5-6-2022

Autonomous Planetary Rover Team Final Project Report

William Whitfield
Trinity University

Jason Guenther
Trinity University

Jackson Robison
Trinity University

Davis Tucker
Trinity University

TRINITY UNIVERSITY

Autonomous Planetary Rover

Team Final Project Report

William Whitfield

Jason Guenther

Jackson Robison

Davis Tucker

May 6, 2022

I. Executive Summary

This design team was tasked to develop autonomous movement, sensing, and navigation capabilities on a rover platform developed by a summer research team working under Dr. Kevin Nickels. The rover is comprised of four subsystems that strive to meet the requirements by establishing movement capabilities through power delivery and control, odometry feedback of the wheels, obstacle detection, and navigation. This report evaluated each design against the requirement they were intended to meet. The following report describes the final design of each of these subsystems, explains the testing performed on each subsystem, and evaluates the results of these tests against the design requirements.

The design constraints of rover size and budget are maintained by our final design by delivering a final design that fits through a standard CSI doorframe, and not exceeding the total budget of $2400. The final deliverable satisfies the requirements of battery specifications, incline traversal, display of map and estimated position, and obstacle detection. The final design failed to demonstrate an ability to traverse over an obstacle of 2 inches. The team was unable to demonstrate completion of the remaining requirements because of significant failures of the motors described later in the report.

In the process of delivering the project requirements, extensive modifications and redesigns to the provided platform were necessary. The frame received from the project sponsor was in a nonfunctional state. The team performed significant modifications to the rover frame to allow for proper movement of the rover. Also, the provided motor drivers failed in preliminary testing, requiring the team to experimentally evaluate the operational requirements of the motors and select and integrate new motor drivers into the final design.

Overall, the team delivered a functioning prototype that met many of the project requirements, and all the design constraints. The rover was able to move, detect obstacles, and plan navigation through an environment. Unfortunately, the motors suffered a thermally induced failure during testing, precluding the completion of the remaining tests.

II. Introduction

The objective of this design project was to develop autonomous movement, sensing, and navigation capabilities on a rover frame (constructed by a summer research team working under our sponsor), to create a platform for continuous development in the field of autonomous robotics. The rover frame provided to the team is displayed in Fig. 1.

Figure 1. Inherited rover platform

The rover was required to autonomously navigate to within 10% of distance travelled to three separate waypoints on 4/5 attempts and to start from random initial positions with different waypoints for each attempt. While traversing to each waypoint the rover was required to avoid any obstacle that were blocking its path, be able to traverse an incline of 4% (2 degrees) and could attempt to traverse over an obstacle of 2 inches in height or less. The rover was required to function for one hour. This project has been made possible by an external donation which doubled the allotted budget from the Engineering Science department. It is the opinion of the team that this project would have not been feasible without the additional funds.

To complete the objective and meet the requirements of this project, specific components were given to the team by the sponsor while other components were selected and purchased by the team. An NVIDIA Jetson Nano [1] (referred to as the Jetson in this report) was provided by the sponsor to act as the processing unit of the rover and run the obstacle detection, navigation, and hardware interface programs. The inherited development platform consisted of a frame

constructed of extruded aluminum T-channel with six dual stage cycloidal gearboxes, six 12 V DC brushed motors, and six wheels attached. The frame and gearboxes were developed by the aforementioned summer research team and project sponsor respectively. Additionally, six optical encoders were provided to attain odometry feedback from the wheels of the rover which is necessary for the navigation algorithm to process the rover's motion. Finally, six IBT-4 motor drivers [2] were provided to modulate and control the pulse width modulation (PWM) sent to the motors. The DC motors, optical encoders, and 6 H-Bridge motor drivers were provided by a 2020-2021 Senior Design Team who worked on a previous iteration of the rover.

To provide the rover with features necessary to complete the objective and meet the project requirements mechanical adjustments were performed to the inherited platform and four subsystems were developed. The mechanical adjustments were conducted to fix fundamental issues with the inherited platform that prevented the rover from properly moving. The components adjusted were the rear legs of the frame, spacing components that keep the wheels a safe distance from the gearboxes, and the gearboxes themselves.

The first subsystem is the wheel assembly and odometry system that provides the rover with movement capabilities and wheel position feedback. The second subsystem is power delivery and control circuitry, which is integrated with the third subsystem, the motor control unit and software, written on the embedded microcontroller (MCU). Together the second and third subsystems provide the electronic components on the rover with the appropriate power and allow the MCU to control the motion of the rover. Finally, the fourth subsystem is the navigation and obstacle detection system which provides the rover with the ability to monitor its surroundings and decide on the best path forward.

III. Final Design Overview

A. Mechanical Adjustments

To allow the rover to move, three components of the inherited platform needed to be modified. First, the wheels were initially mounted in direct contact with the gearboxes. The resulting friction from this contact prevented proper rotation. To fix this issue the team modified the dimensions of the provided spacer. By adding the modified spacer (shown in Fig. B.1.1), the team was able to reduce friction and allow proper rotation of the wheels.

Next, the four rear legs on both sides of the inherited frame bowed significantly during movement, causing unacceptable resistance on the wheels and veering when the rover was driven forwards or backwards. To resolve this issue the team secured two supporting members connecting the left and right sides of the frame. Adding these supports prevented the bowing from occurring. This allowed the rover to travel in a straight line and reduced the current draw of each motor by a few amps keeping it well within the safe operating conditions.

Finally, during testing of the rover while it was elevated on blocks, the team noticed that the wheels were wobbling significantly and rotating at different speeds, with one of the wheels rotating at a significantly slower rate than the others. After some research and discussion, the team decided to add washers and thrust bearings between the outside rotating disk and the plate that held the gearing inside the gearboxes as indicated below on Fig. 2.

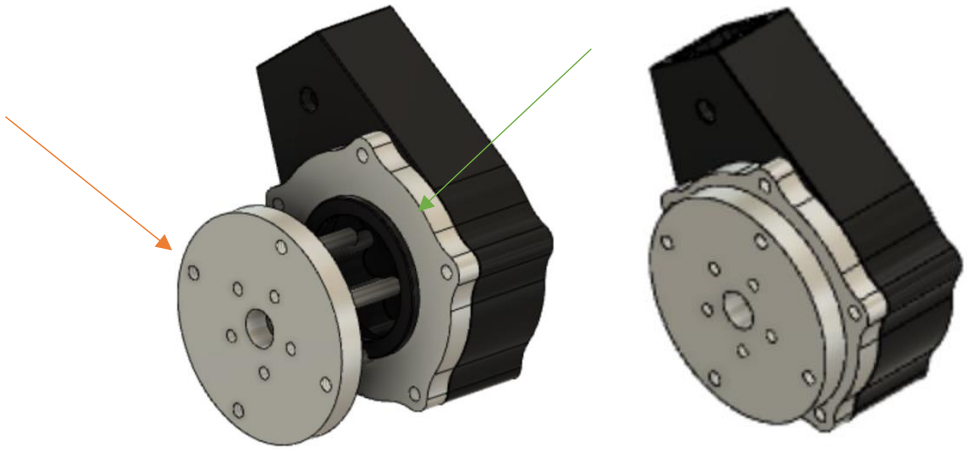


Figure 2. Open view of gearbox (left) with rotating plate (orange arrow) attached to plate holding gearing inside (green arrow) and normal view of gearbox (left)

The presence of thrust bearings eased the friction between the rotating plate fixed to the wheel and the plate holding the gearing inside. The washers took up extra space between the plates, preventing the wheels from wobbling.

B. Wheel Assembly and Odometry Subsystem

This subsystem provides movement capabilities to the six wheels of the rover and captures odometry feedback from each wheel for the motor control software. Each of the six assemblies consists of a 12 V brushed DC motor [3] attached to a dual stage cycloidal gearbox. The shafts of the motor and gearbox were coupled using a 3D printed component and the motor itself is held onto the gearbox by a larger 3D printed housing. This housing holds an optical

encoder [4] directly above the coupling and aligns the encoder shaft parallel with the motors. The coupling was designed with a timing pully built onto its surface that ensures direct translation of shaft rotation through a timing belt connected to a timing pully placed on the encoder shaft. A 3D computer aided design (CAD) model of the assembly is shown below in Fig. 3 and models of each individual component printed by the team can be found in Appendix B.

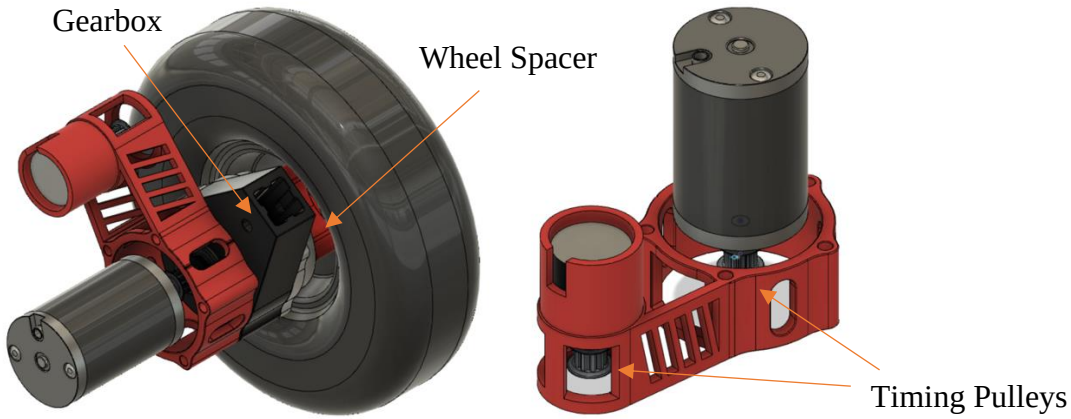


Figure 3. CAD models of wheel assembly (Left) and closer view of encoder housing (right)

C. Power Delivery and Control Circuitry Subsystem

The rover has two separate power networks: one for the high current needs of the motors and the second to power the control circuitry. The first network consists of a 12 V 200 Ah LiFePO4 battery [5] for driving the motors. The capacity of this battery allows the rover to operate for the required one hour while providing the required current to each motor. The battery is connected to a 6-channel 200 A (ASIN: B099ZH8X6X) fuse box with a knife switch on the ground terminal of the battery. This switch allows power to be turned on and off safely without risk of electrocution while the fuse box protects the motors in the event of current draw over the safe operational limit.

Each of channel of the fuse box is connected to one of the 6 motors through a Cytron 30 A DC motor driver [6] (see Fig. B.2.1 for wiring diagram). The decision to switch to the Cytron drivers from the provided IBT-4 drivers was necessary due to the IBT-4 drivers failing during testing. The team conducted extensive research and testing using a DC Load Simulator and determined that the Cytron drivers would satisfy the operating requirements of the rover. The motor drivers set the speed of the motors by pulse width modulation (PWM) and are controlled using an MCU that will be discussed in the next section (see Fig. B.2.2 for wiring diagram).

The second power network consists of a 12 V 10 Ah LiFePO4 battery [7] that powers the optical encoders and Jetson. Two buck converters [8] are used to drop the 12 V of the smaller battery to the required 5 V for input to the encoders and Jetson (see Figs. B.2.3 and B.2.4 for wiring diagrams). The output of each encoder is passed through a level shifter to drop the 5 V output to the required 3.3 V logic input for the MCU (see Fig. B.2.5 for wiring diagram). The MCU is powered through its connection to the Jetson. The overall wiring diagram for the rover can be found in Appendix B.2.6.

The control circuity is currently connected through a combination of wires and breadboards that is mounted on the rover platform. To reduce the likelihood of a wire or component becoming detached from the rest of the circuit, a custom PCB (printed circuit board) was designed and fabricated using Autodesk EAGLE following the circuit schematic and board representation shown in Appendix B.2.7 and B.2.8, respectively, which is functionally identical to the current wiring implemented on the rover. Terminal blocks have been soldered to the final iteration of this PCB to allow for a stable connection to components that are not housed on the board itself. The final board is shown in Appendix B.2.9.

D. Motor Control Unit and Software Subsystem

An Espressif ESP32 WROOM32D [9] was chosen as the microcontroller (MCU) to control the motors and receive feedback from the rotary encoders. The ESP32 provides more pulse width modulation pins than the Nvidia Jetson and has dedicated pulse counter units that can be configured to keep track of the inputs from the encoders. Overall, the ESP32 provides a much better solution for controlling the motors than the Nvidia Jetson due to the abundance of I/O ports that are necessary for handling the 24 inputs and outputs.

The control software takes two types of inputs: two revolution-per-minute (RPM) set points and twelve encoder pulses. The RPM set points are provided by utilizing Universal Asynchronous Receiver-Transmitter (UART) to communicate with the Nvidia Jetson running Robot Operating System (ROS). There is an RPM set point provided for each side of the rover (left and right wheels). Each rotary encoder provides two pulses, one for forward rotation and one for reverse rotation, which is processed and converted by the control software into an RPM value for each wheel.

The control software on the ESP (Fig. 4) also provides three types of outputs: six PWM signals, six direction signals, and six RPM averages. The six pulse width modulation control

signals regulate the speed of each motor individually. The six direction signals control the direction of each motor. The six motor drivers each receive one direction signal and one pulse width modulation signal to provide full control of the motor. The six RPM averages are calculated using the ten most recent measurements for each wheel. They are utilized to maintain the set point velocity for each wheel and sent to the Nvidia Jetson to aid in localization of the rover.

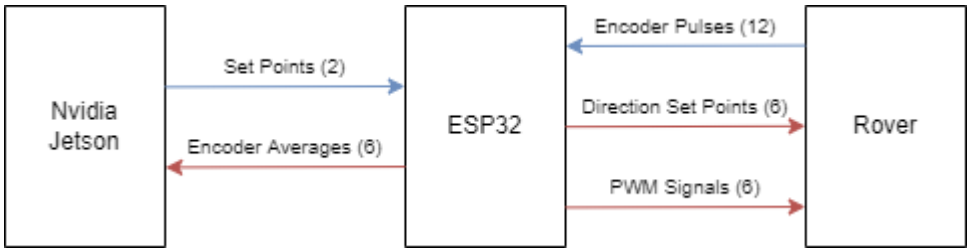


Figure 4. High level I/O flow chart (red – output | blue – input)

The motor control software has two main tasks and one interrupt. The MCU utilizes Real Time Operating System (RTOS) using a task scheduler. RTOS executes tasks based on priority with interrupts having the highest priority. The MCU's pulse counter is utilized to detect and count the rising edges of each encoder input signal. Each encoder has two signals, one signal will increment the counter if the wheel is rotating forward, while the other signal will decrement the counter if the wheel is rotating backward. To process these signals, an interrupt occurs every 10 milliseconds. The software flowchart for the interrupt can be found in Fig. 5. The interrupt calculates the change in pulses since the last interrupt (A), and then places the updated pulse counts for each encoder in a First-in First-Out (FIFO) queue (B) before coming to an end. Figure 5 displays the flow of this interrupt. Semaphores (a mechanism that controls access to common resources) are utilized throughout the program to ensure that the two tasks do not interfere with the same variables at the same time.

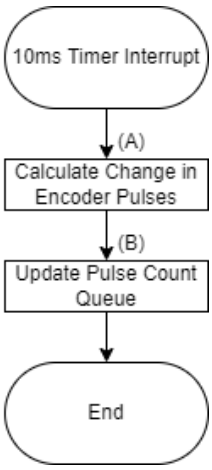


Figure 5. Interrupt flow chart

The motor control thread has the highest priority of the two threads. The whole motor control thread runs on an infinite loop with a delay at the end to free up the task scheduler and allow time for the lower priority task to execute. The software flow of the motor control thread can be found in Fig. 6. The first process that takes place in the motor control task is the reading of encoder data from the pulse count queue. The new encoder inputs are then averaged with the 10 most recent readings to dampen the effect of noise and inconsistencies in the data. The new encoder readings are then converted into RPMs, stored, and then sent to the Nvidia Jetson over UART for processing into ROS (A). Next, a direction check is performed to see if the new set points received by the Nvidia are in a different direction than the current one (B). If there is a direction change, then the rover decrements the duty cycles of each motor (B.1) down to zero while reading the new encoder values after each decrement (B.2). If there is no direction change then this step is skipped, and new duty cycles are calculated. There is a small delay of 5 milliseconds after each encoder reading to ensure that the rover does not come to an abrupt stop (B.3). Once the rover has come to a complete stop (B.4), the new direction is determined based on the magnitude of the set points and the direction pins are changed (B.5). It is necessary for the rover completely stop before changing directions to ensure safe operation of the motors. Next, the old set points are updated with the new ones (C). A new duty cycle for each motor is then calculated using an incremental controller (D). The incremental controller helps maintain the duty cycle of each motor to within ±4 RPM of the desired set points by comparing the desired set point to the encoder readings. The controller also has maximum duty cycle protection to ensure that the motors do not speed up out of control. Finally, the new duty cycles are sent to the motors

(E). There is a 5-millisecond delay once the new duty cycles are sent to the motors before the loop starts over again (F).

