Using Color and Shape Analysis for Boundary Line Extraction in Autonomous Vehicle Applications

by

Sudhir Gopinath

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

December 20, 2002

Blacksburg, Virginia

Keywords: Computer Vision, Autonomous Vehicles, Line Recognition, Color Extraction, Shape Analysis, Binary Image Analysis, Mobile Robots

Using Color and Shape Analysis for Boundary Line Detection in Autonomous Vehicle Applications

Sudhir Gopinath

Abstract

Autonomous vehicles are the subject of intense research because they are a safe and convenient alternative to present-day vehicles. Human drivers base their navigational decisions primarily on visual information and researchers have been attempting to use computers to do the same.

The current challenge in using computer vision lies not in the collection or transmission of visual data, but in the perception of visual data to extract from it useful information. The focus of this thesis is on the use of computer vision to navigate an autonomous vehicle that will participate in the Intelligent Ground Vehicle Competition (IGVC.)

This document starts with a description of the IGVC and the software design of an autonomous vehicle. This thesis then focuses on the weakest link in the system – the computer vision module. Vehicles at the IGVC are expected to autonomously navigate an obstacle course. Competing vehicles need to recognize and stay between lines painted on grass or pavement. The research presented in this document describes two methods used for boundary line extraction: color-based object extraction, and shape analysis for line recognition.

This is the first time a combination of these methods is being applied to the problem of line recognition in the context of the IGVC. The most significant contribution of this work is a method for extracting lines in a binary image even when the line is attached to a shape that is not a line. Novel methods have been used to simplify camera calibration, and for perspective correction of the image. The results give promise of vastly improved autonomous vehicle performance.

Acknowledgements

Though the name on this thesis is mine, the credit mostly belongs to people other than myself. The people I thank below will continue to be an inspiration for me. First and foremost, thanks to my parents, Tara Gopinath and S. Gopinath. Education truly does begin at home.

Virginia Tech is a truly beautiful place and it was my privilege to study in its intellectually stimulating environment. Every faculty member at Virginia Tech struck me as being not just knowledgeable, but also willing to go out of their way to help a student. Dr Reinholtz, as my advisor, has been more than just a technical mentor. I have learnt a lot from him just by observing him being a gentleman. Dr Saunders' class on Mechatronics was an enjoyable hands-on experience. As the first multidisciplinary course I took at Virginia Tech, this course gave me the confidence to jump into the field of computer vision, a field not commonly studied by students of mechanical engineering. Dr Kachroo amazes me with his energy and knowledge. His course on Robotics taught me the basics of robotics at a rate that I did not think possible until I experienced it. I'd also like to thank the other professors I came in contact with at Virginia Tech: Dr Don Leo, Dr A L Abbott, Dr Jan Bøhn, Dr Ken Shaw, Dr Bob West, Dr E F Brown, and Michael Alley. Each one of them has taught me something valuable that I shall carry with me forever.

I'd like to thank National English School, Bangalore, for the important role it played during the formative years of my life.

My brother is an inspiration to me and I do wish I were more like him. I'd like to thank my fiancée, Beena, for being a true friend to me. Thanks also to the friends who have enriched my life in the United States: Sam Parikh, the Trout family, and Steve Edwards.

Table of contents

Table of Figures

Chapter 1: Introduction

Robots are being developed to perform tasks as diverse as milking cows or exploring other planets. One of the areas of research in robotics is the development of autonomous vehicles.

An autonomous vehicle can navigate in an unfamiliar environment without human control or intervention. Autonomous vehicles are also called mobile robots or unmanned vehicles. To navigate effectively, autonomous vehicles need to sense their immediate environment and react accordingly. Since the 1960s, researchers have been trying to build mobile robots that use vision to navigate [McKerrow, 1991]. The problem has not been as simple as it initially appeared. So far, a mobile robot that is completely autonomous has not been developed. This document presents a description of a computer vision algorithm used to make navigational decisions for an autonomous vehicle.

1.1 Autonomous Vehicles

Robots are being developed that either assist or have replaced humans in performing a variety of dangerous, difficult, or boring tasks. Humans use vision as a primary sensory input to perform many of these tasks. However, robotic systems today are unable to “understand” visual information the way that humans do. Even primitive life forms, such as insects, are able to use visual data more effectively than current robotic systems. The ability to extract useful information from visual data has many applications, such as in security systems that use face recognition and in visual inspection systems for quality control. This thesis focuses on the use of visual data to make intelligent navigational decisions.

Research in vision systems has intensified over the past few years for many reasons. One reason is that computer processors are becoming faster and smaller. This suggests that we shall be able to mount processors more easily on mechanical systems and have them direct increasingly complex tasks. Second, artificial intelligence systems are appearing to be increasingly intelligent. Systems have been developed that can play chess better than humans or make more money than humans can in the stock market. Presently these systems perform better than humans primarily because of the vast computations involved in these actions. Still, the effect of these fast computations is that these systems increasingly mimic human intelligence. Third, components for computer vision systems, such as digital cameras and data communication systems, are available at increasingly economical prices. These systems have been designed specifically for interfacing with computer systems. Similarly, cost effective vehicle components, such as actuators, actuator controllers, and related computer interface devices, are readily available.

The only "missing link" in a vision-based autonomous vehicle is the machine intelligence or vision algorithm that would use sensed data to direct vehicle motion. Challenges faced in developing this vision algorithm are described in the section 1.2.

1.2 Using Computer Vision

The problem of using visual data is more complex than it initially seems. The problem is challenging for the following reasons:

Large Amounts of Data

Computer representation of a picture has a large amount of data. Image data (including color information) is typically represented using 3 bytes per pixel. Therefore, a moderately sized picture of 640 X 480 pixels has almost a megabyte of data. The time taken to process this data slows down any operation that uses information gleaned from a picture.

Inefficient Processing Methods

Biological brains are capable of parallel processing of data. Most processors developed are not designed for parallel processing. Parallel processors that exist have been built for specific applications and provide computational performance gains only for programs that are multithreaded.

Complexity of Process

The workings of biological brains are not yet well understood. People who challenge the future of the field of artificial intelligence believe it is impossible to understand the workings of the human brain since we are using the human brain to understand it. The brain manages to look at a complete picture, single out the detail it is looking for, and focus on that. Processors work well with details but are not good at synthesizing the details to look at the larger picture. Also, not enough is understood about the heuristic, or self-learning, capabilities of the brain and whether heuristic algorithms are necessary to implement computer vision.

For the above reasons, it is difficult for a processor to use visual information. A navigation scheme would require that a processor not only recognize an object, but also be capable of extracting information about the features of the recognized object.

The research presented in this thesis concentrates on a single problem, that of recognizing white lines painted on grass or pavement. Attempting to solve a specific predefined problem has allowed deeper analysis of the problem. In the development of this thesis, a specific problem involving visual navigation was targeted. The background and details of this problem are described in the next section.

Chapter 2: The Challenge

The Intelligent Ground Vehicle Competition (IGVC) is organized by the Association for Unmanned Vehicle Systems International (AUVSI) to promote the development of unmanned systems and related technologies. Autonomous vehicles developed by college teams participate annually in the IGVC. The Autonomous Vehicle Team (AVT) at Virginia Tech has consistently been one of the best performers at this competition. The AVT consists primarily of mechanical and electrical engineering students who build autonomous vehicles as their senior design project. The next few sections describe the IGVC and the challenges faced by the vehicles at the IGVC.

2.1 Rules of the Competition

The IGVC consists of 3 different competitions – the autonomous challenge, the navigation challenge and the design competition. In some years, there has also been a follow-the-leader event that will not be described here. Only the autonomous challenge is described here because it pertains to the research described in this document. At the IGVC, each team's vehicle is required to navigate a test course laid out on a grassy field. Figure 1 shows the course being navigated by Maximus, one of Virginia Tech's vehicles that participated in the IGVC 2001.

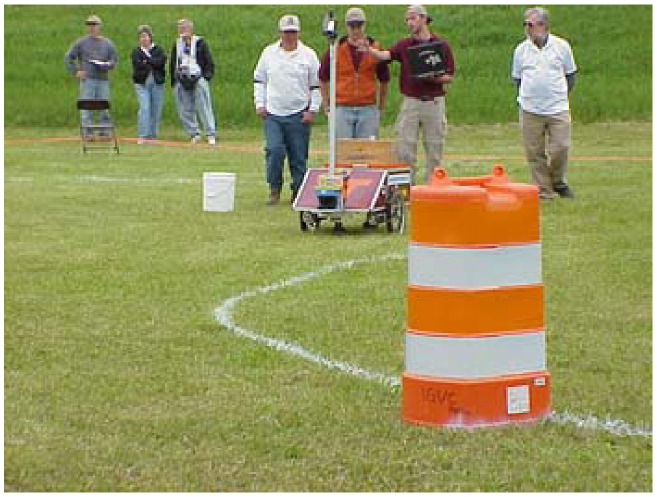


Figure 1. Typical course at the IGVC

The vehicle is required to navigate along a path about 10 feet wide. The edges of the path are marked by white painted lines about 2 inches wide. These lines may be continuous or dashed. In addition to staying between the lines, the vehicle must avoid obstacles placed randomly around the course. These obstacles may be barrels, traffic cones, or white circles painted on the ground. Since neither the shape of the path nor the locations of the obstacles are known before competition, the vehicle has to continually sense its environment, make navigational decisions and then drive in the appropriate direction. The vehicle that completes the course in the least amount of time wins the competition. Interestingly, in the 10 years that the competition has been held, vehicles have successfully navigated the complete course only during one year. On all the other occasions, vehicles that covered the most distance on the course were considered the winners. Other rules of the competition that pertain to the vehicle, such as safety features and vehicle size, are not described in this section because they do not directly influence the vision system of the autonomous vehicle. More information about the IGVC can be found on the web at http://www.igvc.org.

