



PRESENTS:

EnterpRAS

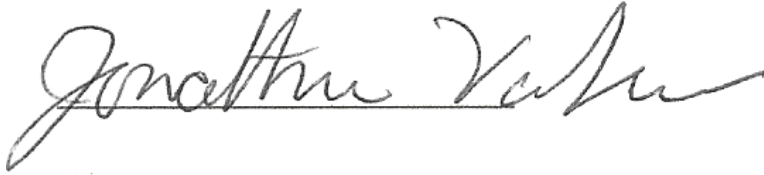


Nicu Știurcă, Josh James, Robby Nevels, Frank Weng, Alan Kwok

Faculty Advisor Statement

I certify that the engineering design of the new vehicle, *EnterpRAS*, 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,

A handwritten signature in black ink, reading "Jonathan Valvano". The signature is written in a cursive style with a horizontal line underneath the name.

Jonathan Valvano

Abstract

This report concerns the production of the *EnterpRAS*, an entry to the 19th annual Intelligent Ground Vehicle Competition. When producing this autonomous vehicle, the authors drew upon past designs, and researched new technology, working as a team to create an innovative robot to perform the numerous challenges of the Competition. The report describes project methods, construction of the robot, electrical system design, sensing algorithms, software architecture, system integration, and safety, reliability and durability concerns. Because of a tight budget, the report emphasizes cost-effective solutions to challenging problem.

1. Introduction

Space: the final frontier. These are the continuing voyages of the starship Enterprise. Her ongoing mission: to explore strange new worlds, to seek out new life forms and new civilizations, to boldly go where no man has gone before.

So said the highly intelligent character Spock from the Star Trek universe [6]. Although he refers to the voyages of a fictional starship, the mission he describes can be applicable to any engineering endeavor. By finding out new, innovative ways to build an autonomous vehicle, map out and explore unknown landscapes, and design electrical and mechanical systems, we believe this project reflects Spock's inspiring words. We named our robot the *EnterpRAS* as tribute to this inspiration as well as our organization's name: The University of Texas IEEE Robotics and Automation Society (RAS).

The purpose of this report is to detail the work that the University of Texas IEEE Robotics and Automation Society team has done to build the *EnterpRAS* from the ground up as an entry to the 19th annual Intelligent Ground Vehicle Competition (IGVC). The rest of the report is structured as follows. Section 2 provides a high level project description. Section 3 explains the UT IEEE RAS team's design process. Section 4 describes the materials and construction of the *EnterpRAS*. Section 5 describes the electrical system. Section 6 describes the software architecture and systems integration, including a microcontroller board designed in-house to control actuators and read rotation sensors. Section 7 details the *EnterpRAS*'s sensing algorithms, including mapping and vision. Section 8 addresses safety, reliability, and durability concerns. Section 9 concludes with an outline of outstanding work to be completed before the competition and with predictions for the competition.

2. Project Description

This section gives a brief overview of the produced robot as well as the motivation behind the project.

2.1 Product Summary

The *EnterpRAS* uses an approximate Ackermann steering drive system implemented in software. It receives input from a variety of sensors, including a digital compass, a planar laser rangefinder, a USB webcam, a GPS receiver, encoders and potentiometers. The wheel encoders provide odometry. The digital compass provides pitch, roll, and acceleration readings in addition to heading. Combined with the GPS receiver, these sensors are used to localize the robot in the world. The USB webcam feeds images to the vision algorithms. The laser rangefinder is mounted onto a tilting servo motor in order to provide 3D data.

We built a custom computer to match our computational and power needs of the autonomous robot. This

computer interfaces with a microcontroller to control the motors and to obtain data from analog sensors as well as the encoders on each drive motor. The power distribution system of the EnerpRAS allows hot-swapping between powering off a wall socket or two 12 volt lead-acid batteries without being turned off. Fresh batteries can also be hot-swapped for drained batteries.

For an exact list of parts, consult Appendix A.

