EDT-SCIPIO

University of Illinois at Chicago
Authors: Justin Delcourt, Krystian Gebis, Paul Gorski, Matt Kelly, Chris Lee, Leslie Malaki, Jasen Massey, Michael Medsker, Akshay Patel, Zach Quinn, John Sabino, Basheer Subei, Steven Taylor, Bart Wyderski

Dr. Miloš Žefran, mzefran@uic.edu

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 self-built gearbox and pulley drivetrain qualify as an innovation, as well as the ease with which our top chassis can be modified to accommodate new sensors. The electrical team has spent their time reducing size and complexity of the circuits required to operate the system, while adding more features. A new smaller backplane and new motor controller are a breath of fresh air to our control system. On the software front, development was moved to 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 members, but the IGVC team is made up of only the most experienced. Figure 1 shows how the team was the organization of members and distribution of responsibilities. The team was divided into mechanical, electrical, and software departments, all overseen by EDT's President and Vice-President. For IGVC 2013 the team invested roughly 2000 man hours to create a new mechanical and electrical platform. With IGVC 2014, EDT has invested another estimated 1500 man hours, largely devoted to software development.

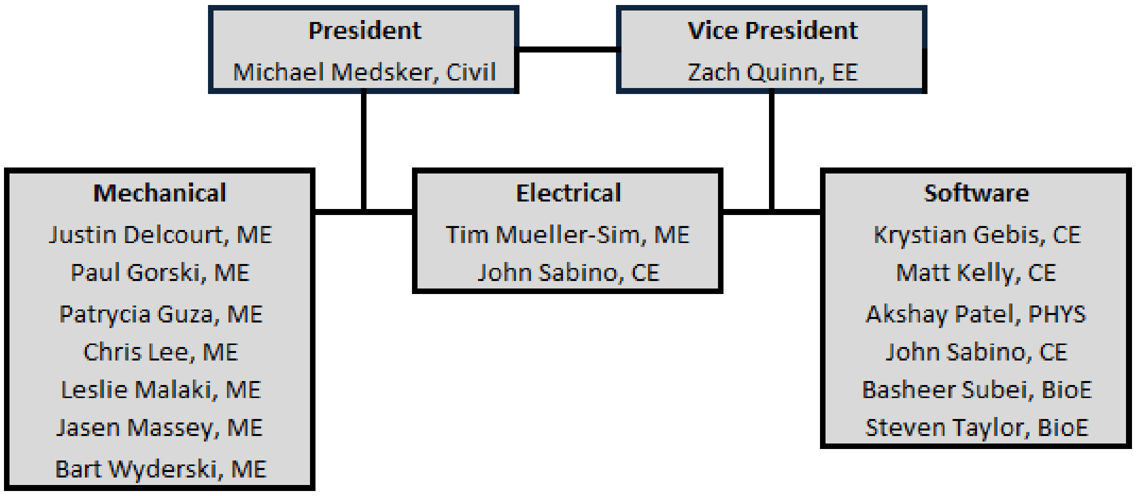


Figure 1. IGVC Team Organization

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.

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

Table 1. Morphological Chart

Functions/ Features	Mean 1	Mean 2	Mean 3	Mean 4
Translate motor rotation to wheels	Chain and Sprockets	Drive Shaft	Spur Gears	Belt and Sprockets
Detect objects	Laser Range Finder	Camera	Sonar	Infrared
Power Source	Lithium Ion Batteries	Lead Acid Batteries	Generator	Solar

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 chassis assembly and bottom chassis assembly, 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.

