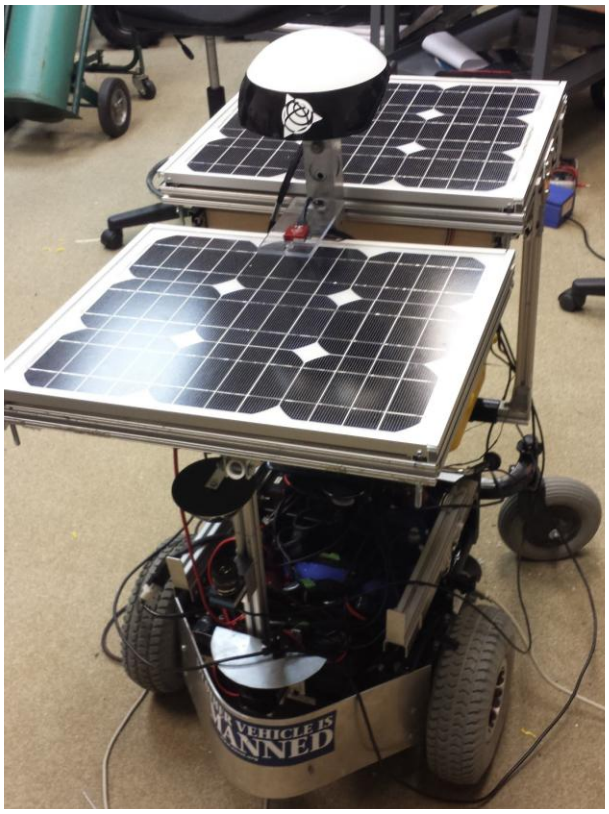
ROBOGOAT 2014



US Naval Academy
Austin B. Taylor, Gavin L. Schelske, and Professor Joel M. Esposito

This document serves to summarize the design and construction of the US Naval Academy's senior design project: The RoboGoat. It will give an overview project as a whole then focus on the biggest changes made this year. The two major foci for this legacy project was the addition and use of an omnidirectional camera system and the continual design of the robot structure. All design aspects are addressed in this report along with figures and illustrations to aid in it' explanation.

INTRODUCTION

The RoboGoat (RG) is a project that has served to educate and develop senior students at the US Naval Academy. It is a legacy project that has been in development since its first debut in 2009. This year we set out to accomplish two goals which would develop our project into a more competitive robot. We desired to develop a camera system that allowed the RoboGoat to have an increased field of view so that it could detect more objects and at farther distances. If the robot can see farther out, it will better be able to react to the environment surrounding it. We also wanted to make significant structural changes to the RoboGoat in order to make the robot easier for the user to control, adjust, and set-up. If we could accomplish these two objectives then it will be easier to successfully navigate the course.

In contrast to the number of additions to the RoboGoat this semester, there are numerous components that will remain the same. With our past years success, we thought it best to keep the things that worked as part of our final system. This included the obstacle avoidance algorithms, LiDAR laser system, lane following code, and a couple of other things. The biggest obstacle is to use these past systems and integrate them with our newly developed systems. In particular it was interesting combining the existing obstacle avoidance and lane following algorithms with our completely new vision system.

DESIGN PROCESS

The vehicle is built explicitly to participate in IGVC, therefore the design process is guided by constraints and functionalities required by the rules, in addition we attempt to optimize some additional attributes deemed beneficial based on our past experience with the competition.

Constraint	**Specification**
Size	3 ft < Length < 7 Ft; 2 ft < Width < 4 ft; Height < 6 ft
Payload	Capable of carrying 20 lb block, 18” X 8 “ X 8 “
Safety	Mechanical e-stop, wireless e-stop, safety light, max speed < 10 mph (see rules)
Budget	< $3000 (Internal FY14 Budget)

Rank	Attribute	Justification
1	Reliable	A run is terminated at the first error (e.g. hit obstacle). During a typical 10 minute run, with sensors updating at 10 Hz there are 6000 frames of sensor data to analyze. A single bad frame can cause a collision.
2	Maneuverable	A zero turn radius reduces the need for sophisticated planning algorithms.
3	Easy to Debug	The project is too large in scope for a single student team to complete from scratch. Next year's team must be able to understand and trouble shoot the design. In addition much of the trouble shooting must be done in the field.
4	Endurance	Longer endurance allows the vehicle to participate in all competition runs, and allows for more testing.
5	Small	The course is comparatively less cluttered / gaps are more open for vehicle with minimum length/width. There is no advantage to having a large vehicle. Transporting a large vehicle to competition requires disassemble or a special motor vehicle.
6	Fast	It must be possible to complete the course in 10 minutes. In event of a distance tie, time is used to score participants. This is unlikely.

