ARMSTRONG: YALE'S 2014 IGVC DESIGN REPORT

Yale University
Alejandro Carrillo, B.S. EE '16
Kevin Abbott, B.S. EE, B.S. CS '16
Jason Brooks, B.S. CS '16
Soham Sankaran, B.S. CS '17
Philip Piper, B.S. EECS, B.S. MENG '16
John Solder III, B.S. EE '16
Han Zhang, B.S. EE '17
Joseph Zinter, Ph.D: joseph.zinter@yale.edu
Professor Roman Kuc: roman.kuc@yale.edu
COL Robert W. Sadowski: robert.sadowski@usma.edu

INTRODUCTION

Yale Undergraduate Intelligent Vehicles is excited to enter the Intelligent Ground Vehicle Competition in 2014 with its inaugural vehicle, *Armstrong*. Founded in August of 2013, YUIV aims to engage computer science and engineering students in sophisticated robotic vehicle projects. Yale first attempted the Intelligent Ground Vehicle Competition in 2003 with *The Yale Jeep*, but did not actually compete. Ten years later, we have picked up the mantle and are ready to represent Yale for the first time in IGVC. *Armstrong* is the culmination of an eight-month, 1,790 man-hour effort by seven students to build a UGV that is capable of handling the rigor of the IGVC course.

DESIGN PROCESS

Our team members, while almost entirely EECS majors, possess a breadth of skills ranging from software design to metal fabrication. Rather than organizing labor into strict subteams based on discipline (electrical, software, etc.), we chose to distribute tasks by amassing a large TODO list prioritized by a critical path, and meeting bi-weekly for half an hour to determine who would tackle which issues. Multiple team members were thus able to simultaneously broaden their skills in machine vision, mechanical design, and other disciplines that are typically delegated instead to a specific subset of people. The team President acted as a systems coordinator by organizing communications, synthesizing ideas into documents, and maintaining our information

repositories.

Our information repositories consist of a Google Drive where we store documents pertaining to project management and finances, a Dropbox where we store CAD drawings, media, and data logs, a Google Site where we organize links pertaining to various subsystems, and a Github repository for our codebase.

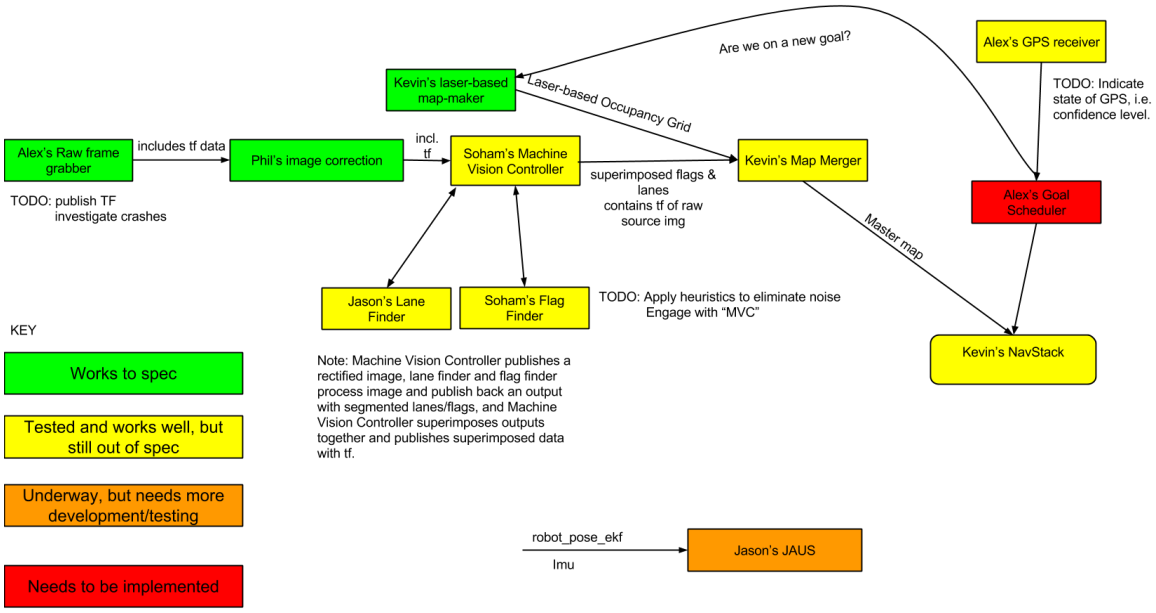


Figure 1. Screenshot Of Central Project Management Diagram.

In order to keep track of project progress, clarify task delegation, and promote team consensus on the "bigger picture" of our software stack, we maintained an original annotated graph of everyone's software tasks and how they communicate with each other. Each node has an assigned developer, and is color coded by its maturity relative to the specification. Keeping task delegation, progress information, and system workflow information in one diagram was an efficient and elegant solution.

Testing

The YUIV team utilizes a continuous integration (CI) server run on *codeship.io*, a cloud-based platform. The CI server is predominantly used to test code used for the Interoperability Profiles (IOP) challenge. Development for the IOP challenge was completed while following the idea of test driven development - tests for each core feature were written before the functionality was implemented, and only when those tests completely passed was the next feature worked on. The CI server allowed us to ensure all tests passed before the IOP code was deployed to our production computer on *Armstrong*.

Furthermore, to test new features and changes, as well as the numerous edge cases, without needing to deploy to the robot and put it in a potentially dangerous situation, we relied on ROS's extensive simulation suite. This allowed us to create simulated worlds and feed the corresponding sensor inputs into our system to determine its stability. This was extremely useful for testing the autonomous control system of our robot, which ensured both the safety of our system and that of our team.

Because of the relatively high power nature of our drive train, we elected one team member to serve as our Safety Captain and robot test coordinator. The Safety Captain maintains a checklist that, if followed, ensures that we have taken every precaution to prioritize the safety of ourselves and those around our vehicle. The Safety Captain is additionally the only person who enables the tractive system.

Mechanical

While our seven-person team consists predominantly of EECS majors, we went to great lengths to learn the required CAM and fabrication skills necessary to design and build our vehicle. Our chassis is designed primarily around our drivetrain, a modified Segway RMP 200. The Segway RMP 200 is an all-terrain vehicle capable of self balancing control, speeds up to 6 mph, and driving periods under load for over eight hours.[1] We chose the Segway RMP 200 for its strong mechanical performance characteristics, robust closed loop motor speed control, and easy system integration.

Using the RMP 200 saved us a great deal of precious design and fabrication time. While we initially wanted to use "Balance Mode" for its slim footprint and zero-turn capability, we decided to opt instead for using non-balancing "Tractor Mode" and installed a free caster wheel in the rear of the vehicle. Although Balance Mode offers many alluring benefits, we chose "Tractor Mode" primarily for its superior safety factor and straightforward reliability.

