The IEEE Robotics and Automation Society (RAS) at

The University of Texas at Austin Proudly Presents:

RASlow

Built by: Chris Derichs, Chris Flesher, Chris Koci, Andrew Lynch, Richard McClellan, Vishal Parikh, Chander Sudanthi and David Yanoshak

Faculty Advisor Statement

I certify that the engineering design of the new vehicle, RASlow, has been significant and each team member has earned or could have earned at least two semester hour credits for their work on this project.

Signed ,

_____

Professor Howard Neal
Department of Electrical and Computer Engineering
The University of Texas at Austin

1.0 INTRODUCTION

This paper describes the University of Texas at Austin's (UT-Austin) design of RASlow for the 14th annual Intelligent Ground Vehicle Competition (IGVC) in June of 2006. This vehicle is the symbol of many hours of design and effort by a diverse background of voluntary student team members. RASlow has been designed and programmed to compete at all three competitive events at IGVC. Our primary focus with RASlow was to build an intelligent robotic platform with autonomous navigation using modular software and integrating commercial off-the-shelf (COTS) hardware at *affordable costs*.

1.1 Team Organization

The UT-Austin group is composed of voluntary members of the IEEE Robotics & Automation Society called "RAS" in short. The group is run entirely by students. There are six undergraduates and one graduate student participating in the contest. In terms of overall work hours, the team approximated people very roughly into the following breakdown below.

Student	Contribution	Major	Year	Hours
Andrew Lynch	PCB/Integration	EE	3rd	250
Vishal Parikh	Software/Integration	ECE/Math	4th	250
Chris Derichs	PCB/Electrical	EE	1st	100
Richard McClellan	Mechanical	ME	1st	100
Chris Flesher	Simulator/Vision	ECE	1st(grad)	100
Chander Sudanthi	Vision/Low-Level	ECE	4th	50
David Yanoshak	Electrical	EE	1st	50
Chris Koci	Labview	ECE	3rd	50
		TOTAL	**(approx)**	**950**

Table 1.0: Work Division Breakdown

2.0 DESIGN PROCESS

The process of designing the entire robotics system involves a variety of constraints on complexity and hardware. There are many deadlines and deliverables we must set to properly allow time for testing and re-evaluation. We set a timeline to have working hardware, modular software drivers and a mix of different electronics and sensor integration. After a few different progress reviews of the design at set deadlines, we made appropriate modifications according to time and available monetary resources.

2.1 Design Subsections

The team divided itself into three sections which would cover various parts of the project and eventually combine to integrate the entire system. The divisions of work were named mechanical, electrical and software. However, the groups interleaved throughout the build phase. The mechanical group was first formed to put together a basic chassis and design drive dynamics for an outdoor environment. The electrical group handled all power, PCB and general cabling across the robot. The software team acted as the overall integrator dictating the overall design and all drivers.

2.2 Design Sequence

With little experience in designing an outdoor autonomous robot, our team took a conservative approach. In early stages, the sequence basically starts with a complete solution including sensors, electronics, software and mechanical design. The overall design was based to a large degree off COTS hardware assuming reliability and a certain level of abstraction from low level devices. After establishing a sufficient roadmap of the various sub-systems, the sequence begins an iterative loop of evaluating an integrated sub-section and redesigning when needed.

This approach differs from the conventional strategy which requires much careful planning and simulation in addition to a large initial investment. Our organizational structure and budget require us to take a slightly different approach. We create a series of rapid prototypes, each one functional in and of itself, but each successive prototype gains in functionality. The prototypes are very cheaply built and mostly demonstrate a proof of concept. This allows us to have a working robot throughout all stages of the design, which maximizes our time in the build-evaluate loop.

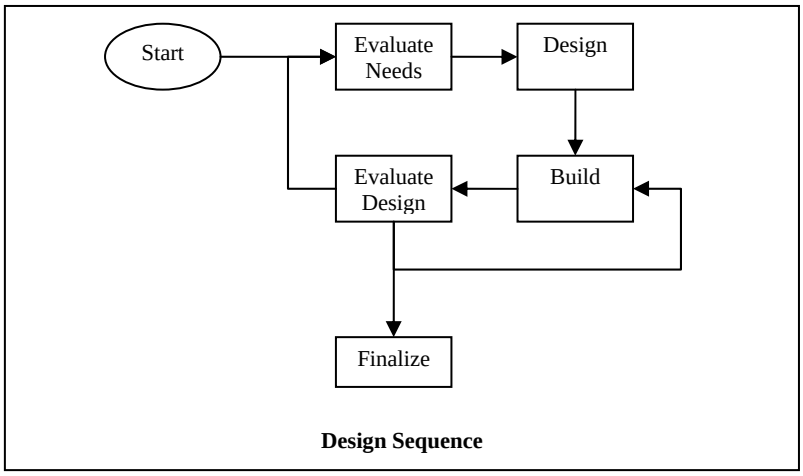

Figure 1.0: Process Flowchart

2.3 Design Innovations

Our main innovation is a custom 2D simulator for our robot and its environment. The simulator is written in Python and is detailed in section 5 of this document.

3.0 ELECTRICAL SYSTEM

The electrical system comprises a Compaq Evo laptop, a Motorola 16-bit microcontoller, a Microstrain 3DM-G Inertial Measurement Unit (IMU), a u-Blox Global Positioning System (GPS) receiver, Ultrasonic Sensors, a Camera, motors and servos.

3.1 Power System

Our system is powered by two 12V 18Ah lead acid batteries wired in series to create a 24V power supply. The batteries provide 36 Ah of power and can power the robot for approximately 6-8 hours with an ideal voltage range depending on load. The batteries power the motors and servos on the robot as well as the embedded electronics and sensors.

