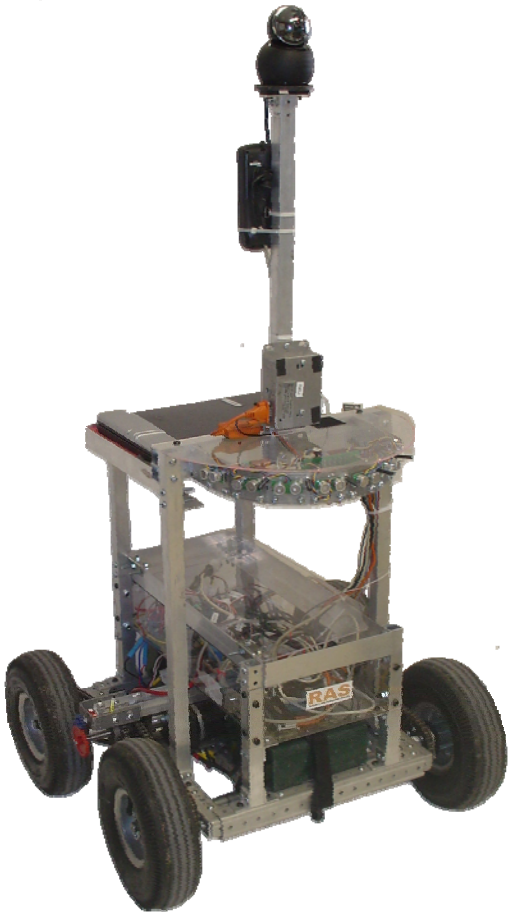
The IEEE Robotics and Automation Society (RAS) at
The University of Texas at Austin Proudly Presents:

BlastyRAS



Designed by Christina Lam, Chris Flesher, Kevin Baker, Andrew Lynch, Richard McClellan, Faizan Kazi, Matt Wanninger and David Yanoshak

Faculty Advisor Statement

I certify that the engineering design of the new vehicle, BlastyRAS, 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 Jonathan Valvano
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 BlatstyRAS for the 14th annual Intelligent Ground Vehicle Competition (IGVC) in June of 2007. This vehicle is the result of numerous hours of hard work and effort by a diverse group of voluntary student team members. BlastyRAS has been designed and programmed to compete in all three competitive events at IGVC. Our primary focus with BlastyRAS is 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 seven undergraduates and one graduate student participating in the contest. In terms of overall work hours, the approximated hours each individual worked is shown in Table 1.

Student	Contribution	Major	Year	Hours
Andrew Lynch	PCB/Integration	EE	4th	250
Kevin Baker	Software/Integration	ECE	3rd	250
Matt Wanninger	PCB/Electrical	EE	1st	50
Richard McClellan	Mechanical	ME	2nd	150
Chris Flesher	Simulator/Vision	ECE	2nd (MS)	300
Christina Lam	Vision/Low-Level	ECE	4th	50
David Yanoshak	Electrical	EE	2nd	50
Faizan Kazi	Electrical	EE	1st	50
		TOTAL	**(approx)**	**1200**

Table 1: Work Division Breakdown

2.0 DESIGN PROCESS

The process of designing the entire robotics system involves many deadlines and deliverables we must set to properly allow time for testing and re-evaluation. Deadlines include working hardware, modular software drivers and a mix of different electronics and sensor integration. Once a deadline approaches, we made appropriate modifications according to time and available monetary resources.

2.1 Design Subsections

The team divided itself into three sections, mechanical, electrical and software. Each section would cover specific parts of the project and would eventually combine to integrate the entire system. The mechanical group was first formed to put together a basic chassis and design the drive dynamics for an outdoor environment. The electrical group handled all power, PCB and general cabling across the robot. The software team programmed algorithms that integrated the hardware and electronics.

2.2 Design Sequence

With little experience in designing an outdoor autonomous robot, our team took a conservative approach. In early stages, the design started with COTS hardware of integrated sensors, electronics, software and mechanical components. The reason for using COTS hardware was due to its reliability and level of abstraction from the low level devices. After establishing a sufficient roadmap of the various sub-systems, the sequence continues with constant evaluation of each sub-system and redesigning when needed.

Due to our organizational structure and budget, the design sequence we adapted differs from the conventional strategy, which requires intensive planning and simulation in conjunction with a large initial investment. We created a series of rapid prototypes, with each prototype being fully functional and independent from one another. Also, each successive prototype in the series included additional functionalities. The goal of each prototype was to be built cheaply to correctly 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 as shown in Figure 1.

