ESP – Environmental Surveying Platform

Benjamin Goerdt, Devon Wilkerson, Sergio Gonzalez, Kris Choudhury

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

***Abstract* — This project is designed to implement a platform for navigating a variety of potentially dangerous environments to identify possible zones for habitation. The goal of the platform is to work with an autonomous vehicle system that utilizes the onboard sensors for data collection. A user interface is used for data integration that has been gathered by different sensors to display the data into visual components. A NVIDIA Jetson Nano running the robot operating system (ROS) integrates a Kinect sensor to detect hazards in real time, while navigating to a GPS waypoint.**

***Index Terms* — Autonomous systems, Data collection, Data integration, Machine vision, Object recognition**

I. Introduction

This project was created with the intent to develop a low-cost data collection device to access remote areas where humans might not be able to access or spend extended periods of time. With rough terrain and limited communication methods the vehicle must be able to calculate a route and travel between waypoints while observing the environment along the way.

Using a high accuracy GPS module, we can ensure that in the event of a cellular service interruption, or radio communication failure, that the vehicle will still be autonomous and able to continue its calculated route with the end goal of a safe return.

Beyond route calculation, the vehicle has onboard sensors and cameras to enable object detection to alert of obstacles along a path. We have achieved SAE Level 4 Autonomy in which the vehicle can take input from its surroundings and make safety decisions. The ESP has the option to reroute around the obstacle and resume its path. If there is not enough physical clearance or a low margin of safety for the maneuver, the vehicle will pull over to safety, summon help, and wait for assistance to arrive.

With the distraction of driving removed, the data is presented on a touchscreen dashboard in a friendly user interface for an operator to view. Simultaneously it is also transmitted to a web portal that can be accessed when cellular service, or WIFI, is available. All sensor and location data is recorded to a rolling window storage configuration onboard.

II. Design

Our vehicle consists of multiple subsystems that are interlinked by I2C to communicate data quickly and effectively. Speed, and data accuracy, are the main concerns when developing and prototyping. A response time of 100ms was needed for the collection, transmission, calculation and display of all the data collected by the vehicle's sensors and camera.

A. Chassis

The chassis that this vehicle's technology is built upon is a power wheels ride on 12V toy. It was chosen due to its weight capacity, mounting area for a solar panel, and remote control. The dimensions are 39 x 26 x 20in and can support a payload of 61lbs. The steering wheel has been removed to allow for the mounting of the touchscreen display, as well as the dashboard region housing various computing devices. The under-seat compartment has been cleaned up to allow components to be more spread out, thus reducing the heat and need for active cooling methods.

To aid in the traversing of rough terrain, a track system has been developed to eliminate the front steering. This allowed us to achieve a tighter turning radius, lower our center of gravity, and add rigidity to the frame.

The platform has many modifications that can be done to ensure reliability and more power, but none of them were required after thorough testing.

B. NVIDIA Jetson Nano

The NVIDIA Jetson Nano runs all autonomous navigation and object detection via the Robot Operating System (ROS). Its small form factor is a great choice for this project due to the small size of the vehicle our team is using. It supports SPI, I^2C, and UART communication, which provides a wide array of sensors that are available to use if needed. The powerful CPU and GPU provide a lot of processing speed for the large number of calculations and data needed for ROS to navigate and detect objects accurately and reliably.

C. Microcontroller

The ATMEGA328P is the microcontroller of choice for our vehicle. This 23-pin device met our needs due to having an adequate number of analog pins to run the distance sensors, as well supporting I^2C communications.

We did not need to order this microcontroller as a few of us already had one on our Arduino devices. Arduino has an

extensive sensor library, which makes integrating all the sensors together a much easier task.

D. Sensors and Circuit Board

The two-layer board was designed with heat dissipation in mind, length matched I2C busses, and 2 oz/ft copper deposition to achieve a compact form factor. Headers were designed to be mounted on the perimeter of the board to allow for long runs of wire for external sensors. The total size is 100x100 mm. Our sensors were chosen by price to performance ratio, where the best performing sensor within our budget would be chosen.

