IGVC 2015-SCIPIO

EDT-SCIPIO

University of Illinois at Chicago

Daniel Budolak, Mohammad Budwan, Justin Delcourt, Jeff Delia, Krystian Gebis, Patrycja Guza, Kevin Huxhold, Nasiruddin Jamali, Tylar Jozefczyk, Evan Keefe, Matt Kubak, Chris Lee, Jasen Massey, Roy Moran, Tim Mueller-Sim, Andrew Pable, Tim Palermo, Dominick Rauba, Joel Rodriguez, John Sabino, Tyler Stillman, Basheer Subei, Steven Taylor, Matt Wiabel

INTRODUCTION

EDT-Scipio (Scipio) was designed from every angle to be a reliable and stable platform that can be used for years to come. The in-house fabricated gearbox and pulley drivetrain qualify as an innovation and ingenuity, as well as the ease with which the top chassis can be modified to accommodate new sensors, electronics, and payloads. The electrical team has spent their time reducing the size and complexity of the circuits required to operate the system; all while incorporating additional features, such as an energy monitor. A further optimized backplane integrates disparate systems, further reducing the complexity of wire routing by eliminating intermediate interfacing. Numerous modifications to the electrical and mechanical hardware allows for easier access and maintenance. These features have allowed simplification of the design, inherently improving performance and reliability of the unit. Conjointly, software development was continued and expanded upon utilizing the Robot Operating System (ROS). ROS is a dedicated open-source platform for robots that provides a standard communication channel between software nodes that monitor the robot's environment, make plans based on present and past information, and act accordingly by sending commands to the motor controllers.

THE IGVC TEAM

The Chicago Engineering Design Team consists of over 40 undergraduate students, with the IGVC team being a subset consisting of one-third of the total member base. Figure 1 provides a breakdown of affiliation and responsibilities. The team consists of three primary departments: mechanical, electrical, and software; all of which are overseen by the President and Vice-President. In preparation for the 2014 competition, members of EDT invested an estimated 1500 man hours, with the primary focus on software development. This year, over 2000 man-hours have been dedicated to gentrifying many key facets of the aggregate design, such as the full renovation of the electrical systems.

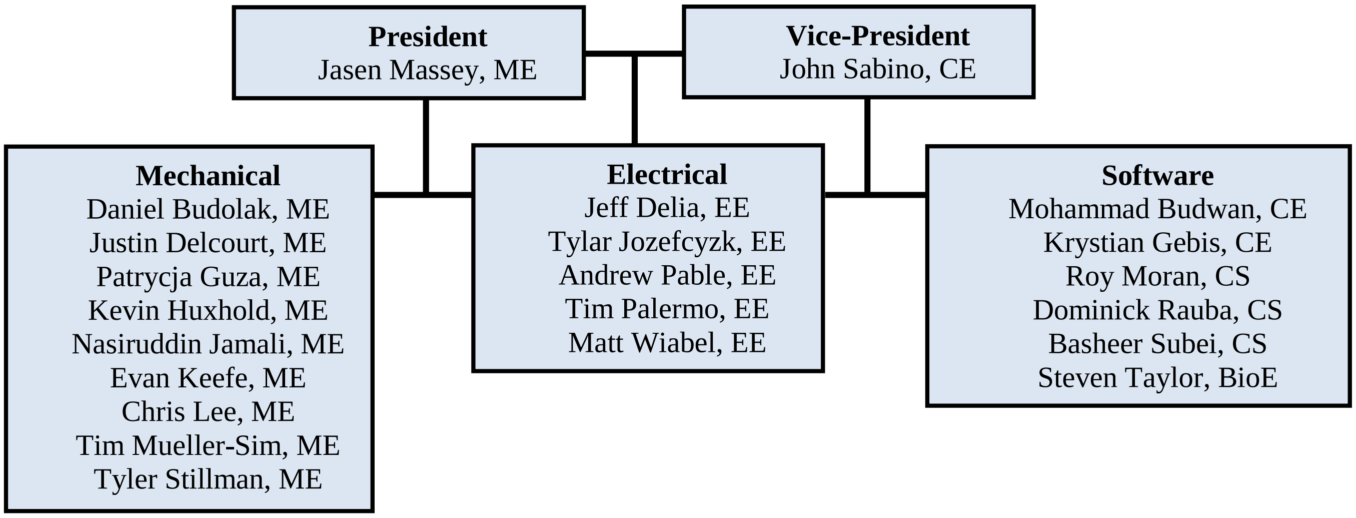
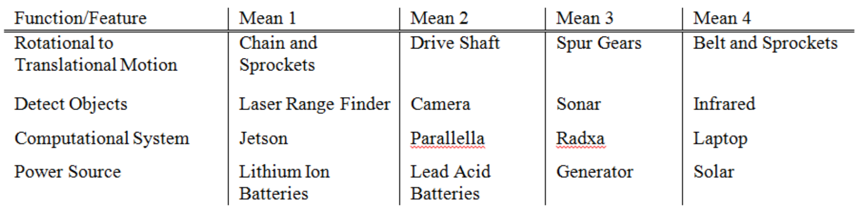


Figure 1. IGVC Team Organization

DESIGN PROCESS

In 2013 the team took on the substantial project of designing and constructing a state of the art autonomous platform. In 2014, the team doubled down on this platform, focusing its efforts on rewriting the software brain and focusing on mechanical and electronic deficits identified during the 2013 competition season.

The design process began by first understanding the design problem, and then formulating design objectives. After the problem was well defined and the objectives were formulated, the constraints and requirements limiting the design were recognized. The constraints included competition rules such as vehicle size, vehicle speed, and safety regulations, as well as also internal constraints such as cost, resources, and manufacturing capabilities. In order to measure how well an objective was met, metrics were developed in order to score different aspects of the design. Metrics pertaining to the higher level objectives were weighed heavier than those pertaining to lower level objectives. The metrics carrying the most weight in Scipio's development were cost and manufacturing capabilities. The metrics were later used in the conceptual design process to compare various design concepts against each other.



Function/Feature	Mean 1	Mean 2	Mean 3	Mean 4
Rotational to Translational Motion	Chain and Sprockets	Drive Shaft	Spur Gears	Belt and Sprockets
Detect Objects	Laser Range Finder	Camera	Sonar	Infrared
Computational System	Jetson	Parallella	Radxa	Laptop
Power Source	Lithium Ion Batteries	Lead Acid Batteries	Generator	Solar

Table 1. Morphological Chart

The conceptual design process began by determining all necessary functions the vehicle needed to perform. Next, all possible means to fulfill the functions were determined and inserted into a morphological chart in order to generate design concepts. A sample of the morphological chart can be seen in Table 1. Concepts were generated from this chart by making various combinations of the means. Many concepts were eliminated because of a failure to satisfy the design constraints. The overall concepts list was then reduced to four top concepts which were further analyzed and compared against each other. Generic drawings and sketches were made for each concept and the overall cost and manufacturability were estimated. A comparison chart was developed and the metrics were used to score the predicted performance of each concept. The method of assigning a score to objectives involving performance was done by researching the concept and making an educated estimation. After compiling the overall scores of each concept, the one that received the highest score was chosen. At this point the departments branched out to work on a detailed design section for their concentrations. The detailed design section consisted of engineering drawings, schematics, CAD models, and a bill of materials. Once the CAD models were completed, dynamic simulations and testing could be performed. Finally, parts were ordered and manufacturing began.

