Robot Communication Systems Final Report

Authors: Amelia Nist, Kailey Brown, Leila Troxell, and Shrinidhi Nadgouda

Statement of Work

I, Amelia Nist, contributed to this project as a member of the software team. My responsibilities included developing code for the robot logic, message formatting, and information transfer as well as creating and executing test cases for these components. I also worked extensively with the XBee 3 microcontrollers, configuring them to align with the project's requirements. Specifically, I tested Python scripts for the XBee 3s, converted the code into MicroPython, and deployed it onto the microcontrollers. I ensured functionality by testing the MicroPython code on the XBee 3s and making modifications to optimize performance.

I, Kailey Brown, contributed to this project as a member of the hardware team. My main role in this project was designing and testing the PCB. I created the schematic and layout in KiCad, designed the buck converter and USB to UART converter circuits, and configured the ZigBee module connections using its datasheet. I verified designs using simulations, resolved a ZigBee footprint issue by rewiring connections to header pins, and coordinated with PCB manufacturers. After manufacturing, I soldered components, tested the buck converter and USB to UART chip, and debugged communication issues, adding an oscillator to the design. I also contributed to presentations and authored sections of the proposal and final report, including societal impact, external standards, test plan, and hardware technical details.

I, Leila Troxell, served as the project leader. I managed the team's gantt chart, organized team meetings, and ultimately ensured we were on time for any given assignment. For other team assignments, I took the lead on the final video, drafting, filming, and editing it.

I was also a part of the software team for the project, involved in all decision logic, sensor information collection and transmission, and the medic user interface. Again I served as a leader of the group, outlining all classes, functions, and test functions for the code, maintaining an overarching idea for the code functionality and connectedness, and updating a to-do list of incomplete tasks. Specifically, I was responsible for the construction of the medic user interface, all code that involved the interaction between the robot and the microcontroller, the majority of the code responsible for the robot movement, as well as half of the test code.

I, Shrinidhi Nadgouda, contributed to this project as a member of the hardware team. I was involved in all decisions around the XBee and PCB design. My responsibilities included researching and testing components of our circuit using the AD2 and WaveForms, and reading datasheets to make sure that all requirements were met for the PCB. I helped select and test the sensors in the XCTU terminal to make sure they transmitted information, and designed and 3D printed a custom PCB and sensor holder for the robot. After the PCB was manufactured I was responsible for soldering and testing components, and troubleshooting the voltage regulator to ensure that the output voltage had the correct frequency and ripple characteristics. I also provided troubleshooting support during full system testing. I contributed to weekly meetings and presentations, and authored sections of the final report, including physical constraints, hardware performance objectives, technical description, and engineering insights.

Table of Contents

Table of Figures

Table of Tables

Abstract

The goal of our project was to design a communication system that enables robots to share critical information with each other and medics during human triage or search and rescue operations. To achieve this, we developed a ZigBee-based communication system that allows robots to efficiently transmit and receive key data such as the location and priority of discovered objects. Our software team wrote Python programs to analyze sensor data from the robots, wirelessly communicate relevant information, and update robot objectives based on incoming data. A ZigBee-based microcontroller was configured and integrated with a PCB, designed by our hardware team, to power the communication system and convert data transmission between USB and Universal Asynchronous Transmitter/Receiver (UART). Additionally, we designed and implemented an algorithm to enable robots to navigate toward goals while avoiding obstacles. Our system enhances coordination between robots and medics, improving the efficiency and safety of operations in critical rescue scenarios.

Background

During natural disasters, time is of the essence - every second wasted could cost a victim their life. Dr. Nicola Bezzo at the University of Virginia, with the support of Northrop Grumman, has been working with a group of students and professionals over the past several years to design and implement a team of robots that could traverse the dangerous terrain after a disaster and report back to nearby medics about victims and the urgency of their injuries.

The role of our team was to create a communication system to attach to the robots so that they can coordinate with other robots during their search procedure, determine what information is the most pressing to present to the medic, and efficiently communicate with the medics.

While robot communication systems exist, none would fulfill all of the requirements that a disaster zone presents. The majority of robotic communication systems rely on WIFI or bluetooth to send messages [1]; these systems are not communicating directly with one another, but rather connecting through a common network and allowing the network to direct their messages to the correct recipient. This is not a resource available in a disaster zone. Some robotic transmission media exist that do not rely on a WIFI system, but are instead reliant on near-field communication [2]. While this is the most viable option for our use-case, it limits efficiency as all robots must then travel back into the limited range of another robot or the medic before transmitting its data; we instead implemented a version of this technique.

Our communication system avoids network connectivity reliance, while also communicating the necessary sensor information from the robots and connecting directly into both a medic's computer and a robot's processor. Our team's experience with the UVA School of Engineering's courses prepared us to engineer this solution. Specifically, Intro to Computer Science, Data Structures & Algorithms I, Fundamentals of Electrical Engineering I, II, and III, Intro to Embedded Systems, and Autonomous Mobile Robots taught us the knowledge we needed to complete this project. With this experience, we were able to program the robots in an organized and intelligent manner. We also knew how to design, simulate, and build a functioning PCB. In addition to these courses, among our team members we had an Applied Mathematics minor which helped with decision analysis algorithms and analytical PCB design.

Project Description

This project has multiple components, actors, and settings. Before discussing the technical details of the project, we will first define the main recurring system components.

The first component is the environment. The end use case of our product will be disaster zones such as natural disasters or war torn cities. The rescue teams will enter the environment and begin to identify, triage, and treat victims. For our testing purposes, all testing was conducted in Professor Bezzo's Autonomous Mobile Robot (AMR) lab in Rice Hall.

The second component is the victims in the environment. During end use, the victims will be analyzed through a variety of onboard vitals detection tools. For abstraction and simplification purposes, the victims during testing scenarios were given a severity based on their location. These relationships are hardcoded into the robot's onboard code. The severities given can be between 1 and 4, with 4 as the most severe and urgent victim.

The third component is the robots. The robots in our system are divided into two separate categories - exploratory and hub. Exploratory robots, which are the on-the-ground robots that explore a given area, locate and rate victims before determining whether the information is urgent enough to deliver back to the hub robot. The hub robot serves as an intermediary between the medic and the exploratory robot. The purpose of the hub is to deliver information to the medic more quickly than the exploratory robot would be able to.

The fourth component is the medic. The medics are receiving the victim information and actively treating the victims. Our communication device is also connected to the medic's laptop, which displays the information about any current victims. With the victim information listing severity and location, medics can efficiently move around and treat victims. For our testing, the medic can inform the robots of a new target and of their own next location, and then complete victims so they are no longer displayed on their UI.

The fifth and final component is the communication device itself. The communication device is composed of an external air quality sensor, a USB cord, the antenna, and the microcontroller. The majority of our project efforts were dedicated to developing and refining this system, which will be described in greater detail below. Nearly all of the decision logic for communication is done through the microcontroller onboard the device. The device is attached to all robots and the medic laptop and enables communication between the host devices. It is also capable of sending commands to the code running on the robots.

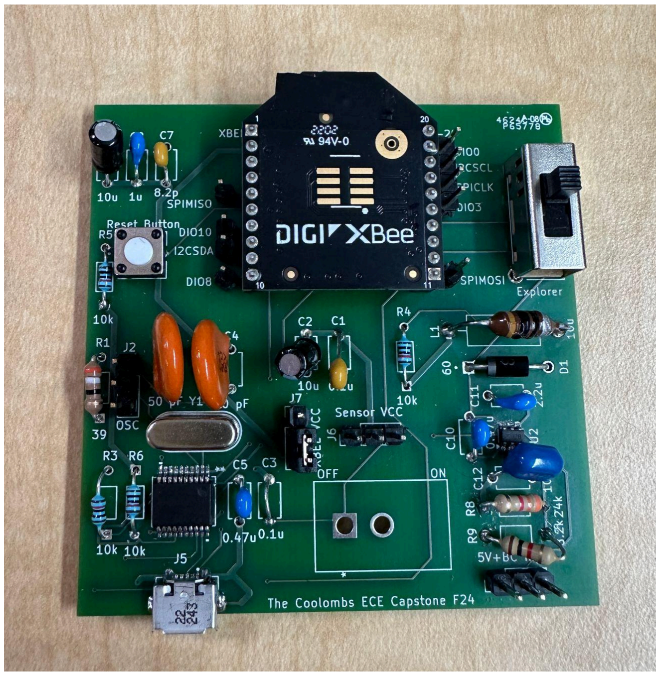


Figure 1. Final Board with Xbee

Performance Objectives & Specifications

The goal of this project is to enable communication between a multi robot and human system, such that it minimizes wasted time and prioritizes the most urgent tasks. The overarching products specifications are as follows:

1) Each communication system must function in long range environments without WIFI or cellular networks
2) Each device must be able to send and receive messages
3) Each communication device can be used either on a robot or connected into a computer. The type of robot to which a communication device is connected is determined by the position of a switch on the PCB

Originally, there was an additional requirement that all devices be interchangeable between a medic or a robot. However, due to the requirements of the ZigBee mesh network, there must always be one device in every network set as a "Coordinator". This is a setting that, once made, is not adjustable by code and must be done manually in the configuration of the XBee antenna. Since we are ensured to always have at least one medic, we decided to have the medic be a specific device and allow the other devices to still be interchangeable between the hub and exploratory robot settings.

Hardware Objectives & Specifications

On the hardware side, our objective was to produce a PCB for the system that, via a microUSB connection, would provide power and a signal transfer path for the XBee microcontroller, ultimately enabling communication between XBee antennas on various hosts in the network. The intended usage of the system is for the user to plug all of the PCB-microcontroller systems into their respective hosts and turn everything on in the correct mode using the power and mode switches.

End user features include a XBee reset button, an XBee mode switch, a PCB power switch, XBee microcontroller breakout pins, and a custom 3D printed PCB and sensor holder to attach onto the ROSbot. The reset button, connected to the XBee microcontroller, provides an easy way to reset the microcontroller without a full system restart. The mode switch is connected to XBee pins and enables the user to easily change the "mode" of the system between exploratory mode and hub mode. The power switch allows the user to shut down the communication system in case the host robot is to be used without it, such as to save the robot's battery. Lastly, we utilized breakout pins with the XBee microcontroller to increase flexibility and overall ease of use when connecting a sensor or other device in the future. On the first iteration of our design, we incorporated a line of sockets to hold our BME 680 temperature and gas sensor. However, that design and those communication connections were specific to the BME680. Per client specifications, to add extra functionality, we adjusted the design to remove this row of sockets and add general breakout pins connected to the XBee microcontroller on the final prototype to enable both SPI (Serial Peripheral Interface) and I2C (Inter-Integrated Circuit) communication protocols between the microcontroller and all compatible devices.

Software Objectives & Specifications

The software had the following specifications and goals. These were loosely defined in the beginning of the project and gained definition as we continued designing.

All messages sent through the communication device include details regarding the x and y coordinates and the severity of the victim.

The goal of the exploratory robots is to locate victims, give them a severity rating and then determine what next step to take. The possible next steps would be to: 1) continue exploring or 2) locate a hub robot to deliver the victim information. Hub robots are sought out when the exploratory robot contains too many victims in their list or they have located a high severity victim. If a hub robot cannot be located, the exploratory robot will instead search for the medic. After locating the hub or the medic, it will transmit all information.

The severity rating is determined by the location at which the robot arrives. Originally this was to be determined by analyzing the color of a block that is located; each color corresponding to a severity rating. However the computational power of the robot was not sufficient to do this analysis live. The delay between seeing the block in the camera vision and recognizing the color was too great, significantly slowing the movement of the robots. Because the color analysis was simply a proof of concept for taking the vitals of a victim, it was eliminated, although only after all of the functioning code was produced. Instead, each target location was given a hardcoded severity rating. Once arriving at a target, the robot would fetch the corresponding rating and continue along with its algorithm, greatly improving the execution speed of the robots.

The hub robot's goal is similar to that of the exploratory. Hub robots rotate between their hub meet up points - predetermined locations where an exploratory robot will attempt to find them. If an exploratory robot is found and information is received, the hub robot will either continue rotating between meet up locations or return to the medic. Hub robots will return to the medic when they have too many victims in their patient list or their patient list has a high severity victim. Upon locating the medic, they will deliver any victim information and pick up any new messages from the medic. Originally the hub robot was to be on top of a drone, however after assessing the cost to configure a drone for our requirements, it was deemed unnecessary. Thus, the hub robot was switched to the same robot type as the exploratory, with their roles staying separate. This still establishes the proof of concept, allowing it to be expanded in the future.

The medic is represented by a device connected to a laptop with an associated UI. This setup enables the medic to perform several key tasks: sending coordinates of new targets and their current location, viewing individual victims, marking victims as treated, and viewing all treated and untreated victims. Additionally, the medic can sort the list of available victims by severity, x coordinate, y coordinate, or overall distance from themselves. When new victims are received, they are populated into the table and sorted according to the chosen method.

Default logic for returning to the medic when the communication device is not functioning and for an automatic initial test of communication devices were excluded from the final design due to time constraints and project priorities. Although these features would not have been overly complex to incorporate, they were ultimately not included.

Functional Description

Below are the details regarding how the device functions and thus achieves the performance objectives listed above. The section is divided into hardware and software decisions.

Hardware

The PCB serves three primary functions: powering the system, managing USB to UART communication, and interfacing with sensors.

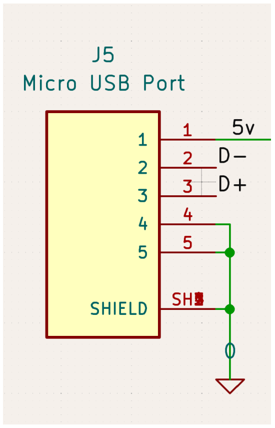


Figure 2. Micro USB Port Schematic

The micro USB port (J5 in Figure 2 above) connects the communication system to the host device, either a robot or computer. This connection enables data transfer and supplies power to the device. The host supplies 5V to the system, which is used to power the USB to UART chip and sensor.

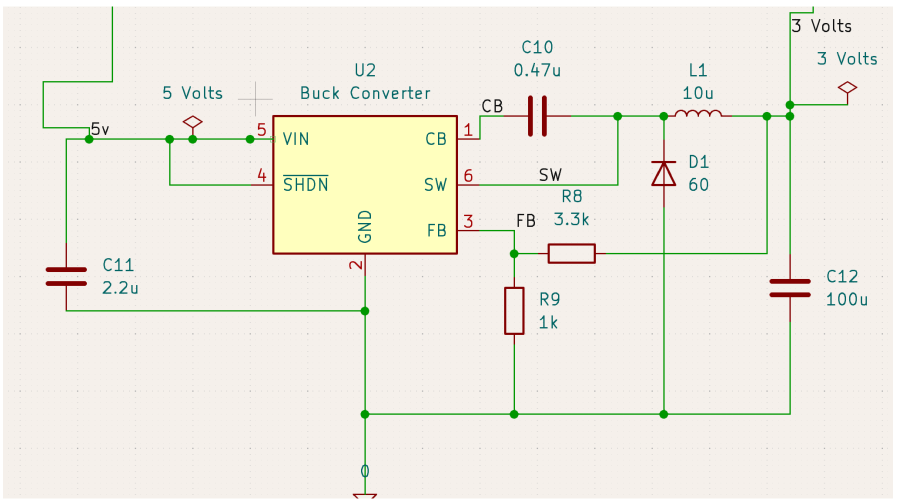


Figure 3. Buck Converter Schematic

That same 5V is also stepped down to 3.3V using a buck converter (U2 in Figure 3 above) to power the XBee microcontroller.

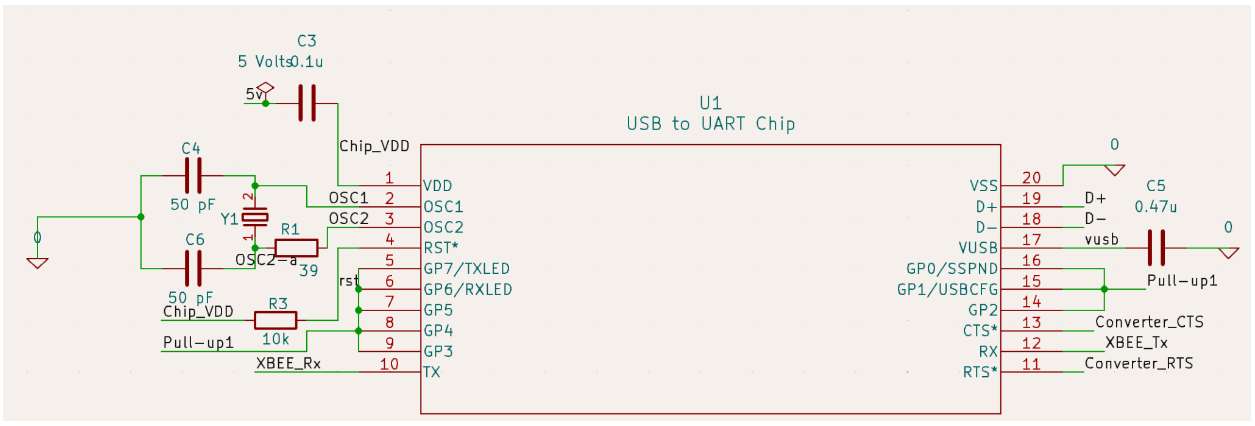


Figure 4. USB to UART Conversion Chip Schematic

Because the XBee microcontroller is incompatible with USB communication protocol, a USB to UART chip (U1 in Figure 4 above) is needed to convert data that is being transmitted to and from the host.

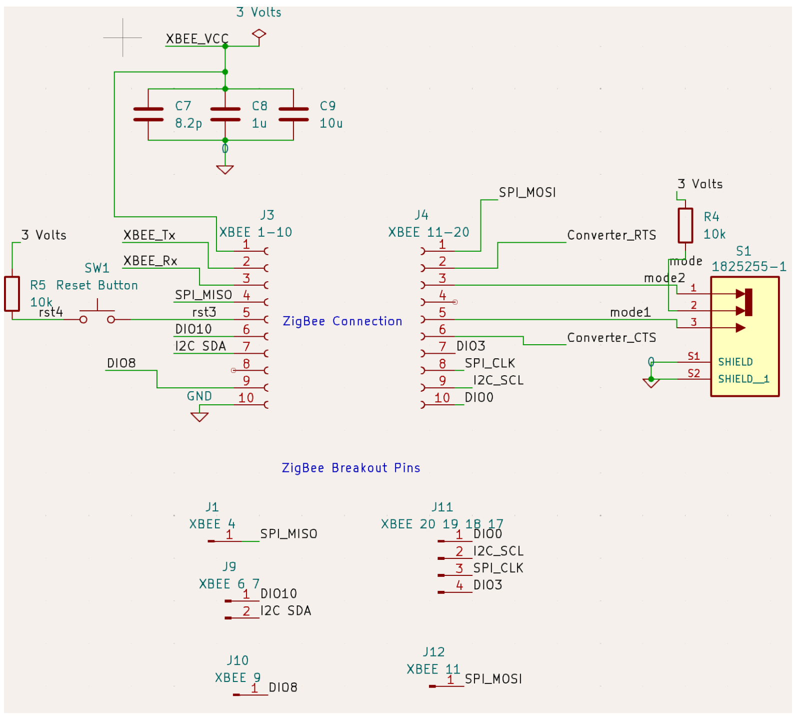


Figure 5. XBee Microcontroller Schematic

The XBee microcontroller connects to the board via female header pins (J3 and J4 in Figure 5 above). Additionally, all unused General Purpose Input/Output (GPIO) and I2C/SPI pins from the XBee microcontroller are connected to breakout pins. This design choice allows for different sensors to be integrated into the system.