Price and communication protocol were the two most important factors that were taken into consideration when selecting what sensors to use on our vehicle. The goal was to get sensors that communicated using I²C for as cheap as possible.

The distance sensors were the only ones that do no communicate using I²C, instead they use analog communications. These sensors are attached to each side of the vehicle to help avoid potential obstacles.

There are different kinds of light sensors attached to the vehicle; the VEML 7700 Light sensor and the VEML 6070 UV Light sensor. Both sensors have an I²C interface and are used to monitor the amount of ambient light around the vehicle as well as to determine how much UV light is hitting the vehicle.

The PMSA003I Particulate Matter sensor, SGP40 Volatile Organic Compound Sensor, and the SEN0322 Oxygen sensor are all used to monitor the air quality surrounding the vehicle. There were other options for these sensors that were cheaper, but they were all analog, so we chose to pay a little more to have I²C communication.

There is also the AM2320 Temperature and Humidity Sensor. This one sensor will allow us to display the current temperature as well the percent humidity in the surrounding area. This device was the cheapest of all the sensors and still allowed for I²C communication.

E. Data Displaying

For our data visualization, we are using a Raspberry Pi 4 and Raspberry Pi 7" Touch Screen. We decided to go with that size of touchscreen as we wanted users to be able to interact with the UI. It was also able to fit on a Power Wheels vehicle. The reason we are using the Pi as a computing device was due to its small form factor as well as the ability for it to be able to host the UI application directly, without the need of a separate server.

The UI application is powered by Node-RED which is a flow-based programming tool for event-driven applications[4]. The application would retrieve data from the sensors to upload onto a dashboard for visualization. There are two main sections for our dashboard: the environmental gauges and the map. The environmental gauges are receiving data from our various sensors through I²C communication. Not every sensor we used had the ability to communicate through I²C, specifically with the Grove Air530 GPS sensor which would be used to mark the current location of the vehicle. To solve this issue, we connected the GPS sensor to an Arduino Uno which would process coordinate data, then send that data through serial communication, and finally send that data straight to the Pi.

For custom made waypoints, our UI sends coordinate data to the Jetson Nano to process that information and let the vehicle navigate to the coordinates as long as there is a stable internet connection. Since internet connection will not be available in almost any remote environment, our team integrated a SixFab 4G/LTE Cellular Modem Kit[5] into the design. The modem connects directly onto the Pi and allows it to connect to cellular networks for internet access.

III. USER INTERFACE

The major components involved in the user interface, that were briefly highlighted in section II, will now be discussed in more technical detail. For clarification, we are using an Arduino Uno which has the same microcontroller that we are using for our PCB, but solely for the purpose of sending the GPS sensor data directly to the Raspberry Pi. The reason for this is that it did not prove to be wise to create a new revision of our PCB to accommodate one sensor especially since it took generally a week to receive a new iteration of the PCB. Not only that, with deadlines coming up we felt that it be much easier to use an Arduino Uno to send that data directly to the Pi.

A. Raspberry Pi 4 Model B

The Raspberry Pi 4 Model B is the computing device that is hosting our UI. The specific model we are using has 4 GB of RAM and 32 GB of Storage. It also comes with various GPIO headers which we used to connect to our PCB to receive all the sensor data via I²C communication. The Pi comes preinstalled with an application called Node-RED. As previously stated, Node-RED is a flow-based programming tool that allows for wiring of data together with flows through a browser. As it is a development tool it has support for libraries to add different functionalities.

One such library which is the foundation for our UI is node-red-dashboard. This library allows data to be "wired" directly to visual components to display. As node-red is always running that means data can be received and updated in real time. Because of this functionality, we can display data in real-time within 100 milliseconds.

