

Line Detection and Lane Following for an Autonomous Mobile Robot

Andrew Reed Bacha

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in
partial fulfillment of the requirements for the degree of

Master of Science
In
Mechanical Engineering

Dr. Charles F. Reinholtz, Chairman
Dept. of Mechanical Engineering

Dr. Alfred L. Wicks
Dept. of Mechanical Engineering

Dr. A. Lynn Abbott
Dept. of Electrical Engineering

May 12, 2005
Blacksburg, Virginia

Keywords: Computer Vision, Autonomous Vehicles, Mobile Robots, Line Recognition, Image
Processing

Line Detection and Lane Following for an Autonomous Mobile Robot

Andrew Reed Bacha

ABSTRACT

The Autonomous Challenge component of the Intelligent Ground Vehicle Competition (IGVC) requires robots to autonomously navigate a complex obstacle course. The roadway-type course is bounded by solid and broken white and yellow lines. Along the course, the vehicle encounters obstacles, painted potholes, a ramp and a sand pit. The success of the robot is usually determined by the software controlling it.

Johnny-5 was one of three vehicles entered in the 2004 competition by Virginia Tech. This paper presents the vision processing software created for Johnny-5. Using a single digital camera, the software must find the lines painted in the grass, and determine which direction the robot should move. The outdoor environment can make this task difficult, as the software must cope with changes in both lighting and grass appearance.

The vision software on Johnny-5 starts by applying a brightest pixel threshold to reduce the image to points most likely to be part of a line. A Hough Transform is used to find the most dominant lines in the image and classify the orientation and quality of the lines. Once the lines have been extracted, the software applies a set of behavioral rules to the line information and passes a suggested heading to the obstacle avoidance software. The effectiveness of this behavior-based approach was demonstrated in many successful tests culminating with a first place finish in the Autonomous Challenge event and the \$10,000 overall grand prize in the 2004 IGVC.

Acknowledgments

Both the work in this paper, and my success throughout my college career wouldn't be possible without the help and support of several people. First I would like to thank my parents for supporting me throughout college. I would also like to thank my roommates for providing the stimulating nerd environment for true learning. I also couldn't have made it without the support of Caitlin Eubank, who would still love me after spending the weeks before competitions with robots rather than her.

This paper wouldn't have been possible without the hard work of the 2004 Autonomous Vehicle Team. You guys built two of the best designed and reliable vehicles that Virginia Tech has fielded in the IGVC. It is a great pleasure to develop software on a robot and never have to worry about wiring problems or wheels falling off. I would especially like to thank Ankur Naik, who programmed alongside me in several of our robotic projects such as the IGVC and the DARPA Grand Challenge. You provided the inspiration for several of the ideas in this paper as well as many Labview tips.

Finally, I would like to thank my advisor, Dr. Reinholtz, for his enthusiasm and devotion to the many senior design projects here at Virginia Tech. Much of my knowledge of robotics and useful experience comes from these projects.

This work was supported by Army Research Development Engineering Command Simulation Technology Training Center (RDECOM-STTC) under contract N61339-04-C-0062.

Table of Contents

Chapter 1 – Background	1
1.1 Introduction to the IGVC	1
1.2 Autonomous Challenge Rules	2
1.3 Past Performance	3
1.4 Examining Johnny-5	6
Chapter 2 – Software Design	10
2.1 Labview	10
2.2 Overview and Implementation	11
2.3 Image Analysis	13
Chapter 3 – Image Preprocessing	16
3.1 Conversion to Grayscale	16
3.2 Brightness Adjustment	21
3.3 Vehicle Removal	23
Chapter 4 – Line Extraction	24
4.1 Thresholding	24
4.2 Line Analysis	26
Chapter 5 – Heading Determination	30
5.1 The Behavior Approach	30
5.2 Decision Trees	31
5.3 Shared Calculations	32
5.4 Individual Behaviors	33
Chapter 6 – Testing, Performance, and Conclusions	36
6.1 Computer Simulation	36
6.2 Preliminary Vehicle Testing	38
6.3 Performance at Competition	40
6.4 Conclusions	42
6.5 Future Work	43
References	46
Appendix A – Sample Images	47
Vita	51

List of Figures

1-1	Johnny-5 competing in the 2004 IGVC.....	2
1-2	Placement of obstacles.....	2
1-3	Close up of Johnny-5.....	6
1-4	The bare chassis of Johnny-5.....	7
1-5	The electronics box from Johnny-5.....	8
1-6	The placement of sensors on Johnny-5.....	8
2-1	Labview block diagram.....	10
2-2	Labview front panel.....	11
2-3	Autonomous Challenge software structure.....	12
2-4	Labview block diagram.....	13
2-5	Flow diagram of image analysis software.....	14
2-6	Results of each step of the image analysis.....	15
3-1	A sample image of the IGVC course.....	17
3-2	Sample image converted to grayscale.....	17
3-3	The sample image from Figure 3-1 is converted to grayscale.....	18
3-4	The sample image is converted to grayscale.....	19
3-5	An image with noise.....	20
3-6	An image containing patchy brown grass.....	21
3-7	The average pixel intensity per row of the sample image.....	22
3-8	Subtraction filter used to account for brighter tops of images.....	23
3-9	The body of Johnny-5 is removed from an acquired image.....	23
4-1	Comparison of (a) a sample course image.....	25
4-2	Comparison of (a) a course image containing barrels.....	25
4-3	The steps of line detection process.....	27
4-4	The image from Figure 4-1a with (a) Gaussian noise added.....	27
4-5	A line is fit to a set of points.....	28
4-6	Parametric polar form of a straight line using r and θ	29
5-1	Decision tree executed when a line is detected.....	31
5-2	Decision tree executed when at least one side of the image.....	32
5-3	Vehicle coordinate frame with desired heading shown.....	33
5-4	A break in a line causes the right side line to appear.....	35
6-1	Screen capture of the simulated course.....	37
6-2	Images acquired from the camera.....	38
6-3	The position of the lines in Figure 6-2a.....	39
6-4	A yellow speed bump on the IGVC practice course.....	41
6-5	Situations where the vehicle might reverse course.....	43
6-6	The software can fail to detect a break in a line.....	44

List of Tables

1-1	Overview of recent VT entries.....	4
1-2	Overview of sensors.....	9
5-1	Output and factors of “2 lines, 1 horizontal” algorithm.....	34
5-2	Output and factors of “Both Horizontal” algorithm.....	34
6-1	Simulator configuration.....	37
6-2	Camera configuration used in NI MAX.....	39
6-3	Computation time for each software step.....	40

Chapter 1 – Background

The work presented in this paper was developed to compete in the Intelligent Ground Vehicle Competition (IGVC). Student teams from Virginia Tech have entered the IGVC since 1996, evolving the vehicle platforms as well as software algorithms to improve results. This chapter covers the goals and rules of the IGVC as well as reviewing Virginia Tech's previous vehicle entries along with a current vehicle, Johnny-5.

1.1 Introduction to the IGVC

The Intelligent Ground Vehicle Competition has been held annually by the Association for Unmanned Vehicles Systems International (AUVSI) since 1993. This international collegiate competition challenges student to design, build and program an autonomous robot. The competition is divided into three events: the Autonomous Challenge, the Navigation Challenge, and the Design Competition. The Autonomous Challenge involves traversing painted lanes of an obstacle course, while the Navigation Challenge focuses on following GPS waypoints. The Design Competition judges the development of the vehicle and the quality of the overall design. Previous years of the competition have included other events focusing on different autonomous tasks such as a Follow the Leader event, which challenged an autonomous vehicle to follow a tractor. Only the Autonomous Challenge will be presented in detail in this paper since it is the target of the presented research.

Virginia Tech has been competing in the IGVC since 1996 with interdisciplinary teams consisting of mechanical, computer and electrical engineers. Many of the students compete for senior design class credit, but the team also includes undergraduate volunteers and graduate student assistants. The 2004 IGVC was another successful year for the Virginia Tech team, with one of Virginia Tech's entries, Johnny-5, winning the grand prize and becoming the first and only vehicle from Virginia Tech to complete the entire Autonomous Challenge course. Johnny-5 navigating the Autonomous Challenge course is shown in Figure 1-1.



Figure 1-1: Johnny-5 competing in the 2004 IGVC

1.2 Autonomous Challenge Rules

The premiere event of the IGVC is the Autonomous Challenge, where a robot must navigate a 600 ft outdoor obstacle course. Vehicles must operate autonomously, so all sensing and computation is performed onboard. The course is defined by lanes roughly 10 ft wide, bounded by solid or dashed painted lines. It is known that the course contains obstacles, ramps, and a sand pit. However, the shape of the course and the locations of the obstructions are unknown prior to the competition [8]. The obstacles can include the orange construction barrels shown in Figure 1-1 as well as white one gallon buckets. These obstacles can be placed far from other obstacles or in configurations that may lead a vehicle to a dead end as shown in Figure 1-2. A winner is determined on the basis of who can complete the course the fastest. Completion time is recorded after time deductions are marked for brushing obstacles or touching potholes, which are simulated by white painted circles. In a majority of previous competitions, the winner was determined by which vehicle traveled the farthest, as no vehicle was able to complete the course. A vehicle is stopped by a judge if it displaces an obstacle or travels outside the marked lanes.

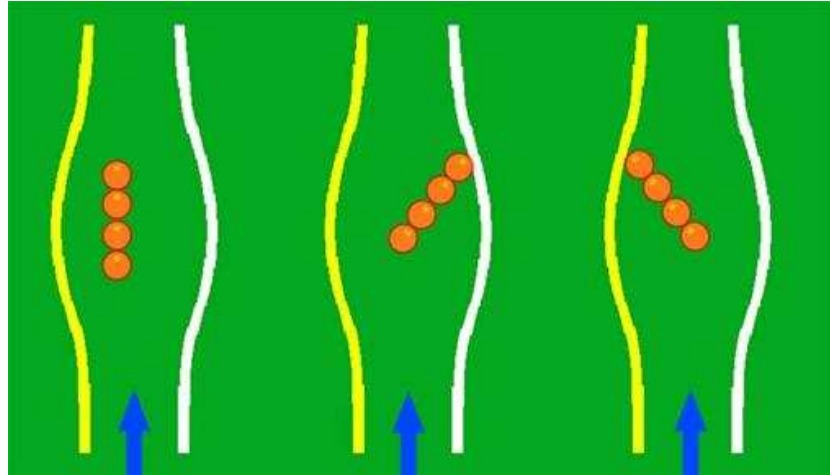












Figure 1-2: Placement of obstacles may lead vehicle into a dead end [8]

1.3 Past Performance

Since first entering the IGVC, Virginia Tech has done well in the Design competition, but was not as consistent in the Autonomous Challenge. A previous team member, David Conner, documented the results of vehicles previous to the 2000 competition [3]. The vehicles prior to the year 2000 were not successful on the Autonomous Challenge course, with problems attributed to camera glare, failure to detect horizontal lines, and sensor failure. The year 2000 however, marked the first success of Virginia Tech in the Autonomous Challenge. Subsequent years have proven equally successful, placing Virginia Tech in the top two positions, with the exception of the year 2002. This change in performance can be attributed to more reliable sensors (switching from ultrasonic sensors to laser rangefinders) and more advanced software. An overview of Virginia Tech's entries in the IGVC since 2000 is shown in Table 1-1.

Table 1-1: Overview of recent VT entries in the IGVC

	Vehicle / Results	Picture	Notes	Software / Problems
2004	Johnny-5 Auton. Chall: 1 st Design : 2 nd		Caster placed in front Gasoline generator Pentium IV Laptop	Behavior based heading Vision: Brightest Pixel Avoidance: Arc Path Completed course
	Gemini Auton. Chall: 1 st Design : 2 nd		2 DOF articulating frame Removable electronics drawer Pentium IV Laptop	Vision: Brightest Pixel Avoidance: Arc Path Exited course through break in line
	Optimus Auton. Chall: 14 th Design : 6 th		Returning Vehicle Integrated with Zieg's old camera mast	Vision: Brightest Pixel Avoidance: Arc Path Poorly calibrated Crossed lines Collided with barrels
2003	Optimus Auton. Chall: 1 st Design : 2 nd		Transforming wheelbase supported 3 or 4 wheels Removable electronics box Pentium III NI PXI computer	Vision: Brightest Pixel Avoidance: Arc Paths Exited course through break in line
	Zieg Auton. Chall: 2 nd Design : 3 rd		Modified Artimus frame Pentium III desktop computer	Vision: Brightest Pixel Avoidance: Vector Shared LRF not available for many runs, collided with barrels

2002	Biplaner Bike Auton. Chall: NA Design : 1 st		Innovative 2 wheel mobility platform Active camera stabilization	Navigation Manager Faulty DC-DC converter prevented vehicle from competing
	Daedalus Auton. Chall: NA Design : 2 nd		Smaller vehicle body Electronics organized in removable E-box Desktop computer	Navigation Manager Vector Field Histogram Blown motor amp prevented vehicle from competing
2001	Navigator Auton. Chall: 3 rd Design : 3 rd		2 Cameras Larger vehicle body Returning vehicle	Vector Field Histogram Path recording Problems with ramp Crossed lines
	Maximus Auton. Chall: 4 th Design : 2 nd		Differential drive at vehicle center Pentium III desktop computer	Navigation Manager Vector Field Histogram Motor control would build follow error, stopping the vehicle
	Artemis Auton. Chall: 2 nd Design : NA		Differential drive with caster Returning vehicle Pentium II laptop computer	Brightest Pixels, steered to center of lane Unpredictable results: Crossed lines Struck obstacles

Examining the past vehicles, it is clear that most development work went to redesigning the mechanical chassis and the electrical distribution systems on the vehicles, yet the software failures remained similar every year without much progress. By switching to brightest pixel based thresholding, software in the year 2003 was able to minimize glare interference, but still had problems with horizontal lines or partial lines. The development of the software presented in this paper was motivated by addressing these issues, and the 2004 results prove its success.