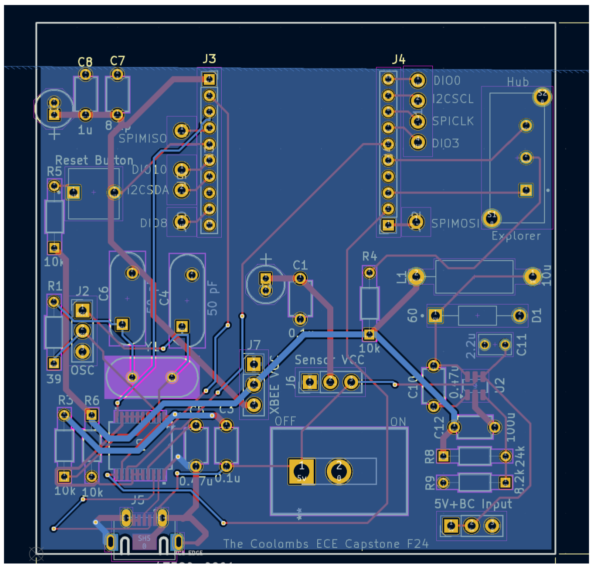


Figure 6. Final PCB Layout

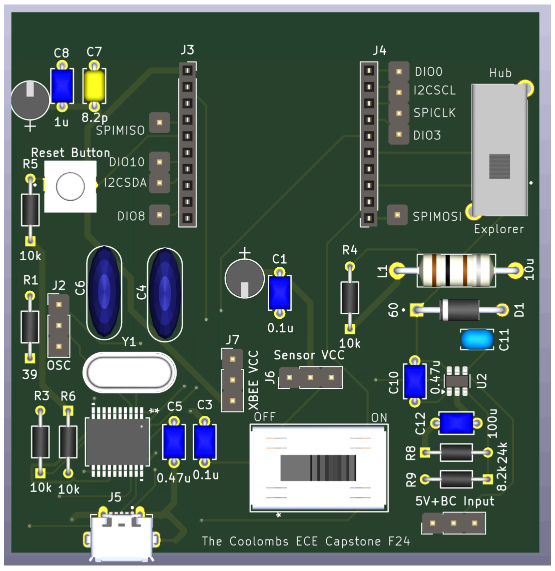


Figure 7. 3D Mock-up

The above figures display the final board layout and 3D model. When the on/off switch is turned on, a 5V supply powers the sensor, USB to UART chip, and buck converter. This then powers the XBee microcontroller, enabling it to transmit data to and from the USB to UART chip, which connects to the USB port and transmits data to and from the host system. The switch to the right of the XBee microcontroller enables the robot to toggle between hub and exploratory modes. The full PCB schematic can be found in Appendix B.

Software

The exploratory robot will have three general states: exploring for victims, searching for the hub robot, and searching for the medic. A decision tree logic, as seen below in Figure 8, was used for the robot to determine what state to proceed to next and what steps it needs to take in order to get there. Victims are assessed at given locations, a change from the initial design of the system, which would originally determine severity based on color of object found. This change is discussed more in Design and Manufacturing Constraints below. An interesting adjustment that we had made in preparation for color recognition was the orientation of the robot when it arrived at the goal. When the severity of the rating was initially based on the object's color, the victim was rated incorrectly by the robot because it arrived at the goal facing the wrong direction. During real exploration this would not be a concern, but due to the configuration of our setup, we had to adjust our code to require the robot to face the correct orientation in addition to going to the proper coordinates.

In order to abstract away the exploration logic, which falls under the authority of a separate team of Systems Engineers, we gave the robots specific locations to navigate to rather than having them search for targets on their own. After the robot either a) locates a victim of high severity (has a rating of 3 or higher) or b) has a victim list that has reached capacity (a length of 4 or more), it will search for the hub at a known meet up location. If the hub is not detected in a certain period of time after arriving at the meet up, the robot will continue on to search for the medic at its stored medic location. Once locating the hub or medic, it will transmit the victims list and return to locating and assessing the remainder of its target list.

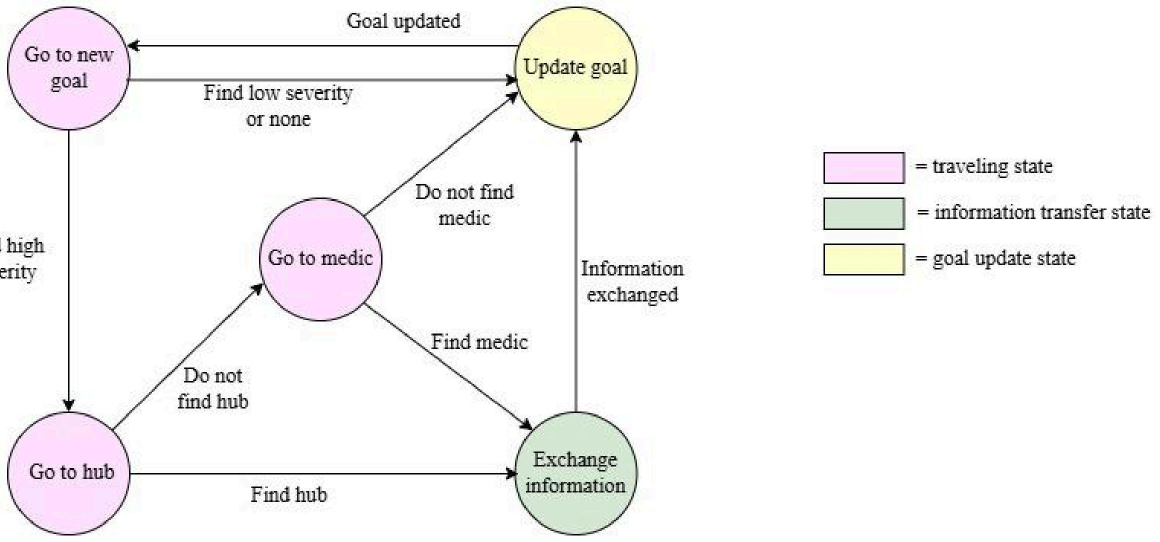


Figure 8. Exploratory Robot Finite State Machine

The hub robot will have two states: traveling to a meet up location, or searching for the medic. The logic can also be seen below in Figure 9. The hub will rotate between two preset hub meet up locations, another simplification in order to abstract away the exploration logic. At each location, the robot will wait for a predetermined period of 20 seconds before leaving for the other meet up location. If an exploratory robot is found at either meet up location, the exploratory robot will transmit its victim list. If the hub's victim list now either a) contains a victim of high severity (has a rating of 3 or more) or b) has reached capacity (a length of 5 or more), it will search for the medic at its known medic location. Once reaching the medic, it will transmit the victims and return to rotating between hub meet up locations.

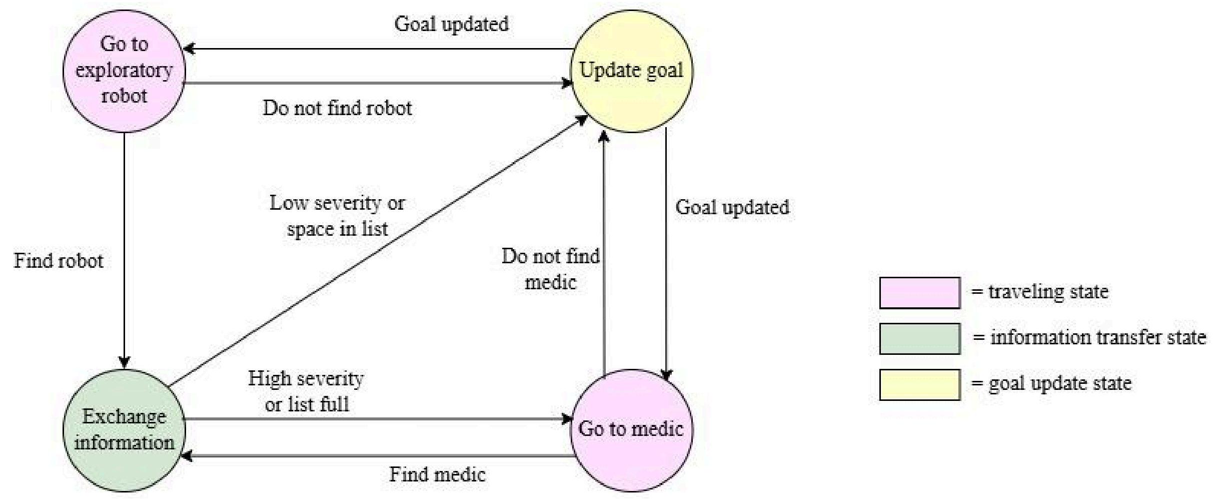


Figure 9. Hub Robot Finite State Machine

All messages sent between the communications devices take the following format

```
{
        "severity": int,
        "x-coordinates": double,
        "y-coordinates": double
}
```

This message format is used to send multiple kinds of messages, beyond just the victim information. The value of severity is varied depending on the meaning of the message. The table below describes the correlation between the severity ratings and their meanings.

Severity Rating	Meaning
1-4	Victim rating
-1	Coordinates are an updated medic location
-2	Robot has arrived hub location
-3	Robot has arrived at medic location
-5	Coordinates are a new goal from the medic

Table 1. Severity Meaning Breakdown

This format is also used when sending and receiving messages within the robot internal system to consolidate the logic for robot decision making. Information from both the robot system and from other robots' communication devices are pertinent to the next decision, which is why the logic is executed in one place.

For navigation and the go-to-goal functionality, we implemented a potential fields algorithm. This is done by having obstacles exert a negative force and the goal exert a positive force on the robot. The robot will then flow between the negative and positive forces to eventually reach its target. The figure below is a photo of the forces at work on our robot from a simulation of the algorithm, where the red line is the negative force, the green line is the positive force, and the light orange line to the left is the resultant movement.

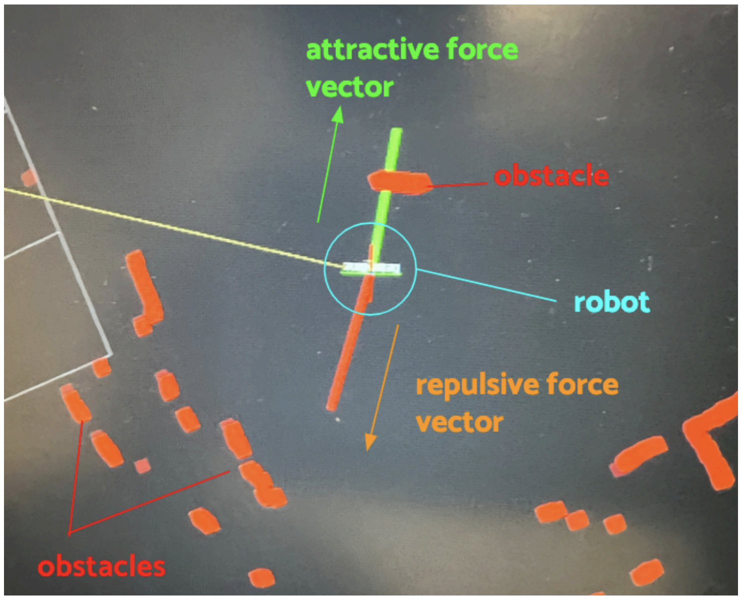
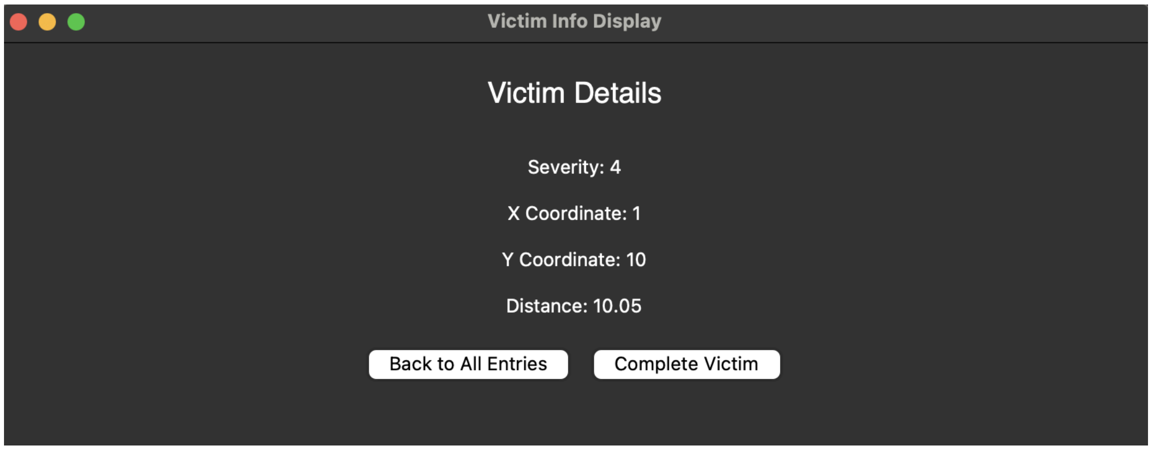


Figure 10. Potential Fields Example

As described above in the project objectives, the medic's UI has multiple functionalities. The figures below are examples of the interface at various stages in the UI usage. The first image below is the page to which the UI opens. It contains the information of the current most urgent victim - if there are multiple victims with that same severity rating, the most urgent victim is then the closest of those with that rating. From here the medic can mark the victim as completed and see a new one, or they can go to the all victim table view.



Victim Details

Severity: 4

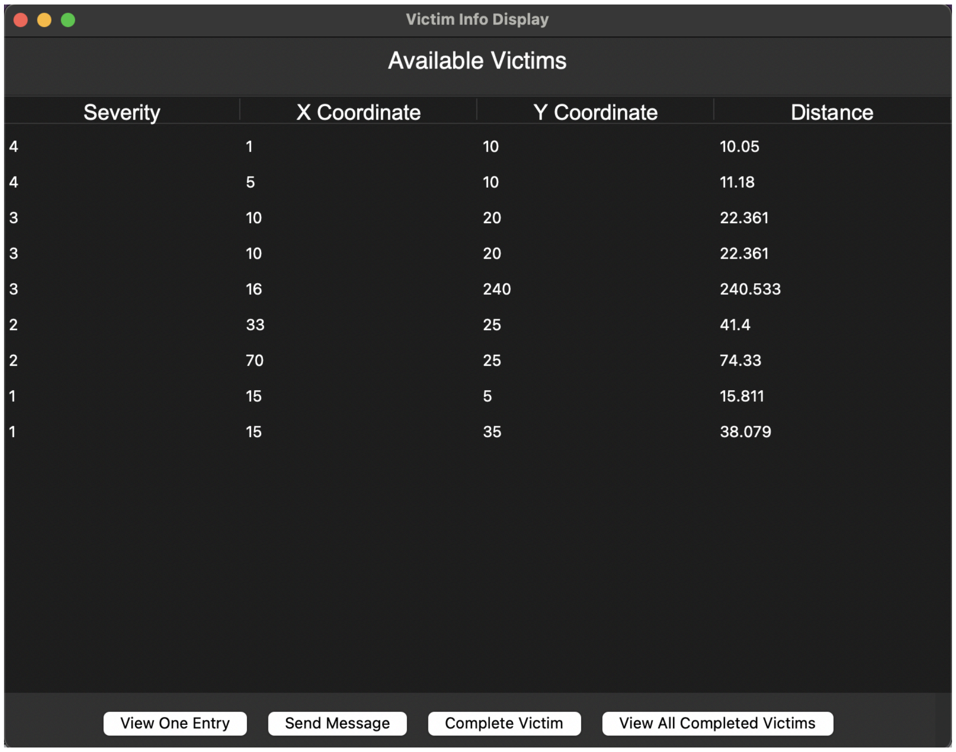
X Coordinate: 1

Y Coordinate: 10

Distance: 10.05

Figure 11. Medic UI Single Victim View

Figure 12 is the list of all victims about which the medic currently has information. The column names are all selectable; if a category name is selected, the table will sort in ascending or descending order of that category. If the category name is clicked once, the table displays the information in descending order, and if you click it again, the table switches to ascending.



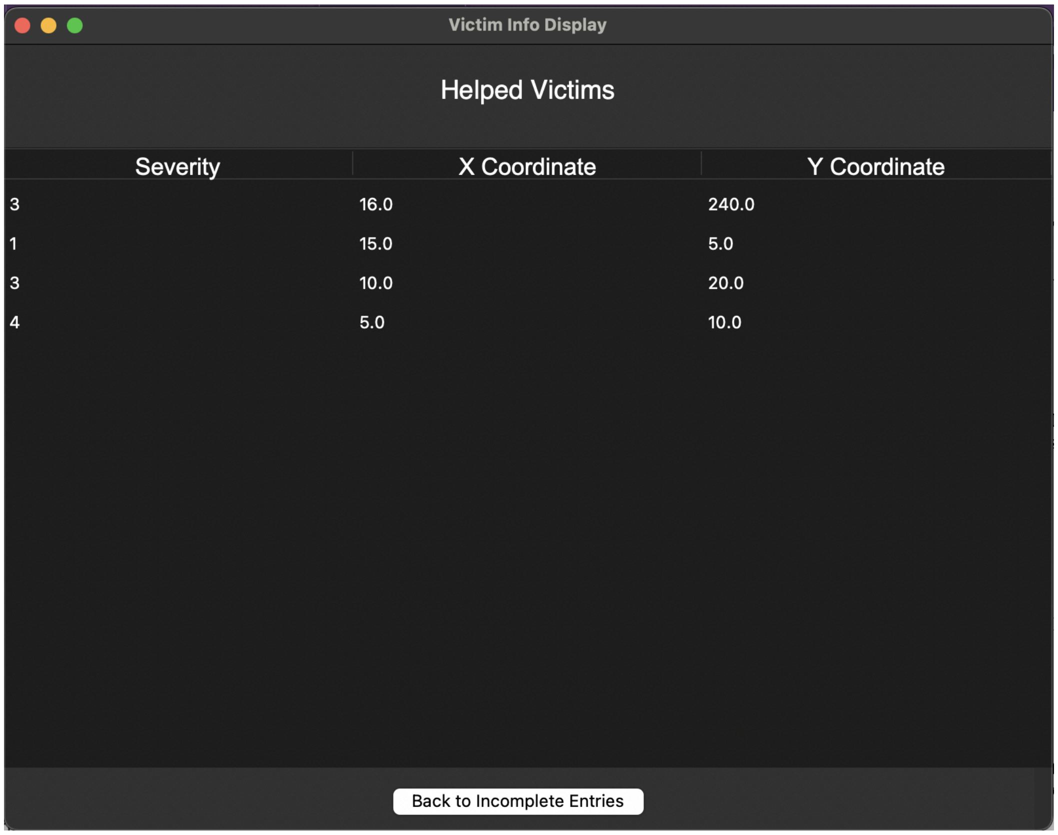
Available Victims

Severity	X Coordinate	Y Coordinate	Distance
4	1	10	10.05
4	5	10	11.18
3	10	20	22.361
3	10	20	22.361
3	16	240	240.533
2	33	25	41.4
2	70	25	74.33
1	15	5	15.811
1	15	35	38.079

Figure 12. Medic UI All Victim Table View

If you select one victim from the table, you can then use the buttons at the bottom of the page to go back to single victim view mode, a less overwhelming view for the medic. The medic is also able to complete a victim from this view.

