Rapid Prototyping of Computer Systems ‘24 : ScottyBot

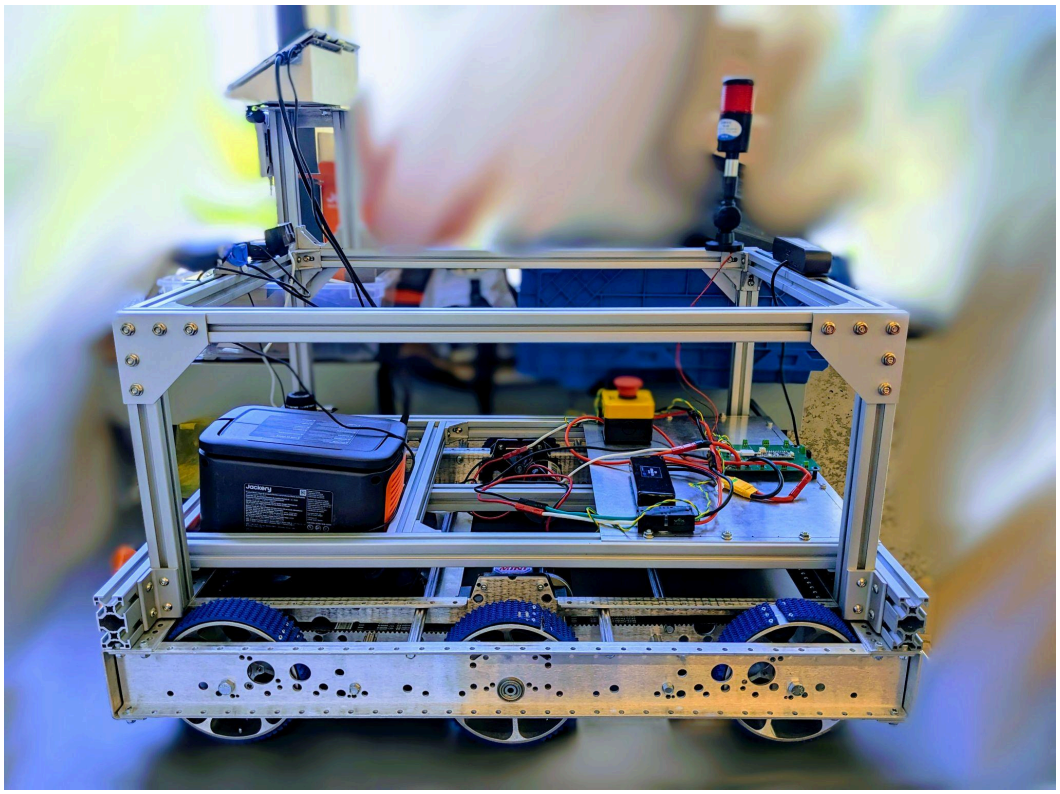


Table of Contents

1. Overview

Objective: Create a LLM-enhanced public robot to solve a mobility problem

Keywords:

- Solving a mobility problem
- Lots of opportunities for robots to provide different services in public spaces.
    - Leveraging the advantage a robot in a physical space provides over a cellphone or other technology.
- Interact with people in socially acceptable ways
- LLMs and VLMs present opportunities for creating public robots that can better navigate public spaces and interact with people
    - Improved scene understanding and plan what actions to take
    - Navigate around obstacles in socially acceptable ways
    - Communicate with people via voice or visual displays
- Explore how generative AI can be used during the engineering and design process.

Developing a Large Language Model (LLM)-enhanced public robot to address mobility challenges represents a significant advancement in leveraging perception systems and artificial intelligence. As robots increasingly integrate into public spaces, there emerges a broader spectrum of potential services beyond the prevalent delivery and map cellphone applications seen today, extending to diverse functions on campuses and city streets. However, the design imperative remains: these robots must navigate real-world complexities while upholding accessibility and social norms in public.

Since the idea of an AI-enhanced delivery robot has been developed and extensively researched, we wanted to create something that does more than just deliver and navigate well.

For these reasons, we will implement and experiment with the idea of an LLM-enhanced public robot, ScottyBot, to solve a mobility problem around CMU campus with potential applicability in various other contexts such as office campuses, recreation areas, and healthcare facilities.

ScottyBot is designed to revolutionize navigation for Carnegie Mellon University (CMU) students and faculty members on the Pittsburgh campus. ScottyBot utilizes advanced language processing capabilities to understand user commands and generate precise audio directions to guide individuals to their desired destinations while leading the user along the way. Whether it's locating classrooms, offices, or campus facilities, ScottyBot ensures efficient and real-time navigation, providing service for CMU's community members with all kinds of

assistance. By harnessing the power of artificial intelligence, ScottyBot embodies innovation at the intersection of technology and campus life, empowering users to effortlessly navigate the dynamic environment of CMU's bustling campus.

2. Problem Definition

2.1. Baseline Scenario

Julie, an undergraduate at CMU, has just finished her class at Gates School of Computer Science and needs to get to her next class in Wean Hall. However, Julie sprained her ankle recently, which makes walking through congested areas and carrying her heavy bags challenging. Without a dedicated service like ScottyBot, Julie has to rely on traditional methods to navigate campus.

She checks the campus map on her phone, trying to find a route that minimizes stairs and walking distance. Julie considers calling a campus shuttle but realizes the wait times are unpredictable, and the shuttles don't drop off directly at classroom buildings. She decides to walk, hoping to find kind strangers along the way who might help her with doors or carrying her bags if needed.

As Julie makes her way, she struggles with the weight of her bags and the pain in her ankle, especially when encountering stairs and crowded areas. The campus's busy sidewalks slow her down, and without real-time traffic updates, Julie finds herself stuck in congested areas, further delaying her progress.

Julie arrives at her class late, feeling frustrated and exhausted from the ordeal. The lack of personalized, immediate assistance highlights the gap in campus accessibility and support services.

Later, another student, Joey, wants to visit a friend's class but can't remember where it's held. Without a service like ScottyBot, Joey has to search through the campus website or ask around, which can be time-consuming and often leads to confusion or wrong directions.

Similarly, Jackson, a blind student, faces significant challenges navigating campus. Traditional methods like using a cane or relying on the occasional help from passersby are helpful but don't offer the same level of guidance and confidence that a dedicated navigation assistant like ScottyBot would. Jackson must plan his routes meticulously, often taking longer to ensure he avoids crowded areas and navigates safely.

At the end of the day, without a solution like ScottyBot, students with mobility challenges or special needs face considerable difficulties in navigating campus efficiently and safely. This scenario underscores the need for innovative solutions that can provide personalized, accessible, and reliable assistance to enhance the campus experience for everyone.

2.2. Key System Requirements

The key system requirements for ScottyBot include navigation and route optimization capabilities, real-time traffic updates, accessibility and personalization, user interfaces, integration with campus infrastructure, and robust robot design.

To ensure that the robot can travel to the required locations autonomously, a key requirement is to have the robot interact with existing building infrastructures, such as door openers and elevator controls. These will be integrated into the existing structures present in buildings to not interfere with their regular operation.

To plan the routes efficiently and to avoid high-congestion areas, congestion monitoring systems at key locations are desirable. These devices should have a small footprint to have minimal impact on the existing infrastructure. Wireless communication capabilities are important so live congestion data can be leveraged for efficient route planning.

Based on its internal map, ScottyBot will navigate itself through an optimal path to reach any required destination within its functional region.

There is also functionally to manually pilot the robot through congested areas with the help of a physical user interface, environmental data, and live-streamed video data. The real time status of the Robots can be monitored with the help of the user interface.

Accessibility is important in this system to provide personalized assistance based on individual user needs and preferences, such as accommodating Julie's sprained ankle or providing specific guidance for visually impaired students like Jackson.

ScottyBot should have an accessible and intuitive user interface accessible via the physical robot or another common platform like mobile. This would allow users to easily interact with the system, request assistance, input destinations, and receive updates.

For ScottyBot to help users navigate complex indoor spaces, there needs to be some level of integration with campus infrastructure. ScottyBot should integrate with building layouts, Wi-Fi networks, facilities and amenities, and certain features in the environment such as elevators.

Finally, to incorporate and provide support for all the aforementioned requirements, ScottyBot needs to have a robust hardware structure with mobility that is up to the task, a power system to support all the components, and a control system to pass the higher-level instructions to the motors. ScottyBot should also have a certain weight capacity to assist users with carrying loads such as backpacks or other school supplies.

3. Initial Solution Concepts

We began to conceptualize solutions to develop an LLM-enhanced public robot. Drawing insights from the key requirements and core functionalities, we carefully selected technologies that align with users' needs and capabilities. Moreover, we prioritized compatibility across platforms with integration in mind during our technology selection process. With these chosen technologies in mind, we began to frame various use cases for our system, and crafted a visionary scenario.

3.1. Table of Selected Technologies

System	Selected Technologies	Purpose it Serves
Robot Chassis	AM14U5 Frame, XL Performance Wheels, 2.5 in CIM Motor	Base Chassis for the robot.
Power System	LiPo 4 Ah 3S	Power for the motors.
	Jackery Explorer 300 Plus Portable Power Station	Power for the NUC.
Control System	Victor SPX PWM/CAN Motor Driver	Motor Driver.
	Adafruit Feather RP2040	Microcontroller.
Navigation (Perception, HCI Intelligence)	2x ZED2i	User tracking, obstacle avoidance, visual odometry, localization.
	4x IR Distance Sensor Short Range-Sharp	Cliff detection.
	NUC	Running models and SDK, connection to cameras and lidar.
	Logitech C922	Streaming.
Route Planning & Optimization	Dijkstra Algorithm	Large scale path planning
	A* Algorithm	Obstacle avoidance
	ROS2	Subscribe to robot attitude updates
	Gazebo	Robot model and simulation environment
	Pybullet	Testing 3D simulation of obstacle avoidance
	MATLAB	Contains ROS toolbox and Optimization toolbox. Integration with Simulink and Gazebo
Natural Language Processing Backend	Microsoft Azure Cognitives Speech Service	Converts textual input into speech/mp3.
	Azure AI Language	Feature extraction with NER.

	Azure Whisper Service	Converts Speech inputs into text.
	Azure GPT Service	Feature extraction with objectives.
	MQTT	Used to send data to P&C and from ID card scan
Data Transport	AWS IoT Core	Connect devices on the cloud across subteams.
	MQTT	Wireless communication protocol. Low bandwidth, pub/sub model, asynchronous.
Data Storage	MongoDB Atlas	A fully managed cloud database service for MongoDB.
	AWS Lambda	Middleware platform to connect to the database and allow serverless computing in response to multiple triggers.
Remote User Interface	React.js	React has strong community support and a rich ecosystem, which can provide extensive resources and third-party tools for developers.
ID Card Reader	ACR122U RFID NFC Reader	Method of authentication: Scan user's ID card and send ID value to HRI's website backend to bypass login page
Live Video Streaming	Zoom	Offers low latency, allows direct peer-to-peer data transfer.
Infrastructure Monitoring & Interaction (HW & SW)	ESP32	Easy to interface with and supports wireless communication
	JSON	Easy to construct and read the JSON message
	AWS	Send and receive JSON message
Software IDE	Arduino IDE	The Arduino IDE provides stable and well-documented libraries for connecting to AWS IoT, ensuring a more

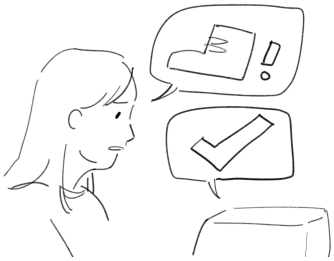
		straightforward integration process.

3.2. Visionary Scenario

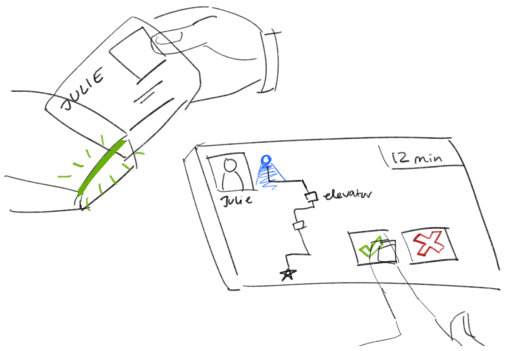
Julie just finished a class in Rashid Auditorium. She needs to get to Wean 7500 for her next class, but she *doesn't know how to navigate there in time*, and she *has a sprained ankle*, so she has trouble carrying her bag and using stairs.

Luckily, she knows about ScottyBot. She sees the robot waiting around Rashid, and says, *"Hey ScottyBot*, can you take me to Wean 7500?" The robot starts listening for "Hey ScottyBot" and processes Julie's input. It says, "I can only go with you until the Wean 4th floor elevators, is that okay?" Julie accepts.

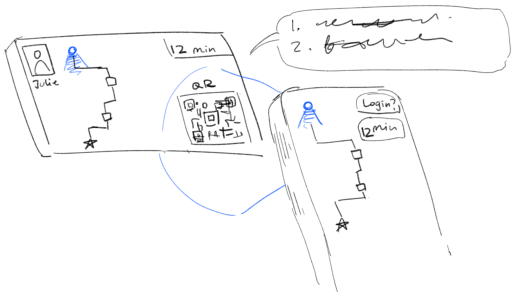
Julie mentions that she has a sprained ankle, and ScottyBot is able to *plan using this constraint*, lengthening the ETA of the trip and noting that it should use a slower pace.



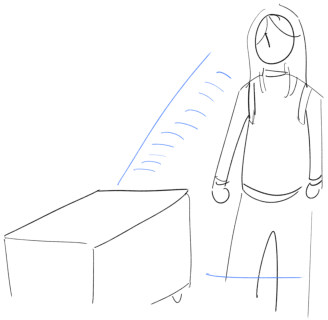
ScottyBot asks Julie to *tap her ID* card on ScottyBot's back to register herself. The robot acknowledges this on the *built-in screen*, and shows the *intended path and estimated arrival time* on the on-robot screen. It confirms with Julie her *intended destination*, and links this journey to Julie's CMU ID.



ScottyBot also generates a *QR code* that Julie can use to open the map on the *ScottyBot website*. ScottyBot asks, "Do you want to make a ScottyBot account with your CMU credentials? This way you can store your preferences in your profile." Julie declines for the time being. ScottyBot will use a website, rather than an app. It will rely on authentication via CMU ID, then CMU login info if the student wants to make an account.

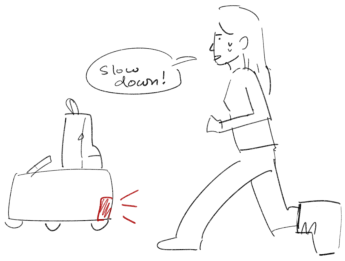


The robot asks Julie to stand in front of it so it can scan Julie's body to track it while on the journey. Julie agrees. In addition, ScottyBot will use body tracking software to enable a better dynamic between human and robot through the journey.

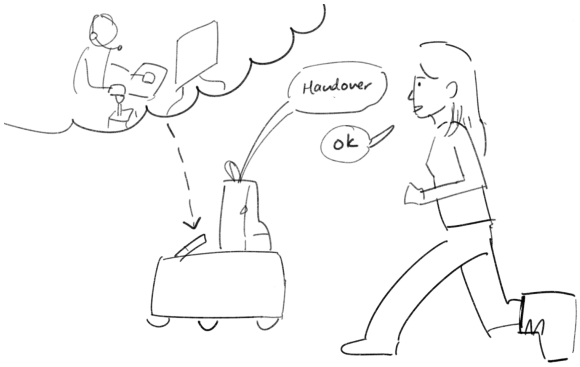


Julie *lays her bag* on ScottyBot, and it begins to lead the way, keeping track of Julie behind it.

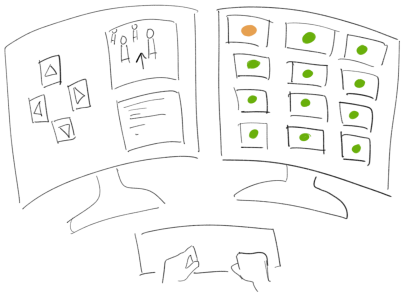
Julie finds that ScottyBot is going too quickly and her ankle is hurting, so she *tells the robot to "slow down."* The robot *processes her natural language input,* pulsing a light to show that it is *thinking about this text*. It then replies, "ok", and telegraphs this action by *lighting up its back LEDs red*, like a car might.

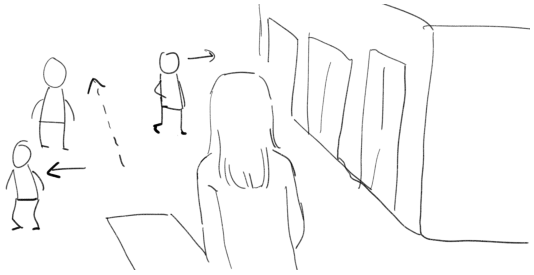


Julie and ScottyBot approach the Wean 4 elevator bank. Through the *sensors around Wean 4, Mission Control is notified of high congestion, and an operator takes control* of the robot. Julie is *notified* of this vocally, and through the screen/app.



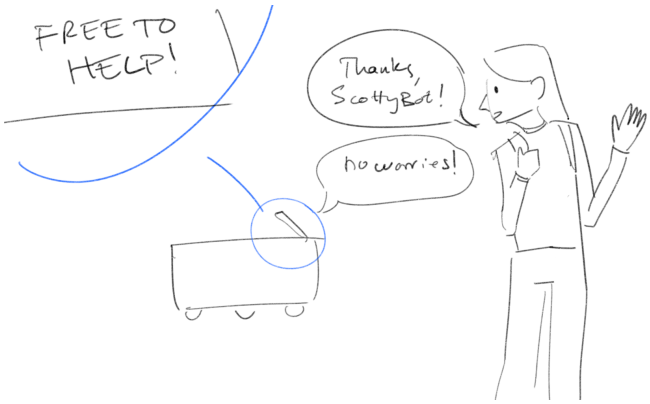
Mission Control is able to pilot the robot with *live data streams* from the *on board camera* and *distance sensors*, integrating this information with *map* and *robot pose data*. They control the robot with the *keyboard on their PCs*.



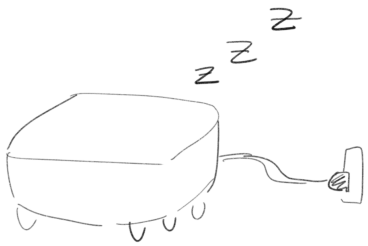


At this point, Julie is *notified* that control has transferred back to ScottyBot. It *calls a Wean elevator* to take Julie to Wean 7 with its integration into the elevator architecture.

ScottyBot says, “I can’t go any further than this, but to get to Wean 7500, just exit the elevator at the 7th floor, and it’ll be right in front of you!” Julie thanks ScottyBot, which ends the journey. ScottyBot’s screen reads: *FREE TO HELP!*

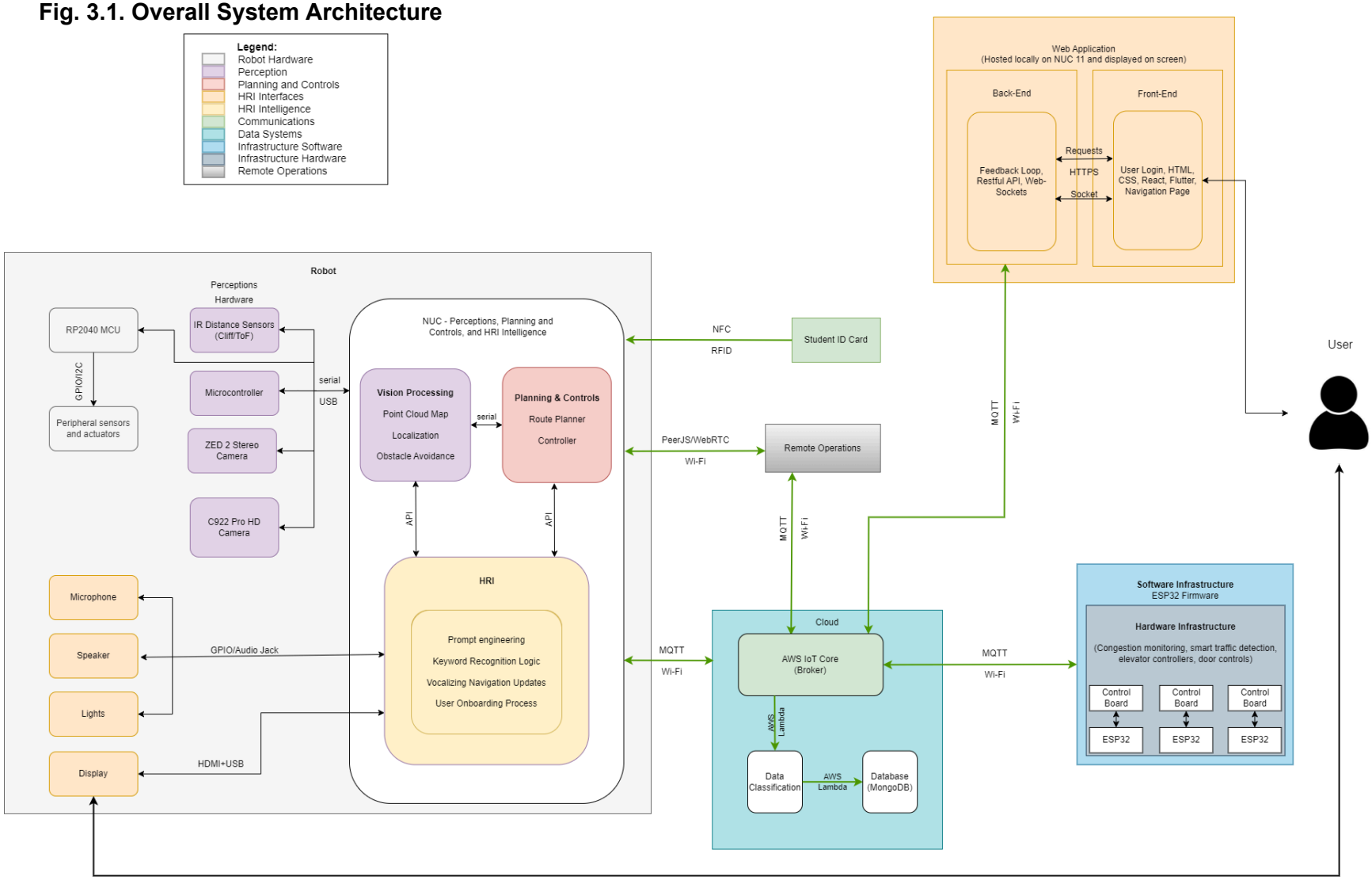


At 8PM, ScottyBot ends its shift. It navigates to its resting location, where *mission control plugs it in for the night.* Sweet dreams, ScottyBot.



4. System Architecture

Fig. 3.1. Overall System Architecture



5. Subsystems/Subteams

5.1. Robot Hardware

5.1.1. Functionality Overview

To fulfill the vision of ScottyBot, a robot designed to aid with campus mobility and navigation, the Robot Hardware team has taken on the responsibilities of developing the physical platform upon which the other teams will integrate their higher-level technologies, as well as the power and control platforms that are essential for the implementations and interactions between the technologies.

5.1.1.1. Mechanical Requirements

The Mechanical framework of the ScottyBot is required to have a robust chassis with reasonable payload capacity and maneuverability, which allows for other teams to physically mount their technological components onto the robot and efficiently implement the navigation efficiently and reliably in diverse and specific scenarios while possibly carrying a payload to meet the users' needs. This chassis design needs to combine durability with the flexibility necessary to handle various outdoor and indoor conditions, from flat surfaces to ramps. However, it is not expected that ScottyBot will go off-road, sticking to paved pathways.

As for the size and the payload capacity of ScottyBot, while the chassis is required to be relatively large, it must also not be too large so that it may cause blockage in the passageways or other pedestrian traffic issues on campus. We need a compact design that can physically support all the electronics, computing platforms, and sensors, but still maintain dimensions that do not exceed half of a typical passageway in the applied scenarios. ScottyBot's payload-carrying feature allows users to transport belongings or packages, for supporting and alleviating mobility difficulties. The chassis will be engineered to support at least some of the weight and bulk of textbooks, laptops, and other personal items without compromising its maneuverability or speed.

All these have led us to the choice of the Andymark AM14U5 chassis, which is large-scale and able to carry large payloads, while still being highly modifiable structurally. The technical specifications of our chassis will be discussed in the later sections. We also have planned on more physical features that will be incorporated into the overall structure of the ScottyBot to facilitate the other teams' design needs and ultimately meet the application requirements in the visionary scenario.

5.1.1.2. Electrical Design

Implementing a robust power supply and management system that ensures ScottyBot can operate for extended periods with a consistent performance without frequent recharges is crucial. This is also extremely important for all components of the robot's on-board system to operate smoothly and correctly. To achieve this, we have included the use of rechargeable batteries capable of sustaining long durations of

activity, and circuitry designed to regulate power supply to the necessary voltages for different electronic components, while ensuring the safety and functional isolation of all the electrical components to prevent any potential hazards.

Furthermore, our responsibilities include housing an array of different interactions across the potentially integrating screens and others that display real-time information, system statuses, and navigational prompts. Essentially, ScottyBoth will offer both visual and auditory feedback through these interfaces to facilitate user interaction and enhance the overall user experience. We will need to supply feedback data to the higher-level systems to enable these functionalities, as well as directly implement some simple functions like the controlling of the LED lights.

It is worth noting that we came to realize most of the electronic components and sensors can be directly powered by the computing device through USB ports, which significantly simplifies the design of the electrical system.

5.1.1.3. Control System

In the visionary scenario, ScottyBot navigates through Newell Simon and Wean, which are experiencing a lot of foot traffic. In these kinds of situations, ScottyBot needs to be aware of its surroundings and move away from nearby objects or walls when required, as well as navigate around foot traffic without colliding into people. In other situations, ScottyBot also needs to avoid navigating into any sudden drops, such as stairs and floor dents.

At the heart of ScottyBot's operation is the Microcontroller Unit (MCU), tasked with executing low-level motor control commands, safety sensing, and interfacing with higher-level software for navigation and user interaction. We will select and integrate an MCU that meets these demands and provides a way for the Planning and Control team to control ScottyBot's movement, ensuring responsiveness and adaptability to user needs and environmental changes.

To ensure ScottyBot can safely navigate through crowded spaces or avoid incoming obstacles, advanced sensors, such as ZED 2i stereo cameras are required to be in the both front and the back of the robot for Perception team to implement their algorithms for obstacle detection and collision avoidance. For cliff detections, cliff sensors, and other sensors are also needed onboard. We will implement the base-level safety hardware that enables ScottyBot to react swiftly to unexpected situations while deferring main navigation and collision avoidance to the Planning and Control and perception teams. This will allow ScottyBot to maintain smooth operation even in crowded and complicated environments.

5.1.2. Robot Chassis Design

We are working with the Andymark AM14U5 chassis as the base and framework for the physical designs of ScottyBot. In Fig 5.1 and Fig 5.2, we show the dimensions of the base chassis. The chassis is a large square configuration consisting of six 5052 Aluminum plates and two 6005A-T61 Aluminum churros, completed with two ToughBox Mini gearboxes and six 6IN XL Performance Wheels. We are using the “drop-center” configuration, meaning that only the middle wheels are directly driven by one 2.5 in. CIM Motors while the other wheels, positioned at the front and back ends of the chassis, are not directly powered by their own motors. Instead, they are connected to the middle, motor-driven wheels by belts. The maximum payload of the chassis can vary largely depending on the materials and structures. In our estimation, with the higher level structure and material we are using, we are optimistic it can satisfy the payload requirements of carrying the users’ personal items on top of housing all the other components, which can be up to 10 kilograms.

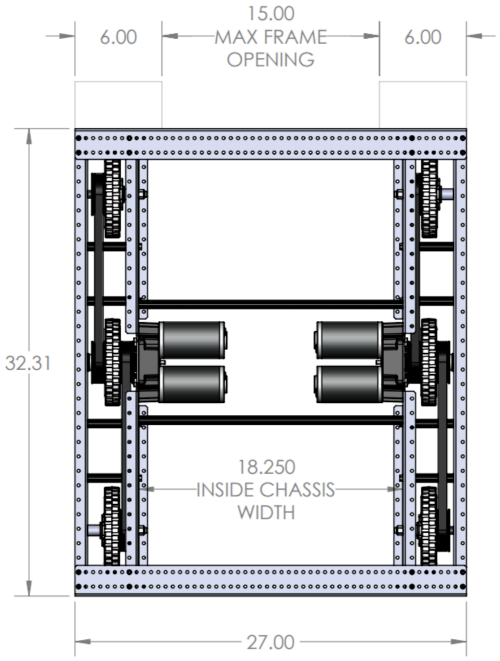


Fig. 5.1 AM14U4 LONG Total Frame Top View with Parameters

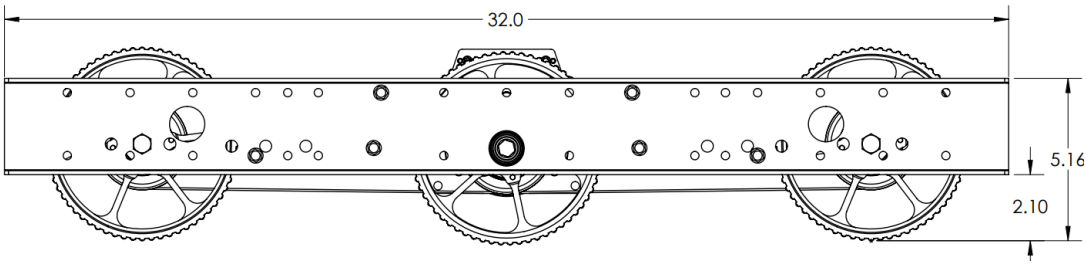


Fig. 5.2 AM14U4_PW6XL Layout Side View with Parameters

From phase 2, the team has decided on using 30 series aluminum extrusions from the 80/20 company to construct the top part of the robot. These extrusions, measuring 30 mm in thickness, ensure a blend of lightweight construction with the necessary strength. A rectangular frame forms the core of the expansion, with two 3060 series beams along the front and back securing the new segment to the AndyMarks base. This addition is projected to raise ScottyBot's height to 45cm (17.7165 inches), while its other dimensions remain unchanged.

The robot's upper sections are reinforced with three horizontal beams and a vertical beam, providing a sturdy support structure. According to our preplanned design schematics, ScottyBot features three distinct platforms. The lower back platform of the chassis is reserved for weightier and larger components like the battery pack station and DC power regulator, as well as the NUC computer which acts as the brain of the operations. Situated just above the base at the front, the middle platform houses mostly lighter electronics crucial for actuator control, such as the control board, the motor drivers, and the motor battery. The topmost platform, which is supported by two beams on top of the frame, acts as the mounting point for a custom 3D-printed holder for the user-interactive touchscreen. The sensors utilized by the perception team and the remote operation team, including high-definition cameras in the front and the back, are mounted directly on the upper frame.

Previously we planned to have the payload held in a basket that is firmly anchored to the horizontal beams toward the back of the chassis, as seen in fig.5.3, with an acrylic plate installed underneath to restrict access to the bottom platform once development ceases. For the prototype we did not have ample physical space for a full-sized basket after mounting all the components, but the load carrying function is viable in future development.

