Battlefield Effects Simulator (BES) Robot

Jared Rymkos, Julia Kemper, Michael Rodriguez, Nicholas Nachowicz

Department of Electrical and Computer Engineering, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract*— 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 the US Army soldiers. This led to the Battlefield Effects Simulator (BES) Robot. This robot is intended to be used on live fire training ranges to assist with pyrotechnic handling. The BES Robot will accurately travel to a specified location on a training field to fire pyrotechnics from the Battlefield Effects Simulator, simulating hostile threat fire and hostile vehicle kill indications, when the proper safety measures have been met. This is the first model of the BES Robot and is intended to be a prototype. The Battlefield Effects Simulator will be connected to the robot, but will not sit on the robot due to the scaled down size of this first prototype.**

I. Introduction

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 developed a robotic system that will connect to 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 was to provide a method to activate pyrotechnics in the correct location 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.

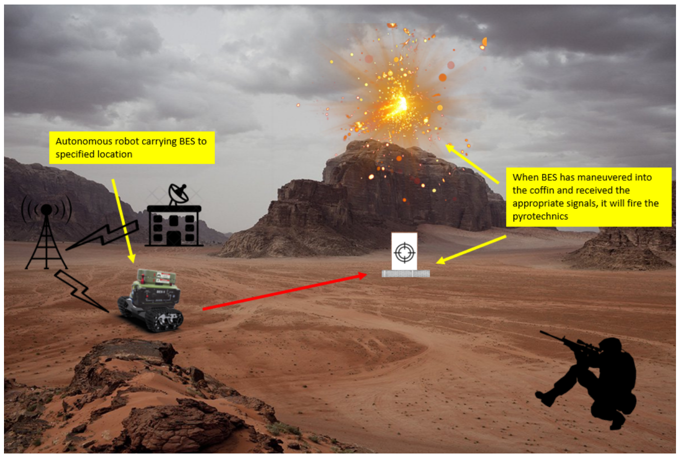
Autonomous robot carrying BES to specified location
When BES has maneuvered into the coffin and received the appropriate signals, it will fire the pyrotechnics

Figure 1: Illustration of the robots functionality on a life fire training range.

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.

II. System Components

The robot utilizes several components in order to be able to accurately function and complete its objective. Each component is utilized in order to create the final model of the robot. This section is a small introduction to each component that is utilized within the design of the robot.

Base

The main factor that will house all the components of the robot is the base that is utilized in order to hold all the other components. The base that we obtained was a purchased pre designed base that was made by the brand DoHome which was obtained through amazon. The base is a 4 wheel drive tank chassis which means that each motor utilizes treads instead of wheels in order to move. The tank chassis made for a good base to design a small scale prototype for the aspect of the robotic project.

Motors that came with the base

The motors are the most important component in terms of movement. The base that we purchased through amazon came with 4 motors to attach to the base. The motor used is the 33GB-520 motors which support a voltage of 12V and has a speed of 170-350 rpm depending on the amount of voltage used. Since these motors did not come with encoders, we are utilizing only 2 of the motors and have them connected to 2 other motors with encoders.

Motors with Encoders

Since the motors that came accompanied with the base that we purchased did not come with any encoders, we had to purchase some motors with the appropriate encoders. To ensure that the other motors work with the encoders, we wired the back motors which are the motors that came with the base to the front motors which contain the encoders to ensure that the motors will work in tandem with each other. These motors function on 12V and has a speed of 100 RPM. The motors encoders help with the odometry data that ROS utilizes to calculate transforms.

Zeee 14.8V Lipo Battery

The battery is the main component that will provide power to the motors of the robot. The 14.8V Lipo battery is regularly used for RC equipment which is why we are using it due to the reliability and the fact that the battery is lighter weight compared to other types of batteries. The higher voltage of this Lipo battery does pose a small problem that is resolved using a buck converter in order for each component to get the proper amount of energy. The other main feature of utilizing this Lipo battery is that it will be easier to charge compared to other types of batteries.

Commercial Battery Pack

This battery is another component that the robot will be utilizing. This battery is used to power the raspberry pi so that the robot will have a longer battery life by utilizing 2 batteries instead of 1. A converter does need to be present because it will be powering it via 5V USB and it has a voltage regulator inside. This battery has 16.75A hours which gives the raspberry pi, PCB, encoders, and logic of the motor controllers much more time than the motors are capable of.