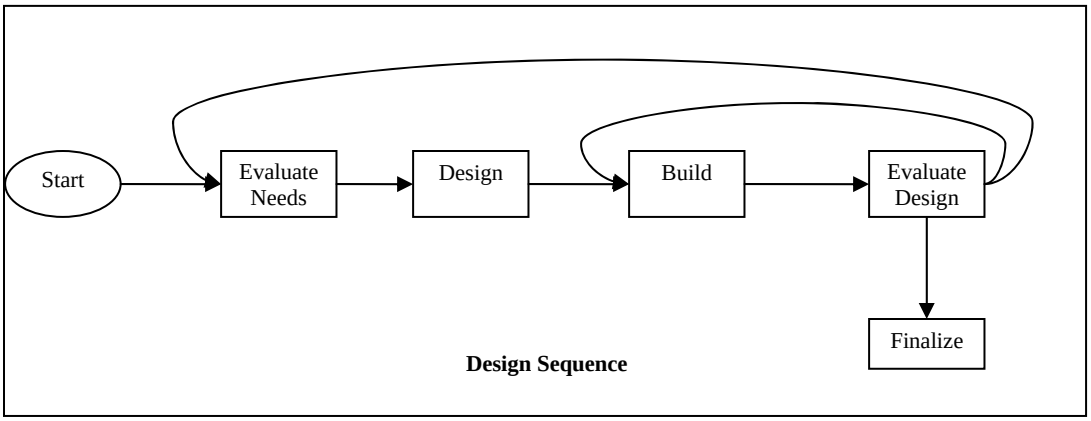


Figure 1: Process Flowchart

2.3 Design Innovations

We created innovation in inexpensive design while still obtaining reasonable performance for achieving the outdoor autonomous routine. A large majority of our work was focused on software algorithms and testing. Our main innovation is a custom 2D simulator for our robot and its environment. The simulator is written in Python and is described in greater detail in section five of this document.

3.0 ELECTRICAL SYSTEM

The electrical system comprised of a Compaq Evo laptop, a Motorola 16-bit microcontroller, a Microstrain 3DM-G Inertial Measurement Unit (IMU), a Garmin 72 Global Positioning System (GPS) receiver, Ultrasonic Sensors, a camera, motors and pan/tilt servos. We created three unique custom printed circuit boards (PCB) for the microcontroller, sonar and relays.

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 18 Ah of power and can power the robot for approximately 6-8 hours with ideal voltage range depending on course terrain. A separate NiMh 12V battery powers the embedded electronics and sensors.

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

3.2 Computers/Microprocessors

We have two computers in our system, a Compaq Evo laptop, and a 16 bit Motorola 68HC9S12XDP256 Microcontroller (S12). The HC12 controls the motors and servos by using pulse width modulation. It also controls and reads the sonars and the optical encoders. The HC12 is on a Technological Arts micro-module board, which plugs into a custom PCB as shown in Figure 2. This PCB regulates power for the embedded electronics and also routes signals between the HC12, sonars, encoders, and motors. For graphical detail of the schematic and layout, refer to Appendix A.

Figure 2: Custom microcontroller PCB for S12

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 sonars.
We have tested two different GPS receivers on the robot. Eventually we decided on the Garmin 72 GPS, which gives WAAS capability and a 3 meter accuracy [1]. We communicate with the GPS using a RS-232 serial link. The GPS has handheld features such as embedded button controls allowing the user to view all information on the GPS for quick debugging.

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 Hz with a precision of 0.01 degrees [2].

The encoders are GrayHill 61R128 Optical sensors. They generate a digital signal indicating the presence or absence of light. The casing of the encoder is completely sealed, allowing the encoder to function in any lighting environment without calibration. We connect the encoders to the shaft by using plastic gears as shown below in Figure 3. The encoders achieve an accuracy of 128 counts per revolution on the output shaft [3].

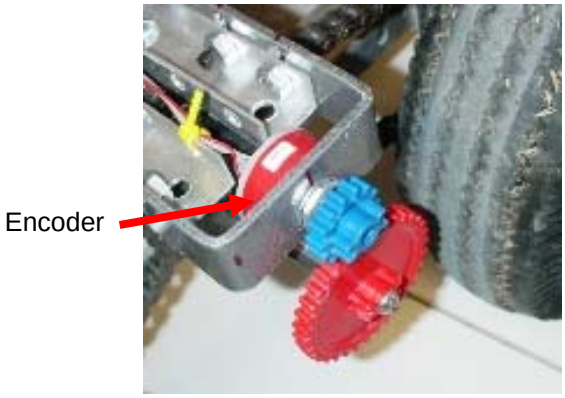


Figure 3: Encoder Mountings

The camera is a Logitech QuickCam Orbit, which uses a variant of the Sony CCD sensor. It has a resolution of 640 by 480 pixels at a frame rate of 30 fps and a still shot natural resolution of 1280 x 960 pixels [4]. From testing, we have noticed the CCD's performance is limited in outdoor lighting due to constant tuning of brightness levels. We hope to remedy the problem by using our quick OpenCV-Python calibration program to adjust for a range of light conditions.

The sonars are SRF04 ultrasonic range finders purchased from Acroname [5]. The have a maximum range of approximately 8 feet and a maximum spread of 45 degrees as described in Figure 4.