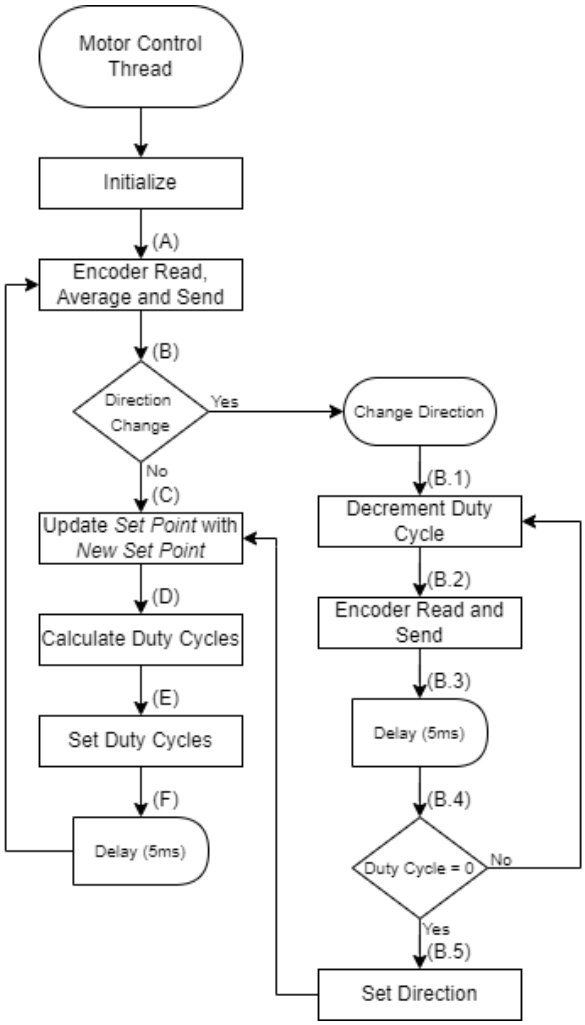


Figure 6. Motor control thread flowchart

The UART event thread is responsible for reading information from the serial communication buffer. A software flow diagram summarizing the UART thread can be found in Fig. 7. If there is a new set point in the UART buffer (A), then the new information is read (B) and stored into a new set point variable (C). Once the variable is updated, the task comes to an end.

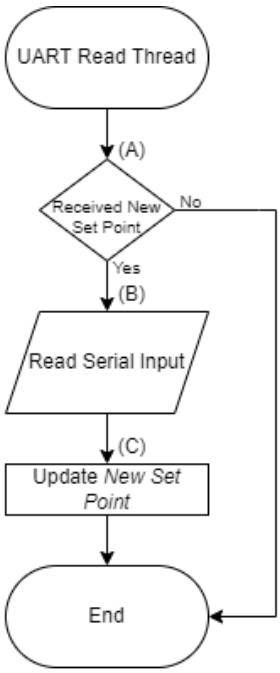


Figure 7. UART read thread flow chart

E. Navigation and Obstacle Detection Subsystem

a. Stereo Camera – Zed2

In order to observe its environment and detect the presence of obstacles, the rover is equipped with a StereoLabs ZED2 stereo camera [10]. The ZED2 is an optical stereo camera and captures video data from two cameras built into the device. This data is then processed by the Zed Wrapper node, a ROS node developed by StereoLabs that provides a ROS-accessible entry point to the ZED Software Development Kit (SDK). The ZED SDK uses an internal AI model to convert the two video streams into a unified depth estimate of each point within the field of view of the ZED. This depth estimate is then processed into a Point-Cloud (the ROS-standard representation of depth data) and published over a ROS topic by the Zed Wrapper node. The Point-Cloud is then ingested by the navigation stack running within ROS to assist in determining the optimal path for the rover to undertake toward its next waypoint. The ZED2 was selected by the team due to its high level of compatibility with ROS. Since the Zed Wrapper node had already been developed by StereoLabs, the selection of the ZED2 allowed the team to continue development on other aspects of the design without concern for completing integrations between the obstacle detection sensor and the navigation stack. Additionally, the ZED Wrapper node publishes both inertial measurement data (calculated from an onboard accelerometer and

gyroscope) and visual odometry data (position estimation based on visual loop closure) over separate ROS topics. This allowed the team to employ an Extended Kalman Filter (EKF) to fuse the wheel odometry data produced by the MCU with these additional odometry estimates to provide a unified odometry estimate, limiting drift in the odometry data caused by error accumulation.

The ZED was attached to the rover frame using a 3D printed mount. Figure 8 shows the CAD model for this design.

Figure 8. 3D printed mount for ZED2 camera

The ZED was fixed to this mount via a screw placed through its central hole. The mount has a built-in angle of 4°, shifting the FOV of the ZED closer to the front of the rover. The mount itself was attached to the T-channel frame using two screws through the wholes on the side of the model fixed to drop in nuts placed inside the T-channel (shown in Fig. 9).

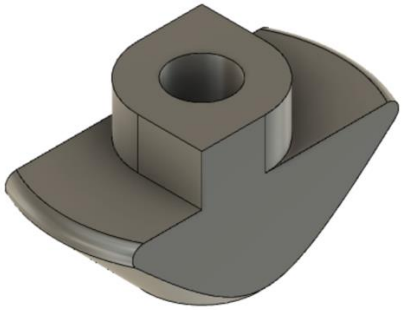


Figure 9. Drop in nut for aluminum T-channel [11]

b. Robot Operating System

The navigation capabilities of the rover are provided by software running on the Nvidia Jetson Nano onboard the rover. The software running on the Jetson is powered by Robot Operating System (ROS), an open-source set of software packages and tools that enable robot development. ROS functions through the use of individual software processes called nodes. Nodes are connected through a "hub-spoke" model, with a master node (the "hub") known as "roscore" controlling the communication between nodes (the "spokes"). Nodes communicate with each other over a network using TCPROS, a transport layer that uses Transmission Control Protocol (TCP) to transmit information between nodes. Information is exposed and ingested by nodes using ROS topics. Topics are buses that allow nodes to exchange information in the form of messages. Messages are packets of information with a prescribed format, allowing nodes to communicate information with minimal parsing logic. For a node to receive messages it subscribes to a topic, and to transmit messages it publishes to a topic.

Additionally, ROS nodes can also expose direct functionality through ROS Services and Actionlib. Unlike ROS topics, where messages are published without awaiting any response from a subscriber, ROS Services function through two-way communication, with the Service caller receiving a response from the Service advertiser. This allows for nodes to expose functionality to each other in a manner similar to a Remote Procedure Call (RPC), meaning that nodes can leverage functionality from other nodes without explicitly handling the necessary communication procedure. Actionlib functions similarly to ROS services but allows for task preemption. While ROS Services only communicate back to the caller upon service completion, Actionlib allows the advertiser to communicate information back to the caller throughout execution and allows the caller to cancel its request while the action is being performed.

The primary set of nodes running on the rover comprises what is known as the navigation stack. The navigation stack receives waypoints from the user and plans and executes a movement for the rover to reach the desired waypoint. A diagram depicting the overall node structure and information flow within the navigation stack is displayed in Figure 10.

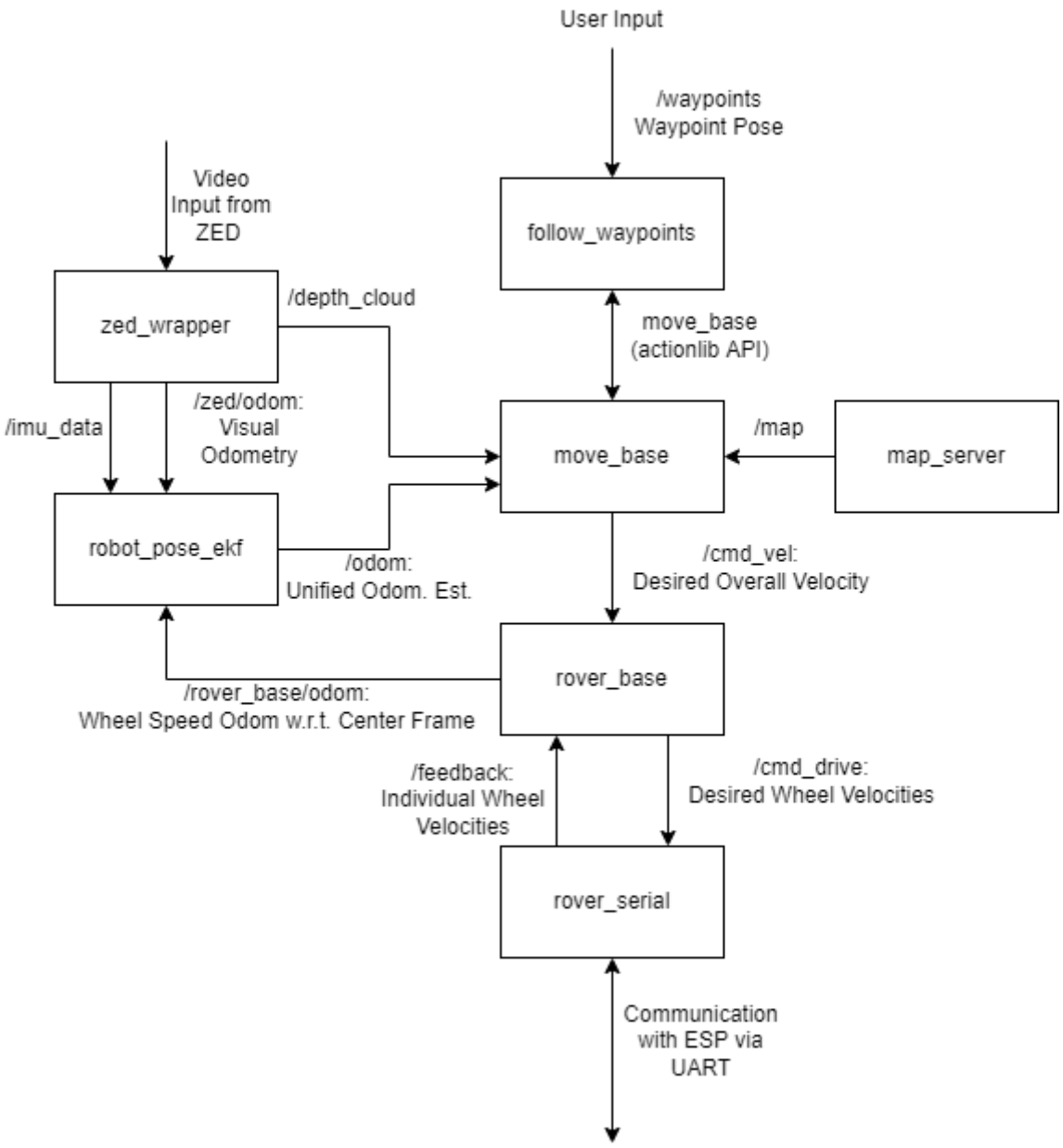


Figure 10. Node structure and information flow of navigation stack

Waypoints are provided to the navigation stack by the user as a six degree-of-freedom pose. The pose fed to the navigation stack represents the desired position of the center frame of reference of the rover relative to the global frame of reference, encoded as translations in the x, y, and z dimensions, and rotation of the frame about the x, y, and z axes (roll, pitch, and yaw, respectively). The waypoint pose is fed into the follow_waypoints node via a ROS topic. The follow_waypoints node buffers these waypoints until it receives a message to begin navigation through a separate ROS topic. Upon receiving this message, the follow_waypoints node begins

issuing waypoints to the move_base node through an exposed Actionlib API, allowing the follow_waypoints node to monitor the progress of the rover and cancel navigation if necessary. The follow_waypoints node functions as a simple state machine, passing the next waypoint to move_base as the current waypoint is reached until all the buffered waypoints have been reached.

The move_base node is responsible for pathing the rover through its environment. A diagram of its functionality is displayed in Fig. 11.

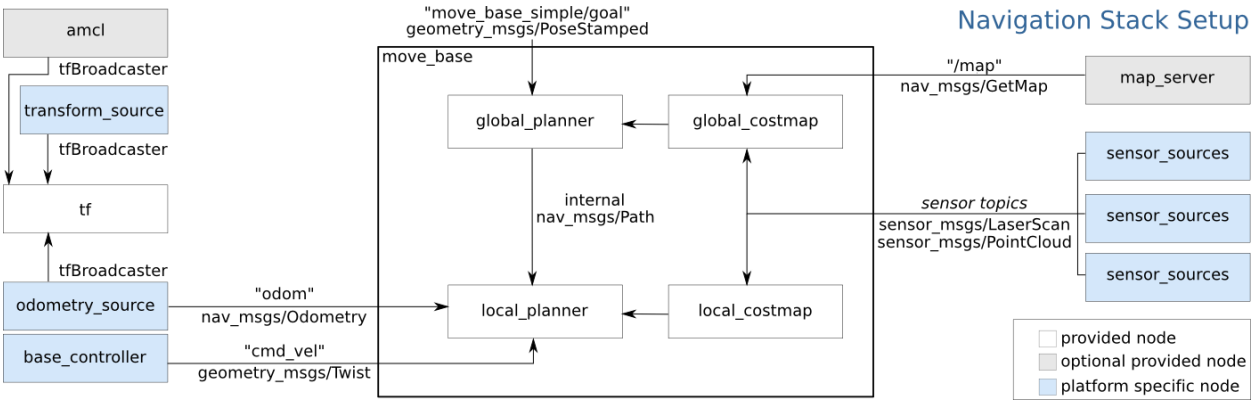


Figure 11. Diagram of move_base functionality [12]

The move_base node consists of four primary elements: the global costmap, local costmap, global planner, and local planner. The global and local costmaps are both maps of the rover's environment, with each point on the map representing the estimated cost of moving the rover to that point on the map. A high cost indicates the presence of an obstacle or obstruction, meaning that the rover is likely to get stuck, while a low cost indicates a position that can be traversed by the rover. In contrast to the provided map, which only encodes the environment in binary (traversable/non-traversable), the costmaps introduce a layer of fuzzy logic, allowing the rover to compare paths that are both nominally traversable. The costmaps are generated by producing an inflation layer around known and detected obstacles. The inflation layer is calculated by placing a fixed radius of decreasing cost around each known/detected obstacle. The local costmap is generated using a rolling window method, meaning that the costmap is continuously regenerated in a 4-meter square window around the center reference frame of the rover, as shown in Fig. 12, with the white rectangular region representing the local costmap, and the greyscale regions in the white rectangle indicating obstacles and the inflation layer.

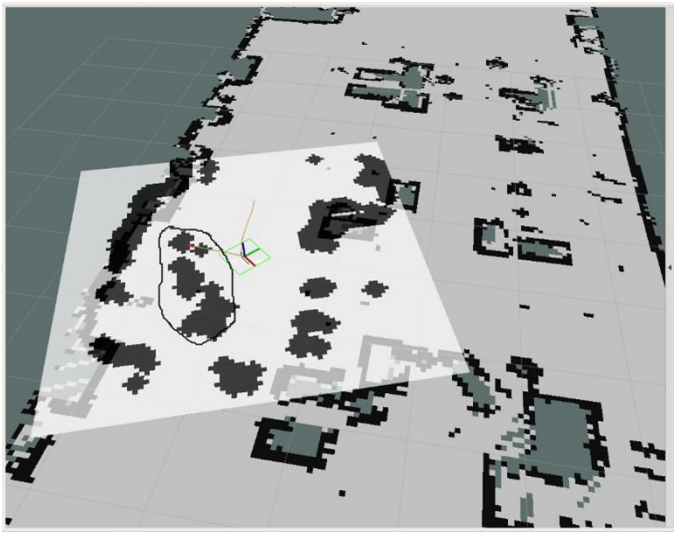


Figure 12. Example of a local costmap [13]

The local costmap incorporates the depth data produced by the Zed Wrapper node in real time (see section b). In contrast, the global costmap is fixed, generating an inflation layer over the entirety of the provided map that includes obstacles previously detected within the local costmap. An example of a global costmap generated by the navigation stack is displayed in Fig. 13, with the red regions indicating obstacles, and the blue regions representing the obstacle inflation layer.