2.2 Motivation

IGVC presents a unique combination of computational, electrical, and mechanical challenges. These include developing algorithms to process and reason through large amounts of streaming sensor data in real time, creating an effective power distribution system to provide the computer, microcontroller, motors, and sensors with adequate power, and creating a mechanical frame to mount all components. All the while, the design must meet safety, reliability, and durability requirements. Each of these challenges will be addressed in forthcoming sections.

3. Design Process

This section describes how the UT IEEE RAS team is organized and how the team proceeds from initial concept to a final design.

3.1 Team Organization

The UT IEEE RAS team started this year's project with a core group of 5 undergraduate students. The team self-organized as an entirely student-run project with little input from any University faculty. Each team member's individual tasks would pertain mostly to one or two subsystem. Starting in the beginning of the Spring 2011 semester, we discussed the necessities of the competition in order to develop a product concept and figure out the requirements of the various subsystems. Several basic components of the physical part of this project which we needed to begin work on: the drive train, the electrical layout, and the computer. The group conducted one or two group meetings every week for the duration of the semester, and individual members also worked on their respective tasks during their free time.

3.2 Continuous (Re)Design and (Re)Evaluation

Our design sequence begins with high level block diagram of sensors, electronics, software and mechanical ideas. Many design decisions involved choosing commercially-off-the-shelf (COTS) hardware to achieve a layer of abstraction from low level devices. Due to a tight monetary budget, cost effectiveness has been a major concern throughout the design process. We often chose to reuse components from previous UT IEEE RAS projects even if newer and improved versions are available. 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 necessary.

Our approach differs from more traditional planning methods which requires more simulation and analysis of each decision and less re-evaluation of past decisions. Our organizational structure and time budgets required us to accelerate the analysis phase. To demonstrate proof-of-concept, we create a series of rapid prototypes to eventually achieve a functional design to utilize in our final system. Our quick design strategy allows us to have a working robot throughout many design stages. A functional prototype improved our time in the overall system evaluation.

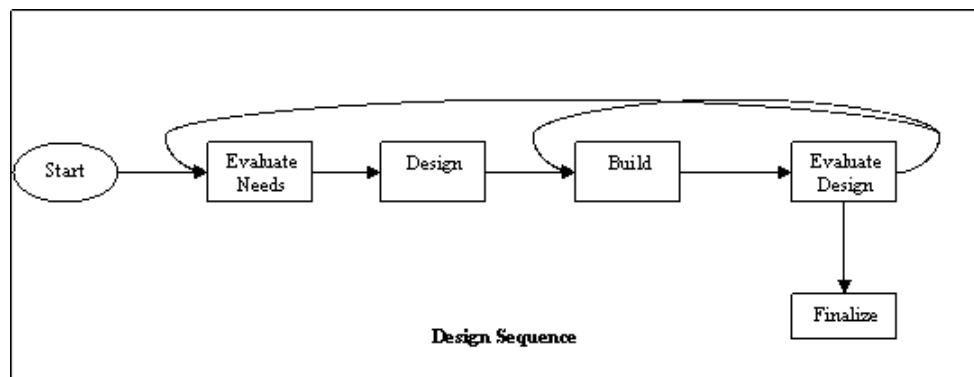


Figure 1: Process Flowchart

An example of this constant reevaluation is presented in Section 4.3.

4. Material Structure

This section describes the mechanical characteristics and construction of the frame and drive systems.

4.1 Frame

The frame is constructed of 1/16" thick 1" aluminum square tubing. The outside perimeter is 2 feet by 3 feet, exactly the smallest allowed for this competition. A sensor mast constructed of 1/16" thick 1" aluminum angle supports all of our various sensors as well as a servo-motor which changes the pitch of the lidar.

4.2 Drive Wheels

The rear wheels are 13.5" diameter with bearings on a stationary 5/8" steel shaft and a #35ANSI chain sprocket attached to the wheels by a custom aluminum hub. The DC motors which drive these by a chain are mounted separately.

