UNIVERSITY OF MINNESOTA

AWESOM-O 2007

15th Annual Intelligent Ground Vehicle Competition

2007 Design Report

Design Report prepared by:

Eddie Arpin Seth Berrier Richard Hoglund

Faculty Advisor:

Max Donath

Faculty Certification

I, Max Donath, Faculty advisor of the 2007 University of Minnesota IGVC team, certify "the engineering design in the vehicle (original or changes) by the current student team has been significant and equivalent to what might be awarded credit in a senior design course."

X_____ Date: _____

1. Introduction

Awesom-O 2007 is the 5th autonomous ground vehicle (AGV) to enter IGVC. The last time the University of Minnesota attended IGVC was 2005, with the G2 AGV. Awesom-O 2007 is almost an entirely new vehicle to IGVC. The software has been completely rewritten from the 2005 team and a new vehicle chassis has been built. There are also new mechanical and electronic components. Basically, the only thing left from G2 is the motors and motion control hardware!

2. Design Process

The design process for AWESOM-O started with brainstorming the necessary design objectives to create a successful AGV for the competition. These design objectives were based primarily on the shortcomings of the 2005 U of M vehicle and of other vehicles from the 2006 IGVC, which was observed by two current team members. Some of the key design objectives were vehicle size, mobility, robust obstacle avoidance, processing power, and low power consumption. The design responsibility was then split between the team members. Throughout the design processes the team kept open communication and advised one another with each iteration of the design.

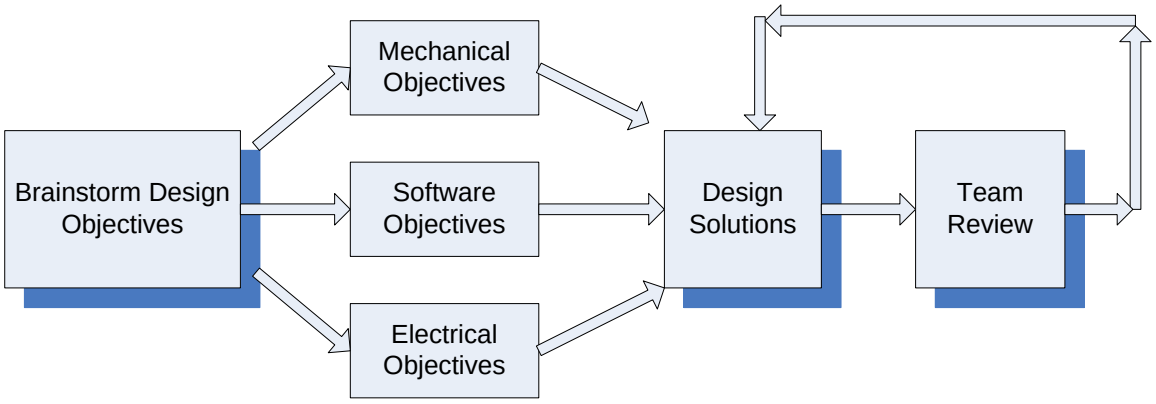

Figure 1 -- Design Process Flowchart

3. The Vehicle

The 2007 vehicle shares the same drivetrain as the 2005 vehicle, but mechanical similarities stop there. AWESOM-O has a completely new vehicle frame, which focuses on a small footprint and increased mobility. The robot maneuvers with two independently driven front wheels and rear dual caster wheels. The rear dual caster wheels improve mobility by reducing the tire forces needed to turn the vehicle. This was a major problem for the G2 and was addressed in the design of AWESOM-O.

3.1 Vehicle Chassis

Pro-E, a CAD software, was used to design the vehicle chassis. Pro-E proved to be a valuable tool because it allowed the team to easily change the design, and allowed 3-D visualization of the vehicle frame.