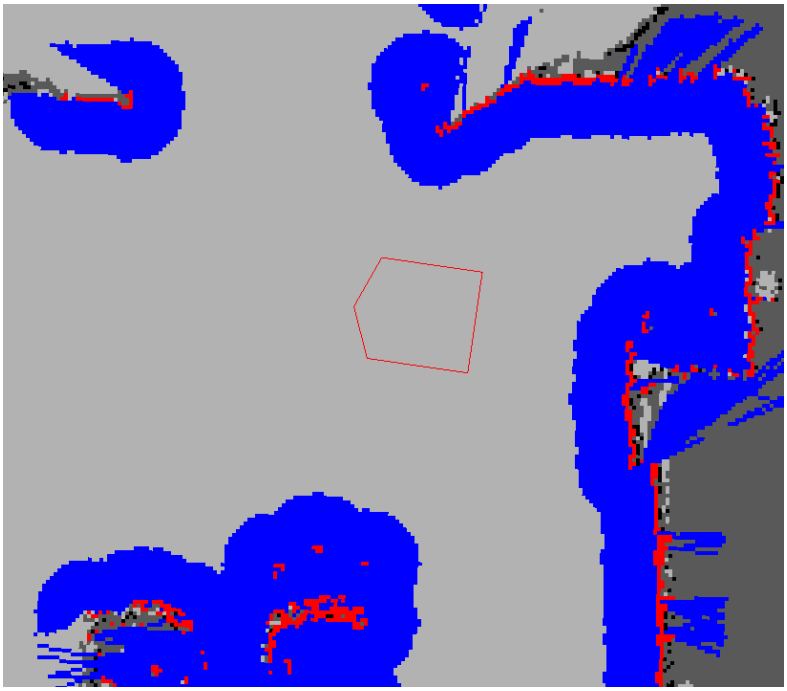


Figure 13. Example of a global costmap, with inflation layers generated over map occupancy data [14] -> GLOBAL COSTMAP

The information produced by the global and local costmaps are consumed by the global and local planner, which determine the path the rover will take toward its next waypoint. The global planner plans the overall trajectory of the rover to its waypoint based on obstacles known when the waypoint is received. The local planner continuously reevaluates its planned trajectory as the local costmap changes. The local planner attempts to uphold the overall trajectory planned by the global planner but will diverge from the global plan if a new obstacle is detected. The local planner implemented on the rover uses a planning method called Timed Elastic Band (TEB) [15]. The TEB planner uses a weighted optimization algorithm to determine the optimal path for the rover, taking into account time of execution, adherence to the global plan, obstacle avoidance, and upholding the kinematic and dynamic constraints of the rover. The trajectory determined by the TEB planner is then processed into a desired angular/linear velocity by the move_base node and published to the rover controller over the /cmd_vel topic.

The desired velocities issued by the move_base node are with respect to the center frame of reference of the rover. In order to actually execute these desired velocities, they must first be processed into individual wheel velocities. The translation of overall velocity into individual wheel velocities is carried out by the rover_base node. The rover_base node is a custom node written by the team which implements diff_drive_controller, an open-source ROS library. The diff_drive_controller library provides a linkage between the commands issued by the navigation stack and the hardware of differential drive and skid-steering robots. The rover_base node reads desired overall velocities from the /cmd_vel topic and uses the diff_drive_controller library to calculate desired wheel velocities. These desired wheel velocities are then issued using a custom message type over the /cmd_drive topic to the rover_serial node. The rover_serial node is another custom node that communicates the desired wheel velocities issued by the rover_base node to the MCU over USB using UART. Additionally, the rover_serial node reads the individual wheel speed feedback from the MCU over the same USB UART connection. The wheel speed feedback is then issued back to the rover_base node and is used by the diff_drive_controller library to produce an odometry estimate for the rover's center frame of reference. This calculated velocity is then issued to the robot_pose_ekf node, which implements an Extended Kalman Filter to fuse the wheel odometry data with IMU and visual odometry data produced by the zed_wrapper node into a unified odometry estimate. The unified odometry

produced by the robot_pose_ekf node is then consumed by the move_base node, which updates the local costmap and local plan based on the change in rover position.

F. Tools and Methodologies

To build the final prototype, multiple tools were used from the makerspace and electronics shop. For maintenance and modification of the frame provided by the project sponsor, various saws from the machine shop were used to cut specific lengths of extruded aluminum T-channel.

To 3D print the shaft couplings, timing pulleys, and camera mount, the Markforged Onyx Pro Gen 2 printer was used. All these components required the level of precision provided by the Markforged printer so that they could be placed with precision and not have any unnecessary movement. Specifically, the couplings were printed out of Onyx because it is an extremely strong and durable material with a high temperature resistance. This allows the couplings to hold up to the large amounts of torque and friction coming from the translation of power from the motors to gearboxes.

The wheel spacers and encoder housings that attach the body of the motors to the gearboxes were printed with tough PLA using the Ultimaker 35 dual extruder. This printer and material were selected because the prints are comparatively large and the cost of material for tough PLA is considerably lower than Onyx. Additionally, tough PLA provides enough strength for these components to behave properly, and the level of detail required for these parts is low enough to warrant using the Ulti-maker rather than the Markforged.

The wiring of the rover was done using crimpers, wire cutters, and various types of connectors found in the electronics shop. The heavier duty wires required the hammer, anvil, and wedge from the makerspace to effectively crimp the large connections over the larger diameter wires. The connectors for the larger wires were purchased from a second party as the electronics shop does not carry components for such large wire.

All software pertaining to ROS was developed using Visual Studio Code with version control handled using GitHub. The software for the ESP32 was developed using the ESP integrated development framework (IDF) and was also managed with GitHub.

IV. Design Evaluation

A. Terrain Traversal

This section discusses the requirements pertaining to the rover's ability to traverse its environment. The test conducted against the requirements and an evaluation of the tests results are discussed. Unfortunately, one of the requirements listed was not able to be tested as a significant failure occurred preventing any further testing. This failure will be discussed in detail in a later part of this report.

Requirement Descriptions

The rover should be able to traverse up an incline of 4% (2 degrees) and may traverse over an obstacle no more than 2 inches tall.

The rover must navigate between waypoints and avoid obstacles without human intervention or control.

Relevant Test: *Movement and Motor Control*

This test was conducted against the first requirement listed above to determine if the rover can effectively move forward, backward, up a slope, and over an obstacle without veering off to either side. The requirement of traversing over an obstacle was not tested, as a significant system failure occurred before that section of the test was conducted.

Objectives and Features Evaluated

The objectives of this test were to ensure that the current draw of the motors did not exceed the limits of the electronic components, verify that the control scheme implemented on the ESP32 effectively produced, limited, and varied the required PWM sent to each motor to successfully reach the velocity setpoints hardcoded into the test code, and to evaluate that the motors and gearboxes can provide the necessary torque to satisfy the project requirements and features dependent on them. The features evaluated were the rover's ability to travel forward and backward without significant veering, travel up a slope, and travel over an obstacle.

Scope

This test exclusively verified the function of the motors and the associated control software and hardware. Accordingly, the test software was hardcoded onto the ESP and no autonomous navigation functionality was tested.

Test Setup

This test required an external computer to flash the test code onto the ESP32, as well as a clamp multimeter to measure the current draw of the motors. Additionally, tape was placed on the floor of CSI in one-meter increments and a camera was secured to the rover frame to measure the time it took for the rover to reach each mark allowing the rover's average velocity to be calculated.

Test Plan

This test was performed by manually setting the velocity set point for the incremental controller in the test code that determined the proper PWM signals to issue to the motor drivers. The commands sent told the rover to move forward and backward, each for a specified amount of time. The current drawn from the motors was measured using the clamp multimeter secured around the ground cable connected to the battery. Next, commands were sent to drive the rover forward onto a ramp. Finally, the rover was to drive forward over an obstacle.

Acceptance Criteria

For this test to have been deemed a success, the rover had to successfully move in all commanded directions as well as travel up an incline without stall or failure. If there was significant variation between odometry and observed velocity, the odometry system would have to be reevaluated or redesigned. If the motors failed to move on either the flat or inclined surface, the motors and gearboxes would have needed to be replaced with motors that have built in gearboxes and odometers to ensure more uniformity in output ratios and better torque performance.

Results

The rover was able to successfully move forward and reverse on a flat surface and was able to successfully travel up and down the wheelchair accessible ramp in front of the Mars Mclean building, which has an approximate slope of 2 degrees. The rover was not able to traverse over an obstacle as before this section of the test was conducted, the motors of the rover failed and were rendered useless. The current measurements are all well-within the 200 Amp limit that the battery can provide. The measured total currents can be found in Table 1.

Table 1. Current draw for different course conditions

Situation	Current Draw (Amps)
Flat surface, forward	60
Flat surface, backward	80
2-degree slope, forward	110
2-degree slope, backward	130

The rover was able to successfully maintain a set point velocity of 0.524, 0.589, and 0.654 meters per second. The externally measured velocity was calculated visually, and it was consistently within 0.05 meters per second of the velocity reported by the encoders, as seen in Fig. 14. The externally measured velocity was only calculated once the rover reached a steady state speed.

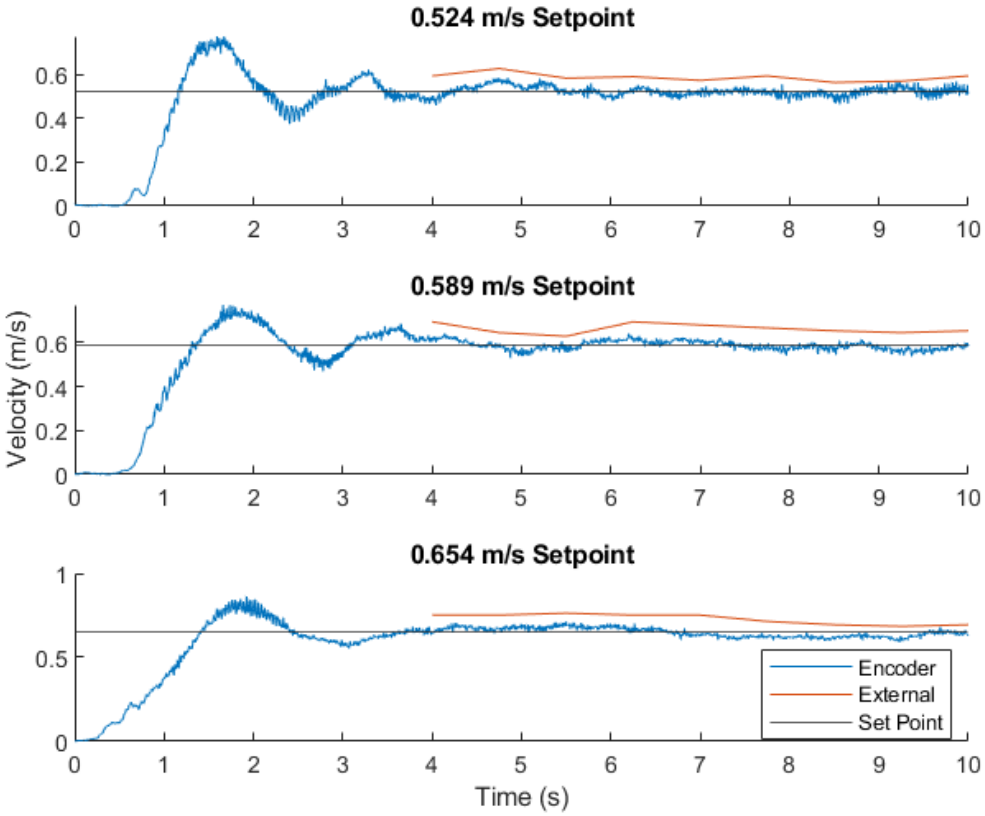


Figure 14. Velocity response measured by the encoder (averaged across each wheel) compared to external measurement

Evaluation

The results of the current draw test demonstrate satisfaction of the current draw requirement, as the highest reported measurement was 130 Amps, while the battery can provide up to 200 Amps. The motors were able to provide the necessary torque to drive on a flat surface and go up a slope thus meeting those requirements. The speed test proved that the motor controller can effectively maintain a set point speed, thus functionality necessary to meet the requirement of traversal without human intervention was established. Finally, the motors were not able to withstand continuous testing and after months of usage and hours of continuous operation were rendered useless, thus the soft requirement of traversing over an obstacle was not tested and therefore not met.

Relevant Test: *Skid Steering Controller and Serial Communication Test*

This test was conducted to verify that the skid steering controller correctly translates overall velocities into individual wheel velocities, and that the serial communication interface between the Jetson and ESP correctly communicates both desired velocities and velocity feedback between the motor PID controllers and the skid steering controller.

Objectives and Features Evaluated

The objectives of this test were to ensure that the serial communication interface between the ESP32 and Jetson could relay messages promptly enough to not inhibit functionality, and to ensure that the skid steering controller was correctly computing individual wheel velocities based on the desired overall velocity. This test evaluated two features of the rover: two-way communication between the Jetson and the ESP32, and directional control of the rover.

Scope

This test exclusively evaluated the proper communication of wheel velocities issued by the skid steering controller. The physical motion of the rover and functionality of the incremental controller under load was not evaluated. Additionally, the velocity commands were issued to the skid steering controller directly, and the navigation algorithm was not enabled during the test.

Test Setup

To test skid steering and communication, the rover was placed on blocks and remained stationary throughout the duration of the test. The skid steering controller and serial

communication test required a laptop to remotely issue velocity commands to the skid steering controller and monitor the velocity feedback from the encoders.

Test Plan

Velocity commands (specific velocity commands displayed in Table 2) were issued directly to the skid steering controller, and the odometry feedback was viewed to evaluate the performance of the serial communications and the skid steering controller. The velocity setpoint and actual wheel velocity were displayed on the laptop for the data to be collected and analyzed.

Acceptance Criteria

Correct functionality of the serial interface was confirmed by a proportional response of the elevated motors to the velocity command issued to the skid steering controller along with observation of odometry feedback from the ESP on the Jetson. If no response was apparent to issued commands or no odometry feedback was received, the serial interface had failed to communicate between the Jetson and the ESP. The proper function of the skid steering controller was confirmed by correct velocity directions for either side of the rover for the desired overall velocity, i.e., both sides having the same direction in pure backwards and forwards motion, and opposite directions in turning motions.

Results

The Skid Steering Controller was able to successfully convert overall velocities into wheel velocities, as displayed in Table 2.

Table 2. Summary of overall setpoints and wheel setpoints issued by controller

Overall Setpoint	Left Side Setpoint [rad/s]	Right Side Setpoint [rad/s]
+0.25 [m/s]	2.016	2.016
-0.25 [m/s]	-1.984	-1.984
+2.5 [rad/s]	-6.147	6.147
-2.5 [rad/s]	6.147	-6.147

The test also demonstrated successful communication between the Jetson and the MCU, with changes in setpoint executed by the MCU without delay, and feedback received by the Jetson without latency or data corruption. Finally, the Skid Steering Controller successfully converted the wheel velocity feedback into overall rover velocity, as displayed in Table 3.

Table 3. Comparison of velocity setpoints and calculated odometry

Direction	Setpoint Velocity	Average Calculated Odometry	Percent Error [%]
Forward	+0.25 [m/s]	0.258 [m/s]	3.20
Backward	-0.25 [m/s]	-0.255 [m/s]	2.08
Left	+2.5 [rad/s]	2.493 [rad/s]	0.26
Right	-2.5 [rad/s]	-2.494 [rad/s]	0.23

It can be observed from Table 3 that the MCU was able to control the motor speed (in an unloaded scenario) with less than 5% error for each direction of motion.

Evaluation

The results of this test demonstrate that the velocity setpoint and feedback communication occurs between the Jetson and MCU without issue. Additionally, this test demonstrates that the software running on the Jetson provides an adequate interface between the navigation stack and the actual rover hardware. Therefore, these features display the necessary functionality to meet the requirements of navigation and traversal without human intervention.

B. Run Time

This section of the report pertains to the rover's ability to operate for a specified amount of time. As mentioned in section IV., sub-section A, a significant system failure occurred during testing that prohibited the team from completing all planned tests. This section discusses one such test and provides the details of the failure and how it informs the requirement listed in this section.

Requirement Description

The rover must be able to run continuously for one hour without charging.

Relevant Test: *Run Time Test*

Objective and Features Evaluated

This test was to be conducted to evaluate the rover's ability to run for one hour on a single battery charge.

Scope

This test was limited to evaluating if the motors and related hardware could withstand continuous operation for one hour. Subsequently the camera and automation software would not have been used during this test.

Test Setup

This test would have been conducted in a large open space on campus where the WIFI signal provided enough strength and coverage. The test would require a laptop to interface with the Jetson to hardcode commands causing the rover to move in a continuous loop. A thermal camera would have been used to monitor the temperature of each component and a clamp current meter would have been used to measure current to ensure operating conditions were not exceeded.

