NineBits Bot

University of Detroit Mercy IGVC-Self Drive 2022 Design Report

Submitted May 15th, 2022

Captain: Giovani Olivo olivogm@udmercy.edu

Team Members: Mariam Bakhaet: bakhaemm@udmercy.edu, Kirols Bakheat: bakheakm@udmercy.edu, Victor Carpenter: carpenvl@udmercy.edu, Tyler Caton: catontb@udmercy.edu, Kevin Conklin: conklike@udmercy.edu, Renata Gonzalez-Navarro: gonzalre@udmercy.edu, Carson Green: greence2@udmercy.edu, Jiyang Guo: guoji4@udmercy.edu, Giovani Olivo: olivogm@udmercy.edu

Faculty Advisor Statement:
We certify that the engineering design in this vehicle undertaken by the student team, consisting of undergraduate students, is significant and qualifies for course credits in senior design and in the undergraduate program respectively.

Advisor Signature: [illegible]

Introduction

Throughout the course of the 2021-2022 school year, the Electrical Engineering and Robotics and Mechatronics Engineering senior classes worked on a Polaris Gem e2 named NineBits Bot, to compete in the Intelligent Ground Vehicle Self Drive Competition (IGVC). This 48V electric vehicle has been converted to be capable of completing various autonomous tasks, and capable of handling primary driving functions, designed for people with disabilities. These primary self-driving functions include lane following, obstacle detection, sign detection, parking, and basic vehicle maneuvering such as turning and lane changing.

In the following document, team organization, design process and organization, innovations, vehicle specification overview, and software/failsafe strategies will be discussed.

Organization

Table 1: Team Organization and Responsibilities

Team Member	Task Responsibility
Mariam Bakhaet	Lane following
Kirols Bakheat	Vehicle Control and Safety
Victor Carpenter	Sign Detection
Tyler Caton	Lane Following
Kevin Conklin	Sign Detection
Renata Gonzalez-Navarro	Obstacle Detection
Carson Green	Vehicle Control and Safety
Jiyang Guo	Sign Detection
Giovani Olivo	Obstacle Detection

Design Process

The senior design class first started working on organizing the team structure and responsibilities for each student. The beginning of the project involved understanding what capabilities the vehicle had prior to beginning our individual responsibility. Once there was sufficient understanding of the onboard technology and capabilities our vehicle was equipped with, the team was able to identify areas for improvements. The team then created subgroups and assigned responsibilities for each subgroup member in the areas where improvements were needed, see *Table 1*. Each student began with a research task that allowed for gaps of knowledge to be filled in order to gain a better understanding on how to proceed with the implementation of their task. Once research had been completed, a Design Matrix was created that allowed the teams to brainstorm solutions to the areas of improvements that were identified during the early learning phase of the project. The Design Matrix was helpful in the decision process to identify the best possible solution to identified areas of improvement during this phase of the project. A failure mode effect analysis (FMEA) was conducted for each of the subgroups' responsibilities in order to consider all failure mode possibilities and optimize the robustness of proposed new designs and the existing designs. Once a design for an area of improvement was finalized, that team would then begin the initial task of implementing, and troubleshooting the new design, as well as to continue to refine the design. The new designs were analyzed and tested at project milestones in order to ensure that the designs would integrate into the final overall design. Once all designs were complete the vehicle went through a final testing and validation phase in order to be able to take the vehicle to the IVGC.

Innovations

Several innovative techniques were used on vehicle designs to complete tasks for the IGVC.

Neural Networks – Sign Recognition

The technology that was used for the traffic sign and lane line recognition task is a Machine Learning implementation using neural networks. The neural networks consist of connected nodes that are able, through mathematical computations, to learn the features of certain objects based on training. Training allows for the detection of objects, and detection is complete when they ‘see’ those objects. A live video feed from a Prosilica 1290c camera is fed to the Nvidia Detectnet network. The network has been trained using Python3 and Pytorch to side load the model with thousands of images. These images are of road signs and include stop, right facing one way, left facing one way, road closed, and no turn. The system will place a bounding box around the object that it thinks it has detected along with the name of the object. The bounding box can be used to crop the image to reduce the amount of image clutter. The network can process up to 20 frames per second using the Prosilica camera.