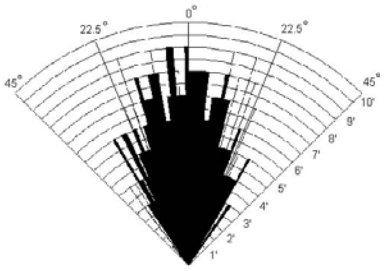


Figure 4: Sonar Beam Pattern [5]

3.4 Communication

We use RS232 asynchronous serial for most of the data acquisition on the laptop. Even though the laptop does not have COM ports, USB to serial converters are readily available and are fairly inexpensive. In addition, the devices

and sensors (GPS, IMU, and HC12) have hardware support for this protocol. Therefore, making this protocol efficient and fulfilling our bandwidth requirements. All interprocess communication is done via User Datagram Protocol (UDP) instead of Transmission Control Protocol (TCP) because UDP is simpler to implement and test. The HC12 reads from the Encoders and sonars as well as generates signals to control the motors. We use USB to receive data from the webcam. Using a protocol that can be layer on top of IP (such as UDP/TCP versus shared memory or pipes) also allows additional modularity and remote debugging.

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. For the final mechanical design as shown in Figure 5, we continued with the COTS ideology and used the chassis and drive train, but switched the IFI gearboxes with 56mm Banebot gearboxes. We attached two CIM motors to each Banebot gearboxes and used #35 chains. Our motors free spin at 5310 rpm, which goes through 12:1 gearbox ratio and a 24/18 sprocket chain ratio to make an overall ratio of 16:1. With four 10" inflatable wheels, we should move approximately 5 feet per second with 5488 oz-inches of torque. With these specifications we will meet the max speed limit of the vehicle with sufficient amount of torque for climbing the fifteen percent grade ramp. In addition to the torque, the four 10" wheels have high tractions to make the climb on the ramp easier.

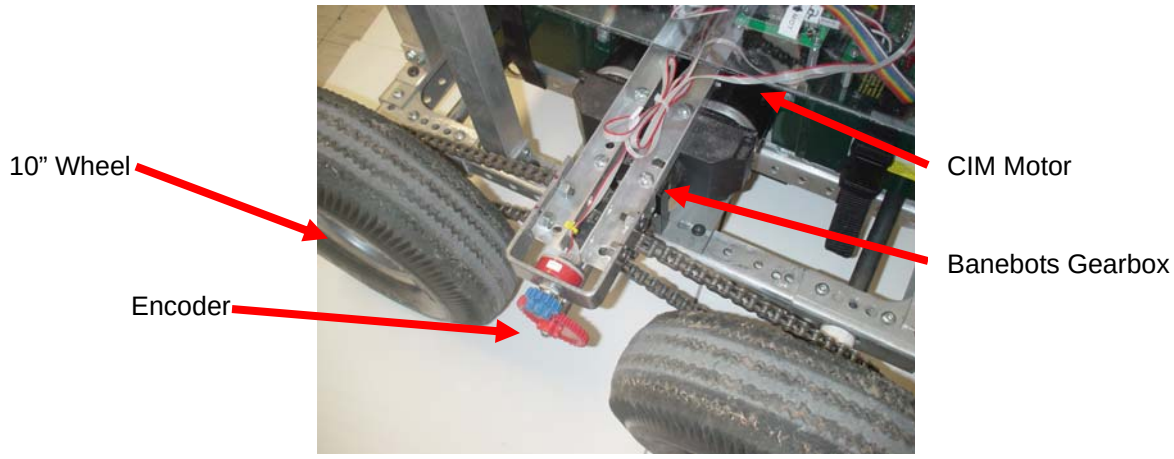


Figure 5: Drive System of BlastyRAS

For mounting structures, we decided on a lexan box to house the majority of the electronics. For the speed controls and power distribution, we built a small lexan compartment underneath the robot. The electronics mounting system was created to keep the center of gravity relatively low. As a result, the payload, speed controllers, breaker panel, 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. The laptop was mounted above the payload to allow for easy access while testing the system. From the base of the robot, two vertical aluminum square extrusions were mounted. Along the entire length of the square extrusions, mounting holes were drilled so the range finders can be easily attached at different

heights. The range finders themselves are mounted on 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 obstacles 8 feet from the front of the vehicle with a total range of 180 degrees. The IMU, Camera, and GPS had to be mounted high for optimum sensor performance, so we put a tower above the two front drive wheels for the mounting of these three components. See Figure 6 for a view of the tower.