Function	Sub-Functions
Basic Mobility	• Average > 1 mph on grassy terrain • Traverse inclines up to 15 mph
Follow Lanes or Flags	• Detect objects • Drive accordingly
Avoid Obstacles	• Detect objects • Drive accordingly
Navigate to Way Points	• Enter way points • Determine current position • Drive on heading
Mobility	• 1 mph < Speed < 10 mph • Inclines < 15 degrees • Grassy terrain • Minimal turn radius
Manual Control	• Ability to take control remotely • Maintain sensor feed • Easy to drive

System Selection

Characteristic	Option 1	Option 2	Option 3	Option 4
Vision Hardware	Parabolic Mirror with Camera	Conical Mirror with Camera	Spherical Mirror with Camera	-
Vision thresholds	Fixed	Global Adaptive	Locally Adaptive	-
Obstacle Avoidance	Ultra-Sonic Sensors	25 Hz Laser Range Finders	10 Hz Laser Range Finder	Camera System
U-turn Correction	Non-existent	GPS Path Tracking	Camera Path Tracking	
User Interface	Purely on Laptop	Android Tablet	Remote Control	Wii Remote
Heading Indication	Magnetic Compass	Gyroscope	IMU	

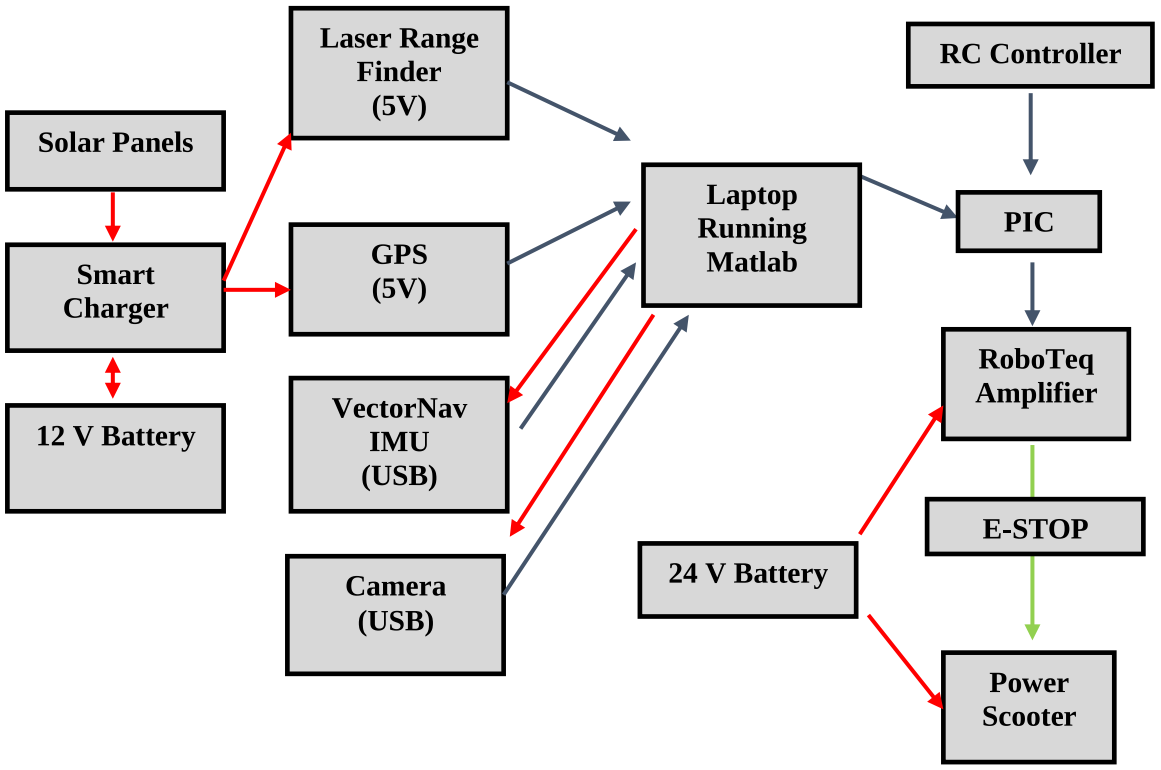
Each of the above selections have independent reasons for being chosen. The first being the choice of which type of mirror we should use for use on in our vision system. Each of the options above have their own pros and cons.

For the catadioptric camera system, we chose to use the parabolic mirror. This would allow a greater field of view along with a higher informational pixel density focused around the vehicle. We decided that with a spherical mirror, we would not get enough pixels covering the immediate area around the robot and thus would have unreliable source from which we can detect close objects and the lanes. The conical mirror allows us to use the center of the camera again vice, it being taken up by a useless image of the robot.