B. Raspberry Pi 7" Touchscreen Display

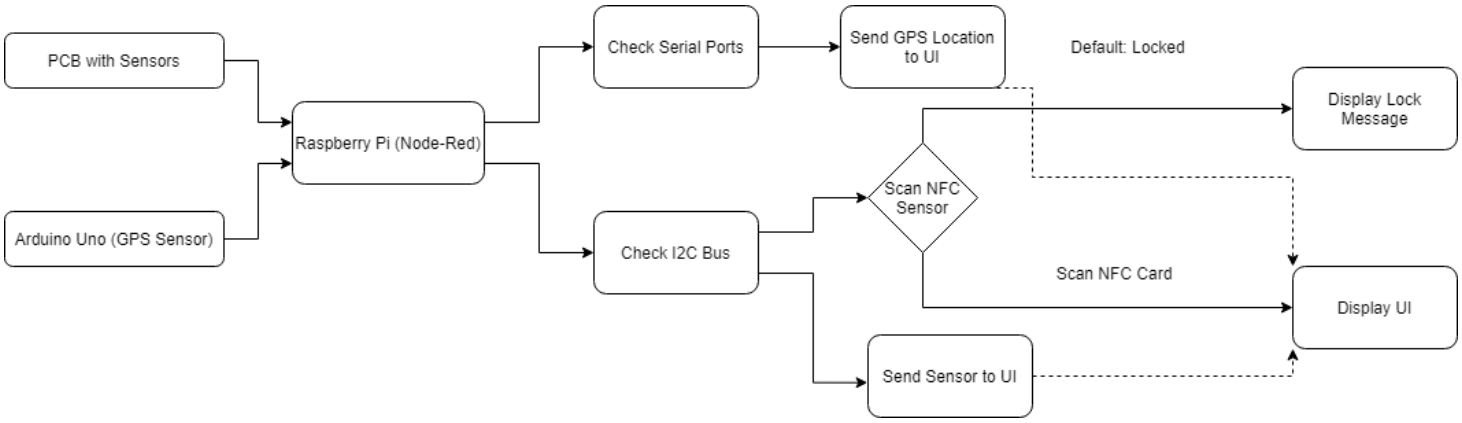


Figure 1: Node-RED UI Flowchart

The Raspberry Pi 7" Touchscreen Display is what is displaying the UI. As discussed in section II, we chose this display specifically for the size, resolution, and connections. For the size we went with 7-inch display as we found that other touchscreen displays were either too big or too small for our purposes. We found that the 7-inch display seemed to work the best for our situation as it was the best size for our prototype.

Resolution also proved to be important as well. As it stands, we found that a higher resolution proved to be more intensive to the CPU. We tested this theory by manually setting the default resolution to 1920x1080p, which proved to be much more intensive than the lower resolutions. Because of this, we had to limit our options for the resolution of the display. The 7-inch touchscreen display has a 800x480 resolution which proved to be sufficient in displaying our UI and is not as resource intensive.

In terms of connections, the 7-inch display has the least amount since all it needed was a DSI cable, a 5-volt wire, and a ground connection. Because of this, we had more pins available to use, and decided to use the extra connections for the I²C communication devices.

C. Node-RED Application

The Node-RED application is a program that comes with all Raspberry Pi devices. This application allows for the manipulation of data to be used for a variety of components. Such components can be from a variety of third-party libraries[4]. Our UI will be using a first-party and third-party library. The libraries we are using are node-red-dashboard and node-red-worldmap. Both libraries include dashboard components related to the UI and is separated into two different tabs: Sensor Info and World Map.

As previously mentioned, we have a GPS sensor connected via UART and all the other sensors would be connected via I²C. The GPS sensor is used for the worldmap tab as it is used to mark the current location of the vehicle. For all the other sensors, which will be discussed in more detail in section IV, most of them are used for the Sensor Info section. The only sensor that is being used for both tabs is the NFC sensor from DFRobot.