The final page view is the table of completed or assisted victims. These are all the victims that have been "completed" and are available for review of who has been assisted and where they were, possibly for post catastrophe analysis.


Victim Info Display

Helped Victims

Severity	X Coordinate	Y Coordinate
3	16.0	240.0
1	15.0	5.0
3	10.0	20.0
4	5.0	10.0

Figure 13. Medic UI Completed Victims Table View

Technical Description

Below we break down the technical decision making that went into the construction of the device. Hardware and software sections are divided accordingly.

Hardware

To demonstrate the functionality of our communication system, we utilized two ROSbot 2.0 robots by Husarion - one serves as an exploratory robot to locate and assess patients and the other as a hub robot, routing information between the exploratory robot and the medic. These robots were provided by the AMR lab and were chosen for their integrated LiDAR sensors and compatibility with the lab's Vicon system, which enabled the retrieval of location data. Additionally, the robots were appropriately sized, being small enough to navigate the AMR lab while still large enough to carry the communication system.

The robots' LiDAR systems detect obstacles in their path, while the Vicon motion capture cameras provide precise positional information. Communication between humans and robots is facilitated via a ZigBee communication protocol, selected for its reliability, low-cost, and ability to cover a large communication radius with minimal power consumption. This wireless protocol was particularly suitable for our project, which is intended to be used in high-risk or power-deficient environments. Furthermore, ZigBee's mesh network further enhances efficiency by enabling information to be routed and rerouted through nodes [3].

To implement ZigBee, our team selected the Digi XBee 3 ZigBee 3 RF Module. This module was chosen for its micro-form factor, which minimizes bulk and weight, making it ideal for smaller robots or drones. Additionally, the Digi XBee 3 can be configured for ZigBee using the Digi XCTU software and directly programmed with MicroPython, an implementation of Python 3 designed to run on embedded hardware [4]. These features eliminate the need for a separate microcontroller, simplifying the design while reducing bulk, power consumption, and cost.

Data transmission between components was achieved using UART through the microcontroller pins specified in the Digi XBee® 3 RF Module Hardware Reference Manual [5]. The XBee microcontroller does not accept USB inputs so we utilized a USB to UART converter chip to enable transmission and reception of serial communication data between the robot and communication system. A XBee 3 development board was used to connect the microcontroller to personal laptops for programming. Once the required files were uploaded to and tested on each XBee 3 microcontroller, the devices were connected to a printed circuit board (PCB) for final operation.

The through-hole XBee module was chosen for this project primarily due to its flexibility. This model allows the XBee to be easily attached and re-attached to the board for programming updates and PCB re-design. The alternative surface mount option would require the XBee microcontroller to be permanently connected to the PCB, making the testing and debugging process very difficult.

According to the XBee microcontroller hardware reference manual [5], the power requirements for the module are as follows:

1) Supply voltage range: 2.1V to 3.6V
2) When using a switching regulator, the switching frequency must exceed 500kHz, and the voltage ripple must be below 50mV

The manual also provides the XBee microcontroller pinout description and recommends placing three capacitors of varying values near the VCC connection to reduce noise. Using this information, two GPIO pins, 13 and 15, were chosen to connect to the mode switch, as they were not already being used for communication protocols and were conveniently located for wiring. A reset button was also connected to pin 5, the designated reset pin (see Appendix B for full PCB schematic).

Using the XBee microcontroller's power specifications, a buck converter was designed. The chosen converter, Texas Instruments' LMR14206, was selected for its high output current capacity, wide input/output voltage range, high switching frequency, compact size, and cost-effectiveness [6].
The input and bootstrap capacitor values were provided by the datasheet, while the resistors were calculated using the following equation from the datasheet:

$$R_1 = R_2 * \left(\frac{V_{out}}{0.765} - 1\right)$$

Initially, the resistor values selected were 24k and 8.2k. However, to reduce noise, these values were adjusted. Setting R1 to 1kΩ resulted in R2 being 3313Ω, which was rounded down to the practical value of 3.3kΩ. Following the buck converter datasheet instructions, the ripple current was set to be 30% of the DC output current (600mA), yielding a ripple current of 0.18A. Using a 12V maximum input voltage and a 3.3V output, the inductor value was calculated with:

$$L = \frac{V_{in} - V_{out}}{V_{in} * I_{ripple} * F_{SW}}$$

Using the numbers provided above, a 10μH inductor was calculated. The output capacitor was determined based on a maximum allowable voltage ripple. Assuming an Effective Series Resistance (ESR) of 0.1 for a 100μF capacitor, the voltage ripple was calculated with:

$$V_{ripple} = I_{ripple} * \left(ESR + \frac{1}{8 * F_{SW} * C_{out}}\right)$$

At a switching frequency (F_{SW}) of 1.25MHz, this yielded a theoretical ripple of 2mV. As the capacitor datasheet did not list the ESR value, it was found experimentally. It was then discovered that the purchased capacitors had an ESR of 0.3, resulting in a ripple of 54mV, exceeding the maximum of 50mV. Fortunately, the bypass capacitors at XBee VCC helped to reduce this ripple to approximately 40mV. Lastly, a 60V 1A diode was selected was chosen using the formula:

$$D_{voltage} = 0.25 * V_{in} + V_{in}$$

The micro USB port was chosen for its simplicity, ease of integration, and extra shield pins for stability [7]. The USB to UART protocol converter chip enables communication between the host device and the XBee microcontroller. Powered by the USB bus (5V), this chip was selected for its compatibility with the board design. Reset functionality is managed via a power-on reset configuration, initializing the chip whenever power is supplied. A 12MHz crystal oscillator provides an internal clock for the chip, ensuring proper detection by the host device.

An unexpected problem with the PCB design arose due to the XBee microcontroller ground pin being assigned to 'GND' instead of 0, causing it to be in a different net. Thus, when the XBee microcontroller was plugged into the system, it did not function as expected. To remedy this, this pin was shorted to a nearby ground pin. The figure below illustrates this problem; the nets are shown in green boxes.

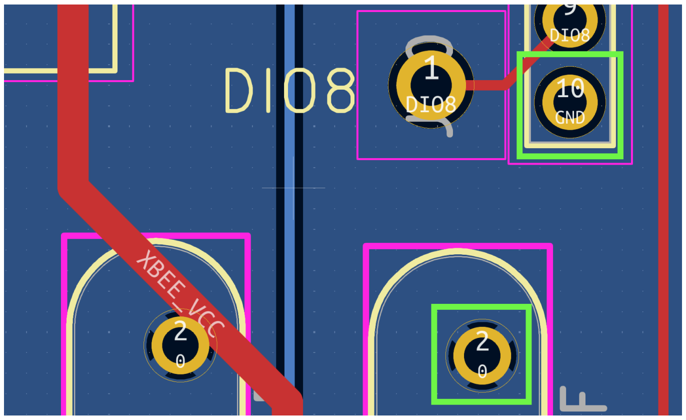


Figure 14. XBee Microcontroller Ground Pin and Board Ground Discrepancy

Another issue occurred with the USB to UART chip, where the bypass capacitor was mistakenly connected in series with the VCC pin instead of being connected to ground (see Figure 15 below). This was resolved by reconnecting the capacitor to a nearby ground net and shorting the 5V connection to the chip VDD pin, enabling the USB to UART chip to be powered correctly. This new connection is shown in Figure 16 below.

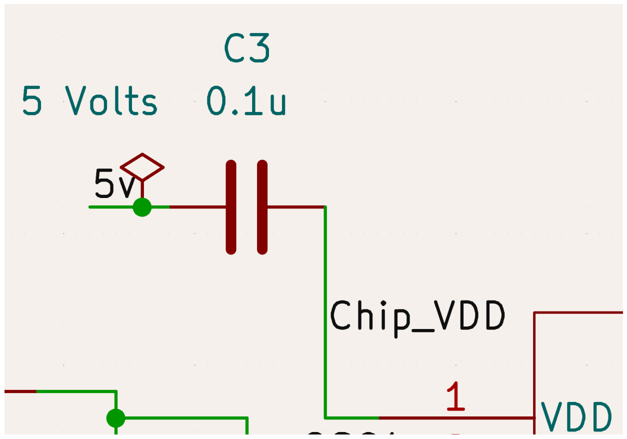


Figure 15. Incorrect Bypass Connection

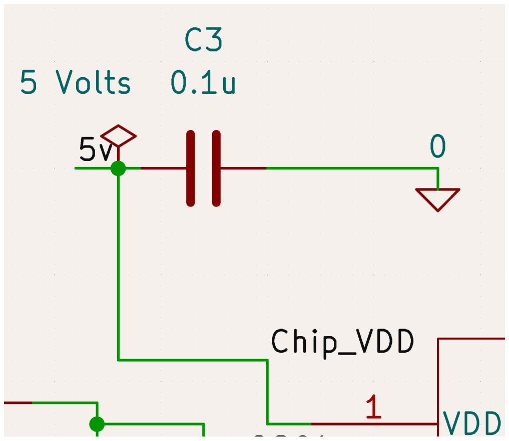


Figure 16. Corrected Bypass Connection

The Adafruit BME680 temperature, humidity, pressure and gas sensor was selected for its ability to measure environmental parameters, its flexibility, and overall low cost. This data is then used as an additional metric for determining severity of a victim in the field. The Adafruit BME680 is a development board that incorporates a Bosch surface mount sensor with a voltage regulator, ports for a STEMMA QT connection, and through holes for I2C and SPI communication protocols [8]. STEMMA QT is a special four wire cable that enables I2C communication; it was used to initially burn the boards. The header pins were later soldered in to create a secure connection between the sensor and the PCB breakout pins. Finally, in order to set up the sensor and correctly collect environmental data, the MicroPython code from a git repository called “Pure MicroPython Bosch BME680 sensor driver” from GitHub user Salvatore Sanfilippo (username - ‘antirez’) was repurposed [9].

There are 2 PCB and sensor holders, with slightly different dimensions but both suitable for the XBee microcontroller and sensor system. The dimensions of the final holder are 97.5x73.0x20.0 mm. The first version of the final holder had no excess space for the microUSB port and cable, so in the second design, the opening was widened. The first version also fit tightly with PCB due to tolerances with 3D printing, and a corner of the XBee microcontroller made contact with the wall. While this was not a fatal error, in the final design, the height of that wall was dropped and the inner cavity was slightly widened. Despite the snug fit with the PCB and sensor, both prototypes were functional and were used for full system testing and the final demonstration.

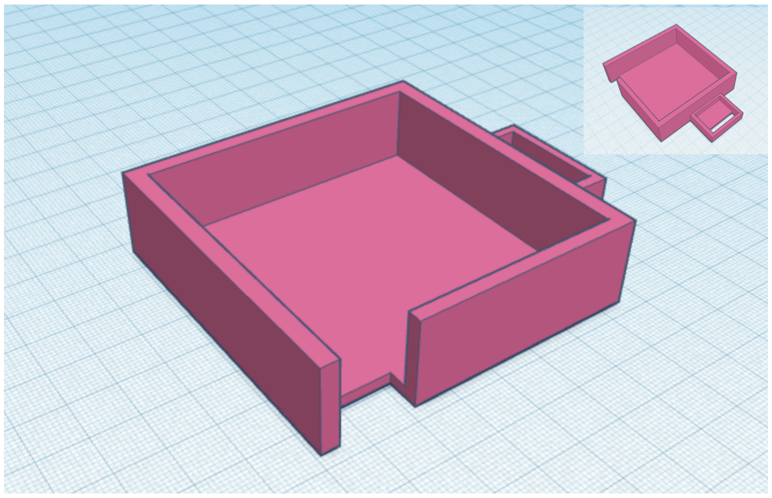


Figure 17. First Edition of PCB Holder

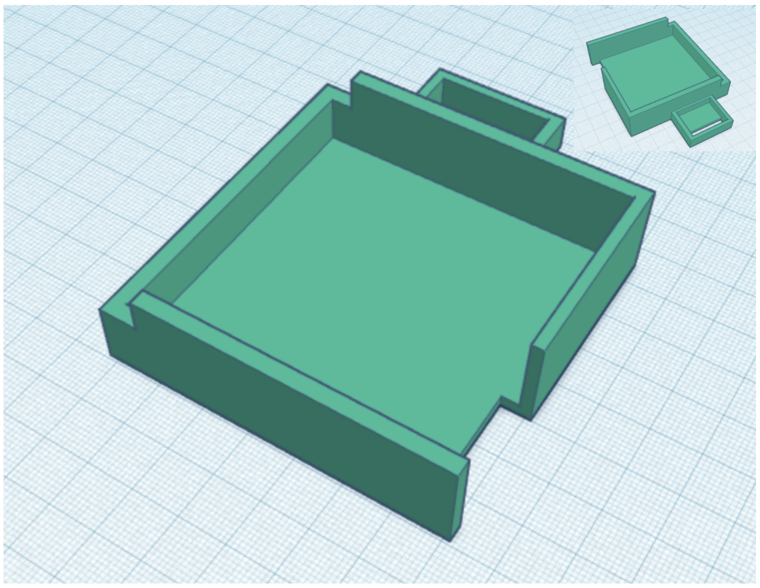


Figure 18. Second Edition of PCB Holder

Software

The Robot Operating System (ROS) was used to program the ROSbots from the AMR Lab. ROS was selected not only for its compatibility with the provided robots but also for its flexibility, code reusability, extensive pre-existing libraries, and open-source software [10]. To write and interface code with ROS topics, services, and parameters, we used RosPy, a Python client library for ROS [11]. Python was chosen for its ease of use and team member familiarity. Additionally, Python code could be easily translated into MicroPython, which was utilized to program our XBee 3 microcontroller.

To implement the robot's go-to-goal functionality, we designed an artificial potential fields algorithm within ROS. This algorithm was selected due to its compatibility with the ROSbot's LiDAR sensor, enabling the robot to navigate around obstacles while moving to its designated goal. The potential fields algorithm subscribed to a ROS node linked to the robot's LiDAR to detect objects and measure distances. Additionally, it subscribed to a Vicon node to obtain positional data for both the robot and its target destination. Vicon consists of 8 cameras positioned around the upper corners of the AMR lab. With these cameras, in combination with a complex and intelligent location processor, Vicon is capable of accurately tracking and publishing the location of multiple items in their field of vision.

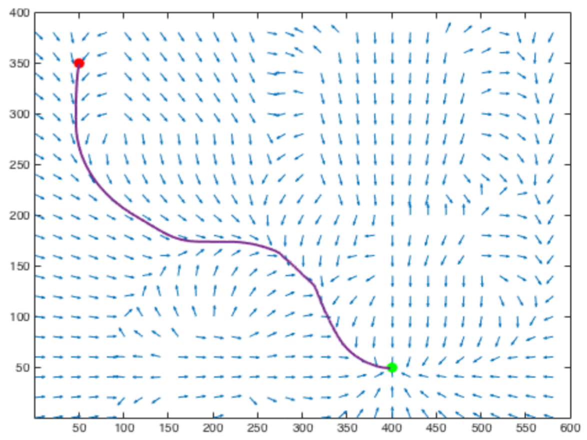
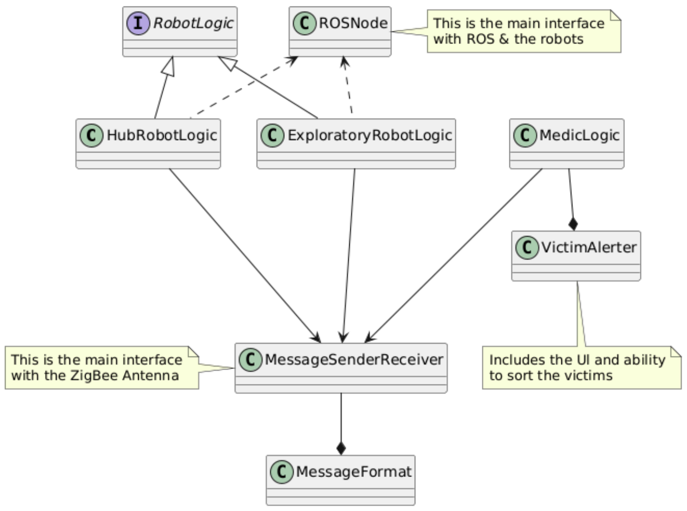


Figure 19. Potential Fields Example

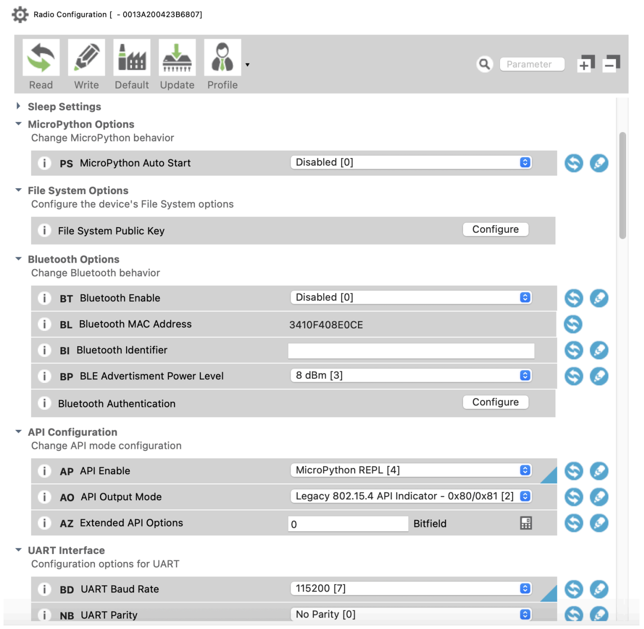
For our code architecture, the project was organized into modular Python files and classes, each focusing on specific functionalities, such as exploratory and hub robot logic, message transmission and reception, medic logic and user interface, and robot interaction. This separation of roles and code decoupling was selected because it enables functional independence of the code, enhancing the refactoring, testing, and code expansion process. At the core of this structure was a Robot Logic interface, which organized and compiled message interpretation, command transmission to robots, and interaction between almost all classes. To differentiate the functionality between the hub and exploratory robots, we developed two specialized classes, inheriting from the Robot Logic interface, both dependent on a ROSNode class. Additional classes were implemented for ROSNode, Medic Logic, and a MessageSenderReceiver, which enabled communication with the ROSBot, presented information to medics, and wirelessly transmitted and received messages, respectively. The interaction of all of these classes can be seen in Figure 20 below. After comprehensive testing with Pytest and the XBee 3 microcontroller development board, the Python code was adapted to be compatible with XBee's MicroPython environment. Using the mpy-cross compiler utility to save memory on the microcontroller, the code was pre-compiled into MicroPython files and subsequently uploaded to the XBee 3's Files Systems Manager for further testing.


