Battle Effects Simulator Robot (BES Robot)

Sponsored by James Todd, PEO STRI

(https://www.peostri.army.mil/)

OSE 4952 | Senior Design II | Summer 2022 | Group 6

Nicholas Nachowicz - Computer Engineering
Jared Rymkos - Computer Engineering
Julia Kemper - Computer Engineering
Michael Rodriguez - Electrical Engineering

Table of Contents

Table of Figures, Diagrams and Tables

1. Description of Project

1.1 Executive Summary

The US Army/ Department of Defense has expressed the need for a Modular Open System Architecture (MOSA) live fire robotic platform/system to support the training of US Army soldiers. The system is intended to be used on live fire training ranges to improve the overall training experience, increase training throughput and improve safety measures by removing the human element. The robotic system will automate the pyrotechnic handling and replacement by supporting the positioning of an existing US Army Battlefield Effects Simulator (BES) that will provide simulated hostile threat fire and hostile vehicle kill indications (as shown in figure 1A,

Figure 1A – Example of hostile fire simulation

provided by PEO STRI). These simulation fire and kill indications are achieved through the use of pyrotechnic cartridges that are activated by the BES (see figure 1B and 1C, provided by PEO STRI.) The robotic system design and control functions will align to the Robotic Operating System Military (ROS-M) framework and ecosystem using open source software to create the robotic framework.

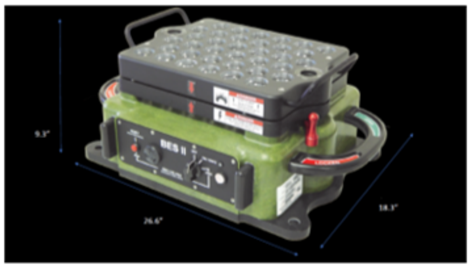


Figure 1B – Battlefield Effects Simulator (BES)

One of the primary benefits of developing a robotic system used for positioning the Battlefield Effects Simulator is to greatly increase safety during training. The robotic system will have multiple safety-related attributes including the ability to traverse the live fire training range terrain without human assistance/entering down range zones, identify when it is in the correct location on the battlefield, and verification that there are no moving elements within a certain vicinity of the robotic platform (i.e. location of the where the pyrotechnics will be fired). When the safety measures are completed, the robotic system logic must determine that the safety conditions have been met before it will allow the control system to send an activation signal to the BES to replicate improvised explosive device events, fire replication, and other direct live fire. Initial RF protocols for control system wireless communication with the robot exist as a US Army standard, the interface specifications for control signals from the robotic platform to the BES will be developed by this senior design team. Due to this specification, the robotic system

Figure 1C – Pyrotechnics cartridge that is place inside Battlefield Effects Simulator

may need to act as a Dynamic Host Configuration Protocol network server that provides a set of parameters to client devices. This will automatically allow the client to communicate on the network.

Along with the ability to support and control a BES, the robotic system will be autonomous with optimal path determination. The design will leverage GPS/GNSS to travel from point A to B smoothly and sensor fusion to detect and avoid any potential obstacles or collisions. This will include using continuous motor control and position tracking to prevent the robotic system from moving more than a specified amount off the correct path. The robotic system will have a sturdy body and wheels/tracks that will allow for it to move in and out of Stationary Armor Target coffin locations without any issues and interference. The robotic system must be power efficient to ensure the system will be able to support the training event and throughput objectives.

The objective of this project is to incorporate automation into a currently manual process, effectively improving the safety of achieving training realism through the utilization of pyrotechnics for simulated hostile threat fire and hostile vehicle kill indications. The goal of this project is to improve the throughput of US Army training exercises and greatly improve safety measures by eliminating the human element.

Update: Due to a smaller budget than originally intended, we had to cut some of the functionality of the robot. The robot that we designed for Senior Design II included all of the features mentioned in this section except the automation of pyrotechnic handling and replacement. The original idea for the robot was for it to have an arm that could replace the pyrotechnic cartridges but that will come in a model of this robot further down the line. We also did not have the robot act as a DHCP server as we stated that we would. Our communication for the robot to BES changed entirely, see the communication section for details. Finally, this robot ended up being a small scale version of what we originally had intended due to the budget cut. Because of this, the robot's size did not allow for the BES to sit on top of it, instead it sat in the testing area, but still performed all other functionality as described.

1.2 Project Description

This section will describe the need/motivation behind developing an autonomous robotics system used to safely travel from one location to another on a live fire training field while firing pyrotechnics when safety conditions are met. The process behind how this will emulate a more realistic training environment, and how it will improve safety will be discussed. Lastly, the overall goal and objective of the project will be presented.

Update: The robot did have autonomous navigation abilities, but per our sponsor's request, it moved both autonomously and by a steering GUI.

1.2.1 Motivation

Currently, the optimal method for the United States Army to simulate live fire and hostile vehicle kill indications on a training range is by the use of pyrotechnics in a Battlefield Effects Simulator (BES). When the pyrotechnics that are used to simulate explosions have all been activated, the BES will need to be reloaded with new pyrotechnics. For this to happen, the training simulation must be stopped and people have to enter the training environment to reload the pyrotechnics before the simulation may begin again. Although this method of training has worked for the army, it still has its drawbacks: lack of reality, and lack of safety. Stopping a simulation to reload these pyrotechnics will not give soldiers an accurate representation of what a constant live fire field is like. Also, bringing a human element into the equation by having a person enter the training environment when the pyrotechnics need to be reloaded can potentially be very dangerous. Taking these drawbacks into consideration, our team plans to develop an autonomous robotics system that will house the Battlefield Effects Simulator and travel to the desired locations the pyrotechnics need to fire at. When the pyrotechnic cartridges have all been used, the robot will then return to a base where it can be safely reloaded without the training simulation having to stop. The aim of this project is to provide a method to reload and activate pyrotechnics without having to stop the training simulation and improve safety measures by reloading the BES without a human having to enter the training field, therefore, eliminating the human element.

When deciding what factors/requirements are necessary for this robot to perform to the level needed, we took into consideration factors that were overlooked when a human was reloading these pyrotechnics: terrain and safety features. Because this robot will be traveling across training fields used to simulate battlefields, the terrain will be extremely rough. It will include holes, narrow paths, and large divots that a human would navigate much easier than a robot. We will ensure the robot is able to safely navigate this rough terrain while still maintaining the desired speed to make it to the specified location for the pyrotechnics to go off. As mentioned earlier, once the robot has reached its desired destination, it will need to have met safety conditions and receive a safety signal for the pyrotechnics to fire. The robot will need to ensure it is at the correct location, it will need to ensure there are no moving objects within a given vicinity, and it will need to make sure it is not on an incline greater than a given degree. These features will continue to enhance safety measures.

1.2.2 Goals/Objectives

When our team was deciding on goals and objectives, our main goal was for this robotic system to be utilized on live fire ranges to support training realism and feedback, threat representation, and pyrotechnical handling/replacement. This robot will transport the BES through rough terrain to and from desired locations to accurately fire pyrotechnics when proper safety signals are sent. The robot will then return to a designated base when the pyrotechnic canisters need to be reloaded. Moving the robot will be done through some kind of tablet application. The user will be able to select which zone the robot will travel to and then fire the pyrotechnics. The robot will interact with the BES to send the safety conditions for when it is clear to fire the pyrotechnics. The robot and BES will also need to interact with the current control center that operates the BES so all three systems are in communication.

Update: The robot is a small scale version of what was originally intended due to budget cuts, therefore it did not hold the BES.

1.2.3 OV-1: High-Level Operational Concept Graphic

Figure 1D: OV-1: High-Level Operational Concept Graphic below shows our OV-1: High-Level Operational Concept Graphic. This graphic is intended to describe the mission of the robot and BES. You are able to clearly see the main operational concepts and the Battlefield Effects Simulator Robot in an actual scenario. The robot will wirelessly receive communication from the control center when it needs to move to its specified destination. Once the robot arrives at its safe location inside the Stationary Armor Target coffin, it will then wait for a signal that a set of safety conditions have been met. When the safety conditions are met and the signal is sent to the robot, the pyrotechnics will then fire, simulating hostile threat simulation and hostile vehicle kill indications.

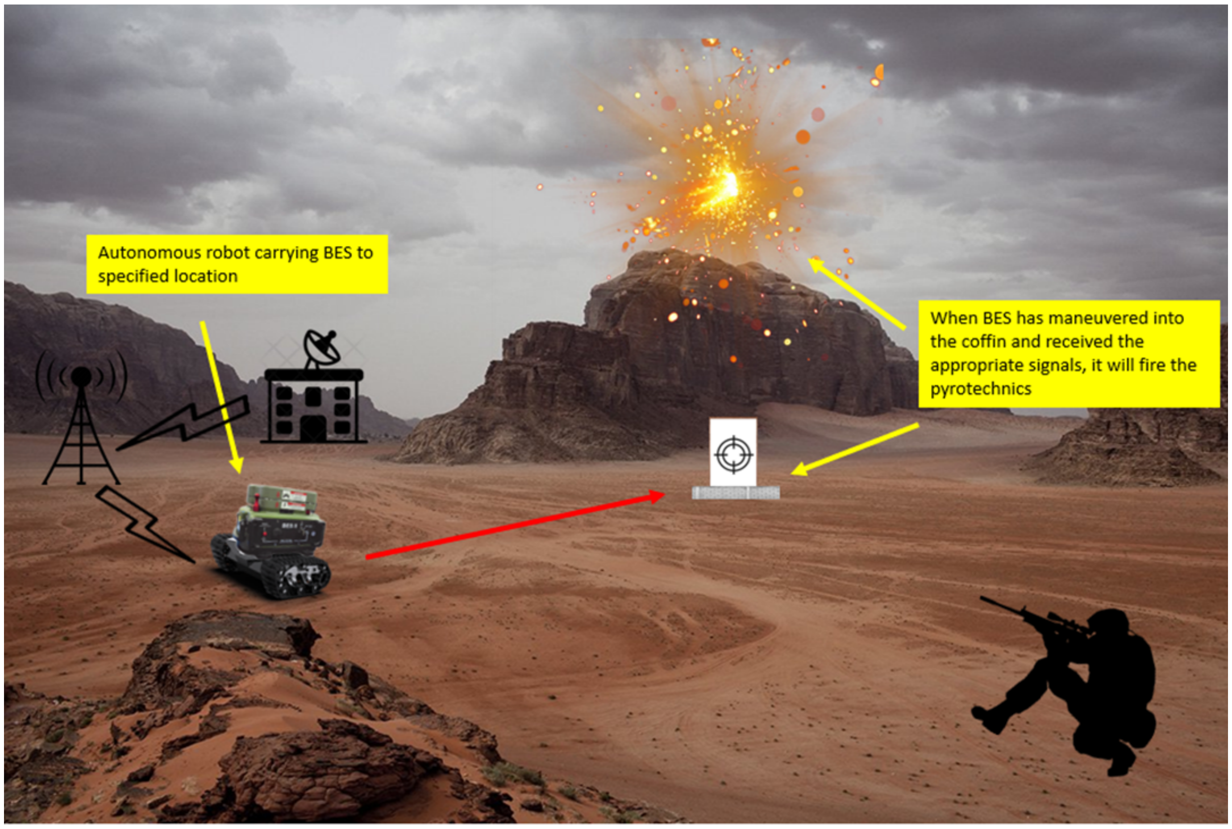

Autonomous robot carrying BES to specified location
When BES has maneuvered into the coffin and received the appropriate signals, it will fire the pyrotechnics

Figure 1D: OV-1: High-Level Operational Concept Graphic

1.2.4 Personnel

This section is intended to describe the personnel involved in this project.

Senior Design Team Members:
Nicholas Nachowicz: As the project leader, I acted as the primary spokesperson during group meetings with sponsors and mentors and delegated tasks when necessary. Within this assignment, my main responsibility was hardware and hardware connectivity. My fundamental achievement in this project is the complete design of the microcontroller as our PCB. However, throughout the duration of the project, I focused on investigating each piece of hardware that is vital for achieving our functional requirements. Additionally, I researched how each piece of hardware would interact with one another to ensure efficiency in the designing process. Through this research, I had the ability to complete the budgeting, hardware block diagram, system subsystem diagram, and house of quality matrix for this project.

Jared Rymkos: My main responsibilities for this project were the Artificial Intelligence/Camera integration and the user interface. I researched different methods for detecting people including Convolution Neural Networks and the algorithmic approaches like the Canny Algorithm.

Additionally I worked on the communication aspect between how the user will communicate with the Robot and how the Robot communicates with the BES.

Julia Kemper: My primary responsibility throughout this project was Robot Operating System (ROS). I did research on what operating system would work best for this project, how we are integrating ROS into this project, how to use ROS, and how we will connect ROS with all other electrical components. I was also very involved in the beginning stages of this project and developing the goals, objectives, motivation, and requirements.

Michael Rodriguez: My primary responsibility throughout this project focused on the design of the robot base itself. I have experience with building robots from previous courses I have taken and was able to use this knowledge to lead the design process. My research focused on whether wheels or tread would be the best option for the robot based on the terrain and its tasks it needs to complete, what type of motor and battery the robot would need based on the derived requirements, and creation of the hardware block diagram.

Sponsors and Mentors:
James Todd: James Todd is our Sponsor for this project. He is the PEO STRI Chief Engineer Force on Target (FoT). We hold weekly meetings with him to discuss our design process, requirements, and potential funding he can provide for this project.

Thomas Waligora: Tom Waligora is our robotics mentor contact. He is the Senior Chief Robotics Engineer at Pratt Miller. Tom attends our weekly meetings and provides insight and advice on our robotics design process.

David Howard: David Howard is our software mentor for this project. He is an engineer at Shock Stream Technologies. David attends our weekly meetings and provides insight and advice on our software design process.

Update: Jeremy Lanman was also another Sponsor for this project. He is the PEO STRI Chief Technology Office (CTO). We held weekly meetings with him to discuss our senior design process, requirements, and involvement at I/ITSEC.

2. Project Requirements Specifications

2.1 Requirements

The requirements shown below in Table 1: Requirements were created in order to show clear, quantifiable, and achievable goals that will be accomplished by the end of Senior Design 2.

Requirements		
2.1.1 General		
Number	Requirement	Value
2.1.1.1	The robot will follow the requirements given by PEO STRI.	
2.1.1.2	The robot will be a small-scale prototype size of the product PEO STRI will implement in the future. a. The robot will be a scaled solution size necessary to support a 6-shot Battlefield Effects Simulator with a width of 8.9”, length of 11.9”, height of 8.3”, and weight of 21 lbs.	
2.1.1.3	The robotic base will move into existing Stationary Armor Target coffin locations without interfering with any existing equipment.	
2.1.1.4	The robot will drive autonomously to a specified destination.	
2.1.1.5	The robot will have no more than a 0.15m path deviation to its specified destination.	
2.1.1.6	The robot will consider terrain and narrowness or paths to perform optimal pathing. a. The robot will potentially sense stable paths while traveling using machine learning.	
2.1.1.7	The robot must be able to detect and maneuver over rough terrain to arrive at its desired safe location. ● A movement map with safe points and traversed paths will be provided.	
2.1.1.8	The robot must be able to detect collisions to avoid existing equipment in or on the way to destinations. a. The robot will slow its speed when rough terrain or narrow paths are detected.	
2.1.1.9	The robot will need to comply with IP65 weather rating.	
2.1.1.10	The robot will travel at a speed 1/6th the scale of the robot desired with a 2mph average.	
2.1.2 Hardware		

2.1.2.1	Connected components inside the frame of the robot: A custom printed circuit board (PCB), Battery bank, Motors, GPS receiver, Receiver for accepting inputs from a person, AI processor, Camera, Microcontroller, Range sensor	
2.1.2.2	A custom PCB will be attached to a range sensor, GPS receiver, receiver to receive data on the desired location, infrared sensor, motors for the treads/wheels, external storage with AI subsystem, and rechargeable battery bank. a. The camera(s) will be connected to the AI subsystem.	
2.1.2.3	A range sensor will be attached to the front and back of the robot for it to avoid collisions while moving forward and backward.	
2.1.2.4	There will be infrared sensors around the robot to investigate if any person/moving object is in the area.	
2.1.2.5	There will be cameras giving the robot a 360° view that will help in identifying people in the surrounding 2m area and if the robot is on the correct path without veering off course.	
2.1.2.6	The robot will be utilizing motors for its treads/wheels so that it can maneuver over rough terrain. This is paired with the AI to ensure no path deviation.	
2.1.3 Software/Firmware		
2.1.3.1	The robot will use open-source software aligning to the Robotic Operating System Military (ROS-M) framework and ecosystem.	
2.1.3.2	The robot will send an activation signal to the Battlefield Effects Simulator once a safety message is received from the control system a. The robot will be in communication with the range control system. b. A safety message will be sent when the following conditions are met: i. The robot has reached its specified location ii. The robot has determined there are no moving elements within a 10-meter vicinity iii. The robot is on a surface that is flat +- 10 degrees.	
2.1.3.3	The robot base will act as a Dynamic Host Configuration Protocol (DHCP) server.	

	a. BES will connect to this server b. Commands will be parsed out from ICD to BES to accurately fire after a safety signal from the robot is received.	
2.1.3.4	Four separate training zones will be defined in the firmware. a. The robot should be able to operate and properly receive commands/fire pyrotechnics in all four zones.	
2.1.4 UI		
2.1.4.1	The robot will be turned on by a power button.	
2.1.4.2	An application will contain the UI used to control the robot. a. The application will be able to control four separate training zones. b. The application will have different destinations that a user can select for the robot to maneuver to. c. The UI will display a fire button that will be able to be pressed only if a safety signal is received from the robot indicating that the safety conditions for firing pyrotechnics are met. d. The application will tell the user the current battery life of the machine.	
2.1.5 Power		
2.1.5.1	Powered by a rechargeable battery a. The rechargeable battery will have a standard wall plug that can be plugged into an electrical outlet to charge. b. The rechargeable battery will have a battery life minimum of 10 hours.	

Table 1: Requirements

Updates: Due to budget cuts and time constraints our robots requirements were cut drastically. Here are the core requirements both the team and the sponsor agreed is feasible due to time and budget constraints. All requirements have a value of 10 because these are all things the sponsor absolutely wanted to see us achieve.

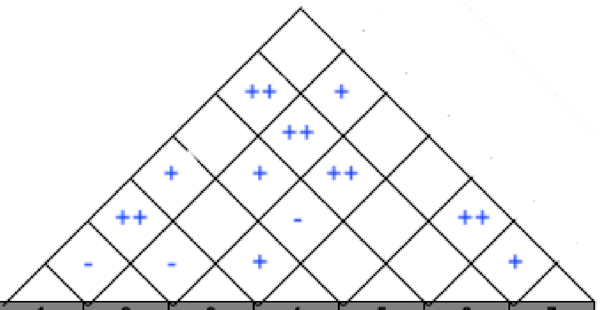
Requirements		
Number	Requirement	Value
1	The robot will use lidar data to create a map of its environment	10
2	The robot will correctly navigate to its specified destination	10
3	The robot will detect obstacles in its path and navigate around them	10
4	The robot will ensure it is level +- 10 degrees when it has arrived at its specified destination	10
5	The robot will ensure it has arrived at the correct destination	10
6	The robot will send signal to GUI via ESP32 that it is safe to fire	10
7	GUI will send signal to BES to fire pyrotechnics	10

Table 2: Requirements Updated

2.2 House of Quality Matrix

House of quality matrix diagram is a good representation of the trade-offs our group has to face when designing the robot. In Table 2: House of Quality, our team compares the main requirements that the customer demands to the engineering requirements that all engineers need to consider.

Legend	
Θ	Strong Relationship
O	Moderate Relationship
▲	Weak Relationship
++	Strong Positive Correlation
+	Positive Correlation
—	Negative Correlation
▼	Strong Negative Correlation
▼	Objective Is To Minimize
▲	Objective Is To Maximize
x	Objective Is To Hit Target



Column #	1	2	3	4	5	6	7
Direction of Improvement: Minimize (▼), Maximize (▲), or Target (x)	▲	x	▲	x	▼	▼	▼
Quality Characteristics (a.k.a. "Hows") / **Demanded Quality** (a.k.a. "Customer Requirements" or "Whats")	Efficiency	Output Power	Quality	Implementation Time	Weight	Cost	Dimensions
Open Source (REQ: 2.1.3.1)	O	▲	O	Θ	▲	▲	▲
Modular (REQ: 2.1.3.1)	▲	▲	O	O	▲	O	▲
Autonomous (REQ: 2.1.1.4)	Θ	Θ	Θ	Θ	O	Θ	▲
1/6 Scale (REQ: 2.1.1.2)	▲	Θ	▲	Θ	Θ	Θ	Θ
Holds the BES (REQ: 2.1.1.2)	▲	O	▲	O	Θ	Θ	Θ
Durable (REQ: 2.1.1.7, 2.1.1.9)	▲	▲	Θ	Θ	Θ	Θ	Θ
Accurate Pathing (REQ: 2.1.1.5, 2.1.1.6)	Θ	O	Θ	Θ	▲	Θ	▲
Accurate Safety Detection (REQ: 2.1.3.2)	Θ	O	Θ	Θ	▲	Θ	▲
Long battery life (REQ: 2.1.5.1)	Θ	Θ	▲	▲	Θ	Θ	Θ
Target or Limit Value	-	Minimum 10 hour battery life	-	1 Semester	-	<= $5000	No larger than target coffin entrance

Table 3: House of Quality

3. Research

3.1 Research Specifications

Before beginning our research, our team outlined specific areas of the project that we would need to take into consideration during that process. Some of these areas include:

- Cost: The finances and resources that are available to the team through PEO STRI.
    - Parts for the robot must be high quality yet affordable. The robot needs to follow MOSA (Modular Open System Approach) compliance. MOSA is defined as a technical and business strategy for designing an affordable and adaptable system.
- Ease of Use: How quickly/easily the team is able to understand and implement hardware, firmware and software during the building process of the robot.
    - The software used for the robot will need to align with the Robotic Operating System Military framework and ecosystem while also being easy to implement/understand by the team.
    - Hardware chosen should be easy to implement/understand by the team, and easily accessible to future teams when furthering this project.
- Functionality:
    - The robot should be able to perform all of the requirements derived by PEO STRI and our senior design team.

3.2 Research Analysis Process

3.2.1 Previous Projects and Similar Robots

When designing something new what are the first things you should research? Researching previous products, projects, and similar things can help get a better understanding of which design paths are best for your specifications. Open source projects can be a significant help when designing something new because anyone can use the contents of the project to build something of their own. When researching for our robot we need something built to withstand terrain, path optimally, and define safe areas. As we started looking at the requirements of our project one of the first robots we thought of was TurtleBot. This is a commercial robot and not just someone's project. This robot is used quite often for open source projects however everything you need is already built and put together making it easy to plug and play. Our group obviously has to build the robot from the ground up however a commercial robot is one of the best options to form ideas.

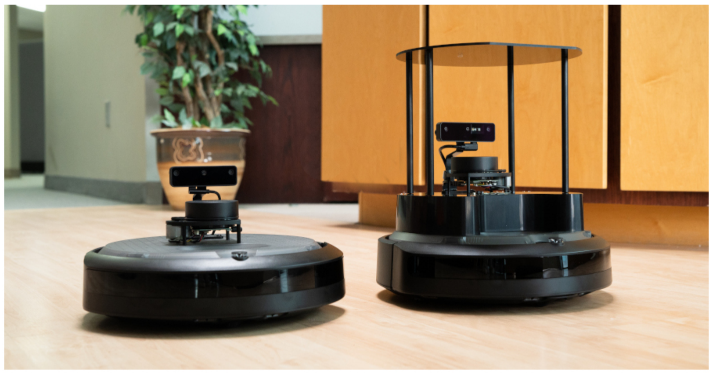


Figure 2A: TurtleBot V4

TurtleBot: TurtleBot is an open-source personal robot kit. Figure 2A: TurtleBot V4(Clearpath Robotics) on the right displays the TurtleBot V4 and the attachments that are available for purchase. Not only is there information on the TurtleBot website but there are a plethora of projects using the TurtleBot online. What are some of TurtleBot's features that can help with building the (BES)Robot? This robot comes with several sensors: OAK-D spatial AI stereo camera, 2D LiDAR, IMU, optical floor tracking sensor, wheel encoders, infrared, cliff, bump, and slip detection all for a low cost. All of these sensors would be incredible for the robot our group is currently building but what is its cost? TurtleBot's newest version has a low-cost value set at $1,850.00 dollars. Our group is clearly trying to achieve a lower price than the newest TurtleBot however this gives us a good idea of gauging our project's cost. TurtleBot also uses ROS, a robot operating system that has several tools to develop applications, drivers, and algorithms. There is plenty of information online about ROS that our group plans on using. The last pro that the TurtleBot has is carrying weight. This robot can carry roughly 33lbs. Our robot needs to carry a 21lb BES which is significantly less than the maximum carry weight. The TurtleBot may sound like the perfect robot from what has been stated so far however there are some cons. The TurtleBot has a top speed of .306m/s which is ok for a robot but not the robot we want to build. Our robot needs to be much faster in order to reduce the time between training safe zones. Our robot will also need to be able to maneuver through harsh terrains whereas the TurtleBot is most functional on flat terrain. The Turtlebot would not be able to operate in grass or gravel.

Pros:

1. Uses sensors we will need in our robot
2. Fair cost comparison that is well within our budget
3. Carry weight.
4. ROS operating system
5. Open source
6. Several projects were created that are open-source online

Cons:

1. Robot speed(which also takes into account motors and battery)
2. Not good in rough terrain

Another great robot that should be talked about is the food delivery robot. These robots are built to take the place of delivery drivers on a small distance scale autonomously. These robots come from a company named Starship Technologies. Starship Technologies makes several different autonomous vehicles including the delivery robot our group researched.

Starship Robot: The starship robot is a food delivery robot created in 2014 and is shown in Figure 2B: Starship Robot. As for some of the specs of the robot: it can travel at 3.7mph, hold up to 22lbs, weighs 50lbs, and each dimension is under 2 and a half feet. We believe this speed is more than enough to get from one safe zone to another quickly and accurately. Not only do we believe that but Starship Technologies has proved that by stating their robots can, “pinpoint its exact location the nearest inch”. This robot utilizes computer vision and GPS to reach its destination accurately. The company also stated the robot has obstacle detection, “ It has a ‘situational awareness bubble’ around it – featuring either twelve cameras, ultrasonic sensors, radars, neural networks and more to detect obstacles, whether that is a dog or a pedestrian/cyclist.”. As stated in the TurtleBot research it uses several of the technologies we plan to use in our robot. The Starship robot has rough terrain capabilities with Starship technologies stating that it can travel up curbs and operate in rain and snow. It may be able to traverse rough terrain however it is unable to flip itself over in emergency situations. One of the major issues with this robot is that it isn’t open source and therefore very difficult to find any information on specific parts to help with our build.

Pros:

1. Fast and accurate pathing
2. Terrain and environmental awareness and protection
3. Obstacle detection

Cons:

1. Not open-source
2. Carry weight is too close to our payload

Now what have other senior design projects achieved? There are several robots that are built in every senior design class.

Autonomous Target Robot: While looking for senior design projects that are similar we found an autonomous target system. This target system was a naval research project for the military similar to ours. The robot would present a target system to a shooter after orders to go to a destination. This project would use an Arduino to wirelessly receive and transmit data. The Arduino would also send specific GPS locations of the robot. The robot would, unfortunately, have poor performances of 7-10 feet in poor conditions. These standards don't match our requirements because we need a .15 meter standard deviation. This project also doesn't seem to have much information on what they did to build the robot. However, there is a list of what hardware they used. This list can give our group a good idea of sensors and controller boards to use for our robot. Below is the poster of the robot built by students at UCI Samueli in Figure 2C: UCI Robot. This robot depicts something achievable by students in senior design. Our project is more ambitious but still achievable with proper research and time. As we research parts it will be important to understand how they connect and branch off of each other.

Pros:

1. List of parts
2. Shows what is already obtainable

Cons:

1. Poor accuracy

All of these robots will help in understanding what we need in order to build our robot. It seems that GPS systems would be the best way to path our robot. ROS systems seem like the best open source pathing capabilities that we can find. Our group will be looking into sensors such as LIDAR to interact with the world and its obstacles, especially in unknown terrain.

3.3 Relevant Technologies

3.3.1 Wheels vs Tank Treads

Our team was tasked with deciding whether wheels or tank treads would be best for the robot's mode of transportation. When conducting this research, we took the following into consideration:

1. **Speed:** In terms of speed, wheels have a higher speed than treads due to the wheels requiring less torque to move from the stationary position and have less friction than treads.
2. **Maneuverability:** Wheels have higher maneuverability than treads due to there being more precision with the wheels turning which causes the robot to have better movement when turning. Treads on the other hand use more power when turning and the movement of the robot is slowed down heavily or stops moving to turn.
3. **Terrain/Traction:** Treads have higher traction than wheels which leads to better movement on different types of terrain. Treads also have less of an impact on the ground due to the distribution of the weight over the length of the treads than having the weight concentrated over several points like wheels which leads to the treads performing better in terrain with soft soil or snow since the treads wouldn’t get stuck in a small rut like wheel would. Treads also use a higher torque which improves the performance when going uphill or over small obstacles while a wheel needs to be at least twice the size of an object in order to go over it.

4. **Steering:** Both wheels and tank treads have their advantages and disadvantages when it comes to steering the robot and keeping it to its specified path. Wheels have an advantage when it comes to steering. It is easier to do sharper turns, but in the case of our robot and the terrain it will be maneuvering on, we are not as concerned about sharp turns as we are about narrow paths. Because of this, tank treads will be the more optional solution. It is imperative that when the robot is traveling that it can handle these tight paths that are potentially on cliffsides/near holes and ruts. Tank treads will not get caught on cliff sides and will not become stuck in holes.
5. **Accuracy:** Due to the higher traction and friction, which leads to there being less of a chance to slip, the treads will have higher accuracy than wheels. Since wheels have a chance to move and slip while in motion, the amount of accuracy that can be obtained on following a path from a wheel would be less than treads.
6. **Cost:** The cost of wheels is much cheaper than treads and are much easier to replace than treads. Wheels also have simplicity in the components that are used since wheels are already set up when purchased. Treads on the other hand have to be set up with their gears that connect to the treads and the tread length has to be set up so that the treads are not too loose. Furthermore, treads are harder to repair or replace since this affects an entire side instead of wheels only needing a repair/replacement at 1 spot.

After our research, the team decided that tank treads would be the best option for the robot. This is due to the better stability and traction on other types of terrain since the robot will spend a lot of time outside running on different types of terrain. The other aspect is the tread's ability to go over objects better than wheels since wheels have to be at least twice the size of an object in order to go over it. Treads also have less of a ground impact which will lead to the robot having less of an effect on the ground in the field which will lead to a smaller probability of evening terrain or covering holes that the robot could have made when a wheel gets stuck in a small rut. Furthermore, since we are making a smaller model of the robot, we have to account for the potential that the robot will become heavier in the future which is why treads were chosen since the treads can support a heavier load than wheels.

Update: We had decided to use a pre-designed base that utilizes 4 treads instead of two which makes all the treads cover less total surface area than having two sets. The base is set up with a 4 wheel drive but instead of wheels, there are treads at the locations where the wheel should have been. This should still provide good terrain coverage and traction in moving the robot.

3.3.2 Rangefinder

There are many types of distance sensors including Ultrasonic, LED, LIDAR (Laser), and VCSEL. We need to decide the best distance sensor to be used on the Robot that meets our needs. The following factors were all taken into account:

1. **Refresh Rate**

    a. This is the time it takes for the sensor to update a new value for the distance, the faster the refresh rate the better.
    b. Ultrasonic sensors have a poor refresh rate which means there will be a delay between recognizing an object.
2. **Sight Distance**
    a. The distance can reliably be seen by the sensor. If the sensor is capable of seeing long distances it can make better judgments on pathfinding..
    b. Ultrasonic sensors are capable of seeing up to 8 meters.
    c. Laser sensors are capable of seeing up to 3000 meters!
    d. VCSEL has a very short distance of up to 2 meters.
3. **Price**
    a. This is not the main factor however, we will likely need multiple distance sensors so we will need to be able to afford to buy many of them.
    b. Laser sensors are generally much more expensive than Ultrasonic and IR which can easily be bought in bulk for very cheap.
4. **Interference**
    a. The main interference we will be facing is sunlight, however, there is also the possibility of dust and fog.
    b. The ultrasonic is not affected by the above interferences because it uses sound waves to measure the distance. However, it is affected by other sensors using the same frequency which can be a problem if multiple distance sensors are used.
    c. The LED sensor is affected by sunlight which makes it not ideal for outdoor use.
5. **Resolution**
    a. The resolution of a distance sensor is basically how small of a distance it can tell apart. This is very important for applications where the distance needs to be very precise at short ranges but not too important for us.
    b. VCSEL has the best resolution at the tradeoff of having a low maximum distance.
6. **Power Consumption**
    a. Ultrasonic and VCSEL do not consume too much power
    b. LIDAR consumes a lot more power which may make it hard to power multiple of them.
7. **Safety**
    a. Some powerful LIDAR sensors may be dangerous to the human eye, so this would be an extremely important factor to look into if a LIDAR sensor is selected.

After the analysis above we were able to rule out VCSEL distance sensors because their max distance is too low. This is an important factor because by having a high max distance we will be able to detect obstacles far away and update the optimal path before being right next to the obstacle. Next, we were able to rule out LED distance sensors because sunlight is a huge interference for them. Because our Robot will be operating outdoors this would not be ideal and cause inaccuracies in the distance. Finally, we ruled out Ultrasonic sensors because their refresh rate is too slow. This is a problem because the Robot will be moving while the sensor sends the wave from the old location, and when the wave returns the Robot will be in a different location causing an inaccuracy. Other limitations with the ultrasonic include only going up to 8 meters and possible interference if multiple ultrasonic sensors are used due to using the same frequency.

These limitations are not present with the LIDAR distance sensors which is why we selected them for the project. There are a few tradeoffs with this decision but we believe it's our best choice. The tradeoffs include making sure the laser is safe for human eyes, being more expensive, and consuming more power than the alternatives. With this decision, we decided to look into different LIDARs. All of the LIDAR sensors in Table 3: LIDAR are 360 range and available through Amazon and can be obtained within a week.

360° LIDAR Sensors				
Number	Name	Sensor radius Distance (meters)	Power Consumption (watts)	Cost (US dollars)
1	Slamtec RPLIDAR A1M8	12m	.5W	$99.99
2	Slamtec RPLIDAR Lidar SLAM A3	12m	3W	$595.99
3	Slamtec RPLIDAR Lidar SLAM S1	40m	> 2W	$649.99

Table 4: 360 LIDAR

As you can see in the table above there are several different LIDAR sensors available for commercial use. Has one difficult obstacle for all LIDAR sensors and that is outdoor capabilities. The first sensor listed is only capable of indoor use. Sunlight can affect the performance of the LIDAR making it impossible for the sensor to work. The two sensors that are listed after are unfortunately much more expensive however they are capable of outdoor use. This type of robot wouldn't need a 40-meter radius and therefore would be fine with the second LIDAR, Slamtec RPLIDAR Lidar SLAM A3. This LIDAR would be a significant expense for our robot and need to be used very carefully. This LIDAR would also have to be placed on the top of the robot where the BES is mounted. Unfortunately, this would mean we need 2 of the same sensor for each side of the BES to avoid having blind spots. Other options would be mounting it higher in the air but this would inevitably be destroyed by pyrotechnics that the BES launches. If the sensor is mounted underneath the robot there would be significant blind spots from the treads. Finally, the only other option would be mounting it on a pole in front of the BES to ensure the most 360 range of view and minimize its blind spots. The fact that there is such a high cost and difficult blind spots to overcome leads our group to investigate our other options more thoroughly.

There are still LIDAR sensors that are more affordable and have better functionality for our design. Single-point LIDAR sensors are a much more affordable way to use LIDAR. The sensor may not be able to detect 360-degree FOV however, the readings it gives are fast and accurate for the price available. One option in building our robot is purchasing multiple single-point LIDARs and attaching them to the robot strategically on the sides. The option of using single-point sensors would be nearly half the cost of using a 360-degree outdoor LIDAR sensor. Below in Table 4: Single Point LIDAR is a few single-point LIDAR sensors that are found on amazon and can be obtained within the week.

Single-Point LIDAR Sensors				
Number	Name	Sensor radius Distance (meters)	Power Consumption (watts)	Cost (US dollars)
1	SmartLam MVR2EB	.3-14m	1.25W	$39.99
2	Stemedu Small TFmini-S	.1-12m	.12W	$43.90
3	Stemedu TFmini Plus	.1-12m	.55W	$53.90

Table 5: Single Point LIDAR

Taking a look at these single-point LIDAR sensors they are all about the same price of each other. The distance on the first sensor is the largest but it also can only read a minimum distance of 30cm from the sensor. If something was to come in the range of 30cm, the only reading we would receive is 30cm which could be an issue. The maximum distance isn't as big of an issue because as it comes into the range we can plan ahead for when we reach the obstacle. That makes the last 2 sensors better options. There are 2 very good things each sensor has. Sensor 2 on the table has a very low average power consumption. Power consumption under a watt is pretty much nothing when looking at what our robot's motors and BES will need and therefore the power they are consuming is practically negligible. The cost of sensor 2 in the table is also 10 dollars cheaper than sensor 3. Sensor 3 however is most likely the best option for our robot because it is IP65 standard and that complies with our requirement 2.1.1.9. The cost of sensor 3 may be more expensive than sensor 2 but it can be used in the environment that we are working in immediately. We estimate that our robot would need 6 sensors to give us the most information needed and FOV while minimizing the cost of a 360-degree LIDAR

Automotive radar could be the answer to rangefinding and obstacle detection troubles. Almost all newer vehicles utilize radar with their backup sensors and blind-spot sensors. This type of radar is called FMCW(frequency-modulated-continuous-radar) radar. This radar does very well when paired with cameras and is usually much more cost-effective than other typical range detection sensors. Continuous-wave radar can measure the instantaneous rate of change but it cannot tell what the distance is of the object it detected. However, with frequency modulation, the sensor will be able to understand the distance of the object. As the sensor sends out a frequency the sensor will receive the same frequency with a delay. The change in frequency will allow you to get a change in time and ultimately you will be able to calculate a distance.

Inside an FMCW radar there are 2 antennas one for transmitting and one for receiving frequencies. The sawtooth generator generates a wave similar to the figure above transmitted through an FM transmitter and then is compared to the frequency received from the receiving antenna. The frequency is compared in a frequency modulator to understand the time and distance an object is. A mixer receives and mixes the signal with a high bandwidth signal. The signal is amplified and then passed into a limiter to remove noise finally passing to the indicator. The indicator will determine whether there is an object present or just noise.

After researching we found a couple of options that look promising. In Table 5: FMCW below we discuss the range of the sensor, the supply voltage, and the cost of the sensors themselves.

FMCW RADAR				
Number	Name	Sensor Distance (meters)	Supply Voltage (Volts)	Cost (US dollars)
1	HLK-LD303	.1-3.5m	5V	$24.33
2	FM24-NP100	.5-20m	5V	$76.50

Table 6: FMCW

We were lucky enough to find a video that would help us understand how the FM24-NP100 works. The angle at which the sensor reads is very focused. It is much harder to detect smaller objects from a far distance. This would still be ok however because it would give us a 3-meter radius even with smaller targets. The sensor is quite expensive and it begs the question of just using LIDAR. If the radar has a very focused signal then it would be more similar to LIDAR and work much less efficiently than LIDAR.

Update: As we determined that this would inevitably be a robot for proof of concept after lowering our budget it was found that the robot will be using a 360 degree RPLIDAR. The BES was no longer going to be a required to be ontop of the robot which made this decision very easy.

360 LIDAR was always one of the best options especially while using ROS however it could not be ontop of the BES because it launches pyrotechnic cartridges inevitably destroying the device. With no BES atop the robot we can get a perfect laser scan of the robots entire surroundings. The new LIDAR has a 12m detectable range, a 8000 sample rate, 2-10Hz, and a 5V input. This LIDAR is plugged into a USB to TTL converter and then the raspberry pi.

3.3.3 Motor Analysis

There are several factors in deciding which motor would be best for the robot-like Torque, RPM, cost, voltage, and size. We need to analyze each factor to decide generally what the specs of the motor need to be in order to move the robot.

Torque: This is one of the most important aspects of the motor because the amount of torque a motor has determines how much weight the motor can move and also determines how well a robot will be able to climb an incline. Essentially in terms of equations, the torque($\boldsymbol{\tau}$) is calculated using force(F) and distance(d) which can be used to calculate how much force the motor provides depending on the distance. This can also be modified using gear ratios to increase the torque but comes at the cost of speed. The other thing that we have to account for with torque is the force from the friction between the wheels/treads and the floor since this will be the force that the motor is required to overcome. Since the robot will be going over several different types of surfaces, we would probably calculate the highest coefficient of friction that the robot will probably come across in deciding what torque the motor needs to run. The following are some formulas to take into account:

a. $\boldsymbol{\tau}$ = F*d (this formula accounts for the torque of 1 motor, when utilizing more motors we have to adjust the formulas based on the number of motors we are using. This also means that the more motors we use, the less torque a single motor needs to move the robot.)
b. $F_f = \boldsymbol{\mu} * F_N$ ($\boldsymbol{\mu}$ is the coefficient of friction) (F_N is the force opposing against the ground)
c. $F_N = F_G \cos\boldsymbol{\Theta}$ (F_G is the force caused by gravity (essentially the weight of the robot)) ($\boldsymbol{\Theta}$ is the angle of an incline the robot would go up)
d. Go/Gi = gear ratio (Go is the output gear and Gi is the input gear)
e. Gear ratio = $\boldsymbol{\tau}$o/$\boldsymbol{\tau}$i ($\boldsymbol{\tau}$o is the output torque and $\boldsymbol{\tau}$i is the input torque)

RPM: This is the rotations per minute (rpm) that the motor will output. This factor is important in deciding how fast the robot will move depending on the size of the wheel or gear that the motor is turning. A high RPM with a big wheel or gear will lead to a higher speed which can further be modified by utilizing gear ratios to increase the speed at the cost of losing torque. Based on the required speed the sponsor would like the robot to go would affect the RPM required in order to achieve that speed. There are some formulas to take into account with RPM and some variables to account like the circumference of the wheel/gear that is being turned and the factor that gear ratio can affect the speed. The following are some formulas that we have to take into account when deciding the speed of the robot:

a. Speed = RPM*2*π*r (r is the radius of the wheel/gear that is being turned and the speed will be based on per minute instead of per hour which can be solved with some conversions)
b. Gear ratio = RPMi/RPMo (RPMi is the input RPM and RPMo is the output RPM which would be calculated in the previous formula if the gear ratio is utilized.