During the manufacturing process, each department performed its own independent tests of components and systems. Once the mechanical and electrical systems were manufactured, integrated testing of all 3 systems was performed. When problems were encountered, improvements were suggested and implemented where required.

MECHANICAL DESIGN

Scipio was designed to be a reliable and stable platform such that the mechanical structure could be re-used in future years. The entire drivetrain is considered a mechanical innovation, as it is different from any previous EDT drivetrain. Scipio is composed of two main sub-assemblies, the top and bottom chassis, which are described in detail below.

Bottom Chassis Assembly

As seen in Figure 2, the bottom chassis is a steel tube frame which houses the drivetrain and its components. The drivetrain is a skid-steer system powered by two 3 HP brushed DC motors operating at 24 volts. A skid-steer system allows Scipio to have a zero-turn radius which is optimal for switchbacks and dead ends. The drivetrain consists of two gearboxes, shown in Figure 3, that were manufactured in house and drive a power transmission belt system. The gearbox and belt system achieve a speed reduction of 12.8 and 2.39 respectively, providing a total speed reduction of 30.63:1. Both of these systems are designed and manufactured within tolerances to achieve efficiencies greater than 93%.

The drivetrain increases Scipio's power and efficiency while reducing the amount of maintenance required. Previous chain-driven robots suffered performance issues and required large amounts of maintenance to compensate for backlash. In particular, chain based drivetrains struggled to turn the robots steadily and accurately. Increasing the speed reduction and establishing the wheel center ratio counteracted the effects of wheel scrub.

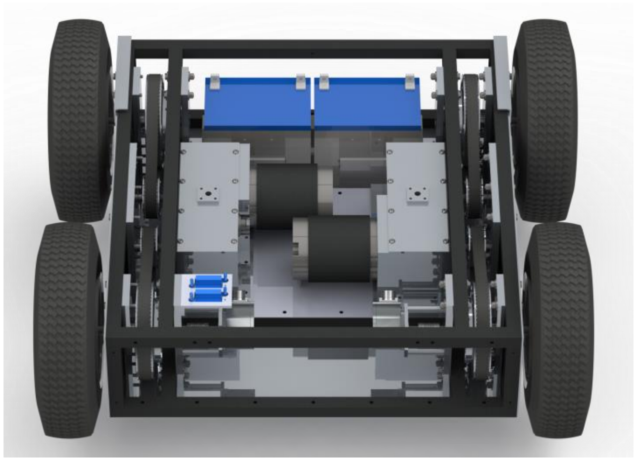


Figure 2. Bottom Chassis Assembly

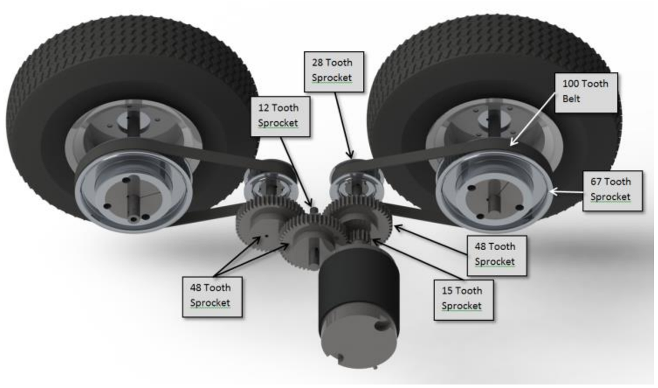


Figure 3. Gearbox Skeleton View

Top Chassis Assembly

The top chassis, shown in Figure 4, is designed to be modular and accessible while maintaining a professional aesthetic. The top chassis stores the logic circuits, payload, and all the sensors. The top chassis is divided into two compartments: the electrical box, and payload cavity. Each area can be accessed by opening the corresponding hinged doors. Two five pound-force gas struts are attached to the inside of the payload cavity door to allow for ease of access to the area without the need to hold it open. There are also compression latches that secure the door in the closed position which are calibrated to counteract all force from the gas struts and vibration. Aluminum T-Slot framing is used as the base structure for the top chassis, allowing for easy assembly and future modifications. Aluminum panels with rubber edge-grip seals are fastened to the T-Slot to provide protection and weatherproofing. The top and bottom chassis are mated with four quick-release pins that provide easy separation for troubleshooting and maintenance.

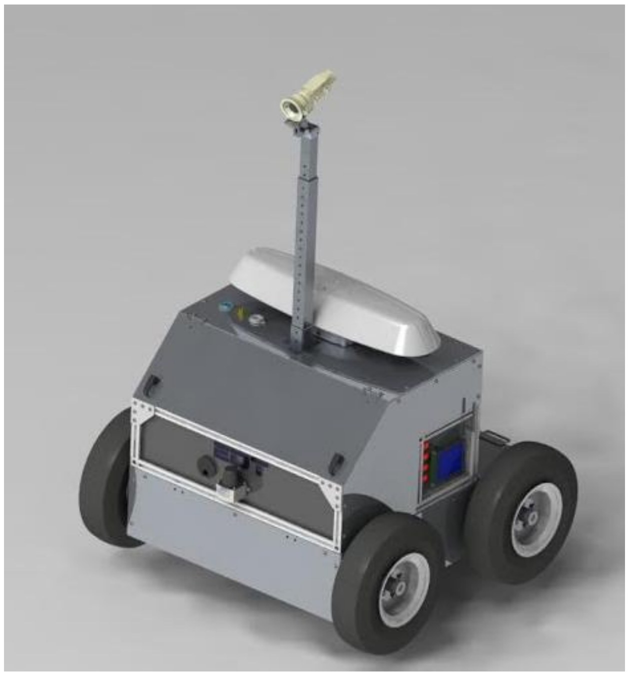


Figure 4. Top Chassis Mated to Bottom Chassis

Innovations

A clevis-hitch trailer was designed and constructed to provide a mobile transportation system to handle this equipment. The design is a simple rectangular frame with a tailgate, similar to that of a pick-up truck. A dual-hitch system is used to prevent jack-knifing, and allow vertical rotation on uneven ground. Free-rotating caster wheels that are used on the trailer allow it to follow the rotation of Scipio.

Designs from previous years required operators to manually disconnect each individual mating between the upper and lower assemblies; requiring precision placement of cables and an extensive amount of time. Scipio’s 2015 design implements blind-mate connectors, shown in figure 5. Once an operator unites the upper and lower assemblies they then connect the two conjoined sixteen prong connectors for a complete electrical connection. The female connector is mounted to the bottom chassis and the male is connected to the wires of the upper assembly. Connection and disconnection of the two sections requires much less effort. Eliminating and rerouting the remaining cables increased the payload volume by 17.8%.

