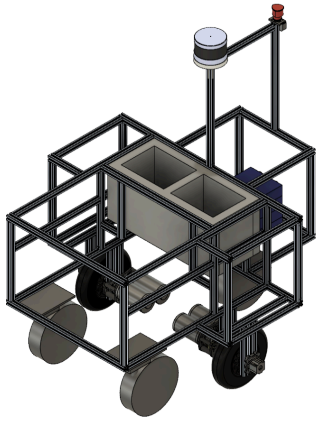
IGVC 2024

Otto

University of Central Florida
Robotics Club of Central Florida



Project Manager
Marcus Simmonds
Marcus.Simmonds@ucf.edu

NAME	EMAIL	ROLE	HOURS
Marcus Simmonds	Marcus.Simmonds@ucf.edu	Software Team Lead	240
Dwight Howard, II	DwightHoward@rccf.club	Electrical Team Lead	300
Hannah Moore	hannahmcleanm@gmail.com	Mech. Team Lead	170
Parker Helms	parkerahelms@gmail.com	Electrical Team Member	40
Mauricio Ferarri	ma350727@ucf.edu	Software Team Member	26
Teja Tiriveedhi	chandrateja.tiriveedhi@ucf.edu	Software Team Member	26
Blake Walker	blakeaw4@gmail.com	Electrical Team Member	100
Cosmo Stellino	dcstellino@gmail.com	Mechanical Team Member	160

Required Faculty Advisor Statement

"I certify that the design and engineering of "Otto" by the 2024 Robotics Club of Central Florida have been significant and equivalent to what might be awarded credit in a senior design course".

Crystal Maraj

Faculty Advisor to the Robotics Club

Table of Contents

1.0 Team Structure & Design Process

1.1 Introduction

Otto is the Robotics Club of Central Florida's 2024 entry to the International Ground Vehicle Competition (IGVC). The Robotics Club is a registered student organization at the University of Central Florida (UCF) that has over 100 undergraduate members. In an attempt to minimize redesign elements of the robot, Otto is heavily based on our 2022 version, with simple improvements and required fixes. The Robotics team that worked on the club's entry to this year's competition is composed of 15 engineering students, starting work on Otto in September of 2023.

1.2 Team Structure and Design

1.2.1 Organization

The Robotics Club Of Central Florida started work on Otto in the Fall 2023 semester with a group of approximately 15 members. The group was organized into three sub-teams, Mechanical, Electrical, and Software. The group members were subdivided into three smaller teams based on their major, skills, and most importantly, interests. Each team included students with a range of technical skills to promote collaboration, better understanding the needs of the two other teams, and a more efficient integration with the other robotics systems. This allowed for an easier and more effective task management system. All teams met at the same day and time weekly, which promoted a more collaborative and productive environment.

1.2.2 Design Process and Constraints

To begin, the Project Manager (PM) generated a set of rough requirements based on the rules of the competition, which were then presented to the group. From there, each sub-team worked to develop their part of the design in parallel, spending the first month of weekly meetings brainstorming and refining their ideas. Team members were encouraged to brainstorm designs while at home and present them at each weekly meeting. After the design phase, the development of systems began. Team Leads delegated tasks to their members to implement their design, with constructive input and deadline reinforcement by the PM.

There were a few major constraints the team was subject to throughout the development and building process. The first and most influential of those is a restriction of funds available to the project. This resulted in a limited development process for the mechanical and electrical teams, as well as a far more cost-conscious design. Many parts and sensors were used that the club already owned which provided a cost-savings, but the drawback of having to build Otto around these pre-existing parts. Table 1 is a rough bill-of-materials for the design as of this writing.

Table 1: Present the materials required for the design of Otto.

Name of Part	Quantity	Cost/unit	Total Cost	Cost to Team
Velodyne VLP-16 LiDAR	1	$4,000	$4,000	$0.00
Intel® RealSense™ Depth Camera D415	1	$272	$272	$272.00
NVIDIA Jetson TX2	1	$499	$499	$0.00
Advanced Navigation Spatial INS Unit	1	$2,900	$2,900	$0.00
VectorNav VN-100 IMU	1	$875	$875	$0.00
VEX Pro Victor SPX	4	$50	$200	$200.00

Esp32 wroom 32ue		$8	$8	$8.00
CUI AMT102-V Capacitive Encoders	2	$24	$48	$0.00
Futaba R617FS (FASST) Radio RX		$80	$80	$0.00
Lithium Iron 60 Ah Battery	4	$85	$340	$0.00
WCP SS Gearbox	2	$181	$362	$0.00
CIM Motors	4	$32	$128	$0.00
1.75 mm PLA 3d Printing Filament, 1Kg	2	$25	$50	$50.00
Miscellaneous Mechanical Parts	N/A	76	$76	$76.00
Miscellaneous Electronic Parts	N/A	$163	$163	$163
Total			$10,001.00	$769.00

2.0 Effective Innovations

Drivetrain Configuration: The gearboxes of Otto are placed in the middle of the chassis on both sides. They are connected to the front wheels through a roller chain. The rear wheels are 4-inch casters. In previous years, the Robotic Club Of Central Florida's entries (e.g., Choo Choo, Bowser, etc.) used a direct drive system of the two front wheels and casters for the back two wheels. While Otto still uses roughly the same system, however, an important innovation was made. There was a choice between a direct drive and a chain-based drivetrain. The team decided to go with the direct drive. This choice was made for the sake of higher reliability and ground clearance. This innovation resulted in the gearboxes being raised into the center of the chassis, thus protecting them from a far larger amount of ground debris. This has the added benefit of increasing ground clearance by not having any parts other than wheels below the bottom of the frame.