The neural network was developed on Ubuntu 18.04, running on an NVIDIA Jetson Xavier AGX platform. The code is a combination of scripts for different parts of the overall process and C/C++ code for the neural network itself was obtained online from a repository on gitHub, Additional scripts were developed by our team to manipulate, rename, resize, and organize the datasets that were used for training. TensorFlow is the main platform that the neural network was built upon. The live video stream from the Prosilica camera is captured and published as a ROS topic, the neural network subscribes to this topic and uses the topic data in the sign detection task. The output from the sign detection task can be combined with LiDAR positioning information for purposes of sign localization.

Picture 1 Sign Labels

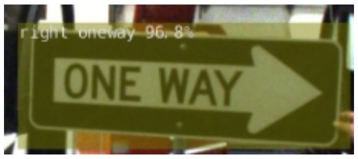
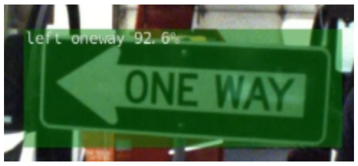
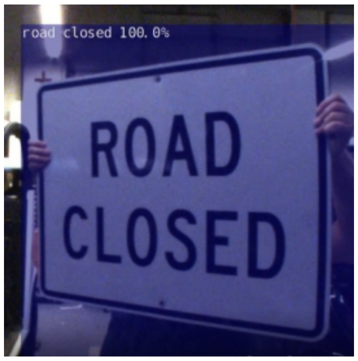
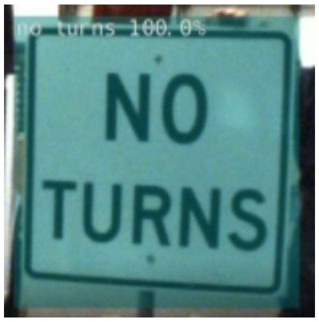
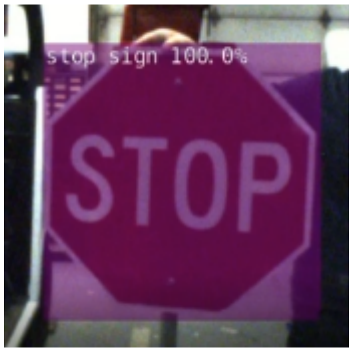
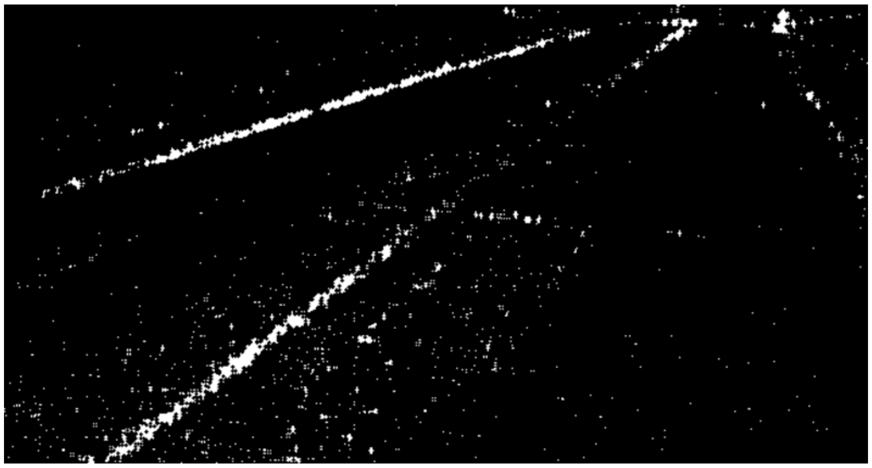
Sign Detection Results for a required signs in IGVC rul		
		