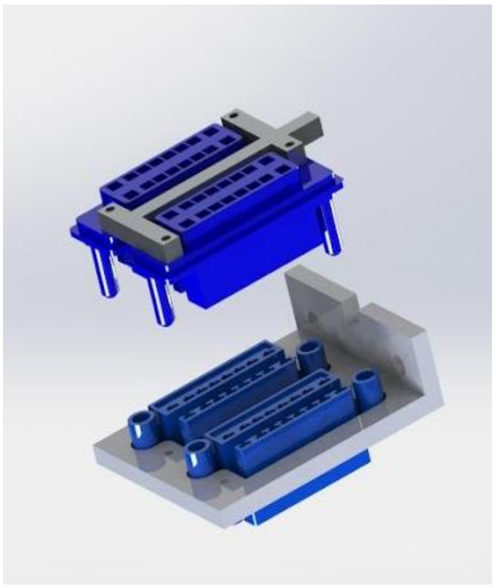


Figure 5. Blind Mate Connectors

To improve upon the weatherproof design of Scipio several exterior panel mounts have been added to isolate the electrical systems from the external environment. To allow for easy access to the on-board computer system, a modular bulkhead has been added to the side panel.

To achieve an optimal field of view with the camera system, a telescoping camera mount has been made. The height ranges from 15.75 to 29.5 inches with an adjustable viewing angle range of 115°. In addition, a camera housing has been manufactured to protect the camera and its sensitive lenses from the environment.

ELECTRICAL DESIGN

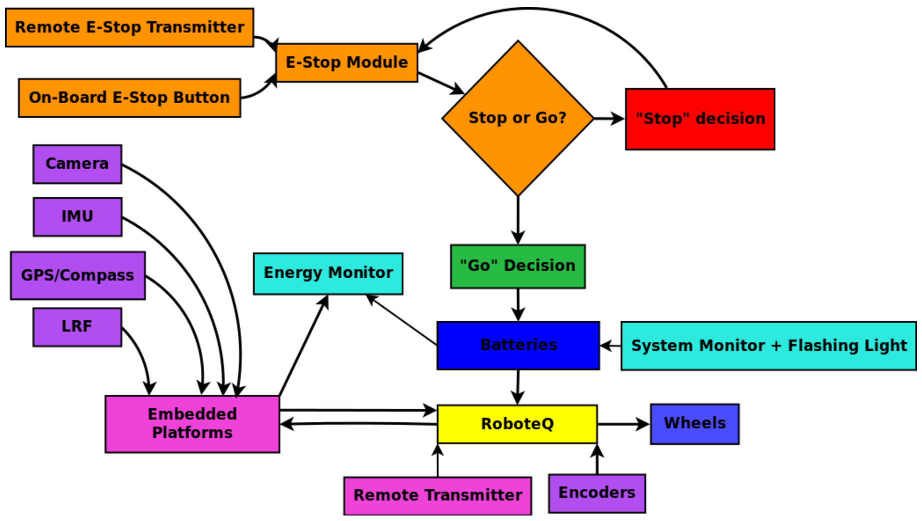


Figure 6. EE Flowchart

This year EDT narrowed its focus to reducing the size and complexity of Scipio's electrical systems in addition to adding a few small features. A new backplane now integrates the systems and eliminates the need for the screw terminal block. These changes allow for simplification of the design, improving performance and reliability. Figure 6 shows a high level overview of the hardware and sensor flow.

Power System

Scipio uses two 12 volt 35 amp-hour sealed lead acid batteries arranged in series, resulting in a 24V nominal system. This configuration yields 90 minutes of drive time and multiple weeks of standby time. Switching regulators are used for improving efficiency and providing 12V, 5V, and 3.3V power lines for sensors and logic circuits. The RoboteQ motor controller receives the full potential of the batteries, and can report useful information to the computer such as battery charge and motor current draw.

Emergency Stop

The Emergency Stop (EStop) system provides the capability to disable Scipio in the event of an emergency. The EStop may be activated wirelessly by remote control or manually by pushing the onboard switch. The system is composed of a wireless handheld transmitter and a receiver in the vehicle. The transmitter unit has a highly visible red pushbutton switch which changes from "GO" command to "STOP" command when depressed. The "GO" signal must be received from both the onboard switch and the wireless transmitter before the vehicle is allowed to move. If no valid signal is received the vehicle will remain stopped. The radio module has an open field range of five miles and uses spread spectrum technology to provide data encryption and prevent interference or jamming from external sources.

The Emergency Stop transmitter has been drastically improved with energy efficiency in mind. The previous design had a maximum battery life of around fifteen minutes with a nine volt battery which proved ineffective for the desired operation time of the robot. To correct this issue several changes were made. The linear regulator has been replaced with a high efficiency switching regulator to reduce energy loss in the voltage conversion. A larger capacity fourteen volt rechargeable battery has been added to provide a wider depletion range that allows the regulator to provide a five volt output. The five mile range was found to be excessive for the given application so the 900 MHz transmitter has been set into a lower power state to reduce the signal strength to under a mile.

Blinker Circuit

The Blinker Circuit receives different signals from the RoboteQ motor controller which identify which mode the robot is in. When in autonomous mode the system flashes high intensity LEDs attached to on the camera mount at 1 Hz. In manual/remote control mode the LEDs stay on to alert bystanders that the robot is powered and in standby mode.

Motor Monitor

The Motor Monitor is a safety centric design that informs bystanders of the robot's direction of travel. It does this by communicating with the RoboteQ motor controller over RS-232 to determine wheel velocities which are displayed as color codes on several high intensity, external LEDs attached to the chassis of the robot. An additional feature the system can also indicate low batteries by flashing the external LEDs in a unique pattern to avoid confusion with the Blinker Circuit. This state is triggered by a GPIO connection to the energy monitor.

System Monitor

The System Monitor is a multi-faceted system. The core purpose is to analyze power consumption of the batteries and display the information on the panel-mounted LCD screen. Additionally, the system monitors and reports temperature data from within the electrical housing of the robot, this allows the user to determine if there are imminent failures in the electrical sub-systems.

To fully utilize the screen size of the LCD, all of the embedded platforms' resource consumptions are reported graphically. This information ranges from individual CPU core consumption to available storage space, but due to limited screen space only so much can be displayed. The bottom button to the left of the LCD screen is used to toggle between the different embedded platform data sets easily. All communications with the embedded platforms are conducted over SPI.

Motor Control

Locomotion of Scipio is controlled by the RoboteQ HDC2450 motor controller. The motor controller allows Scipio to be controlled in closed loop where PID coefficients can be easily adjusted. Velocity commands and measurements are sent to the motor controller through RS-232 communication.