1.4 Examining Johnny-5

A well designed base vehicle platform was a large factor contributing to Johnny-5 winning the autonomous challenge in 2004. Both mechanical and electrical elements must be designed with great attention to detail for a successful platform. The chassis of the vehicle must be sturdy and support travel up to 5 mph, while also providing the mobility to make fast and sharp turns when reacting to obstacles. The power distribution system must provide the proper power to motors and sensors using reliable connectors and still be easy to access for debugging problems. The base vehicle platform of Johnny-5, shown close up in Figure 1-3, successfully executed all the design goals listed above.



Figure 1-3: Close up of Johnny-5, an entry in the 2004

The mechanical design of Johnny-5 refines the differentially driven designs used by past Virginia Tech teams. Johnny-5 uses two differentially driven wheels in the rear for propulsion and a caster wheel in the front for stability. Previous designs placed the caster in the back, causing vehicles to tip when executing a fast stop. Another improvement to stability is a basin

shaped interior that allows components to be mounted below the drive axle, lowering the center of gravity. These improvements can be seen in Figure 1-4, which shows the chassis of Johnny-5 without any components mounted. Fully loaded, Johnny-5 weighs 215 lbs with a 2.8 x 3.4 ft. footprint and a 5.9 ft tall sensor mast [16].



Figure 1-4: The bare chassis of Johnny-5

The electrical system uses a new approach, incorporating a gasoline powered generator that continuously charges two 12 V dry cell batteries. Adding the generator provides run times greater than eight hours. Longer run time makes testing and developing software easier as no additional batteries are needed and more tests can be completed in a single day. Like earlier vehicles, the electrical components of Johnny-5 are packaged in a single box, shown in Figure 1-5, creating an organized layout and allowing easy servicing.

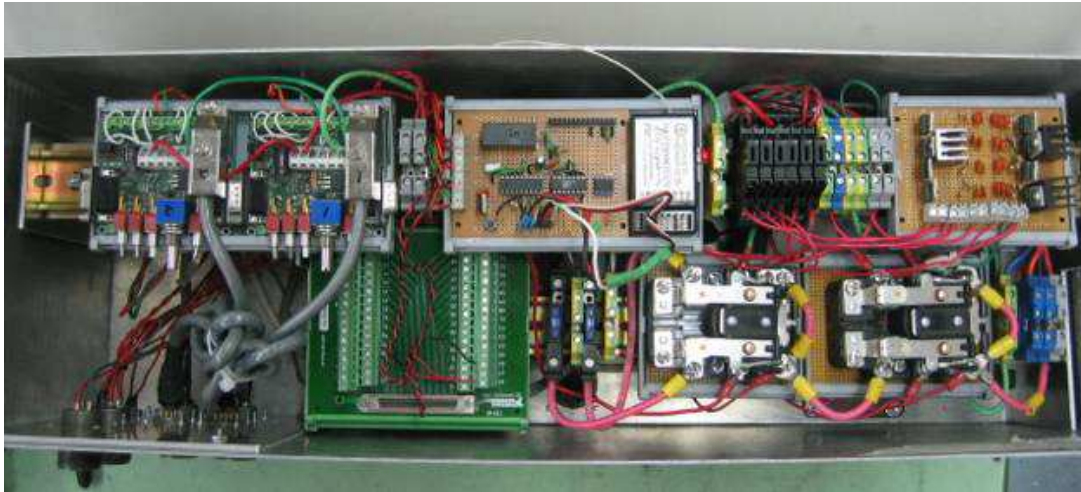


Figure 1-5: The electronics box from Johnny-5

To compete in the different competitions, Johnny-5 uses the sensor suite shown in Figure 1-6. All sensors and motors (with integrated controllers) are hooked directly to a laptop computer. Table 1-2 lists the various sensors with a short description of specifications and function. Only the camera and laser rangefinder are used for the Autonomous Challenge.

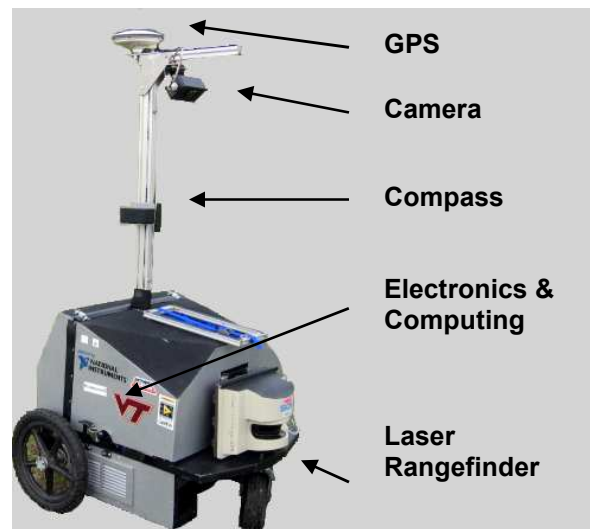






Figure 1-6: The placement of sensors on Johnny-5

Table 1-2: Overview of sensors used on Johnny-5

Sensor	Description	Picture
Unibrain Fire-I Board Camera	The camera provides a 640x480 RGB image at a rate of 15 Hz. The attached lens has a 94 degree diagonal field of view. A weatherproof enclosure was fabricated to protect the board.	
SICK LMS-221 Laser Rangefinder	The laser rangefinder scans in a horizontal plane and returns the distance to any obstacle at a resolution of 1 degree.	
PNI TCM2-20 Digital Compass	The digital compass senses vehicle heading relative to magnetic North. It is tilt compensated and can give pitch and tilt values up to 20 degrees.	
NovAtel ProPak-LB Differential GPS	This dual frequency GPS system is able to improve position information by using the Omnistar HP correction service. Using these corrections, 99% of all position readings will be within 15cm from the true position.	

Chapter 2 – Software Design

In most autonomous vehicle designs, software, rather than the actual vehicle platform is the main factor limiting vehicle capabilities. Starting in 2003, Virginia Tech's IGVC teams switched from developing software using C++ to National Instruments Labview. Labview (Laboratory Virtual Instrument Engineering Workbench) simplifies sensor interfacing and debugging, allowing development to focus on processing data. This chapter starts by discussing the advantages of using Labview as a development platform. The remainder of the chapter outlines the software controlling Johnny-5, discusses the implementation with Labview, and goes into more detail about the image analysis process.

2.1 Labview

All software was written using National Instruments Labview version 7.0, which creates programs using the G programming language. Labview was initially designed to allow engineers with little programming experience to interface with data acquisition systems and perform analysis [15]. This heritage led to the creation of a programming environment that simplifies communication with external devices and allows easy creation of graphical user interfaces with familiar mechanical shapes and knobs. Programs, known in Labview as “Virtual Instruments” or VIs, are written by laying elements on a block diagram and connecting inputs and outputs rather than writing lines of code. The inputs and outputs of the block diagram are linked to the controls and indicators on the user interface, called the front panel. An example block diagram is shown in Figure 2-1 with corresponding front panel shown in Figure 2-2.

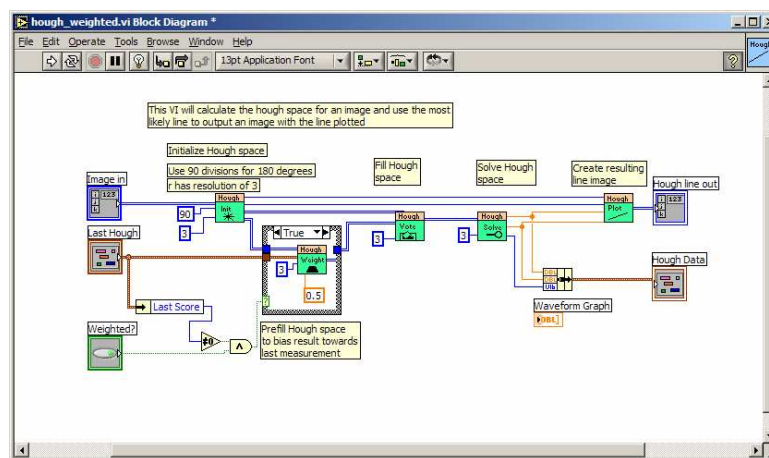


Figure 2-1: Labview block diagram

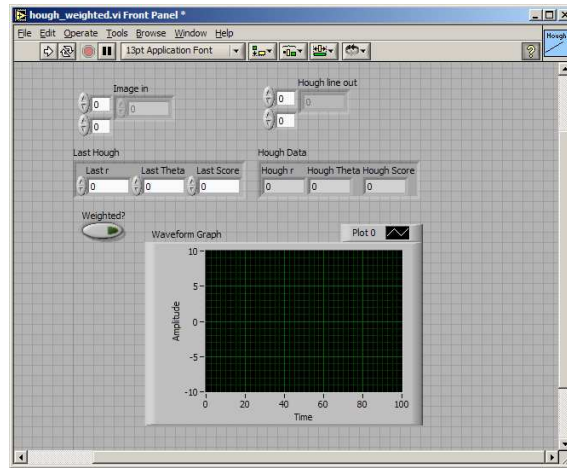


Figure 2-2: Labview front panel

From its simple beginnings, Labview has evolved into a fully functional programming language while still retaining an easy user interface. Programming with a block diagram approach allows the programmer to focus on dataflow, rather than syntax. This approach also makes it simple to create multithreaded parallel tasks since Labview will execute all subroutines that have all necessary inputs in parallel. Labview was created for data acquisition, making it easy to read sensors connected to the computer by serial or firewire. The biggest advantage to using Labview is its graphical nature, allowing users to see the state of any variables while the program is running. A rich library of indicators allows quick implementation of charts and visualization of images or arrays. Using these powerful indicators, debugging programs becomes easier, reducing development time.

Labview is well suited for image processing and machine vision applications, using an image processing library called IMAQ (IMage AcQuisition) [12]. Labview simplifies computer vision development by being able to display the image at any stage of processing with a single click of the mouse, and overlay information on images without affecting the image data. The IMAQ library contains several high level vision tools such as pattern recognition and shape analysis; however, none of these tools were used. Only the simple image manipulation functions such as resizing or extracting parts of images were used in the software.

2.2 Overview and Implementation

The software controlling Johnny-5 uses a reactive system model. Ronald Arkin defines a reactive robotic system as one that “tightly couples perception to action without the use of

intervening abstract representations or time history” [1]. Some important characteristics of a reactive system are that the actions of the robot are based on distinct behaviors, and only current sensor information is used. A behavior is the execution of a specific action in response to a sensor input to achieve a task [11]. During the heading determination part of the image analysis software, features of the detected lines are used as inputs for behaviors that determine where the vehicle should travel (discussed later in this chapter and in Chapter 5).

The software starts by collecting data from sensors, processes this data to determine a desired heading, and commands the vehicle motors to steer towards the heading. After commanding the motors, the process is repeated with no memory of past sensor data (the only exception is storing the last calculated heading for comparison). Any errors in the sensor data will be limited to the current decision cycle and will not propagate into future decisions. This feature of reactive systems is especially relevant since a camera is prone to error sources such as glare that will be consistent until the vehicle moves enough to reorient the camera lens.

The basic steps of the software used in the Autonomous Challenge are shown in Figure 2-3. The software repeats the steps of Figure 3 at a rate of 15 Hz. The process starts by acquiring a color image from the camera. Basic preprocessing is applied to the image, including brightness adjustment, removing the section of the image where Johnny-5 is visible, and converting to grayscale. The image is analyzed, extracting line information such as quality and orientation, and the program determines a heading to the center of the lane. This heading, along with obstacle data from a laser rangefinder, is sent to an obstacle avoidance program that modifies the vision heading to prevent obstacle collisions or driving outside of the painted lane. This path is translated to motor commands, updating Johnny-5’s wheel speeds, and the software cycle starts again.