The electronics, comprising microcontrollers and various sensors are supplied voltage using a PowerTrends 5V switching regulator. The regulator is capable of supplying 2 Amps. We estimate that the combined microcontroller electronics draws around 300mA peak. The current to the electronics is limited by a 2A fuse to prevent short circuits.

3.2 Computers/Microprocessors

We have two computers in our system, a Compaq Evo laptop, and a 16 bit Motorola 68HC9S12 Microcontroller (henceforth referred to as the HC12). The HC12 controls the motors and servos using pulse width modulation. It also controls and reads the Ultrasonic Rangefinders and the magnetic encoders. The HC12 is on a Technological Arts micromodule board which plugs in to a custom PCB. This PCB regulates power for the embedded electronics and also routes signals between the HC12, sonars, encoders, and motors. For graphical detail of schematic and layout refer to Appendix B.

The Compaq Evo Laptop does most of the processing work. It is a 1.5Ghz Athalon XP with 256mb of DDR RAM and runs Windows XP. It has two USB ports which we use to interface to the sensors and an internal WiFi 801.11g card which allows us to communicate wirelessly.

3.3 Sensors

We have two sensor systems: state estimation and hazard detection. Elements of the state estimation system are the GPS, IMU, encoders, and camera. Elements of the hazard detection system are the camera and the rangefinders.

We have used two different GPS receivers on the robot. Eventually we decided our original RoyalTek module was too slow and inaccurate. The RoyalTek has been replaced with a u-Blox RCB-LJ with a similar size and shape which gives a 3 feet circular error probability (CEP) at 4Hz rather than RoyalTek's ten meter accuracy at 1Hz. The u-Blox also provides 3D Doppler velocities which greatly assists navigation calculations [1].

The IMU is a Microstrain 3DMG IMU. The 3DMG uses tri-axial accelerometers, magnetometers and angular rate sensors to provide accurate roll, pitch, and heading information. The IMU provides data at rate of 75 MHz with a precision of .01 degrees [2].

The encoders consist of Optek OU90 Hall-effect magnetic sensors. They generate a digital signal indicating the presence or absence of a magnetic field above 10 Gauss with a hysteresis curve spanning to 100 Gauss. Magnets on the pinion (shown below) of the motor shaft trigger the sensors. The encoders achieve an accuracy of 60 counts per revolution on the output shaft [3].

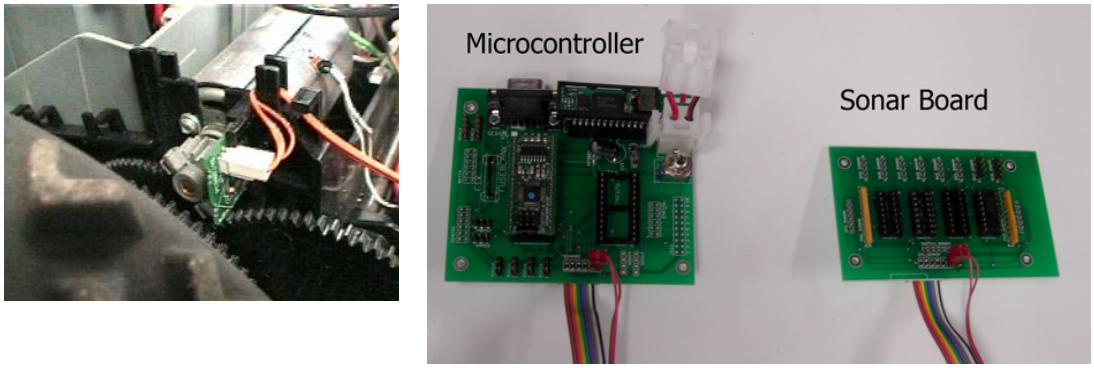

Figure 2.0: Encoder Mountings and Custom PCBs

The camera is a Logitech QuickCam which uses a variant of the sony CCD sensor. It has a resolution of 320x240 pixels at a frame rate of 30 fps and a reduced 15fps for the natural resolution of the CCD, 640 x 480 pixels [4]. We have noticed from testing the CCD is limited in outdoor lighting due to constant tuning of brightness levels. We hope to remedy the problem using our quick Labview calibration program to adjust for a range of light conditions.

The rangefinders are SRF04 ultrasonic range finders purchased from Acroname [5]. The have a maximum range of about 8 feet and a maximum spread of 45 degrees as described in the diagram below.

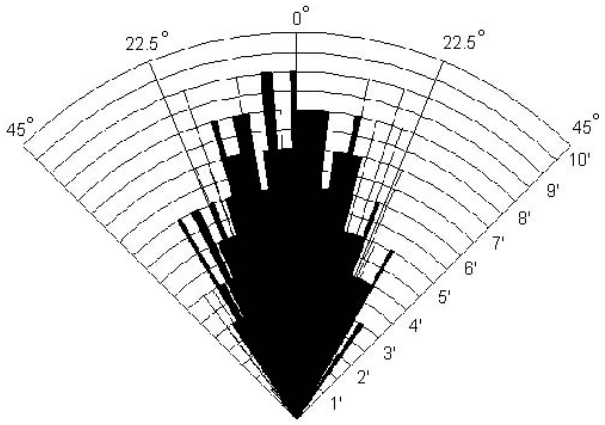

Figure 3.0: Sonar Beam Pattern [5]

3.4 Communication

We use RS232 asynchronous serial for most of the data acquisition on the laptop. The devices have hardware support for this protocol and it is efficient and fulfills our bandwidth requirements. Additionally, USB to serial converters are readily available, and fairly inexpensive. The GPS, IMU, and HC12 all communicate with this protocol. The HC12 reads from the Encoders and

Ultrasonic Rangefinders as well as generating signals to control the motors. We use USB to receive data from the webcam. All interprocess communication is done via User Datagram Protocol (UDP). The reason for using UDP versus Transmission Control Protocol (TCP) is mainly simplicity of implementation and testing.

