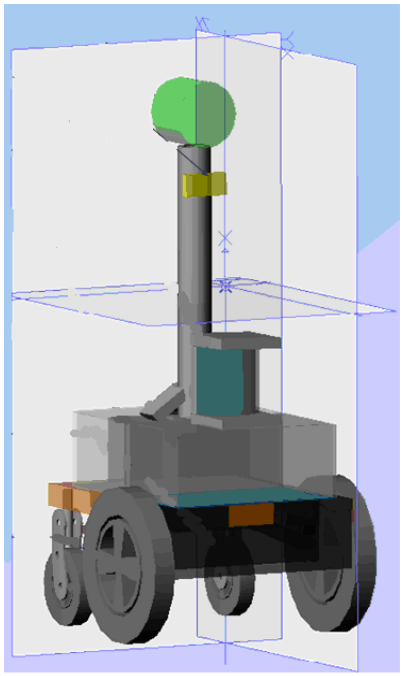
Team ZIPP-E: The University of Akron



Team Members:

(All team members are Senior Electrical Engineering Students)

Adam Rich

Amir Manteghi

Devin Ryder

Margret Dennes

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.

_____

Tom Hartley, Ph.D.

Professor, Department of Electrical & Computer Engineering

The University of Akron

Table of Contents

Path-Finding Navigation / Waypoint Navigation System 4
General Hardware/Other Systems 7
Electrical System 7
Locomotion System 7
Hardware Design 8
Software Design 10
Electronics 10
Printed Circuit Board 10
Lane Detection System 11
Technical Description 11
Hardware Design 12
Software Design 13
Obstacle Detection/Avoidance System 15
Technical Overview 15
Costs (Hardware Only) 17

Table of Figures

Figure 1 - System Block Diagram 3
Figure 2 - Zipp-E 3
Figure 3 - Main Control Software Flow Chart 4
Figure 4 - Path-Finding Navigation Flow Chart 5
Figure 5 - Navigation Graphical Representation (Waypoint and Path-Finding) 6
Figure 6 - Waypoint Navigation Flow Chart 6
Figure 7 - Drive Navigation Flow Chart 7
Figure 8 - Dual 25A Motor Driver 8
Figure 9 - Locomotion System Schematic 9
Figure 10 - PCB Artwork 11
Figure 11 - Populated PCB 11
Figure 12 - Lane Detection System 12
Figure 13 - Hardware Block Diagram 12
Figure 14 - Lane Detection Software Block Diagram 14
Figure 15 - Smart Camera Image 15
Figure 16 - Edge Polarity Profile 15
Figure 18 - Lidar Software Scan Sample Data and Graph (8m range) 16

Table of Tables

Table 1 - PCB IO 10
Table 2 - Lane Detection Hardware Modules 12
Table 3 - Image Processing Module 14

Design Planning Process

Zipp-E, the University of Akron's Intelligent Ground Vehicle, is the product of three semesters of work for the required Senior Design Project. The first semester was deciding what project each of the groups would be doing and putting together proposals for those projects. The second semester involved extensive research into alternative designs and deciding on the accepted design that the team would most likely be using. After discussing each of the alternative designs in the group, the design that is outlined in Figure 1 was chosen. Pseudo code was written for the GPS, Digital Compass, Lidar, Smart Camera, Stepper Motor, the Motor Controller and the Navigation System. This code is what was used as the base for the code that would be written in semester three. Orders for the required hardware that the team did not already have on hand were also placed during this part of the design process.

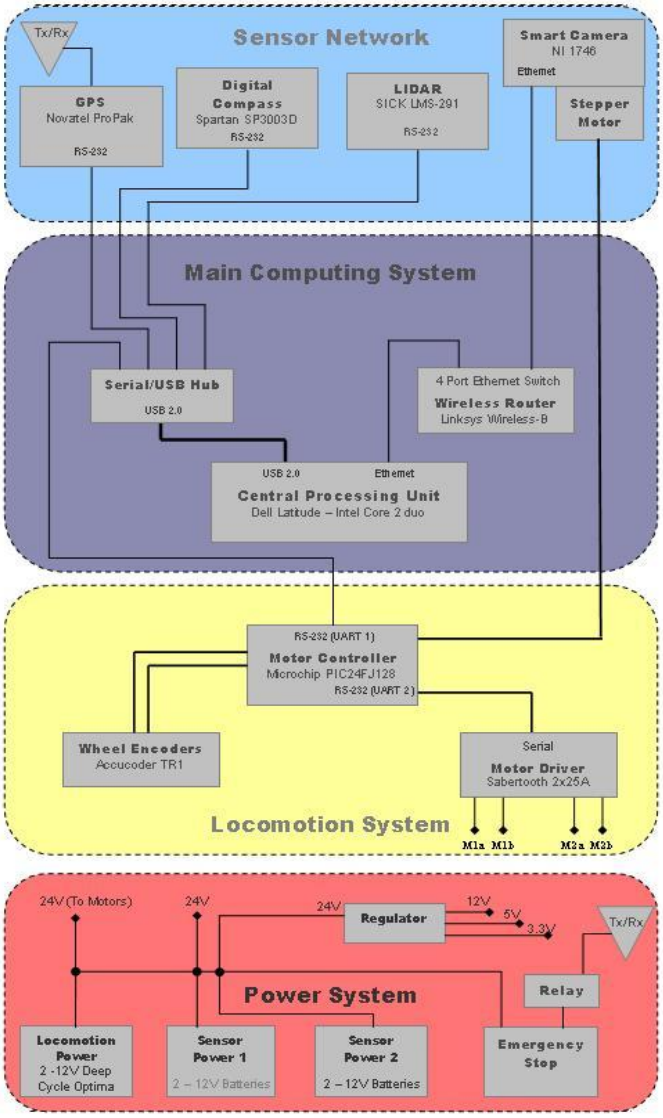


Figure 1 - System Block Diagram

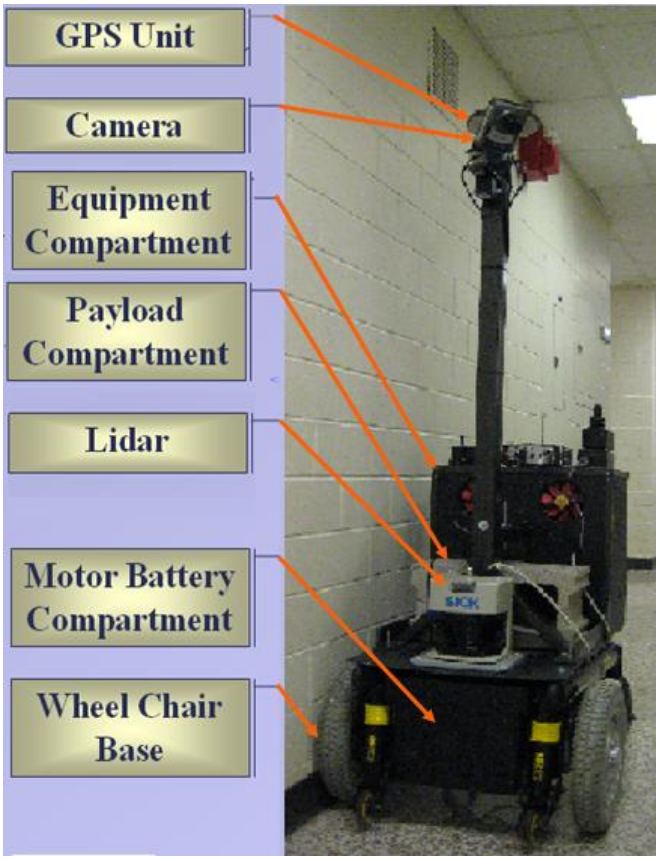


Figure 2 - Zipp-E

The third semester involved building the current robot shown in Figure 2 and writing all of the code required for the operation of Zipp-E. LabVIEW was used for most of the software except for the microcontroller which was written in C code. This completes the design planning process for Zipp-E.