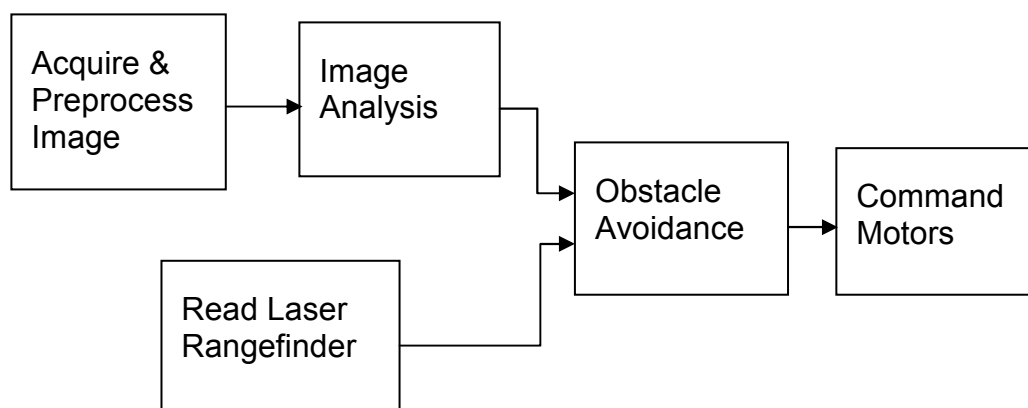


Figure 2-3: Autonomous challenge software structure

The flow diagram above corresponds well to the actual block diagram used in Labview. Each element of Figure 2-3 is implemented using what is called a subVI, Labview's term for a function or subprogram. Figure 2-4 shows the Labview block diagram controlling the main software loop. Each stage of the flow outlined previously is highlighted and labeled. Step 1 reads laser rangefinder data, while steps 2 and 3 read the camera and apply preprocessing. Step 4 analyzes the image and sends the desired heading to the obstacle avoidance in step 5. The finalized obstacle free heading is translated to wheel speeds in step 6 and actually sent to the motors in step 7. Focusing on the computer vision aspects of the software, this paper will focus on the image preprocessing and image analysis (steps 3 and 4).

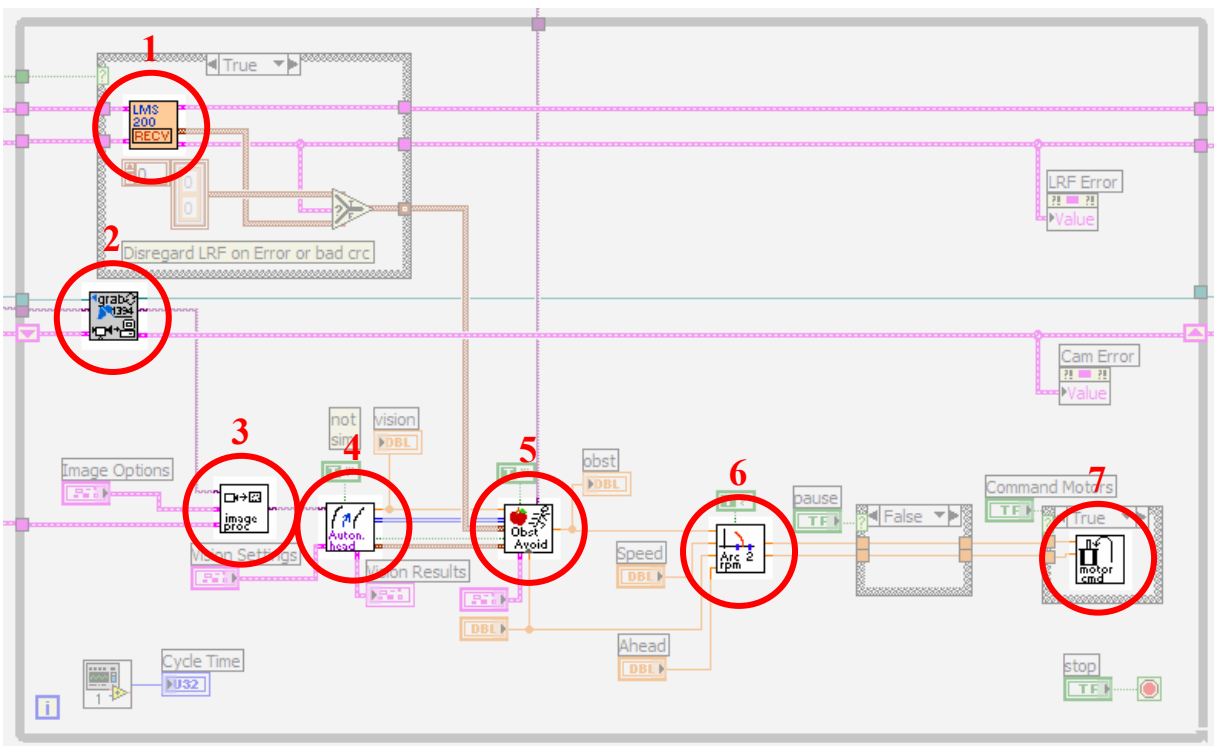


Figure 2-4: Labview block diagram of main software loop with major steps indicated.

2.3 Image Analysis

The image analysis component is the heart of the Autonomous Challenge software and is largely responsible for Virginia Tech's success in the 2004 competition. The software takes in a preprocessed grayscale image of the course in front of the vehicle, and

calculates a heading to the center of the lane in which the vehicle is traveling. The preprocessing steps are discussed in detail in the next chapter. Figure 2-5 outlines the software flow for image analysis.

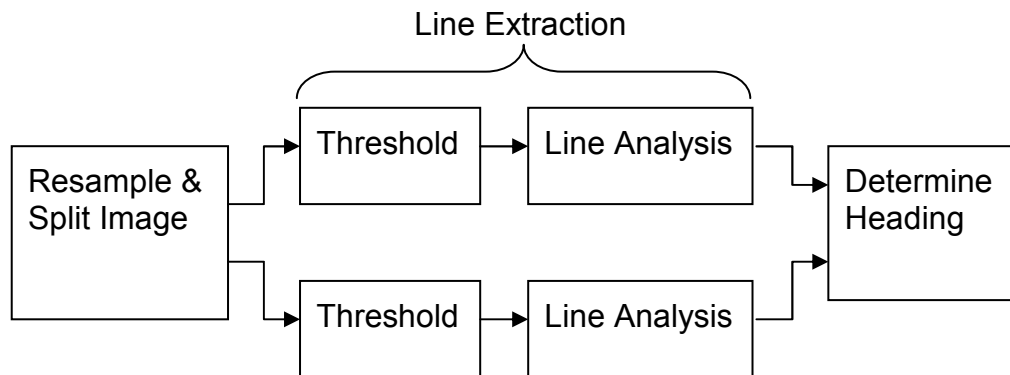
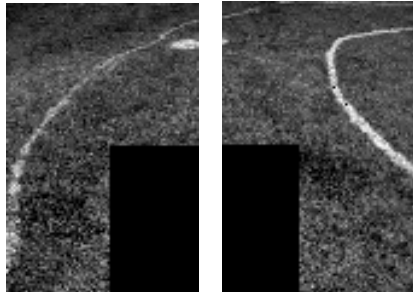


Figure 2-5: Flow of image analysis software

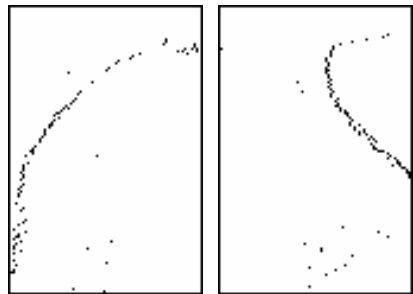
In the first step, the image is resampled to 160x120 pixels for faster computation. The image is also split vertically, resulting in a left and right half image. Each half of the image undergoes a line extraction process consisting of a threshold and line analysis. The threshold step marks pixels that are believed to be part of a line. These marked pixels are used to find the most dominant line during line analysis. After each side of the image has been analyzed, the extracted line information is passed through a series of behavioral rules that will ultimately determine which way the vehicle should head to move to the center of the lane. The details of the line extraction process, as well as the rules used to determine a heading, are presented in more detail in chapters 4 and 5. Figure 2-6 shows the outcome of each step on a sample image.



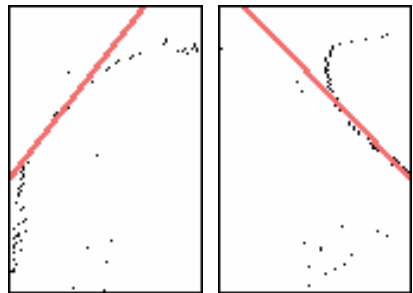
The image is acquired from firewire camera in RGB format with a resolution of 640x480 pixels.



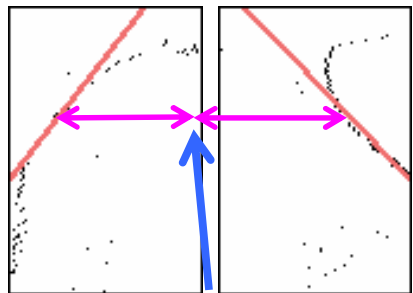
The image is converted to grayscale, and the vehicle body is removed. The resulting image is resampled and then split into two 80x120 pixels images.



A brightest pixel threshold is applied to the greyscale image.



The dominant line on both sides is found using the Hough Transform.



Using the extracted line information and a set of rules, the heading to the center of the lane is calculated.

Figure 2-6: Results of each step of the image analysis process

Chapter 3 – Image Preprocessing

In this phase of the software, the newly acquired camera image is prepared for line analysis. The majority of this process is converting the image from RGB to grayscale. Preprocessing also adjusts the brightness of the image, and removes the shape of Johnny-5 from the image. This chapter is devoted to explaining these steps.

3.1 Conversion to Grayscale

The image acquired from the camera is in 24 bit RGB (Red, Green, and Blue) format. Each pixel of the RGB image can be broken down to a set of three 8 bit values, with each value corresponding to the amount (intensity) of red, green and blue [5]. Each color component of the image is referred to as an image plane or channel. Reducing each pixel to a single value from the original set of three values shrinks the number of operations by two thirds and further simplifies image processing. This process is known as converting to grayscale, as the resulting image pixels will be stored as a single 8 bit number, often represented by a gray value between black and white.

There are several methods used to convert an RGB image to grayscale. A simple method is to add up 1/3 of each color value for the new grayscale intensity. However, this method is rarely used since different colors appear to have different brightness to the human eye. For example, the NTSC television standard uses the formula $Y = 0.299R + 0.587G + 0.114B$ to convert the RGB color space into a single luminance value [7]. For another point of comparison, Intel's Image Processing Library uses the formula $Y = 0.212671R + 0.715160G + 0.072169B$ [10]. A sample image taken of the Autonomous Challenge course is shown in Figure 3-1, and is converted to grayscale by using both 1/3 of each RGB channel and the formula used in NTSC with the results shown in Figure 3-2. Examining Figure X-22, the NTSC method appears to be a more accurate representation of the original image, especially noticeable in the image contrast and overall brightness.



Figure 3-1: A sample image of the IGVC course

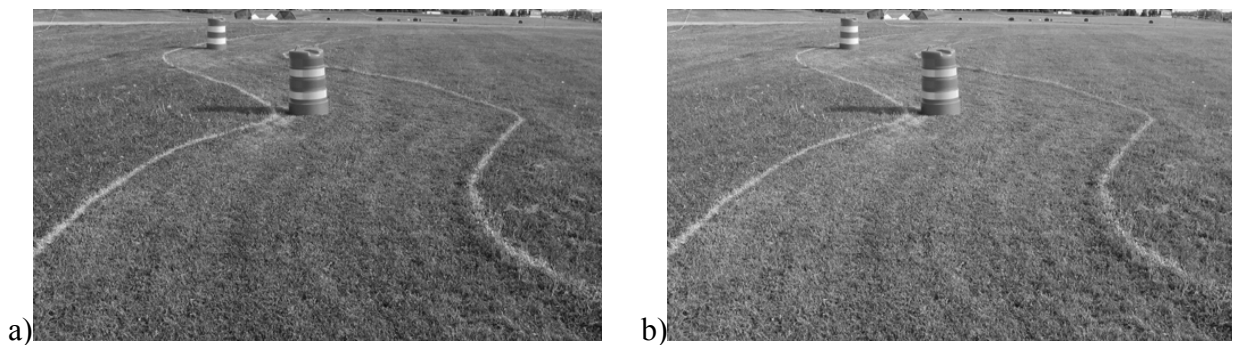


Figure 3-2: Sample image converted to grayscale using a) 1/3 of each color channel, and b) the NTSC formula.