Electronics Access: A challenge in the past with previous entries by the club to the IGVC Auto-Nav competition is a hard-to-reach and hard to work on electronics bay. Currently, the electronics bay for Otto is made to be more accessible for maintenance and repair than prior designs. This was accomplished through the use of hinges on the external paneling that allows for quick access to the electronics. Previous designs also had a crossbar over the top of the electronics compartment, which was removed from Otto because of the difficulties this added to the process of work in that compartment. Overall, this innovative design has decreased build time for electronics and allowed for quicker repairs to power and computing equipment.

Serial Communication: In an attempt to simplify serial communication, the team selectively serialized inter-process messages on the single board computer and multiplexed them on a serial port. In addition to simplifying development by exposing most data streams to the microcontroller and abstracting away encoding and decoding, this allowed us to treat our microcontroller as an extension of the single board computer by defining it as a node that produces and consumes data in the same format as the other processes defined on the onboard computer. This change improved the efficiency of our computer to microcontroller communication.

Improved Odometry: Our previous submission fused a visual-inertial odometry system in conjunction with IMU and GPS data to produce an accurate position estimate. Although this was simple to integrate due to our depth camera producing odometry intrinsically, any change in lighting or obstacles too close to the robot would produce large inaccuracies in our local position estimates. To address this, the team developed a wheel odometry system and disabled the camera's 3D odometry estimation. This not only improves the reliability of our localization scheme, but also decreases our USB bandwidth usage, increasing overall serial communication efficiency.

3.0 Mechanical Design

3.1 Mechanical Design Overview

The Design for Otto is based on predetermined sensors and the best location of the payload. The design allows the payload to be placed between the vehicle's wheels to keep the center of mass close to the midpoint. The electronics are placed in front of the payload with the batteries in the rear; this also helps to distribute the weight. The sensors, including the Velodyne LiDAR, are placed where they could give the best vantage point while remaining inside the size constraints.

3.2 Chassis Design

Otto's frame is made of 20x20, 40x40, and 20x60 V-slot aluminum extrusion. These beams were selected because of the easy modularity between the frame and the things that are mounted to it. The theme of modularity is an important part of the design and development process. This feature of the aluminum extrusion is advantageous because it allows quick access, which can help with easy maintenance during competition and the switching of components. Aluminum L-brackets with set screws were used to fasten the aluminum together. These set screws create strong connections without much rigidity. As a result, vibrations are greatly dampened through these connections.

There are three main sections of the frame: the rear battery compartment, the payload holder, and the electronics bay. The payload holder allows a payload of the specified size to rest halfway in the vehicle while still being sufficiently secured from all sides. This, with a combination of padding, will result in minimal vibrations caused by collisions between the frame and payload. The gearboxes are installed directly below the payload due to their central location. The battery compartment is a mostly open rear area that has a foam floor creating a platform for the battery and any electrical components that are not required to be in the main electronics bay. The battery is held in place by plastic tubing. The front electronics bay has a thin sheet of HDPE that is used to support most of the electrical and computing components contained within Otto. The main electronics bay is able to slide out for easy maintenance. Slots are located between the panel and frame for wired connections to pass back to the batteries and motors.

3.3 Drivetrain Design

The three sections of the chassis are built around a two-wheel drive system. As mentioned, the gearboxes are centrally mounted on each side. Each gearbox is made up of 2 CIM motors with a reduction ratio of 12:60 to increase torque to the required amount. Each CIM motor starts with a stall torque of 2.41 N*M. As a result, each gearbox has an expected torque output of 24.1 N*M. Gearboxes were mounted centrally below the rest of the robot. They are directly connected to the wheels. This has the benefit of keeping the center of mass closer to the ground, and an easier mounting system. They are held up by a piece of 20X60 aluminum extrusion and hose clamps. This was a cost-effective way to keep the motors from dipping down and tilting the wheels under their weight. This system involves the use of a Vex VersaBlock V2, which uses the same mounting pattern as the front plate of the gearboxes. These two pieces are screwed together through a section of 20x60 V-slot aluminum which is mounted to the frame.

3.4 Suspension System

For the purpose of suspension, Otto has been designed to reduce vibrations from the ground. The first of the features designed to accomplish this is the aforementioned adjustable joints. These joints provide enough rigidity that the joints do not vibrate however, they are not too rigid that they will transfer all vibration from

the ground to the more sensitive components in the electronics bay and the mounted sensors. This is aided by soft wheels. The wheels of Otto are pneumatic and are inflated to a low-medium pressure which allows for grip and for more cushioning from the ground.

