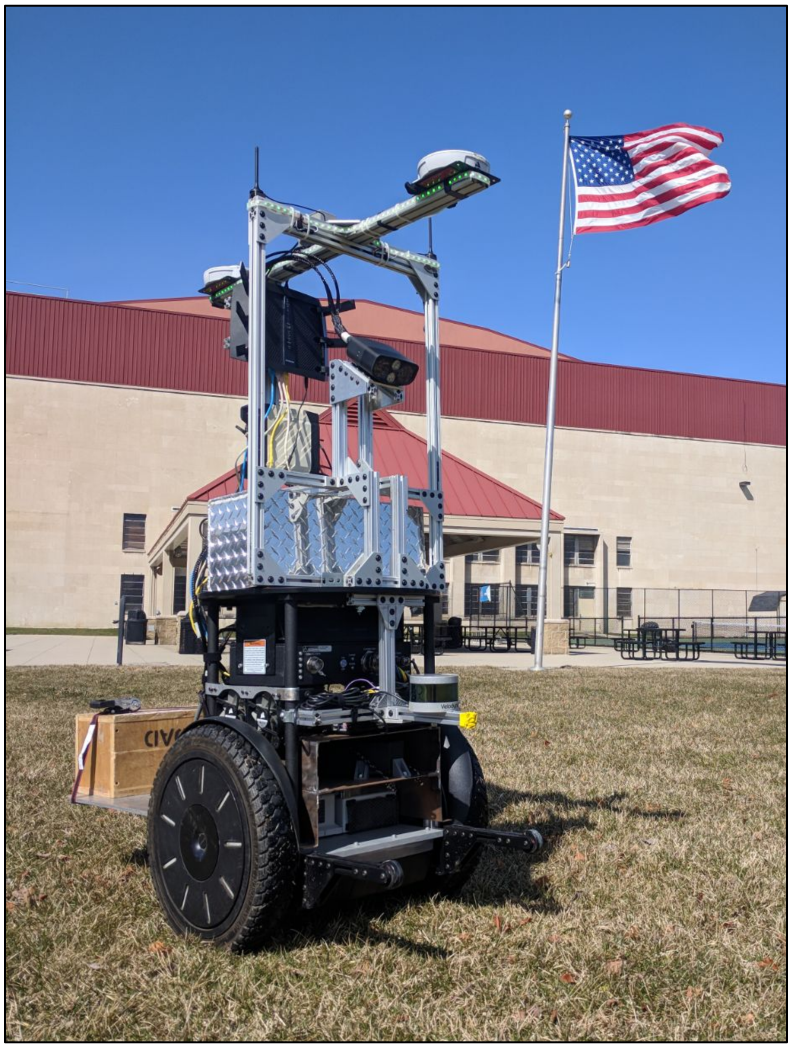
VERTIGO

IGVC 2018 – Design Report

University of Detroit Mercy

UNIVERSITY OF DETROIT MERCY
Build A Boundless Future



Team Captains:

Christine Hillebrand

Marcus Barnett

Team Members:

Ted Chase

Nayan Patel

Isaac Elicea

John Slowik

Ben Kendell

Zach Arnold

Anna Periyappurathu

Joseph Vega

Viken Yeranosian

John Belanger (Volunteer)

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]

Table of Contents

Introduction

The University of Detroit Mercy Senior Design 2018 team is entering the competition with VERTIGO, a successor to Detroit Mercy's 2017 design group's robot Lancebot. VERTIGO sets itself apart from its successor through improved software suite and redesigned hardware components. Modifications have been made to the prior mapping, IP, and goal selection algorithms. In addition, a new redistribution of hardware was made to support easy conversion between Tractor and Balanced modes of VERTIGO's base chassis, Segway RMP 220-v3. The result is more precise localization, maps, and navigation goals. This report highlights the significance of our hardware and software systems that go into the creation of VERTIGO.

Team Organization

The 2018 IGVC team is composed of seven Electrical Engineering students and four Robotics and Mechatronics Students. In addition, four of the seven Electrical Engineering students have a concentration of computer engineering. These diverse areas of study allow for a very versatile and dynamic group with all the skills necessary to make VERTIGO a success.

The work distribution focused on channeling students towards their strengths. Having 11 members in the group gave the team flexibility with the number of people per module. A list of tasks was developed and broken down into modules, then team members were assigned to the tasks and Gantt charts were developed to organize team activities and track progress. The team met twice a week and reported progress though oral reports as well as documented logs on a web-based shared repository. The team facilitator managed the organization of the repository and insured proper submissions in timely manner. A team leader developed agendas for every meeting and reported with the facilitator to the team advisor. The team devoted approximately 12 hours a week for 20 weeks totaling 240 hours during the academic year in addition to 100 hours projected towards the end of the competition. Table 1 lists VERTIGO's team members and their corresponding tasks.