Test Plan

The rover would have been placed at a starting point on a continuous loop around campus and commands would have been sent to set the rover in motion at a constant velocity. The team would follow the rover for the one-hour time limit while monitoring the temperature and current draw of the motors and motor controllers at ten-minute increments.

Acceptance Criteria

This test would have been considered successful if the rover was able to operate continuously for one hour without any component failure.

Results

This test was not conducted due to all six motors failing during testing of the navigation software. During that testing the rover was operated on and off for approximately three hours. While the conditions of that test did not meet the requirement of continuous operation, the failure of the motors constitutes some result of their operational limitations. After on-off operation for approximately three hours, some of the motors stopped operating while those that could still turn on did not provide enough torque. Subsequently the team removed each motor from the rover and took them apart to determine the point of failure. The team determined that the motors overheated resulting in malfunctions of both the brushes and coils of the DC motors. An image of these components is shown in Fig. 15.

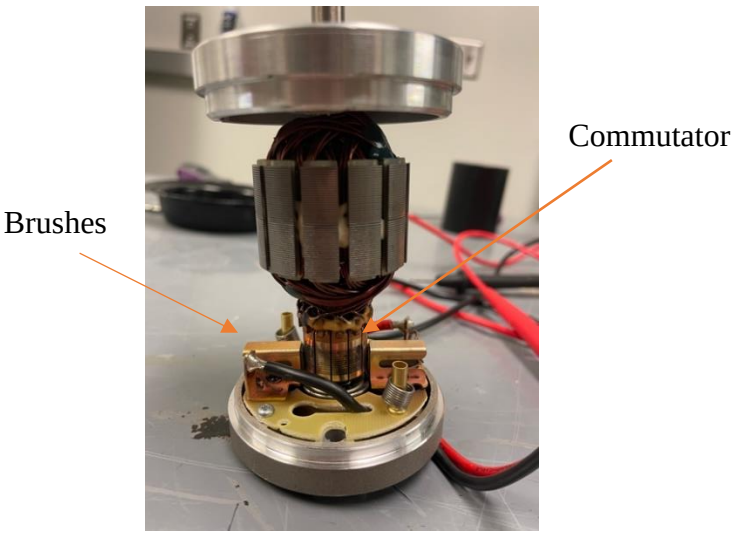


Figure 15. Internal workings of motors (without stator magnet)

Each motor has a commutator comprised of six pairs of panels mounted opposite of each other around the shaft. Each pair is attached to the ends of two oppositely mounted coils that produce the electromotive force (emf) needed to drive the motor while current is ran through them path. The brushes complete the circuit of both coils during its pass over the commutator panel which allows the coils to produce emf. The team discovered that some of the brushes were not in contact with the commutator thus causing the motor to be inert. Visual inspection, seen in Fig. 16, shows that overheating caused melting (indicated by arrows on Fig. 16) on part of the brush mechanism which in turn caused the brushes to be held away from the commutator.

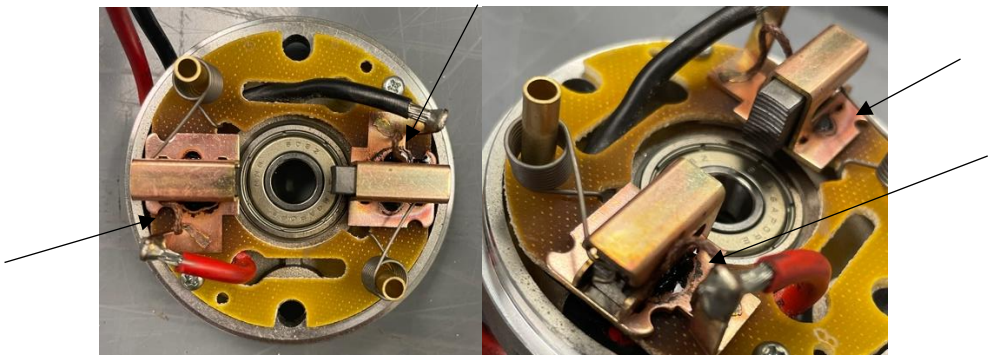


Figure 16. Brush mechanism showing jammed brushes and melted substance

If any of the coils malfunction, the motor will not produce the required emf and thus the required torque. Therefore, the resistance of each of these coil pairs was measured using an Agilent Technologies U3401A Digital Multimeter to determine if they are operating properly. Each coil pair can be treated as two resistors in parallel, therefore if one of the coils is broken, the measured resistance should be greater than or equal to twice the nominal resistance of a

functioning coil which is approximately 0.028 Ω. If the measured resistance is 0 than one of the coils has produced a short circuit. Either case would prevent the motor form providing any useable torque. Measurements of each coil resistance can be found in Table 4.

Table 4. Measured coil resistances after failure

Motor	Coil Pair 1 [Ω]	Coil Pair 2 [Ω]	Coil Pair 3 [Ω]	Coil Pair 4 [Ω]	Coil Pair 5 [Ω]	Coil Pair 6 [Ω]
Front Left	0.06	0.075	0.07	0.09	0.085	0.08
Front Right	0.04	0.045	0.04	0.05	0.03	0.04
Middle Left	0.07	0.08	0.08	0.08	0.09	0.08
Middle Right	0.095	0.08	0.095	0.08	0.08	0.075
Back Left	0.06	0.06	0.07	0.065	0.07	0.07
Back Right	0.06	0.07	0.06	0.07	0.075	0.06

Evaluation

While the test was not technically conducted, the failure resulting from the other test was evaluated. Based on the results in Table 4, it can be shown that all coil pairs had resistances higher than the nominal resistance with each motor having at least one that is twice the nominal value. This means that these motor coils can no longer effectively produce the required emf to provide any amount of useful torque. Therefore, the rover was unable to meet this requirement. It is important to note that the motors were inherited by this team. The team considered replacing them, however, the gearboxes had been explicitly designed for these motors and frame, thus replacing them would have required more time to integrate new motors into the frame which was outside of the scope and purview of the project as was laid out in the project proposal.

C. Terrain Navigation

This section of the report pertains to the requirements governing how the rover navigates terrain and views its surroundings.

a. Display of Position

Requirement Description

The estimated position of the rover should be displayed on the provided map on a connected laptop.

The rover must be able to steer (detect) around obstacles larger than the clearance of the motor and gear box.

Relevant Test: *Encoder to Wheel Output Conversion Factor Calibration and Verification*

This test served to calibrate and verify a conversion factor between wheel and encoder outputs.

Objectives and Features Evaluated

The objectives of this test were to calibrate the conversion factor between wheel speed and encoder output and verify that factor's effectiveness in predicting the speed of each wheel.

Scope

The scope of the test was limited to evaluating the relationship between wheel speed and encoder output at a constant set point velocity in the forward direction.

Test Setup

The RPM of a wheel was measured using a tachometer attached to the body of the rover pointing to the wheel which had reflective tape attached to it. A camera was used to capture the tachometer output. A laptop was required to hardcode velocity setpoints and record odometry output for comparison with the tachometer.

Test Plan

The theoretical encoder conversion factor was calculated and used to convert the output from the encoders to the revolutions per minute (RPM) of the wheels. The encoder conversion factor was then recalculated to match the encoder output to the wheel RPM. To verify the encoder conversion factor, the motors were all set to the same RPM value and a handheld tachometer was used to determine the RPM of each wheel. The set point was controlled by an

ESP32 and the motor drivers. The wheel RPM measured by the tachometer was then compared against the wheel RPM measured by the encoders and the set point. This test was run for every wheel to verify the encoder conversion factor was accurate for each wheel. The test was repeated two more times at different set points to further verify the accuracy of the encoder conversion factor.

Acceptance Criteria

This test is considered successful if the calibrated conversion factor accurately converts odometry output to match the measured wheel RPMs for all wheels.

Results

The theoretical encoder conversion factor was calculated using Eq. 1:

$$\frac{\#\ pulses}{10\ ms} \cdot \frac{1000\ ms}{1\ sec} \cdot \frac{60\ sec}{1\ min} \cdot \frac{1\ shaft\ rev.}{200\ pulses} \cdot \frac{1\ wheel\ rev.}{12\ shaft\ rev.} = 2.5\ RPM \qquad (1)$$

Using this correction factor yielded results that were inconsistent with the measured RPM by the tachometer on the front left wheel. The correct conversion factor was empirically determined to be 1.2 RPM. This conversion factor made the encoder output of the front left wheel within ±2 of the RPM the tachometer measured.

To verify that the encoder conversion factor was the same for all wheels, the RPM for every wheel was set to 40, 45, and then 50. The results for the 50 RPM test can be found in Fig. 17.

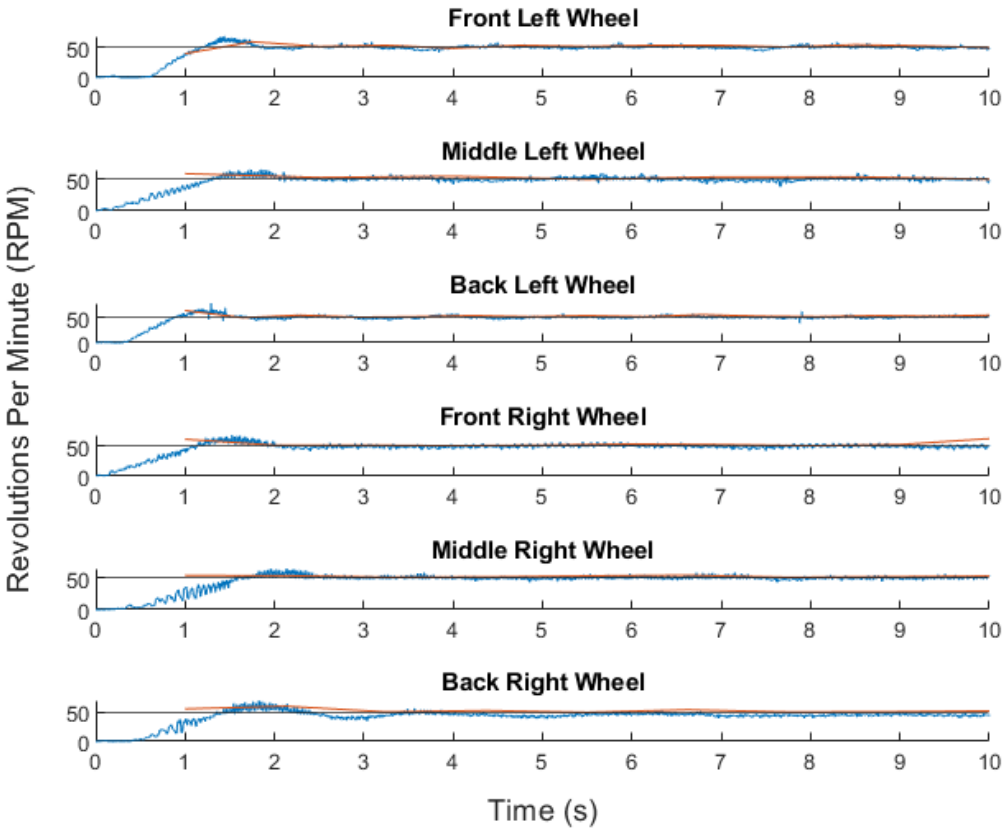


Figure 17. Encoder conversion factor verification test with a set point of 50 RPM (Blue line is encoder output, orange is tachometer measurement, black is setpoint)

Evaluation

The tachometer measurements have a delay compared to the encoder outputs because the tachometer can only make measurements every wheel rotation, while the encoder can make 200 measurements per wheel rotation. The RPMs of each wheel at every setpoint were well within the acceptable range of ±4 (as determined in Section IV, Part B). This test met the objectives by determining an encoder correction factor that was able to accurately represent the RPMs of each wheel at different set points. Determining the proper encoder correction factor allowed the rover to track its movement, therefore this functionality contributes to meeting the requirement of displaying the rovers estimated position.

Relevant Test: *Obstacle Detection Test*

Objectives and Features Evaluated

The purpose of this test was to verify that the obstacle detection system could successfully discern obstacles from navigable terrain. This test evaluated the ability of the

obstacle detection system to detect obstacles larger than the clearance of the motors and the ability of the navigation stack to generate a costmap populated with obstacles.

Scope

The test was conducted on a flat surface, with lighting conditions that correlate to a clear, sunny day. The rover will remain stationary throughout the test. This test will not evaluate the variance of performance with lighting conditions or movement.

Test Setup

The ZED was tested in-place on the rover, with the navigation stack software running on the Jetson. Test data was observed using an external computer. The rover will remain stationary throughout the testing.

Test Plan

The obstacle detection test required an external computer to view the cost map (a 2D map that represents the anticipated difficulty of movement for each point in the map [1]) generated by the navigation algorithm, as well as several obstacles of varying shape with heights of at least 2 inches. The rover platform was placed on a flat surface and 2-5 obstacles of varying shape and size were placed at varying locations within the field of view of the ZED camera. The navigation software was launched, and the local cost map generated by the software was viewed using RViz (ROS visualization software) on a separate computer. The generated cost map was evaluated to determine the number of false negatives (obstacles that should have been detected that were marked as traversable in the cost map) and false positives (distinct regions of traversable terrain marked as not traversable in the cost map). Additionally, RViz was used to add 3 waypoints at random, plan traversable locations within the local cost map, and validate the path generated. The test will be repeated with 2 additional obstacle placements.

Acceptance Criteria

The obstacle detection subsystem should not generate any false negatives on obstacles greater than the 4-inch avoidance threshold. The acceptance criteria of false positives were qualitatively based. The results will be deemed acceptable if the false positives generated by the algorithm are small enough in size and scattered enough in distribution to not prevent the rover from generating a movement plan to any of the three random waypoints. If any regions within the field of view of the ZED 2 produce an unacceptable number of false positives or negatives,

the depth range processed by the navigation algorithm will be adjusted to exclude these region(s).

Results

This test did not meet the acceptance criteria during initial testing, but the test was rerun after performing parameter tweaks to the navigation stack. The results of one of the trials of this second test are displayed in Fig. 18-20, with the other two trials displayed in Fig. D-1 through D-6**.**

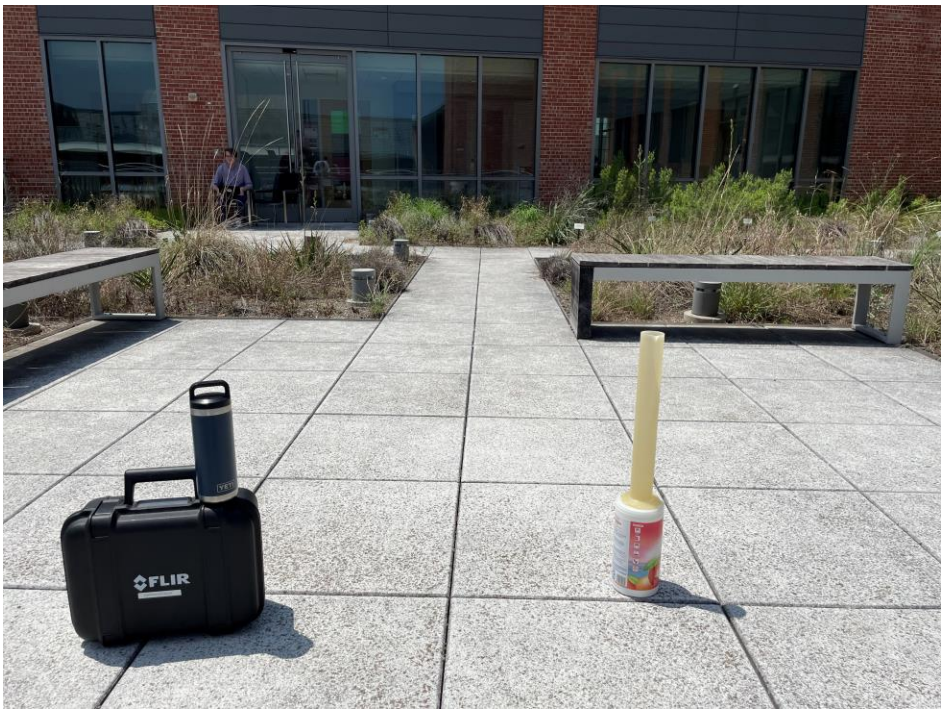


Figure 18. Image of obstacle field for obstacle detection test trial 1

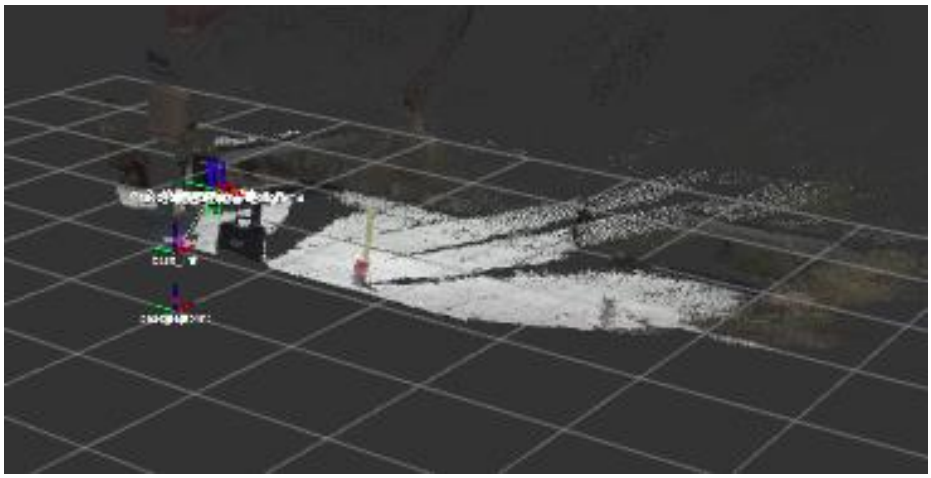


Figure 19. Image of generated depth data for obstacle detection test trial 1

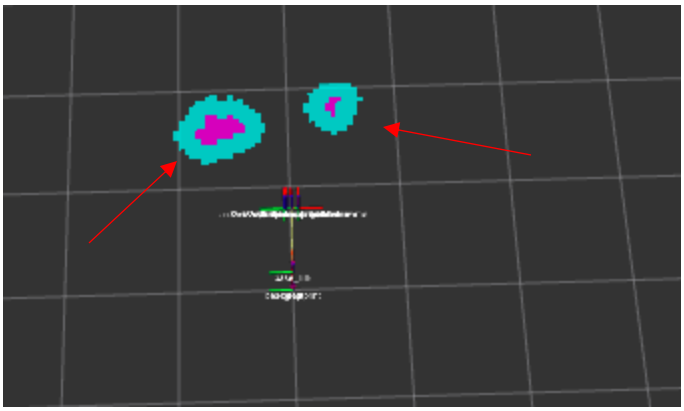


Figure 20. Image of generated costmap for obstacle detection test trial 1 (obstacles indicated with arrows)