The bulk of the vehicle chassis is aft of the drivetrain. Enclosed between our main side plates is a waterproof electronics bay, where we store our computer, our GPS board, our inertial navigation system, and our power distribution system. Projected from the side plates is a rigid tower that encloses our touch screen monitor and raises our omnidirectional imaging system to its operational height. While it is critical for our camera to be as high as possible, it is also critical that the camera sway as little as possible during normal vehicle operation. We also required that the tower be as light as possible in order to ensure a low center of mass. Thus, we put great consideration into minimizing the tower's weight while maximizing its rigidity. The tower is reinforced by a large aluminum sheet that minimizes lateral oscillation during travel and secures the frame of the monitor to the chassis. We additionally desired a center of mass located ⅔ from the rear of the robot. Keeping the center of

[1] http://rmp.segway.com/downloads/RMP_200_Specsheet.pdf

mass $^2/_3$ from the rear caster balances the benefit of maximizing weight on the drive wheels for traction with keeping enough weight on the rear caster so that our robot will not pitch when accelerating quickly. To meet our center of mass goals, we chose to fabricate the chassis with rigid $^5/_8$" aluminum plates and installed the camera tower as close to the rear caster wheel as possible. To keep the electronics bay waterproof while tuning the weight of the system in our model, we machined blind triangular pockets.

At the fore of our drivetrain is a Hokuyo laser range finder rigidly mounted to our gearbox. The chassis design and fabrication process culminated in finishing the aluminum with a translucent powder coat to provide durability and aesthetic touch.

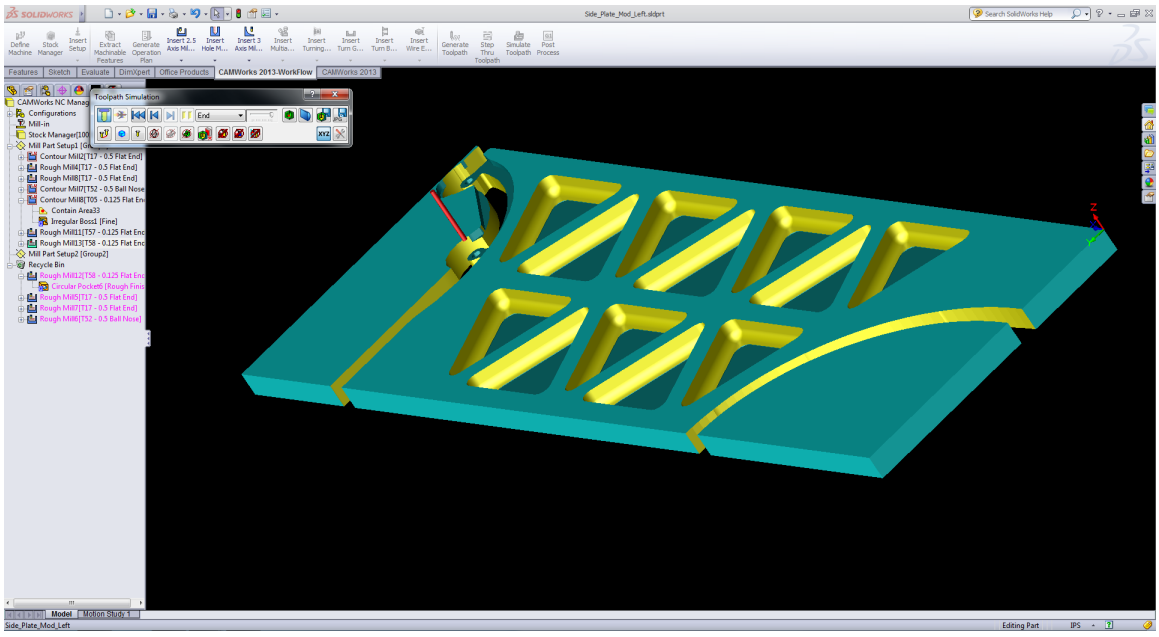


Figure 2. Camworks For Milling Of Sideplate.

The electronics bay is kept waterproof, visible, and accessible with a polycarbonate lid that is in two pieces. The fore subsection of the lid encloses the computer, GPS, and power distribution system, and the aft subsection encloses the battery for easy access. The Payload is strapped to the fore subsection of the lid, and can be slid along the major axis of the robot to tune the center of mass if necessary. In between the two lids is a thin panel of waterproof electrical panel mount connectors for communicating signals and power to the monitor, the camera, and the switch interface.

Electrical

Our system uses two main power sources: a pair of 72V NiMH battery packs for the Segway drive base, and a 12V 22Ah SLA for the control system. The two 72V

batteries are rated to last about 8 hours, allowing the Segway to drive 10-12 miles.[2] Experimentally, we have found the batteries to last 3 hours, which allows our system to travel around 4 miles. Since the drive base is entirely separate from the rest of the system, we were able to isolate our E-stop response to only the Segway base (Figure 3) allowing us to safely stop the robot without needing to shut off the control computer and sensors.

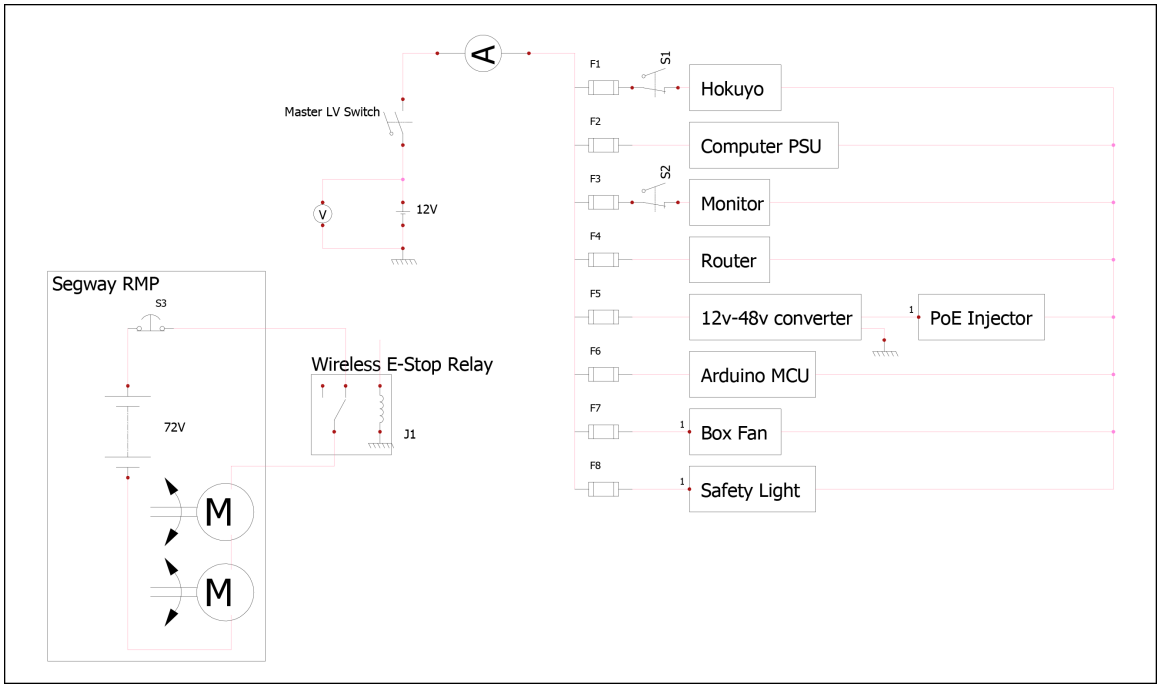


Figure 3. Electrical System Diagram.