Image Processing- Lane Line Recognition

A new lane detection system was developed and implemented on the vehicle using image processing techniques instead of the incumbent machine learning techniques. This decision was made to allow more control over input parameters, as well as easier scalability when adding future features. It also forced the lane detection group to learn the ins and outs of how lane detection worked, instead of relying on a trained model. The incumbent SVC210 fisheye camera was replaced by a standard logitech C90, 1920 x 1080 USB camera, for testing and training with a Prosilica 1290GT serving as the detection camera for competition. The NVIDIA PX2 was also replaced originally with a Jettson Xavier AGX, and then with an intel NUC once the image processing route was decided upon and the reliance on the GPU architecture was reduced. This also mitigated several architecture compatibility issues with most of the hardware being ARM chipset compatible, and the Jettson available to us being ARCH chipset.

Several image processing libraries were considered, however the accessibility and open-source nature of OpenCV, plus the amount of resources available online, meant that this would be the most efficient library to develop with. Integrations between the ROS network and the camera, and the ROS network and our algorithm were ultimately easier to develop as a result of this

decision -although there are more powerful image processing libraries available.



The system receives an input image from the camera, and ultimately outputs a vector of points that is processed and sent to the PointCloud2. The RGB image (figure 1) is converted to grayscale and low-pass filtered to accentuate the important features (white lanes). Then a BW binarization with an adaptive threshold is applied to create 0 or 1 binaries of every image pixel. This is passed through some morphological processing (figure 2), and finally through a connected components analysis (figure 3) to remove some of the visible noise before being passed to the Birdseye view node. This does the perspective transformation and metric image conversion using the camera's intrinsic and extrinsic matrices. With the transformed image, lane tracking is implemented to collect lane points into a vector<point>, which gets sent to the PointCloud2 for publishing (figure 4). In figure 3, the noise is filtered out and after the perspective transform is completed, figure 4 is what will be sent to the PointCloud2.

Figure 1: Input Image

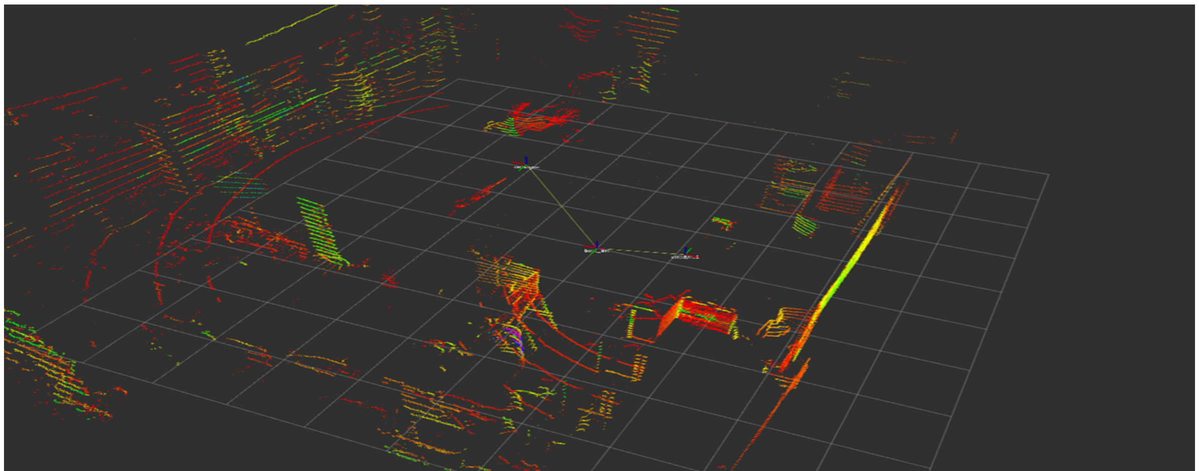
Figure 2: Image before filtering is complete

Figure 3: Image after filtering is completed

Figure 4: Perspective to PointCloud2

Lidar – Sign Detection, Obstacle Detection, Parking Lines

The sensor managing obstacle detection is a VLP-16 Velodyne LiDAR. The LiDAR is mounted on the passenger side of Polaris at approximately side mirror height. Due to this positioning, it emits lasers that reflect off objects in a 270º horizontal (instead of the default 360º) by 30º vertical view with 16 vertical scanning planes at a maximum range of 30 m. Coordinates are collected from each point of a visible surface, thus computing the distance to a point with ease. This aids the team in completing the obstacle detection process. When there is a barrier in the



vehicle's route, the lidar alerts team members via code, therefore collecting information to avoid the object. This LiDAR is very efficient at capturing obstacle data accurately for the purpose of developing maps and navigating unknown environments.

