

Robo Goat 2011



United States Naval Academy, Systems Engineering Department

By Midshipman:

- Adam Albrecht (Jr)
- Caleb Burrow (Sr)
- Amy Lowe (Sr)
- Nicholas Mehalic (Sr)
- Dylan Parrott (Sr)

Adviser Statement: I certify that the design and engineering of the vehicle by the current student team has been significant and was awarded credit for a capstone senior design course.

Adviser: Assoc. Prof. Joel M. Esposito _____

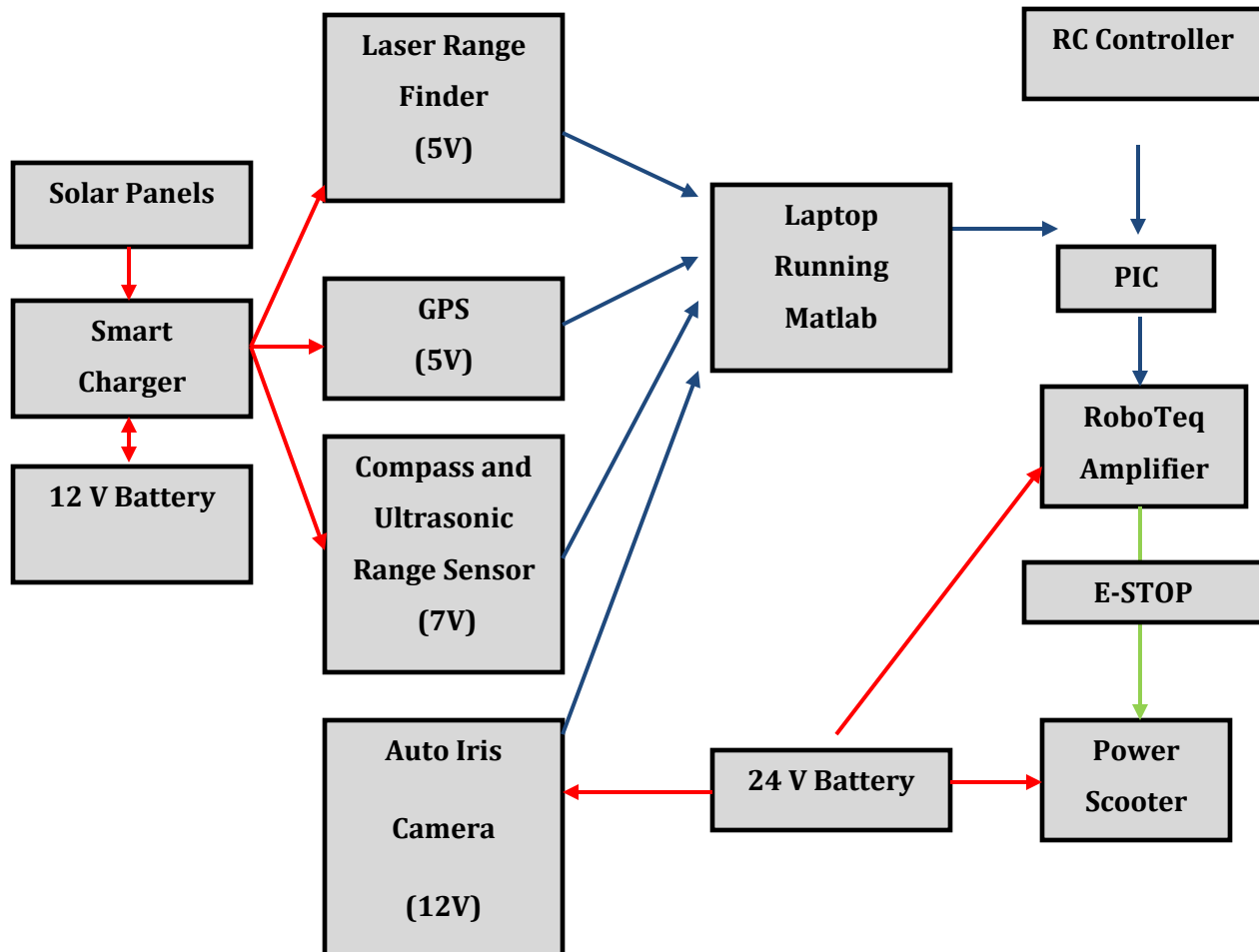
New For 2011 Entry:

- | | | |
|-----------------------------|---------------------------------|---------------------|
| GPS software | Solar panel and charging system | Laptop |
| Auto -Iris Camera | Ultrasonic range finders | Visualization Tools |
| Camera hood/mount | Camera filter | Safety light |
| Integrated RC Control | E-stop | |
| Line thresholding algorithm | Autonomous Navigation Algorithm | |

Overview

The Robot-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 Robo-Goat.