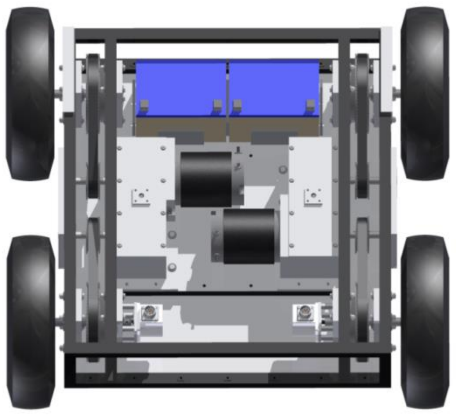


Figure 2. Bottom Chassis Assembly

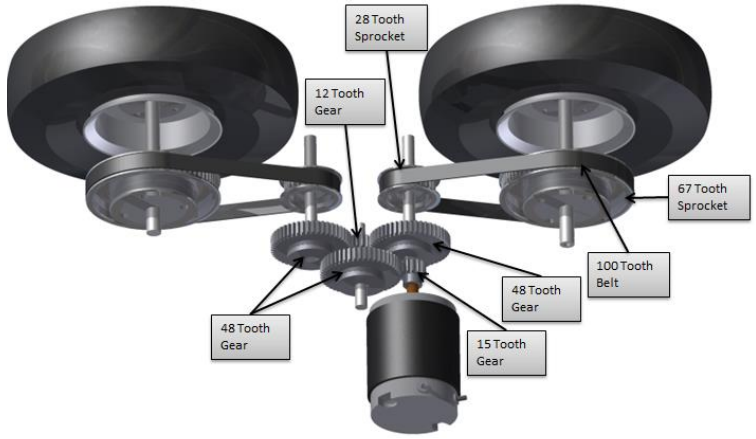


Figure 3. Gearbox Skeleton View

Top Chassis Assembly

The top chassis, shown in Figure 4, was designed to be modular and accessible while maintaining a professional aesthetic. The top chassis stores the logic circuits, payload, and all sensors. The top chassis is divided into three compartments: the electrical box, laptop area, and housing area. Each area can be accessed by either a drawer or a door. The laptop platform slides outward on a drawer, while the electrical box and housing area have hinged doors for access. Aluminum T-Slot framing was used as the base structure for the top chassis, allowing for easy assembly and future modifications. Aluminum panels with rubber edge-grip seals were fastened to the T-Slot. These panels provide protection and weather proofing. The top and bottom chassis are mated with four quick-release pins. The pins provide easy separation for troubleshooting and maintenance.

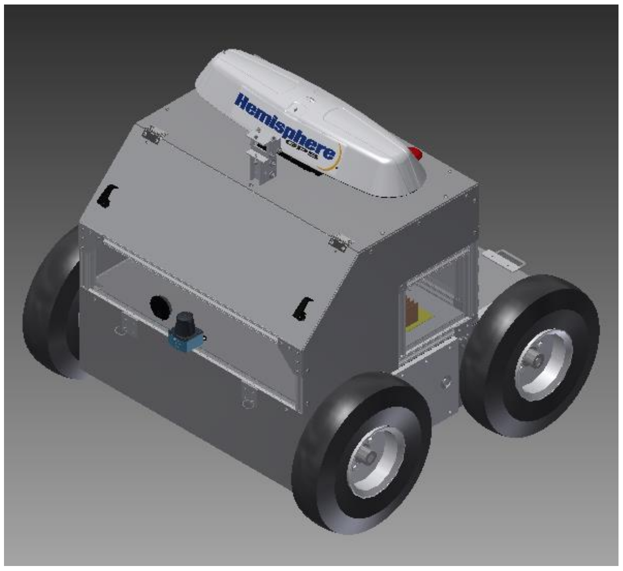


Figure 4. Top Chassis Mated to Bottom Chassis

The T-Slot frame allows for easy mounting of components and sensors. To mount the GPS receiver, pieces of T-Slot were added between the front and rear supports. The receiver was then secured to the supports using L-brackets. Drawer slides were also mounted to the T-slot frame. These slides support the drawer that holds Scipio’s laptop computer, and allow the drawer to be locked in place keeping the drawer either open or closed.

Innovations

Scipio’s 2013 design required a human operator to hold the payload panel open for access. This year two five lbf gas struts were installed to keep the door open. These struts automatically open the panel when the locks are disengaged. Compression latches counteract the force applied by the struts. The latches are adjustable, and were calibrated to counteract all force from the gas struts and vibration. The struts caused minor deformation damage to the original payload panel. Increasing the thickness of the plate from 1/16” to 1/8” eliminated the possibility of deformation. The increased thickness of the plate also supports a new GPS receiver and allows for better placement of the Emergency Stop button.