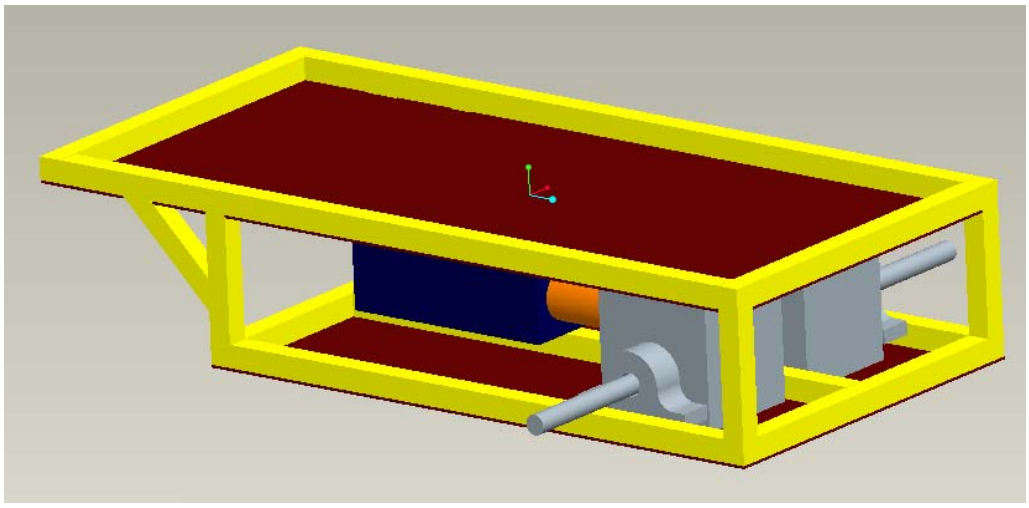

Figure 2 -- Pro-E Model of Vehicle Chassis

It was decided that the vehicle frame be made from 1x1 inch T-slotted aluminum extrusion because of its versatility and ease of construction. $1/8^{th}$ inch aluminum plating was used for the base plate and to support the batteries, computer, electrical box, and SICK LIDAR unit.

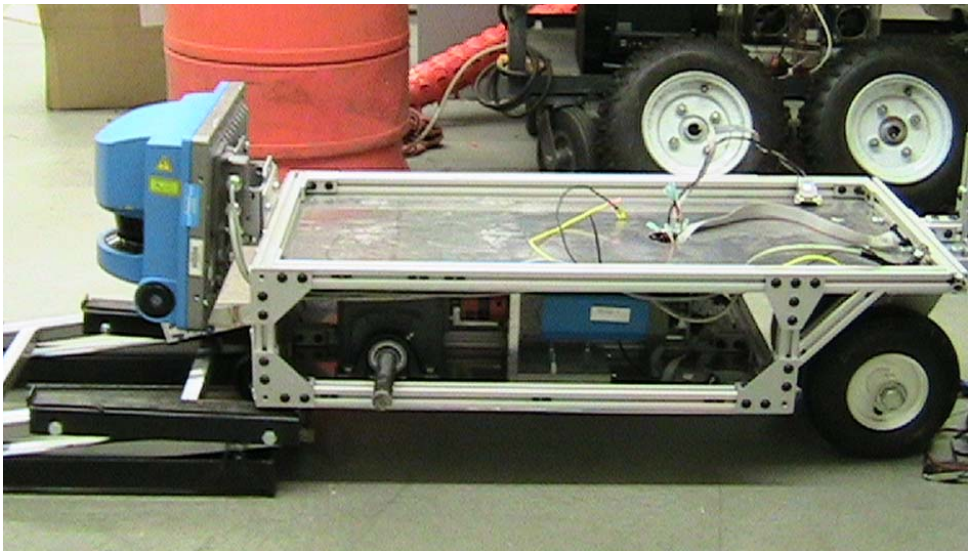

Figure 3 -- Vehicle Chassis

3.2 Drivetrain

AWESOM-O is actuated by two pulse width modulated Kollmorgan servo motors rated at 1.1 horsepower each. The motors are linked to two Boston worm gear boxes via flexible couplings. The two gearboxes have a gear reduction ratio of 20:1. The two drive axels coming out of the gearboxes are run through pillow block bearings to reduce the bending moment in the axle, decreasing the reaction forces on the gearboxes. Two keyed hubs are kept secure on the axle by two shaft collars on each side of the hub. Two 16 inch tubeless snow blower tires are mounted to the hub.

3.3 Electronics

The electronics on AWESOM-O can be split into three main categories: power, motion control, computer, and sensors.