Sensors

Scipio has several sensors that provide feedback to the embedded platforms. Two incremental shaft wheel encoders are coupled directly to the front wheels to measure the exact rotational position and velocity of each side of the vehicle. Each encoder produces two pulse trains with frequencies linearly dependent on wheel speed. The encoder sends the two pulse trains to the RoboteQ motor controller, which determines the wheel speed and direction of rotation.

A Hemisphere V103 weatherproof GPS provides a heading and GPS coordinates within a tolerance of 23.6 inches. The heading is determined by the time difference of a signal between two receivers within the GPS. In the event that satellite communication is lost the GPS contains a gyroscope which approximates position and heading.

A SICK Tim551 laser rangefinder with a viewing angle of 270° and a range of 32.8 feet is used for object detection.

A Microstrain 3DM-GX3-25 Inertial Measurement Unit (IMU) provides filtered accelerometer, magnetometer and gyroscopic data in a discrete package. The IMU is mounted at the center of the robot for optimal data collection.

Computer

Scipio uses three Radaxa Rocks and one Nvidia Jetson TK1 connected together via a gigabit Ethernet router to distribute computational load. This removes dependence on an external power source and greatly reduces weight and space consumption, which were issues with the laptop. The Nvidia Jetson TK1 offers 192 CUDA cores which are used for image processing while the Radaxa Rocks are used for other computational tasks. System tests confirmed that the performance of the embedded platforms exceeded that of the laptop while drawing less power.

Electrical Innovations

The Backplane centralizes all logic circuits and sensors making it the most vital part of the system. The card sockets offer modularity of the logic circuits, this allows easier replacement if a card is damaged or rendered obsolete. High current DC-DC converters have been installed onboard, providing more than ten times the current capacity of the previous regulators.

The Emergency Stop transmitter has been redesigned as higher energy efficient system providing a significantly longer battery life.

SOFTWARE DESIGN

The previous experience with Robotic Operating System (ROS) from last year prompted continue development of the software system using this set of libraries. ROS is a dedicated open-source platform for robots that provides a standard communication channel between software nodes that monitor the robot’s environment, make plans based on present and past information, and act accordingly by sending commands to the motor controllers. The use of ROS allows quick and efficient design of modular nodes which are each made to handle specific tasks and can be modified, replaced, or removed without affecting the function of other nodes. Many nodes and libraries have also been written and shared by ROS' large online community, which is another benefit of ROS as it saves untold amounts of time and effort by not requiring users to reinvent the wheel.

The simulation and testing tools for ROS, including the ability to "bag" data and replay it, RVIZ, and Gazebo, have also each proven invaluable due to their ease of use, and the capability to test the system as a whole, or individual components, without the use of the physical robot. This allows for testing to be performed while other members are making use of the robot or while the robot is not in a state of functionality due to repairs needed or other such issues.

Software Architecture

ROS nodes are contained in packages. Packages can be groups of nodes sharing similar functionality such as mapping, navigation, or individual nodes responsible for unique tasks. Data are collected and modified by the sensory nodes, then sent to the navigation stack. The navigation stack then uses the data to determine Scipio’s location, orientation, and speed. Once the current state is determined, it is compared to the current goal state to determine the steps needed to attain the desired state. The navigation stack then communicates with the motor controller and indicates a speed and direction of movement. The navigation stack monitors the data sent by the motor controllers to measure and adjust progress towards the goal.

Language and Libraries

Scipio's software is written in C++ and Python, while XML and YAML are used for mark-up files. The ROS API handles all primary functionality such as movement, object recognition, line detection, and localization. Libraries used include ROS, JAUS, gUnit, NumPy, and OpenCV for image processing.

Laser Rangefinder

Scipio's laser rangefinder (LRF) is a SICK TiM551. Its driver is a ROS node that requires the parameters of the rangefinder such as the minimum and maximum viewing angle. Once launched, the node connects to the LRF and begins publishing "laser_scan" messages. The navigation stack listens to these laser_scan messages and places the obstacles found onto its map and takes appropriate measures to avoid collisions.

GPS/Compass

A Python ROS node serves as the driver for the GPS receiver. It connects to the receiver through a serial connection and listens for sentences using the NMEA 0183 standard. The GPS internally calculates the heading using the difference in position readings between the two individual GPS receivers. The node converts both the position and orientation readings from degrees to radians and publishes them to appropriate topics for use by the navigation stack.

The navigation stack uses the heading and GPS data to determine its position and orientation in the world. It compares this data to its current goal and any specified GPS waypoint to determine progress and any necessary adjustments.

Line Detection

The computer vision portion of autonomy starts with acquiring the actual camera data using the ROS pointgrey_camera_driver, which sends a stream of image messages to both the line detection and flag detection nodes. The line detection node processes the images and outputs a pointcloud message containing the lane marking as obstacles, which the costmaps receive and use as their obstacle data, stopping it from crossing the markings.

The development of the line detection algorithms had been a bottleneck in previous years due to its long life-cycle of research-implementation-testing. In order to minimize the time spent on writing and implementing code to prototype the algorithms, a generic interface class was written in Python that allows the capability to write multiple algorithms and filters independently, and chain them together in different combinations. In other words, it allowed testing of multiple algorithms in parallel and tying together multiple components for testing, instead of constantly rewriting a single monolithic program many times (with lots of redundant boilerplate code). With different modules running side by side, the performance was able to be evaluated by comparing their outputs with the true desired output. A sample configuration of filters that were used are shown below in Figure 7, where nodes are circled and topics are rectangles.

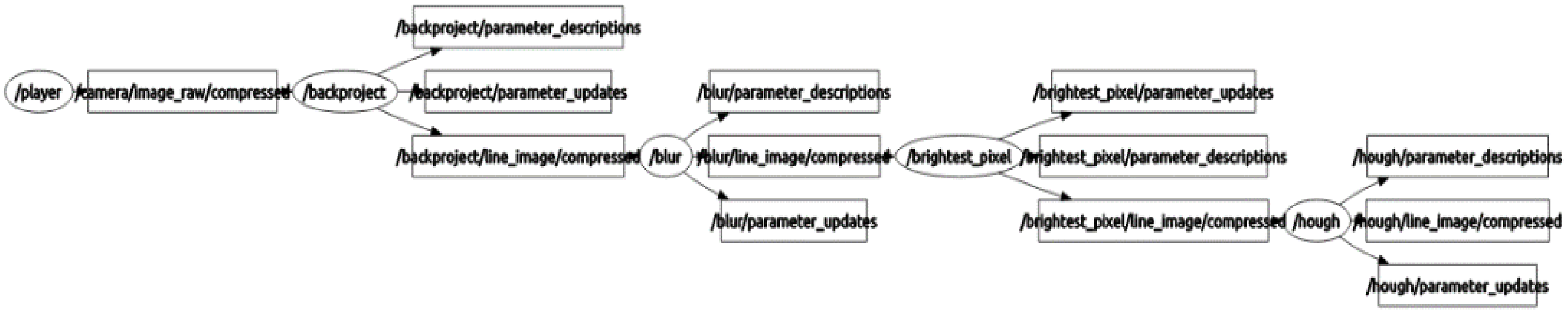

/player
/camera/image_raw/compressed
/backproject
/backproject/parameter_descriptions
/backproject/parameter_updates
/backproject/line_image/compressed
/blur
/blur/parameter_descriptions
/blur/line_image/compressed
/blur/parameter_updates
/brightest_pixel
/brightest_pixel/parameter_updates
/brightest_pixel/parameter_descriptions
/brightest_pixel/line_image/compressed
/hough
/hough/parameter_descriptions
/hough/line_image/compressed
/hough/parameter_updates

Figure 7. Chained ROS nodes and their output topics use for prototyping line detection algorithms.

