WAVE2: Warriors Autonomous Vehicle

2010 IGVC Entry

Wayne State University

Advising Faculty:

Dr. R. Darin Ellis

Team Members:

Justin Ar-Rasheed

Shawn Hunt

Sam Li

Vishal Lowalekar

Prem Sivakumar

Ali Syed

Faculty adviser statement
I certify that the engineering design in the vehicle by the current student team has been significant and equivalent to what might be awarded credit in a senior design course.

Prof. R. Darin Ellis
Industrial and Manufacturing Engineering
Wayne State University

1. INTRODUCTION

This technical report details Wayne State University's submission for the 2009 Intelligent Ground Vehicle Competition. This is Wayne State's first submission to the IGVC, which we all hope will only be the first of many entries in future years. Our team draws on students mainly from Electrical and Computer Engineering.

Section 2.0 describes the hardware architecture that we used including the electronics, electrical system, actuators, sensors, and computers. Section 3.0 describes our approach to signal processing, path following, and the control loop for moving the platform. Section 4.0 describes our system integration plan.

2. HARDWARE OVERVIEW

2.1 Introduction

This section describes the hardware used in our entry. Section 2.2 describes the platform. Section 2.3 describes the electronics and sensors we chose to use. Section 2.4 describes the electrical system. Section 2.5 describes the actuator that we are using.

2.2 Platform

The platform that we used for our entry started off as a wheelchair. Figure 1 is a picture of the platform equipped with some electronics. It is a skid-steer design with a rear passive wheel.

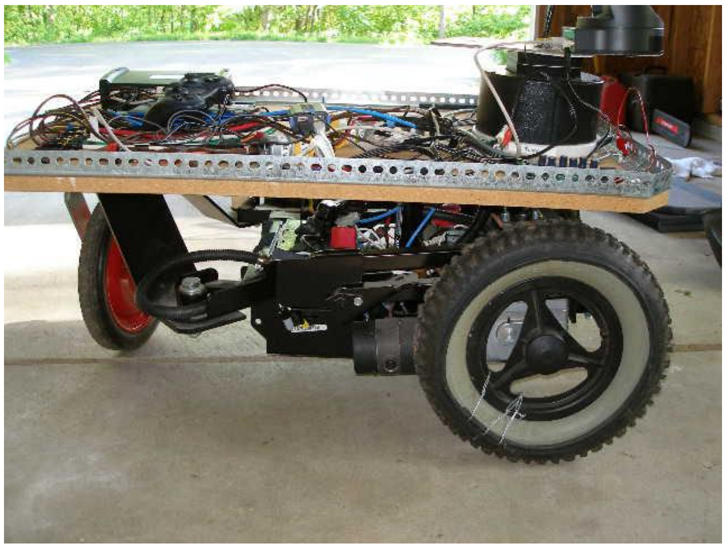

Figure 1: Our modified wheelchair base during the build process.

2.3 Electronics and Sensors

At the time of writing, we are still working on using a Mac Mini loaded with Ubuntu 8.04 as the brains of our robot. In the work we have done on other projects, we are able to run code on the dual processors as well as the NVIDIA GPU that is CUDA compatible. In the interim, ee are using one laptop, a Gateway tablet PC, that has been loaded with Ubuntu 8.04. This laptop does not contain any serial ports. A device made by Quatech accepts a USB connection in and provides eight RS232 ports out, each capable of data rates up to 921 Kbps. The devices that we have plugged into the RS232 box are an inertial measurement unit (IMU), a global positioning system (GPS), an RS232 connection for the Sony pan/tilt/zoom camera, and for communicating with the Roboteq motor controller.

We are using passive optics for our obstacle detection, which is described in more detail in our software overview. The Sony camera outputs an NTSC signal which is fed into a PCMCIA frame grabber from ImperX to the laptop. The overall view of our electronics is shown in Figure 2.