4.0 MECHANICAL SYSTEM

For the mechanical system, we needed a very robust and durable system in order to handle a rugged outdoor terrain. Due to limited resources and the desire for a low maintenance system we planned to use a COTS drive train. Our first prototype was constructed using the chassis and drive train from IFI Robotics commonly used on FIRST robots in the annual competition. While rugged, this platform did not have the flexibility needed for turning on a grassy terrain. Our second prototype began as a COTS R/C Tank and was disassembled and reequipped with our own speed controllers, microprocessor and computer for autonomous navigation. This platform worked quite well for a while for software testing, but later the plastic gears and sprockets driving the treads were torn up by the stress of an outdoor environment, so we looked for another solution.

For the final mechanical design, we continued with the COTS ideology and used the chassis and drive train from a robotic lawnmower, the RoboMow outfitted with our own sensory components and control system. Since it was designed for mowing lawns, we were sure it would be robust enough to handle up to fifteen degree inclines even with payloads up to one hundred pounds. This lawnmower had two main drive wheels with plenty of traction and one large caster in the front to allow a very sharp and easy turning radius, unlike our first prototype. 3D models of the RoboMow are below

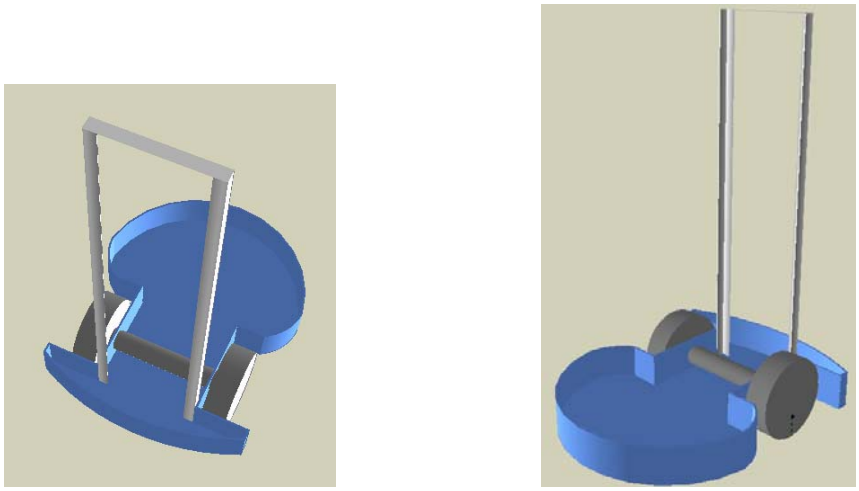

Figure 4.0: 3D Model of the RoboMow

After removing the commercial electronics and cutting off the back end of its protective shell, an innovative mounting system was created that also kept the center of gravity relatively unchanged. A payload, two laptops, speed controllers, a breaker panel, our microprocessor, and eight sonar range finders all had to be mounted relatively close to the bottom of the chassis to keep it from tipping over. To accomplish this, the lawnmower blades and motors were removed. On top of these, we attached a hinged box that rotates ninety degrees, houses both laptops inside and holds the speed controllers and breaker panel on the bottom. The laptops both rest on bubble wrap for shock protection if the robot hits a bump or tips over. On the front two corners of the box, two vertical aluminum angles were mounted with an array of mounting holes for the range finders so that the height can be easily adjusted. The range finders themselves are attached to a piece of flat aluminum bent at 22.5 degree angles. Since the sonars have a range of 45 degrees, this bending strategy gives us double coverage on everything with 8 feet of the front to a total degree of 180 degrees. Our IMU, Camera, and GPS had to be mounted high for optimum sensor performance, so we put a tower above the two drive wheels for the mounting of these three components.

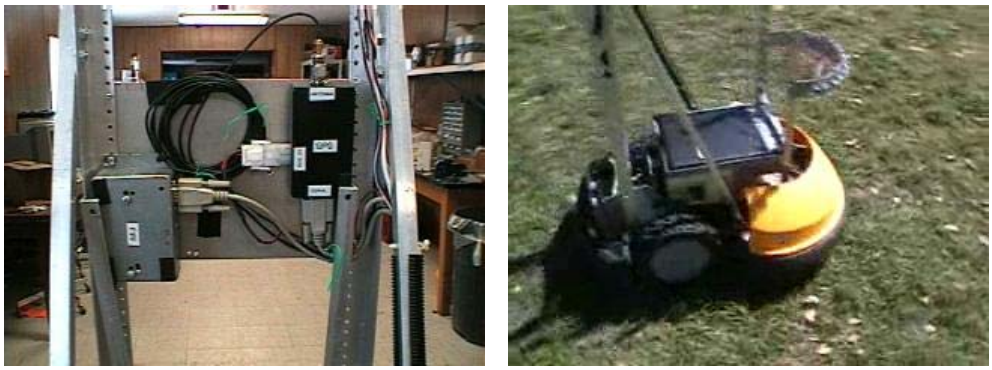

Figure 5.0: RASlow Sensor Mountings

5.0 SOFTWARE

Our software is a distributed model with different processes for the essential portions of the software design. Our goal was to create an efficient and scalable system with a powerful user interface. We also wanted a system that was easy to develop in a parallel programming environment. We chose to use the Labview environment for the user interface and some of the drivers. We used the C programming language for the control system, and some of the drivers. We also used C and specifically the Open CV library for the image processing. Our system consists of 4 main processes as shown in the communication diagram below: Labview, Auto, IMU, and OpenCV.