The algorithms utilized for line detection include various basic image filters (Gaussian blur, median blur, erosion, dilation, subtraction, and others), as well as more advanced filters such as histogram equalization, histogram backprojection, RANSAC, and Hough transforms, all applied in series on each image acquired from the camera at approximately fifteen frames per second.

The most problematic portion of the images for detecting lanes was the grass, because of its unpredictable patterns and varying color/intensity profile. It was quickly determined that relying on static threshold values for brightness and color were not robust enough to filter out the grass, and would need methods to improve reliability. The current grass-filtering algorithm builds a 2-D histogram (hue-saturation channels) from a training dataset, and then backprojects the current image pixel-by-pixel based on that pre-built 2-D histogram. In other words, it filters out a pixel in the current image based on the probability that it belongs in the 2-D histogram that was built based on the training dataset. A sample training 2-D histogram is displayed below as a heatmap, with the resulting filtered image using that histogram in Figure 8.

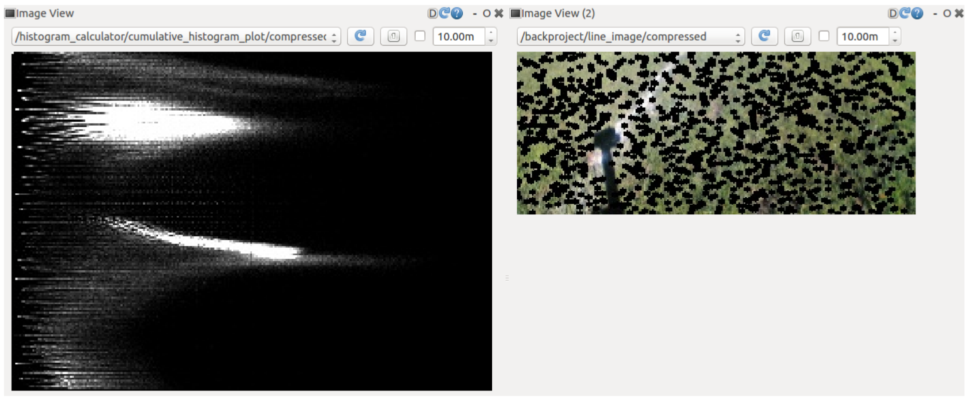


Image View
/histogram_calculator/cumulative_histogram_plot/compressed
10.00m
Image View (2)
/backproject/line_image/compressed
10.00m

Figure 8. Heatmap of constructed histogram from training image (left), where the x axis corresponds to saturation values, the y axis corresponds to hue values, and the intensity corresponds to the relative count of those pixels. The resulting backprojected pixels are subtracted from the camera input during a run, removing a lot of noisy grass areas (right).

The output from most of the filtering stages is shown below in Figure 9. Having most of the grass pixels filtered out, a simple blur filter (top right) was used, and then a brightest row filter (bottom left) was run. The brightest row filter removes all pixels in each row except the brightest pixel in that row. This filter works best when the robot is parallel to the lane markings. Finally, the remaining pixels provide an estimate for the lane markings, which is extracted using a Hough transform operation (bottom right).

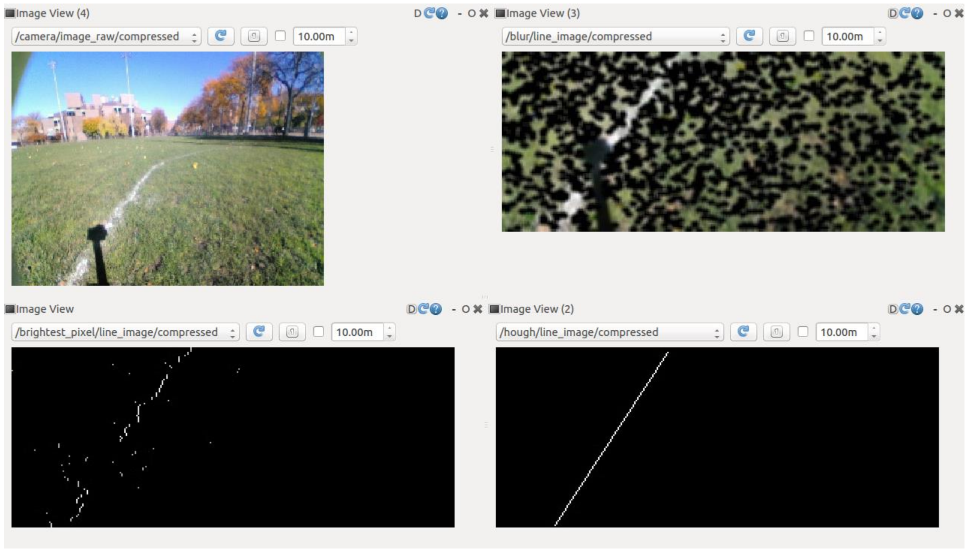


Image View (4)
/camera/image_raw/compressed
10.00m
Image View (3)
/blur/line_image/compressed
10.00m
Image View
/brightest_pixel/line_image/compressed
10.00m
Image View (2)
/hough/line_image/compressed
10.00m

Figure 9. Debug image output of different stages of line detection algorithm. Raw camera data (top left) is backprojected then blurred (top right). Then ran through a brightest pixel filter (bottom left) and finally a Hough transform to extract the lines (bottom right).

Odometry and Localization

Scipio’s location is tracked and monitored utilizing data from multiple sensors. Currently, three nodes provide odometry data: an IMU, GPS, and motor controller node. These sensory inputs are then combined into a filtered odometry output using an extended Kalman filter known as robot localization to produce fairly accurate odometry data. From the filtered data collected and observed in initial testing, the results from traversing a 50 meter by 50 meter square and returning to a starting point gave the results that indicated a 0.5 meter error. Notable sources of error originate from the inherent translational slippage of the wheels and the limited accuracy of the GPS.