A second Velodyne LiDAR was installed on the rear of the vehicle to allow for a 3-D map generation of the car's surroundings. A parent-to-child connection was created between both LiDARs by using the center of gravity of the vehicle. Accurate visualization was created by applying a transformation to each LiDARs. The last steps in the process are to adjust the rotation and verify that the frame of view (FOV) does not overlap. A screen capture of the lidar data shown in rviz, a 3D visualization tool for ROS, is shown in figure 5.

Figure 5 - Lidar data shown in rviz

Localization

The NineBits Bot determines its location using Kalman-filtered GPS and IMU data. The GPS with integrated IMU, PowerPak 7D-E1 Novatel receiver, is coupled with two VEXXIS GNSS-502 Dual Band Antennas. This setup is SPAN-configured, and allows localization within 4-40 cm. A Sparton AHRS-8 IMU is also used for velocity, roll, and magnetic heading. An RTK node in ROS accepts the input for both devices and creates near and far field headings, as well as a filtered IMU heading.

Simulation

The team was able to use a Gazebo simulation world and a 3D model of the Polaris Gem e2, that was constructed by previous years' self-drive groups, to aid in the completion of this project. The simulation has a ground plane that includes lane lines for the 3D car model to follow, the lane lines are a combination of straight runs, left turns, right turns, and intersections. The simulation also includes stop signs. The Polaris 3D model also includes several sensors as part of the model, lidar units and cameras. These additional models allow for the simulation to be closer to the real world vehicle. The simulation allowed the team to verify that the Robot Operating System (ROS) published messages were being received correctly and to know how the vehicle would behave. Figure 6 shows several screen captures of the Gazebo simulation running through several scenarios that the real world vehicle may encounter.