2.2 Autonomous Vehicles at the IGVC

Though vehicles entered in the competition varied widely in their detailed designs, all the vehicles had similar basic subsystems. These basic subsystems are described below.

Mechanical Platform

Most of the vehicles used a structure mounted on wheels as the framework to carry the components of the autonomous vehicle. Most vehicles used electric motors to drive the vehicle because electric motors can be conveniently controlled. Most successful vehicles at the recent competitions used a differential drive system, a system capable of executing zero-radius turns. The differential drive system uses independent drives. For an extensive description of the physical design of autonomous vehicles, the reader is referred to the *Introduction to Robotics: Mechanics and Control* by Craig [1989.]

Sensors

Past vehicles at the IGVC have typically used a range of sensors, such as video cameras and laser rangefinders, to sense the environment. Video cameras are essential for sensing the lines and circles painted on the grass. Most successful vehicles at the IGVC sense obstacles using laser rangefinders. A laser rangefinder senses the distance to obstacles by measuring the time of flight of a laser beam. This beam is sent out repeatedly at different angles to sense objects anywhere within a 180 degree planar field of view.

Navigation Software

The data input from the various sensors is processed by navigation software that runs on computers mounted on the vehicle. The navigation software processes this data and extracts useful information. The output of this navigation software is a decision to move in a particular direction at a particular speed. This software can be written using any high level programming language. Typically, C++ has been used for this purpose because of its modularity, flexibility and power. A user interface enables an operator to interact with the vehicle to modify code, set parameters or define a mission.

Motor Controllers

The motor controller converts the speed and direction decision of the navigation software into a form that can be used by the amplifiers to drive the motors. Motor output, such as speed or position, is usually controlled by a special motor input signal such as a pulse width modulated (PWM) signal or a servo control pulse (SCP). To achieve accurate motor output, an encoder or a potentiometer connected to the motor provides feedback to the motor controller. Motor controllers are processors that are dedicated to the task of creating appropriate input signals to the amplifiers. Most vehicles at the IGVC use commercially available motor controllers.

2.3 Performance of Past Vehicles

Past vehicles at the IGVC have been moderately successful. Though the course at competition can be easily navigated by a human, none of the dozens of autonomous vehicles that have entered over the years has been able to consistently navigate it. Most of the vehicles failed by crossing over path edges or by driving into obstacles.

Past teams at the IGVC have been successful in designing the sensors and the motor controller sub-systems of their autonomous vehicles. The failure of the autonomous vehicles, in most cases, could be attributed to the limited capabilities of the navigation software. The navigation software, and in particular the image processing software, has typically been the weakest link in the autonomous vehicles. The focus of this thesis has been to strengthen this weakest link. The software framework of the vehicle is discussed in the next section.

Chapter 3: The Software

The software of an autonomous vehicle is the main component that defines the autonomous capabilities of the vehicle. This section gives an overview of the software framework of the vehicle.

3.1 The LabVIEW Programming Environment

LabVIEW 6.1 was chosen as the programming language for the development of software for this thesis. LabVIEW is a graphical programming environment that uses the G programming language. LabVIEW was initially developed for enabling easy interfacing between PCs and external instruments but has now expanded into a complete programming environment. LabVIEW was chosen as the software environment for the following reasons:

1) LabVIEW is designed as a measurement and automation software, and therefore communication between PCs and external devices is simplified. This greatly reduces development time of autonomous vehicle software because in a typical autonomous vehicle, the main processor has to interface with a variety of external devices such as sensors and motor controllers.
2) Programming in LabVIEW involves drawing a flowchart, rather than typing in text. This not only simplifies the process of programming, but also makes it easy to visualize and debug a complex program.
3) LabVIEW has an extensive toolset that deals with image processing. Vision Builder for LabVIEW is a prototyping environment for image processing applications. Vision Builder can be used to directly apply image-processing techniques and observe their effects on images. Combinations of techniques that are found to be effective can then automatically be converted into executable LabVIEW programs. The Vision Developer toolset is an extensive library of functions that can be used to create custom image processing applications.

4) LabVIEW programming is designed such that all user interaction with the program is through a graphical user interface. The programmer's design of the user interface is limited to organizing the locations and appearance of objects on the user interface. This is important in autonomous vehicle design because it allows people without programming experience to run and test the vehicle.
5) LabVIEW programs automatically utilize the processors multithreading capabilities. Thus, parallel processing is performed whenever possible, which speeds up execution time. Since the software running autonomous vehicles is typically computation-intensive, this provides an advantage.

LabVIEW programs are written using the dataflow philosophy, which means that programs are designed around the flow of data. A LabVIEW program starts at the point where input data enters the program, and each step processes and converts the data into suitable form until the data is in a form that can be output. A LabVIEW program looks like a flowchart and one can follow the data around as the program executes. The dataflow philosophy of LabVIEW will become apparent to the reader in section 3.3 where the software framework is discussed.

3.2 Overview of Software Design

The software framework developed in LabVIEW is inspired by the NavMan software developed by David Conner [2000.] The software uses a traditional framework for development of mobile robots, that Brooks [1991] called the Sense-Model-Plan-Act framework. The software first senses the environment around it, models the environment, plans its next move, and acts upon its decision. These actions repeat themselves in a continuous cycle. Fast execution times are desirable because the vehicle reacts more often and more quickly to changes in environmental conditions. Faster execution times also mean that the vehicle is less affected by transient errors that may occur at different points in the software.

Figure 2 shows the software framework developed for tackling the IGVC. While the underlying structure of this framework would be suitable for any autonomous vehicle, the choice of sensors was specific to the IGVC. To sense the lane boundaries and simulated potholes, a camera was essential as a sensor. At past competitions, the Laser Rangefinder was found to be the most effective method for sensing obstacles such as barrels. A GPS unit was necessary in order to participate in the Navigation Challenge part of the IGVC.

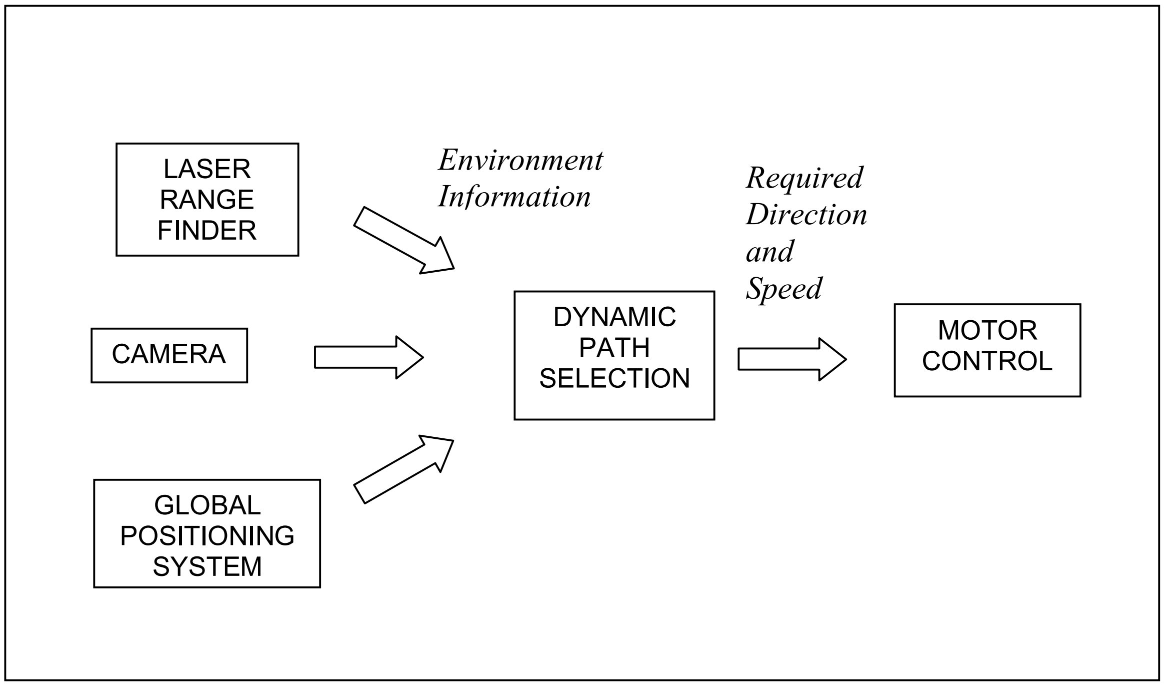
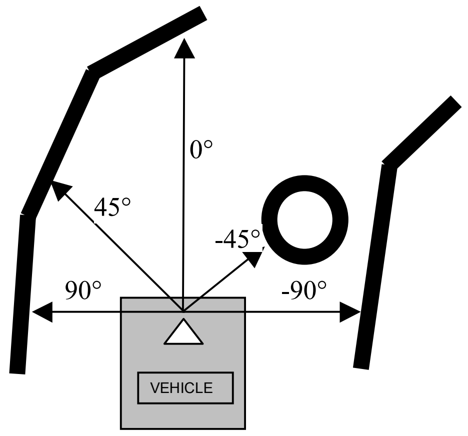