Final Design Overview

The Robo-Goat is designed around four principles:

1. Maximize off the shelf hardware use.
2. All processing done by one laptop running Matlab
3. A system whose state and world view are easily visualized and controlled by the developers, can be debugged more efficiently.
4. Mapping is not necessary to complete the competition tasks and introduces unnecessary complexity.



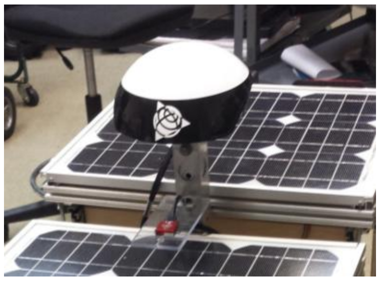
Innovation: We believe these are the most innovative features of the RoboGoat.

- Frame Design: As compared with our previous entries, this version is much closer to ideal.   The robot has a nearly minimal 2' 4" by 3' 2" footprint and maximal height of 4' -- making vision and obstacle avoidance easier.
- Solar Power: it uses solar power to charge a battery while running the onboard electronics, including the laptop, eliminating battery life as one potential limiting factor when testing.
- Robust Vision System: in our experience, this is the most challenging component of the competition.   Our system uses a combination of alternate color spaces, automatic background detection and a new shadow compensation algorithm to adapt to challenges arising from variable lighting conditions.
- User Interfaces: We subscribe to the adage: if you can't see what the system is thinking, you will not be able to debug or improve it.   To that end we have implemented a variety of user interfaces: a robot centered world view, a remote control model that can be switched to on the fly, an integrated power system display and a camera threshold selection tool.
- Maximize Off the Shelf Components: When possible we try to focus on the planning and perception algorithms rather than building custom hardware.

- Runs Entirely in Matlab: We exploit Matlab's extensive library of image processing routines, statistics toolbox, GUI and visualization tools. Programming in Matlab enables rapid prototyping of code, and makes visualization easy. Despite Matlab's reputation for being slow, with proper coding technique and a new laptop we are updating at 10 Hz. We believe we are the only entry running entirely in Matlab. [*]

Mechanical and Mobility Design

Another major change in the structure of the RG is where our netbook is placed. Initially the netbook was placed on top of the circuitry in the plastic bin that held it. This can be seen in the RG 2013 design figure mentioned earlier. This year we decided that we needed to place the netbook in a convenient location for the operator so that he does not need to bend down to access it. We were able to implement a tray just below the rear solar panels of the vehicle. The tray was elevated at about three feet so that the operator did not have to crouch down to access the netbook. The netbook is now separated from the circuitry as well so there is little concern for anything actually shorting now. The tray is also a pull out so that it may be properly stowed when there are no changes being made in the netbook.



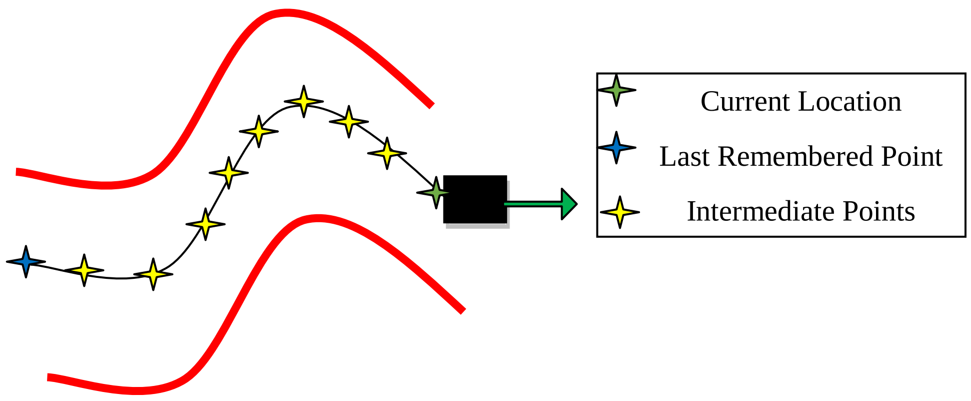
This year we decided to shorten the masthead so that we could be well within the IGVC requirements and make the vehicle more mobile. The rules for the IGVC are 7x4x6 feet. With the old design the masthead was 70 inches compared to our new one which extends upward to a mere 40 inches. This new masthead is only possible because of the new camera system that we have developed for the vehicle.

Software Strategy

U-turn Detection and Correction[†]

In 2012 we ran in to a major problem involving the vehicle making a complete U-turn at the beginning of the course and going in the wrong direction until we had to stop it. Our vehicle had no way to remember where it had been so it never knew it was going the wrong way. In previous years we had experimented with logging all our previous GPS points but we found that this slowed down the computer and used too much memory. This year we added a U-turn detection and correction algorithm using the GPS. The idea behind this is that the vehicle will remember its locations for the last 10-20 seconds therefore creating a tail of GPS points as shown below.