Team Roles and Responsibilities		
Name	**Role**	**Responsibilities**
Christine Hillebrand	Team Leader/ Technical Lead	JAUS/ System Integration
Marcus Barnett	Co-Lead / Facilitator	E-Stop&Controller/Team Organization
Ted Chase	Team Member	IP/ Costmap Integration
Nayan Patel	Team Member	Gimbal/ Camera Hardware
Isaac Elicea	Team Member	JAUS/ NAV integration
John Slowik	Team Member	Odometry/ Gmapping
Ben Kendell	Team Member	UTM Transform/ Goal Planning MUX
Zach Arnold	Team Member	NAV/ Local Planning
John Belanger	Volunteer	Hardware
Anna Periyappurathu	Team Member	LiDAR/ Local Goal Planner
Joeseph Vega	Team Member	IMU/ GPS
Viken Yeranosian	Team Member	GPS/ IMU/ E-Stop Controller

Table 1: Team Roles and Responsibilities

Cost Analysis

The cost of VERTIGO is broken down in Table 2 with an approximate total of $44 K.

Vertigo Team Costs		
Component	Retail Unit Cost	Team Cost
Segway RMP 220 v3 Chassis	$ 24,000.00	$ 24,000.00
Caster Wheel for Tractor mode Capability	$ 247.00	$ 247.00
Auxillary Batteries	$ 729.00	$ 1,458.00
Battery Charger	$ 55.00	$ 55.00
Multisence S7 3D Camera	$ 6,800.00	$ 6,800.00
Velodyne VLP-16 LiDAR Puck	$ 8,000.00	$ 8,000.00
KVH CG-5100 IMU	$ 15,000.00	$ -
ProPak6 Triple-Frequency GNSS Receiver	$ 22,070.00	$ -
Sparton AHRS-8P IMU	$ 1,425.00	$ -
NUC Computer	$ 1,100.00	$ 2,200.00
Mini Box intel Computers (3)	$ 1,000.00	$ -
Router	$ 100.00	$ -
Aluminum framing	$ 400.00	$ 400.00
aluminum sheeting	$ 384.00	$ 384.00
Shelving Unit	$ 400.00	$ -
E-Stop Controller	$ 353.00	$ 353.00
Totals	$ 82,063.00	$ 43,897.00

Table 2: VERTIGO Cost Estimates (blank cells indicate donated items)

Power Budget & Distribution

VERTIGO has two independent power systems: the first is built into the Segway and operates the RMP 220 base platform (motors) and its corresponding built-in computer, and the second is added to the chassis to power all the custom added sensors and computers.

The built-in Segway power system consists of three 72V, 380Wh batteries, two of which power the two Segway motors, and the third powers the onboard computer and motor controllers. The Segway reports a max range of 30 miles on a full charge, and a charging time of about 3 hours. These specs were sufficient to drive the vehicle for a full day under IGVC conditions and rendered no serious challenges to the team.

The second added power system was designed around a quickly swappable 52V, 24A Panasonic GA 18560 battery. This battery provides 1300Wh when fully charged, which can operate VERTIGO's 169W load for about 7.5 hours. With two batteries in-house and a charging time of

4.5 hours, VERTIGO can run for a full day with no power concerns, and a battery can be swapped and fully recharged before the replacement is exhausted.

Panasonic recommends the batteries to be charged to 80% of full capacity to extend their lifetime. This will result in a charging time of three hours and an operation time of six hours per VERTIGO's load. Again, this configuration is sufficient for IGVC conditions and provides plenty of recovery time in case there are gaps in charging.

VERTIGO's power budget was derived by summing the power consumption of all its accessory sensors and computing resources; it is presented in the Table 3.

POWER BUDGET							
		Normal Operating Conditions			Worst Case Operating Conditions		
Device	Quantity	Voltage (V)	Current (A)	Power (W)	Voltage (V)	Current (A)	Power (W)
Velodyne LiDAR	1	12	1	12	18	1.75	31.5
Carnegie Multisense S7	1	24	0.3	7.2	24	0.8	19.2
DVDO G3-Pro Air 3C Pro	1	3	1	3	5	1	5
Netgear NightHawk X6 Wireless Router	1	12	0.55	6.6	12	1	12
Microstrain 3DM-GX2 IMU	1	9	0.09	0.81	1	0.09	0.09
Spartan AHRS-8 Digital Compass	1	5	0.064	0.32	5	0.064	0.32
Mini-Box Computer	2	12	2.5	30	12	3.75	45
Novatel Propak LB Plus GPS	1	12	0.31	3.72	12	0.4	4.8
Indicator LEDs	160	10	0.02	0.2	10	0.06	0.6
Wireless E-Stop	1	12	0.045	0.54	12	0.465	5.58
Total			5.879	64.39		9.379	124.09