Our 12V system (Figure 3) should last about 3 hours given our system's power consumption of 9.65 Amps under full load (Table 1). To allow us to change batteries without needing to reset the system, we implemented a simple "hot swap" battery system by splitting our power cable into a wye.

Overall, our system can drive for approximately 3 hours and travel 4 miles on a single charge.

CONTROL SYSTEM

Sensors

Camera. For both white line and flag detection, we use an omnidirectional imaging setup featuring an Allied Vision Tech Mako G-223 equipped with a Kowa 35mm lens. The Mako is a rugged machine vision camera with a high frame rate and high resolution. It features both auto-exposure and auto-white balancing, which is optimal for

[2]http://rmp.segway.com/downloads/RMP_200_Specsheet.pdf

Table 1. Power Consumption.

Item	Voltage (V)	Current (A)	Power Consumption
Computer	12	3	36
Monitor	12	4	48
Hokuyo LRF	12	0.37	4.44
Mako Camera	12	0.23	2.8
Linksys Router	12	0.50	6
GPS Receiver/Antenna	3.3	0.55	1.8
Average Total:		9.65	99.04
Peak Total:		14.65	171

this outdoor challenge where varying light levels is a problem. Fitted with a lens whose focal length we chose based on our camera's working distance, the Mako allows us to focus solely on our hyperbolic mirror, thus eliminating wasted pixels in our images.

Laser Range Finder. To allow the robot to map its environment, we are using a Hokuyo UBG-04LX Laser Range Finder, which takes planar scans of a 180° area in front of the robot. The Hokuyo LRF has a range of 5 meters and is able to take scans of 0.36° resolution at a rate of 40 Hz.[3] We considered using a larger SICK LRF, which would have offered a range of about 12 meters, but found it to be too large, too slow, and taking too low resolution scans. Overall, the Hokuyo has worked excellently at mapping our robot's environment; and although it is unable to determine an accurate frame size of the sawhorses or A frames, this is easy to compensate for in software with small dead zones around obstacles.

IMU/INS. To attain accurate attitude data on our robot, we are using a Vectornav VM-200 IMU/INS, which offers 0.5° accuracy in roll, pitch, yaw data. The yaw data allows us to accurately determine the robot's current pose in the world, which is needed to map the robot's surroundings and determine a path to the next waypoint. The accurate pitch and yaw data allows us to determine when the robot is tilted and therefore limit the range (for pitch) or angle bounds (for yaw) for which we consider laser scans in order to isolate the ground plane from the map.

GPS. In order to accurately locate and move between waypoints, we are using a Novatel OEM628 GPS receiver and accompanying GNSS antenna, which when combined with the Omnistar VBS service allows us to achieve .6 meter accuracy. Omnistar VBS is an SBAS system that uses a separate satellite as a virtual base station for more accurate GPS readings. The achieved .6m accuracy is perfect given the 2m circumference of waypoints on the field.

[3] http://www.hokuyo-aut.jp/02sensor/07scanner/download/products/ubg-04lx-f01/data/UBG-04LX-F01_spec1.pdf

Software Architecture. At the core of our software stack is Robot Operating System (ROS). We were attracted to ROS because of its rapid system integration, easy debugging, and broad community support. Many systems and devices, such as our drive base, LRF, and IMU, already have 3rd party ROS nodes written, which allows us to integrate them into our system with little overhead. In addition, ROS comes standard with many useful debugging tools. We used Rviz to visualize our robot's laser scans and position, and the outputs of our obstacle mapper and path planner. RQT image viewer was useful in viewing the streams of our image processing pipeline at at any point along the system; and the simulation tools Stage and Gazebo were extremely helpful in testing numerous edge cases by creating simulated obstacles and laser scan data to process through our system. The core feature of ROS that was most useful is the ability to "bag" sensor data, thus allowing us to test changes in our system on the same set of laser scans, image streams, and odometry and attitude data.

System Design

Our AutoNav Challenge strategy starts with the Goal Scheduler. The user first inputs a list of GPS coordinates in the order that the robot should traverse them. While we considered implementing an algorithm to optimize the order in which the robot should traverse goals, we decided that it is simpler for the system user to exercise his or her judgement in organizing the schedule. With the scheduler populated, it publishes a navigation goal relative to the robot's coordinate system to the Path Planner, which is based on the ROS Navigation Stack. The Path Planner then runs a search algorithm through the latest map.

Our maps are generated by merging a laser-based map with a camera-based map. Our laser mapper detects objects by using a Hokuyo UBG-04LX, which is a rapid horizontally scanning time-of-flight sensor. Our camera-based map detects lanes and flags using a slower Mako G-223 machine vision camera aimed at an upright hyperbolic mirror. In order to synchronize the parallel laser and camera mapping processes, we sample the robot's pose estimate when we clock in a raw image. We pass this pose estimate along to every downstream node that utilizes image data until the image processing pipeline reaches the Map Merger. Although the image-based mapping pipeline is the critical path, the Map Merger can retroactively apply segmented lanes to our merged map because the Map Merger receives the original coordinate location of the raw input image.