[*] "US Naval Academy - Intelligent Ground Vehicle Competition." 2013. 24 May. 2014 <http://www.igvc.org/design/2013/US%20Naval%20Academy.pdf>
[†] Ibid.



In order to get the tail of GPS points we start out with an empty array of a fixed length and begin logging our GPS points. Each time we get a new point the old points shift right and the new point is placed at the beginning of the array. This allows the vehicle to drag a tail of its old locations behind it without bogging down the memory of the computer. The basic idea behind detecting a U-turn is that if the current location and last remembered GPS point are within the distance of half the lane (we used 2 meters) then the vehicle has made a U-turn. However we found it to be slightly more complicated than this. If the vehicle was standing still or had moved too slow then the algorithm would detect a U-turn because the vehicle had not been able to move outside of the U-turn detection distance. To fix this we found the length of the tail by summing the distance between every point on the tail. This basically finds how far we have moved in the last 10-20 seconds based on the length of the tail. Then we only check if the first and last points are too close together when the tail exceeds a length minimum that is equal to 3 meters or just barely over the U-turn detection distance. If the tail is long enough and the first and last points are within 2 meters then the vehicle will realize it has made a U-turn at some point.

The next step in this algorithm is the correction portion. In order to do this we must know what direction we are heading. Therefore, once a U-turn has been made and realized by the vehicle the code stops and records the current heading from our compass. Then that data is sent to our drive controller and says that the desired heading is equal to the current direction ±180°. Then the vehicle will turn around and once it is on the correct heading will continue the obstacle course. Then the GPS tail array is cleared and begins to fill again to check for a U-turn.

Flag Detection and Path Planning*

The goal of the our line-fitting code was to use the Hough Transform via the HoughLines MATLAB function in order to 'connect the dots' from each flag of a color to the next flag of the same color. HoughLines works by extracting line segments from particular bins based on the Hough Transform. Many parameters were experimentally tweaked to fit lines to our flags which were of reasonable usefullness when applied to driving the IGV autonomously. These parameters included rhores, the resolution in determining the distance of a line, thetares, the resolution for determining the angle of a line, FillGap, the maximum number of pixels allowed to fill a break in

* "US Naval Academy - Intelligent Ground Vehicle Competition." 2013. 24 May. 2014 <http://www.igvc.org/design/2013/US%20Naval%20Academy.pdf>

a line, and MinLength, the minimum number of pixels used to compose a line, among other parameters.

Once lines had been fit to keep the IGV within the flags, we began brainstorming methods to allow our vehicle to safely enter the flag lines which had been drawn. Many ideas were considered but the method pursued was that of essentially drawing a funnel from the flag nearest in distance to our camera. Using a funnel we believed that there was a high chance of successfully guiding our vehicle into the flag channel without running over any flags and without ending up on the wrong side of the channel. The angle of the funnel was computed by first finding the angle at which the flag line had been drawing; named flag_theta. For the red flags, from which we desired to remain on the left side when navigating the flag channel, the angle of our funnel line was computed using :

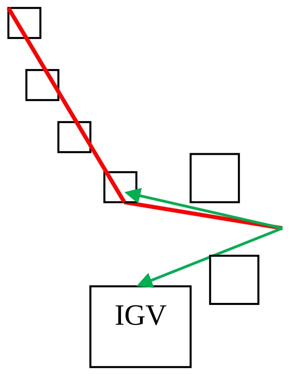
$$funnel\_theta = flag\ theta + .7 * pi$$

Likewise for the blue flags, from which we desired to remain on the right side when naviagting through the channel, the angle of our funnel line was computed using:

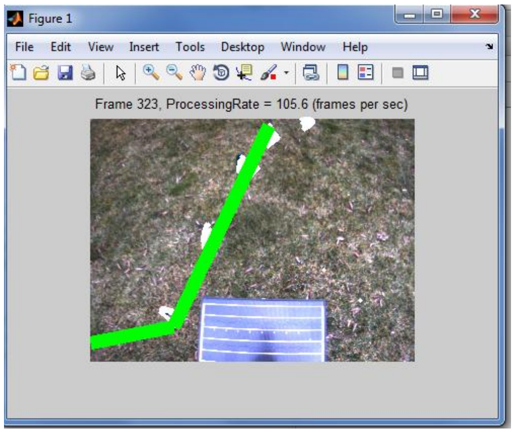
$$funnel\_theta = flag\ theta - .7 * pi$$

The funnel lines were then drawn by taking the cosine and sine of funnel_theta, multiplying this value by a constant to add magnitude (100 was a common value used), and adding the multiplied cosine and sine values to the respective x and y values for the end of the red or blue line nearest to the camera.