Each sensor provides odometry data which allows for a sense of dead reckoning. However, each sensor provides a reading which is relative to a different coordinate frame; the wheel odometry and IMU data are provided in the coordinate frame of the robots chassis, whereas the GPS data provides an estimate of the robot relative to the UTM coordinate frame. Once initial sensor data is acquired, then each of the following wheel odometry and IMU gyroscope readings is set relative to this initial zeroed position and orientation of the robot. This differs to the data provided by the GPS since the position and orientation computed is based off the latitude and longitude readings. A UTM to robot transformation is then applied here in order to place the starting position of the robot's chassis coordinate frame to also be in the UTM frame.

The absolute position and relative velocity of the wheel odometry data can also be separated into two separate coordinate frames. The absolute position is considered to be in the odometry coordinate frame containing the pose from the start of the robot's run, while the relative velocity is considered to be in the local frame of the robot, known as 'base_link'. As absolute position data tends to significantly drift over time, the robot now only provides its relative velocity information to the Kalman filter so that it can be integrated. This integrated value is then fused with the relative velocity data of the IMU.

Mapping

In the previous year ROS' SLAM gmapping library had been used, which builds a map using laser scan and odometry data as the robot runs and contains points depicting objects detected. However, it was deemed that this method was less efficient at keeping track of the robot's position relative to the map coordinate frame than using an a priori map. An a priori map first generates a blank map roughly the size of the course and then adds the accumulated obstacle data from the laser scanner as well as detected lines generated from the line-detection node. Due to the fact that the a priori map has known, fixed dimensions, it is easier to keep track of the robot's position using this type of map than if using a map with no fixed dimensions.

Navigation

Localization, orientation, and obstacle and line detection information are processed by the Navigation Stack (NavStack) to generate the movement commands sent to the motor controllers. NavStack's main package "move_base" is a library native to ROS. Figure 10 shows move_base's interactions between sensor input and command output.

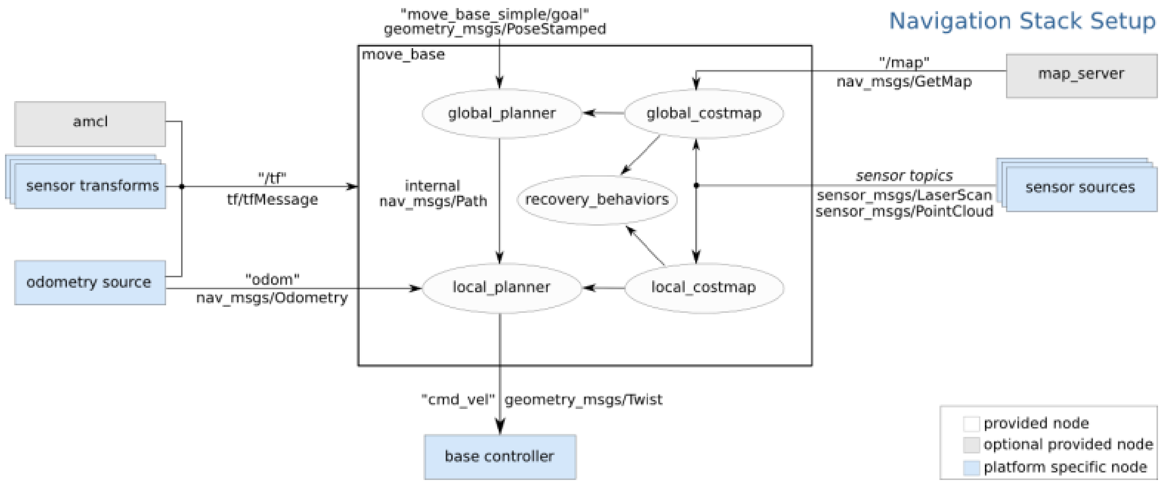


Figure 10. ROS Navigation Stack Structure[*]

All sensor information is sent to both the global and local cost map nodes within the move_base package. The global cost map represents all information Scipio has about its environment. It continuously saves sensor information and builds the cost map until the system is restarted. The local cost map is the pool of information which is acquired from the immediate vicinity, a 4 meter radius around Scipio. The local cost map is constantly updated but is not stored for future use. The global cost map is used for long-term goal decisions, and the local cost map is used for short-term decisions.

As Scipio collects data from sensors, extracting useful information from disorganized data becomes increasingly difficult. Transforming it based on known characteristics of sensor

[*] http://wiki.ros.org/move_base

placement eliminates this difficulty. Converting data systematically to Scipio's point of reference at its center is done via a transform tree as shown in Figure 11. This allows data coming from a sensor such as the LRF to be adjusted by translation and rotation so as to appear that the LRF is at the center of the robot.