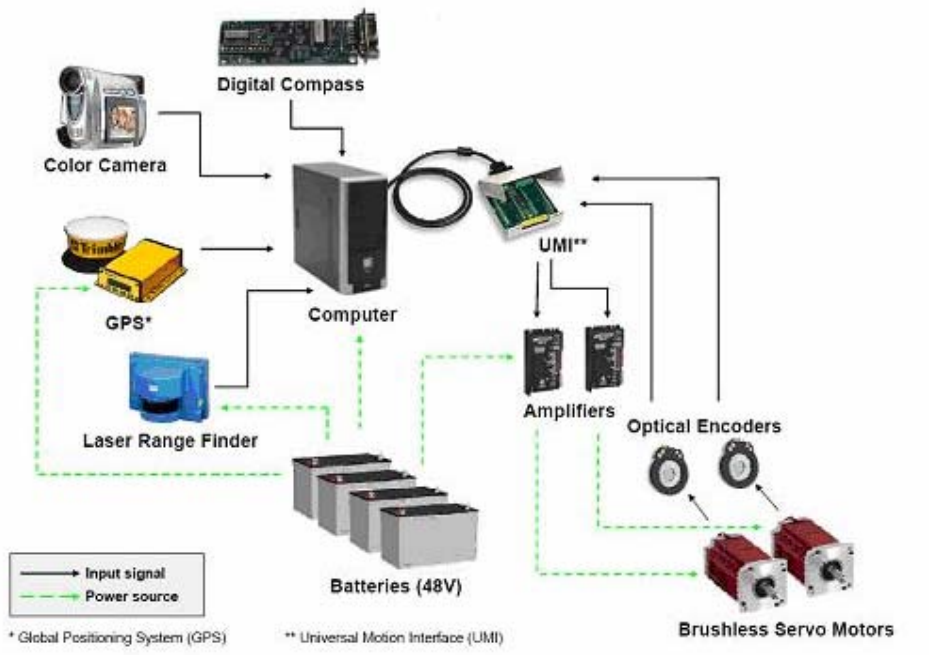

Figure 4 -- Electronics

3.3.1 Power

AWESOM-O is powered with two 12 volt 40 amp hour SLA batteries and four 12 volt 20 amp hour SLA batteries, combining for a 48 volt 40 amp hour system. This power system gives AWESOM-O an estimated 45 minutes of runtime with full load on the drivetrain, sensors, and computer.

3.3.2 Motion Control

AWESOM-O's movements are controlled by a National Instruments PCI-7344 motion control card driven by National Instruments LabVIEW 8.0 software. The motion control card is interfaced to the servo amplifiers through a Universal Motion Interface (UMI) board from National Instruments. The UMI board allows the motion control board to work with 3rd party servo amplifiers. Two Advanced Motion Control servo amplifiers provide the needed PWM signals to drive the servo motors to the desired trajectory. The optical encoders provide the needed feedback signals to close the loop for the motion control boards PID controller.

To add extra safety precautions to AWESOM-O, two emergency stops are built into the vehicle. There is a mechanical E-stop located on the mast (red button) and a wireless E-stop. The wireless E-stop is controlled by a RF controller and actuated by a radio controlled servo motor.

Computer

AWESOM-O's computer is tailored for power efficiency and processing power. The motherboard is an ASUS Micro ATX Intel motherboard. The Micro-ATX platform was choosen for its small footprint. The Intel Core Duo 1.83 GHz processors are energy efficient yet powerful. To supplement the processing power of the CPU, a NVIDIA GeForce 7900 GPU is used to do the image processing, saving the CPU much needed processing power. Other important peripherals for the computer include a multi-port serial card and the NI motion control card. A 100 gigabyte laptop hard drive was selected for its performance in environments experiencing harsh vibrations.

3.3.3 Sensors

All of the sensors for AWESOM-O were reused from the 2005 U of M vehicle, the G2. The sensors include: a SICK LMS, Trimble differential GPS unit, Honeywell magnetic compass, Canon Optura 50 camera, and optical encoders.

3.3.4.1 SICK Laser Measurement System

The LMS is used to detect three dimensional obstacles. It can detect obstacles as far as 50 meters away and can produce ten 180 degree scans a second at one degree resolution with a normal serial port. The SICK has proved to be one of the most reliable sensors on AWESOM-O and is a key component of the obstacle avoidance system.

Figure 5 -- SICK LMS

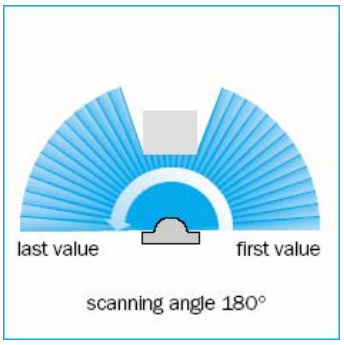

Figure 6 -- Measuring Range

3.3.4.2 Trimble Differential GPS Unit

Global position information is provided to AWESOM-O by a Trimble AG-124 DGPS unit. The unit receives RTCM beacon correction signals provided by the US Coast Guard to correct for errors in GPS signals passing through atmospheric interference. With the correction signals the position information is accurate to less than one meter and under favorable conditions is accurate to less than two feet. The information used by AWESOM-O provided by the GPS unit includes: latitude, longitude, magnetic variation from true north, and GPS quality. This information is provided one time a second to AWESOM-O's computer via serial port.

3.3.4.3 Magnetic Compass

AWESOM-O is provided with heading information by a Honeywell HMR3300 magnetic digital compass. The observed accuracy of the compass is about 5 degrees. These errors can be attributed to magnetic interference from the robots motors and other ferrous material in the compass's environment. In general magnetic compasses are not very reliable but for its limited use in this project it is sufficient.

3.3.4.4 Camera

A Canon Optura 50 DV camcorder is used to acquire video to the computer vision algorithm using firewire at a rate of 30 frames per second. This camera was chosen for it programmability, image stabilization, and image quality. The camera is fitted with a fisheye lens to increase the field of view to approximately 160 degrees.

3.3.4.5 Optical Encoders

The motor position is given to the motion control board by the optical encoders, which have 8000 counts per revolution resolution. The encoders are needed to give relative motor position feedback information to the PID loop for motion control. The encoders seem to be very accurate and reliable sensors.