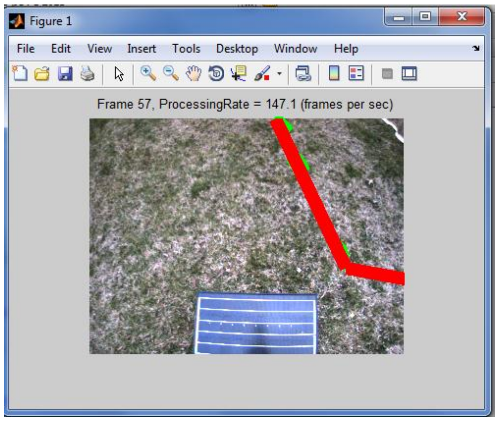
The final goal of our line-fitting code was to implement a script that could detect whether our IGV was on the correct or incorrect side of a certain color flag. Essentially, if we found ourselves on the right side of the red flags, or on the left side of the blue flags, that our vehicle could have some alert that it is driving an incorrect course. Our method for computing this information was to use the cross product between the vector comprised of the points from the end of the funnel line to the front of the IGV and the vector comprised of the points from the end of funnel line to the end of the flag line.



In the picture above the small squares represent red flags, the large square represents the IGV, the red lines represent the flag line and funnel lines drawn, and the green vectors represent the vectors of which the cross product is obtained as shown below:



Frame 323, ProcessingRate = 105.6 (frames per sec)



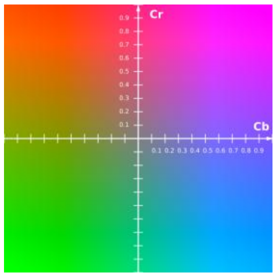
Frame 57, ProcessingRate = 147.1 (frames per sec)

$CrossProductRed = 1x2$

The scripts CrossProductRed and CrossProductBlue have been written to take the Cross product as explained above and display whether or not the IGV is on the correct or incorrect side of the flag channel based on which color flags it is detecting.

Lane Following System[*]

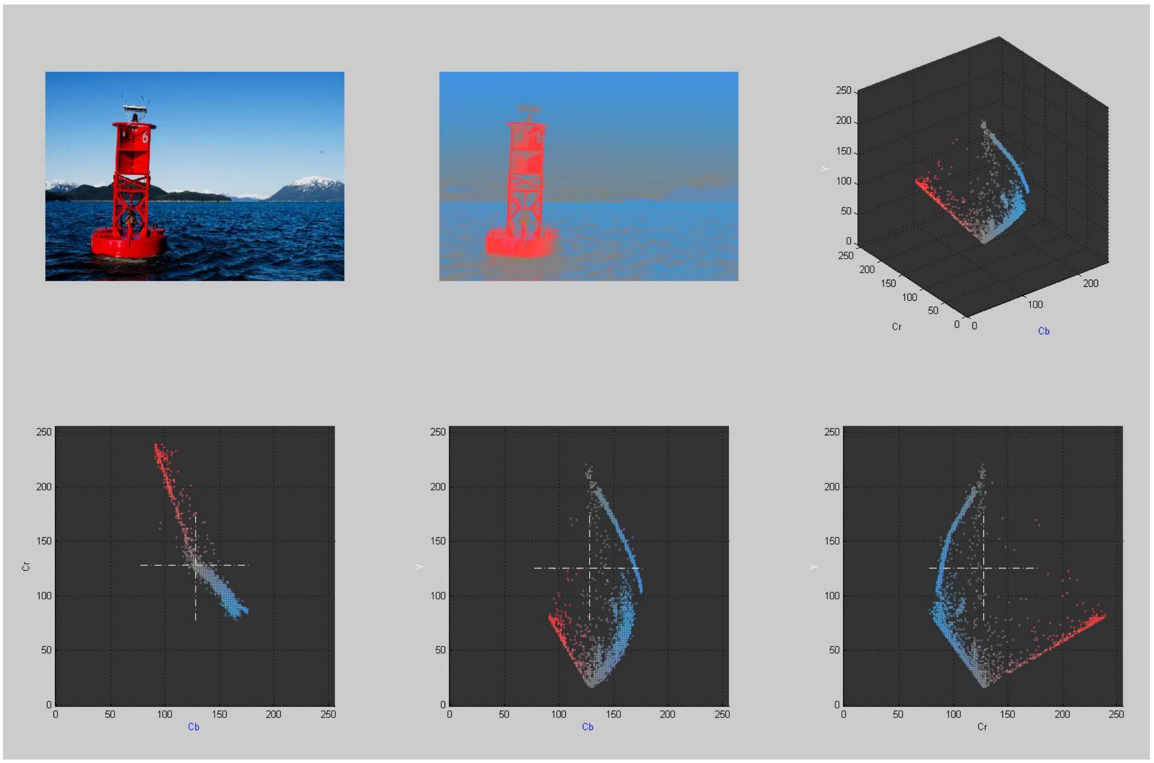
Why YCbCr: The color space that has been chosen for this project is YCbCr. The Y stands for brightness, while Cb and Cr refer to color values (below). If the user only looked at the Y values of an image, they would see a grayscale image with the black areas having a Y-value of 0 and the whitest areas having a Y-value of 1. This color space is handy for the RoboGoat because it distinguishes color from brightness. This means that the thresholds selected with Cb-Cr are more stable in differing lighting conditions, because Cb-Cr values refer to "pure" colors and are not affected by shades or shadows.