In the second test, the obstacle detection system correctly detected all obstacles and did not produce any false positives, satisfying the acceptance criteria for this test.

Evaluation

Through this test, the obstacle detection system demonstrated an ability to correctly discern obstacles from navigable terrain, and the navigation software demonstrated the ability to correctly generate a costmap of its environment. Therefore, this functionality allows the rover to properly detect its environment and make decisions about where it shouldn't go, thus the requirement of detecting obstacles larger than the clearance of the motors was met. Additionally, as seen in the figures above, the estimated position of the rover is displayed on multiple maps, thus that requirement if met.

b. Waypoint Achievement and Adjustment

Requirement Descriptions

The rover must be able to achieve 3/3 waypoints within 10% on at least 4/5 attempts from random initial positions and different waypoints in the test field.

The rover must navigate between waypoints and avoid obstacles without human intervention or control.

The user should be able to add waypoints to the map in the field.

Relevant Test: *Obstacle Avoidance*

The obstacle avoidance test was planned to demonstrate the rover's ability to detect and avoid obstacles without human intervention when given a movement goal.

Objective and Features Evaluated

The objective of this test was to give the rover a movement goal and evaluate if it could detect and respond to obstacles in its path. This evaluates the rover's ability to autonomously detect and avoid obstacles by either driving around them or stopping completely.

Scope

The scope of this test was to evaluate the rover's functionality with respect to obstacle detection and determine if it could effectively avoid obstacles in its path and ignore obstacles that are not in its path.

Test Setup

This test was conducted on the third floor of CSI in the hallway, and it requires the fully assembled rover and a Linux machine. A trashcan was placed along the rover's expected path as an obstacle. Another test run consisted of a trashcan right outside of the rover's path.

Test Plan

The first test run consisted of the trashcan in the expected path of the rover, approximately 10 meters away from the rover's initial position. This would ensure that the obstacle was outside of the rover's initial view so it would be detected once the rover started moving. The rover was then given a movement goal to go forwards for 20 meters. The second test run had the trashcan right outside of the expected path of the rover, approximately 10 meters away from its starting position. The rover was then given a movement command to go forward 20 meters. Finally, the rover was placed on a collision course with a wall approximately 10 meters away from its starting position. The rover was given another movement goal of 20 meters.

Acceptance Criteria

This test would be considered a success if the rover can avoid hitting any of the obstacles placed in its expected path. This includes coming to a stop in front of the obstacles if there is not enough room for the rover to go around the obstacle.

Results

The result for the first test run with the trashcan in the expected path of the rover can be viewed [16]. The rover successfully stopped in front of the trashcan and did not try to avoid it since there was no room for it to navigate around.

The second test run with the trashcan right outside the expected path of the rover can be viewed [17]. The rover was able to determine that the trashcan was not on a collision course with the rover, and it was able to drive past it without stopping.

The final test run with a direct collision path with the wall can be found in [18]. The rover was able to successfully detect the wall and come to a complete stop before colliding into it.

Evaluation

As seen in the videos referenced in the results, the rover was able to successfully stop before colliding into any of the obstacles. This demonstrates that the rover has the ability move towards a navigation goal, detect obstacles that are in its path, and react appropriately to avoid collisions.

Relevant Test: *Full Prototype Test*

The full prototype test was planned to demonstrate that the rover can autonomously navigate to a given waypoint while avoiding obstacles. However, as discussed in section C, the motor failures prevented the team from conducting this test.

Objective and Features Evaluated

The objectives of this test were to verify that the obstacle detection and motor control systems were properly integrated, and that the rover can autonomously navigate to a predetermined waypoint while avoiding obstacles. This test evaluated the rover's ability to avoid obstacles and autonomously reach 3/3 waypoints within 10% of its distance to the waypoint on at least 4/5 attempts from random initial positions.

Scope

The scope of this test covered all aspects of the rover as its purpose is to test the fully functioning prototype against the main project objective.

Test Setup

This test was to take place outside the Center for Sciences and Innovation building on a bright, sunny day. The full prototype test required a laptop to remotely control and monitor the rover's onboard computer. This would allow for the initial position and waypoint to be loaded into the rover's software. The rover would be given its initial position in the test map along with a waypoint that it will travel to, additionally several obstacles would be placed in the test field.

Test Plan

The testing course would have consisted of several obstacles varying from 2 inches tall to more than 4 inches tall along the path to the waypoint. The rover was to start at a given initial position and commanded to move to each waypoint consecutively. Upon arrival at each waypoint, the distance from the rover to the waypoint would have been measured to test its capabilities against the distance requirement listed at the beginning of this section

Acceptance Criteria

The full prototype navigation test would have been successful if the rover was able to avoid obstacles 4 inches and greater, traverse a 4-degree slope, and traverse a 2-inch obstacle. The rover was supposed to achieve 3/3 waypoints within 10% of the distance travelled on at least 4/5 attempts.

Results

As discussed in section C, the motor failures prevented this test from occurring. Therefore, there are no new results to report.

Evaluation

Due to the significant system failure, the requirement of reaching 3/3 waypoints within 10% of the distance traveled on at least 4/5 attempts was not met.

V. Conclusions

The rover met all of the design constraints for this project and met 5 out of the 11 project requirements. It was able to receive a waypoint, plan a path, and execute an attempt to reach the desired waypoint. The rover was able to detect obstacles and avoid collisions, but the biggest challenge was navigating around the obstacles. Due to the limitations of the structural design of the frame and motors, skid steering was not possible as it required too much torque. While performing tests to optimize the path planner for the rover, the motors overheated and succumbed to thermal failure due to the long duration of the test. This resulted in the full test of the prototype not being able to be performed. The final design of the rover can be found in Fig. 21.

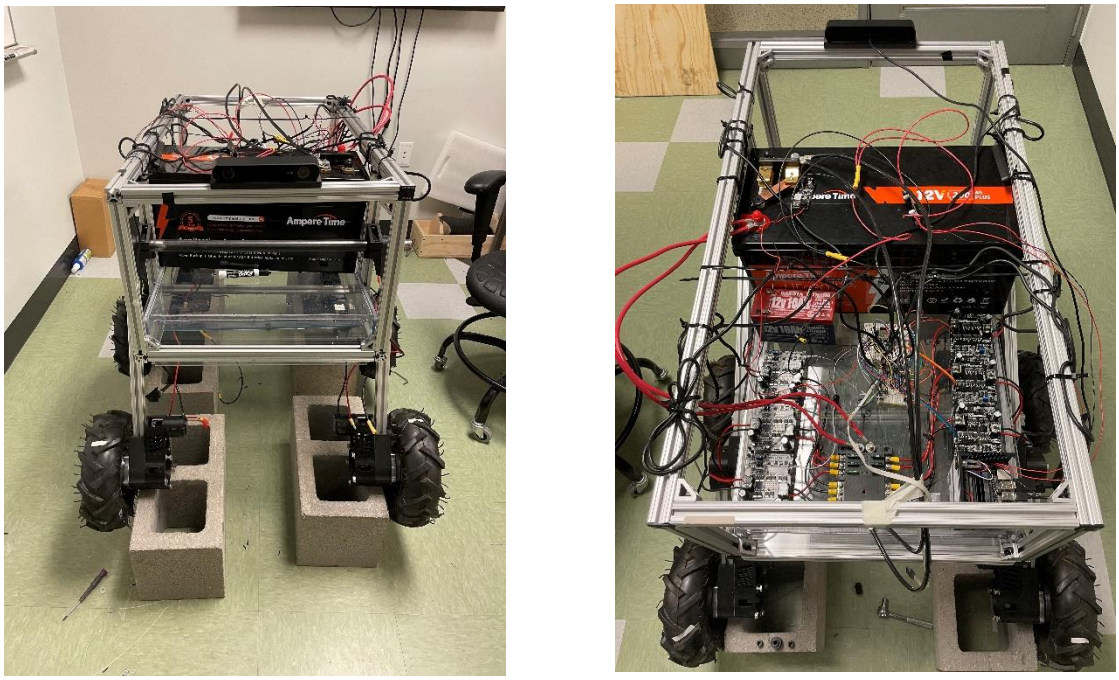


Figure 21. Front and top view of final rover design

There are many changes that need to happen for the rover to become a fully functioning autonomous planetary rover. The first major change that needs to be addressed is the frame. The frame needs to be refactored with all metal parts to prevent bowing of the legs without external support. The overall geometry of the frame also needs to be evaluated and optimized for skid steering as a long rectangular frame requires more torque than a square frame to steer properly. The wheels and tires can be better optimized to allow for more efficient movement of the rover, as the resulting friction applied to the wheels has a significant effect on skid steering performance. Additionally, the radius of the wheels should be evaluated as it directly relates to the amount of force the rover can produce for a given torque from the motors. The wheel tread should also be evaluated and optimized, since the wheel tread can be optimized for the surface the rover travels on, leading to more efficient movement.

The team also recommends switching to a 24 V brushless motor system. While these motors are significantly more expensive than the motors used in this design project, the new motors would be much more energy efficient and produce less heat. Since the motors from this year failed due to overheating, we believe these motors would be able to perform more effectively for this application. These motors also have a gearbox, feedback system, and motor

controller built in. A more detailed suggestion for the future design team can be found in Appendix C.

With a limited amount of time and people on the team, it was extremely difficult to make any major modifications on the rover's frame. The team decided to dedicate the majority of its time to designing and implementing an odometry feedback system, the power and control circuity, finding the correct motor drivers, designing, and implementing a control interface for the motors, and implementing an entire navigation and obstacle detection stack using ROS. Overall, the team believes that the software is ready to perform all of the project requirements, but the rover's hardware became the limiting factor in achieving all of the requirements.

VI. Appendices

Appendix A

A.1 Operation Instructions

Software – MCU

Visual Studio Code has an ESP-IDF plugin that can be used to modify, build, flash, and monitor projects on the ESP32. A video showing how to set up the IDF in VSCode can be found in [19].

To make any modifications to the MCU software are needed, disconnect the USB cable attaching the Jetson to the MCU from the Jetson, connect it to external computer, and use the VSCode plugin to flash the modified software onto the MCU. It is critical to use the correct version of the ESP-IDF, as there are several dependencies used in the current MCU software that are not backward compatible. The code was developed using the current Master branch of the ESP-IDF, so make sure to select this option when installing the ESP-IDF.

Software – Jetson

Setup

To launch the rover's navigation software, an external computer running Ubuntu (preferable Ubuntu 18.04, to avoid compatibility issues with the Jetson) with ROS installed is required. For information on installing ROS, please refer to the ROS documentation [20]. After completing the ROS installation, it is important to set the environment variable ROS_MASTER_URI to the IP address of the Jetson (131.194.115.203) with port 11311. I recommend exporting this environment variable in the computer's .bashrc file so it doesn't have to be reset every time a new shell is opened.

Operation

To launch the navigation stack, first connect to the Jetson over secure shell (ssh) with user "nano" and password "rover2020". Upon successful connection, first launch the ROS master node using by running the terminal command "roscore". Then, open an additional ssh terminal on the nano, source the rover_ws setup file ("/home/nano/rover_ws/devel/setup.bash), and launch the rover_base package with the "base.launch" file. For more information on using ROS launch files, refer to the ROS documentation [CITE]. Next, open an additional Jetson shell using ssh, and launch the rover_navigation package using the "base_navigation.launch" file.

After launching the nodes, open RViz to view the navigation of the rover. Add any information you would like to visualize (map, costmap, depth data) using the corresponding data types in RViz [21]. Once the RViz window displays all the information you would like to visualize, you can supply waypoints to follow_waypoints using the "2D Pose Estimate Tool" in RViz. Once all the desired waypoints have been placed, open a terminal and enter "rostopic pub /path_ready std_msgs/Empty -1". This command triggers follow_waypoints to begin issuing waypoitns to move_base. At this point, the rover should begin moving, and the movement should be visualized in RViz.

A.2 Safety Instructions

The operating current draw from the 200 Ah battery can exceed 100 A. Practice excessive caution with handling any of the components attached to this battery. Always ensure that the knife switch is opened before attempting to touch the battery, motor drivers, or motors. Additionally, the fuse box terminals can reach temperatures exceeding 100°C after extended operation. Wait for several minutes after operation before touching the fuse box and check the temperature with an IR thermometer or FLIR camera if available.