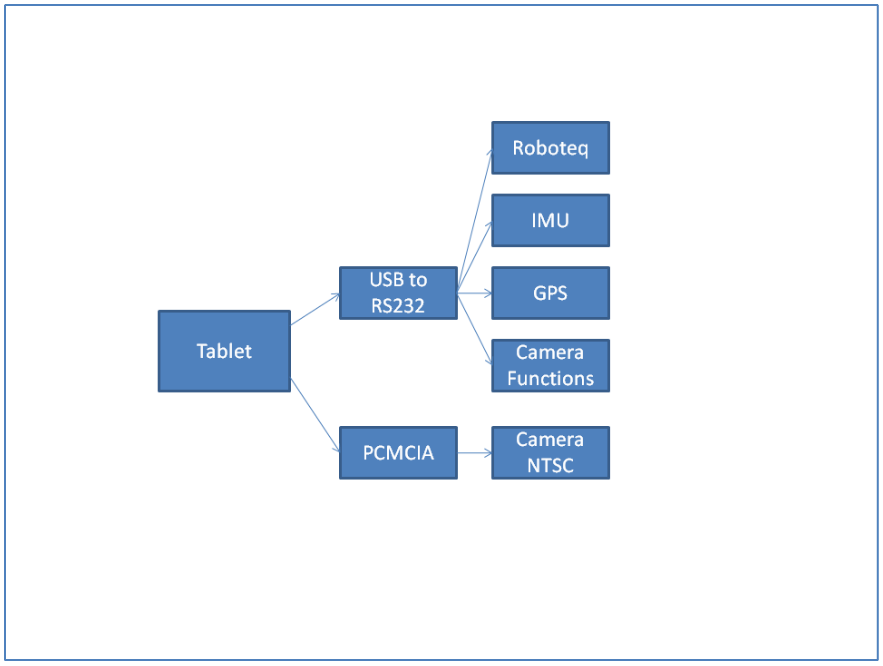

Figure 2: Architecture of our computer and sensors

2.4 Electrical System

There are two 12V batteries in the base of the wheelchair. They are connected in parallel to double the amp hours that are available. There is a connection coming off of one battery to power the equipment on top of the platform.

To avoid excessive power draw by the two motors, a 30A fuse is installed in series with the emergency stop. If the power consumption by any of the devices exceeds that limit, the fuse will disconnect the power supply to the platform. The hardware emergency stop button can also be used to quickly disconnect power to the platform and all devices, in case of abnormal operation.

The sensors being used can be divided into two groups based on their power consumption. The GPS receiver and the IMU both require 9VDC and the Sony camera requires 12 VDC. In order to supply this range, a DC-DC converter is used that accepts 6-30 VDC in and is able to step that voltage down to three separate outputs.