3.5 Sensor Placement

Otto uses a wide array of sensors to analyze the environment and be better able to collect data for the simulation to analyze and interpret. The sensor types present are encoders, LiDAR, a stereoscopic camera, an Inertial Measurement Unit (IMU), and GPS. The encoders are rotary encoders mounted on the ends of the driveshafts and provide accurate information about the rotations and speed of the wheels. They are mounted using a 3D-printed part made of PLA that fastens to the same plate as the gearboxes. The LiDAR is mounted on a mast above the center of the robot and placed at a height that gives it space so that it does not register the frame of the robot. This also gives it the ability to see within a short distance of the robot. For shorter-distance sight, there is a stereoscopic camera mounted to the front of the frame on a fully adjustable mount which clamps tightly to the frame using rubber pads for a reduction in interference from vibrations. The IMU and GPS are mounted in the electronics bay. The GPS is mounted as far as possible from the motors to avoid any magnetic interference.

4.0 Electrical Design

4.1 Overview

The current electrical design of the robot is a continuation of the design found in our previous robot “Choo Choo.” This previous design suffered from deficient electrical connections, causing sporadic behavior when subjected to physical shocks and vibrations found in the competition environment, reducing the robot's reliability. The team improved this design through the use of more robust connector types, as well as replacing assembly techniques found in prototyping with more permanent hardware. Changes include incorporating locking connectors in the design and replacing breadboards with soldered wires or dedicated printed circuit boards to perform the same functions.

However, these were only the beginning of the proposed planned changes. The team recognizes that any electrical connection, even the improved ones incorporated in our current design, present a point of failure, and the reduction of these connections is the best way to improve reliability. A significant amount of time in reworking our design was devoted to the design of a single printed circuit board that combines the functions of several separate devices found in our electrical system into one, significantly reducing the number of connections, resulting in a much more reliable potential design.

4.2 Power Distribution and Usage

Otto uses a 60Ah Lithium Iron Phosphate (LiFe) battery pack. This pack provides a greater run time for the robot because of the battery's greater energy density. With that our calculated runtime is just under two hours of use, this is way more than necessary but it allows the robot to perform full runs of the course and perform testing before needing a recharge (See Figure 1).

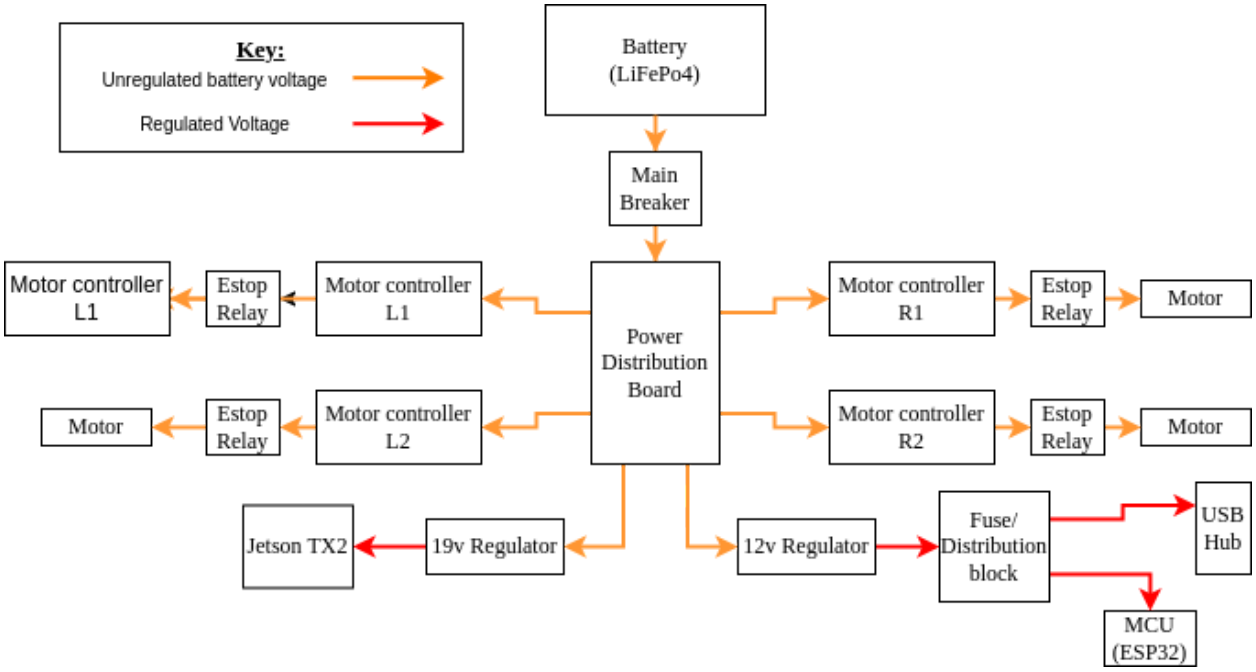


Figure 1: Power distribution diagram.

The pack consists of four 60Ah cells in series to create a 4s 12.8-volt nominal battery. This is a large stability and longevity improvement from previous systems that used lead-acid batteries. LiFe batteries also provide a safer, more stable, alternative to other lithium chemistries along with a closer, more stable voltage for the entire electrical system used on the robot.

