

# Suction Sense Project

## ECE 445 Final Report - Fall 2025

---

### Project #19

Jeremy Lee, Suleymaan Ahmad, Hugh Palin

Professor: Arne Fiflet

TA: Lukas Dumasius

<b>1. Introduction:</b>	<b>1</b>
<b>2. Design</b>	<b>3</b>
Sensor Subsystem Design Procedure	3
Power Subsystem Design Procedure	3
BMS Subsystem Design Procedure	4
MCU Subsystem Design Procedure	5
UI Subsystem Design Procedure	5
Sensors Subsystem Design Description	6
Power Subsystem Design Description	6
BMS Subsystem Design Description	7
MCU Subsystem Design Description	8
UI Subsystem Design Description	9
<b>3. Verification</b>	<b>10</b>
Power Subsystem Verification	10
BMS Subsystem Verification	10
Sensor Subsystem Verification	10
MCU Subsystem Verification	10
UI Subsystem Verification	11
<b>4. Costs:</b>	<b>12</b>
Cost Analysis	12
<b>5. Conclusion</b>	<b>13</b>
<b>6. References</b>	<b>15</b>
<b>Figures and Tables</b>	<b>16</b>
Figure 13: MCU Subsystem Requirements + Verification Table	24
Figure 14: MCU Packet Verification Excerpt	24
Figure 15: UI Subsystem Requirements + Verification Table	24

# 1. Introduction:

Currently, suction systems in hospital operating rooms are left running unnecessarily for nearly 35% of their total runtime, including periods such as overnight when no surgeries are taking place. This results in wasted energy, wear overtime on expensive vacuum equipment, and higher maintenance demands. Without any system to detect or alert staff when suction is left on, hospitals face unnecessary electricity consumption and shortened equipment lifespan. This creates a huge inefficiency that scales across entire healthcare systems.

The financial and environmental impact of this waste is significant. Leaving suction on overnight alone contributes to approximately 8 billion kilograms of CO<sub>2</sub> emissions globally every year. This inefficiency can cause hospitals to incur significant additional costs including: replacement vacuum systems that range from \$100,000 to \$750,000, filters that cost \$2,500 to \$10,000, and annual oil changes that add another \$8,000. On top of that, hospitals spend an estimated \$30,835 each year just on electricity for their vacuum systems. Together, these demonstrate the urgent need for a solution that minimizes unnecessary suction runtime, reduces costs, and lessens environmental impact.

To tackle this problem, we proposed a combined hardware and software solution designed to monitor and reduce unnecessary suction usage in operating rooms. At a high level, the system consists of two parts: pressure sensors installed on vacuum systems in each operating room and a software interface that collects real-time suction data and compares it with the operating room schedule.

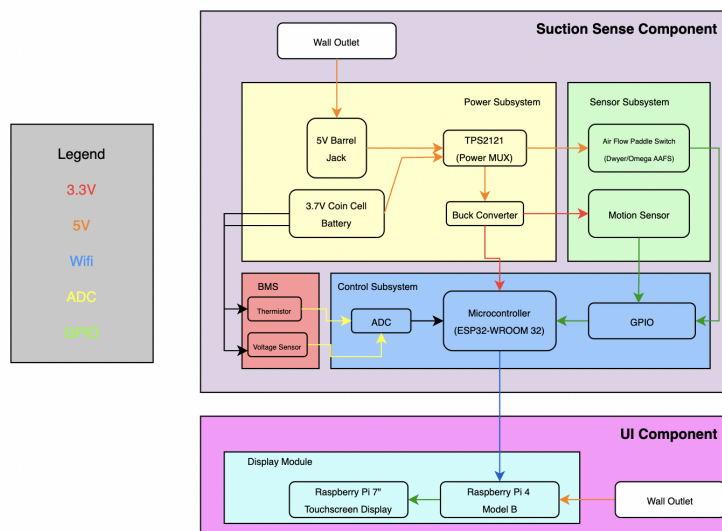


Figure 1: System Block Diagram

Our system is built around five core subsystems: Sensor, Power, BMS, Control, and UI. They work together to deliver reliable, real-time monitoring in the operating room. The Sensor subsystem, composed of a motion sensor and flow sensor, captures suction activity and environmental context. The Power and BMS subsystems provide stable power delivery to the sensors and MCU, managing battery usage when needed. At the center, the Control subsystem uses a custom-designed PCB with an integrated microcontroller to gather suction pressure data and broadcast telemetry over Wi-Fi. This data is received by a Raspberry Pi module, which serves as the processing hub, stores all incoming information, and connects to the hospital network, via Epic integration or manual schedule entry, to compare suction activity against scheduled procedures. Finally, the UI subsystem, running on the Raspberry Pi touchscreen, presents a clear visual interface for medical staff, highlighting each operating room's suction status and flagging unnecessary usage in red. Placed centrally in the hallway, this display ensures staff can quickly identify issues and respond promptly.

We revised our original UI subsystem design after determining that integration with Carle's local scheduling system was not feasible. To compensate, we updated our database schema to incorporate motion data, which allowed us to infer whether an operation is in progress. This adjustment enables our dashboard logic to reliably identify and flag unnecessary suction usage even without direct access to the operating room schedule.



## **2. Design**

### **2.1 Design Procedure**

#### **Sensor Subsystem Design Procedure**

Our sensor subsystem is responsible for detecting whether suction is active on the vacuum line and whether the operating room is occupied. At a high level, we required a design that was compact, affordable, and reliable for continuous monitoring. We began by evaluating several sensor options for measuring flow rate. The two primary approaches were using a differential pressure transducer or using a paddle-type mechanical flow switch. When discussing the pressure transducer option with our collaborators at Carle, they expressed concerns due to pressure transducers only measuring static not differential pressure. We would have to calibrate our sensor setup with each pipe before we begin the calculations. Finally, Pressure transducers also require analog signal conditioning circuitry, which adds cost and complexity.

We therefore selected the AAFS ADJ Air Flow Paddle Switch (Air Flow Switch), which provides a direct mechanical indication of airflow using a simple SPDT contact. This switch operates reliably within the typical hospital suction range of 200–1800 FPM and is rated for continuous use. Which is important for operating in a hospital environment. Its binary output simplifies the electronics needed at the MCU interface and allows us to detect real-time suction state without complex signal processing.

For room occupancy, we chose to include a secondary hardware sensor as a fallback in case integration with OR scheduling data was not feasible. Because of strict HIPAA requirements and potential delays in obtaining data access, we designed the subsystem to remain fully functional using only an external sensor. A passive infrared motion sensor satisfies these constraints by detecting changes in infrared radiation without capturing images or storing identifiable data. We selected the Adafruit MINI PIR motion sensor because it operates at both the supply voltage and logic levels required by our MCU, making integration straightforward while ensuring safe operation.

#### **Power Subsystem Design Procedure**

Our power subsystem is planned to be stable in supporting all onboard electronics, consisting of a few 5 V sensors and a 3.3 V microcontroller. These components need to be supplied at varying

voltages and therefore the system will need to produce both rails in a small and electrically sound manner. The second design requirement is reliability because steady power supply is crucial to the correct sensor readings and continuous work of microcontrollers. This pushed us into giving a lot of thought in the manner we were going to control and allocate power without making the entire design too complex.