- Solar Power: it uses solar power to run the onboard electronics (with the exception of the laptop).
- Robust Vision System: in our experience, this is the most challenging component of the competition. Our robot uses a combination of alternate color spaces, auto-iris hardware, and robust statistics to combat the 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, 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.

New Design Decision Process

Background: The Robo-Goat capstone project is a legacy project that began in 2009. Every year the vehicle has competed in the IGVC (Intelligent Ground Vehicle Competition). In 2009 the Goat placed 20th, and in 2010 it placed 10th. This year, we hope to be within the top five. Last year, the majority of issues with the Goat came from the path detection capabilities of the camera. Also, the onboard laptop was outdated, the object detection system needed improvement, and the Goat suffered from poor battery life.

After identifying these weaknesses, our team developed the following morphological chart to help identify the solutions.

Characteristic	Option 1	Option 2	Option 3	Option 4
Vision Hardware	Auto-Iris Camera System	USB Camera	Stereo Camera System	-
Vision Software	Colorspace	Contextual Cues	Thresholds	-
Obstacle Avoidance	Ultra-Sonic Sensors	Laser Range Finders	Camera System	-
Power	Battery	Solar	Wind	Bio-Fuel
Reach Waypoints	GPS	Differential GPS	2-Axis Magnetometer	3-Axis Magnetometer

The hi-lighted sections of this chart represent the major additions made to our vehicle this year:

- Siemens Solar Industries SM20 Panel (2)
- Auto-Iris Camera
- SRF08 Ultra Sonic Sensors (4)
- HP Tablet Laptop

Power System

Solar Power System: The battery (Fig 1) that runs the electronics on the Robo-Goat is a Power Sonic 12V 7.0Ah sealed lead acid battery (model number PS-1270). The electronics run by the 12V battery are: GPS, Laser, Nav-Board, Ultrasonic range sensors, and RC boards. The total draw on the battery when idling is .64A. There are two Siemens Solar Industries SM20 solar panels (Fig 4). They are individually rated at 20W and designed to support a load of up to 1.38A at 14.5V. These panels were wired in parallel to double the charging current supplied to the 12V battery.

Figure 1



Figure 2



The solar charge controller (Fig 2) is a Morningstar Corp. SunSaver-10. The important feature of the controller is the circuitry that keeps the panels from draining the battery in poor lighting conditions. The controller is rated to handle 10A of input from the solar panels and a 10A load. The controller is designed to run a 12V battery and load and a comparable voltage solar panel. We chose a controller without a LVD (Low Voltage Disconnect) feature. Since all the electronics attached to the battery require between 5V and 6V, there was no danger of equipment damage if the battery voltage dropped below the 11.5V factory set disconnect voltage. In fact, we hypothesized that this feature would prematurely force us to replace the 12V battery, thereby defeating the purpose of the solar recharging system. The capabilities of this controller significantly exceed the requirements of our current system.

Mount: The required functions for our mount were for it to have an adjustable angle, and that it would be mounted above the laptop to help reduce glare on the screen. Several different designs were explored, but we settled on the pin/disc design shown below (Fig 3,4) :

Figure 4

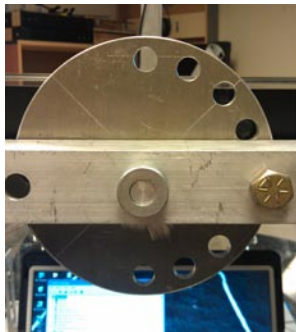
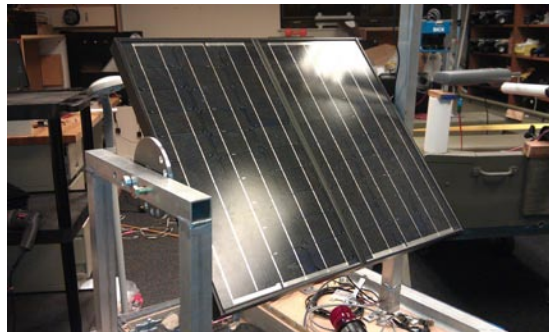


Figure 3



Testing Solar System: Using a 12V motor that draws .4A we tested the solar system to see how well it worked. A baseline test with the motor attached to a fully charged battery caused the battery voltage to drop below 11V in approximately 45 minutes. Note

that this would imply that the 12V battery has a rating of .3Ah, when it is rated for 7Ah. The batteries are 5 or 6 years old and have been used year after year in capstone projects. When the solar panels were attached and tested on a partly cloudy day, the battery voltage had yet to drop below 11V after 2 hrs. Our true test occurred during the DC demonstration of our project in April. On both days we started with a fully charged battery and did not need to change batteries at any time during the day. In fact, the second day was sunny and the 12V battery was still fully charged at the end of the day.