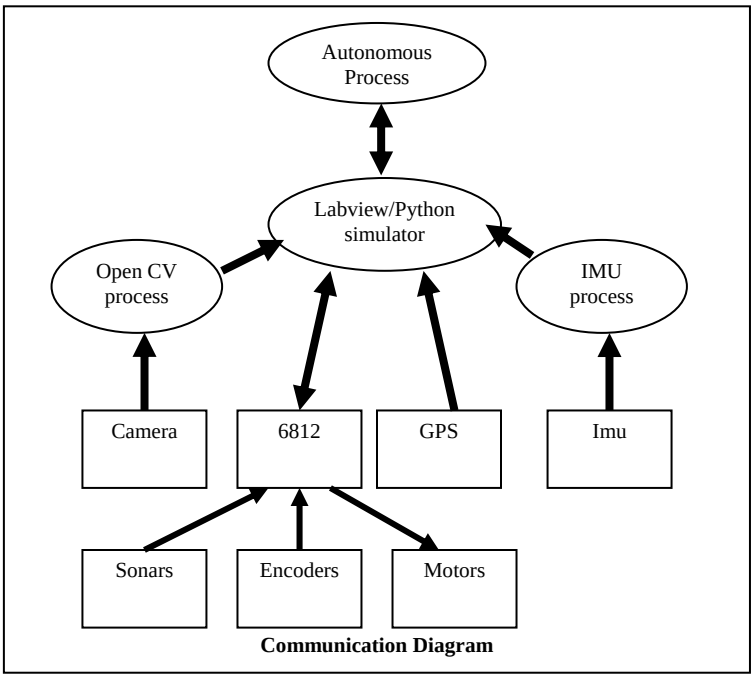

Figure 6.0: Communication System Diagram

5.1 Labview

Labview was chosen as the primary testing and development environment for many reasons. First, it is easy to develop a graphical user interface with indicators and controls for testing and debugging. Second, Labview is inherently multithreaded and event driven, facilitating a distributed and efficient design. Lastly, Labview has a library of useful tools for communication and data processing that we can take advantage of in our design.

The GUI is useful for rapid prototyping and testing, a skill which is essential in our design strategy. The front panel for our user interface can be seen below.

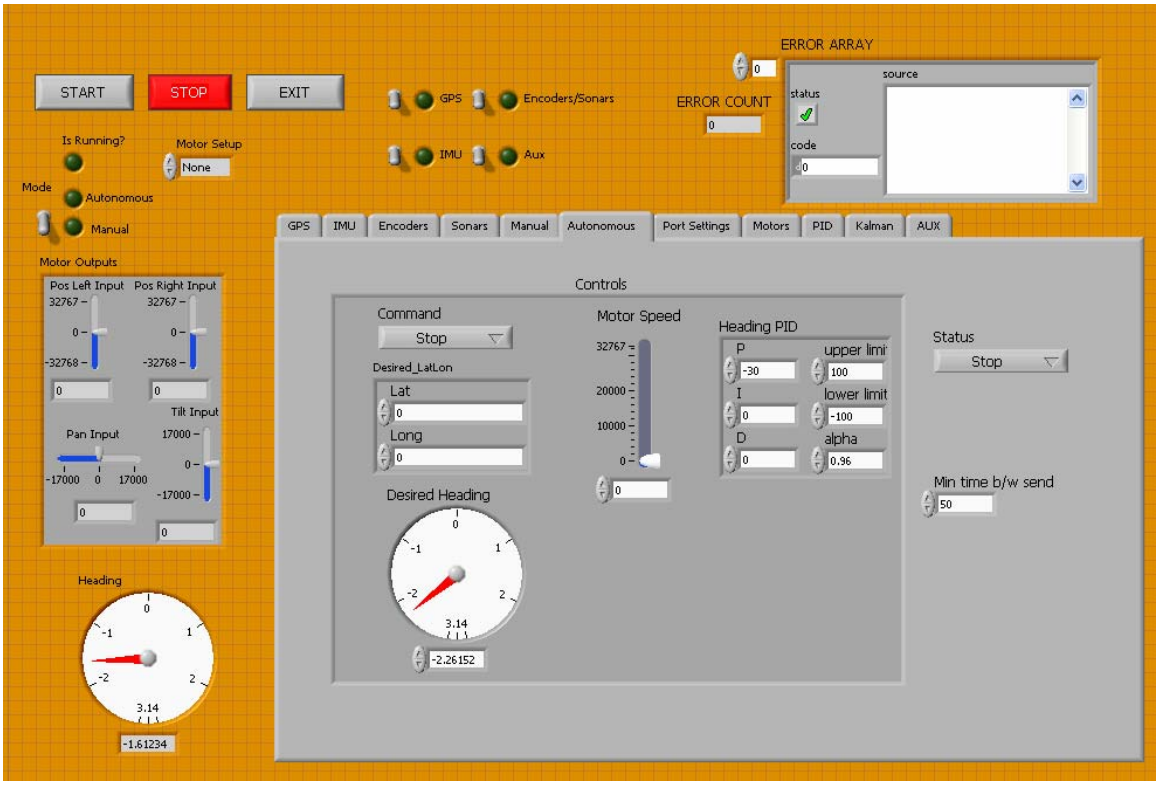

Figure 7.0 : Labview Basestation

A multithreaded, event driven design is essential for reading from and writing to multiple asynchronous I/O sources. As described above, our design has many sensors and devices from which data is gathered simultaneously. Since there is little hardware abstraction between the main CPU and the devices, it is necessary to use a multithreaded design to avoid data loss when reading from and writing to these devices. The event driven structure of Labview allows maximum utilization of CPU cycles, and little time wasted polling devices.