Appendix B

B.1 CAD Models

Figure B.1.1 Wheel assembly inherited from summer research team with arrow indicating wheel spacer

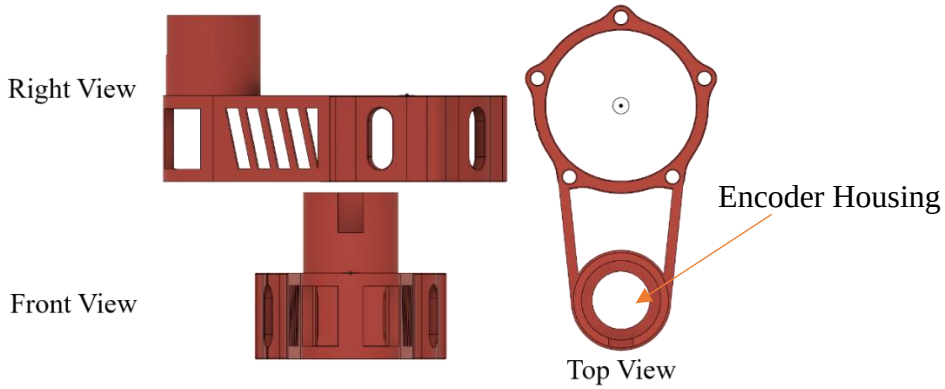


Figure B.1.2 Wheel to gearbox mount with integrated encoder housing

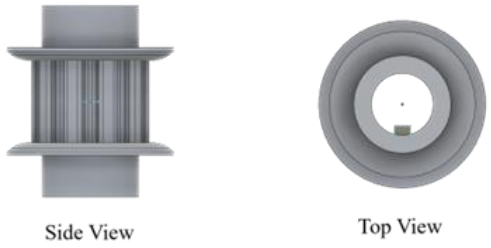


Figure B.1.3 Motor to gearbox shaft coupling with integrated timing pulley

Figure B.1.4 Encoder shaft timing pulley

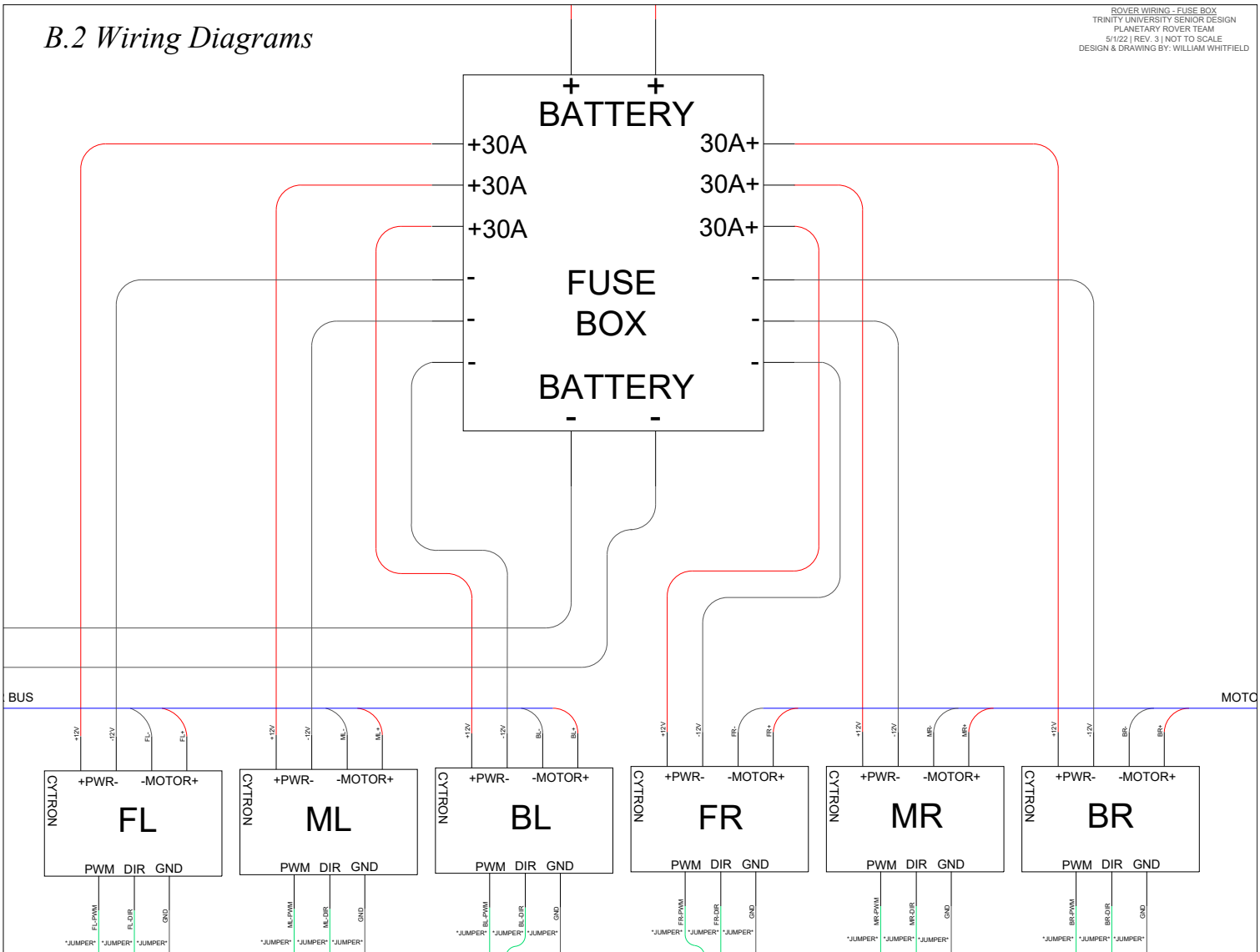


Figure B.2.1 Fuse box wiring

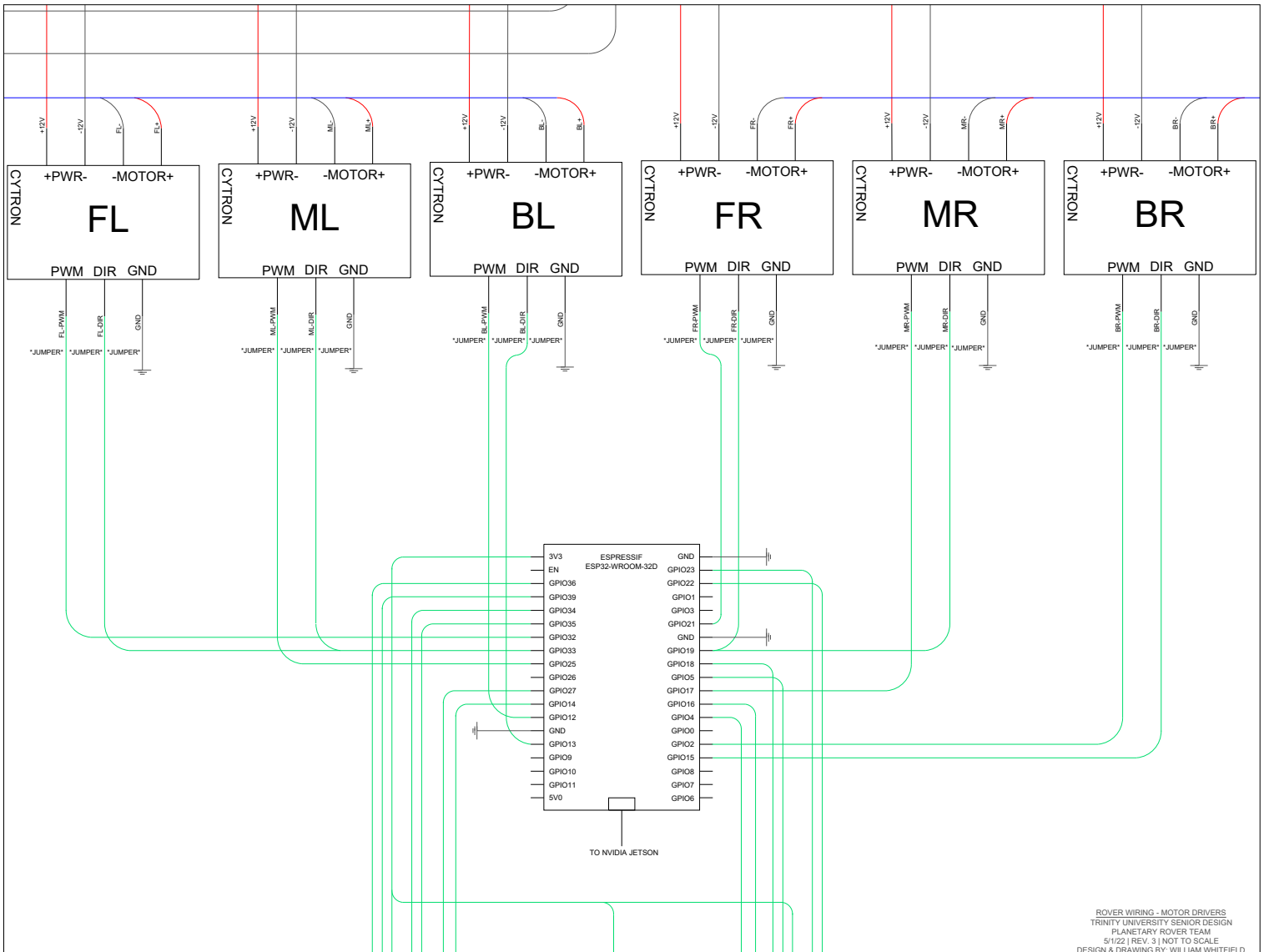
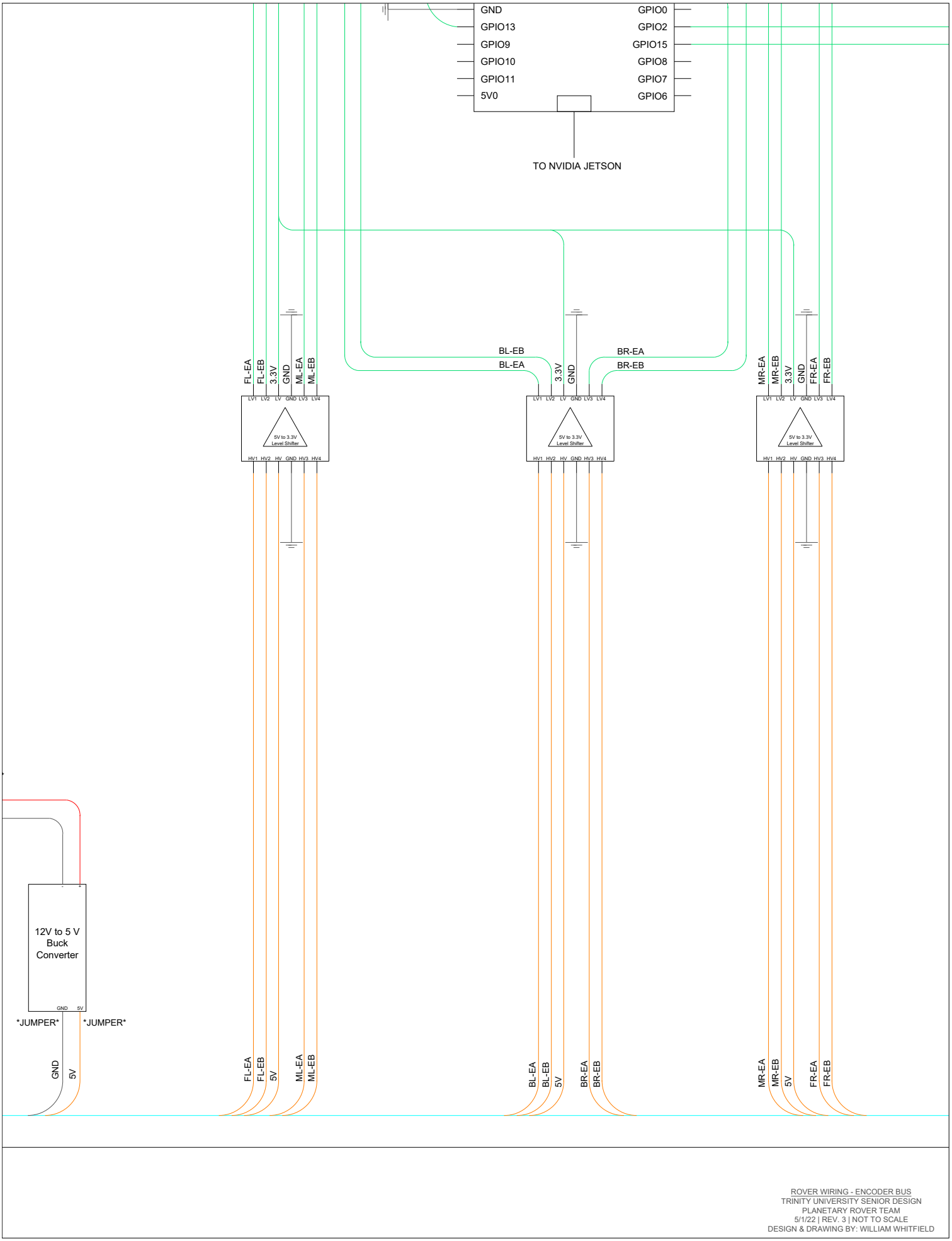
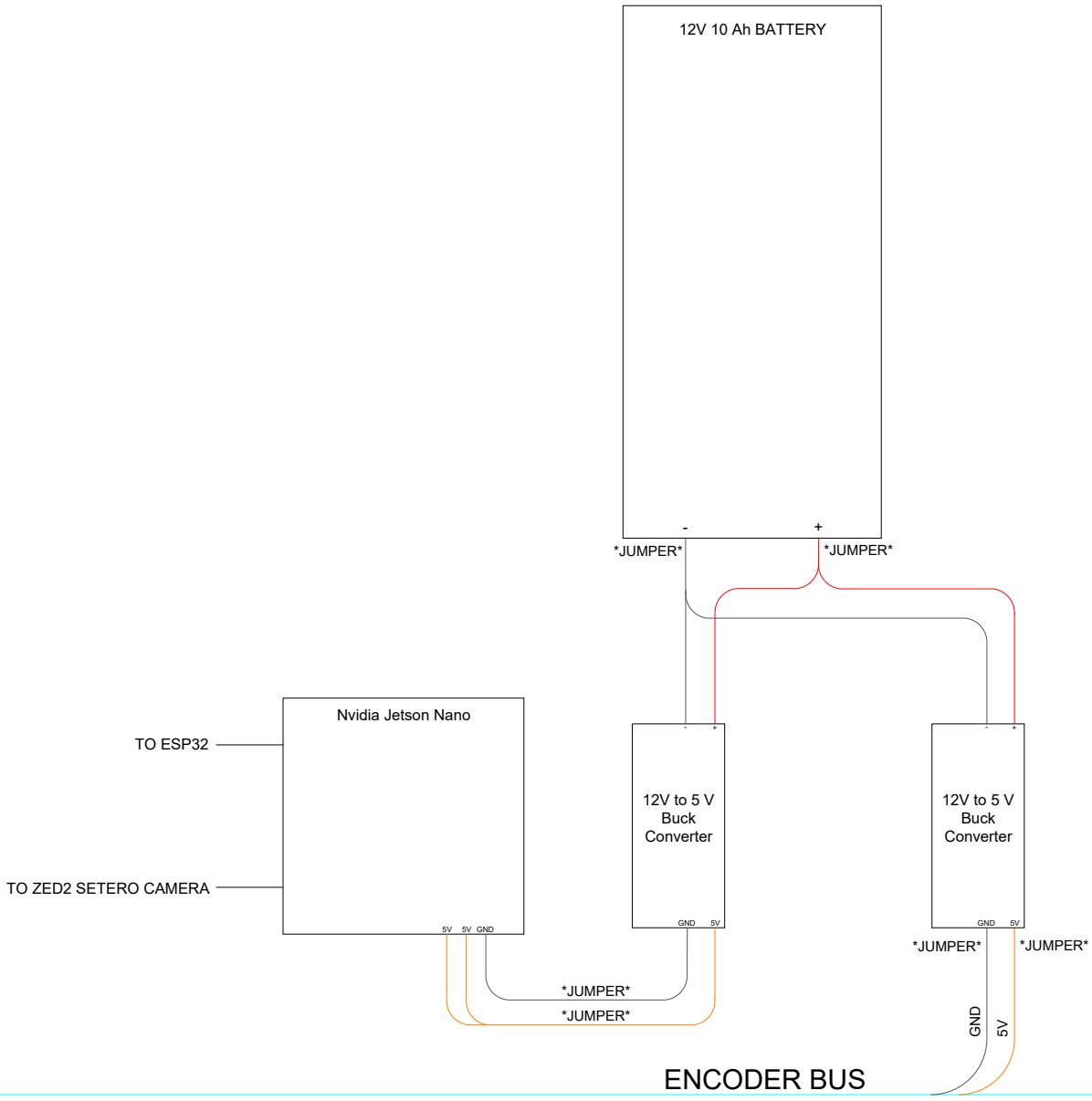


Figure B.2.2 Motor driver wiring



ROVER WIRING - ENCODER BUS
TRINITY UNIVERSITY SENIOR DESIGN
PLANETARY ROVER TEAM
5/1/22 | REV. 3 | NOT TO SCALE
DESIGN & DRAWING BY: WILLIAM WHITFIELD

Figure B.2.3 Encoder bus wiring



ROVER WIRING - NVIDIA JETSON
TRINITY UNIVERSITY SENIOR DESIGN
PLANETARY ROVER TEAM
5/1/22 | REV. 3 | NOT TO SCALE
DESIGN & DRAWING BY: WILLIAM WHITFIELD