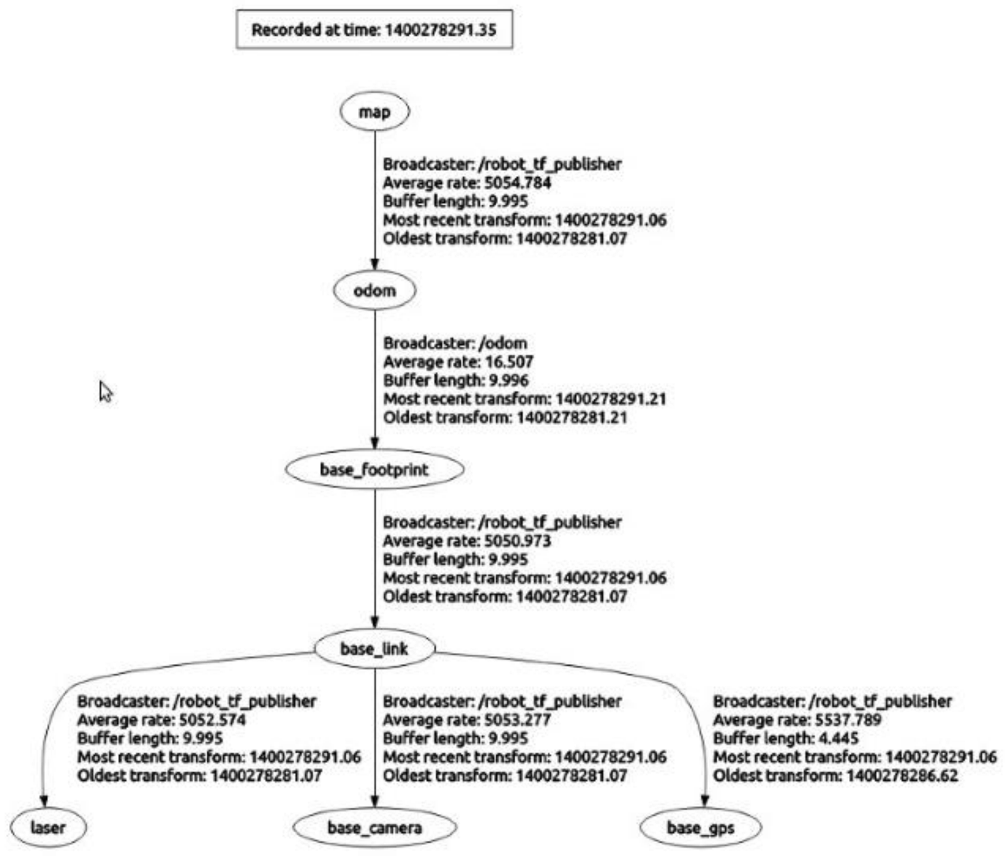

Recorded at time: 1400278291.35
map
Broadcaster: /robot_tf_publisher
Average rate: 5054.784
Buffer length: 9.995
Most recent transform: 1400278291.06
Oldest transform: 1400278281.07
odom
Broadcaster: /odom
Average rate: 16.507
Buffer length: 9.996
Most recent transform: 1400278291.21
Oldest transform: 1400278281.21
base_footprint
Broadcaster: /robot_tf_publisher
Average rate: 5050.973
Buffer length: 9.995
Most recent transform: 1400278291.06
Oldest transform: 1400278281.07
base_link
Broadcaster: /robot_tf_publisher
Average rate: 5052.574
Buffer length: 9.995
Most recent transform: 1400278291.06
Oldest transform: 1400278281.07
Broadcaster: /robot_tf_publisher
Average rate: 5053.277
Buffer length: 9.995
Most recent transform: 1400278291.06
Oldest transform: 1400278281.07
Broadcaster: /robot_tf_publisher
Average rate: 5537.789
Buffer length: 4.445
Most recent transform: 1400278291.06
Oldest transform: 1400278286.62
laser
base_camera
base_gps

Figure 11. Transform Tree

The LRF is located 0.4 meters in the x plane from Scipio's center and 0.124 meters in the z plane from it. Every new data point the NavStack receives from the LRF will have these x and z offsets added to it. Point cloud messages sent by the camera driver have an x offset of 0.14 and z offset of 0.3. The data from the GPS driver is not transformed using offsets as the center of the GPS receiver coincides with Scipio's center, and GPS based altitude readings are not used. These sensor transforms are handled by the nodes, "tf_broadcaster", which publishes the sensor transform offsets, and "tf_listener", which transforms new data based on the published offsets. The base of the robot, "base_link" is the parent of all sensors. For data to be sent to the map frame, which allows the global cost map to analyze information, data from the base link frame must be transformed to the odometry frame. No offset is necessary here as both frames use the same reference point at Scipio's center.

Once determining that navigation should continue the NavStack then analyzes the information acquired from the global and local cost maps and determines a short distance path that will optimally decrease the distance between Scipio and its goal. Once Scipio detects an object in its path the local path planner will attempt to determine a direction to turn to continue navigation toward the target. As this path is determined, the NavStack continues to determine the long term path toward the goal. This is important in situations where the local path planner is unable to determine a path such as when Scipio has moved into a corner and must move backwards before continuing movement toward the goal.

Scipio's objective is a GPS waypoint defined in software before each run. The move_base package determines the path that leads to the desired location. The GPS coordinates are sent to the NavStack in a message containing an x, y, and z location computed by the "gps_goal" node. Once the NavStack has received this goal the global path planner begins determining the overall path and the local path planner begins determining the short term paths.

Once move_base has computed and analyzed all of the sensor information, recovery plans, and has picked a desired path, it sends a velocity command to the RoboteQ driver. This command contains a linear distance to move and an angular velocity to turn. After each velocity command is sent odometry information is collected from the motor controller to determine if Scipio has moved as commanded. If it has not then recovery behavior will be attempted to adjust Scipio towards the desired path.

Joint Architecture for Unmanned Systems (JAUS)

Scipio uses the protocol to report information about its current operating state and to receive waypoints for navigation. JAUS is implemented in Scipio as a ROS node built from JAUS Tool Set (JTS). The node listens for and responds to information requests, and sets the appropriate waypoint in the navigation stack.

Testing and Simulation

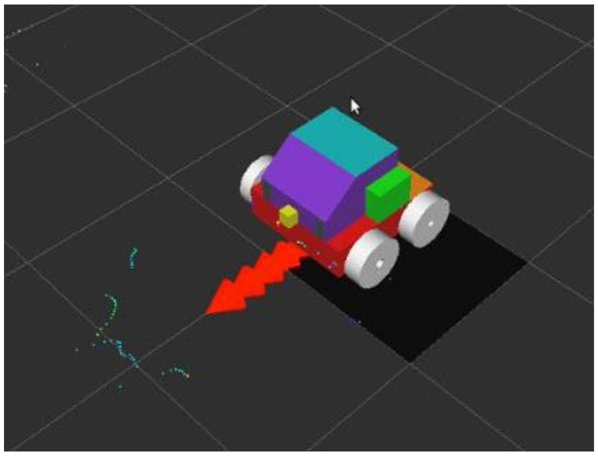


Figure 12. Scipio Modeled in RVIZ

ROS contains a number of testing tools that expedite development and aid in the elimination of bugs early in the process. RVIZ, shown in Figure 12, is a 3D visualization environment developed by the creators of ROS to be used for testing, debugging, and simulation. Gazebo is a 3D simulation tool compatible with ROS and RVIZ with a complex physics engine, allowing for a very accurate simulation of real environments. With these tools, the software team was able to develop individual software components and test them before integration.

RVIZ displays the data being taken in by the sensors and sent through each topic in real time. The visualizer displays the sensor data as seen by Scipio as well as the messages Scipio publishes to each ROS topic. RVIZ uses a markup file containing the robot's physical information in the Unified Robot Description Format (URDF) in order to accurately display the robot in the environment.

Gazebo uses URDF files to simulate the robot’s performance as well as its surroundings. Using additional plugins for each sensor and the motor controller, Gazebo simulates data from the environment being detected by the sensors publishes the data to their respective topics. Premade test objects such as barricades or traffic cones can be placed in the simulation environment to be used in navigation and object detection testing. The simulation works in conjunction with RVIZ to display the simulated data and the active topics. Using both packages allows for software testing without physically operating the robot.

SYSTEM SAFETY

Ensuring bystander and vehicle safety was the top concern during the design process, safety measures were considered at all points of Scipio's design and fabrication. All exposed edges and corners were smoothed to prevent injury. All drivetrain components were properly enclosed and protected. The Motor Monitor alerts bystanders of Scipio's presence and movement. The RoboteQ motor controller checks for valid movement commands and ensures that no random or spontaneous movement will occur in RC mode. The EStop deactivates the vehicle when two "GO" signals are not detected. Finally, Scipio's speed is mechanically limited to 5.98 mph.

PERFORMANCE ANALYSIS

The performance analysis section includes the predicted performance of the vehicle versus the actual performance of the vehicle when tested.

Vehicle Speed

With 14 inch diameter wheels, a speed reduction of 30.63:1, and motors with an output of 4400 rpm at 24 volts, the maximum theoretical speed of Scipio is 5.98 mph assuming no losses due to loading. Testing showed an average maximum speed of 5.59 mph, which is within 6.6% of the calculated speed.