RobotLogic
ROSNode
This is the main interface with ROS & the robots
HubRobotLogic
ExploratoryRobotLogic
MedicLogic
VictimAlerter
This is the main interface with the ZigBee Antenna
MessageSenderReceiver
Includes the UI and ability to sort the victims
MessageFormat
Figure 20. Code UML Diagram

XCTU

To configure, upload, and test code on the XBee 3 microcontroller, we relied on Digi's XCTU software. This tool, provided alongside the microcontroller, facilitated device configuration and the establishment of a ZigBee mesh network. Digi XCTU also allowed us to run code directly in the XBee 3's MicroPython terminal, a capability especially useful during debugging.

To put code onto the XBee microcontroller and run any code, we used a development board and later our PCB [12]. The Radio Configuration (shown in Figure 21), is intense as it requires setting pins and enabling and disabling various parameters for I2C or UART connection. To access the MicroPython terminal, we enabled MicroPython REPL in the API Enable section. To enable the mesh network, we verified that all ZigBee firmware and functions were updated to XBee 3 ZigBee. For a more thorough view of all the settings, see Appendix D.


Radio Configuration [- 0013A200423B6807]

Sleep Settings

MicroPython Options
Change MicroPython behavior

PS	MicroPython Auto Start	Disabled [0]

File System Options
Configure the device's File System options

File System Public Key	Configure

Bluetooth Options
Change Bluetooth behavior

BT	Bluetooth Enable	Disabled [0]
BL	Bluetooth MAC Address	3410F408E0CE
BI	Bluetooth Identifier	
BP	BLE Advertisment Power Level	8 dBm [3]
	Bluetooth Authentication	Configure

API Configuration
Change API mode configuration

AP	API Enable	MicroPython REPL [4]
AO	API Output Mode	Legacy 802.15.4 API Indicator - 0x80/0x81 [2]
AZ	Extended API Options	0 Bitfield

UART Interface
Configuration options for UART

BD	UART Baud Rate	115200 [7]
NB	UART Parity	No Parity [0]

Figure 21. MicroPython REPL and UART Baud Rate Setting in XCTU

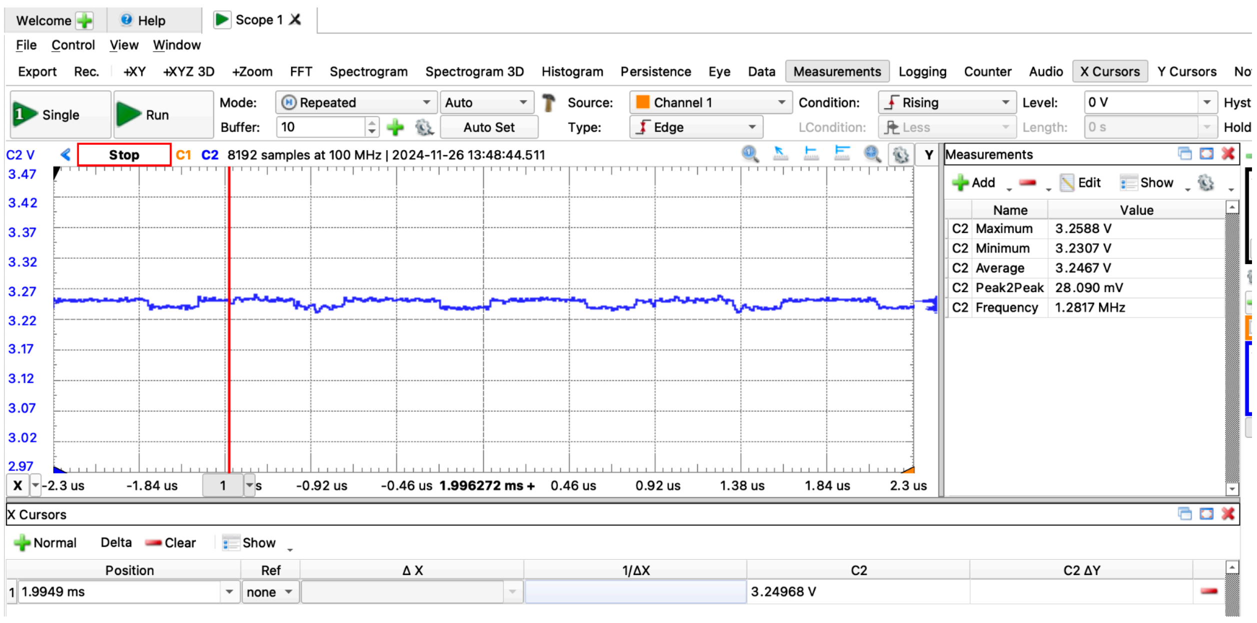
Test Plan

The testing of the device was separated into three steps: the board testing, the software testing, and finally, the full system testing. The division of testing helped maximize the functionality of each component before conducting the full system testing. This allowed us to more quickly narrow down problems to specific parts of the system.

Hardware

To thoroughly verify the hardware functionality, the PCB was split into 3 subsections. Each section was initially tested in isolation.

The first subsection was the buck converter. To set up testing, an AD2 was used to supply power and measure output waveforms. The supplies were set to 5V, and the board ground plane was connected to AD2 ground to establish a common ground. The 3 main criteria used to verify the buck converter function were a measured average voltage of 3V, a voltage ripple under 50mV, and a switching frequency of 1.25MHz. A measurement channel was connected to the output node of the buck converter and the oscilloscope was used to take the measurements. The oscilloscope's measurement tool was used to record the peak-to-peak voltage (voltage ripple), average voltage, and frequency. This is shown in the measurement tab in Figure 22. Verifying the voltage ripple and frequency for the buck converter quickly proved difficult. The voltage ripple was consistently higher than the 50mV max, and the frequency was lower than the expected 1.25MHz. To combat these issues additional output capacitors were added in parallel to help decrease the ripple. The original resistors were also switched out for ones with lower resistance values to help lower noise. The experimental verification of the buck converter is shown in the waveform image below (Figure 22).



Mode: Repeated Auto
Buffer: 10
Source: Channel 1
Type: Edge
Condition: Rising
LCondition: Less
Level: 0 V
Length: 0 s

8192 samples at 100 MHz | 2024-11-26 13:48:44.511

Measurements

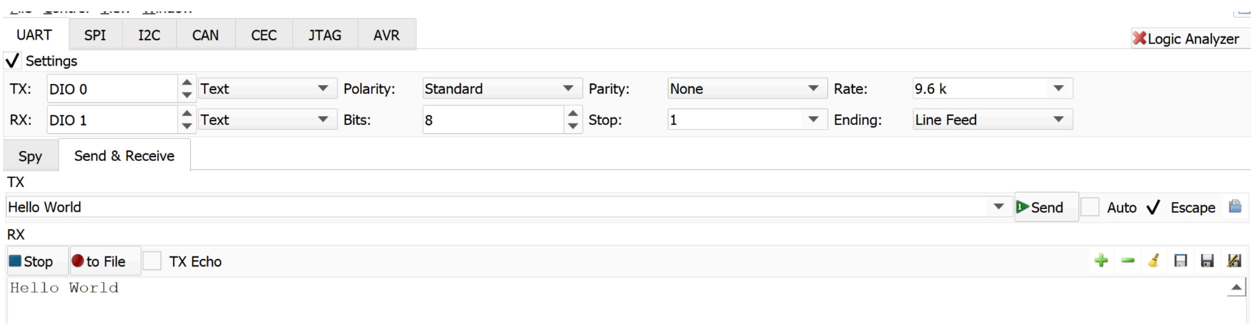
Name	Value
C2 Maximum	3.2588 V
C2 Minimum	3.2307 V
C2 Average	3.2467 V
C2 Peak2Peak	28.090 mV
C2 Frequency	1.2817 MHz

X Cursors

	Position	Ref	Δ X	1/ΔX	C2	C2 ΔY
1	1.9949 ms	none			3.24968 V	

Figure 22. Buck Converter Waveform Verification

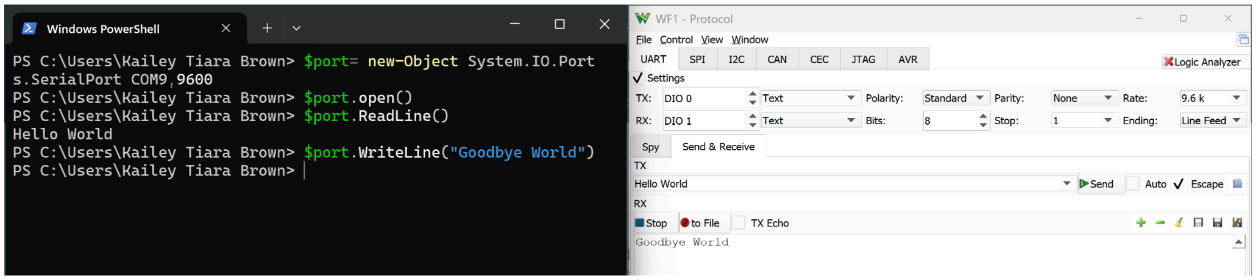
The second subsystem was the USB to UART chip. To set up testing, the AD2 was used to simulate UART communication. A loopback test was conducted to verify that the Tx and Rx channels were wired correctly. To do this a message was sent through the Tx channel and received in the Rx channel without connecting the board to a computer. This result is shown in Figure 23.



```
TX: DIO 0   Text   Polarity: Standard   Parity: None   Rate: 9.6 k
RX: DIO 1   Text   Bits: 8              Stop: 1        Ending: Line Feed
TX
Hello World
RX
Hello World
```

Figure 23. Loopback Test Result

To verify that messages could be successfully sent through the USB to UART chip, the Tx and Rx XBee sockets were connected to the Tx and Rx channels of the AD2. When the board is connected to a computer via the micro USB port, the computer is supposed to recognize the connection and create a virtual port. This was not successful during initial testing and an oscillator needed to be added to the USB to UART chip to help establish a clock signal within the chip. This modification allowed the computer to recognize the board and create a virtual port when testing the second iteration of the PCB. Using the computer terminal, the virtual port was configured to allow messages to be written and read. Messages were successfully sent and read from the terminal to WaveForms, verifying the functionality of the chip (verification shown in Figure 24). Lastly, the chip was configured with a baud rate of 9600 using the downloaded configuration utility.

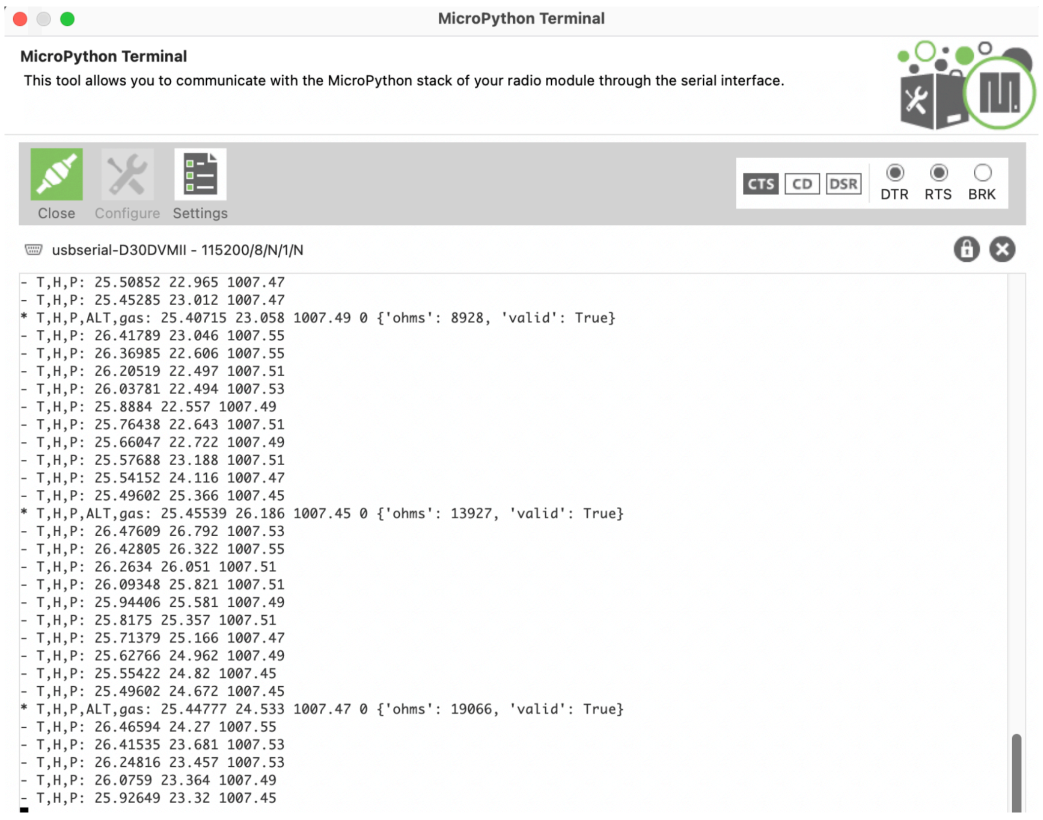


```
PS C:\Users\Kailey Tiara Brown> $port= new-Object System.IO.Port
s.SerialPort COM9,9600
PS C:\Users\Kailey Tiara Brown> $port.open()
PS C:\Users\Kailey Tiara Brown> $port.ReadLine()
Hello World
PS C:\Users\Kailey Tiara Brown> $port.WriteLine("Goodbye World")
PS C:\Users\Kailey Tiara Brown>
```

```
TX: DIO 0   Text   Polarity: Standard   Parity: None   Rate: 9.6 k
RX: DIO 1   Text   Bits: 8              Stop: 1        Ending: Line Feed
TX
Hello World
RX
Goodbye World
```

Figure 24. USB to UART Chip Verification

The third subsystem was the sensor. We first had to burn the sensor before integrating it with our system, a process that included running the sensor for a total of 48 hours. The sensor was then connected to the XBee microcontroller via the development boards. Using the MicroPython terminal in the XCTU software, data was read from the sensor and displayed. A can of compressed air was used to verify the sensor's ability to detect harmful gasses. This process verified the sensor function as well as the XBee microcontroller's ability to interface with the sensor. Within the MicroPython terminal, the XBee reads the correct address of the sensor, and can access the data that the sensor collects and sends it back to the computer.


MicroPython Terminal
MicroPython Terminal
This tool allows you to communicate with the MicroPython stack of your radio module through the serial interface.
Close
Configure
Settings
CTS
CD
DSR
DTR
RTS
BRK
usbserial-D30DVMII - 115200/8/N/1/N

```
- T,H,P: 25.50852 22.965 1007.47
- T,H,P: 25.45285 23.012 1007.47
* T,H,P,ALT,gas: 25.40715 23.058 1007.49 0 {'ohms': 8928, 'valid': True}
- T,H,P: 26.41789 23.046 1007.55
- T,H,P: 26.36985 22.606 1007.55
- T,H,P: 26.20519 22.497 1007.51
- T,H,P: 26.03781 22.494 1007.53
- T,H,P: 25.8884 22.557 1007.49
- T,H,P: 25.76438 22.643 1007.51
- T,H,P: 25.66047 22.722 1007.49
- T,H,P: 25.57688 23.188 1007.51
- T,H,P: 25.54152 24.116 1007.47
- T,H,P: 25.49602 25.366 1007.45
* T,H,P,ALT,gas: 25.45539 26.186 1007.45 0 {'ohms': 13927, 'valid': True}
- T,H,P: 26.47609 26.792 1007.53
- T,H,P: 26.42805 26.322 1007.55
- T,H,P: 26.2634 26.051 1007.51
- T,H,P: 26.09348 25.821 1007.51
- T,H,P: 25.94406 25.581 1007.49
- T,H,P: 25.8175 25.357 1007.51
- T,H,P: 25.71379 25.166 1007.47
- T,H,P: 25.62766 24.962 1007.49
- T,H,P: 25.55422 24.82 1007.45
- T,H,P: 25.49602 24.672 1007.45
* T,H,P,ALT,gas: 25.44777 24.533 1007.47 0 {'ohms': 19066, 'valid': True}
- T,H,P: 26.46594 24.27 1007.55
- T,H,P: 26.41535 23.681 1007.53
- T,H,P: 26.24816 23.457 1007.53
- T,H,P: 26.0759 23.364 1007.49
- T,H,P: 25.92649 23.32 1007.45
```

Figure 25. Verification of XBee Microcontroller and Sensor Interaction

After verifying these 3 subsystems, full system testing was conducted. The step-by-step test plan used for the PCB verification can be found in Appendix C.

Software

The software testing was first completed using Pytest. Pytest is a python framework that enables small, scalable testing with function and class stubbing capabilities [13]. This was used to test each individual function within every class. Each function was tested with a variety of possible inputs; dependencies on other classes, packages, or functions were stubbed, which ensures that only the desired function is being tested. This testing allowed us to debug any individual functions. After this stage was complete, we continued on to complete integration testing with the XBee microcontrollers. The XBee antennas were tested via a development board connected to the laptop. The code was run from our laptops and transmitted to the XBee microcontrollers via USB-C connection. These tests ensured the interactions between the classes and functions were working properly. At this same level, we tested the user interaction with the medic UI - this involved both valid and invalid UI interactions to identify and eliminate potential bugs. Throughout these two stages, no major redesigns were needed as a result of found bugs.

