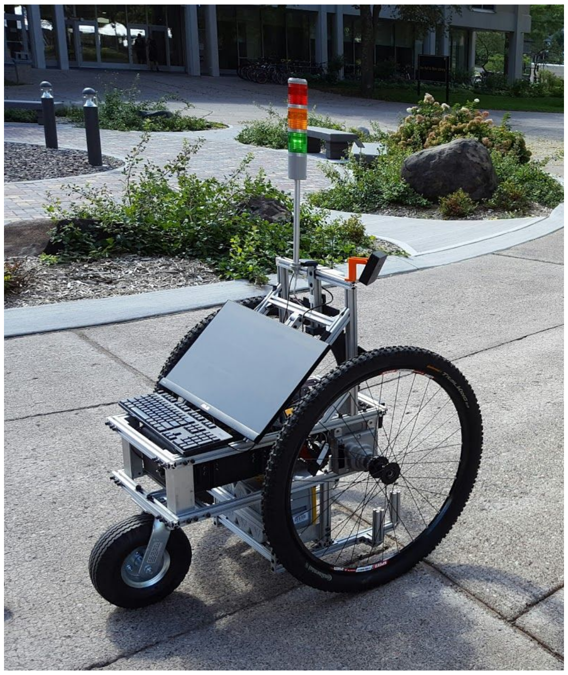
Blue Marble Security Enterprise: Autobot Team

Charlie



April 21, 2017

Team Captain:	Haden Wasserbaech	hwasserb@mtu.edu
Other Members:	Trevor Peffley	tbpeffle@mtu.edu
	Alex McInerney	armciner@mtu.edu
	Christian Baker	cjbaker1@mtu.edu
	Collin Staudacher	cestauda@mtu.edu
	Brian Terry	bmterry@mtu.edu
	Nathan Duprey	naduprey@mtu.edu

I hereby certify, as the faculty advisor, that the design and engineering of the vehicle outlines in this report to be entered in the 2017 Intelligent Ground Vehicle Competition has been significant and equivalent to what might be awarded credit in a senior design course.

[illegible]

Dr. Glen Archer, Faculty Advisor

Table of Contents

Conduct of Design Process, Team Identification and Team Organization **4**
Introduction 4
Organization 4
Design Assumptions and Design Process 4
Innovations **5**
ROS 6
Debug Screen 6
Mechanical Design **6**
Overview 6
Structure and housing 7
Suspension 8
Weatherproofing 8
Electronic and Power Design **8**
Overview 8
Power Distribution 8
Electronics Suite 9
Nvidia Jetson TX1 9
ZED Stereo Camera 9
EVK-M8N 10
Razor 9DOF IMU 10
Custom Interfacer 10
Victor SP Motor Controller 11
Motors 11
Encoder Interface 11
Safety Devices 12
Blinking Light 12
Wireless Emergency Stop 12
Mechanical Emergency Stop 12
Software Strategy and Mapping techniques **12**
Overview 12
GitHub 12
Obstacle Detection 13
Map Generation 14
Line Tracking 14
Occupancy Grid 15

Goal Selection and Path Generation 15

Gazebo 15

Failure Modes, Failure Points and Resolutions **16**

Software Failures and Resolutions 16

Mechanical and Electrical Failures and Resolutions 16

Failure Prevention Strategy 16

Performance Testing To Date **17**

Initial Performance assessment **17**

Conduct of Design Process, Team Identification and Team Organization

Introduction

Michigan Tech's entry into the Intelligent Ground Vehicle Competition is designed within the Blue Marble Security Enterprise as part of the Engineering Enterprise program. The design of this vehicle was started in January 2015, meant to replace the vehicle previously used in the competition. The team is composed of undergraduate students ranging from sophomore to senior status within the Electrical and Computer Engineering departments.

Organization

The team is organized on the basis of administrative responsibilities and technical responsibilities. The three main administrative roles are Project Manager, Documentation Chief, and Financial Manager. The Project Manager is tasked with overseeing team organization, planning, and ensuring that the team stays on the critical path. The Documentation Chief ensures that the entire design process is properly documented, manages repositories, and ensures that all paperwork required from the department are completed and filed properly. The financial manager ensures that our budget is properly balanced and seeks out additional funding if necessary.

Technical responsibilities are given based on class standing, background knowledge, project familiarity and interests. Generally, the more senior members are paired with the junior members to ensure that knowledge is passed down appropriately.

The use of a Gantt Chart was utilized to be able to keep track of tasks that each member is responsible for throughout the course of the semester. This included set dates that each task should start and be done by.

Table 1.1: Membership Information and Roles

Last Name	***First Name***	***Email***	***Administrative Role***
Wasserbaech	Haden	hwasserb@mtu.edu	Project Manager
Peffley	Trevor	tbpeffle@mtu.edu	Financial Manager
Duprey	Nathan	naduprey@mtu.edu	Documentation Chief
McInerney	Alex	armciner@mtu.edu	N/A
Baker	Chris	cjbaker1@mtu.edu	N/A
Staudacher	Collin	cestauda@mtu.edu	N/A
Terry	Brian	bmterry@mtu.edu	N/A

Design Assumptions and Design Process

Project requirements were initially derived based on the IGVC 2015 Rulebook. Additional requirements were gathered based on what was learned from our previous design, physical constraints imposed on us by our laboratory setting, and monetary constraints. The design history of this vehicle is outlined in the chart below.

Table 1.2: Vehicle Development History

Spring 2015	Initial Requirements Capture
	Part Selection
	Chassis Design
	Electronic Design
	Mechanical and Electrical Validation
Fall 2015	Wireless Controller Redesign and Validation
	Sensor Node Development
	Mapping Development
	Pathfinding/Obstacle Avoidance Development
Spring 2016	Pathfinding Validation
	Image Processing Development
	Goal Selection Development and Validation
	Control Algorithm Development
Summer 2016	Vehicles First Competitive Attempt
	Won third place in Design competition
Fall 2016	Switched to ROS for robot framework
	Replaced multiple sensors with ZED camera
	Switched to Jetson TX1 as robot computer
	Minor mechanical hardware modifications
Spring 2017	Replaced broken wheel encoder
	Fixed voltage ringing issue
	Created framwork and basic structure for image detection and map generation
	Finished wheel odometry
	Completed path generation
	Added support for changing line detection on the fly

Innovations

This vehicle's design addresses several of the issues seen with our previous design. The previous design had significant mechanical modifications from its conception to its last competition. Because of this, many of the electronics were not easily accessible, making maintenance difficult. To prevent this from happening, the chassis was designed with 80/20 extruded aluminum, making any future chassis modifications significantly easier to perform. This also makes transportation of the robot much simpler. Most of the electronics are located inside of a weather resistant pelican case shown below. This case is only partially mounted to the chassis and can be easily removed for hardware changes, bench testing, or to make chassis modifications much simpler.