While applications such as television broadcasts and photo editing might demand preserving appearance when converting to grayscale, it is not important when detecting lines. A method that exaggerates the presence of lines is more favorable than one that balances the colors. With this in mind, past IGVC teams at Virginia Tech have simply used the blue color channel of the original RGB image as the new grayscale image. Figure 3-3 shows the improvement in line clarity gained by using the blue color channel, as well as the results of using the red or green channels. The results of the blue color channel are well suited for line detection since the brightness of the grassy areas is very low compared to the painted line. This method of grayscale conversion was used on Johnny-5 during the 2004 competition.

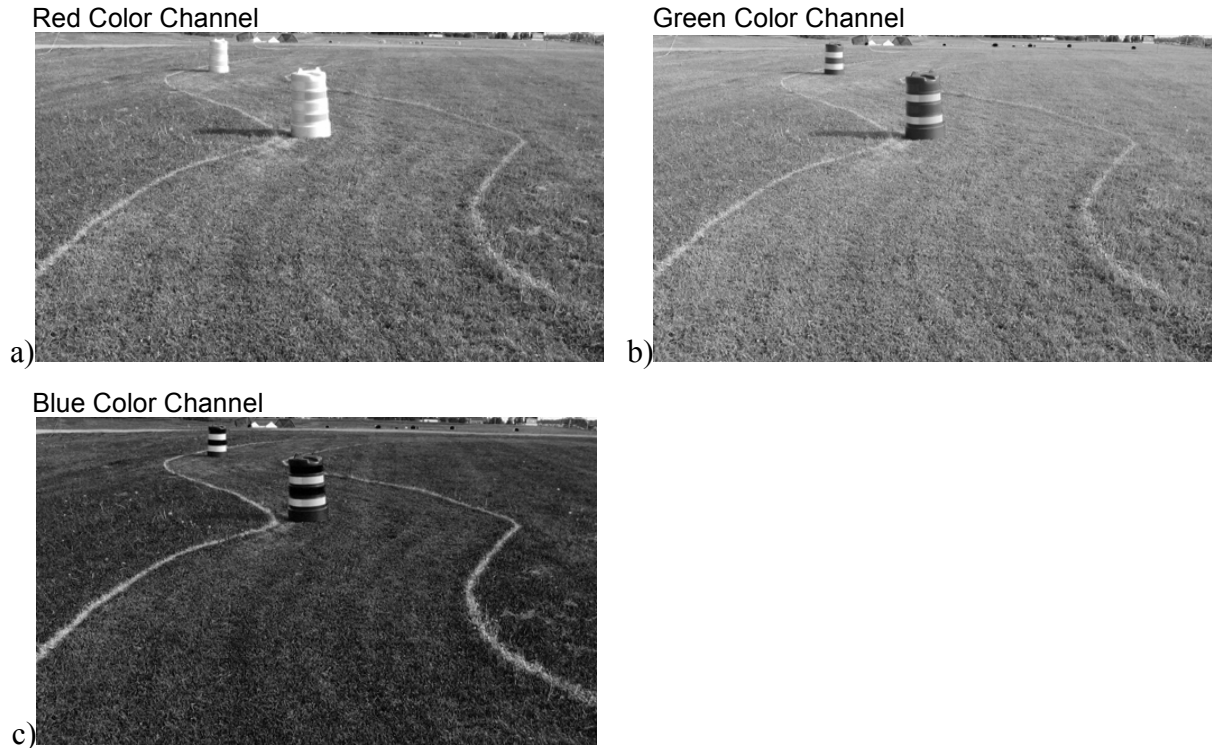


Figure 3-3: The sample image from Figure 3-1 is converted to grayscale by extracting only the a) red b) green and c) blue color channel.

While using the blue color channel to convert to grayscale yields acceptable results, it can also be improved. Further examination of Figure 3-3 shows that the green color channel does a poor job of providing contrast between the lines and grass. The green color channel also suffers from bright spots appearing on the grass. These weaknesses are consistent, and can be used to refine the grayscale image by using a combination of the blue and green channels. Taking twice the intensity values of the blue channel and subtracting the value of the green channel is able to remove noise caused by bright grass. This method will be referred to as mixed channel conversion from RGB to grayscale. The result of this mixed channel conversion is shown in Figure 3-4.



Figure 3-4: The sample image is converted to grayscale the mixed channel method

Taking this extra step becomes more useful if the image contains noise from sources such as dead or brown grass. Figures 3-5 and 3-6 show the added benefits of mixed channel grayscale conversion in these situations. When applying a threshold to the resulting images, the mixed channel method produces a stronger line with more points than the blue channel method of converting to grayscale. The thresholds used were adjusted manually to produce the same amount of noise in both images. A more detailed discussion of thresholding is presented in the next chapter. Mixed channel grayscale conversion was not used in the 2004 competition, due to a simple programming error that was not fixed until after competition. The images used in Figures 3-5 and 3-6 are gathered from the thesis of a previous student, Sudhir Gopinath, which is a good resource for photos with varying conditions of grass [6].

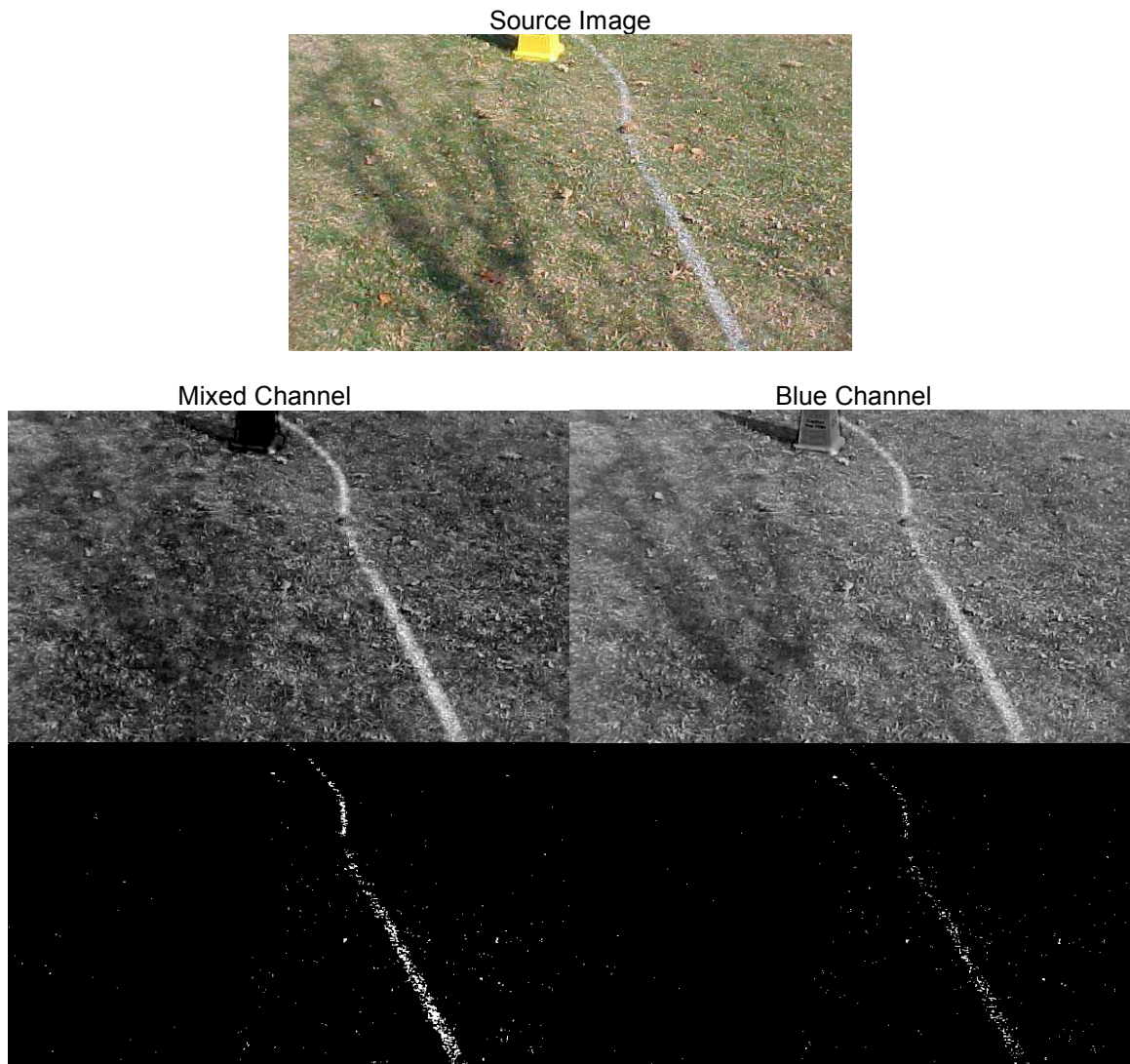


Figure 3-5: An image with noise (from leaves on the course) is converted to grayscale using a mixed channel method on the left and using only the blue channel on the right. After a final threshold step, the mixed channel method has a stronger line.

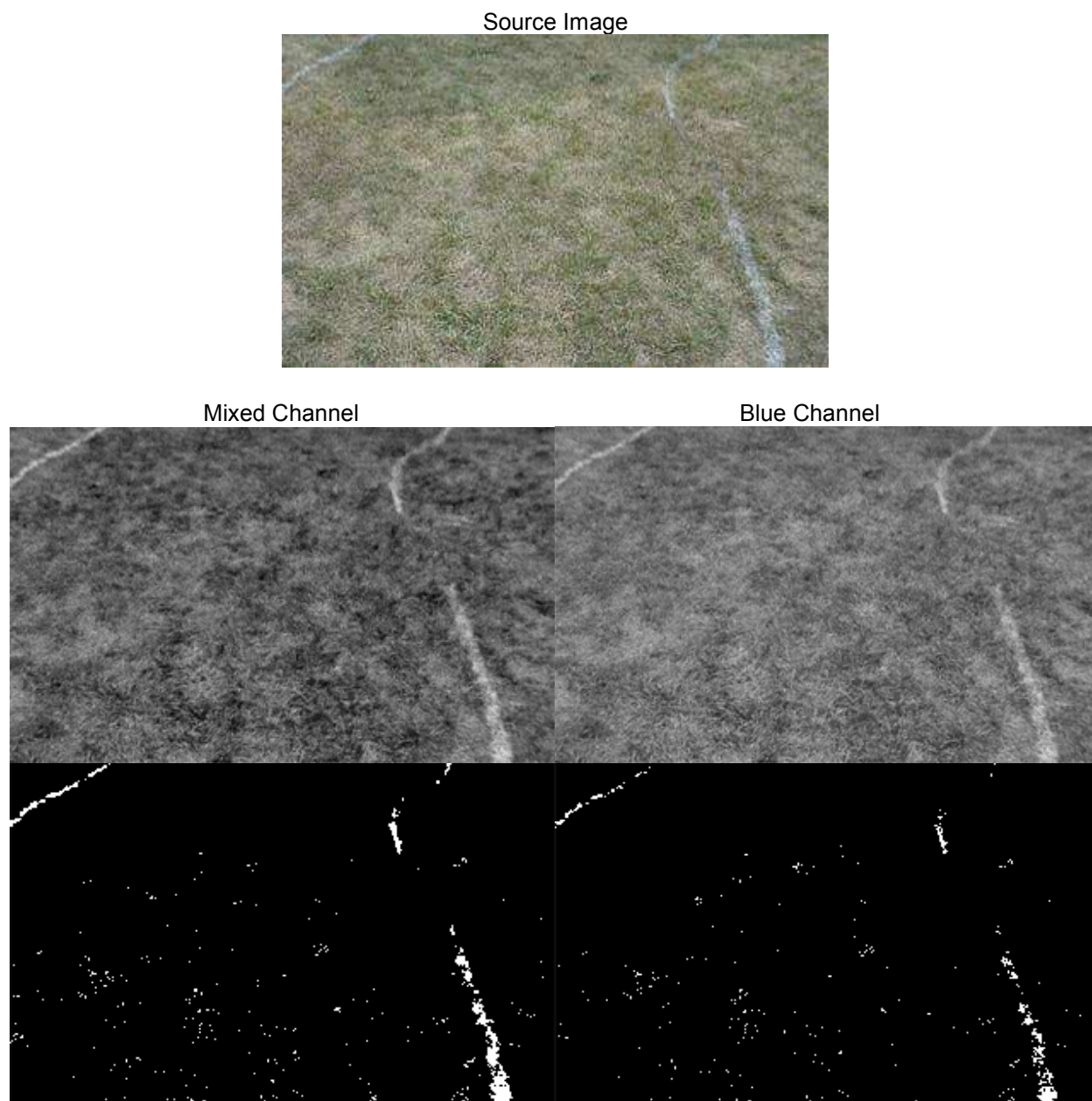


Figure 3-6: An image containing patchy brown grass is converted to grayscale using a mixed channel method on the left and using only the blue channel on the right. After a final threshold step, the mixed channel method has a stronger line.