Path-Finding Navigation / Waypoint Navigation System

In the main control software, there is a drop down list that allows the user to do one of four things;

1) Run Lane Detection Program
2) Run Waypoint Competition Program
3) Enter/Exit Manual Drive Mode
4) Initialize the Current Position and History Maps

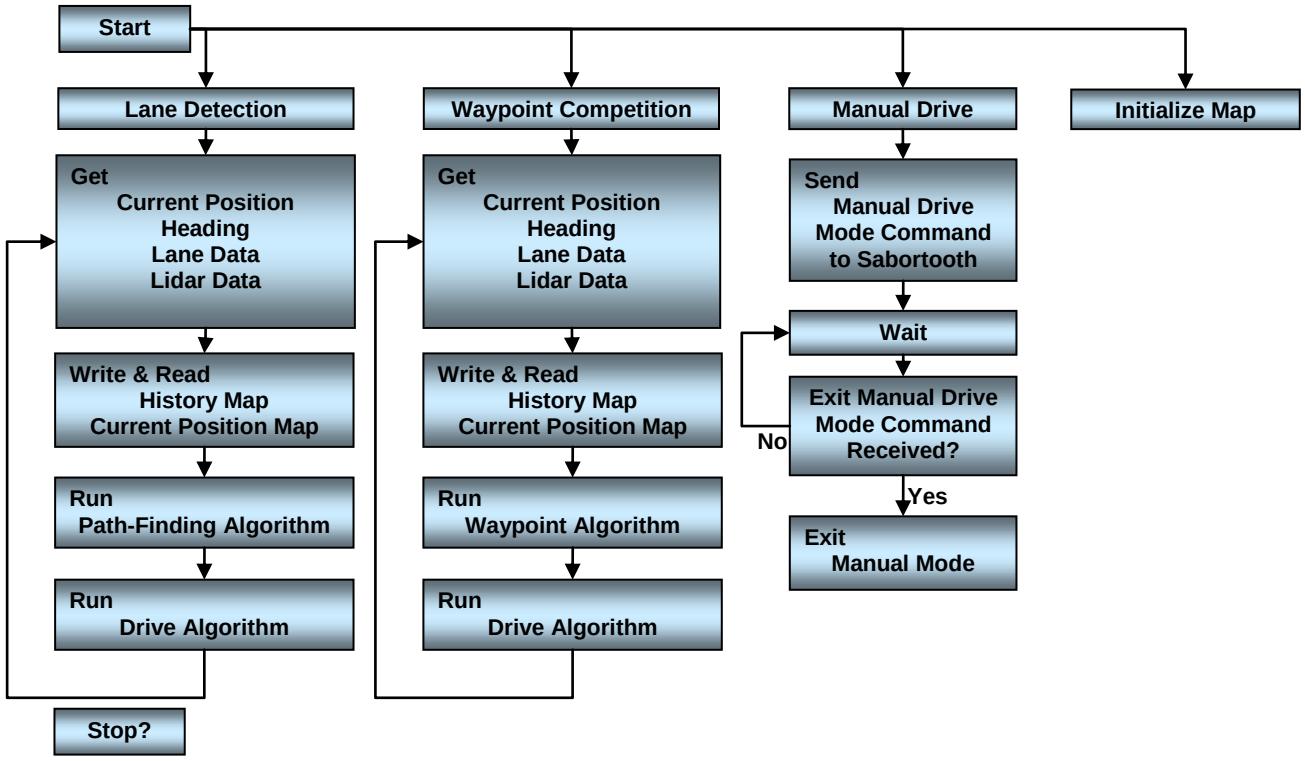


Figure 3 - Main Control Software Flow Chart

The Lane Detection and Waypoint Competition Programs are very similar; the only difference is in the main Algorithms that determine the path the IGV will take. The following software flow charts shown in Figure 4 and Figure 6 will go through this part of the code in more detail. The first block in the Lane Detection and Waypoint Competition Programs is running the subsystem programs in sequential order to get the information from the GPS, Compass, Camera and Lidar systems for the current location, current heading, lane line data and obstacle data.

After this information is collected, the software increments to the next block and takes the collected data and writes it into the main History Map from which the current map gets only the information regarding the immediate 20 foot area around the IGV. After the current position map has been populated, either the path-finding or the waypoint algorithm is run depending on the program that was selected. After the correct path has been calculated, the Drive algorithm is run to drive the IGV forward. See Figure 7 for a detailed description of this algorithm. The Enter/Exit Manual Mode simply sends out the command to the PIC24 to enter manual mode for the Sabortooth. After the command has been sent, the code enters an endless while loop until the user intercedes and tells the program to exit manual drive mode. The Initialize the Current Position and History Maps initializes the maps so that the other programs may be run.

Path-Finding Algorithm

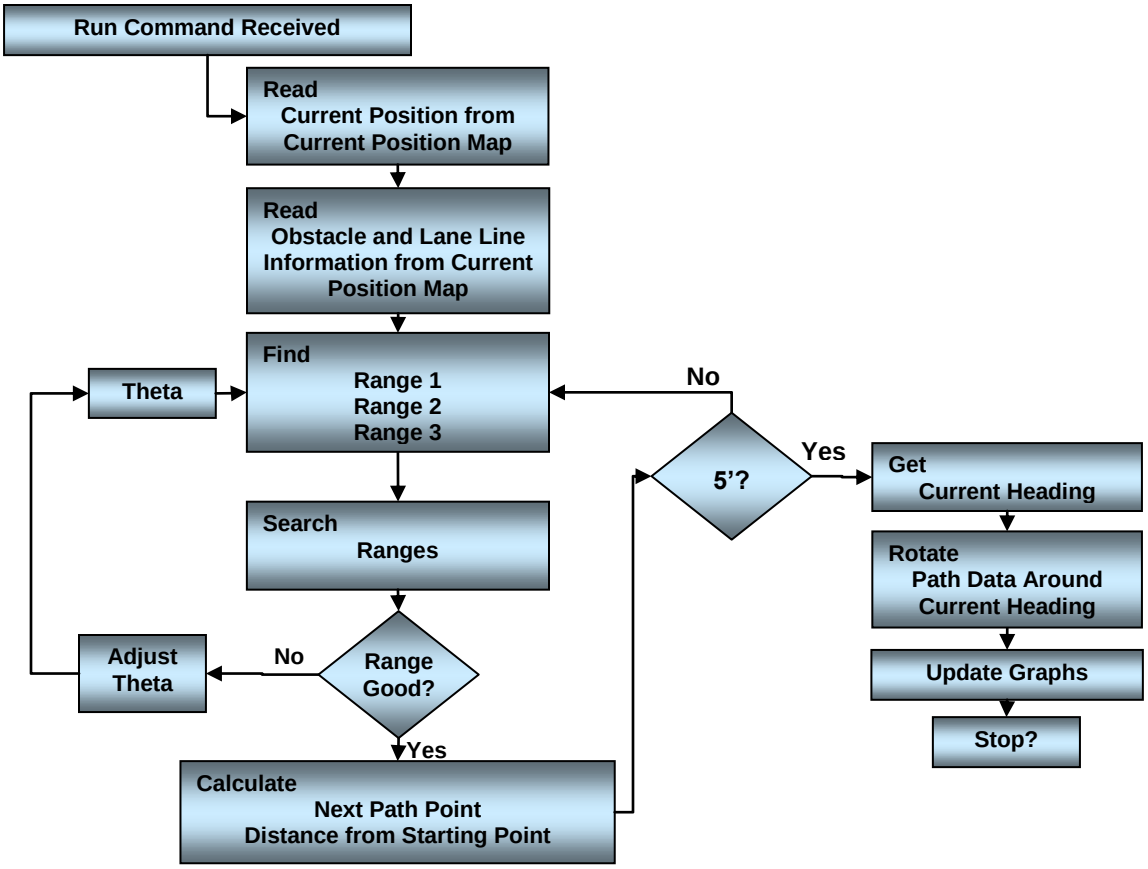


Figure 4 – Path-Finding Navigation Flow Chart

Once the run command has been received by the Path-Finding algorithm, the code reads in the current position that was written to the Current Position Map. Once the code has this starting position, then the obstacle and lane data (X, Y) coordinates is read into a single array. From this array, the angle and distances to those points are calculated from the current starting position with angle zero being the current heading regardless of its actual heading.