Table 3: VERTIGO Power Budget - Normal and Worst Case Operating Conditions

Software Strategy

The team is split up into four groups. The groups are as follows: JAUS, perception, localization, and navigation. The JAUS group worked separately from the group for most of the project. The JAUS module functions on its own without much reliance on the other functionality of VERTIGO. The perception group holds the responsibility of detecting obstacles and lane lines. As time went on, the perception group started working more closely with the navigation group to ensure that the robot does not cross lane lines and avoids obstacles. The localization group holds the responsibility of integrating the GPS, IMUs, and wheel odometry. Each data set is transformed into one coordinate system for uniformity, with Kalman filtering used for sensor fusion. The navigation group holds responsibility for the navigation stack. This consists of mapping, local and global path planning, as well as goal selection, which is an algorithm used to determine where the end point should be for the global path. Having been given some code from last year's team, the team's primary focus was on integration of perception, localization, and navigation. To achieve this goal, the three aforementioned groups were required to work closely together.

Robot Design

VERTIGO is a Segway RMP 220 Chassis capable of a 200lb payload before modifications. The chassis is equipped with reinforced aluminum framing to support our various sensory equipment. The system is run with two Intel Mini-Box Computers using the Ubuntu based ROS platform to run the robot. VERTIGO uses a VLP-16 Velodyne LiDAR, and Multisense 3D Camera for lane line detection and obstacle avoidance. Two GNSS-502 Antennas are mounted on top of the vehicle and feed into the ProPak6 GNSS Receiver. The receiver works with the KVH IMU to give precise heading and global positioning.

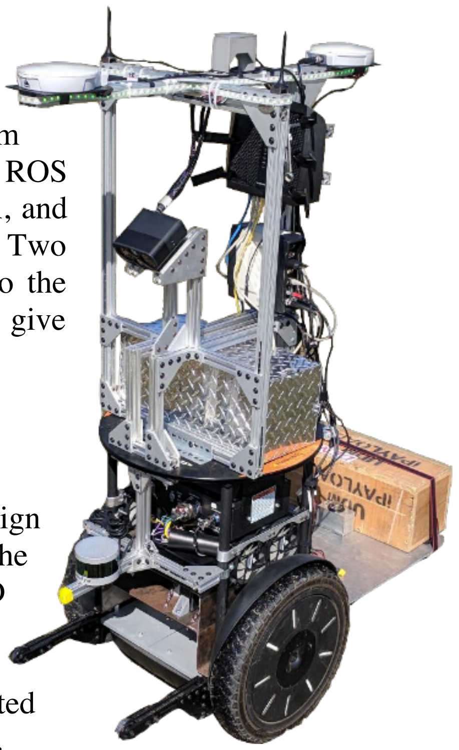


Figure 1: VERTIGO Final Design

Mechanical Improvements

The 2018 team changed some key components to the mechanical design from the 2017 team to make the robot more functional and efficient. The 2017 design included a gimbal, which was configured to stabilize the 3D camera while in balance mode; the team decided to remove it and replace it by an aluminum frame. The gimbal was designed for use with film cameras, which would require some latency of turning. This conflicted with the usage on VERTIGO, skewing the data input of the lane lines. The aluminum frame gives VERTIGO a static camera configuration, simplifying data reception and analysis by comparison with the dynamic configuration of the gimbal.

The team designed and welded a new shelving unit to house the IMU, batteries, and Intel Mini Box Computers. The new design makes the components easily accessible while providing more protection than the previous design. The shelving also creates a static location for the IMU, giving VERTIGO more accuracy in heading and positioning.

Navigation

VERTIGO's software is built on Robot Operating System (ROS), which provides a variety of advantages. Most importantly, ROS is a peer-to-peer networking framework that allows efficient communication between software modules; these modules can be located on different computer platforms, which allows us to distribute computation tasks and increase system speed. ROS also comes equipped with software modules that can be configured to work with different systems; specifically, ROS's navigation stack was used, and some sensors come with ROS-compatible modules. Overall, systems provided with the navigation stack did not work out of the box and needed to be configured in a manner specific to our system.

ROS allows efficient debugging and adjustment of modules, as the topics that modules use to communicate can be monitored, manually fed data, and easily adjusted. Systems such as IOP, image processing, goal selection, and navigation (Movebase) were all developed and debugged modularly, which simplified programming efforts.

ROS provides other advantages suited to our application. We used the ROS Gazebo simulator to test navigation systems before deploying them to the actual robot. We also use the ROS transform library, which allows rapid and efficient transformation of data. This refines the mapping process by combining all sensor data into a common frame of reference, eliminating the possibility of data skew.