Cost: This is not a heavy factor that we are focusing on but the cost of the motor will influence the total price of the robot and the price to repair/replace the motor. The cost also tends (not always) to affect the quality of the motor in which a higher costing motor tends to have better specs or performance than a lower cost motor. Depending on the specs that we get from deciding the other factors of the motor will determine essentially the price range of the motors with the specs that we need to operate with the robot.

Voltage: This is an important factor in deciding the battery that we would need to operate the robot. Each motor has its own operating voltage range which can help in doing the calculation in identifying the size of the battery required so that the robot can run for 10 hours. This factor mostly requires us to identify the speed and torque that the motor requires and find the voltages that the motor with those specs run with.

Size: This factor is important in keeping the robot more compact because a bigger motor will add more weight to the robot and make it bulkier at the base to account for the size of the motor. In this factor, when we are deciding on several different types of motors that have the specs that we need to run the robot, we will pick the motor based on the size since we would like to keep the robot from being too bulky or heavy. The weight that the size could add can also impact the calculations we do for the torque which would require a smaller and more lightweight motor to decrease the impact that it will have on the weight which would increase the torque required to move the robot.

For our analysis of what specification of the motor we need, we are assuming that the robot will be at most around 100 lbs which translates to 45.3592 kg and is equivalent to a force of gravity of 444.52016 N which can be approximated to 444.52 N but for the calculations, we will not use the approximation. We will also be assuming that we will be using tank treads which leads to a higher coefficient of friction. We were notified that the highest gradient that the robot will encounter is 5% which translates to 2.86°. Running with a high assumption that the coefficient of friction would be around 0.7 due to tank treads dealing with higher friction. The torque required would be changed depending on the size of a gear that is being used to apply the torque, but for the assumptions for the calculations, we will assume that the gear that we will use will be around 6 cm in diameter.

Calculations

F_N= 444.52016 N on flat terrain, $F_f = F_N$(0.7) = 311.164112 N

For the robot to move forward on flat terrain, it will need 311.164112 N of force or more to move forward. If we take into account the size of the gear that is turning for this motion, we can find the amount of torque needed. The amount of torque that is needed to move the robot on flat terrain is:

$\boldsymbol{\tau}$ = 311.164112(0.03) = 9.33492336 N*m

But this is not taking into account the highest amount of force that the robot will encounter. We will have to take into account the highest angle of incline that the robot will encounter which is 2.86° since the robot is only going to encounter a gradient of 5%. Next, we will have to calculate the forces by taking into account the gradient.

F_N= 444.52016 cos(2.86) = 443.966481 N on the expected gradient

F_f=443.966481(0.7) = 310.7765367 N

Now that all of the forces are known, we can calculate the summation of the forces in the vertical and horizontal directions to find the amount of force required to make the robot move in the expected gradient. First, we will calculate the summation of the up and down forces which will result in:

F = -444.52016 + 443.966481 cos(2.86) - 310.7765367 sin(2.86) = -16.61307819 N

This means that in order to have an upward motion, the robot needs to have an additional force of 16.613N or more in the up and down direction. Next, we will calculate the summation of the forward and backward forces.

F = -310.7765367 cos(2.86) - 443.966481 sin(2.86) = -332.5414581 N

In order for the robot to have forward motion, it will need a force of 332.54 N or greater. Now that we have both forces required for motion, we can calculate the total force required in order to move the robot forward and upward in the expected gradient.

$F^2 = 16.61307819^2+332.5414581^2$ = 110859.8157

F = 332.9561769 N

THe full force required for the robot to move in the expected gradient is 332.9561769N or more. Taking into account the size of the gear that will be utilizing this force, we find that the torque required to move the robot is:

$\boldsymbol{\tau}$ = 332.9561769(0.03) = 9.988685307 N*m

Based on these calculations we can assume that the minimum torque required to move the robot forward is 9.989 N*m which translates to 7.368 lbs*ft. This is generally the torque required if we were using 1 motor to run the robot, but in the case of 2 motors, we can divide the result by 2 since the forces produced by both motors would equal the force required to move the robot. For 2 motors, the torque required is 4.9945 N*m which translates to approximately 6.684 lbs*ft for each motor. Currently, the sponsor would like for our robot to have a speed of 1 to 2 miles per hour. This can be achieved depending on the rpm of the motor and the gear ratio required in order to achieve the much-needed torque. The other thing is that the calculations done above do not take into account other external factors like air friction which leads to the result we have to not being 100% accurate, but will suffice in helping us find a suitable motor to move the robot. The other factor we haven't taken into account is the terrain itself that we will encounter since we haven't been to the location that the robot will be used, so we will run on the assumption that the robot will be going over dirt and the occasional concrete. Since this will be a small-scale prototype, we don't think that it will fully reach 100 lbs and will be fully expected to fully operate under the preconceived conditions that the final model will have to operate in.

After some searching around, we have found a motor that has a high rpm that we can modify to achieve the required torque and runs on 12 volts. The motor is from amazon.com and is an electric bike motor. Table 6: Electric Bicycle Motor includes the specs of the motor that will be utilized. Due to not knowing the efficiency of the motor, the current is an estimation using the power formula. Furthermore, the horsepower was calculated using the formula of electric power divided with 746. The torque was calculated by multiplying the horsepower with 5252 and dividing the result with the rpm to receive a final result in the unit of measurement of ft*lbs.

Electric Bicycle Motor	
Spec	Value and unit
Voltage (DC)	12 V
Speed	2950 rpm
Torque	0.596628346 ft-lbs
Power	0.335120643 hp
Power (electric)	250 W
Current	≥20.833 A

Table 7: Electric Bicycle Motor

The torque in the table of the motor specs is 0.596628346 ft*lbs which is translated to 0.80891942 N*m which is lower than the torque that was required through the calculation made on how much torque was needed to push the robot. With this, we would have to utilize a gear ratio in order to meet the required torque to move the robot. Below is a calculation to find the gear ratio needed in order to get the required torque.

Gear ratio = 4.9945 / 0.80891942 = 6.174286185 ≅ 6.2

With the calculation, we know that we would need a gear ratio of 6.2 in order to obtain the torque required to move the robot. Using the information from the table on the specs of the motor, we can estimate the rpm that the robot is going to output by calculating the effects of the gear ratio on the rpm. Below is a calculation of the output rpm by utilizing the gear ratio formula.

rpm_o= 2950 / 6.2 = 475.8064516 rpm

With the output rpm, we can now calculate the speed of the robot by utilizing the circumference of the output gear multiplied by the output rpm. The diameter of the gear is going to be about 6 cm which means that the circumference of the gear will be 18.8496 cm which translates to 0.188496 m. Below is the calculation to find the speed of the robot.

Speed of the robot = 0.188496 x 475.8064516 = 89.6876129 m/minute = 5381.256774 m/hour = 3.343766248 miles/hour

The speed seen above does exceed the requirement of the prototype needing a speed of 1 to 2 miles per hour. The next calculation that needs to be done is calculating how much of the battery capacity the motors will be utilizing. The battery capacity is generally seen as amp hour or watt hour so the motors will be calculated in the terms of how many amp hours they will utilize.

Energy draw from 1 motor = 20.833 x 1 hour = 20.833 Amp hour (Ah)
When utilizing 2 motors the energy draw = 41.666 Ah

For the motors to be able to run for 10 continuous hours the battery capacity has to be at least 416.66 Ah if we were only taking into account the motors energy draw. With accounting for other components, the battery capacity will have to be larger than 416.66 Ah.

Update: Due to the decrease in the expected budget that we were supposed to receive, the weight and scale of the robot had to be reevaluated. The weight of the robot was reduced to 8 lbs instead of the expected 100 lbs due to the BES not being mounted on the robot and both the weight and size of the battery was reduced. This led to a reduction in the amount of torque required to move the robot. Furthermore, since the robot base was updated to a 4 wheel drive, the requirements of the motor were lowered even further. The motors that we utilized had a torque of 6.4 kg*cm on each motor and since there were 4 motors, that results in a total torque of 25.6 kg*cm.

3.3.4 Battery Research

For the battery, we need a battery that can last 10 hours due to the recommendation set by PEO STRI. There are 2 things that determine how long a battery will last which are the battery capacity and the current running in the circuit.

Battery Capacity: This tells us how much energy is in a battery and is usually seen as either Ah (Amp-hours) or mAh (milliamp-hours). This tells us how many amps a battery can run in an hour which helps us have the formula for battery life. But this also entails that we have to know the current that will run in the circuit.

a. Battery Life (in hours) = Battery Capacity (Ah) / Load Current (A)

Load Current: This tells us how much current is running in the circuit which helps in the calculation of how long the battery will last. To find this, we have to find the current draw from each piece of the circuit and calculate the total current draw from our system. The other thing is that not every item or part just has the information of how many amps it utilizes and instead tells us how many watts the component uses. To find the amp from the wattage of a component need to know how many volts the component uses and we have to use the following formula:

a. Power (Watts) = I (Amps) * V (Voltage)

There are 2 major components of our robot that will draw the most power are the motors and the BES itself. The motors and BES will most likely rarely be running at the same time however we already know the BES uses 12 volts of DC power. We won't come to a conclusion on the robot until we know how heavy it will be because we will need more torque the heavier it is and the bigger the motor the more power it will draw. One of our first thoughts was using a car battery. A car battery is extremely heavy however it has a large enough battery life to last a full 10 hours. One of the drawbacks is that it will have a parallel trade-off with how strong the motor is because of how heavy the battery will be.

To ensure that we have the best possible decision on what battery to use, we will have to calculate how much energy each component will utilize in the span of an hour. To measure this, we just have to check how much current each component uses and specify that it utilizes that many amps in an hour. The final result in calculating the capacity of the battery will be to multiply the amount of energy that the components will utilize by 10 so that we can see the total amount of energy used in 10 hours. This will give us the capacity for the battery which can help us find a lighter-weight battery than the car battery. The main con of a different battery than a car battery is that the other battery will probably not be as heat resistant as the car battery which will lead us to have to find a solution on preventing the battery from overheating. Table 7: Component energy usage and total energy consumption show the amount of energy that each component uses in amp hours. The bottom of the table contains the total energy used and each section is the labeled component with their respective energy uses. The table is accounting the total usage of each component, so if there will be more than 1 of a specific component, then the energy usage of that component will be accounting for the entire amount that we will utilize for that component like for motors, the table will have the energy used by both motors instead of just a single motor.

Component energy usage and total energy consumption		
Components	**Quantity**	**Energy used (Ah)**
Electric Bicycle Motor	2	41.666 Ah
BES	1	20 Ah
Camera	1	0.8 Ah
Microcontroller	1	1.77 Ah
Raspberry pi	1	3 Ah
Lidar Sensor	4	0.44 Ah
Temperature Sensor	1	0.00042 Ah
Total		67.677 Ah

Table 8: Component energy usage and total energy consumption

The total energy utilized in an hour is 67.677 Ah and if we multiply the result by 10, then the total energy used in a 10 hour period is 676.77 Ah. That means that the battery capacity of the robot has to be equal to the total energy used. The battery capacity that was calculated is under the assumption that every component is running the entire duration during the 10-hour period which isn't very probable at occurring, especially from the BES. We estimate that we can have a battery with a lower capacity by about 30 % which will result in a battery capacity of 473.739 Ah. This demonstrates that the robot needs a massive battery in order to achieve an operation time of 10 hours. We currently do not see a feasible method of doing this for an initial prototype so we either need to lower the operating time goal by a couple of hours or use several batteries in parallel in order to reach the proper capacity. The only issue is that the weight of the robot will be greatly impacted by the type of battery used which can lead to having to reevaluate the motors to handle the increase in the weight. For making a battery bank in parallel, the better option for this tends to be using lithium-ion batteries which also pose their own risk but are quite adept at being able to set the cells in parallel to lead to a larger total battery capacity. The problem with lithium-ion batteries is the risk that they expose and the price in order to set up the battery bank. Another solution would be to just use the car battery but install a solar panel on the robot so that the battery can be charged over time while it is in operation which will lead to a higher operating time.

The next thing we need is how we will charge the battery of the robot. If we are utilizing a car battery to operate the robot, then we can use a car battery charger in order to charge the battery. This will help out by allowing us to use a premade charger instead of having to make one ourselves. The main issue with a car battery charger is that it doesn't utilize a port of any kind but instead has two clamps that attach to the two terminals of a car battery as can be seen in Figure 4: 12V/24V Smart car battery charger from ODOMY. The figure shows a car battery charger from amazon that is a 12V/24V smart battery charger that is from the ODOMY store. The figure depicts the port problem of the battery where it will need two terminals to clamp onto instead of having an easy-access USB port. This means that we will have to design the frame of the robot with the charging of the two terminals in mind which leads to either a bigger charging port area for two terminals or a panel that opens up directly to the battery so that it can be charged.

Figure 4: 12V/24V Smart car battery charger from ODOMY

Another type of battery that we can utilize but is more expensive to get the capacity that we need is the lithium ion battery which can easily be set up with several lithium ion cells in parallel in order to increase the capacity of the battery. Most over the shelf charger for this battery has the same issue as the car battery with the two terminals required in order to charge it. A solution to this problem is to wire a charging port ourselves in order to charge the battery utilizing a more user friendly method to charge the robot. Each recharge port has their own methodology of wiring, so this will depend on the type of port that we will utilize which will also affect the amount of voltage and current that goes into the battery. Once a respective charging port has been wired, then we will have to select a charger based on the specs of the battery and the port that we are utilizing. The only issue with wiring our own recharge port for the battery is the precision required in order to ensure that the charger will safely charge the battery. A single mistake in setting this up can either damage the battery or pose a fire hazard if the wiring itself isn't set up correctly. This means that we will have to check multiple times with a multimeter and some testing to ensure that the wiring is done correctly to ensure no danger can occur from the charger of the battery.

Update: Due to the reduction in the budget that we were supposed to receive, the battery requirement of the robot had to be changed. We no longer had to meet the 10 hour battery life of the robot and were able to utilize a lighter weight with a smaller capacity. We decided to use 2 different batteries instead of 1 so that we wouldn't need to utilize a single larger capacity battery and wouldn't need to utilize many DC-DC buck converters. The batteries that were used to power the motor was a 14.8 V Lipo battery that is usually used to power RC Cars which also contained a charger specific for the battery. The battery that was used to power the other electrical components was a 5 V battery pack that had a capacity of 16.75 Ah and was able to use most phone chargers to charge it. With these batteries, the robot was able to attain a battery life of 1 hour and 44 minutes.

3.3.5 Camera

The camera will be used for detecting people utilizing artificial intelligence. This means that the video footage from the camera will be used as the input for the AI to look over each pixel and try to match similarities to a person. There are two main things that will make human detection easier, the frame rate and resolution. The other two factors that will need to be considered is the lens being able to block out sunlight from distorting the image and the field of view for the camera.

Framerate

a. A video is just a series of pictures that gives an illusion of moving to the human eye. The more pictures in a second the more fluid the video will look. When applying AI to a video this means that it will be able to respond quickly to changes in movement. For example, if video footage was 2 frames per second then there would be a delay between what happened in the real world and what the robot thinks is happening by 0.5 seconds.
b. For our situation, we just want to know if any person is present, and if so the robot will not send the fire signal to the BES. Because of this, the FPS of the camera does not need to be extremely high to the millisecond.

Resolution

a. The resolution of an image is basically the pixel density. The more pixels in an image the clearer the image is which means it's easier to see details. By giving a higher resolution image to an AI, it means that there will be more pixels to process and be more accurate at differentiating between a person and another object.
b. For our situation, greater accuracy is ideal with a higher resolution image because we want to be very sure that no person is around.

Trade-Offs

a. Increasing either the frame rate or resolution will mean more pixels need to be processed per second which increases the computing and power consumption. Thus, this will need to be tweaked later on to find the sweet spot for detecting people accurately with the least amount of computing power.
b. Our current prediction is to prioritize the resolution over the frame rate because the accuracy of determining if a human is in an image is much more important than doing so every millisecond. A person would not be able to move that fast so a lower FPS camera would still be able to capture the person.

Lens

a. Some cameras are not suited for outdoors because the sunlight can distort the image. This would be a very big problem for the AI as it would be unable to tell if a person was present which would be a severe issue. A common solution is an ultraviolet filter which is capable of blocking the sun rays and letting the image maintain its accuracy.
b. Another problem to be aware of is dirt and dust which may build up on the camera and cause issues with the lens. Because the robot will be operating outdoors and traversing rough terrain it should be expected to get dirty. There is dirt resistant lens, however, that will only go so far and the best solution might be to have a person clean the camera lens before the robot is sent off. Another solution may be to have the robot capable of cleaning the lens itself by spraying water and wiping around like windshield wipers. This would require additional motors and is not currently planned for the prototype, but may be considered in the future.

Field of view

a. The field of view (FOV) of a camera is basically the area that the camera can see at one time. At the time of this research, we are still considering if we will have multiple cameras or one camera that can be moved in a 360-degree motion using a motor. If multiple cameras are used then we would probably use four and the FOV of each adjacent camera would need to overlap in order to not miss a location.
b. It may also be more accurate to have the Robot rotate in place to have the camera moving to make sure we are not relying on just one camera for a single location. Using this approach would lead to greater accuracy and insurance because we could run a consensus algorithm where if a majority of the cameras deem a location safe it would be considered safe. This is extremely important because if we just rely on one camera for an area and say it does not pick up on a person due to dirt/malfunction/etc it can be dangerous.
c. The downside is by using multiple cameras we are multiplying the processing and power consumption by 4x which could be huge depending on the FPS and resolution. All of these aspects will need to be considered together in order to find the most optimal and reliable way to detect people accounting for camera failure as well.

While investigating cameras we found OAK-D. The OAK-D is an open-source camera that also uses an OAK-SoM, an AI vision system. In Figure 5: Oak-SoM you can see the board layout of the OAK-SoM which utilizes Intel's Myriad X VPU processing unit.

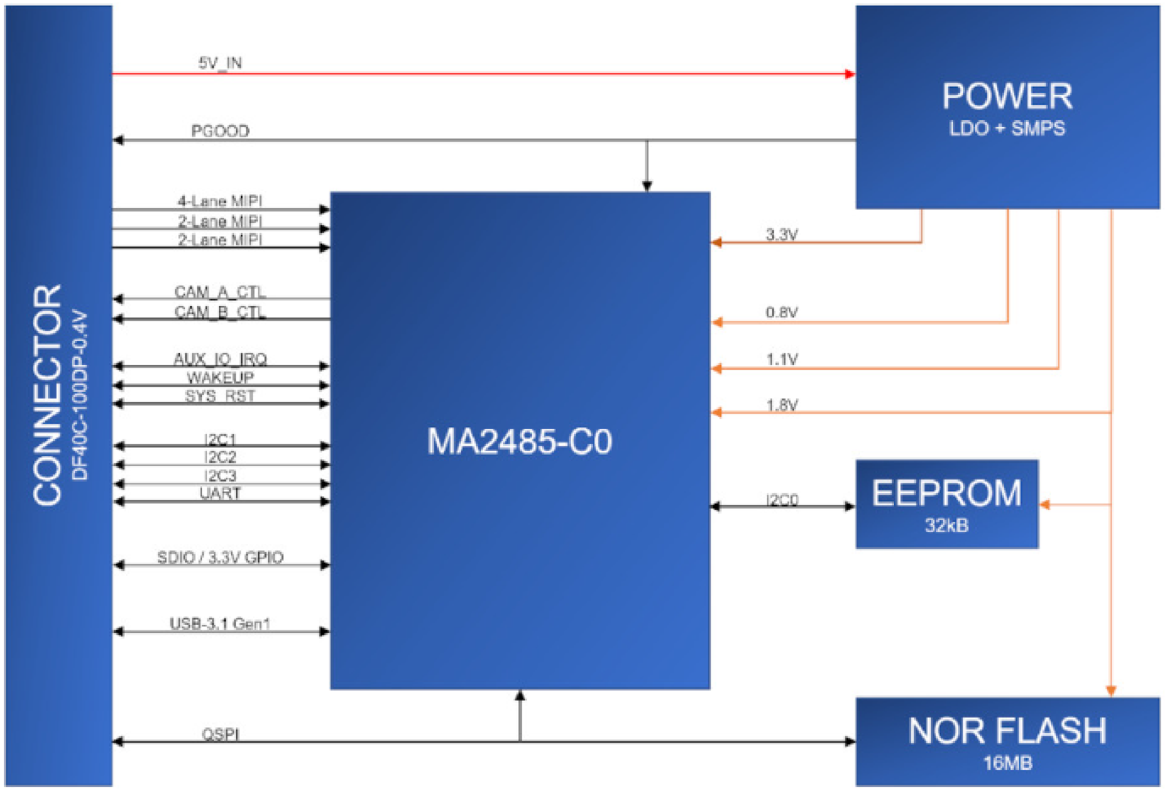


Figure 5: Oak-SoM
OPEN SOURCE

This open-source computer vision camera can run in 4k resolution at 30fps and other resolutions at 60fps. The camera is described as a “spatial AI powerhouse” that can detect anything from masks on a face to pedestrians walking on the street. The camera uses neural networks and stereo cameras for depth. The maximum perceivable depth of the camera is 35 meters. The OAK-D is powered by USB Type-C or a 5 volt, 5.5mm x 2.5mm barrel jack. The maximum power consumption is 7.5 watts and the average is 6 watts.The OAK-D has a cost of $199.99. The price of this product is ultimately fair after researching what it is capable of and the technologies it has on board. Would however need 2 to 4 cameras to make sure the robot is not within range of anything that it could damage/hurt.

In the design process of our robot, we will have to consider the tradeoff of cost and FOV. If we want a larger FOV we will need more cameras. With more cameras comes more cost. One solution to this is simply buying the OAK-SoM and attaching it to cheaper cameras. This would reduce any extra cost that a company may charge for building the entire encasing of the product and heat sink. This would however make the build process take much longer. Another solution that is a bit simpler is making the robot spin in circles when it reaches its destination. This would allow the robot to view a full 360 degrees to determine if the robot's area is safe. The third solution would be purchasing 2 cameras and putting them on a servo motor that can allow them to swivel left and right. I believe the 2nd option is still the best of the three because the first is much more time-consuming and has more room for error. The 3rd option has too much cost and could still have some errors coming from blind spots.

Update: Due to the already high demands of this robot, our limited time frame, and budget cuts, our sponsors recommended that we cut the camera from the project altogether. Implementing a camera that would detect humans using AI would take significant time and our sponsor preferred we focus on other aspects of the project such as the BES communication and the operating system.

3.3.5 GPS

GPS is used in just about every robot that does pathing and moving from point A to point B. Understanding where the robot is in a landscape is crucial for movement. GPS sensors are receivers that communicate with a network of 24 satellites in orbit. The GPS transmits a signal to the satellites and subtracts the time it takes to receive a signal in order to calculate the exact position the GPS is in. With the current build and our choice of pathfinding, GPS won't be a huge role. A GPS sensor is affordable and easy to implement however and we believe it could help us in the future.

1. **GT-U7:** Manufactured by GoouuuTech is a very affordable and high sensitivity GPS module. This board is found on amazon for $11.99 and can be delivered within the week. The supply voltage is 5V and the accuracy is 1-2 meters. The accuracy is too low for our requirements however this would be good to get us in the vicinity and possibly move the rest of the way through the ROS operating system.
2. **NEO-6M:** This GPS is an equivalent GPS system. Manufactured by GoouuuTech this GPS cost $11.99 and has around the same capabilities as GT-U7. The accuracy of this GPS is about 1.5 meters. I believe these are the exact same model just named differently.

While looking for GPS sensors it seems that the cheapest option will be the best option. GPS sensors that have an accuracy of less than a meter are much more expensive. Most GPS don't have any special features as well and therefore makes the decision easy. Buy the cheapest GPS with an accuracy of around 1-2 meters and you will have a decent GPS sensor. We will however be going with the NEO-6M GPS

3.3.6 Angular Level Sensor / IMU

One of the safety checks the robot will need in order to arm and fire the BES is a level check. Gyroscopes are sensors that measure the angle of rotation a physical object has. We do have the option of purchasing an IMU(Inertial Measurement Unit). An IMU is a collection of sensors that have to do with the robot's movements. IMUs usually include a gyroscope, accelerometer, and magnetometer. All of these would be in a 3-axis plane which would make the IMU a 6 or 9-axis IMU. For our robot, a magnetometer is something we do not need, and therefore we would use a 6-axis IMU including only the gyroscope and accelerometer. Accelerometers are sensors that calculate the velocity of an object as it moves. This is not something our robot will necessarily need as it doesn't satisfy any of our requirements but could be something we add in the future.

Hailege L3GD20H 3-Axis High Precision Gyroscope Sensor: This sensor is a 3-axis gyroscope with low power consumption. The board is $7.79 on Amazon and can be delivered within a week. One concern with this board is the definition of precision. I also found that Digikey and Arrow had nothing in stock which is concerning.

We have also looked into Serial input IMU's. These can be a bit more expensive than those that are placed on a PCB however they are simple to hook up and the price isnt significantly different. The 10 DOF is a IMU sensor that cost $36.99 on amazon.com however this sensor has more than enough utilities that would work great with our robot. Another option is MPU 6050 which is significantly cheaper at $12.99 for 5 and $6.00 for 1 on amazon.com.

Update: After further research, our team decided to use the BNO055 IMU. This was due to the fact that there was ample documentation discussing how to connect the IMU, Raspberry Pi, and ROS all together. Our main goal was for the IMU to send a signal to the GUI when the robot was level as a safety check. This IMU allowed us to do that most easily. This was also one of the cheapest options for an IMU and it could be delivered within 48 hours to start development with.

3.3.7 Temperature Sensor

Temperature sensors are pretty self-explanatory. The robot's internals will most likely be hitting temperatures greater than 160°F. These are extreme temperatures that will need to be monitored in order to avoid damage to the components. Most components we have listed are capable of running in these temperatures however they are generating heat themselves and will most definitely malfunction when adding environmental temperatures. We have design functions that will help keep the temperatures of the robot down however we still need something to monitor the temperature. We plan on using a cooling system that utilizes fans and possibly radiators. A temperature sensor will help the robot understand and adjust the fan speed when temperatures are rising far above an acceptable threshold similar to a PC. There are quite a few temperature sensors to choose from

Digital Temperature Sensors

a. These sensors are leading in accuracy with some of the lowest being at 0.1°C
b. They respond quickly to temperature changes with fast conversion times
c. Fit in tight spaces to help get reading all over the robot.

RTD Sensor: These sensors find the change in temperature using resistance.

a. Quite accurate with the most accurate being platinum RTDs
b. Very inexpensive
c. Can be placed in difficult areas to measure temperatures

Thermistors: These Temperature sensors also use resistive properties of materials in order to measure the temperature.

a. Mostly used in the medical field
b. These sensors are a bit more expensive than the previous two.

Thermocouples: This sensor attaches 2 dissimilar metals and measures the temperature at the junction of the two. A small voltage is measured and a control system interprets the findings. These are usually larger and we believe the other 3 options are more functional than this option.
Thermopile IR: This sensor is designed to find the temperature of objects at a distance with infrared. The sensor we are looking for will be going inside of our robot's compartment and therefore this sensor already would not be a good fit.

Now that we have gone over some of the types of sensors let's look at specific components. Assuming we will only need one temperature sensor to understand the component housing area, Table 8: Temperature Sensor displays some good options.

Temperature Sensor				
Number	Name	Type	Accuracy (°C)	Cost (US dollars)
1	G-NICO-018	Digital	+/- .1	$4.09
2	G-NIMO-003	Digital	+/- .2	$3.70
3	NB-PTCO-058	RTD	N/A	$2.31
4	NB-PTCO-029	RTD	N/A	$1.86
5	GA30K5A1IA	Thermistor	+/- .1	$8.43

Table 9: Temperature Sensor

As you can see in the table above the RTDs are the cheapest sensors we can buy. We were unable to find any accuracy for the RTD sensor, unfortunately. Therefore the next best option would be a digital component. Not only can we put it on a board it is the second cheapest and we have an accuracy rating for the component.

Update: Due to requirement cuts we removed the temperature sensor from our robot. Our sponsors advised us to focus on two of the main functionalities that they wanted to see in the robot and remove anything that isn't involved in those areas. Therefore this temperature sensor was never utilized.

3.3.8 Microcontrollers

Microcontrollers are one of the brains of the robot. There are countless microcontrollers that you can choose from. So how do you know which controller would be right for the robot? While it is not as simple as choosing something that looks cool or has the most versatility it can be broken down into a few steps. If you choose a microcontroller that has a plethora of applications you may be spending too much and waste a lot of capabilities that a microcontroller has. When choosing microcontrollers think of the components and software you will be using in your project. For starters, let us look at operating systems. We will be using ROS and although there is mirco-ROS we will most likely be using ROS 2 which microcontrollers obviously can't run. We will need a mini pc such as a raspberry pi that can operate ROS system and pair it with a microcontroller. Operating ROS is something that takes quite a bit to run and therefore we want to buy a commercial board to run it. The microcontroller will be controlling motors and powering different systems when needed. The microcontroller will be taking care of much easier tasks than running ROS. Some microcontrollers we believe will be good options have an ARM (Cortex-Series) and 64-bit Intel processors and many arduino controller boards. These microcontrollers are much easier to program than that of TI microcontrollers. TI microcontrollers often have more capabilities but need every register to be programmed.

The main operations our microcontroller will handle are taking in data readings from the range sensors we choose and turning the value input to output in meters. It will be reading multiple sensors at the same time so it will need to be able to process quite a bit and fast. It will also be reading in from a level sensor to determine the degree of orientation of the robot.

1. **MSP430FR6989:** This microcontroller is a familiar microcontroller that we have used in multiple of our previous classes. It is manufactured by Texas Instruments. The microcontroller run 16MHz and has 2KB of RAM, 128kB of flash, and a 5V input. This would be a very cost-effective option however it would increase the amount of time our robot would take to build. TI microcontrollers are a familiar component to work with however the drawbacks of taking too much time may hurt us significantly.
2. **ATSAMS70J21B-AN:** This microcontroller is manufactured by Microchip Technology and is a much greater powerhouse than the previous. This microcontroller runs 300MHz with a 2048kB flash, 384kB SRAM, and a maximum 3.6 supply voltage.
3. **Arduino Mega 2560:** This microcontroller has an overwhelming amount of inputs you can utilize. Because our robot will have an array of sensors and motors that will need to be controlled, it might be best to obtain something like this. It uses an ATmega2560 chip. This microcontroller has 8KB of ram and an input voltage of 5V.

To operate everything we want, the arduino Mega 2560 is most likely our best option. The board is also barely more expensive than that of its smaller version of the Arduino UNO. We can also get this board within a week's time straight from amazon. The board will work perfectly with a raspberry pi or ordroid that is running ROS using software packages.

3.3.9 Minicomputers

As discussed above because ROS will be CPU intensive we will need a microprocessor in order to run it. In addition, because we will also use image processing we need to account for a microprocessor that can do GPU tasks in order to detect people in real-time. Unfortunately at this time due to shipping delays we may need to pay a premium to get the microprocessor. Most of the microprocessors below have two options with 2GB or 4GB of RAM. We will want the 4GB of RAM because ROS and computer vision tasks will be taking place at the same time potentially causing a bottleneck on the 2GB. Another factor that will change what microprocessor we select is whether we go with a camera that does the person detection or if we only use the camera as a video source to the microprocessor which then processes the image.

1. **Raspberry PI 4:** The raspberry PI has 4 USB ports and a gigabit ethernet port, it also has Bluetooth 5.0 which is critical. It has a 1.5 GHz quad-core Broadcom BCM2711B0 CPU and 4GB of RAM. The MSRP is $55, however, the price in today's market is about $150+. This is the cheapest option and it is capable of running ROS but will struggle with GPU-related tasks like detecting people. This option is only viable if we go with cameras that do the video processing.
2. **Odriod XU4:** The Odriod XU4 is the main competitor of the Raspberry pi 4. It has Samsung Exynos5422 Cortex™-A15 2GHz and Cortex™-A7 Octa core CPUs. It also has 2GB LPDDR3 RAM and 2 USB 3.0 hosts and 1 USB 2.0 host. It has gigabit ethernet, optional Wifi, Bluetooth, and an HDMI 1.4a for display. The power consumption is 5v/4a. This option starts at $79 on Amazon.
3. **Jetson Nano:** The nano has a better GPU than the raspberry pi and roughly the same CPU processing power. It also has 4GB of RAM which means it has roughly the same performance on ROS and it will be able to support at least one camera for people detection. It will not be able to keep up with four cameras and be able to process all of the video streams at a reasonable FPS (15-30). The Jetson nanos MSRP is $99, however, the price in today's market is about $200.
4. **Jetson Xaiver:** The main difference between the nano and xavier is the improvements to CPU, GPU, and RAM. With the addition of much more power, there are now heatsinks on the microprocessor. The RAM is 16GB which is more than enough for our task and the GPU is strong enough to handle people detection on four cameras at the same time. The Jetson xavier MSRP is $400, however, the price in today's market is about $800. This option is expensive, however, it would cut the cost down on cameras because they would not need to do any processing.

   Note that the raspberry pi would fall under 5 fps for people detection because of its low GPU power. For our task of people detection it can be noted that the Xaiver is four times as expensive as the nano but it performs fourteen times better.

3.4 Software Research

3.4.1 Programming Language Choice

Deciding on a programming language is a very important decision. Different languages excel at different things, and you want to pick the best fit for the scenario. The main languages we were considering are C, C++, Python, and Java. The factors for this decision are:

Modularity

a. We need to be able to break up the software into multiple components in order to troubleshoot and make changes seamlessly
b. The object-oriented languages (C++, Java, and Python) are much easier to keep modular due to classes and objects

Speed

a. Because the robot is running in real-time and it needs to constantly make decisions regarding pathfinding it needs to execute instructions quickly
b. Python is much slower than the other languages because it is an interpreted language while the others are compiled
c. C++ and C are about the same speed and allow for deeper control when it comes to memory allocation, pointers, and garbage collection compared to Java

Scalability

a. The program should be able to scale well and complete multiple tasks simultaneously
b. Python does not scale as well compared to the other languages due to the lackluster multithreading and decrease in speed
c. Java and C++ are the clear winners here because they are capable of handling long and complex programs with different classes

Compatibility - The programming languages need to be compatible with the microcontroller and interface

a. C is compatible with pretty much all microcontrollers and it is possible to work with ROS
b. Java is not supported by ROS which would require us to choose a different interface
c. Python and C++ are officially supported by ROS which would mean there is lots of documentation available for our use case

Open Source

a. The software we use must be open source as a general requirement
b. All of the languages are open source

Libraries

a. We want to make use of tools already provided to reduce development time
b. When it comes to machine learning Python has the best libraries by far

Accessibility

a. We want our mentors to understand the language and not pick something obscure. We also want to apply this language to our future jobs.
b. Python is less strict on syntax which can help speed up development time and test out new things

We were able to rule out C language because it is too barebones compared to the object-oriented languages. Using C would also make the scalability and modularity much more difficult, so the slight increase in performance would not be worth it. We ruled out Java because it is not compatible with our interface ROS.

This left us with C++ and Python, and we had a difficult time narrowing it down to one of them. Because C++ would be needed for lower-level tasks like movement and sensors as a compiled programming language is much faster than an interpreted language like Python. However, all of our team is new to C++ so it would be a steeper learning curve compared to the other languages. When it comes to machine learning and other higher-level tasks, Python has a much better framework and libraries.

Our conclusion is that we should use both C++ and Python for different tasks and what they excel at. We may also do the early implementations of the program in Python, and if there is a need for more performance we will optimize in C++ for the final product.

Update: We ended up using C++ for communication, Python for ROS, and Java for the GUI.

3.4.2 Operating System

There are many factors that must be taken into consideration when deciding on an operating system for the robot. A requirement that our sponsor gave the team was that the robotic platform must align with the Robotic Operating System Military (ROS-M) framework and ecosystem. Because of these requirements, I decided to conduct research on the Robotic Operating System (ROS). First, I looked into why ROS would make a good operating system for an autonomous robot. When performing my research, I looked at factors such as financial cost, and cost of time/ease of use.

Financial Cost

a. The Robotic Operating System (ROS) is an open-source framework, so it is free for public use. Many of the packages are released under a BSD license. This allows you to use and modify code for commercial purposes and not have to release it. We do have a substantial budget for this project, but due to all of the requirements, saving money for other factors of the robot, such as parts, is what our group decided was the most financially responsible decision.

Ease of Use

a. It was important that our group use an operating system that was user-friendly and easy to pick up since we have a strict deadline for this project and many other components of the project to work on as well. ROS is extremely well known and relied on in the robot industry. It has plenty of documentation and resources for someone who has never used it before. Most code for ROS is written with C++ or Python, which are common languages that we can find resources on if we run into any issues. Those are also languages that this senior design group is comfortable with.

b. ROS also has many packages that are free for public use. For example, ROS already has code packages for mapping rough terrain. The team can take these code packages and edit them to fit our robot's needs.

c. ROS is supported and tested on Linux, Windows, and macOS. This will allow the team to choose which environment we would like to develop and deploy the robot in, avoiding having to learn how to do this in a new environment.

After conducting this initial research, it was clear ROS would function well as an operating system for our robot due to its features, resources available, and it is open source. I then looked into the different versions of ROS, ROS 1 vs ROS 2, and which would be the most optimal solution for an operating system for this project.

ROS 1

a. **About ROS 1**

   ROS 1 is much older than ROS 2. It was released in 2007. ROS 1 has many mature features which can allow you to move quickly when creating your robot. Many features and packages that have been invented, such as the ability to map terrain or the ability to be autonomous, were created using ROS 1. But, because ROS 1 has been around for much longer, the developers and users of it have had the time to discuss what features they like, don't like, and what needs to be added. Because the needs for a robotic operating system are constantly changing as technology evolves, implementing all the features and changes that are wanted/needed would results in many breaks to ROS 1, so ROS 1 did not get a lot of the new updates that are wanted/needed, instead, these were taken into consideration when developing ROS 2. ROS 1 being older though will mean that eventually down the line the operating system will be outdated and the robot will need to be upgraded to ROS 2 anyway. The last released ROS 1 distribution was May 23rd, 2020, and this distribution has an end-of-life date of May 2025. This distribution is really for those who have already been developing in ROS 1 and just need to finish the project they are working on. There will be no more ROS 1 after 2025.

b. **Writing Code in ROS 1**

   i. **APIs**: ROS 1 allows you to use C++ or Python for your APIs. For C++ you would use the roscpp library and for Python you would use the rospy library. Both of these libraries are independent and used on top of tcp and udp. The APIs and their features will vary between the rospy and roscpp. This can cause issues because you may be using the roscpp library and need a python feature but are not able to use it because that feature is strictly for rospy. The APIs between rospy and roscpp will not be very similar to each other.

   ii. **Language:** ROS 1 only has one distribution that will support Python 3, and that is only until the end-of-life date May 2025. ROS 1 targets C++ 98. C++ 11 and 14 are available in some later ROS 1 distributions, provided that it didn't break any dependencies.

iii. **Nodes:** There is no specific structure that must be used when writing node functionalities. There is no clear convention for writing them.

iv. **Components:** This does not exist in ROS 1; it is new in ROS 2.

v. **Launch Files:** Launch files allow you to start your nodes all from one file Launch files are written with xml. There is a way to write them in python, but it is not popular and there is barely any documentation on it.

c. **ROS 1 Communications**

i. **ROS Master:** ROS 1 must-have ROS master run before the nodes are run. This acts as a server for the nodes so they can retrieve information and communicate. All of the nodes are dependent on the master machine, ROS Master.

ii. **Parameters:** The parameter server is inside the ROS Master.

iii. **Services:** The servers are synchronous.

ROS 2

a. **About ROS 2**

ROS 2 is much newer than ROS 1 and is considered the future of robotics. ROS 2 will take more time to learn most likely since it is newer and has fewer resources on it available. The majority of packages made for ROS were also NOT created for ROS 1. ROS 2 releases new distributions every year, and it is releasing new LTS versions every two years. Unlike ROS 1, these distributions will continue to be released. ROS 2 was based on ROS 1, so for those who are already familiar with ROS 1, ROS 2 will not be very challenging to learn.

b. **Writing Code in ROS 2**

i. **APIs:** ROS 2 APIs have one base library- rcl. This library is implemented in C language and has all of the call features for ROS 2 in it. Although you can implement this C code for your APIs, you can also use another client library such as rclpy, which is python, or rclcpp, which is C++. This version is convenient because any new features that are added/implemented just need to be done in the base rcl library, and then the client libraries (rclpy and rclcpp) will have access to it by providing the binding between the library and rcl. The APIs between rclpy and rclcpp and other client libraries will all be extremely similar.

ii. **Language:** ROS 2 provides many distributions that will support Python 3. ROS 2 also now supports C++ 11 and 14 by default. C++ 17 is also going to be available for use soon for a quicker and safer development process.

iii. **Nodes:** In ROS 2, there is a convention on how to write nodes. You must create a class to inherit node objects (same for python and c++). All of the ROS 2 node functionalities will be implemented in the specific classes. This will save time by providing clean code, an easy-to-follow structure, and easier collaboration with developers. Using this object-oriented programming allows for easier conversion to components.

    iv. **Components:** Many nodes can be handled from the same executable using components. It is a modified node class. This will create a much more efficient program.
    v. **Launch Files:** Launch files allow you to start your nodes all from one file. Launch files are written with python, they are much more customizable and modular. You can use xml if that is preferable.

c. **ROS 2 Communications**
    i. **ROS Master:** The nodes in ROS 2 do not use ROS Master. The nodes have the capacity to discover other nodes. This allows for a fully distributed system. The nodes are all independent.
    ii. **Parameters:** There are no global parameters, the parameters are specific to the nodes. Each node pretty much has its own parameter server.
    iii. **Services:** The servers are asynchronous, but they can be synchronous if needed.