Rocket Switches

The rocket switches will be the component that we will utilize in order to activate or shut down the robot. The switch is used for safety reasons so that there are no electrical problems during recharging or other maintenance being done to the robot. The switch is connected directly with the battery so that the battery will not be able to flow any current while the switch is set to off.

LM2596 Buck Converter

The LM2596 buck converter is utilized to step down the voltage of the battery to some of the other components that require a lower voltage. This buck converter is utilized to provide the motors with 12V instead of 14.8V since the specs of the motors have the working voltage max out at 12V. The buck converter had to be utilized since the Lipo battery came at 14.8V instead of just being a 12V battery.

Raspberry Pi model 4B

This is one of the major technical components of the robot that is utilized to process the information received from the camera. This component holds the programming set up in ROS in order to properly analyze if there is any person in the vicinity of the robot. This component allows us to meet the safety requirement for the robot.

360 Lidar

The 360 Lidar is the component that is utilized in order for the robot to map its environment and detect obstacles in its path while traveling to a specific location. This Lidar analyzes all around the robot and is located on top of the robot. The Lidar scans the surrounding area to identify if there is an obstacle in the way for the robot to traverse around. This LIDAR is capable of running indoors easily and in practice the only place it is difficult to function is under direct sunlight.

PCB

The PCB is another majorly important technical component on the robot. The PCB is designed to act as a microcontroller to handle navigation and driving sending the safety signal to the esp32 inevitably firing the BES. This component was designed by the group instead of utilizing a pre designed microcontroller. The PCB is essentially the central hub that is connected with most of the other components on the robot.

BNO055 IMU

The IMU is used for the various sensor features located on it for providing better safety operation on the robot. The main feature that we will utilize on this IMU is the temperature sensor that it contains so that we can properly assess the internal temperature of the robot. This component will help us in preventing the robot from overheating or having some components get damaged from high temperature.

BTS7960 Motor Drivers

The motor driver is the component that allows us to control the motors with higher accuracy than just having power being directly sent to the motor. The motor drivers allow for the PCB to control the motors and send the proper output voltage to the motors to get the robot to its specific destination.

ESP32

The ESP32 component is utilized to provide the robot with communication functionality. This functionality will allow the robot to communicate with the user to coordinate the location the robot will go and allow the robot to send the safety signal to allow the BES to be activated. This component is the main method of communication with the robot.

GI-U7 GPS module

The GPS module is the component that will allow the robot to identify the location that it is currently in and allow the robot to know the location of the destination that it is being sent to. This module allows us to know where the robot is and allows the robot to move to a destination based on the map data it has and the location due to the module.

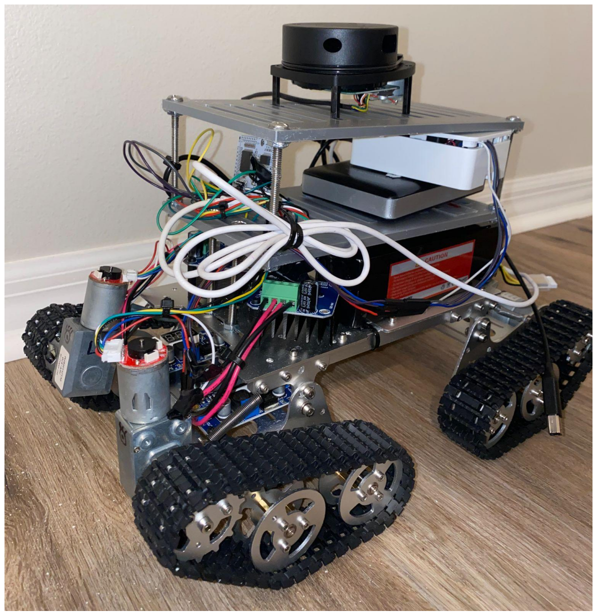

Figure 2: This is an image of the robot with most of the components attached.