Lane Following System

Auto Iris: We've chosen to use an auto-iris camera, purchased from The Imaging Source. It uses a motorized iris to control the amount of light entering the camera. This ability should not be confused with changing the exposure, which alters the frame rate. The reason that we have chosen an auto-iris camera is to compensate for dynamic changes in outdoor lighting. In previous years, when an auto-iris camera was not used, small changes in daylight would negatively affect the program's ability to track selected colors.

Hood, mount and filter, field of view: The camera is mounted underneath a hood to reduce glare from the sun overhead and uses a circular polarizing filter. It is mounted at about 70 inches above the ground, so that it has the largest view of the ground as possible. It is angled downwards, such that the bottom of the viewing window sees 6-inches from the front of the robot. This to reduce the size of a blind spot directly in front of the robot. At this angle, the camera sees 12-inches wider than the robot on each side.

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 (Fig 5). 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. Look back at the thresholded red buoy (Fig 1), notice how all of the red paint has been selected despite shadowing.

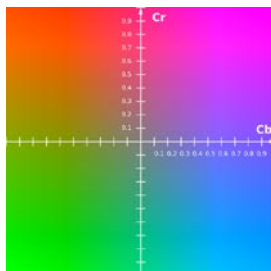


Figure 5

Cb-Cr colorplane, Y set to 0.5

Why don't we use RGB (Red/Green/Blue) for thresholding? RGB thresholds are extremely sensitive to lighting changes. The results are inconsistent, and RGB thresholds that are captured while looking into the glare of the sun tend to be quite different than the thresholds captured while looking away from the sun.

Threshold section: The vision system for the robot is centered on thresholds. This means that the user decides which colors the robot should be interested in - how the robot *uses* those colors is another topic. Every pixel in every frame of a video feed has values (brightness, color, etc) that the program can detect, and the robot checks each pixel in real-time to determine if it fits the threshold criteria. For example, in a picture of a red buoy (Fig 6), the user might threshold the buoy by targeting its red colors. After that threshold is saved by the program, the program will "recognize" any object that matches the colors of that red buoy. In our case, we threshold for white lines on a field.

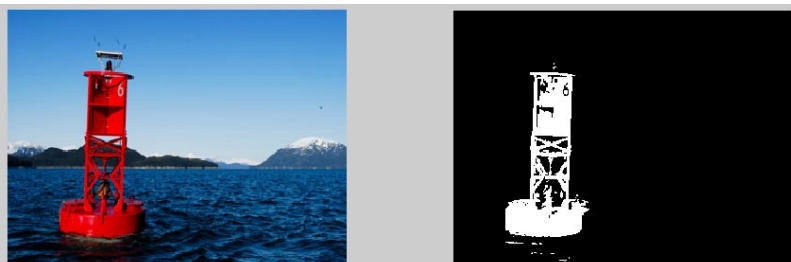


Figure 6

Once thresholded, the robot should recognize those lines every time they appear in the view of the camera.