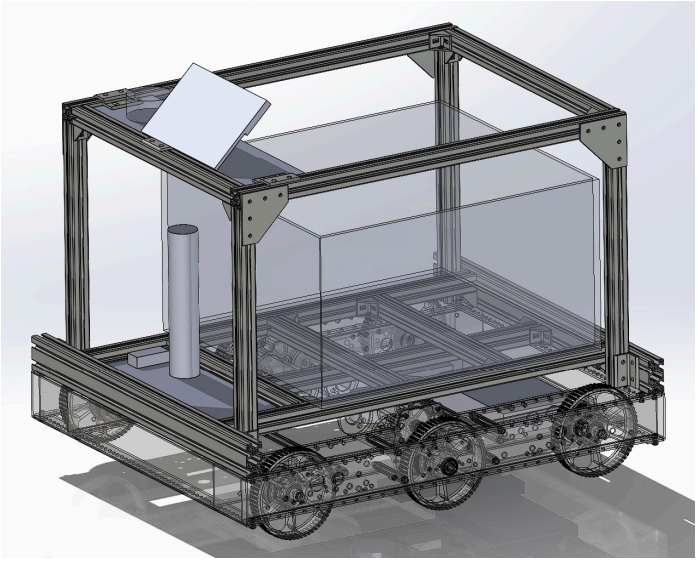
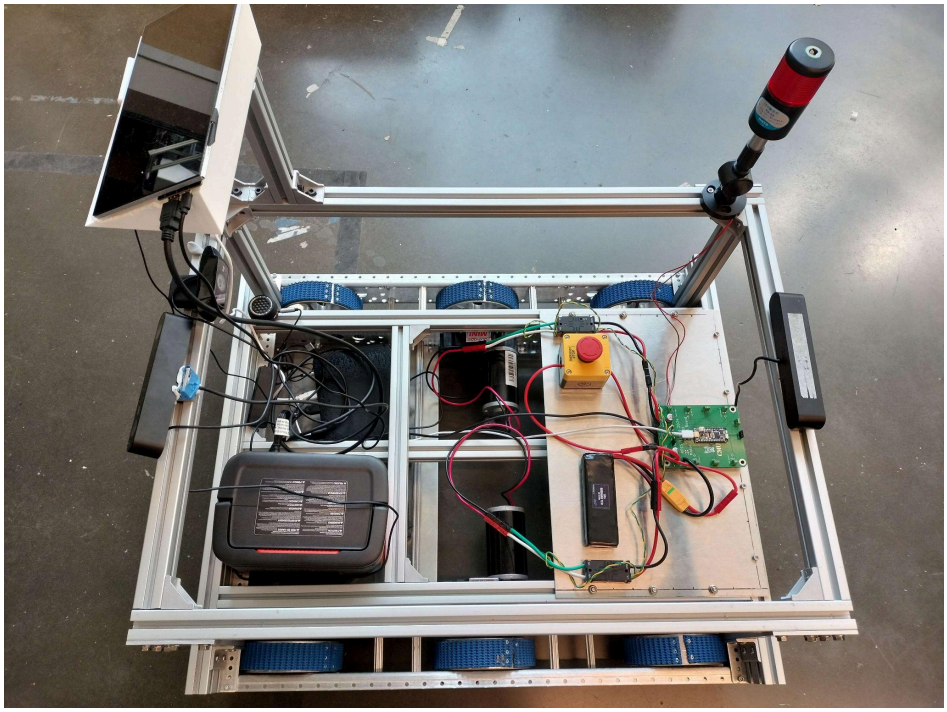


Fig. 5.3 Complete CAD Model of ScottyBot



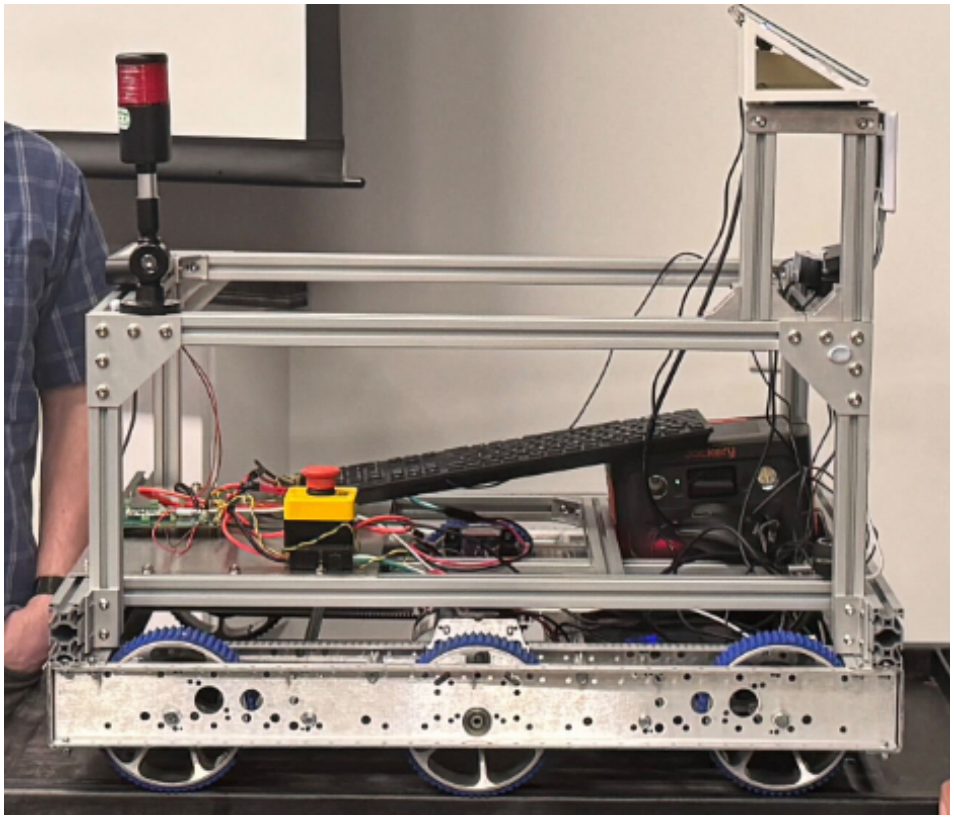


Fig. 5.4 Top and side view of the final chassis design, with the screen mounted on 8020 beams in a back corner.

5.1.3. Control Subsystem

The control subsystem uses an embedded microcontroller (on a Feather spec dev board) to sense the environment through multiple sensors on the chassis, and actuate motors and LEDs to move the robot and interface with the user. This subsystem gets instructions from Planning & Controls, Remote Ops, and the HRI Interfaces teams via the NUC for its actuation. Although we planned on sending collected sensor data to the Perception team to further improve their perception capabilities, there was not enough integration time for this data to be useful to them. Motor control inputs from upstream teams are mapped from an 8-bit integer to a pulse between 1-2 ms to control each motor on ScottyBot using the Victor SPX motor drivers. LED states are also similarly communicated by the HRI Interfaces team.

Data from the distance and cliff sensors is processed onboard the MCU to avoid the robot from driving off an edge (such as a staircase) or running into an obstacle, to prevent unsafe actions not caught by the Perception team's sensing. Inputs from these sensors would overwrite inputs received from higher-level software subteams, while also potentially flagging errors to the NUC to indicate problems in the detection of obstacles or cliff edges. The control loop behavior is outlined below in Figure 5.9.

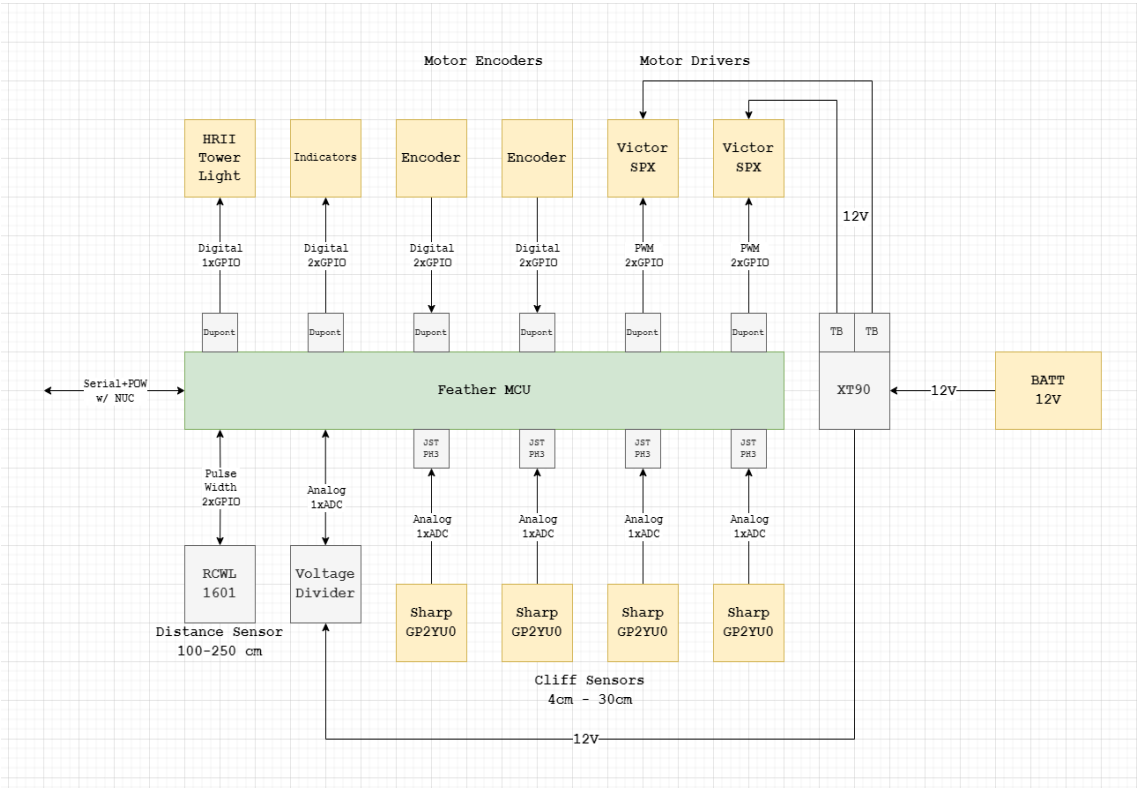


Fig. 5.5 Control System Architecture

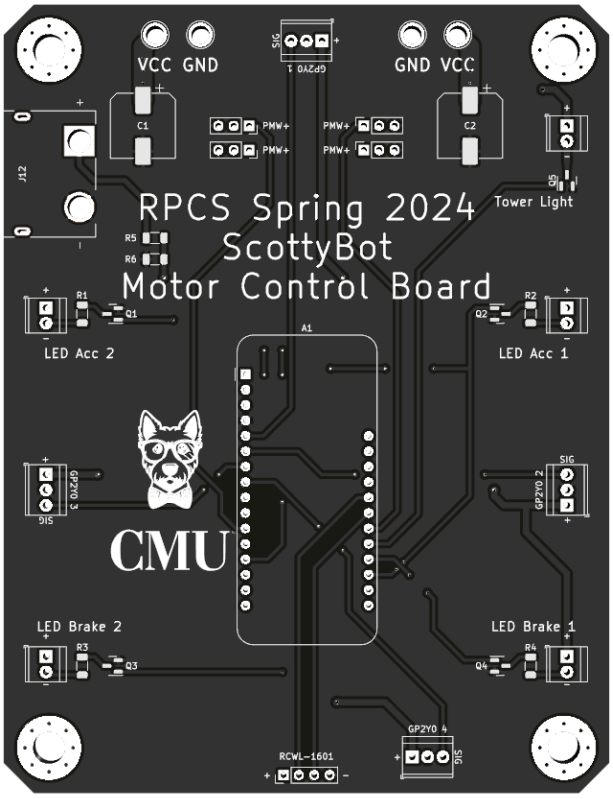


Fig. 5.6 Motor Control Board

OPCODE = 1	Left Motor Speed	Right Motor Speed	LED Values
1 byte	1 byte	1 byte	1 byte

Figure 5.7: REQ message definition

OPCODE = 2	Cliff Sensor 0 Value	Cliff Sensor 1 Value	Cliff Sensor 2 Value	Cliff Sensor 3 Value	Distance Sensor Value	Battery Level
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte

Figure 5.8: DATA message definition

To facilitate communication between the NUC, the entry point of all other teams communication, and the RP2040 microcontroller unit run by our team, we created a serial communication interface based on UART to enable a wired communication protocol to relay messages to and from the NUC and the microcontroller. Serial communication was chosen due to its ease of use and low overhead, and there exist

many libraries on both the NUC side (such as pyserial) and on the microcontroller side, with Arduino's built-in support. This message interface went through a few iterations as the team figured out what data other upstream teams depended on for their own purposes, such as motor speeds and distance sensor data for planning and controls, as well as keeping the interface simple and understandable. In addition, requirements gathering and research were done on the sensors themselves, in order to determine what range of values would be returned from sensors and thus inform how much fidelity would be returned through the interface's data width. Originally, there were more message types, but due to the simplicity of the interface and the need to reduce the overhead of sending messages, all of the data types were combined into a single request and single response message. This approach allows for easy processing logic on the microcontroller side, and reduces the overhead of packing messages and reduces packet congestion. However, in the future, if more data dependencies arise, the interface can be adjusted to allow for more types of messages such as acknowledgements, errors, and even logging messages.

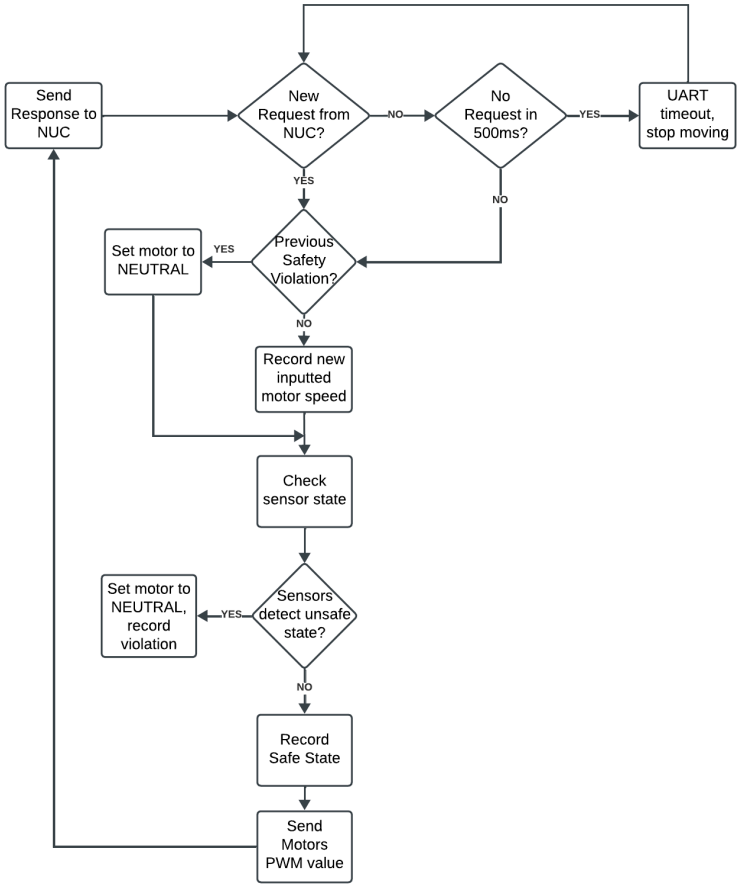


Fig. 5.9 Control System Architecture

The microcontroller logic for robot movement was fairly simple in scope in order to be efficient and performant, but still had an emphasis on safety, by ensuring in each movement action the robot was in a safe state to execute said actions, as seen in Figure 5.9. For example, any time a request from the NUC was received, the control loop would check if there was a recorded safety violation previously, and if there was, it would reject the action without an overriding command from the NUC, which would mean the upstream team had acknowledged the safety issue. Otherwise, in every control loop iteration, based on sensor readings from the distance and cliff sensors, the robot would determine if it is in a safe state before finally executing the motor action, or else it would stop itself. Furthermore, if there is no input received from the NUC within a certain time frame, it is assumed that communications were lost and the robot would stop moving to ensure it is not executing stale commands. Through these two safety approaches, we ensure that the robot is executing safely within the confines of our sensor suite. It should be noted that the final demo did not make use of some of the sensor suite, such as cliff sensors, since there were no cliffs along the path. The distance sensor header also broke off the board at one point, making it inoperable. However, the code logic for these sensors still exists within our repository.

5.1.4. Power Subsystem

There are currently two distinct systems to manage the power, namely the NUC power components and the motor power components. In this way, there is a separate battery pack powering the NUC and all peripherals connected through USB, and another battery pack powering only the motors. The reason behind this is twofold: to isolate each battery component, and to simplify the design of NUC power system to require a single buck converter. This prevents the need for any voltage divisions and allows increased safety through reducing possible power hazards which may occur through fluctuations across independent systems due to effects such as motors trying to overcome stall, etc.

Additionally, a few key safety features were added to the power system. Namely, an emergency stop button as well as connectors to all of the cables, in order to facilitate safe connect/disconnect functionality. The emergency stop is in series with the battery input, which allows it to be a central control for whether current is being delivered to the batteries.

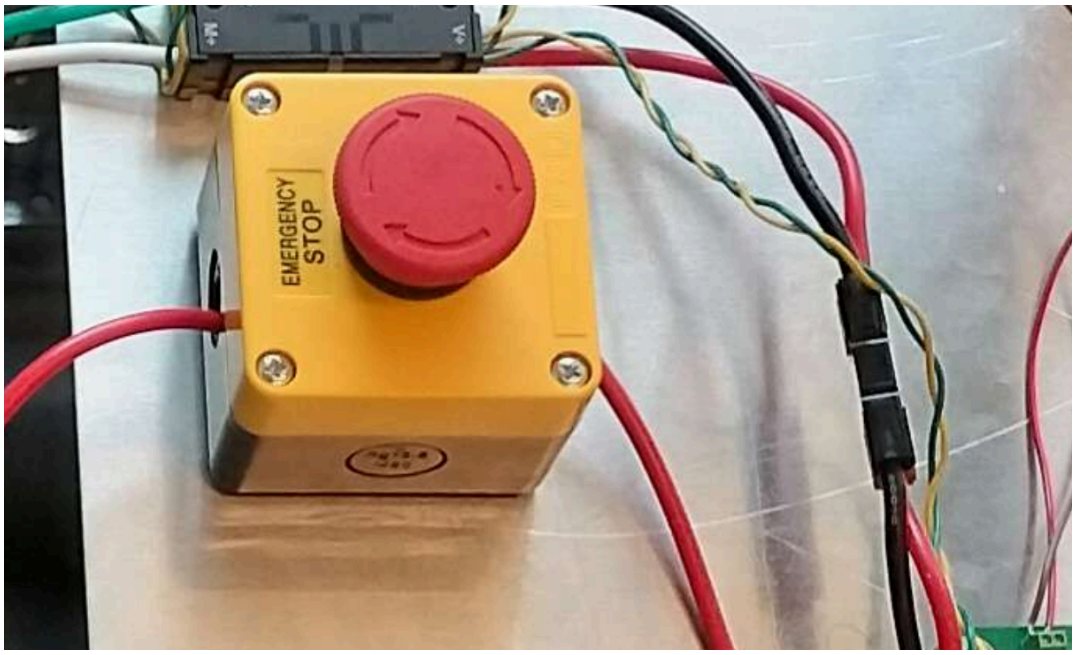


Fig. 5.10 Emergency stop button, in series with the motor battery's connection to the motor control board.

On top of this, the power system for the NUC was simplified. This is because the additional complexity of using a buck converter and barrel jack, although planned out, was delayed due to the time required for rapid ordering of the appropriate parts, i.e. the correct size barrel jack for the NUC. Additionally, the buck converter "looked kinda sus" and thus, the team resorted to a safer approach suggested by someone in a meeting. The newer approach was to use a portable power supply in order to deliver A/C input to the NUC's proprietary power adapter, which would then power the NUC directly. This had its own challenges: now, we had lost some space that was to be used for the payload, since instead we had this sizable chunk of a power supply. Furthermore, this power supply required charging which sometimes prolonged testing runs. Each charge to a 100% level allowed the NUC to run for approximately 1-1.5 hrs of continuous runtime. The advantage, however, was the safety of using a rated, certified power supply instead of waiting on an unreliable barrel jack which might not have connected to the NUC, thus delaying testing to the last week altogether.

Fig. 5.11 Portable Power Supply used to power the NUC, note the reduction in space which would have been used to place a basket to hold the payload.

5.1.5. Team Deliverables

Our team successfully designed and constructed a functional chassis that serves as the physical platform for ScottyBot. Utilizing the robust Andymark AM14U5 chassis,

we achieved a design that supports not only the mechanical stability required for navigating on campus but also the flexibility to accommodate various technological components from other teams. This chassis ensures maneuverability and payload capacity, allowing ScottyBot to carry personal items like backpacks or laptops. The integration of the 80/20 aluminum frame further strengthened the physical structure, providing base architecture and platforms that can house sensors and interactive components.

We also developed a control interface that integrates an MCU to manage motor control, safety sensing, and interaction with higher-level software for navigation and user interaction. This system is designed to handle complex environmental interactions smoothly, such as navigating through crowded areas. We adapted lower-level safety measures like cliff sensors and a front-facing distance sensor, although not showcased in the demo due to time constraints. The control interface utilizes UART for reliable communication between the NUC and the MCU so that ScottyBot responds to user commands and environmental conditions effectively and safely.

Moreover, our team implemented a power supply system capable of supporting the operations planned. This system includes a LiPO battery pack for the motor components and a portable power station for the NUC, which also provides power to other higher-level components, to provide isolated power management and overall system working efficiency. The use of a portable power supply for the NUC ensures that ScottyBot has a reliable and safe power source for intelligence operations. For extra safety and controllability we added the emergency stop feature and easy connect/disconnect functionality safety.

5.2. Perception

5.2.1.1. Visionary Scenario

Perception is responsible for the three main components: localization, user tracking, and object detection for collision avoidance.

We narrowed down the scope for localization according to the CMU cabot map which encompasses Gates, Newell Simon Hall (NSH), and Wean fourth floor. The justification for this decision is so that we would not have to rescan the building to create our own map. Another reason is that we will have to deal with the situation of two glass bridges: the Gates to NSH and the NSH to Wean bridge where Lidar can not accurately detect these areas which will be an obstacle for navigation. Another aspect is that we will need the user to now stand in front of Scotty Bot for a user initialization stage so that we can extrapolate the body for us to begin user tracking. We asked for HRI Interfaces help with directing the user to press a button to complete this process. We have committed to using ROS2 humble because it is compatible with many of the technologies that we bought such as the ZED 2 camera. As for cliff detection, that was previously our responsibility, we now have handed it over to Robot Hardware as it would

be best to directly have the sensors be used with the microcontrollers for use instead of going through our software stack to send to Planning and Controls.

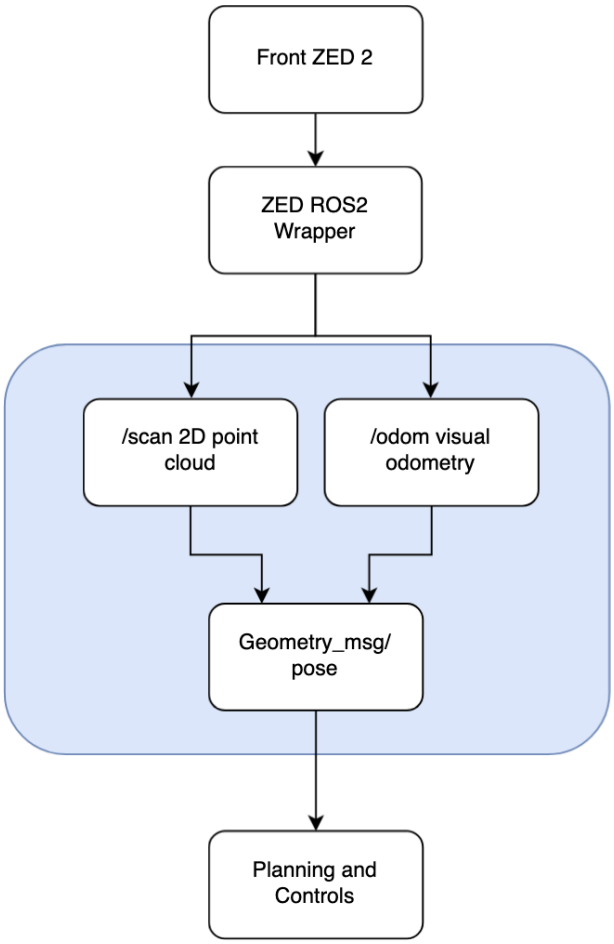
5.2.1.2. Technology Survey

From the perspective of perception, the robot's perception ability can be categorized into four parts, including cliff detection, localization, collision avoidance, and object tracking. For the prototype, our robot will mainly operate on the fourth floor of Wean Hall, Gates Hillman Center, and Newell Simon Hall on the Carnegie Mellon University campus. We limit our operational environment to these areas currently because current 2D point cloud scans already exist for these locations.

5.2.1.3. Localization

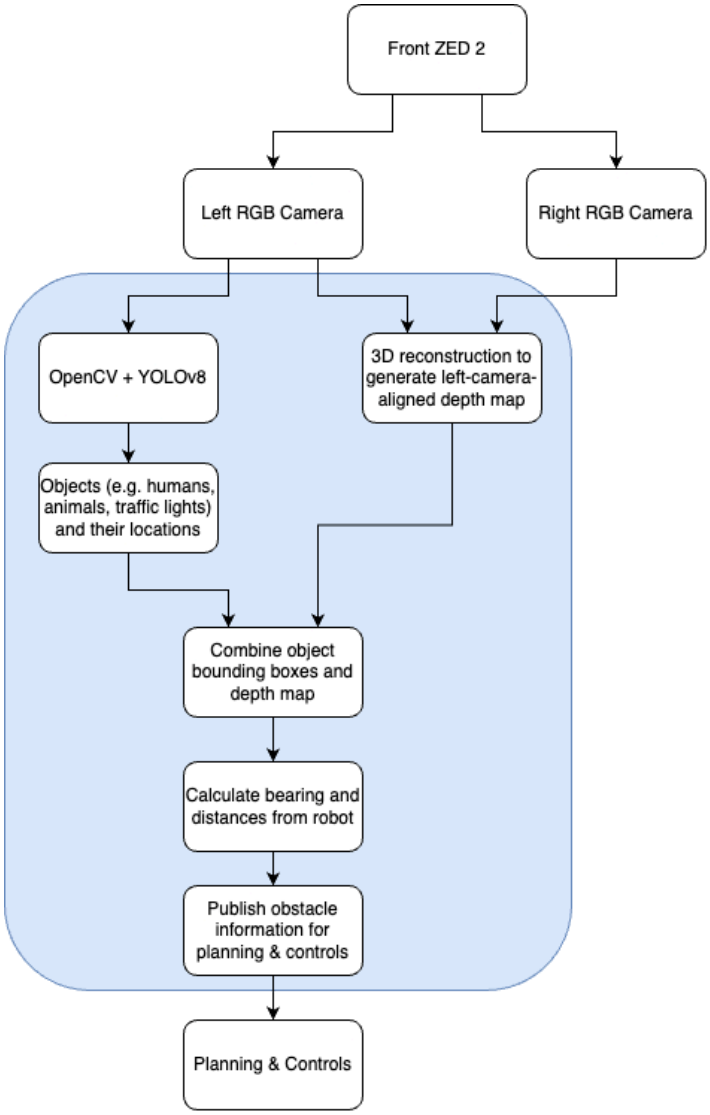
The robot needs to locate its own position in the environment. We initially selected SLAM as an option but due to the prohibitive cost of 3D LiDARs, we then selected VSLAM. However, the complexity of VSLAM is quite high and given that we're only using one computer for on-robot processing (shared between multiple teams), it is infeasible to run VSLAM. Additionally, we have been instructed to not do our own mapping, as a result, we settled on using a 2D LiDAR and existing 2D point cloud maps for localization. 2D LiDARs offer a robust and cost-effective solution for localization in robots, suitable for our prototype robot operating on campus. Utilizing LiDAR for robot localization involves comparing real-time LiDAR scans with pre-existing maps to determine the robot's position and orientation within an environment. We have explored a few algorithms to accomplish this including the Adaptive Monte Carlo Localization (AMCL) and Google's Cartographer.

Unfortunately, during our testing of the LiDAR, we are unable to detect glass walls and windows. As a result, for sections of the campus especially in Gates and the bridge between Gates and NSH will not yield the performance and accuracy we require. This is especially true when the hallways in Gates are only big enough for the robot to have about 6" of clearance on each side. Therefore, we reverted back to using the cameras to localize. The ZED API provides us with a method of creating a map and then using that map for localization. Over a distance of 20m, we got an error of about 20cm when compared with the ground truth (as measured by a Bosch laser rangefinder).



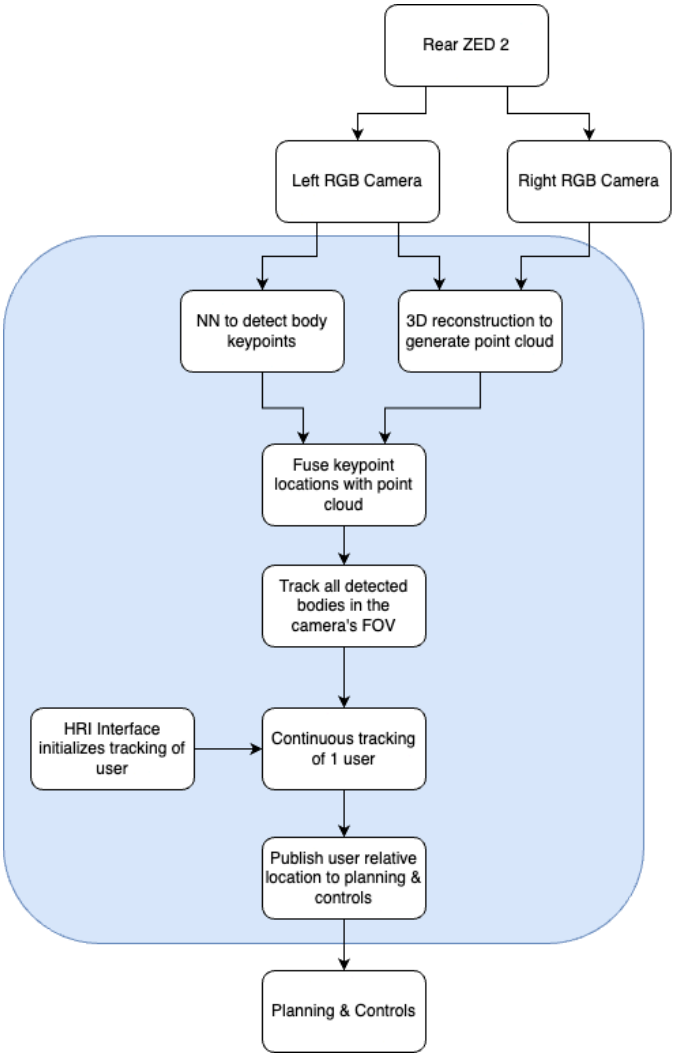
5.2.1.4. Collision Avoidance

Collision avoidance, as part of the 3D sensing suite, ensures the robot will not run into any obstacles, including the user, walls, and other pedestrians. We choose to use a stereo camera to achieve this purpose. Stereo cameras, consisting of two cameras spaced apart, mimic human vision by capturing images from slightly different perspectives. This setup enables depth perception through triangulation, allowing the system to determine the distance to objects in the scene. Using stereo vision, robots can detect obstacles in their path and calculate their distance from the robot. By analyzing the depth information obtained from stereo images, the robot can navigate around obstacles to avoid collisions. We intend to output all relevant obstacle positions to the Planning and Controls team for their consideration. Unfortunately due to time constraints, although we were sending the data, it was not incorporated into dynamic path planning.



5.2.1.5. Object Tracking

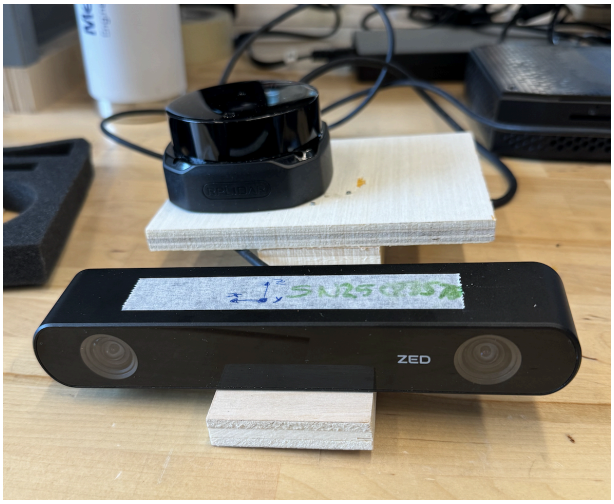
While navigating for the user, the robot needs to maintain the connection established with the user to prevent the loss of the user, and at the same time maintain a certain distance from the user to ensure a safe distance. The solution we propose utilizes 2D sensing with a monocular camera. The monocular camera captures 2D images of the environment from a single viewpoint. Object tracking involves analyzing consecutive frames to detect and follow objects of interest based on visual cues such as color, shape, or motion. We explored multiple different algorithms, including BOOSTING, Lucas-Kanade, and Discriminative Correlation Filter with Channel and Spatial Reliability (CSRT). We believe that CSRT will provide the best performance for our purposes due to its robustness when the object is partially occluded.



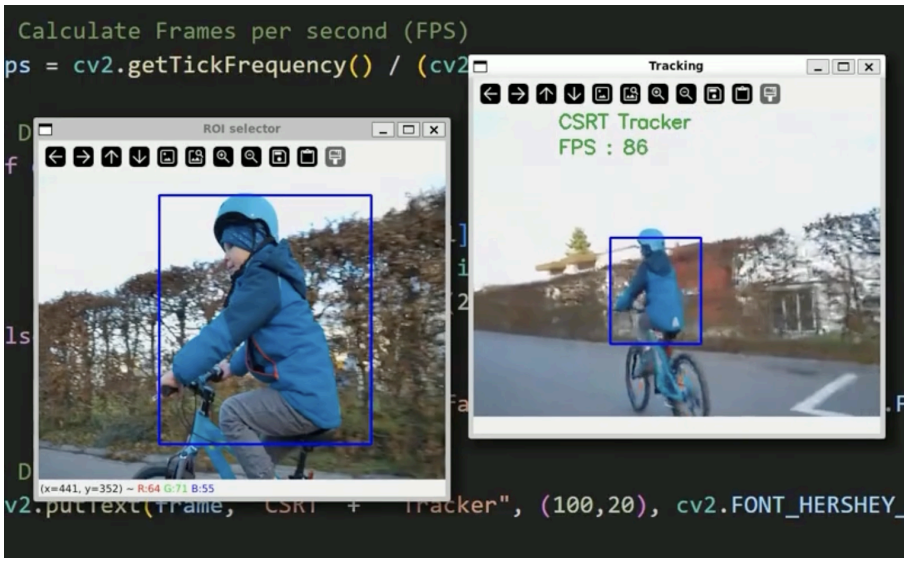
5.2.2. Team Deliverables

For the final robot we were asked to provide four main features, (1) User Tracking, (2) Obstacle Detection, (3) Localization/Pose Estimation, and (4) Streaming. After receiving all our hardware, we worked on setting up the NUC and registered the device with CMU. After setting up the network, we worked with the ZED2 SDK to test out all the capabilities that it has. We found that the most vital components for our purposes are point cloud, body tracking, depth map, and positional tracking.

To facilitate the testing of the LiDAR and camera system, we built a simple wooden structure to couple the two together. This test platform pictured below mimics the coupled-motion of LiDAR and camera to ensure that the visual odometry we get from the ZED camera is accurate.