Figure B.2.4 Nvidia Jetson Wiring

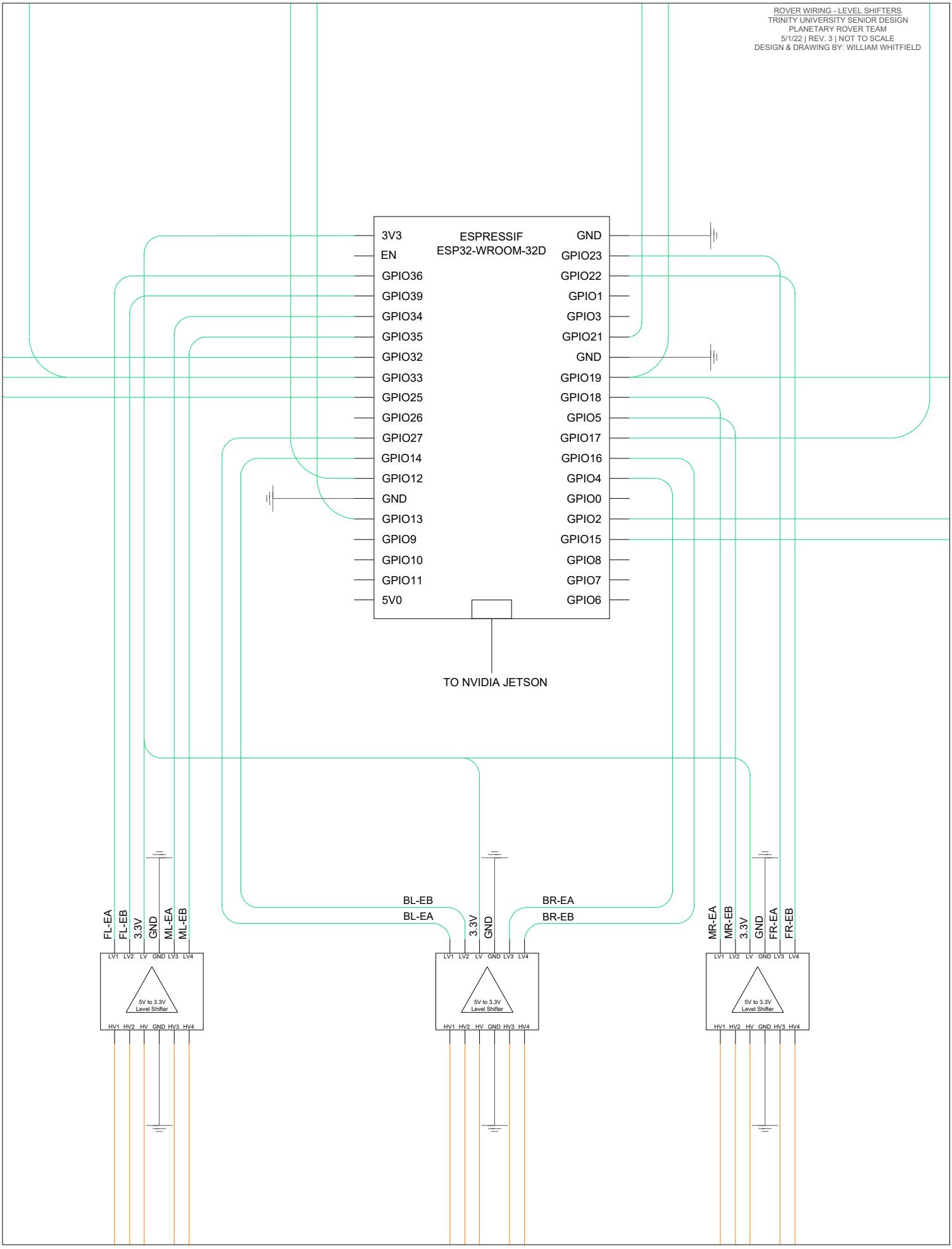


Figure B.2.4 Level shifter wiring

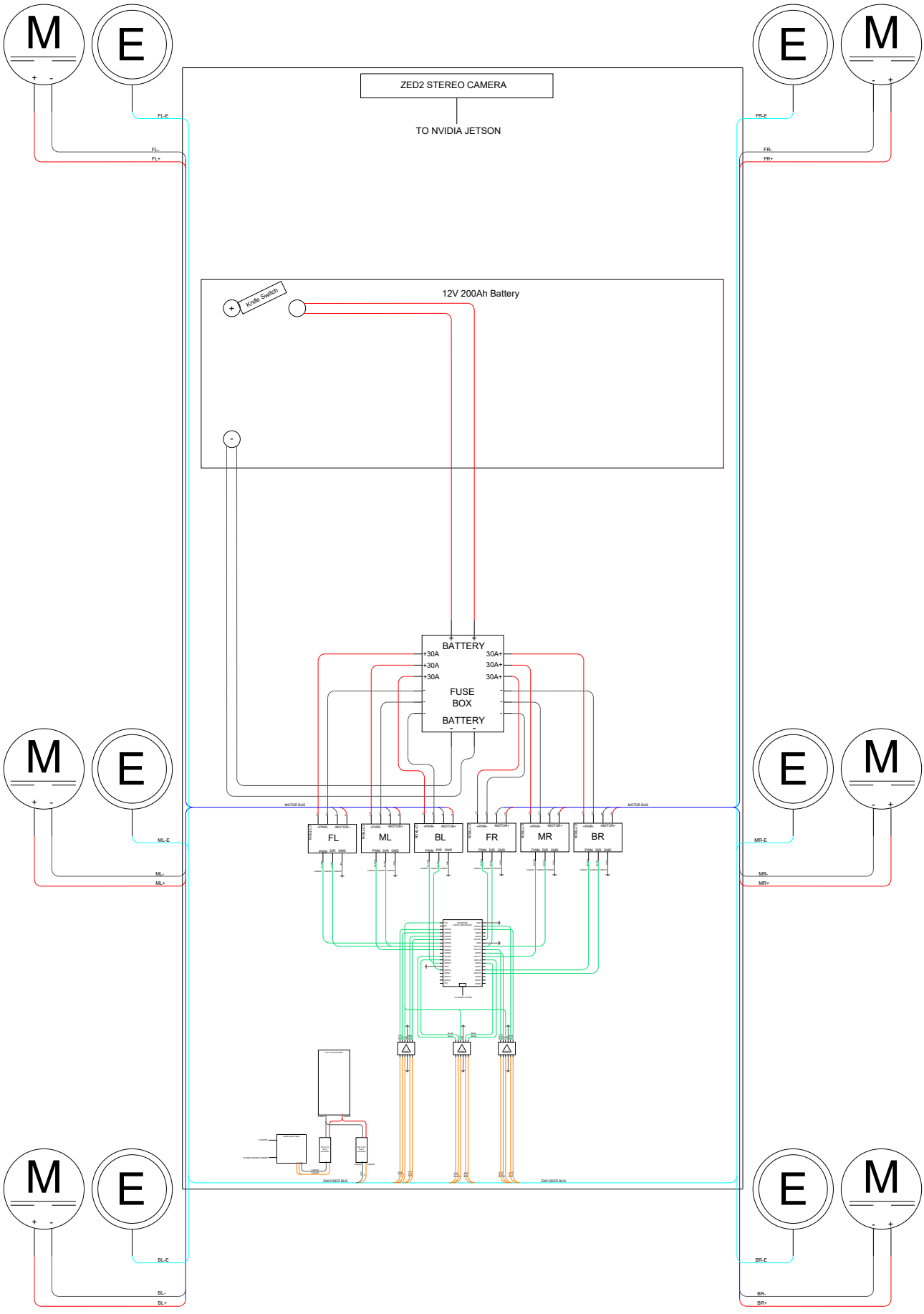


Figure B.2.6 Overview of rover wiring

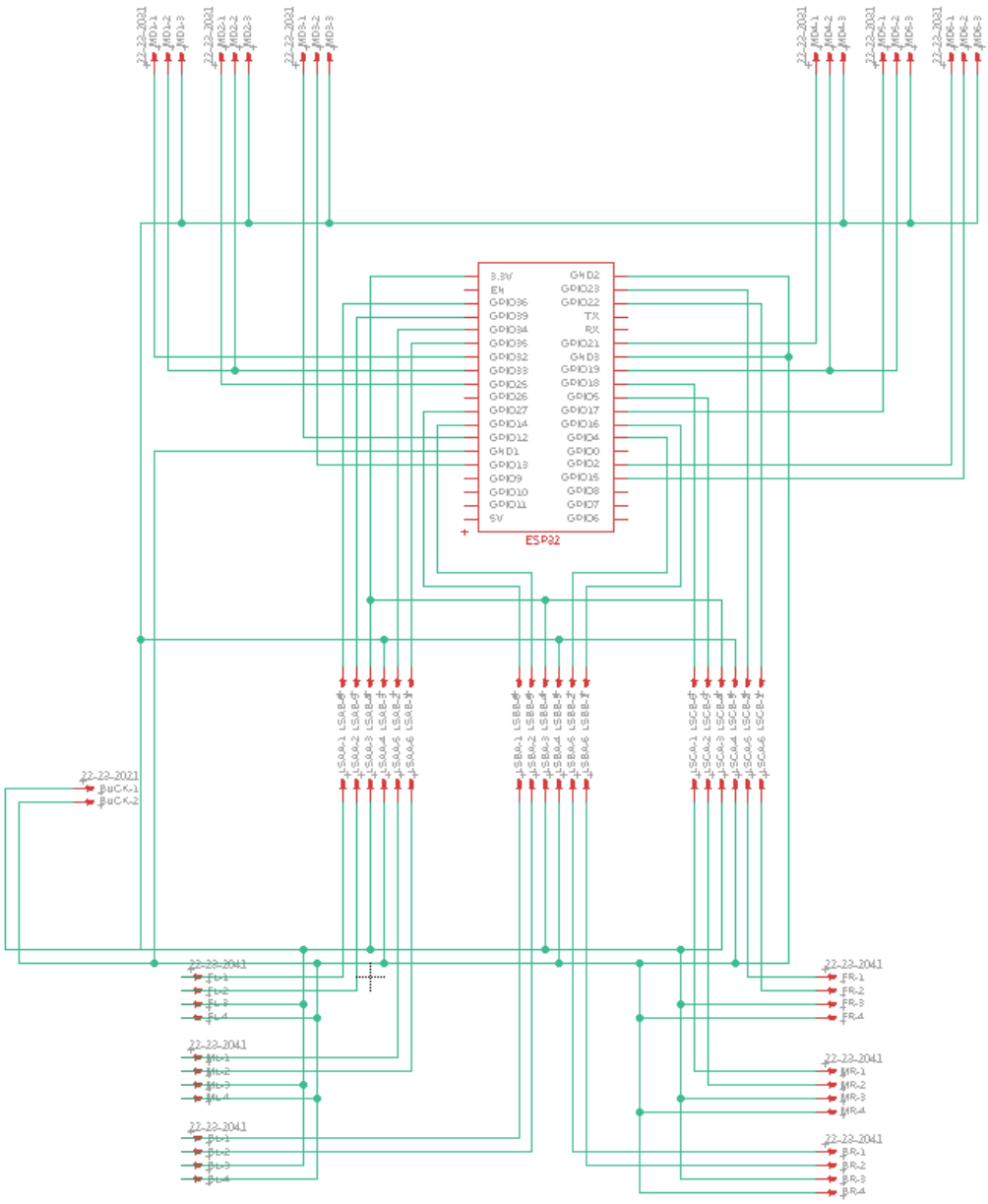


Figure B.2.7 EAGLE Schematic

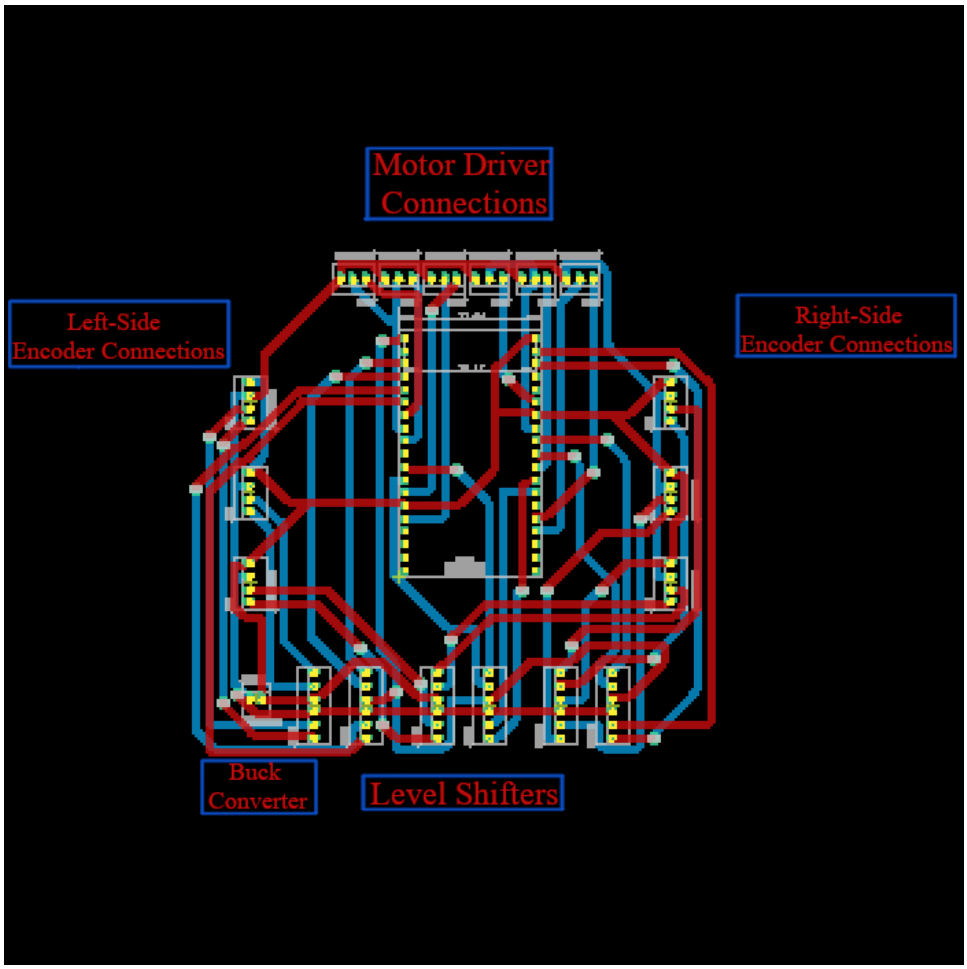


Figure B.2.8 EAGLE Board Representation

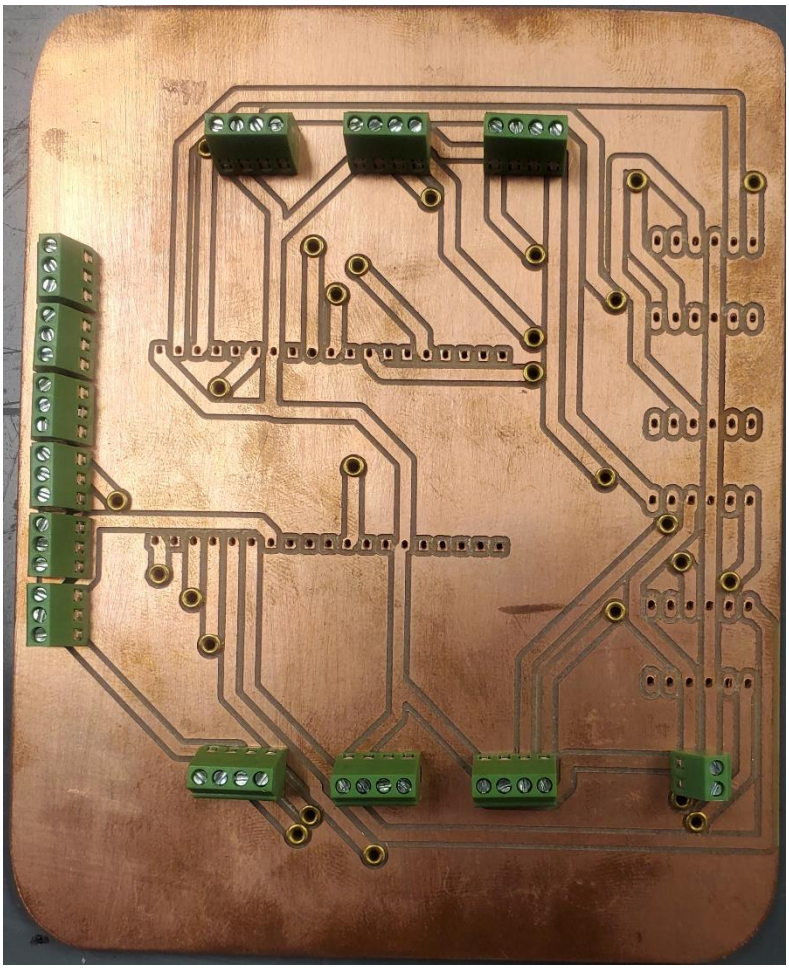


Figure B.2.9 Printed Circuit Board

Appendix C

C.1 Recommendations for Future Design Team

The rover frame needs to be completely refactored with metal only parts and better craftsmanship. The ability to skid steer will also need to be a consideration when reworking the frame. Additionally, we recommend researching the optimal frame and wheel spacing conducive for skid steering as the narrow build of this team's frame and wheels contributed to the difficulties of developing skid steering.