Finally, the last component of software testing was uploading the code onto the XBee microcontrollers and testing their interactions without manual triggering from a laptop. This step identified issues with the MicroPython transferability of our code and tested all automatically triggered scripts. Numerous changes were made so that the code was compatible to run on the XBee microcontrollers themselves. First, the XBee microcontrollers needed to be reconfigured to run in MicroPython mode rather than Application Programming Interface (API) mode, which was used to test the initial Python code. When transitioning from Python to MicroPython, XBee devices no longer needed to be initialized, opened or closed. We also had to replace the send_data() and read_data() methods from the XBee Python library with the MicroPython methods transmit() and receive(). Additional changes included adjusting the UART technique, as the expected class was not available on the microcontroller. Instead of “import UART from machine”, we used the system stdin and stdout, the Digi recommended UART interface. While making the code a little more complicated, it ultimately solved our problem, enabling the essential UART connection that we needed for success.

Full System

Full system testing was the culminating test of our entire device and focused on the interaction between the PCB, the new sensor, the XBee antenna and microcontroller, and the robot itself. Our testing procedure was to first test that the boards could send a message independent of a XCTU interaction. Upon this initial test, some major bugs were found with the power supply into the XBee 3 itself. After various tests, we determined that there was an issue with the Xbee’s ground, resulting in the XBee microcontroller not receiving the power it needed to function. In order to fix this we shorted the Xbee’s ground over to another nearby ground source. In the second stage, we assessed the UART capabilities of the PCB by sending messages from the PCB and reading them through the terminal. After solving the UART import issues during software testing, as described above, this step went quickly. The third stage was connecting the devices to both the robots and to the laptop. We ensured that both devices were acknowledging each other and receiving the desired messages. Once confirmed, we were able to continue with scenario testing.

Several combinations of potential scenarios were constructed, each with a main actor as the focus of the test: either the exploratory robot, the hub robot, or the medic. Each test had a combination of finding multiple low level victims, finding an urgent victim, and either finding or not finding another robot to communicate this information with. This effectively tested all routes through the decision tree, ensuring that all potential code outputs were properly carried out by the robot and antenna. This testing process was slow and tiresome. One issue that we encountered was the variability in the UART transmission from the PCB. A pain point of these problems was discovered to be our print statements within the microcontroller code. Because we are using stdout for UART transmission and all print statements are also sent to stdout, several bugs occurred when the variable messages were received by the robots. Additionally, every minor change in the code would require it to be converted to MicroPython then uploaded to the microcontroller, a process taking about 60 seconds. While this does not sound like a long time, after adjusting several changes, the time quickly accumulated. Another issue we faced was the functionality of the robots that we were using. Specifically, one robot that we had been using stopped functioning on the final day of testing, requiring us to shift the entirety of the code to a new robot. This new robot also required setup and was dealing with its own internal issues that we then had to fix. Ultimately, the full system was proved functional.

Physical Constraints

In the following sections, we break down the design and manufacturing constraints, the tools we utilized, cost constraints and their effects on the prototype and a production version, and any considerations that would be necessary for a production version of the design.

Design and Manufacturing Constraints

We faced the following design constraints for our project:

1) The battery life of the ROSbot
2) Designing a stable power supply
3) Ensuring signal integrity on the PCB
4) Robot size, thus affecting the size and shape of the PCB
5) Computational capability of the ROSbot onboard system

The battery life of the ROSbot is extremely short. So, to avoid draining the robot’s battery life, our XBee microcontroller and PCB design had to consume as little power as possible. At the end of the

PCB design process, we dealt with power supply and signal integrity issues, where the power supply coming out of the buck converter was unstable in terms of voltage ripple and frequency. We solved this by decreasing the voltage divider resistor values, reducing noise in the signal output and increasing stability and robustness of the output waveform. Lastly, we had to design the PCB and the PCB holder around the dimensions of the available flat surfaces on the ROSbot, the height of the central lidar, and the various Vicon triangulation markers on the robot.

We dealt with multiple manufacturing delays and setbacks over the course of the project. These constraints included shipping timelines set by PCB manufacturers and component companies, as well as the turn around time of the electronics company who soldered our surface mount components. The main limitation for our project was the shipping delays and inconsistent communication from PCB manufacturers. Even when we paid for expedited shipping, manufacturing the boards took much longer than expected. Additionally, they did not inform us when the boards had shipped, requiring us to consistently follow up for status updates. With a short timeline, it was often necessary to get the PCBs back within a week of submitting the design, and these delays resulted in a slow down in our testing and a decrease in our available budget as we ultimately paid for faster shipping to minimize the delay we experienced.

An additional issue that was discovered, as discussed above, was the computational capacity and speed of the onboard ROSbot microcontroller. After designing and perfecting the block color recognition code, it was found to be functionally too slow when implemented only on the ROSbot's internal microcontroller. There was a greater than 60 second delay between arriving at a location and finally seeing and recognizing the color of the block. For demonstration purposes, this was too great of a slow down, so the complexity was extrapolated away by assigning severity levels to each target location. Additionally, issues arose from the differences between MicroPython and Python's capabilities. The documentation online is extremely variable about what is possible in MicroPython, ultimately leading to many instances of debugging and needing to generate alternative implementations of a certain piece of code.

Tools Utilized

Multiple tools were utilized over the course of the project, each solving a different problem and assisting with a separate part of the project.

For the preliminary design of the buck converter, we utilized the TI calculator available on the TI site for voltage regulator design. For final buck converter calculations, we used Python in Visual Studio Code (VSCode). A script was written that calculated the component values of the inductor, capacitors, and resistors based on input and output voltage values and other parameters (ripple current, voltage ripple, ESR, etc). PSpice for TI software was used to simulate the buck converter before physical implementation [14].

We heavily utilized KiCad for its schematic editor and PCB editor. We designed all components of our original circuit schematic in KiCad, and used the available components to finalize our PCB for ease of designing footprints and simulation. As we were previously only familiar with Multisim and Ultiboard, using KiCad and all of its features was a learning experience, but ultimately the software was intuitive, smooth, and worked well for our needs.

We also used the Digi XCTU IDE software to connect to and configure the XBee microcontroller and antenna (set the baud rate, upload and test code), and utilized the MicroPython terminal within XCTU to verify that the sensor (connected by I2C) successfully outputted temperature, humidity, and gas information. This IDE was the only free application that offered all of the necessary tools for ZigBee setup. Learning to use and work with XCTU was difficult, since it is a relatively slow and outdated piece of software, but it was the best option for our needs. To construct our code, we had to understand the syntax for MicroPython, learn how to convert Python to MicroPython, and had to be aware of MicroPython's and the XBee microcontroller's computation constraints. Digi MicroPython documentation was often consulted, and mpy-cross compiler utility converted Python to MicroPython while saving memory.

The group utilized GitHub as an easy means of sharing and storing code. This was extremely helpful over the course of the project, as multiple people were working on the software at a given time. Some of the best features of GitHub include branching, which allows different users to have a separate version of the code that they are working on. This lowers the risk of destroying the source, or main, branch and also eases the incorporation of new code into another person's code base. We also found the ability to view past versions imperative, as occasionally some code would be overwritten or removed by another user.

To configure the USB to UART conversion chip with the needed baud rate, a configuration utility was downloaded from the microchip technology website [15] . This again was the best choice for our needs because it had all of the desired capabilities, as it was made by the manufacturer.

We utilized the open source Robot Operating System (ROS) to access robot sensor data, control motion, and determine positioning. Employing ROS required learning its architecture, including the ROS Master, ROS Messages, message buses referred to as ROS Topics, and ROS Nodes that can subscribe and publish to these topics. Additionally, we had to configure the network to link our code with the robot and learn the ROS specific command line tools, such as roslaunch, which enables the execution of packages.

Lastly, we used TinkerCAD to design a custom holder to attach the PCB and sensor on top of the robot. Due to constraints with MacOS, TinkerCAD was the strongest CAD option. It provided a simple, online interface for designing the custom PCB holder according to robot and PCB dimensions. It also allowed the files to be properly exported for printing on the 3D printers in the UVA Shannon Library Makerspace and National Instruments (NI) Lounge.

Cost Constraints

Despite our best efforts, delays and unpredicted costs arose. Some of the cost constraints that affected the price of our final prototype, along with the projected price of a production version, were the cost of the XBee microcontrollers, the price of the PCBs and components, and the price to solder smaller surface mount components, mainly our voltage regulator, UART chip, and microUSB port. The XBee modules were about $23 each, but after purchasing multiple, these began to eat into the budget. While we ordered the first iteration of our PCBs from OSHPark, which was a relatively inexpensive company, we later switched to Advanced PCB, as it is a more reliable vendor for PCB manufacturing. This reliability came with higher associated costs, especially for shipping. Shipping accounted for a significant portion of our budget. We also had to account for PCB component prices, along with the cost to solder three surface mount components per board. While we had originally hoped to solder all components ourselves, upon arrival we realized the surface mount parts were too small to be done by anyone without professional experience and equipment. Outsourcing the surface mount components was relatively inexpensive, as were many of the components, but when producing a small quantity of boards, the unit price per board added up between the cost of manufacturing, shipping, and soldering.

Ideally for a production version the budget would not be so tight. As we describe in the following section, several improvements would need to be made on the board in order for it to be ready for manufacturing, including increasing the quality of materials and purchasing a higher quality Xbee. In the Costs section later in the report we describe the project cost of a mass produced product, coming out to between $66-76. While this price may seem high, the product is extremely adaptable and durable, making it capable to adjust to any new situations the devices might be introduced to. To maximize the efficiency of the system, a more powerful Xbee can be purchased, further increasing the price.

Production Considerations

To make a production ready version of this prototype, several improvements would have to be made. The PCB size, components, and layout should be optimized for ease of use; we have already taken steps towards this with the incorporation of breakout pins, along with power and mode switches. We would improve the PCB component and chip labels to maximize consistency and readability for the user. We would also have all components professionally soldered, or invest in a resolder oven to make soldering individual components much faster, decreasing labor costs. Additional time would be spent testing the board and all components, and conservative component values and components with extremely low tolerances and ESRs would be used to decrease possibilities of noise or error. These conservative values would also help account for host device differences so that the consumer does not have to test or troubleshoot the board. The production device would simply be purchased, plugged in, and ready to go. We would create a comprehensive and digestible schematic, datasheet, and how-to guide to accompany our PCB-microcontroller system, with notes regarding components and troubleshooting options, as well as setup. Included in this how-to guide would be steps to run the medic UI for the first time and the link to an open source repo of the code, so that users can make adjustments that they see fit. Finally, an ideal adjustment would be to make the code running on all robots available for other robot operating systems as well, rather than just ROS. This would be a massive undertaking, but would maximize the product's audience.

Societal Impact

The robot communication system is part of a broader effort to improve the efficiency of triage during mass casualty events. Ideally, this technology will reduce the stress on first responders by providing automated assessments, allowing them to perform their jobs more efficiently while also enhancing victim care through initial one-on-one evaluations. To effectively assess the societal impact of this technology, its effects on individuals, benefits, risks, and economic implications must be considered.

First responders are the ideal end users for this project. Since they are the primary users, first responders will be interviewed to determine how this technology can best assist them during mass casualty incidents. These types of conversations are very important to the future development of this project, as Lieutenant Colonel Joel Herness answered questions regarding triaging logic and gave advice regarding practical future implementation of this system. He mentioned the importance of keeping full control of decision making and treatment in the hands of the field medic. Because of this, it is important to note that this technology is not designed to provide care but to support it, thus posing no risk of replacing first responders or threatening the current efficiency of their work. He expressed excitement for this project as robots have the potential to complete tasks more efficiently than humans can: carrying bodies, entering combat or contaminated zones, and covering ground in efficient times.

The communication system streamlines the transfer of data between the robots and first responders, improving coordination and response time during a mass casualty event. This efficient communication allows first responders to make more informed decisions while reducing their physical and cognitive load. Additionally, patient care is enhanced, as each victim will receive an individual assessment, regardless of their proximity to the majority of victims.

While the potential positive impacts are numerous, this technology does present significant challenges. The first concern is privacy. As personal data is stored and transmitted via the communication devices, there is a risk of the devices being stolen or hacked, leading to the unwanted release of personal information. Allowing autonomous robots to make triage decisions also raises the risk of exacerbating existing biases in healthcare, particularly if AI is involved. If the data used to determine health status is biased against marginalized groups, it could result in inaccurate assessments. This issue was found in Watson for Oncology (WFO). WFO is an artificial intelligence used to produce evidence-based treatment for oncologists [16]. This technology has been found to not be suitable for use on Chinese patients, as the AI was trained with a small dataset of cancer patients that did not reflect diverse patient populations. Additionally, robots are unable to obtain consent from patients in the same way that first responders can, creating uncertainty around how consent will be managed. Furthermore, hardware errors due to inclement weather or malfunctions could potentially injure patients and worsen their conditions. In a study done by the Canadian Urological Association Journal, 4.97% of robotic surgeries performed over a 3 year span logged robotic malfunctions [17]. For this project, robots are not performing a high level of medical care to potential victims. This data is relevant in expressing the possibility of hardware error of robots in the medical field, thus validating the potential concern.

From an economic perspective, the financial burden of this technology could potentially fall on the victims it aims to assist. Hospitals may choose to impose additional fees for robot assessments alongside regular first responder fees. Hospitals could also encounter a financial burden due to this technology, as they would possibly need to train staff to operate, maintain, and interpret data from triage robots. While the end user is clear, the target buyer is not. Whether this technology is purchased by the government, hospitals, or private emergency services, the financial impact on victims could vary significantly.

This product would be the most impactful in countries that encounter a lot of natural disasters. The XBee module used in this project is only certified for use in 7 countries [18], significantly limiting the product's usability in countries that might need it the most. This raises ethical concerns about equitable access to life-saving technology and highlights the need to address certification barriers to ensure broader implementation in disaster-prone areas.

External Standards

The PCB was manufactured by Advanced Circuits. This company complies with the IPC Acceptability of printed boards standard [19]. This standard was considered during the design phase to ensure the PCB met industry requirements for quality and reliability. The buck converter and XBee module used in this project both comply with ROHS environmental standards, which influenced the selection of components to ensure an environmentally safe design [20]. This was especially important since this system would be interacting with harsh environments. The XBee complies with FCC standards for antennas, which is important to ensure that this system can be used and tested in the United States [5]. The "IEEE Standard for Low-Rate Wireless Networks" outlines the regulations for low power, low cost wireless communication devices, including the Xbee [21]. This standard guided the selection and implementation of the XBee module to ensure compatibility and efficient performance along with the project's communications requirements [22]. The USB port is compatible with the USB 2.0 standard, ensuring reliable data transfer and broad compatibility with multiple devices. This standard was considered during the design phase to guarantee proper functionality and support the required data transfer rates for the project.

All code conformed to general Python standards, which include indentation, naming conventions, comments, and file organization [23].

Intellectual Property Issues

Our project integrates existing technologies in an innovative way to demonstrate a proof of concept. Specifically, we designed a PCB and developed software for a ZigBee communication device to enable communication between autonomous robots and humans. ZigBee, an open standard that operates on the IEEE 802.15.4 specification, is not patented. While our capstone design focuses on ZigBee-based communication and networking, the broader goal of the project is to enable robots to operate autonomously even when out of range of other robots or humans. Patent US20210302956A1, held by Intel Corp, addresses challenges in wireless communication methods for AMR, such as network performance inconsistencies caused by interference, shielding, and roaming, with applications in AMR path planning [24]. Future goals of our project include developing algorithms for AMR motion planning that can predict the locations of patients and other robots. However, unless we create new algorithms or implement a novel process for using these algorithms, existing path planning methods may limit patentability.

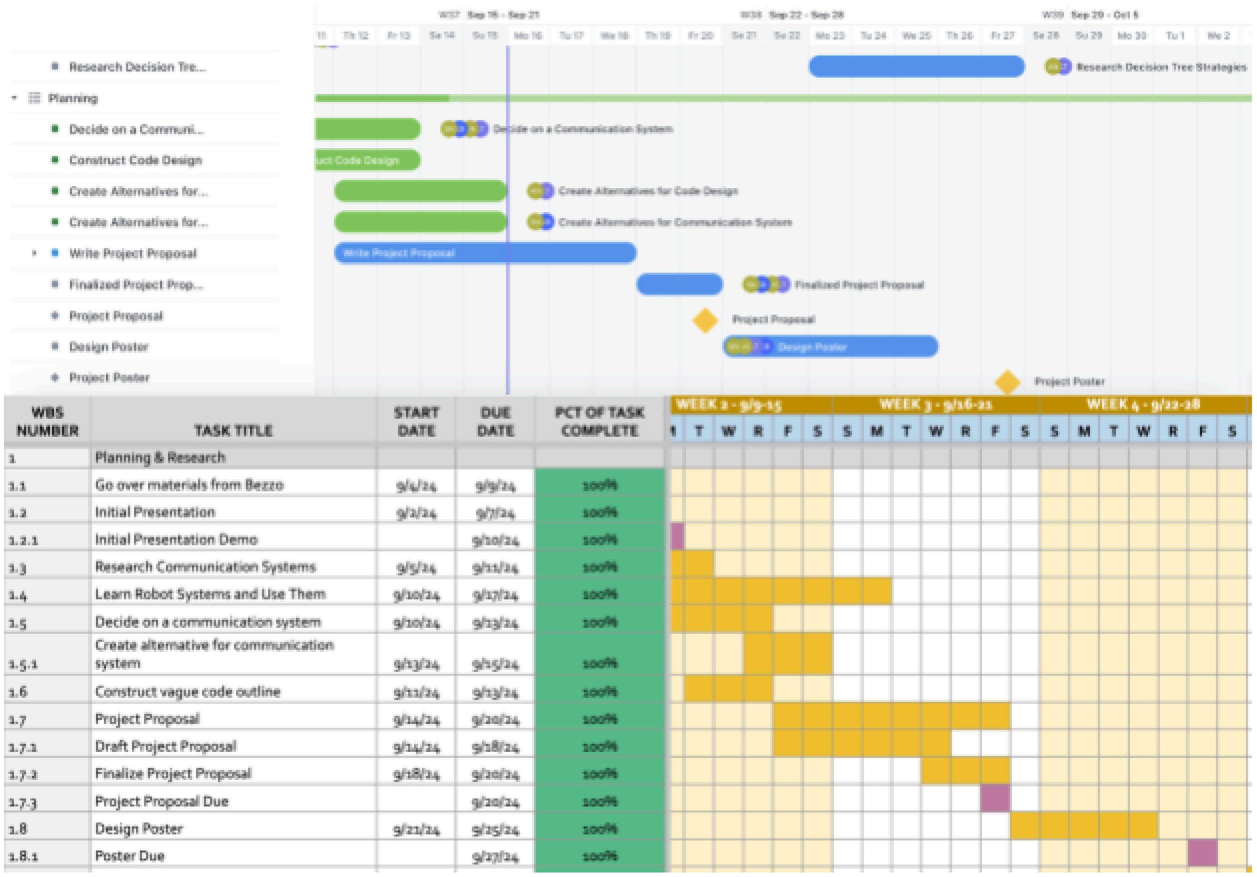
Another related patent, US8708903B2 by Bao Tran, describes a wearable device for recording patient data, analyzing it, and transmitting it to a remote individual such as a nurse, doctor or other caregiver [25]. While this design overlaps with our project in terms of monitoring and transmitting patient information, Tran's design does not incorporate path planning logic for autonomous mobile robots.