For user tracking (1), we implemented the CSRT tracking with the help of OpenCV packages. The performance of CSRT is quite high and the computational requirements are quite low. It is able to handle occlusions and changes in lighting quite well. The figure attached below is a screenshot of CSRT in action. The left image shows the initial bounding box configuration and the right shows us CSRT at work given the initial bounding box configuration. When we weighed our options and compared this with the ZED SDK, we realized that the ZED provides us a method of doing body tracking already. Ideally, we would use the CSRT module, however, given that for our demo, we are only ever tracking a person who is standing up, as opposed to someone in a wheelchair for instance, the ZED body tracking module will also yield acceptable results. As a result, we ended up using the ZED body tracking module.



We implemented obstacle detection (2) with the neural network (NN) using YOLOv8. By using the camera feed to get bounding boxes around people and other obstacles, we are able to fuse the RGB bounding box locations with the depth map provided by our stereo cameras. From this we are then able to use the calibrated

camera intrinsics and the pinhole camera model to calculate the bearing and distance of the obstacle to our camera plane.

For localization and pose (3), we used ZED's visual odometry to map out the area in which we were testing. There were a few maps that we scanned and we determined the best was in Gates where the demo took place. From there we used the ZED API to determine Scotty Bot's pose in relation to the map. We configured a transform matrix to align this map with the cabot map pgm file that Planning and Controls was using for dynamic path planning.

Below is a screenshot of the message containing the position and orientation that we published to the /slam_pose topic which Planning and Controls subscribed to and took this information to determine pose on the map.

```
position:
  x: -3.421552763014629
  y: -7.104174947867413
  z: 0.0
orientation:
  x: 0.08
  y: -0.049
  z: -0.641
  w: 0.762
---
```

In addition, for the functionality of the user initialization stage, we decided to set up a local stream (4) that HRI Interfaces could subscribe to and be able to display on the screen. This is so that the user can stand within a bounding box and click on a button that would "lock" onto this user. It will continue to track them until the end of the trip with Scotty Bot. This locking mechanism is done through ROS2 services where HRI Interfaces provides a button on the website for the user to press and it will call a command line that will lock onto a specific user to track. The local streaming which is done through a simple Transmission Control Protocol (TCP) that opens up a socket over an IP address to send over the ZED camera feed.

Where we originally had concerns about computation expense, we decided to run the software and check the CPU and GPU usage. We found that when running every subteam's code on the NUC, about 70% of all the CPUs were being used and for the GPU it was about 56% utilization rate. As we are the primary subteam using GPU, we found that this would be adequate once other teams place their softwares on the NUC.

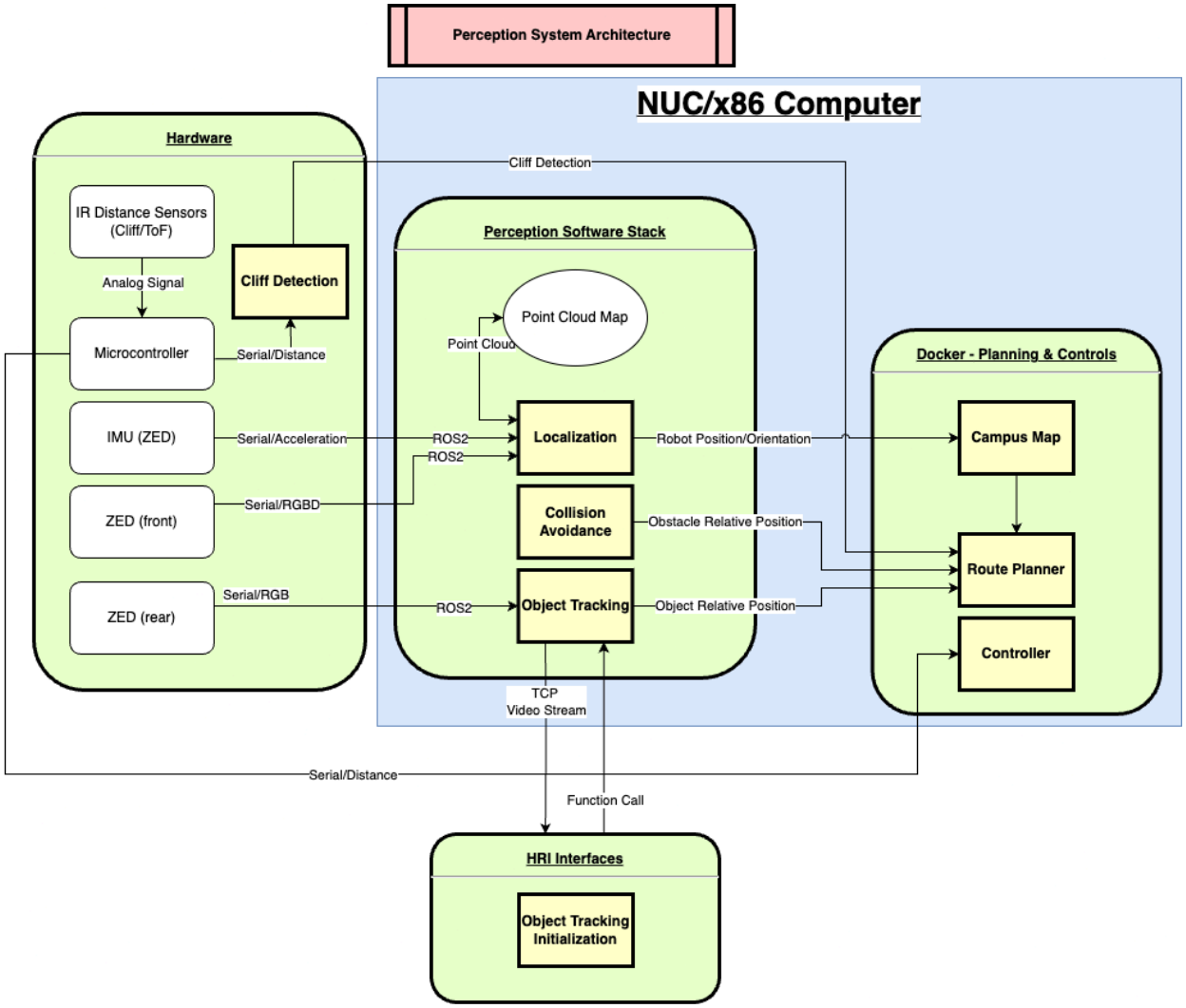
5.2.3. Dependencies

Teams that are most directly dependent on us, Perception, include Planning & Controls, Robot Hardware, and HRI Interfaces. The Planning & Controls team needs to have access to the robots' orientation and location so they can plan the route the robot will take. We are responsible for providing the following: robot pose, localization, and object tracking. Robot pose involves knowing the orientation of the robot such as the angle and distance from various objects. Localization involves using LiDAR to capture the mapping of buildings and to know where the robot is with respect to other buildings. Object tracking is for knowing where the user is at all times to ensure that it is following the robot and that the robot itself has not strayed too far.

Besides what we provide from other teams, we also need support from other teams for us to complete our tasks. Given that the Robot Hardware team is responsible for integrating all the hardware together, we request that they support us with sensor mounting and power supply. We have also given over the task of cliff detection to Robot Hardware.

In addition to the physical robot, our object-tracking pipeline requires the user to be registered with the perception system. This means that when the user scans their ID with the robot, they will register themselves by positioning themselves in a bounding box in front of the camera. This will require written instructions on the robot screen or spoken instructions via the LLM. We received support from the HRI Interface and HRI Intelligence teams on this since it relates to human-robot interaction. We decided, for the best cross-team compatibility, to utilize ROS2 services. So we implemented a service for locking onto a user during the user initialization with a simple command line call.

5.2.4. System Architecture



5.3. Planning and Controls

5.3.1. Visionary Scenario

The Planning and Controls team plays a pivotal role in the visionary scenario by translating real-time data integration from sensors, HRI intelligence, Perception Remote operations etc. We utilize this data to autonomously plan optimal routes for the robot, ensuring accessibility and safety in campus environments. Our algorithms adapt to stationary obstacles and hazards, prioritizing smooth movement and user comfort. We reduced the map scale and focused on key areas like Gates-NSH-Wean (CMU's 4th floor map), optimizing trajectory control and maintaining a friendly distance from pedestrians.

5.3.2. Functional Requirements

5.3.2.1. Real-Time Sensor, HRI, and Remote Ops Data

To achieve the scope of the planning and controls sub team, real-time data of the robot's surrounding environment is required. This includes raw and processed sensor data outlined in the perception and robot hardware team's aforementioned descriptions. This data is used to build a world model for the robot's environment that will be used in planning algorithms to generate optimal paths which will then be used to control actuators for a smooth and natural robot experience. On top of sensor data, information from HRI Intelligence is also required to be able to adapt in real-time to user input. It is expected that this user input will be processed to output numerical goal commands (e.g. goal position or maneuvers) that the planning and controls team can then incorporate into the control pipeline. Lastly, signals from the remote operations team are needed for the planning and controls team to be able to determine whether the robot should be operating autonomously or relinquish control for remote supervisors intervention. Once the planning and controls team gives up autonomous control of the robot, the remote operations team should be able to directly interface with the robot hardware themselves.

5.3.2.2. Planning Algorithm Requirements

The planning algorithm selects planned routes completely autonomously in mapped regions for the robot's path. On top of basic planning, the algorithm optimizes these routes to ensure accessibility and efficiency for users as well as be able to adapt in real-time to dynamic environments where it may encounter obstacles, dangers, and hazards. Given this must all happen in real-time, path planning computational latency should be minimized to achieve natural movement for enhanced user experience.

5.3.2.3. Actuator Controls Requirements

The robot also needs to have an optimal trajectory control policy to keep pace with the user while minimizing jerk. This should reduce wear on actuators, increasing the overall life-span of the robot. Unplanned immediate threats and obstacles will be

unavoidable during robot operation that the control interface will need to be able to avoid to ensure user and robot safety. A “friendly campus” robot needs to be friendly, so the robot needs to maintain a safe distance from intruding on pedestrian’s and user’s personal space to instill a level of comfort when the robot is operating around people.

5.3.3. Technology Survey Summary

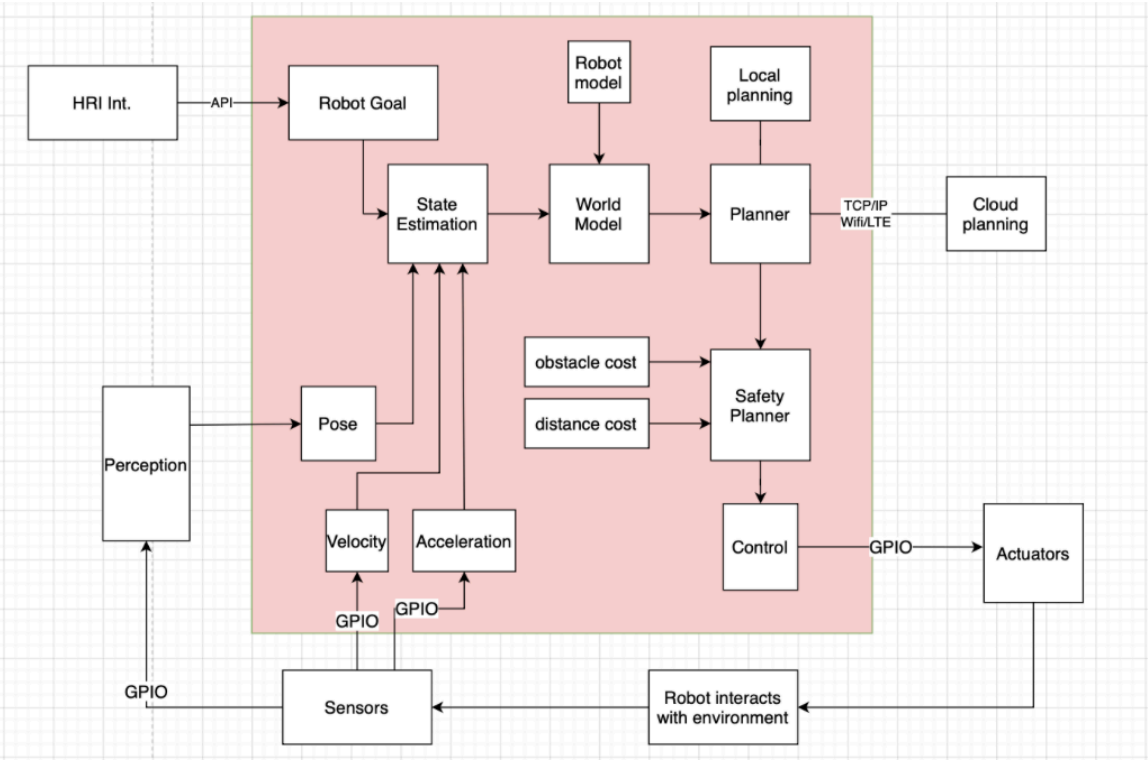
The planning algorithm is broken down into a two step process. Upon receiving a goal destination from the HRI intelligence team, a path is routed through a set of waypoints on a simple 2 dimensional map of the robot’s workspace. The robot then routes a path from waypoint to waypoint using A* and real-time localization data front the perception team. The control side follow the serial command packet protocol outlined by the robot hardware team.

5.3.4. System Architecture

The planning and controls system is designed as a closed-loop system, where the output of the system (the robot's motion) is continuously fed back into the system as input (sensor data). This closed-loop architecture ensures that the robot can dynamically adjust its behavior based on real-time feedback from its environment.

At the top level of the system hierarchy is the planner, responsible for generating optimal paths and trajectories for ScottyBot. The planner interacts with various subsystems, including primarily perception, human-robot interaction (HRI) intelligence, robot hardware, and remote operations. The software architecture starts by subscribing to the perception team’s published ROS2 topics to get robot pose and obstacle locations. Tasks are then imported from the HRI intelligence team to get end destination coordinates for the robot to travel to. This data is then fed into the planner to generate the robot’s optimal path. Once the path is calculated, a control pipeline sends actuator commands to the microcontroller via a serial USB protocol API. The control pipeline asynchronous receives signals from remote operations and HRI intelligence to alter actuator command outputs sent to the microcontroller.

The planner orchestrates high-level decision-making and coordination, while lower-level components, such as control algorithms and actuators, handle the execution of planned trajectories. This hierarchical structure allows for efficient delegation of tasks and facilitates modular development and testing.



5.3.5. Simulation Experiments

Before testing on the actual hardware we conducted simulation experiments. For simulation, the testing and validation of the path planning algorithms were conducted in multiple virtual environments. We focused on integrating various software tools and frameworks to facilitate the development, simulation, and testing of path planning algorithms. Key components include:

5.3.2.1. Gazebo Simulation Environment:

Gazebo, for simulating the Turtlebot's motion in complex environments (eg. a Warehouse). By importing real-world maps and defining dynamic obstacles, we evaluate the performance of path planning algorithms.

5.3.2.2. MATLAB:

MATLAB serves as the primary platform for algorithm development and analysis. MATLAB's rich set of libraries and tools for prototyping and testing various path planning algorithms was used, especially the ROS toolbox.

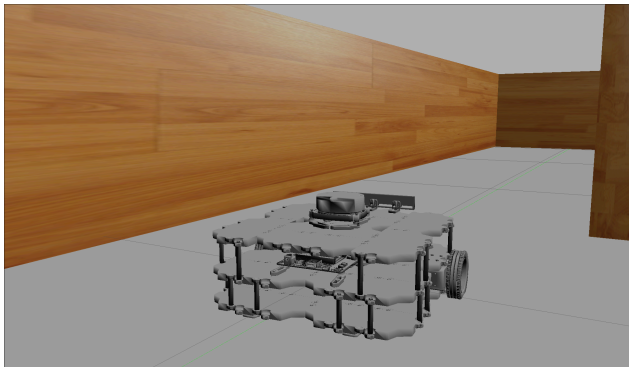
5.3.2.3. PyBullet Physics Engine:

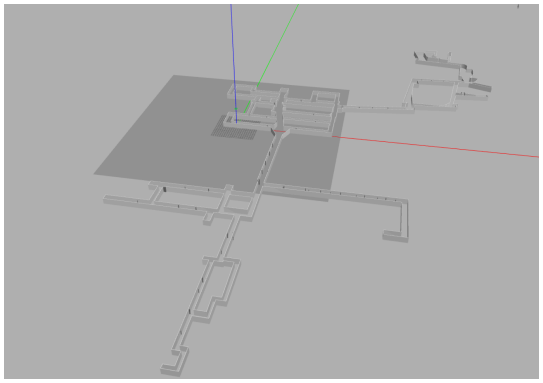
As an alternative physics engine, PyBullet provides a lightweight simulation environment for rapid prototyping and testing of algorithms. We successfully integrated

PyBullet with their path planning algorithms, enabling efficient development in a 3D environment.

5.3.2.4. ROS:

ROS facilitates communication between different components of the system and enables seamless interaction with hardware components. ROS's features were used to test out integration of sensors, actuators, and other peripherals with path planning algorithms. Some examples of simulated environments are given below.

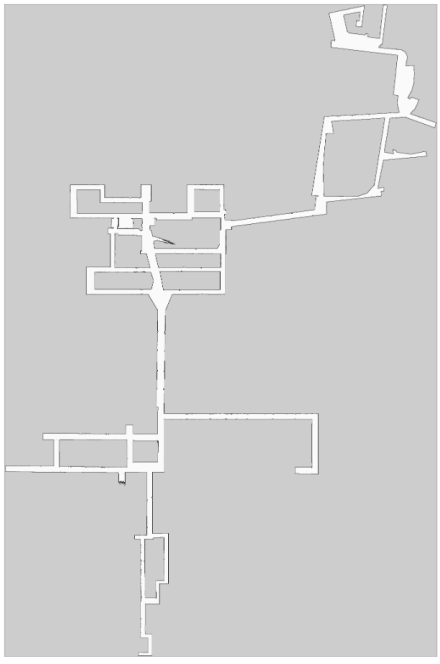


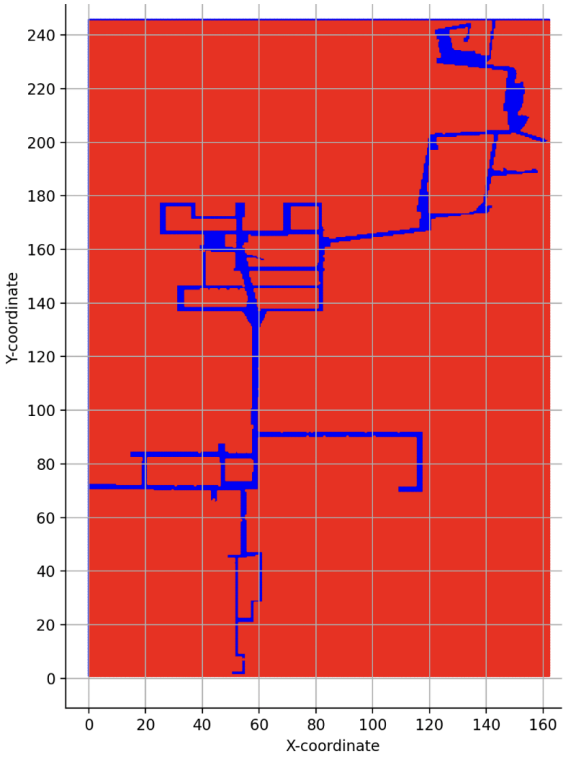


5.3.6. Software Design

5.3.6.1. Map Grid Generator

The Grid Generator converts a pre-measured gray scale map of the buildings into occupancy grids which enables further path planning in this robot's work space.



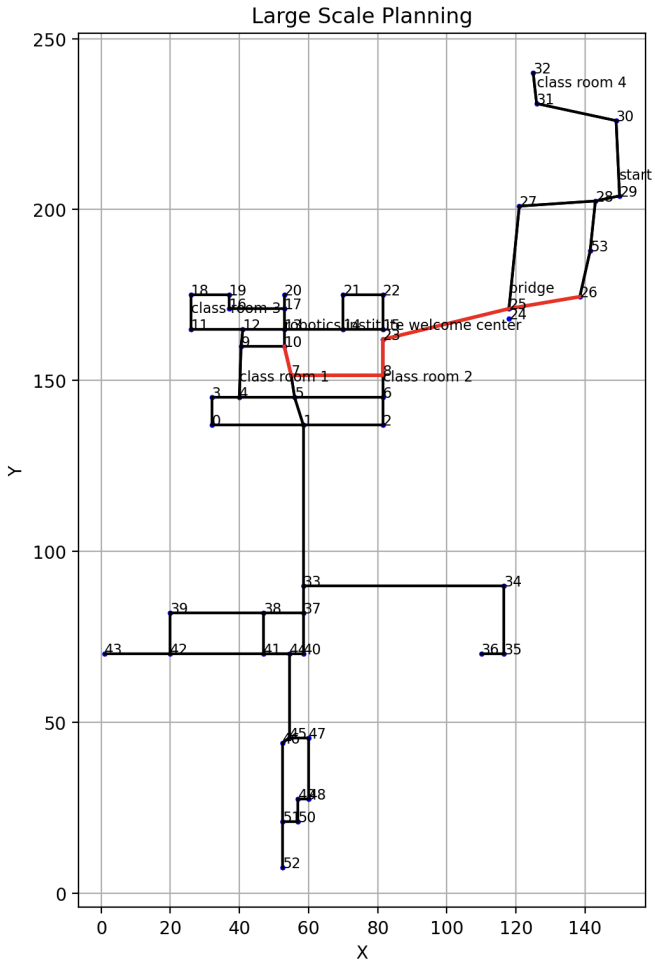


5.3.6.2. Subscriber

The subscriber subscribes to the robot's real-time attitude and the detected obstacles' information from the perception module.

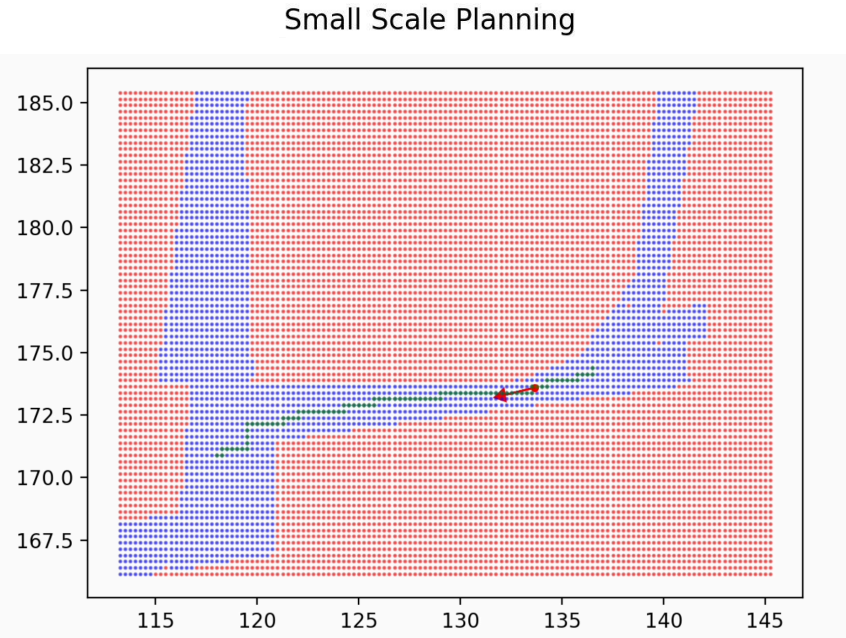
5.3.6.3. Large Scale Map

The Large Scale Map module abstracts a portion of the CMU map (Gates-NSH-Wean) into key nodes such as classrooms, corners of corridors, etc. It records the positions and names of these locations. The module facilitates rough path planning based on user-inputted location names, employing the Dijkstra algorithm. This abstraction prevents real-time planning difficulties caused by the sheer size of the entire map.



5.3.6.4. Small Scale Map

The Small Scale Map module offers detailed planning from one node to another, dynamically generating more detailed maps based on perception's input about surroundings and the pre-generated occupancy map. It plans paths while avoiding obstacles encountered in real-time, employing A* algorithm.



5.3.6.5. Robot Control

The Robot Control module manages the robot's following of planned paths. It receives the robot's current position and the next desired position from the planner, calculates the ideal linear and angular velocities for the robot, and provides feedback to the backend. It also computes motor speeds for the robot's left and right wheels and publishes it to Robot Hardware.

5.3.6.6. Plan and Control Module

The Plan and Control Module encapsulates the functionalities of the above modules and provides interfaces to other teams/groups such as *set_end_point* which allows HRI to change the destination, *suspend* and *resume* which allows switching control of the robot to remote operations in certain situations. It serves as a comprehensive control center for the robot's navigation and interaction with the environment.

5.3.7. Interactions

5.3.7.1. Perception

The perception team provides us with two essential sets of data: the current pose of the robot and detected obstacle points.

Current Robot Pose: We utilize the provided robot pose data to ascertain the current position and orientation of the robot within the environment. This information is crucial for accurately updating the robot's position on the map and refining path planning.

Detected Obstacle Points: By leveraging the detected obstacle points, we enhance our map by marking these obstacles and updating our obstacle avoidance strategy. Incorporating obstacle data enables us to dynamically adjust the planned path, ensuring safe navigation even in complex environments. Due to time constraints, this feature was not completely implemented into our final demo.

5.3.7.2. Robot Hardware

We provide the Robot Hardware team with the left and right wheel speeds required for navigation. This information enables precise control of the robot's movement based on the planned path, utilizing the PID control algorithm. As a next step, we also provide control signals for the ‘in use’ light on the robot.

5.3.7.3. HRI intelligence

The HRI Intelligence team provides us with the destination and we will convert the name of the destination position into coordinates using our dictionary. After the path is planned, we transmit the approximate path to the HRI Intelligence and interface teams. In the next step, we will also provide a signal for changing the state of the robot to the HRI Intelligence group, such as commanding the robot to pause or slow down under specific circumstances, such as encountering pedestrians (not integrated for the final demo).

5.3.7.4. Infrastructure Software

The interface between planning and controls team and infrastructure software for elevator floor selection and automatic door opening involves a coordinated exchange of data and commands. When the Planning and Controls Team makes decisions regarding elevator actions, such as selecting a floor, these decisions are transmitted as JSON messages to the infrastructure software. The infrastructure software, upon receiving these messages, interprets them and triggers the necessary actions, such as sending signals to the elevator system to move to the selected floor and coordinating with the automatic door opening mechanism to ensure seamless entry and exit for users.

5.3.8. Team Deliverables

The Planning and Controls subteam was tasked with making the robot autonomously drive a route for the final demo, for which we completed the following:

- Generate 2D occupancy grid of robot’s work space based on a pre-measured gray scale map in Cabot CMU repository provided by Carnegie Mellon University Cognitive Assistance Laboratory. And hardcoded the key points in it to a graph.

- Take localization data from perception as input with a ROS2 node, and also acquire destination goals from HRI intelligence through MQTT.
- Plan a path through waypoints on the large scale graph using the Dijkstra algorithm.
- Route a fine-grain path between waypoints using A* algorithm.
- Use pure path pursuit to output motor commands via serial connection to robot hardware.
- Provide suspend & resume interface in plan and control module so that remote operators can take over the control of the robot any time they want.
- Implement generating real time plots of robot path planning and trajectory.
- We can deliver a python library that implements these functions, including 6 submodules, a set of map data and a demo main function.

5.4. Human-Robot Interaction Intelligence

5.4.1. Functionality

5.4.1.1. Speech Recognition

HRI Intelligence team provides speech recognition that handles voice inputs and converts it to text and generates voice out after feature extractions. The input voices are in mp3 formats and the output of our pipeline is also in mp3.

5.4.1.2. NPL (Feature Extraction)

One main functionality the HRI Intelligence subteam is providing is to add Natural Language Processing elements to the robot screen and mobile app. We include GPT to process the input messages, and then extract objectives, and label the text with location category.

5.4.1.3. Hey ScottyBot

The “Hey ScottyBot” function is a voice-activated command feature (similar to Siri), which allows users to invoke the robot, without the need to physically interact with the device. It could provide users with a more convenient and accessible way to perform a wide range of tasks using voice commands. By simply saying “Hey ScottyBot”, users can then send requests to the robot, including navigation requests, Q&A requests, etc.

Specifically, it listens for a wake phrase “Hey ScottyBot”. After the wake phrase is recognized, the application will then listen for the command the user wants the robot to proceed.

This functionality is particularly useful when users’ hands are full, or when the device is out of reach, which makes our robot more accessible.

5.4.1.4. MQTT

We deployed a MQTT into the Flask backend to handle team communication such as sending extracted features with objectives and destination to Planning and Control as well as handling card login requests with the RFID ID Card Scanner information.

5.4.2. Technical Details in Components

5.4.2.1. Speech to Text (Whisper)

We use openAI whisper model to transcribe audio from MP3 file to text, the service is based on microsoft azure.

Here’s a detailed breakdown and introduction to the components of the code and their functions:

API Key: The script initializes the OpenAI API by setting an API key, which authenticates the requests to OpenAI's services.

API Base and Type: It specifies a custom endpoint (api_base) and type (api_type), indicating that the code is meant to work with an Azure-hosted instance of the OpenAI API, rather than OpenAI's default cloud service. This setup is part of

OpenAI's offering that allows deploying models on Azure for enhanced data privacy and compliance.

API Version: It also sets an API version (api_version), which defines the version of the OpenAI API being used, ensuring compatibility with the specific features and syntax of that version.

Model Name: The code specifies using the "whisper" model, indicating that it's utilizing OpenAI's Whisper model for audio transcription.

Deployment ID: It mentions a deployment_id, which is unique to the user's deployment of the model on Azure. This ID links the request to the specific deployed instance of the model.

Audio Language: The language of the audio file is set to English (en), which helps the model optimize its transcription accuracy for that language.

Finally, the converted text results are printed for future processing.

5.4.2.2. Feature Extraction

5.4.2.3. GPT

Drawing on a number of papers on prompt engineering for feature extraction, we devised a method for grabbing the objective and destination:

You are a wheeled robotic navigation assistant on Carnegie Mellon University's college campus.

Please extract the (1) objective and (2) destination from the phrase: "Hey ScottyBot, can you tell me how to get to Porter Hall?"

The options for "objective" are: (1) "walk with user", (2) "provide verbal directions", (3) "change pace"

Provide your answer as a JSON in this format:

{"objective": <objective>, "destination": <destination>}

This prompt will be fed to a Microsoft Azure instance of GPT3.5 with the “chat” interface, currently queried through a standard REST API for maximum compatibility. The combination of context “You are a wheeled robot…”, specific instructions, and designated output format yielded satisfying results in feature extraction. This will be sent to planning and controls.

We’ve also utilized GPT to provide natural language feedback to the user based on their vocal input.

Due to the amount of time left, we're proceeding with GPT for our main feature extraction methods.

5.4.2.4. NER

Name Entity Recognition can identify and categorize entities in unstructured text. The build-in NER contains some entities that are useful in our project. By utilizing the NER features, we can examine the extracted text with the entities' category, confidence score, length and offset values. Some category including the Location and Quantity category come in handy when we're extracting an input text of "Hey ScottyBot, I would like to go to Wean Hall Room 7500". After applying NER in our pipeline, it extracts "Wean Hall" and "Room" as Location category and "7500" as Quantity. By anaylsisng the confidence socre, we can get an idea of how good the extraction is. In order to have the extracted entities have a higher confidence score to fit into our application, customized NER is nessary. Due to the amount of time left, we're proceeding with GPT for our main feature extraction methods.

```
Named Entities:

        Text:    Wean Hall      Category:        Location       SubCategory:     None
        Confidence Score:        1.0    Length:          9      Offset:          16

        Text:    Room   Category:        Location       SubCategory:     Structural
        Confidence Score:        0.39   Length:          4      Offset:          26

        Text:    75000  Category:        Quantity       SubCategory:     Number
        Confidence Score:        0.8    Length:          5      Offset:          31
```

5.4.2.5. Text to Speech (Speech Service)

Text to Speech handles outputting the text after feature extraction into a voice output. We picked "Audio16Khz32KBitRateMonoMp3" as our output format to have a easier integration with out teams. Instead of wav, mp3 is more compatible and have a smaller size. As the last part of our pipeline, we create a speech synthesizer and synthesizes it to mp3 file. This will be passed to HRI Interface as a backend support for their screen and mobile website.

5.4.2.6. Hey ScottyBot

"Hey ScottyBot" utilizes a combination of sophisticated voice recognition technology and artificial intelligence to understand and process user commands. We use Microsoft Azure Cognitive Services Speech SDK service to create a web-based application capable of recognizing speech command.

To break down the code and its functionalities:

Subscription Key and Service Region:It begins by defining a subscriptionKey and serviceRegion, which are required to authenticate and connect to the Azure Speech Service. These are obtained from the Azure portal and Azure Speech Service resource.

Speech Configuration: SpeechSDK.SpeechConfig.fromSubscription is used to create a speech configuration object with the provided subscription key and service region. The language for speech recognition is set to English (United States) with speechConfig.speechRecognitionLanguage = "en-US".

Start-btn: An event listener is attached to a button with the ID start-btn. When clicked, it initiates speech recognition.

Continuous Speech Recognition: It uses SpeechSDK.SpeechRecognizer to continuously listen for speech. This is crucial for wake word detection, as it allows the application to process spoken words in real-time without manual intervention to start or stop the listening process.

Wake word detection:The recognizing event handler checks the transcribed text for any variations of the wake word (e.g., "Hey Scotty", "Hey Scotti", and other phonetically similar phrases). This step is vital for activating the application through voice without physical interaction. Upon detecting the wake word, the recognizer stops, and the application indicates that it's ready to listen for a command. A slight delay ensures the recognizer has fully stopped before proceeding.