In order to achieve the reliability requirement, we added two independent 5 V sources in the design. Using a single source presents a single-point of failure risk and therefore introducing redundancy can help a great deal to increase the system uptime. To get a good control over these two sources, circuitry must be provided which will automatically determine which one to use at any particular moment. This is the reason why we have chosen a power multiplexer (power mux). The power mux continuously checks the two sources and chooses the better of the two, in most cases, the one with a higher and more consistent voltage, and allows the switchover to be set up smoothly in case one of the sources drops unexpectedly, disconnects, or becomes unreliable. This operation makes sure that the 5 V rail is not lost, which is important in powering sensitive sensors that can be subjected to malfunction with brownouts or momentary outages.

The constant 5 V of the power mux is then exploited directly to drive the sensors, as well as to input to a buck converter, which reduces the voltage to 3.3 V powering the microcontroller and other low-voltage devices. When the muxed 5 V is used as input to the buck converter, the redundancy and stability of the 3.3 V rail is inherited by the 5 V rail. This design, based on redundant 5 V sources chosen via a power mux and then decreased to MCU power levels, leaves a fault-tolerant space-saving power subsystem, and allows continuous operation of the system even when one of the power sources is unstable or fails entirely. Such a layered method of control and redundancy gives a robust basis of system performance reliability.

## **BMS Subsystem Design Procedure**

A 3.7 V lithium-ion battery is one of our two sources of power, given its high energy density, small size, and the fact that it can be used as a dependable backup power source in the event that the primary source is not available. Due to the extreme protection and control needs of lithium-ion chemistry to guarantee safe functioning, we incorporate a specific Battery Management System (BMS) to protect the cell to all operational conditions. The BMS is used to maintain a battery in a safe condition against the dangerous conditions of overcharging, over-discharging, and excessive temperature which can otherwise result in shorter cell life, system instability or safety hazards.

The implementation of our BMS is based on three main parts: MCP73831-2-OT charging IC, a 10 k - resistor thermistor, and a divider to detect voltage linked to the MCU. The MCP73831-2-OT drives the charging protocol each time the 5 V barrel-jack input is to be

connected with a constant-current/constant-voltage (CC/CV) charging profile to replenish the cell in a safe manner. This machine has also an inbuilt thermal control, charge stop, and current limiting which makes sure that the battery is not charged with wastage and without surpassing harmless electric limits.

A 10 k oh NTC thermistor is put close to the battery holder and read by one of the ADC inlets of the MCU. The system can stop charging or discharging of the battery when the cell gets too hot or too cold by constantly monitoring the battery temperature allowing the battery to last longer without being damaged. Moreover, there is a basic voltage divider across the battery terminals that feeds a scaled voltage into another MCU ADC input. This allows battery voltage to be measured in real time which can be used to estimate state of charge or generate low-battery alarms or avoid deep discharge. These sensing and protection systems combined form a strong and complete lithium-ion battery BMS solution, which makes the battery safe, healthy, and available to supply reliable backup power whenever it is necessary.

## **MCU Subsystem Design Procedure**

To accurately monitor and coordinate the sensor subsystem, we required a low-power, inexpensive, and reliable microcontroller that was compatible with our sensor, BMS, and power modules while also supporting wireless data transmission. We evaluated several candidates, including the MSP430, ATtiny85, and ESP32. Ultimately, we selected the ESP32 due to its small form factor, low cost, and strong performance characteristics, making it ideal for hospital environments where multiple sensing units may be deployed. Its integrated Wi-Fi capability also allows seamless communication with the Raspberry Pi and hospital networks without requiring an external BLE or Wi-Fi module, reducing system complexity and improving reliability.

For firmware development, we chose the Arduino IDE and programmed the ESP32 in C. The IDE provides built-in ESP32 libraries and tooling, which streamlined development, reduced code overhead, and enabled faster debugging and testing throughout the project.

## **UI Subsystem Design Procedure**

To design our UI subsystem, we first broke down our system requirements into a high-level architecture. We needed clear visualization with near real-time updates, as well as modularity to support future scalability when multiple SuctionSense units are deployed across different operating rooms or even different hospitals. We decided that a local message bus would be essential for reliably transporting sensor data to the visualization layer, which led us to adopt an MQTT broker running on the Raspberry Pi. This allowed each ESP32 module to publish flow and motion telemetry in a structured and extensible way. Next, we selected a lightweight SQLite

database to store incoming sensor data and schedule information, giving the UI subsystem a persistent and queryable source of truth. To bridge this data with the actual display, we implemented a Crow C++ web server that periodically queries the database and exposes the latest operating room status through a simple HTTP endpoint. Finally, we chose the Raspberry Pi paired with a 7" touchscreen to host and render the dashboard, allowing the UI to update every few seconds and present color-coded indicators for each operating room. Together, these components, MQTT for messaging, SQLite for storage, Crow for backend logic, and the Raspberry Pi touchscreen for visualization, form the UI subsystem that meets our requirements for responsiveness, modularity, and long-term scalability.

## **2.2 Design Details**

### **Sensors Subsystem Design Description**

The sensor subsystem consists of two primary components: the AAFS ADJ Air Flow Paddle Switch, which detects the presence of suction, and the Adafruit MINI PIR motion sensor, which detects room occupancy. We can see how our sensors integrate with the MCU in Figure 1

The switch outputs two possible states. When airflow exceeds the adjustable threshold, terminals 1 and 2 are electrically connected. When airflow is absent, the switch instead connects terminals 1 and 3. To interface this with the MCU, our circuit supplies a stable 3.3 V logic level to terminal 1. Terminal 2 is then routed to an MCU GPIO pin with a pull-up resistor implemented in our code in Figure 20

This establishes a default logic HIGH (3.3 V) when the switch is open (no airflow), using the ESP32's internal pull-up resistor. A logic LOW (0 V) when the switch is closed (airflow). This allows us to record a change in real time which will then be communicated to the Raspberry Pi.

The Adafruit MINI PIR motion sensor provides us with occupancy sensing using passive infrared sensing. The PIR sensor is powered from the system's 5 V rail. When motion is detected we receive a logic HIGH (3.0 V) and a logic LOW (0 V) when motion is not detected. The IO pin then uses an internal pull-down resistor in Figure 20.

### **Power Subsystem Design Description**

The power subsystem is responsible for delivering continuous and reliable power to our board so that our electronic components such as our MCU and sensors can operate as intended. Our sensors rely on a reliable 5v and our MCU relies on a stable 3.3v to operate. This means our

subsystem would have to reliably provide both. To enhance the reliability of this system, we decided to create a power system that made use of two power inputs, A 3.7v signal from a rechargeable 2032 Lithium Ion as well as 5v input form a DC barrel jack connector that will connect to a wall outlet. In order to make use of both inputs, we connect them to a Power Mux, allowing us to effectively operate in all of 3 different scenarios.

**Scenario 1:** the 5v barrel jack is connected and the 3.7v coin cell is discharged, where the output is 5v.

**Scenario 2:** The 3.7v coin cell is charged and the barrel jack is not connected where the output is the 3.7v signal.

**Scenario 3:** both the barrel jack is connected and the 3.7v coin cell is charged, where the output is the 5v signal since it is a higher voltage.