Finally, patent US11381271B1 granted to Gary M. Zalewski and Albert S. Penilla explores methods to harvest and store power to enable wireless communication devices to sense, process, and send or receive data [26]. In contrast to our device, which routes power from the robot battery using a PCB, this device harvests power using a power pump and stores it in a storage cell. Since this design aims to wirelessly transmit information, it could potentially be useful for reducing the rate at which robot batteries are drained in future iterations of our communication system. All of the mentioned patents are dependent claims as they cite other patents that contributed to their design.

For our code design, we leveraged existing client libraries such as RosPy and utilized GitHub resources to integrate additional sensors with our PCB and microcontroller. Given the existing patents on AMR path planning, wireless networking, and data transmission, it is unlikely that our project qualifies for patent protection. Its primary innovation lies in integrating components through our software and PCB design rather than inventing a new device or process. In the 2007 Supreme Court case KSR v. Teleflex, it was ruled that combining elements, techniques or devices in a way where each component performs its intended function does not meet the criteria for patentability [27]. While our project integrates components in a novel way, the individual components operate as designed. Therefore, our communication system likely does not satisfy the requirements for a "combination" patent.

Timeline

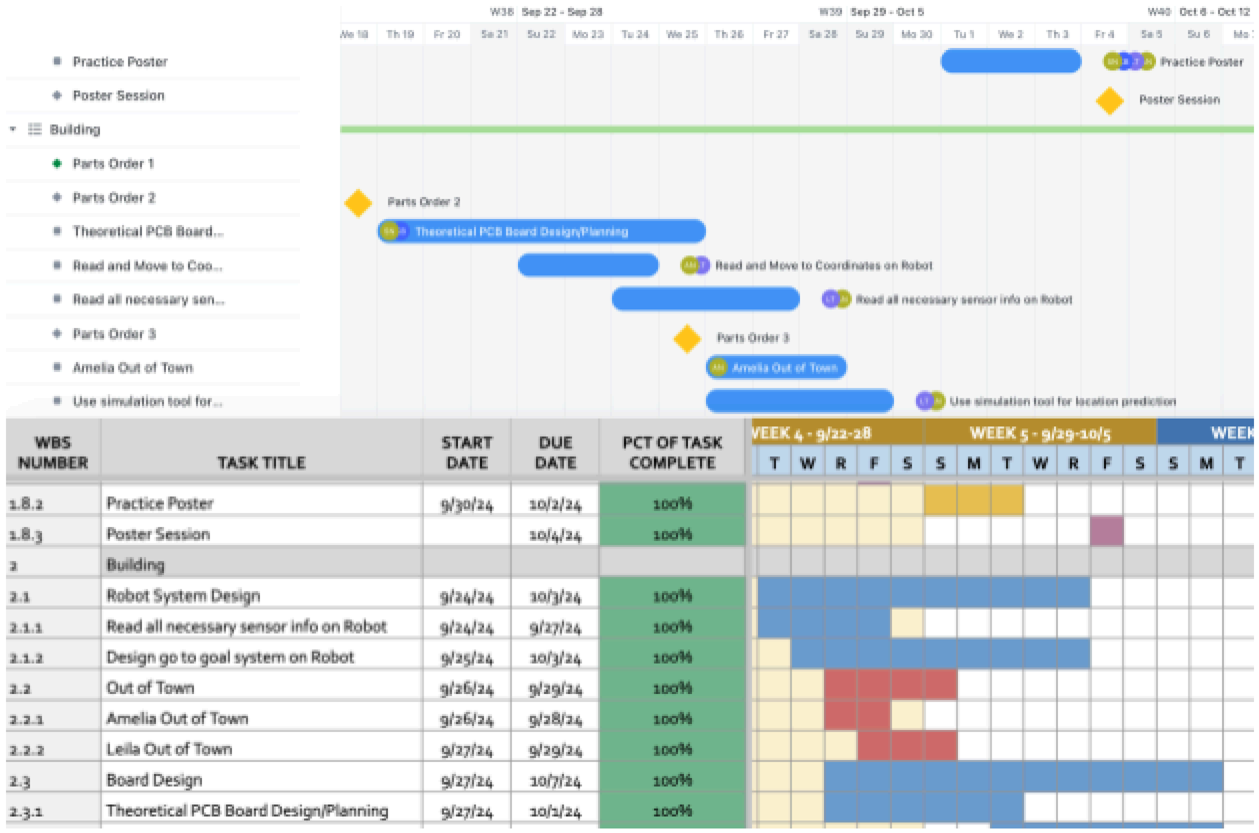
The team's division into two sub-teams worked remarkably well in terms of parallel work and minimizing stoppages from one sub-team's delays. There were a few delays that occurred over the course of the semester which are discussed below. Initially noticeable is our transition to a new Gantt chart tool. It was not originally recognized that the tool that we were using was on a trial basis. Instead of paying for the subscription, we copied all tasks onto another gantt chart and continued use from there.



WBS NUMBER	TASK TITLE	START DATE	DUE DATE	PCT OF TASK COMPLETE
1	Planning & Research			
1.1	Go over materials from Bezzo	9/4/24	9/9/24	100%
1.2	Initial Presentation	9/2/24	9/7/24	100%
1.2.1	Initial Presentation Demo		9/10/24	100%
1.3	Research Communication Systems	9/5/24	9/11/24	100%
1.4	Learn Robot Systems and Use Them	9/10/24	9/17/24	100%
1.5	Decide on a communication system	9/10/24	9/13/24	100%
1.5.1	Create alternative for communication system	9/13/24	9/15/24	100%
1.6	Construct vague code outline	9/11/24	9/13/24	100%
1.7	Project Proposal	9/14/24	9/20/24	100%
1.7.1	Draft Project Proposal	9/14/24	9/18/24	100%
1.7.2	Finalize Project Proposal	9/18/24	9/20/24	100%
1.7.3	Project Proposal Due		9/20/24	100%
1.8	Design Poster	9/21/24	9/25/24	100%
1.8.1	Poster Due		9/27/24	100%

Figure 26. September 9 - 28 Gantt Chart Comparison

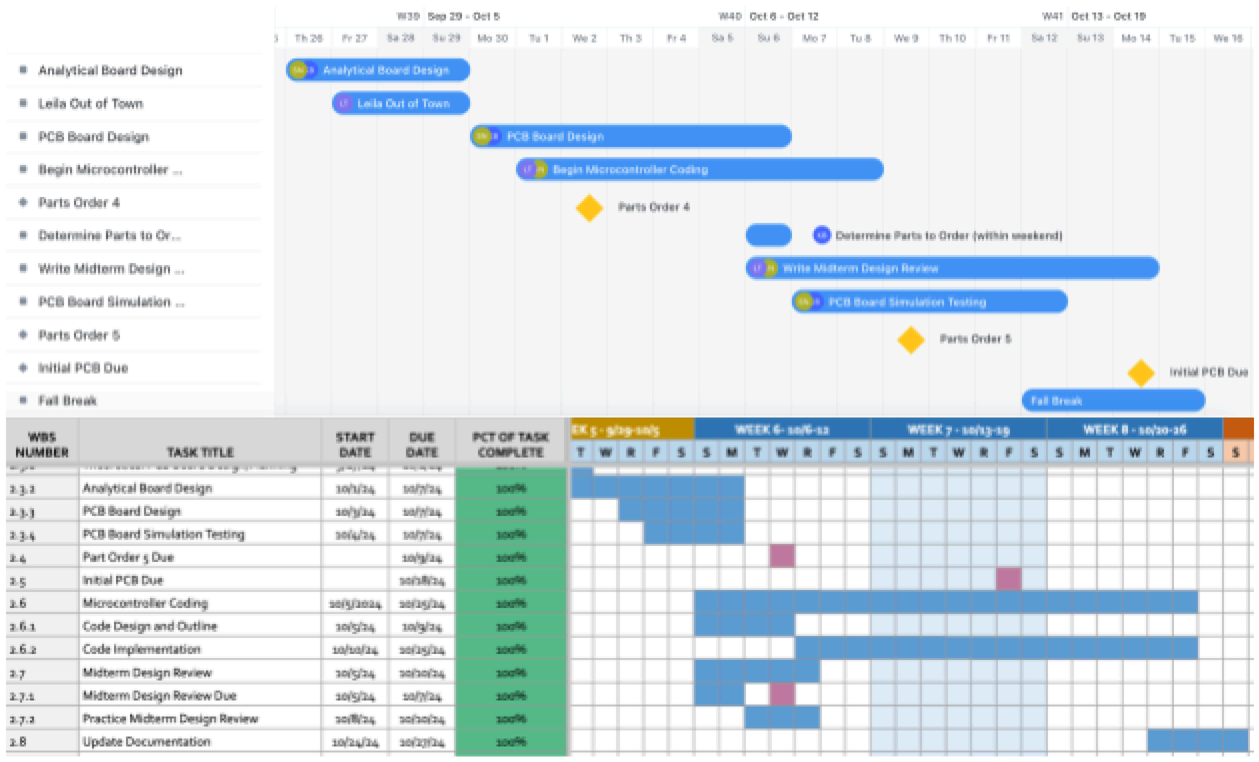
In the above photo you can see a comparison of the beginning weeks of research and planning. In the beginning, we stayed on schedule as it was still near when we created our projected timeline. As we got farther from the gantt charts initial construction, the projected timeline became less accurate, as we will see below.



WBS NUMBER	TASK TITLE	START DATE	DUE DATE	PCT OF TASK COMPLETE
1.8.2	Practice Poster	9/30/24	10/2/24	100%
1.8.3	Poster Session		10/4/24	100%
2	Building			
2.1	Robot System Design	9/24/24	10/3/24	100%
2.1.1	Read all necessary sensor info on Robot	9/24/24	9/27/24	100%
2.1.2	Design go to goal system on Robot	9/25/24	10/3/24	100%
2.2	Out of Town	9/26/24	9/29/24	100%
2.2.1	Amelia Out of Town	9/26/24	9/28/24	100%
2.2.2	Leila Out of Town	9/27/24	9/29/24	100%
2.3	Board Design	9/27/24	10/7/24	100%
2.3.1	Theoretical PCB Board Design/Planning	9/27/24	10/1/24	100%

Figure 27. September 24 - October 8 Gantt Chart Comparison

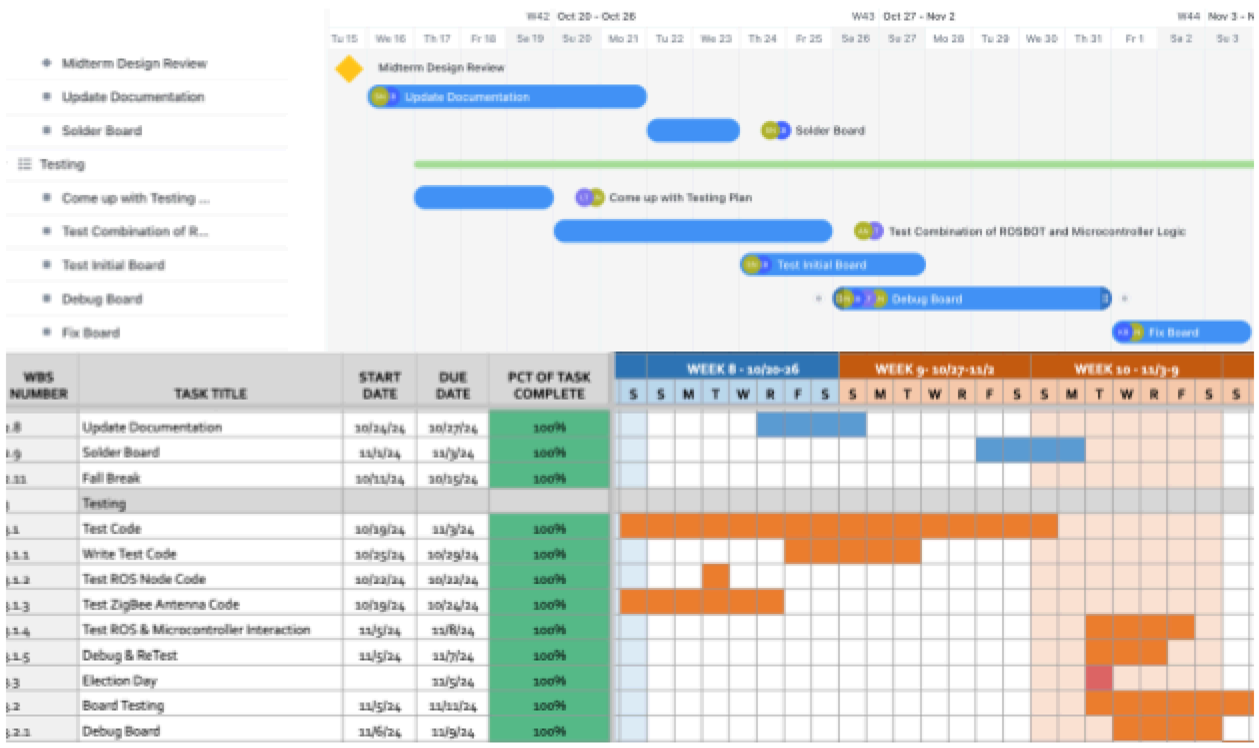
In the images above, larger differences in the anticipated versus actual timelines can be seen. Besides a reordering of the events, certain tasks took much longer than initially planned, specifically designing the go-to-goal system. Additionally, theoretical PCB board design began close to a week later than initially predicted, but did not take as long as was initially allotted.



WBS NUMBER	TASK TITLE	START DATE	DUE DATE	PCT OF TASK COMPLETE
2.3.2	Analytical Board Design	10/1/24	10/7/24	100%
2.3.3	PCB Board Design	10/3/24	10/7/24	100%
2.3.4	PCB Board Simulation Testing	10/4/24	10/7/24	100%
2.4	Part Order 5 Due		10/9/24	100%
2.5	Initial PCB Due		10/18/24	100%
2.6	Microcontroller Coding	10/5/2024	10/25/24	100%
2.6.1	Code Design and Outline	10/5/24	10/9/24	100%
2.6.2	Code Implementation	10/10/24	10/25/24	100%
2.7	Midterm Design Review	10/5/24	10/10/24	100%
2.7.1	Midterm Design Review Due	10/5/24	10/7/24	100%
2.7.2	Practice Midterm Design Review	10/8/24	10/10/24	100%
2.8	Update Documentation	10/24/24	10/27/24	100%

Figure 28. October 1 - 27 Gantt Chart Comparison

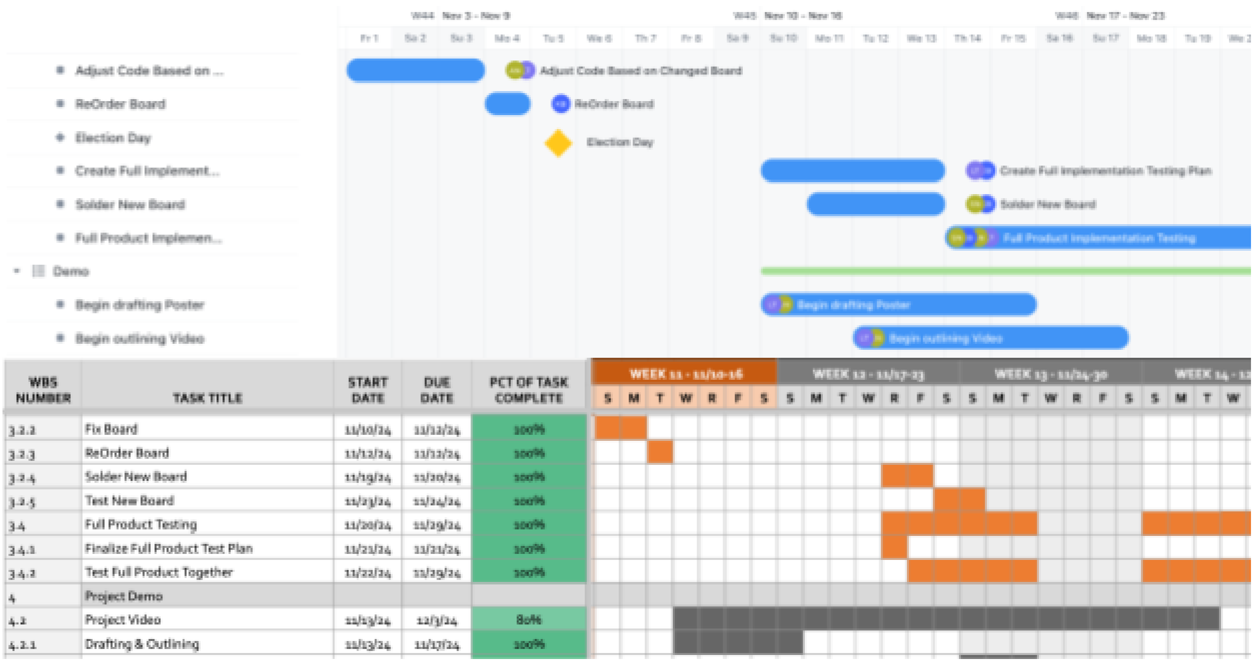
This section of the Gantt chart displays when our plans were the least accurate. Starting at the top, analytical board design started on time but took longer than expected, while the board design and simulation testing were completed quickly and were much more in parallel than anticipated. One of the largest delays or slow downs in the plan was the microcontroller code, being completed over the course of nearly three weeks instead of the initially allotted 1.5 weeks. The midterm design started on time but the deadline was earlier than originally logged, resulting in its quicker completion. Updating of the documentation was done on time, and can be seen in the top portion of Figure 29 below.