The startRecordingCommand: For15Seconds function sets up a new recognizer to listen specifically for a command for 15 seconds. It uses the default microphone input for audio configuration.After 15 seconds, it stops listening and indicates that it's processing the command.

Command Recognition: During this period, the application continuously transcribes speech and updates the displayed text with the recognized command.

Continuous Recognition: The use of continuous recognition allows the application to listen and respond in real-time, enhancing user interaction.

Dynamic Wake Word Recognition: By checking for various phonetic variations of the wake word, the application can better handle different accents and speech patterns.

Delay Handling: The use of a delay before starting to listen for a command post-wake word detection helps avoid capturing the tail end of the wake word as part of the command.

5.4.2.7. MQTT and Websockets

We collaborated closely with communications to integrate MQTT into our subsystem. Rather than assume our server would always be running on the same computer as Planning and Controls' computations, and relying on local mechanisms of communication, we connected our server to the rest of the system with MQTT, using Communication's iot_interface.py module. As further explored in Communication's section, MQTT (Message Queuing Telemetry Transport) is an application layer network protocol that is commonly used as a backend for cloud platforms. Essentially, MQTT

acts as a broker, routing messages between all sorts of entities. All a subsystem has to do to receive relevant information is to subscribe to a specific topic. Other subsystems can publish to that topic to broadcast said information. We're listening on the RFID scanner topic to receive a signal on the Flask backend, that is then ported over Websockets (bidirectional HTTP communication protocol) to the client. Websockets were necessary, because this specific interaction could not be instigated by the client. We're also publishing the destination to the topic where Planning listens to create their path plan.

5.4.3. Interactions

5.4.3.1. Interactions with Interfaces

As backend support for the Interface team, we provide the text-to-speech, feature extraction, and speech-to-text pipeline as backend services and send them to the frontend. We've integrated the Hey ScottyBot feature as a voice support into the HRI Interfaces repo's frontend's navigation page. Additionally, we've provided assistance in three more features:

Login Feature: We use MQTT in Flask backend to implement login authorization according to the RFID Card Scanner information.

Route: When we receive the route images from Planning and Control, we include this with interfaces' implementation and provide it in the navigation page.

Robot Speech: This step is also done by MQTT and can do a complex sequence of jobs.

5.4.3.2. Interactions with Communication

We host the backend server to the website via an AWS EC2 instance, which then needs to be registered as a "Thing" in their IoT network. We communicate and collaborate with them to set up the appropriate server for our Flask backend.

5.4.3.3. Interactions with Perception

We collaborate with perception to design a user-facing map where a user can track their route status and location. We communicate to devise a shared coordinate system such that their internal point cloud representation of the mapped space is synchronized with the user map on the robot screen and phone website.

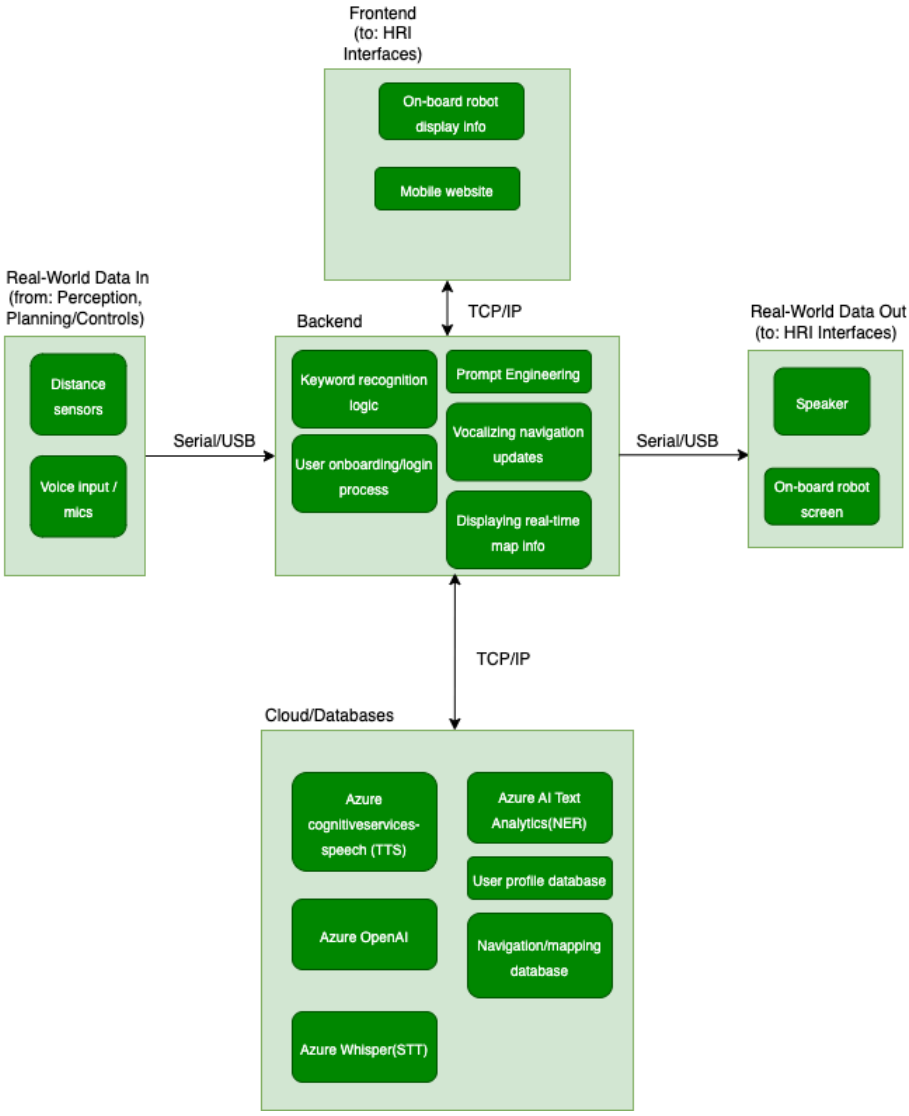
5.4.3.4. Interactions with Data Systems

We utilize Data Systems' AWS lambda functions to manage and interface with databases of user information.

5.4.3.5. Interactions with Planning and Control

We send objectives and destinations with GPT Query and constraints to planning and controls, so that they can plan using that information. We will send this information in JSON format and we get a image of the route back.

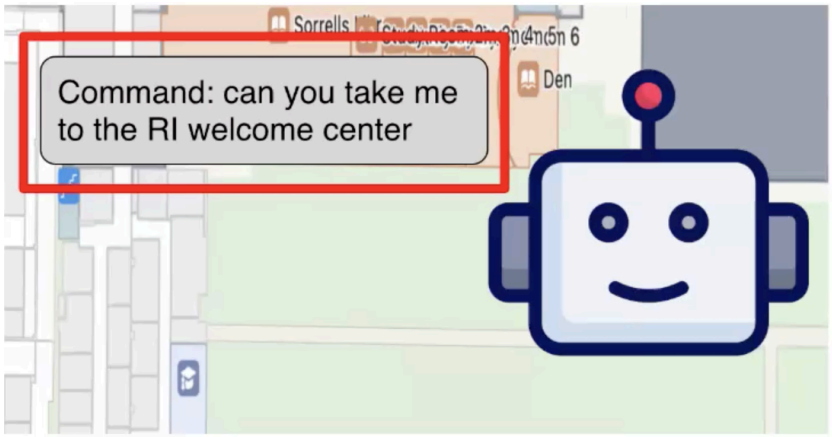
5.4.4. Software Architecture



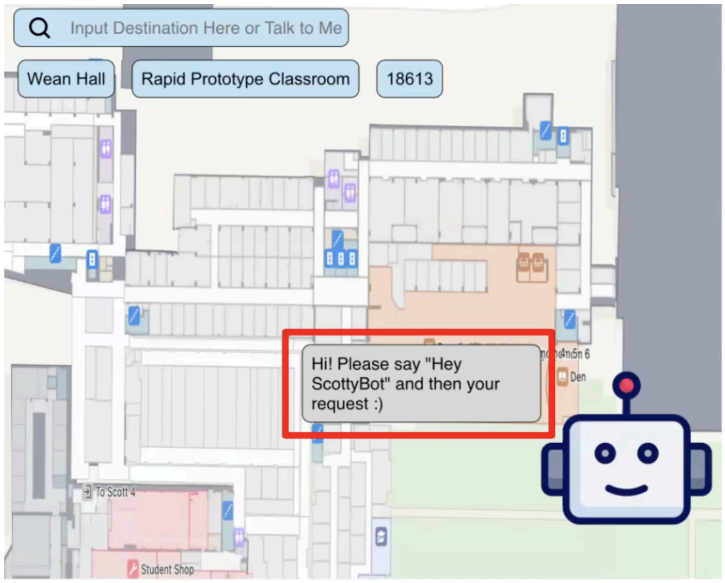
5.4.5. Team Deliverable

In Phase 3, we improved our current GPT feature extractions and integrated our voice in->feature extraction->voice out pipeline into HRI Interfaces' frontend development. In particular, we've added backend support along with MQTT for communication between a couple of sub teams including HRI Interfaces, HRI Intelligence, and Planning and Control. MQTT is integrated with the existing Flask backend along with websocket emits.

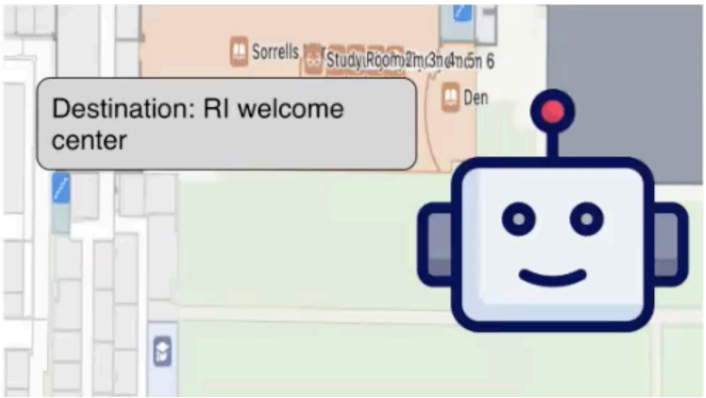
Our pipeline is deployed in the navigation page when the user speaks in the microphone and we use Azure Speech Service to recognize the user commands.



Then, we transcribe the user commands such as a request for direction to RI Welcome Center and print it in the navigation page.



Next, we use Axios requests to backend and send the extracted feature including objective and destination with the GPT Query. This step is done by MQTT which enables the json file to be sent to Planning and Control. Eventually, we prints out the destination in the user interface for a better user experience.



5.5. Human-Robot Interaction Interfaces

5.5.1. Objective

In the third phase of the project, as the Interfaces team, our primary goal is to refine and optimize the user interface of ScottyBot to ensure an intuitive, responsive, and user-friendly operating environment. More specifically, this includes iterating on UI design to make it more responsive to user needs, streamlining user operations, and ensuring that all features are easily accessible. And ensure the accuracy and timeliness of the front and back end information transmission, especially for route planning information and voice processing information transmission.

Moreover, we play a crucial role in cross-team collaboration, primarily responsible for coordinating cooperation among teams to ensure the smooth integration of hardware, software, and database components. This involves not only technical integration but also addressing potential communication issues between each team, which is vital for the overall project progress and quality assurance.

Enhancing ScottyBot's multimodal interaction capabilities is also a key objective in this stage. By developing an integrated text and voice dual-modality input system, the robot is enabled to receive and understand user commands based on both text and voice. Additionally, through a multimodal feedback system, ScottyBot not only displays navigation information on the screen but also provides auditory guidance to users, thereby enhancing the user interaction experience.

This stage of work also includes the preparation and recording of project demonstration videos to showcase ScottyBot's operational interface and its features clearly to stakeholders. Through these system optimizations and demonstration preparations, our team aims to enhance the practicality and market competitiveness of ScottyBot, enabling it to effectively serve the campus community and improve the overall accessibility and friendliness of the campus.

5.5.2. Functionality

Based on the demand analysis and iterative development of the product, we have implemented the following features in the user interface and interaction aspects of the ScottyBot:

5.5.2.1. Registration and Login

- A multimodal login system was implemented, allowing users to choose between traditional email-password login and quick access via NFC by swiping their

campus ID cards. This enhancement improves accessibility and supports rapid user authentication in high-traffic areas.

- The login interface is designed to be clean and intuitive, ensuring that users can easily manage their accounts.

5.5.2.2. Destination Search

- Voice recognition technology has been integrated, enabling users to activate voice input with the wake word "Hey, ScottyBot" (similar to interactions with Siri or Google Assistant).
- A text input option is also provided, allowing users to type in their destination using a keyboard.

5.5.2.3. Route Planning

- Upon receiving a destination input, the system confirms the user's request with immediate feedback, such as responding with: “Of course, I will take you to XXX,” enhancing the interaction's human-like and friendly nature.

5.5.2.4. Navigation

- During navigation, the system can adjust the robot's route based on real-time data (such as congestion and sudden incidents) and dynamically display the robot's current location and direction of travel on the screen.

5.5.2.5. Robot Control

- Users can directly control the robot's movement states through the interface, including start, stop, and speed adjustments. Speed control is available in three settings: low, medium, and high, to accommodate different navigation needs and environmental conditions.

5.5.2.6. Robot Status Indication

- When the robot is in use (the user is logged into the robot's interface), the headlight will light up to indicate the robot's active status, and it will turn off when the user exits the system.

5.5.2.7. Feedback Collection

- A dedicated feedback interface has been designed, where users can provide comments on the robot's performance, navigation accuracy, and user interface friendliness after use.
- The feedback data will be used in future iterations to improve the robot's functionality and user experience.

With these detailed functional designs, we have not only met the basic needs of users but also enhanced ScottyBot's user-friendliness and functional efficiency through technological innovation and meticulous design, making it a highly interactive and practical indoor campus navigation robot.

5.5.3. System Architecture

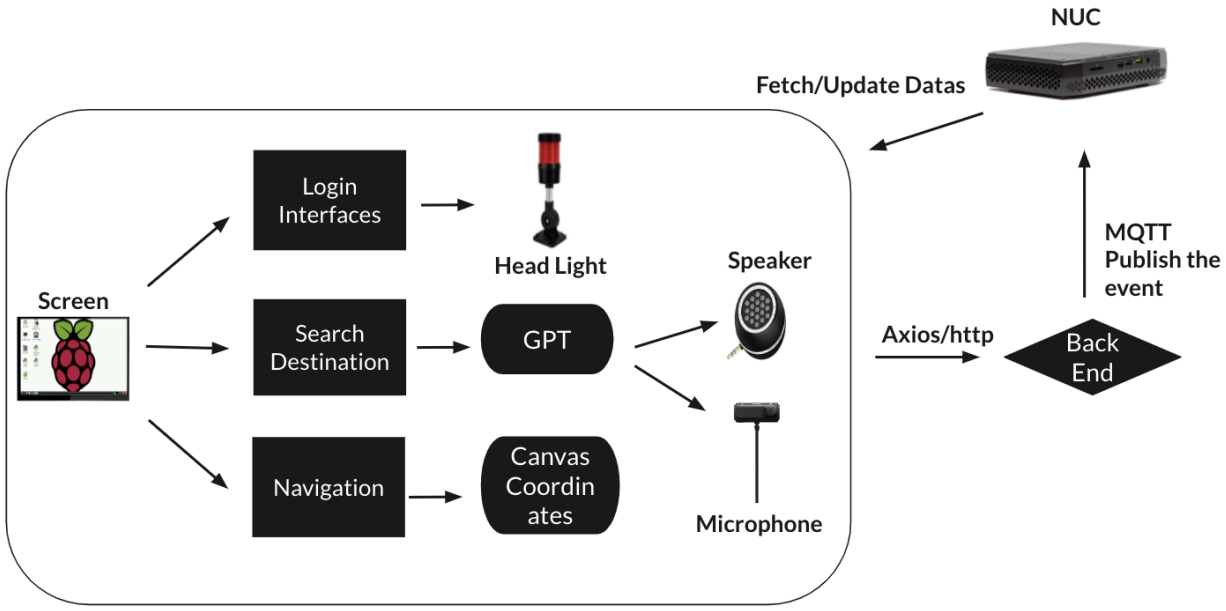
The system architecture of ScottyBot is meticulously planned to ensure efficient navigation within indoor environments and interactive communication with users. At the heart of the system is a NUC, a high-performance microcomputer that coordinates the functioning of various hardware components and software modules, processes data from multiple sensors, and executes complex commands and operations.

In terms of user interaction, ScottyBot is equipped with a touchscreen that displays an intuitive user interface, incorporating features such as user login, destination search, and route navigation. Additionally, the system integrates a microphone and speaker to support voice interactions, allowing users to activate and converse with the robot using the voice command "Hey, ScottyBot" much like interacting with a smart assistant, which enhances convenience and the overall user experience.

The headlight on the robot serves as a status indicator or a navigation aid, providing visual feedback to help users recognize the robot's current state. On the software side, GPT technology is employed to process voice inputs, combined with natural language processing techniques, enabling the robot to more accurately understand user intentions and commands. The Canvas Coordinates system is responsible for drawing navigation paths on the user interface, presenting complex navigation information graphically, which is easy to understand and follow.

The backend system uses Axios and other HTTP communication methods to interact with the frontend interface, while the MQTT protocol is used for publishing and subscribing to events, ensuring responsiveness to real-time events. The architecture focuses on real-time data acquisition and updates to achieve timely and accurate information dissemination.

Overall, the system architecture of ScottyBot effectively leverages advanced computing and communication technologies to integrate complex hardware and software into a coordinated, responsive entity. This design not only allows ScottyBot to provide precise navigation services in a busy campus environment but also offers a user-friendly and interactive experience, significantly enhancing campus accessibility and overall amiability.



5.5.4. Implementation

5.5.4.1. Interface Function

a. User Signup & Login

User login and authentication are prerequisites for all ScottyBot functions.In the process of implementing these functionalities, we first established a new project on the Firebase platform and generated the requisite configuration details. Following this, we enabled the email and password authentication method within the "Authentication" section of the Firebase project. Once this setup was complete, we proceeded to configure our React project to integrate with Firebase.

Within our React application, we utilized the Firebase Auth API. This API facilitated the creation, authentication, and management of user accounts, enabling functionalities such as user registration, login, and logout. By implementing the registration and login forms, we leveraged the methods supported by Firebase Auth API to handle user authentication processes effectively.

This series of actions ensured the robust implementation of the user login feature, providing users with a secure mechanism to access ScottyBot via email and password authentication. This integration not only secures user data but also enhances the overall user experience by streamlining the authentication process. Additionally, using Firebase for backend services provides scalability and reliability, making it ideal

for developing dynamic web applications that require secure and efficient user management.

b. Voice Interaction

Voice interaction is a crucial feature of ScottyBot. To implement voice recording and conversion functionality in the web application, we primarily utilized several key Web Media APIs and React Hooks.

Initially, the ‘navigator.mediaDevices.getUserMedia’ method allows us access to the audio input capabilities of the user's device. This method is part of the Media Capture and Streams API (commonly referred to as MediaStream), which prompts the user for permission to use a media input, producing a MediaStream with tracks containing the requested types of media. Once permission is granted, we could obtain a media stream that includes audio tracks.

Subsequently, we employed the MediaRecorder API to record the acquired media stream. In our code, the MediaRecorder object is initialized with the stream obtained from ‘getUserMedia’. This object offers methods to start and stop the recording and to handle the recorded data.

Regarding event handling, the ‘ondataavailable’ event is triggered when the MediaRecorder outputs media data (in chunks in this case). We used this event to collect audio chunks into an array. When the recording is stopped (either manually or automatically), the ‘onstop’ event is triggered. At this point, the recorded audio chunks are processed into a single audio file, and cleanup tasks such as releasing the media stream are performed.

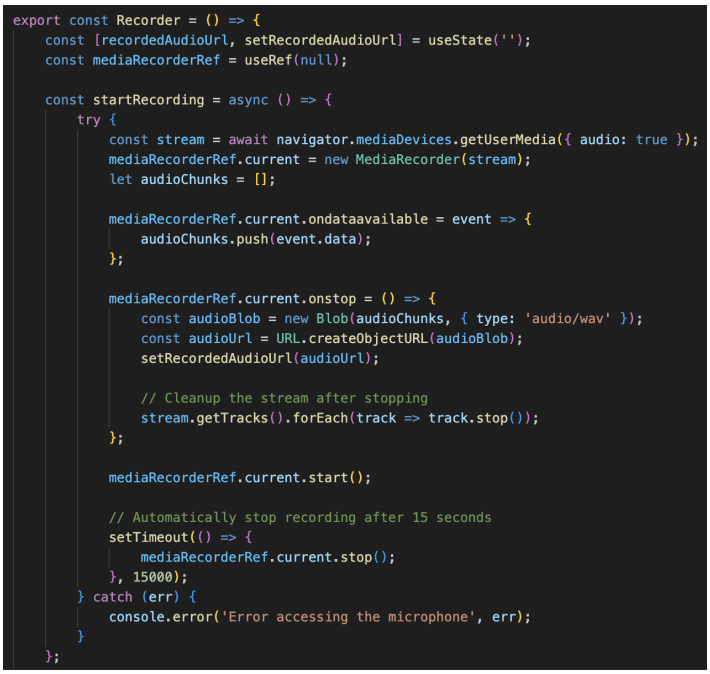
Moreover, we used the Blob constructor to create a new Blob object containing the recorded audio data, set to the MIME type ‘audio/wav’. This object represents immutable raw data. We then utilize the ‘URL.createObjectURL’ method to generate a URL that represents the Blob object. This URL can be used, for example, in an HTML ‘<audio>’ element's src attribute to play the recorded audio.

Within the React framework, we employed several key hooks to manage this process:

- ‘useState’: used to store and set the URL of the recorded audio file.
- ‘useRef’: used to persist the MediaRecorder instance without causing re-renders.

- ‘useEffect’: used to perform side effects in functional components, initiating recording automatically when the component mounts, and stopping the recording and releasing resources when the component unmounts.

By integrating these technologies, we have implemented a comprehensive voice recording and playback functionality within the web environment. This capability is not only applicable to simple recording tasks but can also be extended to more complex scenarios, such as real-time voice recognition and conversion. Additionally, the implementation of these technologies provides robust interactive capabilities for web applications, significantly enhancing service delivery to users who require audio input and processing.



```
export const Recorder = () => {
    const [recordedAudioUrl, setRecordedAudioUrl] = useState('');
    const mediaRecorderRef = useRef(null);

    const startRecording = async () => {
        try {
            const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
            mediaRecorderRef.current = new MediaRecorder(stream);
            let audioChunks = [];

            mediaRecorderRef.current.ondataavailable = event => {
                audioChunks.push(event.data);
            };

            mediaRecorderRef.current.onstop = () => {
                const audioBlob = new Blob(audioChunks, { type: 'audio/wav' });
                const audioUrl = URL.createObjectURL(audioBlob);
                setRecordedAudioUrl(audioUrl);

                // Cleanup the stream after stopping
                stream.getTracks().forEach(track => track.stop());
            };

            mediaRecorderRef.current.start();

            // Automatically stop recording after 15 seconds
            setTimeout(() => {
                mediaRecorderRef.current.stop();
            }, 15000);
        } catch (err) {
            console.error('Error accessing the microphone', err);
        }
    };
```

5.5.4.2. Hardware Installation

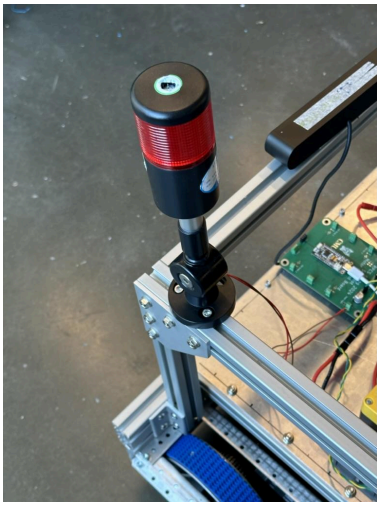
a. Assembly of Head Light & Screen

Firstly, regarding the hardware components, our initial market research for the project considered the ease of use of the product. In selecting components, we opted for plug-and-play devices (microphones and speakers) which connect via a 3.5mm audio splitter to the NUC's audio jack. In determining the installation locations, we considered the following aspects: functionality, aesthetics, and wiring convenience.

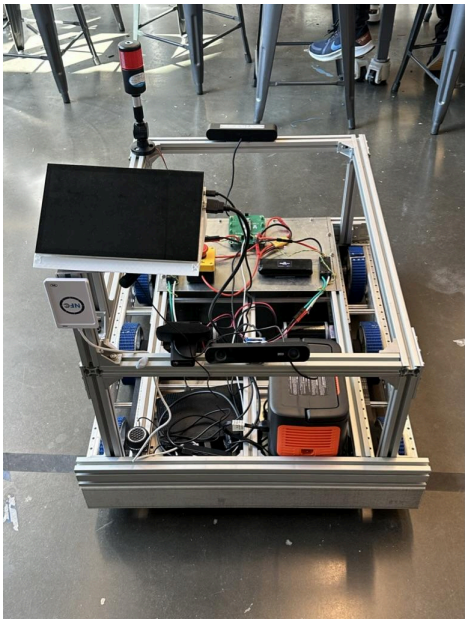
We conducted volume tests on the speaker to ensure its clarity. As a result, we decided to conceal it at the base of the robot for enhanced aesthetics and easier wiring.

Considering the background noise of the operational environment, we aimed for the microphone to be closer to the user; thus, it was installed below the screen to more clearly capture user commands.

Regarding the installation of the Head Light, it required two GPIO ports from the motor drive board for control. We placed it in a corner on the top of the robot where it is sufficiently visible and does not obstruct the camera and radar operations.



The most challenging aspect was the design of the screen, which connects to the NUC via HDMI and USB for power and touchscreen functionality. Given the robot's height, we aimed to make the screen more comfortably visible to users, leading to a raised and side-mounted display. This arrangement does not obstruct the camera and enhances the user experience.



b. Test Microphone and Speaker on NUC

We test the microphone and speaker components after they have been installed on the NUC. We concentrate on the integration with an intelligent recognition system that detects ambiguous start phrases such as "Hey, Sorry Bot" and "Hey, Scooty Boot." Our work with the intelligence team makes use of Azure and OpenAI technologies to improve identification skills, ensuring that the system correctly processes commands with different pronunciations.

5.5.4.3. Communications

In our application, the communication between the frontend and backend is structured around two key processes: requests sent from the frontend to the backend and responses sent from the backend to the frontend.

a. Axios

For sending requests to the backend, we use the Axios library for HTTP transmission, designing RESTful endpoints to handle operations such as login, register, and navigation. This approach leverages the stateless nature of HTTP to ensure that each message transmission is independent and self-contained.

b. Socket.IO

For sending updates from the backend to the frontend, we employ the Socket.IO library. We establish a stable socket connection using text-based sockets, enabling real-time data exchange. This mechanism ensures that any time a user logs in via a card reader, the frontend immediately receives a notification, enabling seamless, real-time interaction.

5.5.4.4. Integration

a. MQTT

In our application, we integrate the Flask backend with other components using MQTT for robust communication and control capabilities. By subscribing to and publishing messages via MQTT, we can precisely manage various functionalities such as card reading and robot motion control. This setup not only facilitates the real-time operation of reading access cards but also governs the speed and movement patterns of the robot, ensuring efficient and responsive interactions within the system.

This method leverages MQTT's lightweight messaging protocol to ensure fast and reliable data exchange, crucial for the dynamic control of robotic operations and immediate response requirements. The integration of MQTT enhances our system's capability to handle complex sequences of actions seamlessly, from user authentication via card readers to precise robot navigation and speed adjustments based on real-time environmental data.

b. Data Synchronization for the Navigation Function

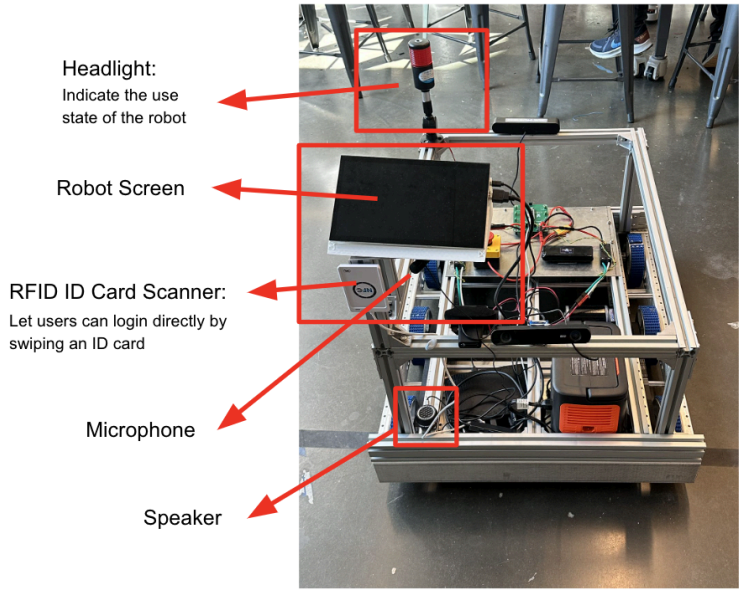
Initially, we developed a backup strategy utilizing animation to simulate the robot's navigation, matching a pre-planned route for the upcoming demonstration with the current GHC map. We then established a backend function tasked with retrieving data-specifically coordinates and directional degrees from the Planning Control's storage system. This function continuously accesses the stored data, reading from the

file at half-second intervals. We employed a JavaScript hook function to facilitate these regular reads and used a settling ratio to accurately render this positional data on the graphical interface.

5.5.5. Team Deliverables

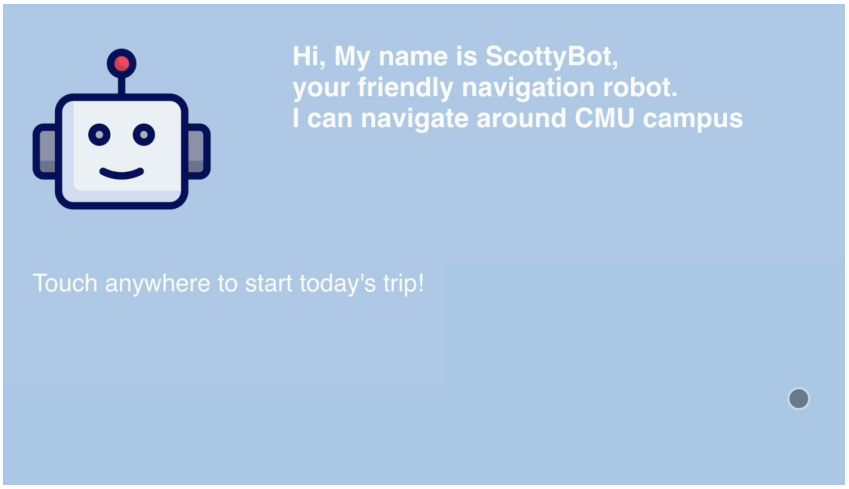
Throughout the project, as part of the Interfaces team, we primarily completed the procurement of robot hardware facilities and the rational layout design, ensuring that each piece of hardware functioned well. Additionally, we designed and implemented the functionalities of the robot's screen interface on the frontend, and collaborated with other teams to integrate, test, and troubleshoot various features, ensuring that each function operated smoothly. Each part of our work will be shown below.