4.3 Ackermann Steering

The front steering system is an Ackermann steering¹ design controlled by a high torque motor. [8] Though not a perfect implementation of Ackermann steering², our design ensures that the wheels are pointing in the desired direction and turn the autonomous vehicle with the minimum slipping. As an Ackermann design, instead of both set of front wheels turning around a common pivot, each wheel has its own axis of rotation close to the hub. This whole systems of rotation motion is controlled by a linkage between thee two hubs; the linkage is in turn driven by a bar directly connected to the high-torque motor. Furthermore, a potentiometer is attached to the axis of motor rotation to ensure accurate turning. Our design is efficient and accurate; however, we consistently ran into problems with structural/material strength. Since our Ackermann design is essentially driven by a bar mounted directly on the high-torque motor, that point of contact experience enormous bending stressing. In the earlier version of the vehicle design, aluminum bars experienced multiple yielding and structure failure. We have since then solved the problem with reinforcing the system with redundancy, and thicker/better placement of linkage/connection.

Due to budget constrains, we had to be sparse in our material usage / equipment purchase. Therefore, we often had to come up with innovative ways to manufacture the frames and steering system out of scrap materials we previously owned.

5. Electronic Design

5.1 Computer Design

We built a custom computer purposed to handle the computational and performance challenges of real-time robotics algorithms as well as Gazebo, a high-fidelity 3D robot simulator. We considered monetary, power, and physical space budget restrictions in choosing the components of our computer. The computer features a quad-core Intel i5 CPU, Fermi CUDA co-processor, 4GB of RAM and a 30 GB solid state drive³. It draws less than 150W under load (less than 25W when idle) and has a small physical footprint. This is advantageous on a battery-powered mobile robot. The total cost is less than \$550. We successfully met all of our various budget considerations without sacrificing computational power.

5.2 Power Distribution

A diagram of our Power Distribution System (PDS) can be found in Appendix C. The function of a PDS is simple: to ensure that all components receive the electricity they need to operate at all times. Our PDS accomplishes this both efficiently and reliably.

5.2.1 Batteries and Voltages

¹E.g., car-like steering.

²Most commercial cars operate perfectly fine with approximate Ackermann steering.

³For exact part descriptions, consult Appendix A.

The PDS must provide 24V to our motors, and 12V to the computer and other sensors. It is possible to accomplish this with a single 24V battery and a voltage regulator. Voltage regulation, however, can cause inefficiencies and power loss. Our PDS eliminates this inefficiency by instead using two 12V batteries in series. Our PDS operates off Pd-Acid batteries since they are rugged, cost-effective, and can supply the high current demands of the *EnterpRAS*'s motors.

5.2.2 Efficiency and Battery Life

The computer was designed with efficiency and battery life in mind. It uses a DC-DC power supply. The use of such a power supply instead of a traditional AC power supply eliminates the need for inverter (the addition of which would introduce large inefficiencies). This allows the *EnterpRAS* to reach power supply efficiencies of up to 96% for the computer. Furthermore, the computer peaks at under 150W under load and draws less than 25W when idle. If the motors are not in use, we expect a fully charged 17Ah Pd-Acid battery to last 3+ hours since we expect the system load to be low on average. Depending on the amount and speed of driving, we expect to get 1-2 hours of battery life.

5.2.3 Hot-Swap

The PDS was also designed with the ability to hot-swap in mind. The PDS board has four input ports for batteries to allow for either of the two batteries to be swapped out without the robot losing power. Additionally, our PDS has the ability to switch the 12V rail to wall power during periods of time in which the robot is not field testing. The switch is made without losing power to the computer or the sensors. The ability to hot-swap between various power sources allows the PDS to supply power reliably at all times. Our PDS also protects the devices it is attached to with fuses.

6. Software Architecture and Integration

The *EnterpRAS* runs Willow Garage's Robot OS (ROS) Diamondback version on top of Ubuntu 10.10 (Maverick). [5] A brief summary of ROS follows in Section 6.1, and the remaining subsections describe how various features of ROS are being used.

6.1 ROS Overview

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more.

The *EnterpRAS*'s software is organized as a collection of ROS nodes. Generally, each ROS node is responsible for a single piece of hardware or a single algorithm. ROS nodes communicate with each other primarily through ROS topics, unidirectional many-to-many data streams. ROS nodes may also invoke

ROS services offered by other nodes. ROS services are, in essence, remote procedure calls. We are using existing ROS drivers for most of our sensors, with the exceptions being the digital compass and our custom microcontroller.