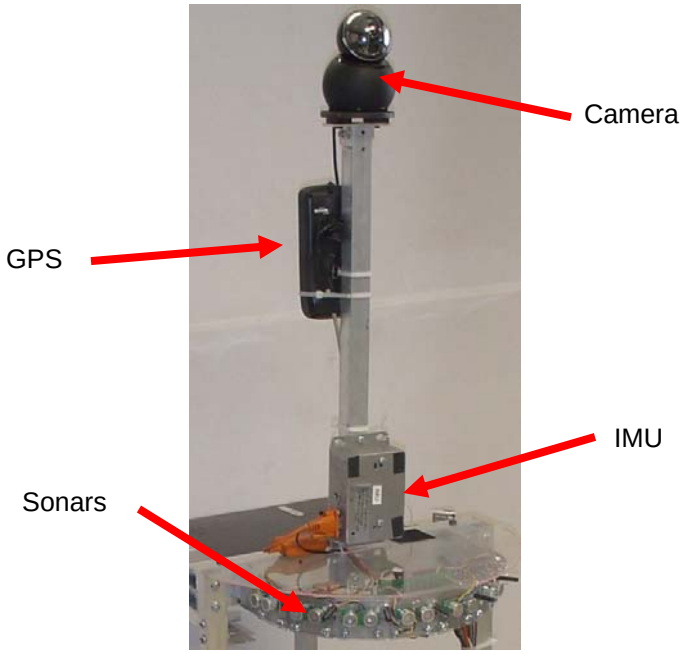


Figure 6: BlastyRAS Sensor Mountings

5.0 SOFTWARE

The main design objective was to make it easy for our programmers to prototype and debug new ideas. This is important, especially at competition where new changes may have to be coded up quickly with limited manpower. The software is broken up into several different programs that run in parallel. Each program communicates with the others by sending and receiving time stamped messages. Figure 7 shows a high level flow chart of the communication system.

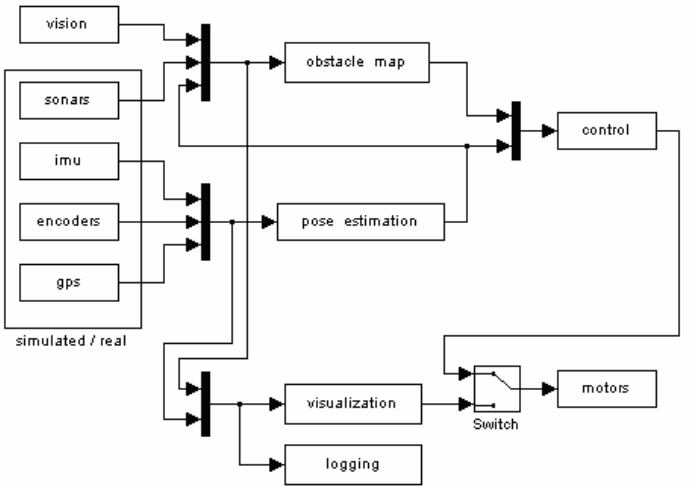


Figure 7: Communication System Diagram

Our team has an efficient method to debug algorithms that can minimize the amount of time in testing. If a situation arises that caused the vehicle to fail, we have the ability to log data and replay any troubling scenarios after they occur. Our graphical user interface allows us to visually inspect the state of the robot at any moment in time. GPS, sonar, position, heading, and camera data are displayed together in an intuitive form. In addition, custom debugging information specific to the vehicle mode of operation can be displayed. While the vehicle is running, all of the sensory information is continuously logged. We have also developed a playback mode that can be used to debug the robot from logged data.

We chose to use the Internet Communications Engine (ICE) to link together programs by using a publish/subscribe architecture. ICE is a middleware alternative to other remote procedure call libraries like CORBA or COM and has support for a number of languages including C++, Java, Python, Ruby and several others. The publish/subscribe architecture allows several programs to listen to a piece of data simultaneously. To allow judges to communicate with the robot, we have also built an ICE to JAUS bridge using the open source implementation OpenJAUS.

This architecture offers a number of benefits. The first is message abstraction. Most of our code is written in Python, but we have the ability to incorporate C++, Java, and code written in several other languages. This allows programmers with varying backgrounds to contribute to the project. Also, we have the ability to add multiple CPU's very easily if more processing power for a specific algorithm is required. The second benefit is reduced latency. Since programs are being run in parallel, a single computationally intensive program will not keep the computer from reading a serial port. This is similar to multithreading except we can focus on testing each individual driver instead of debugging multithreaded code, which can be very difficult. There are several other benefits of this architecture that our group did not have time to take advantage of. One example is reliability, if a message passing program crashes unexpectedly the main communications engine will recognize flawed messages and restart the troubled program.

Our design choice does have a couple of significant drawbacks that must be addressed: timing and network overload. Data from different time stamped messages must be fused together. But what if a packet arrives late due to some communication error? Also what happens when we send more data over the network than it can handle? We get around the timing issue by time stamping data. Each program has its own way of dealing with the data it receives. Usually if some sensory data packet gets delayed it is simply dropped in favor of more recent data. To prevent the network from becoming saturated with data we have taken a couple of steps. First of all data is broadcast using UDP, so in the worst case scenario if sensory data cannot be transmitted during a cycle it is simply dropped. Next data coming from high bandwidth sources (e.g. a webcam) must be requested before the data is broadcast over the network. The bandwidth priority sequence scales the network traffic down if the current network cannot support the traffic going across it.