Despite the increase in safety afforded by this chemistry, and the excess capacity, the team decided it was necessary to include a Battery Management System into our electrical design. The Dali BMS is connected to each cell in the pack individually to balance out charging between the entire pack, and ensure no individual cell is overcharged or over discharge, reducing cell wear and avoiding potentially dangerous situations involving fire and panicking.

Further increasing safety over our previous design is the inclusion of a much more robust power distribution and fuse block into our design. The specific model we used is legal in and commonly used for the FIRST Robotics Competition, and offers robust electrical connections and several fuse/circuit capacities.

4.3 Electronics Suite Description

4.3.1 Control Board

The main “control board” responsible for computation on our robot is the NVIDIA Jetson TX2, a common single board computer (SBC) used for robotics. The control board is the brain of the robot, and thus needs to be connected to our sensors (described below) as well as our motor controller, lights, and RC receiver via our peripheral board (also described below). The connection of the Jetson to the electronic components of the robot is summarized in figure 2.

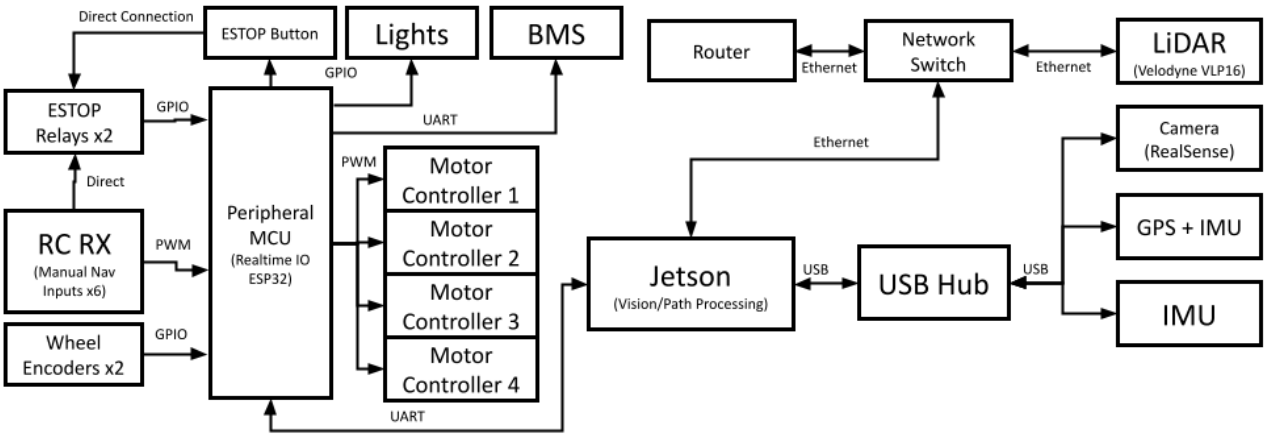


Figure 2: Depiction of the Jetson Electronic Component

4.3.2 Sensors

The **Velodyne VLP-16** is a 360-degree FOV LIDAR sensor, mounted at the top of the "LIDAR mast" for maximum visibility. The VLP-16 generates 3D point clouds of the environment surrounding the robot at 300,000 points/sec, with a resolution of +/- 3cm and measurement range of up to 100m. The output of the VLP-16 is used as one source of obstacle detection for our mapping.

The **Intel RealSense Camera** is a stereo camera that provides both camera images and a 3D point cloud with a max range of 3m and an accuracy of <2% at 2m. The raw camera outputs are used for computer vision detections of lanes and potholes, while the point cloud output is used for general obstacle detection in concert with the VLP-16. The RealSense camera is mounted to the front of the robot, pointed slightly down.

The **Advanced Navigation Spatial GNSS/IMS** is a combination GPS and IMU module that provides both GPS data (in the form of NMEA sentences) as well as orientation, acceleration, and velocity data from the IMU. The output of the Spatial will be used to fuse GPS and positioning data into our localization solution and track the movement of the robot. The antenna for the Spatial is mounted on the exterior of the robot, on the underside of the LIDAR stalk, while the Spatial itself (and the IMU) is mounted central to the robot.

As a second source of inertial data, we're using the **VectorNav VN-100** IMU, which provides orientation and acceleration data. This data will also be fused into our localization solution. The VN-100 is mounted centrally.

For wheel encoders, we're using a pair of CUI Technologies **AMT102-V capacitive encoders**. These are incremental, quadrature encoders that will report rotational velocity of our wheels, used as feedback for our motor controllers as well as odometry information to fuse into our localization. The encoders are mounted to the end of the motor drive shafts, and connected to the peripheral board.

4.3.3 Peripheral Board (MCU)

To interface with the low-level elements of our robot, such as the motor controllers, lights, and RC receiver, we use a paired **ESP32** microcontroller (MCU). The peripheral board is connected to the control board by USB, and communicates via serial protocol. Commands to low-level components travel from control board (Jetson) to the peripheral board (ESP32 MCU), where they're parsed and implemented. This separation of peripheral and control boards was done to increase I/O and as a workaround to software limitations with the NVIDIA Jetson, which cannot run the I/O in real-time in our environment.