3.4 Vehicle Characteristics

Dimensions	28 x 52 x 72"
Max Speed	4.7 mph
Battery Life	45 minutes
Climbing Ability	30 degree grade

4. Software

The 2007 team decided to follow in the footsteps of the 2005 team by using National Insturment's LabVIEW 8.0 programming language to develop all of AWESOM-O's software except for the image processing. LabVIEW 8.0 was chosen because of its dense built in functionality and extensive libraries. The image processing software was written in C++ and developed in Microsoft Visual Studio.

5. Motion Control

AWESOM-O's motion control software controls the motors to produce the desired trajectory of the vehicle. The desired trajectory is either produced by a user via remote controller or by the path planning software for the autonomous and navigation challenges.

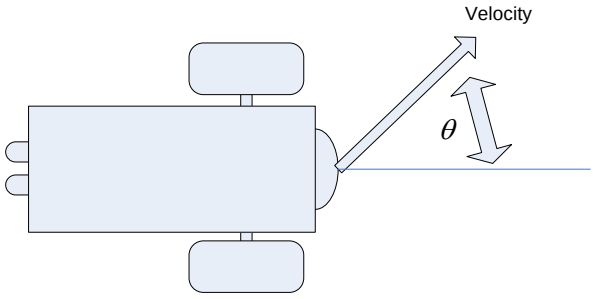

Figure 7 – Vehicle Model

The kinematic model for a vehicle with front differential steering is as follows.

$$\frac{d\theta}{dt} = \frac{V_r - V_l}{l_w}, \quad V = \frac{V_r + V_l}{2}$$

Where V_r, V_l, V, and l_w is the right wheel velocity, left wheel velocity, vehicle velocity and track width respectively. This model assumes zero tire slip. The control laws are then created from the kinematic equations, and are as follows.

$$V_r = \left(1 - \frac{\theta_{des}}{\pi}\right)V_{des},\ V_l = \left(1 + \frac{\theta_{des}}{\pi}\right)V_{des}$$

Where V_{des} is the desired vehicle velocity. The left and right motor velocities are then fed into the motion control PID loop executed by the motion control board.

6. Obstacle Avoidance and Path Planning

AWESOM-O has a unique path planning algorithm. First, all obstacles sensed by the LMS or camera are placed into a local map. The local map consists of an 81 by 81 two dimensional digitized grid where each zone is an eighth of a meter. A "1" is placed in the grid zone where an obstacle is detected and a "0" is placed every where else. This map is then convolved with a Gaussian kernel, which gives values to zones that are near obstacles. This way, AWESOM-O is penalized for traveling near obstacles. The path planning algorithm chooses the goal zone by looking at a Manhattan distance of 10 zones away from the vehicle and selecting the zone with the lowest cost. Then, a trimmed A-star algorithm is employed to determine the path with the lowest cost to the goal zone. A trimmed version of A-star was used because the full A-star algorithm expands too many nodes, making the system perform slower than desired. Once the path is known, the desired steering angle is calculated and fed into the motion control algorithm.

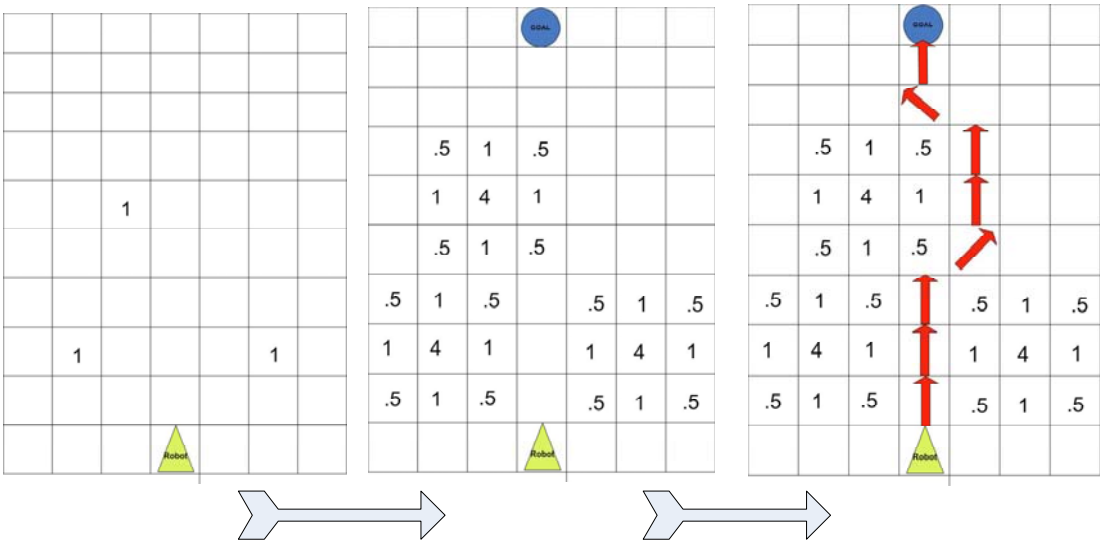

Figure 8 -- Path Planning Diagram

7. Autonomous Challenge

The autonomous challenge software takes images from the camera and the LMS data as inputs and produces a steering angle and speed for the vehicle. The images are processed to find the lane boundaries and the LMS is used to detect the position of the three dimensional obstacles. A path planning algorithm is used to guide the vehicles motion through the course.