3.2 Brightness Adjustment

As the vehicle travels through the Autonomous Challenge course, it may point towards or away from the sun, as well as travel in and out of the shadows of nearby trees or other cover, causing large brightness changes. These changes in brightness are compensated by the auto-exposure feature of the camera. In most cases the auto-exposure feature on the camera works in

a positive manner, but poor design of vehicle body shells can degrade performance. While this is a physical design issue, not software, a programmer should be aware that an hour of painting can save days of programming. During testing, it was found that light was reflecting off the body of Johnny-5, causing the camera to shorten the exposure and not register lines as strongly. On Johnny-5, the camera mast is located at the rear of the vehicle, making these vehicle reflections a much larger issue. Once both tops of the vehicle body and laser rangefinder were painted a flat black color, the image quality was noticeably improved. Teams in the future should consider impact on the camera rather than aesthetics alone when deciding how to paint their vehicles.

Another brightness issue is that the top of an image may be brighter than the bottom, becoming more pronounced as the camera is tilted upward to see more of the course. This change in brightness is due to the corresponding pixels at the top of the image receiving light from a larger area. Figure 3-7a shows an image where this is a problem and Figure 3-7b shows the average intensity of each row of the image from top to bottom. The auto-exposure feature of the camera will not correct these changes in brightness since it adjusts the image uniformly.

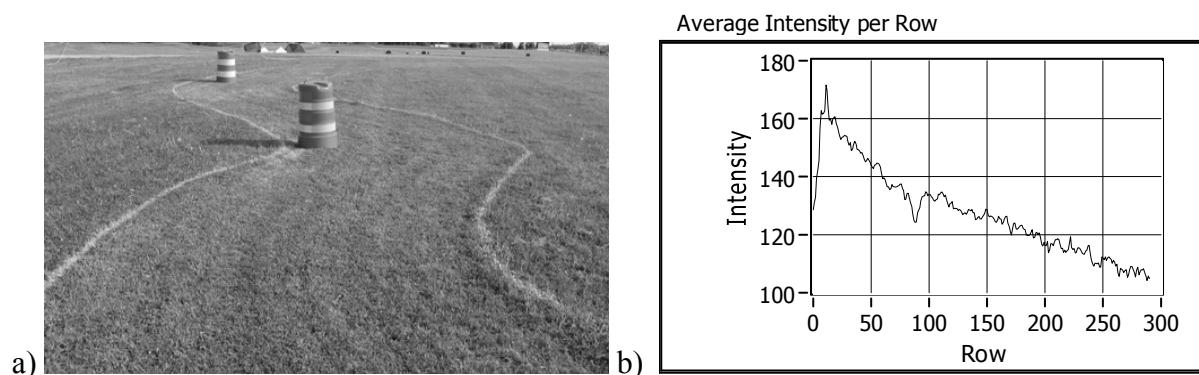


Figure 3-7: The average pixel intensity per row of the sample image (a) is graphed in (b) showing the image is brighter at the top

To account for the upper areas of images being brighter, a subtraction filter was used to lower the intensity of top of each image. Figure 3-8 shows an exaggerated version of such a filter. Each pixel of the source image is subtracted by the value in the corresponding location on the subtraction filter. This process will lower values of the source image more towards the top of the image and less towards the bottom of the image. The actual filter used during the competition subtracted 30 from the image intensity at the top image row, linearly decreasing to subtracting 0 at the top quarter of the image.



Figure 3-8: Subtraction filter used to account for brighter tops of images

3.3 Vehicle Removal

Johnny-5 is designed with a camera mast in the back of the vehicle allowing lines to be detected on the sides of the vehicle as well as the front. This camera placement puts the bulk of the vehicle body in the camera field of view. To prevent the vehicle body from interfering with any line detection, a black box is drawn over the captured image to cover the vehicle. Figure 3-9: shows an example image with this applied.

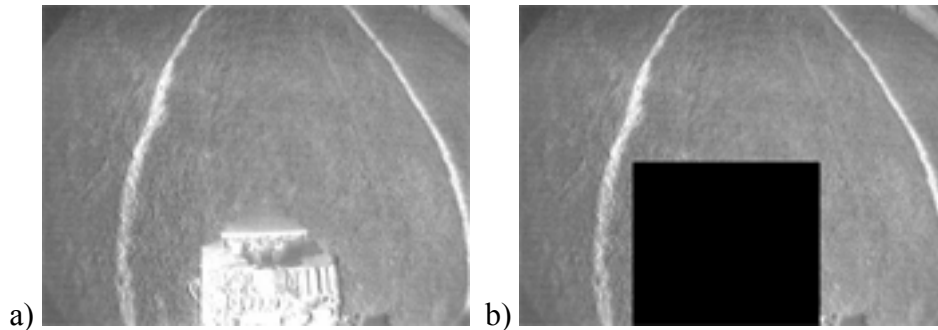


Figure 3-9: The body of Johnny-5 is removed from an acquired image (a) by overlaying a black box shown in (b)

Chapter 4 – Line Extraction

During the line extraction process, the equation of the most dominant straight line in an image is calculated. The source image has already been converted to grayscale and split into separate left and right half images. Line extraction is applied separately to both images and starts by applying a “brightest pixel” threshold, marking which pixels are believed to be part of the line. The results of the brightest pixel threshold algorithm are analyzed using the Hough Transform, which fits a line to the data and returns information on the line quality and orientation. This chapter describes why these algorithms were chosen and how they work.

4.1 Thresholding

At some point in the routine of vision processing, the computer must decide which pixels might correspond to lines painted on the grass. At the end of this process, the initial camera image will be converted to a binary representation where black pixels represent lines and white pixels represent free space. Two popular approaches to achieve this result are to use an edge detector, such as a Sobel or Canny [5], or to apply a pixel intensity threshold. Past experience has shown the edge detector methods to be unreliable and sensitive to noise, so recent Virginia Tech IGVC teams have favored the pixel intensity approach [14].

A simple approach to performing an intensity threshold is to set an intensity level where brighter pixels are considered part of a line and darker pixels are free space. Figure 4-1a shows a sample image of a line on the course under poor lighting conditions. Figure 4-1b shows a typical result using an intensity threshold and Figure 4-1c shows the result of using the brightest pixel threshold (discussed later in this chapter). The intensity threshold method assumes that the painted line will generally be the brightest part of the image. Assuming the desired output will be brightest is a common approach with several well researched methods of dynamically determining the threshold intensity [13].

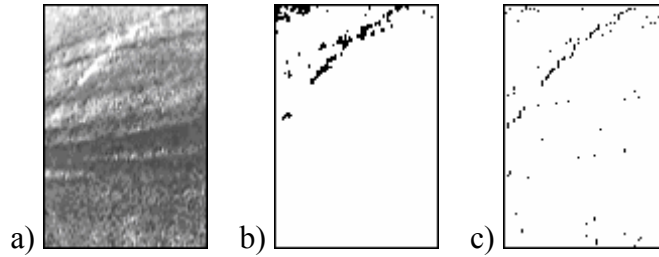


Figure 4-1: Comparison of (a) a sample course image with (b) an intensity threshold and (c) a threshold by brightest pixels

While successful for many other conditions, the dynamic threshold approach fails when the line is not the brightest object in the image. This happens frequently on the Autonomous Challenge course when a robot encounters construction barrel obstacles with retro-reflective stripes. Figure 4-2 shows an example of one of these barrels on the course and the result of using a dynamic threshold. The stripes on the barrel overpower the threshold and become the only object present in the threshold image. If the threshold is adjusted so that the line and barrel stripes are both present, the barrel stripes dominate the subsequent line fitting algorithm. As a result, any use of a dynamic threshold must also implement some object recognition to filter out regions that are not part of the line.

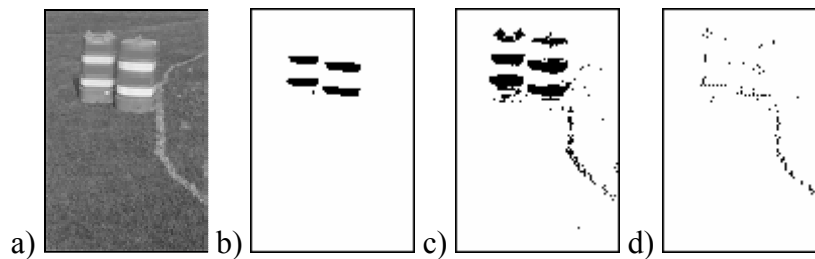


Figure 4-2: Comparison of (a) a course image containing barrels with (b) an intensity threshold, (c) a weaker intensity threshold, and (d) a threshold by brightest pixels

The weaknesses in the dynamic threshold approach can be mitigated by reducing the effect of large, bright objects. One way to do this is to take each row of an image and only accept the one brightest (highest intensity) pixel. Using this method, high intensity areas, such as barrels or glare, will only affect the rows in which they are present. Also of importance, this method prevents the number of falsely detected pixels from being large in comparison to the number of pixels correctly detecting lines. Unless there are many false high intensity areas in the image, most of the line will still be detected. This algorithm of focusing on each row

independently and choosing the brightest pixel came to be known as “brightest pixel thresholding” or simply “brightest pixel”.

Virginia Tech used the brightest pixel method in the 2003 IGVC by splitting the image into a right and a left region and heading to a point half way between the centroid of the brightest pixels in the two regions [17]. While very simplistic, this strategy allowed the team to win the 2003 Autonomous Challenge. Unfortunately, the software still had similar limitations as past attempts such as being unable to detect horizontal lines or handle breaks in lines. To improve on these limitations, the brightest pixel per row is combined with the brightest pixel per column, allowing detection of lines in any direction. Another change is adding a lower threshold, where if the brightest pixel intensity is less than the threshold, no pixel for a given row or column is returned.

The binary output image obtained using this brightest pixel threshold method is compared to the dynamic threshold results in Figures 4-1 and 4-2. These figures show the brightest pixel method is successful detecting the lines and minimizes the effects of other objects. Comparing Figures 4-2c and 4-2d shows a similar amount of points generated by the line, yet Figure 4-2d has less false points generated by the barrels. Noise is randomly scattered throughout the image rather than being concentrated in one area. With noise scattered, the lines still remain distinct with a low threshold as seen in Figure 4-1c. During the 2004 competition, the lower threshold feature was not implemented yet, so the lower threshold was effectively set to zero. More examples of the brightest pixel threshold are shown in the image processing examples of Appendix A.

4.2 Line Analysis

After the brightest pixel threshold is applied to the image, the Hough Transform [4] is used to fit a line to the remaining pixels. The Hough Transform has several qualities that make it favorable for detecting painted lines. The result of the Hough transform is the equation of a line, which is used to analyze the position and orientation of the course boundary line. The process of the Hough Transform also generates a score that is used to determine if it was a successful fit or falsely fitting noise. If a line is present in the image, the Hough Transform can identify it reliably; the addition of moderate amounts of noise will not change the output. Figure 4-3 shows

the resulting line fit from the Hough transform on the previous threshold results. Figure 4-4 demonstrates the immunity to noise.

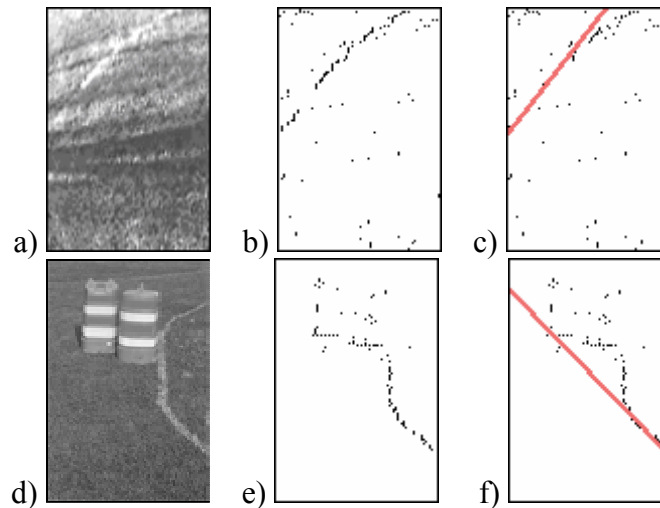


Figure 4-3: The steps of line detection process seen in (a,d) original image (b,e) brightest pixel threshold and (c,f) Hough Transform line fit

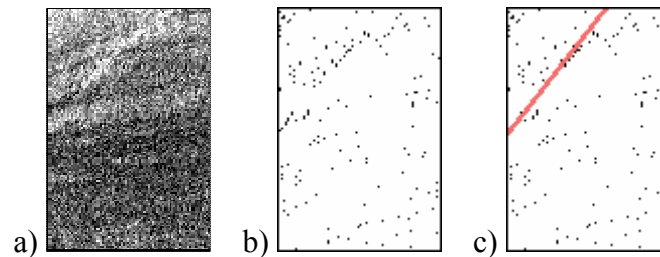


Figure 4-4: The image from Figure 4-1a with (a) Gaussian noise added (b) brightest pixel threshold, and (c) Hough Transform result

The Hough Transform operates by finding every possible line (reduced to a discrete set) that passes through each point in the image [4]. The occurrences of each line are totaled and the line with the most occurrences is selected as the dominant line. This number is also used as a score to separate boundary lines from noise. A simple example with only 4 points is shown in Figure 4-5a. Each point in Figure 4-5a generates a set of lines. Each of the lower three points generates the same diagonal line, while every other line in the image is only generated once. The diagonal line marked in red is the dominant line since it occurs three times while all other lines only occur once. A key feature of the Hough Transform is that the placement of the outlying point does not affect the fit line. Immunity to outlying points is an important distinction from

fitting a line using a method such as least squares, where outlying points will generate error as shown in Figure 4-5b.