6.2 Mircocontroller/Computer Interface

In order to communicate between the computer and various hardware-based peripherals such as the motor controllers, encoders, potentiometers, and servos we have decided to interface with a microcontroller through a USB port.

6.2.1 RASBoard

The *EnterpRAS* features the RASBoard, a custom microcontroller board to handle digital I/O with motors and encoders and analog input from potentiometers. The board was initially designed by a member of RAS in Fall of 2008. The board has undergone several upgrades and revisions, the latest of which happened in Fall of 2010. It features a Texas Instruments Stellaris 3S811 ARM Cortex M3 microcontroller operating at 50Mhz. The board exposes two dedicated motor ports, two dedicated encoder ports, four dedicated servo-motor ports, a dozen general purpose digital I/O, and four ADC input ports.

The microcontroller is loaded with custom firmware which takes velocity commands from the computer and controls the motors and servos accordingly. It also interprets raw wheel encoder signals and provides odometry to the computer. Implementing this system in software consists of two major parts: first, custom firmware running on the microcontroller, and a corresponding ROS node running on the computer sending and receiving data through a USB port.

6.3 Data Flow

A diagram depicting the data flow to and from hardware devices is available in Appendix D.

6.4 Logging

ROS provides excellent logging facilities. The UT IEEE RAS team uses the rosbag utility to collect sensor logs of raw sensor data and any other messages being passed between nodes. The logs can then be replayed by the rosbag utility as needed in order to aid development or debugging. The replayed logs can be analyzed directly as appropriate (eg, graphing estimated robot pose over time) or fed into higher-level nodes. Nodes do not differentiate between incoming logged data vs data coming directly from sensors.

6.5 Simulation

ROS also integrates well with Gazebo, a high fidelity 3D simulator. [1] The simulator uses Open Dynamics Engine, a rigid body physics simulator. [7] Gazebo uses a plugin architecture to tie in simulated sensors that

produce ROS messages the same way a real sensor would. Higher level nodes cannot distinguish whether the data is coming from real or simulated sensors. All sensors on the real *EnterpRAS* have pre-existing plugins corresponding to each type of sensor. UT IEEE RAS wrote a special plugin which controls a simulated Ackermann drive vehicle since ROS does not yet have a Gazebo controller plugin matching the *EnterpRAS*'s drive system. UT IEEE RAS plans to release this plugin to the ROS community after the competition since there appears to be some demand for such a plugin. To summarize, Gazebo integrated with ROS allows the UT IEEE RAS team to cheaply (in terms of both time and money) set up arbitrarily complex and repeatable testing scenarios.

7. Sensing and Navigation

The *EnterpRAS* project focuses on achieving sophisticated perception through cost-effective sensors despite their inherent sacrifices in sensing quality. Once such perception is achieved, navigation can be achieved without a need for novel algorithms.

7.1 Localization

The robot's position is estimated using a Kalman filter integrating estimates from wheel odometry, IMU sensor, and GPS sensor. Since the GPS sensor provides absolute position, the *EnterpRAS* mapping algorithms do not need to worry about localization drift over time so a computationally expensive SLAM algorithm is not required. The filter is implemented by the `robot_pose_ekf` ROS node available in the ROS navigation stack. [5]

7.2 Mapping

Autonomous ground vehicles traditionally rely on a two-dimensional perception and representation of the world. Although this approach can simplify some aspects of robotic mapping and reduce computational requirements, it is prone to losing important information, leading to incorrect behavior. For example, ground vehicles that rely on traditional planar rangefinders would detect a ramp as a wall, thus compromising liveness⁴ of the robot since it would be unable to proceed up the ramp out of fear of hitting a wall. Worse yet, such a robot might detect the supporting legs of a barrier but not the crossbeam, thus compromising safety. The *EnterpRAS* employs a richer representation to allow correct operation at IGVC.