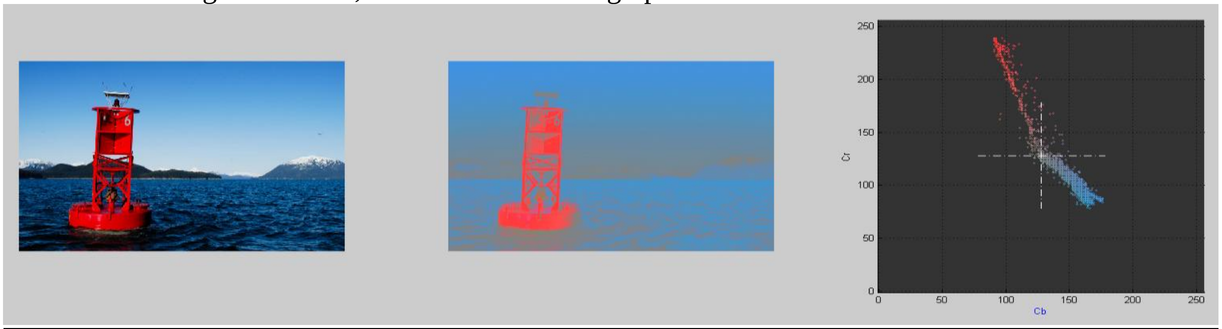
Cb-Cr color plane, Y set to 0.5

The 6-Plot Figure is shown below and is the most powerful source of image information in the program. The top three plots, moving left-to-right, display: the original image (top-left), the image shown in YCbCr with a uniform/normalized brightness level (top-middle), and a rotatable 3D scatter plot of the image's pixels within the YCbCr color space (top-right). The colors of the dots in this 3D scatter plot match the colors in the YCbCr image, so the user can see what part of the image they refer to. The bottom three plots, moving left-to-right, display the three 2D perspectives of the 3D scatter plot: Cb-Cr (bottom-left), Y-Cb (bottom-middle), and Y-Cr (bottom-right).

[*] "US Naval Academy - Intelligent Ground Vehicle Competition." 2013. 24 May. 2014 <http://www.igvc.org/design/2013/US%20Naval%20Academy.pdf>

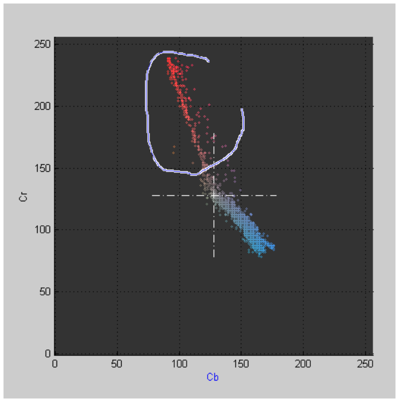


The three 2D graphs are very handy. In the example of using the picture of the red buoy, all three 2D plots show the dots for the red buoy from various angles of the 3D graph. The dots in first 2D graph (Cb-Cr), however, include <u>every value of Y (brightness)</u>. This means that if the red dots are selected from the Cb-Cr graph, every shade of the red buoy will be included. The other 2D graphs have Y in the vertical axis, so the red shade-ranges are spread vertically up and down the graph. But if the user wanted to <u>specifically</u> choose the lightest or darkest red pixels, they could choose the Y-Cb or Y-Cr graphs. Once the user chooses the 2D graph that they would like to use for threshold selection, the program will display a new window. This window displays three items: (left-to-right) the original image, the image shown in YCbCr with uniform/normalized brightness level, and the selected 2D graph.



Original Image *YCbCr view, uniform brightness* *Distribution of pixels in color space*

In the 2D graph, the user will freehand-select (draw) around the pixels they want to threshold. This is done by left-clicking in the graph, holding down the mouse button, and dragging the cursor around the desired dots. Figure 3 shows what the freehand selection looks like, in-progress:

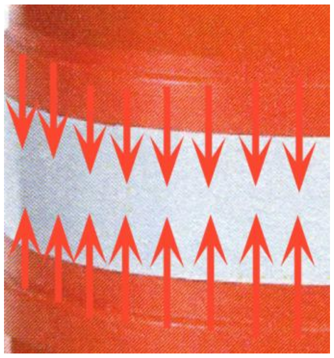


The program reads the coordinate values of every selected pixel. In this example, the coordinates are Cb-Cr values (just like the x-axis and y-axis). The program then takes the maximum and minimum values from both axes - these four values become the thresholds.