III. System Concept

Power System

The power system includes all components that are controlling voltages and supplying power to devices throughout the robot. All components will be running off of a 2 battery system. The largest battery (Zeee 14.8V lipo battery) will be directly connected to a rocker switch via hard wiring and controlling whether the motors will be able to receive power. This batter will then be attached to 2 separate LM2596 buck converters. These buck converters control the voltage and reduce the battery's voltage to 12V. The 12V buck converter then gets hard wired directly to the BTS7960 motor drivers inevitably powering the motors themselves.

The second battery is a commercial power bank that will be powering the raspberry pi and devices that are connected to it. This power bank will be powering the single board computer using USB-C. The components that are powered by this battery through the raspberry pi include: PCB, 360 LIDAR, ESP32, BNO055, GI-U7, Encoders on motors.

The BES will be powered outside of the system because our robot cannot hold its weight and size. The robots main purpose is a proof of concept in creating a working system.

Our team decided to create a 2 battery system in order to sustain a decent battery life. Our initial requirement was 10 hours with the BES being on top of the robot. Due to the need of a very large budget in order to fulfill this requirement with various reasons including, the robot needing to be much larger and able to carry a much heavier weight, we have reduced this requirement. Our team decided we would like to be able to have a battery life of a minimum 1 hour.

With a 2 battery system we are well above the battery life of 1 hour with the robots battery life ranging between 2 and 4 hours. The main crunch on our battery life is actually their large Zeee battery. The maximum battery life of the commercial battery we are using is 16.75A hours which is much larger than the battery life of the battery powering the motors. While the motors won't be taking 12V at all times the battery life of the robot is dependent on how long the motors will run simply because the commercial battery bank is so large.

Drive system

The drive system consists of all the components that involve robot propulsion or controlling motors. The base is included in this system because of the treads the robot is running on. There are 4 separate tread tracks on the base each attached to its own motor. Each pair of motors is connected to its own motor driver. The motor drivers and encoders are connected to PCB.

The back 2 motors are motors that came with the base that we purchased. These motors do not have encoders and therefore do not control any of the actual steering or transform calculations in order to determine the robots position in the ROS navigation stack. These back motors are then connected via hard wiring directly to the front motors that are on the tread track inline with it. The front motors are motors that do have encoders and therefore are involved with calculating the transform of the robot's current position using motor rotation ticks. These motors did not directly come with the robot's base so in order to attach these to the base we had to do extensive modifications to the frame. The specific pinout for power to the motors are PIN0(red wire) for motor positive power and PIN5(white wire) for motor negative power. These are not specifically labeled as PIN0-5 on the encoder which is why there is a color next to the pin names. These pins are directly connected to the motor driver along with the back motors mentioned previously.

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.

Sensory System

The sensory system of our robot helps the robot understand and evaluate the world around it. This system will be able to detect obstacles, understand the level of the robot, understand its speed and also let the world know where it is exactly.

The 360 LIDAR is directly connected to the raspberry pi via USB to TTL serial converter. This is connected to the raspberry pi for easy data transfer from the LIDAR to raspberry pi. The power draw from the LIDAR is also too much for the PCB.

The BNO055 IMU will be attached to 2 devices in order for easy information delivery. It will be connected to both the PCB and Raspberry Pi. This will allow the PCB to transfer information to the esp much easier while having ROS directly connected to the IMU for transformation calculations. The IMU is connected via I2C with SDA and SCL going to both SDA and SCL respectively on the only I2C pins on the PCB and raspberry pi on GPIO pins 8 and 9 respectively. While Vin and GND on the IMU going to 3.3V and GND on just the raspberry pi. This only needs power from one device and will be sending data to both devices.

Lastly is the GI-U7 GPS will be able to tell the map where exactly the robot is in correlation with the robots safe zones on the GUI's map. This will be connected to the esp32 so that data transfer does not have to go through several devices to get to the GUI. This will be connected via serial connection. TX and RX pins on the GPS will go to TX0 and RX0 on the esp32. The GND and VCC pins will go to GND and 5V on the esp32.

Misc. Connections

There are 3 Miscellaneous connections that can fit into multiple groups. These connections are very important but more of just connections between the brains of the operation. The raspberry pi will be connected to the PCB via micro USB. The esp32 will be connected to the PCB via serial connection.