The image pipeline has additional parallel processing considerations. We chose to separate the tasks of segmenting colored flags and lanes into concurrent processes, as each machine vision procedure treats the common input image differently. It is possible for each individual image processing node to pass on its information to the map merger. However, because the orientation of the image is always fixed to the orientation of the robot, and the robot moves relative to the global (merged) map, the

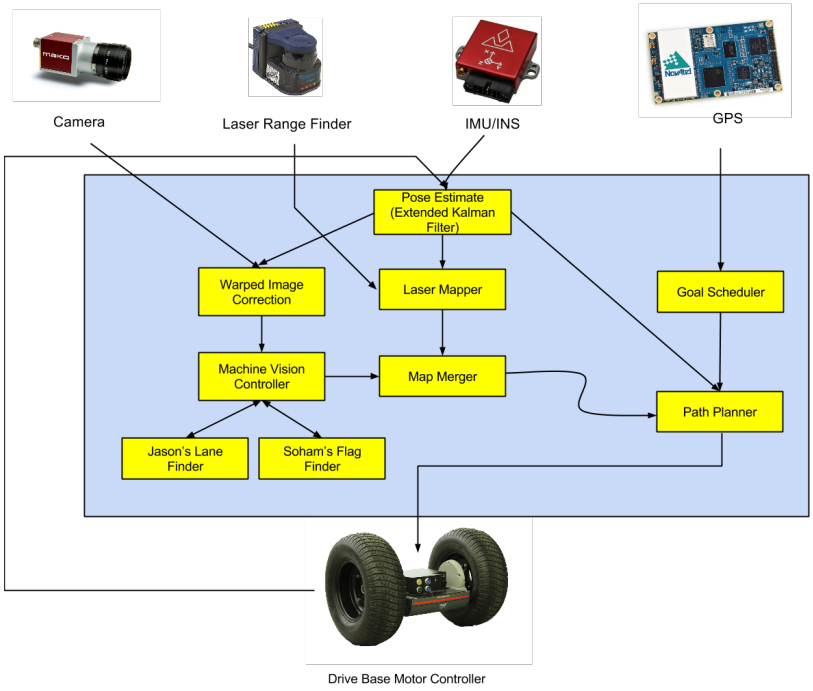


Figure 4. System Block Diagram.

Map Merger must rotate the output of the lane and flag detectors before "stamping" the lanes and flags onto the world frame. Allowing the lane and flag detectors to independently publish data to the Map Merger would require two image rotations per input image, as the lane detector is faster than the flag detector and will publish its data to the Map Merger before the flag detector. Thus, we implement a Machine Vision Controller that takes in a rectified image with the raw image pose, publishes the image to the separate lane and flag detection tasks, bitwise-ANDs the output of each task, and passes this combined image with the raw image pose to the Path Planner.

User Interface

Our emphasis on safety, aesthetics, and utility extends to our UI design. On the top plate of the monitor and camera tower is a small switch panel, containing an E-Stop, a master 12V on/off rocker switch, a computer power pushbutton switch, and a toggle switch to turn off the monitor during heats in order to save power. The buttons to energize the Segway drivetrain are hidden underneath the Hokuyo mount at the front of the robot in order to minimize accidental enabling of the drivetrain. The E-Stop will immediately open the drive train power circuit and leave the robot to safely coast.

Below this top control plate lives a 17" touchscreen, sunlight readable monitor from AIS Pro. Both the ability to read the monitor in the sunlight and its touchscreen

capability allow us to use the monitor as the lead interface for the entire system. On bootup, the user has the ability to choose between either a IOP Challenge interface or a normal navigation course interface. The IOP Challenge interface allows the user to view rich data analytics about the current state of the robot's networking functionality, as well as information pertaining to the Segway base status. The navigation course interface functions as both an input and output system. For input, a user has the ability to configure camera settings (white balance, exposure, etc.), teleoperated settings for testing (joystick speed), and general system information. For output, our control system allows us to view CPU usage, Segway data (velocity), and position information from a ROS node.

THE VISION SYSTEM

Vision capture and object detection represents a major component of Armstrongs software pipeline. Using our implementation, we are able to run the entire image process at 5Hz.

Raw Image

An image grabber is implemented at the very start of the image pipeline. The camera is first configured using a set of parameters, which include white balance, exposure, frame rate, and acquisition mode (single frame versus continuous capture). This configuration can take place through a GUI provided by Allied Vision Tech, or through an original configuration program in our raw image ROS node.

After configuration takes place, raw images are captured using the pymba python library, which is a wrapper for Allied Vision Techs Vimba API. This image is then stamped with the robots pose (a combination of position and orientation information) relative to the global map. With our sensor, we produce images with a resolution of 2048 x 1088, which allows the image processing pipeline to retain rich information with obstacle and line information while segmenting out the sky.

Unwarping Raw Image Data

At the heart of our lane and flag detection strategy is our choice of an omnidirectional camera system. By coupling a camera with a 35mm Kowa lens to an upright hyperbolic mirror, we can produce 360 degree panoramic images of the area around the robot.

The primary advantage of this system is the elimination of certain edge cases that can plague a non-omnidirectional approach. With a top-down perspective, there is less risk of inadvertently mapping obstacles with white features as lanes that could critically impede the quality of our global map. The original image captured is not perfect due to inconsistencies in the manufacturing of the mirror (it is very difficult to create a perfect hyperbola out of shiny glass). Since the raw image data is

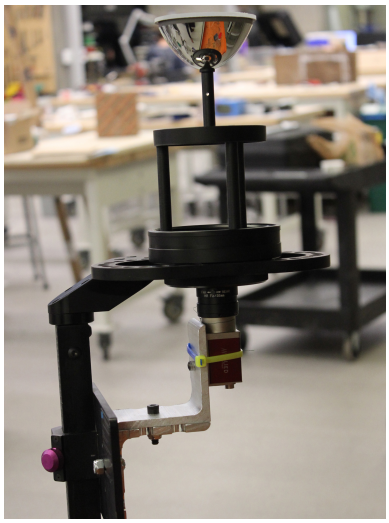


Figure 5. A Mako 223 Camera With a 35mm Lens Pointing at a Hyperbolic Mirror.

warped from the mirror, images need to be normalized before feature detection and processing. Doing so is a two phase process — first, a calibration process takes place, followed by running a ROS node that unwarps the image.

Armstrong uses an original calibration program written in C++. This program uses a polynomial regression based on known distances in a sample captured image. To get these known distances, we use a 16'x16' giant chessboard with 24" squares. Therefore, we know the distance between two adjacent intersections of white and black tiles is 24".