After the distances and angles have been calculated, this information is searched from all the points that are within ±30°s of the current heading. This data is then split into three different ranges with which only the shortest distance is taken on each angle that is looked at.

1) Range 1 – Theta to Theta - 30°s
2) Range 2 – Theta - 20°s to Theta + 20°s
3) Range 3 – Theta + 30°s to Theta

The first range that is searched is Range 2 for an open space to put the IGV through. If there is no area in this range that is large then or equal to 3 feet, then the next range searched is number 1, then three. What the code is doing while it is searching the ranges is it takes the furthest angles and starts looking for distances that are less then or equal to the following:

Distance = Shortest Distance + (Larges Distance – Shortest Distance)/4

As these angles move in, they are saved until all of the range has been search. Once the angles have been found, the law of Cosines is applied to find the distance between the two points that were found. See Figure 5 for a graphical representation of what has been described. If there is no open path found, the angle that is the current Theta is rotated by 30°s to the left and the searching is repeated. Again, if there is no clear path found in that the range, the current Theta is then rotated to the right by 60°s. After a good range has been found, then the next point in the path is calculated and then the process repeats itself.

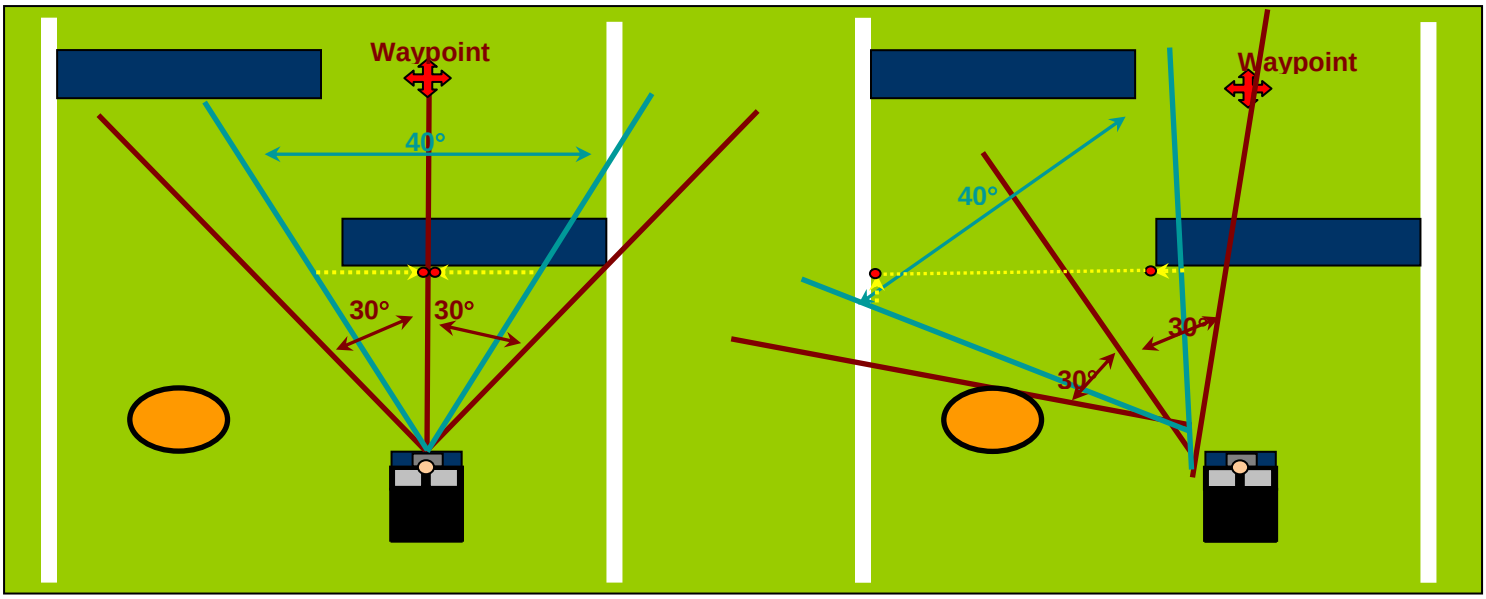


Figure 5 – Navigation Graphical Representation (Waypoint and Path-Finding)

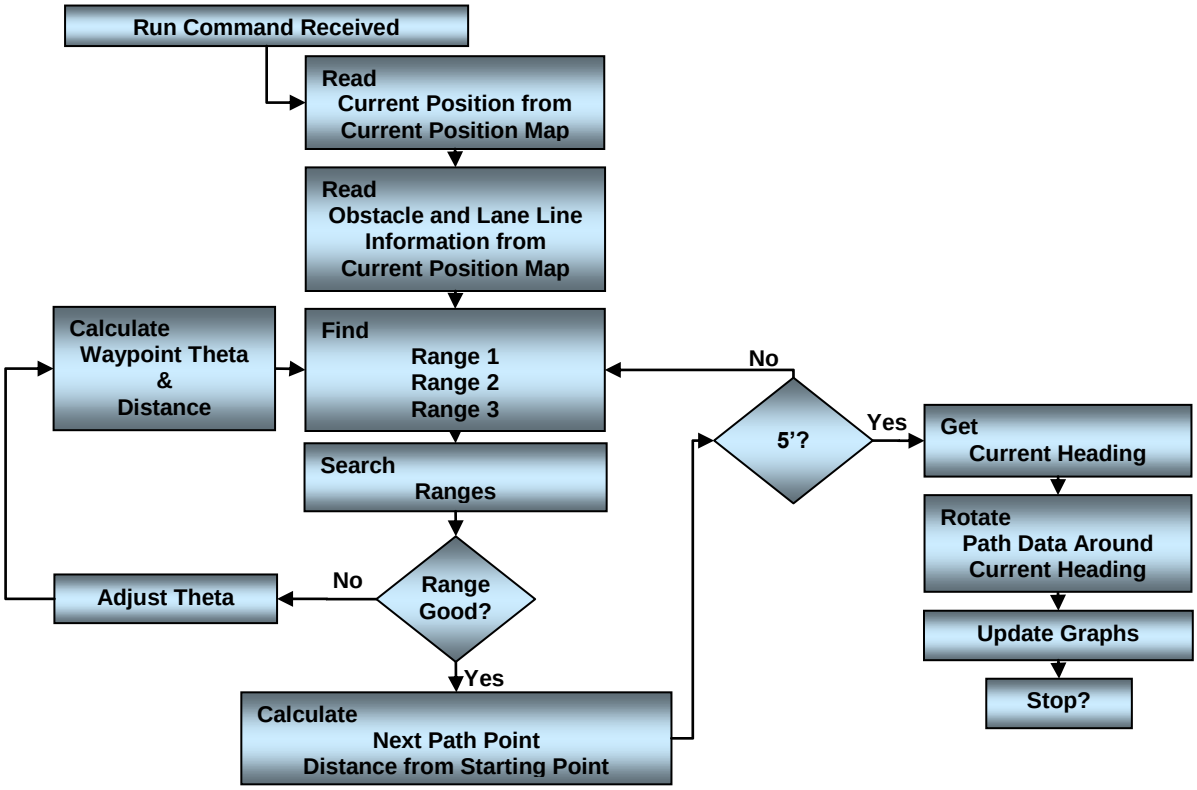


Figure 6 - Waypoint Navigation Flow Chart

Once the waypoint algorithm has received the run command, the code reads in the current position that was written to the Current Position Map. Once the code has this starting position, then the obstacle and lane data (X, Y) coordinate data is read into a single array. From this array, the angle and distances to those points are calculated from the current starting position with angle zero being the current heading regardless of its actual heading. This angle is then adjusted to the current point and the waypoint to see if there is a clear path there first.