7.2.1 Approach

In our approach, we deem it critical that the *EnterpRAS* can correctly and effectively deal with a three-dimensional world. In order to do so, we map the world in three-dimensions, represented as a voxel grid. Our approach is derived from Marder-Eppstein [4], but we extend it to deal with semi-permeable obstacles (such as fences) and to utilize the full scanning range of the lidar. We extend the capabilities of a planar laser rangefinder

⁴We borrow the terms liveness and safety from systems literature. Informally, liveness properties of a system guarantee that the system eventually does something good; safety properties guarantee that the system never does anything bad.

by mounting it on a tilting servo-motor, thus generating a 3D point cloud. The point cloud is used to build a voxel grid representation of the world. The raw lidar data along with the tilt angle of each scan and the robot's position in the world is passed to a mapping algorithm which updates the voxel grid. Each scanned point in the cloud is ray-traced through all blocks in the voxel grid from the robot's position to the detected end point of the ray. The traced blocks are updated according to Algorithm 1.

7.2.2 Implementation Considerations

In this algorithm, each block is in one of four states, so only two bits of storage are needed for each block. Thus, a column of 32 blocks can be efficiently represented as a 64 bit integer. Bitwise AND and OR operations can be used to efficiently update blocks. 32 blocks can represent a column of space 1.6 meters high (the height of the *EnterpRAS*) with a vertical resolution of 5 cm. In the interest of simplicity, the forward and lateral resolutions have also been defined as 5 cm. Since the update operations on the set of blocks traced by a ray are independent of each other, they can be performed in parallel. The *EnterpRAS* exploits this by implementing the algorithm on the CUDA co-processor in order to offload a significant computational burden from the CPU.

```

foreach ray r in scan
  B = list of blocks intersected by r
  # update the blocks up to but excluding
  # the last block
  foreach block b in B[1..n-1]
    if b = UNKNOWN or b = OCCUPIED
      b := SEMI_FREE
    else if b = SEMI_FREE
      b := FREE
    else if b = FREE
      no-op
  # update the last block
  if B[n] = UNKNOWN or B[n] = FREE
    B[n] := SEMI_FREE
  else if B[n] = SEMI_FREE
    B[n] := OCCUPIED
  else if B[n] = OCCUPIED
    no-op

```

Algorithm 1. Updating Voxel Grid

Some care must be taken to safely deal with the case where one or more rays passes straight through a block therefore clearing it while some other ray's endpoint is in the block therefore marking it occupied. In this case, the block is not entirely FREE and must not be marked as such. This is accomplished in the CUDA multithreading model by introducing a barrier after the inner **foreach** loop which assures that updating the last block as SEMI_FREE or OCCUPIED only occurs after all the updates which might mark it as FREE.

7.3 Vision

The *EnterpRAS* uses a combination of several standard vision processing algorithms to detect the lane boundaries. The *EnterpRAS* vision nodes use OpenCV, an open source, state of the art computer vision library. [2] OpenCV is fully integrated with ROS, and it provides efficient implementations of standard vision algorithms. As an optimization, the newest version of OpenCV available as of this writing also includes support for CUDA-enabled GPUs to parallelize algorithms and speed up processing while freeing up the CPU. The *EnterpRAS* lane detection algorithms described in the following subsections are built on top of OpenCV vision processing primitives.

UT IEEE RAS developed two algorithms for detecting the lane boundaries and obstacles. One is based on

edge detection, and the other on flood filling the region of the image corresponding to the lane. Each algorithm operates on single-channel images, so raw color images are first converted into 320×240^5 grayscale images, or alternatively they can be converted into the hue-saturation-value color space and the algorithms can operate on the hue (color) channel. After the conversion to a single channel image, the image is further preprocessed with a Gaussian smoothing filter, and it undergoes an inverse perspective warp to transform the the image from a camera-centric coordiante frame to a robot-centric coordinate frame⁶.

7.3.1 Edge Detection

Performing edge detection with OpenCV is simple. The 2011 algorithm for edge detection is based on the team's 2010 lane detection code. The preprocessed image is processed with a Canny edge detector, as implemented in OpenCV. Since edges present in the processed image will correspond to either the painted, white lane boundaries or to the edges of obstacles, the processed image is interpreted as an occupancy grid where pixels marked as edges correspond to occupied blocks in the grid.