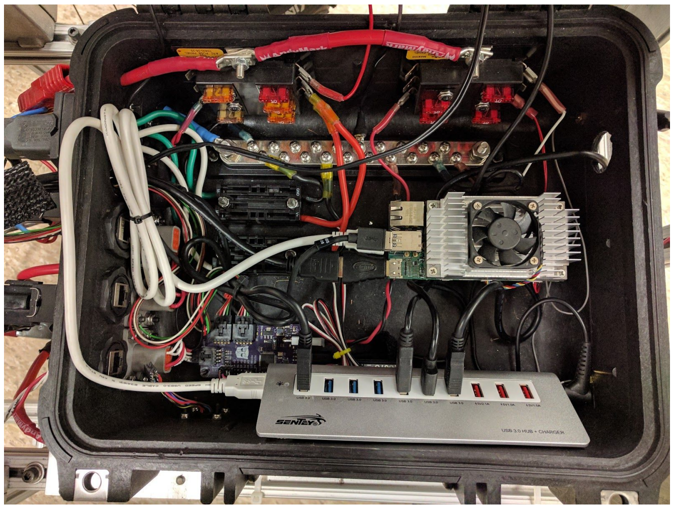


Figure 1.1: Electronics Housed in Pelican Case

ROS

We chose to use ROS (Robot Operating System) as the basis of our software system. ROS runs on top of Linux and is used for handling all back-end communication between nodes, providing data logging capabilities, sensor data replay, and plug-and-play capabilities with various popular sensors. Because ROS is the industry standard for mobile robotics and has extensive community support, there are many built in libraries in addition to freely available ones developed by the community. These libraries are able to perform useful tasks such as navigation, obstacle detection, mapping, localization, and much more. Due to these attractive features we determined that ROS is the best platform available for the development of a robot such as ours. While in years past ROS was not as developed or widely used, it has become goto standard for many mobile robotics applications and is now used in many self-driving cars, nearly all academic robotics, and is even being used for an increasing amount of factory automation. Due to the wide adoption of ROS it has been significantly easier to find documentation than some of our previously used platforms.

Debug Screen

We have greatly utilized our nicely placed debug screen directly on our robot for code edits, to view the current generated map, and to see live sensor readings. This will be handy at competition to be able to make any on the fly code edits and changes before the robot competes.

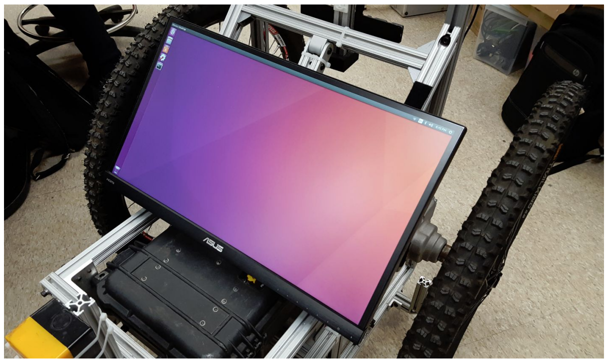


Figure 1.2 Robot Debug Screen

Mechanical Design

Overview

The chassis was designed to comply with IGVC Competition rules as well as our physical constraints in our laboratory. These requirements, as gathered from the IGVC Rules, are outlined in the table below.

Table 2.1: Mechanical Requirements

Requirement	Details
Vehicle Type	Ground Vehicle; Must have direct contact to the ground
Length	Minimum: 3ft Maximum: 7ft
Width	Minimum: 2ft Maximum 2ft 8in (Doorway Width in Laboratory)
Height	Maximum: 6ft

Structure and housing

The structure of the vehicle (Figure 3.1) is composed of 80/20 and aluminum sheeting when appropriate. This material was chosen based on its light-weight properties and modularity. A three-wheeled design was chosen with two 29” bicycle wheels in front and an unpowered caster wheel in the rear. This is based on the previous design, which had little issues with mobility, and was chosen for this vehicle because of its mechanical simplicity. One small issue included when the wheel got stuck in the wrong direction when trying to go up a hill or when a 180 degree turn was made. To help solve this and decrease the amount of friction from the large back caster wheel, tape was wrapped around the entire wheel which has proved to be effective with allowing for better movement of the robot. All of the electronic components, minus the wires to and from the motors and external components, are housed in a water resistant pelican box.

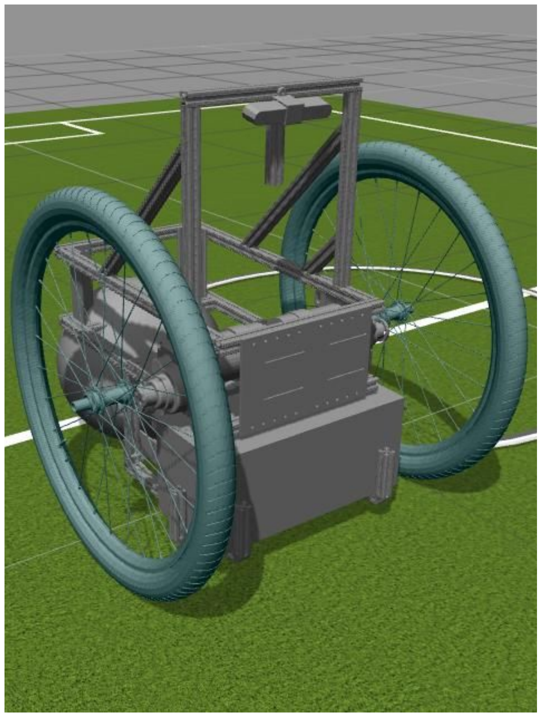


Figure 3.1: Vehicle Structure Model

Suspension

Based on our previous robot designs we determined that suspension is not needed.

Weatherproofing

A majority of electronics are placed in a weather resistant pelican case. Some smaller electronics are currently outside of the main case such as the ZED camera, GPS antenna, E-Stop, Monitor, and wheel encoders. Most of these components are weather resistant and the screen, which is not, can be covered with a 10mm bisquick water deflection system.

Electronic and Power Design

Overview

Our electronics system was designed with the chief criterion being ease of implementation and power efficiency. We ultimately decided to take advantage of the powerful communication backend of ROS and designed the system with a distributive computing model in mind.