The purpose of the NFC sensor is that it is being used to secure the vehicle. The way we are securing it is by locking the UI from user interaction. This works by hiding the visual components for both tabs and presents a default component that asks the user to scan the correct RFID card, which will then "unlock" the UI and display the map and sensor information.

In Figure 1, it demonstrates the flow of our UI to receive the correct data. The Raspberry Pi picks up data from its I²C bus as well as from its serial ports. Then in Node-RED, it can access that data from those ports and then manipulate it. For the I²C data, each sensor is connected to a corresponding visual component. The NFC sensor, which is part of the I²C package, is not connected to a visual component but is connected to a function. By default, the function sets that the lock screen to be displayed and hides the sensor and map component. Once an NFC card is scanned, the lock screen is hidden, and the visual components are shown. As an added feature, there is an option to lock the UI again if the user scans the card again.

The map tab utilizes the GPS sensor that's being received from the serial port. The data from the sensor is sent to a world map node which receives coordinates to mark the current location of the vehicle.

Since the application is browser-based, we have the Pi set to kiosk mode and dedicated solely for the UI app.

D. Grove Air530 GPS Sensor

The Grove Air530 GPS sensor[6] is what is being used to always mark the current location of the vehicle. The sensor runs off 5V with a baud rate between 9600 – 115200. The sensor picks up GPS signals from up to 6 satellites at the

Figure 2: UI Design

same time and supports most GNSS positioning systems. Map. In terms of accuracy, the sensor can get a 2.5 meter horizontal and vertical accuracy.

The sensor is connected to an Arduino Uno which accesses the Tx and Rx pins of the GPS. Once the connection is made, a library called TinyGPS++ which is a library that parses NMEA data streams[7]. Without this library, the data that the sensor receives is cannot be used directly. Once the library parses the data, the latitude and longitude and printed to a serial monitor. From here, once the Arduino Uno is connected to the Pi through a serial port, Node-RED accesses that data stream and constantly receives the latitude and longitude of the sensor at its current location and marks it as a waypoint on the map.

E. SixFab 4G/LTE Cellular Modem Kit

The SixFab modem kit[5] extends the networking capabilities of the Raspberry Pi. Normally if we were just using the Pi by itself, we would need to be connected to a Wi-Fi network to use the internet. Since the map portion of the UI requires an internet connection to display it in the first place. This means that if we tested in environments which do not have an established Wi-Fi network then the part of the UI would not work.

As the UI is essential to the design, we felt that it was necessary to include this component in the vehicle even if the Pi was connected an established Wi-Fi network. The reason for this is so the UI has something to fall back on for two reasons. The first being if the vehicle happened to navigate outside the range of the network, it would something to fall back on to avoid any issues with marking the current location of the vehicle as well as making new waypoints. The second would be if the network that the vehicle's connected to would go down during operations then the map feature would fail entirely. With the kit, we can avoid all these issues as well as open up further options for this networking.

IV. Sensors

The purpose of the vehicle is to be able to be deployed in different kinds of environments and gather information of the surrounding area. To do this, it will need to be equipped with different kinds of sensors. There will be a total of 12 different sensors on the vehicle.

A. HC-SR04 Distance Sensor

Four of these sensors are the HC-SR04 ultrasonic distance sensors will all run on 5V. Each sensor emits a high frequency sound wave while sending a signal HIGH to the microcontroller. The sensor will signal LOW when it detects the reflected sound wave. The microcontroller will determine distance using the formula.

$$distance = \frac{time * 0.034}{2}$$

The constant 0.034 is the speed of sound, in units of cm/µs.

B. PMSA003I Particulate Matter Sensor

There is also a Particulate Matter sensor (PMSA003I) that takes in samples of air in a small chamber and his hit with a beam of light that gets scattered and read by a photodiode. The Particulate Matter sensor used on our vehicle will tell the microcontroller how many particles of varying sizes are in the air sample. The PMSA003I is capable of distinguishing are 3, 5, 10, 25, 50, and 100 micrometers (um). This sensor needs 5V to operate.