7.3.2 Flood Fill

Performing flood fill operations on an image with OpenCV is, again, simple. The flood fill algorithm is seeded with a pixel corresponding to a point on the lane just in front of the robot. By assuming that point is part of the lane, the algorithm marks all the points that correspond to the lane: given some pixels that are already known to correspond to the lane, all of their neighbors are checked; if a neighboring pixel being checked is within a small floating threshold of a neighboring pixel known to correspond to the lane, the new pixel is marked as part of the lane. This process identifies all the pixels that correspond to the lane since pixels that correspond to the lane boundary lines or obstacles would differ from pixels in the lane by a value greater than the threshold. The processed image is again interpreted as an occupancy grid where pixels that were marked as belonging to the lane are considered free.

7.3.3 Combining Estimates of Lane Boundaries

The two variations on preprocessing raw images (grayscale or hue channel) times two lane detection algorithms yields four estimators of the lane boundaries. The *EnterpRAS* must somehow determine which is the best estimate of the lane given each camera frame, and the best estimate of the local occupancy grid computed from each camera frame is incorporated into a global occupancy grid.

In practice, each estimator usually yields acceptably few false positives or false negatives in its computed

⁵Although shrinking an image introduces some information loss, it was necessary in order to achieve good processing performance. The image size was chosen empirically in order to balance efficiency of processing with information loss.

⁶The *EnterpRAS*'s coordinate frame is that commonly used by roboticists: the x axis points forward, the y axis points left, and the z axis points up.

occupancy grid⁷. That is, blocks falsely detected as free usually do not produce a large enough gap for the robot to fit through. Likewise, blocks falsely detected as occupied can be usually planned around in a way that does not hinder progress, although the robot may have to slow down or adjust its trajectory. It is expected that successive occupancy grids should be very similar when an estimator is producing good estimates since it will be detecting (roughly) all the obstacles in needs to detect. When an estimator suddenly produces occupancy grids dissimilar to what it produced previously, it is an indication that the estimator suddenly started producing too many false positives, too many false negatives, or both. In this case, confidence in the quality of the estimator's output drops and a different estimator which appears to be operating correctly (or at least less erratically) should be chosen.

By quantifying the similarities of successive occupancy grids yielding from each estimator, the *EnterpRAS* can decide on the best estimate at each time step and incorporate the estimate's local occupancy grid into the global map. To pick the estimator with the highest similarity between its last and penultimate occupancy grids, the *EnterpRAS* picks the estimator with the smallest difference between the successive occupancy grids. Since occupancy grid cell states are either occupied or not, Hamming distance is used. In order to account for the fact that one of the estimators may naturally detect more obstacles, the difference is normalized (i.e., divided) by the number of cells which are occupied in both successive grids. Care must be taken to account for the motion of the robot in between the two frames. Each frame must be transformed from the local reference frame at the time the camera frame was captured to a global reference frame. ROS has a transform engine which can easily facilitate this transformation. When the two occupancy grids are compared in the global reference frame, some parts of each grid are not visible in the other if the robot moved. By design, this is the usual case, and care is taken to ignore these parts when calculating the relative difference between the occupancy grids.

7.4 Navigation

The *EnterpRAS* uses the ROS navigation stack for path planning. The ROS navigation stack takes as input a 2D map represented as an occupancy grid and a destination, and it outputs the velocity (translational and angular) that the robot should travel at each time step. In order to feed the navigation stack the expected input, the voxel grid is flattened to a 2D occupancy grid and the occupancy grid from the lane detector is superimposed. When flattening the voxel grid, the height of the highest OCCUPIED or SEMI_FREE block is given as the height of the occupancy grid. If no OCCUPIED or SEMI_FREE blocks are present, the height is 0. If a column with no OCCUPIED or SEMI_FREE blocks has more than 5 UNKNOWN blocks, the height is -1. The height map is then used to compute traversability. All cells with height 0 are traversible. Cells are also traversible if it is adjacent to a traversible cell whos height differs by at most 1. This marks ramps with a slope of less than 45 degrees as traversible. The goal given to the navigation stack is the next GPS waypoint the robot is trying to

⁷This has only been observed qualitatively since it would be too time consuming to manually identify false negatives and positives in order to quantify this observation.

reach.