Power Distribution

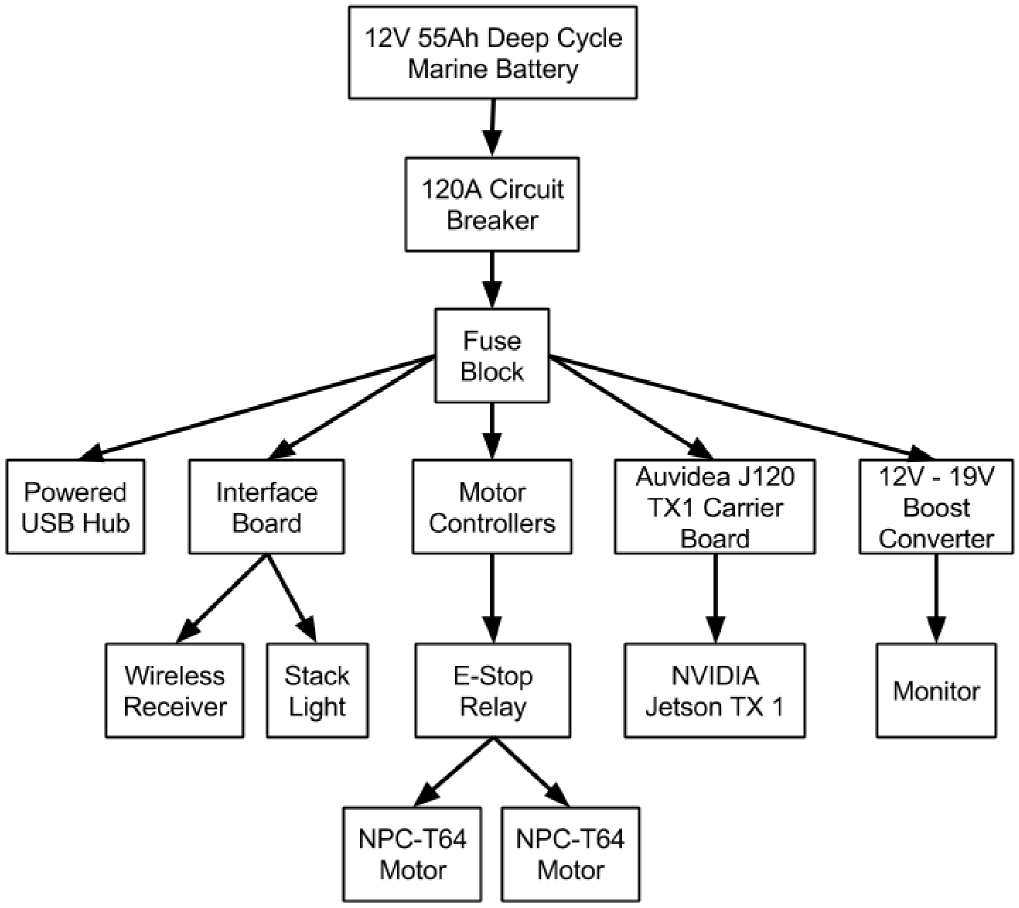
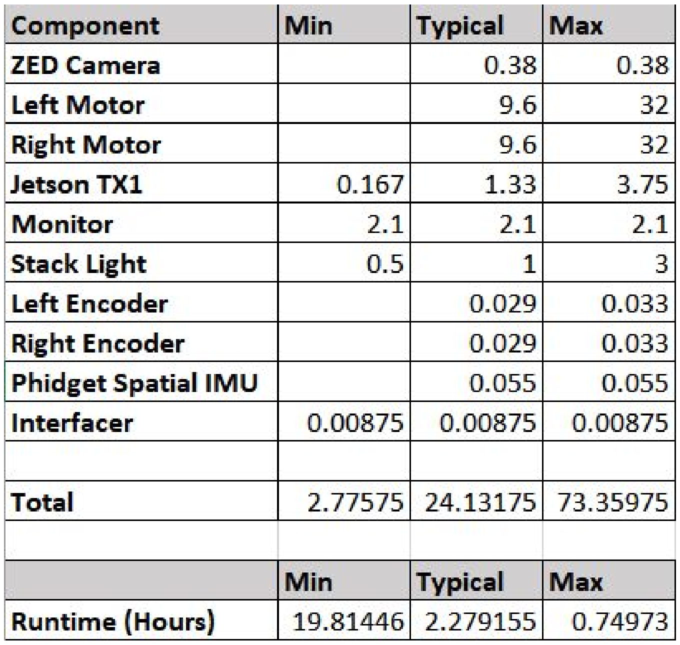


Figure 4.1: Power Distribution

The vehicle is powered by a 12V Deep-Cycle marine battery. With our current setup, we expect a typical run time of just over 2 hours on a constant run. However, empirical testing has shown runtimes closer to 6 hours. If all motors are running at full power, we calculated that the battery could sustain 45 minutes worth of run time. Knowing this and the typical competition usage is only about 10 minutes or less on the course, we know that our battery will last for more than what we will need it to at competition. In case of a unknown power shortage, we do have a backup battery that can be easily swapped in only a couple minutes to be able to get back on the course while the other battery is charging.



Component	Min	Typical	Max
ZED Camera		0.38	0.38
Left Motor		9.6	32
Right Motor		9.6	32
Jetson TX1	0.167	1.33	3.75
Monitor	2.1	2.1	2.1
Stack Light	0.5	1	3
Left Encoder		0.029	0.033
Right Encoder		0.029	0.033
Phidget Spatial IMU		0.055	0.055
Interfacer	0.00875	0.00875	0.00875
Total	2.77575	24.13175	73.35975
	Min	Typical	Max
Runtime (Hours)	19.81446	2.279155	0.74973

Figure 4.2 Estimated Run Time

Electronics Suite

Nvidia Jetson TX1

The Nvidia Jetson TX1 was picked for its powerful and efficient chipset (Tegra X1) and it's robotics friendly platform. It features a quad core ARM Cortex A57 CPU as well as a separate single core low power CPU that is great for reduced power consumption. Additionally it contains a Maxwell architecture GPU with 256 CUDA cores with about 1 TeraFLOPs of compute power. Nvidia also provides a specially optimized OpenCV library specifically for the Jetson TX1 to provide significantly improved image processing performance. Because the Jetson is actually a very small module it can be installed in different carrier boards similar to how a CPU can be installed in different motherboards. The carrier board that comes with the Jetson allows for a large voltage input range of 5.5-19.6VDC, this make is very easy to run from the onboard battery without the use of a dedicated buck boost converter. The many peripheral interfaces such as UART, I2C, CAN bus, SPI, USB, Ethernet, and CSI camera interface. The is very useful in the robotics applications as the interfaces used in sensors vary greatly.