C. VEML 7700 Light Sensor

The VEML 7700 is the Light Sensor that is equipped on the vehicle. The sensor can take a maximum supply voltage of 4V but thanks to the 3.3V voltage regulator on the breakout board, it allows us to plug it in to our 5V power supply without concern of burning out.

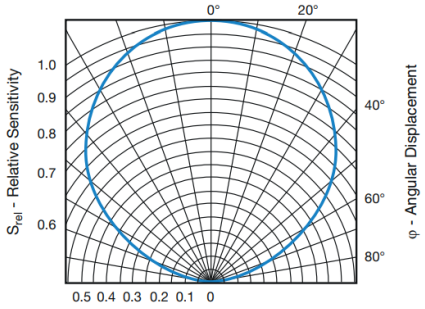


Figure 3: Radiant Sensitivity vs. Angular Displacement

The angle at which light his hits the sensor affects the output value[8]. This sensor can measure up to 120 kLux.

D. VEML 6070 UV Light Sensor

Our vehicle is also equipped with a VEML 6070 UV Light sensor. This device takes in 5V and communicates via I²C. Contrary to other low-cost UV light sensors on the market, the VEML 7700 can detect light in the UV spectrum. It doesn't output a UV index value, instead it outputs a value depending on how intense the UV light is. The maximum value that can be outputted is 65535.

E. AM2320 Temperature and Humidity Sensor

The AM2320 will monitor the temperature as well as the relative humidity around the vehicle. Takes in 5V to operate and also uses I²C for communication. It has a range of -40 °C up to +80 °C for the temperature sensing and has range of 0% RH to 99% RH.

F. SEN0322 Oxygen Sensor

The O2 sensor equipped on the vehicle is the SEN0322. It takes in 5V and communicates with I²C. The amount of oxygen in the air we breath is around 21%, 78% belongs to nitrogen with the remaining percent going to argon and other gases. The SEN0322 can measure the amount of oxygen in the air up to 30%. This allows us to ensure that the environment is safe enough to breath. Unsafe oxygen percentage is below 19%.

G. SGP40 VOC Sensor

The Volatile Organic Compound Sensor equipped on our vehicle is the SGP40. This sensor takes in a maximum of 3.6V but has a voltage regulator on the breakout board that allows for a 5V supply voltage to the device. This device allows us to determine the VOC index of the area. VOC index's range from 0 to 500, with 100 being the average index indoors.

H. DFR0231-H NFC Sensor

Our vehicle will have the DFR0231-H NFC sensor to simulate unlocking and locking. This sensor requires 5V to operate and has both I²C as well UART. The communication protocol can be interchanged with a flip of the switch. It supports the Reader/Writer protocol, the Card Emulator protocol, as well as the Peer-to-Peer protocol.

I. Microcontroller Programming

To program the ATMEGA328P, we used the Arduino IDE. In the program is a custom data structure that will hold all of the information so that it can be sent via I²C to the Raspberry Pi. The data structure is a total of 31 bytes so the ATMEGA328P will not have to send multiple packages since the Node Red flow that will be running on the Raspberry Pi has a maximum data package of 32 bytes.

The NFC sensor will determine if the touchscreen UI will be locked or not. As the user taps the NFC with a supported NFC tag, it will invert a Boolean value.

When the Raspberry Pi sends a request for data, the microcontroller will jump to Interrupt Service Routine that sends each byte from the custom data object to the Pi.

V. Navigation

The autonomous navigation is controlled by a NVIDIA Jetson Nano running the robot operating system (ROS). To successfully navigate, The Jetson Nano is using multiple packages in ROS, which includes the navigation package[1] for localization and navigation, the robot_localization[2] package for sensor fusing and localization, the freenect_stack[3] package for depth information from the

Microsoft Kinect, the nmea_navsat_driver package for GPS data, the adafruit_imu package for gyroscopic and linear acceleration data, an encoder package for velocity data and localization, and a motor controller package for navigation. The Jetson Nano communicates with a gyroscopic sensor, and an accelerometer over I^2C, a GPS over UART, and an Arduino Mega and Microsoft Kinect over USB.