Labview's library of low level tools greatly simplifies design. Labview has controls for RS232 serial communication, TCP/IP, UDP, USB and other protocols that are essential to our design. The ability to use the built in Labview components, while not as efficient as a custom C or C++ design, was much faster to design and test, and is much more stable. In the Labview process we receive and parse data from the GPS and the HC12, we format and display this data for monitoring and we pass it via UDP to the autonomous process.

5.2 Control

The autonomous process implements the main control system; it receives data from Labview, processes it and sends motor commands back to Labview. All inter-process communication is done over UDP. This modularization allows transparent replacement of Labview with the Python simulator described below. We also have a control system implemented in Python that can replace the autonomous process.

We are currently testing several options for the automatic control system. One option is to use a Bayesian filter to estimate the state of the robot and its environment based off a thorough model of the system. The state estimate gives us a map of the robot's surroundings and a confidence level that the map is correct. We can then use this map to plan a path to the goal that gives us the highest chance of success.

The IMU process is for acquiring data from the IMU, parsing it and sending it to Labview. We elected to have this as a separate process and not built into Labview like the other drivers so as to minimize latency from the IMU. Labview can deliver a minimum of 100ms latency, but for accurate state estimation, we desire 10ms of latency.

5.3 Vision

The OpenCV process is for acquiring and processing the image from the webcam. We used C++ and the OpenCV library to process the image and extract information about hazards. We tried to remove as many user defined parameters as possible and eliminate false positives due to noise while at the same time keeping our algorithm as robust as possible.

The vision algorithm first gets the raw image from the camera and rescales the image size to 160x120 pixels. This resolution was empirically chosen because the processing time and information losses were deemed acceptable. The algorithm converts the image to grayscale, splits the image into two sub-images and begins the search for lines in each of the two half images.

The line detection algorithm uses the OpenCV implementation of the Hough transform to find the three most prominent lines in each image. If no prominent lines are in the image the Hough transform will often detect lines around the square edges of the image or detect non-stationary lines due to noise, in which case the data is ignored. Note that the Hough transform is much more

robust than any methods that binarize an image using a user defined intensity threshold because there are no parameters to be adjusted.

The line extraction algorithm then converts the three most prominent lines into robot centric coordinates using an inverse perspective transformation. This transformation allows us to represent the detected lines in a robot centric frame of reference, which is of more value for control purposes than a camera centric frame of reference.

To filter spurious data, the angle and perpendicular distances of the three lines are found and the lines that have the most common properties are averaged. This filter, combined with the filter that ignores edge lines, greatly reduces the number of false positives the algorithm encounters. Shown below is a distorted grass sample image [7] which displays the final Hough green line and binary image output of the brightness threshold.

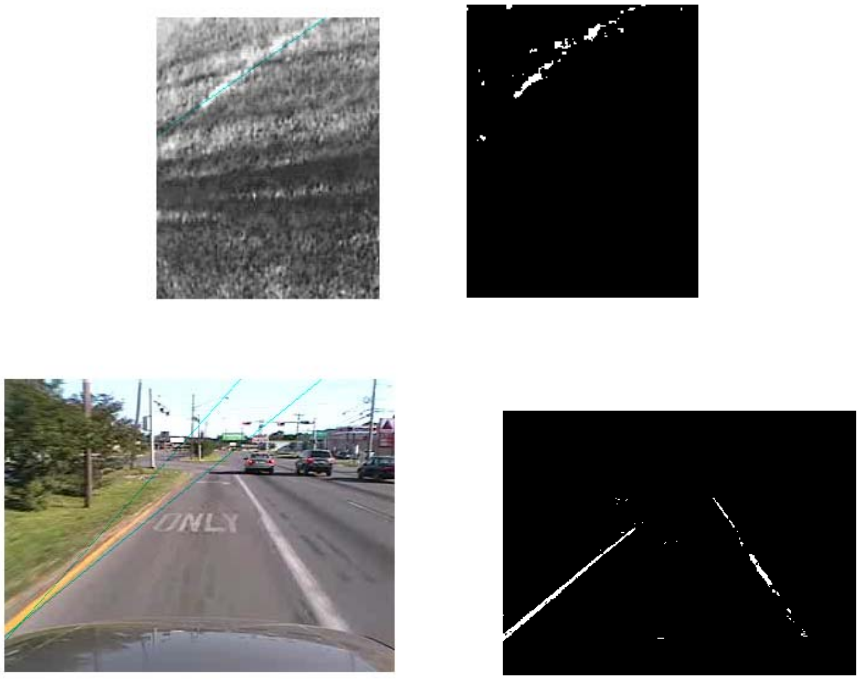

Figure 8.0-8.4: Vision Processing Images

We also tested in a few driving scenarios shown above to prove the algorithm will be robust for a wide variety of terrain shown below. Notice how the grass and line are much easier to distinguish than the highway line. As a stable testing platform we implemented “OpenView Vision” a Labview-OpenCV testing environment to quickly test and modify parameters on the fly [8]. The

debug console is shown below with graphical Labview switches to modify Hough parameters and boundaries of detected images.