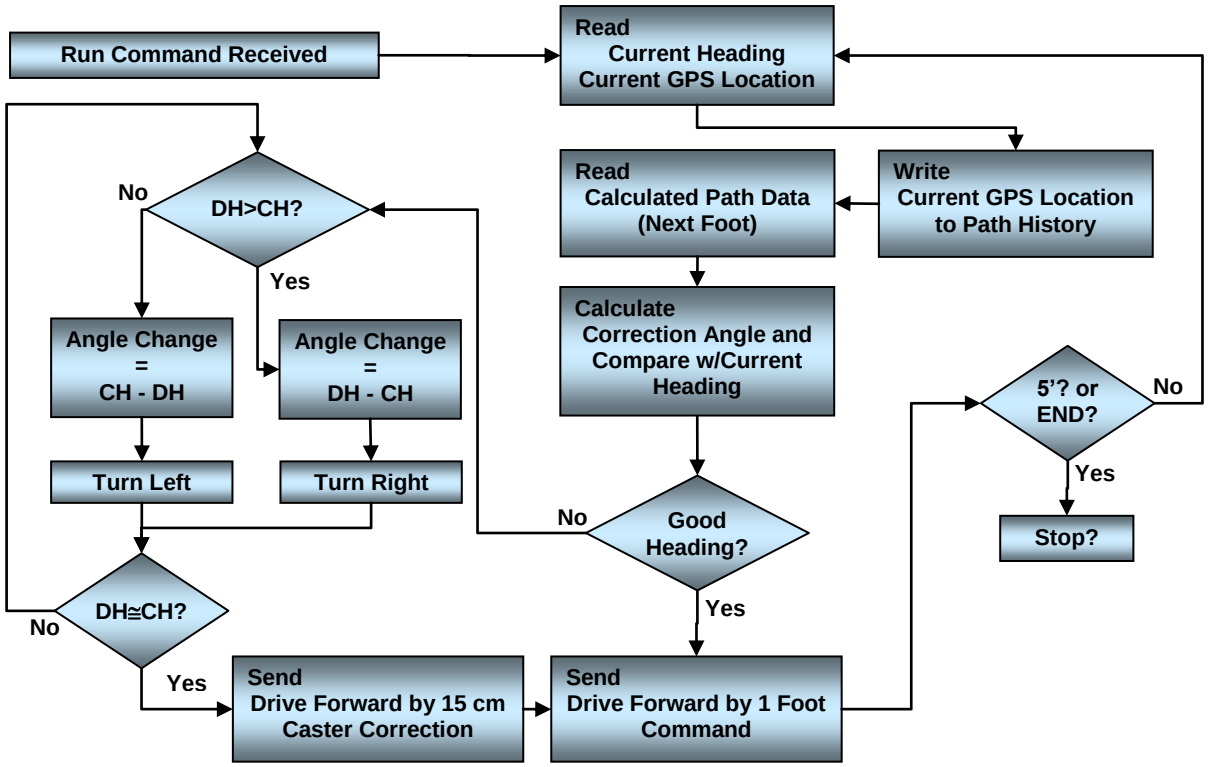


Figure 7 - Drive Navigation Flow Chart

After the Path has been calculated the Drive Algorithm is then run. The first thing that it does is reads in the current heading and GPS location. Once it has this information, it saves the GPS location into the history array. After the current location is saved, the code reads in the first foot of data from the path array. Then the code determines if a correction angle is required and which direction the IGV needs to turn. Once the heading correction has been made, the command to move forward one foot in that direction is sent. After the IGV has finished that movement, the process is repeated until the IGV has moved either five feet or as far as the data had been calculated. After the movement has completed, the next iteration is then started.

General Hardware/Other Systems

Electrical System

The power system consists of the following batteries:

- (2) 55A-hr 12V Batteries
- (3) 12A-hr 12V Batteries
- (1) 24V Regulator
- (1) 12V Regulator

Two 12A-hr batteries supply power to the all the main systems except for the motors. One 12 A-hr battery supplies power for the stepper motor and the two 55A-hr batteries connected in series to supply the motors and the driver with power. Power to the motors was isolated to avoid problems with noisy power due to switching.

Locomotion System

The locomotion system will be responsible for driving the intelligent ground vehicle. Ultimately the main computing system will be providing the appropriate information to the locomotion system or, looking at a lower level, the motor controller. The motor controller will interpret the transmitted data from the main computing system and signal the motor driver. The motor driver will then power the wheels of the ground vehicle. Wheel encoders as well as the main computing system (via global positioning, digital compass, and obstacle detection) will provide the motor controller with the desired information to move intelligently.

To communicate with the main computing system, Microchip's 16-bit programmable instructions controller will be used. The Explorer-16 development board will be used to interface with the PIC since the a serial communications port is already

integrated on to the board, as well as an LCD, temperature sensors and the ability to easily interface with all the I/O via PICtail Plus Daughter Board. The PIC itself was chosen to be Microchip's 16-bit microcontroller.

Since the PIC provides a myriad of I/O configurations we will be using this as the motor controller. The motor controller will be providing control information to the motor driver. The motor driver will be a dual 25A Sabertooth from Dimension Engineering. The 25 Amps per channel is why we chose this motor driver over designing our own. The wheelchairs motors were calculated to draw around 15 Amps of current, so 25 Amps provides enough source/sink and plenty of headroom. The Sabertooth also makes development much simpler by providing packetized serial communication support. In other words it does a lot of the dirty work for us already, so the integration will be much easier. Another feature that was welcomed was the support for an emergency stop, when implementing the packetized serial protocol.

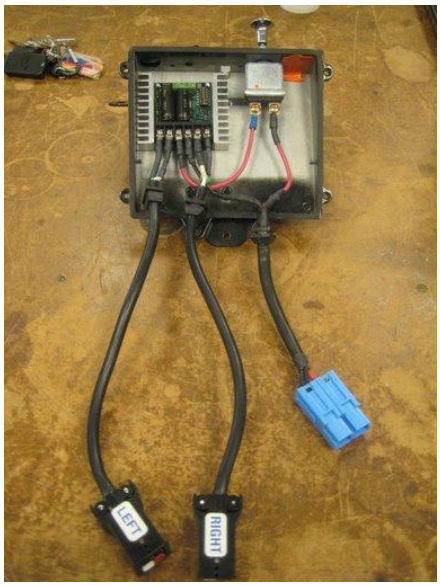


Figure 8 - Dual 25A Motor Driver

A few different forms will be providing the system with feedback during autonomous motion. One of which is local to the motor controller. Accu-Coders from the Encoder Products Company will provide the motor controller with direct feedback during motion. This way distance actually traveled may be logged by the PIC and passed to the main computing system when it is necessary.

Hardware Design

Logically handling all the information of the driver system will be handled by the 16-bit microcontroller. It has 2 UART communication lines that will both be utilized. One UART will be used to communicate with the main computing system and the other to communicate with the Sabertooth motor driver. Figure 16 shows the hardware schematic that will be used for the locomotion system. In addition to handling the locomotion, the motor controller will also provide control for the lane detections stepper motor and camera, the outputs to drive system cooling fans when temperature thresholds are breached, updating status LED's to indicate it's current operation and provide an interface for manual drive via analog joystick.