8. Safety / Reliability / Durability

EnterpRAS has several features specifically designed to increase its safety. A warning strobe light will always be on whenever the robot is in autonomous mode. This lets everyone around it know that they may wish to be cautious around it. The voltage to the motors is controlled by two switches in series. The first switch is the large red button. We placed this switch in an easily accessible location so that the robot would be easily disabled in the event of a malfunction in the other systems. The second switch is a wirelessly controlled relay. It can be toggled remotely from the Nintendo Zapper that will be given to the judges. If either switch is toggled while the robot is moving, the robot will stop immediately. We have also added bumpers to all sides of the robot so that if the robot is not stopped in time, the damage from collisions is minimized. In terms of electrical safety, all voltages and connections on the PDS have been labeled. The voltages of each rail have also been color coded to reduce confusion. Additionally, we have taken precautions that no voltage rail is likely to touch the chassis.

9. Conclusion

The UT IEEE RAS team build the *EnterpRAS* to compete in the 19th annual Intelligent Ground Vehicle Competition. The team created an innovative robot emphasizing simplicity and cost-effectiveness in order to deal with the numerous and complex challenges of the IGVC competition.

References

- [1] Anonymous. (November 2, 2006). Gazebo. In The Player Project. Retrieved May 6, 2011, from <http://playerstage.sourceforge.net/index.php?src=gazebo>.
- [2] Bradski, Gary. (April 1, 2011). OpenCV. In OpenCVWiki. Retrieved May 6, 2011, from <http://opencv.willowgarage.com/wiki/>.
- [3] Conley, Ken. (March 16, 2011). ROS Documentation. In ROS.org. Retrieved May 6, 2011, from <http://www.ros.org/wiki/>.
- [4] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, Kurt Konolige. (September 27, 2010). PapersICRA2010_Marder-Eppstein. In ROS.org. Retrieved May 6, 2011, from http://www.ros.org/wiki/Papers/ICRA2010_Marder-Eppstein.
- [5] Meeussen, Wim. (March 4, 2011). robot_pose_ekf. In ROS.org. Retrieved May 6, 2011, from http://www.ros.org/wiki/robot_pose_ekf.
- [6] Meyer, Nicholas. (Director). (June 4, 1982). Star Trek II: The Wrath of Khan [Film]. Retrieved May 6, 2011, from <http://www.imdb.com/title/tt0084726/quotes>.
- [7] Smith, Russell. (May 28, 2007). Open Dynamics Engine. In Open Dynamics Engine. Retrieved May 6, 2011, from <http://www.ode.org/>.
- [8] Wikipedia: The free encyclopedia.. (n.d.). Ackermann steering geometry. In Wikipedia. Retrieved May 6, 2011, from http://en.wikipedia.org/wiki/Ackermann_steering_geometry.

Appendixes

Appendix A: Bill of Material

Part	Quantity	Retail unit price	Total cost to us
Intel Core i5-760 CPU	1	\$209.99	\$209.99
Gigabyte H55M-S2V motherboard	1	\$69.99	\$69.99
G.SKILL 4GB DDR3 RAM	1	\$40.99	\$40.99
Zotac GeForce GT 430 GPU	1	\$79.99	\$59.99
Kingston 30GB SSD	1	\$85.99	\$55.99
M4-ATX DC-DC 250W PSU	1	\$89.50	\$89.50
Construction Material (Aluminum and Steel)	1	\$250.00	\$150.00
ShaYang Ye DC Geared Motor	3	\$200.00	\$0.00
Wheels	4	\$10.00	\$0.00
Chain and Sprocket	2	\$20.00	\$5.00
Lead Acid Battery	6	\$60.00	\$0.00
Victor Motor Controller	3	\$80.00	\$0.00
RASBoard	1	\$50.00*	\$0.00

OceanServer 5000-S Digital Compass	1	\$270.00	\$0.00
Logitech QuickCam Pro 9000	1	\$90.00	\$0.00
Ublox AEK-4H GPS Evaluation Kit	1	N/A	\$0.00
Hokuyo UHG-08LX Laser Rangefinder	1	\$3950.00	\$0.00
Total:		\$5536.45	\$681.45

Notes:

* This is the estimated cost of the raw parts of the board. The item is made in-house and is not available for retail.