Figure 2. Software architecture of the autonomous vehicle

Sensors

The sensors are the first blocks of code to be executed. Each sensor block is sensitive to a particular feature of the environment. The sensors pass information about the environment to the next sections of code. NavMan used a combination of evidence grids and a vector field histogram to achieve sensor fusion [Conner, 2000]. Putting together data obtained from different sensors is often a complex problem because sensors measure different parameters at differing rates and resolutions [McKerrow, 1991]. The information gleaned from the sensors is usually in very different formats. For instance, the data from the Laser Rangefinder is a one-dimensional array of numbers, where each number represents distance to obstacle at a particular angle. Data from a camera is usually an image and is stored in the computer as a 2-dimensional array for a monochrome image or a 3-dimensional array for a color image. Data from a GPS receiver in its simplest form consists of two numbers – latitude and longitude. To simplify the other sections of the code, all sensory data was converted into a common format before being passed on. Thus, each of the sensor blocks in Figure 2 senses and models the environment. Data output from each of these blocks is represented in the form of a polar passability plot, inspired by the polar histogram used by Borenstein and Koren [1991] to generate Vector Field Histograms. The polar passability plot was chosen as a common format because it represents the environmental data as seen from the vehicle's point of view. A sample polar passability plot with a resolution of 45 degrees is shown in Figure 3. The array in Figure 3b is obtained from the scene shown in Figure 3a. The data in the second column corresponds to distances to obstacles at different angles.



(a)

Angle in degrees (relative to straight ahead)	Distance to obstacle in inches
90	20
45	25
0	35
-45	10
90	20

(b)

Figure 3. Generation of the polar distance array

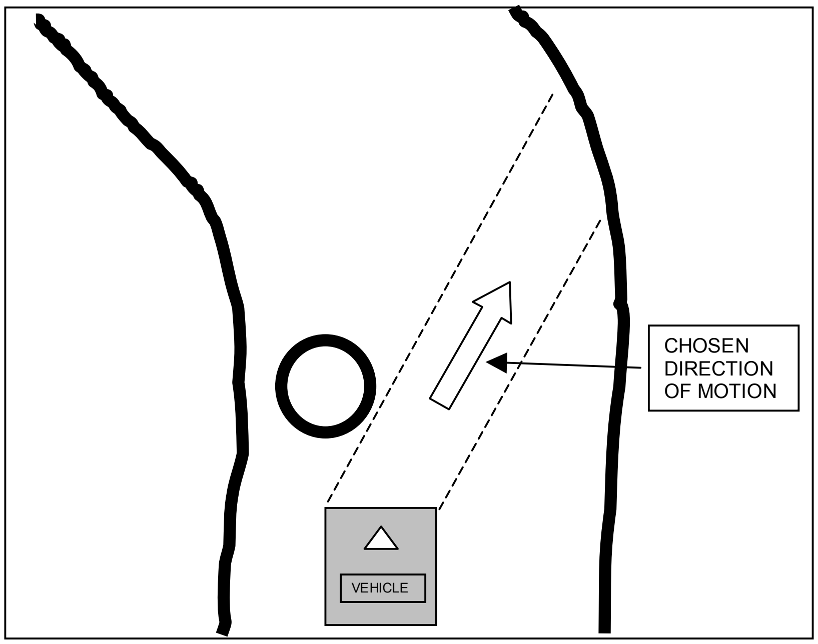
To sense the typical obstacle encountered at the IGVC, the resolution of the polar passability plot was chosen to be 2 degrees. This angular resolution would sense an obstacle approximately 2 inches wide at a distance of 10 feet. Also, the range of the plot was chosen to be from +90° to -90°. Standardizing the resolution and range of the polar passability plot meant that it was sufficient to pass just the second column of the polar distance array to remaining parts of the program, thereby improving the computational efficiency of the algorithm.

Dynamic path selection

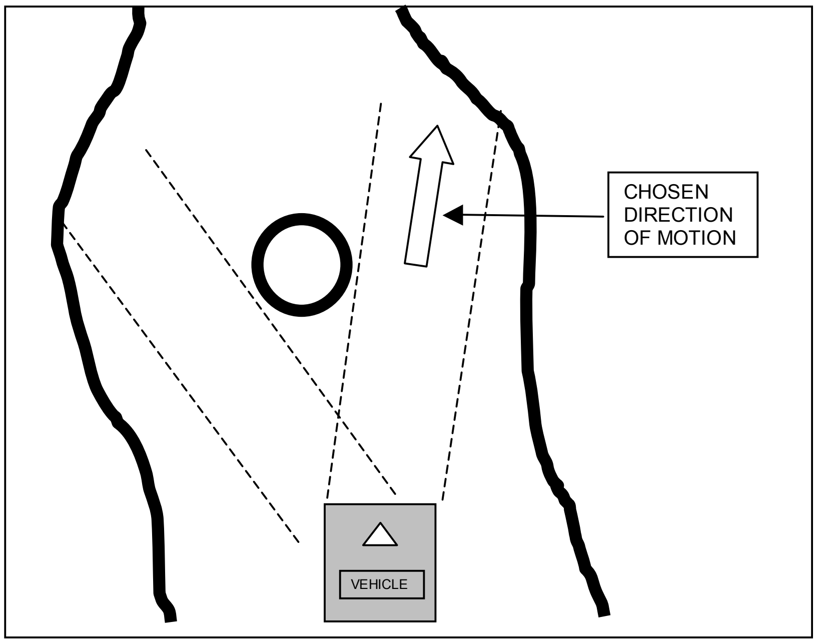
The dynamic path selection algorithm receives environmental data in the form of polar distance arrays and finds the longest straight clear path available to the vehicle. The first function of the dynamic path selection block is to combine the data. If the data from sensor i at angle x is called data i(x), then the integrated data from n sensors is given by

$$\text{Integrated data }(x) = \min \{\text{data }1(x), \text{data }2(x), \ldots, \text{data }n(x)\}$$

The one-dimensional array containing integrated data contains information about the immediate environment of the vehicle, as perceived by the vehicle. The dynamic path selection algorithm tries out different paths available to the vehicle and chooses a suitable direction to follow. The vehicle looks for the longest straight clear path available as shown in Figure 3a. Given a choice between straight paths of equal length, the vehicle would choose the path that meant the least turning angle, as shown in Figure 3b. In other words, the vehicle would have a tendency to move straight ahead unless this decision was subsumed by the presence of obstacles.



(a)



(b)

Figure 4. Selection of suitable straight path using Dynamic Path Selection Algorithm. (a) The vehicle chooses the longest path available (b) that requires the least turning angle

The output of the dynamic path selection algorithm is the desired relative direction and desired speed. At this point in the algorithm, the data passing through the software is in its simplest form.

Motor Control

The motor control block takes the desired direction and speed and actuates the vehicle. Commercially available motor controller boards that are compatible with LabVIEW can be used to interface the PC with the motors. Another important function of the motor control block is to deal with real-time controls issues, since the vehicle is a dynamic system. LabVIEW comes with a toolset for implementing PID control and Fuzzy Logic control. Also, the motor control functions that come with LabVIEW have functions that smooth the motion of the motors. The output of the motor control block is the motion of the vehicle.

3.3 Implementation of Software using LabVIEW

Figure 5 shows LabVIEW code that runs the vehicle. Each block seen in the diagram is called a subVI, which is the LabVIEW equivalent of a subroutine or a function. The whole code is placed in a while loop. The program inside the while loop executes block by block, from left to right until an iteration is complete. Once an iteration is completed, it loops back and the program executes again. The sensor subVIs can be switched on or off using LabVIEW. When switched off, a sensor sends in a polar passability plot with default values that indicate a clear field ahead.