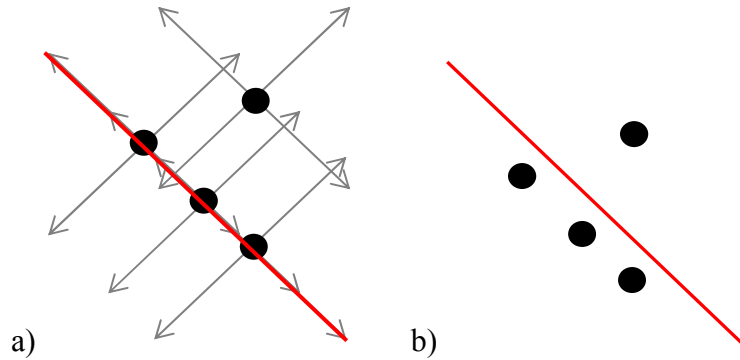


Figure 4-5: A line is fit to a set of points using (a) the Hough Transform and (b) least squares

While the basic theory of the Hough Transform is simple, other complications arise when implementing it in software. The familiar slope-intercept ($y=mx+b$) Cartesian form cannot be used to represent lines since vertical lines require infinite slopes. Instead, lines are represented in parametric polar form, $r = x \cdot \cos(\theta) + y \cdot \sin(\theta)$, allowing representation of a line in any direction [9]. The parameters r and θ , shown in Figure 4-6, are the length of the normal from the line to the origin, and the angle the normal makes with respect to the x-axis, respectively. The value of r is calculated using values of θ ranging from 0 to 180 degrees with 2 degree increments for each (x,y) coordinate of pixels present in the threshold image. The calculated value of r is rounded to a multiple of 3 pixels and the counts of $\{r, \theta\}$ pairs are incremented. Once all pixels have been processed, the $\{r, \theta\}$ pair with the most occurrences determine the equation of the most dominant line.

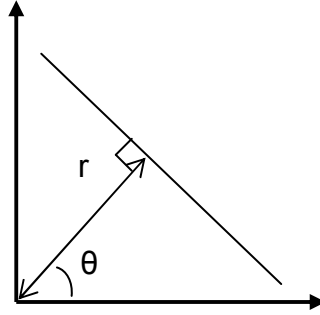


Figure 4-6: Parametric polar form of a straight line using r and θ

The Hough Transform is applied to the left and right side images separately, resulting in a calculated r , θ , and score (number of occurrences of parameter pair) for each image. These parameters are used to classify some basic attributes of the line. The score must be above a set threshold to be considered a successful line detection, and not fitting noise. At the 2004 IGVC, this threshold was set to 30 for each image half. The resolution of each image half is 80 x 120 pixels yielding a maximum theoretical score of 160. Appendix A shows the result of the Hough Transform applied to several example images as well as the resulting score. A line was also classified as horizontal if the θ value was between 75 and 105 degrees. These classifications as well as the $\{r, \theta\}$ pair for each line are the parameters used to determine the heading to center of the lane.

Chapter 5 – Heading Determination

After vision processing and line detection, the equation of the most dominant line on the left and right half of the source image is known. Using these results, the software determines a heading to the center of the lane. Unfortunately, no generalized solution could be determined that would solve for a heading given only the line equations and still work for all combination of line orientations. Instead, the line information is broken down into cases that are solved using distinct methods. This strategy causes the vehicle to execute different behaviors depending on the characteristics of the lines.

Changing the vehicle behavior for different situations was a new approach for Virginia Tech and solved many performance problems plaguing previous software. Past algorithms converted detected line pixels into obstacles and set headings based on the longest obstacle free path, leaving it susceptible to driving through breaks in the painted lines. Another flawed strategy was always setting the heading to drive between both lines when lines appeared on both left and right images halves. This approach failed to account for the same line appearing on both image sides.

5.1 The Behavior Approach

The development of behavior-based reactive mobile robots stem from work done by Rodney Brooks [2]. Brooks broke away from the mainstream complex hierarchal intelligence architectures by developing robots that operated using simple independent parallel behaviors [11] [1]. The heading determination process described in this chapter fits well into the reactive philosophy, using independent behaviors to interpret the position of the robot within the painted course lane. As the robot travels the course, the most current line information is used in a decision tree to select the appropriate behavior.

Developing software by building a set of behaviors has several advantages [11]. Behaviors are inherently modular allowing easier software testing and development. Changes made to a single behavior will not affect the rest of the system, so each behavior can be tested independently. The software can also be designed and implemented with an incremental approach by adding more behaviors on top of the previously tested and working behaviors.

5.2 Decision Trees

Once the boundary lines have been extracted, a decision tree is used to interpret the situation and selects the appropriate vehicle behavior. Each behavior interprets the line information in a unique way and produces a heading to the center of the lane. The first decision factor examines if the left and right halves of the original image both contain a line. When true, the decision tree shown in Figure 5-1 is used. If both sides of the image do not contain a line or if only one of the sides contains a line, the decision tree shown in Figure 5-2 is used instead. The behavior used to determine heading is at the end of each path. These behaviors are explained in more detail later in the chapter.

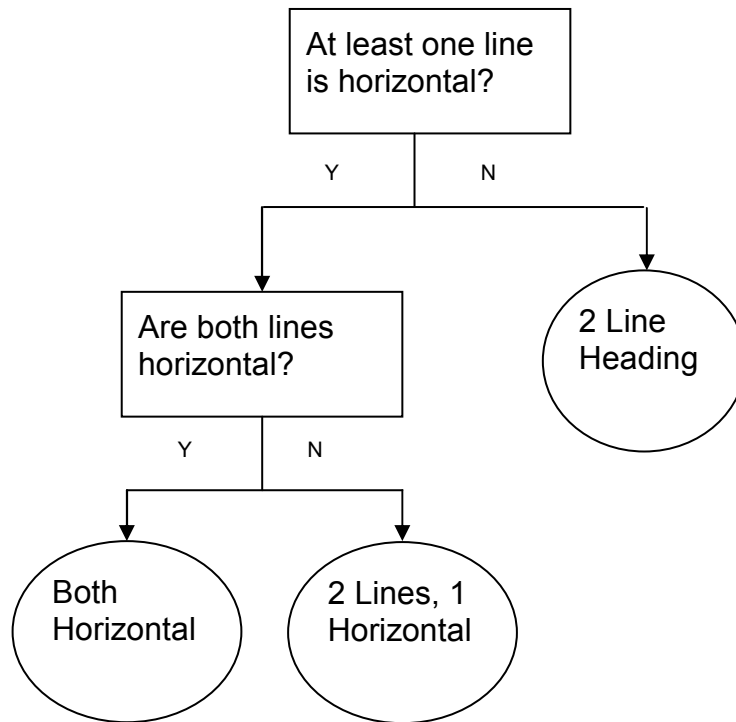


Figure 5-1: Decision tree executed when a line is detected on both sides of the source image

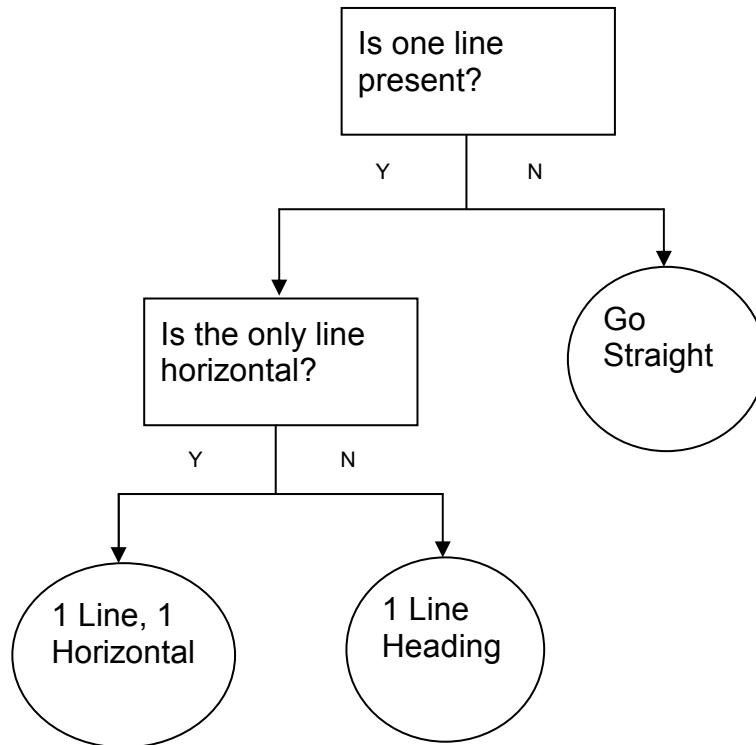


Figure 5-2: Decision tree executed when at least one side of the image does not have a detected line

5.3 Shared Calculations

Many of the vehicle behaviors apply similar calculations when determining the heading to the center of the lane. In many of the algorithms, it is easier to work with lines that are in the slope-intercept ($y=mx+b$) Cartesian form. Lines close to vertical will become slightly distorted, but will not affect any calculations significantly. Cartesian form allows for simple checking to see if the same line crosses on to both sides of the image and if the lines of both images are in the same direction (i.e., they have nearly the same slope). While converting from polar to Cartesian coordinates, the equation of the line is also converted from pixels to meters as well as correcting for perspective distortion.

Using the r and θ parameters found from the Hough transform, two points on the line are found and corrected for perspective. These coordinates are then used to solve for the equation of the line in Cartesian form. The coordinate frame is selected such that its origin is instantaneously coincident with the point half way between the two drive wheels with the X axis to the right and the Y axis pointed in the forward direction. Desired vehicle heading is measured in a right-hand sense from the X axis, as shown in Figure 5-3.

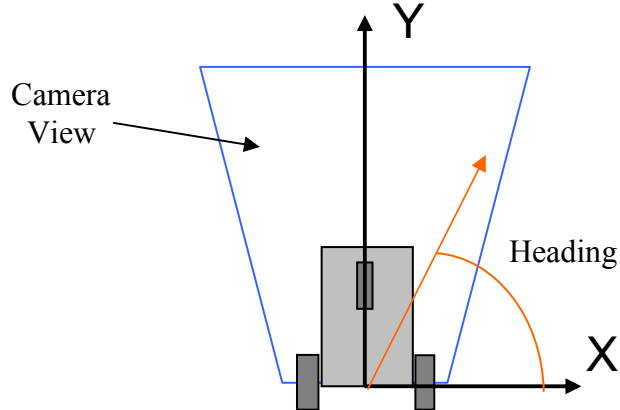


Figure 5-3: Vehicle coordinate frame with desired heading shown

5.4 Individual Behaviors

This section explains the vehicle behaviors named in Figures 5-1 and 5-2 in more detail.

2-Line Heading: The “2 line heading” algorithm first checks to see if the lines on both sides of the image are actually parts of the same line. This is accomplished by checking if the lines have similar y-intercepts that occur near the field of view of the camera. If the 2 lines are not part of the same line, the desired heading is set to be midway between the 2 lines. If the lines are part of the same line, their slopes are compared. If both lines have positive slopes, the heading is set for a gentle right turn. If both lines have negative slopes, the heading is set for a gentle left turn. If the slopes are different, no information can be extracted, so a heading for a zero-radius right turn is returned. A right turn was chosen because the Autonomous Challenge course forms a clockwise loop, so a right turn would be more likely to be correct.

2 Lines, 1 Horizontal & Both Horizontal: In the cases where the “2 lines, 1 horizontal” and the “Both horizontal” algorithms are used, no calculations are performed (other than converting to Cartesian form), and the heading is found by using slopes in a lookup table. The possible outputs of the “2 lines, 1 horizontal” algorithm are listed in Table 5-1. The possible outputs of the “Both horizontal” algorithm are listed in Table 5-2. The factors and outputs used in the “Both horizontal” algorithm are similar to what would happen in the “2 lines heading” algorithm, but the heading output is more extreme. Also like the “2 lines heading” algorithm,

having different slopes yields no useful information, so the heading for a zero-radius right turn command is returned.