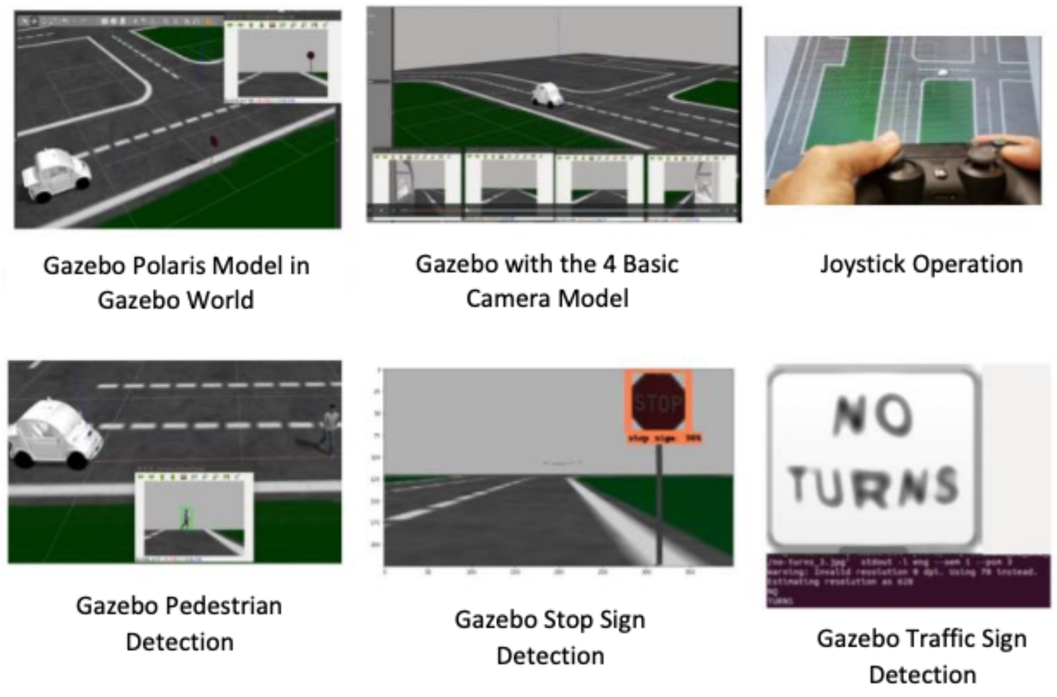

Gazebo Polaris Model in Gazebo World
Gazebo with the 4 Basic Camera Model
Joystick Operation
Gazebo Pedestrian Detection
Gazebo Stop Sign Detection
Gazebo Traffic Sign Detection
STOP
NO TURNS

Figure 6 - Screen shots of the Gazebo Simulation

Mechanical, Electronic and Power Design

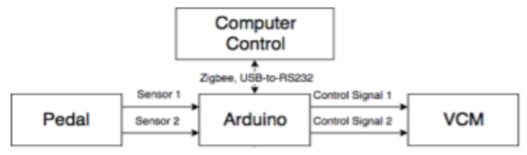
Power

The Polaris is a commercially available electric vehicle. The Polaris' electric motors are directly controlled with a 6.5HP Sevcon 450A controller, which is powered by 48 volts produced by a series of 6V batteries. The 48V battery packs are 6kW and capable of giving around a 30-mile range. The charger is capable of 1kW. The autonomous hardware is powered from two 48V to 12V DC-DC converters. All of the 12V power is fused and contained within the cabin of the vehicle. A Pure Sine Wave 2500W 120VAC converter provides AC power to systems that require it, these include are not limited to, computing devices running the Robot Operating System (ROS) nodes and a LCD monitor

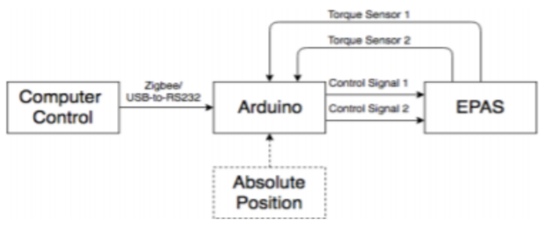
Vehicle

The Polaris is converted to drive-by-wire for throttle, braking, and steering. The suspension of the vehicle is the stock suspension that comes with the Polaris Gem e2. There are three microcontrollers that interface into ROS for the control of the vehicle. The first is the vehicle control unit (VCU), the second is for encoder capture, and the third is for communication with the remote control.

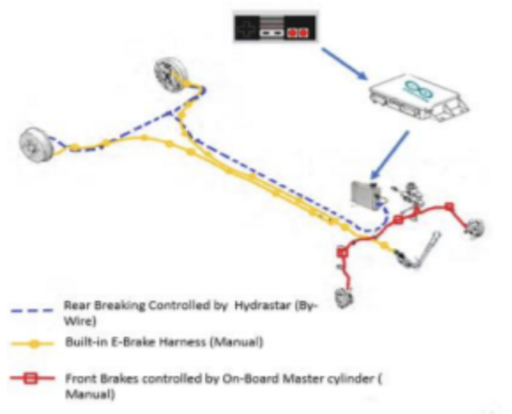
The first microcontroller is an Arduino Mega2560, which is the vehicle control unit (VCU). The VCU controls the throttle, steering, gear, and brake of the vehicle, additionally the VCU monitors the emergency stops on the vehicle to communicate to ROS when those have been pressed. The VCU only manipulates control signals when the vehicle is put into computer control for safety reasons; and when the vehicle is not in computer control the original factory connections are made. The analog control from the Arduino is done using a 10-bit digital-to-analog converter, LTC2465. The autonomous steering for the Polaris is done by manipulating the torque sensors in the Electronic Power Assisted Steering (EPAS) System, this is done using the first Arduino. A dual channel absolute encoder is attached to the steering column of the vehicle to allow for steering effort to be changed under computer computer control. The dual channel absolute encoder consists of two potentiometers that are opposite each other and give inverted readings of each other. Dual encoders are implemented to add an additional safety feature to ensure that the steering position is being read correctly. The position of the steering column is controlled by a proportional controller programmed into the Arduino. The braking system is also controlled with the first Arduino. The braking system is split between the front and rear brakes. The front brakes are under user control at all times, for safety, and the rear brakes are activated by the emergency brake or the Hydrastar System. The Hydrastar system controls the autonomous braking and is connected to the VCU via an H-Bridge module, During an emergency stop the VCU incrementally increases over 0.5 seconds the Hydrastar braking until 100% brake signal is achieved. The control for the throttle, the steer-by-wire system, the braking system, and the Hydrastar system are depicted in figure 7.