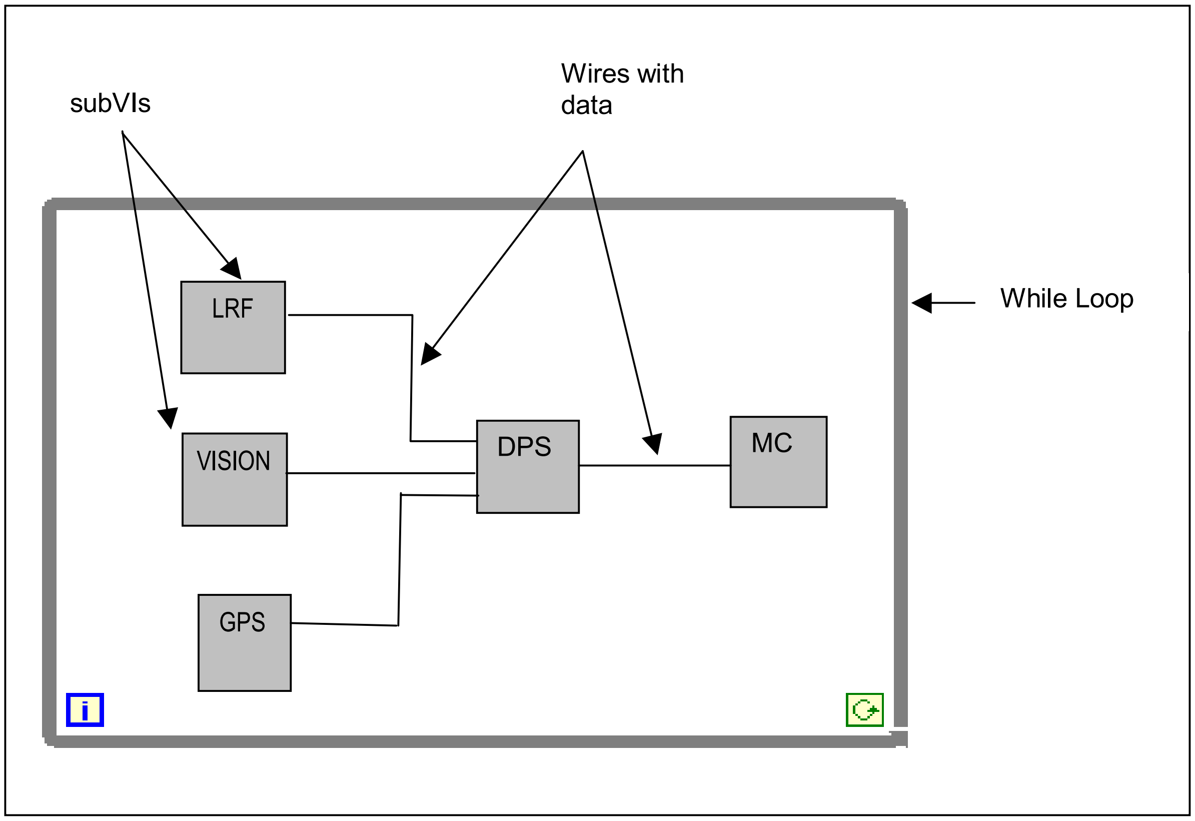


Figure 5. Main features of LabVIEW code that runs the autonomous vehicle

The time taken to execute most of the subVIs is not deterministic, because it depends on the kind of environment the vehicle is in. For instance, if the vehicle is in a position where it has a clear path straight ahead, the dynamic path selection subVI takes very little time to decide on direction and speed. If the vehicle were close to many obstacles, the subVI would take more time trying to find a suitable direction for the vehicle to take. Therefore, the program within the while loop would take varying amounts of time to execute. Each iteration of the while loop would execute as often as possible. The vehicle would therefore be reacting to the environment as quickly as the processor speeds would allow. Thus, running the same code on a faster processor would improve the performance of the vehicle by making it less susceptible to transient errors in sensor data.

The dataflow programming technique is a method of programming that is built upon sequentially connecting perception to action. Figure 5 is a good illustration of the dataflow philosophy. Data enters the program at the sensors and is output by the program as a motor control command to the vehicle. Moving from left to right in the program, we can see the data being converted into different forms. Each subVI could be looked at as a function that converts data from one form to another. The image data, for instance, gets converted to a polar distance array, then to a vector, and then to a motor control command. The effect of this data conversion is that the vehicle starts by "seeing" the scene around it and then "drives" in the appropriate direction.

The details of the subVIs for motor control, dynamic path selection, the Laser Rangefinder, and the GPS receiver, are not discussed in this document. The vision subVI is described in the next section. It is critical to the operation of the autonomous vehicle and is the focus of this thesis.

Chapter 4: Vision

Software written for any computer vision system has to first acquire an image from a camera and then process it to extract information. Design of the image acquisition system for our software was simplified by the availability of LabVIEW-compatible hardware and software.

4.1 Using LabVIEW for Computer Vision

A Standard S-video signal from a camera is sent to a National Instruments framegrabber that plugs into a PCI slot in the host computer. National Instruments framegrabbers are designed to have Direct Memory Access (DMA.) DMA reduces execution time of the vision algorithm because the image data is directly stored in memory without having to move through the main processor of the PC.

There are many standard ways to store image information. Some of the best known formats are bitmaps, JPEGs, and TIFFs. One of the advantages of using LabVIEW software for computer vision is that the programmer does not need to understand or convert between formats. LabVIEW uses a proprietary format called AIPD and all interactions with the image can be done through a set of high-level subVIs. For instance, performing an FFT on an image involves selecting the appropriate subVI and wiring in appropriate inputs. The following sections describe the image-processing block of the software.

4.2 Image Processing

The image-processing block is the part of the code that takes image data and sends information in the form of a polar passability map to the remaining part of the code. Given an image such as in Figure 6a, the image processing block would have to recognize the regions where the vehicle is not allowed to go, such as lines and simulated potholes. It would then have to convert this information into a polar passability array. After the lines and simulated potholes are recognized as the obstacles, perspective transformation can be performed to convert locations of obstacles into a polar passability array. The most difficult part of this process is going from the image in Figure 6a to the image in Figure 6b.

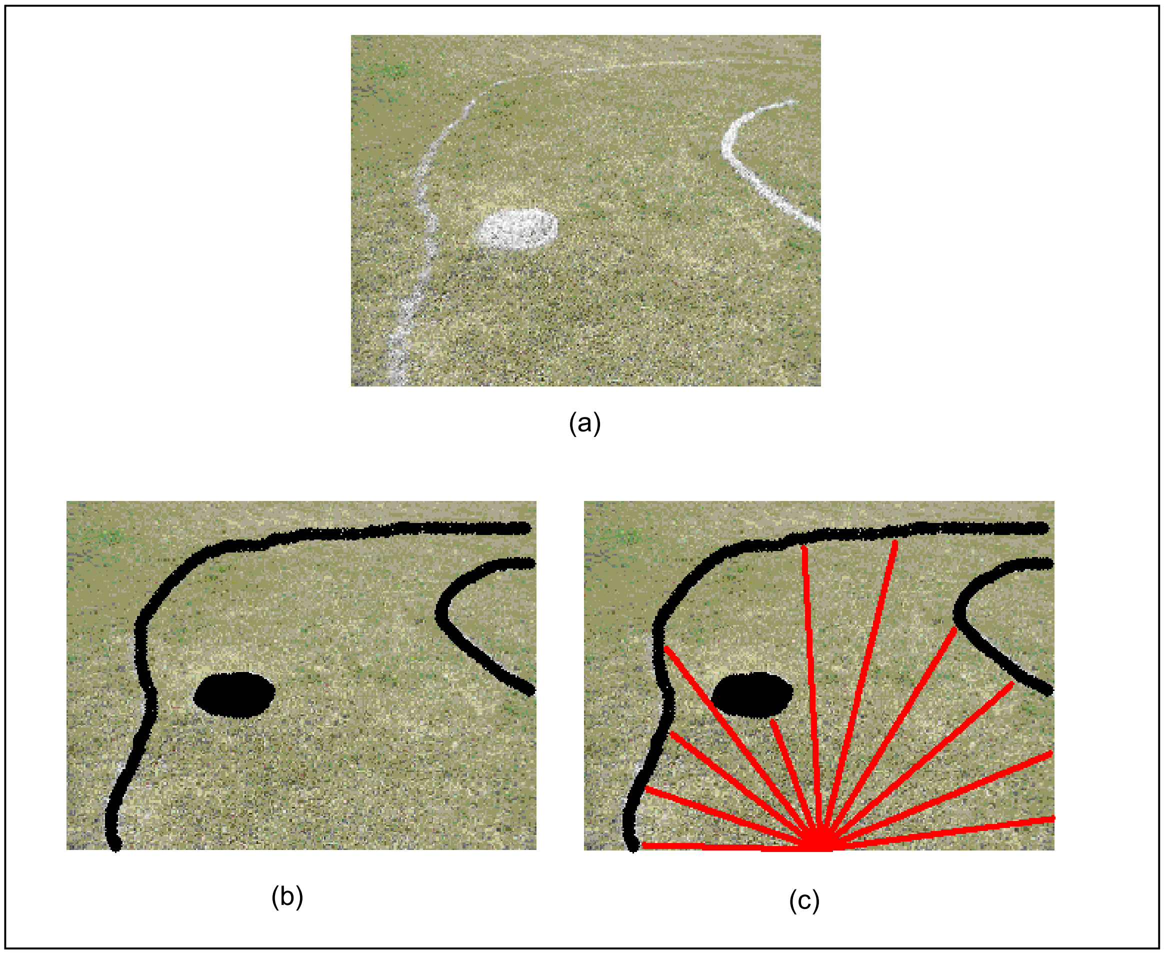


Figure 6. The goal of the software block is to (a) acquire an image, (b) select regions that represent lines or potholes, and (c) build a polar passability array.

4.3 The Need for Object Recognition

One of the most successful autonomous vehicle application softwares developed at Virginia Tech was called NavMan. NavMan was developed by David Conner [2000] using Visual C++. The image processing component of NavMan consisted of determining all parts of the image that were bright, such as lines and simulated potholes, and treating them as obstacles. This code worked very well at competition. Under most circumstances, the image processing code did guide the vehicle between the lines and around the obstacles. However, this code sometimes failed when patches of sunlight reflected from the grass looked like bright obstacles. Also, on occasion, when the vehicle faced a dashed line, it navigated through the gap between the dashes.