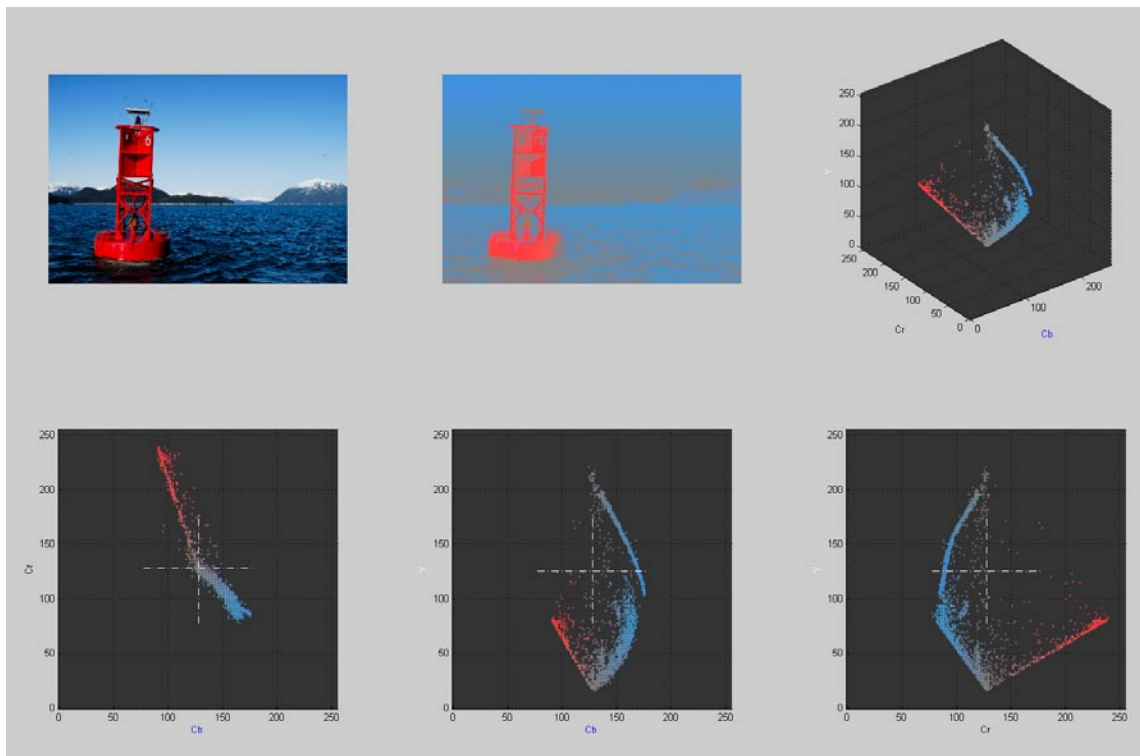
Original Image (left);

Binary image of threshold

(right)

The 6-Plot Figure is shown below (Fig 7), 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).

Figure 7

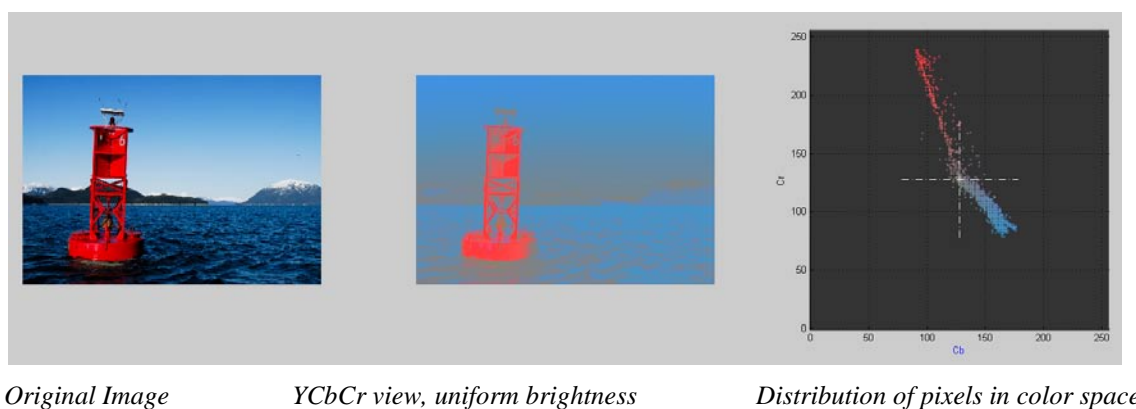


The three 2D graphs are very handy. In the example of using the picture of the red buoy, all three 2D plots show the dots for the red buoy from various angles of the 3D graph. The dots in first 2D graph (Cb-Cr), however, include every value of Y (brightness). This means that if the red dots are selected 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 specifically 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. See Figure 8 (below):

Figure 8



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 9 shows what the freehand selection looks like, in-progress:

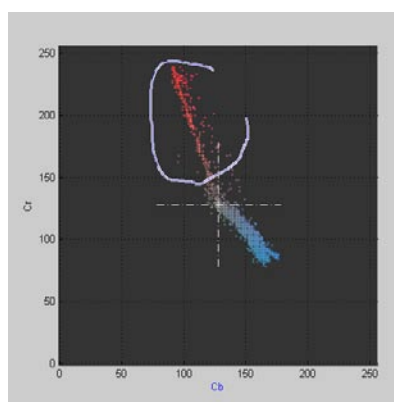


Figure 9

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. Figure 10 shows a picture of several such barrels on the IGVC course - the lane line is visible in this figure too

Figure 10



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.