5.5.5.1. ScottyBot Overall Interface Design

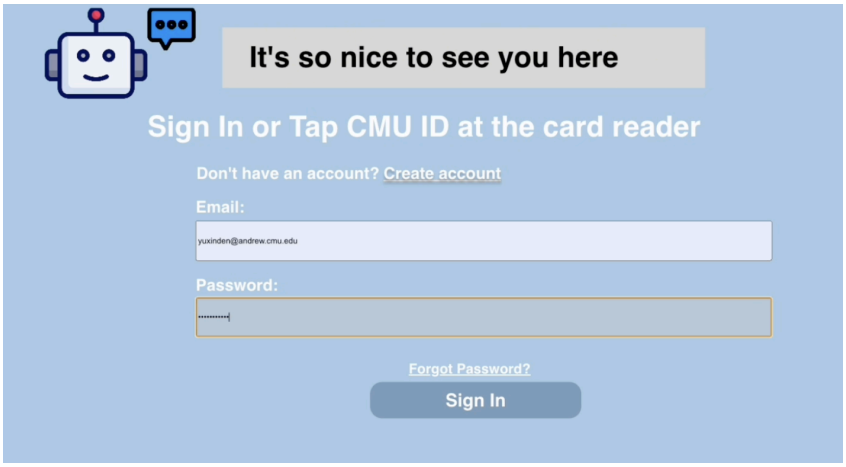


5.5.5.2. ScottyBot Screen Design

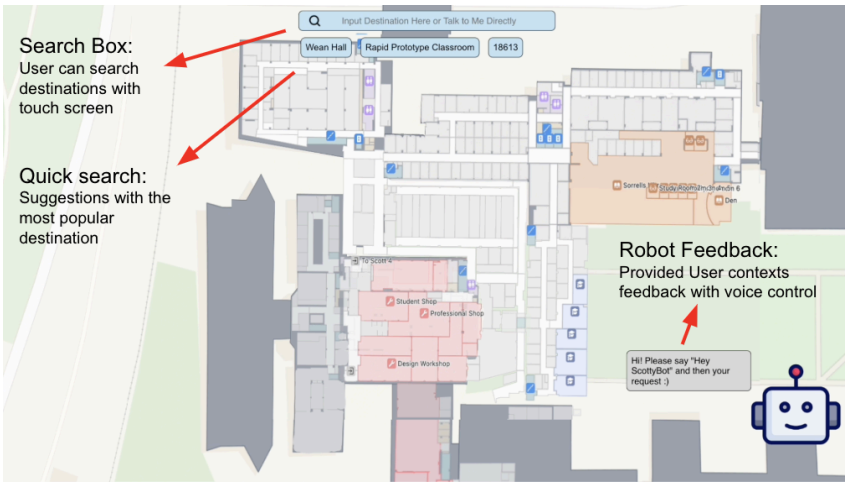
a. Welcome & Introduction Interface

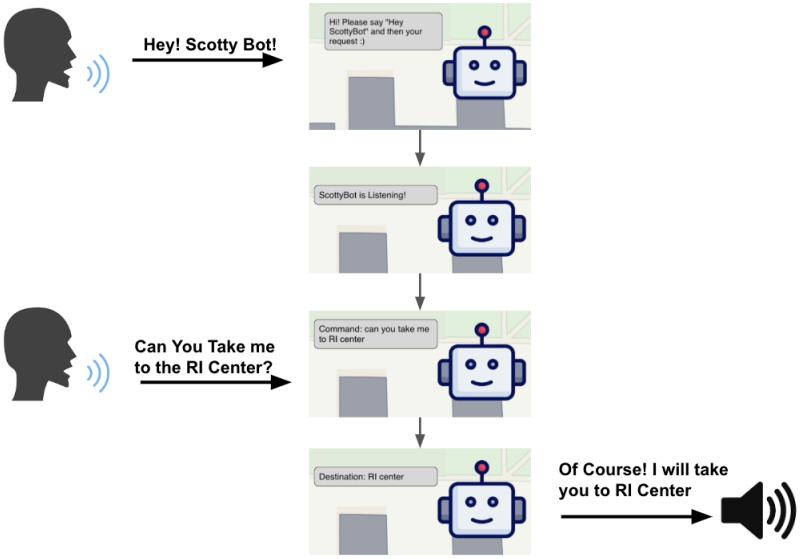


b. Registration & Login Interface

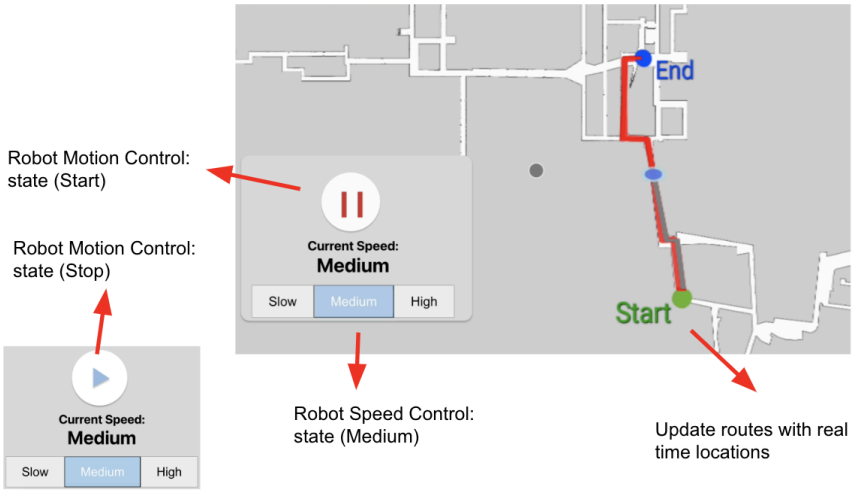


c. Destination Search Interface





d. Route Planning Interface



5.5.6. Next Steps

As the Interfaces team, our future work will concentrate on several key areas to enhance the robot's user interaction capabilities and overall performance:

5.5.6.1. Mobile Interface Development

By persistently developing the mobile interface, we aim to facilitate users to interact with the robot seamlessly through their smartphones, adding convenience to the utilization of the robot.

5.5.6.2. User Interface Optimization

We will continue to refine the design of the user interface to align more closely with user intuition and habits. This includes adjusting visual elements such as colors, fonts, layouts, and icons, as well as streamlining the interaction process by reducing the number of clicks and simplifying the steps involved, thereby improving user operational convenience and efficiency.

5.5.6.3. Customization Features

Development will focus on allowing users to personalize their interface and functions according to individual preferences. Users will be able to choose interface themes, set shortcut commands, and more, thereby increasing the product's appeal and user retention.

5.5.6.4. Hardware-Interface Integration

In collaboration with the Hardware team, we will ensure that the user interface fully leverages the robot’s hardware capabilities, such as optimizing screen display effects, adjusting speaker volume and sound quality, and enhancing the microphone's range and precision of sound capture.

5.5.6.5. Enhanced Assistive Features and Accessibility

Acknowledging the special needs of various users, including the elderly, visually impaired, or hearing impaired, we will develop an array of assistive features to ensure ScottyBot is accessible to everyone.

5.5.6.6. Social Media Integration

We consider integrating social media functionalities into ScottyBot's user interface, such as the real-time sharing of location, routes, and experience feedback, to increase user interactivity and the robot's social connectivity.

Through these efforts, we anticipate ScottyBot to not only become an essential navigational assistant within campus confines but also to extend its applications into a wider indoor navigation market, becoming a pivotal tool for fostering community inclusivity and convenience.

5.6. Communications

5.6.1. Functional Requirements

5.6.1.1. Inter-Subteam Wireless Communication

In order to facilitate wireless communication between multiple subteams and relay information to and from ScottyBot and its surroundings, Communications must provide a network protocol over which devices can send and receive information. The wireless network and protocol should not impede ScottyBot's mobility and must be scalable.

5.6.1.2. Wi-Fi Capability

Relevant devices on and off the robot must be connected to Wi-Fi in order to communicate over our network. The NUC 11 computer and the ESP32 microcontrollers have Wi-FI capabilities.

5.6.1.3. Student ID Card Reader

Only CMU students are allowed to use ScottyBot. As a result, scottybot must be able to read a student ID card and verify that the user is a student at CMU by cross referencing the database and the ID value.

5.6.1.4. Real Time Video Streaming

ScottyBot must be able to communicate other types of information, such as a real-time video, to the Remote Operations subteam while in commission. Reliable real-time video streaming guarantees that time-critical information can be sent from the robot to Remote Operations hub for various scenarios, such as manual control takeover.

5.6.1.5. Remote Operations (Ops) Robot Controls

In case of emergency, there must be a Remote Ops to Robot (NUC) wireless connection for controlling the robot with a simple and intuitive interface.

5.6.2. System Architecture

Communication's system architecture is shown below.

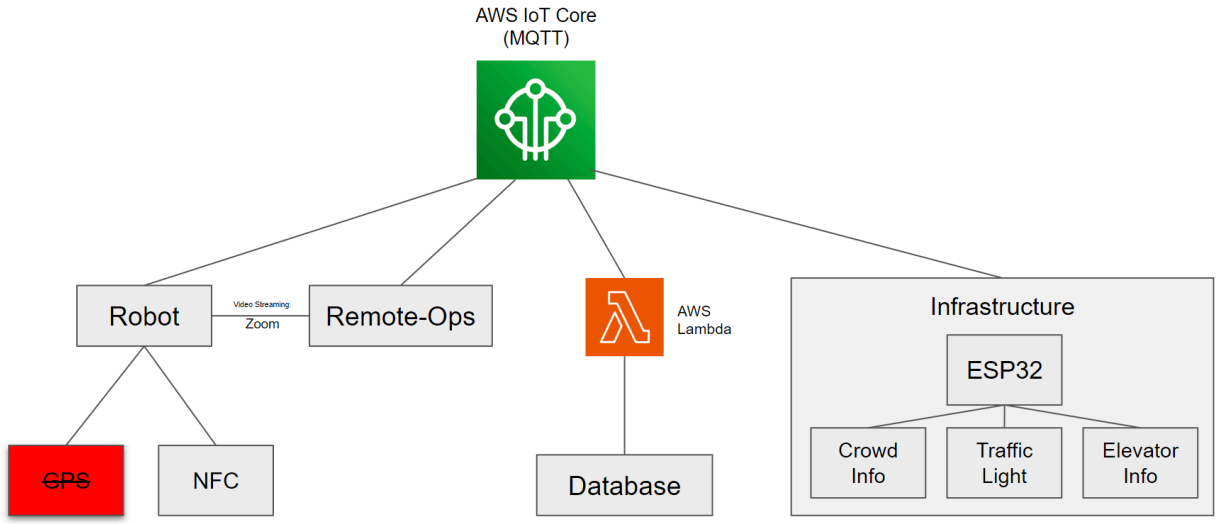


Figure 5.6.3.1: Communications System Architecture

5.6.3. System Design

5.6.3.1. Inter-Subteam Wireless Communication

The wireless system design consists of a network of connected devices over AWS IoT Core. IoT Core allows us to connect multiple devices to the AWS cloud. These devices can then communicate with each other by publishing and/or subscribing to MQTT message topics.

MQTT (Message Queuing Telemetry Transport) is an application layer network protocol that is commonly used as a backend for cloud platforms. Using an MQTT broker - an intermediate entity to which devices (clients) connect - we can establish "topics" that devices publish to (write/send messages to) and/or subscribe to (read/request messages/data from). The broker eliminates the need for direct peer-to-peer connections between devices; it receives a message that a device publishes to a topic and then forwards that message to the devices subscribed to that topic.

From a IoT device perspective, our AWS IoT Core and MQTT configuration allows it to subscribe to specific topics, receive messages published to those topics, and process the data according to their specific use case with callback functions. Our team was responsible for integrating these callback functions with subteams' code.

Data is fundamentally sent wirelessly over Wi-Fi using Websockets (for Remote Ops browser running javascript) and TCP. The lightweight MQTT protocol is used as a sophisticated top layer to efficiently manage the data flow between devices in our IoT system. It directly sends only the necessary messages between relevant devices using topics and the notion of pub/sub (publish/subscribe).

By wrapping AWSIoTPythonSDK functions, we created a Python interface called iot_interface.py that subteams would import to:

1. Connect their device(s) to IoT Core
2. Asynchronously subscribe to MQTT topics
3. Publish to MQTT topics
4. Remove subscribed-to topics
5. Correctly process the received data messages with callback functions.

This python interface was used as a reference for Remote Ops' JavaScript application. Devices connected to AWS IoT Core included the NUC 11 on the robot, Remote Ops's MacBook Pro, and several ESP32 microcontrollers.

5.6.3.2. Wi-Fi Capability

The NUC computer and ESP32 microcontrollers have built-in Wi-Fi capabilities, reducing the scope of our subteam in helping deploy with Wi-Fi. The current scope of ScottyBot is on campus, so we chose to use campus Wi-Fi (CMU-Secure and CMU-Device) for the NUC and a hotspot for the Infrastructure teams' ESP32's. In the future, wireless connectivity can be extended using a small, on-robot Wi-Fi hotspot in case of Wi-Fi dead zones or disconnect. This would expand ScottyBot's range.

5.6.3.3. Student ID Card Reader

The student ID Card Reader scans CMU Student IDs and relays the relevant information to Data Systems, Remote Ops, and on-robot systems. We deployed a Near-Field Communication (NFC) Card Reader Device that scans IDs and sends the ID value to the main computer where it will later be authenticated. This ensures that only CMU students can access the robot. Their user profiles are stored in the Data System subteam's MongoDB. We have successfully connected the NFC reader to AWS IoT Core and published messages with the scanned ID to verify that the ID value is being published and received over the network. This authentication method shall allow the user to bypass the login page on ScottyBot's screen.

5.6.3.4. Real Time Video Streaming

To enable the Perception subteam to live stream video to the Remote Operations subteam, we will use WebRTC, an API for real-time video streaming with low latency over Wi-Fi. Browser-to-browser communication is supported, and therefore live streaming through a set of standardized protocols. It requires 4GB of RAM and there exists a JavaScript API that can easily be used for browser-to-browser streaming.

We take care of interfacing the video streams of the robot with Remote Ops. We achieve this by using WebRTC and PeerJs. WebRTC is a set of protocols and APIs that enables real-time communication directly between web browsers without the need for additional plugins or software installations. It allows for peer-to-peer communication for

voice calling, video chat, and file sharing. WebRTC supports real-time audio and video communication by capturing media streams from the user's microphone and camera using the “getUserMedia” API. These streams are then encoded and transmitted securely using encryption protocols.

For the final demonstration, we ended up using Zoom with the CMU license to stream video from ScottyBot’s front camera to the Remote Ops MacBook Pro. This was the quickest and most reliable implementation, as both the on-robot NUC 11 and MacBook Pro used CMU-Secure Wi-Fi.

5.6.3.5. GNSS GPS Module

As a stretch goal, GPS capabilities on ScottyBot could be implemented for better navigating outdoor environments. There are several other caveats to consider when extending the scope of ScottyBot to outdoor environments that include but are not limited to congestion monitoring outdoors, weather conditions, power management, and localization accuracy. Therefore, GPS will not be implemented on ScottyBot at this time.

5.6.4. Interactions

5.6.4.1. Remote Ops

The Remote Ops team is in charge of sending wireless control actions to the robot through a JavaScript React application.

In order to facilitate their communication, we registered their personal laptop as an AWS device. We then create an AWS topic for Remote Ops to publish their commands (up arrow = “forward”, down arrow = “back”, left arrow = “left”, right arrow = “right) from the registered laptop to the Planning and Controls team on the on-robot NUC. The command then gets propagated to the Robot Hardware Team as motor commands.

5.6.4.2. Planning and Controls

The Planning and Controls (P&C) team will interact with the IoT Core through the Python interface we have provided to set up a callback function, which will trigger upon a publish event to their shared topic with Human-Robot Interaction (HRI) to get a route destination. After HRI extracts the destination from the user’s speech, it publishes it for P&C. P&C’s callback function then processes this desintaiton, plans a path accordingly, and begins autonomous navigation via SLAM.

5.6.4.3. Human-Robot Interaction

The Human-Robot Interaction (HRI) team’s webapp backend will interact with the IoT Core through the Python interface we have provided. Their webapp is hosted locally on the NUC.

We subscribed the NUC (more specifically HRI's webapp) to the NFC ID card topic and set up a callback function which will trigger upon an NFC ID card read. This allows the user to bypass the login page by emitting a message to the webapp's frontend via a socket.

Additionally, after HRI's LLM extracts the destination from the user's speech, it publishes it for P&C. P&C's callback function then processes this desintaiton as described above.

5.6.4.4. Infrastructure Software

Infrastructure Software will interact with the IoT Core interface through their ESP32 software, and use this to send and receive traffic congestion monitoring, elevator control, and traffic light controls. Other teams such as the Planning and Controls team will receive this data through a subscription to a congestion topic, and provide a callback function that halts autonomous control if the area is too congested.

5.6.4.5. Data Systems

Data systems will interact with the Communications pipeline via Lambda functions. Our infrastructure is set up to invoke a specific Lambda function when specific topics are published to. These Lambda functions will then be leveraged to process and store relevant data. Most notably, Data Systems receives user information from NFC ID card readings.

5.6.5. Deliverables

Detailed below is the On-Robot and Off-Robot code structure that was used for wireless communication flow.

5.6.5.1. On-Robot

The demo_main.py file contains the NUC 11 configuration with AWS IoT Core and MQTT that:

1. Connects the NUC to AWS IoT Core as a device client using a certificate, private key, and root.
2. Subscribes to necessary topics, such as congestion from Infra SW and destination from HRI.

Next, cyclic-executive code runs to do the following:

1. Waits until the NFC ID card is scanned (publish to HRI), configures P&C and Perceptions libraries, and waits for a destination from HRI (P&C subscribe)
2. Continue autonomous navigation and planning, and when congestion is too high, quit autonomous navigation, subscribe to Remote Ops's manual control topic, and continue listening to their commands and translate them to Robot HW.

```
"""
device_client defines a data structure for a Thing Object with the following attributes:
 - root_cert: Path to the root certificate file.
 - private_key: Path to the private key file associated with the device.
 - cert: Path to the certificate file associated with the device.
 - subscriptions: List of MQTT topics to subscribe to.
 - publications: Dictionary mapping MQTT topics to lists of messages to publish.
"""
class device_client:
    def __init__(self, root, key, cert, client_id, sub_topics, pub_topics):
        self.root = root
        self.key = key
        self.cert = cert
        self.client_id = client_id
        self.sub_topics = sub_topics
        self.pub_topics = pub_topics

    def connect(self):
        self.myAWSIoTMQTTClient = AWSIoTPyMQTT.AWSIoTMQTTClient(self.client_id)
        self.myAWSIoTMQTTClient.configureEndpoint(ENDPOINT, 8883)
        self.myAWSIoTMQTTClient.configureCredentials(self.root, self.key, self.cert)
        self.myAWSIoTMQTTClient.connect()

    def disconnect(self):
        self.myAWSIoTMQTTClient.disconnect()

    def publish(self, topic, msg):
        data = "{}".format(msg)
        self.myAWSIoTMQTTClient.publish(topic, json.dumps(data), 1)

    def add_subscribe(self, topic, callback):
        if topic not in self.sub_topics:
            self.sub_topics.append(topic)
            print("Subscribed to the topic: " + topic)
            self.myAWSIoTMQTTClient.subscribe(topic, 1, callback)
        else:
            print("Error subscribing to " + topic)

    def del_subscribe(self, topic):
        self.sub_topics.remove(topic)
        self.myAWSIoTMQTTClient.unsubscribe(topic)
```

Shown above: The device_client class in iot_interface.py containing a constructor and relevant methods

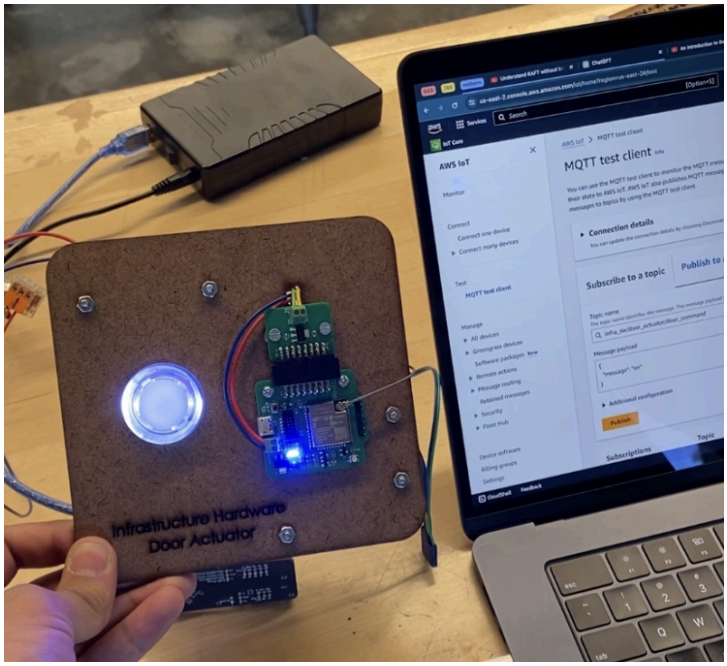
```python
def main():
    global motor_port
    global arrived
    global destination
    global congestion_status
    # Initialization
    nuc_computer = device_client(
                root = r"/home/snuc/Communications/AmazonRootCA1.pem",
                key = r"/home/snuc/Communications/6be221a2f7c31b06e6085277bee22e2e730ae5f1479975da26dae2685416d369-private.pem.key",
                cert = r"/home/snuc/Communications/6be221a2f7c31b06e6085277bee22e2e730ae5f1479975da26dae2685416d369-certificate.pem.crt",
                client_id = "nuc",
                sub_topics = [], #empty for now - add topics using add_subscribe()
                pub_topics = {}
    )

    nuc_computer.connect()
    nuc_computer.add_subscribe("infra_sw/congestion_sensor1/congestion_data", congestionCallback)
    nuc_computer.add_subscribe("robot/control/destination", destinationCallback)
    nuc_computer.add_subscribe("robot/motors/control_status", arrivalCallback)
```

Shown above: The NUC 11 is registered and connected to AWS IoT Core as a device client/"thing" and is subscribed to relevant "topics" with subscribe() with relevant callback functions as the second argument
The NUC sends/publishes data with nuc_computer.publish("topic", message_as_dict)

5.6.5.2. Off-Robot

```
37        net.setCACert(AWS_CERT_CA);
38        net.setCertificate(AWS_CERT_CRT);
39        net.setPrivateKey(AWS_CERT_PRIVATE);
40
41        // Connect to the MQTT broker on the AWS endpoint we defined earlier
42        client.setServer(AWS_IOT_ENDPOINT, 8883);
43
44        // Create a message handler
45        client.setCallback(messageHandler);
```

Shown above: Similar to the NUC 11, each ESP32 is registered and connected to AWS IoT Core as a device client/"thing" and is subscribed to relevant "topics" with subscribe().
Its callback is set with setCallback().



Shown above: testing the Infrastructure teams' door open actuator.
The AWS IoT Core's MQTT test client tool was used to act as the device sending the signal to ESP32s.

The structure of the Remote Operations team's AWS IoT Core/MQTT configuration Javascript code running on the MacBook looks similar to the Python code running on the NUC.

5.7. Data Systems

5.7.1. Functionality

5.7.1.1. Data Storage

For our tasks, it was essential to find a database solution for storing, organizing and managing data. We have to select an appropriate implementation that aligns with the unique requirements of our IoT environment, complex data types, the project's scope, and cost considerations.

5.7.1.2. Data Ingestion

To enable the collection and processing of data from diverse sources and teams, it's crucial to have an efficient data handling mechanism in place. Additionally, it's important to consider practices that ensure flexibility and scalability to accommodate future use cases, including real-world scenarios.

5.7.1.3. Data Access and Querying

We require a good approach to establish connectivity between the data handler and the database. Additionally, it's crucial to provide appropriate interfaces to enable access to data for other teams for storage/loading/modification.

5.7.1.4. Data Transfer/Integration

Data integration provides data exchange and communication among various components within the system. Specifically, it addresses the interaction between cloud-based systems and hardware devices. This aspect is particularly important for enabling effective communication between components and supporting the system's overall functionality.

5.7.1.5. Data Management

Data management policy involves removing outdated information from a database collection based on a specified timestamp. This functionality helps manage the data stored in the MongoDB collection by periodically removing older records, ensuring that the database remains up-to-date and optimized for efficient querying and analysis.

5.7.2. Interactions

5.7.2.1. Communication

The team collaborated closely with us to ensure seamless data transmission and storage. Our system serves as the backbone for storing, organizing, and processing vast amounts of information, while communication channels facilitate the exchange of data and insights among various stakeholders, and develop efficient data processing pipelines using AWS Lambda and MongoDB. To be more specific, each team submits data handling requests to the Communication Team, which subsequently forwards them and calls AWS lambda function, which then processes the requests and provides the results accordingly. By integrating data systems with communication platforms and tools, organizations can streamline collaboration, enabling real-time sharing of data-driven insights, updates, and reports.

5.7.2.2. Infrastructure Software

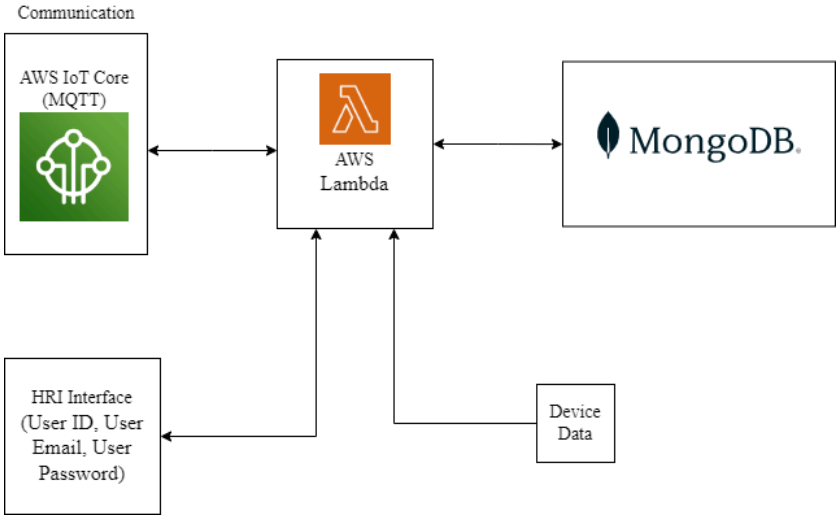
The team worked hand in hand with us to optimize data interaction architecture. They focused on enhancing the communication mechanism between the front and back end, aiming to streamline data flow processing and improve system response speed and reliability. Through collaborative efforts, they introduced advanced API management tools and middleware solutions, ensuring smoother handling of complex data requests and enhancing the overall user experience. The Infrastructure Software team's

feedback and requirements guided us in refining their database design and data handling processes, resulting in a more efficient and scalable infrastructure.

5.7.2.3. Remote Operations and Control

The team collaborated closely with us to facilitate real-time monitoring and control of the robot. We store and fetch critical location data, enabling personalized and efficient navigation experiences. The Remote Operations & Control team provided input to the Data Systems team regarding their data storage and retrieval needs, ensuring that the database design and data processing pipelines met the requirements of the remote control interface. Through effective communication and collaboration, both teams worked together to address challenges and refine their systems for optimal performance during the project's execution.

5.7.3. Software Architecture



5.7.4. System Design & Status

5.7.4.1. Data Storage

In terms of data storage solutions, we explored several options including AWS DocumentDB, self-managed MongoDB instances on EC2 and/or MongoDB Atlas and relational databases like postgreSQL. While AWS DocumentDB offers a fully managed native JSON document database ideal for handling critical document workloads at scale without the need to manage infrastructure, we found it to be cost-prohibitive for our project, which is relatively small in scope. For postgreSQL, the data we are using is better suited to using JSON vs using a relational database, as well as schema management. As a result, we opted to deploy and manage a MongoDB instance on EC2 ourselves, which helped reduce costs significantly but required ongoing efforts to maintain the instance and environment. Eventually, we transitioned to using MongoDB Atlas, a fully managed database service that provides all the features of MongoDB while

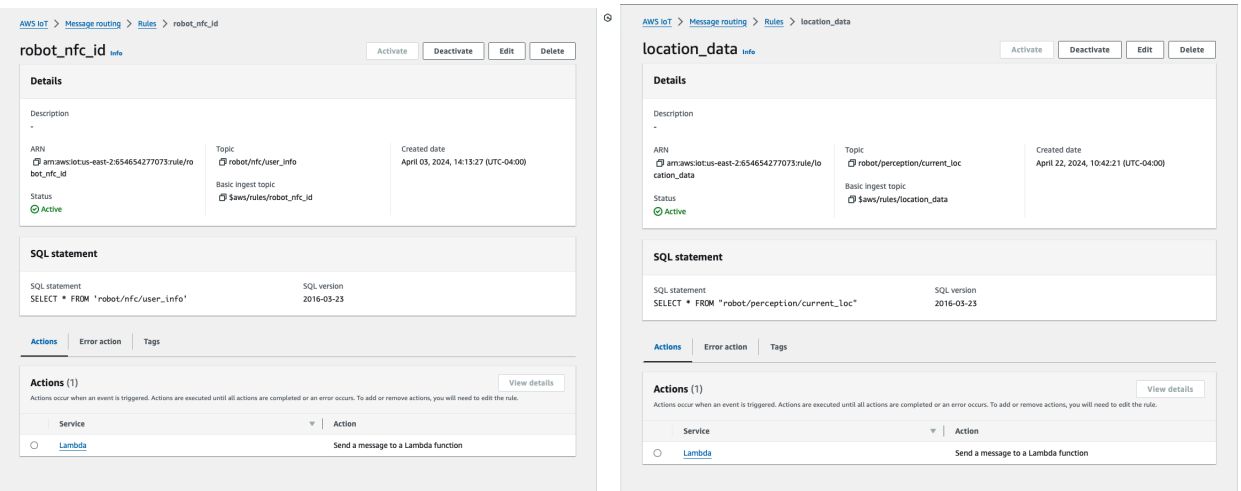
offering the convenience of being fully managed. Overall, MongoDB Atlas proved to be the optimal choice for our needs, offering advantages in terms of usability, maintenance, and cost-effectiveness.

5.7.4.2. Data Ingestion

To facilitate the ingestion of data from diverse sources and teams, we leveraged AWS Lambda as a versatile handler for data processing tasks. This approach offers the flexibility of event-driven functions, ensuring extensibility and scalability for future requirements. For instance, in scenarios where other teams necessitate database interaction via HTTP connections, we can seamlessly integrate AWS Lambda with Amazon API Gateway. As our project scales, we have the capability to adapt the Lambda function to queue data ingestion through AWS SQS for subsequent processing, or to distribute data across multiple database nodes. In summary, the utilization of AWS Lambda provides a resilient foundation for accommodating evolving data ingestion needs while maintaining flexibility for future enhancements.

5.7.4.3. Data Access and Querying

Within our AWS Lambda function, we established connectivity with MongoDB Atlas by importing the pymongo library and storing the connection URL in Lambda environment variables. While best practices dictate securing sensitive information in real-world deployments, such as utilizing services like AWS Secrets Manager, our current setup suffices for the scope of this project. We have implemented separate functions to handle requests from other teams, ensuring comprehensive support for CRUD operations. Moving forward, this setup ensures good integration and efficient data management for our project's requirements.



5.7.4.4. Data Integration

We have integrated AWS Lambda functions with AWS MQTT for data handling. These functions are event-triggered and subscribe to data-related topics on MQTT,

receiving data published by other teams. Additionally, the functions have the capability to publish data to specific MQTT topics. Through AWS IoT console subscription to these topics, we ensure efficient interaction and data exchange between cloud-based systems and hardware devices. This integration enhances our system's flexibility and enables effective communication across various components.

5.7.5. Team Deliverables

In phase 3, one critical aspect was the adaptation of data interfaces and types required for seamless integration with other subsystems. As the other teams finalized their preparations, our team commenced the reception of real-world/dummy data and extended the relevant Lambda functions accordingly. This process involved accommodating various data types or refining data processing mechanisms to ensure compatibility and efficiency. Additionally, as a stretch goal/future work, we were looking at exploring the integration of a second database framework, such as PostgreSQL, provided that all data requirements from other teams have been successfully met. This expansion would enhance the system's flexibility and scalability, accommodating diverse data processing needs effectively.

```python
client = MongoClient(host=os.environ.get("ATLAS_URI"))

def lambda_handler(event, context):
    # Determine accessed database and collection
    print("event is ", event)
    if "team" not in event:
        event = event['body']
        print("event body is ", event)
    if event["team"] == "perception":
        db = client.perception
        if event["data"] == "location":
            collections = db.location
        else:
            return {"message": "Invalid data name"}
    else:
        return {"message": "Invalid team name"}

    if event["type"] == "store":
        return store_Info(event, collections)
    elif event["type"] == "fetch":
        return fetch_Info(event, collections)
    else:
        return delete_info(event, collections)

def fetch_Info(event, collections):
    num_row = event['n']
    loc = collections.find({}).sort({"timestamp": -1}).limit(num_row);
    if loc:
        documents = [{**doc, "_id": str(doc["_id"])} for doc in loc]
        # documents = list(loc)
        print("docs is ", documents)
        for doc in documents:
            doc['timestamp'] = doc['timestamp'].strftime("%Y-%m-%dT%H:%M:%S.%fZ")
        return documents
    else:
        return {"message": "No match timestamp found!"}
```

Aligning with the data management policy is another essential aspect of our team's deliverables for phase 3. This involves determining retention periods for collected data to ensure compliance with regulations and minimize storage costs. For example, we plan a 60-day retention period for device data, which includes tracing the time and location of ScottyBot in our system for future reference. Other information passing through the communication system, such as logs identified with important attributes, will also be retained. Additionally, our team will track registered and unregistered users who attempt to use our service, ensuring that data is kept only for as long as necessary for its intended use, thereby reducing the risk of unauthorized access or data breaches. Furthermore, our team will oversee data and device disposal processes to ensure that sensitive information is properly handled and disposed of in accordance with security protocols. We will also establish classification and handling protocols for different types of data, including private, internal, and public data, to enforce strict access controls and maintain data integrity.

5.8. Infrastructure Hardware

5.8.1. Functional Requirements

5.8.1.1. Automatic Door Opening

In order to allow ScottyBot to travel between different buildings, it must be able to open entrance doors to buildings. Rather than creating a complex solution that would use a physical actuator to open the door or press the handicapped switch to open the door, we are electing to use a purely electronic solution. Our solution PCB will electrically connect to the two wires that are attached to the handicapped switch and short the two wires together electrically, mimicking a button press. Since we only need access to the wires connected to the button, our PCB will be unobtrusive and easy to install on any automatic door with a handicapped button.