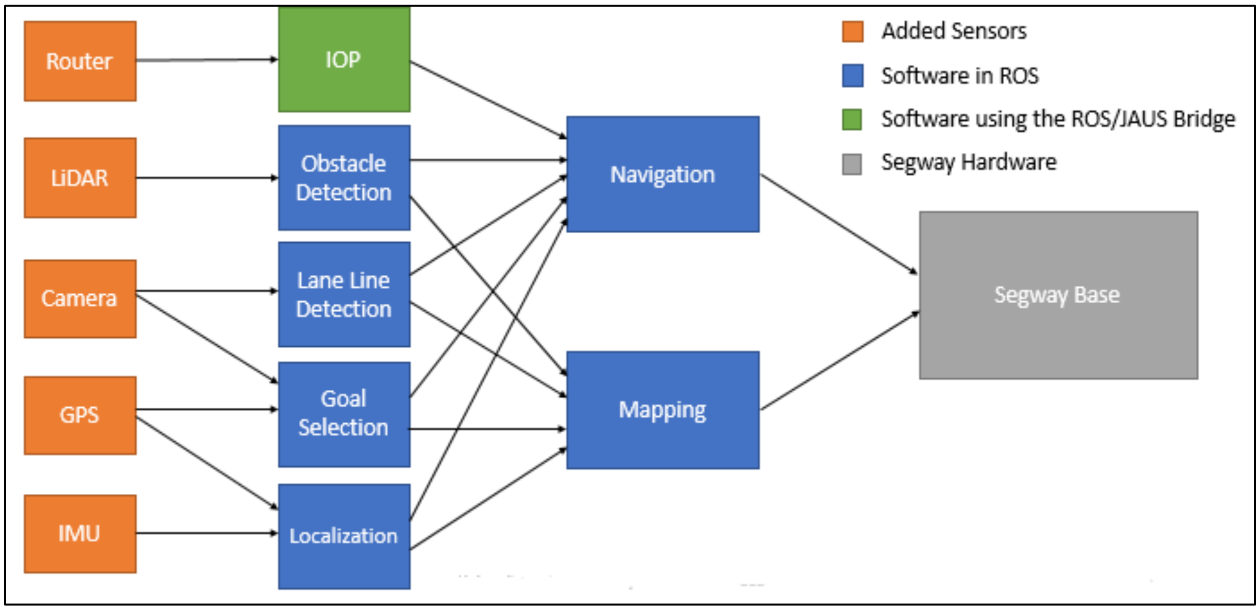


Figure 2: Block Diagram of System Architecture

Odometry & Kalman Filter

VERTIGO's odometry consists of a built-in ROS Kalman filter, which is fed data from a variety of position sensors. The Kalman filter is a mathematical approach of calculating actual data values with respect to positioning of the robot and, in this context, also handles the fusion of these disparate data types (e.g. IMU, GPS, etc). Since the GPS and IMU already have a built-in Kalman filter, we found that one Kalman filter was sufficient to our application.

We also analyzed the use of other tools provided in the ROS navigation stack, including Adjustable Monte Carlo Localization (AMCL) and SLAM Gmapping. AMCL accepts a static map and a LiDAR scan and will attempt to approximate the robot's position within that map. Gmapping is similar but produces a ROS transform between the given static map and the robot's odometry frame within that map. The benefit of using such localization systems is that they reduce the uncertainty in the Kalman filter's final approximation by providing world-referenced position data. We simulated both modules, but ultimately decided to use the Kalman filter to fuse our globally-referenced and dead reckoning sensor data.

Local & Global Planner

VERTIGO uses a program called Movebase, an executable with plugins that influence different behaviors of the robot. Movebase takes in all the filtered sensor data and constructs a map, then navigates that map based on the setting configuration. Those behavior settings affect how VERTIGO will respond to obstacles and recover when no path plan exists.

More specifically, Movebase is a plugin manager that manages the global planner, local planner, costmaps, and recovery behaviors. The global planning stage will keep track of the map as a whole and attempt to select a path between start and end location; in our case, the system uses a simple version of Dijkstra's algorithm, although more efficient navigation methods could have been chosen if this proved too taxing computationally. The path this generates is sent to the local planner which operates on a smaller window in the immediate vicinity of the robot; we used a plugin called Trajectory Rollout, which samples and simulates possible linear and angular velocities of the robot to determine the optimal motor speed of VERTIGO. This system considers obstacle proximity, goal proximity, and global path proximity in making decisions.

The Global Planner can use Dijkstra's algorithm since the environment is tracked as a weighted cost map; this map can be interpreted as a node weighted graph. The algorithm searches the path to find the lowest weighted path to the final destination from the starting point.

Goal Selection

The local and global planner generates the path to the goal given by the goal selection algorithm. The purpose of the goal selection algorithm is to keep the robot from losing its heading and turning around unexpectedly. The robot uses a system given by image processing to map right and left lane lines separately in a costmap. With this information, VERTIGO knows how to keep its heading in the forward direction by comparing the history of the right/left lane line map to the instantaneous image processing data. The goal selection algorithm toggles between image processing goals (local goals that are generated in the presence of lane lines) and GPS goals (provided in the Auto-Nav and IOP challenges).