The analog joystick was implemented by use of the analog to digital converter built-in to the microprocessor. Since the conversions are based on a reference of 3.3V and the voltage range of the analog joystick is 0V-5V, a voltage divider was designed to provide the appropriate quantization range of 0-1024. These analog voltages were then digitized and given to the appropriate method to determine which of the five states described above we were currently in and send the corresponding command to the motors via UART to the Sabertooth.

Output pins on the microprocessor control three cooling fans. When the control signal is high a Darlington pair is turned on and so is the fan. The status LED's are implemented to visually see the current state of operation. Below are examples of how operations are shown visually

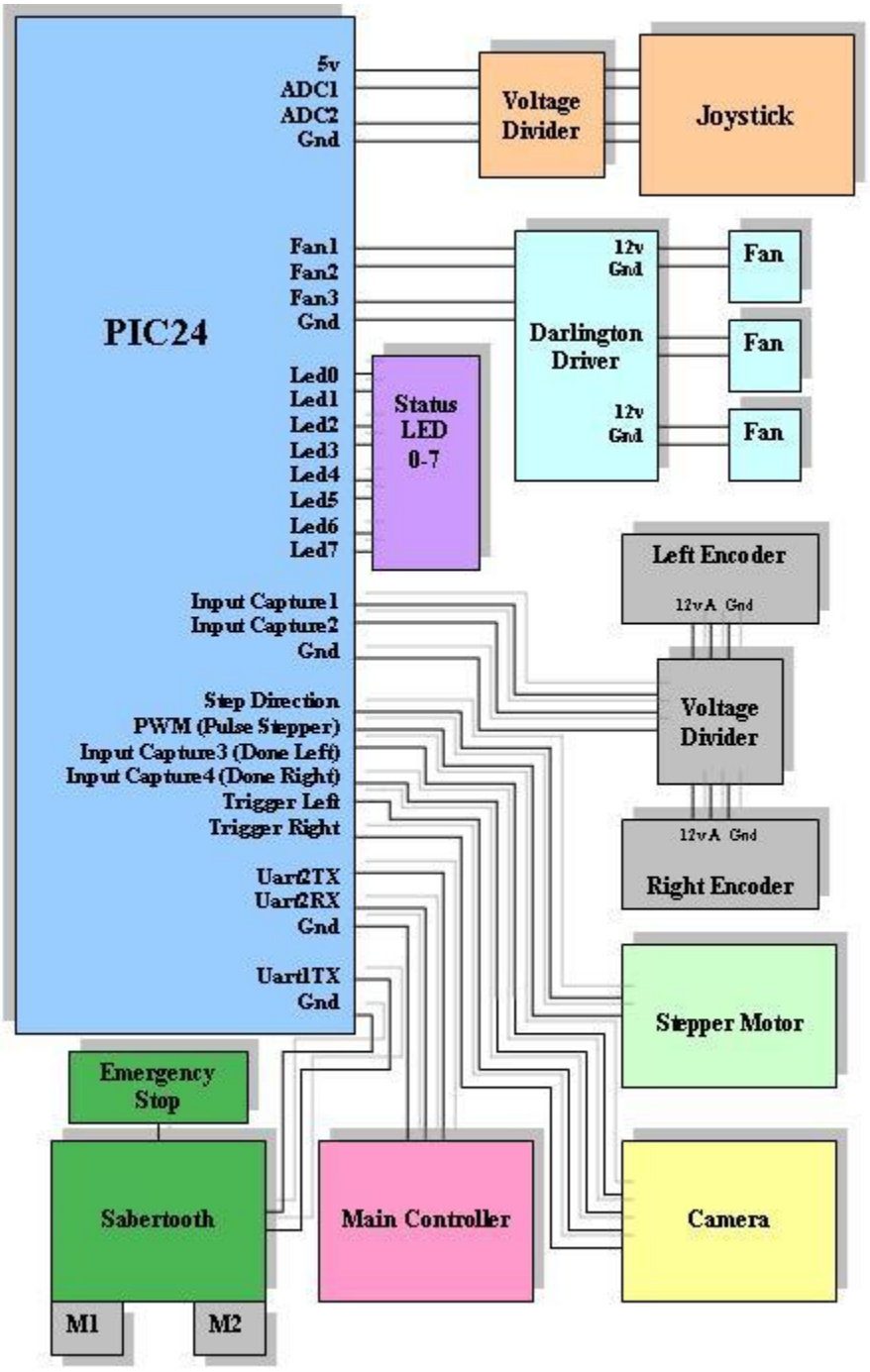


Figure 9 - Locomotion System Schematic

To control the motion of the stepped motor PWM is used to reliably step the motor at a desired frequency of roughly 80Hz. This output signal was then pulled up to 5V to provide the appropriate logic level for the stepper motor driver. To control the direction a standard output pulled is exercised high for clockwise and low for counter-clockwise. Other output pins are used to trigger pictures in the cameras state machine, one for each side (Left, Right). Two Input captures are also used to accept signals from the camera to synchronize when it is done processing an image on the left and the right. In order to interface with all of this I/O the PICtail Plus daughter board was used.

The Sabertooth will be configured to differentially drive the motors using the packetized serial protocol. Since we are using a microcontroller, it only requires two wires, the UART transmission line and a common ground with the PIC. The encoders are implemented using the input capture capability of the microprocessor, but first the encoder's 12V logic must be stepped down to 3.3V, which is a solid logic high for the microprocessor. This was achieved through use of a voltage divider implemented on our custom designed PCB. The encoders provide 500 voltage pulses per revolution, and from the wheel size each pulse

corresponds to 0.33cm distance traveled. So at the end of each movement period the overall distance traveled by each wheel can be calculated.

Software Design

The motor controller will communicate with the main the computing system via RS-232 UART protocol. Through this communication channel the main computing system will provide the direction for each motor, and a scaled value based on the desired angle at which to turn. The value is scaled based on the packetized serial protocol that will be used to communicate with the Sabertooth motor driver. Packetized serial mode uses TTL level RS-232 data to set the speed and direction of the motor. The packet consists of an address byte, command byte, data byte, and a 7 bit checksum. Packetized serial automatically detects the baud rate based on the first character sent, which must be 170.
The mixed mode command set will be used to control the motors. This is where mixed drive and turn commands are sent. The Sabertooth requires both valid drive and turn commands to be set before it will operate the motors. The PIC24 microcontroller instructs the motors to move forward or turn in a certain direction. The accelerate function slowly brings the motors up to the maximum speed as instructed, and then retain that speed and direction until the motor encoders notify the microcontroller that the vehicle has reached the desired destination. Once this notification is received, the microcontroller instructs the motors to stop.

The microcontroller will mostly be acting as a slave to the main system, since it is controlled by commands sent from the main system. The encoder output will be captured by utilizing the change of notification interrupt service routine. This way an interrupt will be generated every time the specified data line goes from high to low, giving us an easy way to count the pulses and compute the distance traveled. Almost Every Interrupt other then the UART receive, is turned on only when it is necessary for them to be used. This avoids erroneous line toggling or any other transients on the I/O lines when they are not in use.

The implemented watchdog timer is called upon to avoid system deadlocks by making all looping conditions dependent upon the timeout value set before the operation is called. This provides a form of time slicing and time deadlines as in a real-time operating system. Primarily this is used to avoid communication deadlocks with UART1 in the situation where the number of received bytes is under or over the expected value.

Electronics

Printed Circuit Board

Originally, the printed circuit board was intended to be for the stepper motor driver circuit alone; eventually, it expanded to include a multitude of different components including the PIC, stepper motor driver, motor encoders, smart camera, and the fans with Darlington driver. The following table explains the IO on the board.