It would be ideal that the team uses ROS 2. The goal for this project is for it to continue with other senior design teams after we have completed our part. Because of this, we want to use the most updated operating system possible to keep the robot modern for the next team. Using ROS 1 will eventually result in a future team needing to fully transition the robot over the ROS 2. This will cause a lot of extra work and issues. It is clear that ROS 2 also includes features, such as executables and the standard way or programming nodes, that would add to the ease of use when developing in ROS. To confirm that we will be using ROS 2, our team needs to ensure that the packages needed to create our robot are available. Below, we conducted research on the packages available in ROS and which would be useful for our robot.

ROS Navigation Package

a. It is vital that the robot for this project be autonomous. In order for this to happen, the robot must navigate successfully from point A to B without collision. Figure 6: Robot

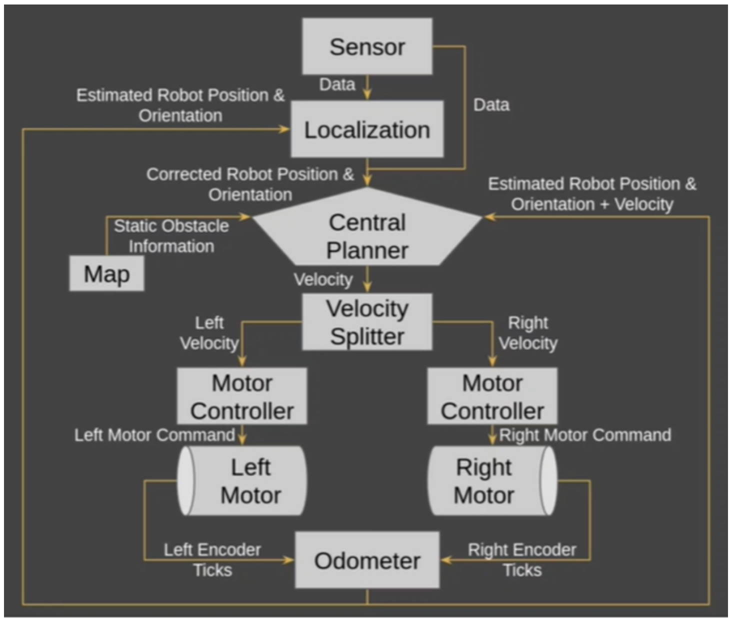


Figure 6: Robot Navigation

Navigation from ROS.org is showing the key components to make a robot navigate autonomously. The sensor gathers information about obstacles in the environment. This information is then passed to the localization and central planner. The localization has an estimated robot position/orientation on the map. This location is made using odometry and is then passed to the central planner. The map also provides additional information on the obstacles in the environment the robot is running in. After this information is collected, the central planner finds the value that is needed for the velocity of the robot based on the current velocity of the robot and the position of the robot on the map. After this velocity is calculated, it is then passed to the velocity splitter. The velocity splitter computes individual wheel velocities that are then passed to the motor controllers. This is then converted to motor commands that are passed on to the left and right motors to control the wheel accordingly. This also provides information to the odometer to estimate the location of the wheels and the location/orientation of the robot on the map.

b. ROS has what is called the ROS 2 Navigation Stack. This stack is a collection of software packages to help a robot accurately move from point A to point B. This stack utilizes mapping path planning and localizing to make this happen. This navigation stack uses what is called behavior trees that call modular servers to complete a given action. According to the ROS navigation site, these actions can include computing a path, control effort, recovery, or anything similar to that. Figure 7: ROS Stack from navigation.ros.org shows the structure of how this navigation stack works.

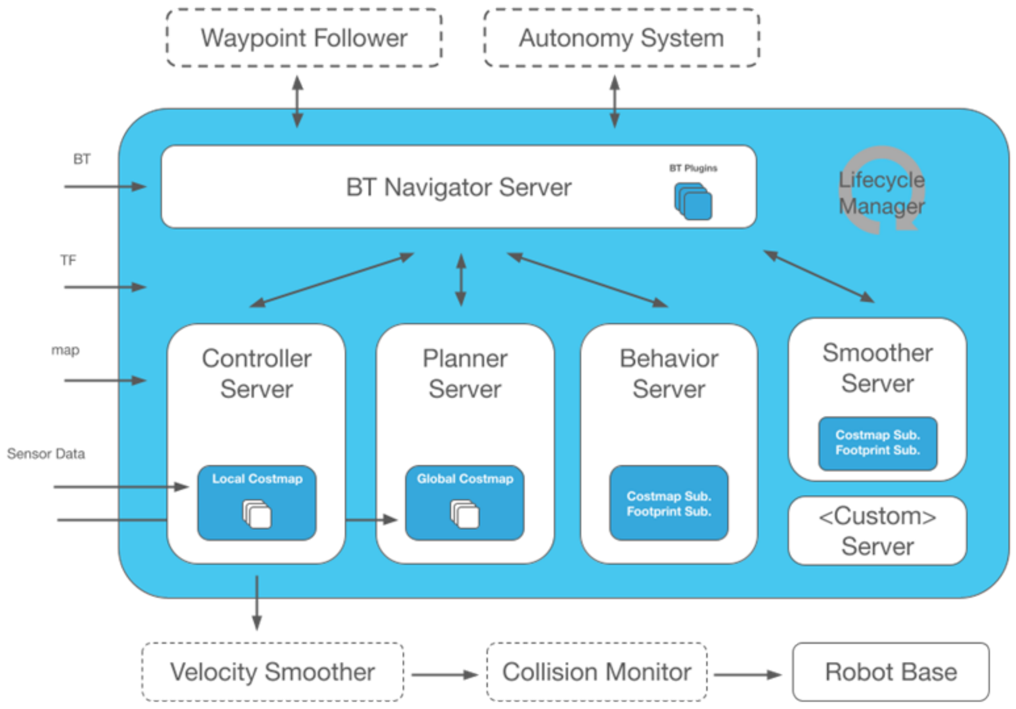


Figure 7: ROS Stack

The ROS navigation stack also uses sensor information as well. This will help the robot avoid obstacles and navigate through rough terrain, which is very important for a robot that will be on training fields. This stack also allows for a map to be used, but it is not required. If one chooses to import a map of the environment the robot will be running in, you are then able to set an initial start spot for your robot, and end goal destinations. This stack also has cost map configuration. There are two kinds of cost mapping it allows: global and local. Global cost mapping can perform functions such as finding the shortest path from point A to point B, which is a requirement that our sponsor mentioned. A local cost map can be used to optimize paths by avoiding obstacles and collisions. Finally, this stack requires the use of Adaptive Monte Carlo Localization (AMCL). This will keep track of where the robot is on a given map. It utilizes the input map, LIDAR scans, and then transforms messages. Figure 8: ROS Stack below shows the ROS Navigation stack setup. This diagram comes from ROS.org. To autonomously navigate, a map is needed. This is provided by the map_server node. Once we have a map, we must know the location of the robot on the map. This is done by the amcl node. It uses sensor data and odometry information to localize the robot on the map. The heart of the stack is the move_base. This generates the path to the position the robot needs to arrive at and detects obstacles

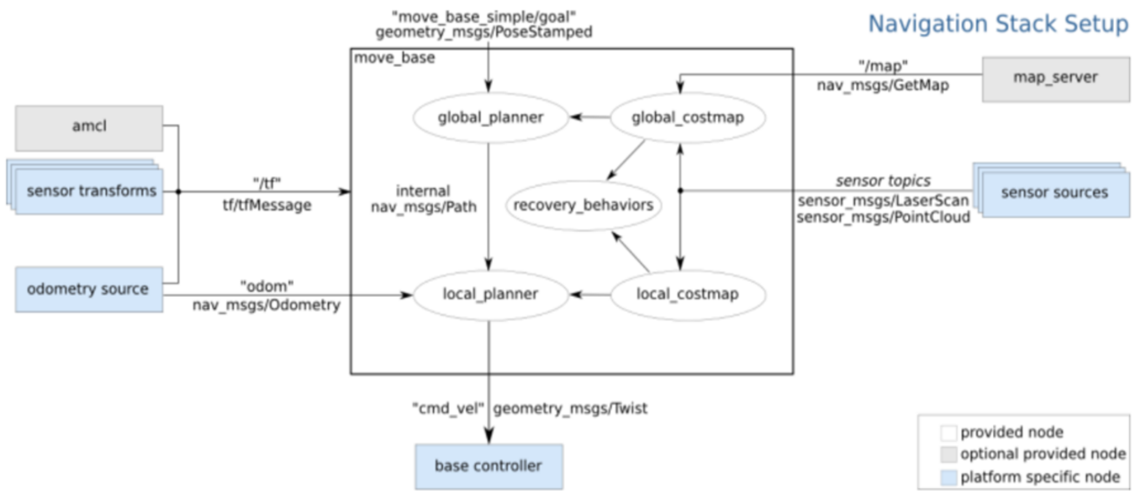


Figure 8: ROS Stack

using global and local planners. This computes the robot's linear and angular velocities required to follow the planned path and sends that information to the base controller. The base controller then converts robot velocities received to individual wheel velocity commands for straight and accurate path navigation. The odometry source takes feedback from the wheels and motor and provides the robot's current position to the move base. The white and gray components in the diagram are already implemented in the stack, they just need to configure the parameters of the nodes according to the robot. The blue component nodes are robot dependent and must be written. This package does have some limitations. It performs best on square or circular robots. It may have difficulty on large, rectangular robots. It also requires that the robot have a planar laser planted on the base of it.

After conducting this research on the ROS Navigation Stack, we have come to the conclusion that this would be a great stack to use for this project. It aligns with the ROS-M framework and ecosystem like our sponsor requested, and it provides us with a great starting framework for our robot to run autonomously and have optimal terrain mapping, both of which are requirements for this project.

Update: The team decided to use the ROS 1 framework. ROS 1 has been around much longer than ROS 2 so there is ample documentation regarding it. ROS was a much larger learning curve for the team than expected, so we decided it would be a better use of time to utilize an operating system where we can actively search for answers and find an abundance of solutions to try. ROS 1 also had established packages for many parts we were using such as the BNO055 and RPLIDAR. The team did use the navigation stack as planned.

3.4.3 Issues with AI

By using the pictures from the video footage as input the robot will be able to process each pixel and use machine learning to determine if a person is present. This is a critical process because people's safety relies on this so we need to prioritize accuracy over speed for the detection process.

Constraints

a. One of the main constraints for the AI will have to keep up with the framerate of the camera. This means that we want the runtime of scanning over every pixel to be less than 1 divided by the framerate. If this condition is not met then the AI will be using outdated images and the amount of time it is outdated will scale with runtime. This will be difficult to maintain depending on the number of cameras we use with the expected amount being

Overfitting

a. Overfitting is a problem that occurs when training an AI. Basically, the AI will pick up on unnecessary details or "noise" and learn bad habits from the data. An example would be if we were training the AI to detect people and we only used pictures of people on crosswalks. The AI would pick up on things like stop signs, traffic lights, and road markings which is not what we want it to be recognizing. If this AI was to look at a picture with a person at the beach it might be able to detect the person because it was only trained in a different setting.
b. In other words, it can be thought of when the AI is very good at handling the images given in training but it does not perform well on images not used in training. Basically, we want the AI to think for itself in these new situations instead of over-relying on the training data and not being able to recognize new situations.
c. The key to solving this problem will be to train the AI of all sorts of different people in different locations to minimize the noise and background coincidences that the AI would pick up on.

Underfitting

a. This is the opposite of above where the AI does not perform well in training or when given images outside of training. Obviously, this is not desirable and it is much easier to observe this problem because it will not be working in training.
b. This problem occurs because the AI is unable to understand the relationship/context, meaning it does not recognize the patterns. If this is happening it may be better to use images more closely related so the AI can start to pick up the pattern.

False Negatives

a. For our situation, a false negative is when the AI thinks a human is not detected when someone is present. This is extremely dangerous because the robot will proceed with the launch sequence because it thinks it is safe to launch when it should not be launching.

False Positives

a. For our situation, a false positive is when the AI thinks a human is detected when no one is present. This is not dangerous and is more of an annoyance to the operators of the robot. The robot will be programmed to be more susceptible to false positives because of the risk factor involving humans getting harmed by the robot. The exact values will be tweaked later, however, if the robot is in doubt if a human is present or not we will err on the side of caution and not send the signal to launch.

Update: Due to the already high demands of this robot, our limited time frame, and budget cuts, our sponsors recommended that we cut the camera from the project altogether. Implementing a camera that would detect humans using AI would take significant time and our sponsor preferred we focus on other aspects of the project such as the BES communication and the operating system.

3.4.4 Neural Networks

Neural networks are like a human brain where there are many nodes that can process information and communicate with each other. There are many types of ways to configure the neural network of the AI and each has advantages in certain situations and tasks.

Artificial Neural Network (also called Feedforward)

a. This is a type of neural network where the information only moves forward meaning it travels in one direction. It is one of the simplest neural networks because there are no loops or any way to send information back up to nodes on a previous level.
b. The way it works is by using weighted values and it basically adjusts the weight on each path through trial and error. Trial and error are also known as the machine learning part where it will find the optimal values to accomplish the task.
c. Because of the simplicity, there are multiple limitations like nodes not being able to communicate with other nodes on the same level or previous level.

Recurrent Neural Network

a. This is very similar to the Artificial Neural Network with the difference being that information can now flow backward to nodes on a previous level. What this means is that feedback is basically thrown backward which allows for more complexity and accuracy in doing a task.
b. This type of network basically has a memory and is aware of what is going on so it would be suited for a task like a prediction. For example, whenever you are typing a message on

your phone there will be auto-filled words at the bottom and this is where an RNN would excel at predicting and recommending the word it thinks you want.

c. A downside is that it's harder to train Recurrent Neural Networks because you don't just have to look at data from previous nodes but also account for data from nodes that will send info back up.

Convolutional Neural Network

a. This network is very different from the above because it does not use weights and it has convolutional layers which use filters. The filters allow it to excel at detecting edges, circles, and squares. That information would then be sent to the next layer which would use the already detected primitive/geometric stuff like edges and circles to detect more complicated things like the eyes of a person. Basically, by using the filters the AI is able to build up and be great at object detection by being able to recognize patterns.
b. This network is classified as deep learning which is different from machine learning (ANN/RNN) because it is more self-sufficient and is able to learn on its own with less direction from a human. Normally in machine learning, a human would help with assigning the weights but that is not present in this type of learning.
c. A downside is that more advanced hardware is required for this type of learning meaning we will need powerful GPUs to train the AI. The reason GPUs are used is that they allow lots of parallel computations to take place which is needed for the various different routes an AI will go when learning.
d. This type of network is more powerful than the others which makes it more suited for image classification tasks. Because our task is to basically look over an image and detect if there is any person, we believe the CNN is more suited and will be our chosen neural network.

3.4.5 Edge Detection

Edge detection is the process of looking through an image for an area where there is a big difference between pixels. Imagine the letter “A” in a black font printed on a white piece of paper, edge detection on this image would recognize that the pixels are different around the A. This is the basic idea and for images taken from the real world, the color difference of adjacent pixels will be slightly different adding on to the complexity.

Image Behavior

a. The key to understanding how pictures work is to think of them as two dimensions. There is the horizontal dimension which is the location of the pixel, and there is the vertical dimension which is the color of the pixel. A darker color pixel means it is farther on the vertical axis compared to a lighter color pixel. Below is Figure 9: Gradient Intensity Example.

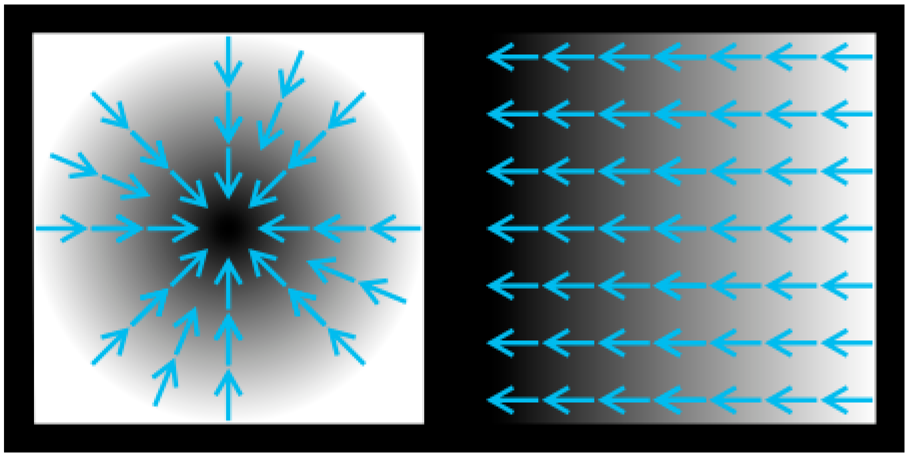
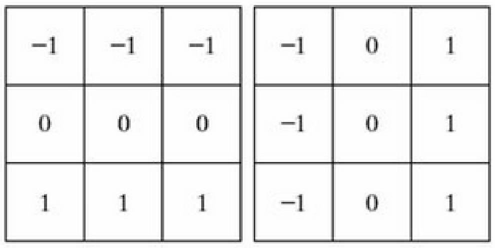


Figure 9: Gradient Intensity Example

Simple Edge Detectors

a. Edge detectors will use gradients in order to take an input (the picture) and produce the output which is the picture with edges outlined. Gradients are basically the rate of change, in the context of a picture you want to get the horizontal and vertical gradient.
b. All of the following edge detectors will use kernels which are basically a matrix. Convolution is performed on the image and the kernels for the x axis and the y axis, and the result is being able to see the edges of the image through the Gx and Gy components. Gx is the horizontal component of the gradient and Gy is the vertical component and you can calculate the magnitude using both. The result will be different depending on the kernel used, however, these edge detectors are more prone to noise because of their simple nature.
c. Roberts Cross - This edge detector is interesting because it uses 2x2 kernels compared to the 3x3 of the others. The benefit of this is being able to compute the gradient much quicker because fewer calculations need to be done. However, there is a tradeoff where the resulting image will be more susceptible to noise in the image. This is not ideal, especially for our use case where we do not want there to be inaccuracies that can cause the AI to think a human is there when it's not or vice versa.
d. Prewitt Operator - This edge detector uses a 3x3 kernel which means there is more computation time and it's less prone to noise as seen in Table 10: Prewitt Kernel. This could be used as an edge detector effectively on its own, but there are still more optimizations that can be done in order to make it more accurate at the cost of more computation time. This will be talked about later where the Canny Edge Detector can build off of these simple edge detectors by applying more filters and transformations in order to get a much cleaner result.

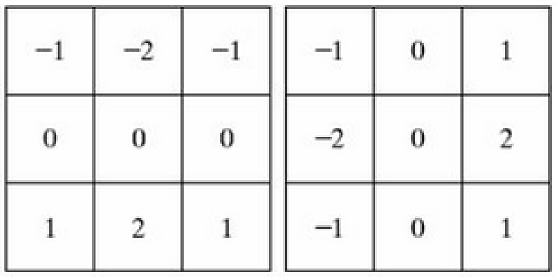


−1	−1	−1
0	0	0
1	1	1

−1	0	1
−1	0	1
−1	0	1

Table 10: Prewitt Kernel

e. Sobel Operator - This edge detector also uses a 3x3 kernel meaning it is very similar to sharing the same benefits and cons as Prewitt. The main difference between the kernels is that the weighted value in the middle can be changed to 2 or greater as can be seen in Table 11: Sobel Kernel. This means that this operator will have darker or more pronounced edges which are usually better for trying to use AI to recognize specific shapes and patterns.



−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

Table 11: Sobel Kernel

Canny Edge Detector

a. The Canny edge detector involves five main steps compared to Sobel, Prewitt, and Roberts which means the runtime of this algorithm is much greater. However, the result of the Canny edge detector produces much smoother edges and is more clear. For our task, we want to be as accurate as possible and we will choose accuracy over runtime so Canny is looking more appealing to be used as an edge detector.
b. The first step in this process is to use a Gaussian filter to remove noise from the image. A Gaussian filter is used to blur an image by making adjacent pixels the same colors which reduces noise in the image as shown in Figure 10: Gaussian Filter Example. This is desirable because when taking a picture from the real world there are often pixels that do not match what they should be representing due to various reasons like lighting, dirt, etc. Here is an example below and you can see the amount of unique colors in an area is reduced.

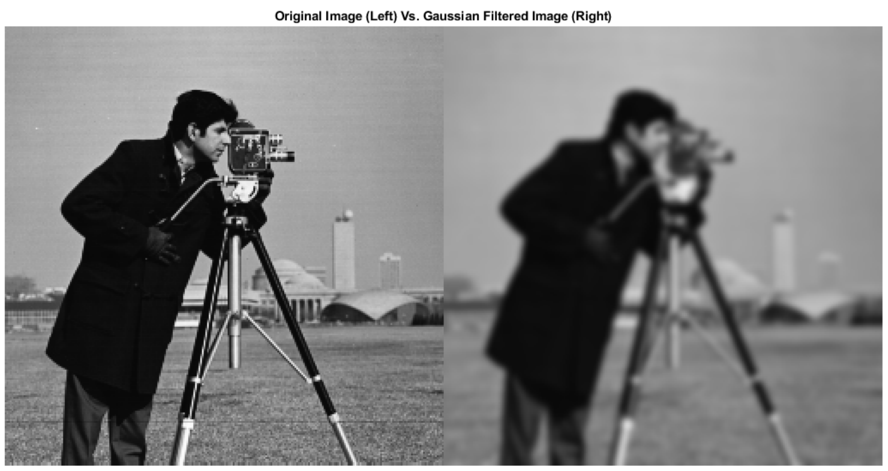


Figure 10: Gaussian Filter Example

c. The next step is to get the intensity gradient of the picture. This can be done using any of the Simple Edge Detectors and normally Sobel is chosen. After this step the edges are visible and you could stop here, but the process of Canny edge detector goes further. One problem with the result of this step is that some edges are thicker than others when ideally they should all be the same, this will be solved in the next step.
d. The next step is Suppression which is a way to basically thin the edges to make them much more similar to a slim line. This is desired because with thin edges it's easier to recognize patterns and simple geometric shapes compared to if the lines have a varied width. The way this step is accomplished is by looking at the adjacent pixels on an edge and it replaces the outer pixels with the background color. This is basically leaving the middle pixels of an edge untouched while removing the outer ones which complete the goal of having thin edges.
e. After the previous step the edges become much thinner, however, there is still the problem that some edges are less visible than other edges that were very thick before the thinning. The reason that problem was caused is generally because of noise/color vibrations which caused the difference in thickness. This is not ideal because the AI would have a harder time detecting patterns when the edges still have a different width/intensity. The solution is the “Double Threshold” step which is basically marking edges pixels as weak or strong based on their intensity or strength. The strong pixels are usually colored in white while the weaker ones are colored in gray. After this step, all of the important edges are about the same intensity and width which will greatly help the AI.
f. The final step in the algorithm is edge tracking by hysteresis which is basically the clean-up step that gets rid of some of the weaker pixels from the previous step. All of the strong pixels will remain in this step because they have already proven to be “essential” to the edge while it's debatable whether some of the weaker ones are needed. The way to

determine whether the pixel should be kept is by comparing the adjacent pixels of a specific weak pixel, and if there is a stronger pixel adjacent it will be kept. When a weaker pixel is decided to be kept it effectively changes color to white. This process is called Binary Large Object Analysis (BLOB-analysis) and by the end hopefully only the important pixels are left. The result is usually very basic having only the important edges around a picture, and these five steps will have suppressed most of the noise.

Conclusion

a. After understanding the common ways to detect edges from an image, we believe for our situation the Canny Edge Detector featuring the Sobel Operator will be best for our task. This is because we are prioritizing accuracy over computation speed in order to have a more reliable robot. Roberts cross suffers too much from noise so the tradeoff of a faster runtime is not worth it. Sobel generally works better with Canny compared to Prewitt with Canny because slightly more edges can be detected due to the additional weight value. The remaining steps in the Canny algorithm will then decide if those pixels are important or not, meaning generally a more accurate image but more processing is needed on the additional weaker pixels.Here is a demonstration of what an image looks like after each of the five steps in the Canny algorithm as shown in Figure 11: Canny Algorithm Example after each step. The final result is an image with only two colors where the background is white and the edges are white. You can actually see the difference between just using the Sobel operator and finishing the remaining 3 steps in the Canny algorithm by comparing the 2nd and 5th images. It can be observed that the person in the background is still visible in the 2nd image, while in the final image that noise is mostly removed. There are still a few edges that should have been removed in the final picture, however, the Canny algorithm is not perfect and some noise will make it through. After applying the canny algorithm the next step would be to look for patterns in the remaining lines and in this case, it would be very easy to recognize the human face through the eyes, nose, ears, and mouth.

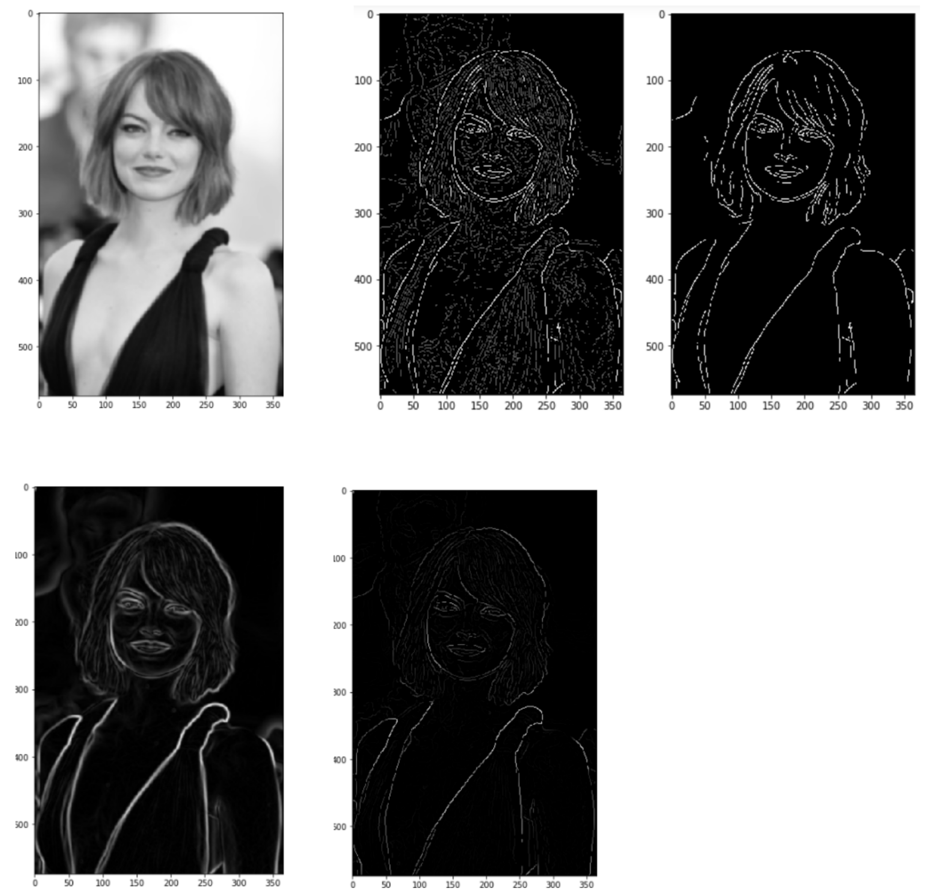


Figure 11: Canny Algorithm Example after each step

3.4.6 Dynamic Host Configuration Protocol (DHCP)

Functionality

a. The way a DHCP connection will work is through a client-server which means that something will act as the server (robot) and it can communicate with multiple clients (BES). The server will also be able to communicate with the network which means that all requests the client makes have to go through the server in order to get to the network. The server is responsible for assigning unique Internet Protocols (IPs) to each client. This is essential in order for the network to differentiate between multiple devices coming from the same server.

b. The main idea between DHCP connections is that when a client connects they are automatically assigned an IP. This is useful because you can think of it as a pool of IPs and when a device connects it takes an IP from that pool and makes it unavailable. This is useful when lots of devices are constantly disconnecting and connecting because IPs will not be wasted on inactive devices.

c. An example is how a router works when it's connected to multiple devices, say a computer and laptop. Each device will be assigned a unique IP automatically which is why multiple devices can be connected to a router. There can be a point after 200 devices where the router can not handle any more devices, however, this will not be a problem for us because we will only have the BES and potentially GPS hooked up as clients.

Alternatives to DHCP

a. The main alternative to DHCP is statically assigning IPs. The main advantage to statically assigning the IP is that you can always know which device the traffic came through by looking at the IP because it will never change even on a new session. The disadvantage to this is that the network operator will have to manually assign and unassign IPs depending on which devices are connecting to the network.

Ways to Connect

a. The two main ways to connect clients to a DHCP server are through ethernet or wireless. Ethernet is usually more reliable and faster because there is less room for interference from other devices and noise compared to when done wirelessly.
b. The main pro to wireless connections is that it is sometimes not feasible to connect a wire over such a long distance. For example, when connecting the robot to the network we will have to use wireless communication because an ethernet cable would get in the way and potentially be run over by the robot. One issue we expect to run into is the interface between connecting to the network and the various sensors we will be using like the rangefinder. We plan to resolve this issue by using an antenna to get a better signal and create some distance between the various components.
c. The BES has an ethernet port so we have chosen that to be the way to connect it to the robot because the distance is so short between the BES and the server. This is also ideal to reduce interference as the more wired connections we make the less noise for connections that are required to be wireless like connecting to the network.

Communication between clients and network

a. The robot will effectively act as a middleman between the main network used by the military and the clients (BES and GPS) as shown in Diagram 1: Communications Diagram. The situation when the BES communication will be needed is when the user presses the launch button on the app. The BES will not receive the signal to launch until the server (robot) has verified the conditions are safe for launch by using the AI to confirm no people are around. Once the BES has finished launching it will send a signal back to the robot and then the robot will send it to the network and user app stating the launch was successful. If conditions are deemed unsafe to launch the robot will send a message back to the network and user app which states that it was unsafe to launch and the reason why.

b. The role of the GPS will be different compared to the BES because we want it to constantly be updating the location of the robot on the network. The network will not need to communicate with the GPS and instead only receive updates on the current location which will be broadcasted in the user app.

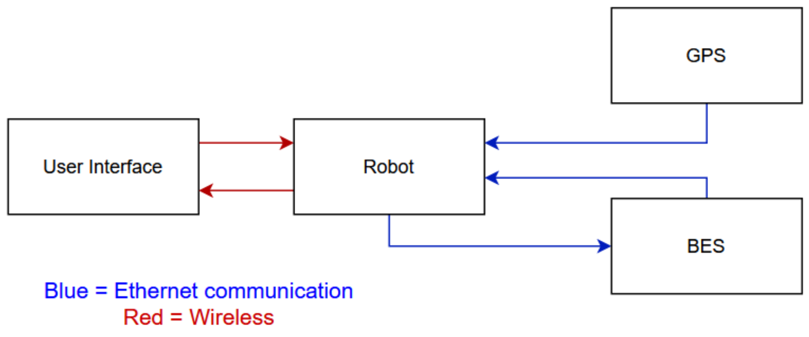


Diagram 1: Communications Diagram

Update: We ended up using a router which assigns the IP to the BES because it was too heavy to fit on the robot. The updated communication diagram looks like:

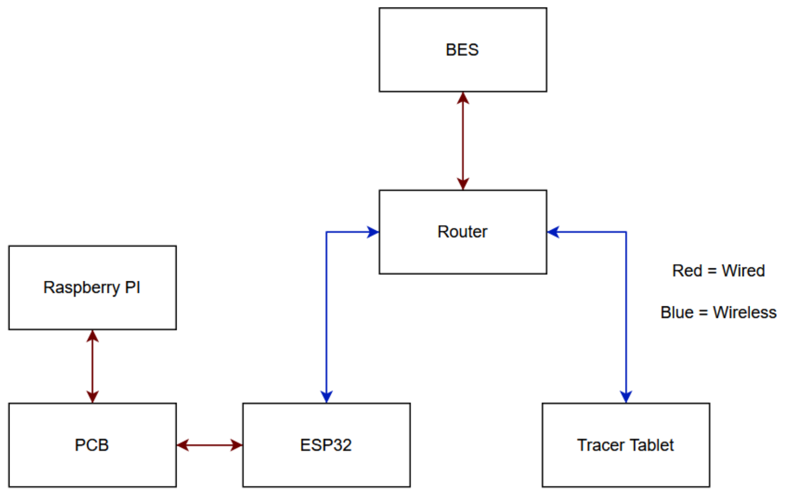


Diagram 2: Communications Diagram Updated

3.4.7 User Interface

Platform Choice

a. One of the first steps for our user interface will be researching which mobile device we will be using to communicate with the robot. The two main choices are Android and IOS devices. The main way for the phone to communicate with the network/robot will be through Bluetooth. There is also the option to have a windows based tablet/laptop. This would make the development easier but it would restrict the ease of getting a new device to work. Ultimately it would be ideal if the app could work on multiple platforms, but for now, we will focus on one.

b. Most modern tablets have Bluetooth; however, there is a big difference between Bluetooth 5.0 and Bluetooth 4.2. Bluetooth 5.0 can reach 800 feet while Bluetooth 4.2 can only reach 200 feet. For reference, a football stadium is 360 feet meaning Bluetooth 5.0 can reach farther than two football stadiums in theory. Although 200 feet would likely be enough for our current scale model, because future groups will be upscaling the robot we will be choosing a device with Bluetooth 5.0 or higher. iPhone 8 and up support Bluetooth 5.0. For Android, it's more dependent on the device but the mainline Samsung Galaxy S8 and up support 5.0.
c. Another factor to consider is the cost to publish an app on the IOS and Google Play store. For IOS, it costs $100 per year compared to a one-time fee of $25 on the Google Playstore. The Google Playstore is the preferred option here because the app will be removed on IOS if the recurrent fee is not paid each year.
d. The computation power of the phone is not important as it will just be sending commands and displaying feedback from the robot. The screen size is important and will factor into how the app is designed. If we go with a tablet more information like the coordinates and sensor information will be able to be shown without scrolling.
e. IOS also has much more security/privacy which can potentially cause issues when creating the app and trying to make a connection with the robot. There are known issues with connecting with non-apple products which can be a big problem for our project. Android would be preferred as it allows users to give permissions to certain apps which would allow us to give the app root permission if needed. This is not possible on IOS without jailbreaking the phone which will cause issues if the phone is updated. A windows based tablet would allow us to build a Java application that doesn't have to deal with any of the platform-specific API which would make development smoother.

Features

a. There will be a connect button that will attempt to pair the mobile device with the robot and if it's successful the output information below will be shown. There will also be a disconnect button which will terminate the connection between the robots.
b. The first feature will be displaying output information from the robot to the user. The most important information would be the coordinates and battery percentage. Then there is less important information like from the level sensor and rangefinders which can be displayed through scrolling.
c. The main ways the user will interact with the Robot is by specifying which coordinates the robot will travel to. There will be an input field allowing an integer to be inputted for x and y coordinates and then a travel button. The other button will be the launch button

which will activate the sequence of making sure it's safe to launch and then outputting the result back to the app.

d. A safety feature that can be added to the app is a toggle switch that will automatically start off and must be turned on in order to give commands to the robot. The reason for this is to prevent the user from accidentally miss clicking the launch/travel. Another way to prevent this from happening will be to add a confirmation message where the user must select yes again in order to give a command to the robot.

Additional Observations

a. There may be additional features added later in development like a way to turn off feedback from certain sensors on the robot. This can be extremely important for preserving the battery on the robot if it didn't need to update its GPS coordinates constantly for example.
b. There are two types of Bluetooth communication for android devices which are Classic Bluetooth and Bluetooth Low Energy. Our project would use Classic Bluetooth because we need a constant stream of information from the app to the robot which will consume more power. There are Bluetooth APIs that allow the app to scan for Bluetooth devices, transfer data, and use RFCOMM channels. The RFCOMM channels can be used to directly communicate with specific sensors which are ideal.
c. Another thing to be improved on in the future is reducing battery consumption on the mobile device and robot. For example, when the app is running as a background app the sensors should not be sending information.

4. Related Standards and Design Constraints

4.1 Related Standards

There are multiple standards that we must follow for our project in order to create a uniform and safe product. Furthermore, because future senior design teams will be building off of our project, it is very important that we make clear the protocols and standards that will be followed in the project. The following sections will specify the standards we will follow and why they were chosen.

4.1.1 IEEE 1725-2021

This is a standard for rechargeable batteries that is mostly used for telephones. We are going to follow several of the standards here since the robot will be utilizing a rechargeable battery that will recharge through a USB port. It goes over several different requirements that the battery must have before being utilized in the respective system. We will use this standard to make suitable operating system requirements based on the condition that the robot will go through. The battery will need an appropriate voltage and current to be able to operate the robot in a consistent manner so that there will be no errors that would cause a problem in any of the subsystems.

Furthermore, we will have to identify the safe temperature operating range for the battery and make a protective measure to prevent any damage from occurring due to high temperatures. Since the robot will spend most of the time outside, there will be exposure to high temperatures from both the sunlight and the BES which would make temperature operating conditions an important factor when selecting a battery to utilize with the robot.

Another standard that we will utilize from this IEEE standard is meeting the minimum subsystem requirements to set up the battery system in the robot. The subsystem requirements that the robot must at least meet in order to have a functioning battery system is AC/DC adapter or charger and battery. Depending on whether we utilize a lithium-ion battery would affect whether we follow the other minimum subsystem requirements which is the battery pack and cell, but that is only if we decide to utilize a lithium-ion battery. The robot will require an AC/DC adapter or charge since the requirement of the robot is to charge it from an outlet utilizing the USB port on it and this will convert the energy from AC to DC which would then be stored in the battery. The robot must also have a battery that can safely store all the energy from recharging it and must have enough capacity to meet the requirement set by the sponsor of a battery life of 10 hours. The AC/DC adapter must also follow the safety standards of IEC/UL 62368-1 which go over creating safeguards to prevent any exceeding of dangerous temperatures and utilizing safeguards to prevent the transfer of energy to affect the user in a dangerous and harmful way. Furthermore, if we decide to utilize a lithium-ion battery, which we probably would not decide to utilize, we have to take into consideration the design of the battery so that it will be able to power the robot in a safe manner and not cause strain on other components f the robot by taking into account the size and design for feasibility to maintain or replace.

There is also a standard that goes over thermal protection and testing. The standard would require that we test the battery in an oven at 130℃ for an hour and ensure that the battery does not flame or explode during the test, but this applies to lithium-ion cells which we probably wouldn't use. Furthermore, due to the cost of any dangers occurring or the cell being damaged due to the excessive heat, we will avoid doing this test and only test it in a more normal manner like being placed under sunlight temperature or making a box to create a similar effect as a car that heats up since this will be the temperature that the battery is most likely to experience. The thermal consideration that we will have to account for would depend on the battery that is being utilized since each battery has its own specific temperature operating range. The standards for thermal protection will be the ones that we will take most into consideration since the robot will be exposed to high temperatures. The standard that we will follow is the addition of a thermal sensor to read the temperature that the battery is currently at which will allow us to monitor whether the battery stays within the safe temperature operating range or exceeds the range and becomes hot enough where it is no longer safe to operate the robot. The other standard in thermal protection that we will have to follow is a protective measure or action that the robot will take if the sensor detects that the robot has exceeded the safe temperature operating range. Some actions that the robot would have to take if the temperature increases to unsafe levels would be shutting down the entire robot, disabling the charging of the battery if the increase in temperature occurs during battery charging, or initiating some cooling system to help lower the temperature.

The other battery standards that we will have to take into consideration are the standards for overcharging. This will take into consideration the max that the battery can hold to prevent the battery from being overcharged. The charge specification that we would have to follow would be dependent on the battery that is utilized since each battery has its own limitations on the current that gets inputted into the battery and the voltage requirements. We do have to follow the standard of overcurrent protection in the charger to prevent the charger from delivering more than the max safe current allowed to charge the battery so we would have to utilize a limiter that limits the max current that goes through the charger to prevent a high current from damaging the battery or the robot itself. We also have to take into consideration the design of the charger itself, so that there is a limit to the voltage and current which would lead to the charger containing the limiters for current and voltage to prevent overcharging. The battery must also contain a limiter to prevent any over current in the case that there is a failure in the limiters that were placed in the charger. The other standard in overcharging that we would have to follow is the addition of 2 protective functions that would protect the battery in the event of an overcharge. The protective measures that we can take are LEDs that light up when the battery is full, an active circuit, resistors, running the thermal sensor so that the temperature of the battery is monitored, or including a system that emits a sound so that the robot can notify the user that the battery is fully charged. These protective functions allow the robot to discharge any excessive energy that it is obtaining through charging to prevent the battery from being overcharged and causing damage to the battery and robot. Furthermore, the battery will need to follow the standard for over-discharge. This standard sets the minimum discharge voltage and current that the battery must adhere to so that there is no damage to the other systems of the robot. Furthermore, the battery must have a safety mechanism that prevents it from discharging to any external system if there is a detection of an over-discharge. A safety mechanism that we can take in the event of an over-discharge can be a fuse or a breaker that trips if there is a detection of an over-discharge.

We also have to take into consideration the standards for mechanical considerations of the battery. The space allotted for the battery must take into account the size of the battery and have some tolerance in the case that there are changes to the size of the battery. There should be some isolation on the terminals of the battery so that there is no accidental electrical discharge in the robot that can essentially short out the battery and cause a potential fire hazard. There should also be appropriate spacing to the cable and connectors to prevent abrasion, wear, or damage from occurring. There should also be appropriate relief from strain to the cable and connectors so that there is no damage in the cables or connectors of the battery system. The inside of the frame of the robot must also be spacious enough to prevent any cables or connections from being stretched in an unsafe manner. Furthermore, most of the electrical connections should be soldered in place so that there is an increase in the stability of the design but the main connections to the battery should not be soldered in place in the event that the battery has to be replaced. There should also be clearance in the space for the battery in the case that the battery becomes deformed due to damages or external forces and this clearance also applies to any of the electronic components on the robot so that no sensitive parts are too close to the battery in the case that it becomes deformed. If the battery utilizes a circuit board, then the circuit board must be mechanically isolated to prevent any accidental electrical connections from occurring. Furthermore, the battery must be designed to mitigate the effects of external mechanical forces like shaking, bending, twisting, pressing, shock, and drops. This design should lessen the probability of the battery being deformed due to these external mechanical forces. Since the robot is going to be moving around on a firing range the battery mostly has to take into account the vibrations created from the movement of the robot itself and the shock from the impact of potential bullets from soldiers training in the range. Since this is just an initial prototype, we will not take into account the impact of bullets since this is a smaller scale that will not take into account any bulletproofing. The connectors and the terminals of the battery must also be designed to prevent an accidental short circuit from occurring and prevent shorting from contact with common conductive objects. Since the robot will probably be utilizing a metallic frame in the final version, this would have to be taken into consideration when designing the battery of the robot so that the frame of the robot itself does not cause an accidental short circuit.