LiDAR

VERTIGO uses a VLP-16 Velodyne LiDAR Puck equipped with a 360-degree horizontal field of view (FOV). The LiDAR on VERTIGO is configured for a 270-degree FOV horizontally with a 30-degree vertical FOV. Raw data from the LiDAR comes as a 3D point-cloud. The 3D point-cloud is reduced to a 2D laser scan primarily to reduce computation. The data runs through two filters during this reduction; a voxel grid filter and a radius outlier filter. The radius outlier filter only passes points with a given number of neighbors in a threshold distance. The voxel grid filter applies a 3D grid of cubes measuring 343 cubic centimeters (7cm x 7cm x 7cm) over the input data. The data is then downsized by passing the centroid of each voxel cube to the laser scan.

GPS & IMU

VERTIGO uses a Novatel Propak6 GPS receiver coupled with two VEXXIS™ GNSS-502 Dual Band Antennas, making VERTIGO capable of SPAN technology. This technology provides continuous 3D positioning, velocity, and altitude. When the GPS receiver and antennas are running, VERTIGO can localize its position to within 4-30cm. The longitude and latitude are transformed into UTM – Universal Transverse Mercator – to generate a transform for the data, enabling exact positioning.

VERTIGO also uses a KVH CG-5100 IMU for extra certainty in navigation. The IMU collects data on the positioning and heading of the robot, which integrates the data with that of the GPS for use in localization, goal selection, and navigation.

Mapping

VERTIGO tracks its current and past environments using ROS implemented costmaps. These maps are marked using our LiDAR and camera sensory equipment. The sensors are also able to clear obstacles off the map, dependent on the situation. Obstacles identified by the sensors are marked with confidence values, such that VERTIGO can choose to recognize an obstacle as being absent or present.

VERTIGO's LiDAR and camera each have their own individual costmaps. This is done to prevent one sensor from clearing objects that the other sensor sees. We then integrate the two maps together by converting the data types of the individual maps into one recognized universally by the ROS control system.

Image Processing

Image processing is used to detect lane lines. VERTIGO uses a Multisense S7 3D Camera by Carnegie Robotics. The camera provides the distances of each pixel in a 3D point cloud format. The information below explains how the locations of the lane lines are extracted from the raw camera data.

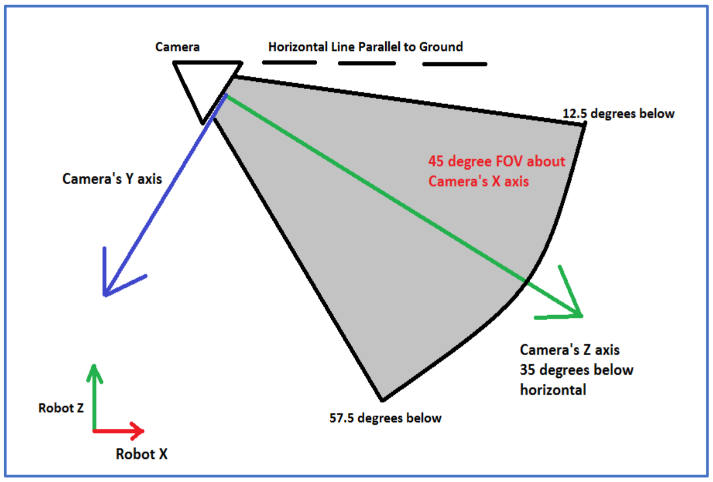

Camera
Horizontal Line Parallel to Ground
12.5 degrees below
45 degree FOV about Camera's X axis
Camera's Y axis
Camera's Z axis 35 degrees below horizontal
Robot Z
Robot X
57.5 degrees below

Figure 3: Detail of Camera Mounting Angle

Ground Plane Extraction

VERTIGO creates a 2-D image of the ground plane from the 3-D point cloud data. Since the camera is mounted at an angle of 35-degrees below horizontal, the ground plane can be extracted by accepting points at a certain distance along the camera's z-axis. Every point in the ground plane forms a triangle with the camera in which the hypotenuse is the camera's Z-axis, the base is the distance from the robot base, and height is the camera height. If the Z-value for a point in front of the robot is less than the expected ground hypotenuse value, then this point is rejected from the ground plane image. We can adjust the threshold for the Z-value cutoff in case there are variations in the height of the ground plane.