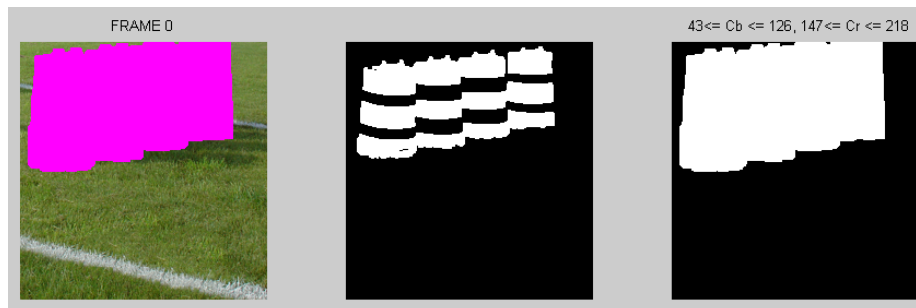
Figure 11



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. Figure 12 shows this dilation executed on the image of the course (Fig 10):

Figure 12



Calibration to ground plan with CalTech

Toolbox: Using the CalTech Toolbox camera calibration software, we are able to determine the intrinsic and extrinsic

parameters of the camera. This software allows the robot to know where the camera is, in reference to the ground. Since the camera is not stereoscopic, and without true depth perception, it calculates "depth" by assuming that all objects are flat on the ground.

Gains on distance for steering: Center islands are navigated using a balancing of the repulsion of obstacles and lanes. The Robo-Goat wants to maintain a minimum set distance from each according to different gains. The avoidance distances for obstacles are set such that the Robo-Goat will only turn enough to barely avoid the obstacle. Lanes, however, have a higher gain and avoidance distance and the Robo-Goat will try to maintain that distance as

well. Then these two inputs are detected they force the Robo-Goat to take a path that balances the repulsive forces of the obstacle and lane. This balanced path typically exists much closer to the obstacle.

Obstacle Avoidance System

Laser: The primary obstacle avoidance system is a laser scanner -- the Hokuyo URG-04LX. The laser is configured to scan every 2 degrees in a 240 degree arc. The laser measures phase difference and time of flight to determine the range to an object.

Our algorithm is patterned off of the Dynamic Window Obstacle Avoidance Method. After accounting for the robot's size and speed limitations, it finds the best "gap" or heading direction. "Best" is defined along the following criteria (1) closest to desired heading (specified by camera or GPS); (2) closest to current heading; and (3) maximum clearance.

The laser is capable of seeing up to 4 meters out; however, we set the maximum range that our navigation algorithm will recognize lower so we can navigate switchbacks. A switchback can be seen as three layers of obstacles. When the distance threshold of the laser is set properly, only one layer of obstacles can be seen at a time. This allows the Robo-Goat to enter the first gap before trying to avoid the second layer of obstacles. If the distance threshold is set too high, the Robo-Goat will see both the first and second layer at time, which will look like a solid wall, making the Robo-Goat perform a u-turn.

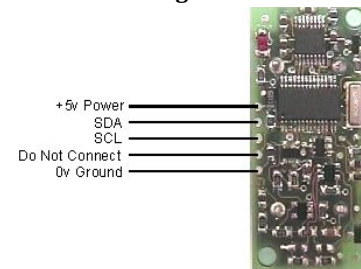
The layering of the switchback obstacles is important because our algorithm that analyses the laser, camera, and ultrasonic data finds gaps in the obstacles. Lane data from the camera and returns from the ultrasonic sensors are combined with the laser data to form a complete picture of the obstacles around the Robo-Goat. After finding the gaps, it compares the width of the gaps to the parameter that defines the width of the Robo-Goat and determines if a gap is large enough. The code then finds the gap that is closest to the desired heading, which was defined using either GPS waypoints or lane information from the camera.

Enhanced Object Avoidance: For object avoidance last year, a laser range finder mounted at the base of the goat was used. However, a single laser range finder makes it difficult to detect objects such as sawhorses. The laser would only strike the upright supports and believe it could navigate between them. This year, we are using four ultra-sonic sensors in addition to the laser range finder.

Ultra Sonic Sensors: There are four SRF08 ultra sonic sensors (Fig 13) newly mounted to the Robo-Goat. Two face forward while two face outward. The sensors use the I2C bus to communicate with the NavBoard, which was already mounted to the Robo-Goat. The sensors beam is very similar to that of a cone, giving the sensors a wide viewing area.