Scipio travels with a large amount of testing and backup equipment. A trailer provides mobile storage space for this equipment. The design is a simple rectangular frame with a tailgate, similar to a pick-up truck. Since a single hitch system has the potential to jackknife, a double hitch system was created. The dual-hitch system prevents jack-knifing, yet allows vertical rotation on uneven ground. Caster wheels allow the trailer to rotate when Scipio turns in place.

ELECTRICAL DESIGN

This year EDT narrowed its focus to reducing the size and complexity of Scipio's electrical system in addition to adding a few small features. A new smaller backplane integrated our systems and reduced wiring. A new motor controller reduces the latency of command execution, and allows for easy installation and configuration of a real-time drivetrain control loop. Numerous modifications to the hardware compartment allow for easier access for maintenance. These changes allowed the team to simplify the design, improving performance and reliability. Figure 5 shows a high level overview of the hardware and sensor flow.

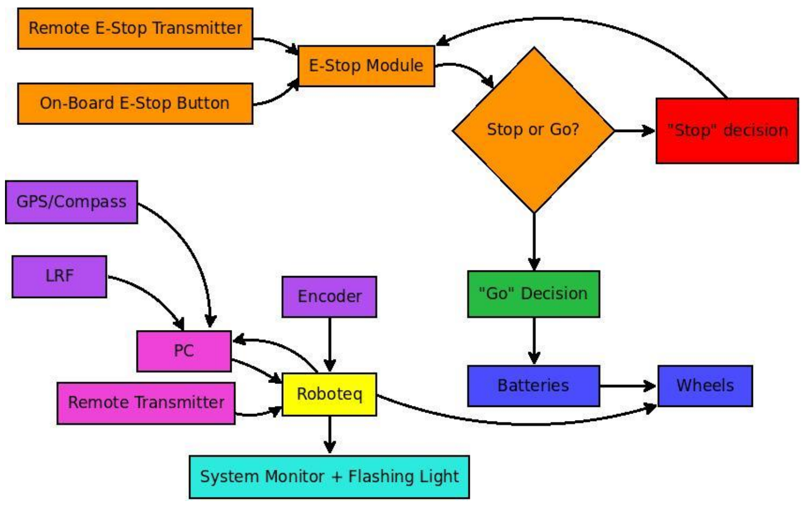


Figure 5. EE Flowchart

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 were soldered directly to the backplane, improving efficiency and providing 12V, 5V, and 3.3V power lines for sensors and logic circuits. A simple low pass filter eliminates switching ripple effects. The new 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. Using a workstation class laptop provided the needed computational power, while eliminating need for an inverter, and increases overall robot runtime.

Emergency Stop

The Emergency Stop (E-Stop) system disables Scipio whenever an emergency situation occurs. The E-Stop may be activated wirelessly by remote control or manually by pressing the onboard switch. The system is composed of a wireless handheld transmitter and a receiver on 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 a range of 5 miles and uses spread spectrum technology to provide data encryption and prevent interference or jamming from external sources.

System Monitor

The system monitor is a twofold system. Its primary focus is safety, and will alert bystanders to the current mode of operation. It controls four high output LEDs, one facing out from each of Scipio's sides, mounted under the webcam. This circuit receives the current drive mode from the Roboteq, and places itself in one of two states. When autonomous mode is engaged, the LEDs blink with 1 Hz frequency, giving people adequate warning to stay clear of the robot. When not in autonomous mode, the LEDs are on, but do not flash.

The second part of the system is to provide status information to the operator, including battery voltage and motor current draw. By communicating with the Roboteq using RS-232, the system can alert the operator to an upcoming problem, such as bad motor or batteries that need to be charged. This circuit can change the color of several high output LED's placed on the rear of the robot.

Sensors

Scipio has several sensors that provide data feedback to the computer software. 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 motor controller, which determines the wheel speed and direction of rotation. A weatherproof GPS receiver is used to help with goal planning and navigation by heading and GPS waypoint. A laser rangefinder is used for object detection and replaces a previously less accurate system of sonars and a stereoscopic disparity camera.