Solving the "Barrel Problem": [*]

Certain objects on the IGVC course pose a problem for the robot's vision system. The robot is designed to use its camera to look at chalked/painted white lane lines on the ground, using them as a visual reference to stay within the lane. There are many obstacles to avoid along the route, the most common of which is an orange construction barrel. The figure below shows a picture of several such barrels on the IGVC course - the lane line is visible in this figure too.

When pixels in each frame of the streaming camera fit the thresholded criteria of a "lane line" - when our robot sees what it believes to be a white lane line - it assigns a trend line to those pixels. This is a simple concept, the same process occurs in the brain when one sees a less-than-perfectly painted line on a field. However, the robot's line-fitting ability cannot distinguish the difference between white paint in the grass and the white stripes on the barrels. If they both fit the color-criteria for a lane line (white), then the program marks those pixels as "lane line" pixels. Then, when it calculates the trend line for the "lane line", the result is completely incorrect. The robot thinks that the lane line is pointed/angled in a certain direction, when any person looking at the field knows otherwise. Fortunately, there is a way around this problem through the use of structuring elements. A structuring element is a shape created around a pixel. For this particular task, we use a rectangular-shaped structuring element.

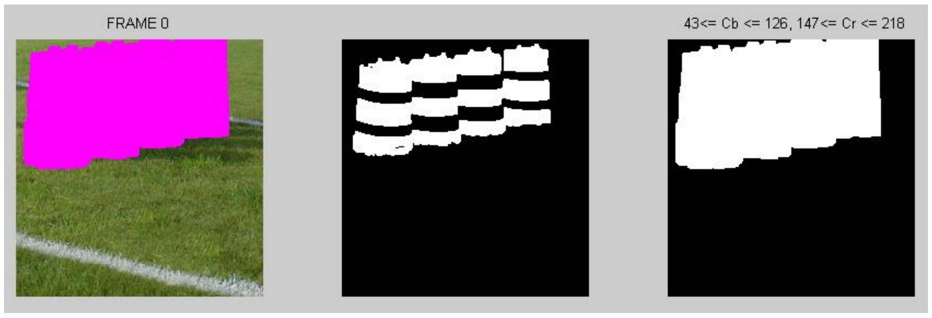


Orange pixels are dilated to overlap and conceal the white stripes

The result of this method is that the program is free to threshold and track the "actual" lane lines [in the grass], using the trend line calculations, since it is no longer confused by the white regions on the barrels. The following figure shows this dilation executed on the image of the course:

[*] "US Naval Academy - Intelligent Ground Vehicle Competition." 2013. 24 May. 2014 <http://www.igvc.org/design/2013/US%20Naval%20Academy.pdf>

Figure 6



Electrical System

We replaced the old GPS system, which was a two piece component, with a newer GPS (Ag 162) from the same Trimble Company. The new GPS only had one component which helped in simplifying the RG. The second component to the last 0047PS iteration (the Ag 132) simply displayed the GPS information to the operator. We did not need the display box associated with the new GPS since the data only needed to go to the netbook where both the user and the code could see and utilize the data. The Ag 162 simply takes data from the satellites and sends them directly to the netbook.

This year we also replaced the Honeywell compass with an Inertial Measurement Unit (IMU). The IMU (VN-100) has a compass built into it so it has all the same features of the compass but with added components like the gyroscope and the accelerometer. We can now know things like where gravity is pointing with respect to the vehicle as well as how fast it is accelerating or at what speed it is turning at in any direction. This will be incredibly useful when developing code that stops the vehicle based on sensor inputs.

The first mechanical subsystem to make a note of is the new light that we used which is a 12 volt strobe that has an electrical connection for flashing mode and one for solid mode. The new light is much brighter than the one used last year even though it only utilizes half of the voltage. The light was attached to the far back of the RG to allow the operator to easily see which mode the vehicle is in. The light wiring is temporarily connected to our switchboard; however, it will eventually be attached to some relays coming off of the Arduino board so that it can switch upon changing between autonomous and manual mode.