Each sensor is shipped with a default address, 0xE0. Using a simple code, it is possible to readdress the sensors. This will allow multiple sensors to use the

Figure 13



same I2C bus but fire at different times. By placing only the sensor that needs a name change on the bus and using the simple code, , the address can be changed to one of sixteen different addresses. Once the address is changed, every time the sensor is powered up, the LED will flash out the address. The sensors will also maintain any new address. It is important to ensure all sensors used have different addresses. Failure to do so may result in a bus collision and very unpredictable results.

The SRF08 sensors only use a 5V power source, making it easy to use the same power that is used to power the NavBoard. The SDA and SCL are the data and clock lines respectively. The resistors used for the system are 1K Ω . The beam of the SRF08 ultra sonic sensor is a cone like shape. The sensors are sensitive up to a range of about 80 inches. From the sensor, the beam spreads 45° from center for about two feet. Past that two feet mark, the cone stops spreading.

The range for the SRF08 sensors is determined by the time between firing the ping and receiving it. As the time increases, so does the max distance. The current setting has the sensors pausing for 65 ms before reading the return. This allows the sensors to reach their top distance. The range for the sensors is based on the range of the laser. The computer will take the lesser of the two ranges, between the laser and the ultra sonic sensors, before considering a turn.

The original code had all the sensors fire in sequence before reading any returns. This would work for three or less sensors, but errors occurred when using all four sensors. The maximum distance dramatically decreased to roughly 15 inches. This error was fixed by firing and reading one sensor at a time and can be found in the code. This would eliminate the error of having one sensor read pings from the other sensors.

Mounts: The side mounts for the ultra sonic sensors are simple L brackets and are recessed into the body of the Robo-Goat. The brackets are tall and have an opening that runs the length of the bracket. This allows future teams to adjust the height of the sensors. Currently the sensors are mounted in such a way that they can detect the flags used for lane markings. The mounts are recessed to save the sensors from hitting obstacles while turning. The front two sensor mounts utilize the camera mast. The two sensors are not as adjustable but there is room left to move them.

GPS Navigation System

GPS: The GPS receiver we use is a Garmin 18x-5Hz. The receiver sensitivity is -185dBW and is accurate to within 3m using WAAS differential GPS. The receiver is attached to our laptop via a serial to USB adapter. The GPS receiver is mounted atop our mast to have a clear view of the satellites and is powered by the 12V battery through a

5V regulator. The receiver sends a NMEA (National Marine Electronics Association) sentence every .2 seconds to the serial object in MATLAB.

This year we started with new code to read the GPS receiver. We discovered through experimentation that when the serial object buffer in MATLAB overflowed the new data was thrown away, not the old data. To deal with this, our code reads through the entire buffer until it has read every complete NMEA sentence. Using headers, delimiters, and a checksum at the end of the sentence, the code verifies that a full and accurate sentence has been received before returning values from the sentence to the main code. To reduce the time it takes to read through the buffer, we initialized the buffer to hold 1 second of measurements. This corresponds to 5 sentences. Since our navigation loop runs at approximately 3Hz, we will always call our function to read the NMEA sentences before the buffer overflows, thereby always receiving up to date data. This method, however, requires a full run through the navigation loop in the beginning to remove all the old data in the buffer from the initialization of the GPS serial connection in MATLAB.

The Robo-Goat uses GPS information to navigate by comparing GPS waypoints. The MATLAB function that does this converts the GPS coordinates to radians and then uses two off the shelf equations to determine the distance between the points and the direction from the first point to the second. This gives the Robo-Goat the desired heading and the distance to the waypoint so we can set a threshold that determines when we arrive at the waypoint. As soon as we get to a waypoint the Robo-Goat accesses the next entry in the matrix defining our waypoints and repeats the process.

The Robo-Goat does not use any form of mapping. In fact, the Robo-Goat has no memory. In an attempt to achieve a 5Hz speed on our navigation loop and the reaction benefits that that brings, we decided to not map our environment. This, however, is making it difficult to keep ourselves from performing a u-turn when we encounter an obstacle or from crossing a line that has disappeared beyond our field of view. Our planned compromise is to implement a Kalman filter that will, to an extent, remember where the lines are that leave the field of view of the camera.