Computer

Scipio uses an onboard laptop, which eliminated power system requirements such as DC-AC inverters. The laptop is durable and withstands vibration and temperature shock. Replacing the mechanical drive with a solid state drive provided additional vibration resistance. It contains an NVidia Quadro K3000M graphics card, which will allow for GPU processing in the future. A secondary battery was installed, giving an average runtime of four hours.

Electrical Innovations

The largest innovation this year was the move to the Roboteq HDC2450 motor controller. This new controller made the reduction in backplane size possible. One important feature of the motor controller is its ability to monitor battery voltage and motor current draw. The motor controller also allows the user to define Scipio’s mode of operation. A mode is set by pressing one of the four pushbuttons on Scipio’s left side. First is the safety mode, the default mode of operation. Scipio remains stationary in this mode. The second mode provides autonomous control to Scipio. The third mode allows a user to direct Scipio by remote control. If Scipio fails to detect a valid RC signal it will switch to the safety mode. The final mode allows the vehicle to switch between autonomous and radio control. Control is determined by a toggle switch on the remote control.

The backplane is a vital part of the design, and the new motor controller was vital in its improvement. Moving to the new Roboteq motor controller allowed the team to design a new backplane that is 28.6% smaller. The backplane has DC-DC converters onboard, reducing the amount of wiring between components. The previous backplane included a signal multiplexor and two custom motor controllers. These were removed and re-implemented with a combination of software and the new motor controllers.

In the past the team needed to manually unplug batteries, sensors, and communication lines to separate the top and bottom chassis. This year, the team installed self-aligning blind mate connectors, which allow the top chassis to be effortlessly removed and replaced. This made disassembling Scipio easier and faster, reducing time needed for maintenance and modifications. Eliminating and rerouting the remaining cables increased the payload volume by 17.8%.

Lastly, Scipio received a major GPS update. Previously, it relied on a receiver that was only accurate to an average of 2.5 meters. This receiver was replaced with a Hemisphere V103 GPS

Compass. This receiver is accurate to 0.6 meters, allowing for much more accurate GPS navigation. This unit also contains a precision compass and gyroscope, allowing for navigation by heading. The receiver is mounted perpendicular to Scipio's direction of forward movement. Because the receiver is designed to be mounted parallel to forward movement the compass readings are shifted 90 degrees before being broadcast. This compass communicates with the GPS to stay calibrated when moving between different electromagnetic regions of the earth.

SOFTWARE DESIGN

In 2013, Scipio ran an in-house software platform named Deimos. Deimos presented two major difficulties. First, the team spent a large amount of time on tasks such as navigation and drivers, consuming time that was needed for integration and testing. Second, integration proved difficult as each member had developed their portion in isolation. Because Deimos had no well-defined communication interface, Scipio was unable to reliably process sensor data and respond properly.

To address these problems, development was moved to 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. Nodes are able to publish or subscribe to any information topic and act independently.

ROS's organization as nodes that communicate through a standard channel has eliminated the integration problems that the team experienced during Deimos' development. Each member is able to develop and test each node in total isolation or with any number of other nodes. Thanks to ROS's adoption by the robotics community, there are many libraries available reducing the need to re-implement existing code. ROS also has a powerful testing, logging, and simulation tool chain that allow the team to ensure nodes function as intended.

Software Architecture

ROS nodes are contained in packages. Packages can be groups of nodes sharing similar functionality such as mapping or 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 find the steps needed to attain this 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 was written in C++ and Python. XML and YAML are used for mark-up. 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.

Movement

Locomotion of Scipio is controlled by the Roboteq HDC2450 motor controller. The motor controller allows Scipio to be controlled in closed loop where PID constants can be easily tuned. For comparison, in 2013 Scipio's movement was controlled by querying two independent drivetrain control systems for wheel velocities. This inefficient high latency process was eliminated by the switch to the Roboteq motor controller, which provides a programmable interface that decreased communication and process latency while allowing for tighter control specifications. Velocity commands and measurements are sent between the motor controller and laptop through serial communication.

Laser Rangefinder

Scipio's laser rangefinder (LRF) is a SICK TiM310. 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. At that point, 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 C++ 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. GGA, the first sentence type, contains Scipio's position in degrees latitude and longitude. HDT, the second sentence type, contains Scipio's heading in degrees. The node converts both 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.