There are also some battery assembly considerations that we will have to take into account. We will have to ensure that the battery has all of its internal components properly soldered and does not contain any solder balls, flashes, or bridges so that the battery is safe to operate. We also will have to ensure that the inside of the frame of the robot follows the IP65 rating since the standard for the battery involves that no foreign objects get into the battery. Foreign objects also cannot get into any of the protection-based circuits for the battery to prevent any damages from occurring that can cause the battery to lose protection from overcharging and over-discharging. The battery itself must also not contain any short circuit either internally or externally due to the hazard that a short circuit can present to both the battery and the robot itself.

There are also standards over critical testing practices on a battery to ensure that it meets most of the protective-based standards. We will avoid doing any extreme degree of testing to prevent the battery itself from being damaged which would increase the cost of the robot if the battery were to be damaged. We will test the battery but to a lesser extent than what is generally covered by the standards. We will test if the overcharge protection is functioning as intended by connecting the battery to the charger and attempting to charge the battery beyond the max capacity to see if any of the protective functions for the battery will activate whether it is an LED or a sound that the robot will make. We will also test the thermal protection of the robot by placing it in a hot environment like outdoors with the car heating effect to see if the robot will activate a cooling system or shut down to protect the inside components from too much heat. Normally the test in the standards is to verify that the battery or cell doesn't flame or explode which is what we are trying to avoid doing to avoid any damage from occurring.

The other battery standards take into consideration the device that it is being utilized in as a whole. In our case, the whole device that it is operating is the robot which must follow some of the standards. The robot must be marked in some way with an indication of the voltage and current that it must take in order to charge and also be marked with an indication of how much voltage and current that it will output. We will probably put this indication on either the maintenance panel of the robot, or place this somewhere near where the USB port to charge the robot will be placed. The robot as a whole must also not allow damaging power surges to occur so it will require some kind of surge protector. Furthermore, the robot must not override any of the protective features placed for the battery like overriding any limiters or protectors from overcharging. The robot must also be capable of withstanding the maximum allowed voltage and current of the battery. The robot must also be able to isolate the input connection from the rest of the system in the case that the protective functions of overcharging were to fail in order to prevent damage to the rest of the system. The robot must also be designed so that it can terminate the charging process in the event that the user leaves the robot in the charger for an extended period of time and if it is not able to terminate the charging process, then utilize a method that is capable of safely discharging the excess energy as mentioned in the protective functions of a battery. The robot must also have the charging port be robust and last over many insertions of the charging cable so that the system will last for an extended time period without the charging port from being damaged over the many insertions. The electrical connections inside the robot must be set up to prevent any damages where the ground connection/pin of any circuit or component must be connected before any other connection/pin. The robot must also contain specifications about the operating temperatures that the robot can work under and must also contain the qualifications of the battery itself so that the end user knows the information about the battery itself. The robot must also undergo testing practices to ensure that all the systems are set up correctly and there is no issue regarding the connections with the battery itself. As mentioned before, we will test the robot under the conditions that it will normally encounter during the operation that it will normally undergo and will not be tested under any extreme conditions that will damage any components or the robot itself.

The adapter of the charger itself must also comply with some of the standards mentioned before. The adapter must contain information about the specifications of the adapter itself like the maximum and minimum voltage and current that it is rated for. The adapter must also have the specifications for the power and qualifications. The adapter must also follow the specs for the battery to prevent any overcharging and should contain any limiters or safety measures to prevent any overcharging to the battery. We also need to add any cautionary note for the adapter itself, for example, if the adapter is meant for indoor use only, then the adapter needs a cautionary note on it that specifies it is only for indoor use so that the user does not connect the adapter outdoors and can potentially cause a problem if the adapter gets wet. The adapter must also contain any markings from where it was made or the manufacturer responsible for making it. In our case, depending on the battery we use, we will either use an adapter that was made for our specific battery, purchase an adapter that meets the specification of the battery, or have to make our own adapter or modify an existing adapter. If we make/modify an adapter, we will have to make sure that the adapter contains all of the information associated with it like the voltage, current, power, and any safety notices. We will also have to prevent any foreign objects or liquids from entering the adapter so that no short circuit occurs in the adapter and potentially causing major hazards. The adapter will also have to withstand any effects from vibrations or shock that can occur during normal usage so that it can reliably charge the battery without creating any faults. Furthermore, we will need to test the adapter to verify that everything is functioning correctly. The way that we will test the adapter and the battery is by charging the battery while it is not connected to the robot in the event that a fault or more severe error occurs, the damages would only be limited to just the battery and the charger. The battery will be connected to a simple circuit to test and verify that it did receive charge and that the proper protection functions are working as intended.

The other standards that are for batteries lie with the end user of the products where they will have some responsibilities on their part in following the correct information that is mentioned on the adapter and the battery. We have to ensure all safety-related information is posted somewhere either on the robot or in some kind of user manual so that the user can follow the proper safety procedure to prevent any problems from occurring in the battery. We will have to put in information about what to avoid, recommended areas to connect the robot, and proper information on how to handle the battery in the event that the battery needs to be replaced. The robot must also be able to give out alerts in the event that the robot encounters unsafe conditions like unsafe temperatures, high current draw, and alerts for any malfunction. These alerts should help the end user identify that there is a problem and attempt to either shut off the robot or in the event of an overcharge, to disconnect the robot from the charger. In the event that a hazard does occur to the battery, depending on the type of battery, there should be information pertaining to the proper procedure to take in the event of a damaged or leaking battery and also any recommendations on medical help in the event that the battery releases any toxic fumes or there is an ingestion of a component of the battery. Furthermore, all components and parts of the robot and the battery system must be appropriately marked and labeled. We could do this by either using colored tape with markings on it to mark each component or having each component marked before putting it all together by having the markings being a part of the frame for each component. We will have to follow the common symbols that are used in marking each component so that it is consistent with what is normally used in other components that users regularly use so that they recognize what the symbol means.

4.1.2 IEEE 802.15.1-2002

This is a wireless communication standard specifically targeted at Bluetooth. We will be using bluetooth to communicate with the robot through the tablet and there are many potential issues that can arise including security, frequency, and communication issues. It talks about how the frequency that should be used for Bluetooth is 2.4GHz and how the encryption protocol should be done to prevent people from intercepting the data. We will reference this standard when establishing a connection between the tablet and robot if any issues arise and in order to make sure the connection is secure. This standard is slightly dated and we will be using the newer Bluetooth 5.0, however, lots of the original methods regarding the connection through RFCOMM channels remain the same.

4.1.3 IEEE/ISO/IEC 29148-2018

This is a Software Requirements Specification standard that will be used to create adequate requirements for our project. This is important because all of our requirements should follow a standard in order to ensure they properly represent the project. The requirements should be clear to the team members, mentors, and faculty in order to guarantee that everyone is on the same page and understands our project. This standard says that each individual requirement should have the following characteristics:

- Necessary - The requirement should not be redundant and it should represent an essential function. The primary reason to have this characteristic is to focus on requirements with a specific reason it should have a clear purpose.
- Appropriate - The requirement should appropriately identify exactly what is needed with the appropriate amount of detail. The reason for this characteristic is to not contain too much-unneeded information or to not have enough information in the requirement.
- Unambiguous - The requirement should be clear as to the purpose and not be unambiguous on the interpretation. The reason for this is to make sure everyone will interpret the requirement as the same thing and be consistent.
- Complete - The requirement should be able to be understood on its own without needing to know other requirements or external information. The reason for this is because the requirements should be independent from each other and should be able to clearly present the information.
- Singular - The requirement should state a single thing and not be a combination of requirements. The reason for this is because we want the requirement to present only one function as a single node and not combine multiple requirements.
- Feasible - The requirement should be realistic according to the given constraints like the budget, timeframe, and knowledge. The reason for this is that all of the requirements should be doable to complete within our scope.
- Verifiable - The requirement can be verified once the project has been completed. The reason for this is that we will need to clearly show that all of the requirements have been satisfied through testable means.

- Correct - The requirement should be accurate pertaining to the needed function. The reason for this is that we do not want mislabeled requirements and they should clearly represent the objective.
- Conforming - The requirement should conform to the desired standards and templates. The reason for this is that we want the requirement to follow all of the rules we have set and conform to the named standards.

Additionally it is recommended to avoid ambiguous terms or “loopholes” like ‘if possible’ or ‘as appropriate when creating the requirements. It is also important to consider the risk and difficulty of the requirements in order to make sure they are realizable within our scope. By following this standard, requirements will be clear and concise to our professors and mentors which will help everyone understand the project.

4.1.4 ISO/IEC/IEEE 29919

This is a standard for software testing and it was created in five parts going over concepts and definitions, test processes, test documentation, test techniques, and keyword-driven testing. One common problem in testing software is you know there is an issue but are not sure which part of code the issue exists. This standard helps provide a systematic way to apply testing methods in order to troubleshoot and find bugs within our code. This will be important as we are on a heavy time constraint which means we will have limited time for troubleshooting. The main processes it recommends using are:

Specification Based Techniques - These techniques are testing different inputs across the code where generally the user will derive the test cases. This will be the bulk of the troubleshooting where we will be deriving test cases through the various techniques listed below.

a. Combinatorial Testing is where a certain amount of test cases are derived to cover a wide variety of cases. This is the most significant testing and when deriving the test cases it is very important to cover different inputs that will cause the code to react in different ways.
b. Boundary testing is where the lower bound and upper bound input is tested. The reason for this is to generally test overflow and underflow errors that can potentially happen with large test cases.
c. Syntax testing is where test cases that specifically invalidate the desired input syntax are used. For example, if the program asked for an int from the user and they gave a string the program would need to account for this and throw an invalid input error.
d. Random testing is where random test cases are generated and the output is checked to see if it's correct. The pro to this technique is that it can be automated to speed up time. However, the downside is that the random test cases will not be geared towards the problem and are less targeted.

Structure Based Techniques - These are very simple techniques that are generally accomplished by the IDE and compiler because they can run through the whole structure of the program. These techniques should be done before specification-based testing, as they can be automated much more easily without having to manually derive test cases.

a. Statement testing is where each statement in the code is executed once. This is basically just running the code and seeing if any errors occur.
b. Branch and decision testing are where you traverse each decision/branch in the code in order to make sure you cover all the code. This is important because sometimes there can be errors only on the true side of an if statement or vice versa.
c. Data flow testing is looking through the code to find issues related specifically to variables. The common examples are a variable being defined twice before it is ever used, being created and never used, and using a variable before it is created. Luckily most IDE's are capable of finding these errors as they can quickly parse through the code and highlight these errors.

Experienced Based Techniques - These techniques use prior experiences in order to create test cases or guess the error. The standard does not focus too much on these techniques as it is really only used for simpler programs/functions.

a. Error Guessing is the only experience-based technique talked about in the standard and the goal is to basically guess the reason for the error. For example, simple mistakes like dividing by zero or null pointers exceptions can be guessed based on previous experiences of similar errors.

4.1.5 PEP8

PEP8 is a standard for Python with the goal of helping code readability and consistency. For all of the AI work and ROS we will be using Python so this is a very relevant standard to follow. Some of the key rules to follow are limiting the length of each line to 79 characters, importing libraries on separate lines, and variable naming rules like not using single letters as variables. For example, under this convention variable names like x or i are not recommended because they are arbitrary and do not give a description of what the variable is. The reason 79 characters is recommended as the max per line is because this will allow all devices to view the code without having to use a scroll wheel which can impair the readability. Another aspect of PEP8 is regarding commenting on the code. It is recommended to use block comments to describe what the function/class is and give a brief description of how it works. Inline commenting (commenting on the same line as code) should avoid being done because generally they explain the obvious and can clutter the code. By following this standard it will be easier for outside observers to understand our code and for future teams to build upon it.

4.2 Design Constraints

The following sections describe the possible constraints this group may face throughout the development of this project. These constraints, excluding time, were formed from the Accreditation Board for Engineering and Technology (ABET). The time constraint was formed by our team. We saw time as our largest constraint due to the number of requirements we have to incorporate into this project per our sponsor, and the limited amount of time we have to do so.

4.2.1 Economic Constraints

The Battle Effects Simulator Robot is sponsored by PEO STRI. We have been in communication with our sponsor about a budget for this project. Our sponsor has informed us that funding for this project will be possible, but the amount has not been confirmed yet. Our sponsor is estimating a budget of $5,000 for this project. Although this is a large budget, we must take into consideration that this robot will need to be army grade and comply with all the requirements our sponsor has given us. This budget is an estimate currently, and we will continue to update this amount when we have more information from our sponsor. To help with this constraint we will make sure to consider all options when looking at parts for our robot. We will make sure we choose parts that are both affordable and reliable. This is extremely important because this is a Modular Open System Approach (MOSA) robot. MOSA is a set of guidelines and requirements that results in affordable and efficient projects.

It is also a possibility that our sponsor may not be able to provide us with a budget at all, or we will not be able to receive the budget within the timeframe this group needs. If that is the case, our team will build an even smaller scale version of this robot with a budget of $500 or less provided by the team. Our team has taken this possibility into consideration when researching what parts we will buy to build our robot, and are prepared for this potential situation. We have discussed this situation with our sponsor and he has agreed that if a budget is not attainable by the time we need it, a smaller scale prototype with a smaller list of requirements is fine.

Update: The economic constraint of the robot has been changed due to a reduction in the expected budget for the robot. The group was provided with a budget of $1,000 to work on the BES robot. Due to this change in the budget, we had to reevaluate the requirements that the robot had to achieve and the scale of the robot.

4.2.2 Environmental Constraints

Our robot will need to comply with the IP65 enclosure rating for weather conditions according to our sponsor. The IP65 enclosure rating states that the robot needs to provide protection against low-pressure water jets from any direction. This includes condensation and spraying of water. Our robot will not need to account for extreme conditions such as flooding. Our robot will need to function in the outdoor elements for most of its lifespan. We will be adding features that will help with the IP65 rating, but this will affect our cost constraint. The terrain must also be considered when making this robot as it will not always be a flat surface. The robot must be able to make it from point A to point B accurately while traveling on gravel, into Stationary Armor Target coffins, and other non-ideal terrains. The last part of the environment that should be considered is the type of impact that the robot may be taking. This robot will be holding dangerous pyrotechnics and needs to be able to withstand multiple launches every day. The hardware needs to be mounted well and protected or else the whole machine will be vulnerable to failures. It is expected that our sponsor or future senior design groups will reinforce the robot to withstand certain impacts when necessary. Our team still plans to be considerate of where hardware should go to avoid damage regardless of reinforcement.

Update: The environmental constraint had to be reevaluated due to the reduction in the budget. The robot no longer had to achieve the IP65 enclosure rating and instead only had to be somewhat water resistant equivalent to entering a small puddle. Furthermore, the robot will not be holding any pyrotechnics for this model which removes any added protection we needed to add from heat and explosives.

4.2.3 Social Constraints
We do not see any social issues arising during the development process of this project.

4.2.4 Political Constraints
We do not see any political issues arising during the development process of this project.

4.2.5 Ethical Constraints
We do not see any ethical issues arising during the development process of this project.

4.2.6 Health and Safety Constraints
To ensure the safety of our team, we will not be using actual pyrotechnics during the design and testing process. Although using pyrotechnics during our presentation would allow for a much more desirable and realistic demonstration, both UCF and PEO STRI believe it is not safe for our team to have access to pyrotechnics on campus. To ensure we are able to still simulate the pyrotechnics going off at the correct time and under the correct circumstances, PEO STRI will provide us with cartridges to put in the Battlefield Effects Simulator in place of the pyrotechnics that will shoot a beam of light to simulate when the pyrotechnics will go off. This way, the team can test and accurately demonstrate the function of the robot.

4.2.7 Manufacturability Constraints
Manufacturability is one of the main constraints for this project. There are many sections and subsections in a project of this magnitude. The design process for all of these sections is not simple, but our team has taken many steps to make the process as simple as possible for ourselves. When researching what parts we will use to build our robot, our goal has been to find all of the parts we need at a physical location, so we can immediately purchase it and begin our building. This will eliminate the possibility of parts not arriving on time or arriving broken. When it comes to our software, we are using free and open-source software for our operating system. This program is called ROS. It is extremely well known and respected within the robot

community, so we will have plenty of documentation to help us learn from which will result in an easier design process.

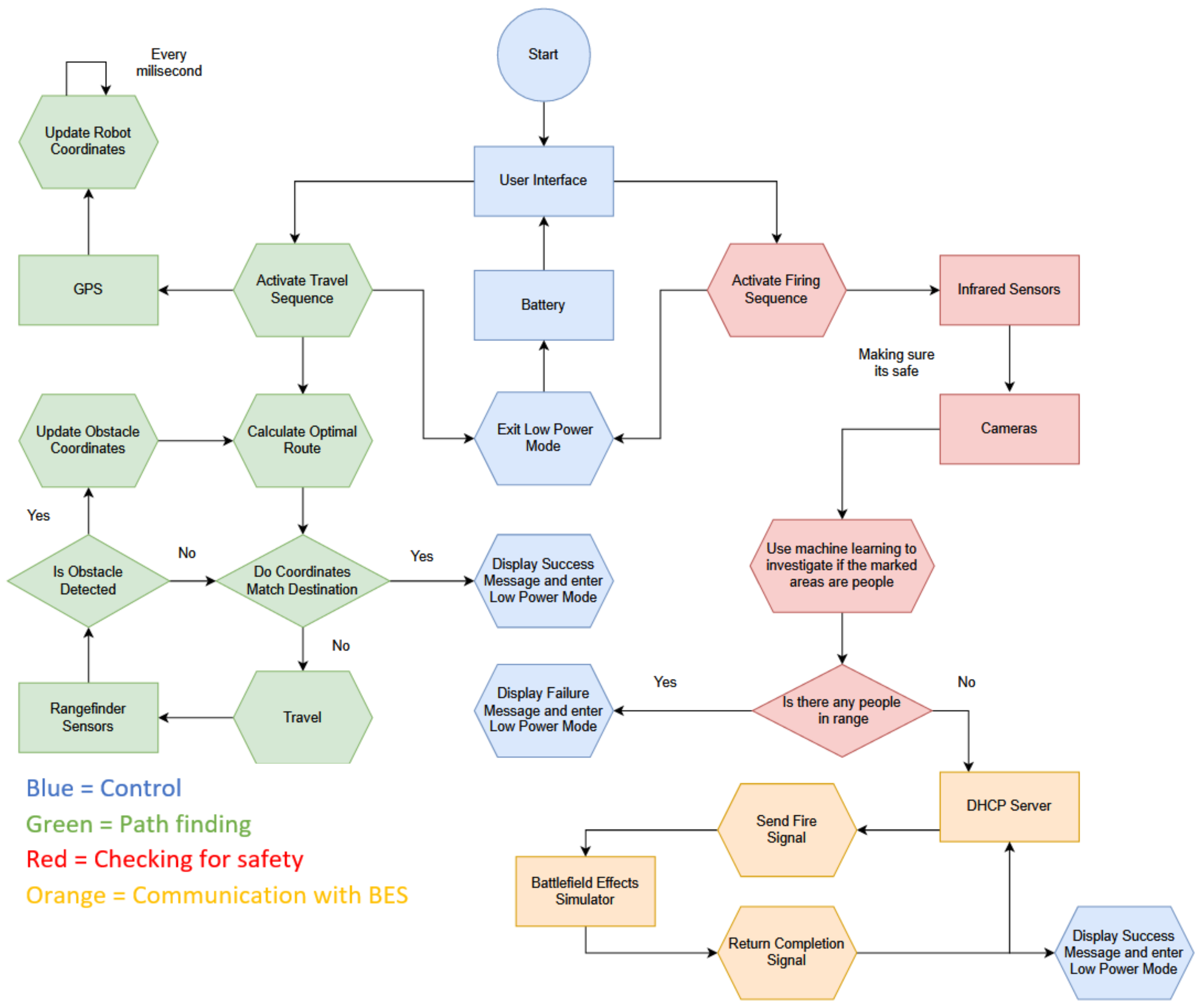
4.2.8 Sustainability Constraints
We do not see any ethical issues arising during the development process of this project.

4.2.9 Time Constraints
Our final constraint for this project, which is not an ABET constraint, but instead a constraint the team has discussed and felt was important enough to include here, is time. We are on a strict deadline to complete this project and have been tasked with building the Battlefield Effects Simulator robot over the span of two semesters. Our sponsor has expressed eventually turning this project into a multi-year project with our team building the initial phase 1 prototype. Our sponsor has given us the goal of creating a robot that is a quarter of the size of the final product robot. Although this is a small-scale prototype, our team still has to incorporate all of the requirements we have discussed with our sponsor. We have discussed with our sponsor that if time starts to become an issue during the building process of this robot, we will have to return to him with a shorter, more achievable list of requirements that is attainable for the team, but still satisfies what he needs from this project. Another factor that will affect our time constraint is how easily we can acquire parts. Electronics have been in shortage for some time now and it's affecting the availability and delivery time of parts. But as mentioned in the Manufacturability constraint section above, we will be continuously researching and attempting to acquire parts during the course of Senior Design 1 semester that we can receive quickly, or that we can pick up in person to ensure parts are available.

5. Project Block Diagrams

5.1 Software Block Diagram

Below is Diagram 2: Software Block Diagram. This shows the flow of our software processes as well as some of the tools and systems we will be using to connect to the control system.



Every milisecond
Update Robot Coordinates
Start
User Interface
GPS
Activate Travel Sequence
Battery
Activate Firing Sequence
Infrared Sensors
Making sure its safe
Update Obstacle Coordinates
Calculate Optimal Route
Exit Low Power Mode
Cameras
Yes
Is Obstacle Detected
No
Do Coordinates Match Destination
Yes
Display Success Message and enter Low Power Mode
Use machine learning to investigate if the marked areas are people
No
Rangefinder Sensors
Travel
Display Failure Message and enter Low Power Mode
Yes
Is there any people in range
No
DHCP Server
Send Fire Signal
Battlefield Effects Simulator
Return Completion Signal
Display Success Message and enter Low Power Mode
Blue = Control
Green = Path finding
Red = Checking for safety
Orange = Communication with BES

Diagram 3: Software Block Diagram

Update: After changing the logic and removing the Oak-D this is the final software logic/block diagram.

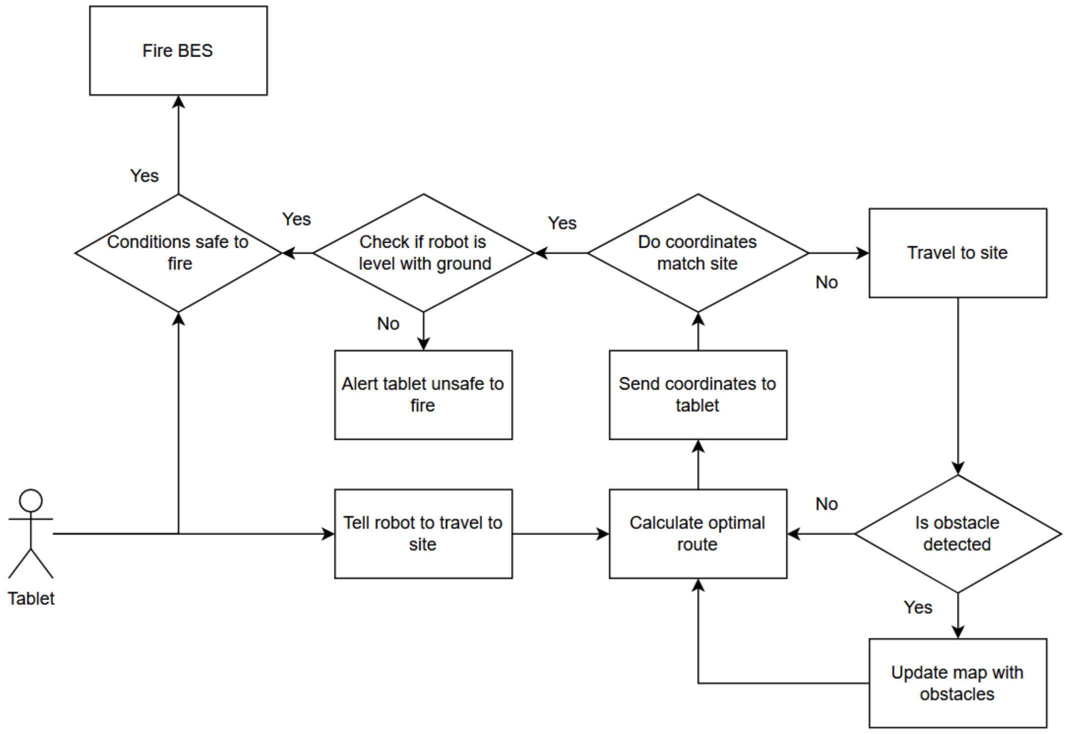
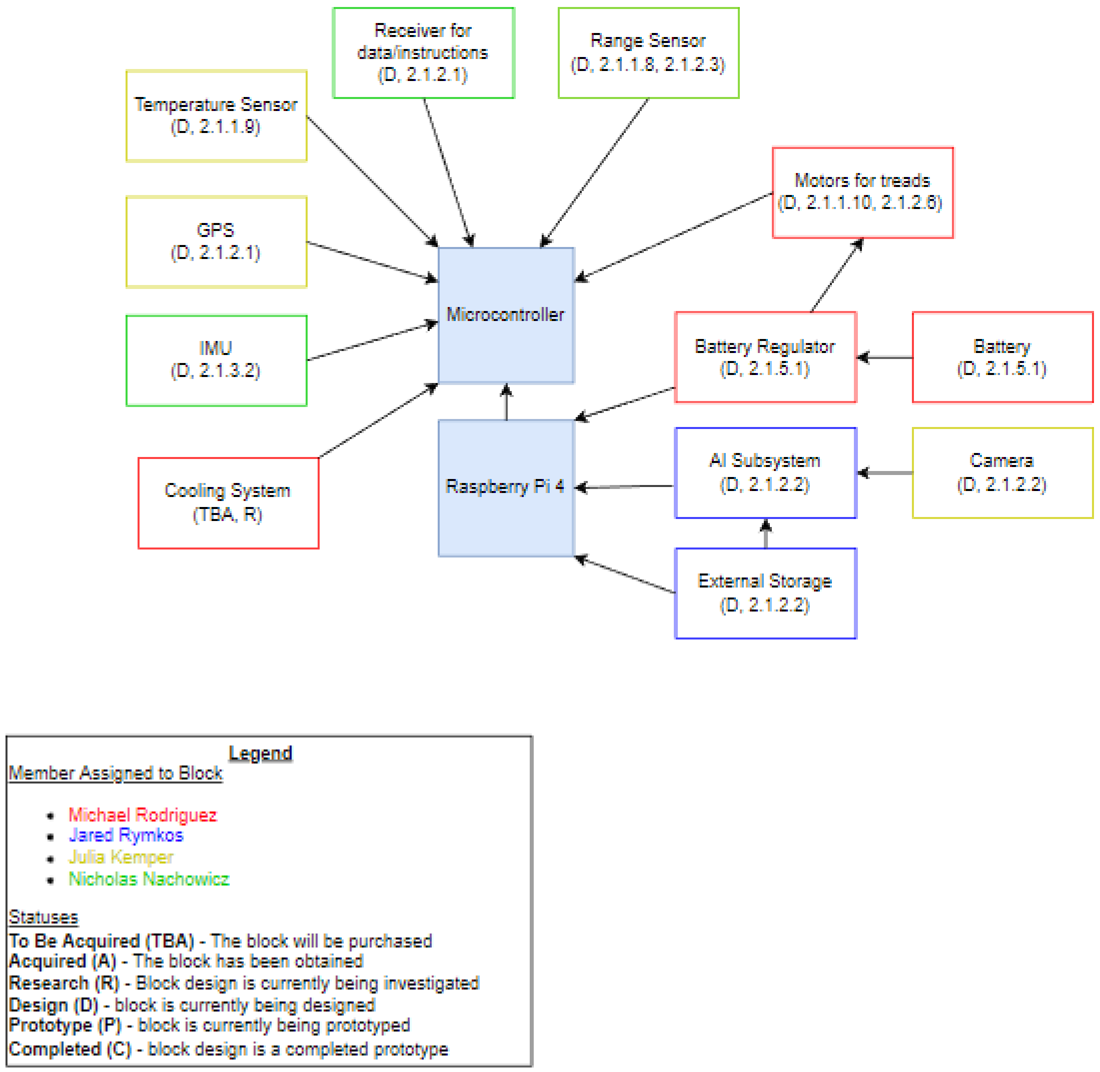

Fire BES
Yes
Conditions safe to fire
Yes
Check if robot is level with ground
Yes
Do coordinates match site
No
Travel to site
No
Alert tablet unsafe to fire
Send coordinates to tablet
Tablet
Tell robot to travel to site
Calculate optimal route
No
Is obstacle detected
Yes
Update map with obstacles

Diagram 4: Software Block Diagram Updated

5.2 Hardware Block Diagram

Below is Diagram 3: Hardware Block Diagram. This diagram shows the hardware and some of the connections it will contain. The diagram also has requirement numbers labeled in each block that pertain to each of the components.


Receiver for data/instructions (D, 2.1.2.1)
Range Sensor (D, 2.1.1.8, 2.1.2.3)
Temperature Sensor (D, 2.1.1.9)
Motors for treads (D, 2.1.1.10, 2.1.2.6)
GPS (D, 2.1.2.1)
Microcontroller
IMU (D, 2.1.3.2)
Battery Regulator (D, 2.1.5.1)
Battery (D, 2.1.5.1)
Cooling System (TBA, R)
Raspberry Pi 4
AI Subsystem (D, 2.1.2.2)
Camera (D, 2.1.2.2)
External Storage (D, 2.1.2.2)

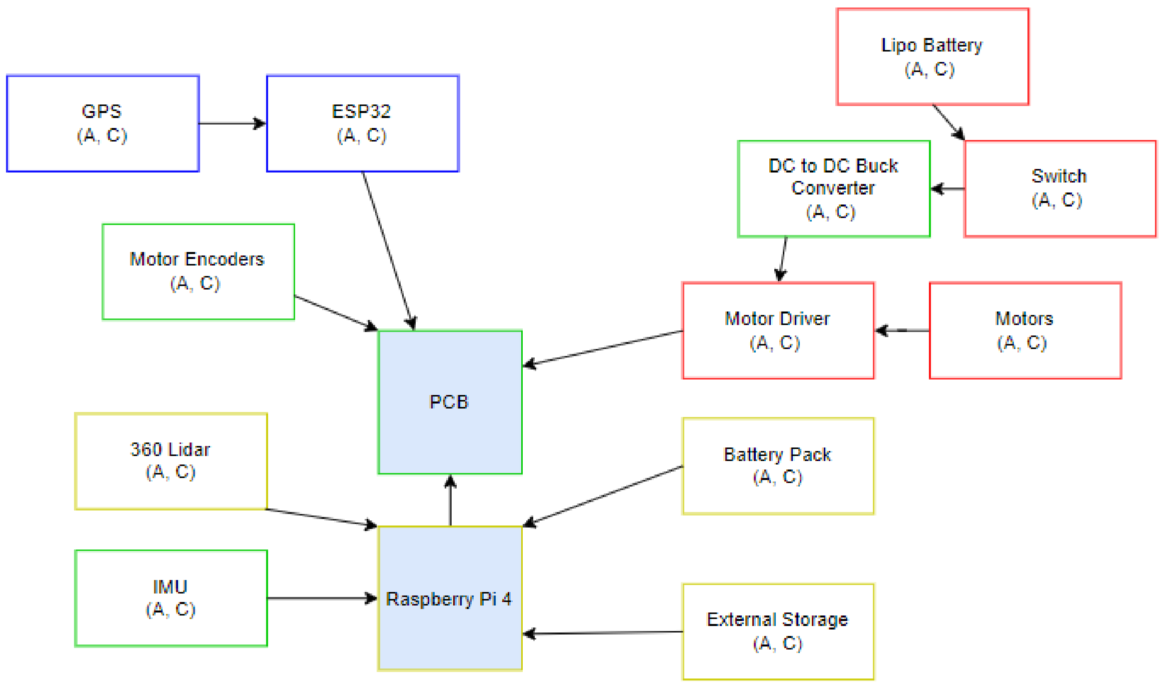
Legend
Member Assigned to Block

- Michael Rodriguez
- Jared Rymkos
- Julia Kemper
- Nicholas Nachowicz

Statuses
To Be Acquired (TBA) - The block will be purchased
Acquired (A) - The block has been obtained
Research (R) - Block design is currently being investigated
Design (D) - block is currently being designed
Prototype (P) - block is currently being prototyped
Completed (C) - block design is a completed prototype

Diagram 5: Hardware Block Diagram

Update: After finishing the prototype, we designed a new hardware block diagram based on the prototype that was made. Changes had to be done due to the robot being smaller scaled due to the decrease in budget and since the robot didn't need to meet as many requirements as before.



Legend

Member Assigned to Block

- Michael Rodriguez
- Jared Rymkos
- Julia Kemper
- Nicholas Nachowicz

Statuses

To Be Acquired (TBA) - The block will be purchased
Acquired (A) - The block has been obtained
Research (R) - Block design is currently being investigated
Design (D) - block is currently being designed
Prototype (P) - block is currently being prototyped
Completed (C) - block design is a completed prototype

Diagram 6: Hardware Block Diagram Updated

6. System Design

6.1 System Design Diagram

A robot has many subsystems that collectively play a role in creating something that can be considered functional. While making Diagram 4: System Subsystem we had to consider what are the main subsystems of a robot. We found 6 main subsystems that can define all of the components of a robot. These subsystems are the mechanical system, control system, human-machine interaction, sensing system, robot environment interaction, and drive system. The mechanical system makes up the components such as wheels on a robot or the chassis.

Control systems are the systems that control the robot such as a microcontroller. Human-machine interaction is a system that works as a user interface in our case it's a button on the robot that powers it. Sensory or sensing system is the system of sensors that the robot uses to understand the world and itself such as LIDAR and temperature sensors. Robot environmental interaction is the way the robot interacts with the world around it. Our robot doesn't have much environmental interaction besides driving through the terrain. This could be changed in the future by using an arm on the robot to unload and load shells from the BES. Drive systems are systems such as motor systems and power systems. With all of these systems come systems underneath them and then finally components that are the bottom layer of the diagram. The BES Robot is found in the middle of the diagram in yellow. All of the subsystems can be found in the color blue. The systems within the subsystems are in green. Finally, the components are in red and they can't be simplified any further.

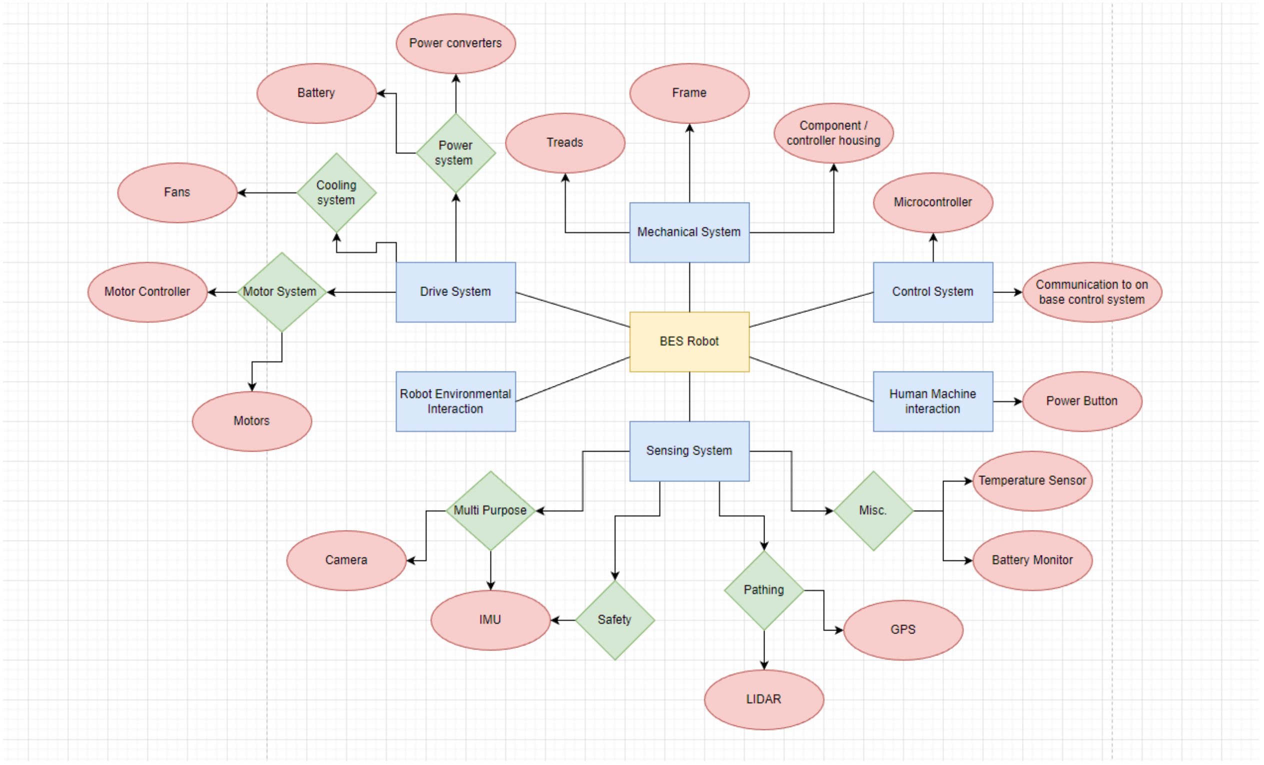


Diagram 7: System Subsystem

The block diagram below shows a general layout of how the components of this project will be connected and interact with each other.

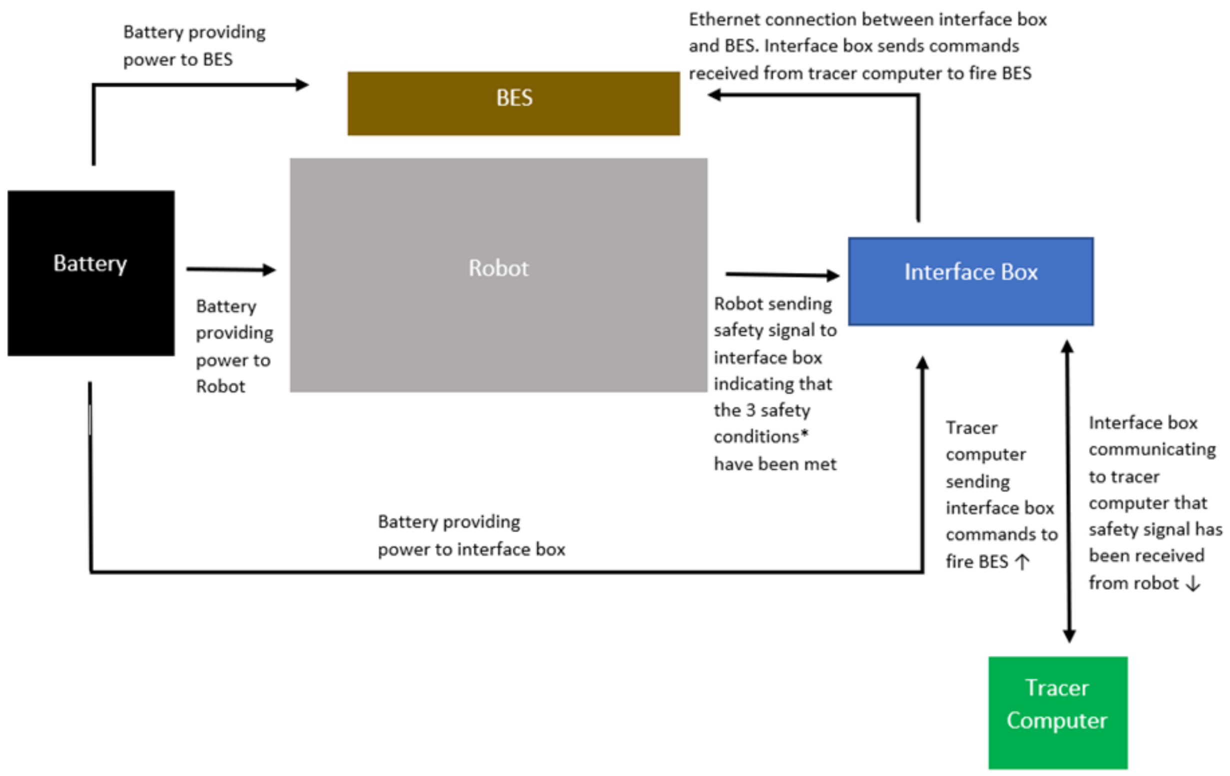

Battery providing power to BES
Ethernet connection between interface box and BES. Interface box sends commands received from tracer computer to fire BES
BES
Battery
Robot
Interface Box
Battery providing power to Robot
Robot sending safety signal to interface box indicating that the 3 safety conditions* have been met
Tracer computer sending interface box commands to fire BES ↑
Interface box communicating to tracer computer that safety signal has been received from robot ↓
Battery providing power to interface box
Tracer Computer

Diagram 8: General Block Diagram of System

* The safety conditions include the following:

1. The robot has arrived at the correct location on the map
2. There are no humans or cows within a 2-meter vicinity
3. The robot is on a flat surface plus or minus ten degrees

Update: The system block diagram that we would utilize throughout our project is depicted in the image below.