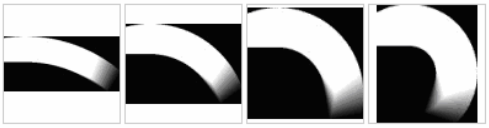
5. 1 Obstacle Avoidance

In order to navigate through constrained spaces we use a slightly modified version of the dynamic window approach [6]. This method is one of the few obstacle avoidance algorithms that can handle vehicles with complex shapes and severe dynamic constraints.

The idea behind this approach is to find the set of admissible velocities by first considering the velocities that are physically allowed by the chassis and dismissing velocities that do not meet other dynamic constraints. Next out of this reduced set of velocities, the algorithm picks the velocity that optimizes a pre-specified cost function.

All trajectories of the system can be represented by the curvature $c = w/v$ with the special case of pure rotation when $v = 0$. The set of all admissible velocities are described by $V_r = V_d \cap V_a \cap V_c \cap V_o$. A table of these constraints along with the meaning of each set is given in Table 2. Any commanded velocity outside of this set is not considered to be an acceptable command.

Symbol	Constraint Type	Constraint
V_d	differential chassis	$\{(v,w) \mid \lvert v\rvert + r\lvert w\rvert \le v_{\max}\}$
V_a	maximum acceleration	$\left\{\begin{matrix} v \in [v_c - a_v\Delta t, v_c + a_v\Delta t] \\ w \in [w_c - a_w\Delta t, w_c + a_w\Delta t] \end{matrix}\right\}$
V_c	comfort constraint	$\lVert \dot{v} + \dot{w} \times r_c + w \times (w \times r_c) \rVert \le K_c$
V_o	obstacle collision	$\lvert v\rvert \le vel_v(o)$ $\lvert w\rvert \le vel_w(o)$



Here are four sample trajectories from our lookup table. Each of these trajectory maps undergo a bitwise AND operation with the local occupancy grid. After this operation we can solve for the distance to the nearest obstacle along a given trajectory.

Table 2: Obstacle Avoidance Trajectories and Constraints [7]

First we update the occupancy grid map by alpha blending the previous occupancy grid map with a set of triangular polygons that represent the current sonar range measurements. Careful consideration of pixel alignment must be taken into account to keep the grid cells from bleeding into neighboring cells due to round off errors. Next we obtain the robot-centric occupancy grid map by drawing a rotated version of the occupancy grid map as a texture into a framebuffer that represents a smaller, local map. The pixels from this map are copied from GPU memory to CPU memory and put into an array [8]. This binary array tells whether a grid cell is occupied in local space or not. Finally we obtain the distance to the nearest obstacle along each possible trajectory specified in our lookup table. To do this the local obstacle map is multiplied element-wise with each of the trajectory maps (see Table 2) and the maximum value of the result is taken. This value corresponds to the distance to the nearest obstacle along a trajectory and can be used to limit the possible vehicle velocities using the dynamic window method [9] [10].

5.2 Vision

The vision algorithm uses an open source computer vision library developed by Intel called OpenCV. 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 in Figure 8 is a distorted grass/concrete sample image which displays the final Hough green line and binary image output of the brightness threshold.

Figure 8: Vision Processing Image on road

We also tested in a all concrete driving scenarios to prove the algorithm will be robust for a wide variety of terrain. Notice how the grass line is much easier to distinguish than the highway line. As a stable testing platform we implemented “OpenView Vision”, a Labview-OpenCV testing environment that quickly test and modify parameters

on the fly [11]. The debug console is shown in Figure 9.

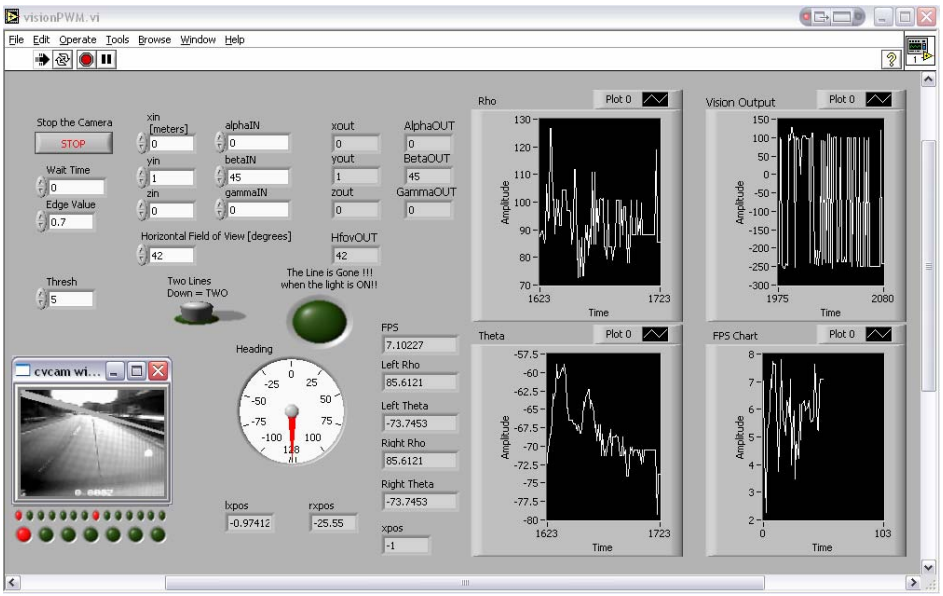


Figure 9: Vision Base station Console

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 robot 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 in Figure 10.