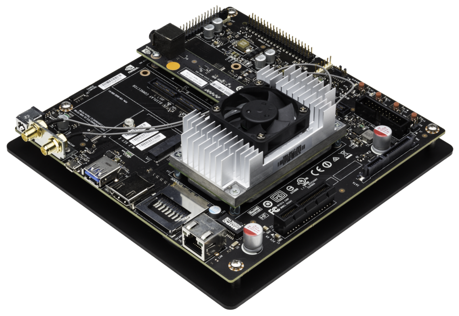


Figure 4.3 Jetson TX1

ZED Stereo Camera

Image acquisition is fulfilled by the ZED 2K Stereo Camera. The ZED camera is a stereo depth sensing camera. It is able to take in RGBD values, so each point has a color and depth value. This can be used to replace the LIDAR as a much cheaper alternative for depth mapping. The ZED camera has a 110° range of vision which is more than twice the range of the Raspberry Pi camera modules that it is replacing from last year. It can calculate a depth ranges for 0.7 - 20m. Since we are only dealing with short ranges as a slow speed, the camera will give us more data at short ranges than a LIDAR could even though the LIDAR gives more further ranged data. The ZED software package also performs some tracking to determine the orientation of the camera. This additional odometry data can then be fused with the IMU, wheel encoders, and GPS to provide a more accurate pose estimation. The ZED camera connects to the Jetson TX1 over USB 3.0 and has dedicated support through NVIDIA.

Figure 4.4 ZED Camera

EVK-M8N

The ublox EVK-M8N is a GPS evaluation kit based on the ublox M8 chipset. This module has multiple interfacing modes including an onboard USB to UART adapter allow easy connection and integration with linux. The unit can be powered straight from the USB port and eliminates the need for an external power supply. The included active antenna provides a stronger, more accurate GPS fix when compared to some passive ceramic antennas that are typically used in many cheaper modules.

Razor 9DOF IMU

The Razor IMU is a low cost 9 DOF IMU that connects over USB. The IMU performs some of the sensor fusion of the 3 axis accelerometer, gyroscope, and magnetometer onboard before sending the pose estimation over USB. The IMU is easily integrated in linux through the provided driver and sample code.

Custom Interfacer

Figure 4.5 New RC Controller

While the custom interface board that was designed last semester worked well enough to drive the robot, there were enough changes and extra features wanted to necessitate a new design. The new design would still need to perform the same main function as the previous design: receiving drive commands from the Jetson. The board is responsible for turning commands from the Jetson into PWM signals for the motor controllers. Additional functions include handling manual control of the robot with a standard RC controller, reading the quadrature encoders on the drive motors, and controlling the status lights on the robot. The new board connects to the Jetson over USB and shows up as a basic serial device in software.

Previously, a PIC32 microcontroller was used as the processor but for the revised design, an ARM Cortex M4 was used. The microcontroller used was one of NXP / Freescale's Kinetis K20 devices, specifically the MK20DX256. This was picked because it is the same device used for the Teensy 3.2, a low-cost, open-source microcontroller development board. The basic Teensy schematic was used as the starting point for the new interface board.

Main power for the board is taken care of by a high-efficiency buck converter to step down the 12 volts from the battery to 5 volts for powering the RC receiver and wheel encoders. The 5 volts from the buck converter is further stepped down to 3.3 volts with a linear regulator to provide stable, clean power for the microcontroller. The 12 volt input is routed directly to the stack light and each light is turned on by a low-side MOSFET switch. The specific buck converter used in the design can handle up to 17 volts input and can supply up to 2 amps on the 5 volt side.

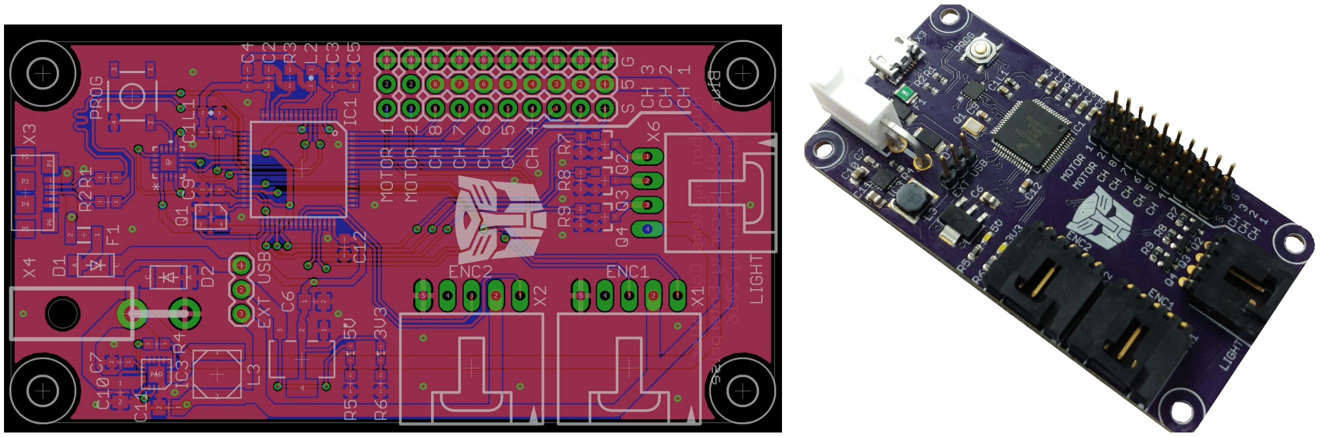


Figure 4.6: Interface board PCB design

The new design has been mostly successful, however there were a couple hardware-level bugs that need to be fixed in the next revision. The biggest problem was a reversed footprint for the micro USB connector. This was able to be worked around by soldering a USB cable directly to the board but a new PCB should be manufactured. The connectors for the encoders also turned out to be reversed, but this was easily worked around by re-arranging the wires in the connectors. All these issues have been fixed so a new board can be ordered and assembled at the start of the next semester.

Victor SP Motor Controller

The motor controllers have a small form factor and have passive cooling while maintaining high performance. It has a wide input voltage (6-16 volts) that is a perfect fit for our 12 volt system. They can handle 60 Amps continuous current draw and are controlled over a PWM interface.

Motors

The NPC-T64 motors were chosen based on the team's prior experience and their built in gearbox. The gearbox has mounting points that allow it to be easily integrated into the mechanical design. The motors are rated for 24 volts but provide ample power when run at 12 volts.