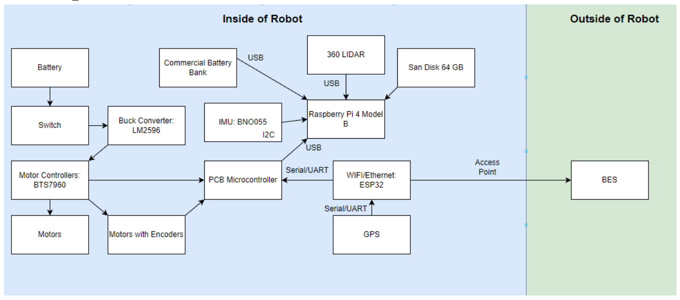

Figure 3: Robot connectivity block diagram.

Communication/Software

The arduino is connected to the ESP32 and will send the information through serial communication. The programming language is C++ which allows us to have a deeper control of the hardware compared to other languages. We transmit information from the ESP32 to the windows tablet by connecting to the same network and communicating through sockets. We decided to connect the tablet to the ESP32 over the raspberry pi in order to reduce the strain on the raspberry pi from running ROS. This is a client-server type architecture and we decided to make the ESP32 be the server so it can be looking for clients while it is turned on. The tablet is the client and it will search for the server when the connection button is pressed in the GUI. Once a connection is established the ESP32 will send updated information about the sensor data every 2 seconds to the tablet. This will be done by having an identifier byte which signals the type of data (X coordinate, Y coordinate, etc) followed by the data and then a new line symbol representing the end of the data. Through the tablet by pressing the travel button a byte will be sent to the ESP32 signaling which site to travel to or to stop. The Windows tablet was selected because we were developing the GUI code in a windows environment which would allow us to easily transfer the code. In the future it may be possible to run the client code on android devices instead of a windows tablet because it was written in Java. The BES is connected to the ESP32 through wifi utilizing a router. Originally we were planning on having the BES attached to the robot, however, due to budgeting constraints the BES would be too heavy for the smaller scale version of our robot. There are three main safety checks that must be met in order for the fire signal to be given to the BES. These checks include the robot's GPS coordinates being within the respective sites coordinate bounds, no objects being within 2-meters, and the robot's IMU reading is within 10 degrees from being flat. The lidar will be used to scan if any movement is detected around the robot. The checks are important to make sure the robot is in a safe position to fire. If all the conditions are satisfied the BES will be fired and the result of the safety checks will be sent to the GUI.

IV. Hardware Detail

PCB

As we have stated in the previous section the PCB is one of the pinnacle points in our project. Our team decided to make a microcontroller PCB that can run arduino IDE code. Our team not only wanted something easily programmable but also something that can have an abundance of pins because there are several components it will be controlling and talking to. Our team decided on the ATmega2560 micro-chip which has been used previously in the Arduino mega2560. This also makes things easy for testing because of our easy access to the Arduino board itself and start testing without the PCB being finished. This board has a total of 8 PWM pins, 7 digital GPIO pins, 8 analog GPIO pins, 4 serial communication bus, 1 I2C. Some of these pins aren't being used at this time however in future the company hopes to bring in new senior design teams to expand on what we have. This board also has 3.3V and 5V capability. In the future the plan for the PCB is to have an IMU built into the board itself. While building the PCB some errors were found and corrected. The original PCB had some footprint issues, 2 fewer PWM pins and the usb chip was not implemented into the default serial port on the atmega2560. Our solution for the first two was creating a new board and ordering it. The solution for the serial lines was jumping our serial port the usb chip was connected to onto the serial port the atmega 2560 uses as a default port. After this connection was made the only thing that needed to be made sure of is that the original serial port was never activated to prevent any damage to the board.