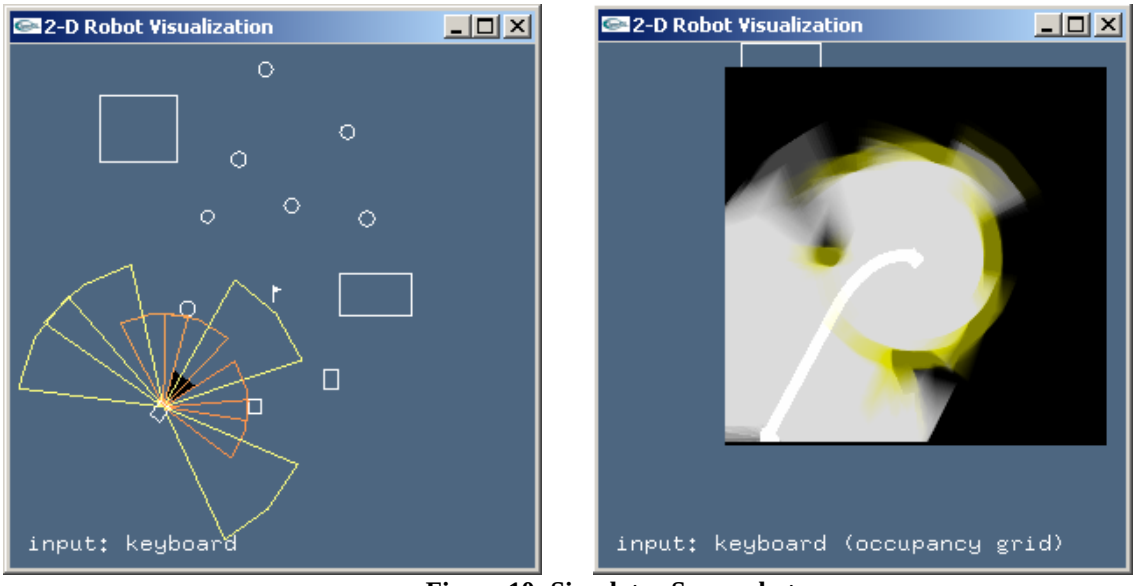


Figure 10: Simulator Screenshots

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 developed a test stand by mounting a sonar on a servo and wired a microcontroller to send control signals to the servo, pulse input to the sonar, and receive the returned data from the sonar. This test stand was placed in an open field in front of a cylindrical object; a plastic trash can, to resemble a construction barrel. Data was collected every one to two 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 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 the 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 communicated 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 Figure 12.

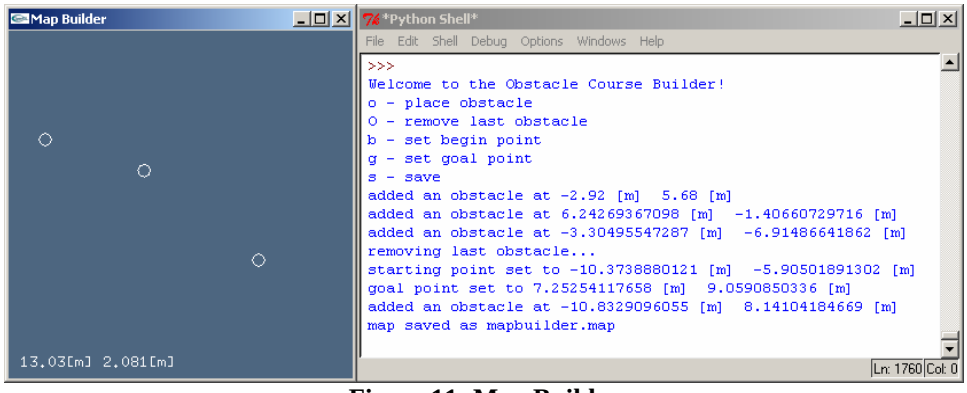


Figure 11: Map Builder

6.0 PREDICTIONS

Our low-cost drive train and configurable electronics will provide a reasonable opportunity for success at the 2007 competition. The motors and gear box in this drive train allow a maximum speed of around 5 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.