A. Motors

The motors used on the vehicle are RS390 motors. These motors are powered by a 12V battery and are controlled by the Arduino Mega. The Arduino Mega is used as a motor controller since the Jetson Nano is unable to have a digital low signal sent to the pins, which causes floating voltage levels throughout the enabled pins. This in turn produces unpredictable behavior in the motors causing them to randomly turn on when the pins receive voltage even when the pin should be in a low state. The motors also have two BTS7960 motor drivers controlling the voltage applied to the motors for control. These motor drivers are placed between the battery and motors since the Arduino Mega is incapable of providing the necessary voltage to power the motors, and the Arduino Mega controls the voltage going through the motor drivers via digital pins.

B. Odometry

The odometry is calculated via encoder ticks from a Quadrature encoder attached to the shaft of the motors. These encoders calculate the velocity of the vehicle by using the following formula:

$$V = \frac{2*\pi*r*\Delta tick}{\Delta t} \quad \text{(X)}$$

where r is the radius of the wheel, Δtick is the difference in encoder ticks from the current and previous counts, and Δt is the difference in time measured in milliseconds. This approach is not accurate over a long distance due to wheel slippage and false tick counts; however, it is aided by the accelerometer sensor and the GPS, which are both fused with this approach for velocity measurements. Since the engineering requirements specify a position error by a maximum of two meters, the use of estimated velocity will not cause any issue when fused with the GPS, which has a maximum position error of two meters.

C. Object Detection

The vehicle uses a Microsoft Kinect for depth data using an infrared camera that uses a point cloud to estimate distance. The resolution of the IR camera is 640 × 480 pixels and can support a resolution of 1280 × 1024 at a lower frame rate. The IR camera also has a practical range of 1.2 – 3.5 meters and has a maximum range of 6 meters. The field of view of the IR camera is 57° horizontally and 43° vertically. Due to the large size of the depth data, which would often be over 10MB in size, the depth information is only updated at a rate of 3 Hz to avoid overtaxing the Jetson Nano's processor. During testing this would lead to a total system failure, due to the Jetson Nano being unable to process over 300MB of data per second from the Kinect alone. The depth information is processed by the navigation package to populate an occupancy grid via the costmap_2d package within the navigation package. This package will keep track of any potential obstacles of any size and will allow the vehicle to properly plan its route accordingly. The vehicle does not have any kind of mapping since it's designed for outdoor use in an unfamiliar environment, so having an accurate occupancy grid is a key part of the vehicle successfully navigating.

D. Navigation

The vehicle navigates with the navigation package. This package provides the necessary behaviors for successfully navigating with or without a map and relies on the user to provide information about the robot. The first thing it requires is a consistently updated sensor transformation message. This allows for calculating the position of sensors in relation to the base link, or the center of the robot. The next thing it requires is an odometry source. This is provided by the encoders on the wheels as discussed in subsection B. A type of vision is also required for successful navigation. This is provided by the Microsoft Kinect as discussed in subsection C. The final requirement is a base controller that the navigation stack can send movement commands to in the form of X, Y, and Yaw commands and receives velocity information VX, VY, and VYaw. Since our vehicle is using a differential drive, it only takes X, and Yaw commands and sends VX and VYaw information. A map is an optional part of the navigation package that we are not providing since the vehicle is designed to navigate through unfamiliar areas. The behaviors the navigation package provides are a global planner, a local planner, a global costmap, a local costmap, a common costmap and recovery behaviors. The costmaps are how the navigation package stores information about potential obstacles and required information about the robot such as: dimensions, observation sources (LiDAR or Kinect), observation range, global frame (non-moving sources such as odometry origin and maps), base link, update frequency, and whether a map is used. The global and local planner behaviors require information about the

maximum velocity and acceleration limits of the robot, as well as if the robot is holonomic or differential.