Table 1 - PCB IO

Device		
PIC24	**Inputs**	**Description**
	ENC1/ENC2	The encoder logic outputs. They are stepped down from there default voltage of 12V to the 3.3V that the PIC can handle
	RET_L/RET_R	These inputs are used for synchronization between the PIC and camera. They tell the PIC that the camera is done processing the left and right images respectively, and it is ok to proceed forward.
PIC24	**Outputs**	**Description**
	TRIG_L/TRIG_R	These outputs of the PIC control when the camera will take a picture, and also, whether the camera is facing to the left or right so that the
	FAN_1/FAN_2/FAN_3	These outputs drive the bases of the darlington drivers to turn the fans on.
	CLK	The PWM signal that drives initiates steps on the motor controller. Each low-to-high transition equates to one step on the stepper motor.
	CW/CCW	Sets the turning direction of the motor. Logic hight for Clockwise, logic low for Counter-Clockwise
	5V	The 5V line from the Explorer 16 board is used to pull up the 3.3V logic lines on the PIC for use on the stepper motor driver, which requires 5V logic.

Encoders	Inputs	Description
	VIN/GND	Supplies 12V to power the Encoder
	Outputs	**Description**
	ENC	The pulse output of the encoder
Camera	**Inputs**	**Description**
	24V	Supplies Power to the Camera
	GND	Supplies a ground path for power, a ground reference for logic, and grounds the shielding on the camera cable.
	TRIG_L/TRIG_R	The logic inputs from the PIC, which are stepped up using a MOSFET as a pull up/down switch.
	Outputs	**Description**
	RET_L/RET_R	Logic inputs to the PIC, which are stepped down through voltage division.

The PCB was designed in PCB123, and the artwork for the PCB, unpopulated board, and fully populated board are shown below.

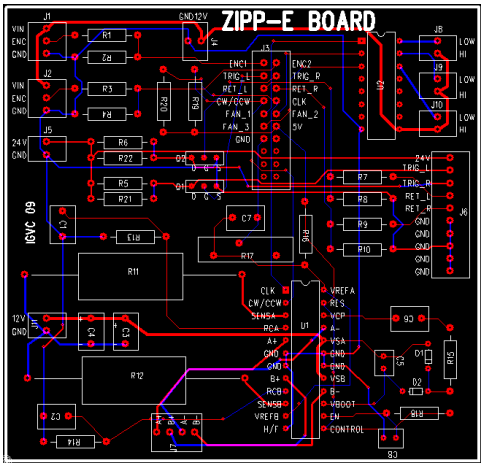


Figure 10 PCB Artwork

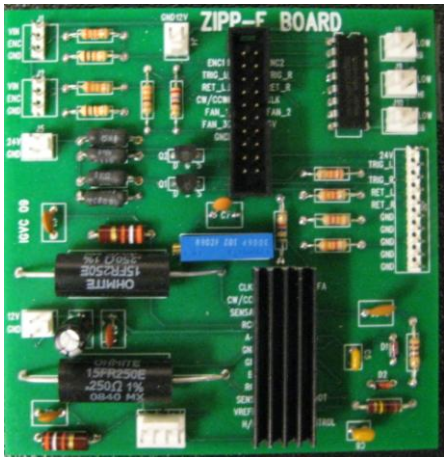


Figure 11 – Populated PCB

One of the main concerns with the implementation of the PCB was the stepper motor driver cause electrical or electromagnetic noise in the rest of the circuit. The first step in isolating the motor was to give it its own 12V rail. Also, the stepper driver circuit was separated, as much as possible, from the other components on the board by shifting all of ti to the bottom half of the board. Finally, as can be seen in the PCB artwork, the power path loop was kept to a minimum and the phase lines on the motor were overlapped; this was done to reduce current loops, and therefore, reduce EMI.

Lane Detection System

Technical Description

The Lane Detection System is implemented using a multitude of components both hardware and software including the National Instruments 1764 smart camera, the Surestep 17048 stepper motor, the STI L6228 stepper motor driver, dsPIC24 Microcontroller, National Instruments LABview, National Instruments Vision Builder AI, and C. The hardware components and implementation have not changed since the original design (with the exception of a more powerful camera mode 1764 instead of the 1722); the software system has been changed to better make use of the software available of the camera. A picture of the Lane Detection System is given in following Figure

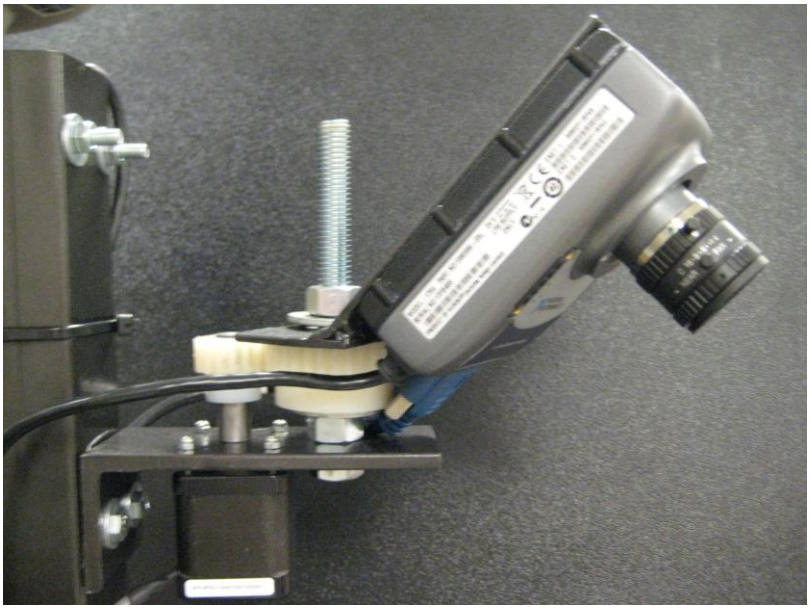


Figure 12 Lane Detection System

To overcome some limitations of the camera, a stepper motor is used to rotate the camera and two images are processed to affectively double the area of coverage. A detailed description of both the hardware and software systems involved is presented in the following sections.

Hardware Design

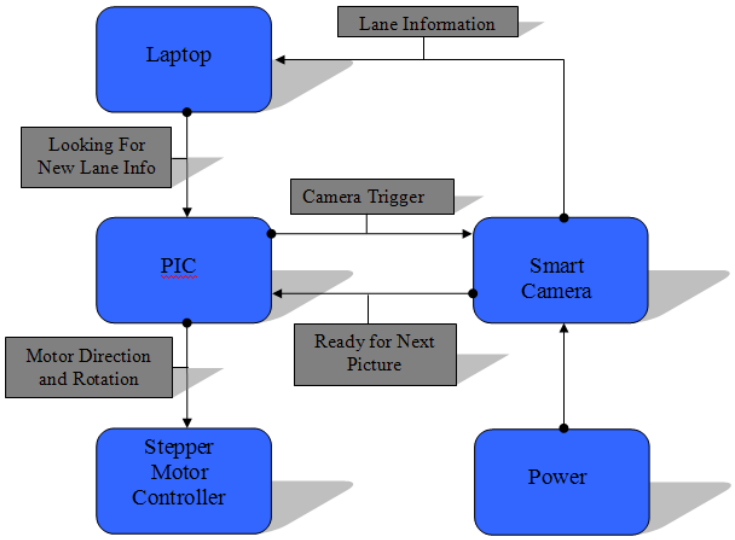


Figure 13 Hardware Block Diagram

Table 2 Lane Detection Hardware Modules

Module:	Lane Detection
Designer:	Adam Rich
Inputs:	Power, and Looking for New Lane Information