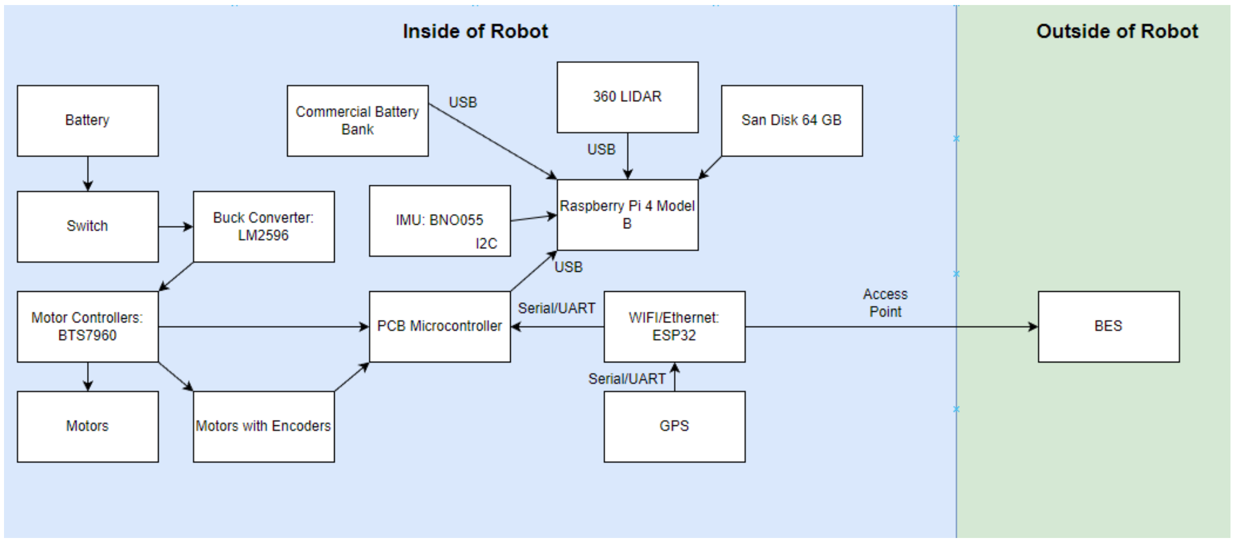


Diagram 9: System Subsystem Updated

We created four groups that would break down all of the components in our build. These would come to be the power system, drive system, sensory system, and communication system. This image also displays connectivity throughout the robot that isnt clear.

6.2 Robot Design

For the design of our system, we had to take into account several of our research topics in order to design a robot that will be able to perform well in the environment that it will be in. Figure 12A: Preliminary Design of the BES Robot is a 3D model of the preliminary design that we have made based on what we think would be best for the robot. The model is only a view of the exterior of what the frame of the robot would be and does not include how the inside of the robot will look like.

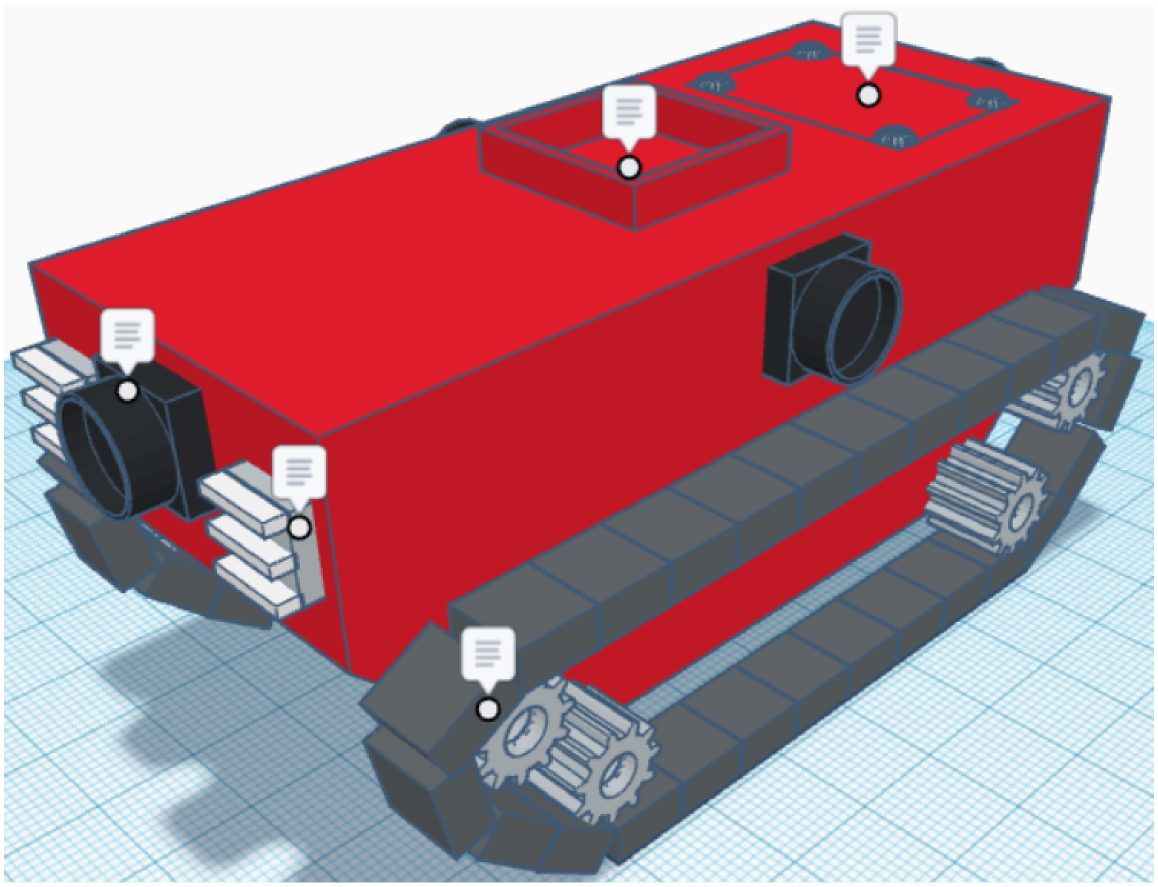


Figure 12A: Preliminary Design of the BES Robot

We have decided to utilize 4 cameras on the robot with each camera facing a different direction so that the robot has a 360° view to better identify the surroundings and be able to easily identify if a person is in the vicinity of the robot. By having 4 cameras, the robot has to spend less time looking around at the surroundings to identify if a person is in range before sending a signal to the BES. Also, the extra cameras will help in expanding the pathfinding capabilities of the robot by allowing the robot to be able to see all around it and identify obstacles or drop-offs that could cause the robot to fall. This design decision would be changed depending on the situation of budget or resources. The other decision we would make based on the camera is to utilize only 1 camera on the front and have the robot do a 360° turn in order to identify if there is any person in the area. Furthermore, if we want to make the camera more protected from the elements and down the line be bulletproof, then we can have the camera be inside the frame of the robot and have the front of the camera have a glass that is resistant to bullets or have a sliding door that covers the camera while in transit. Of course, for this prototype, we are not designing it to be bulletproof but this is an idea for down the line in case this robot is revised or modified.

The other design decision that we made was making a mount for the BES on top of the robot instead of having an interior compartment for the BES. This decision was made due to the fact that the BES is a pyrotechnic and having a pyrotechnic enclosed in a small area near sensitive components of the robot presented some risks to the robot suffering from damage due to the BES. With the BES mounted on top of the robot, the damage caused by the pyrotechnics to any of the components of the robot is minimized in case the worst outcome occurs with the BES. The other factor that led to this decision was the heat that a pyrotechnic emits which can damage electronic components or wiring which is why we decided to increase the distance of the source of heat by mounting the BES on top of the robot.

We also made the decision to go for treads instead of wheels for the design of the robot. This decision was made due to the treads having increased traction and torque which is a big necessity for the robot since it will be mostly maneuvering over uneven terrain and will probably be moving over terrain that doesn't consist of concrete. The treads provide better off-road capabilities than wheels and are more capable of going over small objects or crossing over small holes. This decision took into account the fact that the robot will be spending most of the time moving over terrain that is essentially off-road, which is where treads tend to be superior. Wheels can make up for this by utilizing wheels that are specific for off-road but wheels have to be at least double the size of an object in order to go over it which can cause path deviation due to one wheel getting stuck due to a rock. Treads on the other hand are less susceptible to getting caught on a rock or getting big path deviation because when the treads get stuck and can't move forward, then the entire robot will stop which can allow the robot to identify that there is something in the way that the range sensors did not identify and reverse away from the object and adjust course accordingly. The main design of the treads that we have decided on can be seen in Figure 12B: Tread design on 3D model. The reason we designed the treads to be in this specific format is so that the robot will have an easier time going over small objects and obstacles. This tread design allows the robot to have an upward force when the treads bump into a hill or small object which will allow it to more easily climb objects and have a smaller probability of getting stuck.

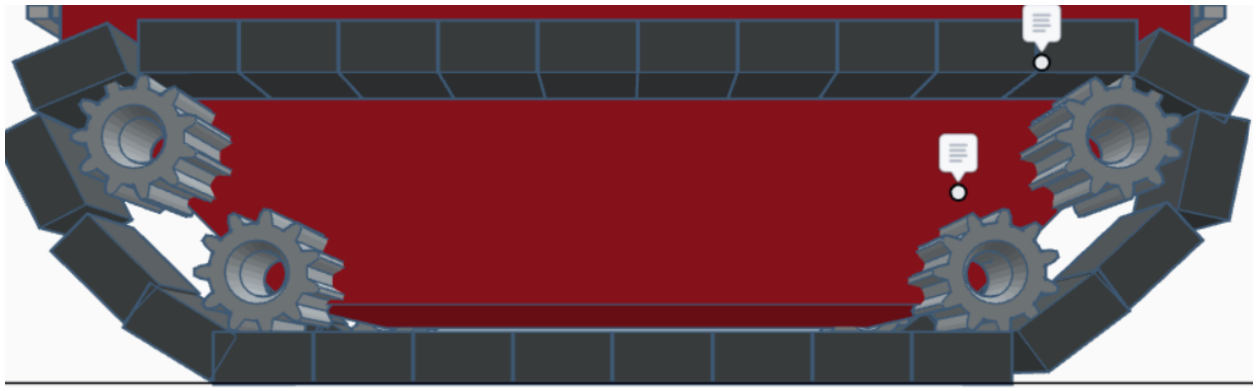


Figure 12B: Tread design on 3D model

Currently the model is utilizing range sensors in the front and the back, but that design only applies if we decide to use a range sensor that isn't LIDAR. If we use range sensors similar to ultrasonic sensors, then the sensors will have to be placed on the front and back of the robot so that it can detect objects in the path. Depending on the LIDAR sensor that we use would affect where the sensor can be placed on the robot. If we utilize a LIDAR sensor that has a 360° view, then we can put the LIDAR sensor on top of the robot but would probably have to utilize 2 of the sensors since the BES would get in the way and interfere with a small angle on the view that the LIDAR would see. If we utilize a LIDAR sensor that has a similar angle of view to the camera, then we can put the sensor alongside the camera so that it can be used in sync with the camera in order to better set up the pathfinding of the robot. The decision on the type of sensor that would be utilized would be dependent on the ability to acquire said sensors and the budget constraint to where we can't utilize sensors that are too expensive.

The other design choice that was made on the BES robot was a choice that is taking into account future maintenance, which is the addition of panels that can be unscrewed in order to access the inside components of the model. Currently, the model is only utilizing 1 panel for maintenance that is currently located on top of the robot which can be changed to be on the side of the robot depending on what will make the maintenance more feasible for the robot. We are also planning on adding another panel on the bottom of the robot so that we can conduct maintenance on the motors of the robot. The thing we would have to add to the bottom maintenance panel is some waterproofing features so that water underneath the robot doesn't get in the motor or end up stuck inside the robot. A way that we can avoid damage from water is either by utilizing components that are waterproof or water-resistant or by making the panel in such a way that water can't get inside the panel like utilizing a rubber-type border around the inside portion of the maintenance panel. An example of the protection that we plan to add to the maintenance panel can be seen in Figure 12C. The example seen is a maintenance panel that is used for electrical boxes that are located outside, which must be protected from water seen in strong weather conditions to prevent damage to the home electrical system. Since this is generally used for outside electrical boxes, we believe that this will be the best option when it comes to further increasing the resistance against water for the robot. The reason we designed the maintenance panel to be screwed in place is due to the fact that we don't expect the robot to go through maintenance often since it needs to be running for 10 hours, so we need to make sure that the robot has the resilience to maintain a long uptime. Furthermore, having the maintenance panel screwed in place instead of utilizing a door would increase the water-resistant capabilities of the maintenance panel.

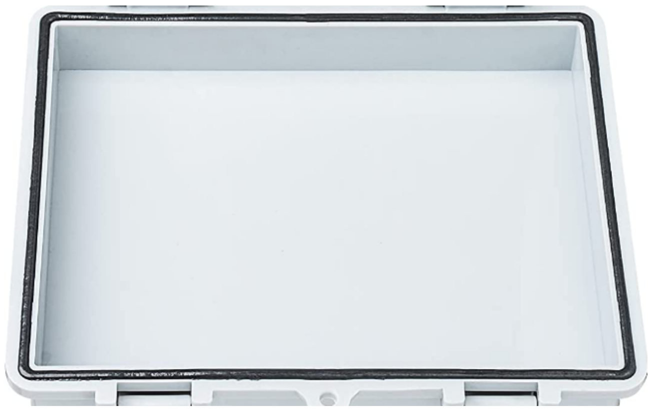


Figure 12C: A maintenance panel that has a protective border inside. (Protective border is the black line that can be seen)

Another design decision that isn't on the model but we would have to take into account is the addition of a charging port on the robot so that the battery of the robot can be recharged. The charging port would have to be located in an area where it would not suffer damage from the pyrotechnic or damage from water which would probably be located on the side of the robot. The other feature that the charging port would need is a cover so that while the robot is in use, the port would be better protected from potential damages. The cover of the port will have to be set up in order to remain closed while the robot is in operation, but be able to be easily accessed by a person so that they can charge the robot with little to no hassle. An example of protection we might put for the charging port can be seen in Figure 12D. The example seen is an Otterbox phone protector that utilizes some rubber-like flaps in order to protect the charging port from dust and moisture. The flap would be utilized since it stays secure in the closed position until someone tries to open it and it is an easier way for a user to access the charging port without having to go through the hassle of unscrewing a small panel. Another way that we could protect the charging port is using a door-like mechanism that has a waterproofing border inside but the problem with this is generally that the charging port is small so making a door to access a small compartment is a little bit of a waste and we wouldn't want to make this door utilized for maintenance as well due to the fact that while the robot is recharging, dust would be getting inside or someone cleaning the area near the robot could have an accident and potentially damage the inside components. We believe that the best option to protect the charging port from damages would be to utilize a flap as seen in the example since it is easier to modify the size of the flap and it will increase the ease of use for the user to charge the robot. Another type of design that can be utilized to protect the USB charging compartment would be to utilize a protector that plugs into the charging port which is an easier alternative to set up and would work with this prototype but would not be a good idea in the final design since the protector that is plugged into the port can get hit by a bullet if used in the final design that would be utilized in a firing range. The impact from a bullet would generate sufficient force into the protector that it would end up damaging the charging port which would render it unusable until repairs are made.

That would make this other type of protector to be more costly in the long run due to having to continuously repair or replace the charging port.

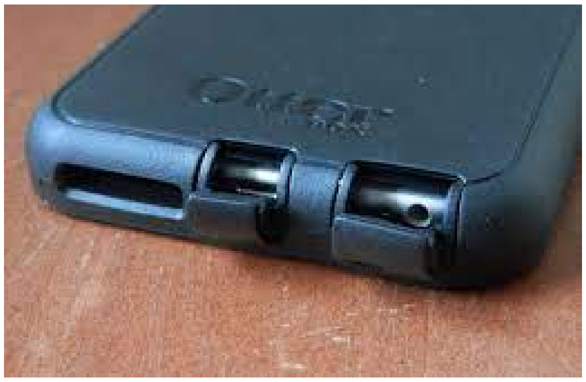


Figure 12D: An example of what will be utilized to protect the charging port. This example comes from an Otterbox phone protector.

The main thing that we have to account for when building the frame of the robot is that we have to make sure that the robot has an IP rating of 65 which means the electronic components inside have to be protected from dust and water from a nozzle. This means that the frame must have little to no openings and must have some kind of protection inside in case something gets through a small gap or opening in the frame. With this in mind, We have designed the bottom of the robot to have a similar design to a tank so that the robot won't have a sharp box-like shape which could lead to difficulties in maintaining the IP 65 rating. The tank design also allows the robot to be able to go over small obstacles without the frame getting caught. The IP 65 rating is also the reason why we need to add protection in the maintenance panel and protection to the charging port. Since the robot will spend a big amount of time outside, it has to follow the IP 65 rating in order to protect the sensitive components inside. The main hazards we expect the robot to encounter is dust hazard due to driving over off-road conditions and the water hazard we expect the robot to encounter will probably be due to rain or a puddle. The robot will not be expected to drive through deep puddles of water or streams since this will be utilized in a firing range. Due to the risk of water being relatively low due to the expectations we have of the conditions the robot will be operated in, we have decided to go for the IP rating 65 instead of going for a higher rating.

The other design decision that we have to make is an internal design which is adding heat resistance to the components inside. Many electronic components don't do well in high temperatures and the robot will be utilizing pyrotechnics which increases the intensity of temperature near the robot. The first thing we did to lower the amount of temperature that the robot will be exposed to was placing the mount for the BES on top of the robot instead of having an inside compartment for the BES which increases the distance of the BES from the electrical components. The next thing we would have to do to protect the electronic components is to add a cooling system to help decrease the internal temperature of the robot, but for an air cooling system to be efficient and functional, the robot must have an airflow that would bring a problem to making the robot resistant to water or having the components being protected by dust. If we go for an air-based cooling system, then we would have to have the inside electrical components in a box and that box has to be designed for ease of access for maintenance in order to remove the box and access the electrical components. Another thing we can add to the design of the robot inside is to add insulation to increase the amount of time that the inside of the robot will take to heat up from the high temperature. The bulk of the insulation will probably be located at the top portion of the robot since that is where the bulk of the heat will be generated from due to the BES.

The other design decision that we are making is the sizes of the gears to utilize for the gear train in order to achieve the desired gear ratio of the robot. We have decided that the best course of action will be to utilize a compound gear train in order to avoid having to use really big gears. This will help in order to conserve space for the inside of the robot for other components like the battery and the rest of the electric components. Figure 12E Contains a drawn image of the design the compound gear ratio will have along with the calculation to show that the design meets the required gear ratio to move the robot. The output gear was designed to be smaller than the gear that will be connected to the treads so that the output gear can be placed inside the frame of the robot without affecting the frame of the robot itself. If the output gear was bigger than the gear that will be connected to the treads, then the gear outside of the frame will have to be placed a further bit away from the edge of the frame so that the gear would be able to fit inside. For the sake that the image was hand drawn, there were no teeth added to the gears, so the calculation was accomplished using the diameters of each gear to represent the gear ratio. In the figure, gear A is the input gear and Gear D is the output gear. By utilizing a compound gear train, the final output gear was only 5 cm in diameter instead of having to utilize a gear that has a diameter of 12.4 cm if the input gear had a diameter of 2 cm which is more than twice the size of the gear that is connected to the treads which would definitely impact the placement of the gear that is connected to the treads or affect the frame of the robot itself and also impact the amount of space in the robot itself that could have been utilized for electrical components or as mentioned about protecting the robot from heat, space that could have been used for insulation.

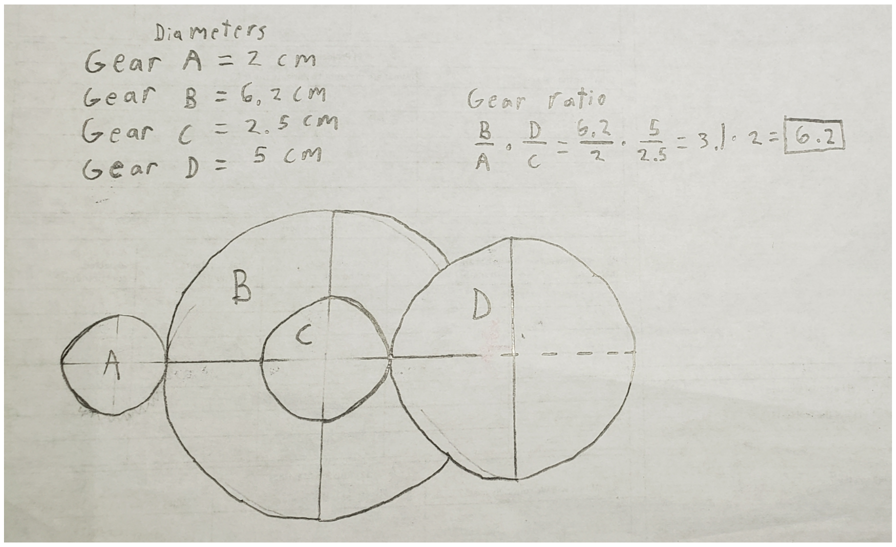


Figure 12E: Drawing of the compound gear ratio design

After going over a preliminary design review, we have decided to make changes on the initial design of the model of the robot. Seen in Figure 12F is the new updated design of the frame of the robot that is taking into consideration the changes that we discussed and have decided to modify. The frame and treads of the robot mostly remained the same, but some of the components that were mentioned before but were not in the model of the robot have been added into the new design.

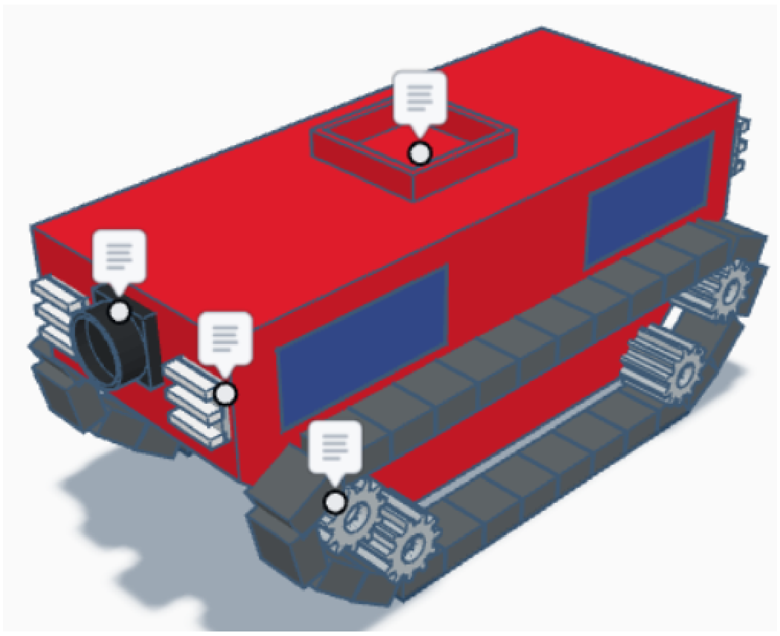


Figure 12F: Design of robot after preliminary design review

The things that were modified with this design was the removal of all but the front cameras, added maintenance panels and the change of locations of maintenance panels and the addition of the charging port onto the frame of the robot. The figure above shows the top front right angle of the robot which will not show all of the additions that were done to the model. Currently, the robot will only have 1 camera located on the front of the robot that will be utilized to identify if a person was in the vicinity of the robot and the robot will do a 360° turn to identify if a person is in the area. The other things that were modified were the movement of the maintenance panel to the side of the robot that is marked with the color blue in the image. We have decided to put the maintenance panel on the side so that there will be less exposure to the heat from the BES from the top of the robot and decided to have 2 maintenance panels on each side so that we can have easier access to some of the components of the robot. We kept the range sensors on the front and back of the robot since the range sensor will be our method of detecting any obstacles or collisions in the route that the robot will take. Figure 12G shows the bottom orientation of the view of the robot to reveal that we also decided to add a maintenance panel on the bottom so that we can have access to the motors and gears of the robot.

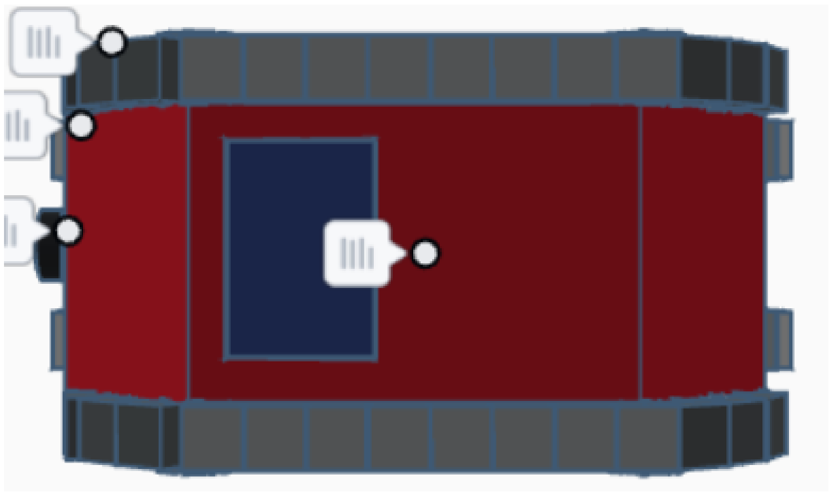


Figure 12G: Bottom view of the robot

As seen in the figure above, we have added the maintenance panel at the bottom marked by the color blue to indicate that a maintenance panel is in that location. As stated before, we will probably have the maintenance panels screwed in place for better protection of the components inside the robot which will mean that any maintenance that takes place will require that the maintenance panels be unscrewed. The other thing that we have added to the design of the robot as mentioned before is the addition of the charging port on the robot. This can be seen on Figure 12H which shows the back view of the robot. The charging port was placed on the backside since the robot will mostly be moving forward and the battery of the robot will probably be placed towards the back end of the robot. The charging port of the robot is marked in yellow to identify that it is a charging port and it will probably utilize the flap system that was mentioned earlier in the design of the robot. The range sensors can be seen on the back of the robot in the case that the robot will have to go in reverse and will potentially encounter an obstacle while going in reverse.

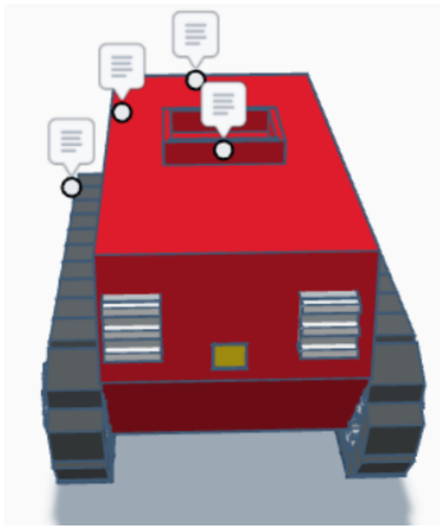


Figure 12H: Back slightly top view of the robot

After having done research on batteries and how to charge the batteries, there are a couple of different methods that can be utilized to charge the robot. We have 3 different designs in the port area where the robot will be charged. The first design is the one seen in the figure above where we wire a charging port to the battery so that the robot can be more easily recharged by the user. The next design can be seen in Figure 12I which will utilize a panel that opens up and give the user direct access to the battery to charge it utilizing a car battery charger or some other charger that connects to the two terminals of the battery. The drawing seen in the figure is not drawn to scale but is more for a conceptual view of how it will be set up. The image shows the side internal view of the backside of the robot to show where the battery will be positioned on the backside of the robot and it also shows the back view of the robot to see the panel that the user will be able to open to get access to the battery. The dotted line on the back of the robot marks the location of where the panel that the user can open will be located and the battery will be right next to the panel so the user will be able to connect the charger directly to the terminals of the battery. This design will utilize only 1 range sensor on the back of the robot and will probably be a single-point LIDAR sensor.

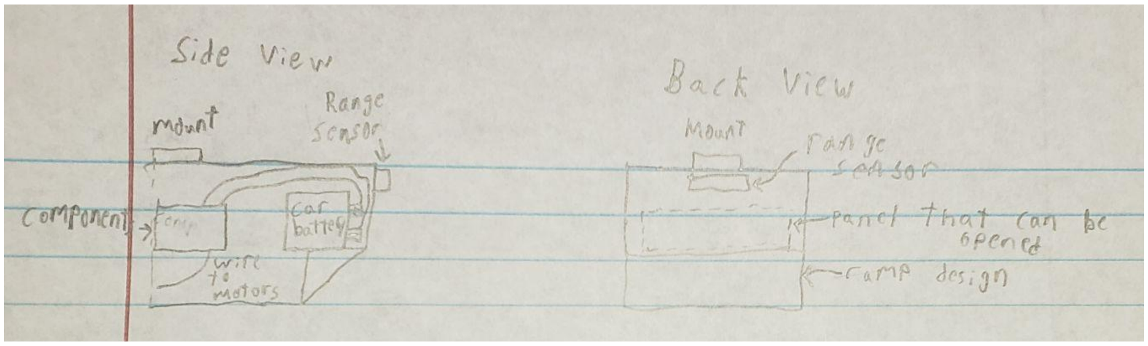


Figure 12I: Internal side view and back view of robot for charging

The other design for the charging port of the robot will be to hook up a jumper from the battery to two nodes to the back of the robot that will be insulated around the nodes so that the electricity from the battery does not flow throughout the robot which will result in loss of power or damage to some of the robot's components. The two nodes will each be covered by their own respective cover and will be separated so that there isn't an accidental short circuit. This design can be seen in the hand-drawn image of Figure 12J which shows the internal side view of the robot and the back view of the robot while the cover for the nodes are closed. The design of the cover can be a type of flap as mentioned previously but will probably be a flap that opens in the down direction so that the flap will not apply pressure to the cable that is charging the robot since the cable will be clamped like the car battery charger. The dotted lines show where each node will be and the respective size of the flap that will house each node. Figure 12K is a hand-drawn image as the same design discussed but with the flap of the charging nodes open to show the orientation that the flap will open and what the user will see when they open it. Inside of the compartment will just be a node that the user will be able to directly connect the charger of the robot to. Both hand-drawn images are not drawn to scale and are mostly meant to be a conceptual representation of what we are going to do to set up the battery.

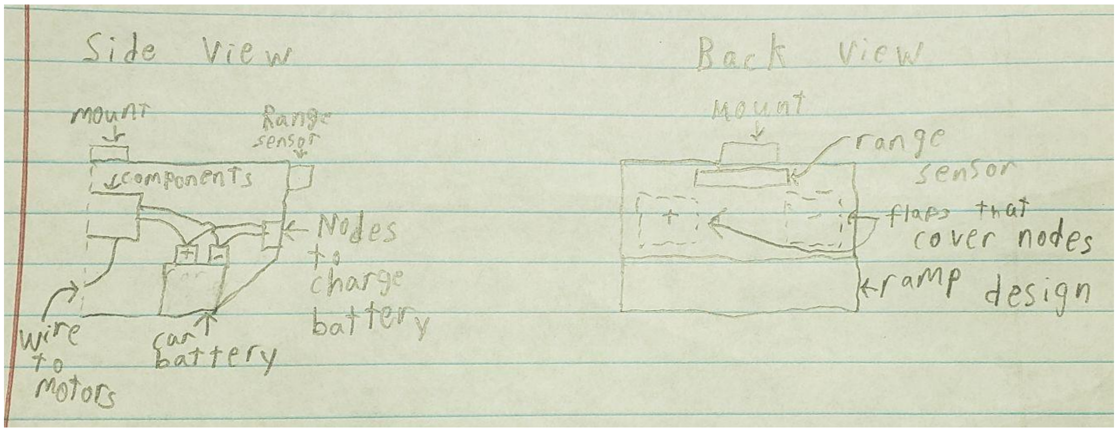


Figure 12J: Side and back view of robot utilizing nodes and a jumper to charge the battery

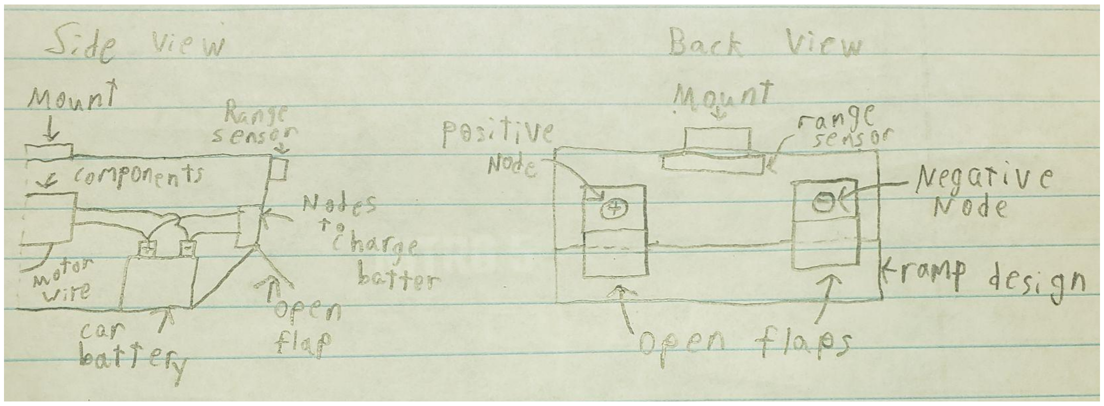


Figure 12K: Side and back view of robot utilizing nodes and jumper to charge the battery (Open Flaps to nodes)

With the node method as seen in the figure above, there will be a smaller probability of user error occurring in charging the battery because the nodes will be separated and clearly marked. The previous method that allows the user to connect the charger directly to the battery is more dangerous because the user will have a higher chance of accidentally touching the two nodes together which will create an accidental short circuit that will cause major damage to the battery and can potentially harm the user. By utilizing the jumper and the nodes the user error will be heavily reduced which will increase the safety of the user. The only problem with this method is that we will need to solder most things in place so that the wires connecting to the nodes don't come loose or create an accidental short. We will also have to design the cables connecting to the battery in such a way that it will allow us to disconnect the battery in the event of maintenance or in the event that the battery needs to be replaced. The other benefit for utilizing the node method will be that we will have the freedom to place the battery in many different locations in the robot since the jumper from the battery to the node is the only connection it requires for the charger. The method of directly connecting the charger to the battery limits the location of the battery to just the backside of the robot, right next to the panel that the user opens.

Update: The design of the robot had some alterations due to the decrease in budget and the utilization of a pre-designed base. The new design of the robot utilizes 4 sets of treads instead of 2 which led to the robot utilizing a 4 wheel drive. The robot was also no longer using a car battery which lowered the size and scale of the robot. Furthermore, since the robot was no longer meeting the IP65 rating, the safety protection mentioned previously was not utilized in the design of the robot. The robot also doesn't have a charger port, since we utilize smaller batteries that we just disconnect from the robot when we are charging it. The robot also utilized a single 360 Lidar sensor on top of the robot and only utilized 1 camera instead of 4 cameras.

6.3 Hardware Connectivity

6.3.1 Components list

While researching and preparing to build this robot our team has defined specific hardware components that will be used. In Table 12: Components List below our team gives specific reasoning, cost, and where we will be obtaining the components from. This list of components doesn't consider any mechanical structures that the robot will need such as treads or a frame. The table only focuses on parts that are electronics hardware.

Components List		
Name	Reason	Distributor and Price
Arduino Mega2560	The ATmega2560 is a microcontroller powerhouse. This microcontroller has 256KB of flash, 8KB of RAM, 86 I/O	Amazon.com: $34.68

	pins, 12PWM channels. See section 5.4.3 to read about the communication between the Arduino Mega2560 and ROS.	
SanDisk Ultra 64GB microSDXC	This storage will be for installing Ubuntu and ROS onto the minicomputer we choose	Amazon.com: $11.85
Raspberry pi 4	The raspberry pi was chosen as the single board computer. These minicomputers are great options for booting ROS. There are also a large number of open source projects and packages we can utilize for raspberry pis. They are probably the more well-known minicomputer around. The drawback is how expensive or unobtainable they are.	Amazon.com: $155.20
TFmini plus	TFmini is a single-point LIDAR that is capable of working outdoors. This LIDAR is capable of UART or I2C connections with the microcontroller. The LIDAR can read anywhere from 0.1m to 12m distances. The reason we choose this is that it is already IP65 rating and comparable in price to other outdoor single-point lidars. The LIDAR will get the distance between the robot and the obstacle around it.	Amazon.com: $53.90
BTS7960	BTS7960 is a high-power motor controller for motors that are drawing high amperage. This will be perfect for our 12V car battery and 12V motors. They will be drawing some significant power. The BTS7960 also has a heatsink that helps with the cooling of the system.	Amazon.com: $15.99
12V Battery	12V Car batteries are our best bet for obtaining a 10-hour battery life. They do weigh a significant amount and cost quite a bit however they hold a long charge and we will need that. This battery will fulfill the requirement of a 10-hour battery life while unfortunately making us take a big hit on weight making us look for more powerful motors.	Amazon.com: $200.00

OAK-D	OAK-D is a computer vision camera that is already optimized for AI identification. The camera can identify specific objects we want as well as identify how far they are. This will be perfect for determining how safe the area around the BES is. This camera will help us with our requirements on safety and determine the area is safe.	https://store.opencv.ai/products/oak-d: $199.99
12V Electric Bicycle Motor	12V Electric Bicycle Motors accompanied with gear ratios will create a significant amount of torque that allows our robot to move the weight it is in rough terrains. The fact that we will also be using treads will cause a significant amount of friction between the treads and the ground. With 2 motors and gear ratios, we will be able to reach our requirement of 2mph.	Amazon.com: $128.34
G-NIMO-003 Temperature Sensor	When researching temperature sensors this was the second cheapest variation of a temperature sensor and we were able to find the temperature rating. The temperature sensor will help us regulate the internal temperature of the robot and understand when our cooling system should be active.	Digi-key: $3.93
IMU MPU6050	The IMU that we will be using can be found on amazon. This sensor includes 3-axis gyroscope, 3-axis accelerometer, and 3-axis compass/magnetometer. This will help us with understanding the robot's position speed and level. This IMU will help us with the requirements 2.1.1.10, 2.1.3.2.	Amazon.com: $6.29
DC - DC Step Down LM2596	The LM2596 is a voltage regulator that will step the voltage of the battery from 12V to a voltage of 5V to connect the battery to the minicomputer.	Aliexperss.com: $1.93
GPS NEO-6M	The NEO-6M will be a good fit for our robot. This will help ROS understand our position if we ever want to use GPS	Amazon.com: $11.99

	tracking.	

Table 12: Components List

Updates:

Components List		
Name	Reason	Distributor and Price
PCB	Our PCB is built off of the Atmega2560 and is modeled to be a smalled Arduino mega2560. We needed a microcontroller with a lot of gpio pins and serial ports. The price of the PCB is large because of the cost of components and a version of the PCB.	jlcpcb.com: $89.60
Raspberry pi 4	The raspberry pi was chosen as the single board computer. These minicomputers are great options for booting ROS. There are also a large number of open source projects and packages we can utilize for raspberry pis. They are probably the more well-known minicomputer around. The drawback is how expensive or unobtainable they are.	Bestbuy.com: $134.99
RP LIDAR	RP LIDAR is a 360 degree LIDAR that we decided to use after we decided the BES will not be ontop of the robot. This will make it much easier to understand where obstacles are rather than using single point LIDAR.	Amazon.com: $99.19
BTS7960	BTS7960 is a high-power motor controller for motors that are drawing high amperage. This will be perfect for our 12V car battery and 12V motors. They will be drawing some significant power. The BTS7960 also has a heatsink that helps with the cooling of the system.	Amazon.com: $15.99
Zeee 9000mAh	This battery is an easy top use LIPO battery that can be recharged. This battery can give our motors bare minimum of a 1 hour life span. It is also much easier to recharge than other types	Amazon.com: $83.69

	of batteries.	
IMU BNO055	The IMU that we will be using can be found on amazon. This sensor includes 3-axis gyroscope, 3-axis accelerometer, and 3-axis compass/magnetometer. This will help us with understanding the robot's position speed and level. This IMU will help us with the requirements 2.1.1.10, 2.1.3.2.	Amazon.com: $33.70
DC - DC Step Down LM2596	The LM2596 is a voltage regulator that will step the voltage of the battery from 14.8V to a voltage of 5V to connect the battery to the minicomputer.	Amazon.com.com: $5.15
GPS NEO-6M	The NEO-6M will be a good fit for our robot. This will help ROS understand our position if we ever want to use GPS tracking.	Amazon.com: $11.99
Battery Bank	This is a simple rechargeable battery bank that can be purchased at any retailer.	Already owned
ESP32	This will handle the robots communication between the BES and the GUI.	Amazon.com: $55.00

Table 13: Components List Update

6.3.2 PCB

PCB or printed circuit boards are designed for custom projects. Many projects don't want to pay more than they have to when trying to make a specific circuit that does a specific thing. Why buy an Arduino if you don't plan on utilizing most of the capabilities it has to offer? Many people utilize printed circuit boards when building microcontrollers and we believe it may be our best plan of action. The microcontroller that we will be utilizing is the ATmega2560. This board is an absolute monster and is used in the Arduino mega series boards. The chip has a total of 54 digital input/output pins. With this many pins, we have plenty of room to work within our PCB.
The Schematic below in Figure 13: PCB Schematic shows how our PCB will be wired together.