Autonomy Philosophy

The RoboGoat is able to switch into an autonomous mode when the user activates a switch on the remote control. Once the RG (RoboGoat) is in autonomous mode, it will turn on its flashing red LED light to make observers aware that it is indeed in the correctly designated mode. The vehicle should function well on its own without the user having to intervene. Ultimately, safety is the crucial point of autonomous mode. It should be the first aspect to be catered to and developed. After safety comes the rest of the RG. We want the RG to be able to run the course on its own with ease. It should avoid any traffic lane violations swiftly and easily. This includes hitting other vehicles, running outside of the white lines that mark the boundaries of the course, and avoiding obstacles all while maintaining a speed below the 10 miles per hour designated in the rules.

User Interface[*]

We subscribe to the adage: if you can't see what the system is thinking, you will not be able to debug or improve it. To that end we have implemented a variety of user interfaces: a robot centered world view, a remote control model that can be switched to on the fly, an integrated power system display and a camera threshold selection tool. The user should be very comfortable with handling and operating the RG. This year we have made the RG much easier to assemble and disassemble for the user to make adjustments and fix bugs in the system as necessary. The circuit board is detachable so the operator may inspect individual circuit components, as well as replace or add new ones as necessary. The netbook is also placed in a tray which allows the user quick and comfortable access to the code running the RG. To sum everything up, we made the RG easier for a user to understand, operate, and adjust. RG is a legacy project, so changes will be made often throughout the years, thus it is vital to the success of RG for the system to be very flexible and easily alterable.

Systems Integration to Achieve [†]

When considering our major systems, lane following, GPS, obstacle avoidance, our system is comparable to the 2013 design. We believe our new additions are fully integrated with the old system and we can expect similar results. Specific testing points:

1. GPS Navigation: Our GPS did not change from the 2012 competition where we were able to hit all GPS points in the navigation course. We conclude that our GPS will have the same success this year.

2. Speed: The vehicle's max speed is 5 mph. However, at this time the autonomous navigation has only been tested at speeds up to 2 mph. Initial experiments with our laptop suggest an update rate of 10 Hz. We believe this will permit running the course at max speed.

3. Battery Life: At a recent test evolution the power systems were initially fully charged using AC power. Over the next 48 hours, we ran for about 4 hours using only the preexisting charge, and the energy contributed by the solar cells.

4. Obstacle Avoidance: Our obstacle avoidance algorithm is nearly flawless. It is perhaps the strongest feature of RoboGoat. At this time we intentionally use a detection distance of 1.5 meters, even though our hardware is capable of up to 4 meters.

5. Lane following: Over the course of 48 hours, we observed the vision system worked nearly 100% of the time. More importantly we did not adjust the color thresholds despite the fact that the

[*] "US Naval Academy - Intelligent Ground Vehicle Competition." 2013. 24 May. 2014 <http://www.igvc.org/design/2013/US%20Naval%20Academy.pdf>
[†] Ibid.

light conditions changed. This is extremely promising. Note it is very difficult for us to replicate the lane markings on campus because we cannot paint the grass

Testing and Performance

We will test our vehicle outside on a local soccer field. We will bring the vehicle out there then set-up our netbook and use our final drive code. We will set up certain segments of the IGVC course and run the robot at individual segments to test how it can handle the individual portions of the competition. We will first make a course that is approximately 80- 100 feet with white painted/roped lines on either side to act as the borders of the course. We will then set-up orange cones and orange barrel barriers inside of the miniature course. The last thing we will incorporate is large box of some sort inside of the competition track to simulate a stopped vehicle.

The other portion of the test will test the flag algorithm that we develop. We will set-up a small 50-60 foot course that consists only of the flag portion of the IGVC competition. The flags will be set-up as closely as possible to the configuration shown in the competition guideline.

Another navigation portion of our test will be to map out waypoints across the field and use the same orange cones and barrels and obstructions. We will only use the GPS of the robot to have it move to those waypoints while avoiding the obstacles we set in between itself and the destination.

Of course we will have already tested the RC controls as well as the electronic E-stop. We will switch the robot to RC mode and drive the robot the maximum distance possible to see how far it can go until the controller no longer communicates with the robot so that we have an idea of the maximum range upon which we can safely electronically E-stop the robot.

CONCLUSION

Our robot this semester will be in top fighting shape. With a newly designed camera system, completely new dimensions, and a more ergonomic interface, we expect to steal the show and earn first place. With last year's robot doing so well, we can only improve. The robust and tolerant vision algorithms as well as the increased field of view will serve well enough to defeat the flag obstacles which were our biggest problem last semester. While we don't have the prettiest robot, we have high expectations on its performance.

ACKNOWLEDGMENTS

We give special thanks to all of the faculty and staff that have helped us to implement the additional components to our project. They were a great help and this project would not be possible without their help:

Our Capstone Advisor: **Professor Joel Esposito**
Our School's Technical Support Department staff to include:

- **Mr. Tyson**
- **Mr. Wick**
- **Mr. Bradshaw**
- **Mr. Keith**