VI. Electrical

The electrical system of the vehicle is separated into two power sources with a common ground between. This configuration was chosen to prevent a sharp voltage drop when the motors are activated, as well as allowing the hot-swap functionality of the main chassis battery without requiring a reboot of the microcontroller and data collection devices.

An additional power source to aid in extending the runtime was a solar panel attached to the hood. Rated at 100W, the panel had an actual output of 15W, so compensation was needed after initial testing. The panel has an integrated charge controller that connects to the chassis battery. At 1.25A of output current, this will provide enough power trickle charge the chassis battery, or supply power to offset a majority of load with no passenger weight in the vehicle.

A thermal cutoff fuse is mounted in-line with each motor to shield the vehicle and passengers from overload if the wheel gets stuck and the controller fails to cut power. The average current draw from the battery is under 4A but will peak over 10A when an excess load of 3x rated capacity is tested. Even with the large load being tested, there was no thermal fuse triggering nor wires generating concerning amounts of heat.

A 12V-4.5Ah Sealed-Lead-Acid (SLA) battery was selected and total runtime was not an issue even after receiving a partially defective panel. Even with the solar panel, charging can still be done using the built-in power wheels battery controller at 0.5A, or an external 12V charger up to 10A. This aligns with our initial specification as golf carts are charged via the method of an external charger.

The sensors and circuit board are powered by a 12V Lithium-Ion battery comprised of 3-18650 cells. A 2Ah Milwaukee M12 battery was the easiest source of power during testing and demonstration. Located on the exterior edge of the board are two screw connector terminals that are used mutually exclusive for powering the board. A 5V terminal is connected to the main trace of the circuit board as a backup option if the on-board converter fails at any point.

To connect directly from a 12V source, a 5V step down regulator (Pololu D24V50F5) allows us to power the sensors with a maximum sustained current of 5A. The module features reverse polarity protection to sanitize the cruder screw terminal inputs. This was beneficial over a barrel jack during testing and allows the 5V input to be dual

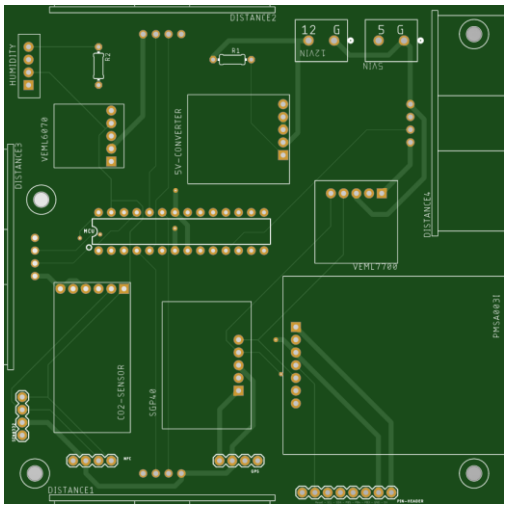


Figure 4: PCB Layout

purposed as an output port if needed. To further increase the safety of powering our devices, a waterproof inline fuse was added to the input of the board.

Powering a Kinect and transmitting data is done via a proprietary cable from Microsoft that is connected to a 110V AC power source. To combat the issue of ruining an expensive cable, allow modularity during prototyping, and providing adequate wattage, an Uninterruptable-Power-Supply (UPS) was chosen to be mounted on the vehicle. It will be charged via an extension cord while the vehicle is stopped. It has a reserve capacity of 200W, which is enough for 6 hours of runtime. Rather than be configured in the method it is meant for, the device will be used as a portable 110V source. The Kinect, our machine vision components, Raspberry Pi, and screen will be powered by this source. Rather than design our own solution, we found the safest and most compact option to be purchasing a device with a full sin wave inverter.