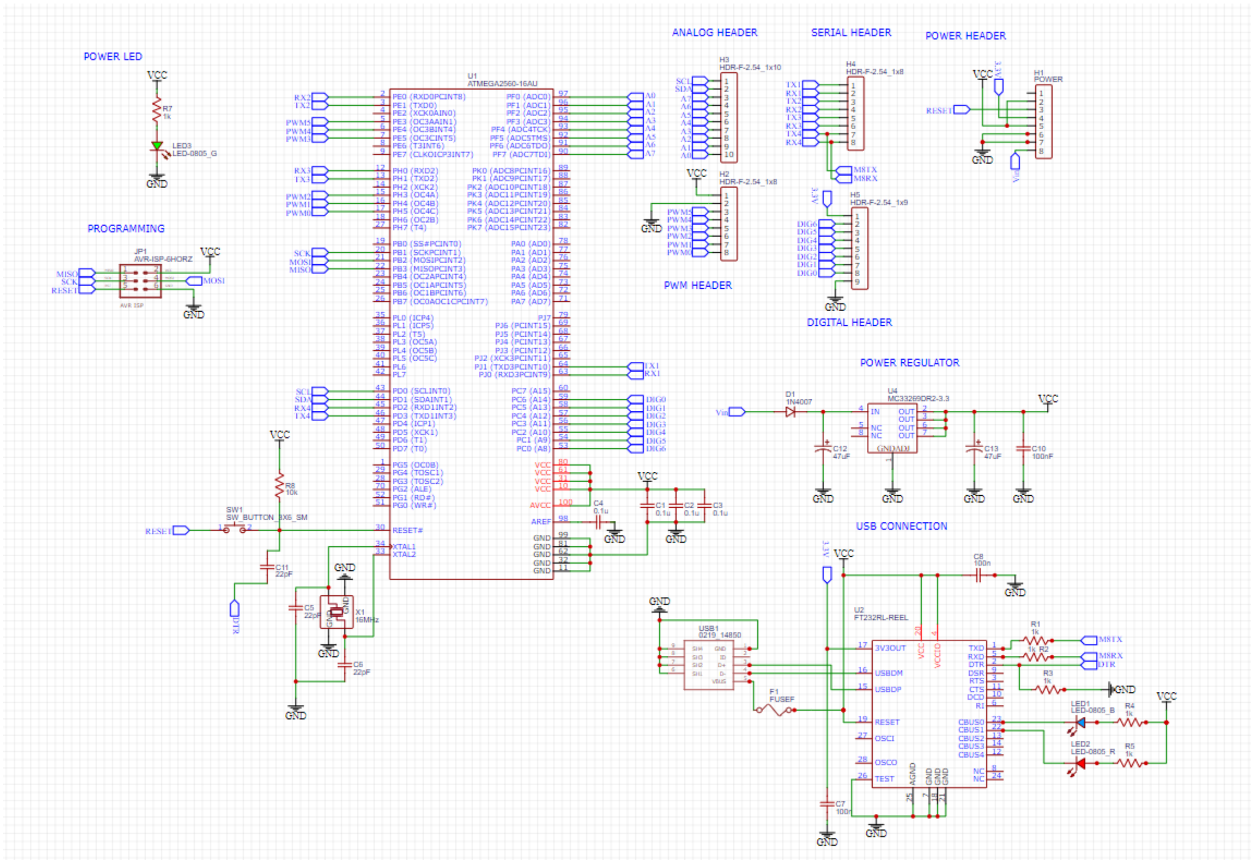


Figure 13: PCB Schematic

Taking a look at the schematic you can easily find the ATmega2560 in the center of the image. The software used to build this schematic is EasyEDA which links easily to LCSC when purchasing parts and later JLCPCB to print the board.

Starting with the clock in the bottom left of the ATmega2560 there is a 16MHz crystal paired with two 22pF capacitors. In the ATmega2560 datasheet, it shows the connections for a full swing crystal oscillator and what values it runs best with. In Figure 14: Recommended Circuit below those connections are displayed.

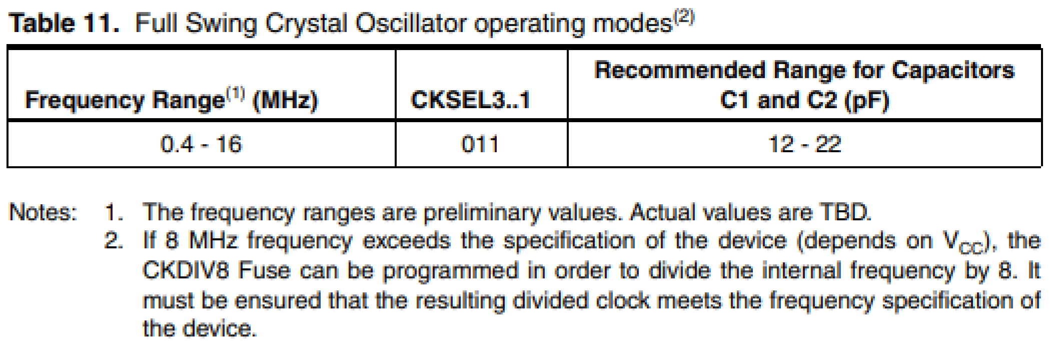


Table 11. Full Swing Crystal Oscillator operating modes[2]

Frequency Range[1] (MHz)	CKSEL3..1	Recommended Range for Capacitors C1 and C2 (pF)
0.4 - 16	011	12 - 22

Notes: 1. The frequency ranges are preliminary values. Actual values are TBD.
2. If 8 MHz frequency exceeds the specification of the device (depends on V_{CC}), the CKDIV8 Fuse can be programmed in order to divide the internal frequency by 8. It must be ensured that the resulting divided clock meets the frequency specification of the device.

Figure 23. Crystal Oscillator Connections

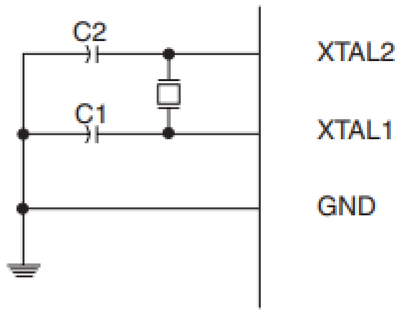


Figure 14: Recommended Circuit

As you can see the crystal range can be anything from .4-16MHz and the capacitors can be anything from 12-22pF. If a 22pF capacitor isn't used then the crystal clock won't be at 16MHz. We choose an external clock because the built-in oscillator will be too slow and very inaccurate compared to what we need. RC resonators would change too much in the heat that our robot will be enduring. Because cost and space are not worries for our board we decided that a quartz crystal is probably best for this project.

Next, we will take a look at the reset button in the top left. The reset button is will reset the board to its original state. The button itself will just be a push button. The reset button doesn't use the autoboot loader that Arduino uses because we will need to use the serial ports. If you use the auto boot loader every time the serial ports are accessed the auto boot loader will then reset the board and maybe a problem later on. The push button will then reset and will not affect the serial ports and every time code will be added you will need to use the reset button. This will be a push to ground causing the whole board to reset.

The board will be powered via micro USB which can be displayed in the bottom right. This will eliminate the need for a linear regulator and the USB will power with a 5V input supply. The USB connection is also giving and receiving serial data. The circuit shown attached to the FT232RL-REEL is modeled after the example circuit found in the chip's datasheet. Finally, there is a blue led that will light up whenever the USB is giving or receiving data.

I/O pins are the bread and butter of a microcontroller. It is the way that the microcontroller can communicate with the world. The Atmega has a plethora of pins that can do digital or analog as well as some that can do serial input. Let's first talk about the GPIO headers or general purpose I/O pins. In total, we have 5 headers. The analog header has a total of 8 analog pins as well as an SCL and SDA pin to help attach the IMU. The analog pins will be mostly used with the IMU. The serial header carries all 4 sets of RX TX pins that the ATmega2560 has. The pins will be mostly used with LIDAR, WIFI, and GPS and are attached to the USB communication device (FT232RL-REEL). The power header has a VCC pin or 5V pin a 3.3V pin a reset, ground, and Vin. The Digital header has 7 digital pins as well as a 3.3V pin and ground pin. These pins will be useful when attaching the IMU. Lastly is the PWM header that holds A VCC and ground pin as well as 6 PWM pins. These pins will be used to control each motor. Both motors will need to utilize a LPWM and RPWM. These header pins A0-A7(analog) are attached to pins PF0-PF7 respectively for analog pins and header pins DIG0-DIG6 are attached to the pins PC6-PC0 for digital pins respectively. The PWM header is the header that has pulse width modulation pins. The pins PWM0-PWM5 are attached to the chip's pins PH5, PH4, PH3, PE5, PE4, and PE3 respectively. The serial pins RX1-RX4 are connected to the pins PJ0, PE0, PH0, PD2. The serial pins TX1-TX4 are connected to the pins PJ1, PE1, PH1, PD3. The single-point LIDAR we are using will use serial input. The SCK MOSI and MISO will help us program the board to be attached to pins PB1-PB3. The SCL and SDA pins are pins that the IMU utilizes and are attached to the ATmega2560 at the pins PD0 and PD1. Below in Table 13: Pins there is a better visualization of what pins go where.

Table of Pins	
PIN Name In schematic	**Pin Name on ATmega2560**
A0 - A7	PF0 - PF7
DIG0 - DIG6	PC6 - PC0
PWM0 - PWM5	PH5, PH4, PH3, PE5, PE4, PE3
RX1 - RX4	PJ0, PE0, PH0, PD2
TX1 - TX4	PJ1, PE1, PH1, PD3
SCK	PB1
MOSI	PB2
MISO	PB3
SCL	PD0
SDA	PD1

Table 14: Pins

The decoupling capacitors can be found on the Vcc ports. The decoupling capacitors will help with noise and make sure that the microcontroller runs accurately to the 16MHz. These are to provide a local instantaneous charge source to prevent the voltage source from dipping and a bypass path that dampens ringing.

Lastly, because the chip is a surface mount part we will need a header to program the initial bootloader. On the left middle of the schematic you see an ICSP breakout with its labeled pins. The MISO, MOSI, and SCK are attached to the pins PB3, PB2, and PB1 respectively on the chip.

The fact that we are using an ATmega2560 allows us to put in more features later on. This will not only be good for if we overlook something but also good for the groups that will possibly be taking on this project in the future. When they are trying to build add-ons to the robot it won't be difficult with the number of pins that are still available on the chip. One other reason the ATmega2560 is a good choice is the number of serial ports. The TFmini plus LIDAR sensor utilizes serial communication. With more serial ports we can add more LIDAR sensors to our design.

Below Figure 15: PCB Front is an image of the finished PCB that we will be ordering as soon as senior design second semester starts. This figure shows all the wiring of both the front and back of the board with copper ground layering. The first figure is of the front of the PCB and where everything will be soldered and the second figure is the back of the PCB. The routing of the wires isn't the most efficient and doesn't follow some unspoken rules. This is however the first PCB we will be testing in our second semester.

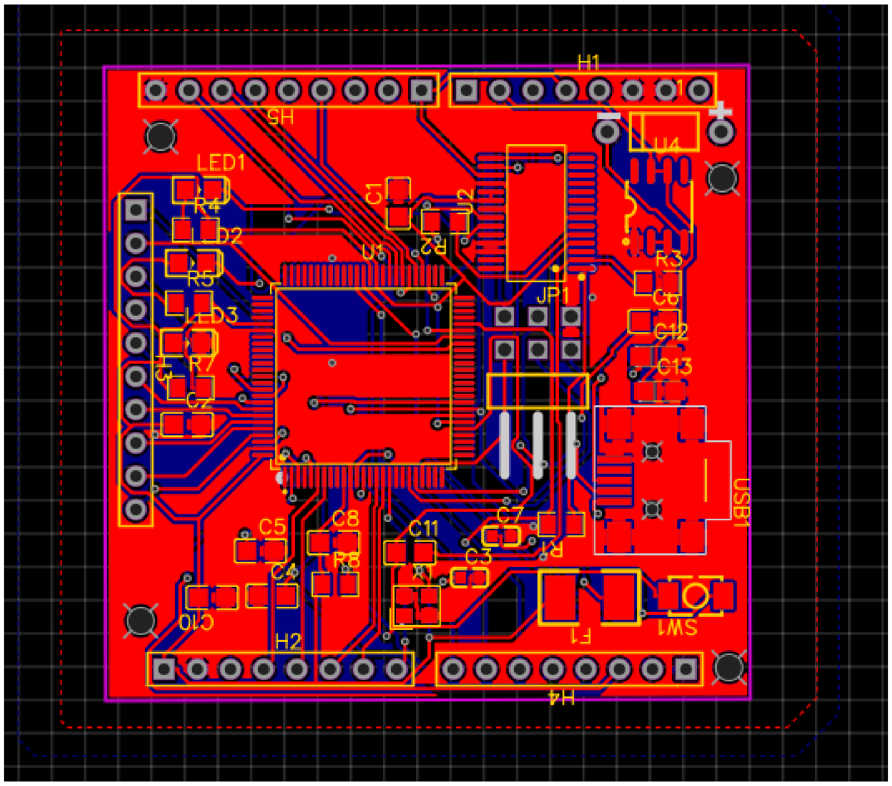


Figure 15: PCB Front

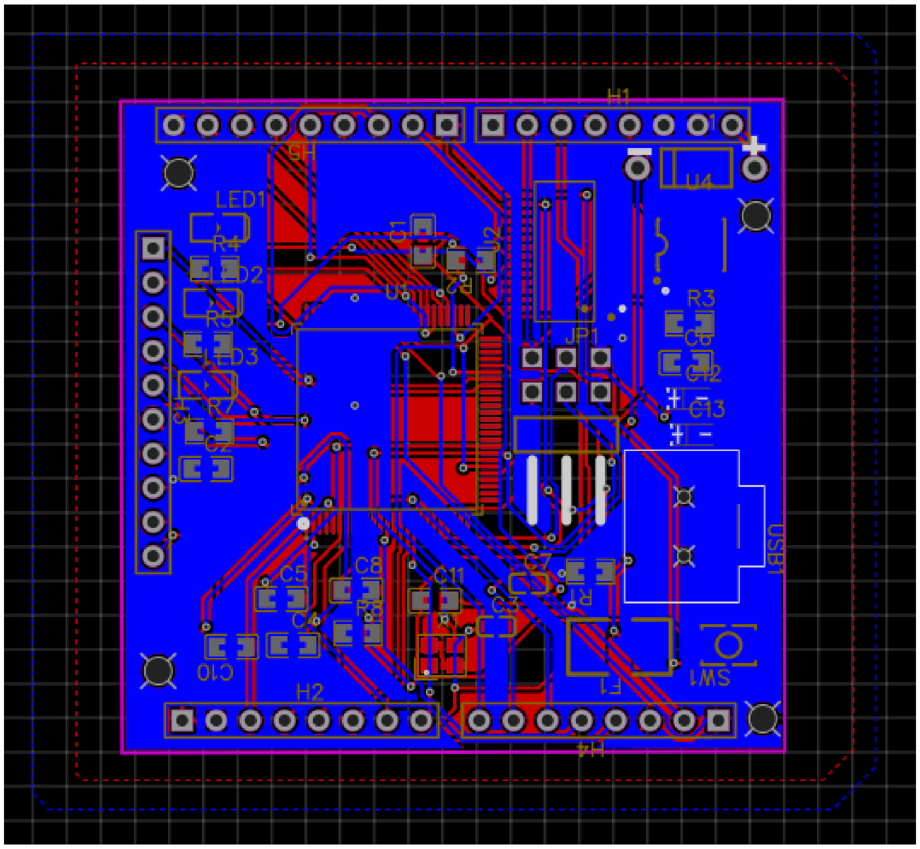


Figure 15: PCB Front

Throughout the paper whenever the term microcontroller is used or another microcontroller is displayed it is safe to assume this PCB microcontroller is the microcontroller that is being used.

All of the parts for the PCB will come out to around $35 dollars with a significant portion coming from the ATmega2560 chip itself.

Update: Throughout building the pcb there were a few errors that would make the initial board a dud. After making updates to the board this is what the schematic looks like.

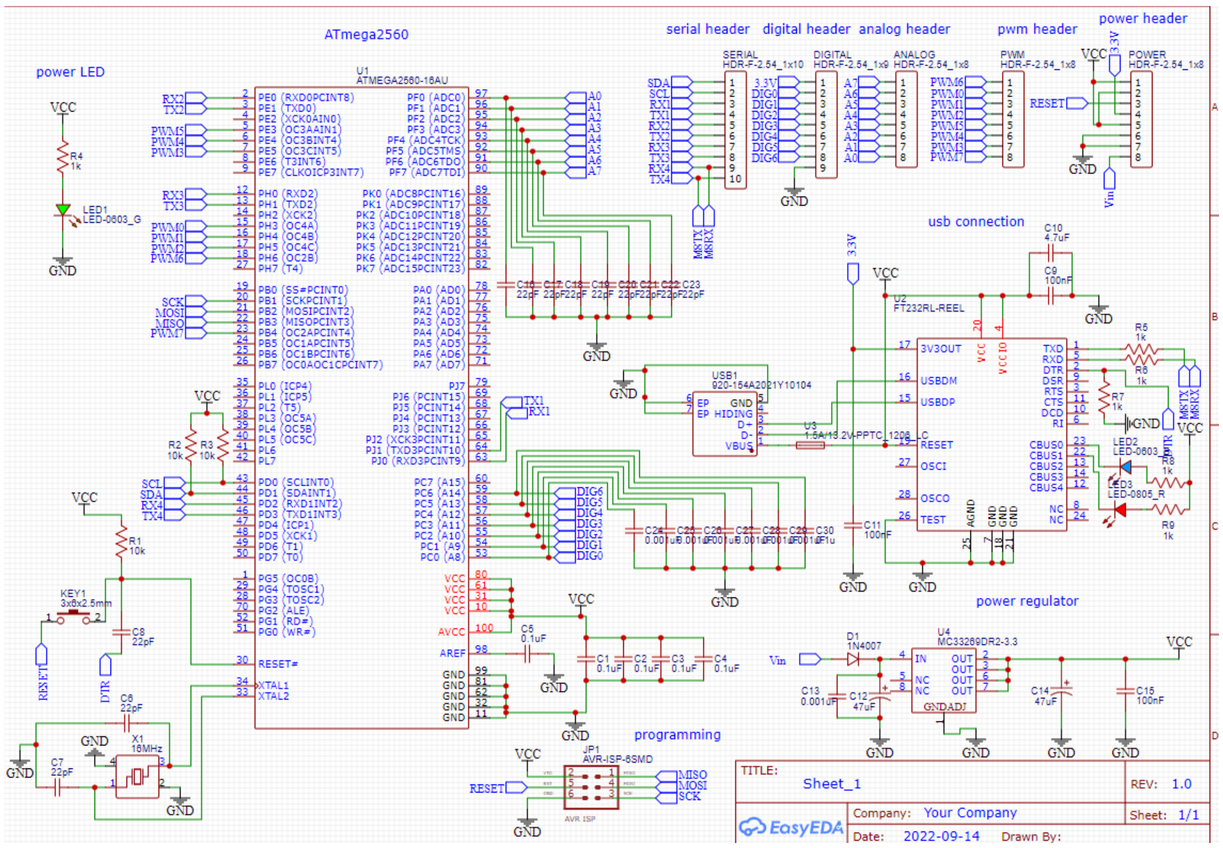


Figure 17: PCB Schematic Updated

As you can see there are decoupling capacitors havebeen added to digital and analog pins in order to reduce noise. Some headers have changed including adding 2 more PWM pins in order to account for 2 encoders and 2 motor drivers. SDA and SCL have also been moved to the serial header because it is another form of communication.

Several footprints have been edit in order to match the products that were purchased. Mainly the switch footprint was changed.

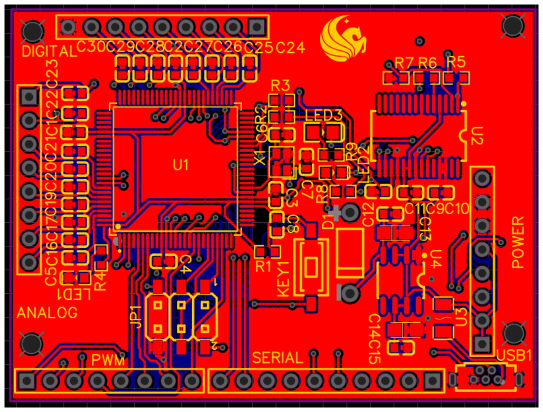


Figure 18: PCB Front Updated

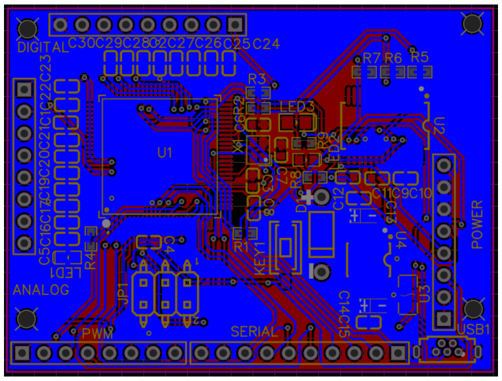


Figure 19: PCB Back Updated

The motor encoders have 4 other pins that are attached to the PCB. The encoder's positive power pins are PIN4(blue wire) and negative power PIN1(black wire). These pins on the motor encoders are connected to 5V and GND on the PCB. The last 2 pins on the motor encoders are interrupt pins. PIN2(yellow wire) and PIN3(green wire) on the left motor encoder will be connected to PWM7 and PWM3. The right motor encoder pins will be attached to PWM4 and PWM5.

Each BTS7960 motor controller has 6 pins that are connected to the PCB. The GND pin will be connected to the GND pin on the PCB. On each motor controller the VCC, R_EN and L_EN will be attached to the 5V pin on the PCB. Lastly the left motor controller RPWM and LPWM pins will be attached to PWM1 and PWM2. On the right motor controller the RPWM and LPWM pins will be attached to PWM6 and PWM0 on the PCB. These pins are all interrupt pins in order to stop the motor's process when ROS understands that the robot is in the position that is desired.

The ESP32 connected to TX3 and RX3.

On the PCB TX4 is connected to RX2 and RX4 is connected to TX2. This is to correct an error in the hardware.

6.3.3 Connectivity

Connectivity is a huge part of the design and it can go several ways. There are many components that can be connected in different ways. Diagram 6: Connectivity displayed our current best understanding of how our robots' hardware components will connect.

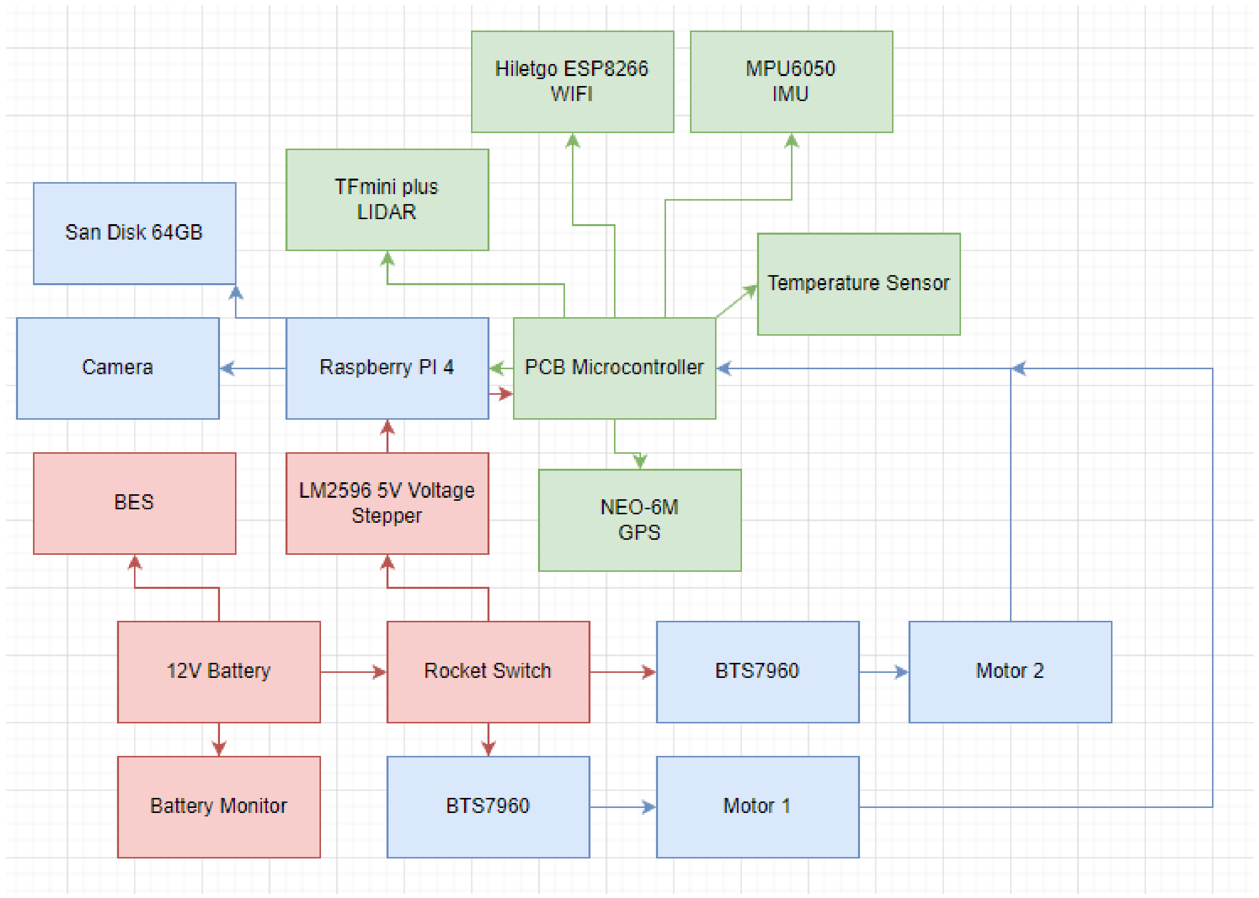


Diagram 10: Connectivity

One important thing to consider in this diagram is the LM2596 12V voltage stepper. This is a component that may not actually be needed. The stepper itself may offer a more stable 12V voltage for the raspberry pi as well as a stable 3A current. However the 12V battery already supplies the 12V.

Understanding how components will connect to each other is one of the key parts of the design. This can also be one of the most time-consuming portions because you have to look at component schematics to understand where they fit together. Starting with the most basic connection is the battery. What will the battery be connected to immediately without the use of a voltage regulator or step down? Using a 12V DC battery it will be directly connected to the 2 BTS7960 motor controllers and the BES.

The BES will however need a safety signal in order for it to turn on. This will be made possible with a simple and gate logic circuit. Only when the safety signal is passed to the circuit and the circuit has power from the battery then the BES will be powered.

The battery will have a green LED light to show that there is power being supplied to the robot. When the BES is turned on this LED will become red. The green signifies that the robot is on and safe to approach and the red LED will signify the robot is powered on and it is not safe to approach because the user is capable of firing the BES.

These motor controllers are built for high-power motors and are capable of a maximum 27V and 43A in. 2 12V DC electric bicycle motors will then be connected to the BTS7960. These motors will be accompanied by gears to gear ratio our torque to an acceptable amount and reach our goal of 2mph. The connection of the battery to the motors will look similar to Figure 17: Battery Connections.

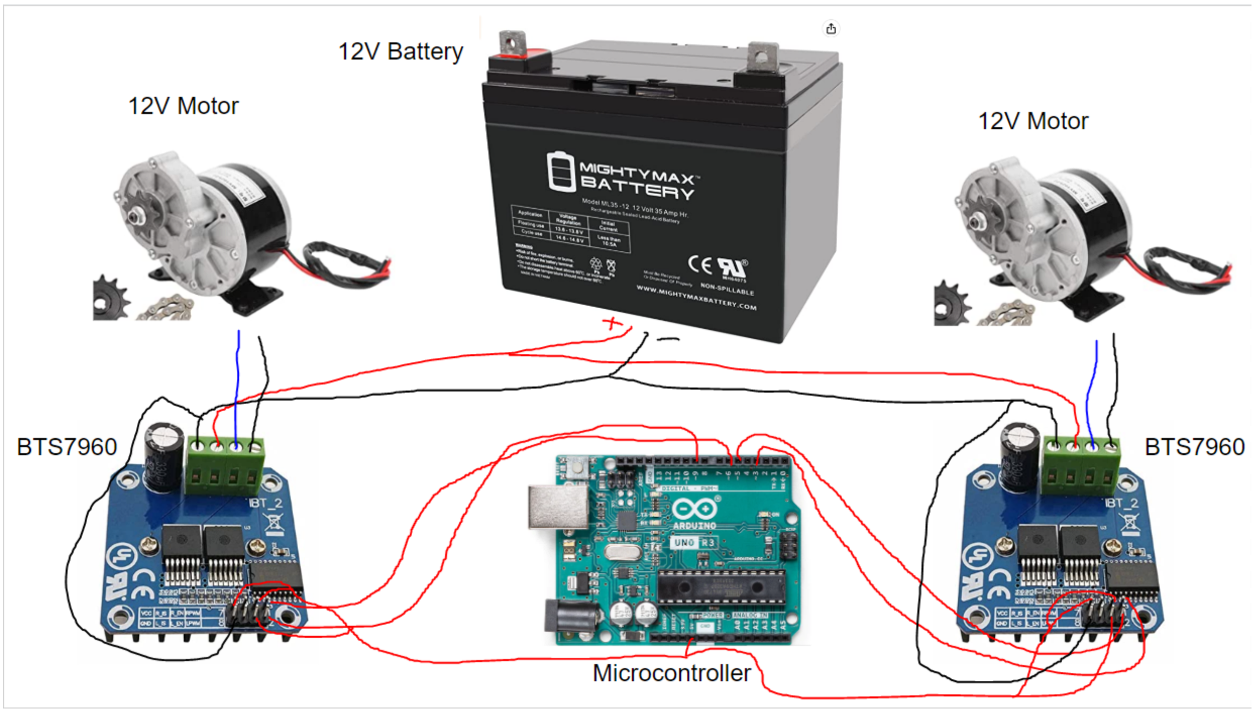


Figure 20: Battery Connections

As you can see in the figure above the Arduino UNO is representing our PCB that utilizes an ATmega2560. This isn't a big issue because they will still be using pins that have similar functionality. The ATmega2560 will be connected to both of the motor controllers. Each motor controller will be utilizing Arduinos PWM pins. The first BTS7960 will utilize pins PWM0 and PWM1 of the Arduino for its RPWM and LPWM pins. The second BTS7960 will be utilizing pins PWM2 and PWM3 of the Arduino for its RPWM and LPWM pins. Each of the motor controllers' VCC, L_EN, and R_EN pins will be utilizing the 5V pin on the Arduino and the ground will be attached to its own ground. The Arduino will then be connected to the raspberry pi that is running ROS.

The Raspberry pi 4 will be connected to the battery through a voltage regulator. The voltage regulator will take the 12V voltage of the battery and bring it to a voltage anywhere between 3V and 40V. We will be using 5V to power the raspberry pi. The raspberry pi will also utilize 2.5A and the step-down voltage circuit we have chosen (LM2596) delivers that as well. We will have to connect and make an adapter that fits the needs of our raspberry pi from our step-down voltage circuit. The raspberry pi will then power the microcontroller via micro USB. In Figure 18: Power Connections below you can see the battery and how the power will get to the microcontroller and raspberry pi. The battery is separated by the voltage steppers and switch that will control the power.

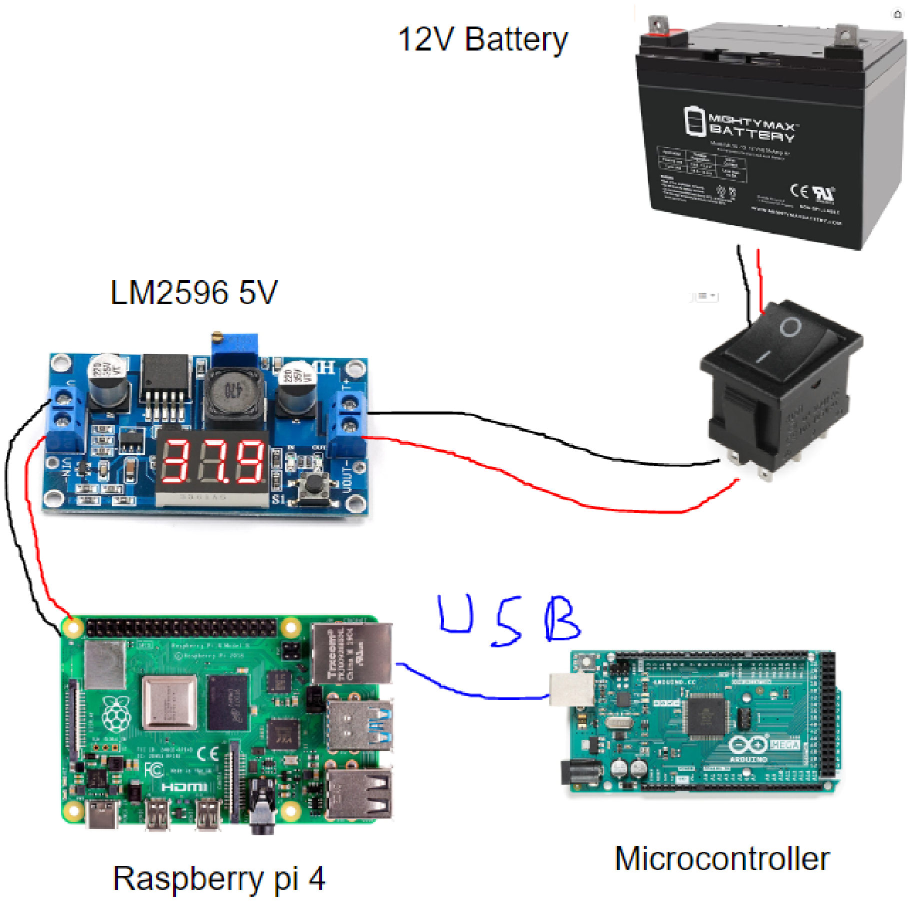


Figure 21: Power Connections

The battery will also be directly attached to the BES. The BES will however need a safety signal in order for it to turn on. This will be made possible with a simple and gate logic circuit. Only when the safety signal is passed to the circuit and the circuit has power from the battery then the BES will be powered.

The battery will have a green LED light to show that there is power being supplied to the robot. When the BES is turned on this LED will become red. The green signifies that the robot is on and safe to approach and the red LED will signify the robot is powered on and it is not safe to approach because the user is capable of firing the BES.

Communication between the control system and the BES will be helped on a windows tablet and special board that will possibly be provided by PEO STRI. If the board and tablet are not obtainable by PEO STRI then the BES will be on a tether. This ethernet cable will be a maximum of 300ft and connect directly to the BES that way there is no need for a DHCP server. This makes the communication process more achievable in the time that we have to build the robot. It also cuts the cost of the robot.

The IMU MPU6050 will be attached directly to the microcontroller. This will be hooked up utilizing Vcc, GND, SCL, SDA, and a single digital GPIO pin.

HiLetgo ESP8266 will utilize serial input pins on our microcontroller in order to get WIFI. The pins needed will be Vcc, GND, RX, and TX.

TFmini plus LIDAR will also be attached to serial pins and will be utilizing the same pins listed as the HiLetgo wifi however it will use different serial pins.

GPS NEO-6M will also be attached to serial pins and will be utilizing the same pins listed as the HiLetgo wifi however it will use different serial pins.

The OAK-D camera will be attached to the raspberry pi 4 through USB. This could be an issue later on because of the power the OAK-D needs in order for it to work.

Updates: The Diagram below depicts the robots connectivity and its subsystems. In red is our power system. Blue is our sensory system. Green is our communication system and finally yellow is our drive system.

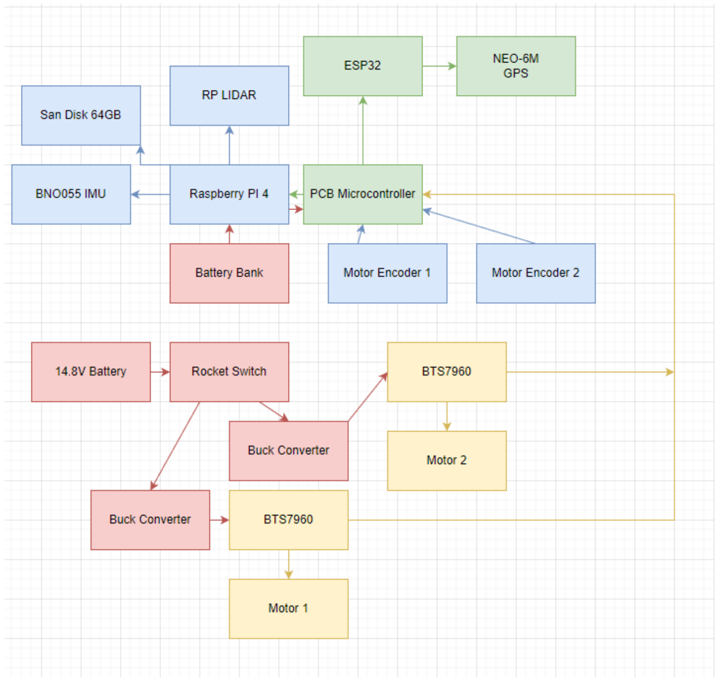


Diagram 11: Connectivity Updated

The Raspberry Pi is connected via USB-C to a battery bank in order for a stable connection and power supply. This also gives the robot a much longer run time because of the second battery.

The IMU BNO055 is connected to the Raspberry Pi directly via I2C connection.

ESP32 is connected to the PCB via serial connection to RX3 and TX3.

RP LIDAR is connected directly to the Raspberry Pi via USB to TTL serial coneverter.

GPS NEO-6M will also be attached to serial pins and is connected to the ESP32.

The OAK-D camera is not utilized.

6.4 Operating System Design

ROS has thousands of libraries to help people build their robot applications. This section is intended to list the general hardware and software we will be using that will interact with ROS to make the robot autonomously navigate. After the pieces that are needed for this robot to navigate are determined, this section will go into depth on which ROS packages will be used for this project.

6.4.1 ROS Navigation

Below is a list of the basic components/software that will be used for the robot to navigate autonomously with ROS.

1. Sensors: In order to perform navigation, ROS requires a map of the environment the robot will navigate in. This map will be created by the robot using sensor readings to gather information about the environment. This is referred to as “mapping.” This can be done by launching the ROS map builder program. The sensor data that ROS supports are laser scans, such as lidar, or point cloud, such as depth camera or stereo camera.
2. Odometry: Odometry gets an estimated robot position/orientation with respect to the starting position. This is done using data from an encoder. Common encoders that are supported by ROS used to create autonomous robots include magnetic encoders and optical encoders.
3. Base Controller: The base controller takes velocities from the local lanner and converts them to velocities for each wheel (left and right). Then they are transformed into motor commands to make the wheels move.
4. Map Server: The map server reads the map that was created by the robot and sends the map information to the planner to generate a path for the robot to travel on.
5. Adaptive Monte Carlo Localization (AMCL): Determines the correct position of the robot on the map. This is done using the sensor information, odometry, and the map itself.
6. Move Base: Components that work together to allow for the robot to move from point A to point B successfully. It is comprised of five main components:
    a. Global Planner: Performs functions such as finding the shortest path from point A to point B considering non-mobile objects.
    b. Local Planner: Takes into consideration moving obstacles when finding the most optimal path to the destination.
    c. Global Cost maps: Takes into consideration the non-mobile obstacles in the path and inflates them to create a path that will easily avoid it.
    d. Local Cost maps: Performs functions such as optimizing paths by avoiding obstacles and collisions. This is done using laser data.
    e. Recover Behavior: This allows the robot to have a set of operations that can be performed if the robot is stuck due to an obstacle.

6.4.2 ROS Packages Used

Below in Table 14: ROS Packages is a list of the ROS packages we will be using to help build this robot, a brief description of them, and the necessary requirements to implement them.

ROS Package	Package Description	Requirements to Use Package
ROS Navigation Stack	The ROS Navigation stack uses odometry and sensor stream information to send velocity commands to the robot to allow movement. See section BLANK for more information on the Navigation stack.	1. The mobile robot must be a differential drive or holonomic wheeled robot. 2. The mobile robot must have a planar laser mounted on the mobile base (to be used for map building and localization) 3. The robot must be generally a square or circular shape. It may have difficulty with robots of large, rectangular shape. 4. The robot must have a tf transform tree in place.
ROS Gmapping Package	The Gmapping package provides laser-based Simultaneous Localization and Mapping (SLAM). This is done through the ROS node, slam_gmapping. According to ROS.org, this node allows you to create a 2-D occupancy grid map from the laser data collected by the robot, and position of the robot.	1. The mobile robot must provide odometry data. 2. The mobile robot must include a horizontally-mounted fixed, laser range-finder.

ROS Gazebo Packages **gazebo_ros_pkgs**	Gazebo is a simulation tool. It has ROS integration that will automatically converts between Gazebo and ROS protobuf messages. The packages required to use Gazebo are the gazebo_ros_pkgs. These packages provide the necessary interfaces to simulate a robot in Gazebo using ROS features.	1. N/A other than ROS installation.
ROS tf Package	Both the ROS Navigation Stack and the ROS Gmapping package require the robot to have a tf transform. The tf package lets a user keep track of multiple frames during a period of time. According to ROS.org, “tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time”	1. The Gmapping package requires a fixed value, broadcasted by the root_state_publisher or the tf static_transform_publisher.
Rosserial Package	This package allows for communication between the Arduino board and ROS running on a machine. According to ROS.org, it is a “protocol for wrapping standard ROS serialized messages and multiplexing multiple topics and services over a character device such as a serial port or network socket.”	1. Requires ros_lib Arduino library for Arduino board to communicate with ROS.

ROS Library Package **ros_lib**	The ros_lib package allows for the Arduino and the ROS to communicate.	1. Install on a Linux or Windows system.
AMCL Package	The AMCL package implements the Adaptive Monte Carlo Localization approach. This is a localization system for moving a robot in 2D. According to ROS.org, this package "uses a particle filter to track the pose of a robot against a known map."	1. Robot must have the ability to provide laser scans. 2. This package must take in a laser-based map.
RVIZ	RVIZ is a ROS graphical interface 3D visualization environment. It basically allows for the user to view what the robot is seeing, thinking, and doing. It allows for the visualizations of things such as grids, maps, robot models, TF hierarchies, and many other things.	1. Robot must utilize cameras, lasers, or encoders.

Table 15: ROS Packages

6.4.3 Part Communication with ROS

This section will discuss how our chosen parts for the robot are communicating/interacting with ROS

Arduino Mega2560 and ROS Communication

Our group has chosen to use an Arduino board as our microcontroller, as discussed in section 5.3.1. One of the reasons this board was chosen is due to its ability to communicate with our chosen operating system, ROS. The Arduino controller performs low-level/embedded control and sensing. Figure 19 below from ROS.org shows what control the Arduino board includes. A ROS package will be used to allow ROS and Arduino to communicate. This package utilizes the Arduino UART communication to convert the board to a ROS node. That ROS node is then able to publish the ROS messages and subscribe to ROS messages. The Arduino ROS node publisher sends data from the sensor or robot state from the Arduino to the machine that ROS is running on. The Arduino ROS node subscriber will get instructions from the machine running ROS.