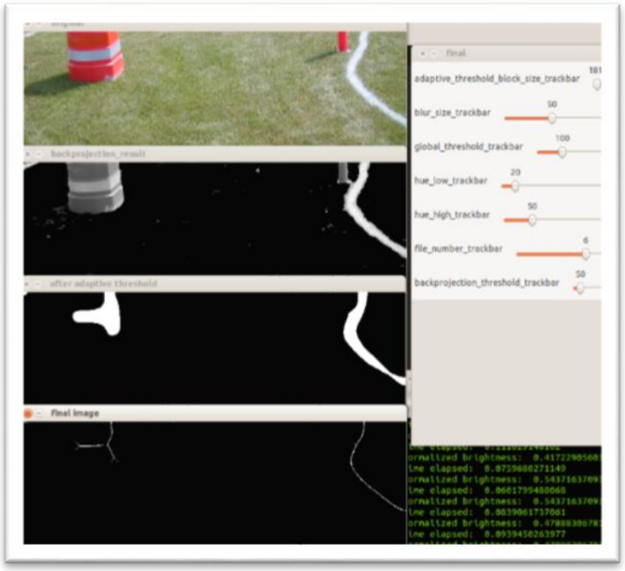
Line Detection

A Python ROS node processes data from the webcam to detect and stay inside lanes. The node makes extensive use of the OpenCV library in conjunction with the NumPy library to manipulate and process the images. It captures images from the webcam, removes distortion, filters and processes the images, then sends the lines it found as point cloud data to other nodes. The point cloud contains the line points in 3-D space, which are published to the navigation stack and placed as objects on the map.

The images are captured from the webcam using OpenCV's VideoCapture.imread() function, which returns a 2-D matrix containing the pixel values for the red-green-blue (RGB) channels. Radial and tangential distortion is then removed. Before each run, the camera focal length, skew coefficient, and other parameters are determined from pictures of a checker board pattern. OpenCV's undistort() function uses these patterns to return an RGB image free of lens distortion. Scipio converts this image to hue-saturation-value (HSV) format which is better suited to color detection in varying lighting settings. A histogram back projection is performed on the HSV image to filter out undesirable pixels such as grass, barrels, and mud. A series of training images determine the colors that should be ignored by the line detection processes. The image is then converted to grayscale, and high-frequency luminance noise is filtered using Gaussian blur.

The image is then thresholded using a global value based on the image mean brightness, so that all pixels below that value are filtered out. The resulting histogram is then equalized to use the entire 0-255 scale instead of only the upper half. An adaptive threshold is then run on the image in order to adjust for lighting differences in different regions of the image.

The final image contains only the white lines to be followed. Since the resolution of the camera is 1080p the lines each contain thousands of pixels. The algorithm assumes that all lines are roughly the same thickness and forms topological skeletons of the lines based on the images. This decreases the amount of information that must be processed and sent to other nodes. The skeletons are obtained from the line images by performing morphological operations such as erosion and dilation on the lines iteratively until only the skeletons remain. The resulting image contains pixels depicting only the line skeletons. Figure 6 shows the progression of image processing. The top window shows the original image, while the bottom window shows the final processed image only containing lines.



```
adaptive_threshold_block_size_trackbar   181
blur_size_trackbar                        50
global_threshold_trackbar                100
hue_low_trackbar                          20
hue_high_trackbar                         50
file_number_trackbar                       6
backprojection_threshold_trackbar         50
```

```
ormalized brightness:  0.4172290560
ime elapsed:  0.0759600271149
ormalized brightness:  0.5437163709
ime elapsed:  0.0601799400068
ormalized brightness:  0.5437163709
ime elapsed:  0.0039061737061
ormalized brightness:  0.47888300678
ime elapsed:  0.0939450263977
```

Figure 6. Progression of Image Filtering

The images are then converted into real-world spatial points. The pixels are measured from the bottom left corner of each image, which makes the calculations easier than those using measurements taken from the top left corner. Angles are measured from the horizon in the image, represented by a solid horizontal line skeleton. Because each pixel corresponds to an angle difference, the angle difference can be multiplied by the vertical pixel difference between the horizon line and the pixel of interest. The angle is decreased by ninety degrees and used to calculate the vertical component of the radial distance. The result is a function of θ which is the height of the camera multiplied by the tangent of 90 subtracted by θ, where θ is the angle previously calculated. To find the horizontal component, the vertical component is multiplied by the ratio of the pixel's horizontal measurement to the dimension of the picture multiplied by a constant. With these points stored in an array, a helper function creates and publishes a point cloud object.