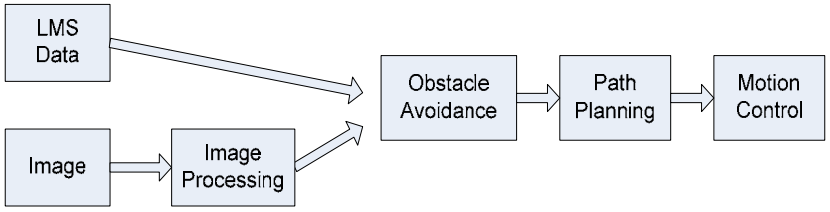

Figure 9 -- Autonomous Challenge Flow Chart

7.1 Image Processing

In designing the vision system, AWESOM-O leverages the superior image processing capabilities of modern programmable graphics hardware. The central processor on a programmable video card is often referred to as a Graphics Processing Unit (GPU). Programmable GPUs offer much higher performance than more expensive specialized hardware but still have the benefit of offloading the vision computation from the CPU. In addition, the memory bandwidth between the GPU and the CPU is much higher than for typical daughterboards allowing fast exchange of the vision data with the central AI program.

AWESOM-O was already being designed around a small form-factor, desktop computer which could easily be equipped with a powerful video card for nominal additional cost. Programming the image processing routines and the proper filters for the vision system is also eased by the examples in the publicly available OpenVIDIA library. What follows is a description of the final hardware setup, software structure and algorithm design of our vision system.

7.1.3 Image Processing Software

A calibration of the wide angle lens needs to be performed before utilizing the vision software. This is done using the Omni-directional Camera Calibration Toolbox for Matlab. By taking images of a checkerboard from many different arbitrary angles this toolbox is able to compute a mapping from each pixel to a vector into the world. It can also compute a transformation from the camera location to a plane containing any one of the checkerboards. By adding a picture of

the checkerboard taken with the camera on the robot and the checkerboard on the ground, the final mapping from pixels in the video image to points on the ground plane is obtained.

There is a very simple image processing library for computer vision on the GPU called OpenVIDIA. This library is not a robust, marketable library but rather a "proof-of-concept" showing that extrememly efficient vision algorithms can be designed that use the GPU at a fraction of the cost of their commercial counterparts. Utilizing the same framework demonstrated in OpenVIDIA, a new library called Flex-OpenVIDIA (FlexOV) was written.

With this framework, a bridge between FlexOV and the LabVIEW AI code is needed. LabVIEW provided a superior framework for data I/O and processing as well as a simple, effective system for parallelizing our code taking advantage of our dual-core CPU. The new PCI Express system bus offered fast enough speed that this consideration was not necessary. The system was able to quickly move the data to main memory without introducing lag.

Once in LabVIEW, a small amount of additional processing was in order so Intel's OpenCV library was utilized within LabVIEW to complete the vision system. In summary, the pipeline starting from the camera comes in on the firewire bus. Then it is decompressed and de-interlaced in the CPU, passed to the GPU, processed and reduced by our filter-chain. Finally it is passed back to main-memory and into LabVIEW where it is processed again and added to the data from our other sensors for the AI system to utilize.

7.1.4 Vision Algorithm Breakdown

1) Video Acquisition (Fig. 1-4)

The video stream is decompressed and deinterlaced using the Microsoft DirectShow library. Then a histogram of the blue channel of data is created. This histogram will be utilized later in the filter process. The video frame is then sent to the video card as a texture.

Figure 10: Input Video w/ Histogram &

2) Image resizing and Masking (Fig. 1-5)

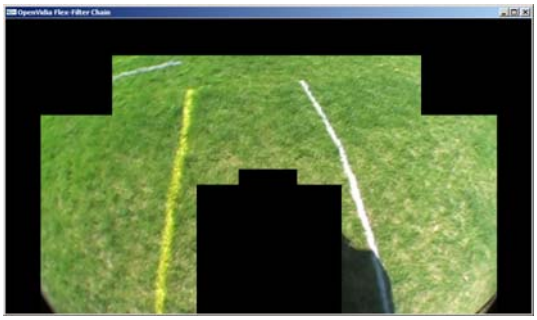

Figure 11: Image Resizing & Masking

The data coming off of the camera has a resolution of 720x480 pixels. However, the aspect ratio of the image should be 16:9 (the typical widescreen TV aspect ratio). 720x480 is 3:2. This means the pixels are not square. To rectify this we resize the image to be 853x480. This is the nearest resolution that results in square pixels without losing any detail. Also, pixels that contain the robot and the distant horizon are masked out.