Ramp Climbing Ability

Due to Scipio’s low center of mass and powerful drivetrain, it was expected to be able to climb an incline of 40°. Testing showed that Scipio was able to easily climb a 45° incline, significantly greater than any incline on the IGVC course.

Reaction Time

Taking images via the camera and processing them to detect lines is the slowest process in navigation. Each image takes 90 milliseconds to acquire and filter. Once the line skeletons have been obtained, there is negligible additional overhead to update goal progress.

Battery Life

Scipio can operate for a maximum of 90 minutes before the drive motor batteries must be replaced.

Obstacle Detection

The SICK Tim551 is rated to have a detection distance of up to 32.8 feet indoors and outdoors.

Complex Obstacle Handling

To determine if Scipio is stuck or has come to a corner or other obstacle which must be navigated around, the information from both the local and global costmap nodes is sent to a recovery behavior node which decides between four different recovery options as shown in Figure 13. If a recovery behavior cannot be achieved the system will abort the navigation to avoid any further unwanted movement.

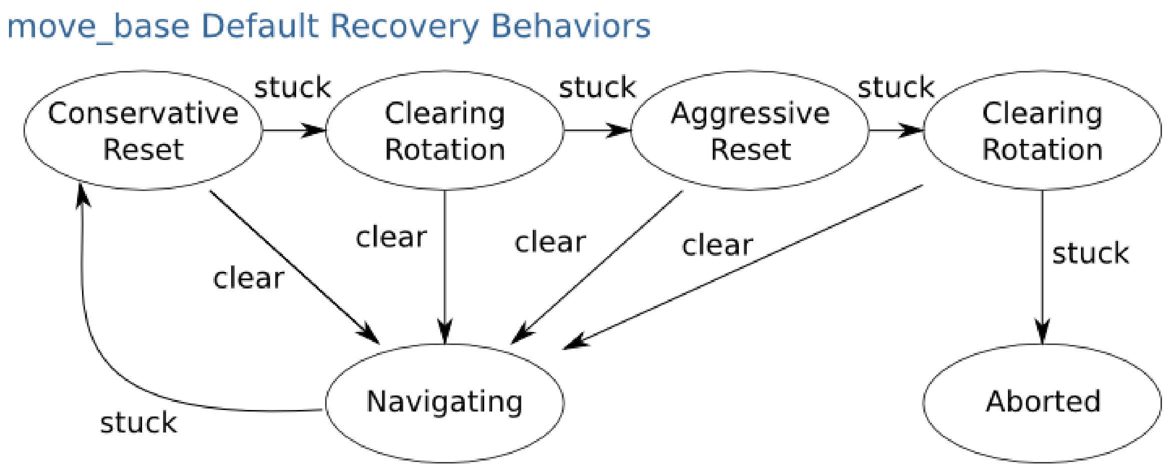


Figure13. move_base Flowchart[*]

Positional Accuracy

Scipio now uses the Hemisphere V103 GPS Compass, which is accurate to 0.6 meters. This unit also contains a precision compass, which is internally filtered to increase total perceived GPS accuracy.

[*] http://wiki.ros.org/move_base

BILL OF MATERIALS

Table 2 represents both EDT's lifetime and fiscal year monetary investment in Scipio.

Table 2. Bill of Materials

Part Description	Part Use	QTY	Retail Price ($)	Cost to Team This Year ($)	Vendor
Batteries: Power Sonic PS-12350	Drivetrain Power	2	130	0	BatteryPlex
GPS: Hemisphere V103 Smart Antenna	Navigation	1	3200	0	Hemisphere
Radaxa Rock	Software	3	297	297	Radaxa
Nvidia Jetson TK1	Software	1	192	192	Nvidia
Laser Range Finder: SICK Tim 551	Object Detection	1	2627	0	SICK - Donation
Motors: Palmer Industries 200	Drivetrain	2	1100	0	Palmer Industries
Black Fly Camera	Line Detection	1	250	250	Point Grey
Camera Lens	Line Detection	1	500	500	Edmund Optics
Wheel Encoders: US Digital HD25-1000	Position and Velocity Determination	2	681	0	US Digital - Donation
Wireless Tranceiver: RadioTronix Wi.232 928 MHz	Emergency Stop Transmitter and Reciever	2	119	0	Mouser
Microstrain 3DM-GX3-25	Inertial Measurement Unit	1	1600	1600	Microstrain
Circuit Elements (Copper Boards, Microcontrollers, etc..,)	Circuit board prototyping	N/A	100	0	Jameco + Mouser
Switches, Wires, Crimps	Various	N/A	100	0	Jameco + Mouser
New Haven Graphic Display 160x 128	System Information Display	2	134	134	Mouser
0.083 wall 1" x 1" x 6' steel tubing	Bottom Chassis Frame	6	132	132	McMaster-Carr
Aluminum Flat Stock	Gearbox Casings/Drivetrain brackets	1	300	120	Online Metals
Spur Gears	Gear Box	10	660	0	McMaster-Carr

Bearings	Gearbox/Drivetrain	20	660	120	McMaster-Carr
Drive Belts (Gates Poly Chain)	Drivetrain	4	312	0	Murph Haines, Inc.
Drive Pulleys (Gates Poly Chain Sprockets)	Drivetrain	8	1548	0	Murph Haines, Inc.
Taper-Lock Bushings	Drivetrain	8	176	0	McMaster-Carr
3.50" Dia., 0.75" Dia., 0.625" Dia. Steel rods	Drivetrain, Gearboxes, Wheel hubs	3	60	0	Murph Haines, Inc.
1" T-Slotted Extrusion (10 Feet)	Top Chassis Frame	4	176	124	McMaster-Carr
T-Slotted Framing Accessories	Top Chassis Frame	N/A	500	400	McMaster-Carr
Polycarbonate sheet	Top Chassis Windows/Doors	3	116	116	McMaster-Carr
Aluminum Sheets	Paneling/Bottom Plates	N/A	315	315	Stainless Supply
Fasteners	Fastening	N/A	150	60	McMaster-Carr
Rubber Seals & Stripping	Weather proofing	N/A	100	100	McMaster-Carr
Kenda 14 inch Diameter tires	Drivetrain	4	200	0	Northern Tool
Totals			**$16,435.00**	**$4,460.00**	

CONCLUSION

EDT-Scipio represents the combined efforts of seventeen members of the Chicago Engineering Design Team. This year's model has seen great improvements from last years across all areas. Numerous mechanical modifications allow easier maintenance of Scipio and support new equipment and sensors. Reworked electrical systems decreased complexity while adding features and hardware. The decision to expand development on ROS allowed the software team to focus on strategy and greatly improve Scipio's navigation capabilities. Having seen improvements to all aspects of its design, Scipio stands as the finest of EDT's work.

Faculty Advisor Statement

I certify that the engineering design documented in this report and implemented into this vehicle by the current student team is significant and equivalent to the work required to receive Senior Design credit.

[illegible]

Dr. Miloš Žefran
Department of Electrical and Computer Engineering
University of Illinois at Chicago