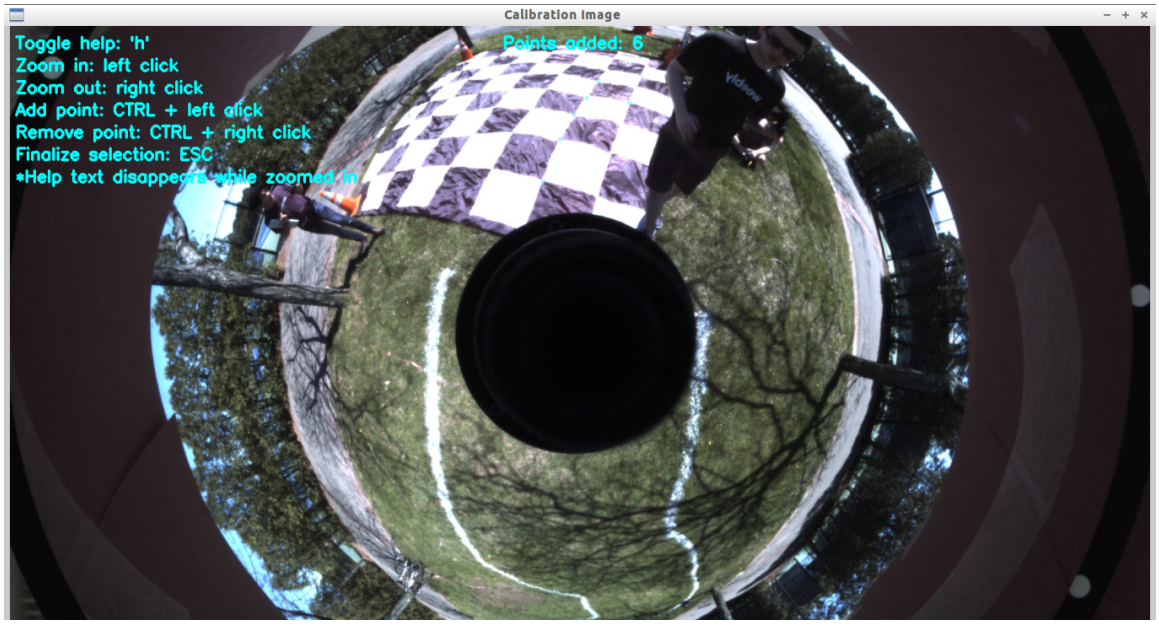


Figure 6. Our Custom Program for Calibrating Images.

The original idea was to use the C++ version of the OpenCV image processing library to detect the chessboard and automatically find the distances between tiles. Instead, Armstrong uses a UI that allows points on an image to be hand selected down to within 2 pixels. After these points are selected, we assume the distance between the two points closest to the center of the mirror/image to be 24 apart, and create a polynomial regression that adjusts the distances between every set of two adjacent points to be the number of pixels representing 24".

The effect of a world image reflected by a hyperbolic mirror is to displace points exponentially as a function of radius from the center of the mirror. Our custom image unwarping program first generates a lookup table TransformMatrix that maps datapoints in warped image A at coordinates (i, j) to coordinates (k, l) in rectified image B. The value of the element $TransformMatrix[i][j]$ is the vector $< k, l >$. Expressed in pseudocode, $< k, l >= TransformMatrix[i][j]$.

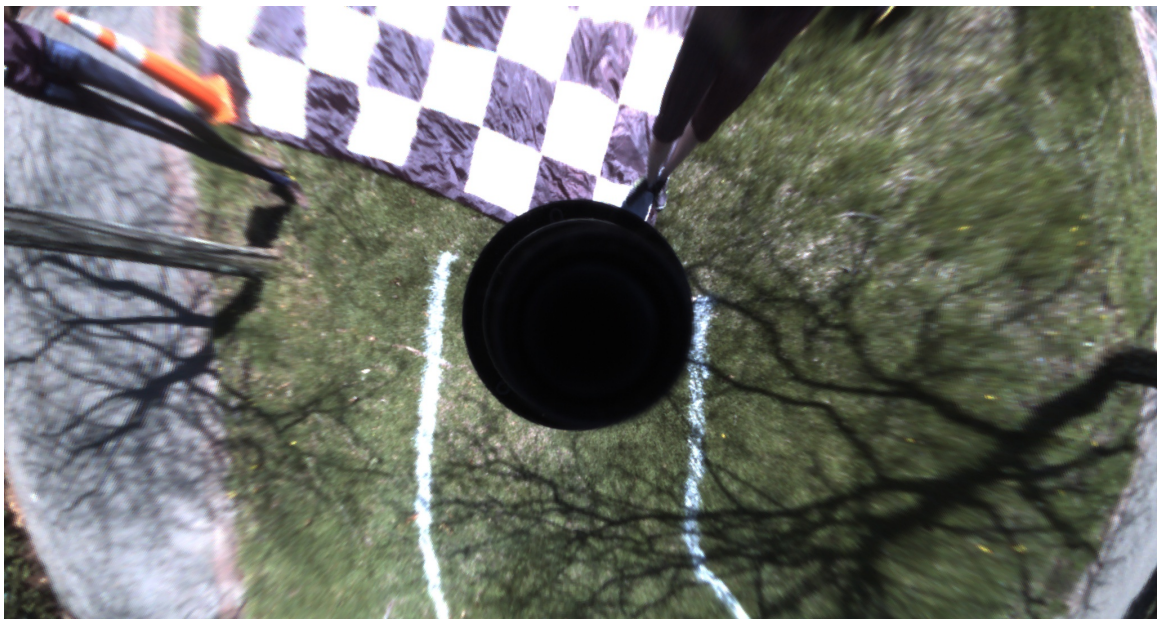


Figure 7. A Sample Output Image From Our Image Rectification Software.

Let $TransformMatrix[0][0]$ be in the upper left-hand corner of the matrix. In order to generate $TransformMatrix$, for each pixel $< i, j >$ in the input image size we find the magnitude r and angle θ of the vector formed by the subtraction of pixel $< i, j >$ with O, where O is the center, or origin, of the hyperbolic mirror in the raw image. Expressed formally, $O =< \frac{width}{2}, \frac{height}{2} >$. Relative to the center of the mirror, the hyperbolic mirror translates world coordinate $< r, \theta >$ to warped point $< r', \theta >$. To retrieve r given r', we let $r = func(r')$, where func is a polynomial function of r' whose coefficients are set by the calibration program. To convert polar coordinate $< r, \theta >$ back to rectangular $< k, l >$, we simply let $k = rcos\theta + (width/2)$ and $l = -(rsin\theta - (height/2))$.

With k and l thus calculated and stored in a lookup table, we can simply map $A[i, j]$ to $B[k, l]$ with the high-speed operation $B[k, l] = A[i, j]$.

This transformation takes advantage of the symmetry of the mirror, namely that in cylindrical coordinates: $z = hyperbola(r, \theta) = hyperbola(r, \theta')$ for all θ'.

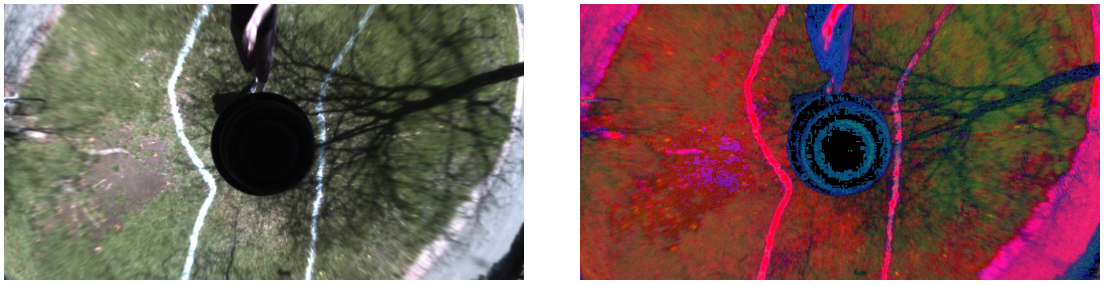
Lane Detection

Armstrong uses a four stage image pipeline to detect and filter white lanes as they appear on grass:

1. Conversion and Normalization
2. Filtering
3. Clustering
4. Cluster Analysis

The pipeline utilizes OpenCV for conversion and processing.