4.3.4 Low-level Components

The brushed DC motors on the robot will be controlled by four **VEX Victor SPX** motor controllers. Each motor controller will control an individual motor, and controllers on the same side will receive identical commands. This built-in redundancy ensures that the team can continue operation in the event of a motor controller failing, and allows us to use multiple CIM motors to maximize torque without pulling too much current. The motor controllers will be connected to the peripheral board using PWM.

The RC receiver is a **Futaba R617FS** receiver, using the proprietary FASTT protocol. Despite the proprietary protocol, we've managed to interface the FASTT receiver with the Arduino using PWM inputs. Data from the RC receiver will simultaneously be routed to each motor controller and the Arduino Mega, allowing the paired RC controller to drive the motors without computer intervention, while still passing RC inputs to the software system to control the robot software.

4.4 Safety Devices

The electrical system has the two necessary E-stops. There is one physical button located on a mast on the rear-center of the robot, and one microcontroller relay. These two E-stops are wired in series to provide double protection. The series circuit controls 4 relays, seen in Figure 2, that disable all motor power in the event anything goes wrong. Further, Otto's electrical system was designed to be fault tolerant using auto resetting breakers for the individual motors if they temporarily exceed the fuse rating. This provides safety while still allowing the robot to continue after a short time. All other devices have regular ATC fuses strategically placed. These fuses should never be exceeded and if they are exceeded then all robot functions should stop to resolve the issue.

5.0 Software Strategy and Mapping Techniques

5.1 Overview

The Software team developed the software stack for Otto by leveraging the Robot Operating System (ROS), an extremely popular robotics middleware and SDK used by industry and research teams alike. The system consists of independent software "nodes," connected to each other by ROS message-passing features.

As the "executive" of our system, tasked with high-level behavioral control, the team have implemented a finite state machine (FSM) with the SMACH Python library. The executive makes use of a paired microcontroller, which acts as the interface between the high-level, ROS-enabled software stack and the low-level components of the robot, such as the lights, motor controller, and RC receiver.

Autonomous navigation capabilities such as map generation, environmental modeling, path planning, and obstacle detection/avoidance have been implemented as states in the FSM. Localization of our robot in space is handled via sensor fusion with an Extended Kalman Filter (EKF). The EKF fuses data from multiple positioning sensors onboard Otto to generate a state estimation. Obstacles are detected with a mix of LIDAR data and computer vision detections, which are fused together using the ROS package costmap_2d, to generate our map. This map is used for path planning with the ROS-standard planners, A* for global, and DWA for local. Once we've generated a path, we use the ROS move_base stack to generate a set of base commands (translation/rotational speeds) that get passed to our drivetrain, which will actuate the robot. The ultimate "goal" of our robot will be the end-of-course waypoint, in combination with any other intermediate or parallel goals implemented to support the request.

5.2 Obstacle Detection and Avoidance

5.2.1 Detection

Otto will detect obstacles using a mixture of point cloud data from our VLP-16 LIDAR, RealSense stereo camera, and computer vision detections.

Computer vision techniques were implemented in a python script using the OpenCV python library to detect the lanes and the potholes present in the course. To get a better view of the ground directly in front of the robot, a perspective transform is used to isolate the ground plane in each frame in the video captured by the ZED camera. Once isolated, the image is blurred and converted to a grayscale image to remove noise. Simple thresholding is then applied to the image to isolate any white pixels visible in the frame. A Canny edge algorithm is then used to detect any edges visible in the frame. Once the edges have been extracted, a Hough Line and Hough Circle transform are used to identify any lines and circles made by pixels, which correspond to lane lines and potholes. The identified pixels are then re-transformed to match their locations in the original image. The pixels are located relative to the robot by finding the same pixels in the depth image, calculating the angular difference per pixel given the camera's field of view, then calculating the horizontal distance from the center of the frame using simple trigonometry.

Once an obstacle is detected by LIDAR or CV, the detection is converted to our preferred format and "marked" on our map. For obstacles detected by our LIDAR system, the marking is handled by the costmap_2d package natively, as detailed in section <MAP_GENERATION>. However, for those detections that come from our computer vision system, we have defined a custom costmap_2d plugin (CVDetectionLayer) that takes in individual detections and marks them each on a new "layer" of the map. This newly created map layer is then additively combined with the rest of the layers of the costmap, such that each cell is the sum of all layers. This final costmap is used for tracking obstacles and generating paths to avoid them.

5.2.2 Avoidance

Once obstacles are placed on the map, the path planner and base controller will handle the evasive maneuvers. At this time, there are no purely "reactive" components to our obstacle avoidance: we don't actively search for obstacles using our camera, and we don't preempt the path planner to dodge them with manual commands to our base controller. While this functionality could be implemented given the adaptability of our finite state machine, we've had good results using the strict path-planning approach.

Taking this approach suggests that the obstacle avoidance relies entirely on the ability to correctly mark our costmap so that the path-planner can generate paths around obstacles, which makes it a potential failure point.