3) Dynamic Color Threshold (Fig. 1-6)

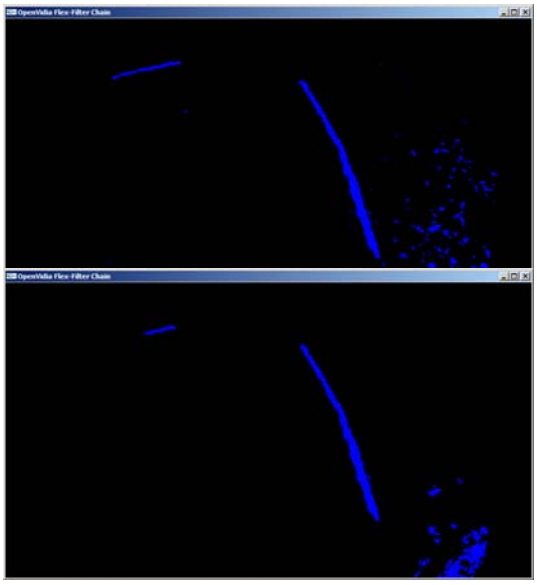

Figure 12: Particle Filtering

As a first step towards isolating the painted lanes of the autonomous course a color threshold to the blue channel is applied. The threshold used is calculated using the histogram computed earlier. The global max of the histogram is found, and then a pre-determined histogram bucket size is located past the peak. This value becomes the threshold. Removing all pixels below this value eliminates much of the background and the orange of the construction obstacles. It is also partially effective in removing the grass. There is still a significant amount of noise in the grass at this point.

4) Particle Filtering (Fig. 1-7)

The first layer of noise reduction is a particle filter. This filter examines the relative size of each detected blob by examining surrounding pixels. If a blob is small and disconnected from other blobs it is removed by this filter. This is effective in removing small particulate sized noise in the image.

5) Re-map to ground plane (Fig. 1-8, left)

Now, the results are ready for LabVIEW. To do this, a precomputed lookup table that associates an XY coordinate in the ground plane with each pixel in the image is used. Then a list of points in the ground plane is built (one for each pixel) and placed in a shared page-file space for LabVIEW to retrieve.

6) Blob Detection and Removal

The second layer of noise reduction is a full blob detection and removal algorithm. A blob detection library is used that sits on top of the Intel OpenCV library. It detects large blobs and removes them from the image while still preserving lines.

7) Probabilistic Hough Transform (Fig. 1-8, right)

The last layer of noise reduction is the probabilistic form of the Hough transform. This transformation function is implemented in the OpenCV library. It returns to us a list of endpoints of highly probable lines in the image. These line segments are then rasterized at a reduced resolution to create the final noise reduced image.

8) Final Resolution Reduction

The rasterized line segments are then reduced in resolution again to a final matrix of 1's and 0's where a "1" indicates the presence of a lane and "0" the absence. This matrix is used as is for the path planning algorithm running in another thread.

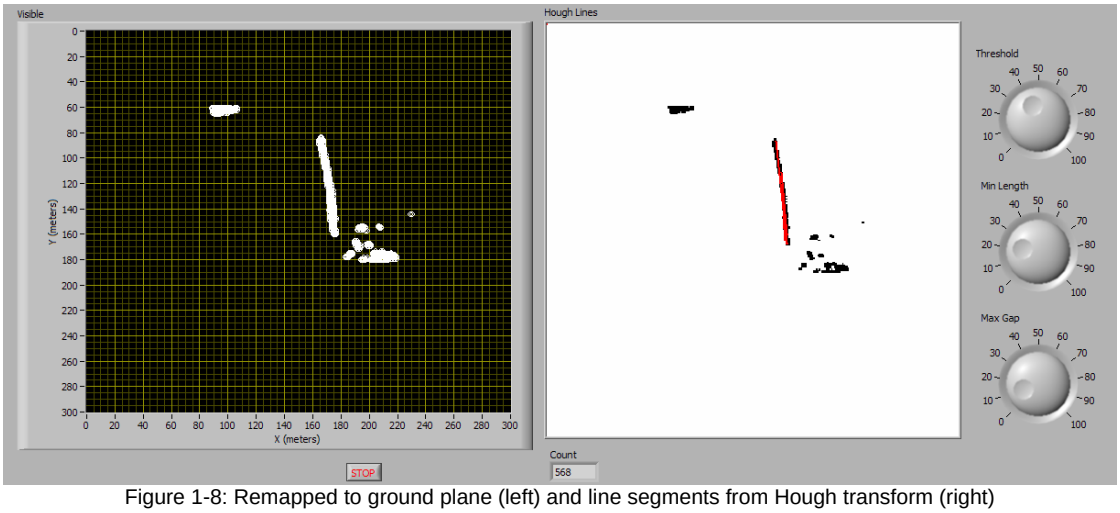

Figure 1-8: Remapped to ground plane (left) and line segments from Hough transform (right)

7.2 Autonomous Challenge Path Planning

To navigate through the autonomous course the path planning algorithm fuses the image processing data with the LMS data to produce the local map described in section 6. This local map is then convolved with a Gaussian distribution to penalize the vehicle for attempting to move close to obstacles. A heading grid is then added to the convolved grid to reward the vehicle for driving straight ahead. The heading grid has a constant gradient in both the lateral and longitudinal directions of the vehicle. The final grid is then given to the modified A-Star algorithm to determine the path and desired steering angle.