VII. Conclusion

This project has given our team experiences and insight to multiple different areas of engineering due to the complexity of making an autonomous vehicle. We have adapted to failures and learned from mistakes. Choosing this project was a difficult decision and a big leap out of what we were comfortable with. We chose to create a project that had aspects of the fields all our team members wanted to get a career in: Electrical Systems, Software Development, Robotics, and Embedded Systems. Given the time frame of the project we wanted to push the boundaries of what we thought we could do, while also trying to succeed in making a functioning project.

ACKNOWLEDGEMENT

This project had a lot of troubleshooting involved. Thankfully there were many people and forums that were eager to help us succeed. We would like to thank all of the users from the NVIDIA Developer Forums for assisting with Jetson Nano problems, all the users from ROS Answers for helping with navigation issues, OpenKinect and ros-drivers contributors for Kinect troubleshooting. We would also like to thank the users on the Node-RED forums as well. The user David Burrows, known on the forums as meeki007, solved most of the issues we were having with our UI, as well as gave some suggestions on what could be improved. We would also like to thank the professors, who have kindly agreed to be a part of our review committee, as well as the professors we

REFERENCES

[1]"Wiki," ros.org. [Online]. Available: http://wiki.ros.org/navigation. [Accessed: 28-Nov-2021].

[2]"Robot_localization wiki¶," robot_localization wiki - robot_localization 2.6.11 documentation. [Online]. Available: http://docs.ros.org/en/melodic/api/robot_localization/html/index.html. [Accessed: 28-Nov-2021].

[3]Ros-Drivers, "Ros-drivers/freenect_stack: Libfreenect based Ros Driver," GitHub. [Online]. Available: https://github.com/ros-drivers/freenect_stack. [Accessed: 28-Nov-2021].

[4]"Documentation," *Node*. [Online]. Available: https://nodered.org/docs/. [Accessed: 28-Nov-2021].

[5]*Sixfab Docs*. [Online]. Available: https://docs.sixfab.com/. [Accessed: 28-Nov-2021].

[6]B. Zuo, "Grove - GPS (AIR530)," *seeedstudio*. [Online]. Available: https://wiki.seeedstudio.com/Grove-GPS-Air530/. [Accessed: 28-Nov-2021].

[7]"TinyGPS++ : arduiniana," *TinyGPS++ | Arduiniana*. [Online]. Available: http://arduiniana.org/libraries/tinygpsplus/. [Accessed: 28-Nov-2021].

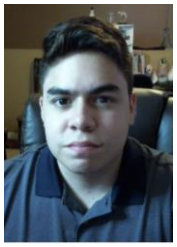
[8] *High Accuracy Ambient Light Sensor with I2C Interface.* [ebook] Vishay, p.3. Available: https://media.digikey.com/pdf/Data%20Sheets/Vishay%20Semiconductors/VEML7700.pdf . [Accessed 7 November 2021].

BIOGRAPHY

Kris Choudhury, a senior student of the Department of Electrical and Computer Engineering at the University of Central Florida will be receiving his Bachelor of Science in December 2021. After graduating, he will enter a career as an IT Software Developer at World Fuel Services.

Benjamin Goerdt, a senior student of the Department of Electrical and Computer Engineering at the University of Central Florida and will receive his Bachelor of Electrical Engineering in December of 2021. Ben's career aspirations are to work in the automotive industry furthering the research and development of electric vehicles. Rapid prototyping keeps him engaged with his work.

Devon Wilkerson, a senior student of the Department of Electrical and Computer Engineering at the University of Central Florida. He plans to continue his education and work towards a MS in Computer Engineering. His career goals are to create robotic prosthetics after graduating.



Sergio Gonzalez, a senior at the University of Central Florida and will receive his Bachelor of Science in December of 2021. He will begin his career as a Software Engineer at Laserstar Technologies.