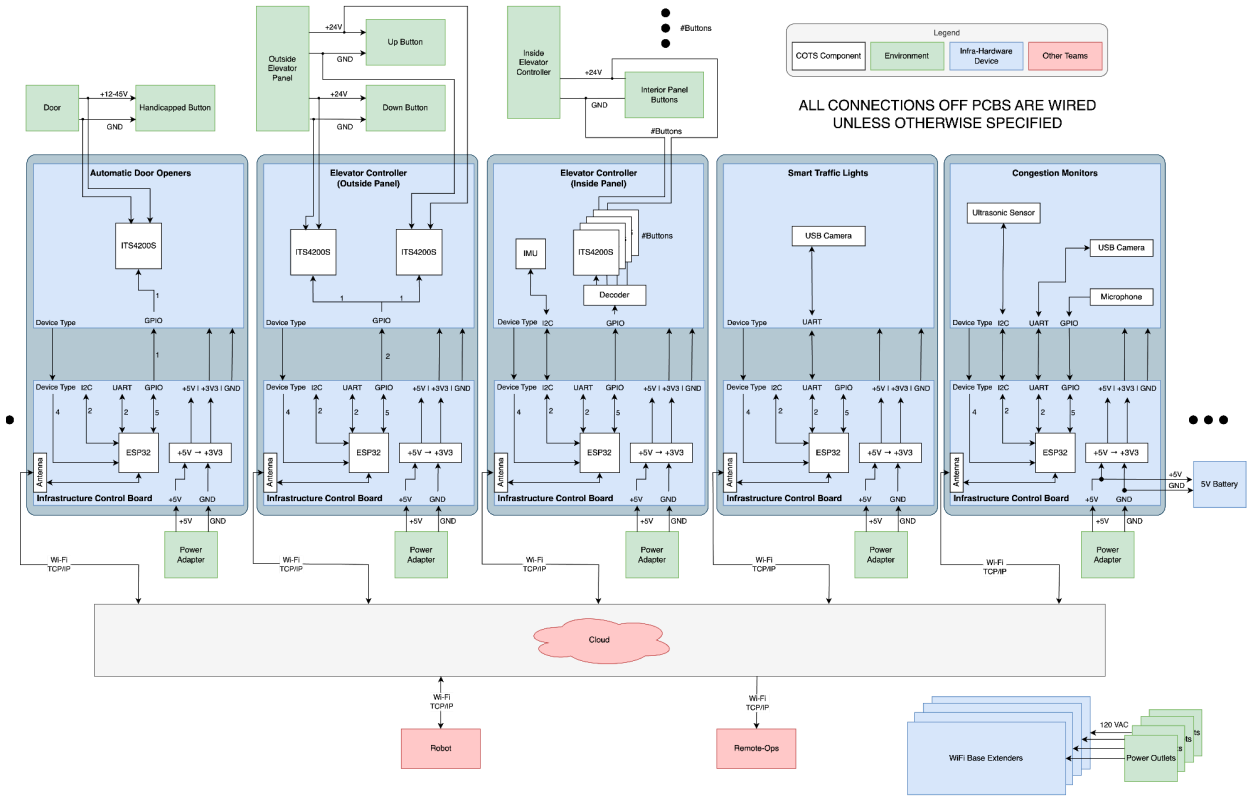
5.8.1.2. Elevator Opening/Control

In the future, ScottyBot must also be able to traverse between different floors of buildings; since it is a wheeled robot, it must use the elevator to do so. We are planning to use a similar solution to our automatic door opening PCB discussed above to press the elevator buttons. Our solution for elevators will consist of two separate PCBs: one for the panel outside the elevator to call it, and one for the panel inside the elevator. The PCB connected to the panel outside the elevator will be electrically connected to the buttons to call the elevator. The PCB inside the elevator will use a decoder to electrically connect to the array of buttons on the elevator panel. This PCB will also contain an IMU to track which floor of the building the elevator is currently on, and bluetooth signal connection with the PCB outside the elevator to detect the current floor.

5.8.1.3. Congestion Monitoring

In order to ensure students and ScottyBot can get to their destination safely, it is important for ScottyBot to take routes with low congestion. To provide ScottyBot with this information, we plan to create a congestion monitoring PCB containing both visual observation with ultrasonic sensors, and auditory monitoring with microphones. With this information, ScottyBot will be able to tell if certain hallways/areas have lots of people and can assist with pathfinding to avoid high congestion.

5.8.2. System Architecture



Our system will consist of six different types of PCBs: an automatic door opening PCB, an outside elevator panel PCB, an inside elevator panel PCB, a congestion monitoring PCB, and a control PCB.

The control PCB is responsible for controlling each of the other five environment PCBs and for communicating with ScottyBot. Each control PCB will have an ESP32 microcontroller and a connector going off board that will interface with the ESP32's GPIO, I2C, and UART pins. The other PCBs will connect to the control PCB through the connector so that the control PCB can write to or read from the other PCBs based on information from ScottyBot. Rather than having an ESP32 on each environment PCB, we elected to have it only on the control PCB so that the infrastructure software team only has to write firmware for one PCB.

Four of the pins in the connector between the control PCB and environment PCBs will be dedicated to a board ID - each type of environment PCB will have a different ID that can be recognized by the control PCB. This way our system can be plug and play - any one of our environment PCBs can be plugged into a control PCB, and it will automatically be recognized and work without needing to change code.

The ESP32 on the control PCBs will connect to ScottyBot using a wifi connection. We will install antennae onto each control PCB so that we can extend the range of our PCBs and avoid dead zones.

The button actuating PCBs (automatic door opener, outside elevator controller, inside elevator controller) will use voltage controlled switches that will be connected to

the buttons that need to be actuated. The control PCB will then give the button actuating PCB a command to bridge the connection using the switch, which will then cause the button to be “pressed”. Since the inside elevator panel will have many more buttons than the other two PCBs, we will be using a decoder to keep GPIO pin usage low on our ESP32.

The monitor board will communicate and be managed by the control board through several GPIO pins available for use. The 3 ultrasonic sensors and microphones on each monitor board will provide data on movement activity.

5.8.3. Hardware Design

5.8.3.1. Control Board

To reduce the setup time required for each new device/peripheral designed, we decided to design a singular control PCB. We will then reflow multiple of this, which can then be plugged into any peripheral to control. This makes it easier for the software team to interface with, as they will only need to write for one standardized MCU. Our control board is designed around the ESP32, with a standardized pinout that all the peripheral boards are designed around.

5.8.3.2. Door Actuation

The door actuation boards are essentially just relays that are designed to be easily integrated into door panels. The door actuator consists of a singular relay that will integrate into existing motorized doors (typically for the handicapped). These will be utilized by ScottyBot to access areas which are separated by a door.

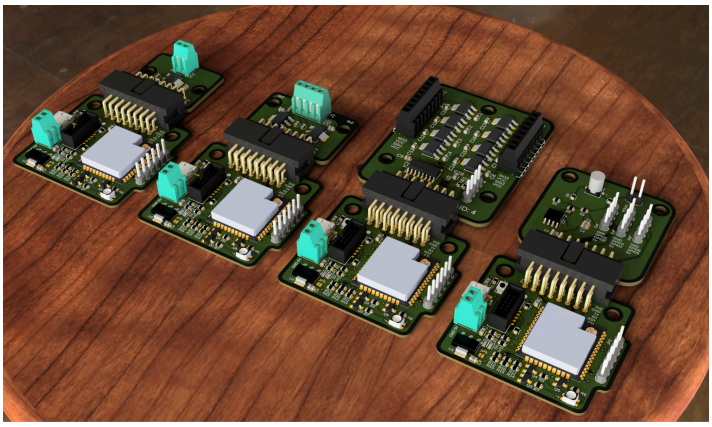
5.8.3.3. Elevator Actuation

Similar to door actuation, we have designed boards to integrate with both the outside and inside panels of elevators. The outside panel is just two relays to actuator the up and down buttons. Meanwhile, the inside panel utilizes a 3-to-8 decoder to expand how many relays we are capable of utilizing. These will be utilized by ScottyBot to traverse multiple floors of a building.

5.8.3.4. Congestion Monitoring

Separate from physical actuation, we have also designed a system that can monitor the congestion level of specified areas of the campus. These utilize 3 ultrasonic sensors that monitor activity. Through the communications server setup by the software teams, these can help inform the pathfinding on which areas are less optimal for ScottyBot to traverse through during times of heavy traffic.

5.8.4. Final Deliverables



5.8.5. Next Steps

Now that we have working mockups of each of the designed demos, there are several improvements that can be developed if given the time for another revision. Firstly, the congestion monitoring system can be improved upon to utilize metrics beyond just ultrasonic sensor detection. In addition, the final demo did not demonstrate the robots capability to remotely request a door to be opened in an end-to-end manner, so this would be a logical next step in creating a full implementation. Furthermore, although the actuation is functional for the mockups, they have yet to actually be tested within doors/elevators. This would entail more closely working with the communication and other software teams.

5.9. Infrastructure Software

5.9.1. Functionality

As part of the infrastructure software team, our main task is to develop the software that enables ScottyBot to interact smoothly with its surroundings. We focus on two core functions: sensing data and taking action. There are three main functionalities we will support:

5.9.1.1. Congestion Monitoring

We manage the congestion data, acquired through ultrasonic sensors and store it in a database for ScottyBot or operators to access. This data helps ScottyBot plan its routes efficiently and warn users about potential traffic jams. Integrating data sensing and actuation functionalities, our infrastructure software empowers ScottyBot to sense and control dynamically changing environments.

5.9.1.2. Automatic Door Opening

The ScottyBot will have to travel into classrooms and therefore it needs to open the door automatically. We receive automatic door opening messages from the communication team and execute the door opening command. Our solution is to use a simple ESP32 program to represent the door opening signal and pass it to the infrastructure hardware team.

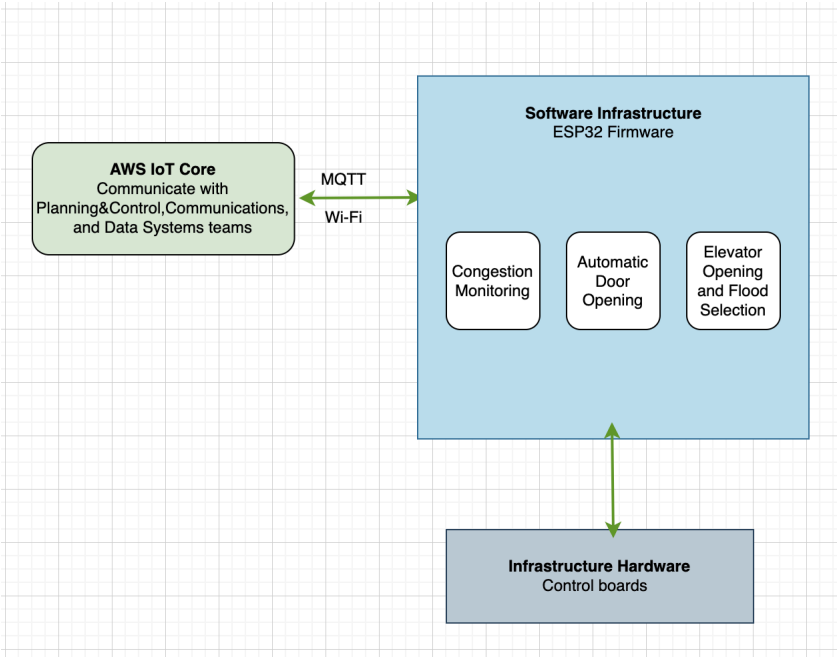
5.9.1.3. Elevator opening and floor selection

ScottyBot also needs to call the elevator to travel to different floors. In order to do that, we created two separate ESP32 programs for the elevator opening and floor selection functionalities. For the elevator opening functionality, ScottyBot calls the elevator by selecting whether it wants to go up or down. For the floor selection functionality, ScottyBot picks which floor to go, there are 3 signals to represent 8 floors.

5.9.2. Software Architecture

We leverage the AWS IoT Core platform along with the MQTT message framework to expose the software layer, facilitating efficient communication with other

teams such as the Planning and Controls team, Communications team, and Data Systems team. Our core software functions are built on the ESP32 firmware, it consists of three main modules: Congestion Monitoring, Automatic Door Opening, and Elevator Open and Floor Selection. The ESP32 code of these 3 modules runs on the control boards built by the Infrastructure Hardware team, simulating the infrastructure functionalities.



5.9.3. Interactions

5.9.3.1. External Interactions with the Real World

The Infrastructure Software team is instrumental in bridging the gap between the digital and physical realms within the facility's ecosystem. Through the deployment of sophisticated ultrasonic sensors, our team captures real-time data on pedestrian traffic, exemplifying our direct engagement with the external environment. This initiative not only enhances the safety and efficiency of pedestrian movement but also provides valuable insights into space utilization patterns. Moreover, the automatic door and elevator control systems represent pivotal points of interaction, receiving external inputs from the Planning and Controls Team to execute precise operations that affect the physical space. These systems, governed by an API we developed, stand as a testament to our commitment to creating responsive and adaptive infrastructure that meets the dynamic needs of the facility.

5.9.3.2. Internal Connections Within the System

Internally, our team's efforts are characterized by a seamless web of connections that facilitate communication and data exchange among various technical teams. The

API that interfaces with the Planning and Controls Team is just one layer of this intricate network. To clarify, the Planning and Controls Team will transmit robot action decisions as JSON messages to the communication team. The communication team will then relay these messages to us, enabling our software to issue commands to the infrastructure hardware team. This communication is facilitated through our APIs, utilizing MQTT and AWS. Beyond acting as a conduit for operational commands, this API serves as a cornerstone for interoperability, ensuring that different systems and teams can collaborate effectively without silos.

The collaboration with the Data Systems Team showcases our internal connectivity, as pedestrian traffic data is systematically streamed to the cloud and integrated into a centralized database. This harmonized data flow enables comprehensive analysis and strategic decision-making, reinforcing the infrastructure's adaptability and intelligence.

Our partnership with the Infrastructure Hardware Team further underscores the symbiotic relationship between software and hardware. By aligning our software developments with the physical components managed by the hardware team, we ensure a cohesive and efficient infrastructure. This alignment is crucial for the real-time responsiveness of our systems, from sensor-based monitoring to the activation of doors and elevators, illustrating the deep interconnectivity that underpins our operational framework. Specifically, we have implemented the connection of the ESP32 to Amazon AWS IoT Core using MQTT. Additionally, we can program the ESP32 using the Arduino IDE to control physical components.

5.9.3.3. Software Modules and Status

As introduced previously, the infrastructure software along with infrastructure hardware are mainly responsible for four main components of the final system, namely: congestion monitoring, door opener, elevator call, and elevator floor select. The four systems have many commonalities, and it was designed with expandability and customization in mind. All 4 modules communicate data through AWS cloud’s IOT Core, using one set of secret keys and WiFi info. This allows simple expandability in the future to include other infrastructures such as traffic signal monitoring.

For congestion monitoring systems, it uses an HC-SR04 ultrasonic sensor and an ESP32 microcontroller. The system operates by continuously detecting and logging the distance to an obstacle in front of the sensor into a queue every second, utilizing a rolling window of 1 minute. By comparing consecutive distance measurements within this window and checking if the difference exceeds a predefined sensitivity threshold, the system determines the number of people passing within the last minute. Based on this count, the program categorizes the congestion level as low, medium, or high using predefined thresholds. To provide real-time feedback, both the congestion level and the number of people passing in the last minute are printed to the serial monitor every

second, as well as pushed to the AWS cloud for the planning and controls to use in the planning algorithm. The congestion level thresholds, rolling window size, and other parameters are easily tunable by changing the defined parameters at the beginning of the code file. This implementation offers a versatile and adaptable solution for congestion monitoring in various environments.

The door opener and elevator call functions are similar in logic. Both functionalities subscribe to an AWS topic. When a message is sent to actuate the door or call the elevator, the code will decode that message and turn a corresponding GPIO pin into HIGH. The GPIO signal will then trigger infrastructure hardware's simulated elevator PCB, causing a relay to close and in turn triggering the physical door opener button or the elevator call button.

The elevator floor select function is a little more complicated, since most buildings have multiple floors that easily exceeds the available GPIO pins on an ESP device if the floor to GPIO mappings are one to one. Instead, a binary combination of the three GPIO pins are used to trigger the floor select commands. As shown in the diagram below, different combinations of the three GPIO pins represent different floors or door open/ close commands. An enable pin is added, which only toggles after the three GPIO pins are set to signal to the PCB board to send the actuation commands. This avoids any misalignments in the signal and avoids potential errors. The program can be easily extended to operate in buildings with higher floors by increasing the number of GPIOs used in the binary select process.

GPIOs S0, S1, S2	Enable	Floor Select
000	Enable? ——>	1
001		2
010		3
011		4
100		5
101		6
110		Door Open
111		Door Close

The team also demonstrated the seamless transmission of sensing data to the cloud and the reception of actuation commands. The ESP32 can connect to the AWS IoT core using MQTT. After configuring and registering the ESP32 with the necessary permissions, the ESP32 can start communicating with AWS over wifi using MQTT. The AWS MQTT broker sets up two topics, congestion monitoring topic and actuation topic . The ESP32 module can send congestion monitoring data to AWS IoT core by publishing it to the congestion monitoring topic. The ESP32 module receives actuation commands by subscribing to the actuation topic that the AWS IoT core publishes to. This capability can be extended to send data from any sensor or control any actuator with control commands. For better management, some example topic names used in the final version of the code are: "infra_sw/door_actuator/door_command", "infra_sw/elevator_out/elevator_command", and "infra_sw/congestion_monitoring/congestion_data".

5.9.4. Final Team Deliverables

For phase 3 , our team delivered the following features

Door opener:

Receiving open/close command from planning and controls and opening the door(toggle GPIO pin for infrastructure hardware door mockup)
topic name: “infra_sw/door_actuator/door_command”
message format: “on”

Elevator call:

Receiving a up/down from planning and controls through and controlling the elevator(, toggle up GPIO or down GPIO for the infrastructure hardware elevator mockup)
topic name: “infra_sw/elevator_out/elevator_command”
message format: “up” or “down”

Elevator floor select:

Receiving a command of integer number (1-8) from planning and controls , turning a combination of 3 GPIOs high (binary representation of the integer) accordingly and the select a floor (toggle the necessary GPIOs on the infrastructure hardware elevator mockup to select a floor)
topic name: “infra_sw/elevator_in/elevator_command”
message format: “1” or “2" or ... “8”

Congestion monitoring:

Sending the congestion level (low = 0, medium = 1, high = 2) to planning and controls through communications
topic name: “infra_sw/congestion_monitoring/congestion_data”

message format: “0” or “1" or “2”

5.10. Remote Operations & Control

5.10.1. Functionality

The Remote Operations & Control team is responsible for monitoring, managing, and controlling the ScottyBot fleet through a comprehensive User Interface application. Their primary focus is to ensure the smooth and efficient operation of the robots in real-time, leveraging various features and functionalities provided by the application.

The team utilizes the Overview page to gain a holistic view of the system, including the map view, live streaming, and robot details. This allows them to quickly assess the overall status and performance of the ScottyBot fleet. The Map View page enables the team to monitor the real-time locations and trajectories of the robots, identifying any potential issues or congestion. Through the Live Streaming page, the team can visually assess the robots' surroundings using real-time video feeds captured by the robots' cameras. This functionality, powered by WebRTC technology, facilitates informed decision-making and enhances situational awareness.

The Robot Details page provides the team with comprehensive information about each individual ScottyBot, including operational status, battery level, speed, and maintenance status. This data, sourced from the Robot Hardware team, enables the Remote Operations & Control team to monitor the physical condition of the robots and ensure their optimal performance. Additionally, the Robot Details page displays mission-related information such as ETA, destination, user's name, number of people in the queue, and request status, which is obtained from the Communications team. This allows the team to track the progress and status of each robot's assigned tasks and missions.

In situations where manual intervention is required, such as navigating through congested areas, the Remote Operations & Control team utilizes the Controls page. This page provides a manual control interface, allowing operators to remotely operate the ScottyBots using the WASD keys on the keyboard. By taking direct control of the robots, the team can navigate them through challenging situations and ensure smooth operation.

The Remote Operations & Control team plays a vital role in ensuring the efficient and effective deployment of the ScottyBot fleet. By leveraging the functionalities provided by the User Interface application, they can monitor, manage, and control the robots in real-time, making informed decisions based on the data and live video feeds available to them. Through their expertise and the tools at their disposal, the team strives to optimize the performance of the ScottyBots, enhance the user experience, and maintain a seamless and reliable service.

5.10.2. Interactions

5.10.2.1. Overview Page

The Remote Operations & Control application features a user-friendly interface divided into several key sections: the Overview page, Map View page, Live Streaming page, Robot Details page, and the Controls page. The Overview page integrates the Map View, Live Streaming, and Robot Details into a cohesive dashboard, providing a comprehensive snapshot of robot operations. The Live Streaming page displays real-time video from the robot's front camera, while the Map View page tracks the robot's current location.

5.10.2.2. Live streaming

The live streaming page is designed to display a real-time video feed captured by a camera mounted on the front of a robot. This camera, which is part of the robot's perception system, captures visual data that is essential for operations that involve navigation and interaction with the environment. The video stream is transmitted and displayed using WebRTC (Web Real-Time Communication) and PeerJS technologies. WebRTC enables direct, real-time video communication between browsers and devices over the internet, while PeerJS simplifies the process of WebRTC peer-to-peer data sharing by providing a comprehensive API.

5.10.2.3. Robot Details

The Robot Details page provides comprehensive information about the robot, including its operational status, battery level, speed, and maintenance condition. This section also displays mission-specific details such as the estimated time of arrival (ETA), destination, user’s name, queue length, and the status of requests. The robot's technical data and user-related information and mission details like the user's name, destination, and ETA are retrieved from the Communications team through subscriptions to AWS IoT Core. This setup ensures that all pertinent information is centralized and updated in real time, facilitating efficient monitoring and management of the robot's operations.

5.10.2.4. Controls page

The Controls page facilitates manual operation of the Robot using the WASD keys on the keyboard, which is particularly useful in congested areas. Control commands are relayed to the communications team by publishing them to the AWS IoT Core. These commands are subsequently forwarded to the planning and controls team to maneuver the Robot. Additionally, the interface includes a lock button that turns green to indicate when manual control is necessary due to congested conditions. There is also a 'Release' button provided, which, when activated, allows the Robot to either return to autonomous operation or proceed with its pre-programmed tasks once manual control is disengaged.

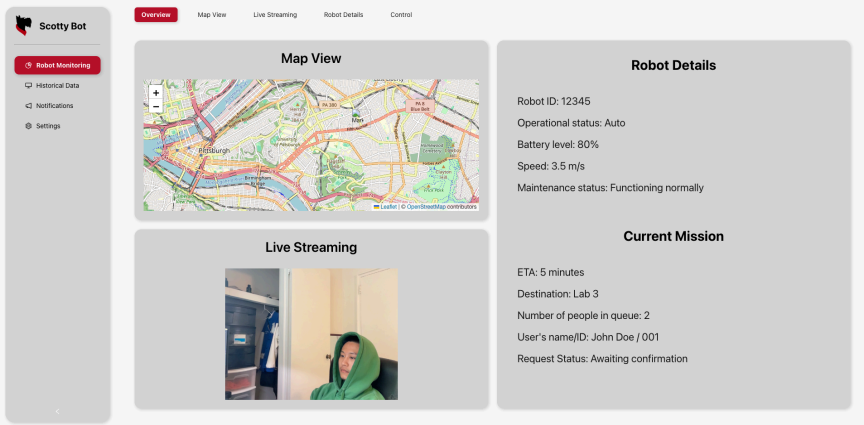
5.10.3. Screens

5.10.3.1. Robot Monitor

The "Robot Monitor" screen is the most crucial and relevant for this project's scope. The other screens cover additional functionality that could be implemented if time permits. The "Robot Monitor" section displays the robot fleet (outside current scope) and essential information about each individual robot, such as ID, activity status, and functionality status. When a user clicks on a specific robot, they are taken to second-level screens with five different views.

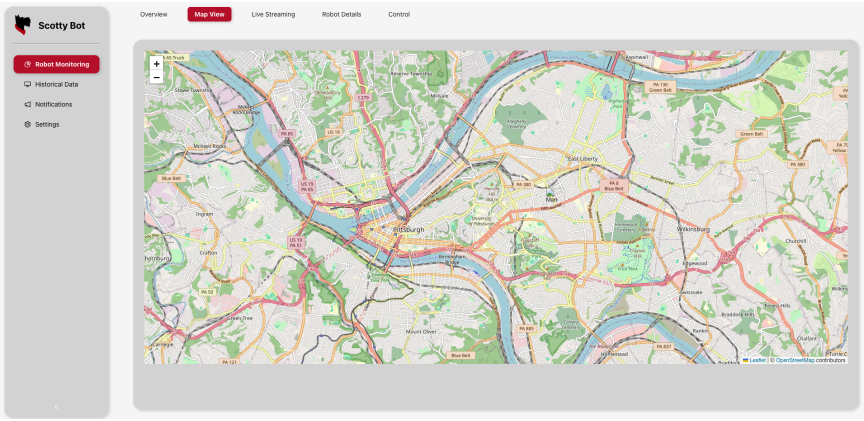
Overview

An integrated view showing the robot's location on a map, live camera feeds from the front and rear, and robotic details like ID, activity, and functionality.



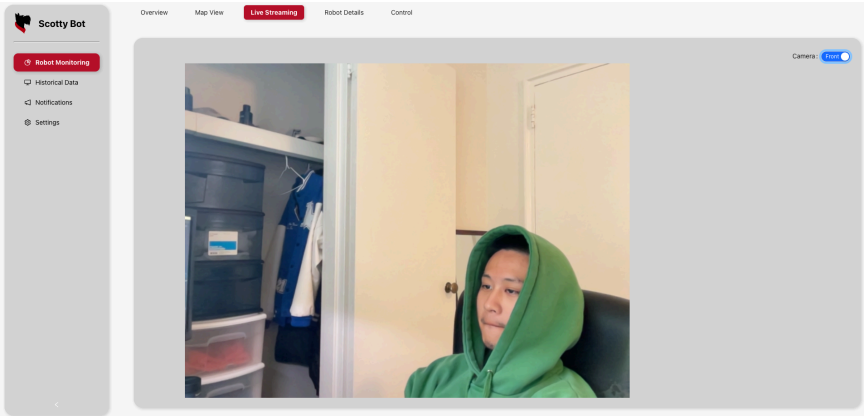
Map view

Displays the robot's current location, intended destination, navigation route, estimated arrival time, etc.



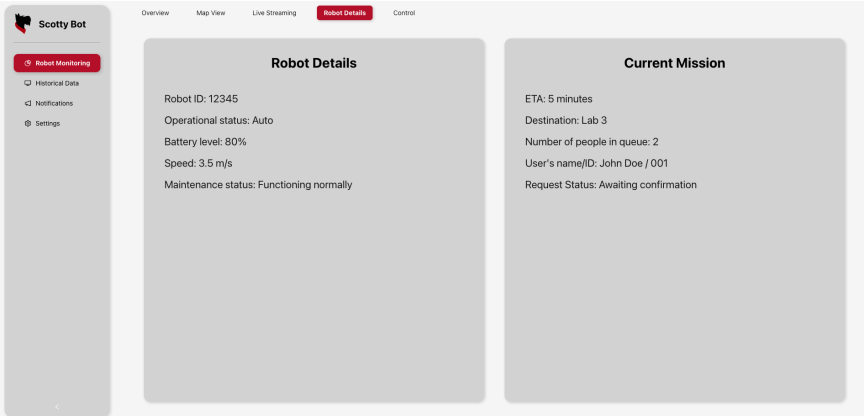
Live streaming

Shows the live video feeds from the front and rear cameras on the robot.



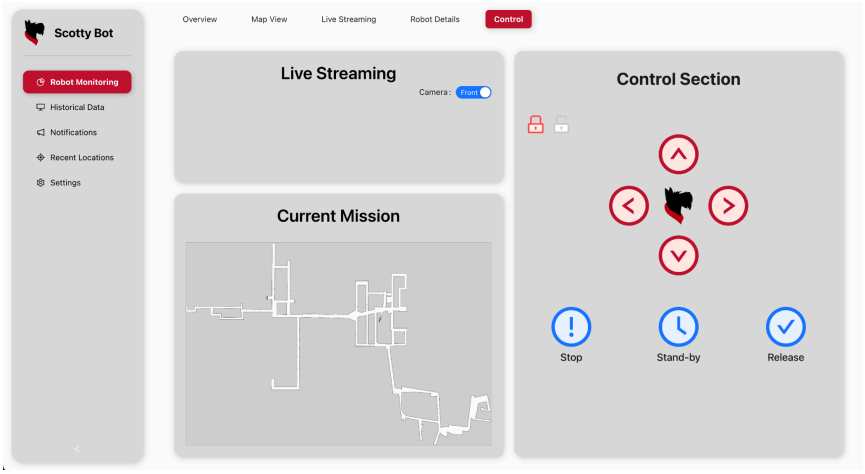
Robot Details

Presents comprehensive information about the robot, including ID, operational mode (auto/manual), battery level, speed, and maintenance status.



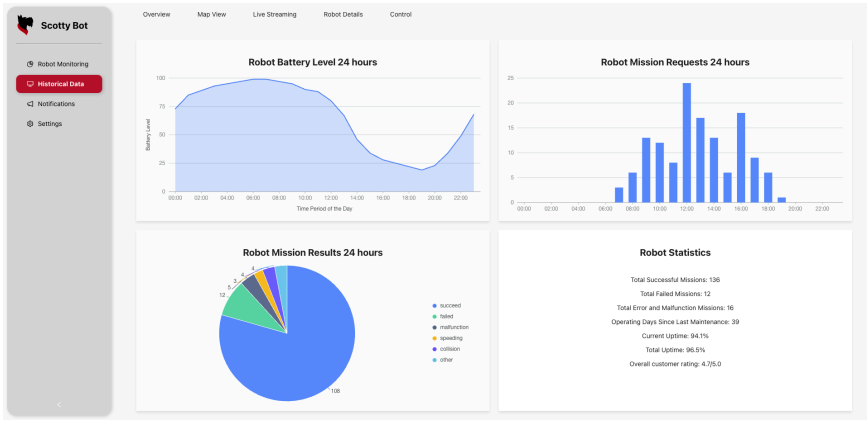
Controls

An interface for remotely operating the robot, incorporating live front and rear camera views. The red lock indicator will turn green once send signal to AWS and when it's green the controller is able to control the robot. Also update the map which now can correctly show the robot's current position.



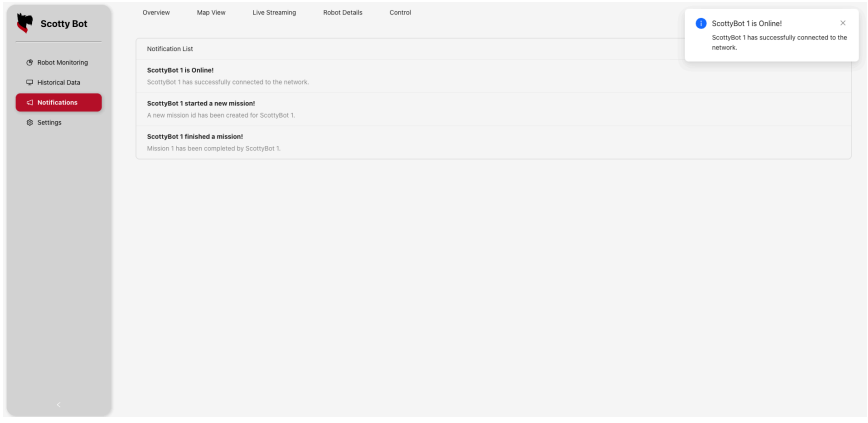
5.10.3.2. *Historical Data*

This page provides the status of a robot in terms of mission history and battery history.



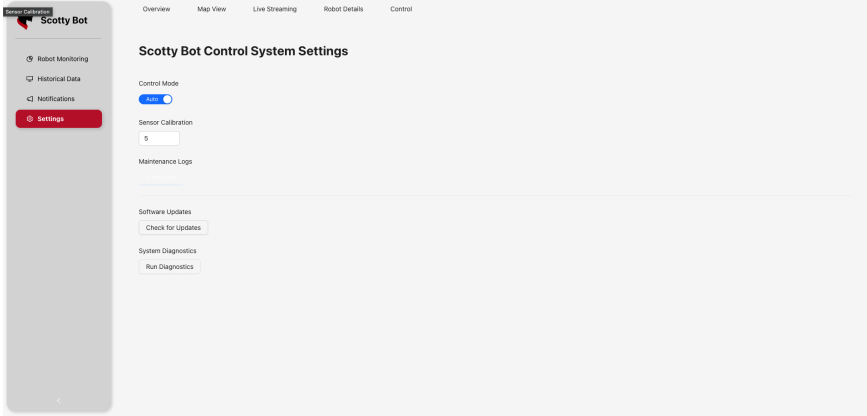
5.10.3.3. *Notification*

This section provides a feature that a notification window will pop out on the upper right corner when there is a notification.



5.10.3.4. *Settings*

This section provides the option to switch between auto control mode and manual control mode, for the human monitor to take over the robot when a situation occurs, such as navigating through crowds and emergencies.



5.10.4. Software Architecture

5.10.4.1. Design Philosophy

Our design philosophy is centered on creating a codebase that is modular, intuitive, and forward-looking. We achieve this through strategic modularization and judicious use of React's functional components and Context API. These approaches ensure our components remain focused, reusable, and adaptable, which is essential for both current utility and future development.

5.10.4.2. State Management and Components

Contextual State Management: We leverage React's Context API to facilitate a more fluid state management system, allowing us to transmit stateful information across the component tree without prop drilling. This keeps our component interfaces clean and focused on their visual roles.

Functional Components with Hooks: Our components are built as functional units with the useState and useEffect hooks handling local state and effects. This pattern aligns with the modern React paradigm, promoting simplicity and encapsulation while allowing us to tap into lifecycle features previously only available in class components.

Core Components as UI Building Blocks: We maintain a library of core components - LiveStreaming, Map, and RobotDetails, among others - which act as the foundational elements of our interface. These components are built to be autonomous, ensuring that changes or enhancements to one do not necessitate alterations to others, thus streamlining the update process and minimizing side effects.

5.10.4.3. Scalability

By structuring our application in this manner, we not only prepare our system for scalability and maintainability but also create an environment where testing and quality

assurance processes can be implemented with greater ease and efficiency.

State and Dependency Management: Utilizes React Context and Hooks to manage and distribute the application state across components, preparing for the eventual integration with Redux or another state management library for more complex state handling in future if needed.