Figure 11 -- Grid Formulation Flow Chart

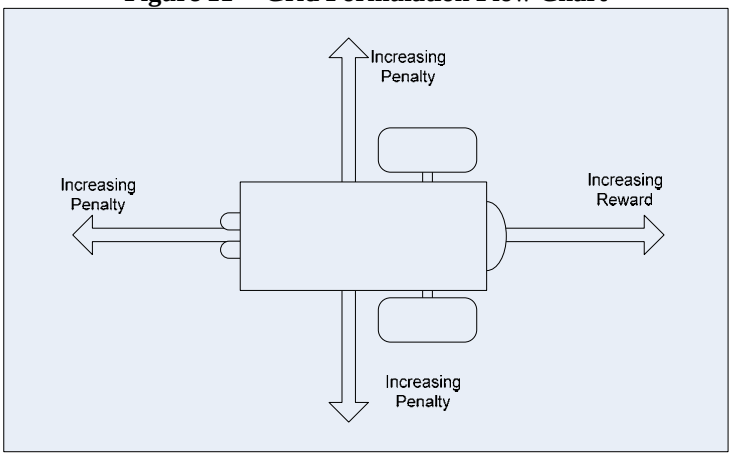

Figure 12 – Heading Grid

8. Navigation Challenge

AWESOM-O's navigation software consists of 3 different driving modes. The first is global mode where navigation is determined by a global obstacle map. The second is local mode where AWESOM-O heads directly to the way point, and the third is autonomous mode where AWESOM-O uses the same path planning algorithm used in the autonomous challenge.

8.1 Global Mode

The navigation challenge path planning algorithm utilizes a global map so AWESOM-O does not get trapped in concave obstacle configurations. If AWESOM-O does not see any obstacles in its field of view and is more than 15 meters away from the way point, then the global map

determines the best path toward the way point using a full A-star searching algorithm. Here, the full A-Star algorithm is used because the scores are binary yielding fewer expanded nodes, increasing the speed of the algorithm. The A-star algorithm is able to calculate the best possible path to a goal state by minimizing a heuristic function. The heuristic function assigns cost values associated with traversing specific zones. The global map is a 100 by 100 square grid where each zone is one square meter. Once an obstacle is detected on the course, the particular zone that contains this obstacle is deemed untraversable and given a high cost, meaning the A-star algorithm will not choose a path to the way point that travels through this zone. This means that AWESOM-O has the ability to go into traps, detect that a trap is present in a particular location, leave, and not return.

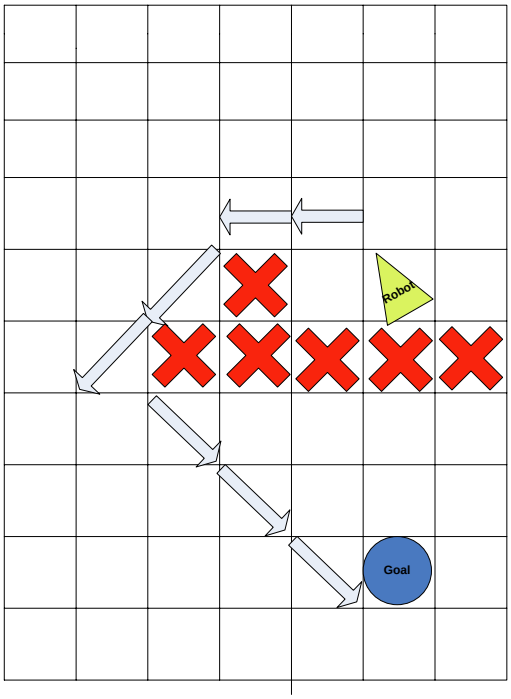

Figure 13 -- Global Map

8.2 Local Mode:

If AWESOM-O is less than 15 meters away from the way point and does not see any obstacles then it heads directly to the way point.

8.3 Autonomous Mode

If AWESOM-O does see obstacles and is within 15 meters of a way point, then it does not utilize the global map. Instead, it travels in autonomous mode which uses the same path planning algorithm as used in the autonomous challenge.

9. Conclusions

This year's IGVC team created a smaller and more maneuverable vehicle and developed completely new software for both the autonomous and navigation challenge. The path planning and obstacle avoidance algorithms are more advanced and robust than the 2005 University of Minnesota IGVC team. As a result, we hope to have great success in this year's competition.