5.3 Software Strategy and Path Planning

Our current strategy for implementing autonomous navigation is to heavily leverage off-the-shelf ROS libraries and packages to provide the advanced behaviors for our robot, like path-planning and localization, and then coordinate those behaviors with a custom finite state machine as the "executive." Behaviors that require the actuation of the robot, such as movement commands, are handled by a microcontroller interface to the low-level components of our system. When needed, we route or supply inputs/outputs to the various nodes of the system with custom software nodes.

For low-level commands, the software stack sends commands across a serial connection to the microcontroller, which in turn communicates with our lights, motor controller, RC receiver, etc. This serial connection is encapsulated in a ROS-enabled software node that can turn ROS messages from any process

into commands for the MCU, and any data from the MCU into ROS messages to be consumed by any process.

For path planning, the team is leveraging the ROS Navigation stack and all of its associated path and trajectory planners. The Navigation stack defaults to an A* path planner implementation. Given a goal position, in this case the end-of-course waypoint, as well as a costmap and a localization estimate, we can use the A* planner to generate a coarse path to the goal. As we travel along the course, that coarse path will be refined by local sensor detections, obstacles, updated state estimations, or other changing variables. Along the way, we can place additional waypoints, either dynamically or by hand, that will assist in navigating the course. A useful set of waypoints would be the ramp coordinate pair, provided by the competition.

5.4 Map Generation

To effectively track obstacles and plan our movement in the real world, the team generates costmaps (sometimes called occupancy grids) using the costmap_2d package. costmap_2d implements a layered costmap, wherein each layer is a grid of cells representing a physical space in our environment. We're using three costmap layers:

The first layer tracks 3D obstacle detections from our LIDAR and stereo camera and is implemented natively in costmap_2d as the configurable VoxelCostmapPlugin. It uses point clouds, in the ROS PointCloud2 format, to generate a map where each 2D cell represents a column of voxels containing occupancy information. The voxel columns are projected down into 2D such that a three-dimensional obstacle, like a cone or person, is represented in the map by its footprint.

The second layer of the costmap is our custom CVDetectionLayer, implemented using the plugin interface of costmap_2d. CVDetectionLayer takes "polygons" of input detection points, generated by our computer vision obstacle detection, and places them on the map. Since we're detecting potholes and painted lines with our CV program, that is effectively what this map layer will represent.

Finally, the team uses an inflation layer to create a safe buffer zone around all obstacles. The inflation layer uses the size of our robot to generate additional markings around obstacles intended to ensure that the robot never gets close enough to touch them. As the final layer of the map, this will inflate the results of the 3D obstacle layer and the CV detection layer.

The output of the costmap_2d package will be the local- and global-frame costmaps, representing the sum of all costmap_2d layers, that we will use to track obstacles and plan our movements.

5.5 Goal Selection and Path Generation

Paths will be generated by our trajectory planner, an off-the-shelf ROS implementation of the Dynamic Window Approach, called base_local_planner. The coarse path generated by the path planner in the global frame is augmented by the output of the trajectory planner to generate a smooth path from point to point.

Goal selection will be among a combination of the initially provided waypoints representing start/end of course and start/end of ramp, as well as a list of waypoints derived from the first mapping run of the robot.

5.6 Localization and State Estimation

In order to maintain our environment modeling and navigate autonomously, our robot needs localization information to track its position in space. We generate this localization information through the use of multiple Extended Kalman Filters, implemented with the robot_localization package. The Extended Kalman Filter takes in multiple sources of potentially errorful positioning information, and fuses them into an accurate estimation of our position.

The team is using multiple positioning sources to feed our EKFs: wheel encoder odometry from our drivetrain, IMU orientation/headings from both the Vectornav IMU and Spatial Navigation GPS + IMU

combo, and finally discrete GPS information from our GPS. Using these sources, the robot_localization package will generate local- and global-frame position estimates for use by the rest of our autonomy stack. This information is summarized in figure 3.