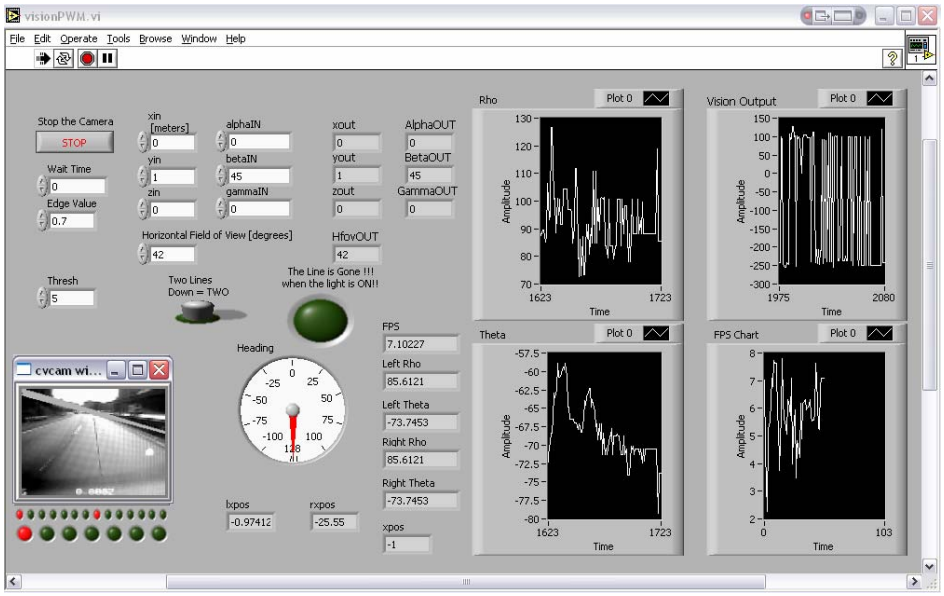

Figure 8.5: Vision Base station Console

5.4 Simulation

Testing a control system on a physical robot can be very time consuming. Battery life, setup time, and minor setbacks can limit the amount of time available for testing. For these reasons, testing an automatic control system in a simulated environment can speed development time by orders of magnitude. We decided to develop a custom robotic simulator, reproducing the input / output characteristics of every component of our robot in simulation in order to model how the robot would behave in real life. Screenshots are shown below.

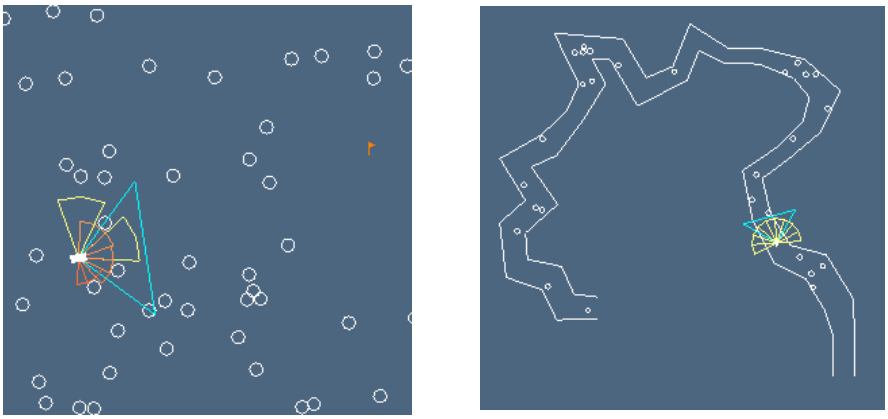

Figure 9.0: Simulator Screenshots

To fully model the robot we had to consider the input / output relationships of the sensors. There are four sensors to be modeled: the sonars, inertial measurement unit, GPS, and the camera.

To model the sonars we plan on developing a test stand consisting of a single sonar mounted on a servo and some electronics to send control signals to the servo and pulse input to the sonar and to receive the returned data from the sonar. This test stand will be placed in an open field in front of a cylindrical object (a plastic trash can) made to resemble a construction barrel. Data will be collected every 1-2 degrees in order to build an input / output map for the sonar.

To model the GPS unit we simply monitored the GPS signal over a three hour period in order to develop a noise model for the GPS. We model this noise either by designing a FIR or IIR shaping filter based off of the power spectral density or by multiplying the power spectral density with a Fourier transformed sequence of randomly generated white noise and then taking the inverse Fourier transform of the result.

Because the inertial measurement unit had a fairly quick response time and we anticipate no magnetic distortion during competition, we did not feel that it was necessary to generate a complicated model for this sensor. Instead we simply used the robot's heading as the IMU measurement. This approximation works well as long as there are no strong magnetic sources near the IMU unit.

The camera was modeled by finding the lines inside of the simulated camera's bounding box and performing a weighted average of the line distance and angle, closer lines being weighed more. Noise was not modeled for the camera because the true camera's noise model was deemed too difficult to model.

The simulator was written in Python / OpenGL and communicates with the control program over a UDP socket. In this way the simulator emulates the sensory information generated by the real sensors. Course information for both the GPS waypoint part of the competition and the obstacle course part were either randomly generated or manually specified by a companion application, MapBuilder (shown on the next page).