The only scenario where our Subsystem has no power is when the coin cell is discharged and the barrel jack is not connected.

To ensure seamless operation, we choose our Power Mux with the following requirement: at most a 10us switching time to avoid a transient that would brownout our MCU. This is why we choose the TPS2121[10], which is a Power Mux that is manufactured by Texas Instruments that makes use of an Ideal Diode O-ring mechanism to seamlessly transition between two power sources. Figure 5 below contains the specification information from the datasheet of this device, demonstrating that it more than fits our requirements.

In order to operate our MCU and Sensors we require a 3.3v signal, the signal that our Power Mux emits is 5V, which does not currently meet our needs. To resolve this we will make use of a regulator that can step our voltage down to a safe operable voltage. To do this we have decided to use an AP62150 synchronous buck converter that will take the inputted signal from the Power Mux and step it down using this device by configuring the Vout using the correct component values according to the table below.

## **BMS Subsystem Design Description**

One of our two power sources is a 3.7 V lithium-ion battery, which serves as a compact and energy-dense backup supply. To ensure safe and reliable operation under all conditions, a dedicated Battery Management System (BMS) has been integrated to monitor and protect the battery against unsafe states such as overcharging, over-discharging, and excessive temperature.

The BMS consists of three primary components: the MCP73831-2-OT Li-ion charging circuit, a 10 k $\Omega$  NTC thermistor, and a voltage sensing circuit. The MCP73831-2-OT manages the

charging process when the 5 V barrel jack is connected, using a constant-current/constant-voltage (CC/CV) profile to safely charge the cell while providing built-in thermal regulation and charge termination. To ensure thermal safety, a 10 k $\Omega$  NTC thermistor is placed near the battery holder and connected to an MCU analog input, allowing continuous temperature monitoring and enabling the system to halt charging or discharging if the temperature falls outside safe limits. Additionally, a voltage divider circuit connected to the battery terminals feeds into another MCU ADC input to measure the battery voltage and estimate its state of charge, ensuring that the system can make intelligent power management decisions such as switching sources, issuing low-battery warnings, or preventing deep discharge. Together, these components provide a comprehensive and reliable BMS solution that ensures the lithium-ion battery operates safely, efficiently, and within optimal performance parameters.

## **MCU Subsystem Design Description**

The MCU subsystem provides system control and wireless connectivity for communication with the external software subsystem. It reads output data from the BMS to manage power switching between wall and battery sources for optimal energy efficiency, while monitoring sensor signals through its GPIO pins. We selected the ESP32-WROOM-32E-N4 for its powerful microcontroller core with integrated Wi-Fi, compact form factor, low cost, and proven reliability. Its built-in Wi-Fi enables direct communication with the Raspberry Pi and hospital networks without the need for an external BLE module, simplifying both hardware design and system integration[5]. Figure 4 describes the pin-out of the ESP-32.

The first function of the MCU subsystem is to monitor and record key values from the Battery Management System and power system. The MCU will read inputs from the thermistor and voltage divider connected to the BMS to ensure the battery is operating within safe limits for temperature and charge. In addition, the MCU will track which power source is currently active by monitoring the output of the power mux, allowing it to log transitions between wall power and the battery supply for reliability and energy efficiency.

The MCU also monitors the AAFS flow sensor and the Adafruit motion sensor to detect instances of unnecessary suction within an operating room. While continuously polling and collecting this data, it streams the readings over Wi-Fi to the external software subsystem hosted on a Raspberry Pi, where the results are visualized through a custom user interface. To support this communication, an MQTT client will be implemented on the ESP32 to serve as a local message bus, enabling the MCU to periodically publish telemetry data that can be received and processed by the Raspberry Pi module.

## UI Subsystem Design Description

The Raspberry Pi 4 Model B paired with the Raspberry Pi 7" Touchscreen Display will serve as the central monitoring and alert system. The Raspberry Pi was chosen for its quad-core processing power, I/O support, and strong software ecosystem, which will allow us to easily integrate with the Epic scheduling system[6]. A diagram displaying the pi capabilities can be seen in Figure 5. The 7" touchscreen will allow the module to be mounted in the hallway, providing an interface that allows staff to quickly view operating room suction status, with clear color-coded indicators and alerts. This combination also enables both visual and audio notifications when suction is unnecessarily left on, ensuring staff can respond promptly.

Figure 6 outlines the flow diagram for our application. To accurately receive and process data being streamed out from the ESP32, we will first run an MQTT broker service on the Pi to collect sensor telemetry data[3]. The ESP32 firmware will connect directly to the broker, sending flow and motion sensor data for processing. Next, the data will be inserted into a locally hosted SQLite database, that will additionally store operating room schedule data.

A lightweight Crow web application written in C++ serves as the interface between the SQLite database and the user[4]. It periodically queries the database and exposes an HTTP endpoint that returns each operating room's latest suction status in JSON format, while hosting a static dashboard on the Raspberry Pi's 7" touchscreen. The dashboard polls this endpoint every few seconds and displays color-coded tiles(green for normal use, red for unnecessary suction, and gray for no suction) allowing staff to quickly identify rooms with suction left on. This setup provides an automated, real-time monitoring system that updates continuously with no manual input required.

## **3. Verification**

### **Power Subsystem Verification**

Our Power Subsystem successfully met the requirements we discussed in the requirements & verification table. We aimed to create a dual-redundant system that can be powered by a battery and a barrel jack. We were able to create this with the use of a TPS2121RUXR power mux, that outputted a stable 5v source from either source as shown in Figure 16. This was good enough to power our sensors but we needed a 3.3v voltage source that could take high loads for our MCU. This was created using a AP62150Z6 buck converter which successfully stepped our 5v down to 3.3v as shown in Figure 16. The last requirement was that the transient of switching was below 200ma so that it would not damage our buck converter as illustrated in Figure 17, this was more than met, as the in rush current was only about 4ma.

### **BMS Subsystem Verification**

Our BMS Subsystem successfully met the requirements we discussed in the requirements & verification table. This was because: MCP73831-2-OT has a thermistor built in Thermistor to control shutdown 60°C, Charging circuit worked for 3.7 LIR2032 battery that we discharged and operated correctly as measured through the multimeter. We also tested the device under extreme temperature conditions to make sure the IC worked correctly to shut up the charging to protect from over voltage and over heating, increasing the safety of the product.

### **Sensor Subsystem Verification**

Our Sensor Subsystem successfully met the requirements (Figure 7) we discussed in the requirements & verification table. We aimed to detect flow rate sensors and return the correct state of flow with 99% accuracy. We validated this by cycling the system through known suction on/off states and comparing the sensor's reported readings to the expected values, this is shown in Figure 8. We then measured the voltages of the flow sensor and confirmed that it consistently received a stable 3.3 V supply and that its output remained conditioned when cycling the system between no suction and maximum suction conditions. Measurements taken with a DMM showed correct voltage transitions corresponding to airflow changes. The motion sensor was validated by supplying it with 5 V and monitoring its output pin with a DMM while repeatedly introducing and removing motion in the room.