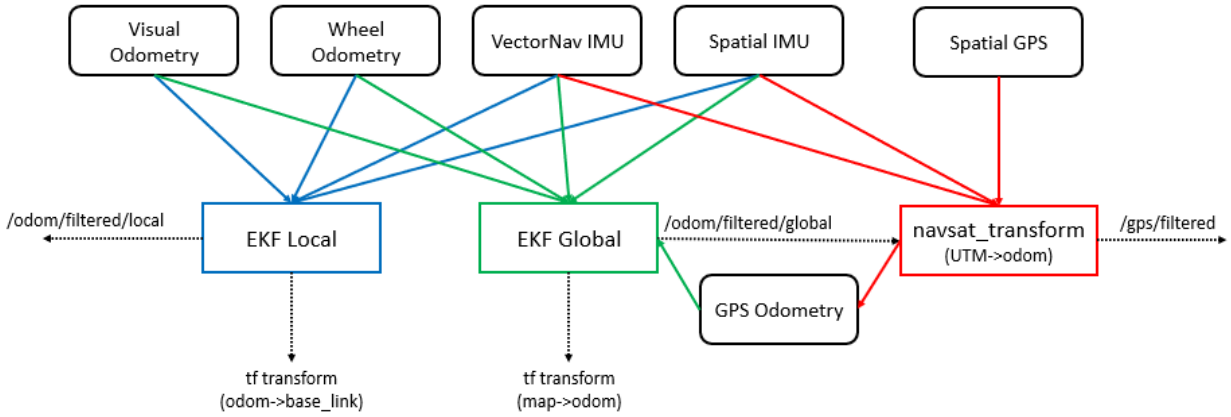


Figure 3: The robot localization plan

6.0 Failure Modes and Procedures

6.1 Vehicle Failure Modes (Software) (See Table 2 below).

There are several scenarios in which Otto's software will reach a state of inoperability. We have designed our executive as well as our map building process with these failures in mind, allowing Otto to immediately stop operation without any potentially catastrophic system failures.

Table 2: Provide the Failure Description and Potential Resolution.

Failure Description	**Resolution**
The CV system detects false positives, or the same obstacle multiple times, potentially flooding the map with false obstacles.	Built-in decay on our custom costmap_2d layers clears old detections over time. Adjusting decay rate can resolve this issue.
Loss of sensor input(s).	Watchdog process checks that all required topics are published at minimum rates. If a data stream is lost, hierarchy determines if we continue without it, or stop and wait for user intervention.
FSM/Executive crash; loss of software control of robot.	Built-in timeout in peripheral MCU will stop any motor control to robot, throw E-stop; RC controller has software E-stop built.
Loss of radio control of robot.	Detected by peripheral MCU as lack of change in PWM inputs; immediately throw software E-stop.

6.2 Vehicle Failure Modes (Electrical)

Electrically there are many possible failure points that are strategically placed points. Most points being fuses that once blown will prevent the robot from operating, or failed electrical connections between devices. These points are a part of the electrical design because of a lack of team knowledge, the team has never worked on a from scratch electrical design. The team took a safe approach to the overall power distribution design and when sufficient knowledge was not within reach the team stuck to basic protection of the systems.

The motion system of Otto can avoid complete failure and loss of power, if any one controller or two motors were to fail the system would still be able to run. Each side of the drivetrain is powered by two motors, each being able to provide enough power to the wheels to continue running. As well as having each of the two controllers be mirrored across the entire drivetrain, if a controller fails one motor on each side will continue to run.

6.3 Vehicle Failure Modes (Mechanical)

Mechanically there are a small number of possible failure points. The first of which is the joints of the frame. These joints are made to be sturdy and to stay in place, however, due to their screw-in design it is possible for them to come loose. While the chance of this is small Otto is designed to accommodate this possibility by using multiple joints per connection point when possible for all structural and other load-bearing joints. This as well as preventative actions such as the use of Lock-Tite on important connections are meant to make the chances of any sort of structural failure close to impossible.

Another major failure point is the master links in the chains. Each chain has one master link which is secured with two pieces making them less vulnerable to failure. A failure of the chain may result in parts of the chain moving at high speed. This is one of the reasons for using corrugated plastic as external paneling for Otto. Corrugated plastic is a soft material but this will result in better impact absorption for any moving parts of the chain or drivetrain, thus adding to the overall safety of the system by containing any damage. In the case of a failure of the chains, Otto will be disabled on that side until repairs are able to be made, but the damage will be contained.

Weatherproofing is an important element of an outdoor robot, especially one carrying sensitive computers, sensors, and other electronics. That is why multiple layers of protection are used for weatherproofing. For the large scale, corrugated plastic sheets are used, however, to mount these some holes had to be cut into them. This creates a need for a second layer of protection which is rubber weatherstrips and caulk for sealing. With the combination of these three materials, the top, and sides of the robot are highly resistant to rainwater. However, it is possible some weatherproofing could fail, and water from rain or puddles could enter Otto. This is not an issue for any compartments other than the electronics bay which has two layers of corrugated plastic for higher certainty against any failures. In the case of a failure of weather-proofing in the electronics bay all power and computing components can be replaced quickly.

6.4 Testing

As of this writing, the software failure modes are being actively developed and tested on our simulation platform (detailed in section 7.0). The team plans to fully test these in simulation before attempting to recreate the failures on the real robot to avoid last minute catastrophes.

Electrical failure modes have been tested throughout construction: fuses and breakers are tested every time we make a configuration change to ensure that they can handle current to values that we can reasonably expect. Further, we've ensured that the robot's motors can be driven with only one set of motors active, and also with only one motor controller active.

To date, testing for mechanical failure resolutions has not occurred, because the focus is on last-minute reconfiguration of our robots mechanical design.

6.5 Vehicle Safety design

The primary safety concepts we worked towards for this robot were redundancy and electrical safety. At IGVC 2019, our team had difficulties due to damaged parts that couldn't be hot-swapped and significant electrical issues. Currently, Otto has redundancy built into many of its critical systems: two sets of identical motors, where any one set can drive the robot by itself; two motor controllers receiving the same data; multiple sources of each data input to minimize risk of "going blind;" and more. As for electrical issues, we took great care to use fuses and breakers on nearly every electrical connection, as well as over-speccing wire gauges and fuse values to minimize potential mistakes.