WBS NUMBER	TASK TITLE	START DATE	DUE DATE	PCT OF TASK COMPLETE
.8	Update Documentation	10/24/24	10/27/24	100%
.9	Solder Board	11/1/24	11/3/24	100%
.11	Fall Break	10/11/24	10/15/24	100%
	Testing			
.1	Test Code	10/19/24	11/3/24	100%
.1.1	Write Test Code	10/25/24	10/29/24	100%
.1.2	Test ROS Node Code	10/22/24	10/22/24	100%
.1.3	Test ZigBee Antenna Code	10/19/24	10/24/24	100%
.1.4	Test ROS & Microcontroller Interaction	11/5/24	11/8/24	100%
.1.5	Debug & ReTest	11/5/24	11/7/24	100%
.3	Election Day		11/5/24	100%
.2	Board Testing	11/5/24	11/11/24	100%
.2.1	Debug Board	11/6/24	11/9/24	100%

Figure 29. October 19 - November 10 Gantt Chart Comparison

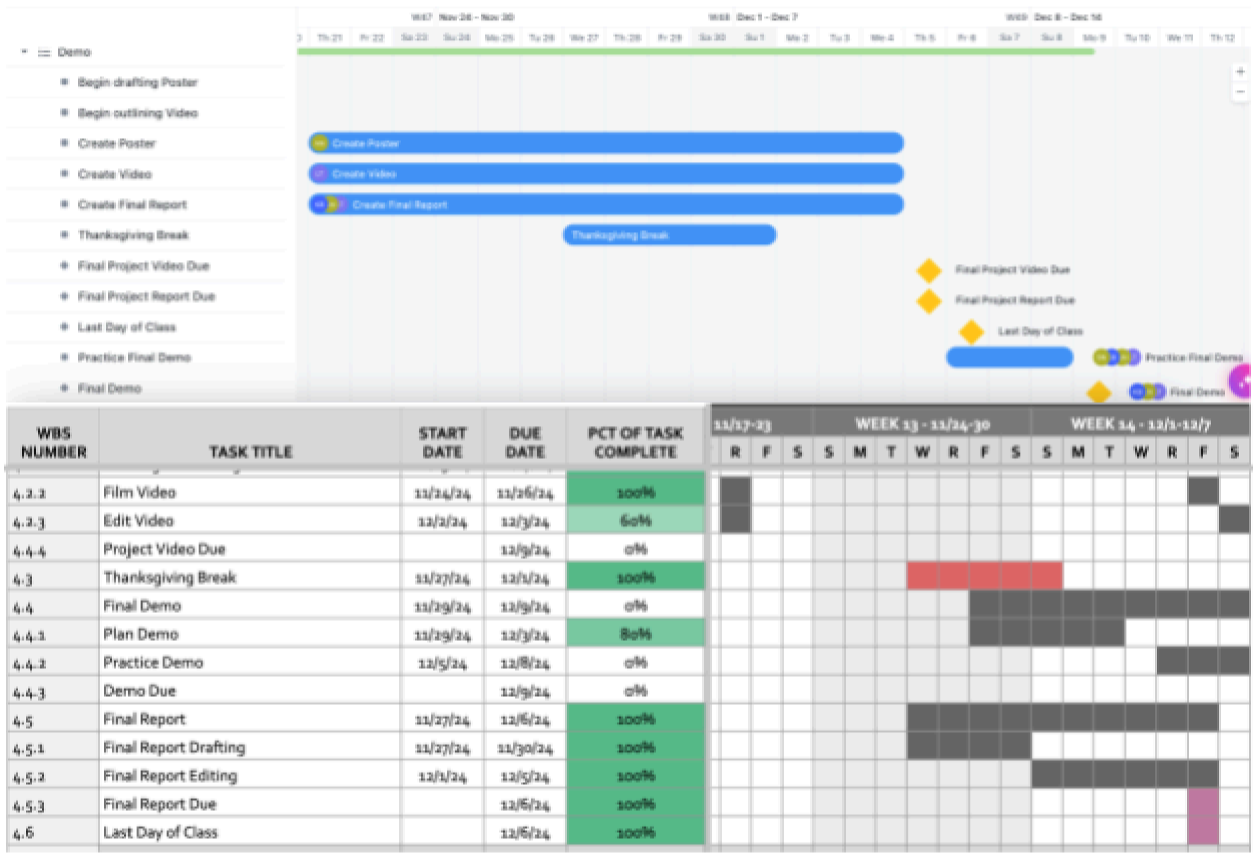
For this section of the Gantt chart, we created more detailed tasks as we progressed. The largest delay over the course of the project occurred with the arrival of the PCB. Due to manufacturer issues, the board arrived close to 3 weeks after it was ordered, roughly 2 weeks later than expected. This caused a delay in board testing, board reordering, and full system testing, as seen in the following figures. Ultimately this put us about a week behind schedule. Code testing started on schedule but exceeded the initially allocated time frame because more comprehensive testing functions were developed for each section of the code than originally anticipated.



WBS NUMBER	TASK TITLE	START DATE	DUE DATE	PCT OF TASK COMPLETE
3.2.2	Fix Board	11/10/24	11/12/24	100%
3.2.3	ReOrder Board	11/12/24	11/12/24	100%
3.2.4	Solder New Board	11/19/24	11/20/24	100%
3.2.5	Test New Board	11/23/24	11/24/24	100%
3.4	Full Product Testing	11/20/24	11/29/24	100%
3.4.1	Finalize Full Product Test Plan	11/21/24	11/21/24	100%
3.4.2	Test Full Product Together	11/22/24	11/29/24	100%
4	Project Demo			
4.2	Project Video	11/13/24	12/3/24	80%
4.2.1	Drafting & Outlining	11/13/24	11/17/24	100%

Figure 30. November 10 - December 4 Gantt Chart Comparison

The week-long delay from manufacturing can again be seen here, with the new board ordered eight days after planned. This again delayed full product testing, along with some of the final deliverables of the project. It was discovered that there was no poster assignment due, even though it was on the original timeline, which increased our time for other tasks. While waiting for the second iteration of boards to be delivered, we outlined the final video and filmed two sections of it.



WBS NUMBER	TASK TITLE	START DATE	DUE DATE	PCT OF TASK COMPLETE	11/17-23			WEEK 13 - 11/24-30							WEEK 14 - 12/1-12/7						
					R	F	S	S	M	T	W	R	F	S	S	M	T	W	R	F	S
4.2.2	Film Video	11/24/24	11/26/24	100%	■															■	
4.2.3	Edit Video	12/2/24	12/3/24	60%	■																■
4.4.4	Project Video Due		12/9/24	0%																	
4.3	Thanksgiving Break	11/27/24	12/1/24	100%							🟥	🟥	🟥	🟥	🟥						
4.4	Final Demo	11/29/24	12/9/24	0%									■	■	■	■	■	■	■	■	■
4.4.1	Plan Demo	11/29/24	12/3/24	80%									■	■	■	■	■				
4.4.2	Practice Demo	12/5/24	12/8/24	0%															■	■	■
4.4.3	Demo Due		12/9/24	0%																	
4.5	Final Report	11/27/24	12/6/24	100%							■	■	■	■	■	■	■	■	■	■	
4.5.1	Final Report Drafting	11/27/24	11/30/24	100%							■	■	■	■							
4.5.2	Final Report Editing	12/1/24	12/5/24	100%											■	■	■	■	■	■	
4.5.3	Final Report Due		12/6/24	100%																■	
4.6	Last Day of Class		12/6/24	100%																■	

Figure 31. November 21 - December 7 Gantt Chart Comparison

The final section of the project can be seen above. The delay in the original board delivery can again be seen, along with a much longer final testing window. Upon delivery of the board, we had to identify a faulty ground within the Xbee header pins. Due to the resources required for board testing and board adjustments, little work was completed over Thanksgiving break. Instead during that time our team focused on the final report. Although it took up until the last day, we were able to do a full system test and verify our system. Despite the delays, we were able to maximize our time from delays and use it to begin work on the final report ahead of time, maximizing our time available to work on the device at the conclusion of the semester.

Costs

The tables below outline our budget for the entire semester. A portion of our funding, specifically for ordering the final PCBs, came from Professor Bezzo's lab. This allowed us to be under budget for the allotted Capstone budget, but over budget in terms of total spent. A specific component and supplies breakdown is given in Appendix A.

Date	Item(s)	Total Price	Funding Source
9/25/2024	2 XBees	47.04	Capstone Budget
10/8/2024	2 Development Boards + 20 XBee Header Pins	74.2	Capstone Budget
10/8/2024	2 Adafruit Sensors	46.42	Capstone Budget
10/9/2024	Parts order	114.37	Capstone Budget
10/15/2024	Osh Park 3 PCBs	45.15	Capstone Budget
10/27/2024	Dev Board + XBee	$50.76	Capstone Budget
11/14/2024	Mouser parts order	31.4	Capstone Budget
11/14/2024	Canned air	13.15	Capstone Budget
11/04/2024	3W Manufacturing(1st board iteration)	5.9	Capstone Budget
11/21/2024	3W manufacturing(final Board iteration)	6.8	Capstone Budget
	Advanced PCB Final Boards	122.41	Prof. Bezzo's Lab
Total Spent		435.19	
Budget Remaining:	$64.81		

Table 2. Budget Overview

When considering the cost to mass produce our final product, the unit prices drop significantly. For example, Mouser and Digikey offer lower prices for bulk purchases, anywhere from 40-75%. However, the unit price of our chosen XBee3 Module and BME680 (temperature sensor) do not decrease with quantity. Thus, the price of our product, even scaled to 10,000+ units, will have a base cost of $41 from the XBee and sensor alone. Without considering the XBee, the cost of components per board would amount to $10-15, depending on the quality and availability of specific components. Our PCB is double-layered and about ~9 square inches; prices can go below $1 per square inch for mass production. We also have to account for professional soldering of components. A reasonable unit price for a manufactured and professional soldered PCB would be in the range of $25-35 per board plus the extra costs of the XBee and temperature (or other compatible) sensor brings the unit price to $66-76.

Final Results

In the end, our device functioned as anticipated. After thorough testing, the robots follow their intended path - the exploratory going to given victim locations and the hub rotating between meet up points. If at any point the exploratory robot's victim list is urgent, meeting the requirements described earlier in the document, it travels to its known hub robot location. From there, it waits for a predetermined length of time. If it does not successfully transmit the information, the exploratory robot will travel to the medic instead. The hub robot, similarly, waits at a meeting point. If it does not receive any messages, it continues on to the next meeting point. If the hub robot does receive messages, it travels to the medic's location, transmits those messages, and then travels to the next meeting point.

Our original criteria for an A included the following:

1. Communication between all of the following pairs is fully functional without the use of WIFI or Bluetooth: exploratory robot and hub robot, hub robot and computer, exploratory robot and computer
2. All systems send and receive an initial test message to both the medic and the aerial vehicle before beginning exploration.
3. Default backup procedures occur as expected (robot travels until it finds the human using only the predictive algorithms)
4. If hub robot intercepts an exploratory robot, the exploratory robot continues exploration instead of returning to the human
5. A functional PCB and microcontroller communication system is delivered and demonstrated.

Of this criteria, we met requirements #1, #3, #4, and #5.

A slight change was made to #3 - our robots do not use predictive algorithms, but instead go to a known medic location. Predictive algorithms were not within the scope of the ECE one semester capstone, as the systems students were tasked with that portion and they are on a two semester capstone schedule as opposed to one.

For #2 we did not create an initial test message as we ran out of time. As stated in the beginning of the report, this would be a relatively simple addition, but given our time constraints and the delays we experienced, the initial test message was not a priority.

Figure 32 shows the full system prototype, integrating the ROSbot, PCB holder, XBee, and temperature sensor.

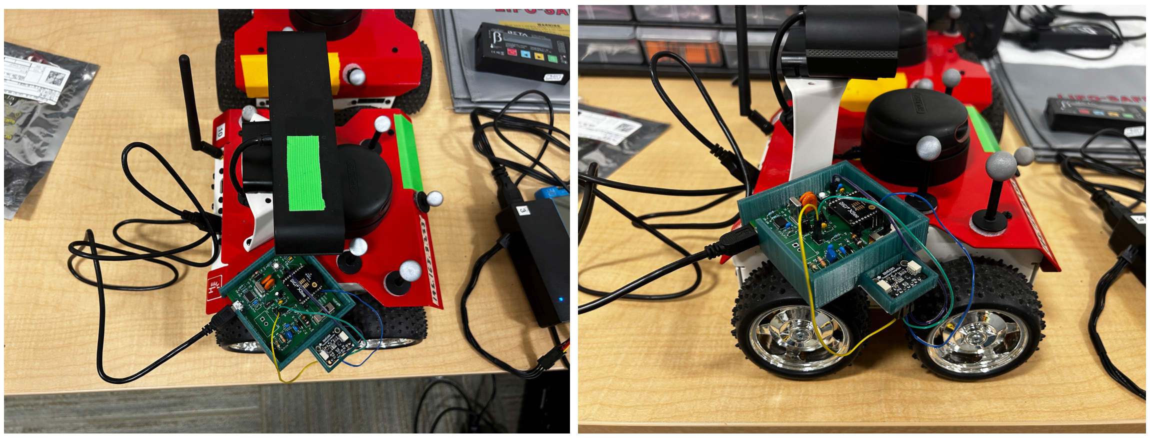


Figure 32. Full System Prototype

Engineering Insights

In this project, we applied foundational skills from our introductory electrical and computer engineering (ECE) courses while also developing new technical and soft skills.

During the PCB development process, we gained extensive experience with the schematic and PCB editing tools with KiCad, having to create the PCB schematic from scratch and follow it through until generating the Gerber files for manufacturing. We improved our ability to read and analyze datasheets for integrated circuits (ICs), having to pay attention to external required components, tolerances and other specifications. Additionally, we developed our soldering and problem solving skills, particularly in addressing challenges like replacing or correcting incorrectly soldered parts. If a part needed to be replaced or modified, we utilized a combination of the solder sucker, solder wicks, clamp, and heat gun, depending on part complexity and size. Over the course of the device holder design, we strengthened our CAD skills, along with our familiarity with the various 3D printers on grounds. Overall, through the PCB manufacturing and holder 3D printing processes, we learned about the importance of setting early deadlines and planning for multiple iterations of a design while working to ensure that every design is the best it can be.

Throughout the software development process, we gained experience in creating software from scratch and learned lessons about teamwork and communication. Being in control of the product design from start to finish provided us flexibility to structure and organize our code as we saw fit. Originally, we intended to maintain a straightforward and well structured code architecture, with classes separated into strict purposes with minimal dependencies between classes or repeat code. However, while we began the project with a strong focus on maintaining a clear and well defined code structure, we found that this intention was difficult to sustain under the pressure of completing tasks and making progress with our project. Over time, the code design became more convoluted as we shifted our focus toward code functionality rather than maintainability. This process also revealed the challenges of dividing tasks in a collaborative environment. At times, one team member would make a change without fully explaining the intent, leading to delays and confusion. As a result, we learned to prioritize communication when constructing our code. Despite these challenges, our coding and software development skills grew significantly. We strengthened our Python skills, explored tools like Pytest and MicroPython, and applied software engineering principles learned in CS 3140. To stay organized and set achievable goals, we implemented a to-do list system for tracking team member functions or tests still needed. This approach was crucial for maintaining our week-to-week progress.

With regards to the engineering process, we learned about time and resource management, along with teamwork, communication, and maintaining morale. We had to manage our time and account for delays in designing and manufacturing, while juggling our capstone with other classes and commitments. This sometimes proved difficult as capstone deadlines could feel less urgent than other courses. We had to manage our budget and funding, determining how to minimize our costs while accounting for extra parts and selecting reliable, quality components. We also developed skills in teamwork; when we decided to split the work among members, it allowed everyone to do a similar amount, helping prevent any one person feeling extremely overwhelmed. This also created the opportunity for others to provide support when necessary. However, due to this divided structure, it was extremely important to communicate updates and delays to the rest of the team immediately to help others adequately plan next steps. Lastly, we learned the importance of balancing dedication with self-care. While pushing through to complete a task late into the night can boost morale and provide a sense of accomplishment, recognizing when to rest and recharge is equally vital for maintaining productivity and well-being.

If we had to give advice to future capstone students, it would have to be to set deadlines earlier than expected, because delays are guaranteed, and to maximize your planning and outlines. While things are guaranteed to change, it is much easier to make progress when you have a fleshed out goal to work towards, regardless of whether they are small or large.

Future Work

For future expansion on this project, we recommend modifying the code to improve scalability and adaptability for use across different systems. Currently, the project code is specifically designed for the XBee 3 ZigBee antenna and microcontroller and ROS for the ROSbot 2.0 by Husarion. However, the project's primary aim is to develop a communication system to be utilized by robots and medics in large-scale human triage scenarios. It is unlikely that the autonomous robots and antenna used for our proof of concept will be the same as those ultimately used for the triage. Therefore, the code should be made more flexible for easy adaptation to various devices and systems.

Numerous components of this project were designed separately - the robot's goal-to-goal navigation mechanism, message interpreter and logic, the XBee3 microcontroller, and the PCB - before being integrated together. While designing each component individually was manageable, integrating them together proved more challenging, both conceptually and practically. Thoroughly mapping out the code structure and the relationships between components in advance could help make future integration smoother.

Our capstone focused on designing a communication system to enable the robots to communicate with each other and the medics. However, the overarching project also aims to implement algorithms that 1) allow the robots to categorize the severity of the casualties they find and 2) use epistemic logic to determine the robots' next target location as well as where other parties are likely to be. Future students interested in joining this larger project could contribute by integrating the severity categorization and epistemic logic algorithms with the communication system developed for our capstone.

Initially, our test plan included an aerial robot, or drone, to serve as the hub robot and quickly route information between the ground robots and medics. However, due to time constraints and the added complexity of configuring the available drones in the AMR lab, we were unable to include an aerial vehicle in our testing. Future iterations of this project could incorporate this element.

In our tests, the severity ratings were determined using the locations of objects of interest. In real-world applications, robots would assess the vitals of human patients. Therefore, developing a system to measure patient vitals is a crucial next step in the larger project. Additionally, enhancing the communication system to operate over greater distances, such as by selecting a higher-range XBee device and using GPS data instead of Vicon, would further improve the system's capabilities.

Finally, the last addition that we would recommend is the creation of thorough documentation, both about the PCB construction and the software. This would allow for greater flexibility in the future, as well as widen the audience for both engineers working on the product and users looking to implement the product on their devices.

References

[1] ZettaScale, “Overcoming Wireless Communication Challenges in Robotics,” *zettascale.tech* [Online]. Available: https://www.zettascale.tech/news/overcoming-wireless-communication-challenges-in-robotics/. [Accessed: Sept. 19, 2024].

[2] J. Gielis, A. Shankar, and A. Prorok, “A Critical review of Communications in Multi-robot systems,” *Current Robotics Reports*, vol. 3, no. 4, pp. 213–225, Aug. 2022, doi: 10.1007/s43154-022-00090-9.

[3] D. Gislason. (2008, Oct 9). *ZigBee Wireless Networking* [Online]. Available: https://books.google.com/books?hl=en&lr=&id=up8Oa7456I8C&oi=fnd&pg=PP1.

[4] Digi, “Digi XBee 3 ZigBee 3 RF Module,” *Digi Xbee* [Online]. Available: https://www.digi.com/products/embedded-systems/digi-xbee/rf-modules/2-4-ghz-rf-modules/xbee3-ZigBee-3. [Accessed Sept. 18, 2024].

[5] Digi, *Digi XBee® 3 RF Module Hardware Reference Manual* [Online]. Available: https://www.digi.com/resources/documentation/digidocs/pdfs/90001543.pdf.