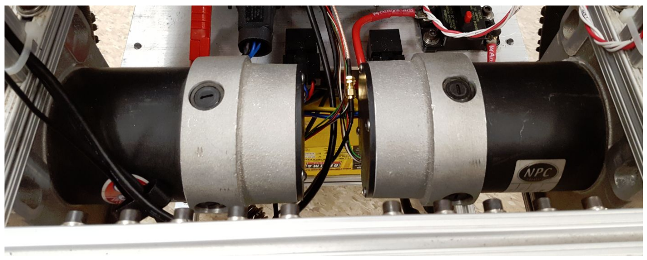
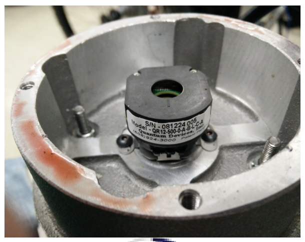


Figure 4.7 NPC-T64 Motors

Encoder Interface

The previous version of the robot used an off-the-shelf USB encoder module made by Phidgets, but this was eliminated as the new interface board handles the encoders. The microcontroller used on the board has hardware-level support for encoders, so this change



does not put extra work on the CPU of the microcontroller. This change also simplifies the code on the Jetson for reading encoders, as the raw encoder count is directly transmitted over serial when the Jetson requests it. No additional drivers or libraries are needed. Since the interface board both reads the encoders and controls the motors this will make it possible to implement PID control for more consistent robot motion.

Figure 4.8 Motor Encoder

Safety Devices

Blinking Light

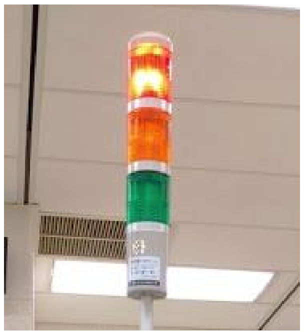


Figure 4.9 Stack Light

The blinking light design that was used for the previous robot had many good aspects so this design was improved upon for the Charlie. Three lights, one red, yellow, and green, were used again Charlie. The multiple lights allow the team to communicate more from the robot when it is at a further distance from the team. A new board was made that would allow for easier interface with the lights. Soldering the new board allowed members of the team to gain valuable surface mount soldering experience. The red light signifies that the robot is stopped and cannot move, the yellow light represents manual control mode and the green light indicates autonomous mode.

Wireless Emergency Stop

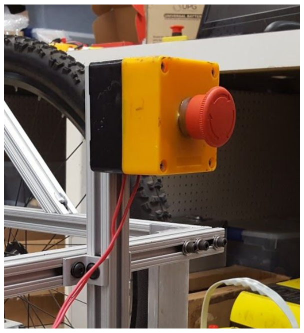


Figure 4.10 Hardwired E-Stop

Our wireless emergency stop is built into the control system; the robot will stop moving if the signal from the RC controller is lost or if a dedicated switch on the controller is toggled.

Mechanical Emergency Stop

A mechanical emergency stop is mounted facing the rear of the vehicle, as per IGVC rules. Activating this button will cut off power to the motors via relays.

Software Strategy and Mapping techniques

Overview

Our software system distributes core functionality across several nodes on several pieces of hardware. Each node is capable of communicating with each other via ROS (Robot Operating System). ROS is an outstanding platform to use above what was used in previous years due to its many libraries of prewritten code that helps heavily with navigation and obstacle avoidance. Since it is open source, there is also plenty of forums filled with helpful information regarding bugs and issues that others face so that individual projects can debug their systems more easily. Since ROS is filled with nodes, the multi-node approach allows us to made changes to each core system independently without disrupting development on a different node. This approach also allows us to take advantage of the NVIDIA Jetson TX1s exceptional multi-tasking abilities with its powerful hardware. This node network is summarized in the figure 5.1 below.

GitHub

We use GitHub as a way to keep all of our code organized. Every change is pushed onto the repository and submitted with a comment with what was changed, so each member knows what had been edited. This allows us to have issue tracking (continuity) to roll back if a newer revision of the code does not work or breaks some other code. We also have a wiki page to be able to have easy documentation access. Here we keep some useful commands and other code that is useful to have on a regular basis with working on the robot code.

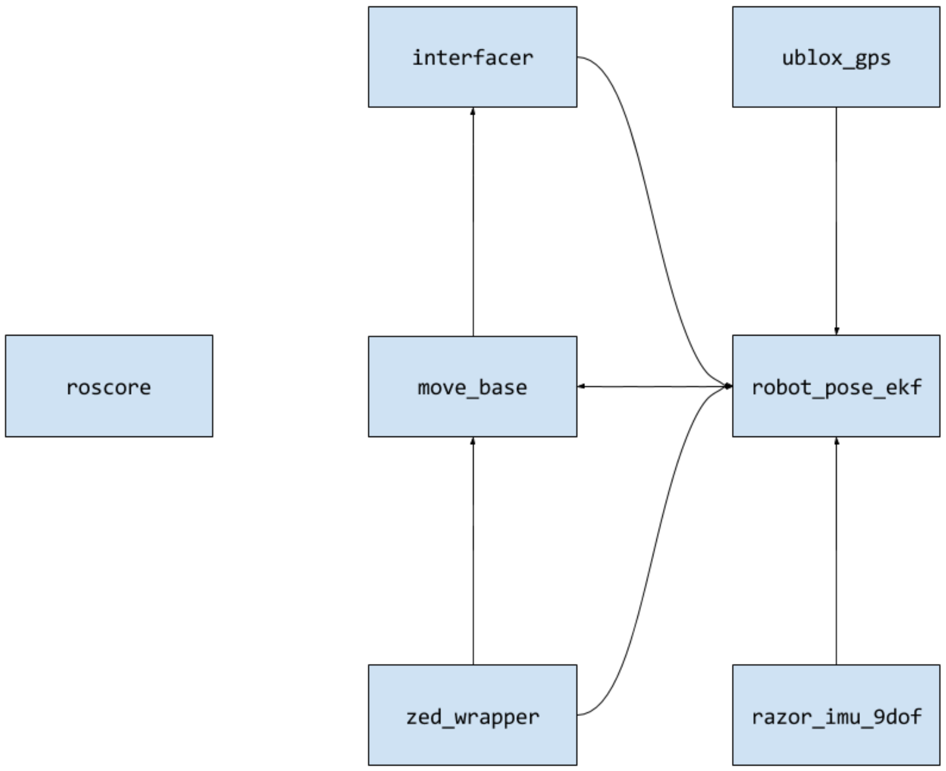


Figure 5.1: Network design

Given how the higher level nodes such as navigation and mapping rely heavily on the lower-level “reader” nodes, our design employed a bottom to top design approach where the core functionality of each lower-level node was stress-tested and validated before the development of the higher-level nodes that rely on the integrity of the data coming out those nodes.