Table 5-1: Output and factors of “2 lines, 1 horizontal” algorithm with an example image for each line case

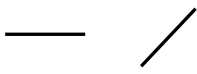
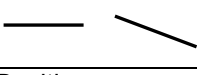
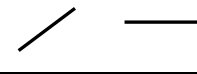
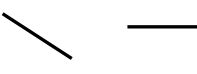
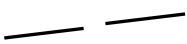
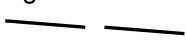
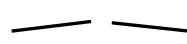
Image Side Containing Horizontal Line	Slope of Other Line.	Desired Heading (deg from right)
Left	Positive 	60°
Left	Negative 	180°
Right	Positive 	0°
Right	Negative 	120°

Table 5-2: Output and factors of “Both horizontal” algorithm with an example image for each line case

Same Slope?	Slope	Desired Heading (deg from right)
Yes	Positive 	0°
Yes	Negative 	180°
No	Mixed 	0°

No Line Heading & 1-Line Horizontal: The cases where no line or only one line is detected are simpler than if both lines are detected. If no line is detected, it is assumed the vehicle is in a wide lane and the heading is set to 90° (straight ahead). If there is only one line detected and it is horizontal, the desired heading is set to a zero radius turn in the direction away from the side containing the line.

1-Line Heading: In the remaining case, only one line is detected and it is not horizontal. This case is handled by the “1 line heading” algorithm, which first checks to see if the line is on

the wrong side of the image based on its y-intercept. This situation typically occurs when there are breaks in the painted line, as shown in Figure 5-4. In this figure, the right boundary line appears only in the left half of the camera image. While not always accurate, this case often occurs when the y-intercept is positive while the slope is negative for the left side of the image or positive for the right side of the image. Once it has been determined which side of the course the single line is on, a heading is selected to move the vehicle back toward the center of the lane.

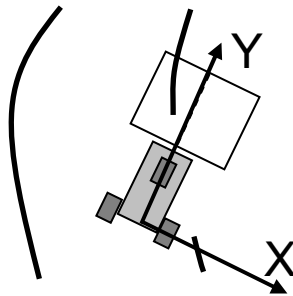


Figure 5-4: A break in a line causes the right side line to appear on the left half of the camera image

Chapter 6 – Testing, Performance, and Conclusions

The most important phase of any autonomous vehicle software development is the actual implementation on the final vehicle platform. Especially when entering a competition such as the IGVC, the success of software is judged on performance alone. Before the IGVC competition, the software presented in this paper was tested on the final vehicle platform, Johnny-5, as well as custom made simulator. The following chapter covers the testing process of the software, the changes made as a result of this testing, and conclusions based on the performance of the software during testing and the final runs on the IGVC course.

6.1 Computer Simulation

Software debugging and quantification of the software performance are often the most difficult parts of developing software for autonomous vehicles. While it is critical to test software on the target vehicle platform, real world testing often involves several factors that cannot be controlled, reducing testing repeatability. Some of these factors include starting vehicles in a different starting position for each test, and course conditions such as lighting, health of grass, and visibility of painted lines. These changing conditions can make it difficult to correlate software changes with changes in vehicle performance.

To eliminate uncertainties from testing, most software testing was done using a computer simulation. The simulator is a set of software that constructs a virtual world with simulated lines, obstacles and vehicle kinematics. Within the simulation software, users can monitor simulated sensor data, vehicle response, and an overhead view of the virtual course. A screen capture of the overhead view with a magnified section containing the vehicle is shown in Figure 6-1. Navigation software interfaces with the simulator by using a set of functions that return data in the exact format as a real sensor, such as a laser rangefinder. In order to port software from simulator use to controlling a real vehicle, the simulated sensors and motor controller can be swapped with the real sensor interfaces, so no major changes are needed in software. Only minor differences need to be accounted for, such as needing to disable perspective correction when using simulated data.

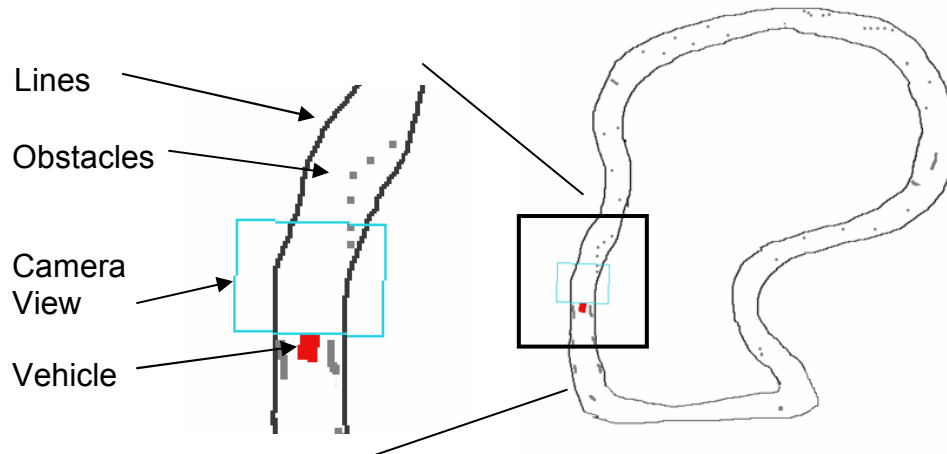


Figure 6-1: Screen capture of the simulated course

The simulator is flexible and can be used to model a variety of sensors and vehicle platforms, allowing it to be used by other autonomous vehicle teams at Virginia Tech. Each simulated sensor has configurable settings such as refresh rate, camera field of view or laser rangefinder resolution. These options allow a user to configure the simulator to better represent an actual vehicle, or allow a user to reduce the processing time taken by the simulator. Both the laser rangefinder and camera are simulated with lower refresh rates than the actual sensor to save processing power. The configuration used while testing Autonomous Challenge software for Johnny-5 is listed in Table 6-1.

Table 6-1: Simulator configuration

Vehicle		LRF		Camera		Refresh Rates (Hz)	
Rear Track (m)	0.7	Ang. Range (deg)	180	Width (m)	6	LRF	10
Wheel Base (m)	1.04	Ang. Res (deg)	3	Length (m)	4.5	Camera	10
Steering	Differential	Start Angle (deg)	0			Motor Rate	30
		Detect Range (m)	15			Display Rate	20

Using a simulator allows software to be tested in a static environment where all sensor data is perfect. The ability to test with perfect sensor data was critical to the development of the behavioral rules presented in Chapter 6. By testing with the simulator first, any problems with image preprocessing or sensor configuration could be ruled out, accentuating any problems with higher level algorithms. Testing on the simulator introduced cases that would be hard to detect during real testing such as negotiating breaks in the painted lines as illustrated in Figure 6-4.

6.2 Preliminary Vehicle Testing

After exhaustively testing the Autonomous Challenge software on the simulator, testing was moved to the final platform, Johnny-5. With most logic errors debugged on the simulator, testing on the vehicle was focused on verifying the simulator's results, calibrating the camera, and tuning software parameters. Some changes were made to the computer vision software, however, most vehicle testing involved testing and refining obstacle avoidance software. With only a couple days of testing, Johnny-5 was able to complete practice courses successfully.

The largest change in moving from previous testing (course photos and simulation) to actual vehicle implementation was the camera. A sample image taken from the camera on Johnny-5 is shown in Figure 6-2a. The camera lens had a wider field of view than any test images, introducing slight amounts of barrel distortion (also known as fisheye distortion). The effect of the barrel distortion can be seen in Figure 6-3 which plots the perspective corrected points from the lines from Figure 6-2a. Barrel distortion is worse near the corners of the image, causing the false curve seen at the bottom of Figure 6-3. However, barrel distortion is assumed to be negligible since the majority of a line is not affected by the distortion. Other than accounting for the position and tilt of the camera for use in perspective correction calculations, no software changes were needed to accommodate the camera when transitioning from simulation and sample images to physical vehicle testing.

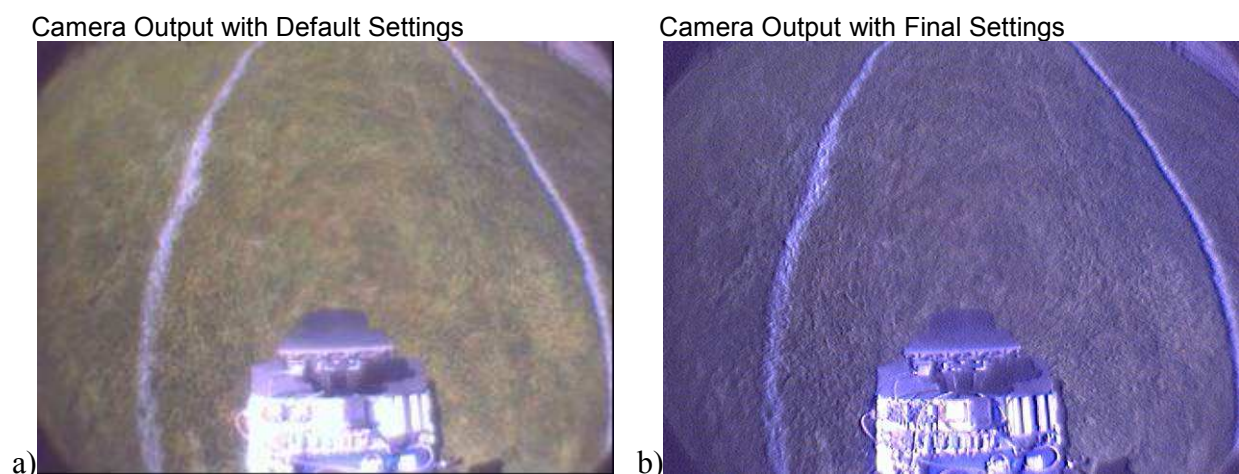


Figure 6-2: Images acquired from the camera with (a) default settings and (b) final settings

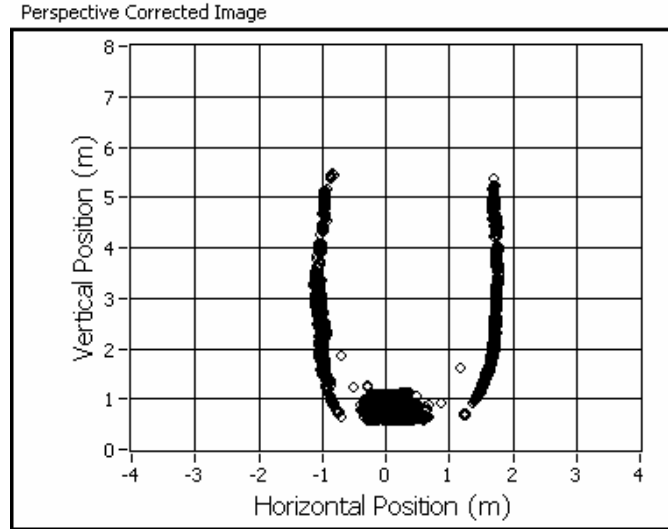


Figure 6-3: The position of the lines in Figure 6-2a after perspective correction

While not directly affecting the software, proper configuration of the camera had a significant impact on vehicle performance. By adjusting the shutter, brightness, and saturation of the image, the contrast between the grass and lines was increased. The result of changing the camera settings from default settings is shown in Figure 6-2b. Even with the decreased saturation, reducing the presence of color, the blue color channel still provides the greatest contrast between the grass and painted lines. All configuration parameters were set in National Instruments Measurement and Automation Explorer (MAX). MAX uses the D-cam protocol to configure and acquire images from IEEE-1394 cameras. Table 6-2 lists the configuration parameters of the camera used.

Table 6-2: Camera Configuration used in NI MAX

Camera Settings	
Setting	Value
Brightness	383/383
Auto Exposure	Auto
Sharpness	50/255
White Balance U	130/255
White Balance V	87/255
Saturation	79/255
Gamma	0/1
Shutter	4/7
Gain	118/255
Video	RGB, 640x480 at 15fps

Testing on the final platform also validated that the software could complete all calculations in a short enough time to keep the vehicle under control. Since a simple arc path model was used to model the vehicle travel in the obstacle avoidance portion of the software, a computed path would only be valid for travel roughly equivalent to the length of the vehicle, 1 meter. Traveling at full legal speed, 5 mph, the vehicle travels 2.2 meters per second.

Operating in the worst case scenario, the complete decision cycle of the software took 105 ms to complete. In 105 ms, the vehicle would travel .23 meters, well below the limit of 1 meter. The computational time for major steps of the software including vision and obstacle avoidance is shown in Table 6-3. The timing measurements were done on a Pentium 4M 3.06 GHz laptop computer. Since the camera operates at 15 Hz, it produces a new image every 67 ms, the worst case time for image acquisition. For typical operation, the software would complete all calculations before the next video frame is ready, lowering the total cycle time to the camera rate of 67 ms. The laser rangefinder only operates at a rate of 12 Hz (this can be increased to 75 Hz by using RS-422 rather than RS-232), but does not limit the execution time of the program. The laser rangefinder interface will always return the latest data and not force the program to wait for a new set of data.