To limit the amount of heat the motors have to endure, we recommend switching to the 42D Brushless DC Planetary Gear Motor - 24V 116RPM made by E-S Motor and sold by Robot Shop (Product Code: RM-ESMO-0ET, Supplier Product Code: 42PG-4260BL-67 24V). The recommended motor operates with twice as much torque as the old ones, with a rated speed of 65

RPM. The brushless motors are also much more efficient, with each motor using 20 A less than the brushed motors used this year. In order to use the same 12 V 200 Ah battery from this year, boost converters can be used to convert 12 V to 24 V and provide enough voltage for the 24 V motors. The suggested motors have an integrated gearbox and encoder, allowing for easy integration into the new frame. Finally, these motors will not require motor drivers, which will simplify the electronics on the rover.

For the tires, we would recommend finalizing design expectations as the tread of the tires will depend greatly on the surface that the rover is operating on. The radius of the wheel could also be evaluated, since a smaller wheel will require less torque to generate the same amount of force.

Finally, once the testing phase comes closer, we recommend trying to reserve a good spot with a good Wi-Fi signal to conduct controlled tests. When we were testing outside of CSI, we kept losing Wi-Fi signal (causing the rover to stop working) and people were getting in the way of the camera, resulting in the rover stopping due to it detecting the people as obstacles.

Appendix D

D.1 Additional Subsystem Test Results

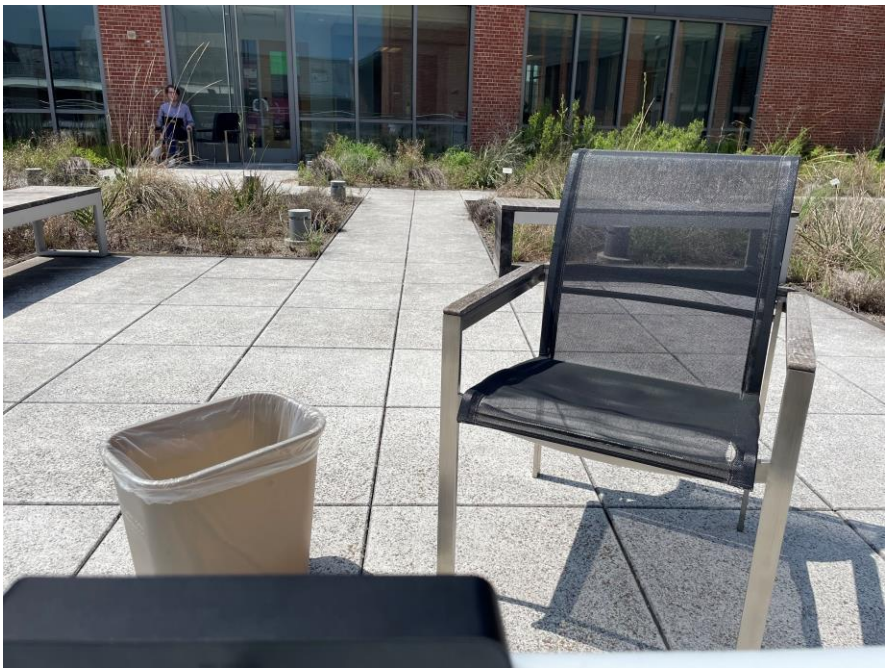


Figure D-1. Image of obstacle field for obstacle detection test trial 2

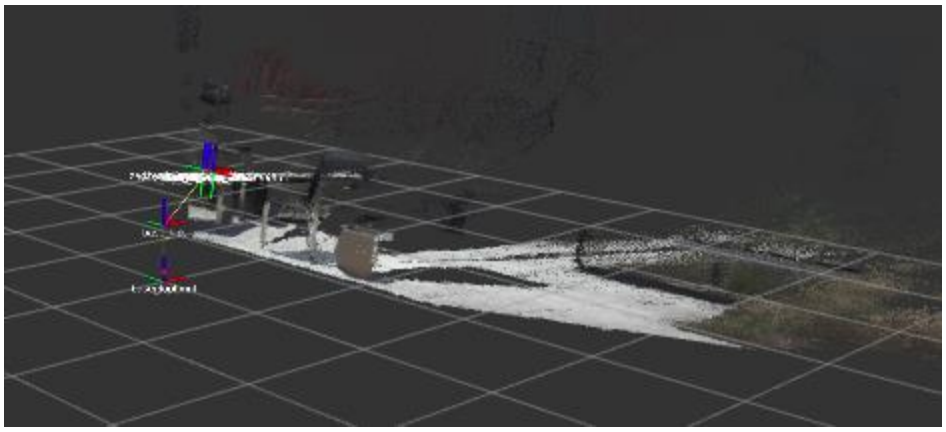


Figure D-2. Image of generated depth data for obstacle detection test trial 2

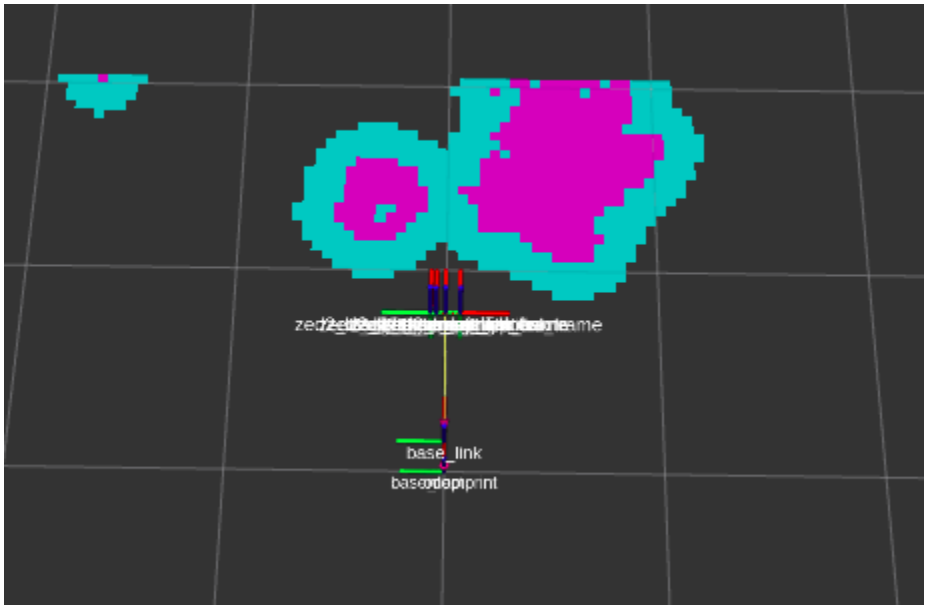


Figure D-3. Image of generated costmap for obstacle detection test trial 2

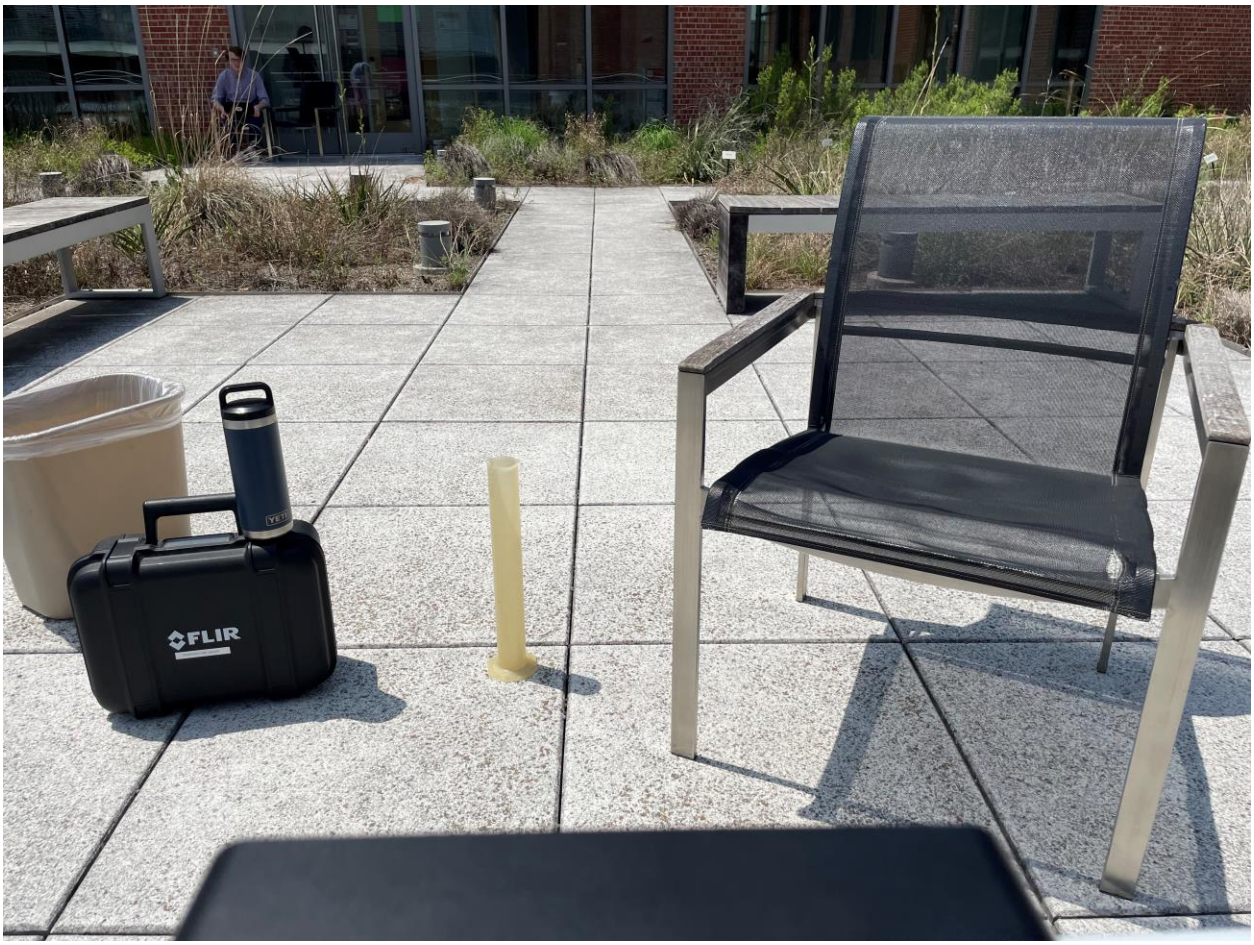


Figure D-4. Image of obstacle field for obstacle detection test trial 3

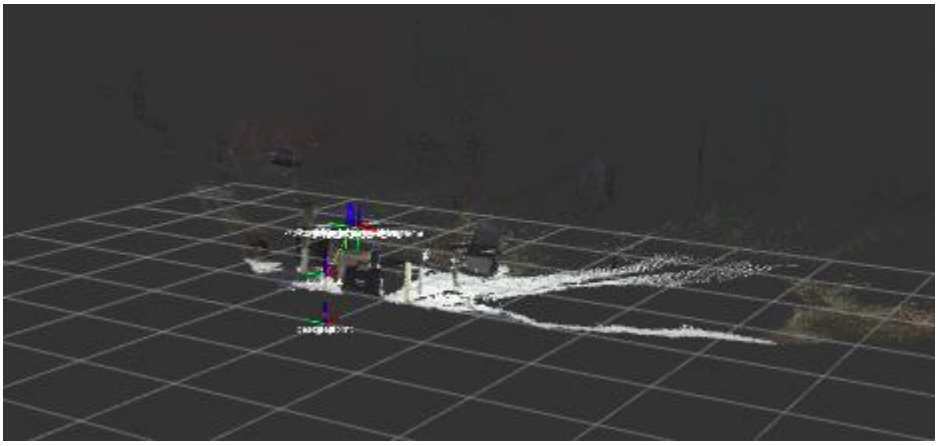


Figure D-5. Image of generated depth data for obstacle detection test trial 3

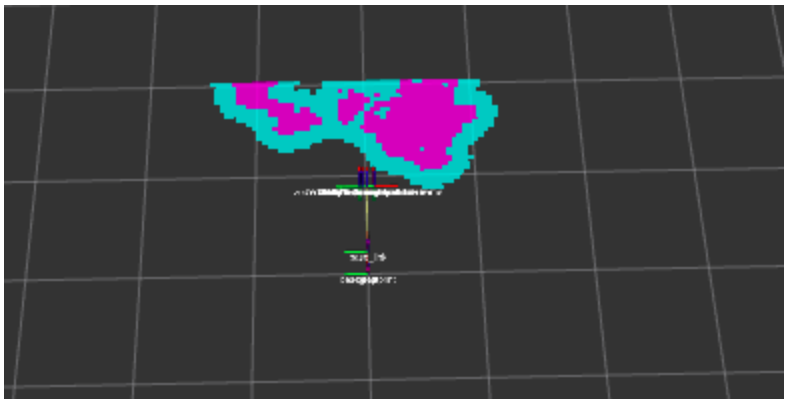


Figure D-6. Image of generated costmap for obstacle detection test trial 3

VII. Bibliography (IEEE)

[1] "Jetson Nano," *NVIDIA Developer, 29-Mar-2021.* [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano.

[2] "Loviver IBT-4 MOSFET high current H bridge driver, motor ... - amazon.com." [Online]. Available: https://www.amazon.com/LOVIVER-Mosfet-Current-Bridge-Driver%EF%BC%8CMotor/dp/B07MKQLQ22.

[3] "Home," *63zyt01c High Torque High Speed 10000rpm 12v 24v Dc Motor 200w - Buy Hight Torque High Speed Dc Motor,12v 200w Dc Motor,10000rpm Dc Motor Product on Alibaba.com*. [Online]. Available: https://smartmotor.en.alibaba.com/product/594978218-213798039/63ZYT01C_high_torque_high_speed_10000rpm_12V_24V_dc_Motor_200w.html.

[4] "Shaft rotary encoder ghs38 series," *CALT Sensor*, 26-Sep-2021. [Online]. Available: https://caltsensor.com/product/shaft-rotary-encoder-ghs38-series/.

[5] *Lithium Iron Phosphate Battery (LiFePO4) 12V 200Ah (200A BMS) Product Manual.* Shenzen Ampere Time Technology Co., Ltd.

[6] "30Amp 5V-30V DC Motor driver," *Cytron Technologies*. [Online]. Available: https://www.cytron.io/p-30amp-5v-30v-dc-motor-driver.

[7] A. Jay, “Dakota Lithium 12V 10ah battery - half the weight & twice the power,” *Dakota Lithium Batteries*. [Online]. Available: https://dakotalithium.com/product/dakota-lithium-12v-10ah-battery/.

[8] “Amazon.com: Buck Converter 12V to 5V, DROK 5a USB voltage regulator DC ...” [Online]. Available: https://www.amazon.com/Converter-DROK-Regulator-Inverter-Transformer/dp/B01NALDSJ0.

[9] “ESP32-devkitc V4 getting started guide,” *ESP*. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html.

[10] *ZED 2 Camera and SDK Overview*. Stereolabs, 2021.

[11] “Drop-in M4 and M5 T-slot nuts for 80/20 aluminum extrusion,” STLFinder. [Online]. Available: https://www.stlfinder.com/model/drop-in-m4-and-m5-t-slot-nuts-for-80-20-aluminum-extrusion-llIYCjuW/1586593/.

[12] E. Marder-Eppstein, “move_base” ros.org. [Online]. Available: http://wiki.ros.org/move_base.

[13] “Clearing the obstacles form local costmap edit,” Clearing the obstacles form local costmap - ROS Answers: Open Source Q&A Forum. [Online]. Available: https://answers.ros.org/question/352299/clearing-the-obstacles-form-local-costmap/.

[14] “Costmap2D” *ros.org*. [Online]. Available: http://wiki.ros.org/costmap_2d.

[15] C. Roesmann, W. Feiten, T. Woesch, F. Hoffmann and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," *ROBOTIK 2012; 7th German Conference on Robotics*, 2012, pp. 1-6.

[16] “Rear View Obstacle Detection,” *YouTube*, 05-May-2022. [Online]. Available: https://youtube.com/shorts/wnEbELEjU5k.

[17] “Close pass,” *YouTube*, 05-May-2022. [Online]. Available: https://youtube.com/shorts/b0mAng4yUbU.

[18] “Wall detection,” *YouTube*, 05-May-2022. [Online]. Available: https://youtube.com/shorts/vs1SdI8x7E8.

[19] “Getting started with VS code IDE,” *ESP*. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/vscode-setup.html. [Accessed: 06-May-2022].

[20] “ROS Installation Guide,” *ros.org*. [Online]. Available: http://wiki.ros.org/ROS/Installation.

[21] “Rviz User Guide,” *ros.org*. [Online]. Available: http://wiki.ros.org/rviz/UserGuide.