Obstacle Detection

The ZED camera has been an outstanding sensor to map the area around the robot using ROS’s built in functions to gather points and plot them on the world. It is proving to be reliable in multiple lighting conditions. The main plan is to use the ZED camera, imu, gps, and encoders to help create an accurate map. ROS takes in the stereo camera data and will note points that are at a specific height above the ground to flag as an obstacle. There is also the capability to train the system to learn what a barrel looks like for increased reliability that can be further looked into.

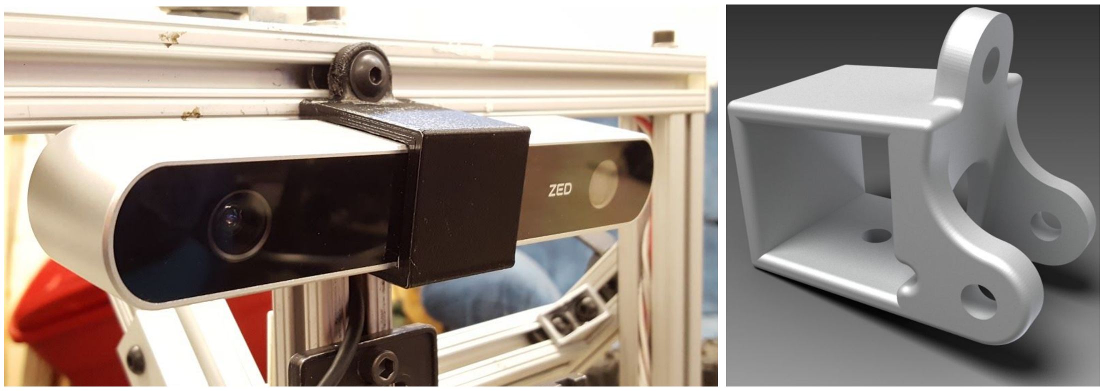


Figure 6.1: ZED camera in 3D printed mount

Map Generation

Line Tracking

Our line tracking software pulls the raw camera images from the ZED camera that are already available in ROS. This means that we don't need another camera to do line detection.

Using OpenCV, the line tracking software takes image data in from ROS and runs a perspective shift on the image to make later distance calculations easier to perform. It then filters the image based on color, and runs that through a canny edge detection filter. We then run a Hough Transform on the resulting image to give us lines (Figure 7.1). This Hough transform allows us to disregard any unintentional detection that occurs from things like white flowers in the grass. These lines are then populated onto a point cloud to be sent back to ROS to be added to the occupancy grid as obstacles.

We also have written a calibration program for our color filtering. This program allows us to quickly calibrate our line tracking for any situation using sliders and a visual representation of the final filtering. It also allows anybody on the team to calibrate the camera without much knowledge of the system. In our testing we were able to go from no calibration to competition ready calibration in less than five minutes.

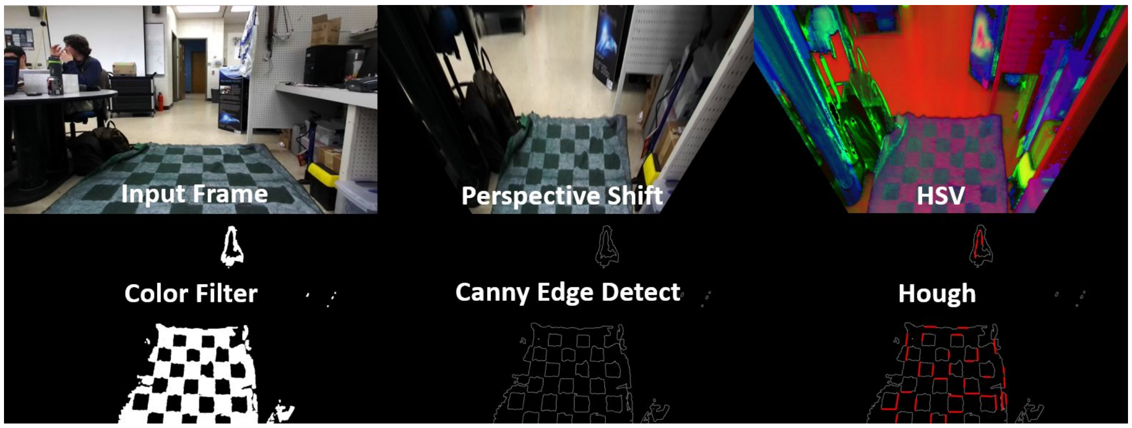
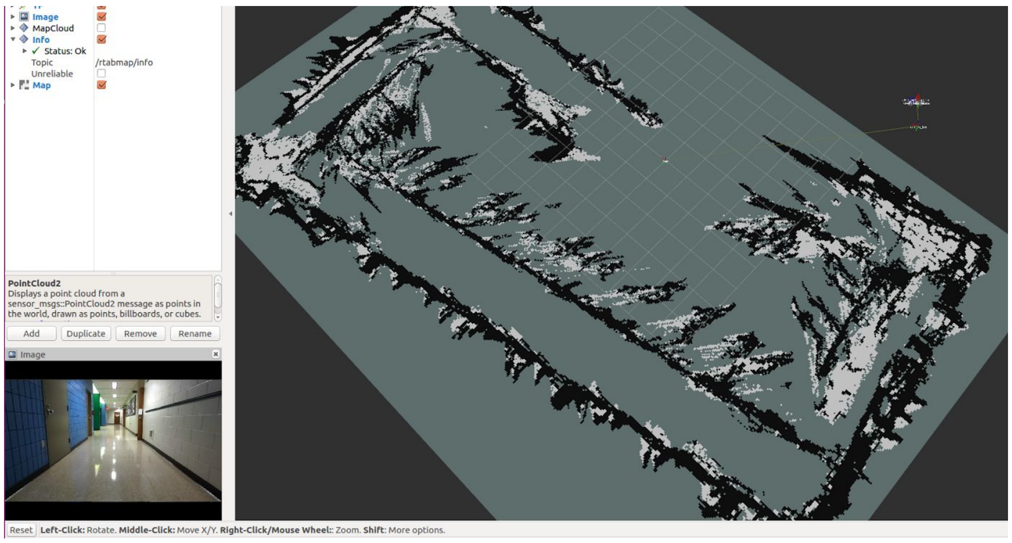


Figure 7.1: Image processing system

Occupancy Grid

The objects detected from the ZED camera data are overlaid onto the lines detected from our cameras. These systems are married together into their final form as an occupancy grid. This occupancy grid is published via ROS.