Wiring Diagram for Throttle Module



Wiring Diagram for Steer-by-Wire System


Rear Breaking Controlled by Hydrastar (By-Wire)
Built-in E-Brake Harness (Manual)
Front Brakes controlled by On-Board Master cylinder (Manual)

Wiring and Hydraulic Line Diagram for Brake-by-Wire

Hydrastar Module

Figure 7 - Hydrastar Braking System

The second microcontroller is programmed to decode the outputs from two through bore incremental encoders, Encoder Products Company (776-B-S-1024-QOC-C-P/6-A-N-N), that are installed on each of the drive shafts of the Polar-RS. The output from the two incremental encoders are also being captured by the third ATMega2560 and sent serially to ROS to be published as a topic, this also supplements the SPAN solution from the Novatel system. The sensors can also be utilized to generate odometry data. The remote-control system consists of two Arduino Unos that have Zigbee wireless communication. There is also a user node that transmits the desired mode of the vehicle and a vehicle node that receives commands from the user node and communicates it to ROS as a published node. The remote control allows the vehicle control to be operated in manual drive mode, joystick mode, and autonomous mode. The VCU Arduino also controls the operation of the transmission through a set of 3 relays. One rela, the enable relay, switches between the dash forward and reverse rocker switch and the arduino controlling the gear shift. The other two relays control direction when under computer control, one for forward and the other for reverse. The relays are only enabled one at a time, however if both relays are switched on at the same time it is considered an error condition and the vehicle is put into neutral.

Computation

Sufficient computing power is need for Polar-RS due to the wide array of sensors utilized, therefore there are four computing systems used in the NineBits Bot: three general processing computers, and one NVIDIA Jetson AGX Xavier

The first general computer is responsible for lane detection and LiDAR processing, the second general processing computer is used for localization, and the third general processing computer will serve as our vehicle control unit as the master to allow for integration. The sign detection neural network runs on the Jetson computer, the live video feed from the Prosilica camera is fed into the Jeston computer to be processed by the sign detection neural network.The lane detection is run on one of the Intel NUC general processors, and due to the use of image processing techniques it ultimately to reduced GPU reliance. Using general Intel NUC in conjunction with the ROS network also allowed for the groups to reuse topics with necessary information as opposed to having redundant publishers.

The Robot Operating System (ROS) is installed on all computational platforms. All of the computing units are connected to a private network inside the vehicle via wireless or wired interfaces and will be synchronized when started.

Software Strategies

Obstacle Detection and Avoidance

The Polaris is currently able to detect and generate coordinates where objects are located utilizing the input data of the LiDAR. This is accomplished by gathering intensity values of real world objects at various distances from the vehicle. A higher intensity value is given to surfaces such as reflective vests and street signs to be distinguishable from the surrounding environment.

Then, a 3-D map is created in Rviz, a visualization tool for ROS applications, with the display type PointCloud2, a collection of N-dimensional points in ROS. In order to have an appropriate output to use during integration, the raw data from the LiDAR must be continuously processed. This is where the use of filtering comes into play. VoxelGrid Filtering is used to gather points approximated with their centroid. Intensity Filter is used to ensure highly reflective surfaces remain in the list of points. Radius Outlier Removal (ROR) keeps regions containing a user-specified number of "neighbors" to remain in the list of points. Statistical Outlier Removal (SOR) removes points who have a distance larger than some standard deviation of the mean distance to the query point. When these filters have been completed for a given environment, we are left with a region of interest (ROI) or the collection of points that are important to a given task. The points that are important to object detection are the highly reflective surfaces. Therefore, the entire environment is filtered from the PointCloud except for any high intensity points.. This does create limitations in our object detection capability. If something important like another vehicle does not have high intensity, it can be filtered out when we want to keep that information.