Odometry

Scipio uses odometry based on multiple data sources to track where it moved relative to the starting position. Currently, two nodes provide position data, which is combined using an extended Kalman filter to produce odometry data. These two nodes are a custom-written dead reckoning node that uses encoder data and a modified ROS GPS driver node. In the future visual odometry will be included as another source of position data for the odometry extended Kalman filter.

Since the rotary encoder data suffers from creeping error in the long term due to slippage and GPS data suffers from low refresh rate and low accuracy , neither data source can be used alone to determine where the robot is. By combining both sources through an extended Kalman filter position data with high accuracy and low creeping error can be obtained at a high refresh rate. ROS has an extended Kalman filter node that receives messages from multiple data sources and publishes odometry messages, which the Navigation stack uses to determine Scipio's position in the map.

Mapping

As Scipio moves through the course, it creates a map using ROS's SLAM gmapping library. SLAM builds this map using laser scan and odometry data. To run gmapping both the LRF and motor controller drivers must be initialized. The map contains points depicting objects detected and depicts progress made toward the goal relative to the starting position. This information is saved to disk for later use when running the same course multiple times.

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 7 shows move_base's interactions between sensor input and command output.

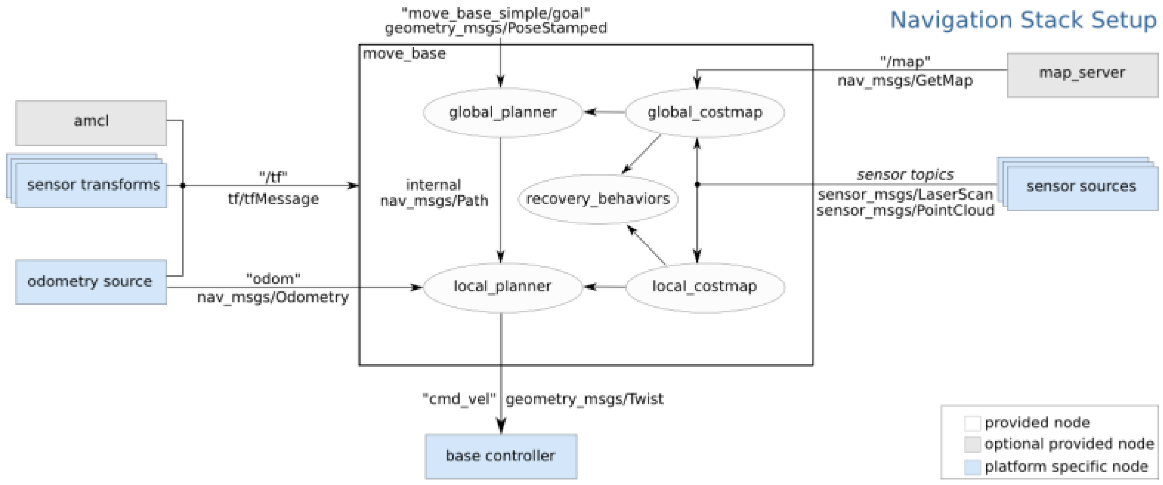


Figure 7. ROS Navigation Stack Structure*

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 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 8. 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.