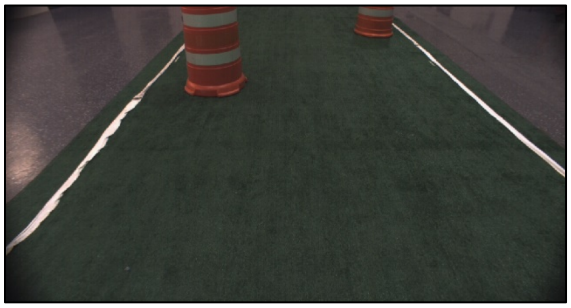


Figure 4: Standard Image

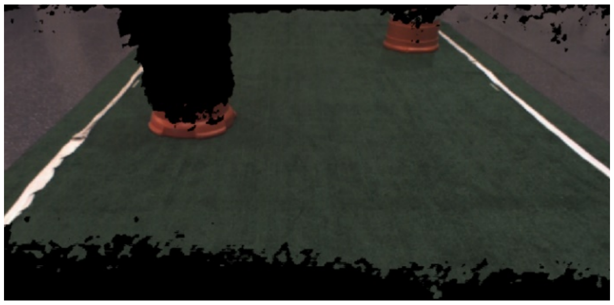


Figure 5: Ground Plane Extracted Image

Metric Image

In the extracted image, the distance per pixel is not uniform due to the optical characteristics of the camera which causes difficulties during the sensor fusion operation with the LiDAR. Hence, a uniform number of meters per pixel in the ground plane image is important to accurately mark the locations of the lane lines and fuse then into the navigation maps.

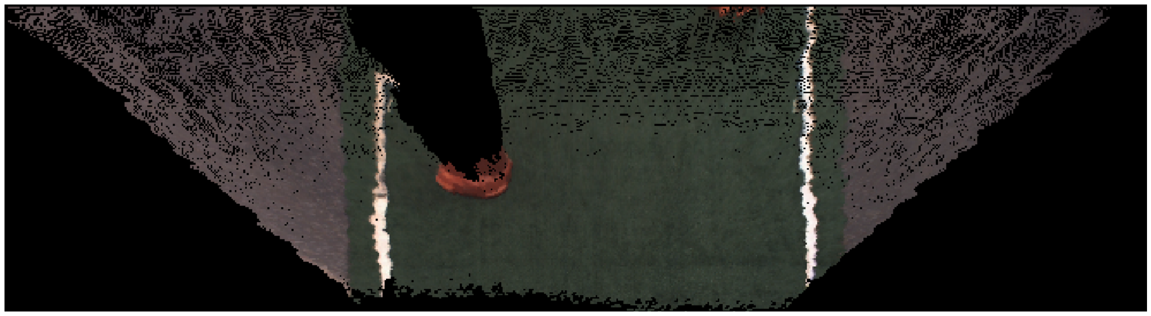


Figure 6: Metric Image Re-shaping

Vision Code

VERTIGO uses vision code to extract a binary (black and white) image from the metric image. This is the final output used for navigation. The vision code uses the openCV C++ library, which has a list of standard image functions and classes. The RGB planes are extracted from the metric image. Because grass typically has a higher red content than blue content, the red plane is subtracted from twice the blue plane. In doing so, the image is kept on a 0-255 gray-scale interval, while creating a stark contrast between the grass and the white lane lines. Intensity thresholding is applied to create the binary image. Canny edge and median blur functions are used to reduce computation for the Hough transform and increase accuracy through elimination of salt and pepper noise. The Hough transform is used last to pass only points that fall into a certain set of lines. These lines are the lane lines, thus giving the final output image.

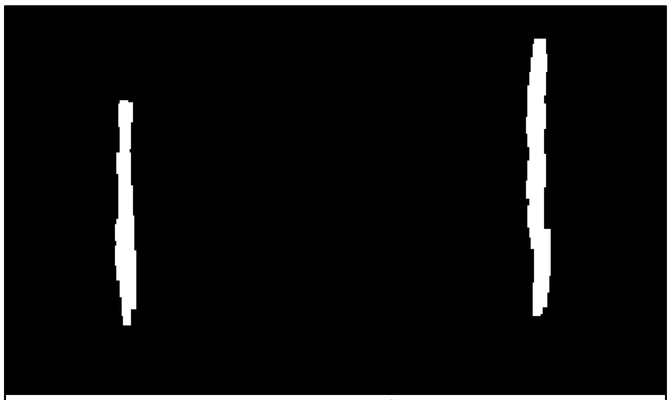


Figure 7: Binary Black/White Image

Mapping Integration

VERTIGO converts the binary image into a laser scan allowing us to mark and clear cost maps. Because the binary image came from the metric image – a geometrically uniform image in meters per pixel – we can derive the location of the lane line pixels in VERTIGO's X-Y coordinate system. The X and Y distances are converted to polar distances to match the laser scan data type. VERTIGO's system adds this layer to the overall cost map and gives authority to mark and clear only on this layer.