Nav-Board: The Nav-Board used on the Robo-Goat (Fig 14) is designed by the technical staff in the department for use on numerous different projects. The board is based on a Rabbit 3000 microprocessor and hosts a XBee module, 3-axis accelerometer, 3-axis magnetometer, a GPS receiver, and numerous I/O ports (including analog, serial, and interrupts). We use the Nav-Board's magnetometers as a compass and the analog I/O to read data from our ultrasonic sensors. Data is collected by MATLAB from the Nav-Board via serial port.

The compass on the Nav-Board is calibrated by recording the raw numbers from the magnetometers while spinning the Robo-Goat in a circle. The result when the x and y axes are graphed simultaneously is a circle. The values needed to zero the compass are

Figure 14



the x and y coordinates of the center of the circle, taken by averaging all the x and y values separately. These numbers are then written into the MATLAB code that reads the data from the serial object through which the laptop and the Nav-Board communicate. These numbers are added to the raw data so that heading can be accurately determined. If improperly calibrated, typical errors include all headings being in the same quadrant or headings appearing to skip a quadrant when the Robo-Goat is rotated.

Drive System

Speed/Performance: The Robo-Goat is hardware limited by its capabilities to 4.77 MPH. We also discovered that in order to maintain our minimum speed we have to, on average, send at least 6V to our motors.

Voltage (V)	RPM	MPH (10" wheel)
5.93	37	1.1
23.2	160.3	4.77

Roboteq: The Robo-Goat is propelled by two 24V DC motors rated at 4.5A each. We control the motors with the Roboteq motor control board via commands from the laptop through a serial connection. We used the supplied RoboRun utility to measure the limits of our system by using a tachometer to measure our rotational velocity at varying voltage outputs. Since the motors are powered using two 12V lead acid batteries in series, we have an available voltage range of 24V. The RoboRun utility has a power setting that ranges from 0 to 127 and is proportional to the voltage range. This was the range used to test the speeds of the motors. The MATLAB function that sends commands to the Robo-Goat, however, has translational and rotational inputs that range from 0 to 1. These are combined to form individual commands for each motor that correspond to the desired voltages. Forward and reverse changes are handled by changing which pin the Roboteq board uses to output a positive voltage to the motor with.

Wheel Chair: The chassis for the Robo-Goat is a Sunfire Plus SP3C Power Mobility Chair. It weighs 82 lbs. The chair has a range of up to 20 miles, making it well suited to the competition. The chair is powered by two 12V U1 34 Ah batteries in series. The ground clearance on the chair is only 2.5 inches, meaning we can navigate small pot holes, but not large ones.

Safety Systems

Safety Light: A new requirement for the IGVC this year is a safety light. The light must be solid when the vehicle is in ROV/manual mode and be flashing when it is in autonomous mode. The safety light we chose to use is a Federal Signal LP3TL-024R. We chose it for two reasons. First, it has an LED bulb that has a low current draw, .08A. The second reason we chose the light was its 4X weather rating, which means it is able

Figure 15



to withstand dust, hose directed water, and corrosion. With a 100,000 hour life on the LED bulb, this light should remain useful to the project for many years.

Emergency Stop: The Robo-Goat is paired with a 6CH Futaba transmitter and receiver. The Futaba transmitter manually controls the Robo-Goat, has an Emergency Stop, and has the ability to switch the input of the Roboteq motor control board. A PIC12f675 is used to monitor CH5 which is the Emergency Stop channel. If tripped, the PIC disengages a relay that cuts power to a solenoid, which in turn cuts power to the motors running the Robo-Goat. Another PIC12f675 monitors the other channels to manually control the Robo-Goat and to change the input to the Roboteq board. The input for the Roboteq board can be set to the RS-232 input from the laptop running MATLAB or it can be set to receive input from the second PIC through a MAX 323 which converts the signal from the transmitter to RS-232 levels.¹

Integration Testing

We participated in the AUVSI demonstration on DC's National Mall in April. There the system performed well, traveling over 500ft autonomously in the two days we were there. Specific testing points:

1. GPS Navigation: Based on 10 way points tested, the goat got within 2 meters of 9 of the 10. The final way point was declared "reached" once the goat was within 3 meters of the actual point. This can be difficult to test without surveying equipment.
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 new laptop suggest an update rate of 10 hz (vs the current rate of 5 hz). We believe this will permit running the course at max speed.
3. Battery Life: At the DC Demo neither the 12 or 14 V power systems were recharged using AC power. Over 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 robo goat. 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 work nearly 100% of the time. More importantly we did not adjust the color thresholds despite the fact that the light conditions changed. This is extremely promising. Note however that the lines at the DC demo were made from lime chalk rather than paint, making them easier to see.