### **MCU Subsystem Verification**



Our MCU subsystem met all the requirements (Figure 13) we discussed in our requirements & verification tables. We started by logging a 2-hour period of telemetry of our MQTT wifi transmission before running the full 24-hour test. From our initial 2-hour test, we observed minimal packet loss when comparing packets sent versus received, we lost 5 packets. With this result we extrapolated the full 24-hour verification which be only a loss of 60 packets, which was well below our allowable limit of 86, thereby meeting our reliability requirement (Figure 14). We verified this requirement by applying known suction states using our test stand and having the MCU sample the flow switch output while we manually toggled between suction ON and OFF conditions. The MCU's readings were compared against these known states over repeated trials, confirming that the sampled values stayed within the required  $\pm 2\%$  error margin. We tested automatic reconnection by powering our MCU and Raspberry Pi on and off and seeing if it automatically reconnects. In each case, the MCU successfully reconnected and resumed MQTT publishing without user intervention, since our firmware automatically retries the connection and re-subscribes to the broker after any network loss.

## **UI Subsystem Verification**

Our UI subsystem successfully met the majority of requirements outlined in the Requirements & Verification table. The objective was to create a fast, reliable, and clinically intuitive interface that could be used by medical staff in a centralized location. As such, our requirements described in Figure 15 emphasized efficiency, availability, and functionality.

To evaluate UI responsiveness, we triggered suction events using our testing valve and measured the time from telemetry publication to on-screen visualization; in all cases, the UI refreshed within our 10-second requirement. Motion-based updates using the PIR sensor also met this requirement, though with slightly longer refresh times due to inherent debounce behavior. We additionally validated system availability by disconnecting the suction sensor from Wi-Fi and later stopping the MQTT broker on the Raspberry Pi. In both cases, the UI continued functioning, displaying the last known state and demonstrating fault tolerance aligned with our scalability goals. Once connectivity was restored, the system seamlessly resumed normal updates without manual intervention. Finally, although our initial design included an audio alert subsystem for highlighting unnecessary suction usage, project time constraints and practical implementation concerns prevented full integration in this iteration. Instead, we prioritized improving dashboard responsiveness, reliability, and visual clarity. The audio alert framework remains partially implemented and could be re-introduced in future iterations.

## 4. Costs:

### Cost Analysis

According to the UIUC ECE Department statistics, the average EE graduate makes close to \$90,000 annually, which comes out to about \$43.27/hour. Assuming that each of us works 6 hours a week to complete this project, we can estimate the cost of labor to be  $\$43.27/\text{hr} \times 2.5 \times 6\text{hrs/week} \times 14\text{ weeks}$ , which comes out to \$9086.7 per group member or \$27260.10 total. For the cost of materials, we estimate it will come out to \$264.03 to build our entire system. Thus, combining both the cost of labor and cost for parts, the grand total for our project will be \$27,525.13. The table for costs can be found in Figure 18.

### Schedule

For our schedule, we divided the project into 3 main parts: Hardware, Firmware, and Web Server Software. Suleymaan was mainly responsible for designing and verifying the PCB, while Jeremy worked on developing the firmware for the ESP32 and also the backend web server application code. Hugh worked on the implementation of the SQLite Database as well as the UI design for the application. He also assisted Suleymaan in the design of the PCB. Our full schedule can be seen in Figure 19

## 5. Conclusion

Suction Sense demonstrates a complete, functional solution to the widespread problem of unnecessary suction usage in hospital operating rooms. Through integrated hardware and software design, our system reliably detects suction activity, cross-references it with operating room schedules, and provides clear, near-real-time feedback to clinical staff. Our prototype met all major high-level requirements, including stable 24-hour telemetry transmission, sub-10-second UI refresh rates, accurate sensor readings, and power failover, giving strong evidence that our design will work effectively in a real hospital environment. Together, these accomplishments show the feasibility of reducing millions of dollars in wasted energy and equipment wear through a low-cost, scalable monitoring solution in Suction Sense.

Across the semester, we successfully designed and tested each subsystem: a sensing module using a flow paddle switch and PIR motion sensor, a dual-source power architecture with safe battery management, a Wi-Fi-enabled ESP32 MCU capable of continuous operation, and a Raspberry Pi software platform featuring an MQTT ingestion service, a local SQLite database, a Crow-based backend, and an intuitive touchscreen UI. By fully integrating our system and testing it with the custom-built suction valve simulation testbench, we were able to confirm that suction events propagate from sensor to display reliably and within our timing specifications. Given these results, we are confident that the system’s architecture is sound and that hospitals could deploy Suction Sense with minimal infrastructure changes.

There are some uncertainties remaining with our project. First, we are concerned with long-duration reliability testing and integration with real Epic scheduling systems. While our simulations and controlled tests indicate stable behavior, extended multi-week testing in a production environment would be necessary before true clinical deployment. Additionally, access to real patient-scheduling data is subject to HIPAA and institutional approvals. If such access is restricted, our fallback method using PIR-based occupancy detection may require additional tuning or modified performance specifications to reduce false-positives and also improve accuracy during edge cases such as low-motion procedures.

Ethical considerations guided our development throughout this project. In accordance with the IEEE Code of Ethics and relevant medical-technology standards[8], we designed Suction Sense to avoid harm, clearly communicate system limitations, and protect privacy. As emphasized in our design document, the system is advisory-only and cannot control suction equipment, ensuring clinicians retain full authority over patient-critical systems. To uphold HIPAA principles, no personal health information is stored or displayed; only room-level operational data is processed, and all communication channels can be secured through encryption and access control. Furthermore, our hardware attaches non-invasively to existing medical infrastructure, avoiding interference with clinical workflows. These choices reflect our commitment to safety, honesty, and respect for all stakeholders.

Beyond its technical performance, Suction Sense offers meaningful broader impacts. Reducing unnecessary suction usage directly lowers hospital energy consumption, extends the lifespan of expensive vacuum systems, and decreases global CO<sub>2</sub> emissions, addressing environmental and economic concerns at scale. Improved visibility into OR suction status allows staff to operate more efficiently, reducing operational overhead while also promoting sustainable healthcare practices. By designing a low-cost and scalable solution, we contribute a tool that can benefit not only large medical centers but also smaller or resource-constrained facilities seeking to also reduce suction usage.

In summary, our project demonstrates a viable, ethically responsible, and environmentally beneficial system that can be deployed across hospitals to reduce waste and improve operational efficiency. While further long-term testing and potential specification adjustments may be required for clinical deployment, our results strongly indicate that Suction Sense can achieve its intended impact and serve as a practical step toward more sustainable healthcare infrastructure.