Our camera can see objects in excess of 20 feet depending on the ambient brightness and the contrast, but our sonars have a maximum range of 10 feet. Our GPS has a best accuracy of 2 meters, but in practice on a clear day, far from buildings, we achieve an accuracy of 3-5 meters. Our waypoint navigation is bounded by our GPS navigation.

7.0 SAFETY

Throughout the different robot vehicles we have built, small or large, flying or ground, there is always a safety plan built into the design. BlastyRAS is 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 3. 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	IFI Robotics Kit Parts & Banebot Gearboxes	$450	$200
1	Compaq Evo laptop	$1,299	$0
1	Motorola 68HC9S12 Microcontroller	$50	$0
1	Microstrain 3DM-G IMU	$1,495	$0
1	Logitech QuickCam Orbit	$90	$90
1	Garmin GPS 72 Unit	$110	$110
8	Ultrasonic SR04 Rangefinder	$200	$0
2	GrayHill 61R Encoders	$75	$75
2	IFI Victor 884 Speed Controller	$350	$350
1	Custom PCB for Sonar and Microcontroller	$33	$33
Total		$4,152	$858

Table 3: System Level Budget

9.0 CONCLUSION

BlastyRAS is a culmination of effort of the RAS team from the University of Texas at Austin. As a second year team into the competition, our contributions software infrastructure and low-cost electronic design are quite portable. However, BlastyRAS 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 Python-OpenCV Vision Engine and configurable robotic chassis will be a defining aspect of our presence at the IGVC competition.

10.0 REFERENCES

[1] Garmin 72 GPS datasheet , https://buy.garmin.com/shop/shop.do?pID=214

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

[3] Grayhill Encoders http://lgrws01.grayhill.com/web/images/ProductImages/I-37-41.pdf

[4] Logitech QuickCam Orbit Spec Sheet, http://www.logitech.com/index.cfm/products/details/US/EN,CRID=2204,CONTENTID=10628 [5] Acroname Sonar SR04 spec Sheet, http://acroname.com/robotics/parts/R93-SRF04.html

[6] D. Fox, W. Burgard, S. Thrun, "The Dynamic Window Approach to Collision Avoidance" IEEE Robotics & Automation Magazine, 4(1), March 1997.

[7] C. Schlegel, "Fast Local Obstacle Avoidance under Kinematic and Dynamic Constraints for a Mobile Robot" Proceedings of the 1998 IEEE-RSJ Intl. Conference on Intelligent Robots and Systems Victoria, B.C., Canada, October 1998.

[8] M. Yguel, O. Aycard, C. Laugier, "Efficient GPU-based Construction of Occupancy Grids Using several Laser Range-finders" Proceedings of the IEEE-RSJ Intl. Conference on Intelligent Robots and Systems, 2006.

[9] S. LaValle, *Planning Algorithms*, Cambridge University Press, 2006

[10] S. Thrun, W. Burgard, D. Fox, *Probabilistic Robotics*, 2005.