Outputs:	Camera Trigger, Motor Rotation and Direction, Ready for Next Picture, and Lane Information
Description	**Power:** Power is an input to the camera. The Camera runs of 24V D.C. with +20% -15% tolerance range, with a maximum current draw of 450mA for a max power dissipation of 10.8 W **Lane Information:** Lane location is an output of the smart camera and an input to the laptop. The lane locations are sent over the Ethernet connection via an FTP server to the laptop as a set of x,y coordinates of discrete points on the lanes. **Looking for New Lane Infromation:** Looking for new lane information is an output of laptop and an input to the Pic. This output lets the Pic know whether or not the system is currently looking for lanes, and therefore whether the Pic should perform a shutter trigger (take a picture). **Camera Trigger:** The shutter trigger is an output of the Pic and is communicated to the smart camera via the 24 pin cable that connects to the smart camera. When the shutter trigger is initiated by the Pic, the camera will take a picture. The 3.3V logic of the PIC is stepped up by the use of a MOSFET pull up/down network. **Ready for Next Picture:** This is an output of the camera that informs the PIC that the camera is done processing an image. The PIC waits for this return signal to avoid synchronization issues. **Pic:** The same Pic24 used for wheel chair motor control **Laptop:** Dell Latitude Laptop, used for main system control **NI Smart Camera:** The NI 1764 camera

Here a description of the camera system block diagram is given. The laptop communicates to the PIC via serial and initiates the new lane information process. The PIC starts by initiating a picture; then waits for the ready for next picture return on the camera. When the return line goes high, the PIC moves the camera to left through the stepper motor, and then, once the camera has been moved, the PIC initiates another trigger. After the second image is processed, the PIC will move the camera back to the original position. Finally, the camera sends the lane information back to the laptop for further processing.

Software Design

The goals of the lane detection software are to efficiently and effectively recognize white lines in the image, calculate distances, adjust the lines to the same reference point, and control the stepper motor to accurately and reliably move the camera. To accomplish this, the software system utilizes three software languages. The stepper motor and logic returns from the camera are controlled by the PIC and are therefore written in C (described in locomotion section). The image processing that is done on the camera for lane detection uses National Instruments Vision Builder for Automated inspection. Once the camera obtains lane information, the data is sent over the Ethernet connection via a FTP server (open source baby ftp was used) and saved into a text file. Next, National instruments LabVIEW is used to read the lane data out of the text file. Once the data is in LabVIEW, linear interpolation is preformed to fill in the gaps between points. Finally the data points are quantized to 5cm blocks, which is the minimum resolution of the map.

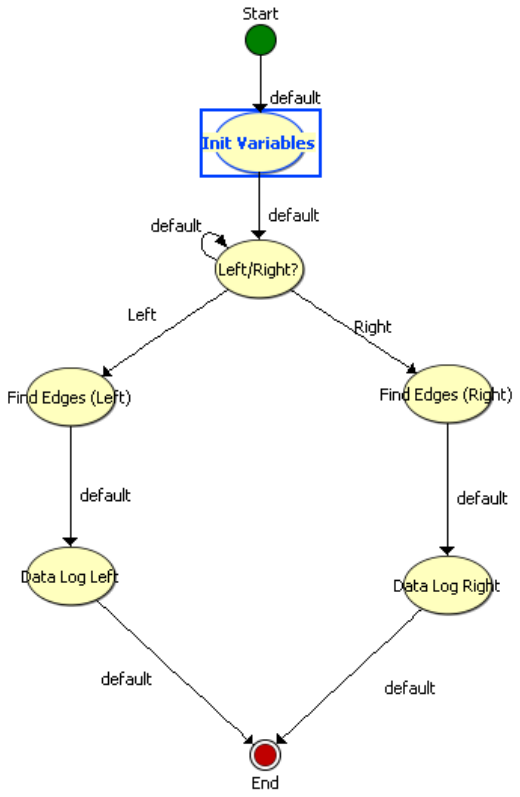


Figure 14 Lane Detection Software Block Diagram

Table 3 Image Processing Module

Module:	Image Processing Software
Designer:	Adam Rich
Inputs:	Trigger Left, Trigger Right
Outputs:	Done Processing Left, Done Processing Right, Lane Line Right, Lane Lines Left
Description	**Trigger Left**: Reads the IO line from the PIC and moves the state machine out of the Left/Right? state and into the Find Edges Left **Trigger Right**: Reads the IO line from the PIC and moves the state machine out of the Left/Right? state and into the Find Edges Right **Done Processing Right:** A return command form the data log right sate that tells the PIC that the camera done processing the image and ready to move. **Done Processing Left:** A return command form the data log right sate that tells the PIC that the camera done processing the image and ready to move. **Init Variables:** This states initializes all the variables back to there default values and moves to the Left/Right? state when done. **Left/Right?:** This is a wait state. The software reads the input lines and waits for either a trigger left of trigger right command from the PIC. The camera will move into the find edges left or find edges right state from the trigger left or trigger right command respectively **Find Edges Right/Find Edges Left:** This state performs all of the image processing. First, the software will take a picture. The image is then calibrated to remove the distortion from being angled instead of directly facing the ground. The software then performs edge detection along a line for 9 different lines spaced evenly across the screen. **Data Log Left/Data Log Right:** This state takes the points found from the find

		edges right/left state and saves them in a text file.	

To detect lane lines, the software scans across lines in the image. The pixel intensity (overall brightness) is measure. A large jump in pixel density represents a sharp jump from dark to light. The following image was taken from the smart camera and illustrates the edge detection and the edge profile.

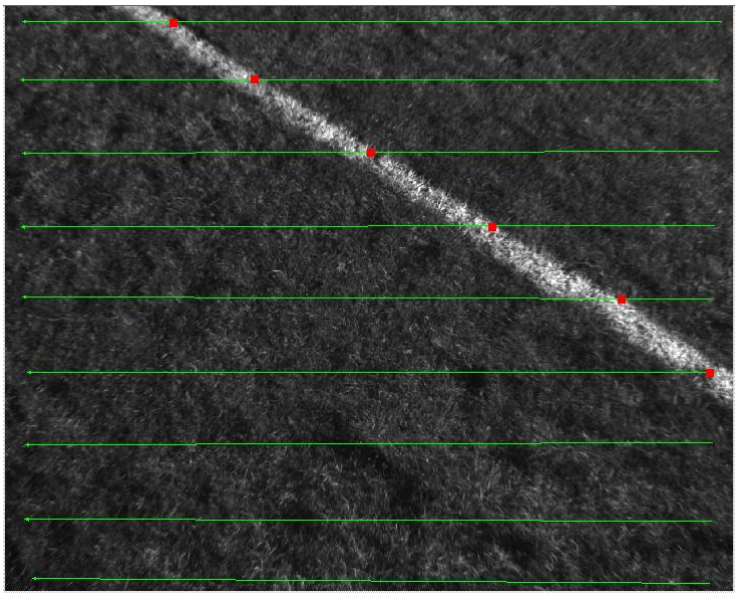


Figure 15 Smart Camera Image

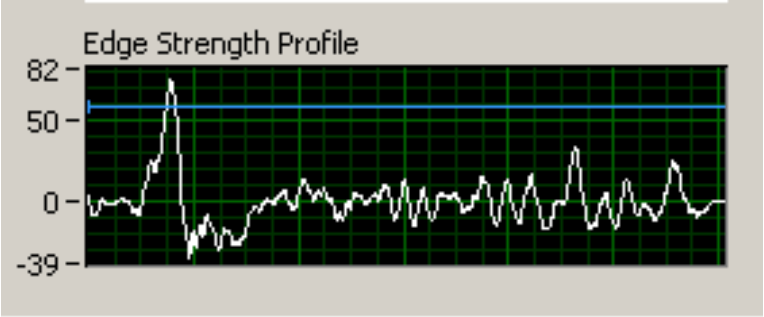


Figure 16 Edge Polarity Profile

In Figure 37, the edge detection is implemented on each of the green lines (9 in total). The red blocks represent the locations in which the software has detected a large enough discontinuity to read it as a lane line. Figure 36 shows the edge detection profile along one of the lines in the previous picture. The edge detection software is calibrated such that it looks at a large enough piece of the image (kernel size) and looks only for large discontinuities. The blue line represents the minimum edge strength discontinuity for the software to detect it as a lane line. As can be seen, the software filters out the lower intensity transitions.