## 6. References

- [1] PIR Motion Sensor Created by Lady Ada, ADA Fruit, [cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf](https://cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf). Accessed 13 Oct. 2025.
- [2] Inc., Alpha Controls & Instrumentation. "AAFS - Adjustable Air Flow Paddle Switch." Alpha Controls & Instrumentation Inc., Dwyer Instruments, [www.alphacontrols.com/AAFS-Adjustable-Air-Flow-Paddle-Switch/model/7411](https://www.alphacontrols.com/AAFS-Adjustable-Air-Flow-Paddle-Switch/model/7411). Accessed 12 Oct. 2025.
- [3] "1 Introduction." MQTT Version 5.0, [docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html](https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html). Accessed 12 Oct. 2025.
- [4] "Crowcpp." Crow, [crowcpp.org/master/](https://crowcpp.org/master/). Accessed 12 Oct. 2025.
- [5] ESP32-WROOM-32E ESP32-WROOM-32UE Datasheet Version 1.9, [www.espressif.com/sites/default/files/documentation/esp32-wroom-32e\\_esp32-wroom-32ue\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf). Accessed 13 Oct. 2025.
- [6] Raspberry Pi 4 Model B Published February 2025 Raspberry Pi Ltd, [datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf](https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf). Accessed 13 Oct. 2025.
- [7] MCP14628 Data Sheet, [ww1.microchip.com/downloads/aemDocuments/documents/APID/ProductDocuments/DataSheets/MCP14628-Family-Data-Sheet-DS20002083.pdf](https://ww1.microchip.com/downloads/aemDocuments/documents/APID/ProductDocuments/DataSheets/MCP14628-Family-Data-Sheet-DS20002083.pdf). Accessed 13 Oct. 2025.
- [8] IEEE, "IEEE Policies - Section 7-8 - IEEE Code of Ethics," [Online]. Available: <https://www.ieee.org/about/corporate/governance/p7-8.html>. [Accessed: 18-Sep-2025].
- [9] Chao, Sharon. "Suction Sense Lecture Proposal." Carle Illinois College of Medicine, [courses.grainger.illinois.edu/ece445/lectures/Fall\\_2025\\_Lectures/Lecture2/lecture\\_2\\_suction.pdf](https://courses.grainger.illinois.edu/ece445/lectures/Fall_2025_Lectures/Lecture2/lecture_2_suction.pdf)
- [10] TPS212x Data Sheet <https://www.ti.com/lit/ds/symlink/tps2120.pdf?ts=1760390503520>
- [11] NCP21XV103J03RA Data Sheet <https://pim.murata.com/en-us/pim/details/?partNum=NCP21XV103J03RA>

# Figures and Tables

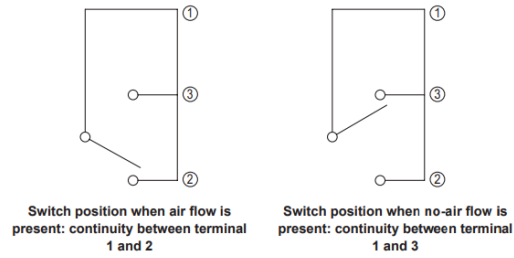


Figure 2: Circuit diagram of our AAFS Switch

Table 10-3. Automatic Switchover Design Requirements

DESIGN PARAMETER	SPECIFICATION	DETAILS
IN1 Voltage	$V_{IN1}$	12 V
IN2 Voltage	$V_{IN1}$	5 V
Load Current	$I_{OUT}$	2 A
Load Capacitance	$C_L$	200 $\mu$ F
Maximum Inrush Current	$I_{INRUSH}$	100 mA
Switchover Time	$t_{SW}$	TPS2120: 5 $\mu$ s
Mode of Operation	Automatic Switchover	TPS2121: XCOMP

Figure 3: Power Mux Voltage Specifications[10]

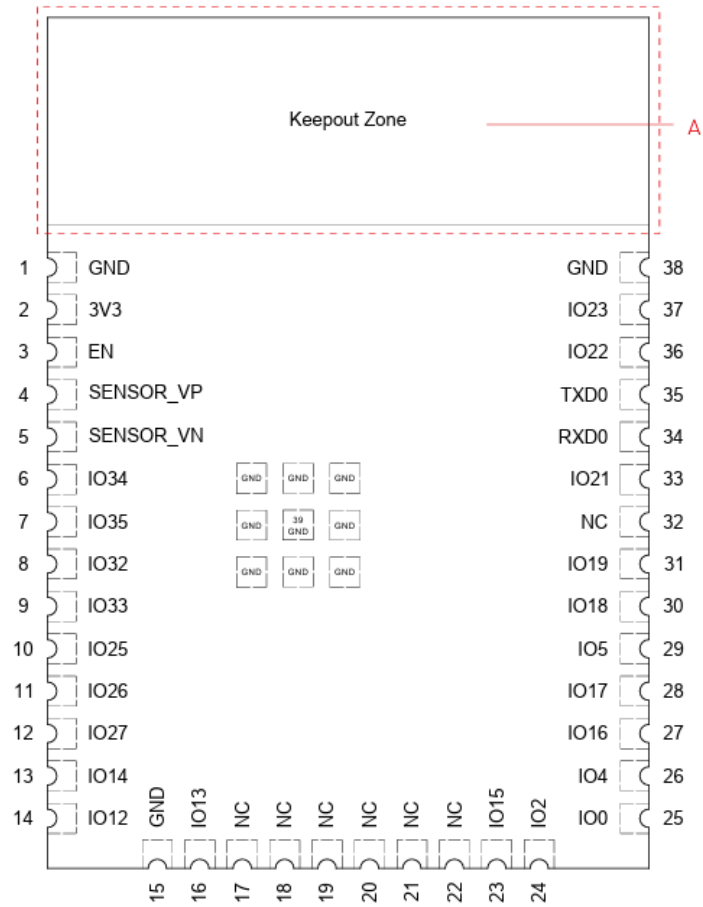
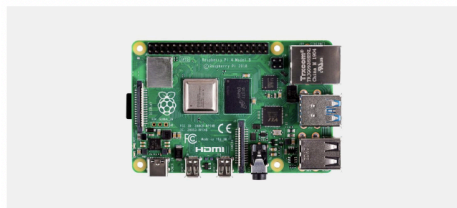


Figure 4: ESP32-WROOM-32E Pin Layout [5]



Raspberry Pi 4 Model B features a high-performance 64-bit quad-core processor, dual-display support at resolutions up to 4K via a pair of micro HDMI ports, hardware video decode at up to 4Kp60, up to 8GB of RAM, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 3.0, and PoE capability (via a separate PoE HAT add-on). For the end user, Raspberry Pi 4 Model B provides desktop performance comparable to entry-level x86 PC systems.

This product retains backwards compatibility with the prior-generation Raspberry Pi 3 Model B+ and has similar power consumption, while offering substantial increases in processor speed, multimedia performance, memory, and connectivity.

The dual-band wireless LAN and Bluetooth have modular compliance certification, allowing the board to be designed into end products with significantly reduced compliance testing, improving both cost and time to market.