Lane Detection

The lane detection system had a few notable limitations, as it was designed with the specific constraints of the competition in mind. The first of which being that the window tracking actually doesn't not track the top 10 pixels of the image (as shown in Figure 8), due to potential errors trying to read pixel values that don't exist. Under the speed conditions the vehicle is tested against, this won't be an issue however that could change under different test conditions. Furthermore, there is significant counter-based and proximity based error checking on outlier points to ensure safe lane starting segments and safe tracking practices. In a controlled environment like IGVC, this will have little impact on runtime or execution accuracy. However, this could compound and slow the system should too many processable elements be introduced. Finally, because I am publishing a collection of points as opposed to publishing a polynomial that tracks the curvature of the lane segment, should too many points be totalled for the lane this could also bottleneck the throughput of the system. Under our specific environmental and requirement-based constraints these issues are mitigated, but they could play a larger role should those things change.

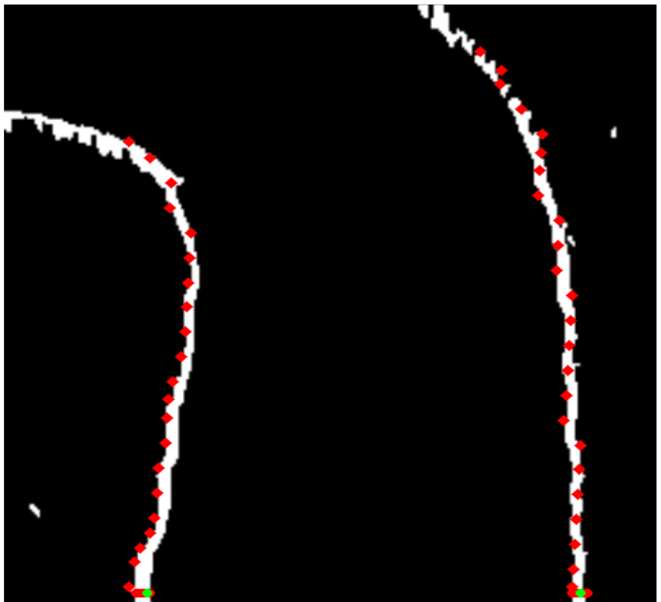


Figure 8

Goal Selection and Path Generation

Path planning involved first gathering data from our lane detection. For lane detection, we are using a lane detection algorithm that is able to detect lines from a camera view. These lane lines are then converted from the image to PointCloud2 data. This PointCloud2 data is then used to be able to determine a goal point between the lane lines for the vehicle to get to.