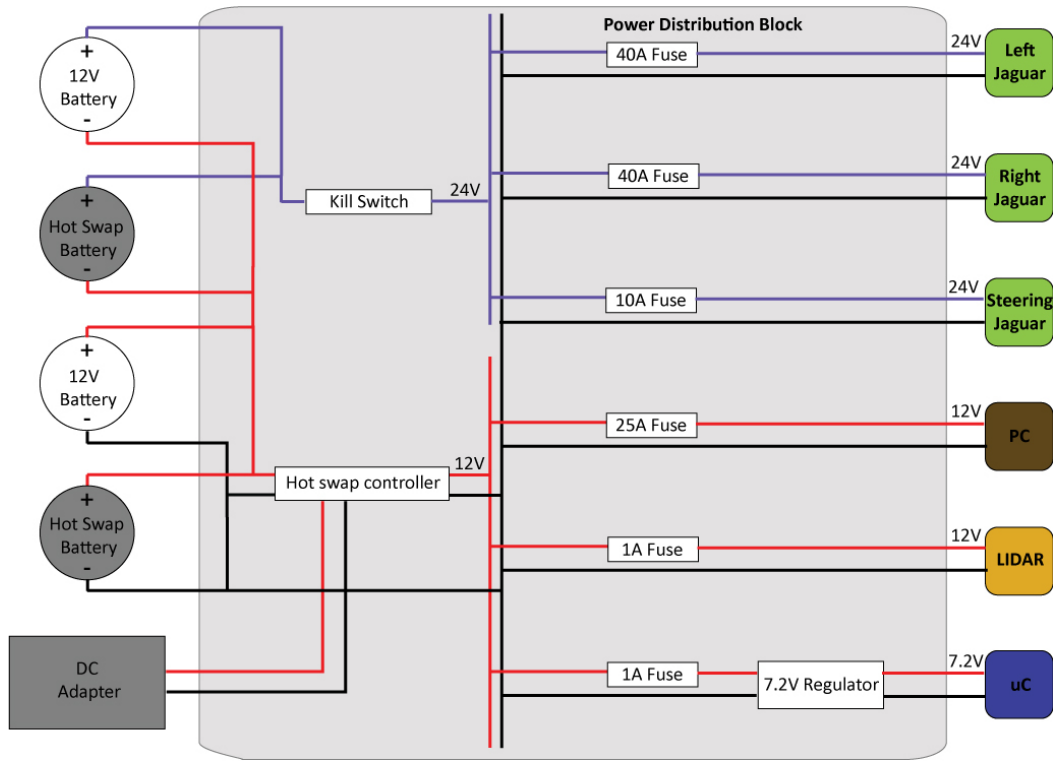
Items marked with a cost to us of \$0.00 are items that have been reused from old UT IEEE RAS projects.

Items marked with a cost to us lower than retail value are items that we purchased at a discounted price.

Appendix B: Work Division Breakdown

Student	Major	Year	Focus of Contribution	Approximate Work Completed
Nicu Ştiurcă	Computer Science	4th	Team Lead; Software	150
Josh James	Electrical Engineering	2nd	Electrical	125
Frank Weng	Electrical Engineering	2nd	Mechanical	100
Robby Nevels	Electrical Engineering/ Computer Science	3rd	Software	50
Alan Kwok	Mechanical Engineering	2nd	Mechanical	40
Emily Ledbetter	Electrical Engineering	1st	Electrical	15
Juan Ortiz	Aerospace Engineering	4th	Mechanical	15
Charlie Manion	Mechanical	4th	Mechanical	10

Appendix C: Power Distribution System



Appendix D: Data Flow Chart