Displays a point cloud from a sensor_msgs::PointCloud2 message as points in the world, drawn as points, billboards, or cubes.

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.

Figure 7.2: Vehicle's occupancy grid

Goal Selection and Path Generation

Goals are selected by running through an ordered list of GPS waypoints stored in a text file. The selected GPS waypoint location is given in degrees longitude and degrees latitude.

These points are then input into ROS as goals. ROS then utilizes the occupancy grid it has generated from the mapping node to determine the shortest path between the vehicle and the GPS waypoint. The resulting path is stored and the vehicle is commanded to move along the generated path.

Gazebo

This useful software helps to be a virtual representation of how our robot will act in the real world. It includes a comprehensive, real time physics engine capable of simulating complex robot behavior. Using gazebo we are able to construct virtual simulations of outdoor environments complete with glare, ambient sensor noise, coefficients of friction, and varying terrain height and grade. This allows us to replicate almost any environmental conditions the robot might have to operate in.

In addition, gazebo also offers us the ability to tack almost every aspect of the simulated robot in real time. This includes: all ROS components, simulated power, wheel slippage, and much more. Gazebo also offers the ability to playback the simulation for further analysis of any bugs that may be found. Since the full code was not complete while the weather conditions outside were inadequate for testing, we did most of our testing directly on the robot in inside and outside conditions since it was the easiest to test with.

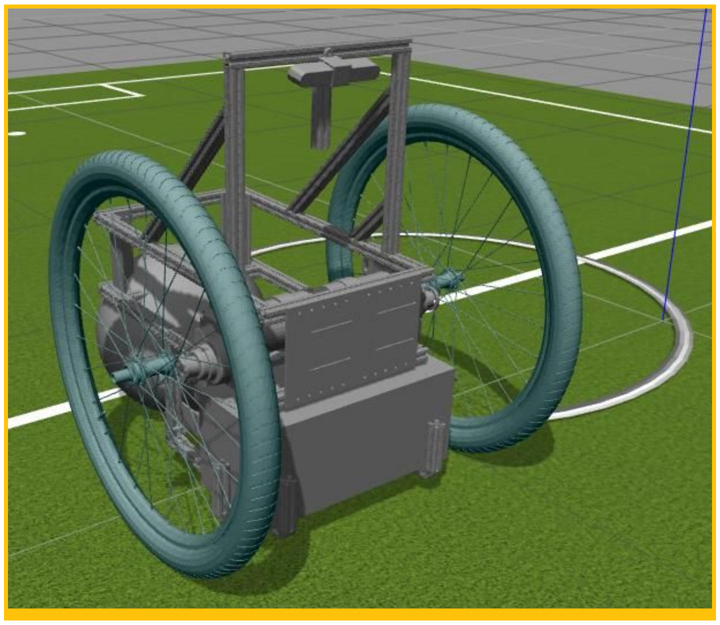


Figure 7.3: Model can be used to put accurate sensor locations on robot

Failure Modes, Failure Points and Resolutions

Software Failures and Resolutions

The most likely source of failure in software is an error in the data that is sent into the mapping node. If the data points do not properly correspond with their real-life positional relationship with the vehicle, then we are prone to collisions with obstacles. The address this concern, we apply probabilistic filters to all data coming of the nodes that feed into the mapping node to help eliminate any extraneous data. Our mapping node also creates a “no-fly” zone around all obstacles and lines. This gives our path-following algorithms a large margin for error and minimizes the effects of both failures with our control algorithms and mapping nodes.

Mechanical and Electrical Failures and Resolutions

Our chassis has several redundant supports to minimize the effects of any mechanical failure in the chassis.

All devices in our system are properly connected with fuses to minimize the possibility of an electrical failure from damaging our more sensitive electronics.

Our sensitive electronics are all enclosed within the body of the vehicle, which is enclosed between the two front wheels. In the case of a motor lockup or a control failure, all sensitive electronics are properly protected from damage.

Failure Prevention Strategy

While we have planned for more redundant safety features, we lack the human and monetary resources to implement them at this stage in the vehicle’s design. Currently, our chief failure prevention strategy is less about preventing failures, but more minimizing the risk of failure. To do so, we have design each system with exceptionally large margins for error at the expense of vehicle efficiency. Our navigation node will also cease operation if any of the nodes that it relies on fail.

Performance Testing To Date

Because we focused on the core systems, our testing focused on competition of a qualifier-like course. This course consisted of a 5 meter long lane indicated by two perpendicular white lines separated by five feet painted on artificial grass and a single barrel.

Before ROS was implemented ,additional navigation tests were performed to test the robustness of our older pathfinding system by giving the vehicle a waypoint 10 meters in front of it and placing three to five barrels in its path.