10. Acknowledgments

Special thanks needs to be given to the Intelligent Transportation Systems (ITS) Institute at the University of Minnesota for financially supporting this year's team. Thanks to our graduate advisors for being so understanding of our lack of research progress during the spring semester. We would also like to thank Jed Rutherford from National Instruments for donating LabVIEW 8.0 and teaching the team essential LabVIEW programming skills. We would also like to thank Shawn Brovold, a 2005 team member, who helped us get started with the project and tell us what worked and what didn't work in 2005. Lastly, we would like to thank everyone affiliated with IGVC for making the competition possible.

Appendix A: AWESOM-O BOM

Description	Qty	MSRP	Team Cost
Sensors			
Sick Laser Range Finder	1	5700	0
Honeywell HMR3300 Digital Compass	1	450	0
Trimble AG-124 DGPS Receiver	1	2995	0
Cannon Optura 50 DV Camera	1	450	450
Computer			
2.0 GHz Micro-ATX PC	1	1100	1100
Geforce 7900 Video Card	1	260	260
7" Touch Screen Monitor	1	400	400
NI Motion Control Board		1650	1650
Hardware			
Aluminum Frame	1	700	700
Kollmorgan 1HP DC Brushless Servo Motors	2	1200	0
Boston 20:1 Worm Gears	2	800	0
Advanced Brushless Servo Amplifiers	2	950	0
Toyo 12 Volt 40 Amp-hour batteries	2	120	120
Toyo 12 Volt 20 Amp-hour batteries	4	160	160
Software			
Labview 8.0	1	2399	0
Visual Studio 2005	1	2145	0
Total		21479	4840

Appendix-B JAUS Challenge

Our robot is able to respond to standard JAUS commands sent over UDP. Here we discuss our strategy for developing JAUS compliance including our research approach, robot design implications and the final implementation.

B.1 Team Education and Research

To learn JAUS we went right to the source documents. We read the Reference Architecture document paying particular attention to the JAUS message header structure, the data types and precision, and the specific commands that would be used in the competition. The material emphasizing the purpose and design methodology of JAUS gave us a good concept of what a JAUS compliant system should emphasize.

We also did a lot of experimenting using the JAUS Compliance Test Suite (JCTS) so we could be sure that the packets generated by JCTS were accurate. Intercepting and dissecting the JCTS messages allowed us to verify our packet processing logic. In addition, we developed a simple test application using the Qt GUI library to send and receive JAUS packets (fig. 16). This system incorporated a user friendly GUI which considerably hastened our development and testing process.

B.2 Design Considerations

All of the components of our system are isolated. They run in their own thread of execution and process their own information before passing this information to a central path planning thread. This design fits well with the JAUS system. Each sensor or control device is capable of sending JAUS messages to other devices or to the central control system. The protocol of data communication is independent of the system architecture.

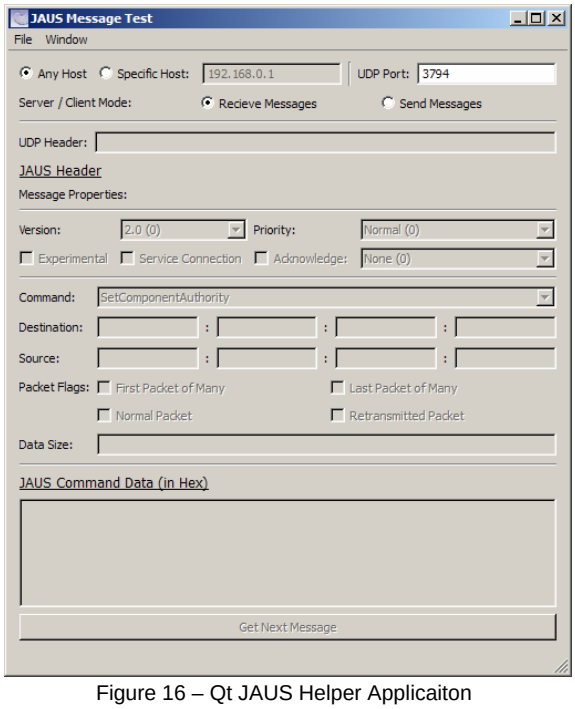

Figure 16 – Qt JAUS Helper Applicaiton

This isolated design allows us to create a central arbitration system that receives JAUS messages and routes them to the appropriate devices in a format understood by that device. The JAUS arbitrator can also respond to queries for information by first retrieving the message and then restructuring it into a JAUS message to be delivered to the system that requested it. Additionally, this approach keeps the JAUS implementation decoupled from the individual components making it reusable for other systems and other devices.

B.3 Implementation Details

Our message parsing code centers around a single C struct. The struct has members that align with the different fields of the JAUS header. It also contains several functions that allow access to the sub-byte data level. These functions use bit masking and bit shift operations to isolate individual bits. To parse a message, we simply find the start of the message in memory and then type-cast it to this struct. As long as the struct is packed tightly (i.e. there is no multi-byte memory alignment taking place) then the struct members will line up perfectly with the JAUS message fields.