To avoid erasing lane lines, the scan is given a variable size based on the locations of lane lines in the binary image. The scan only has authority to mark and clear lines from a region determined by the minimum and maximum angles of the white pixels with respect to the robot, i.e. the areas that are known to be see clearly.

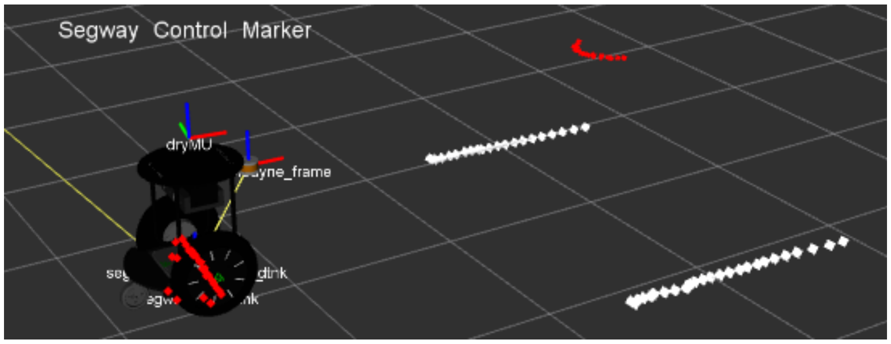


Figure 8: IP Costmap

E-Stop and Controller

The controller has both a joystick for manually controlling VERTIGO, and an E-stop button for stopping the robot in case of emergencies. What makes the controller unique is its versatility, the controller is custom made with multiple mode selections. There are two distinct ways to stop the vehicle when the E-stop button is pressed that are differentiated by a switch: Decelerate-to-zero (DTZ) and Emergency stop. DTZ mode stops the robot without turning it off, while E-stop mode shuts VERTIGO down, making it ideal for extreme emergency situations. There is another mode select for switching VERTIGO between autonomous mode and remote-control (RC) mode. Lastly, there are push buttons for the Segway's balance mode; pushing the correct button allows for VERTIGO to operate with or without its caster wheel attached.

The controller's programming also manages the indicator LEDs, setting them to different colors to indicate connectivity between VERTIGO and the controller, autonomous mode/remote-control mode, and when the robot has been E-stopped.

The controller is designed such that it does not need to be disassembled to access the internal Arduino for reprogramming; this allows limitless modes or functionalities to be edited, added, or removed.

The LED functionality can be reprogrammed to indicate current status, which carries the potential for real-time feedback from the robot. In this way, the team can potentially detect errors while testing or during the competition based on the color and status of the LEDs.

Hardware

VERTIGO uses a custom RC controller based on two Arduino Unos for data processing and control. The Arduino has an operating voltage of 5V with a 16 MHz clock. To cut down on the amount of processing and potential delay resulting from the Arduino's multiple functions, an Arduino Pro-Mini powered by ATmega328 was used to control the LED programs for VERTIGO. This allowed for less than one millisecond delay in processing for both E-stop and joy-stick commands, and less than one millisecond delay for the corresponding LED indicators. Two Xbee Pro SB3 wireless transmitter/receivers create the serial point-to-point network connection to send data between VERTIGO and the wireless controller.

IOP Challenge

One of the communication systems implemented in VERTIGO is the Joint Architecture for Unmanned Systems (JAUS) protocol. VERTIGO uses the 2010 version of the Aerospace standards. These standards specify the unique structure, transportation, and responses for each message and service. The three types of communication that JAUS enables are discovery, navigation commands, and reporting.

For the implementation of JAUS, VERTIGO uses the JAUS Toolset (JTS), an open source software which uses a Graphical User Interface (GUI) input system to autogenerate the code for JAUS functionality. This software relies on JR Middleware to create the network connection. However, JTS is built using a different build tool than the rest of the module systems on VERTIGO. Thus, there is no direct way to interface between the JAUS networking system and ROS.

The ROS/IOP bridge is an open-source tool that, as the name suggests, creates a bridge between an IOP communication node and a ROS robot control system. The bridge allows the JAUS components to access data about the robot's status, position, velocity, and more, to be used in communications with an external controller or other entity.

The system that we have implemented includes only one component, which will contain all the services needed for the competition. Each service is defined in the SAE standards as a finite state machine, which is implemented by the ROS-IOP bridge as a plugin.

Innovations

Hardware

E-Stop Controller

Teams competing in IGVC are required to have a hard E-stop on the base of the robot, as well as a remote-control E-stop that the judges will hold during the competition. This is standard for all teams. VERTIGO and Detroit Mercy took wireless e-stop a step further with a controller that is efficient, versatile, and easy to use. The controller has modes for E-Stop, remote control, balance mode, tractor mode, and DTZ/Emergency Shutdown functionalities. In addition, the controller also commands the LEDs on VERTIGO. Likewise, the software in the controller acts as a fail-safe: VERTIGO will not run without a successful connection with the controller, and if the controller were to disconnect for any reason, the E-stop is immediately activated. Lastly, the controller is easy to reprogram for any additional use the robot might need.