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

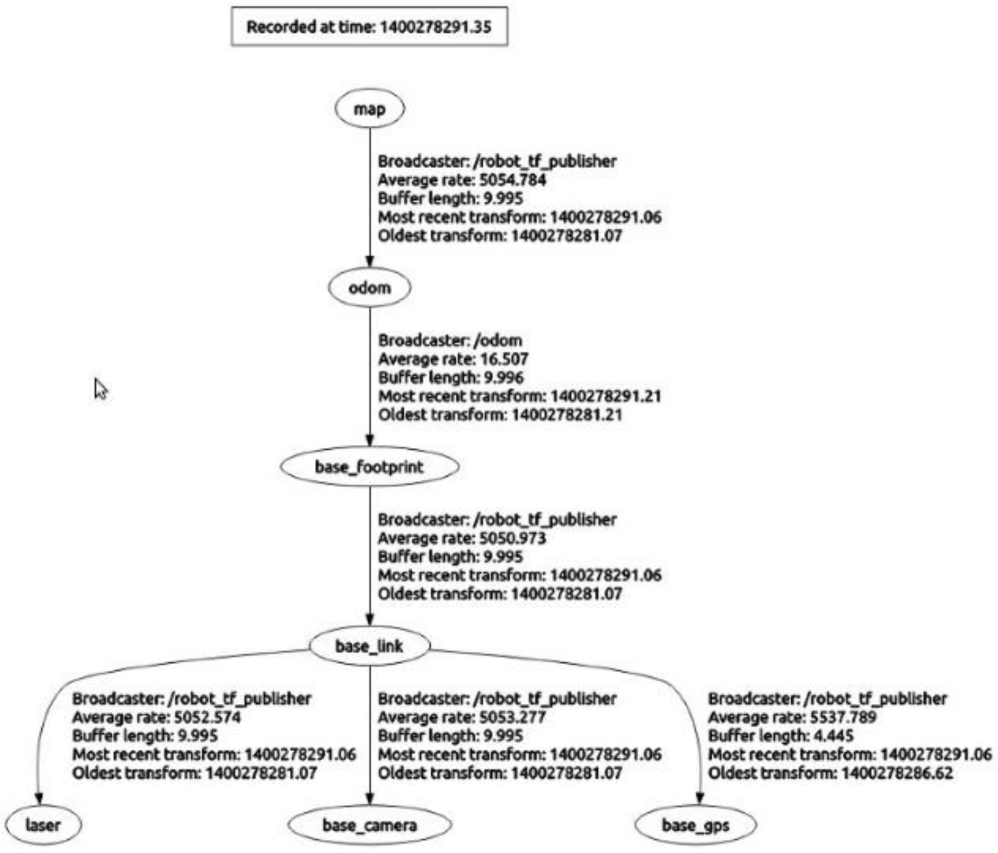

Recorded at time: 1400278291.35
map
Broadcaster: /robot_tf_publisher
Average rate: 5054.784
Buffer length: 9.995
Most recent transform: 1400278291.06
Oldest transform: 1400278281.07
odom
Broadcaster: /odom
Average rate: 16.507
Buffer length: 9.996
Most recent transform: 1400278291.21
Oldest transform: 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 8. 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 goal. 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 ultimate goal 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)

The JAUS protocol allows for communication between autonomous systems. 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 using JAUS++. The node listens for and responds to information requests, and sends the appropriate command to the motor controller.

Testing and Simulation

ROS contains a number of testing tools that greatly speed development and eliminate bugs early in the process. RVIZ, shown in Figure 9, 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 integrating all of the systems.

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.

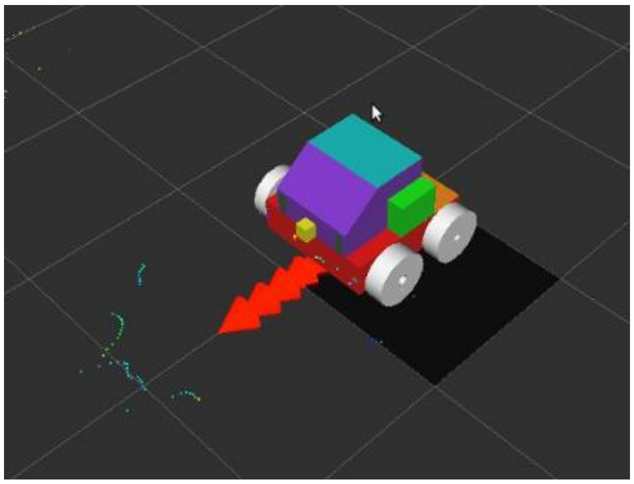


Figure 9. Scipio Modeled in RVIZ

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 operating the robot, allowing other team members access and decreasing time spent charging batteries.