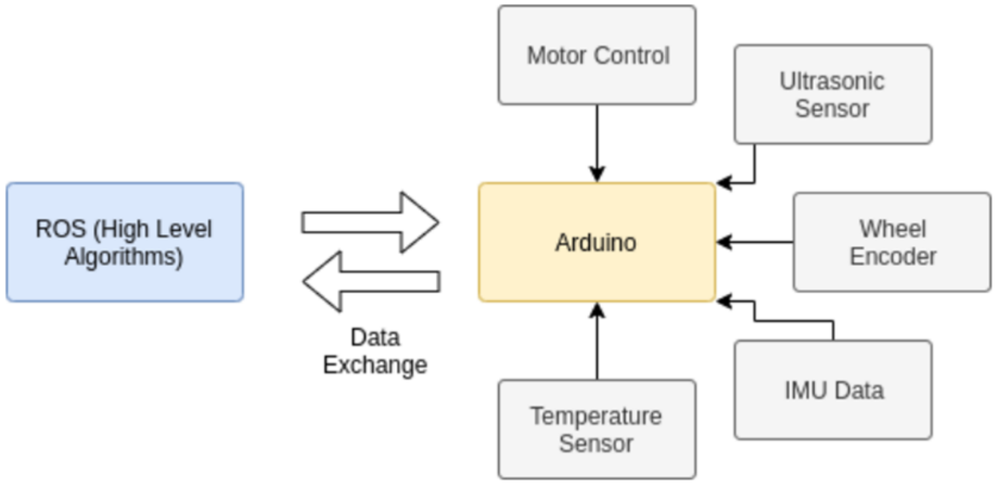


Figure 22: ROS/Arduino Communication and Control

Raspberry Pi 4 and Arduino with ROS
Our group has chosen to use the Raspberry Pi 4 as our Minicomputer for this project. The Raspberry Pi 4 being our minicomputer and the Arduino Mega2560 being our microcontroller means that these two devices will have to communicate with each other. The Raspberry Pi 4 will be running the ROS navigation stack, and the Arduino Mega2560 will communicate with ROS and control certain components of the robot. This communication is called Serial Communication and is done with Universal Asynchronous Reception and Transmission (UART) protocol. This is a multi-master protocol that allows the connected devices to send data when necessary. This is key to success between both of these devices, they both need to communicate with each other, which is why the multi-master protocol was chosen over the slave-master protocol. These boards are simple to physically connect, they just require a USB between the two boards. The Arduino will need to first be connected to your computer and have code uploaded to it before the two boards can be physically connected though.

6.4.4 Full Autonomous Stack Set Up

This section will discuss the process of getting our full autonomous stack running among our robot, Raspberry Pi, Arduino, and Linux virtual machine.

1. First, we will need to install Robot Operating System (ROS)
    a. Download Ubuntu with pre-installed ROS. This is done by downloading the image of your choice and flashing it to an SD card (Etcher is a good option for this).

    b. Next, we will connect our computer to our Raspberry Pi 4 WiFi network.
    c. We will then connect our Raspberry Pi to our computer using a command in the terminal.
    d. Finally, we will run the command `roscore` to ensure everything is working as it should without any warnings or errors.
2. After installing ROS, we need to be able to remotely connect to it. This will allow us to access ROS communication messages from our laptop. This will also allow us to use RVIZ. Please see RVIZ in the Operating System Design section for more information.
    a. We will need to spin a Linux machine using our ROS distribution of choice, ROS Melodic Morenia. This can be done using a virtual machine or a real machine. For the sake of this project, we will be using a Linux virtual machine. We will be using the ROS Melodic Morenia distribution because it is supported on Ubuntu Bionic 18.04 which also supports the Raspberry Pi 4. We will be referring to the virtual machine as the Observer machine, and our robot computer used to run ROS, the Raspberry Pi 4, will be the Master machine.
    b. Once step a is completed, we will find the ROS_IP, the IP address of the master computer, and ROS_MASTER_URI. These will be used to allow for the Observer and Master to communicate.
        i. For the Raspberry Pi (Master): ROS_IP is the Raspberry Pi IP address that we will set using commands and ROS_MASTER_URI has the address of the Master machine.
        ii. For the Linux VM (Observer): ROS_IP is the Observer machine IP address and ROS_MASTER_UI is the robot master machine UI.
        iii. Our ROS Master is controlling the communication so we exported the ROS_MASTER_URI and ROS_IP (IP for Raspberry Pi). In order for the observer to connect to the Master we then executed the ROS_MASTER_UI and ROS_IP (IP for observer) on that.
    c. After step b, we will then need to connect our Raspberry Pi 4 and computer to WiFi so they are able to communicate over SSH.
        i. Use the `pifi add` command to connect to your WiFi.
        ii. Restart the Raspberry Pi and it will connect to the WiFi as it turns on.
        iii. Connect your computer to the same network as the Raspberry Pi.
3. After connecting the lidar to WiFi, we need to next get the lidar running. To do this we need to install the necessary package for our lidar, tfmini, to run with Raspberry Pi and ROS.
4. After testing the lidar, we have to connect ROS and the Arduino Mega 2560. This will allow for Raspberry Pi to send commands to the Arduino. To do this, we will use the rosserial ROS package. More information about this package is in section 5.4.2 ROS Packages Used. First, we will install ROS for the Arduino. In the Arduino IDE install the

ROS rosserial library. After the installation, we will test it to ensure it is working correctly and that we can see the messages being sent.

5. After ROS and the Arduino are able to communicate, we can then install the Hector-SLAM ROS package. This package allows for us to create maps for localization and navigation using the lidar.
6. After setting up the SLAM package, it is time to set up the ROS package that will allow for the robot to actually physically move now. We are creating a node that will run on the Arduino. The node will then listen for commands from they keyboard that are being sent to the robot. This allows the robot to be controlled from a computer.
    a. After setting up SLAM, we will create launch files to make the set-up process easier and quicker.
7. At this point, we can finally start creating and saving maps. First, we have to download the map server onto our Raspberry Pi. Next, we will get the robot and the lidar spun up. To record a map, we will execute the appropriate commands and move the robot around the area we want to map slowly until we have an accurate map simulation on RVIZ. We will then save it to a safe location on our computer to use for later.
8. Next, we need to allow the navigation stack to localize the robot. To do this, we will give it access to the map that we just created by launching the map. To then localize the robot in the map that was created, we will be using the lidar sensor, odometry, and the map itself.
    a. We need to extract our odometry data from inertial measurement units (IMUs). To collect this data from the IMU, we will use code to get the sensor data and combine it into a single string. It is then ready to be sent to the ROS node. This data will then be extracted and converted in the subscriber node into integers. The Arduino is a publisher node and it is sending the readings and the subscriber node receives the data and sends it to the ROS node. Figure 20 below from ROS Wiki shows the process of extracting and using the IMU data.

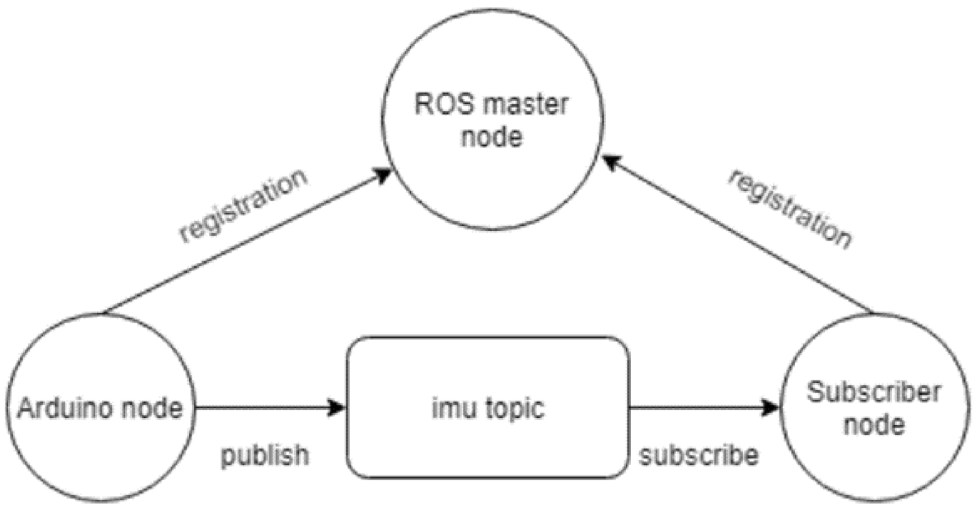


Figure 23: Extracting and Publishing IMU Data

b. The other option for extracting our odometry data is through wheel encoders. If we were to use this option, we would need to create an odometry publisher based on wheel encoders. To do this, we will use the tick count from the right and left motor encoders and the initial position/orientation of the robot. When it is time to "publish" the odometry data, we will publish the orientation z variable. This is an Euler angle. We will also publish the position and velocity estimate

9. The next step in localizing the robot in a map is to provide the AMCL package with the odometry information and the previously created map. Please see section 5.4.2 ROS Packages Used for more information about this package. This is also the time we will use the transform trees.
    a. Once this portion is done being set up, we will test it to ensure it is working. Using RVIZ, we can publish the 2D Pose Estimate. From there, we can click on the robot location and move the robot around the map. We will check to make sure the pose of the robot is changing as well to accurately reflect where the robot is.
10. Finally, the ROS Navigation stack is used to allow the robot to physically move autonomously around the map. The robot will use the following files to plan paths with obstacles:
    - base_local_planner_params.yaml
    - costmap_common_params.yaml
    - |global_costmap_params.yaml
    - local_costmap_params.yaml

    These files have the robot dimensions, intended velocity of the robot, and parameters when navigating in maps.
11. Once the Navigation Stack is implemented, we can open RVIZ, set a navigation goal, and watch as our robot autonomously navigates there.

6.4.5 Safety

The most important thing when designing the software for the robot is to maintain safety and not put anyone at risk. This is a tradeoff because by putting extra safety checks into the code the performance or speed at which the task is accomplished will be reduced. Due to the nature of the BES we can not afford for there to be any misfiring or other unintended action that can cause harm. The following safety features we plan to implement in the code:

1. There will be two states for the robot to be in which are Safe Mode and Fire Mode. The default mode when the robot is powered on is Safe Mode. During this mode, it will be impossible for the BES to launch and it will send an error message back to the user.
2. In order to initiate the firing sequence Fire Mode must be enabled which is done through the user interface. Fire Mode is basically the first check in the firing sequence and if it's not enabled the sequence will not begin.

3. The second check in the fire sequence is making sure the robot is on a level surface within a margin of 10 degrees. This will be done using the level sensor and this step is also important to make sure that the camera will be able to see the area in clear for the next step.
4. The third check in the firing sequence is using the AI and cameras to scan the area in 360 degrees for any person within 10 meters. Once the robot has determined the area is clear of people the BES will launch.
5. When the robot is moving to specific coordinates there will be multiple Lidar sensors that track the distance between potential objects/people. The robot will not move at full speed when objects are near and this is to prevent collisions especially with moving objects like a person.
6. There will be a kill button that the user will be able to press on the tablet that will stop whatever action the robot was doing. This will effectively terminate the task whether it was traveling to a location or scanning the area for people.

There will be some redundancy in some features but it is important to have multiple safety checks. For example, there will be a switch on the robot that makes it physically impossible to give power to the BES and there will also be a software lock that prevents the launch. This is important because even if one of the safety checks were to fail the other would still prevent the robot from causing harm.

6.5 Image Processing

This section is intended to explain how the people detection will work using the onboard camera and microprocessor. It will go over the hardware and software used in the process for the robot to determine if a person is present when doing the launch sequence.

Update: Due to the already high demands of this robot, our limited time frame, and budget cuts, our sponsors recommended that we cut the camera from the project altogether. Implementing a camera that would detect humans using AI would take significant time and our sponsor preferred we focus on other aspects of the project such as the BES communication and the operating system.

6.5.1 Camera

After our research on cameras and microprocessors we have concluded the best option that will meet our budget and have enough power for the task is one OAK-D as shown in Figure 21. The people detection operations will be performed on the camera which means the microprocessor will not need to have a GPU. The OAK-D is capable of specifically detecting people at about 15 frames per second. It is powered directly through the microprocessor USB which means we will not need an external power supply for it. The OAK-D supports USB C 2.0 and 3.0, we will use a

3.0 cable which will bring the benefits of faster transfer speed and power efficiency. This will allow us to spin the robot in 360 degrees and capture the whole area with just one camera.

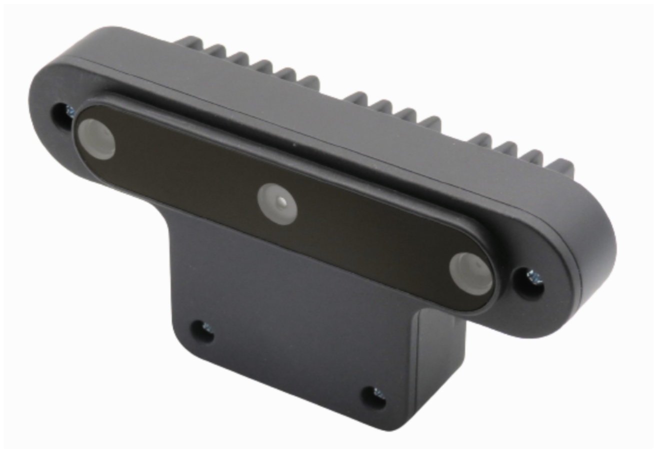


Figure 24: OAK-D Camera
Permission from Luxonis

The tradeoff to using only one camera is that the process of detecting people will take a maximum of four times as much compared to using four cameras. This is because with one camera the robot will spin 360 degrees, whereas with four cameras the robot would spin 90 degrees. The cameras do consume a lot of power meaning the battery performance of the robot will be improved with our decision. We decided to use only one camera because the max power consumption of the OAK-D is approximately 1A. The raspberry pi 4 will only support 1.2A through USB ports meaning we would need a more expensive microprocessor like the Jetson Xaiver which can supply 5A to the USB ports. The microprocessors are currently around twice their normal price so we do not think it is worth the slight speed up. In the future, if it is deemed the process of spinning the robot is too slow, more cameras can be added to reduce the number of degrees needed to spin the robot.

This OAK-D is designed specifically for spatial AI tasks which is why it incorporates two stereo cameras allowing it to sense the depth of an image. The main camera in the middle is an RGB camera which is what the AI uses to determine if an object is there. The left and right cameras are used to determine the distance of the object from the camera by using different angles. This function will be critical as we need to determine the distance from the person in order to see if they are closer than 10 meters. The way this is represented is like a heat map where the person gets brighter the closer they get to the camera.

6.5.2 OpenCV

OpenCV stands for Open Source Computer Vision Library which basically contains multiple functions that will allow us to interact with the OAK-D. This is how we will specify the frame rate, resolution, and algorithms that will need to be run for people detection. This library is compatible with Python and C++, we will be using Python for our AI tasks. The main tasks that OpenCV will allow us to accomplish are:

1. Image Manipulation - OpenCV allows us to recognize images in Python. There are many functions that will allow us to resize images to different dimensions, rotate, or translate. This will potentially be used to translate the frames from the camera into the correct dimensions that were used when the CNN was trained.
2. Control Resolution - The OAK-D technically has three cameras and we may not want all of the cameras to run at the resolution. The main middle camera should be at least 1080p because that is what the CNN model will use to determine if a person is present. The left and right cameras are only used for depth perception so we could run them at a lower resolution like 720p which will save bandwidth and processing power.
3. Control Frame Rate - We will want to synchronize the framerate of the cameras with the rate at which the CNN model can process the frames. This will need to be done dynamically and we can use OpenCV to control the camera's frame rate.

6.5.3 CNN Model

The CNN Model is basically how the AI was trained to recognize different things. Different models can be used for different tasks and it will affect the amount of processing that the AI needs to do on each frame. It is possible to create a model ourselves however, it would likely not be accurate due to needing powerful GPUs in order to train across large datasets. For example, the YOLO-V3 (You Only Look Once - Version 3) model was trained across 80 classes of objects meaning it's capable of detecting humans, animals, vehicles, and more. The downside to this is that by looking for all of these different objects on each frame of a video it would greatly increase the processing power needed. For our use case, we want to specifically look for people and not waste resource processing if there are cars. The models below in Table 15 are relevant to our purpose of verifying its safe to launch.

Model	Purpose	FPS on OAK-D
deeplabv3p_person	The purpose of this model is People Segmentation which is basically taking an image as an input and the output is just two colors representing pixels with the person and the background representing pixels without the person (usually black).	22.1
pedestrian-detection-adas-0002	The purpose of this model is People Detection which is basically taking an image as an input and the output is a box around all people in the picture.	13.1
person-detection-retail-0013	This model generally achieves the same output as the pedestrian but it sacrifices speed for accuracy.	10.7
yolo-v3	The purpose of this model is general object detection where it will take an image as an input and place a box around various different animals/people/vehicles.	1.9
tiny-yolo-v3	This model has the same intention as yolo-v3 but it greatly sacrifices accuracy for speed in order to run on embedded devices.	29.9

Table 16: CNN Models

We will experiment with the various CNN models listed above in order to find a balance between accuracy and speed. By having a greater FPS we will be able to spin the robot in 360 degrees faster because more frames will be able to be processed. We also may modify a model like the yolo in order to also account for animals in the area like dogs, cats, and cows while excluding objects like apples, crosswalks, etc. This will come down to lots of trial and error to find the best model for our purpose.

6.6 User Interface

The user interface or app will be very important for controlling the robot and telling it what to do. It will also be useful for receiving data from the various sensors on the robot so that it can be monitored. In the research section, we went over the three main platforms of IOS, Android, and Windows and we have decided on using a Windows tablet. The main reason for this decision is because there are many security/privacy restraints placed on IOS devices that will likely cause issues connecting to the microcontroller which the phone will not recognize as a trusted device. Additionally, if we used an Android device we would be more constrained to follow their functions and standards for designing apps. By using Windows we can use Java to create an application that will connect with the robot through Bluetooth. We will also be able to run it on a laptop or tablet depending on what is available meaning we will not have to add a mobile device to the cost of the project. In the future, it will be possible for other teams to create an app that works with IOS and Android but now we will be solely focusing on Windows for the user interface.

Our Sponsor also has base code that they use to communicate with the BES and they currently use it on a Windows machine through Java. By using the same platform we will be able to use this code as the backend as we will be able to use the already built functions for transmitting and receiving data to the microprocessor and BES. This is desirable as it will allow us to design the User Interface on the same platform we will be programming on and reduce the work spent on creating an app for a mobile device. The main task for us will be designing the Graphical User Interface which will call these backend functions depending on what the user inputs.

6.6.1 Home Tab

The home tab is the default tab that will be displayed when running the application as shown in Figure 22. This is where the user will mainly be when communicating with the Robot and it will contain all of the critical actions including powering the BES, traveling, and launch sequence. Powering the BES will be a toggle meaning the red outline around the true or false will represent whether power is being given to the BES currently. This is also how the tabs at the bottom work including the Home, Sensor Data, and More details where the red outline will represent which tab the user is currently viewing. The green buttons are used for connecting and disconnecting from the robot and the travel/start sequence. In order to initiate the travel sequence, the user must input the coordinates of the location they want the robot to travel to. It will verify if the coordinates are valid when the user clicks the travel button, and if they are invalid it will display a popup message.

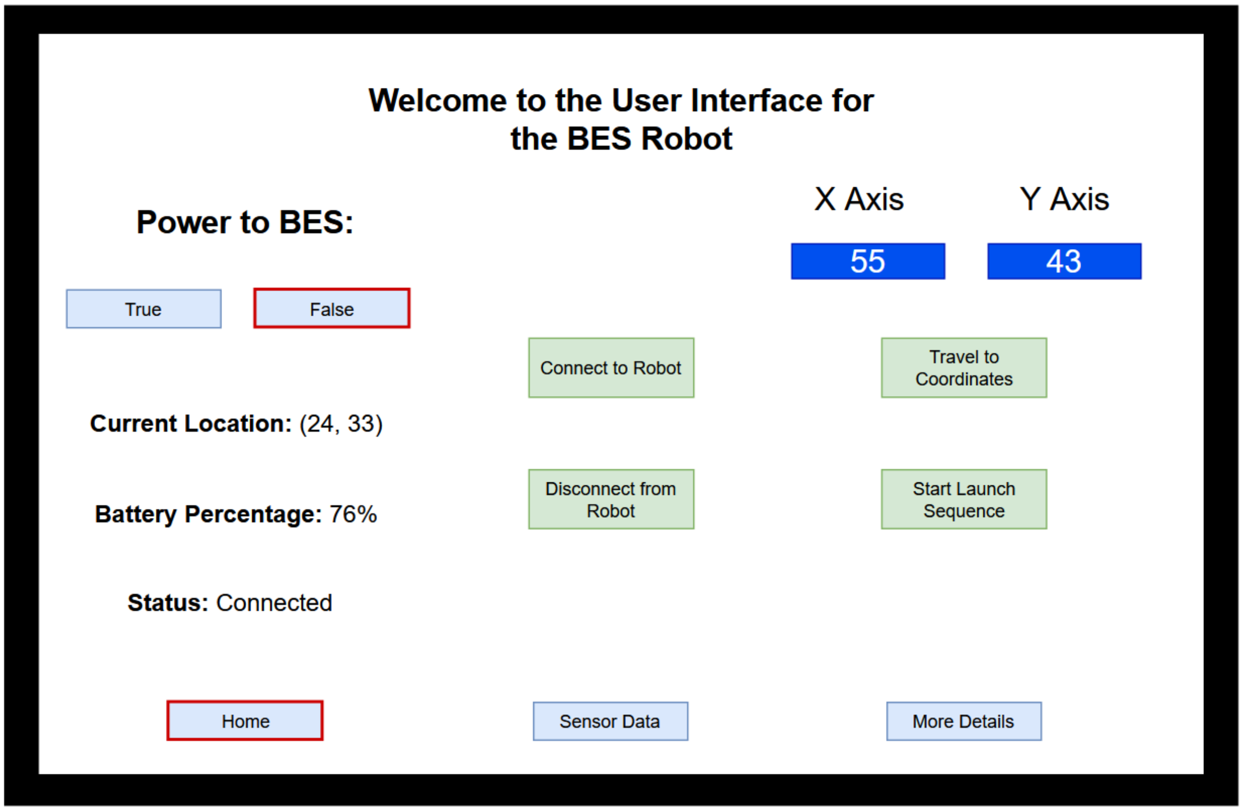
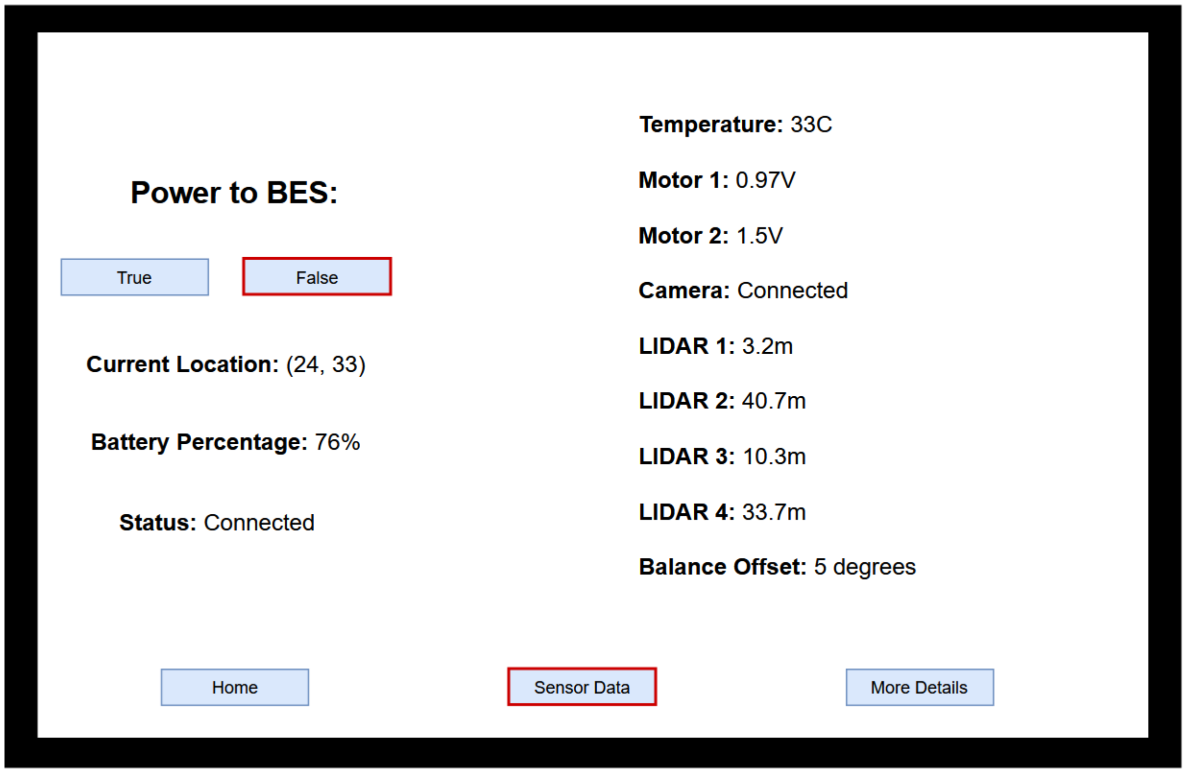


Figure 25: Home Tab

6.6.2 Sensor Tab

The sensor tab will display all of the information about the sensors/motors. This will be useful for troubleshooting and monitoring the Robot while it is running. Once configured properly we will be able to use this tab to safely monitor the sensor on the robot from a distance. For now, we plan on monitoring the temperature of the robot here in celsius in order to make sure no thermal throttling is occurring. The voltage to each motor will be displayed here in order to make sure they are functioning properly. The camera will either display as connected or disconnected which will be useful to make sure we know it is functioning. The LIDAR sensors will each display their distance value to make sure they are working properly. Lastly, we will read the balance offset which will be helpful to verify the angle of the robot. In the future, we may change the number of sensors used and there may be a need for different information to be displayed like the current through each component.



Current Location: (24, 33)

Battery Percentage: 76%

Status: Connected

Temperature: 33C

Motor 1: 0.97V

Motor 2: 1.5V

Camera: Connected

LIDAR 1: 3.2m

LIDAR 2: 40.7m

LIDAR 3: 10.3m

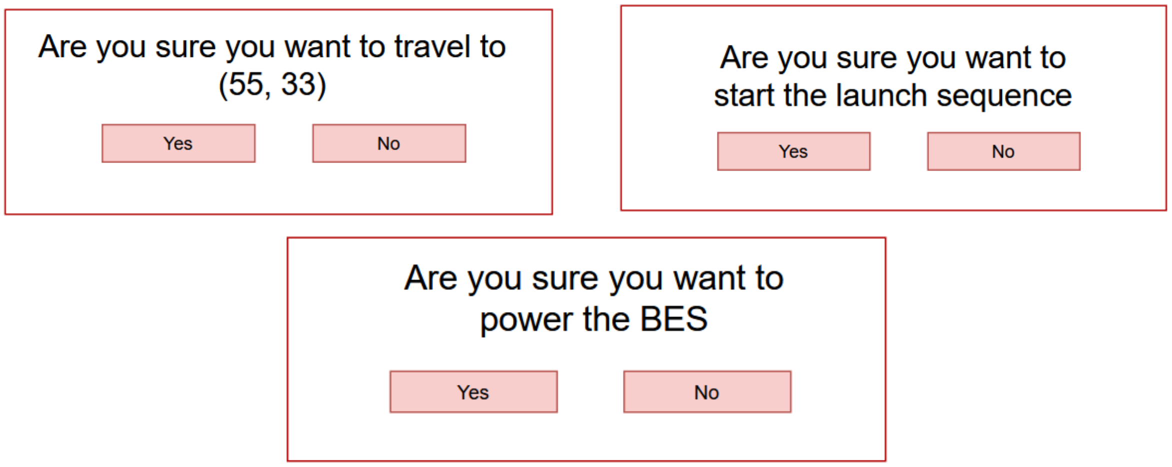
LIDAR 4: 33.7m

Balance Offset: 5 degrees

Figure 26: Sensor Tab

6.6.3 Popup Messages

Popup confirmation messages will appear when a user does a critical task. The purpose of this is because our device will be a touch screen there is a possibility that a button might accidentally get pressed. For some buttons like switching to sensor data, it's not a big deal, however, for something like starting the launch sequence we will require additional confirmation. This is a tradeoff of safety vs extra steps, so we have concluded that any button from the user interface that causes the robot to perform an action will be deemed a "critical task" and will require verification. The three critical tasks we have defined right now are pressing the travel/launch button and giving power to the BES. The pictures in Figure 24 below demonstrate what the popups will look like:


Are you sure you want to travel to (55, 33)

Are you sure you want to start the launch sequence

Are you sure you want to power the BES

Figure 27: Critical Popups

Additionally we will have alert popups that will appear to tell the user about something important. The first alert message is related to the robot's battery and it will appear whenever the robot's battery percentage is below 20% as shown in Figure 25. The reason for this is to alert the user that it may be a good idea to return the robot back to a safe location so that it will not run out of power on the field.

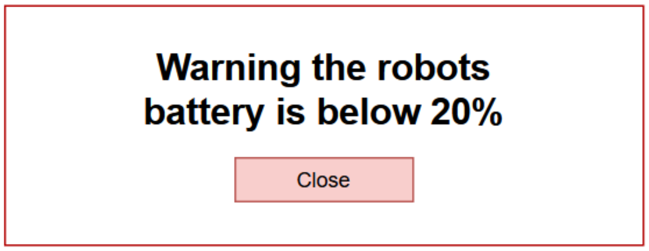

Warning the robots battery is below 20%

Figure 28: Battery Popup

The other two alert messages will appear to tell the user if the travel/launch sequences were successful or unsuccessful as shown in Figure 26. If the sequence was unsuccessful the user can look in the "more details" section to see the exact reason. If it's unsuccessful the word will be colored red while if it's successful the word will be colored green. For example, if a person was detected in the launch sequence it would say that in the more details section.

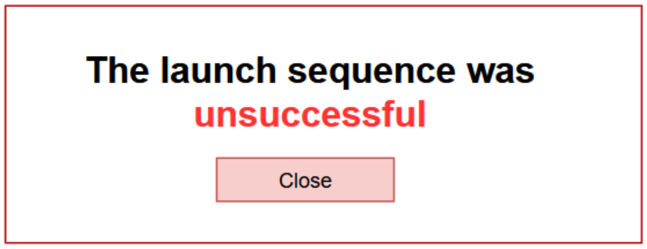

The launch sequence was unsuccessful

Figure 29: Feedback Popup

Another alert message is whenever the input boxes for the X axis and Y axis are null or they contain invalid values/out of bounds as shown in Figure 27. The inputs will be validated after the user presses the submit button and the following popup message will be displayed if the inputs are invalid. In the more details section, it will explain why the inputs were invalid.

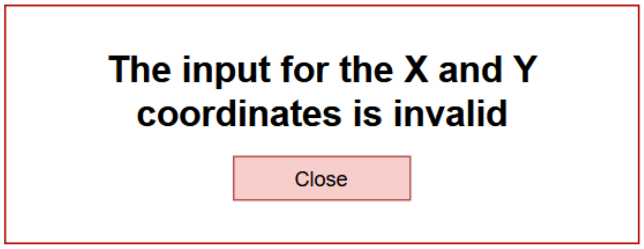
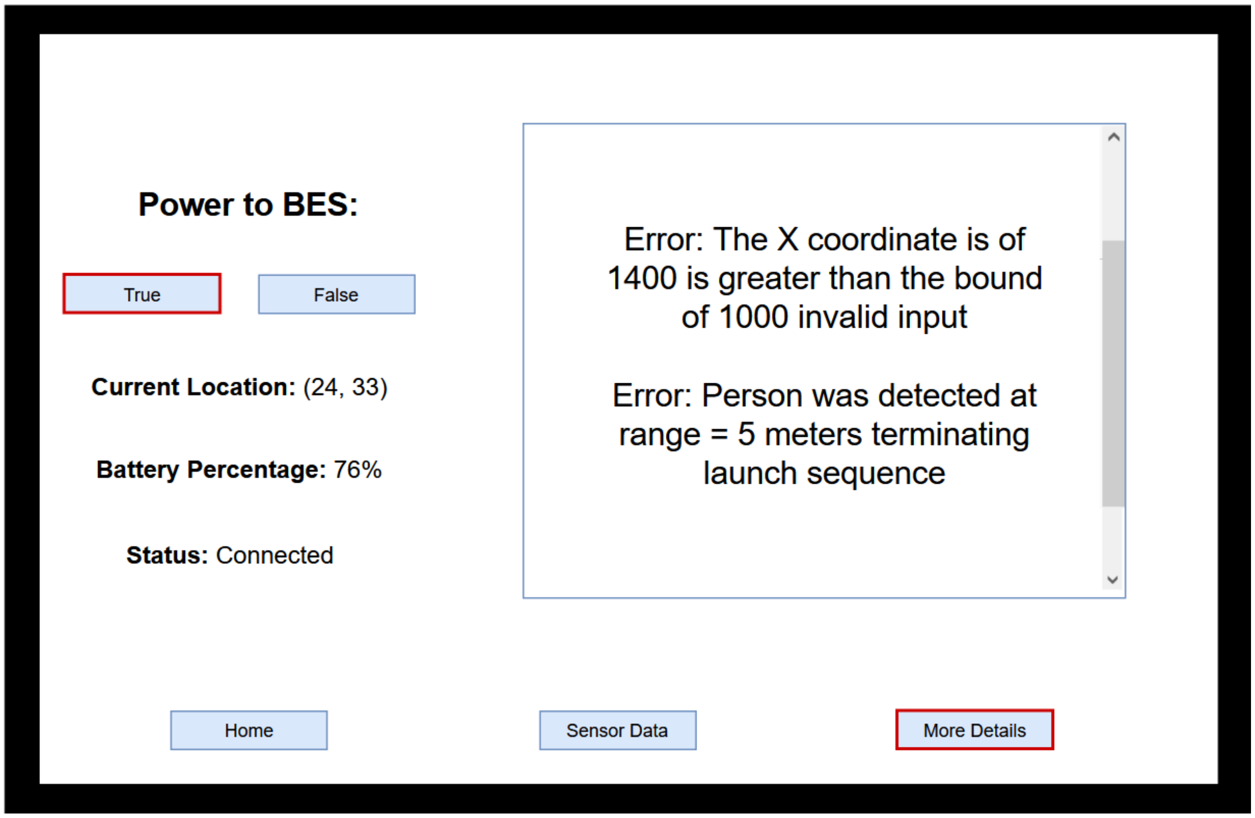

The input for the X and Y coordinates is invalid

Figure 30: Invalid Input Popup

6.6.4 More Details Tab

The purpose of this tab is to display the logs that the robot makes as shown in Figure 28. Basically, we will be displaying the console here allowing us to print useful information about where the robot is in the code or what it is doing. This tab will be a big help for debugging as we can then trace back the problem. The components on the left are present in every tab because they are very important for the user to know at all times.



Current Location: (24, 33)

Battery Percentage: 76%

Status: Connected

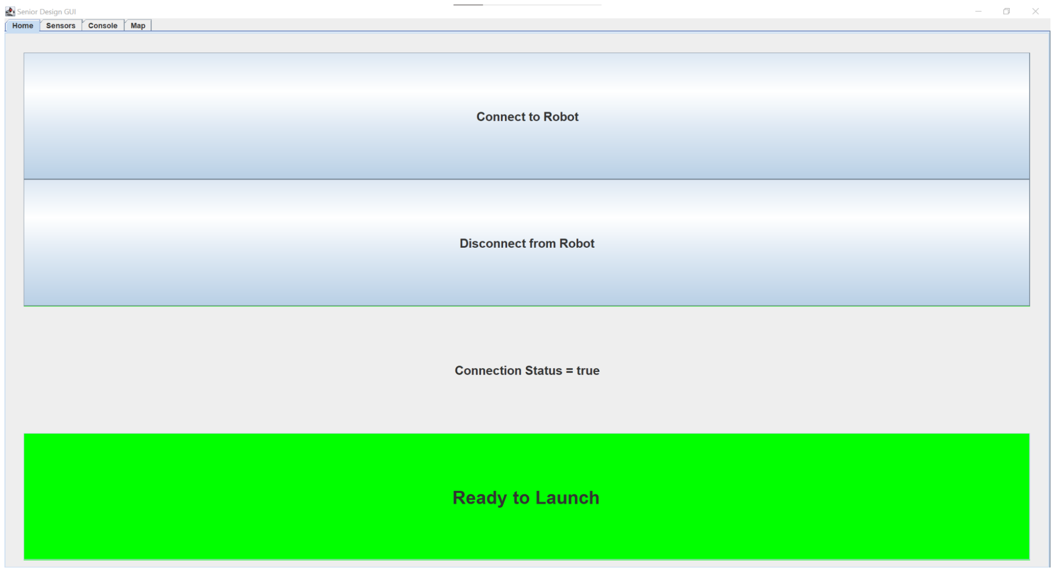
```
Error: The X coordinate is of
1400 is greater than the bound
of 1000 invalid input

Error: Person was detected at
range = 5 meters terminating
launch sequence
```

Figure 31: Details Tab

Update: The GUI has been changed over the course of the project to add things we did not expect at the time like a firing status, and a more in depth map using jxmapviewer2 library. The sensor tab was left in but not used in the final product so future work will be able to implement it when more sensors like the camera are added.

Home Tab

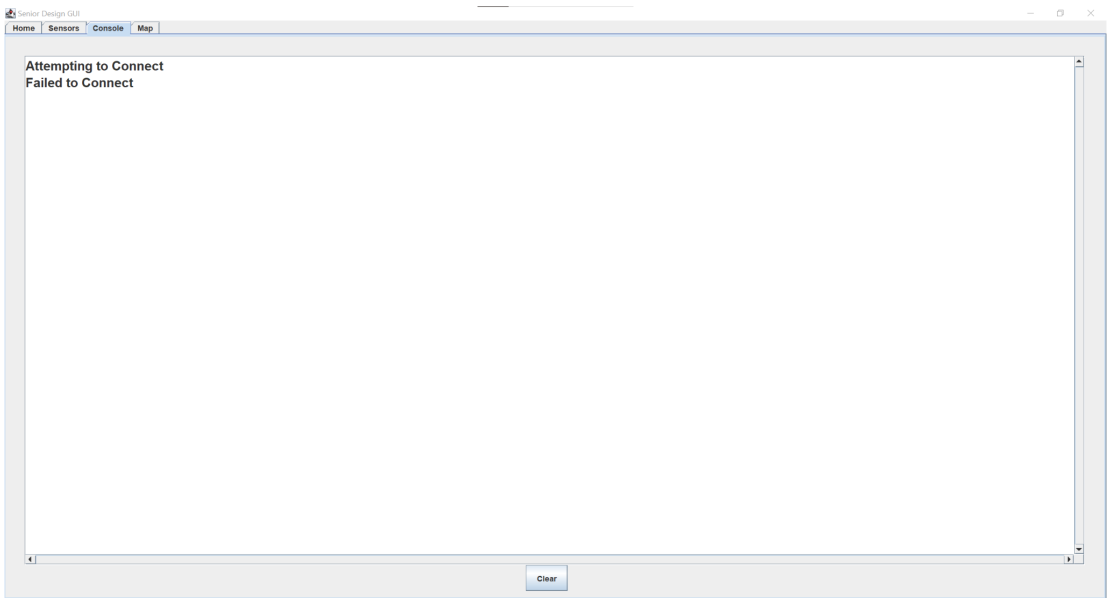


Connection Status = true

Ready to Launch

Figure 32: Home Tab

Console Tab



```
Attempting to Connect
Failed to Connect
```

Figure 33: Console Tab

Map Tab

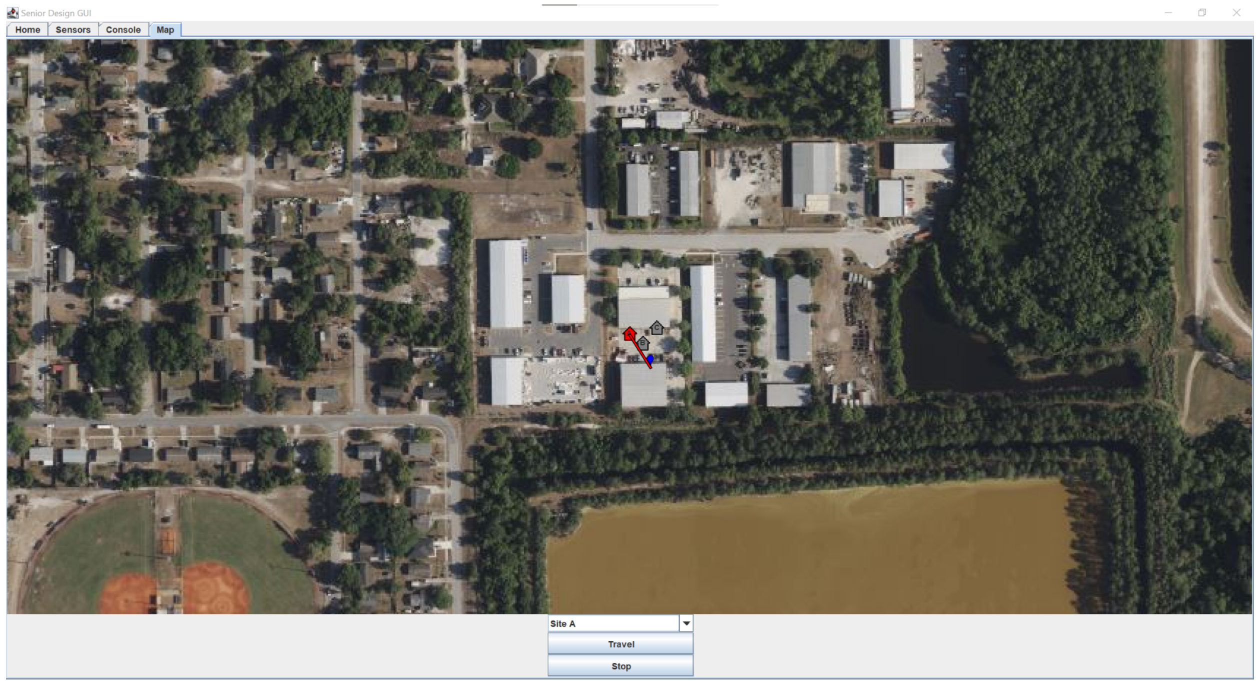


Figure 34: Map Tab

7. Testing and Control

7.1 Version Control

Multiple people will be working on the coding so we will need a reliable way to make sure everyone is on the same page and there are no compatibility issues. GitHub has been chosen and we will make sure to follow certain criteria when making changes. Every update that is pushed will have a short description highlighting what was added/fixed. This will be important because we may want to test certain components of the software in an isolated environment in order to rule out issues. Another important factor is to pull all of the updates before making changes. The reason for this is that minor things like changing a variable name can end up causing problems later down the line if everyone is not up to date. Finally, when concurrently working on the same file we will make sure to only make isolated changes to the needed function/class. This is important because we do not want two people working on the same area at once causing pull errors.