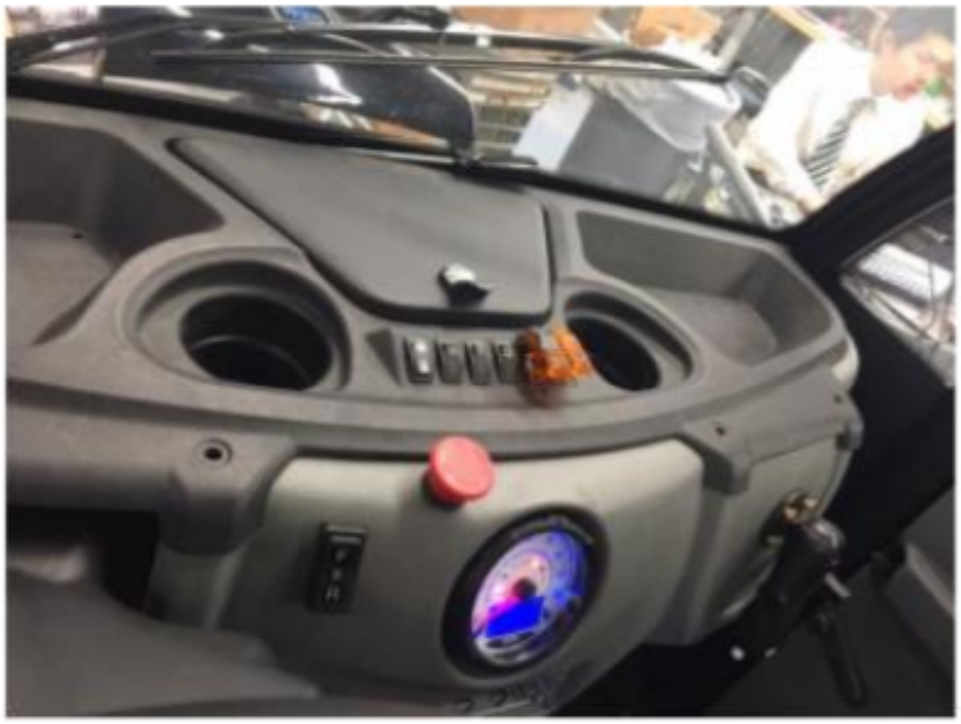
The vehicle dynamics are taken into consideration such as the velocity of the vehicle and the current steering angle to be able to determine the smoothest way to correct the path of the vehicle. For lane keeping, our team can identify the difference in angle between the goal point and the vehicle heading and can use this to appropriately correct the steering angle. The change of the steering angle is proportionate to the speed of the vehicle to ensure the vehicle's jerk is reduced to a minimum.

Map Generation

Localization is a foundational requirement which involves the use of various sensors including a GPS and IMU in tandem with a Kalman filter. These are used for estimating the vehicle's location in space relative to a given origin. This location estimate was produced by instantiating the modules that are built in the Robotics Tool Kit (RTK) for localization. The RTK modules use the GPS, Sparton IMU, and speed from the wheel speed sensors. This allows for the vehicle to position the vehicle for the near field and far field mapping. These maps are able to localize the vehicle to a current location, as well as a goal location as set by the path planner, which uses programmed points to generate a path to eliminate the difference in location between the vehicle and the next point. This involves setting a steering angle based on the curvature of the difference between the points.

Safety Considerations/Failure Modes

Safety is a very important consideration in a vehicle such as this, and there are many areas that need to be taken into account when designing an autonomous electric vehicle. A brief list of areas of concern would include, dead battery or power failure, sensor failure or damage, poor weather conditions, undetectable high speed objects crossing the path of the vehicle. The safety concerns listed can be mitigated to some degree by charging the battery when the vehicle is not in use, be keeping the sensors clean and regularly inspecting for damage, increasing the sensitivity of the sensors during poor weather conditions, and having easy access to emergency stop buttons, E-Stop, for those unexpected emergencies. During competition the vehicle is never run above 5mph, which allows for longer times to react to emergency situations, both inside and outside of the vehicle. The vehicle can be stopped remotely via the wireless E-Stop should the need arise during autonomous driving. There are also E-Stop buttons located on the front, sides, and rear of the vehicle, as well as one located inside the vehicle, see figure 9.



Emergency E-Stop on Dashboard Inside Vehicle

Figure 9

Testing and Validation

The test and validation tasks were distributed amongst the team, we were able to utilize both simulation and real-world testing to complete this project. As mentioned previously in this report, we were able to use a Gazebo simulation of the vehicle in a test course to verify the functionality of the controls, lidar, and vehicle maneuvers, such as breaking, accelerating, lane keeping, lane changing, and turning. Once these models and systems were verified within a simulated environment, the systems were then verified for functionality on the vehicle on a test track located in a campus parking lot. The real world testing would be the final stage to ensure that all the subsystems worked correctly.

Operational to this Date

At the time of submission of this report, our team has completed Localization, lane recognition and tracking - up to the point of metric conversion completion and the publishing of PointCloud2 data, detection of obstacles within a desired distance threshold, and detection of highly reflective surface obstacles to where only the intensity points are retained in a PointCloud.