Object recognition could improve the performance of vehicles at the IGVC, because vehicles would not react to bright patches of sunlight, unless they looked like a line or a pothole. The remainder of this work focuses on recognizing the lines in the typical image encountered at the IGVC.

To extract lines from the image, it is first necessary to determine the defining characteristics of a line. These characteristics are listed below.

Color: In the case of white lines painted on grass, the primary characteristic distinguishing a pixel that is a part of a line from a pixel that is not a part of a line is the color. Chapter 5 describes in detail the process used to extract lines based on color.

Shape: Extraction of white regions is not enough to recognize lines in the image. Simulated potholes and patches of reflected sunlight can also appear white in an image, but these are not lines. Therefore, it is necessary to analyze the shapes of the white to distinguish lines from white regions that were not lines. Shape analysis is discussed in Chapter 6.

Chapter 5: Using Color Information

Color thresholding is the process of extracting all the pixels in an image that lie within a specific range of colors. Color information of each pixel in an image is typically represented as a point in a 3 dimensional space called a color space. Color thresholding is thus a process of selecting pixels that lie within a particular region of the color space.

5.1 Color Spaces

The most commonly used representation of color space is the additive color space, or the RGB color space. In the RGB color space, each color is represented as a mixture of varying amounts of the basic colors: red, green and blue. If the color of a pixel is stored in a computer as 3 bytes of data, each byte would correspond to a particular component of the color. For example, the color pure green would be stored as [0,255,0]. This indicates that the red and blue components are zero, while the green component would be maximum range or 255. By varying the proportions of the red, green and blue components, a very large range of colors can be described. 3 bytes of data can thus be used to describe 256 X 256 X 256, or more than 16 million different colors.

Colors can also be represented using color spaces other than the RGB. Different color space representations are appropriate for different applications. For example, a color space called the Cyan, Magenta and Yellow (CMY) color space is used to describe colors for printing applications. This color space is called the subtractive color space because it describes the color that is obtained by the subtraction of different amounts of cyan, magenta and yellow from white.

An investigation into color spaces showed that most color spaces are not perceptually uniform. For example, in the RGB color space, the colors that appear red to humans occupy a larger volume than the volume occupied by colors that appear yellow. This means that humans are less sensitive to minor differences in color in the red region than they are to differences in the yellow region. Further, if you started in the color space at a location that represented pure red and moved 10 units in a certain direction, you might still be in a red region, but if you moved 10 units in another direction, you might end up in a totally different color region. Because most color spaces are not perceptually uniform, they are not well suited for extraction of colors. If a human looked at a color image and divided the image into four regions based on color, the result would depend on the human perception of color. A similar operation done by a computer would give different results.

The HSL color representation was chosen for our application for 2 reasons. One, it decouples the color information from the intensity component and two, the chromaticity is closely related to the human perception of color [IMAQ Vision Concepts Manual, 2000.] Color spaces such as 'L*u*v*' or 'L*a*b*' have been designed as perceptually uniform color spaces, where numerical distance in the space is proportional to the perceived color difference. However, the HSL color space was found to be sufficient for extracting white regions for this application, and thus the computational cost of using perceptually uniform color spaces was not justified.

5.2 Color Thresholding

To extract regions in the picture that are white and could possibly be lines, it is necessary to select regions in the HSL colorspace that correspond to white regions. Figure 2 shows histograms of a sample picture. Notice how regions with different colors have different frequency profiles in the HSL colorspace. These histograms show that white regions can be extracted from an image by thresholding in the colorspace if appropriate threshold values are chosen.

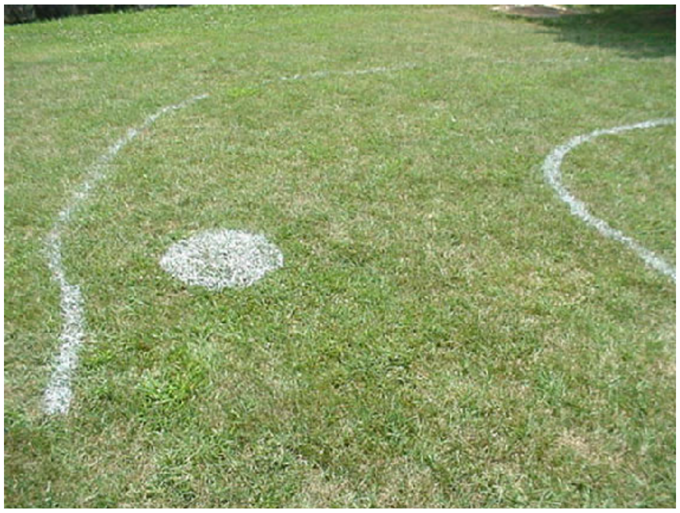
Figure 7a shows a sample image. The green regions in the image were manually selected and a histogram of the HSL values was plotted as shown in Figure 7c, Figure 7e, and Figure 7g. Similarly, the white regions in the image were selected and the histograms plotted as shown in Figure 7b, Figure 7d, and Figure 7f. Based on observation of these histograms for many sample images, the following values were chosen as ranges for thresholding.

Hue plane – 30 to 135

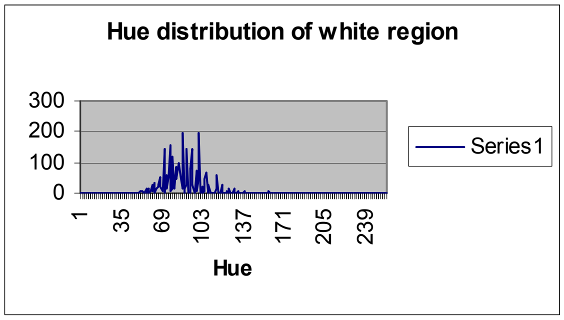
Saturation plane – 0 to 26

Luminance plane – 158 to 255

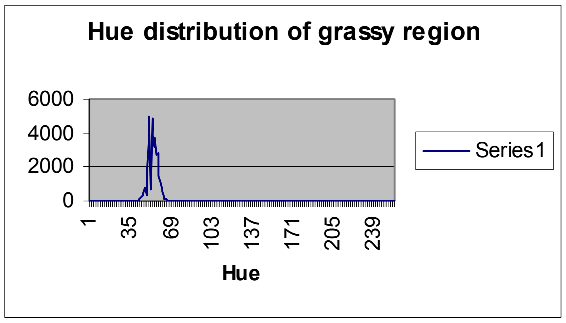
Pixels with values that lie within this range have a high probability of being white pixels.



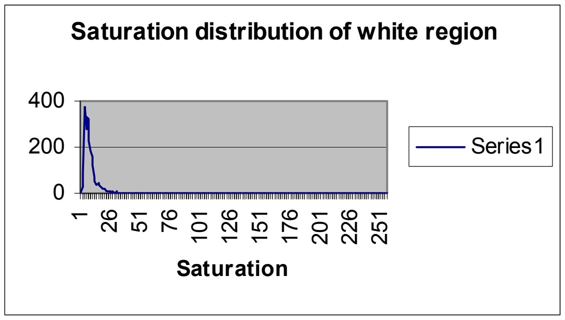
(a)

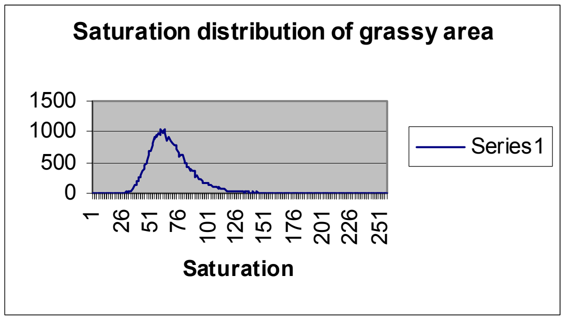


(b)



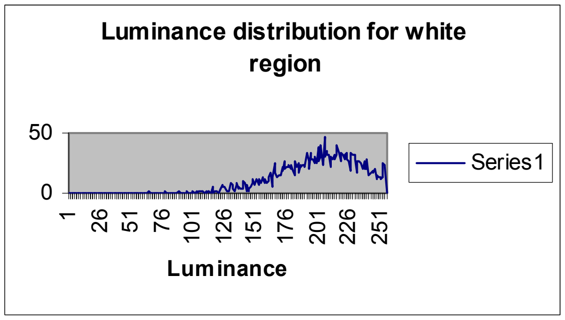
(c)

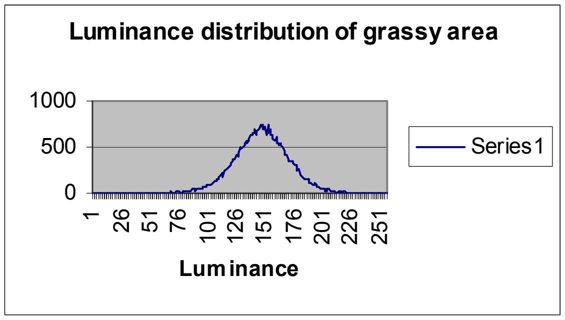




(d)

(e)





(f)

(g)

Figure 7. Sample image and a comparison of HSL distributions in the white and grassy areas of the image

5.3 Binary images

The result of a thresholding operation is a binary image in which all background pixels have the value 0 and all foreground pixels have the value 1. For purposes of display, the selected pixels are given a value of 255 and they appear white in an image.