7.0 Simulations Employed

7.1 Simulations In Virtual Environment

We simulate our vehicle in Gazebo, a full-physics, ROS-integrated 3D simulator. The native ROS integration in Gazebo means that the software stack running on the simulated and real robots are near-identical, minus some input/output differences. The Gazebo simulation shown in figure 4.

The vehicle is described via a Universal Robotics Description Format (URDF) file, the same filetype used by ROS to manage vehicle geometries, and then augmented by additional macros and plugin definitions to enable the Gazebo simulation. By using the same URDF for the simulated and real robot, we ensure updates to the mechanical design are immediately represented in the simulation. Sensor plugins generate sensor outputs that match actual sensor data formats, meaning we can fully test the software stack in simulation.

In addition to modeling the robot and its sensors, we make use of multiple custom Gazebo "worlds" that imitate the IGVC course, qualification scenario, and multiple obstacle scenarios to test robot behavior in each situation as fully as possible.

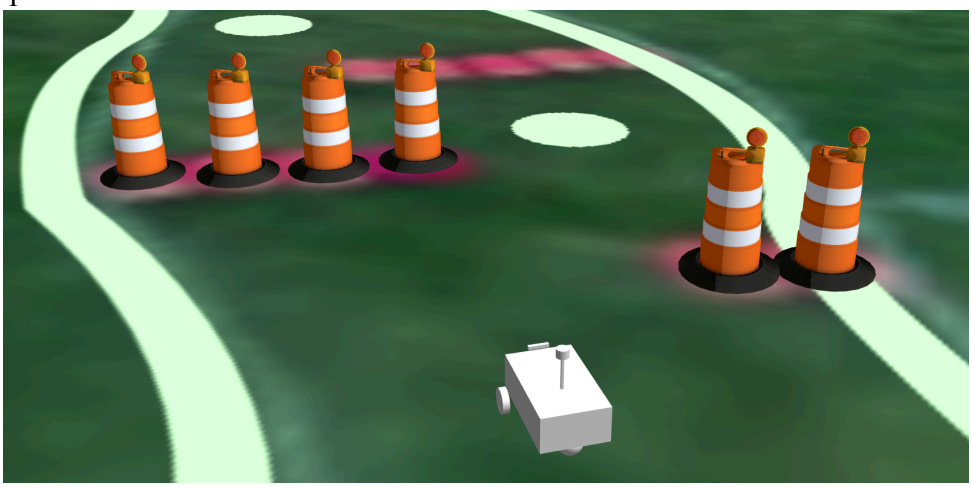


Figure 4: Full-scale auto-nav course simulation

8.0 Performance Testing to Date

8.1 Component Testing

To date all major mechanical parts have been tested in a number of ways. Each motor has been tested independently to verify no issues with the motors before assembly, and to ensure no excessive current draw occurs.

All 3d printed parts were tested before being used. The level of stress each part was subjected to in testing depends on how structural, or vital the part is in the performance of Otto. For example, all of the 3d printed hubs were designed to have high strength and were also tested by putting a shaft through each set of hubs and applying approximately 100 pounds to each side of the shaft. None of the Hubs showed any signs of damage from these tests. Sensor mounts were tested by applying force to the part while it is mounted to ensure no failures would occur when in normal use.

A small batch of L-brackets from the same order as those used for fastening the chassis together was tested by installing them in two pieces of aluminum extrusion and testing the amount of force that would be required to compromise one of these parts. It was found that each bracket failed only at forces far beyond what any one bracket may experience in normal use.

Each sensor onboard Otto has been independently tested and verified to output data that is both correct and useful. To test the sensors, we connected them to the control board and verified the output of the sensor using the ROS data visualization tool, RVIZ. We also used real-world sensor data to test the fidelity of the simulation with the rationale that the more accurate our simulated outputs are to the actual outputs, the more likely we are to be able to trust the simulation for development.

8.2 System and Subsystem Testing

To date mechanical subsystems have been tested by running the gearboxes while Otto is raised from the ground on blocks. The motor controllers were plugged into an external computer which allowed for the reading of current drawn from the motors. Each motor was found to be pulling about five amps when spinning at maximum R.P.M. with no load. This has been reduced from seven through optimizing alignments between sprockets, and lubrication of moving parts.

The software stack for our robot is currently under extensive testing in our simulation environments as we iron out last-minute kinks and race to the finish on our intended functionality. The basic FSM and localization approaches have been tested relatively fully by manipulation of the simulation environment and removing/sensor inputs for stress-testing. When the robot is functional, the team is confident that this testing will pay off by reducing the overall time to a “fully working software system.”

9.0 Initial Performance Assessments

9.1 Software Performance

As of the time of this writing, Otto’s localization, mapping, and obstacle avoidance performance has only been assessed with a visual odometry system as a stand-in for our wheel odometry system. Our mapping system is able to observe and clear obstacles at near real-time speeds (greater than 1hHZ). Our computer vision is capable of detecting a circular white object on the ground that is 3 inches in diameter up to 3 feet away. Our path planner is capable of executing evasive behaviors within 2 seconds of encountering obstacles near it.