In the first phase, a ROS image is read into python, and converted to the HSV color space. Representing Hue, Saturation, and Value, the HSV space is a cylindrical-coordinate representation of points found in the RGB color model. The HSV space was chosen since, unlike RGB, the intensity of pixels in the image is a distinct channel from the color value of the pixels. As a result, it is much easier to normalize the image, and ultimately remove certain features like shadows, dead grass, and obstacles. After this conversion, shadows are normalized by using the Yin-Shi et al.[4] Shadow Removal Algorithm for HSV color spaces - a ratio is taken between H, S, and V in the foreground and background to determine if certain pixels contain shadows.



(a) Original Image. **(b) Image in HSV Colorspace.**

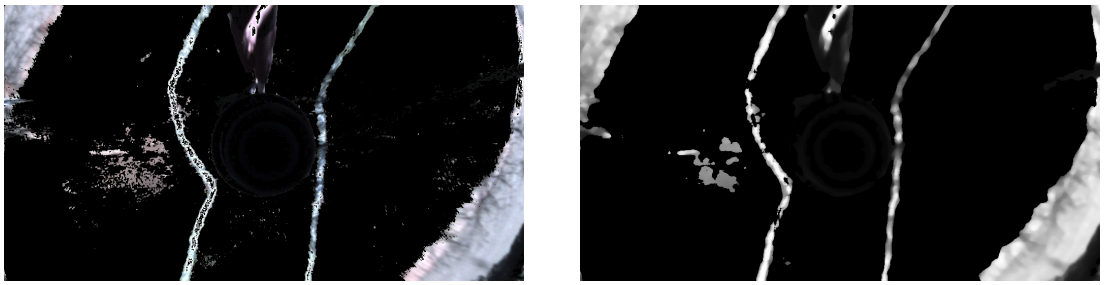
Figure 8. Phase 1 of Image Pipeline.

[4]http://www.atlantis-press.com/php/pub.php?publication=icmt-13&frame=http%3A//www.atlantis-press.com/php/paper-details.php%3Fid%3D10471

Phase two aims to filter and segment white lane data from the rest of the image, which predominantly includes obstacles, grass, trees, and people, among other objects. To start, range thresholding takes place in the HSV space, where green and brown hue values are identified at all saturation and brightness levels. After this identification, a binary mask is created of the identified pixels, and the bitwise AND operation is used to quickly create a composite mask of the original image minus these pixels. Finally, another HSV threshold operation is run, which filters out pixels with low saturation and high brightness. This filter further isolates the white lanes from other light parts of the image. At this point, the composite image contains our lanes, along with outliers. Since the outliers take on a different geometry than the white lanes, these are possible to remove while clustering.

Before moving on, we need to also consider the removal of halation as a result of sunlight on the hyperbolic mirror. After the image unwarping process, halations appear with a very similar geometry and color structure to the lanes. We can run a similar thresholding process to remove the halations as we did to remove grass, however, we first convert the HSV image back to the RGB color space.

In phase three the algorithm begins to loosely follow a methodology proposed in Kobayashi et al.[5] First, we split the image into its blue, green, and red channels, and then use the blue grayscale channel, which prevents errors in detecting the sun (R component) and errors in detecting grass and the road (G component). Next, the image is blurred with a median filter, and we cluster the remaining points by finding the contours in the image.



(a) Image After Return to RGB Space. **(b) Image With Median Blur.**

Figure 9. Phases 2 and 3 of Image Pipeline.

In the last phase, we run analyses on the clusters produced to determine what is a lane and what is a false positive. This is a two part process. First, we run a bright line detector, which looks for clusters with bright centers and dark points on the outside. Second, we look at a skeleton representation of each cluster. The algorithm checks that the two lines representing the edges of each cluster are relatively parallel. If this is the case, we can assume the line is a lane. Otherwise, the skeleton representation

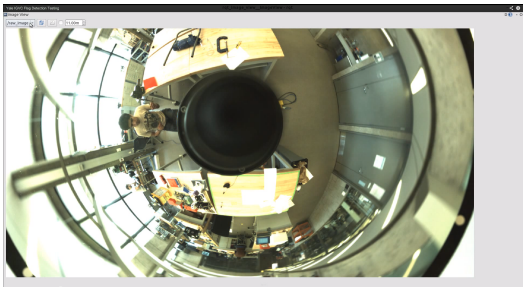
[5]http://onlinelibrary.wiley.com/doi/10.1002/eej.21193/abstract

diverges, and we know this is a false positive. A bitmap is then published to the machine vision controller with lane information.

Figure 10. Final Output Image.

Flag Detection

To detect both the red and blue flags, two separate filters are used. To start, the machine vision controller feeds an unwarped RBG image. In a similar methodology to lane detection, we extract the red channel for blue flag detection and the blue channel for red flag detection to remove excess noise.



(a) Original Flag Image.

(b) Filtered Flag Image.

Figure 11. Flag Detection.

The image is then converted to the HSV color space, where hue thresholding takes place for each flag color. A simple blob detection filter is run to remove excess noise. Finally, the algorithm looks for blobs in close proximity and equal distances. We assign a probabilistic weight to each weight, and based on these weights, output a bitmap to the machine vision controller.

OBSTACLE DETECTION AND MAPPING

Laser Mapper

In our first iteration of an obstacle mapper, we attempted to use Simultaneous Localization and Mapping via the ROS package Gmapping, but quickly found that it was unsuited for our needs.