Figure 5: Raspberry Pi Description and Capabilities from Datasheet

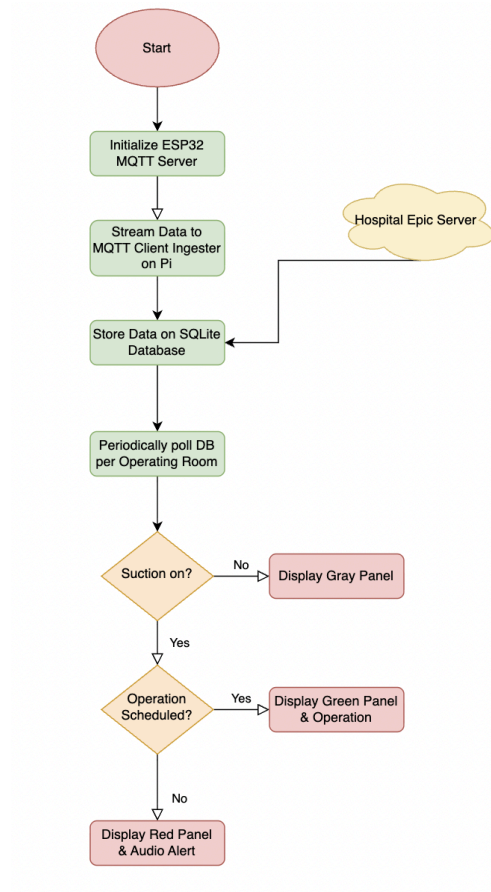


Figure 6: Chart displaying the software application flow from our Suction Sense Module to Raspberry Pi

Requirements	Verification
The flow rate sensor must return the correct state of flow with 99% accuracy.	When producing a vacuum on our testing stand, connect a DMM between terminal 2 (output) and terminal 1 (ground) of the AAFS paddle switch to measure its logic state. Compare the measured output voltage to the known on/off state of the vacuum pump over repeated cycles, recording results in a data table. Verification is achieved if the switch output matches the pump state in at least 99% of trials.
The flow rate sensor requires a 3.3 V supply, so we will supply it with a stable 3.3 V rail while conditioning its output for the MCU.	We will use our DMM to measure input voltage being a stable 3.3 V from our rail. Then measure the output voltage of our



	switch on the data terminal, ensuring it is still a stable 3.3 V when the flow rate is changing. We will change flow rate from none to maximum hospital suction.
The motion sensor shall be supplied with 5 V and will present a 3.0 V output signal compatible with the MCU input.	Check the power rail the motion sensor is supplied with is 5 V, and its output pin shall be monitored with a DMM connected between signal and ground. The measured output voltage will be compared to observed room motion (person entering/leaving) over repeated trials, and results will be recorded in a data table. Verification is achieved if the output consistently switches to ~3.0 V during motion.

*Figure 7: Sensor Subsystem R&V Table*

<b>Trial</b>	<b>Known State (Ground Truth)</b>	<b>Sensor Reading</b>	<b>Match (Y/N)</b>
1	OFF	OFF	Y
2	ON	ON	Y
3	OFF	OFF	Y
4	ON	ON	Y
5	OFF	OFF	Y
6	ON	ON	Y
7	OFF	OFF	Y
8	ON	ON	Y
9	OFF	OFF	Y
10	ON	ON	Y
11	OFF	OFF	Y
12	ON	ON	Y
13	OFF	OFF	Y

14	ON	ON	Y
15	OFF	OFF	Y
16	ON	ON	Y
17	OFF	OFF	Y
18	ON	ON	Y
19	OFF	OFF	Y
20	ON	ON	Y

*Figure 8: Suction Sense Testing Data*

## 9 Setting the Output Voltage

The AP62150 has adjustable output voltages, starting from 0.8V, using an external resistive divider. The resistor values of the feedback network are selected based on a design trade-off between efficiency and output voltage accuracy. There is less current consumption in the feedback network for high resistor values, which improves efficiency at light loads. However, values too high cause the device to be more susceptible to noise affecting its output voltage accuracy. R1 can be determined by the following equation:

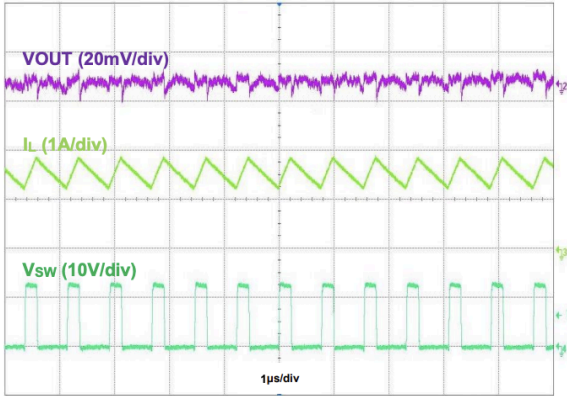
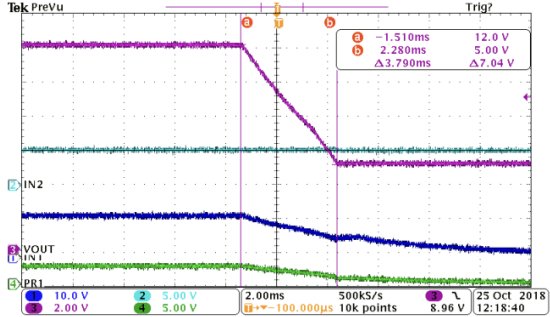
$$R1 = R2 \cdot \left( \frac{V_{OUT}}{0.8V} - 1 \right) \quad \text{Eq. 8}$$

Table 1 shows a list of recommended component selections for common AP62150 output voltages referencing Figure 1. Consult Diodes Incorporated for other output voltage requirements.

Table 1. Recommended Component Selections						
AP62150						
Output Voltage (V)	R1 (kΩ)	R2 (kΩ)	L (μH)	C1 (μF)	C2 (μF)	C3 (nF)
1.2	4.99	10	1.2	10	22	100
1.5	8.66	10	1.5	10	22	100
1.8	12.4	10	1.8	10	22	100
2.5	21.5	10	2.2	10	22	100
3.3	31.6	10	3.3	10	22	100
5.0	52.3	10	3.3	10	22	100

*Figure 9: Buck Converter Voltage Specifications*

Requirements	Verification
<ul style="list-style-type: none"> <li>The 3.3 V rail must remain within ±5% regulation to ensure stable MCU operation.</li> </ul>	Operate board in standard operation with each power source and verify with waveform from data sheet

	 <p>Figure 16. Output Voltage Ripple, VOUT = 3.3V, IOUT = 1.5A</p>
<ul style="list-style-type: none"><li>• The power subsystem must seamlessly switch to battery power via the BMS when the external 5 V supply is removed, with no loss of operation.</li></ul>	<p>Operate board with single power source and then disconnect power source to force power mux. Probe signals for Vin to Buck converter and Vout, and compare with datasheet to observe the transient by zooming in really close on time scale. Ensure operation is as expected per datasheet</p>  <p>Figure 10-10. Automatic Switchover from IN1 to IN2</p>
<ul style="list-style-type: none"><li>• Buck converter must be able to step down 5 to 3.3v with load current &lt;200ma</li></ul>	<p>Measure current through Vout using a rogowski coil or current probe or through shunt resistor. Should be similar to iL in this graph from datasheet:</p>