5.10.4.4. Dependency Management

We have implemented a strict dependency management strategy, encapsulating third-party dependencies such as Ant Design components within our custom components. This abstraction layer ensures that we can adapt to changes in third-party libraries with minimal impact on our codebase. CSS and Ant Design components are utilized with a mobile-first approach, ensuring responsiveness across devices.

5.10.4.5. Documentation

To ensure the maintainability of our codebase, we use comprehensive documentation that details the responsibility of each component and container. The application's dependency tree, as defined by the package.json and locked with package-lock.json, ensures consistent installations across all environments.

Our build process is optimized to use these lock files for faster builds and to prevent accidental updates. Code quality is maintained through disciplined use of package-lock.json for managing dependencies and .gitignore files for excluding non-essential artifacts in our Git repository, maintaining a clean and efficient codebase.

5.10.5. Final Team Deliverables

In Phase 3, we successfully completed the integration of all essential components within the interface, utilizing live AWS services to facilitate seamless communication between remote operations and other teams. Our interface is designed to display comprehensive details about the Scotty Bot and its current mission. It features an intuitive control mechanism that allows for easy manual operation of the robot's movements via keyboard inputs. Additionally, the interface is structured to support straightforward extensions for simple data analysis and system adjustments through its existing pages and components, ensuring versatile adaptability for various purposes.

During the demonstration, the interface promptly displayed the current user's name and ID upon swiping a student card. Once the mission commenced, the Scotty Bot initially operated on autopilot. If the congestion sensor detected a crowded environment, it triggered a signal that was captured by our system, enabling us to illuminate a status light to indicate the need for manual remote control intervention. Operators could then manually navigate the robot using keyboard controls while monitoring through live streaming. Upon reaching the destination, pressing the 'Release'

button allowed the robot to either resume autonomous control or continue its programmed tasks.

6. Conclusions

6.1. Requirement Feature Table

		Requirement Fulfilled by							
		Chassis/Hardware Design	Infrastructure (HW & SW)	Perceptions	Planning & Controls	HCI Interface	Comms and Data Systems	Remote Operations	HCI Intelligence
Navigation	Navigate across crowds				X				
	Plan routes				X				
	Avoid obstacles and people		X	X	X				
	Traffic light control		X						
User interaction	Scan IDs and authenticate						X		
	Interact through voice					X			X
	Interact through web app					X			X
Mobility	Carry loads	X							
	Manual override							X	
	Move autonomously	X		X	X				
	Able to open doors, elevators		X						
Communication	On robot data can be sent off robot						X		
	Off robot data can be sent to the robot						X		

6.2. Summary of Key Design Issues

6.2.1. Hardware Integration

In terms of electronic hardware, we had to ensure that all communication channels were well-defined and that the messages being sent and received followed a predefined protocol. This will make sure the messages are successfully decoded into useful information to actuate the correct functions. For example, the GPIO pin input/ output interface requires tight integration between teams to generate the correct behavior. The hardware team initially had some trouble interfacing with the planning and control operations, but worked out an efficient and reliable protocol eventually.

Although the current physical structure of ScottyBot stands up to the functionality requirements and has quite some potential in terms of capacity and movability, we have space for improvements if the project were to move forward. For example, the screen needs to be higher for more practical human interactions, which calls for the total height of the robot to be raised. For the demo, we did not implement a storage space on the chassis due to supply and space constraints, although it is highly viable and easy solvable in further development as we have the payload capacity and overall space on ScottyBot.

6.2.2. Sensor Fusion and Perception

The Perception team uses 4 cameras to perform all of its responsibilities. The fusion of two front-facing cameras in the ZED 2i sensor allows us to generate depth maps in which we can identify obstacles and output the relevant obstacle information to Planning & Controls. Using another 2 cameras from the rear-facing ZED 2i allows us to perform user tracking. This ensures that we don’t lose track of the user that is following ScottyBot.

Originally, we planned to use the LiDAR in here. For the LiDAR, we experienced issues with setting up Adaptive Monte Carlo Localization (AMCL). This is partly due to the relatively new ROS2 navigation package lacking documentation. We’ve managed to set up the map server and output the appropriate odometry messages, but we’re still figuring out how to properly configure transform frames and integrate everything. In the end, these problems were superceded by the issue of using LiDAR on glass walls and windows.

6.2.3. Software Complexity

Another key design challenge was adapting the logic of the app to better reflect our design priorities. The process involved synchronizing information between the mobile app and the robot's screen as well as providing easy access for users to interact with the robot through a variety of methods (e.g., code scanning, touch screen, and ID card verification). In particular, we focused on providing a seamless service experience, including a real-time display of the robot's usage status and voice interaction features,

the latter of which enables users to control the robot via voice and provide navigational assistance for the visually impaired. In addition, key decisions during the design process, such as the replacement of the Google Maps API to better accommodate indoor navigation and where to mount electronic components, aimed to optimize user interactions and functionality. These efforts reflect a commitment to providing an integrated, multi-dimensional HCI platform that focuses not only on intuitive operation but also inclusivity to encompass users with various needs.

6.2.4. Cross Team Communication

One challenge we faced was establishing an intuitive way to store MQTT messages in the database. Since the database is not a device like Remote Op's laptop or Infrastructure Hardware's sensors, it cannot connect to the IoT network in the same way. To solve this issue, we used IoT rules and AWS lambda to interact with the database. An IoT rule would be created so that whenever a message is published to a specific topic, a specified AWS lambda function would trigger with the message as an argument to the function. The lambda function would then process the message as necessary and store the data in the database.

Another challenge we faced was establishing AWS Iot Communications with Remote Ops. This was because Remote Ops only worked with React and Javascript, and our AWS topics were all done in Python by this point. We had to do some research into how to transmit signals from JavaScript to an AWS topic. We decided to register the Remote Ops Laptop device through Node.js to solve this problem. Additionally, we had to configure endpoints and subscriber/publisher functions on a team-by-team basis. Luckily, the Remote Ops team also helped with fixing this issue and we were able to solve it within two days.

7. Design Process

7.1. Summary of Work Log Hours

7.1.1. Robot Hardware

Team Member	Role	Contribution
Akshat Sahay	Presenter	Designed, ordered and assembled motor control board, helped integrate the board with the chassis and our other electrical hardware, helped debug integration challenges with the motor control software when working with Planning and Controls, assisted during full system testing for the last week or so of the project.
Akshunna Vaishnav	Leader	Coordinated inter-team interfaces on the robot, added safety electrical components to the robot, tested planning and controls interfaces, verified end-to-end Hardware components, developed demo plan w/ other leaders, helped dry runs and demo runs go smoothly by being available as a debug resource, responsible for cable management and chassis layout organization, delegated SSH task to Data Systems leader, resolved key inter-team issues such as remote ops camera placement and HRI interfaces layout and screen placement on robot. Performed the demo with my peers to showcase key functionality to Honda, answered any questions they had after the presentation.
Joey Wildman		I implemented the control loop code for the RP2040, being directly responsible for all of the lower-level firmware to drive the motors, the cliff sensor, and the distance sensor. I also implemented the UART interface last phase and debugged communication between the RP2040 and a Python script I wrote which sent commands to the RP2040, which was eventually used in the demo by Remote Ops. I also helped Planning and Controls integrate their planning stack with our microcontroller so they could drive the robot as well. I assisted with final hardware development and assembly of various

		components on the robot platform and the PCB.
Yuk Ian Khor		Worked with members from HRI and Perception teams on the placements of the screen/sensors and other peripherals. Worked with teammates on the construction of the physical chassis and placement of different components, also worked with teammates to ensure everything worked on the chassis. Collaborated with TA William to design and print out the screen holder for it to be secured onto the chassis.
Zhonghui Cui	Editor	I worked on finalizing the design as well as the construction of the robotic chassis for ScottyBot, ensuring it was ready for the integration of key components and the demo. I also took charge of mounting important hardware, including power electronics such as batteries and PCB, sensors, and human-robot interaction (HRI) components like screens and speakers. I was actively involved in preparing for the demonstration, taking notice of how to debug the hardware to ensure everything functioned smoothly during the demo. Additionally, I headed the outlining, writing, and compilation of report for sections of our team.

7.1.2. Perception

Team Member	Role	Contribution
Alexis Duong	Editor	For this phase, I collaborated with Christian and used the Python API to create an interface with the ZED ROS2 wrapper. We scanned multiple areas on campus and tested the accuracy of the ZED to produce precise pose estimations over the course of several sessions. After finding that the ZED output area file was not compatible with the ROS2 wrapper, we instead pipelining the computations directly to a topic instead. In the final stages of the robot I acted as technical

		support to other subteams, staying for most of the sessions where integration took place. I was also Editor for this phase so I took lead in the discussions for what is necessary in the final report.
Christian Luu	Leader	For phase 3, Alexis and I worked on porting the code we wrote to interface with the ZED ROS Wrapper to using the Python API. This is because during testing, we found that the map files that the Python API generated were incompatible with the ROS Wrapper. This meant a rewrite was required. I also, with the rest of the team, performed a lot of testing on our perception stack with a mock robot using a push cart. Then during the last two weeks, my role switched over to providing technical support for other teams.
Patrick (Penggao) Li	Presenter	For this phase, I worked with Sirui on implementing the user tracking function using ROS service so that the robot an look / unlook on a specific user when they scan their id card. Along with all other teammates, we used ZED camera to mapped the testing area and generate map files for localization. I collaborated with planning and control on integrating perception stack with their path finding functions. I also collaborated with all other members from other groups to prepare for the final demo. I offered technical support to other teams on what commands we need to call and how to use our perception stack.
Tony (Sirui) Qi	Presenter	In this phase, I was working on doing the localization with all of the teammates with the ZED camera and the push cart. Except these, Patrick and I was also working on implementing the user tracking function with the ROS service which can let the tracking system following the people who we want (like locked on a people with scanned id). Also, I was offering the technical support to the planning and control team to support their collaboration with us (like bug fixing) and let them know how to use the things which we

		offering to them like data and API on route planing. Finally I was also offering help on robot testing for the final demo

7.1.3. Planning and Controls

Team Member	Role	Contribution
Evelyn Bang	Editor	In this phase, I mainly focused on the logic of the algorithm. Some tweaks needed to be made in order to integrate Dijkstra and A* into the code. For example, the robot was scraping the wall, because it was not accounting for its width during automation testing. In order to combat that, we took the width of the robot and counted how many dots on our chart it would take up in order to have enough space to move freely. One challenge was that there was a very narrow hallway that pushed us to create more checkpoints (to combat high calculation load) and increase the rendering (to combat guaranteed scraping) for the dot chart. I also worked together with Qing and Ruiyi to make sure that the processing power wasn't overloading the bot in order to make sure there weren't any pauses in between processing checkpoints. In order to do that, I made sure the point chart had a higher rendering, avoided edge points, and created more checkpoints.
Nicholas Beach	Presenter	For this phase, I started by writing a python script that subscribes to relevant ROS2 topics published by the perception team needed to run through our pipeline. I then discussed integration with the robot hardware team for sending commands and communications team to integrate with cloud infrastructure. I also tested the robot and fine-tuned our planning algorithm once the full robot was assembled. This included aligning the perception teams map with our 2D map to ensure proper path planning, adding waypoints to maps, and adjusting hyper parameters after test runs. After testing, I

		helped debug a precision issue which then caused a speed issue and then ran more tests. Lastly, I communicated with the presenters on how the presentation should look, created slides for the team, and presented during the final demo day.
Saadhikha Suresh	Leader	I collaborated with the group to discuss integration tasks and align it with the updated demo scenario. I finalized cross-team requirements pertaining to planning and controls during the leader's meetings. Worked on researching dynamic map generation, conducting code reviews, and fine-tuning algorithm parameters. I focused on adjusting hardcoded coordinates on the map and optimizing the algorithm to ensure a smoother path. Additionally, I worked on programming and debugging plans, integrating with other subteams, testing simulations and resolving errors from porting over our code on the actual hardware.
Qinman Wu	Editor	Convert grayscale png in cabot to occupancy grid. Hard code more nodes for the large scale map, and make them consistent with actual coordinates. Use the data from grayscale png to build the small scale map instead of just creating a rectangular area. Finish the update logic in PlanControlModule. Refine the __calculate_wheel_speeds in control part to make the output of motor speed in range (-wheel_speed_scale wheel_speed_scale) based on angular speed and linear speed. Write a ROS2 subscriber to subscribe to perception. Encapsulate and organize all code, and write a demo main script. Put code into the full demo. Robust test with 50% noise when updating attitude in simulation. Add function of set end point by (x,y) coordinates. Handle some edge situations. Add error handling. Integrate hardware code to publish wheel speed to hardware. Work on the integration between Perception and Plan. Work with perception to test the plan module with the perception module on. Change subscriber to successfully subscribe robot attitude from perception. Add more nodes to the largeScale map. (finished hardcoding whole cabot 2d map)

		Test the plan module on the route, change the initial robot attitude parameter to align the map perception use with the map we use. Double the map resolution to make the path plan more accurate. Since this will make the calculation slower, change to a* algorithm to compensate. In order to prevent hitting the wall, change the logic of picking effective neighbors so that the possible neighbors and the wall must be separated by one node. Add real-time plotting for debug. Set placeholder for suspend/resume planControlModule for communication group's requirement. Store real-time plot/real-time robot attitude as png/json/pkl file to meet HRI interface's requirement. Finetune parameters, completing the route for the first time. Refine the planning algorithm to make it faster (instead of recalculating the path every time, determine whether it has deviated the path first, only recalculate when needed.) add a thread to the subscriber so that the spin of subscriber doesn't block the main thread.
Ruiyi Lian		In this phase, I was involved in discussions and negotiations with the robot hardware group about the format of the data, and ultimately decided to still control the robot's traveling and steering by outputting different speeds for the left and right motors. I also updated the map, as at first we had set half a meter from the wall as a no-travel zone, but we were using different maps than the perception group, so we needed to keep the starting point the same as theirs during the debugging process. Their starting point is against the wall. So I updated the map near the start point. Also during the debugging process, Qin and I found and debugged with the other groups a problem with map mirroring and differences in the angles of the two maps. Since perception's map could not be visualized, we mapped the robot's positioning in order to debug the offset between the two maps, and eventually made the two maps almost the same after several debugging sessions. I assisted in choosing the right speed for the

		robot to travel. Code on bypassing obstacles was written by me, but due to the fact that pedestrians can cause problems with localization. And also because obstacle detection is passing us the closest point of the obstacle to the camera, which requires recalculating the path at all times and can slow down the cart considerably, this feature was not implemented in the end.

7.1.4. Human-Robot Interaction Intelligence

Team Member	Role	Contribution
Shenai Chan	Presenter	Created GPT feature extractions and improved it to integrate into the pipeline. Integrated the text to speech, feature extraction, and speech to text into a pipeline. Integrated Hey ScottyBot code into Interfaces' react code Communicated with multiple teams for Integration process Successfully finished presentation, with diagrams to trace the system from user input to destination output. Helped lead organization of the final presentation narrative. Fixed the voice input screen by removing the 15 second recording feature. Set up personal laptop as a Thing in IOT network for dummy testing. Connected subsystem to MQTT network on the topics of destination, user info, and RFID card scanning. Brought up website on the NUC and debugged cross-platform issues. Set up websockets to allow for bidirectional communication and debugged cross-origin platform issues between client and server. Set timeout for next page trigger on receipt of the destination. Published destination to planning's topic, manipulated the HTML to show user destination, modified TTS code to speak over speaker. Was generally present for debugging each

		day in the final push.
Vivian Zhou	Editor	Created Text to Speech using Azure Speech Services and generated mp3 file from the input messages. Created NER model using Azure AI Language Services and was able to output entity text, entity category, confidence score and other evaluation matrix. Helped integrating Hey ScottyBot into Interfaces' code and added the printing functions back. Modified HTML outputs Communicated with the demo leader and other sub teams in the integration process and for the full demo details.
Yudi Chen	Leader	Created Speech to text using Whisper model and generated corresponding text from mp3 voice in. Created Hey ScottyBot feature with voice command in js and was integrated into the backend part Participated in Leader's meeting and completed leader's meeting's note.

7.1.5. Human-Robot Interaction Interfaces

Team Member	Role	Contribution
Yuxin Deng	Presenter	I successfully distributed and managed a broad spectrum of tasks, ensuring seamless integration and collaboration across multiple subteams and technological domains. I implemented voice functionalities on screens and diligently coordinated with subteams to prepare for comprehensive demos. My efforts included integrating these components with the work of the intelligence group and planning control group, culminating in an end-to-end demo that I presented. I implemented the route abstract and navigation features on the screen. Additionally, I focused on the frontend by researching and implementing voice recording and recognizing , which integrated with the backend. I tackled deployment

		challenges on Linux, optimizing system compatibility and functionality. Throughout this phase, I continuously engaged with various teams to monitor progress and integrate their work, keeping the project aligned with its objectives. This proactive approach not only addressed immediate technical challenges but also set the stage for future enhancements and testing, ensuring the project's ongoing success and adaptability.
Sitong Shang		In the integration phase, a large amount of work is in cross-team collaboration by facilitating communication alignment across 3-4 teams, ensuring clarity in requirements and communication protocols. I also spent a lot of time learning and researching new tools like websockets and MQTT. As for implementation, I refactored code structure to unify HTTP routes for mobile and screen interfaces. I also continued on debugging existing code and explored frameworks such as Bootstrap to boost our mobile interface's responsiveness for improving user experience across devices.
Yichi Zhang	Editor	In this phase, to enrich the robot's functionality, I added functional interfaces to the robot's screen interface as needed, such as the robot movement status control interface. Additionally, I adjusted the layout of the robot's screen interface to ensure proper display of each interface. Furthermore, I participated in testing various features of the robot interface, and corrected bugs in the voice input function. I also took part in resolving issues related to data transmission between the frontend and backend. As an editor, I promptly recorded team meetings, participated in editing meetings, and completed most of the work on the Phase 3 team report.
Mingze He		I tackled front-end and back-end integration challenges, notably in voice recognition and data transmission, and streamlined the robot's

		control interface for simplicity and efficiency. My contributions in interface design and functionality testing, including refining voice inputs and updating map routes, played a crucial role in aligning the project with its goals and preparing for future enhancements.
Yongzhi Gao	Leader	As a leader, I participated in leaders' meetings, oversaw the progress of the project, coordinated with other teams during the integration phase, and managed the methods and content of integration. Additionally, I was responsible for establishing the backend Flask framework, writing the HTTP communication protocol between the frontend and backend, and collaborating with the HRI Intelligence team to develop the WebSocket communication protocol. I also handled the deployment of the voice module and the integration of the MQTT protocol. Throughout the project, I participated in the testing and debugging phases, adjusting the interface logic including the map routes update and suggestions feedback based on test results to achieve a reasonably effective presentation.

7.1.6. Communications

Team Member	Role	Contribution
Adrian Kam	Leader	As the demo master for the entire class, created the script for the demo that was presented to Honda. Oversaw the integration of the robot by talking with representatives from every subteam, getting updates, encouraging subteams to work with each other, answering their questions regarding what needs to be done for the demo. Monitored the progress of the project and was the decision-maker on what to include/delete in order to meet the project deadline. Oversaw testing of the robot to ensure it was capable of following the demo script and it was robust. As the communication subteam leader, ensured external systems

		(infrastructure, remote-ops, database) were ready to go and capable of interacting with the robot.
Lexi Batrachenko	Editor	Completed the Phase 3 report for the Communications subteam by rearranging the sections and detailing the updates, esp for the demo_main.py script. Kept track of big picture of the demo. Helped develop the IoT MQTT interface and communicated with several on-robot teams regarding data pub/sub. Set up NUC computer as IoT device on AWS IoT Core, downloaded the necessary files to represent it as a device_client object in the iot_interface and helped install the necessary libraries to enable MQTT and NFC card reading capabilities (briefly tested it). Worked with HRI to first test the flask webapp with IoT Core and MQTT on a laptop and MQTT test client; next, helped integrate the webapp onto the NUC and explain callback functions for HRI <-> P&C with NFC card reading and spoken destination receiving. Created annotated version of the world map for displaying to the user, included start and end points for better user experience. Helped manage central documentation for rapidly expanding list of MQTT topics, including JSON format for each, as well as the teams publishing and subscribing on them.
Sanjana Shriram		Connected infrastructure code to the central AWS IoT Core and fully integrated all 4 infrastructure functionalities. Validated congestion monitoring, door opening, and debugged both elevator functions and became familiar with infrastructure setup for the end to end demo. Demonstrated a fully working pipeline from a simulated planning and controls to communications to infrastructure software to infrastructure hardware. The door actuation button lit up when the MQTT message "on" was received. Completed the end-to-end demo with a successful demonstration of all parts of the

		infrastructure! In particular, I led the timing of the congestion monitor switching over control from Planning & Controls to Remote Ops.
Rohan Raavi		Connected and registered the Remote ops team into the AWS IoT Client Devices group. Also aided setting up and achieving video streaming to the Remote Ops. We first tried with peerjs, but this approach was eventually abandoned due to networking issues. Because of this we switched to installing Zoom on the NUC and used zoom hosting for live streaming the robot's camera. Finalized and helped debugging integration of HRII Interface. Specifically, the LLM module sending voice commands over the AWS topics to the NUC. Lastly, I helped with overall communication integration across all of the subteams
Brendan Wilhelm	Presenter	Integrated remote ops and other subteams into the AWS IoT MQTT interface, and helped to develop and debug the remote ops video streaming peerjs script which did not appear in the demo due to network permissions issues out of our control. Also met with presenters from other teams, designed the communications portion of final presentation, and presented this portion of the presentation to the Honda investors. Also coordinated with other subteams to smoothly integrate them into the communications infrastructure.

7.1.7. Data Systems

Team Member	Role	Contribution
Connie Liu		Developed and integrated AWS Lambda with MongoDB, enhancing data processing efficiency sourced from MQTT/IoT Core. Studied AWS API gateway usage and assisted in figuring out the solution for the usage of other teams. In close collaboration with the communication team, I ensured the smooth execution of our project. Additionally, I assisted in preparing our final presentation

		and demo, highlighting the collaborative efforts that propelled our project's progress.
Bowen Han		Cooperating with the communication team and the remote control team. Programming for the lambda function to store and fetch data and try to communicate data with the remote operations team. Test connection with the remote control team. Contributed to the final documentation and presentation preparations. Make the final demo for the data system presentation.
Tianyi Ren	Editor	Collaborating closely with both the communication and remote operations and control teams; Crafted AWS Lambda functions to seamlessly receive data from the communication team; ensured efficient data handling by debugging code, testing connections, and refining lambda functions for optimal performance; culminating in a successful end-to-end demonstration.
Yihan Wang		Developed and optimized AWS Lambda functions, capitalizing on the serverless architecture to focus more on coding and less on maintenance. This approach enhanced efficiency and scalability, allowing for seamless data processing between teams. By integrating Lambda with MongoDB, it can ensure rapid data access and address connectivity challenges faced by the remote operations team. This work not only deepened my expertise in AWS services but also underscored the cost-effectiveness and agility of serverless computing. Contributed to the final documentation and presentation preparations.
Yao Xu	Presenter	Developed and integrated AWS Lambda with MongoDB to achieve automatic database operation triggering with MQTT/IoT Core. Communicate with other teams to understand the issues and difficulties they encounter while using the lambda functions we provided. Assist them in debugging their code that

		interacts with the lambda functions, such as the remote control team. Prepare and be responsible for the presentation in the third phase. Participate in the preparation work for the demo.
Takshsheel Goswami	Leader	As leader of phase 3, my main tasks were to ensure a clean transition into the final building stages. It started off with the end-to-end demo, where I worked along with my team to set up lambda functions for the communications team, which we end up using in the final deliverable scottybot. I also attended leaders' meetings to report on Data Systems' progress and assist with demo planning/mapping and integration tasks. I was involved in a lot of discussion with Planning and Controls and Remote Operations pertaining their data requirements and pathways, as well as assisting remote ops program with lambda functions to receive data, moving from API gateway to AWS SDK. After this, in the last 2 weeks, the data systems team assisted with integration tasks, and I personally was involved in getting openSSH to work on the NUC so the whole team could upload code/debug the NUC remotely. After this, I oversaw the demo pertaining to the collaboration of our team along with Communications, Remote Ops, as well as Planning and Controls. In the last week of class I assisted with integration tasks with HRI/Remote Ops and Perception system verification. Also did work for a little bit on the postgreSQL but didn't take it to completion given lack of time at the final weeks. This was followed by me assisting with the final demo and presentation.

7.1.8. Infrastructure Hardware

Team Member	Role	Contribution
Navod Jayawardhane	Member	Assisted in the push for final integration and coordinated with team members, Infrastructure Software, and Communications

		for the final demo. Began the phase by producing a working implementation for the end-to-end demo, and by developing the second revision of PCBs and completing the ordering process. Manufactured five copies of the control board followed by the new inside elevator PCB. Performed hardware testing on all systems. Developed the CAD models of the demo mockup panels and schematics for mockup PCBs. Built and integrated the electronics for all three paneled mockups and wrote firmware for the demo. Did final integration with the mockups and our custom PCBs and worked with infrastructure software to test the whole system. Coordinated with the final demo efforts to highlight our work during the live demonstration, and wrote technical details about our systems for the final presentation.
Joey Mok	Editor	Designed and prototyped the enclosure for the congestion monitor system. 3D printed and laser cut all the structures required for the final mockups of the actuators and monitor. Created revisions of hardware & re-manufactured items when required, and collaborated with the software team for full system integration. Worked with other editors to finalize and complete the Phase 3 report.
Devan Grover	Leader	Worked with other leaders in coordinating demo efforts, and helped develop the demos for all the PCBs that were integrated into the mockups. Reflowed and manufactured the extra PCBs for demos. Helped with final integration with the infrastructure software team and achieved the ability to flash our control boards with our own custom software.
Julian Rodriguez	Presenter	Tested and attempted code for I2C communication with on-board IMU's for the inside elevator control panel. Cut and mounted protoboards to the elevator and door actuator mockups. Reflowed a spare set of 2nd revision boards to be displayed as a part of our presentation outside of the mockups.

		Wrote and delivered an overview of Infrastructure Hardware's role in Phase 3 integration with other teams.

7.1.9. Infrastructure Software

Team Member	Role	Contribution
Bangjie Xue	Presenter	Working on integration with other teams for final testing, especially the infrastructure hardware, communications, and planning and controls. Finalizing the 4 softwares for the infrastructures, uploading the final version to Github and adding a detailed ReadMe file for other teams to reference. Helped on several integration work sessions to solve issues live as they come up. Helped on the final demo, worked on the presentation slides, and presented in class for the final presentation.
Yufei Liu	Member	1. Edited the report and presentation slides. 2. Set up the Arduino IDE environment and wrote the "Hello World" demo. 3. Collaborated with other teams to identify responsibilities. 4. Worked on AWS and ESP32 and attempted to connect the ESP32 to the AWS IoT framework to send and receive messages. 5. Developed and tested JSON-based communication protocols to enhance data transmission between the ESP32 microcontroller and AWS IoT Core, ensuring robust and secure message handling for real-time operations. 6. Assisted in preparing detailed documentation and system diagrams to support ongoing maintenance and future scalability of the infrastructure software components. 7. Participated in comprehensive debugging sessions to optimize the system's performance, focusing on reducing latency and improving the reliability of network connections.

Wenxin Shi	Leader	In phase 3: 1. Finalized the design and implementation of the message format that meets the demand of communication 2. Held team meetings to assign tasks and used Slack to follow up with timelines and assign work. 3. Coordinated with communication team, plan and control team, as well as infrastructure hardware team about task dependency 4. Reported to TAs and professors the progress and tasks of Infrastructure Software team 5. Actively participated in the integration test and demo. 6. Completed the report.
Amy Zhao	Member	Wrote the simulated elevator control ESP32 code for elevator calling and floor selection functionalities, collaborated with teammates to integrate with other teams. Communicated with the communications team to finalize the API and data type. Worked on the software architecture design diagram. Worked on our team's section on the final report.
Zilong Zhou	Member	Investigating Amazon Lambda, LoT core. Setting up a development environment and setting up Hello World Demo. Incorporating ESP32 with LoT core and Lambda for multiple purposes. Finalized the design and implementation of the message format that meets the demand of communication

Krish Sethi	Editor	Created an API which can be used to send sensor data from esp32 to AWS IoT core and receive actuation commands Presented a Hello world demo. The demo use the API to send DHT sensor data to the cloud and received command from the cloud to control an LED Used the API to send Congestion monitoring data to the cloud Worked on Phase 2 presentation, recorded demo for sending Congestion monitoring data to the cloud Worked with infrastructure hardware and communications team for integrating our API. Coding and writing the 4 final modules the infrastructures, uploading the final version to Github and adding a detailed ReadMe file for other teams to reference. I also effectively communicated the our teams status and key findings in the report.

7.1.10. Remote Operations & Control

Team Member	Role	Contribution
Zhiyuam Chen		During this phase, I actively participated in testing and integrations at Tech Spark, ensuring seamless progress. I collaborated with various teams, addressing their inquiries and facilitating smooth communication. Additionally, I assisted team members in testing streaming functionality under school wifi restrictions and engaged with data system groups multiple times to implement data fetching procedures. Moreover, I contributed to testing manual controls to ensure their functionality ahead of the final demo.
Chi Hsiang Lo	Presentor	In phase 2, I presented the hello-world-demo and phase-2-presentation. In the website, I developed the page responsible for live streaming, transitioning from an initial computer video input to a live video input via

		the computer's webcam. Additionally, I explored how to use a Logitech camera as a video input source. This ensures that our website can display what the robot sees, offering a live video feed for manual monitoring and intervention during critical moments. In phase 3, I was deeply involved in various stages, from planning and testing to the final execution. I led the implementation of a crucial control system toggle between manual and automatic modes and was integral in setting up live streaming capabilities. Collaborating closely with the team leader and other members, I contributed significantly to resolving issues and ensuring all map and location data were seamlessly integrated. My role extended to presenting at the final demo, where I effectively communicated our project's achievements and also crafted the final presentation slides, culminating in a successful demonstration of our technological capabilities.
Tanvi Mahatme		During this phase, I played a key role in coordinating with other teams to ensure our project progressed smoothly. I set up the server on my machine, to help other team members with any questions about the application's interface. This helped our team work more efficiently and respond quickly to any issues. I also participated actively in the testing sessions before our final demonstration. I focused on carefully checking our systems to find and fix any issues before they could become problems. This careful testing helped make sure our application was ready and worked well during the demonstration.
Suhan Lu	Leader	In this phase, as the team leader, I actively participated in all leaders meetings to ensure

		seamless cross-team communication and collaboration. I was responsible for developing the backend server, integrating it with the communications team's AWS IoT using MQTT protocols, and interfacing with the data system team's AWS Lambda functions. I successfully implemented and tested live streaming under restricted conditions. Additionally, I attended every testing and integration session prior to the final demo to address any issues proactively, ensuring a flawless delivery of both the interface and the demo. My leadership and hands-on involvement were instrumental in driving the project towards its successful completion.
Abuduwaili Yierpan	Editor	In this phase, I actively participated in final demo meetings to ensure the quality and consistency of our project. I collaborated closely with the team leader and other team members, providing valuable insights and suggestions for improvement. I also attended many testing and integration sessions prior to the final demo to gain a comprehensive understanding of the project's progress and help fix any potential issues. This helped me effectively communicate the project's status and key findings in the report.

7.2. Task Dependencies

7.2.1. Robot Hardware

The Robot Hardware team is responsible for developing the physical platform to house all the subsystems and components, as well as implementing the navigating and mobility functionalities that have been envisioned. To ensure an efficient streamlined workflow, we try to avoid direct dependencies on the higher-level architectures but only rely on the control data from a few teams. We will require the motor control instructions provided by the Planning & Controls team for the microcontroller to execute the movements needed, and will pass back the 1D cliff sensors and distance sensor data to the Perceptions team for further data processing. In addition, We will directly interact with the HRI Interface team to control the LED lights, and transmit battery level to the Remote Operations team in the final integration stage.

7.2.2. Perception

The Perception subteam is mostly working on the items detection with the localization. Mostly we just want to communicate with the HRI Interface team to add a button which can lock and unlock onto that user for tracking which can help us to do this. Additionally, we also need to send the streaming from the front camera and the location of the robot to the HRI Interface team to let them show more information onto the UI design. Also we need the Robot Hardware team to support the power onto the cameras. In addition, we also need the Robot Hardware team to help us put the camera onto the robot to make this work.

7.2.3. Planning and Controls

The Planning and Controls subteam is in charge of the reading of collected data by the Perception subteam and running algorithms and pipelining the resulting data to the Hardware team to execute certain movements and logistics. We do a lot of interfacing with the Perception and Hardware team as well as the HRI Intelligence team in order to know the end points for our planning. Particularly, the Planning and Controls team worked on the algorithm for automated movement to allow movement from one location to another via checkpoints and shortest distance calculations within a small scale and large scale map.

7.2.4. Human-Robot Interaction Intelligence

The main dependency HRI Intelligence team will be doing is to collaborate with the HRI Interfaces, Planning and Controls team. We will be providing backend support including serving routes and view, enabling login features, connecting to the database and providing “Hey ScottyBot” voice starting commands. This requires the data system’s user table, and Planning and Control real-time localization and map data. In addition, we used MQTT to instantiate objects of classes with appropriate parameters and then

call the provided methods to manage MQTT connections and subscriptions. This is used to communicate and send json data from backend to P&C as well as reading RFID card login data. Once we received this information, we integrated that into the Interfaces team.