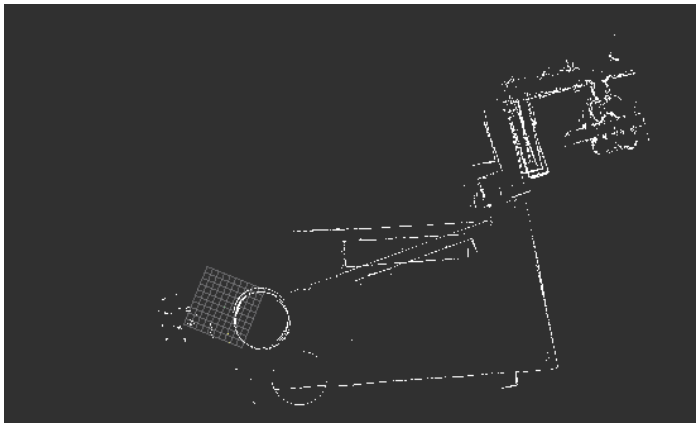


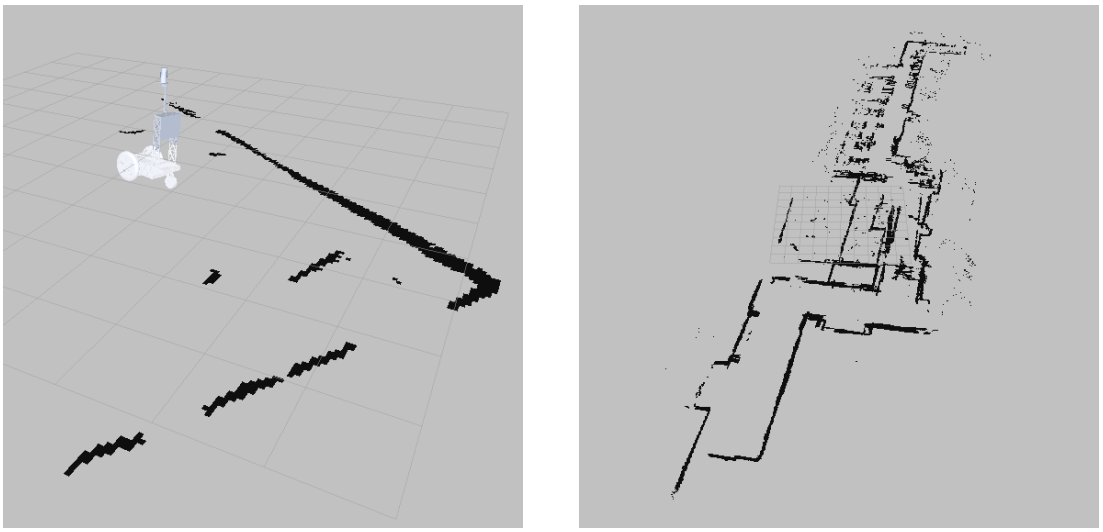
Figure 12. Skewed Gmapping Data.

Given the repetitive nature of the obstacles and lack of distinguishing features on barrels–such as corners–the Localization aspect of SLAM based mapping resulted in a heavily skewed map of the robot's world, as the robot would continuously recentralize its map of the world on different "identical" barrels.

Fortunately, given the high accuracy of the Segway's odometry data and our Vectornav IMU's attitude data, we were able to develop a mapping algorithm that produced maps with little skew or drift, even in large environments with long travel distances. To do this, we create a probabilistic occupancy grid of the the robot's world, and plot the planar Hokuyo scans based on the robot's current position in its environment. Then, to reduce the effect of noise and non-static objects, when the LRF detects an obstacle it merely increases the probability of an object being located in the corresponding occupancy grid location, therefore requiring numerous laser scans on a location before an object is said to be there definitively. Similarly, if an LRF scan shows a clear point in a previously marked location, it will decrease the probability of an obstacle being there to allow for the decay of transitive objects–such as people–within the map.

Map Merger

Since the camera setup provides a near top-down view, once the lines and flags with interpolated obstacles are isolated, it is simple to make a complete map of the environment. By correlating a pixel location in the line and flag image with a real-world distance from the center of the camera, the lines and flags can be combined



(a) LRF Scan Plotting. **(b) Map of Yale Engineering Space.**

Figure 13. *Armstrong's* Mapper Implementation.

with the laser mapper's occupancy grid to create a complete obstacle map of the robot's surroundings.

NAVIGATION

Goal Scheduler

Once we have a relative map of the robot's environment, it is possible to relate the absolute GPS location of a waypoint with a pose in the robot's frame, which is the input into the path planner. To do this, our goal scheduler reads a GPS coordinate from a list of waypoints input by the user and then calculates the goal pose for the robot based on the robot's current GPS location and pose.

As the robot moves towards a waypoint, the goal scheduler continuously refreshes its GPS location in order to update the goal pose and to determine when the robot has hit a waypoint. For the waypoints in No Man's Land, when the goal scheduler determines it has reached a waypoint, it simply moves on to the next input. When the goal scheduler determines it has reached the waypoint at the entrance to No Man's Land, it signals then mapper to do a reset, because up to this point there have been people moving along the robot in No Man's Land who may have accidentally been mapped and therefore are false positive obstacles in the robot's map. Then to ensure that the robot will not try to reenter No Man's Land after reaching the last waypoint, the goal planner signals the mapper to place a line of obstacle behind the robot when it reaches the waypoint at the exit of No Man's Land.

Path Planner

Once the goal scheduler provides a pose goal for the robot, our system uses the ROS navigation stack to find a path from the robot's current location to the next waypoint and carry out this path by sending the Segway's controller the appropriate velocity commands. To do this the navigation stack is split up into two main components: the global planner and the local planner.

The global planner, which uses the map produced by the map merger to create a costmap for traversing any part of the robot's environment, is responsible for creating a complete path from the robot to its next waypoint. For our global planner, we are using the Anytime D* algorithm, which is an "anytime" variant of the D* lite path planning algorithm, meaning it can run under heavy time constraints by first producing a suboptimal path and improving upon it with more time and iterations.[6] Although it may initially produce a suboptimal path, we determined this to be the best path planner given the minimum speed requirements of the robot and that the robot is moving in an unknown environment, so it will have to frequently recalculate the path to the next waypoint as more obstacles are discovered.

The local planner portion of the navigation stack is responsible for determining intermediate waypoints along the global planner's path for the robot to move along. This is useful in situations where the robot starts out with a static map and must localize itself and rescan locally for changes in the environment; but in our case, where we are building the map as the robots moves through course, all we needed was for the robot to follow the path determined by the global planner. From this we are using an algorithm that simply interpolates pose vectors along the global path and sends the Segway controller the appropriate velocity commands to move between the positions.

INTEROPERABILITY PROFILE (IOP)

In order to enable systems interoperability and adherence to the Joint Architecture for Unmanned Systems (JAUS) set of standards, we developed a JAUS controller, which acts as a communication system between the robot's underlying control stack and external systems like the JTC. Our JAUS controller is a simple UDP server that is subscribed to the robot's state variables, and is able to provide velocity and pose information and command the robot to a new waypoint dependent on the external request received. Our JAUS controller offers a level of abstraction above the direct control of the robot, which results in a simple interface for external command of the system.