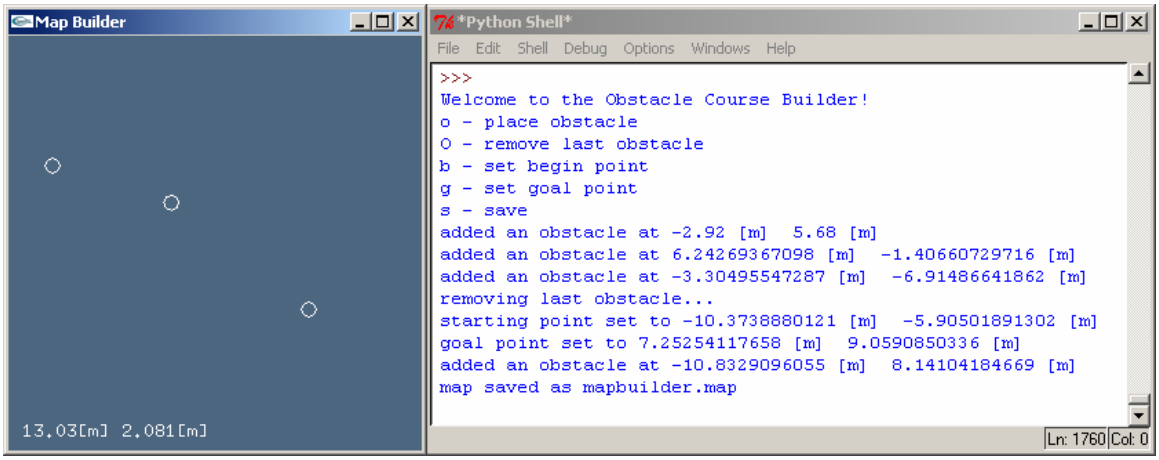

Figure 9.1: Map Builder

6.0 PREDICTIONS

Due to cost and time constraints, we are using a modified RoboMow Chassis and drive train. The motors and gear box in this drive train allow a maximum speed of around 2 MPH. However, as the robot was designed for outdoor use, we predict that we should be able to carry the payload up 15% grades with no problem. We predict that the 24V lead acid battery will provide six to eight hours of battery life; however, we are limited by the 4 hour battery life of the laptop.

Our camera can see objects in excess of 20 feet depending on the ambient brightness and the contrast, but our Ultrasonic Rangefinders have a maximum range of 9 feet. Our GPS has a maximum accuracy of 3 feet, but in practice on a clear day, far from buildings, we achieve an accuracy of 10 feet. Our waypoint navigation is bounded by our GPS navigation.

7.0 SAFETY

Throughout the many different robot vehicles we have built, small or large, flying or ground, there is always a safety plan built into the design. RASlow has approximately 100 lbs in weight driven by powerful high-torque motors. When objects get in the path of the robot, the high torque can cause significant ramping of the motors, therefore an electrical safety stop (E-stop) was essential to the operation of the robot. On top of the hardware E-stop, we built features into the software to immediately stop any operation in case of a problem. From the early stages of construction, we incorporated a comprehensive safety design for interaction with the vehicle such as eliminating sharp edges, adding bumpers and covering exposed wires. For all power wires we used Mate'N'Lock connectors that will avoid any type of reverse polarity. Overall, we looked

closely at the IGVC regulations as well as other teams' safety designs to plan extensive safety when around the robot.

8.0 COST

Our costs are detailed in the table below. Most of our equipment was donated or inherited from previous projects. Therefore, cost for this project has remained extremely low.

Quantity	Part	Retail Price	Our Price
1	Friendly Robotics RL850 Robomow	$1,300	$400
1	Compaq Evo laptop	$1,299	$0
1	Motorola 68HC9S12 Microcontroller	$50	$0
1	Microstrain 3DM-G IMU	$1,495	$0
1	Logitech Webcam	$40	$25
1	Ublox GPS Unit	$110	$0
8	Ultrasonic SR04 Rangefinder	$200	$0
2	Optek OU090 Hall-effect Sensor	$3	$3
2	IFI Victor 884 Speed Controller	$230	$0
1	Custom PCB for Sonar and Microcontroller	$66	$66
Total		$4,793	$494

Table 2.0 : System Level Budget

9.0 CONCLUSION

RASlow is a culmination of effort of the RAS team from the University of Texas at Austin. As a rookie team into the competition, our contributions to high speed navigation are somewhat constrained due to hardware. However, RASlow has broken the boundary for achieving low-cost reliable robots which could be applied in consumer applications such as mowing lawns or surveillance. In addition, our unique Labview-OpenCV Vision Engine and Robotic Lawnmower chassis will be a defining aspect of our presence at the IGVC competition.

Appendix A: REFERENCES

[1] GPS U-blox datasheet , http://www.u-blox.com/customersupport/Data_Sheets/RCB-4H_Data_Sheet(GPS.G4-MS4-06034).pdf

[2] 3DMG IMU datasheet, http://www.microstrain.com/3dm.aspx

[3] Optek Magnetic Encoders http://www.optekinc.com/pdf/OH090-3075.pdf

[4] Web Cam Spec Sheet, http://www.brickfilms.com/wiki/index.php?title=Logitech_QuickCam_Pro_4000

[5] Acroname Sonar SR04 spec Sheet, http://acroname.com/robotics/parts/R93-SRF04.html

[6] UT-Austin IARC 2005 paper, http://iarc1.ece.utexas.edu/~lynca/final_documentation/utiarc2005.pdf

[7] VT - Johnny 5, Grass Image 2004 , http://www.igvc.org/deploy/design/reports/dr96.pdf