SYSTEM SAFETY

Ensuring vehicle safety ranked highly during the design process, the team considered safety measures 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 System 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 E-Stop 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 degrees. Testing showed that Scipio was able to easily climb a 45 degree incline, significantly greater than any incline on the IGVC course.

Reaction Time

Taking images via the webcam and processing them to detect lines is the slowest process in navigation. Each image takes 90 ms 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. The laptop batteries last an average of 4 hours. The goal was to achieve 2 hours of runtime.

Obstacle Detection Distance

The LRF specifications state that it will detect objects at distances up to 4 meters indoors and over 3 meters outdoors. Testing revealed that the laser rangefinder was able to detect objects as distant as 3.96 meters in direct sunlight, exceeding the expectations.

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 10. If a recovery behavior cannot be achieved the system will abort the navigation to avoid any further unwanted movement.

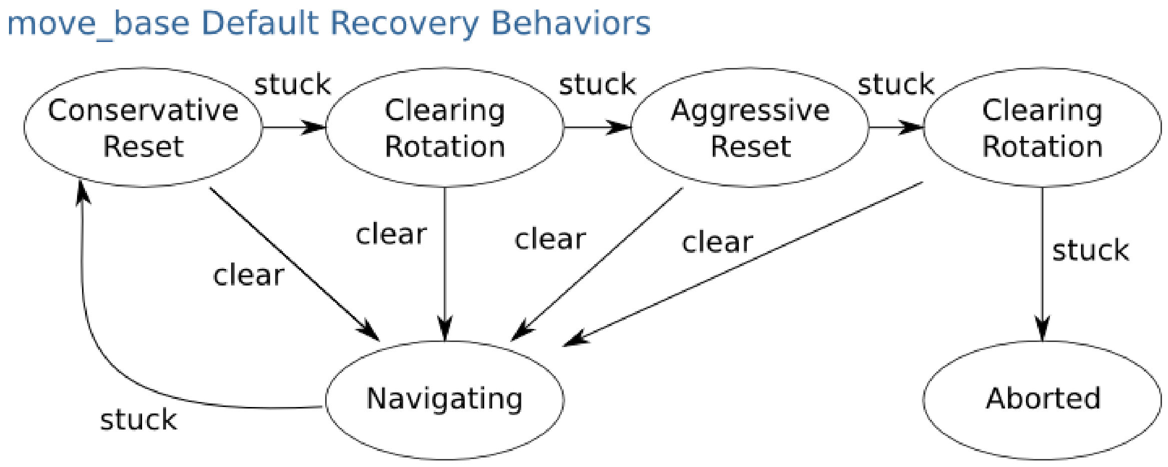


Figure 10. move_base Flowchart[*]

Positional Accuracy

Scipio now uses the Hemisphere V103 GPS Compass, which is accurate to 0.6 meters. This unit also contains a precision compass and gyroscope, which are 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
Laptop: HP Elitebook 8770w	Software	1	3729	0	Hewlett Packard
Laser Range Finder: SICK Tim 310	Object Detection	1	2000	0	SICK - Donation
Motors: Palmer Industries 200	Drivetrain	2	1100	0	Palmer Industries
Webcam: Logitech C310	Line Detection	1	53.99	0	CDW
Wheel Encoders: US Digital HD25-1000	Position and Velocity Determination	2	681	681	US Digital - Donation
Wireless Tranceiver: RadioTronix Wi.232 928 MHz		2	119	0	Mouser
Circuit Elements (Copper Boards, Microcontrollers, etc..,)		N/A	100	100	Jameco + Mouser
Switches, Wires, Crimps		N/A	100	0	Jameco + 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, Gear-boxes, 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				**$ 16,617.99**	**$ 2,268.00**	

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 in 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 switch to ROS and a Roboteq motor controller 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.

The engineering design documented in this report and implemented into this vehicle by the current student team is significant and equivalent to the credits that would be awarded in a senior design course.

Dr. Miloš Žefran, mzefran@uic.edu

Signature: Miloš Žefran