[6] http://www.cs.cmu.edu/ ggordon/likhachev-etal.anytime-dstar.pdf

EXPECTED PERFORMANCE ANALYSIS

Speed

The Segway RMP's nominal top speed is 10 mph. Experimentally, we have not observed a top speed above 7mph. The Segway RMP's motor controller has firmware that limits the Segway to 10 mph. The Segway will even actively drive against its direction of motion if a force tries to push it. Because our vision processing pipeline operates at around 5 frames per second, however, we have chosen to limit our top speed to 3mph, which we have experimentally observed to yield the fewest path planning mistakes.

Ramp Climbing Ability

Given that our drivetrain's maximum power output is 2hp, and that we have more than enough weight to enable full traction, tractive power will not be a limiting factor to our ramp climbing ability. Our objective when designing the chassis was to place the center of mass ⅔ away from the rear caster from a top view, and as low as possible from a side view. While we were successful in attaining our top-view center of mass goal, our effort to keep the height of the center of mass as low as possible was made difficult by our tall side towers and heavy monitor.

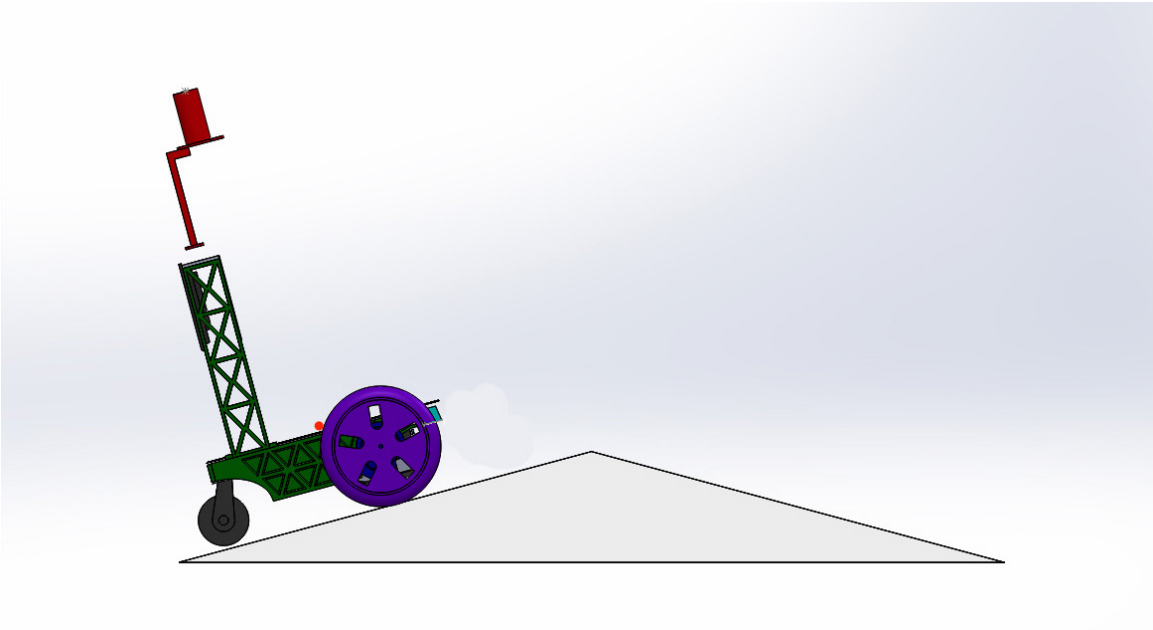


Figure 14. Red Dot Represents Center Of Mass

Nevertheless, our robot is projected to succeed in climbing and descending 15 degree inclines. We may experience an unstable condition if the robot travels the ramp at an angle to the gradient of the incline. We have made room in the electronics

bay for steel ballast plates in case we must pull the center of mass down further to be safe.

Reaction Times

While our laser-based maps are published at a rate of at least 10Hz, the fastest that our path planning program can publish a velocity command update is about 5Hz. Under ideal conditions, then, the fastest reaction time possible to a new obstacle is 200 milliseconds. This calculation does not consider frame rate of our lane and flag mapper, however, which can be as slow as 4Hz. Therefore our reaction time is projected to be within 200-250 milliseconds, which is similar to that of a human being.

Distance at Which Obstacles are Detected

The Hokuyo LRF is able to detect obstacles at a range of up to 5 m in a 180° area in front of the robot. Our omnidirectional camera is able to detect line obstacles in a 3m radius around the robot.

Complex Obstacle Performance

Our path planner is configured to know not only the footprint dimensions of our robot, but also the center of turn. This enables the planner to make intelligent decisions about how to approach oncoming switchbacks. We additionally chose the location of the laser carefully to minimize the danger of blind spots in our laser mapper. By keeping the center of the laser range finder as close to the center of turn of the robot as possible, we can ensure that our path planner will not inadvertently rotate the body robot into an unseen obstacle. Had we placed our laser rangefinder on the rear of our robot, we would have run a greater risk of swinging the chassis of the robot into an obstacle.

With the navigation system discussed thus far, there are a number of critical edge cases present in the Advanced AutoNav Course to consider. When the robot is in its starting position and the Path Planner is given its first navigation goal at the entrance of No Man's Land, the Path Planner may want to consider making an 180° turn in search of a low-cost path. Similarly, when the robot has reached the waypoint at the exit of No Man's Land and is scheduled to return to its starting position, the Path Planner may investigate a solution that requires re-entering No Man's Land. Our solution to these similar problems is to superimpose a virtual obstacle behind the robot at the start.

Given our robot's 24" drive tire diameter and 10" caster wheel, our system has little trouble with divots or potholes in the ground. Furthermore, the segway's robust

control system utilizes an extended kalman filter to maintain its velocity commands in the face of changes in traction or ground characteristics.

Accuracy of Waypoint Navigation

Our robot is equipped with a GPS system with .6m accuracy and we are continuously integrating GPS data into our goal scheduler and path planner, which means we should be able to navigate our robot to within a .6m radius around a waypoint. In our initial tests, we have found that our robot can navigate to within 1m of a waypoint, which is well below the 2m circumference of waypoints on the field.

BUDGET

Item	Cost to team ($)	MSRP ($)
Hyperbolic Mirror	400	400
GoPro HD Hero 3 + lens	400	400
Allied Vision Mako 230 Machine Vision Camera	1,800	1,800
NovAtel OEM628 GPS	1,200	2,700
SICK LMS 200	—	500
Hokuyo URG	—	2,000
Test equipment (Barrels, etc.)	400	200
Electronic cabling & components	500	500
Batteries & Chargers for control system	400	400
Touch Screen Monitor UI	950	950
VectorNav Inertial Measurement Unit	—	3,000
Development laptop	400	400
Robot Computer Control System (Mini-ITX)	700	700
Segway RMP200	—	20,000
Fabrication materials (Aluminum plate, sheet, etc)	1,000	1,000
Metal finishing job	—	600
Total	*8,150*	*35,550*

Figure 15. Estimated Cost for the 2013-2014 year. Items with a dash are donations.