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

Appendix A: SCHEMATICS

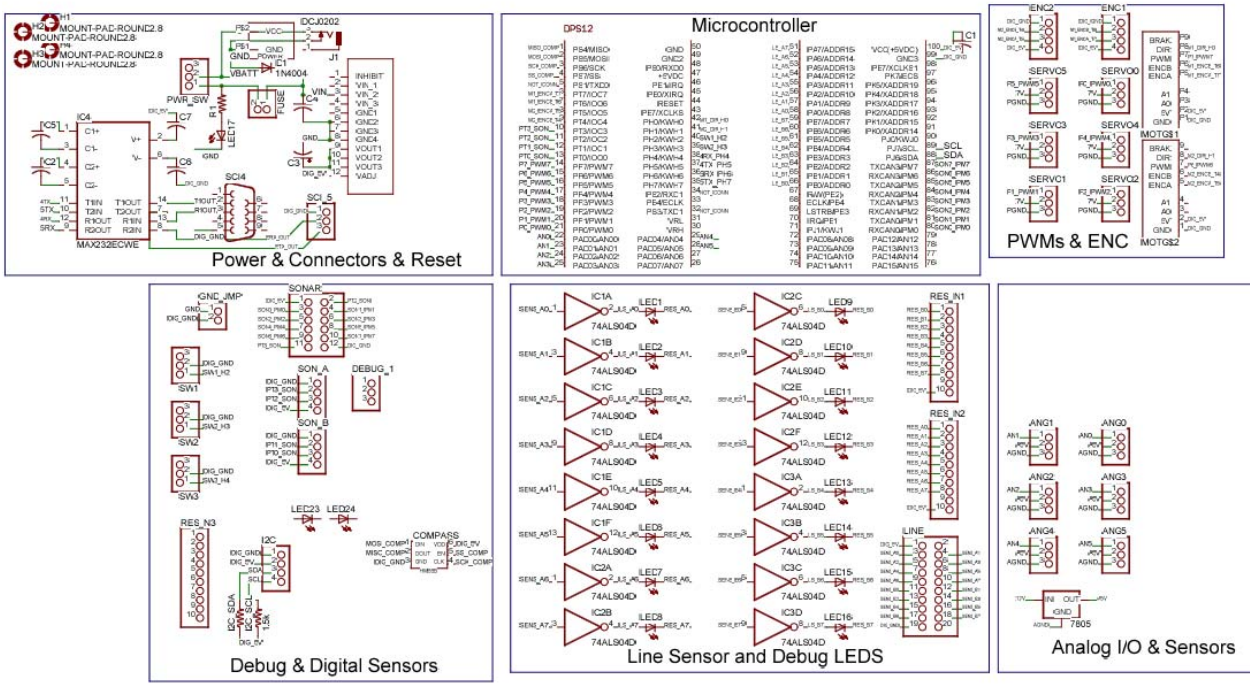


Figure 12: Microcontroller Schematic

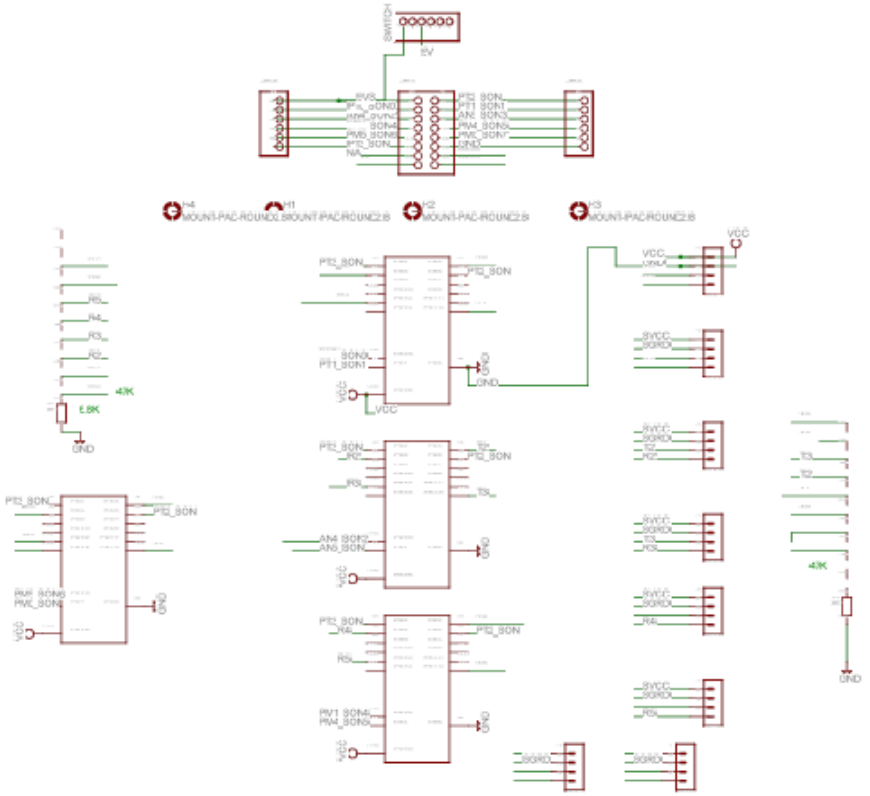


Figure 13: Sonar Controller Schematic

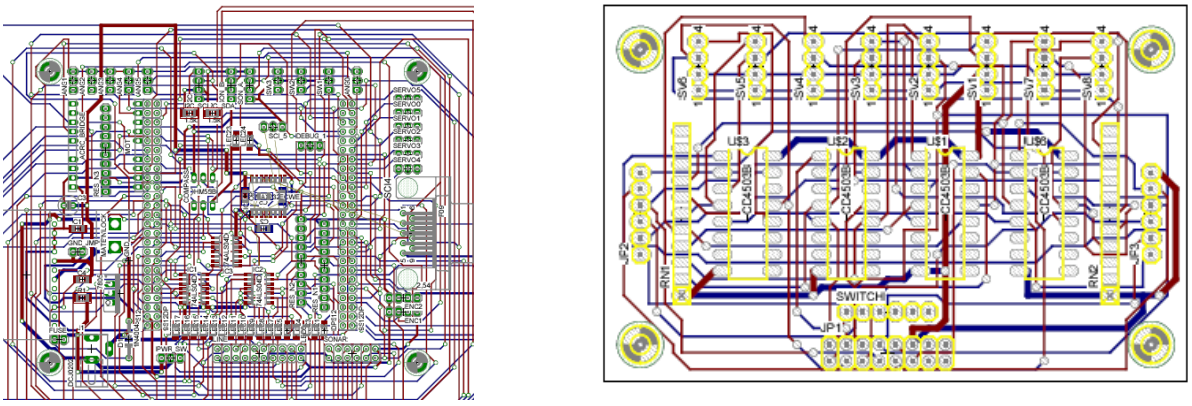


Figure 14: Layout of Sonar (right) and Microcontroller Circuit (left)