Sample results of this color thresholding operation can be seen in Appendix A. It can be seen that this thresholding operation works well on some images where the lines are clearly defined. It does not work as well on images where sunlight on the grass creates bright regions that appear close to white. However, this was not considered a shortcoming of the thresholding operation since these falsely selected pixels are pixels that appear white even to the human eye.

The color thresholding operation takes about 270 ms on an image of size 640 X 480 pixels. Running the same operation on smaller images showed that the processing times varied almost linearly with image size. Running this process on a faster computer would decrease the processing time involved.

5.4 Grayscale thresholding

To save image processing time, thresholding could be performed on a grayscale image rather than on a color image. Thresholding a grayscale image of size 640 X 480 takes only about 3 ms, which is about 1% of the time taken to threshold a color image. This reduced processing time is important for an application that makes real-time decisions on an autonomous vehicle. Figure 8 shows a grayscale image thresholded with several different values.

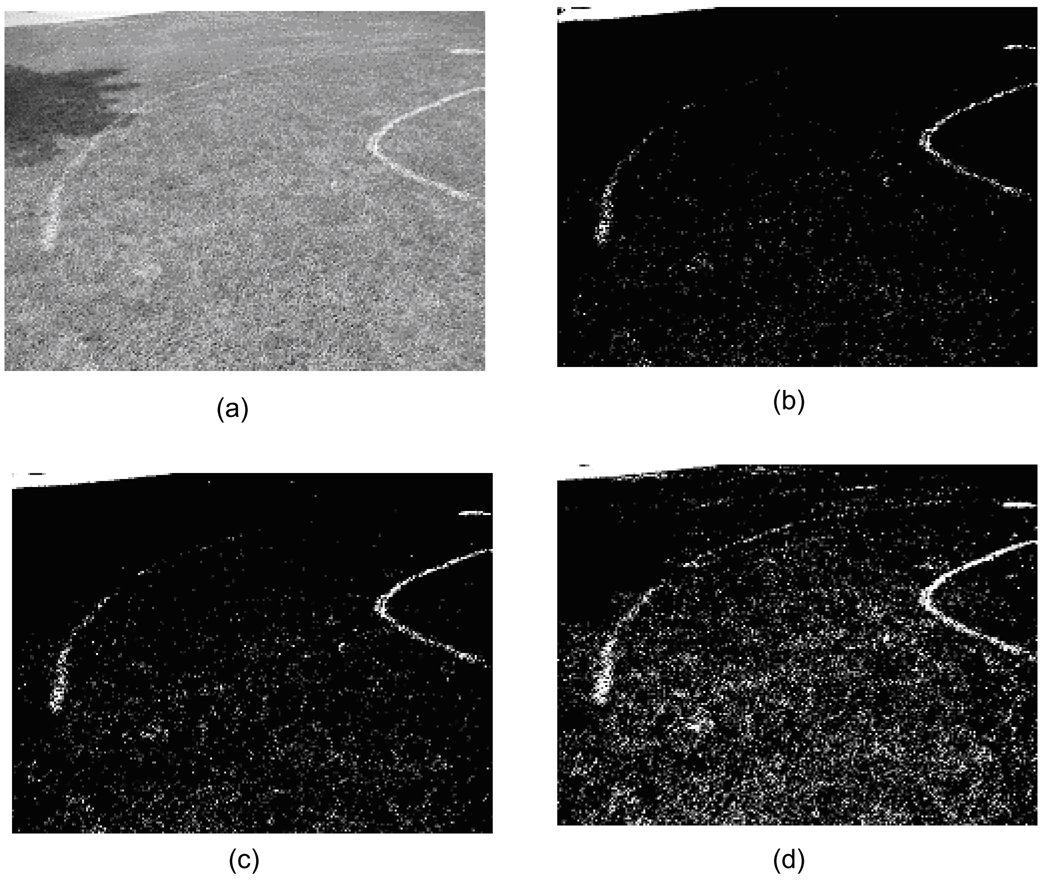


Figure 8. Effect of grayscale thresholding. The grayscale image (a) is thresholded at the following levels: (b) 210, (c) 200, and (d) 190.

A look at the results of grayscale thresholding shows that along with the white regions, many regions get selected that are not actually white regions. Trying to eliminate these false positives wrongly eliminates areas in the image that are parts of lines. Increasing the range of threshold values improves the chances of detecting all pixels that are part of a line. However, this also increases the chances of selecting areas of the picture that are not white.

Dynamic grayscale thresholding techniques were explored using the LabVIEW Vision Builder. Sample images of size 640 X 480 took about 15 ms to be dynamically thresholded. The dynamic thresholding techniques attempted were clustering, entropy, metrics, moments and intervariance. The best results obtained were with the moments method. Dynamic thresholding did not work too well on these images because the histograms of these images were not bimodal. Figure 9 shows the results of dynamic thresholding using moments on the sample image of Figure 8a.

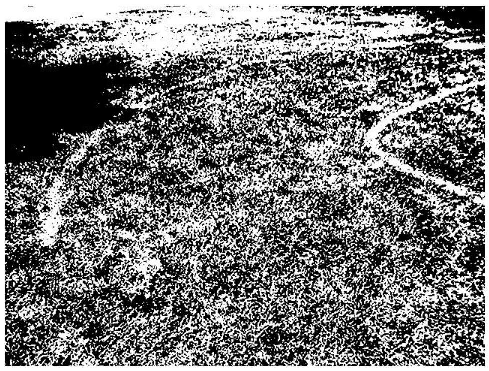


Figure 9. Result of dynamic thresholding using moments

A comparison of the results of color thresholding and grayscale thresholding is shown in Figure 10. The image in Figure 10b is the result of color thresholding. It appears to have brighter lines and a brighter pothole because more pixels in this region are selected. The lines and potholes are less bright in Figure 10d, which is the result of grayscale thresholding. Reducing the threshold value during grayscale thresholding would brighten the lines to the level seen in Figure 10b, but would also add to the noise in the picture. More examples comparing color and grayscale thresholding can be found in Appendix A.

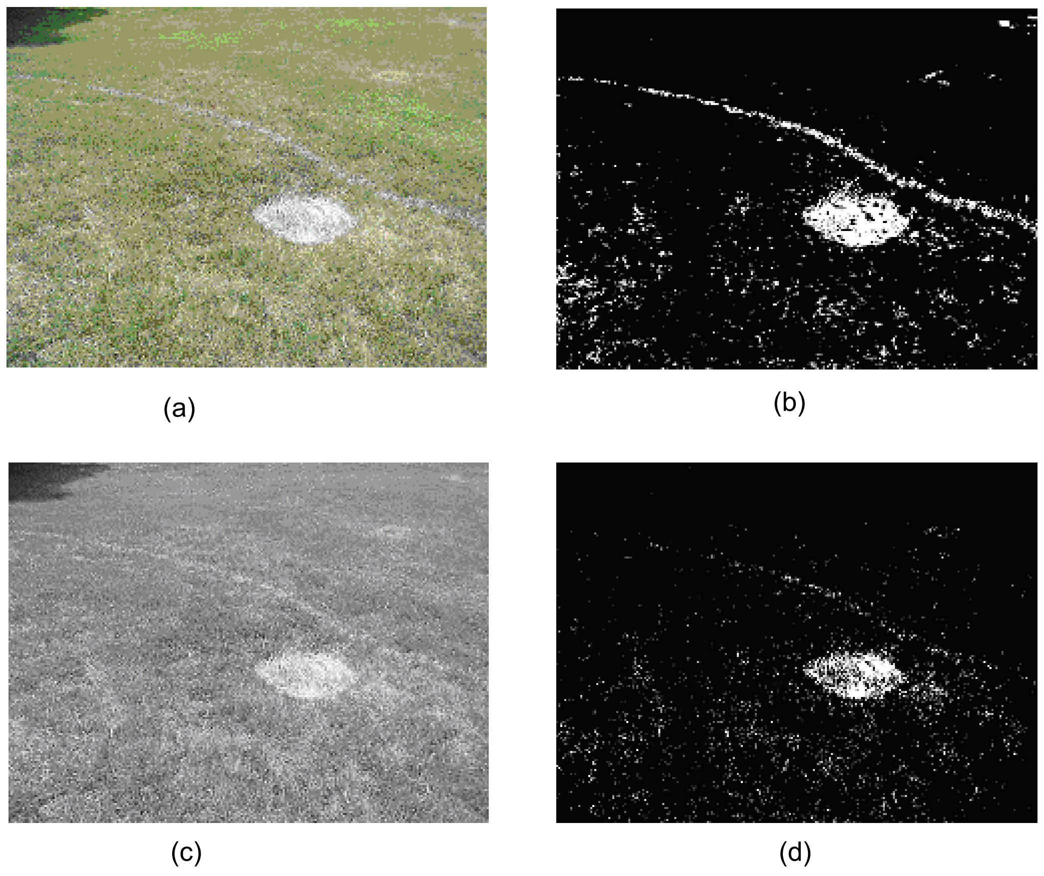


Figure 10. Comparison of color thresholding with grayscale thresholding. Color thresholding is performed on the image in (a) to obtain image (b). Image (a) is converted to a grayscale image shown (c), and thresholded to obtain image (d).