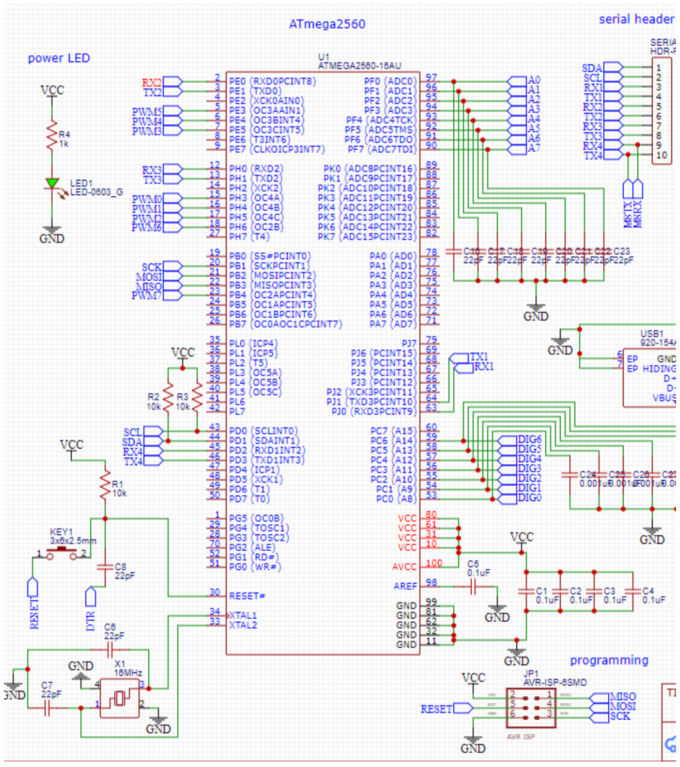

Figure 4: The ATmega2560 Chip and its connections in the center. This image also includes the Jtag that will help in programming the chip in the bottom right, the power LED that turns on when receiving 5V in the top left, the serial header in the top right, and the clock and reset buttons in the bottom left.

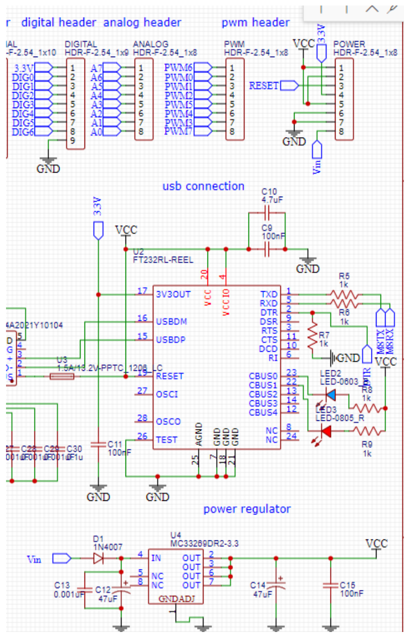

Figure 5: In the center of this image is the usb connection circuit. In the bottom half of the Schematic is the power regulation circuit. In the top half of the schematic is all the headers other than serial that were displayed in the previous image.

V. Software Detail

ROS

Our team chose to use the Robot Operating System, or ROS, as the operating system for the BES robot. ROS is a set of libraries and tools used to help build robot systems. This includes drivers, algorithms, and development tools. ROS served many purposes in this project. The overall purpose it served was to make the BES robot navigate accurately from point A to B. ROS has what's called a Navigation Stack Package. This is a set of packages and libraries that are installed to ROS with Ubuntu on the Raspberry Pi. This is a 2D navigation stack that takes in odometry information, sensor information, and a goal pose for the robot to end at, and it outputs velocity commands that get sent to the mobile base to move the robot safely. I began by installing this navigation stack to the Raspberry Pi. Next, I worked on the transform configuration. For the Navigation Stack to work properly, the information about the coordinate frames of the robot has to be published.

After this information was published using a ROS Transform Package, the sensor needed to be connected to ROS. Our team used the RPLidar to help the robot map and detect its environment and avoid any obstacles by publishing the sensor data and sending the data over ROS. The lidar is connected to the Raspberry Pi via USB port. I cloned the RPLidar package from Github that had the necessary packages for the lidar to run. After the lidar was spinning, I was able to output the data to a terminal window. Next, I installed Rviz, which is a ROS 3D visualization software program. I connected the lidar to Rviz and was then able to visualize the laser data on the screen.

After the lidar data was being visualized, I began working on the odometry information. I began by installing the necessary software that enables ROS to speak with the Arudino. It was crucial these two components were able to speak because the motor controllers/encoders were hooked up to the Arduino (and eventually the PCB after initial testing). Next, I needed to be able to control the robot's velocity remotely from my computer using ROS. I first needed to create a file in the Arduino IDE that would act as a ROS node to publish the ticks of each wheel of the robot and subscribes to the velocity commands to the robot so it will drive correctly. After this code was written and published from the Arduino IDE and onto the Arduino, it was then time to launch the motor controllers. I used the ROS robot steering interface which publishes Twist messages, or linear/angular velocity commands, to the appropriate robot topic. The robot can be manually moved forward/backward and left/right using the slide bars.