Obstacle Detection/Avoidance System

Technical Overview

Obstacle detection system hardware consists of the Lidar sensor, two 12Vdc batteries, placed in series, to provide power. An RS-232 cable will then connect the Lidar directly to the system laptop (main processor) where all of the software functions will be implemented. The following block diagram depicts the basic setup of the obstacle detection hardware.

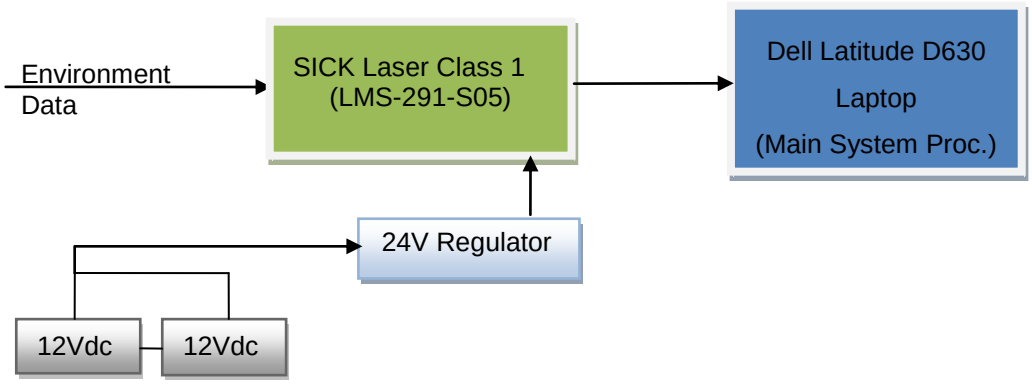


Figure 17: Obstacle Detection Hardware Block Diagram

Configuration of the Lidar system settings will be handled in the following manner in order to adequately meet the needs of the vehicle naviga incrementing (allows maximum 361 distance values). These settings can be examined using the software that was provided with the Lidar. The software creates a digital image of the sensor data and outlines the maximum distance at each point of the viewing field with a green line. The resulting image of one such scan test is provided below.

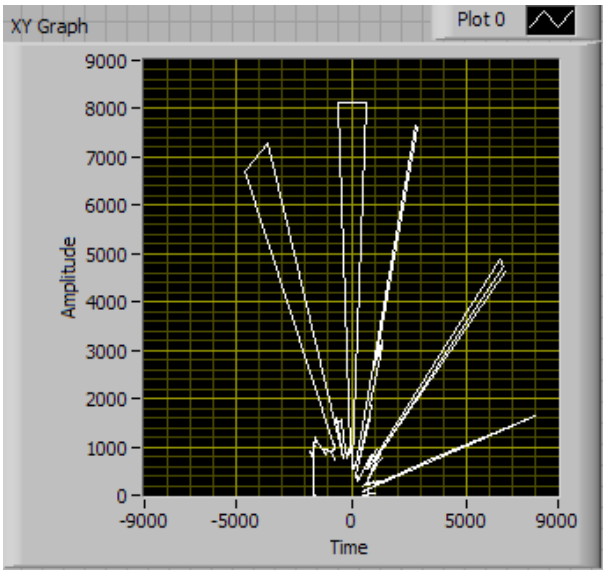


Figure 18: Lidar Software Scan Sample Data and Graph (8m range)

The communication between the Lidar and the main system processor running LabView is handled via an RS232 (DB9) connection and UART transmission. Upon request of object detection data by the system software, the Lidar provides 2 bytes of data for each distance value that is recorded calculated for an object in the field of view. At the 1° laser pulse increment setting, the maximum number of distance values sent is limited to 361 (722 bytes). Given this information, it is not possible to generalize the maximum number of obstacles that are detectable at any one time, because the size (width) of the object has a direct effect on how many distance values are necessary to accurately describe its position.

Costs (Hardware Only)

Qty	Part Num.	Description	Cost (ea.)	Total Cost
5	C320C104K5R5HA	100nF	$0.15	$0.75
5	C320C102K5R5HA	1nF	0.13	0.65
3	C320C224K5R5HA	220nF	0.43	1.29
2	C320C562J1G5TA	5.6nF	0.43	0.86
2	C320C103K5R5HA	10nF	0.13	0.26
2	C320C683K5R5HA	68nF	0.22	0.44
5	1N4148-B	Small Signal Diode	0.03	0.15
3	15FR250E	Current Sense Resistor	1.53	4.59
4	RC1/2393JB	39k 1/2 W 5% Resistor	0.36	1.44
4	RC1/4104JB	100k 1/4 W 5% Resistor	0.20	0.80
2	30BJ500-5.1K	5.1k /14W 5% Resitor	0.29	0.58
2	RC1/4153JB	15k /14W 5% Resitor	0.29	0.58
1	L6228N	Drivers DMOS Stepper Motor	9.21	9.21
1	STP-MTR-17048	2 Phase Steper motor	19.00	19.00
1	43020-0400	4 Circuit connector	0.36	0.36
1	780403-01	1742 Smart Camera, 640 x 480, 60 fps	2,249.10	2,249.10
1	197818-05	17xx 5m Pigtail breakout cable	26.10	26.10
1	780024-01	C-Mount Lenses, 8mm, Computar	188.10	188.10
1	780240-01	17XX Series Smart Camera Panel Mount Kit	26.10	26.10
1	182219-05	E1 Ethernet Cable, Twisted-pair, 5 m	18.00	18.00
1	777859-09	LabVIEW Vision Development Module	899.75	899.75
2	TD62002APG(5,J,S)	Toshiba 7 Channel Darlington Driver	0.89	1.78
3	A2368	Thermaltake 120mm Case Fan - Retail	6.99	20.97
2	10425	Dynamat Xtreme Wedge Pack 18" x 32" Sheet	14.99	29.98
1	Explorer-16	Microchip Development Board (Donated)	0.00	0.00
1	PIC24FJ128GA010	16-Bit microcontroller (Donated)	0.00	0.00
2	Sabertooth2x25	Dimension Engineering Dual 25A motor driver (Donated)	0.00	0.00
2	DE-SWADJ3	3A 25W Switching Voltage Regulator (Donated)	0.00	0.00
1	Vreg Breakout	Regulator Breakout Board (Donated)	0.00	0.00
1	SP3003D	Spartan SP3003D Digital Compass w/Dev kit (Donated)	0.00	0.00
1	ProPakV3	Novatel GPS Reciever w/L1,L2 antenna (Donated)	0.00	0.00
2	TRI	Accu-Coder wheel encoders (Donated)	0.00	0.00
1	LMS-291-S05	SICK Lidar laser class 1 (Donated)	0.00	0.00
1	2400P	Emergency Stop Button (Donated)	0.00	0.00
1	653-G2RL-24-DC5	K1 Relays (Donated)	0.00	0.00
1	509-RK-433-RC	Radiotronix Wireless Reciever (Donated)	0.00	0.00
2	D34 Yellow Top	Optima Deep Cycle Batteries (Donated)	0.00	0.00
2	547-PS-12180F	Lead Acid Batteries (Donted)	0.00	0.00
1	D630	Dell Latitude laptop (Donated)	0.00	0.00
1	Ranger-x	Invacare Wheelchair Base (Donated)	0.00	0.00
			Total	$3,500.84