7.2.5. Human-Robot Interaction Interfaces

As the HRI Interfaces team, our dependencies on other teams are critically intertwined. Specifically, in implementing the voice interaction functionality, we rely on the HRI Intelligence team to process the voice data on the back end. This requires our frontend system to accurately capture and transmit the user's voice inputs, while the backend needs to process these inputs swiftly and accurately to provide feedback.

In terms of the visualization of the robot's route planning, we depend on the data provided by the Planning and Control team. They are responsible for calculating and updating the movement points of the robot, and our task is to accurately visualize these points on the map within the user interface. This not only demands that our system is capable of receiving and processing data in real time but also ensures that the data visualization is both precise and readable.

Moreover, our work is highly dependent on the Hardware team. The performance of the hardware directly impacts the responsiveness and stability of our interface. We maintain frequent communication with the hardware team to ensure that all devices, such as screens, microphones, and speakers, operate stably under various conditions to support complex human-machine interaction tasks.

Through close collaboration with these teams, we not only ensure the smooth completion of tasks but also foster the exchange of knowledge and technology, enhancing the cohesion and innovative capacity of the entire project team.

7.2.6. Communications

The Communications subteam's main responsibility is to ensure reliable wireless communication between all related devices on and off the robot (“devices” includes both software and hardware developed by subteams).

Our role also extends to guaranteeing MQTT publish/subscribe network communication among teams. We closely collaborated with P&C, Infrastructure teams and HRI to establish internet connections, connect devices to the AWS IoT Core system, manage the transmission of time-sensitive data, authenticate users via an NFC ID card reader, and facilitate real-time video streaming to the Remote Operations team. We were responsible for creating MQTT topics and being consistent in how devices, and specifically subteams, publish and subscribe to them.

We ended up needing to understand the big picture of the project, which meant learning what teams needed us for and what they did independently. Since the were

responsible for integrating code, we needed to understand what code we needed to add and what code was relevant to our role.

7.2.7. Data Systems

The database system plays a vital role in driving data collection efforts and establishing a standardized data format across teams. Additionally, the system must be flexible enough to accommodate diverse data storage needs, including the creation of collections in MongoDB and handling complex data manipulations. In scenarios where data originates from additional platforms beyond MQTT, the implementation of corresponding event listeners within Lambda functions becomes necessary. Similarly, for evolving data processing requirements, the addition of corresponding Lambda functions is imperative to ensure efficient data handling. [FIX THIS]

7.2.8. Infrastructure Hardware

The Infrastructure Hardware team is responsible for designing the boards that the Infrastructure Software team will then interface with, and utilize to transmit messages to the other software teams. As such, the Infrastructure Hardware team itself does not have many dependencies besides the direct connection with the Infrastructure Software steam. As long as all our ESP32 microcontrollers are capable of transmitting data through wifi modules, we leave the communication to other subteams to implement and integrate.

7.2.9. Infrastructure Software

The Infrastructure Software team plays a pivotal role in the seamless integration and operation of various automated systems within the facility. This team's efforts are intricately coordinated with those of the Infrastructure Hardware Team, Planning Controls Team, and the Data Systems Team, ensuring a cohesive approach to infrastructure management and optimization.

For the implementation of automatic door controls, the process begins when our team receives a request signal from the Planning and Controls Team. This signal explicitly specifies which door is to be opened. The communication is facilitated through a sophisticated Application Programming Interface (API) designed and maintained by our team. This API acts as a conduit for requests, enabling us to efficiently process them and trigger the corresponding door's opening mechanism. This same protocol is adeptly adapted for the control of elevator systems, ensuring a unified and streamlined approach to managing different facets of the building's infrastructure.

Pedestrian traffic monitoring represents another critical component of our responsibilities. Utilizing advanced ultrasonic sensors, we are tasked with the accurate and continuous monitoring of pedestrian flow at strategic locations throughout the facility. These sensors are capable of detecting and measuring the proximity and

density of individuals, allowing us to gather substantial data on traffic patterns and volumes. The collected data is then transmitted to the cloud, where it is securely stored within a database. This database is meticulously managed by the Data Systems Team, ensuring the availability and integrity of the data for analysis and decision-making processes.

Moreover, our collaboration with the Infrastructure Hardware Team is essential for ensuring that the software solutions we develop are perfectly aligned with the physical hardware components they oversee. This collaboration encompasses the integration of our software code with various microcontrollers and hardware devices, laying the foundation for a robust and reliable infrastructure system. Our mutual efforts are focused on achieving optimal compatibility and performance, ensuring that both software and hardware components work in unison to support the facility's automated systems.

By working closely with these teams, the Infrastructure Software team not only contributes to the operational efficiency and safety of the facility but also paves the way for innovative solutions that enhance the overall experience of those within the building. Through our concerted efforts, we aim to establish a state-of-the-art infrastructure that is both responsive and resilient, capable of adapting to the evolving needs of the facility and its occupants.

7.2.10. Remote Operations & Control

The Remote Operations and Control team plays a crucial role in guiding the Robot through densely populated areas by sending control signals to our Communications team. These signals are crucial for the Robot's movement and are meticulously forwarded to our Planning and Controls team for precise execution. Furthermore, our team is responsible for streaming the Robot's live video feed, acquired through the Communications team, enabling real-time visual monitoring. Additionally, we oversee monitoring the Robot's operational status, including battery levels, by liaising with our Robot Hardware team. Our team also gathers essential user information, such as names and destinations, from our Communications team, allowing for a personalized and efficient navigation experience. This harmonized approach allows our Remote Operations and Control team to effectively manage the Robot's navigation, ensuring safety, efficiency, and a tailored service to users.

8. Reflection

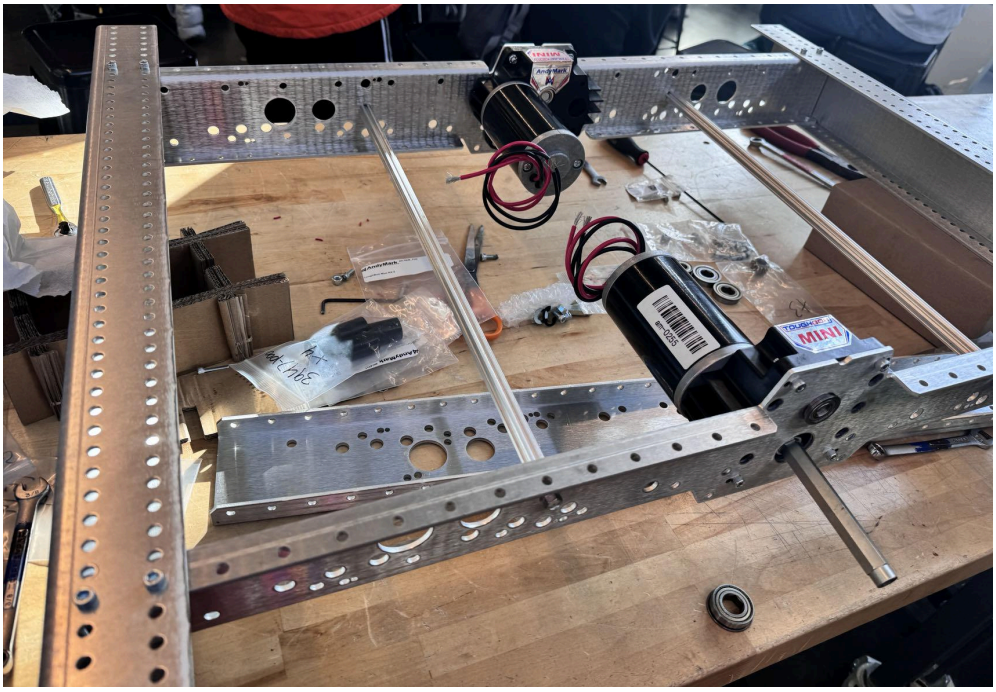
8.1. Robot Hardware

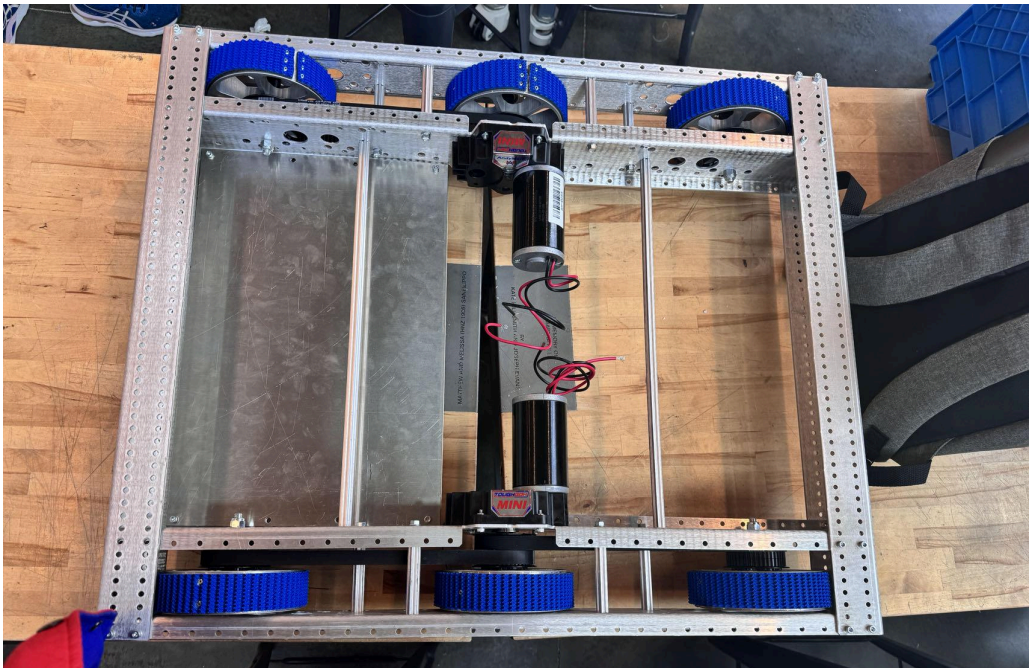
As the Robot Hardware Team, we took on the responsibilities of developing the hardware components and systems of ScottyBot. While there were challenges along the way, the experience has definitely helped with our understanding of robotic development and provided some invaluable memories.

8.1.1. Project Progression and Achievements

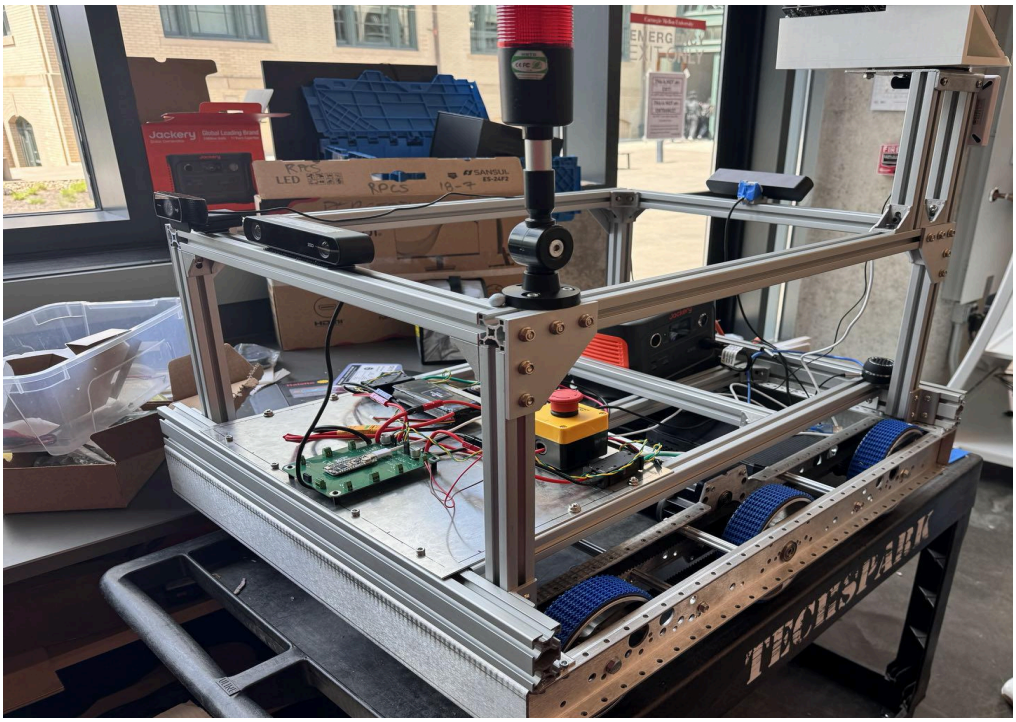
In the initial stage, all students did some brainstorming and defined the visionary scenarios. After the formation and deliberations about plans, our Robot Hardware team began by selecting and surveying different chassis and microcontroller, and other hardware components options. After making some decisions, we got down to designing and construction.

The second phase revolved around the physical construction of ScottyBot and developing its fundamental hardware-related systems. We constructed a robust chassis using SolidWorks CAD to ensure durability and easy access to internal components. Our team also designed for the power and control systems, integrating a microcontroller with the necessary drivers to control motors and sensors.





In the final phase, we integrated the functionalities that allowed ScottyBot to interact with users and navigate through CMU's campus smoothly. This required collaboration across different teams to ensure that the hardware and software components worked smoothly together. At last, we planned and presented the final demonstrations showcasing ScottyBot's ability to interact with users, to provide real-time navigation, and other functions that we have achieved.



8.1.2. Reflections and Improvements

If we were to start over, these are what would have done differently:

We would complete the design and order the PCB earlier, so we could be working on other aspects of our work while waiting for it to arrive in parallel. his would allow two things: first, integration with higher level teams happens earlier if the PCB is on hand (but this also depends on other team being there to make use of the motors in time), and second: if any changes arise (more ports than RP2040 supports so need diff microcontroller, etc.), we can respond to them faster from our end and change PCB and re-order if needed.

We would consider getting a different chassis/pre-built chassis, which would take a lot of the load off from our team having to design every hardware component custom through SOLIDWORKS and then making modifications. Although in this process, we did learn a lot about CAD design and hardware construction as a whole.

We would start out with the portable power supply as the backup to power NUC, for ease of testing, and then build a power system using on-robot batteries if needed. In hindsight, understanding the exact power needs and load capacity for the demo showcase would have led to better estimates for our power system requirements, and to just buy a portable power supply even if only for testing makes runs of components which rely on A/C power much simpler.

We would plan out the payload by doing some testing to get more accurate estimates. Had we known how much current our CIM motors consume, how much weight we need to and could handle, or how large the robot really needs to be in order to accomplish the functionality, which is not as large as we estimated, we would have ordered different power and hardware parts that were more suited to our needs and cut down the cost.

Further, we would emphasize more on safety and power load management from the start. We were lucky that no power related accidents happened despite we did not took the most secure safety precautions during the development and testing, but looking back and moving forward we definitely should add more safety measures, for example a more reliable circuit breaker at the power source.

8.2. Perception

Through phase 3, the most important thing we've learned is how to translate the ideas and designs from the previous phase of the project into a concrete realization and, in the process, identified new problems and gradually improved the previous deficiencies. Technically speaking, we've learned how to utilize the openCV library as well as embedded camera functions to implement our tasks. Specifically, we learned using YOLOv8 for object detection, CSRT for object tracking, how to call ZED's built-in API and integrate them into our system. We also learned how to set up the NUC and link equipment including cameras and LiDAR to it. We also learned that LiDAR does not

work well on glass. We learned to use ROS service to communicate synchronously between different nodes, which is crucial for cooperation with other groups.

Overall, this semester has been invaluable in providing our Perception team with hands-on experience in working on a project with many moving parts. The sheer number of teams and other students all collaborating with each other makes this project become not just a technical one, but also an organizational one. Despite our team being relatively separated from the remaining parts of the robot (as we only really limit our interface with Robot Hardware and Planning and Controls), in the end, we found that being the point of contact for the NUC means that we provide a lot of technical support to other teams. Also, during the integration stage we spent a lot of time watching Scotty Bot take its “first steps” and it was a lot of fun to be with all the other teams in one area where there was constant communication and debugging happening. In retrospect, if we were to redo this semester, technologically, we would forgo the LiDAR completely and request that we have wheel odometry data so that we can run a Kalman filter to provide an even better position estimate.

8.3. Planning and Controls

The Planning and Controls subteam previously invested significant time in researching simulation environments, but acknowledged its disconnection from the overarching project goals. Consequently, we shifted our focus to direct coding efforts, completing the code framework, and increasing interaction with other teams. We learned that it is always a good protocol to prepare as much as possible to get ready for real data and be able to calculate that into usable outputs while making a hardcoded environment that is a good baseline for other teams to reference.

Overall, we learned that it would be important to have different algorithms for the larger map as a whole with long distance checkpoints using Dijkstra’s algorithm, while using A* algorithm for smaller scale local obstacle planning. During these calculations, we found some bugs that we had to fix that helped run the automation smoothly, which successfully allowed the robot to go from one checkpoint to another. If we had more time and resources, we would have spent more time tweaking the code and allowing smoother turns that aligns better to the goal of leading a person to a prompted location. With these smoother turns, the person following the robot would not have to abruptly stop in certain areas to wait on the robot.

8.4. Human-Robot Interaction Intelligence

In this phase, we integrated our developed pipeline of voice in-> GPT feature extraction-> voice out from Microsoft Azure services into HRI Interfaces’ frontend screen and mobile development. This phase is challenging since a lot of our code is incompatible with other teams’ programming language as well as exploring the tools to allow such integrations. We implemented MQTT to allow machine to machine communication. The MQTT and websocket emits allows our code to be able to get the RFID ID Card Scanner data and ensures login is successful. It also allows us to pass

our extracted features including objectives and destination to Planning and Control. That way Planning and Control can use this information to compute the route and we can get the image with the route information to display the result to the users. One of the most important lessons we learned is to communicate with other teams frequently and be in sync with any changes that everyone is implementing.

If we would start over, we would've communicated with HRI Interface regularly because our team is very closely related to each other. We are responsible for providing backend support to them and passing route information. Since the majority of Phase 1 and Phase 2 is more individual group work, the integration phase is very challenging since we changed backend a couple of times to make sure it would work with the frontend as well as be able to fit into the full demo deployment. We could set up flask backend earlier and integrate our pipeline at the end of phase 1 or in phase 2 so that we have a better idea of what needs to be done to run Scottybot.

8.5. Human-Robot Interaction Interface

The ScottyBot project, as an innovative experiment in indoor navigation robots, has offered an invaluable opportunity for our Interfaces team to learn and grow. We have come to understand that the design and implementation of the user interface play a critical role in the usability of the robot. Moreover, we have observed that the integration of voice and visual inputs within a multimodal human-robot interaction system is essential for enhancing the user experience. From a technical integration standpoint, we have realized that efficient communication between the front and back end is key to ensuring the stable operation of the system. Particularly in the transmission of route planning and voice data, the real-time and accuracy of data have a direct impact on the quality of navigation services. Therefore, selecting the appropriate communication protocols and technology stack, as well as ensuring the reliability of data synchronization, is foundational to the efficient operation of the entire system. In terms of cross-team collaboration, we have deeply felt that close communication and coordination are indispensable when integrating multiple subsystems.

Based on these reflections and summaries, assuming that the ScottyBot project is restarted, as an interface team we plan to adopt strategies aimed at integrating user interface-related work more efficiently and effectively. Firstly, we intend to employ an agile development approach, implementing and testing each interface function in phases to quickly identify and resolve issues. This iterative method not only speeds up development but also enhances our capacity to adapt to changes, ensuring that strategies can be adjusted in a timely manner throughout the different project phases to meet new requirements and challenges.

Secondly, to strengthen communication and collaboration among teams, we will organize regular cross-team meetings, focusing on sharing progress, challenges encountered, and upcoming needs in interface development. This approach ensures the

close integration of various modules such as hardware, software, and data processing, thereby facilitating more effective overall project integration.

Furthermore, our Interfaces team will focus on optimizing the data interaction architecture, particularly the communication mechanism between the front and back end. We plan to introduce more advanced API management tools and middleware solutions to streamline data flow processing and enhance the system's response speed and reliability. Through these technological improvements, the user interface will be capable of handling complex data requests more swiftly, offering a smoother user experience.

8.6. Communications

The most important takeaway is to design systems with scalability and ease of integration in mind from the beginning. We learned how important modularity, simplicity, and scalability are. By using MQTT and AWS IoT Core, we were able to make the communications network simpler for other subteams to use. At the end of the day, integration is the most difficult task, and using a simple technology like what we did helped the entire team come together more effectively.

As the communications team, our main priority was the integration of on and off-robot systems. Thus it was difficult to begin a large part of our work until other subteams were done with theirs. It would be prudent to think about integration long before it actually needed to happen, since this leaves a lot of the work for the end.

As we developed what is now a clean Python interface for connecting devices to the AWS IoT Core, we had several iterations where we had to modularize, rewrite, and restructure our code. There was also the initial learning curve of digging through documentation and figuring out how to connect devices and build the pub/sub model with AWS, so we learned a lot as we developed our system from scratch. Connecting various devices with different interfaces also taught us how flexible AWS IoT Core is and what is required to get an ESP32, NFC card reader, etc integrated with the system. Lastly, we realized that we needed to also go beyond what felt like our team’s role and understand other subteams’ structure and ideas. Doing this allowed us to figure out how to integrate our MQTT implementation in the correct way.

Along the same vein, If we were to start over with this project, here are some ways in which we would work differently:

If our Communications team could restart the project, we would focus on scalability and modularity from the start. We didn’t start prioritizing that until it became obvious to us that we would have to integrate our code with other subteams.

We would also begin working with our respective subteams much earlier - working together and beginning integration earlier could have prevented a lot of last-minute work being done in the week leading up to the end-to-end demo.

We would also propose assigning overall leadership roles like “demo master” earlier. Basically, have someone be assigned as the project lead during phase 1 or at the start of phase 2. It could make phase 3 go by smoother.

Also, we would define communications interfaces and protocols much earlier, and ensure that other subteams understood the basics of the protocol and were aware of their role within these protocols. Also pressure other subteams to begin integration much earlier.

8.7. Data Systems

During Phase 3, the most important thing we have learned is how to cooperate with other teams. In phase 2, we only considered how to connect to our lambda function in our local environment but not pay attention to the other team’s environment. In phase 3, we need to handle the data from/to the remote operation team for storing or fetching the current location data. Since they mainly implement the frontend code, they plan to let us use the AWS gateway to connect to our lambda function and they will try to connect to our AWS gateway endpoints. However, when we try to set up the AWS gateway, we cannot get the data payload from the remote control team side. In this way, we implement it by connecting to the AWS lambda function directly on the remote control side. This method has safety drawbacks that may expose some important variables. Therefore, if we were to start over with this project, we will try to communicate with other teams more frequently and figure out how to connect with different teams early with different methods.

In addition to prioritizing early communication and collaboration with other teams, the Data Systems team acknowledges the importance of considering cost and system requirements at the project's outset. One key lesson learned is the need to assess the cost-effectiveness and necessity of selected technologies and platforms from the beginning. For instance, the team identified that DocumentDB, while a powerful database solution, may have exceeded the project's storage requirements and incurred unnecessary costs. In hindsight, a more thorough evaluation of storage needs and cost-effective alternatives could have been beneficial. By carefully weighing the cost and necessity of each subsystem at the project's inception, the team could use MongoDB atlas which successfully optimizes resource allocation and ensures efficient use of project resources. This proactive approach would not only contribute to cost savings but also streamline the overall project execution process.

Another thing that was applicable that we have seen in hindsight is that the Data System team is limited by the scope/requirements of the other teams, and one thing our team could have done is to assist the teams in direct communication and relevancy with our system to have better streamlined experiences with their work as well.

8.8. Infrastructure Hardware

At the start of phase 3, we first started with the reflow and testing of the 2nd revision of all of our PCBs before starting to work with the other teams in creating our mockup demos. After the first week, we worked closely with the Infrastructure Software team and the Communications teams to create working prototype demos for the elevator actuation, door actuation, and congestion monitor.

Reflecting back on this experience, since the visionary scenario for the project changed several times as we progressed, the elevator capabilities actually became obsolete towards the middle of phase 2. However, as we had already spent numerous man-hours on these aspects we retained them for future developments to Scottybot. In the case we are able to redo our work starting 3 months ago, we would have focused our efforts instead on infrastructure that Scottybot would utilize in the final demo. While the congestion monitoring system & door actuation would remain, we could instead replace the elevator interfaces with other systems such as infrastructure to call Scottybot to a call button, or further aids to assist in navigation through buildings.

Through this process, we were able to acclimate ourselves with using ESP32 microcontrollers, and designing PCBs to work with these. In addition, we were able to gain experience working in a large project context, where we offload all the software development to other teams and focus solely on hardware development, which helped streamline our process.

8.9. Infrastructure Software

In Phase 3, we collaborated closely with numerous teams, offering software solutions and being readily available to address any identified bugs. Our contribution to the ScottyBot involved designing and implementing several crucial components, enhancing its ability to interact with its environment. The primary emphasis was on refining the algorithms powering these components, which demanded significant time and effort to ensure ScottyBot's enhanced intelligence. Through this process, we gained valuable insights into embedded systems programming and real-world task-oriented programming. We were thrilled to witness the successful execution of the final demonstration and the delivery of a high-quality product.

Reflecting on our experience, we realize that, in hindsight, we should have prioritized task execution over technical refinement. While we dedicated considerable resources to investigating and optimizing algorithms, many of them were already sufficient for their respective tasks, rendering excessive refinement unnecessary.

We learned how to use a rolling window and queue for ultrasonic sensors to create a congestion monitor. We learned about the input type that the infrastructure hardware is expecting for the simulated elevator. We learned about the AWS framework, lambda

functions and how to connect ultrasonic sensors/ ESP32 to Amazon AWS IoT Core using MQTT, and some example codes for congestion monitoring. We learned about how to set up the IDE and program the ESP board and testing examples. We learned about the design and implementation of message type and format, especially for JSON. We learned how to implement door opener, elevator call, elevator floor selection based on ESP32. We learned how to flash our ESP software code within the Arduino IDE environment, to the ESP board. We learned how to set proper thresholds to indicate congestion. We also learned the standard for integration testing, demos, and reports.

8.10. Remote Operations & Controls

During Phase 3, we dedicated significant effort to implementing live streaming using PeerJS/WebRTC and initially achieved success under dorm WiFi and mobile hotspot conditions. However, when we tested the technology at Tech Spark using campus WiFi, we encountered numerous unexpected errors and blockages. These issues required extensive debugging and testing to pinpoint the causes. With the final demo approaching and limited alternative options available, we decided to use Zoom as a backup solution, which ultimately met our streaming requirements. While we were pleased that everything worked out in the end, given more development time, we would prefer to fully integrate and refine the live streaming feature within our interface to ensure functionality across all network conditions.

9. References

9.1. Robot Hardware

1. https://content.vexrobotics.com/vexpro/pdf/VictorSPX-UserGuide-20190117
2. https://www.andymark.com/products/am14u5-frame-only
3. https://cdn.andymark.com/media/AM14U4_Frame_Size_Compariso.pdf
4. https://cdn.andymark.com/media/AM14U4_PW6XL_Layout.pdf
5. https://learn.adafruit.com/adafruit-feather-rp2040-pico/overview
6. https://www.chiefdelphi.com/t/arduino-code-for-cim-motor-via-victor-spx-speed-controller/389511
7. https://www.sparkfun.com/products/12728
8. https://www.instructables.com/Simple-Arduino-and-HC-SR04-Example/Datasheet

9.2. Perception

1. https://learnopencv.com/object-tracking-using-opencv-cpp-Python/
2. https://www.teamviewer.com/en-us/
3. https://docs.ultralytics.com/zh/datasets/detect/open-images-v7/
4. https://github.com/CMU-cabot/cabot_sites_cmu/blob/main/cabot_site_cmu_3d/server_data/attachments/map/cmu/ROS2.png

5. https://github.com/CMU-cabot/cabot_sites_cmu/tree/main
6. https://broutonlab.com/blog/opencv-object-tracking/
7. https://www.stereolabs.com/docs/api
8. https://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams

9.3. Planning and Controls

1. https://github.com/CMU-cabot/cabot_sites_cmu/tree/ros2/cabot_site_cmu_3d/worlds
2. https://app.gazebosim.org/fuel/worlds
3. https://cdimage.ubuntu.com/jammy/daily-live/current/?_gl=1*1v10afr*_gcl_au*MTE2MDg4MjQyNS4xNzA4OTgxNTY3&_ga=2.107582150.555330762.1708983488-1945734171.1708983488
4. https://ros2-industrial-workshop.readthedocs.io/en/latest/_source/navigation/ROS2-Turtlebot.html
5. https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php#:~:text=Dijkstra's%20algorithm%20finds%20the%20shortest,all%20the%20unvisited%20neighboring%20vertices.
6. https://www.geeksforgeeks.org/a-search-algorithm/#
7. https://github.com/arshadlab/gazebo_map_creator/tree/master

9.4. Human-Robot Interaction Intelligence

1. https://github.com/Azure-Samples/cognitive-services-speech-sdk/blob/master/samples/Python/console/speech_synthesis_sample.py
2. https://learn.microsoft.com/en-us/answers/questions/693848/can-azure-text-to-speech-support-more-audio-files
3. https://learn.microsoft.com/en-us/azure/synapse-analytics/machine-learning/tutorial-text-analytics-use-mmlspark#key-phrase-extractor
4. https://learn.microsoft.com/en-us/azure/ai-services/language-service/named-entity-recognition/quickstart?tabs=ga-api&pivots=programming-language-Python
5. https://learn.microsoft.com/en-us/azure/ai-services/language-service/named-entity-recognition/concepts/named-entity-categories?tabs=ga-api
6. https://github.com/Azure/azure-sdk-for-Python/blob/main/sdk/textanalytics/azure-ai-textanalytics/samples/sample_recognize_custom_entities.py
7. https://learn.microsoft.com/en-us/azure/ai-services/language-service/language-detection/quickstart?pivots=programming-language-Python#code-example
8. https://learn.microsoft.com/en-us/azure/synapse-analytics/machine-learning/tutorial-text-analytics-use-mmlspark
9. https://learn.microsoft.com/en-us/azure/ai-services/language-service/custom-named-entity-recognition/quickstart?pivots=language-studio
10. https://medium.com/@mjghadge9007/building-your-own-custom-named-entity-recognition-ner-model-with-spacy-v3-a-step-by-step-guide-15c7dcb1c416

9.5. Human-Robot Interaction Interfaces

1. https://www.figma.com/file/FInHhHOizIqfMS0MRedOOy/Mobile?type=design&node-id=0%3A1&mode=design&t=zF1qFZD1DlSg78x5-1
2. https://www.figma.com/file/KPEBscq9cx4f6QmT61niAY/Robot-Screen?type=design&mode=design&t=zF1qFZD1DlSg78x5-1
3. https://github.com/CMU-RPCS-2024/HRI-Interface
4. https://ant.design/docs/react/introduce
5. https://socket.io/docs/v4/
6. https://axios-http.com/docs/intro

9.6. Communications

1. https://how2electronics.com/connecting-esp32-to-amazon-aws-iot-core-using-mqtt/
2. https://www.youtube.com/watch?v=6w9a6y_-T2o&t=298s
3. https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html
4. https://aws.amazon.com/blogs/database/build-a-sensor-network-using-aws-iot-core-and-amazon-documentdb/#:~:text=In%20this%20post%2C%20we%20discuss,DocumentDB%20(with%20MongoDB%20compatibility)
5. https://realtimelogic.com/articles/How-to-connect-ESP32-to-AWS-IoT-Core-Using-MQTT#:~:text=This%20ensures%20that%20the%20data,IDE%20as%20our%20development%20environment.
6. https://how2electronics.com/connecting-esp32-to-amazon-aws-iot-core-using-mqtt/
7. https://medium.com/@aysunitai/building-a-simple-video-chat-app-with-react-and-peerjs-5650cf89569d
8. PeerJS - Examples
9. jmcker/Peer-to-Peer-Cue-System: Cue system for simple two-way communication and visual signaling using a PeerJS peer-to-peer connection.
10. https://ourcodeworld.com/articles/read/496/how-to-create-a-videochat-with-webrtc-using-peerjs-and-node-js
11. https://www.youtube.com/watch?v=QsH8FL0952k&t=1103s

9.7. Data Systems

1. https://aws.amazon.com
2. https://www.mongodb.com/
3. https://www.mongodb.com/atlas/database
4. https://www.mongodb.com/docs/manual/reference/method/
5. https://docs.aws.amazon.com/lambda/
6. https://docs.aws.amazon.com/lambda/latest/dg/lambda-python.html

7. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_PostgreSQL.html
8. https://docs.aws.amazon.com/iot/latest/developerguide/creating-a-virtual-thing.html
9. https://www.mongodb.com/developer/products/atlas/awslambda-pymongo/

9.8. Infrastructure Software

1. https://aws.amazon.com
2. https://cloud.google.com
3. https://azure.microsoft.com
4. https://realtimelogic.com/articles/How-to-connect-ESP32-to-AWS-IoT-Core-Using-MQTT#:~:text=This%20ensures%20that%20the%20data,IDE%20as%20our%20development%20environment
5. https://how2electronics.com/connecting-esp32-to-amazon-aws-iot-core-using-mqtt/

9.9. Infrastructure Hardware

1. https://docs.espressif.com/projects/esp-at/en/latest/esp32/Get_Started/Downloading_guide.html
2. https://www.espressif.com/en/products/socs
3. https://wiki.dfrobot.com/LIS2DW12_Triple_Axis_Accelerometer_SKU_SEN0405
4. https://stackoverflow.com/questions/76970801/can-i-program-an-esp32-with-another-esp32

9.10. Remote Operations

1. https://react.dev
2. https://ant.design
3. https://leafletjs.com

-