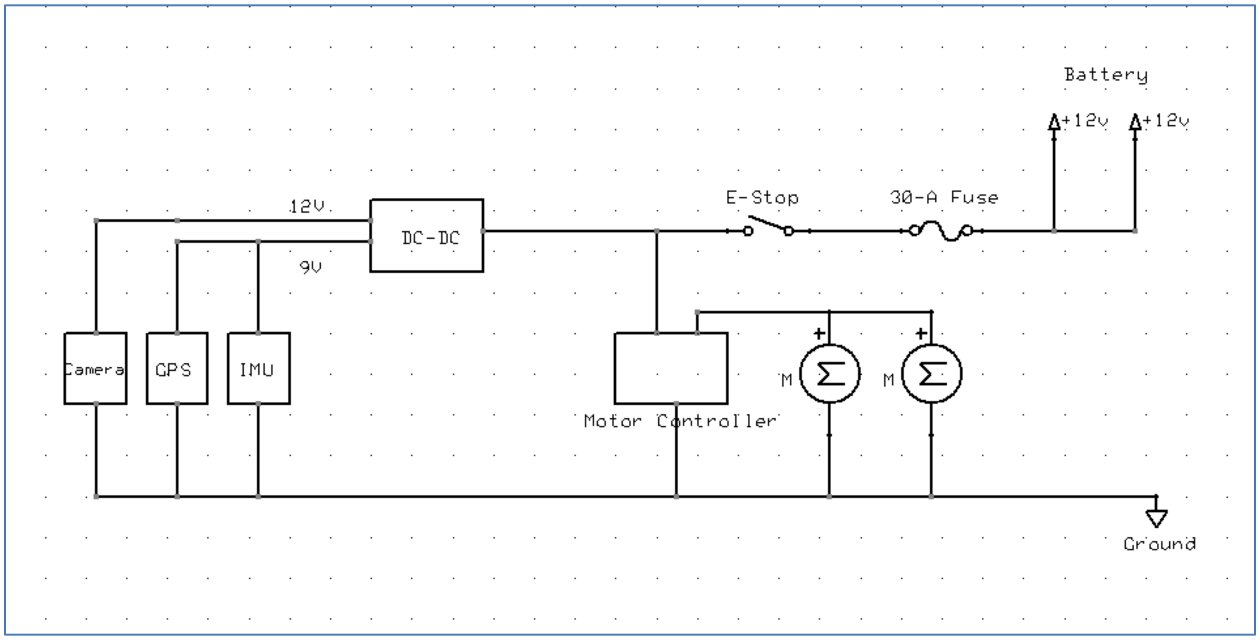

Figure 3: Electronic wiring diagram of our IGVC platform

2.5 Mounting

This year we are using a fixed pole to mount our camera and DGPS receiver on. Last year, we used a zipper mast that caused problems when we drove the robot with it extended at any length.

3. SOFTWARE OVERVIEW

3.1 Introduction

This section describes the software architecture that we used for our entry. Section 3.2 describes the signals we are processing. Section 3.3 details the approach we used to handle JAUS communications. Section 3.4 describes our algorithm for lane detection and path following. Section 3.5 describes our stereo vision processing and Section 3.6 summarizes our control loop for autonomous navigation.

3.2 Signal Processing

Our approach to reading data from each of the sensors was to use shared memory on the laptop. The program that reads from each device spawns a thread for each device and writes the current reading to its area in shared memory. There is a location in shared memory for image data, GPS data, IMU data, and feedback from the motor controller. The module that does the path planning reads this data in and then acts upon it. There is a mutex that has to be acquired by either the process doing the reading or the writing of the data in shared memory. This protects the data from becoming corrupted.

The images from the camera come in at 640x480 resolution. The pan, tilt, and zoom functions of the camera are available through the RS232 interface but right now, we are not using them in our control loop.

The Trimble GPS outputs NMEA data. The NMEA sentences that we are processing are GGA and RMC. The GGA sentence gives latitude and longitude. The RMC sentence gives us fix information. The data rate of the GPS is 1 Hz.

The data that we are reading from the IMU are the acceleration, angular rate, and Euler angles (pitch, roll, and yaw). The data rate of the IMU is around 15 Hz.

The data that we are reading back from the Roboteq motor controller are amps consumed by each motor, current battery voltage, and the relative encoder feedback from each motor.

3.3 JAUS

In accordance with IGVC's JAUS Challenge requirements, the platform will respond to a periodic status and information requests from the Judge's COP. The development of our JAUS code is based on Jr Middleware software development tool. We created six JAUS services to meet the Interoperability Challenge. The JAUS services include: (1) transport and capability discovery service, (2) access control service, (3) management service, (4) local pose sensor service, (5) velocity state sensor service, (6) local waypoint driver service. Our team also developed and implemented the JAUS messages from the JAUS Core Service Set (AS5710) and the Mobility Service Set (AS6009) to meet the requirement. The JAUS messages that are used in our platform are shown in the figures below. Figures 4 and 5 show the JAUS services and messages that we implemented.