Software

IOP System

The Interoperability Profiles (IOP) system designed for the challenge works as a self-contained software system. It reuses several functions from the main Segway control system by reading from the existing ROS topics, especially those broadcasting sensor and position data. It also incorporates several unique functions, which help to integrate with the robot control system.

The innovation in JAUS is in the way that the system was created. We used several open-source tools, which we modified to meet the specifications of our system. The most significant roadblock was with incorporating the ROS/IOP Bridge software. This software was designed for a different version of ROS and Ubuntu than those being used for the rest of the robot's systems. Therefore, we had to adjust the code to meet our specifications.

Right and Left Lane Differentiator

The right and left lane line differentiator is an algorithm that differentiates the two lane lines. This is important in determining if the robot is driving in the correct direction. The differentiator requires an initial condition of seeing at least one lane line and facing within 90-degrees of the forward direction. The algorithm uses connected component analysis to determine if the bottom of the lane is in the right or left half of the image. This, along with knowing the angle of the line, can determine if a lane line is a right or left lane. The differentiator converts the image to a laser scan and publishes it to a cost map for each lane. Having a cost map for each lane is important in correcting failures in the detection, because the histories of the lanes are known. Ultimately, this algorithm will be used to keep the robot driving forward throughout the course.

Failure Points and Resolutions

Hardware

Segway Electrostatic Battery Discharge

While working on VERTIGO, there were system faults appearing claiming a completely dead battery, manifested by a loud beeping and fault logs. The measured battery voltage proved this false by showing as fully charged. After contacting tech support for the Segway base, a software reset tool was acquired which reset the faults, and if the problem occurs again we will understand it. We have learned to check the battery voltage, as well as the necessary measures to turn off the beeping noises to continue full functionality.

Velodyne LiDAR

The LiDAR has a minimum range of 40 cm. When an object is within 40 cm of the LiDAR, it will be cleared from the map, if clearing privileges are given. This is a problem, because the global planner hugs corners (typically from barrels marked in the map) to minimize the distance of the planned path. When the barrel is within 40 cm of the robot, the barrel will be erased from the map and a new path will be created. Without the object in the map, the path planner will tell the robot to drive through the barrel. One possible solution is to increase the inflation radius of objects in the map to greater than 40 cm. This, however, could block potential paths. A better solution in this situation is to simply remove the clearing privileges from the LiDAR.

With the clearing privileges removed from the LiDAR, the barrels will not be removed from the map, even when the LiDAR is not reporting that it sees them. Removing clearing privileges from a sensor is not typically a proper solution to integration problems between perception and mapping. This is because if the robot sees an obstacle while its localization has a failure, the robot will mark the obstacle in an incorrect location in the map. If clearing is removed, the obstacle will be reported in more than one location in the map, and the problem cannot be fixed. While it is a risky decision to remove clearing from the lidar, it was determined that the shape and size of the objects detected by the lidar would not cause considerable marking to the map with the minimal amount of odometry slippage that VERTIGO has.

Software

Right and Left Lane Differentiator

The lane differentiator has a likelihood of failure when the robot is perpendicular to the lane. A failure here can cause the robot to think that it is facing the opposite direction, making the robot want to face backwards. This can be fixed by comparing the output of the differentiator with the history of the right and left lanes from their individual costmaps.

Testing and Debugging

VERTIGO has gone through rigorous testing and debugging. One of the bonuses of using ROS is that it lends itself to debugging. ROS enables the printing of topics to verify the actual vs. expected values of our modular systems, allowing us to check values in real time and handle deficiencies in code proactively as they appear.

In relation to JAUS, setting up the multicast network for testing proved an area of difficulty. The university public wi-fi network, which we used for initial testing, did not allow multicast communication. We checked to ensure that the router on VERTIGO would allow multicast communication, then we obtained a separate wi-fi router to test in a way that would not monopolize use of the robot.

Conclusion

The innovations and technology that surround VERTIGO are already being widely used in the automotive industry. Our exposure during school gives us an advantage to other students, as it gives us a chance to practice lifelong learning. Lifelong learning is an important component of an engineering education; being able to practice it in school makes us more adaptable to new technologies and processes. In this project we have encountered a variety of new technologies that have forced us to practice this skill. Additionally, to complete the competition tasks we have had to read and follow a variety of technical standards, another important transferrable skill that will follow us as we move on to industry jobs. The automotive industry, along with other industries that employ electrical and robotics engineers, is replete with standards and other sets of technical rules to be followed. By ensuring that we are able to understand and follow these rules, we prove that we can function successfully in a highly regulated industry.