[6] Texas Instruments, “LMR14206 Simple Switcher 42Vin, 0.6A Step-Down Voltage Regulator in SOT-23,” SNVS733D datasheet, Oct. 2011 [Revised Apr. 2013].

[7] Mouser Electronics, “Molex 47590-0001,” *Mouser* [Online]. Available: https://www.mouser.com/ProductDetail/Molex/47590-0001?qs=B2dI1vFk2x%252BDCUfk3sC4bA%3D%3D&mgh=1&utm_id=17222215321&gad_source=1&gclid=Cj0KCQjw05i4BhDiARIsAB_2wfBPlXjhNAjFqXSP1CJK5f4cpVLjKqgxjL-2CvT85TZInl-JfjwcK8caAtVEEALw_wcB.

[8] Adafruit Industries, "STEMMA QT / Qwiic JST SH 4-pin to Premium Male Headers Cable - 150mm Long" [Online]. Available: https://www.adafruit.com/product/4209. [Accessed: Dec. 5, 2024].

[9] S. Sanfilippo (‘antirez’), "bme680-pure-mp: MicroPython Driver for BME680 Sensor", GitHub repository. [Online]. Available: https://github.com/antirez/bme680-pure-mp/tree/main. [Accessed: November 14, 2024].

[10] ROS, “Why ROS?” *ROS* [Online]. Available: https://www.ros.org/blog/why-ros/. [Accessed: Sept. 18, 2024].

[11] ROS, “rospy,” *ROS Index* [Online]. Available: https://index.ros.org/p/rospy/github-ros-ros_comm/#noetic. [Accessed: Sept. 18, 2024].

[12] SparkFun Electronics, "SparkFun Qwiic Atmospheric Sensor - BME680 (Qwiic)," [Online]. Available: https://www.sparkfun.com/products/21636. [Accessed: November 15, 2024].

[13] Pytest, “pytest: helps you write better programs,” *Pytest Documentation* [Online]. Available: https://docs.pytest.org/en/stable/.

[14] Texas Instruments, “PSpice for TI design and simulation tool,” *Texas Instruments* [Online]. Available: https://www.ti.com/tool/PSPICE-FOR-TI.

[15] Microchip, “1.2.2 Download and Install the USB Serial Driver,” *Metering Demo and Developer User Guide* [Online]. Available: https://onlinedocs.microchip.com/oxy/GUID-7AF360DD-C920-4EAF-93E5-FA00C76B2FC5-en-US-2/GUID-D8794665-89E7-491E-B113-44B4D574D0D6.html.

[16] F. Zou, Y. Tang, C. Liu, J. Ma and C. Hu, “Concordance Study Between IBM Watson for Oncology and Real Clinical Practice for Cervical Cancer Patients in China: A Retrospective Analysis,” Front Genet, vol. 11, pp. 1-8, Mar. 2020. doi: 10.3389/fgene.2020.00200. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7105853/pdf/fgene-11-00200.pdf.

[17] E. Rajih et al., “Error reporting from the da Vinci surgical system in robotic surgery: A Canadian multispecialty experience at a single academic centre,” Canadian Urological Association Journal, vol. 11, pp.1-6, Jan. 2017. doi: 10.5489/cuaj.4116. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5426941/.

[18] Digi, “Certifications,” *Digi* [Online]. Available: https://www.digi.com/resources/certifications?pn=XB3-24Z8PT-J.

[19] IPC, “IPC-A-600 - Revision K - Standard Only: Acceptability of Printed Boards,” *IPC* [Online]. Available: https://shop.ipc.org/ipc-a-600/ipc-a-600-standard-only/Revision-k/english.

[20] ChemSafetyPRO, “EU RoHS 2 (Directive 2011/65/EU),” *ChemSafetyPRO* [Online]. Available: https://www.chemsafetypro.com/Topics/Restriction/EU_RoHS_2_EU_RoHS_Directive.html

[21] "IEEE Standard for Low-Rate Wireless Networks Corrigendum 1: Correction of Errors Preventing Backward Compatibility," in IEEE Std 802.15.4-2020/Cor 1-2022 (Corrigendum to IEEE Std 802.15.4-2020 as amended by IEEE Std 802.15.4z-2020, IEEE Std 802.15.4w-2020, IEEE Std 802.15.4y-2021, and IEEE Std 802.15.4aa-2022) , vol., no., pp.1-22, 10 Jan. 2023, doi: 10.1109/IEEESTD.2022.10014667. keywords: {IEEE Standards;Wireless networks;Personal area networks;Data processing;IEEE 802.15 Standard;corrigendum;870 MHz band;920 MHz band;IEEE 802.15.4;low data rate;low power;wireless personal area network},

[22] Molex, “Micro-USB AB Receptacle, Right-Angle, Top Mount, Surface Mount, Lead-Free,” 475900001 datasheet.

[23] Python, “PEP 8 - Style Guide for Python Code,” *Python Enhancement Proposals* [Online]. Available: https://peps.python.org/pep-0008/.

[24] S. Sudhakaran et al., “Network aware and predictive motion planning in mobile multi-robotics systems,” U.S. Patent US20210302956A1, Mar. 24 2021.

[25] B. Tran, “Patient monitoring appliance,” U.S. Patent US8708903B2, Apr. 29, 2013.

[26] G. M. Zalewski and A. S. Penilla, “Energy harvesting sensor device with wireless communication,” U.S. Patent US11381271B1, July 05 2022.

[27] B. Farkas, “When is a Combination Invention Patentable?” *Nolo* [Online]. Available: https://www.nolo.com/legal-encyclopedia/combination-invention-patentable-patents-29891.html. [Accessed: Dec. 05, 2024].

Appendix

Appendix A - Budget

		Electronics & electrical components							
Index	Manufacturer Part #	Digikey Part #	Mouser Part #	Qty in Stock	Qty Req'd	Per Unit Price	Cost	Team Name	Comments
1	K104K15X7RF5UL2	56-K104K15X7RF5UL2CT-ND		8,447	10	0.10700	1.07	The Coolombs	0.1μF ceramic capacitor
2	C322C334M5U5TA7301	399-9882-1-ND		8,440	10	0.32000	3.20	The Coolombs	0.33 uF capacitor
3	FG28X7R1H474KRT06	445-173600-1-ND		2,597	10	0.20900	2.09	The Coolombs	0.47 uF capacitor
4	FG28X7R1E105KRT06	445-173583-1-ND		14,934	20	0.19300	3.86	The Coolombs	1.0 μF ceramic capacitor
5	ECE-A1EKA100B	10-ECE-A1EKA100BCT-ND		3,621	10	0.09300	0.93	The Coolombs	10 uF decoupling capacitor, 20% tolerance
6	FG18X5R1E225KRT06	445-173253-1-ND		1,326	10	0.20900	2.09	The Coolombs	2.2 uF capacitor
7	C317C829D5G5TA	399-8923-ND		10,122	10	0.24700	2.47	The Coolombs	8.2 pF ceramic capacitor
8	FK22X5R0J107MN006	445-FK22X5R0J107MN006CT-ND		67	10	1.53700	15.37	The Coolombs	100 uF capacitor
9	SB160	1655-SB160CT-ND		6,622	10	0.09700	0.97	The Coolombs	60 V 1 A diode
	B78108E1103K009	495-76013-1-ND		11,609	10	0.29500	2.95	The Coolombs	10 μH Inductor
11	RNMF14FTC10K0	S10KCACT-ND		113,186	50	0.04940	2.47	The Coolombs	10k resistor, 1% tolerance
12	RNMF14FTC8K20	S8.2KCACT-ND		14,989	10	0.08700	0.87	The Coolombs	8.2k Ohm Resistor, 1% tolerance
13	RNMF14FTC24K0	S24KCACT-ND		12,631	10	0.08700	0.87	The Coolombs	24k Ohm Resistor, 1% tolerance
	TL59NF160Q	EG2532CT-ND		10,474	6	0.29000	1.74	The Coolombs	Tactile Switch Reset Button
15	61301611121	732-5327-ND		3,288	4	0.96000	3.84	The Coolombs	Jumper/header pins
16	47590-0001		538-47590-0001	12,246	10	0.50200	5.02	The Coolombs	Micro USB ports
17	1825255-1	450-1567-ND		5,179	10	1.11300	11.13	The Coolombs	SPDT Switch (mode switch)
18	GF-123-0054	CWI333-ND		1,170	10	1.48800	14.88	The Coolombs	SPST Switch (power switch)
19	MCP2200T-I/SS	MCP2200T-I/SSCT-ND		4,028	10	2.61000	26.10	The Coolombs	USB to UART converter chip
20	LMR14206XMKX/NOPB	296-41446-1-ND		5,057	5	2.49000	12.45	The Coolombs	TI Buck Converter
21	PPTC071LFBN-RC	S7005-ND		13,135	5	$0.69	3.45	TheCoolombs	CONN HDR 7POS 0.1 TIN PCB
22	XB3-24Z8PT-J	602-2188-ND		8,397	2	23.52	47.04	The Coolombs	RF TXRX MOD 802.15.4 TRC ANT TH

Figure A1. Weekly Parts Order Record

Index	Manufacturer Part #	AdaFruit Part #	SparkFun Part #	Digikey Part #	Mouser Part #	Qty Req'd	Per Unit Price	Cost	Comments
1		4209				2	18.95	46.42	Adafruit Sensors
2		3660				2	0.95		Adafruit STEMMA QT Cables
3	WRL-21636			1568-WRL-21636-ND		3	19.95	59.85	Development Boards
4			PRT-10112			20	$1.15	$23.00	ZigBee Header Pins
5						1	12.49	13.15	Canned Air
6	561R10TCCQ50				75-561R10TCCQ50	10	$0.70	31.4	50pF Ceramic Disc Capacitors
7	830003206B				710-830003206B	5	$0.73		Crystal Oscillators
8	PRT-12796				474-PRT-12796	2	$1.95		6in F/F Jumper Wires
9	LMR14206XMKX/NOPB				926-LMR14206XMKXNOPB	4	$2.21		Buck Converter

Figure A2. Personal Purchases Record

Index	Free Items	Source
1	3D Printing Material	Shannon Library MakerSpace, UVA Electrical and Computer Engineering Department
2	AD2 and Breadboard	Shrinidhi (on loan from UVA Electrical Engineering Department)
3	Various resistors and capacitors	Kailey (materials purchased for courses from UVA Electrical Engineering Department)
4	Soldering equipment	UVA Electrical and Computer Engineering Department
5	ROSbots	Bezzo Lab

Figure A3. Previously Paid For, or On Loan Materials Record

Appendix B - Full PCB Schematic

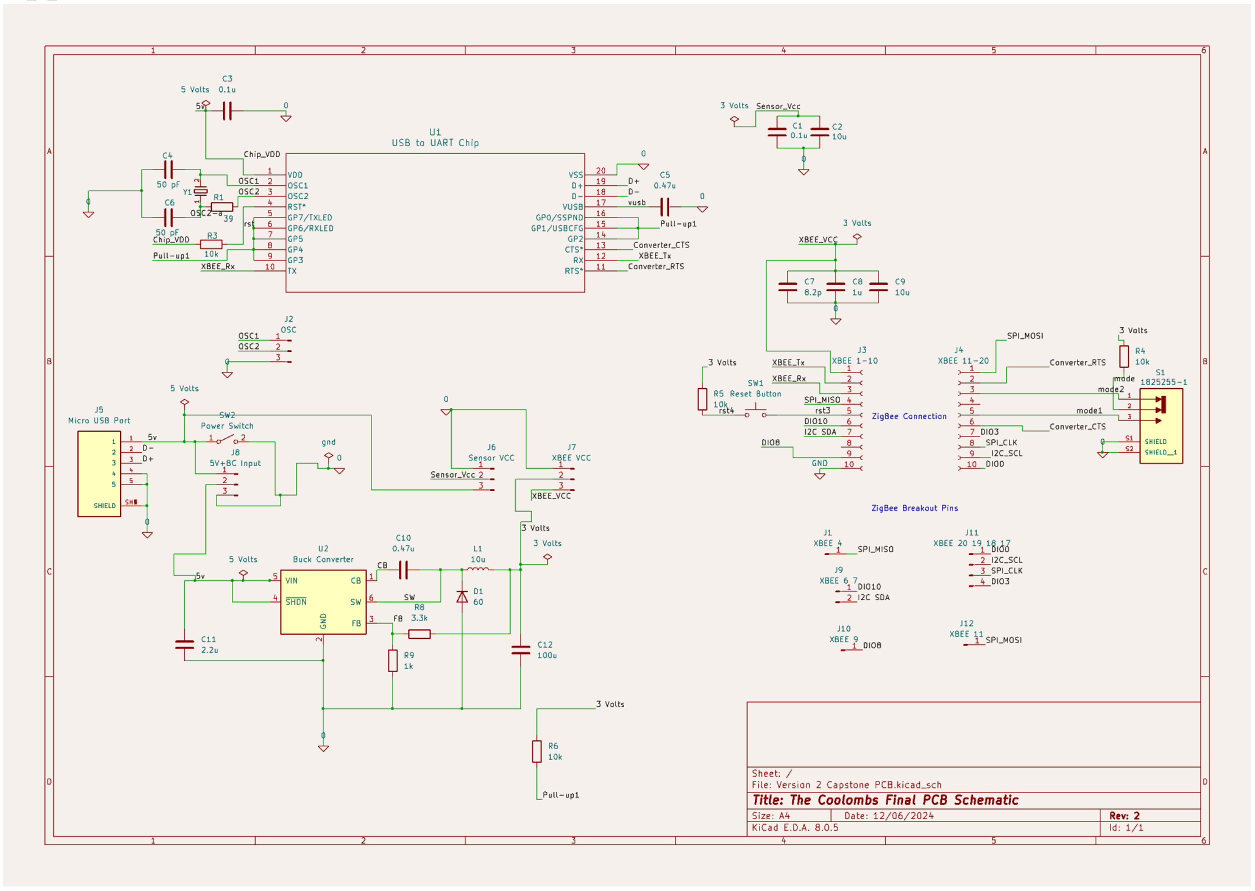


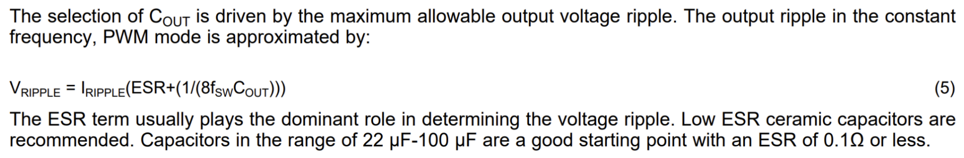
Figure B1. Full PCB Schematic

Appendix C - PCB Test Plan

Buck Converter

1. Physical connection check
    a. Make sure that soldering is done properly: No shorts or loose connections
2. Using WaveForms generator, place a 5 volt input into the Buck converter input(pin 2, header pin J8)
    a. Use supplies tab pos 5V

3. Connect header header pin J8 (pin 3) to ground using Waveform generator
4. The buck converter powers the Microcontroller.
5. Connect a waveform channel to pin 2, J7
6. Using the oscilloscope, measure the voltage of that pin, and check that voltage ripple is under 50mV
7. If Voltage is incorrect:
    a. Check R4 and R5, ensure that the correct resistor values are soldered in
    b. Use formulas found in the Buck Converter data sheet to check if values theoretically produced desired 3V output voltage.
8. If Voltage ripple is incorrect:
    a. Check that the correct inductor value is soldered in.
    b. Refer to formulas given in the data sheet to ensure that values are producing the correct theoretical ripple.



The selection of C_{OUT} is driven by the maximum allowable output voltage ripple. The output ripple in the constant frequency, PWM mode is approximated by:

$$V_{RIPPLE} = I_{RIPPLE}(ESR+(1/(8f_{SW}C_{OUT}))) \quad (5)$$

The ESR term usually plays the dominant role in determining the voltage ripple. Low ESR ceramic capacitors are recommended. Capacitors in the range of 22 μF-100 μF are a good starting point with an ESR of 0.1Ω or less.

9. Place a probe on the SW pin of the Buck converter, and verify that the switching frequency is 1.25MHz.
10. Finally, once all separate component testing has been completed connect the loads(Chip, XBee) to the buck converter to check that they are properly being powered.

USB to UART Chip

1. Connect AD2 to a computer, open WaveForms and use the digital logic sensor in WaveForms
2. Use the digital logic sensor and select UART mode. Match settings to ZigBee UART settings.
3. Send message through Tx line on waveform, if loopback setup is correct, this message should appear on the Rx line
4. Next verify that a full message can be sent through the USB chip and delivered to a computer.
    a. For windows devices drivers for the USB to UART chip can be downloaded here:
        i. https://www.microchip.com/en-us/product/MCP2200
    b. Use the instructions found here (https://learn.sparkfun.com/tutorials/terminal-basics/command-line-windows-mac-linux) to configure your terminal for serial communication
    c. Configure UART communication using the WaveForms logic generator
    d. Send text phrases from Waveforms on the RX line and those phrases should be seen in the terminal.
    e. Send phrases from the terminal and make sure they are seen on the Tx line in Waveforms.

Ada-Fruit BME 680 Sensor

1. Burn sensor
2. Verify sensor is communication via I2C correctly
    a. Configure waveforms for I2C communication using the logic generator
    b. Connect the AD2 to the Sensor, and see that values can be read from the sensor
3. Verify that sensor can correctly detect temperature
    a. Using the same connections described in part 2, take the sensor outside and see that it reads a temperature value within the expected error range given in the data sheet.
    b. Place sensor near an open freezer and see if it can detect a colder temperature
    c. Make note of how fast temperature readings change
4. Verify that sensor can correctly detect harmful gasses

Switches and Buttons

1. The following switches and buttons need to be tested
    a. SW1
    b. SW2
    c. S1
2. For SW1 *Starting with buttons on Breadboard*
    a. Connect one side of the button to pull up resistor to 3 volts
    b. Connect the other side to a voltmeter
    c. Press the button and check that the voltage on the voltmeter is 3V
    d. Release the button and check that the voltage goes back to 0V
    e. Solder button into PCB and Repeat the testing process
3. For SW2 *Starting with Switch soldered into board*
    a. Connect switch to a pull up resistor to 5V
    b. Place a voltmeter on the other side of the switch
    c. Before moving the switch note the voltage. Is it 5V or 0V? This will help determine the on/off positioning for the switch
    d. Slide the switch over and check that the voltage changes. If it was originally 5V it should now be 0V. If it was originally 0V it should now be 5V.
4. For S1 *Starting with Switch soldered into board*
    a. Connect Switch to a pull up resistor to 5V
    b. Place voltmeters on the 2 output pins of the switch
    c. Note which output pin has a 3V output at the initial switch position
    d. Slide the switch over and check that the voltage on one output goes to 0V and the voltage on the other goes to 3V. Slide the switch again and observe the opposite effect.