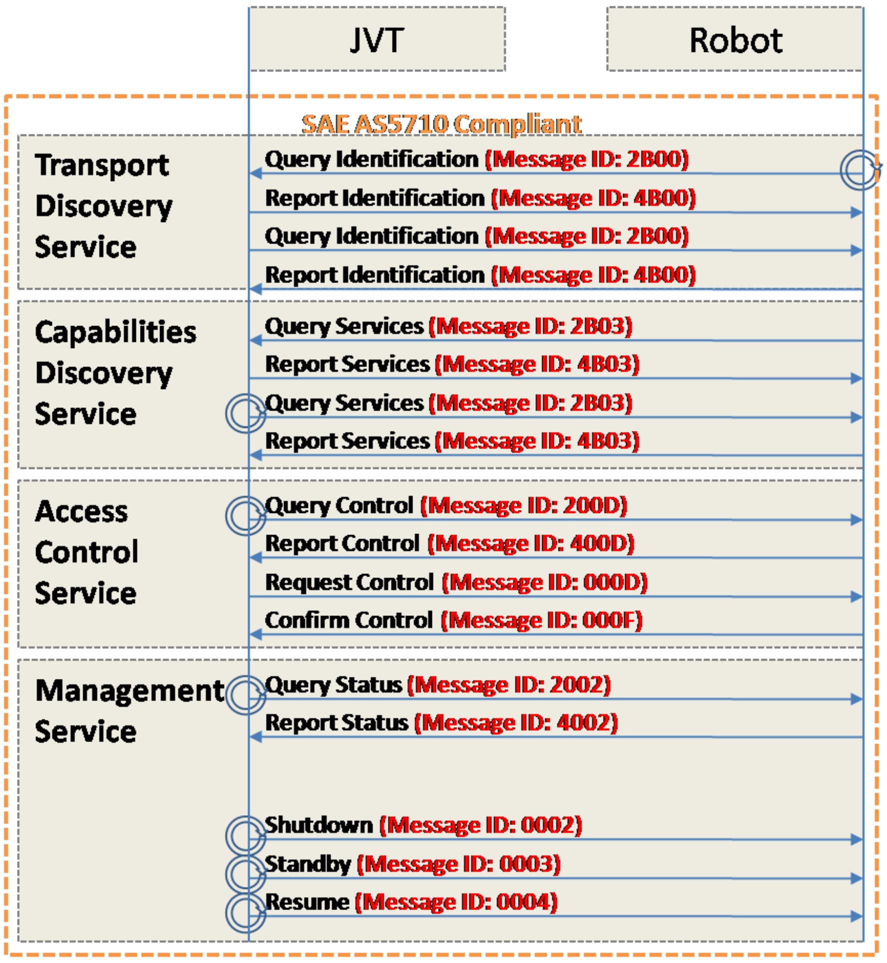
JVT
Robot
SAE AS5710 Compliant
Transport Discovery Service
Query Identification (Message ID: 2B00)
Report Identification (Message ID: 4B00)
Query Identification (Message ID: 2B00)
Report Identification (Message ID: 4B00)
Capabilities Discovery Service
Query Services (Message ID: 2B03)
Report Services (Message ID: 4B03)
Query Services (Message ID: 2B03)
Report Services (Message ID: 4B03)
Access Control Service
Query Control (Message ID: 200D)
Report Control (Message ID: 400D)
Request Control (Message ID: 000D)
Confirm Control (Message ID: 000F)
Management Service
Query Status (Message ID: 2002)
Report Status (Message ID: 4002)
Shutdown (Message ID: 0002)
Standby (Message ID: 0003)
Resume (Message ID: 0004)

Figure 4: The JAUS service and messages from SAE AS5170 that are used in our system.