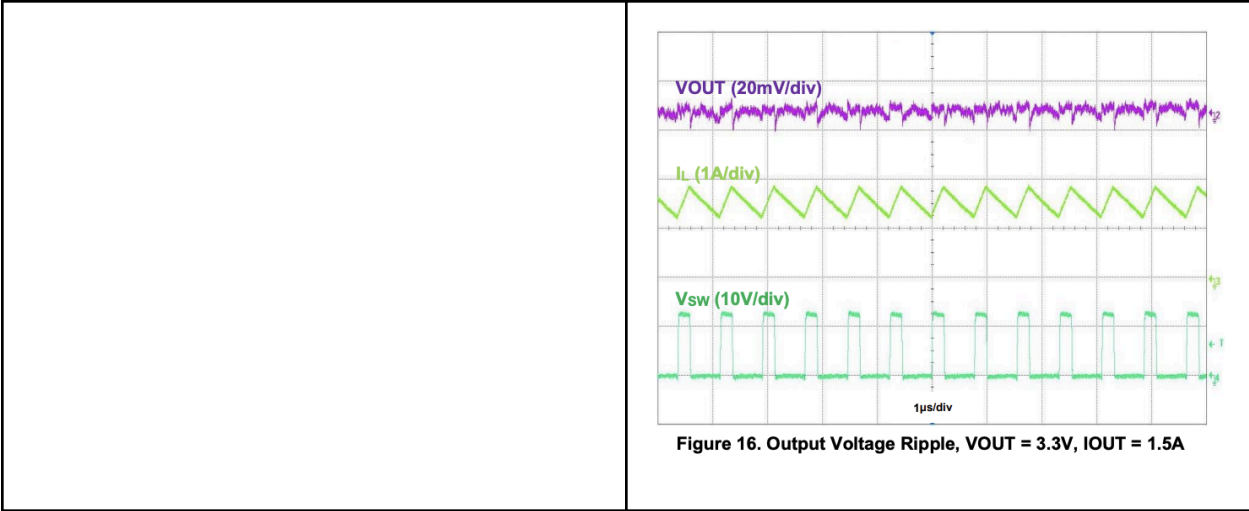


Figure 10: Power Subsystem R&V table

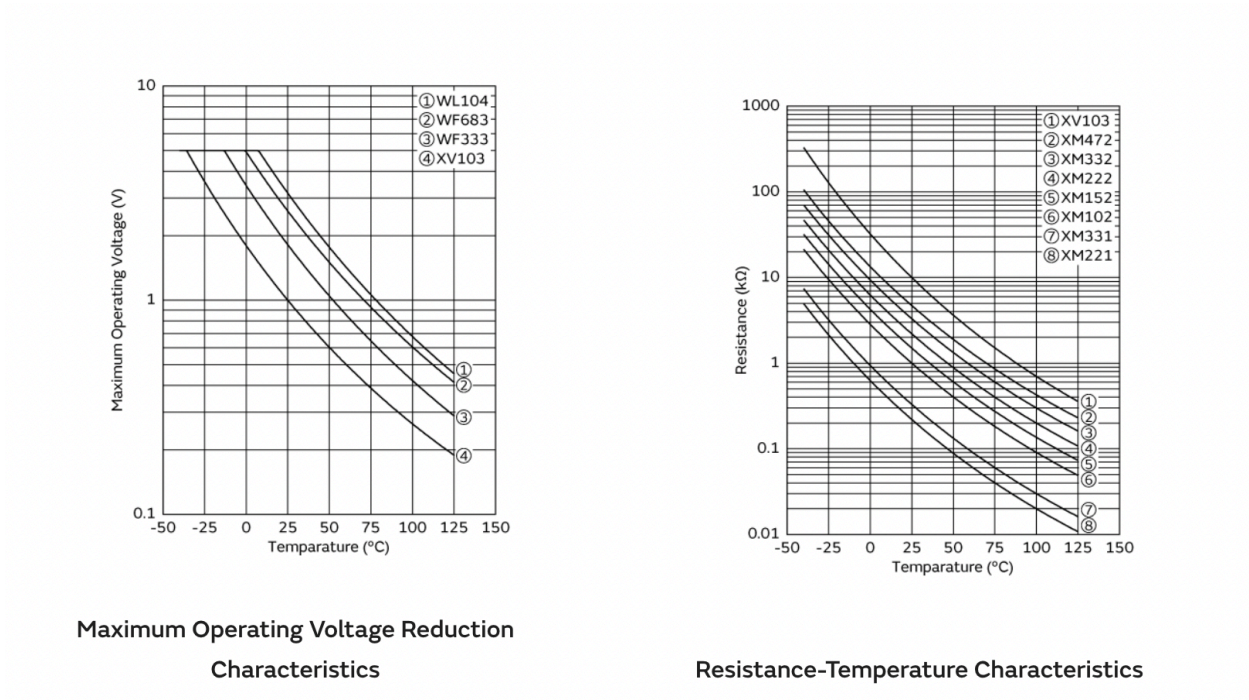


Figure 11: NTC Thermistor Voltage Characteristics[11]

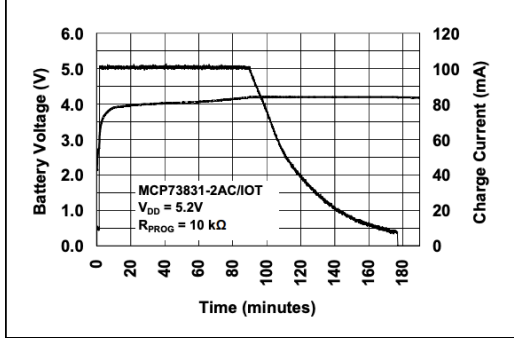
Requirements	Verifications
The thermistor must be NTC 10 kohm to reduce power draw, but accurately provide temperature values.	Probe thermistor voltage on an oscilloscope and do a verification test where we artificially heat up the board and see if the voltage across is reflected.
Charging circuit should be able to accurately charge 3.7v li-ion battery.	<p>Probe battery as it is in the charging state, and verify with expected charge profile form datasheet shown below.</p> 
Voltage Divider should be able to be stepped down to voltage that can safely read by MCU and data can be streamed with less than 5% error.	Probe output of voltage divider to ensure voltage is within acceptable threshold to be read by GPIO pin.

Figure 12: BMS Subsystem R&V Table

Requirements	Verification
The MCU's wifi streaming must ensure that $\geq 99.9\%$ of 1 Hz telemetry messages reach the Raspberry Pi over a 24-hour period. This ensures system reliability and minimal data loss during continuous operation.	Create a custom MQTT subscriber with timestamps and run the MQTT client for 24 hrs; count total vs. received messages on the Pi. Verify $\leq 86$ missed of 86,400 total.
The MCU must be capable of sampling the flow rate sensor output with a correct reading within a $\pm 1-2\%$ margin of error.	Provide suction to the flow rate sensor and have the MCU sample the output and compare the recorded readings to the known state. Verification is achieved if the MCU's readings stay within $\pm 2\%$ of the known flow state in repeated trials.

The MCU must reconnect automatically if the WiFi link to the Raspberry Pi is lost.	Disable the WiFi link between the MCU and Raspberry Pi, then restore it. Verify that the MCU automatically reconnects without user input and resumes data transmission. Verification is achieved if reconnection occurs reliably in repeated trials of various times such as, 1 second, 5 seconds, and 10 seconds.
--	--

*Figure 13: MCU Subsystem Requirements + Verification Table*

timestamp	packet_se	packet_re	loss_flag
12/7/2025 0:37	1	0	1
12/7/2025 1:05	1	0	1
12/7/2025 1:22	1	0	1
12/7/2025 1:58	1	0	1
12/7/2025 2:11	1	0	1

*Figure 14: MCU Packet Verification Excerpt*

Requirements	Verification
The Suction Sense UI should update room tiles every $\leq 10$ s. This will ensure our system provides clinically useful “near-real-time” feedback.	Publish via a change for an operating room to turn suction on from an off state. Start stopwatch. Observe tile color/text change from gray to green on the kiosk. Must update $\leq 10$ s
The system should continue to serve the UI even if the MQTT broker is down. This ensures that the UI won’t crash even if one of the telemetry systems is down	Stop the MQTT service. Ensure that the software application still returns last known statuses (HTTP 200)
Audible alert from Pi plays on transition to red and can be acknowledged	Trigger red. System plays alert sound once. Tap “Acknowledge” (or button) to silence the alert for the configured window.