Next, I needed to use ROS and Rviz to create an initial pose and goal pose of the robot publisher. This will allow for the robot to navigate to a specified location, or goal pose. The initial pose button allows the user to set the pose of the robot on a given map and initialize the localization system. The navigation goal button sets the goal for the desired location of the robot. Once the code for these has been written, we can see the coordinates in a terminal for the initial/goal pose of the robot when they are selected in Rviz. After this, I needed to publish the wheel odometry information over ROS. This includes data from the wheel encoders and is used to estimate how much the robot's position/orientation has moved over time in regard to a given map. I had to create an odometry publisher that would take in the ticks of the wheels and the initial positioning of the robot and publish it to the correct topics for the robot to later access. When this code was added in, we could then see the relationship between the initial/goal publishers and the actual location of the robot when they are set.

After the odometry information was correctly set up, it was time to get the IMU running. The data from the IMU needed to be published using ROS. After hooking the IMU up to the Raspberry Pi, it was time to set up the serial communication protocol. The BNO055 IMU used I2C serial communication protocol. I registered the IMU to our robot and then checked that the bus of the IMU was at the correct address. When this was confirmed, I then installed and built the required IMU package that would allow for the BNO055 and ROS to communicate. Next, I installed the Rviz plugin that would allow me to see the IMU data. From there, when I launched the IMU, I was able to see the data in the terminal and visually in Rviz changing live.

Next it was time to fuse the wheel encoder data and IMU data together. This was done by using an extended Kalman filter. This allows for a more accurate estimate of the location and orientation of the robot on a map. This was a simple set that just involved creating a file that would fuse the odometry and IMU data, but it was very crucial to getting the robot to run smoothly.

After the odometry and IMU data was fused correctly, I could then create a map of the environment the robot would be moving in. This allows me to set an initial and goal pose for the robot to move between. This map was created using our lidar and the ROS Hector-SLAM (Simultaneous Localization and Mapping) package. This is a package that can map an unknown environment using lidar while also keeping track of where the lidar is in the map. To start the mapping, I launched the lidar and then Rviz. I then slowly moved the lidar around the environment until I was satisfied with the visual of the map in Rviz. Next, I saved the map as an image so I could later pull it up to set the current position of my robot and then the goal pose for it to travel to.

Next, it was time to set up the costmap configurations for the robot. The global and local costmaps store information about obstacles in the robots environment. The code for these configurations was stored in .yaml files to be accessed by the robot during its navigation. In addition to the costmap configurations, the base local planner needed to be configured as well. The planner computes the velocity commands that get sent to the base controller of the robot. After all the configurations were set up, it was time to set up the move base node with ROS. That is the node responsible for the path planning of the robot. It will create a path that is collision free. The piece of code for that was added to the robots launch file.

Next, the localization system AMCL, Adaptive Monte Carlo Localization, had to be set up. This system tracks the robots position using the map created, the scans from the lidar, and the transform configuration readings and estimates where the robots output position will be. This just required a block of code being added to the launch file.

Finally, it was time to launch the robot to move with all the moving pieces. This is done by starting the launch file that contains all the blocks of code from the previous steps. Then, in Rviz I determine the current position and desired position of the robot. And finally, the robot uses its configurations to map a collision free path to its destination.