A look at the results of thresholding shows that it is necessary to discard regions of the image that are white but are clearly not part of a line. The next section describes the algorithms that were used to extract lines from binary images.

Chapter 6: Using Shape Information

Ideally, the results of the thresholding operation on an image would give us a binary image such as in Figure 11b, which shows a set of well defined lines on a black background. Color thresholding performed on the typical image of the IGVC course often gave us binary images that looked like Figure 11c. The problem faced is therefore a problem of pattern recognition.

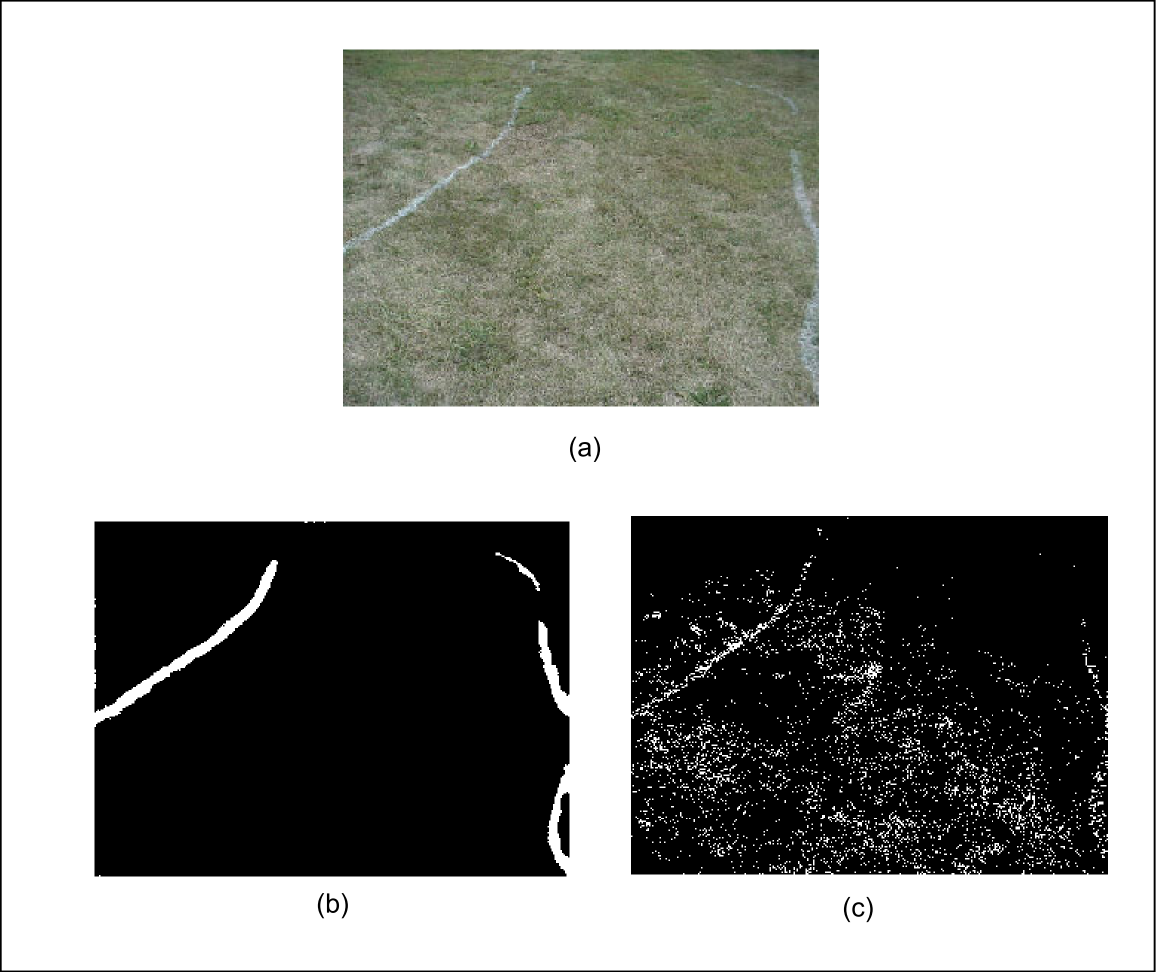


Figure 11. The need for binary morphological analysis. Color thresholding an image (a) would ideally result in an image similar to (b), but usually results in an image similar to (c).

We see in Figure 11c that the lines are not well defined. Most of the lines have holes in them because of blades of green grass showing even in the painted areas. Patches of reflected sunlight show up on the binary image as white amorphous particles. Thresholding out reflected sunlight is difficult because reflected sunlight appears to be just as white as the lines. The next section describes a few of the binary morphological operations that were performed on the binary image to prepare it for shape analysis.

6.1 Binary Morphological Operations

Morphology is the study of shape. Morphological operations performed on binary images exploit the shapes present in images to perform functions needed for machine vision applications [Jain, Kasturi, Schunk, 1995.] Binary images consist of pixels that are either background pixels or foreground pixels. Groups of foreground pixels that are connected to each other are called connected components or particles.

Removal of small particles

Particles in the binary image that are small correspond to sunlight reflected by individual blades of grass or other spurious white points in the image, such as a gum wrapper or a clover blossom. Therefore, removal of small particles helps reduce noise in an image. Removal of small particles in an image is performed by application of a lowpass filter on the image. For a given filter size of N, the lowpass filter eliminates particles whose widths are less than N-1 pixels [IMAQ Vision Concepts Manual, 2000.] The effect of removal of small particles is shown in Figure 12b.

Filling of holes

The Euler number of a particle is defined as the number of particles minus the number of holes. For example, a particle shaped like the alphabet N has an Euler number of 1, while a particle shaped like the alphabet B has an Euler number of -1. It is noticed that none of the expected shapes in the image, such as lines or potholes have holes in them. Each of the particles in the image was checked for Euler number, and holes filled up. The effect of this operation is shown in Figure 12c.

Smoothing of particles

The binary image at this point had only particles of a certain minimum size, and with an Euler number of 1. Most of these images appear to have jagged edges. To smooth the shapes of these images, the process of successive dilation and erosion was adopted. The process of dilation involves making a particle larger. This is done by selecting the pixels neighboring the particle and making them part of the particle. Erosion is the opposite of dilation and involves making a particle smaller by removing all pixels at the border of the particle. Successive dilation and erosion is typically used as a method for removing small holes in particles, while erosion followed by dilation is used as a method for removing small noise particles. For our application, dilation followed by erosion is used for its smoothing properties as illustrated in Figures 12d and 12e. We see that many sharp edges and curves in the particle are removed as a result of this operation.

It is noticed that many of the above operations require user-determined parameter values, such as number of erosions and filter size. The software has been designed such that these parameter values could be changed through the user interface.

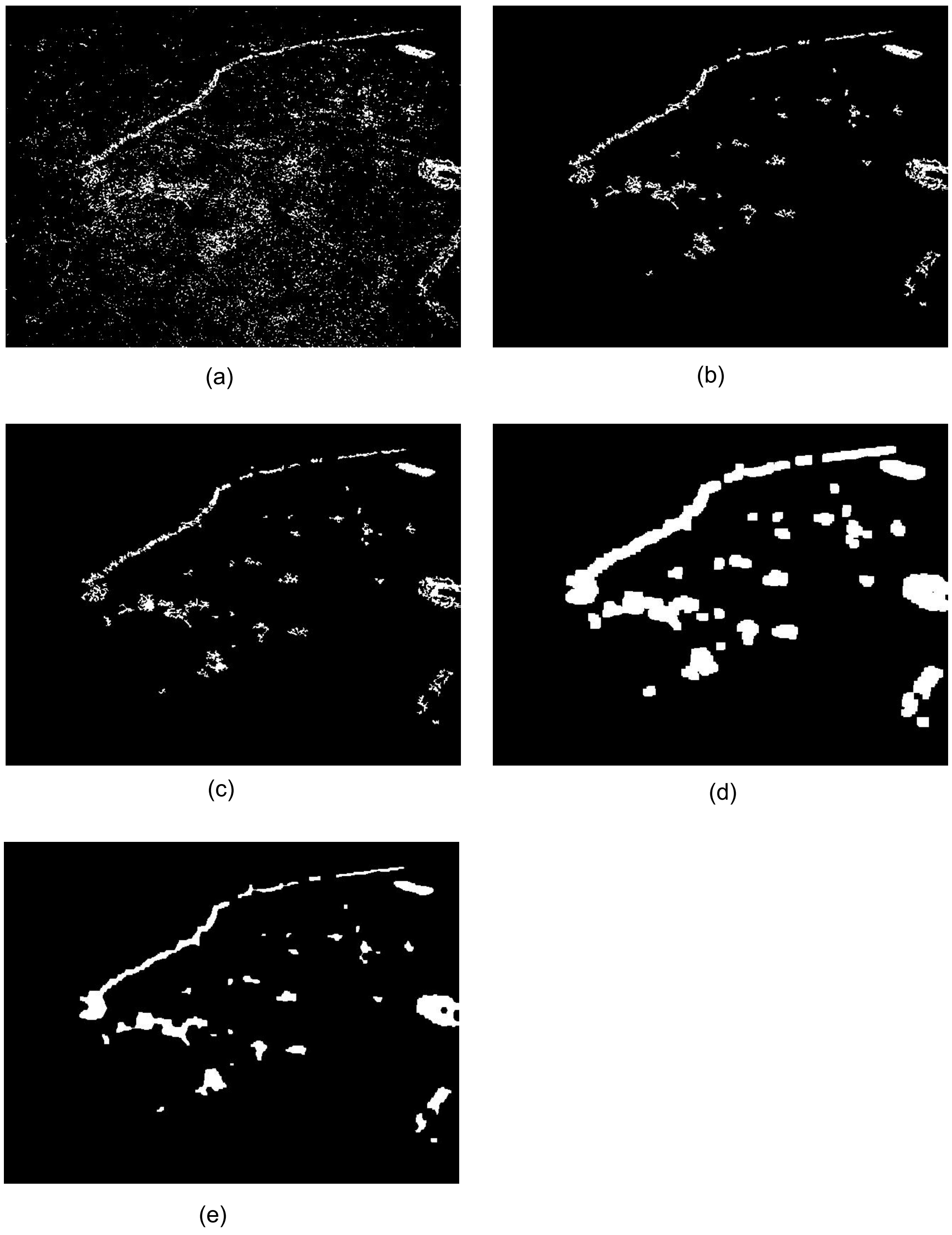


Figure 12. Binary morphological operations performed on an image (a). Shown here are the effects of a low-pass filter (b) followed by a hole-filling operation (c), dilation (d) and erosion (e).