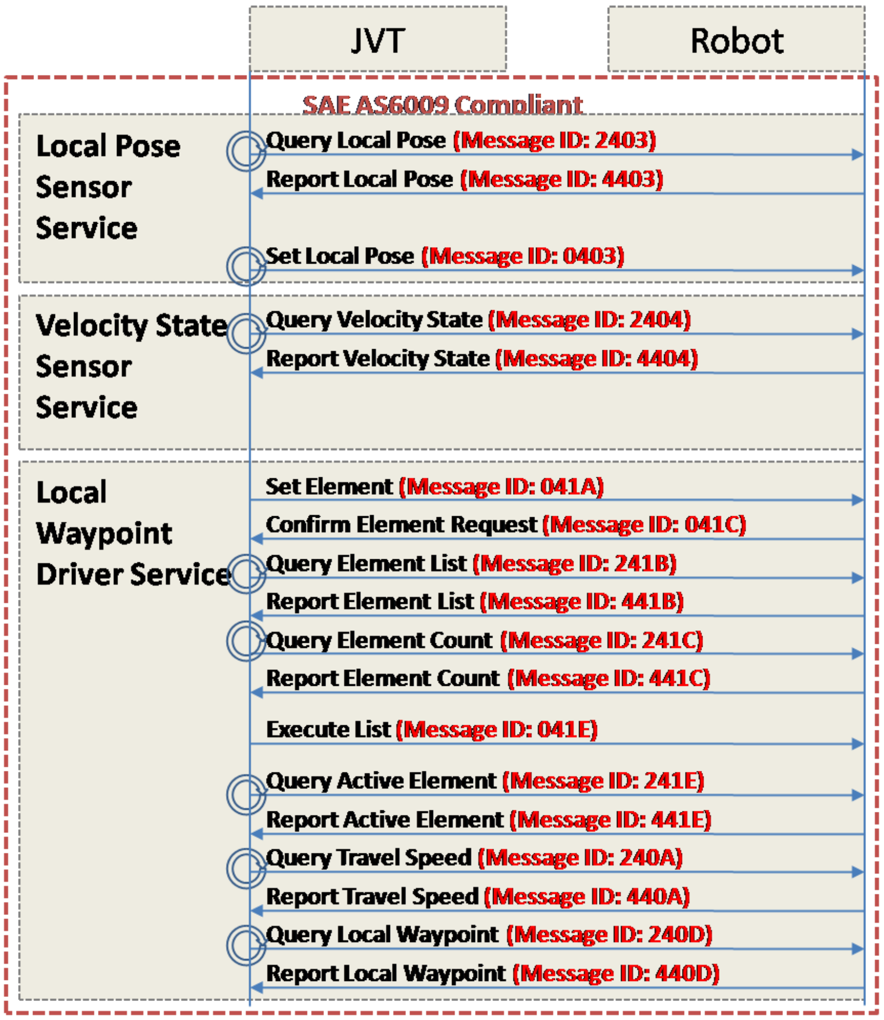
JVT
Robot
SAE AS6009 Compliant
Local Pose Sensor Service
Query Local Pose (Message ID: 2403)
Report Local Pose (Message ID: 4403)
Set Local Pose (Message ID: 0403)
Velocity State Sensor Service
Query Velocity State (Message ID: 2404)
Report Velocity State (Message ID: 4404)
Local Waypoint Driver Service
Set Element (Message ID: 041A)
Confirm Element Request (Message ID: 041C)
Query Element List (Message ID: 241B)
Report Element List (Message ID: 441B)
Query Element Count (Message ID: 241C)
Report Element Count (Message ID: 441C)
Execute List (Message ID: 041E)
Query Active Element (Message ID: 241E)
Report Active Element (Message ID: 441E)
Query Travel Speed (Message ID: 240A)
Report Travel Speed (Message ID: 440A)
Query Local Waypoint (Message ID: 240D)
Report Local Waypoint (Message ID: 440D)

Figure 5: The JAUS services and messages from SAE AS6009 that are used in our system

3.4 Lane Detection

We took a simple approach to lane detection this year. We implemented several lane detection algorithms including some simple thresholding techniques along with edge detection algorithms such as Canny [1] and Sobel [2]. Our algorithms operate in both the RGB (red, blue, green) colorspace and HSV (hue, saturation, and value). We chose the HSV colorspace because our experience has been that it is less susceptible to being affected by shadows. We took data that we collected from the course from last year as well as from the test course we constructed to see which method works the best. These algorithms are affected by lighting conditions so our plan is to take our toolbox of algorithms with us and decide which one(s) work the best during the competition.