7.2 Developmental Testing

Because we will have so many different pieces of hardware communicating with each other we need to develop a plan to integrate them. The main phase of development will be Unit and

Functional testing, which will allow us to work up steadily to the final goal and to meet the design requirements.

7.2.1 Unit Testing

The goal of unit testing is to test the software in an isolated environment without any dependencies or interactions with other software/hardware. This will be the first step in the development and the main goal is to usually accomplish being able to communicate with the device. For example, when unit testing our motors we will be testing individually if we are able to apply different speeds and directions. This is the lowest form of testing and the goal is to simply be able to command the motor and not actually cause the robot to travel in a direction. A big benefit to unit testing is that it can help rule out individual issues with hardware like a faulty sensor which can save time down the line where multiple components are involved. In Table 16 below it shows the main goals in Unit Testing.

Individual Unit	**Goal Accomplished**
Battery	All of the devices are able to be powered on by the battery. It's also possible to recharge the battery once the power has been depleted.
Motors	Being able to control the speeds and direction of each motor individually. Various power levels will be tested to ensure it works properly.
Camera	Being able to perform the image processing using an existing CNN and sending the output image to the microprocessor.
LIDAR Sensors	Being able to read the distance of each LIDAR individually. We will write down the maximum and minimum distance that can be detected. Additionally, they will be tested in different lighting environments (with and without sunlight).
User Interface	The windows tablet is capable of running the application that provides the interface to control the robot. The buttons are able to be pressed and call the corresponding functions (just printing a success message for now).
Level Sensor	Being able to read the orientation of the robot through the level sensor. This should be tested

	on slopes and level grounds to ensure its accuracy.
GPS	The coordinates of the GPS are accurate and it updates in real-time.

Table 17: Unit Testing Goals

7.2.2 Functional Testing

The goal of functional testing is to test the components when used with other components directly related to its subsystem. This step is important as after unit testing we know that each of the components can work on its own but we don't know if they will work together. For example, testing motors in the functional step would be running them at the same time with the same power and direction. We want to make sure that the function of “driving” works and that a motor is not stronger than the other. In this step we would notice any differences in the motor and we can adjust accordingly in the software by adjusting the power to each motor. In Table 17 below it shows the main goals in Functional testing.

Robot Function	**Goal Accomplished**
Drive System	The robot is able to travel forwards and backward without veering off the path. Also being able to rotate clockwise and counter-clockwise about a single point.
Image Processing	The camera is able to process the image with boxes around people at 15 FPS and send the output to the microprocessor. The microprocessor can read the output to determine if a person is present.
Virtual Mapping	The robot is able to create a virtual map of its environment using the LIDAR sensors. This function will also need to integrate the drive system in order to ensure the map can be updated while moving and rotating.
Communication	The windows tablet is able to connect with the microprocessor through bluetooth and the user interface. Instructions and feedback are capable of being sent and received.
Orientation	The orientation of the robot is able to be obtained at a constant rate. This function will also need to integrate the drive system to ensure it is updated continuously.

Table 18: Functional Testing Goals

7.3 Acceptance Testing

7.3.1 Testing Location

The environment we have chosen to test and train our robot at is the field outside the UCF softball field, as shown in Figure 29 from Google Maps. This area will be at least 50 meters by 50 meters, per our sponsor's request. This area was selected as our testing grounds for many reasons. We picked the training area keeping in mind what features our robot would need based on our derived requirements, and what training area would allow for us to best test those features. This training location will allow us to test whether Bluetooth is able to autonomously navigate over rough terrain with holes, slopes, and bumps. Per the requirements of our sponsor, the training area

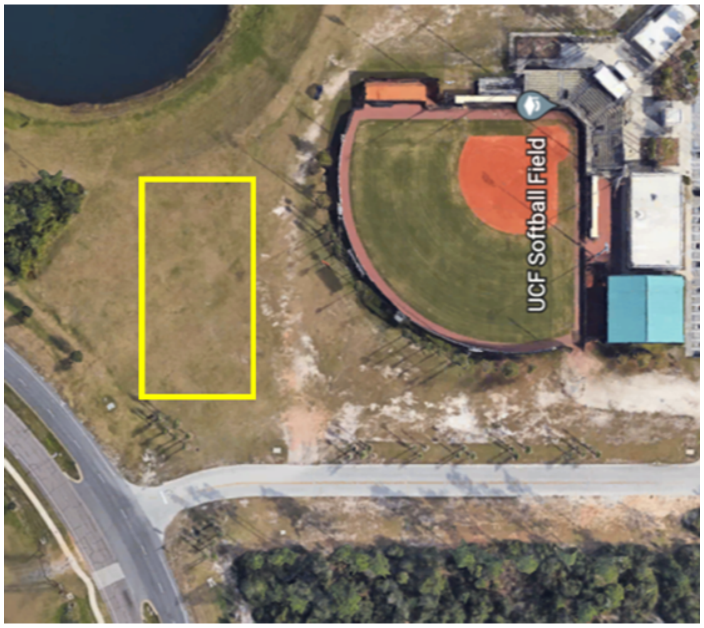


Figure 35: Aerial view of the selected training location

would need to have 2-5 test Stationary Armor Coffins, each at 1/6 size, which is about 2m wide. The coffins will need to each be about 10 meters between each other as well. Figures 30 and 31 below, provided by PEO STRI, are examples of a full size Stationary Armor Target and its design we will roughly follow on a smaller scale for this project. The robot is also required to have an IP65 rating, protection from light dust and water projected from a nozzle. This outside environment will allow for testing again light dust, and we are able to demonstrate the water protection by

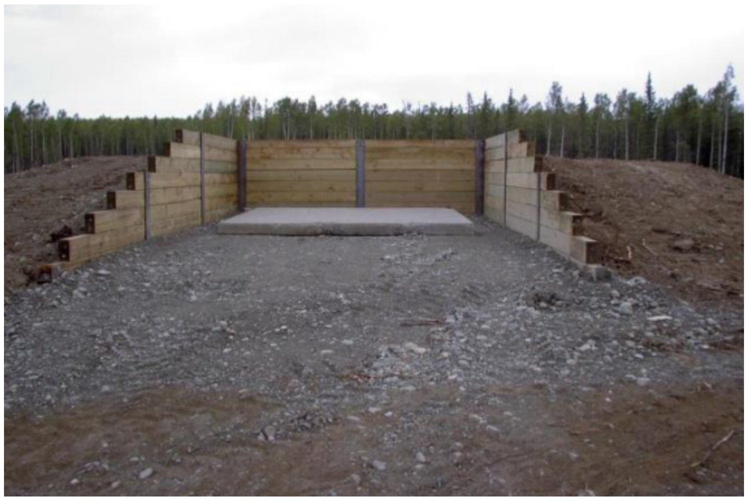


Figure 36: Example of a full scale Stationary Armor Target

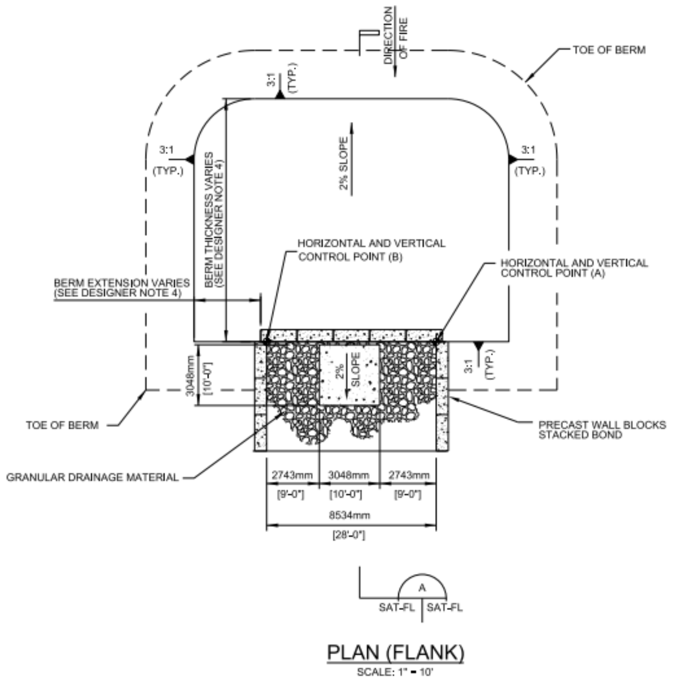

DIRECTION OF FIRE
TOE OF BERM
3:1 (TYP.)
2% SLOPE
BERM THICKNESS VARIES (SEE DESIGNER NOTE 4)
HORIZONTAL AND VERTICAL CONTROL POINT (B)
HORIZONTAL AND VERTICAL CONTROL POINT (A)
BERM EXTENSION VARIES (SEE DESIGNER NOTE 4)
3048mm [10'-0"]
2% SLOPE
TOE OF BERM
PRECAST WALL BLOCKS STACKED BOND
GRANULAR DRAINAGE MATERIAL
2743mm [9'-0"]
3048mm [10'-0"]
2743mm [9'-0"]
8534mm [28'-0"]
A
SAT-FL SAT-FL
PLAN (FLANK)
SCALE: 1" = 10'

Figure 37: Stationary Armor Coffin Design

spraying the robot. The training environment is also large enough to have multiple paths to one safe location. This way, we can test if the robot will select the most optimal path when heading to a specified location.

Update: Per our sponsor's request, we changed our testing zone to the parking lot at ShockStream. This is our sponsor's place of employment and it was easy for him to monitor our progress/provide any help necessary if the testing was held there. Also, due to budget cuts, it was no longer a requirement for our robot to be IP-65 rating.

7.3.2 Acceptance Testing Methods

The following table describes the features of the robot we will be testing based on our derived requirements, and the testing methods that will be used.

Robot Feature	Testing Method
The robot will be able to navigate over rough terrain with uneven surfaces, holes, and inclines/declines.	This feature will be tested by allowing the robot to autonomously navigate over the rough terrain. It will use machine learning and detection of rough terrain through ROS to determine when terrain is too rough to navigate through/over, and it is necessary to find a new path.
The robot will be able to navigate around any detected obstacle.	This feature will be tested by allowing the robot autonomously navigate through the training field. We will place obstacles of different sizes and shapes in random spots on the field. The robot will then have to navigate around the object and either return to its pre existing path, or continue on a shorter path to the destination.
The robot will be able to navigate into Stationary Armor Coffins without affecting any pre existing equipment	This feature will be tested by creating a demonstration Stationary Armor Coffins and setting it as the destination for the robot. The demo coffin will contain a number of objects it will have to maneuver around while still moving into the coffin to a space that is clear to stop in.

The robot will be able to navigate through narrow paths, slowing down its speed if needed.	This feature will be tested by setting up narrow pathways in the training field using inclines and borders to make them. The robot will then have to determine one, if it is able to maneuver through the path, and two, if it needs to slow its speed.
The robot will use optimal pathing when determining its path to a specified location.	This feature will be tested by setting a destination for the robot to move to in the middle/far back of the training field. The field will have clear obstacles the robot is aware of (such as Stationary Armor Coffins). The robot will have to determine with what it knows about the current obstacles on the training field what the most optimal path to travel is.
The robot will signal for the BES to fire the pyrotechnics only when the specified safety conditions are met. Safety conditions: a. The robot is at its specified location b. There are no people/cows within a 2-meter distance (this distance is 1/6 the size a full-size BES robot will need to determine) c. The robot is on a surface that is flat +- 10 degrees.	Each of the safety conditions will be tested individually. a. This safety condition will be tested by letting the robot autonomously navigate to its destination. We will then track the robot on the application to ensure it is in the right position. b. This safety condition will be tested by standing/moving near the robot to see if it can detect a human. (The robot will not be loaded with pyrotechnics during this.) c. This safety condition will be tested by placing the robot on inclines of different degrees and trying to set off the pyrotechnics.

Table 19: Acceptance Testing Methods

Update: The robot was no longer required to slow its speed when navigating due to budget cuts.

8. Administrative Content

8.1 Project Budgeting and Financing

As of this time in the R&D process our sponsor has not specified a budget for this project. Our group has talked to the professor to get a better understanding of our budget. Our professor states a project like this can cost upwards of 800 dollars. When thinking about this cost we need to consider possible equipment we need in order to test the product, equipment that fails or malfunctions, and parts we may overlook. While in the design process it is our job to make sure that we cut the cost of a product down as much as possible. Even though we are students, we are going to work hard to ensure we choose the best part in terms of cost and functionality. Below is a very rough estimate we gathered for parts we know we will need in Table 2: Budgeting and Financing.

Below is Table 19 that goes into detail about the components and their prices. It also shows the quantity of the component that we will need.

Item	**Price (USD)**	**Quantity**
IMU	$6.29	1
Camera	$200.00	1+
Microcontroller	$10.00 - $15.00	1
Misc. Parts for PCB	$15.00 - $25.00	1+
LIDAR	$50.00 - $60.00	3 - 4
GPS retriever	$6.00 - $15.00	1
Cooling System	$25.00 - $50.00	1
Motors for Axles / Treads	$120.00 - $200.00	2
Motor Controller	$21.99	1(pack of 4)
Power Switch	$1.00 - $5.00	1
Voltage Step Down	$1.00 - $10.00	2
Battery	$200.00 - $500.00	1
External Storage	$10.00 - $25.00	1
Wheels/Treads	$100.00 - $150.00	2
Robot Frame	$50.00 - $100.00	1
Total	$917.28- $1763.28	—-

Table 20: Budgeting and Financing

After seeing this chart our group believes a budget of 2000 - 3000 dollars would be sufficient. Our group had a meeting with our sponsor and they stated that this is a much smaller budget than expected. Our group and sponsor still haven't determined a set budget, however, our sponsor has asked the company for a budget of 5000 dollars for this project. Our sponsor believes this will be sufficient for all expenses. They also think it is better to ask for a higher budget now rather than later. It looks much worse on our part as the product provider to ask for more money later in the design process.

Update: Our sponsor was kind enough to give this project a 1000 dollar budget. The budget was much lower than anticipated and we inevitably had to reduce the requirements and size the the robot. With that being said, here is our total budget breakdown. Some of the components that were purchased could not be used or were used for the pruposeof actually building the robot.

BUDGET	
NAME	**PRICE (TAX not included)**
CanaKit - Raspberry PI 4	$134.99
OAK-D Lite	$149.00
ESP32	$55.00
BNO055	$33.70
TFmini Plus	$49.90
360 LIDAR	$99.19
Robot Base	$109.99
Motor Controller	$15.99
Zeee 9000mAh	$83.69
ESP32 with Ethernet	$15.99
ESP32 module	$10.99
Solder Paste	$7.95
Flux Paste	$12.45
Battery Connecters	$8.99
VK-162 GPS Mous	$15.49
Solder Wick	$11.50
LM2596 Buck Converter	$5.15

Motors with Encoders	$32.68
Rocker Switch	$1.60
PCB Parts	$42.54
PCB	$47.06
TOTAL	
SPENT	**$943.84**
REMAINS	**$56.16**

Table 21: Budgeting and Financing Updated

As you can see our final budget spent was actually under the 1000 dollar range.

8.2 Project Milestones by Semester

8.2.1 Senior Design I

Below is Table 20: Senior Design I Milestones. This table shows goal dates for specific deliverables and milestones throughout senior design I.

Description	**Dates**
Discuss Project Ideas	May 18 - May 28
Project Selection	May 28- May 31
Work on Divide and Conquer	June 1 - June 3
Divide and Conquer submission	June 3
Research on Project	June 3 - June 10
Communicate with Sponsor	June 6
Update project requirements/specification	June 10 - June 14
Update Divide and Conquer	June 14 - June 16
Divide and Conquer submission 2	June 17
Continue research on project design and implementation	June 17 - July 1
Documentation	June 24 - July 7
60-page Senior Design 1 Draft submission	July 8
Continue research/update draft	July 9 - July 21
Updated 100-page report submission	July 22
Finalize report/ make any necessary changes	July 23 - August 1
Final senior design 1 report submission	August 2
Begin Building Prototype Before Senior Design 2	TBA

Table 22: Senior Design I Milestones

8.2.2 Senior Design II

Below is Table 21: Senior Design II Milestones. This table shows goal dates for specific deliverables and milestones throughout senior design II.

Description	**Dates**
CDR with Sponsor	August 8
Continue building prototype	August 22 - September 19
Communicate with Sponsor	TBA
Testing and Redesign	September 20 - October 17
Finalize Prototype	October 18 - November 15
Peer Presentation	TBA
Final Report Submission	TBA
Final Presentation	TBA

Table 23: Senior Design II Milestones

Update: Significant milestones include:
I/ITSEC Presentation: 28 & 29 Nov 2022
Final Presentation: 30 Nov 2022
Website Submission: 6 Dec 2022

9. Conclusion

The Battlefield Effects Simulator (BES) Robot design concept was derived from the US Army/Department of Defense's expressed need for a Modular Open System Architecture (MOSA) life fire robotic platform/system to support the training of US Army soldiers. During the development process, we created a design for a system that is intended to be used on live fire training ranges to improve the overall training experience, increase training throughput and improve safety measures by removing the human element.

Main components of this project consisted of designing the robot base/platform that would hold the BES and can navigate through rough terrain, the connection between the BES, robot, interface box, and control center, creating the UI, implementing safety conditions and signals, and designing and implementing the operating system using ROS.

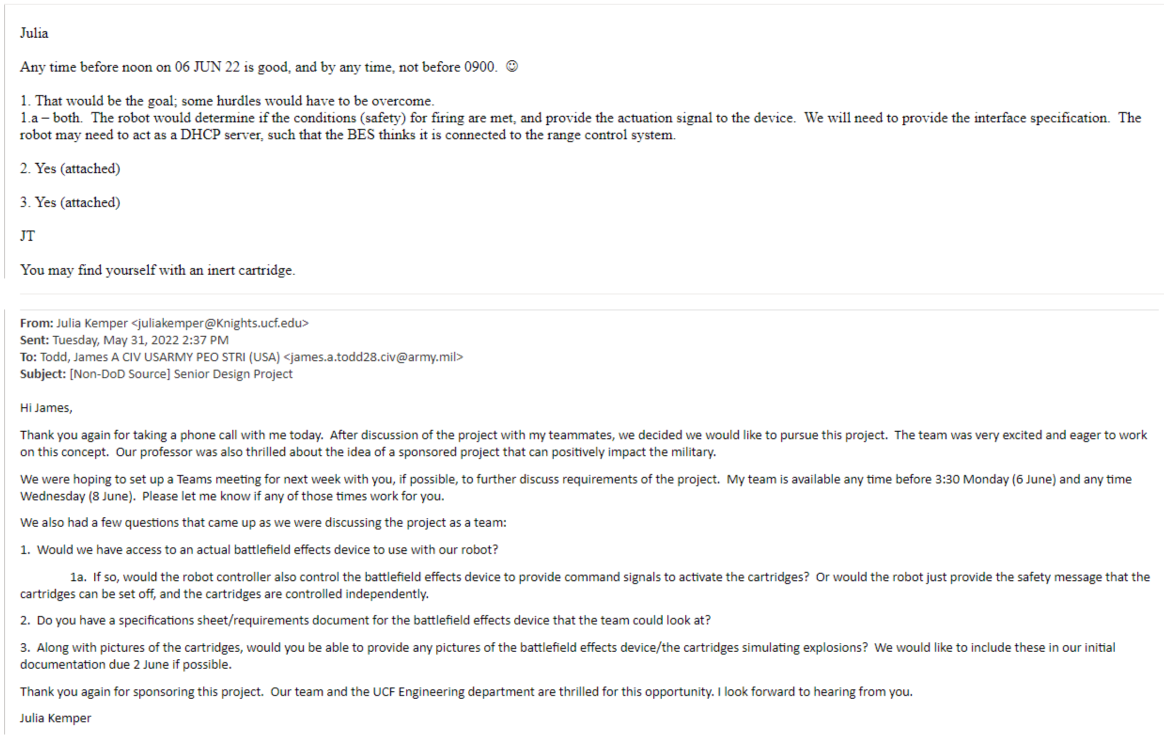
As we designed and developed the Battlefield Effects Simulator, our team encountered many changes and obstacles. Our team was able to work with any new requirements given to us by our

Sponsor and continue development on schedule by constantly communicating, working together with each other and our mentors, and staying flexible. Although this project has been difficult, we chose this project to not only push ourselves as engineers, but to be able to give back to our country in any way we can. Overall, we want the end product of this project to provide the army with a better, safer, and more realistic training environment.

10. Appendices

10.1 Copyright Permissions

Receiving BES information/images to use in paper:



Julia

Any time before noon on 06 JUN 22 is good, and by any time, not before 0900. ☺

1. That would be the goal; some hurdles would have to be overcome.
1.a – both. The robot would determine if the conditions (safety) for firing are met, and provide the actuation signal to the device. We will need to provide the interface specification. The robot may need to act as a DHCP server, such that the BES thinks it is connected to the range control system.

2. Yes (attached)

3. Yes (attached)

JT

You may find yourself with an inert cartridge.

From: Julia Kemper <juliakemper@Knights.ucf.edu>
Sent: Tuesday, May 31, 2022 2:37 PM
To: Todd, James A CIV USARMY PEO STRI (USA) <james.a.todd28.civ@army.mil>
Subject: [Non-DoD Source] Senior Design Project

Hi James,

Thank you again for taking a phone call with me today. After discussion of the project with my teammates, we decided we would like to pursue this project. The team was very excited and eager to work on this concept. Our professor was also thrilled about the idea of a sponsored project that can positively impact the military.

We were hoping to set up a Teams meeting for next week with you, if possible, to further discuss requirements of the project. My team is available any time before 3:30 Monday (6 June) and any time Wednesday (8 June). Please let me know if any of those times work for you.

We also had a few questions that came up as we were discussing the project as a team:

1. Would we have access to an actual battlefield effects device to use with our robot?

1a. If so, would the robot controller also control the battlefield effects device to provide command signals to activate the cartridges? Or would the robot just provide the safety message that the cartridges can be set off, and the cartridges are controlled independently.

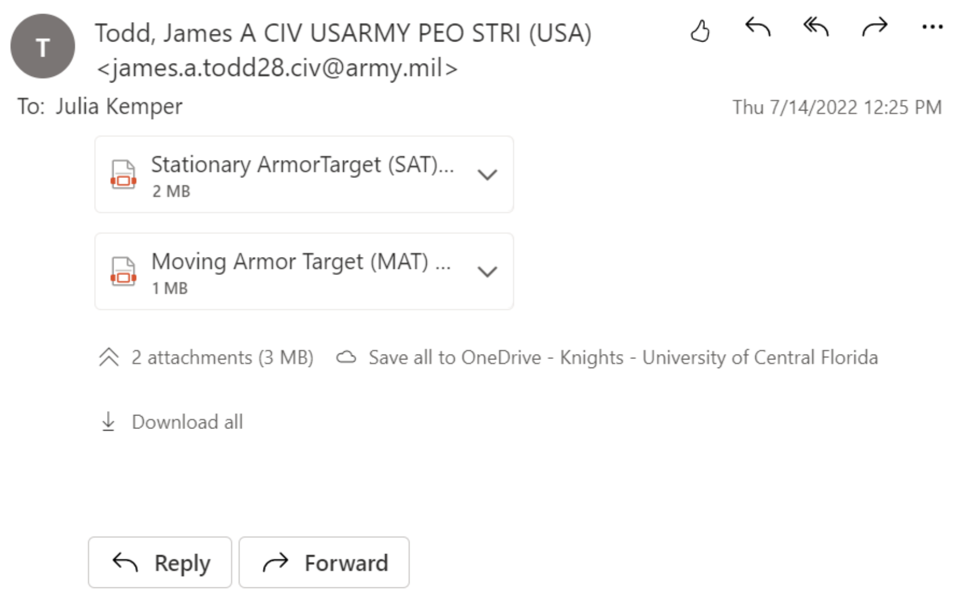
2. Do you have a specifications sheet/requirements document for the battlefield effects device that the team could look at?

3. Along with pictures of the cartridges, would you be able to provide any pictures of the battlefield effects device/the cartridges simulating explosions? We would like to include these in our initial documentation due 2 June if possible.

Thank you again for sponsoring this project. Our team and the UCF Engineering department are thrilled for this opportunity. I look forward to hearing from you.

Julia Kemper

Receiving Stationary Armor Coffin Details/Images to use in paper:


Todd, James A CIV USARMY PEO STRI (USA) <james.a.todd28.civ@army.mil>

To: Julia Kemper

Thu 7/14/2022 12:25 PM

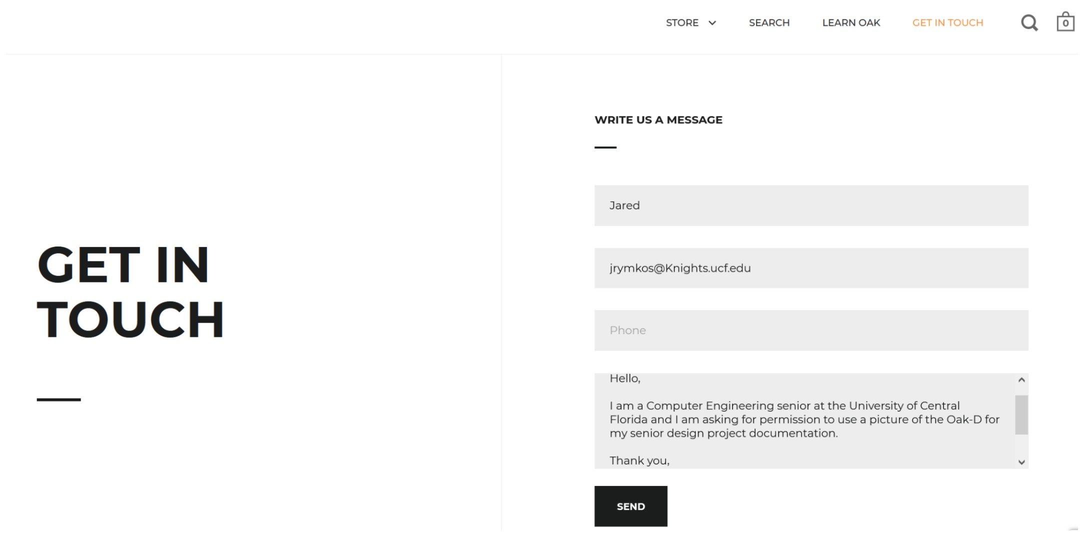
Stationary ArmorTarget (SAT)... 2 MB

Moving Armor Target (MAT) ... 1 MB

2 attachments (3 MB) Save all to OneDrive - Knights - University of Central Florida

Download all

Receiving permission for OAK-D camera:



Jared

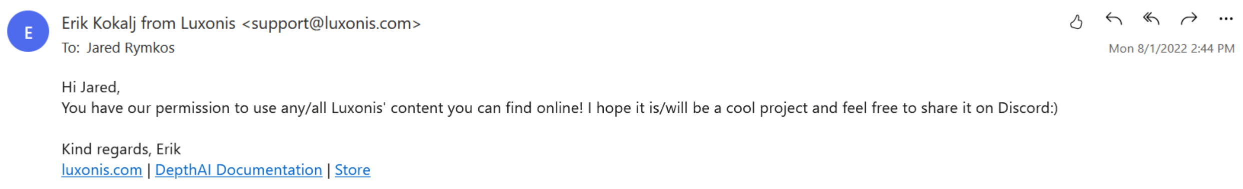
jrymkos@Knights.ucf.edu

Phone

Hello,

I am a Computer Engineering senior at the University of Central Florida and I am asking for permission to use a picture of the Oak-D for my senior design project documentation.

Thank you,


Erik Kokalj from Luxonis <support@luxonis.com>

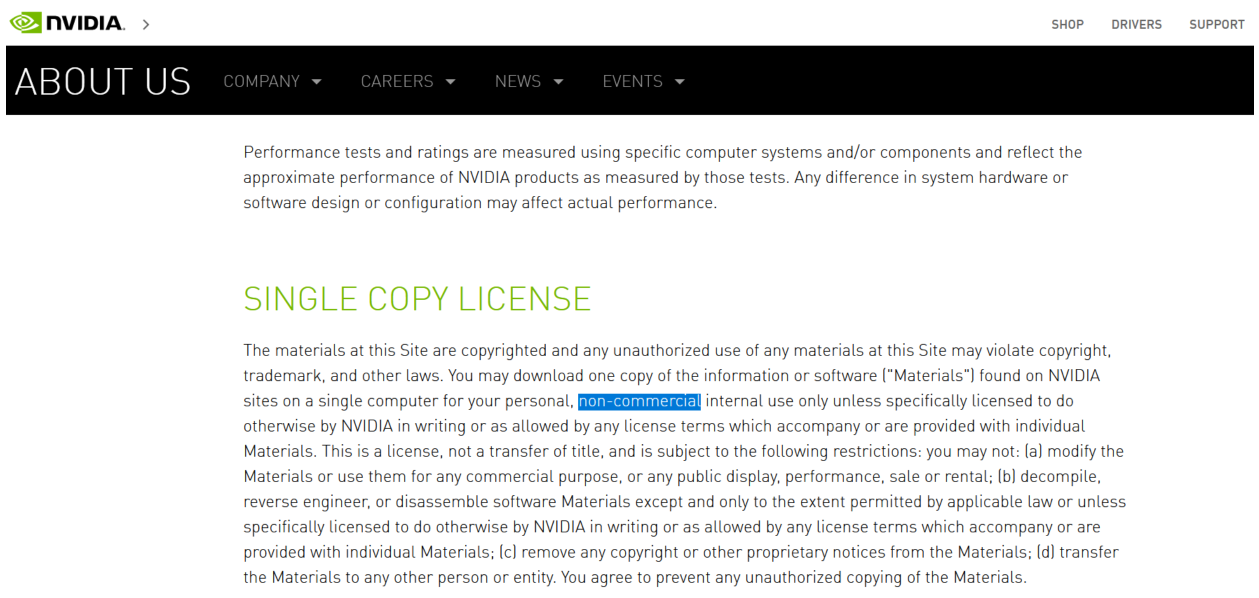
To: Jared Rymkos

Mon 8/1/2022 2:44 PM

Hi Jared,
You have our permission to use any/all Luxonis' content you can find online! I hope it is/will be a cool project and feel free to share it on Discord:)

Kind regards, Erik
luxonis.com | DepthAI Documentation | Store

Permissions Listed on Websites:



Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of NVIDIA products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

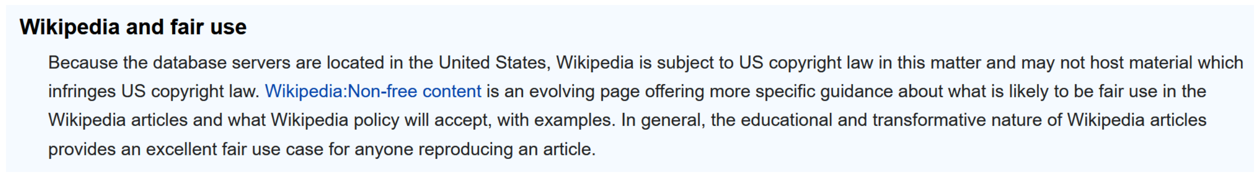
SINGLE COPY LICENSE

The materials at this Site are copyrighted and any unauthorized use of any materials at this Site may violate copyright, trademark, and other laws. You may download one copy of the information or software ("Materials") found on NVIDIA sites on a single computer for your personal, non-commercial internal use only unless specifically licensed to do otherwise by NVIDIA in writing or as allowed by any license terms which accompany or are provided with individual Materials. This is a license, not a transfer of title, and is subject to the following restrictions: you may not: (a) modify the Materials or use them for any commercial purpose, or any public display, performance, sale or rental; (b) decompile, reverse engineer, or disassemble software Materials except and only to the extent permitted by applicable law or unless specifically licensed to do otherwise by NVIDIA in writing or as allowed by any license terms which accompany or are provided with individual Materials; (c) remove any copyright or other proprietary notices from the Materials; (d) transfer the Materials to any other person or entity. You agree to prevent any unauthorized copying of the Materials.

7. Licensing

ROS is an Open Source project. We aim to support a wide variety of users and developers, from grad students to entrepreneurs.

- We prefer permissive Open Source licenses that facilitate commercial use of the code.
- The preferred license for the project is the BSD license. Whenever possible, new code should be licensed under BSD. All ROS core code is licensed BSD. Reasons for choosing BSD
- Any OSI-approved license is acceptable (for non-core code).
- We strongly recommend using a non-copyleft license (e.g. BSD) for ROS `.msg` and `.srv` files so that the auto-generated source files and data structures are not encumbered.
- The full text statements of all licenses used in the project should be placed in the LICENSES directory at the top of the repository. If you add a package under a license that is not included in LICENSES, add the license statement.
- Every source file should contain a commented license summary at the top. For convenience, the LICENSES directory also contains summaries, appropriately commented for different languages.
    - We preserve license and copyright statements on all third-party code that we redistribute.
- We rigorously obey license terms on third-party software that we use. For example:
    - If a library is licensed GPL or LGPL and you modify it, you must release the modified code. Ideally you would send a patch to the library maintainer. Keeping the modified code in a publicly accessible repository (e.g., at SourceForge) also suffices.
    - If your package uses a GPL'd library, then your package code must also be licensed GPL.
    - If your package uses a GPL'd library, then it must not also use any code governed by a GPL-incompatible license. For example, the Creative Commons Attribution-Noncommercial-Share Alike license is GPL-incompatible, because it imposes extra constraints that the GPL does not (namely, requiring attribution, and prohibiting commercial use).
- Whenever possible, each ROS package is governed by a single license.
    - Common special case: BSD-licensed code (e.g., the ROS core) is used in a package that also uses GPL-licensed code. To comply with the GPL, the BSD-licensed code is multiply-licensed under BSD and GPL, with the user given the choice of which license to use. This is not a big deal, because the GPL simply adds restrictions that are not in BSD.
- The ROS packaging and communication system allows for fine-grained licensing. Because nodes communicate via ROS messages, code from multiple nodes is not linked together. Thus the package provides a kind of "license boundary."
    - Exception: when a package is a library that is actually linked to by other packages. In this situation, license terms mix in the resulting binary.
- For reference: Maintaining Permissive-Licensed Files in a GPL-Licensed Project: Guidelines for Developers


Wikipedia and fair use

Because the database servers are located in the United States, Wikipedia is subject to US copyright law in this matter and may not host material which infringes US copyright law. Wikipedia:Non-free content is an evolving page offering more specific guidance about what is likely to be fair use in the Wikipedia articles and what Wikipedia policy will accept, with examples. In general, the educational and transformative nature of Wikipedia articles provides an excellent fair use case for anyone reproducing an article.

Creative commons licsense for Figure 3A.
https://www.newmediarights.org/guide/legal/copyright/licensing/creative_commons/citizens_legal_guide_using_creative_commons_licenses

10.2 Datasheets

ATmega2560: https://docs.arduino.cc/hacking/hardware/PinMapping2560
FT232R USB UART: https://ftdichip.com/wp-content/uploads/2020/08/DS_FT232R.pdf
MC33269: https://www.mouser.com/datasheet/2/308/1/MC33269_D-2315567.pdf

10.3 References

"A." *2-D Gaussian Filtering of Images - MATLAB*, https://www.mathworks.com/help/images/ref/imgaussfilt.html.

"Canny Edge Detector." *Wikipedia*, Wikimedia Foundation, 24 Jan. 2022, https://en.wikipedia.org/wiki/Canny_edge_detector.

"Distance Measurement: Laser or Ultrasonic?" *Sensor Partners*, 3 Nov. 2020, https://www.sensorpartners.com/en/knowledge-base/distance-measurements-choosing-laser-or-ultrasonic-sensors/.

Ed. (2021, November 15). *Raspberry pi arduino serial communication - everything you need to know - the robotics back*. The Robotics Back-End. Retrieved July 21, 2022, from https://roboticsbackend.com/raspberry-pi-arduino-serial-communication/

"First Steps with Depthai." *Luxonis*, https://docs.luxonis.com/en/latest/pages/tutorials/first_steps/#default-model.

"IEEE Standard for Rechargeable Batteries for Mobile Phones," in IEEE Std 1725-2021 , vol., no., pp.1-80, 23 Aug. 2021, doi: 10.1109/IEEESTD.2021.9514813.

"IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN - Specific Requirements - Part 15: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)," in IEEE Std 802.15.1-2002 , vol., no., pp.1-473, 14 June 2002, doi: 10.1109/IEEESTD.2002.93621.

"Image Gradient." *Wikipedia*, Wikimedia Foundation, 18 Nov. 2021, https://en.wikipedia.org/wiki/Image_gradient.

"Introduction." *OpenCV*, https://docs.opencv.org/4.x/d1/dfb/intro.html.

"ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering," in ISO/IEC/IEEE 29148:2018(E) , vol., no., pp.1-104, 30 Nov. 2018, doi: 10.1109/IEEESTD.2018.8559686.

"ISO/IEC/IEEE International Standard - Software and systems engineering--Software testing--Part 4: Test techniques," in ISO/IEC/IEEE 29119-4:2015 , vol., no., pp.1-149, 8 Dec. 2015, doi: 10.1109/IEEESTD.2015.7346375.

Joseph, L. (2020, August 22). *How to choose a brain for your robot?* Robocadamy. Retrieved July 21, 2022, from https://robocademy.com/2020/04/18/how-to-choose-a-brain-for-your-robot/2/

Kumar, A. (2022, July 21). *How to use Arduino with Robot Operating System (ROS): Arduino*. Maker Pro. Retrieved July 21, 2022, from https://maker.pro/arduino/tutorial/how-to-use-arduino-with-robot-operating-system-ros

"Nvidia DeepStream SDK." *NVIDIA Developer*, 31 May 2022, https://developer.nvidia.com/deepstream-sdk.

"Oak-D¶." *D*, https://docs.luxonis.com/projects/hardware/en/latest/pages/BW1098OAK.html.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *The Journal of machine Learning research* 12 (2011): 2825-2830.

Prechelt, Lutz. "Are scripting languages any good? A validation of Perl, Python, Rexx, and Tcl against C, C++, and Java." *Adv. Comput.* 57 (2003): 205-270.

"Robot Operating System." *ROS*, https://www.ros.org/.

Ros - data display with Rviz. Stereolabs. (n.d.). Retrieved August 1, 2022, from https://www.stereolabs.com/docs/ros/rviz/#:~:text=RVIZ%20is%20a%20ROS%20graphical,many%20kinds%20of%20available%20topics

ROS1 vs Ros2, practical overview for ROS developers - the robotics back. End. (2021, December 16). Retrieved July 6, 2022, from https://roboticsbackend.com/ros1-vs-ros2-practical-overview/

Sahir, Sofiane. “Canny Edge Detection Step by Step in Python - Computer Vision.” *Medium*, Towards Data Science, 27 Jan. 2019, https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123.

Sears-Collins, A. (2021, June 14). How to publish wheel odometry information over Ros. Automatic Addison. Retrieved August 1, 2022, from https://automaticaddison.com/how-to-publish-wheel-odometry-information-over-ros/

“Sensors.” *Temperature Sensors*, 8 June 2022, https://www.te.com/usa-en/products/sensors/temperature-sensors.html.

Specification of ROS_MASTER_URI and ROS_HOSTNAME edit. Specification of ROS_MASTER_URI and ROS_HOSTNAME - ROS Answers: Open Source Q&A Forum. (n.d.). Retrieved July 20, 2022, from https://answers.ros.org/question/272065/specification-of-ros_master_uri-and-ros_hostname/

Staff, R. B. R., & Staff, R. B. R. (2019, May 13). *Starship Technologies Inc.*. Robotics Business Review. Retrieved August 2, 2022, from https://www.roboticsbusinessreview.com/robotic-company/directory/listings/starship-technologies/

Stroustrup, Bjarne. *The C++ programming language*. Pearson Education, 2013.

Subramanian, R. (n.d.). *Build Autonomous Navigation Robot: ROS Simulation & Hardware*. Udemy. Retrieved July 21, 2022, from https://www.udemy.com/course/build-your-own-ros-powered-autonomous-robot/

“TurtleBot 4.” *Clearpath Robotics*, 21 June 2022, https://clearpathrobotics.com/turtlebot-4/.

Wiki. ros.org. (n.d.). Retrieved July 6, 2022, from http://wiki.ros.org/navigation/Tutorials/RobotSetup

Wiki. ros.org. (n.d.). Retrieved July 21, 2022, from http://wiki.ros.org/gmapping#Required_tf_Transforms

Wiki RVIZ. ros.org. (n.d.). Retrieved July 22, 2022, from http://wiki.ros.org/rviz

Wolff, Dipl.-Ing. (FH) Christian. “Radar Basics.” *Radartutorial*, https://www.radartutorial.eu/02.basics/Frequency%20Modulated%20Continuous%20Wave%20Radar.en.html.

Yoraish. (2021, August 8). *A Full Autonomous Stack, a Tutorial | ROS + Raspberry Pi + Arduino + SLAM*. Yoraish. Retrieved July 23, 2022, from https://yoraish.com/2021/09/08/a-full-autonomous-stack-a-tutorial-ros-raspberry-pi-arduino-slam/

8 reasons why you should use ROS for robotics projects. Niryo. (n.d.). Retrieved July 6, 2022, from https://service.niryo.com/en/blog/8-reasons-use-ros-robotics-projects

16-385 Computer Vision - CMU School of Computer Science. https://www.cs.cmu.edu/~16385/s17/Slides/4.0_Image_Gradients_and_Gradient_Filtering.pdf.