[8] OpenView Vision Code, http://ras.ece.utexas.edu/learning.php

Appendix B: SCHEMATICS

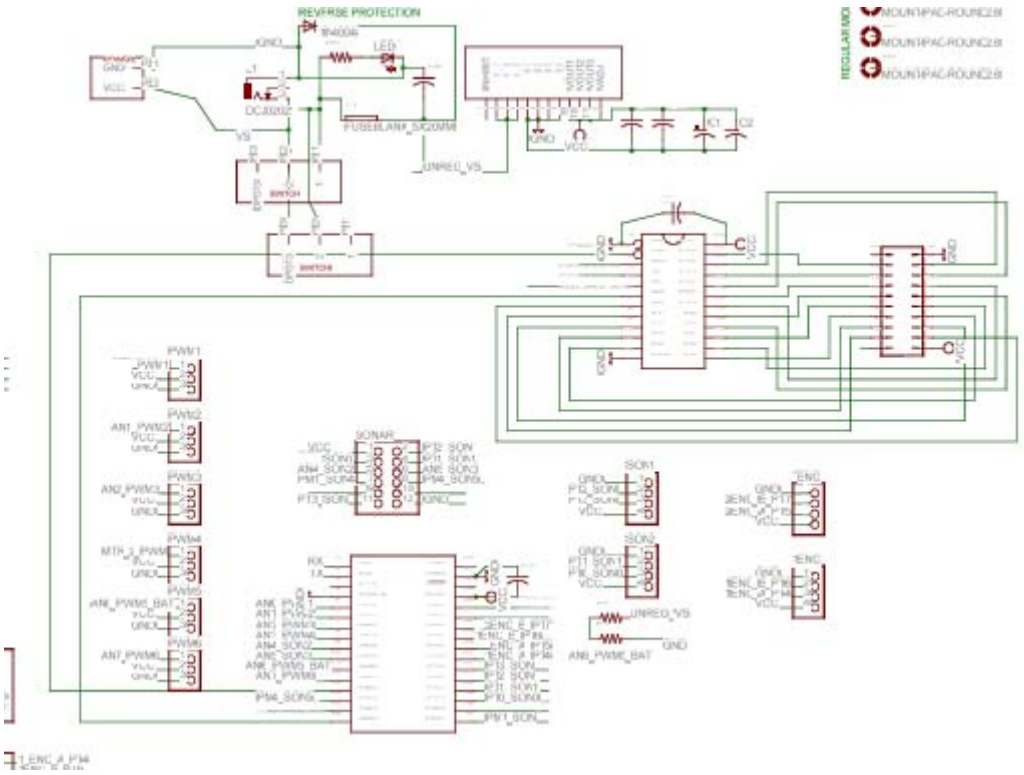

Figure 10.0: Microcontroller Schematic

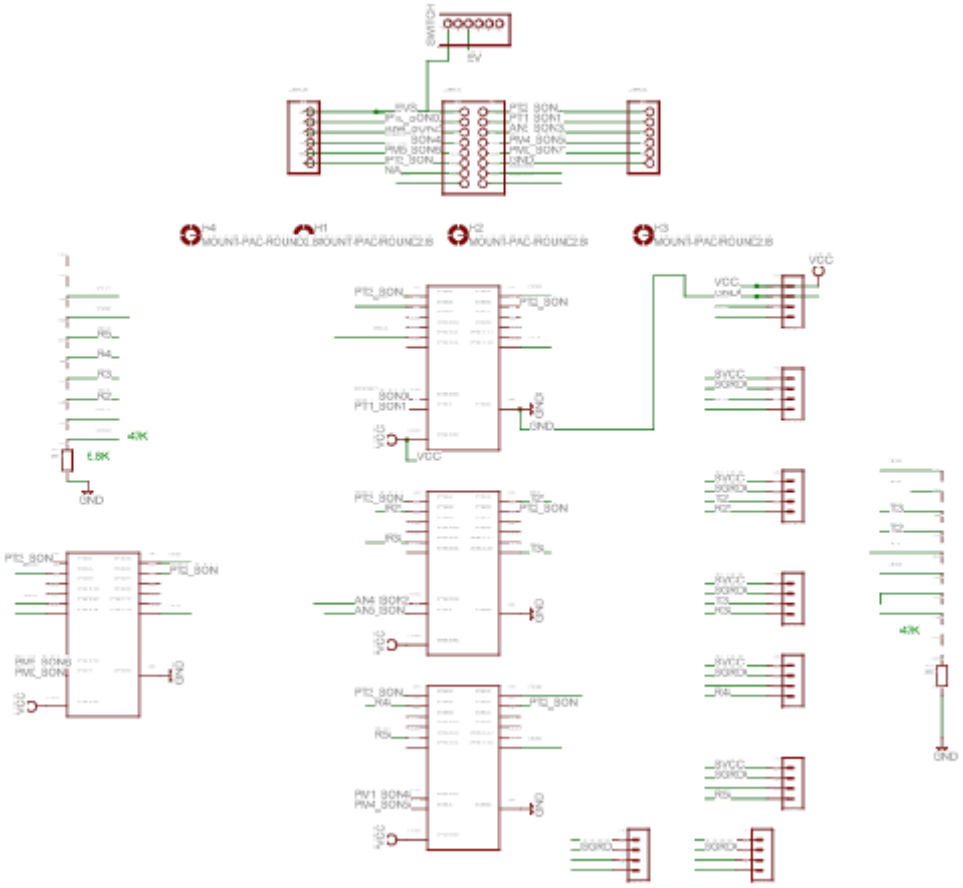

Figure 10.1: Sonar Controller Schematic

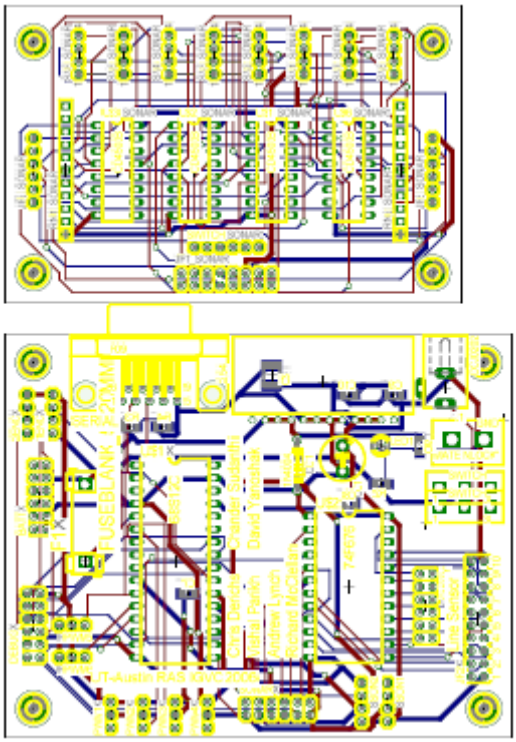

Figure 10.2: Layout of Sonar (top) and Microcontroller Circuit (bottom)