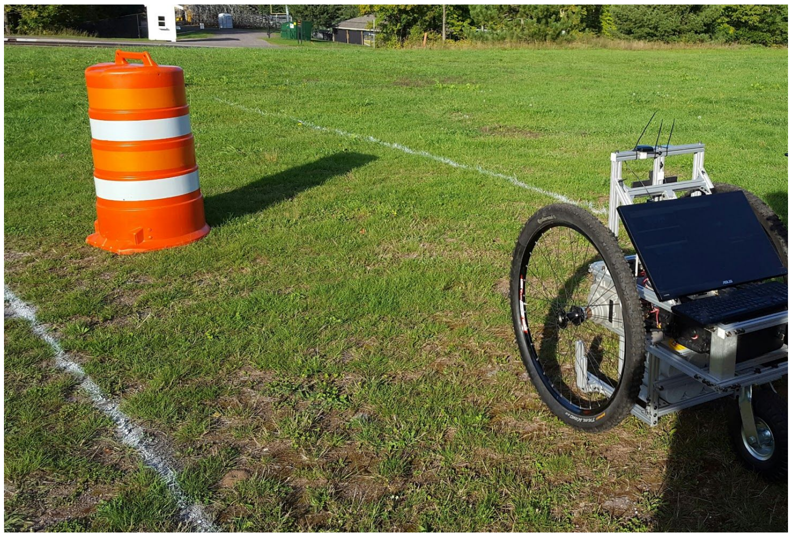
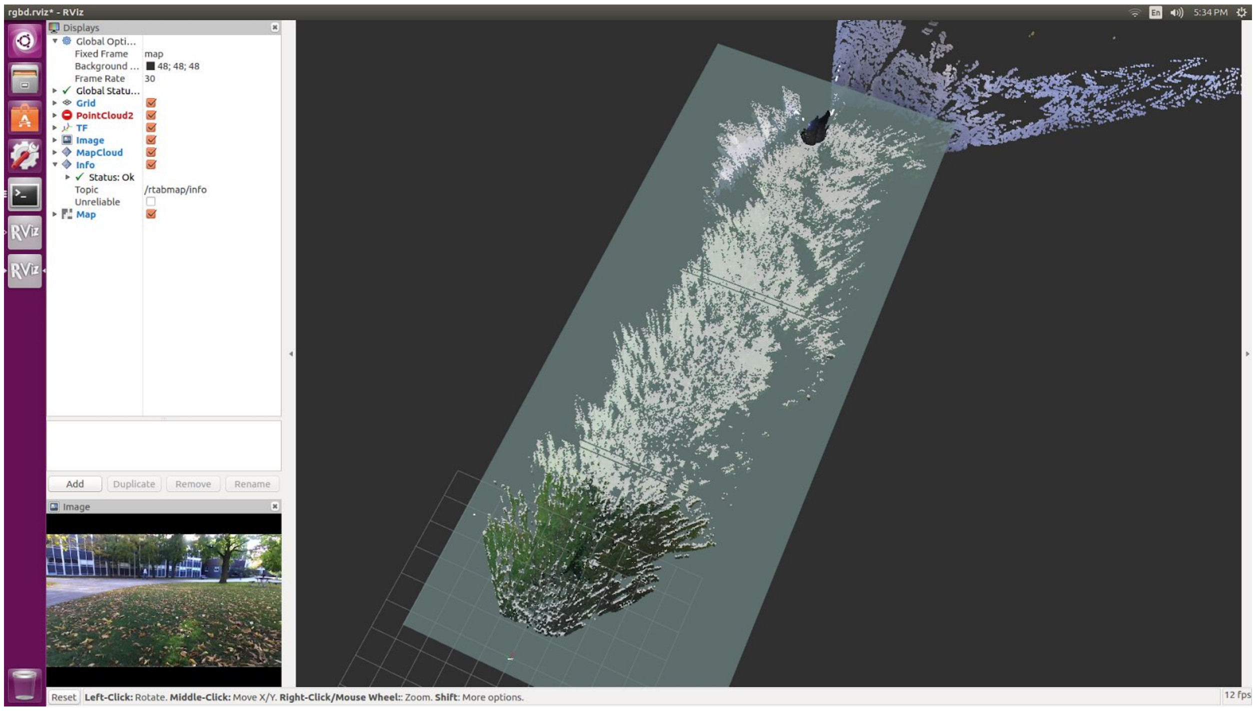


Figure 9.1: Field Testing Qualification Course

Initial Performance assessment

Our newly design vehicle outperforms the previous designs in past years—which struggled to even qualify even after three years of development. We do not expect the vehicle to perform every task during the auto-nav portion of the competition as the new ROS driven vehicle is still in its infancy, but should be able to complete the qualification course.

Testing does suggest that a chassis modification might be necessary to either redistribute weight off of the caster wheel or move over to a four-wheeled design. Both of these modifications would aid in the vehicles mobility and ease the burden on future control development. Our trouble comes when the robot wants to move from the forwards direction to the backwards direction. Since there is a good amount of weight on the caster wheel and that the swivel bearings are not the greatest, the wheel has a hard time rotating accordingly. This would require a large redesign of a main component in the semesters to come.



Global Opti...
Fixed Frame map
Background ... ■ 48; 48; 48
Frame Rate 30
✓ Global Statu...
Grid ✓
PointCloud2 ✓
TF ✓
Image ✓
MapCloud ✓
Info ✓
✓ Status: Ok
Topic /rtabmap/info
Unreliable ☐
Map ✓

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.

Figure 9.2: Occupancy Grid of Dillman Lawn

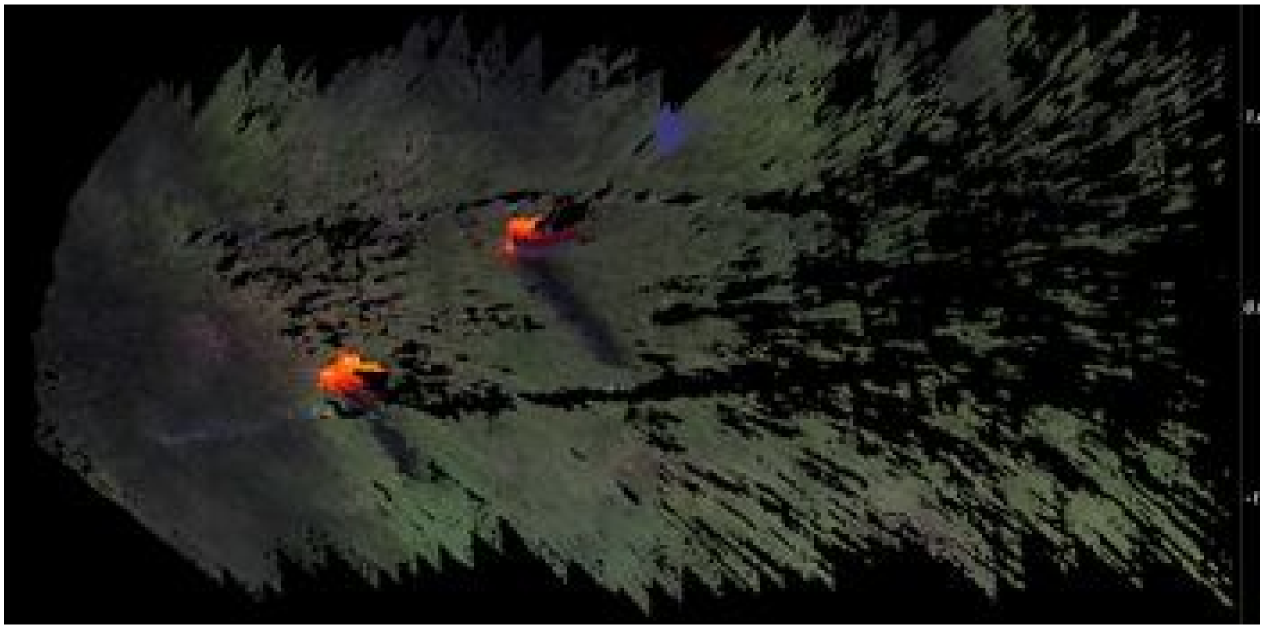


Figure 9.3: Occupancy Grid of Qualification Like Course