Table 6-3: Computation time for each software step

Computation Time	
Process	Time (ms)
Image Acquisition	67
Vision	
Preprocessing	2
Thresholding	0.4
Hough Transform	12
Heading Determination	<1
Total	14
Obstacle Avoidance	24
Total	105

6.3 Performance at Competition

The navigation software on Johnny-5 was well tested before arriving at the 2004 IGVC. However, there were still unexpected situations that resulted in software changes. After further refining the software during the 2004 IGVC, Johnny-5 was able to successfully negotiate the competition course, finishing in first place.

The most significant software change made at competition was switching to brightest pixel based thresholding. All testing done before arriving at competition was on student-made test courses using large solid colored trash cans as obstacles. During these tests, a dynamic thresholding algorithm was used with successful results. The obstacles used in competition were orange construction barrels with retro-reflective stripes, causing the dynamic threshold software to eliminate painted lines in favor of the barrel stripes. To fix this problem, the thresholding algorithm was changed to the brightest pixel algorithm, reducing the effect of the barrels. A more in depth discussion of thresholding and this problem is presented in Chapter 5.

Another unexpected condition was the placement of speed bumps on the competition practice course. These speed bumps were plastic strips roughly 4 inches wide and the length of the lane. The speed bumps were placed perpendicular to the direction of travel, as shown in Figure 6-4. If the vehicle approached the speed bump with an orientation that was close to perpendicular, the true course lines would still be correctly identified, but the speed bump would appear as a solid object that would be passed later to obstacle avoidance software. To address these speed bumps, the brightest pixel threshold was adjusted to only take a pixel from every other column rather than every column. This broke the previously solid line into many pieces that would be filtered out by the obstacle avoidance software. While solving the speed bump problem, this has the drawback of lowering the sensitivity to horizontal lines. The speed bumps were only present on the competitions practice course, and not the official course. It is not known if the speed bumps will appear on future competition courses.



Figure 6-4: A yellow speed bump on the IGVC practice course

With the final changes made on the practice course provided at the 2004 IGVC, Johnny-5 got off to poor start during the official Autonomous Challenge event. The laser rangefinder had stopped sending data as soon as Johnny-5 started its first run on the competition course. As a

result, Johnny-5 struck a barrel within 20 feet of the starting line, ending the run. After the laser rangefinder problem had been resolved, Johnny-5 attempted a second run on the Autonomous Challenge course, completing the entire course running at a conservative speed of 1 mph. The reliability of the software was highlighted, when the judges allowed the vehicle to continue circling the course, completing a second “victory lap”. Johnny-5 was able to complete the course on each subsequent run, with a higher speed each time. The final run was completed using a top speed of 2.3 mph.

6.4 Conclusions

A good benchmark for the success of the software developed in this paper is the actual performance at the 2004 IGVC Autonomous Challenge compared to other vehicles. The software developed in this paper finished in first place, and was able to complete two laps of the Autonomous Challenge course without error. The success of the software can be attributed to reliably detecting lines, using the detected line information to set a desired heading, and by completing all calculations fast enough to keep the vehicle on course. The development of a simulator and other testing applications was essential for debugging and refining the final algorithms used.

During testing, it was found that lines could be reliably detected on practice courses that were more challenging than courses found at competition. The practice course often contained tree shadows and patches of dead grass that the competition course would not have. The line detection process consists of conversion to grayscale and thresholding, which can be found in Chapters 4 and 5 respectively. Line detection was more reliable compared to past years because the camera used in the 2004 year produced better images than camcorders used in the past, and the brightest pixel algorithm was able to reject local disturbances such as glare or barrels.

Using a simulator allowed for testing the interpretation of line information independently of the ability to reliably detect lines. By devoting more time to the problem of what to do with line information, older software problems could be solved. These solutions include detection and handling of horizontal lines, as well as calculating a heading to the center of the lane rather than relying solely on obstacle avoidance.

The algorithms presented in this paper are only one component of the overall navigation problem. The vision-based heading to the center of the lane must be integrated with reliable

obstacle avoidance procedures. For success in competition, it is also critical to have a solid vehicle platform, a robust sensor set, rapid navigation decision update rates relative to the speed of the vehicle and responsive motor control, among many other things.

6.5 Future Work

The software presented in this paper was successful in competition and could run multiple laps on practice courses without error. However, if an observer waited long enough, the vehicle would inevitably run off the course or strike an obstacle. Also, if someone was familiar with the design of the vehicle software, it would not be difficult to create a course that would cause the vehicle drive out of a space in the lines or turn 180 degrees and head the wrong way down the course. Below are some problems noted with the vision software in its current form.

180 Degree Turns – There are situations where the software will command the vehicle in a path that will turn the vehicle 180 degrees and it will head where it came from. Example course section layouts that could cause this problem are shown in Figure 6-5. Figure 6-5a shows a barrel trap configuration that will cause the obstacle avoidance software to override the vision heading until the vehicle is turned away from the barrels. If the vehicle has already turned more than 90 degrees to avoid the barrel, the vision software will read the now horizontal line in view and command a further turn until the vehicle is rotated 180 degrees. In Figure 6-5b, the turn contains an elbow shape, the vehicle will enter the elbow rather than continuing the course, causing an eventual reversal in direction. These problems could be addressed by implementing a memory of previous vehicle headings or even GPS positions to detect when the vehicle has reversed course.

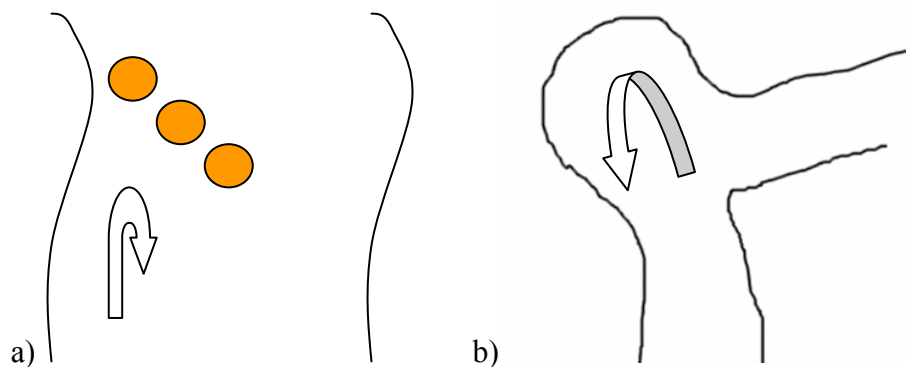


Figure 6-5: Situations where the vehicle might reverse course

Horizontal Lines – While the software is better at handling horizontal lines than previous software, it still occasionally crosses horizontal lines. A large part of this problem stems from crippling the detection of horizontal lines to allow for speed bumps as discussed earlier in this chapter. If it is confirmed that future competitions do not contain speed bumps, than the speed bump modifications could be removed.

Breaks in Lines – The heading determination software accounts for the presence of breaks in lines, but it can easily be fooled. Figure 6-6a shows a line break that the software could detect and stay in the lane. By changing the shape of the line slightly to have a different orientation in the camera view, shown in Figure 6-6b, the software will be fooled into driving out of bounds. A possible improvement to software would be to detect these breaks in the line, and command the vehicle to turn to look for the other line for guidance. While Figure 6-6b shows an extreme case not likely to be found in competition, placement of obstacles could also steer the vehicle out of a line break.

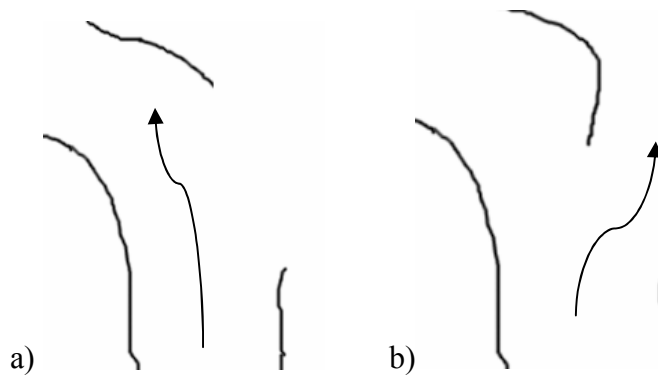


Figure 6-6: The software can fail to detect a break in a line

Too Many Obstacles – The use of brightest pixel thresholding reduces the effect of disturbances such as barrels on line detection. However, the line can only be detected if it contributes more brightest pixels than the barrel. If most of the field of view is covered by barrels rather than a line, line detection will fail and it will fit a line through the barrels rather than the line. This could be improved by implementing a barrel detector that can detect barrels by color or shape and remove all nearby pixels, eliminating the retro-reflective stripes on the barrels.

Higher Driving Speed - Johnny-5 traveled at 2.3 miles per hour on the final run at the 2004 IGVC. During testing, it was found that the software was only reliable up to speeds of 3 miles per hour. The most likely cause of increased failures at higher speed is the obstacle avoidance software. While not the focus of this paper, upgrading the obstacle avoidance would most likely allow the vehicle to operate at higher speed. This is currently an educated guess and data will need to be logged on a higher speed attempt to confirm that the vision software was still providing accurate data during a failure.

References

- [1] Arkin, R., *Behavior-Based Robotics*, MIT Press, 1998.
- [2] Brooks, Rodney A., "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, RA-2, April, 1986, pp. 14-23.
- [3] Conner, D. C., *Sensor Fusion, Navigation, and Control of Autonomous Vehicles*, Master's Thesis, Virginia Tech, July 2000.
- [4] Duda, R. O. and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Comm. ACM, Vol. 15*, pp. 11–15 (January, 1972)
- [5] Forsyth, Ponce, *Computer Vision A Modern Approach*, Prentice-Hall, 2003.
- [6] Gopinath, Sudhir, *Using Color and Shape Analysis for Boundary Line Extraction in Autonomous Vehicle Applications*, Master's Thesis, Virginia Tech, December 2002.
- [7] Hoffmann, Gernot, *Luminance Models for the Grayscale Conversion*, Available: <http://www.fho-empden.de/~hoffmann/gray10012001.pdf>.
- [8] IGVC Rules, Available: <http://igvc.org/deploy/rules.htm>.
- [9] *Image Transforms – Hough Transform*, Available: <http://cs-alb-pc3.massey.ac.nz/notes/59318/111.html>
- [10] Intel Corp., *Image Processing Library Reference Manual*, 200, Available: <http://www.cc.gatech.edu/dvfx/readings/iplman.pdf>.
- [11] Murphy, R. R., *Introduction to AI Robotics*, MIT Press, 2000.
- [12] National Instruments, *NI Vision, IMAQ Vision Concepts Manual*, January 2005.
- [13] Otsu N. (1979) A threshold selection method from gray-level histograms, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 9(1), pp. 62-67.
- [14] Roman, Chris and Reinholtz, C. F., "Robust Course-Boundary Extraction Algorithms for Autonomous Vehicles", *IEEE Intelligent Systems*, Nov./Dec., 1998, pp.32-39.
- [15] Travis, Jeffery, *LabVIEW for Everyone*, Prentice Hall, Upper Saddle River, 2002.
- [16] Virginia Tech - Johnny-5 2004 Design Report, Available: <http://www.igvc.org/deploy/design/reports/dr69.pdf>.
- [17] Virginia Tech – Optimus 2003 Design Report, Available: <http://www.igvc.org/deploy/design/reports/dr69.pdf>

Appendix A – Image Analysis Examples

This appendix shows several example images with the results of the brightest pixel threshold, and Hough Transform with score for each half of the image. These examples use the blue channel method of grayscale conversion, and use a brightest pixel lower threshold of zero. A fit is considered a line if the score is above 30. Note that not all line detections are correct.



Scores: 50

76



Scores: 58

62



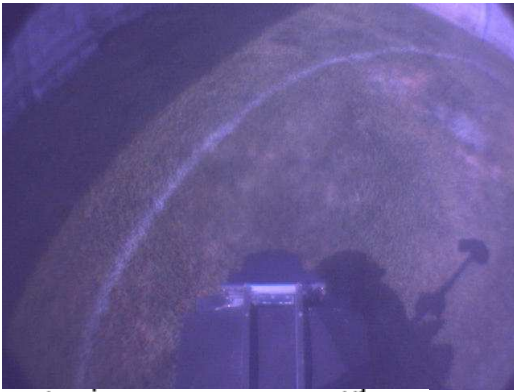
Score: 40

39



Score: 44

39





Score: 47

25



Score: 38

42

Vita

Andrew Bacha was born on July 6th 1981 to Robert and Linda Bacha of Reston, Virginia. He graduated South Lakes High School in 1999 and went on to attend Virginia Tech. Andrew graduated with a Bachelor of Science in Mechanical Engineering in 2003. Afterwards, he enrolled in graduate school at Virginia Tech, completing his Master of Science in Mechanical Engineering in 2005. Andrew will be further pursuing his graduate studies and work with autonomous vehicles in the Mechanical Engineering department of Virginia Tech as a PHD student.