GUI

For the user interface, our team decided to use Java in order to make use of the swing library. The swing library allows us to implement components like text boxes, buttons, drop down menus, and different frames into an application that will run on the Windows tablet. The four main tabs that the user can navigate to in the GUI are the Home, Sensors, Console, and Map. The Home tab is the initial starting tab and is where the connection status and BES firing status are displayed. If there is no connection to the robot it will be indicated by the status being false. The BES firing status will be red if it's unsafe to fire and will turn green when the safety conditions are met. There are also two buttons that allow the user to connect and disconnect the wifi connection with the robot. Next, we have the sensor tab which relays sensor information from the ESP32 to the tablet. This information includes the number of people found from the OAK-D, the applied voltage to the motors, the degree offset from the IMU, and the current velocity of the robot. This tab will be implemented in the future with the following senior design team. Next, we have the console tab which displays information about what is currently going in the code. This is useful for troubleshooting as it indicates what the Robot is attempting to do. For example, if the user presses the connect button from the Home tab it will be displayed in the console that the tablet is searching for a connection. After 5 seconds if the Robot is not found it will display that in the console or say that the connection was successful. Additionally the result of the safety checks will be displayed in the console, it will indicate which of the three conditions were satisfied and if the BES launched. There is a button to clear the console which will erase all of the history. Finally, there is the Map tab which is where the robots location is displayed with respect to the site locations. The coordinates are found using the GI-U7 and are translated into being displayed on the map. We used the jxmapviewer2 library in order to integrate map tiles into the GUI. The map tiles are found using Bing Maps satellite images and they are accessed from the server live which means there may be a delay in loading the tiles. The robot's location is indicated by a blue marker and the sites are indicated by a gray house with the designated letter (A, B, C). Currently we have three sites to demonstrate the path finding logic, however, this can easily be scaled up when applied to the field training. By selecting a site from the drop down menu and hitting the travel button it will highlight the respective site by turning the color red and drawing a line to the site from the robot's current position. The robot's marker is updated in real time and if connection with the robot is lost the marker will disappear and the highlighted sites will return to gray. There is also an option to stop the travel sequence by pressing the stop button. Stopping the sequence will cause the robot to stay in place and await further instructions from the GUI.

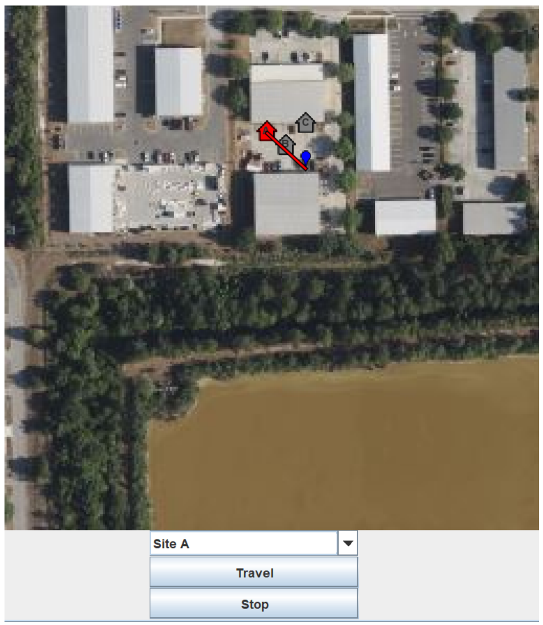

Figure 6: The Robot currently traveling to Site A displayed on the map in the GUI.

VI. 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 as a proof of concept for a robot that would later 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 holds components such as the PCB, Raspberry Pi, Lidar, and motor controllers and can navigate through rough terrain, the connection between the BES, robot, PCB, Raspberry Pi, and GUI, 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 Robot throughout the course of two semesters, 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. We were constantly changing hardware and software components being used when our team found better solutions to our problems. 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.

The Engineers

Jared Rymkos is a 20-year old graduating Computer Engineering student who is currently seeking Software Engineering positions.

Julia Kemper is a 22-year old graduating Computer Engineering Student. Kemper recently accepted a job with The MITRE Corporation as a Software and Simulation Engineer. She will continue to work as an engineering contractor for the US Army.

Michael Rodrigues is a 24-year old graduating Electrical Engineering student who is currently seeking Electrical Engineering positions.

Nicholas Nachowics is a 24-year old graduating Computer Engineering student who recently accepted a position with Northrop Grumman as a Software Engineer.

Acknowledgement

The authors wish to acknowledge the assistance and support of Sponsors: Jeremy Lanman and James Todd, Mentors: Tom Waligoria and David Howard, and Professor: Dr. Samuel Richie.

VII. References

[1] Sears-Collins, A. (2021, June 27). How to set up the Ros Navigation Stack on a robot. Automatic Addison. Retrieved July 10, 2022, from https://automaticaddison.com/how-to-set-up-the-ros-navigation-stack-on-a-robot/