*Figure 15: UI Subsystem Requirements + Verification Table*





Figure 16: Powersystem when board turns on

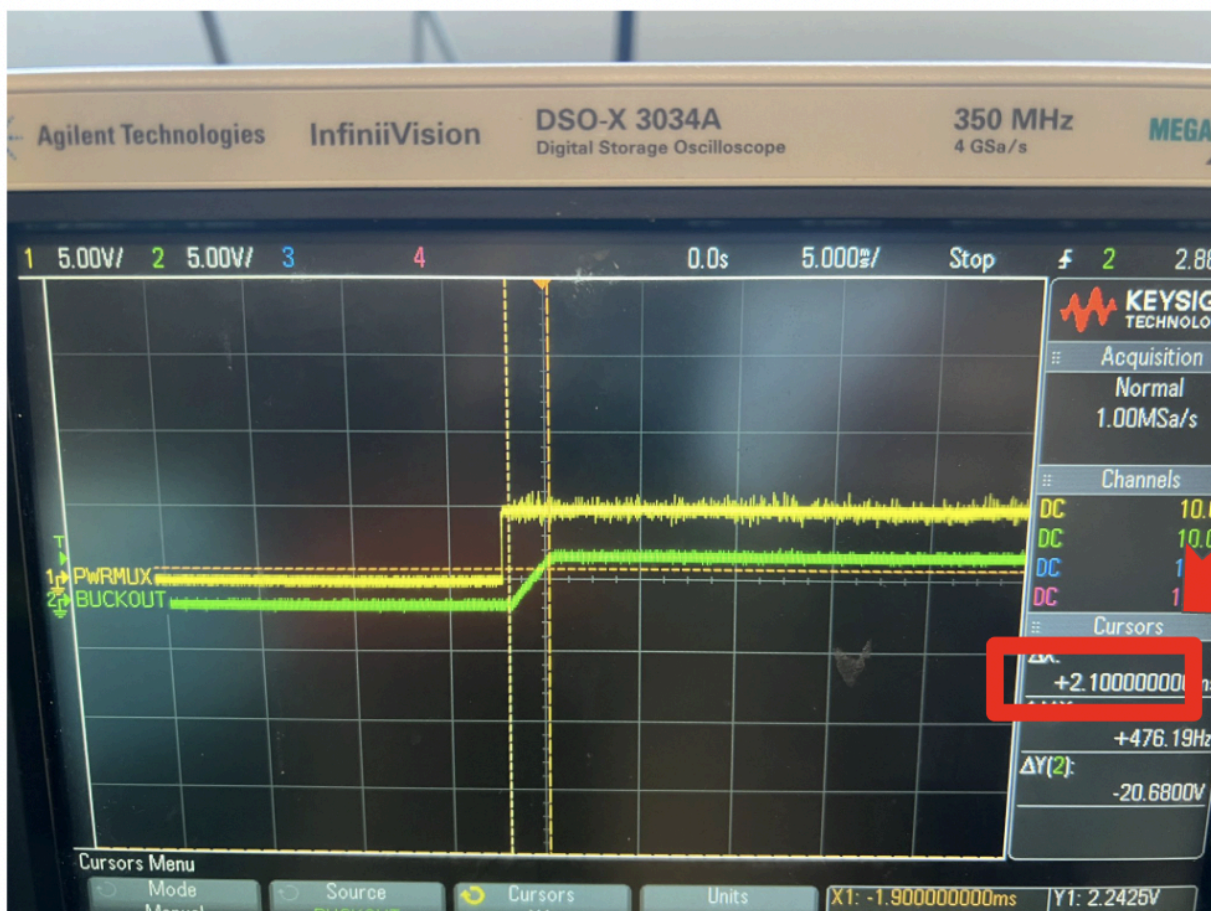


Figure 17: transient between switching sources

<i>Description</i>	<i>Manufacturer</i>	<i>Part Number</i>	<i>Quantity</i>	<i>Unit Cost</i>	<i>Total Cost</i>
Boost Converter	Texas Instruments	<a href="#">TPS61222</a>	1	\$1.22	\$1.22
LCD Display	Raspberry Pi	<a href="#">SC1635</a>	1	\$81.25	\$81.25
Raspberry Pi	Raspberry Pi	<a href="#">Raspberry Pi 4 Model B 2019</a>	1	\$63.88	\$63.88
Airflow Sensor	Dwyer	<a href="#">AAFS</a>	1	\$106.05	\$106.05
Motion Sensor	Adafruit	<a href="#">HC-SR312</a>	1	\$3.95	\$3.95
MCU	Espressif	<a href="#">ESP32-WROOM-32E-N4</a>	1	\$4.84	\$4.84
Thermistor	Murata	<a href="#">NCP21XV103J03RA</a>	1	\$0.16	\$0.16
Buck Converter	Diodes Inc.	<a href="#">AP62150Z6-7</a>	1	\$0.31	\$0.31
Power Mux	Texas Instruments	<a href="#">TPS2121RUXR</a>	1	\$2.37	\$2.37
					<b>Grand Total:</b> \$264.03

*Figure 18: The Parts List and Cost for Suction Sense*

Week	Task	Person
October 12th-18th	Order parts for prototyping	Jeremy
	Software Planning & Environment Setup, Basic App	Jeremy, Everyone
	PCB Revision	Suley
	Machine Shop	Hugh
October 19th-25th	Power Subsystem bring-up and testing	Suley
	BMS Subsystem bring-up and testing	Suley
	MCU Subsystem & Firmware	Jeremy



	Sensor & Peripheral Subsystem	Hugh
	Test Wifi Message from ESP32 -> Pi	Jeremy, Hugh
October 26th-November 1st	Assemble PCB	Everyone
	2nd Breadboard Demo	Everyone
	Design and implement SQLite DB	Jeremy, Hugh
	PCB Testing	Suley
	Design Enclosure for PCB	Hugh
November 2nd-8th	3rd PCB Order	Everyone
	PCB Revisions	Suley
	Design UI for display, connect to Crow app	Jeremy, Hugh
	Connect backend Crow app to DB	Jeremy, Hugh
November 9th-15th	4th PCB Order	Everyone
	PCB Revisions	Suley
	Software Integration testing	Jeremy
	3D Print Encloser	Hugh
November 16th-22nd	Integration Testing	Everyone
	Final Software Testing	Everyone
November 23rd-29th	Fix Minor Bugs	Everyone
	Fall Break	Everyone
November 30th-December 6th	Final Demos	Everyone

*Figure 19: Team Member Schedule*

```
pinMode(FLOW_PIN, INPUT_PULLUP); // HIGH = no suction, LOW = suction ON  
pinMode(MOTION_PIN, INPUT_PULLDOWN);
```

*Figure 20: Coding Figure*