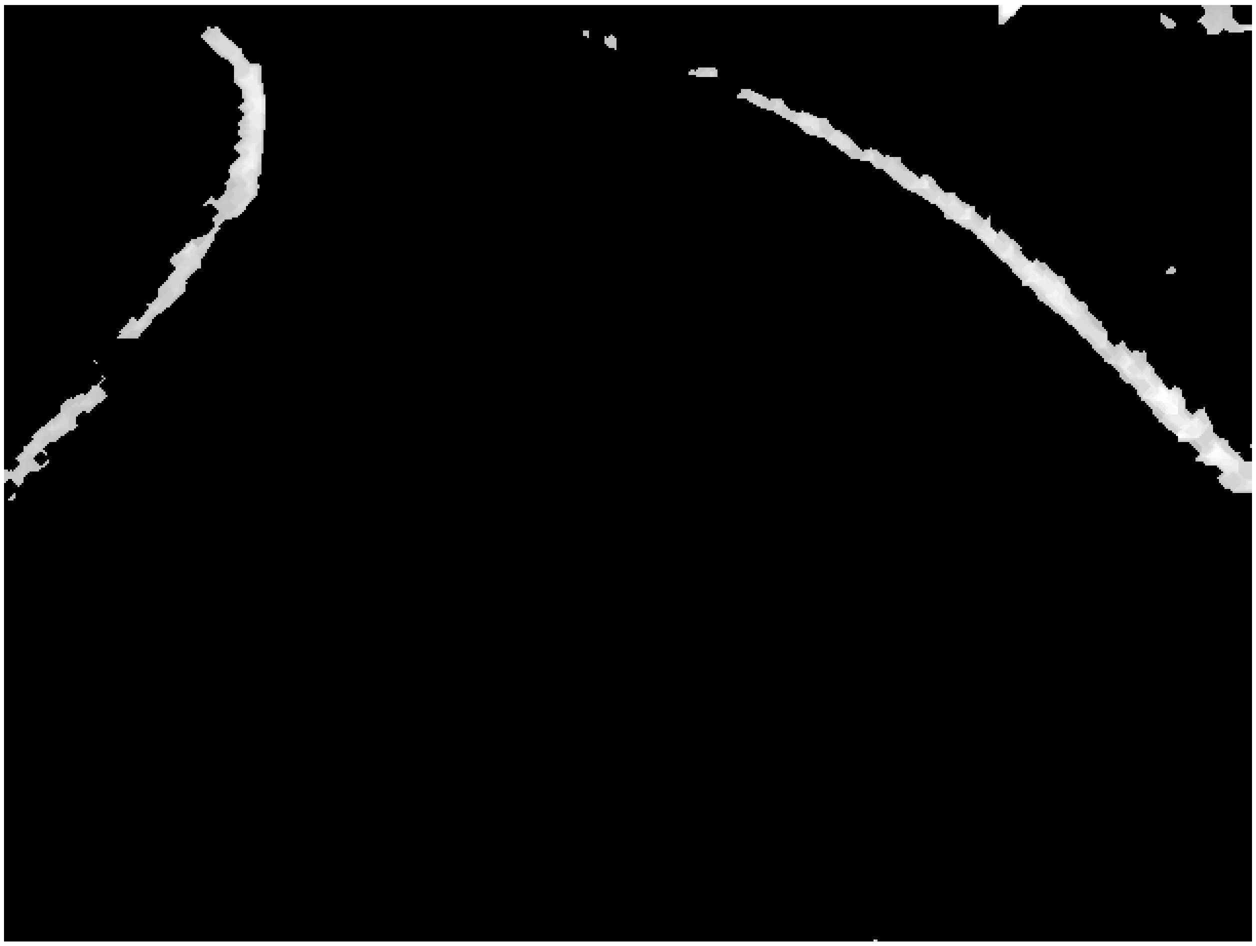

Figure 6: The results of our simple thresholding algorithm run on one of the images we took of the course last year.