¹ Information taken from Adam Albrecht's report "Robo-Goat: Integrated Radio Control System" written 09DEC10.

7. **Figure 17**

6. **Ramp climbing:** The IGVC states “...expect natural or artificial inclines with gradients not to exceed 15% and randomly placed obstacles along the course.” With this year’s addition of the solar panel, our team was concerned that the vehicle would become too heavy, and unstable on inclined surfaces. We tested the Goat by placing it in its most precarious orientation atop a 15 degree inclined ramp. (Fig 16-17) The vehicle remained sufficiently stable despite bumps and jostling.

**Figure 16**

Cost Estimate

Cost:

	Number	hr/wk	\$/hr	# of weeks	Total # of Hours	Total Cost
Students:	5	12	26	16	192	\$24,960.00
Shop	1	-	25	-	25	\$625.00
Advisor	1	2	50	16	32	\$1,600.00

Parts		
Body Work:	Chassis	\$1,741
	Weather Proofing	\$80
Total Body Parts:		\$1,821
Hokoyo Laser (1):		\$2,500
Roboteq Amp		\$700
GPS:		\$199
Accelerometer:		\$10
Compass:		\$40
Rabbit Board:		\$99
Ultrasound Range Finder (4):		\$80
Emergency Light:		\$100
Solar Panel (2):		\$280
Charge Controller:		\$60
Camera Lens (1):		\$113
Laptop and software:		\$2,200
Camera (1):		\$670

Total Parts: \$8,912

Direct Labor Cost:	\$ 27,185.00
Indirect Labor Cost:	\$ 27,185.00
Overhead:	\$ 27,185.00
Total Cost:	\$ 89,367.00

Work Breakdown:

Amy Lowe

Total Hours: 73

- TSD provided 4 SRF08 Ultra Sonic Sensors
- The sensors can with a skeleton code by Joe Bradshaw that was the base of my work
- Wired the SRF08 Sensors
- Wrote code to fire SRF08 Sensors
- Wrote code to address each SRF08 Sensor
- Tested different obstacles and angles of attack
- Integrated SRF08 Sensors into master code
- Helped with design concept with MIDN Parrott to mount SRF08 Sensors
- Worked on integrating tilt detection for hills

Dylan Parrott

Total Hours: 72

- Developed multiple GoogleSketchUp designs for Solar Mount.
- Submitted work order for Solar Mount.
- Drilled holes in vehicle for securing solar mount (assisted by machine shop)
- Disassembled old laptop casing. (Lid was too large, would have hit solar panel)
- Constructed new casing (received help from wood shop and TSD)
- Submitted work order and schematic for ultra-sonic sensor mounts
- Mounted sensors (TSD provide the plastic standoffs)
- Mounted Solar Charge controller to roof (To accommodate for payload)
- Constructed new GPS mount
- Sanded and painted PVC camera hood (inside and out)
- Mounted ultra-sonic sensor to camera hood
- Currently working with 2/C Albrecht to wire the safety beacon

Caleb Burrow

Total Hours 95:

- Laptop
 - I picked out the laptop, choosing the model and customizing the hardware
 - Made contact with HP Sales Agent to begin order process. Filled-out and forwarded USNA Order Forms through Systems Department
 - Installing all drivers and software on laptop

- Basic Thresholding: First six weeks, worked half-and-half with Nathan Miller on developing thresholding software
 - Six-plot figure
 - Free-hand selection interface
 - User's decision to choose Cb-Cr, Y-Cb, or Y-Cr
- Camera: Worked half-and-half with Prof Esposito on setting up the new camera. This work was split evenly:
 - He picked out the camera
 - I figured out the firewire expresscard solution
 - Under his instruction, I took charge of determining camera settings. I wrote Weather Presets into the code.
- Program and its parts:
 - Software threshold decision tree (Main)
 - Dilation of Barrel Thresholds, through the use of structuring elements.
 - Wrote scripts to combine Ignored and Tracked Thresholds into a Final Threshold
 - My work; advice given by Professor Esposito

Nicholas Mehalic

Total Hours: 86

- Procured, Tested and Integrated Solar Charge Controller and Panels
- Drew wiring diagram for Robo-Goat Electronics
- Fix poor wiring connections
- Streamline GPS code (Significant help from Prof. Esposito)
- Procure Safety Light
- Build power supply for firewire card
- Integrate GPS and Camera Navigation (work in progress)d

Adam Albrecht

Total Hours: 85

- Developed wireless e-stop
- Reconfigured roboteq board
- Programmed PIC and RC control board