3.5 Stereo Vision

This year we did not have access to a SICK laser scanner so we opted to use stereo vision for depth detection. We are using two FireWire cameras mounted at the front of the robot. Our stereo vision algorithm isn't anything novel and readily found in published literature. At the time of writing this, our algorithm uses a simple Sum of Squared Differences (SSD) approach to find pixels in the left and right camera views. The disparity value is computed using the Euclidean distance between each pair of points. Once the distances have been computed, they are assembled into a disparity map where the intensities represent the relative depth of points in the scene.

3.6 Control Loop

The control loop for the navigation challenge first requires the input of the waypoints. Our software grabs an image from the camera, segments it, checking for obstacles. It next grabs the current latitude and longitude from the GPS. If the path is clear, it moves toward the waypoint. If an obstacle is present, the platform attempts to navigate around the obstacle. Once the obstacle has been cleared, the current GPS location is obtained and a new path is calculated. This is repeated until the goal location has been reached.

The control loop for the autonomous challenge is similar to the control loop for the navigation challenge except here, once an image has been segmented; we pass it through our naive Bayes classifier for lane detection. The control module will align the robot so that a lane is to the left and to the right. Once the lanes have been found, the platform will move forward, checking for obstacles.

In addition to the hardware emergency stop described earlier, there is also a wireless emergency stop. There is a separate control box that houses a microcontroller and a ZigBee module. Once power has been applied to the circuit, the microcontroller starts transmitting a message that is received by a microcontroller that we have connected to the motor controller. If this message stream is not received by the microcontroller on the platform, the microcontroller will send a command to the motor controller ceasing all movement.

4. SYSTEM INTEGRATION

4.1 Introduction

This section describes how our system operates as an entire entity. We describe how our platform is able to handle inclines and obstacles. We also touch briefly on our algorithm for GPS waypoint navigation.

4.2 System Integration.

We have set the current limit of our motor controller to 30A. This setting does not permit the platform to travel faster than the 5 MPH limit as set by the competition rules. Our platform is able to handle ramp climbing. We have tested this out on thirty-degree ramps without any trouble. This was problematic prior to setting the current limit setting on our motor controller. We would pop fuses if the incline was too steep or the platform became stuck in mud. Our platform is also able to handle potholes without any trouble. We tested on potholes that were three feet in diameter and four inches deep.

Our platform is able to react instantaneously to the hardware emergency stop being pressed. We are able to detect an incline within one second by reading the pitch value from the IMU and detecting that the current draw from the motors has increased by reading the data from the motor controller.

We had thought to use a separate power supply on top of the platform to power the electronics but the lithium-ion batteries would last for approximately one hour before needing to be recharged. We were concerned that using the power supply from the base would introduce power spikes that would cause unpredictable results with our sensors. This has turned out to not be the case.

As of the time of this writing, our waypoint navigation code has not been finalized yet. The preliminary results of our code are that we are able to navigate to waypoints with approximately two meter accuracy. We also have not yet addressed the more complex obstacles that we will encounter such as switchbacks and dead ends.

5. REFERENCES

1. Canny, J., *A computational approach to edge detection.* IEEE Transactions on pattern analysis and machine intelligence, 1986: p. 679-698.

2. Sobel, I. and G. Feldman, *A 3x3 isotropic gradient operator for image processing.* Presentation for Stanford Artificial Project, 1968.

6. APPENDIX

A. Cost Breakdown

Item	**Cost**
Wheelchair Platform (donated)	
Roboteq AX3500 Controller	$ 395.00
Sony EVI-D70 PTZ Camera	$ 875.00
ImperX VCE-Pro Framegrabber	$ 599.00
Microstrain 3DM-GX1 IMU	$ 1,495.00
Trimble AgGPS	$ 5,000.00
Robotics Group Inc. DC-DC Converter	$ 295.00
Quatech USB to RS232	$ 800.00
Total	**$ 9,459.00**

B. Man Hours

Group	**Person**	**Man Hours**
JAUS	Sam	200
Vision	Shawn	250
	Vishal	80
Hardware	Ali	200
	Vishal	40
	Prem	40
	Justin	200
Total		**1010**