The above "cleaning up" of the binary image prepared the image for particle analysis. The aim of the particle analysis is to recognize particles that belong to boundary lines. In certain cases when lines are attached to potholes, it is necessary to recognize only the parts of the particle that belong to the lines. The next section describes particle shape analysis.

6.2 Shape Analysis

Shape analysis involves looking at a particle and classifying it as a particular shape. Commonly used shape analysis methods involve measuring features of the particle such as width, height, size, location, moments and orientation. An extensive discussion of pattern recognition techniques can be found in textbooks by Devijver and Kittler [1980], Schalkoff [1992], and Duda and Hart [1973.] A few of the best known particle measurements and their relevance to our problem are described below.

Particle size measurements such as width, height, area and moment are calculated using pixels as units. The area of a particle cannot indicate whether particles are lines or not because the lines are not of a known size. Straight lines such as those in Figure 13a have small widths and low moments about their axes of orientation. However, curved lines such those in Figure 13b have neither of these properties. Thus particle size measurements cannot be used to recognize lines. Another well-known particle measurement used for shape analysis is the Heywood circularity factor, defined as the ratio of the particle perimeter to the perimeter of a circle of the same area. A perfect circle would have a Heywood circularity factor of 1, while a square would have a circularity factor of 1.128. Longer particles would tend to have a high Heywood circularity factor. The Heywood circularity factor can be used to classify particles such as those in Figure 13c that are not very distorted. The particle shown in Figure 13d has a very high Heywood circularity factor but yet it is more a circle than a line and illustrates a case where the circularity factor fails to classify a shape. Other elongation measurements such as the Waddel disk diameter and the elongation ratio are not discussed here because they too depend on measurements of the perimeter and area of a particle, and are as prone to failure as the Heywood circularity factor.

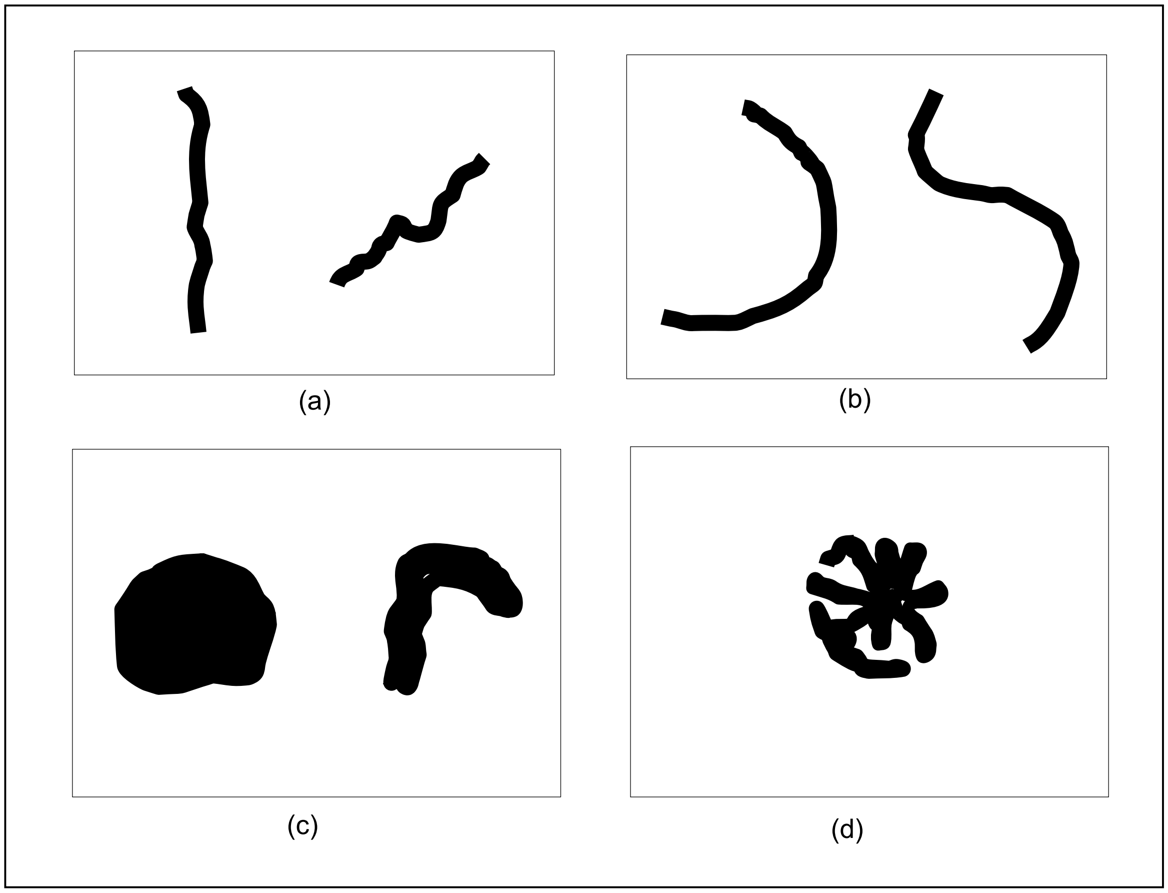


Figure 13. Shape features cannot always be used to distinguish lines from other particles. Features of lines such as low width and low moment about axis of orientation would work on the lines in (a), but would fail with the curved lines in (b). The Heywood circularity factor can be used to distinguish shapes such as in (c), but would incorrectly classify (d) as a line because of the large perimeter to area ratio.

Pattern matching technology is successfully used today in many applications. Pattern matching can be performed using a variety of techniques such as cross-correlation and pyramidal matching. These techniques can be used to locate a known reference pattern in an image. Since the reference pattern we are looking for could be in a variety of shapes, ranging from small straight lines to long S-shaped lines, pattern matching is inappropriate for our application.

Another problem in shape analysis occurs when a line touches a simulated pothole or a large patch of reflected sunlight as shown in Figure 14. Separation of the line from the pothole can be done only if we are able to recognize the lower part of the particle as a line, and the upper part as a pothole. However, to recognize the lower part of the particle as a line, we would have to first separate it from the pothole. This Catch-22 situation makes it difficult to use shape properties to recognize and extract lines from the typical binary image encountered in our problem.

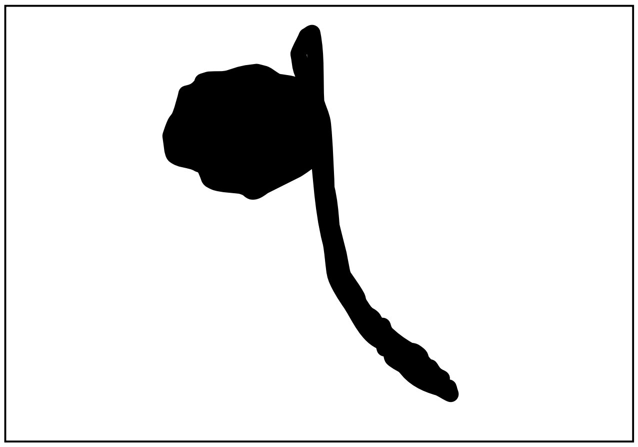


Figure 14. The problem of overlapping particles. Separation of particle into 2 parts is necessary for recognition of each part. But, recognition of each part is necessary for separation of particle into 2 parts.

To extract lines from a binary image, it became necessary to go deeper into the definition of a line, and attempt to use this definition for the recognition of lines.

Definition of a Line

Lines, by definition, are shapes that are formed by the tracing of a point. Since a point by definition is of infinitesimal width, lines also are considered to have an infinitesimal width but a finite length. In the real world, points are approximated by circles of a specific diameter. A line drawn by the tracing of such a circle would have a width equal to the diameter of the circle. Thus, two defining characteristics of a line could be written as

i. The width of the line at any location is constant.

ii. The length of the line is far greater than the width.

The following section describes how the above observations were used for the recognition of lines in an image.

Particle labeling

To analyze the particles in a binary image, it is necessary to treat each particle as separate from other particles. To achieve this, the particles in the image were labeled. Particle labeling is a method of assigning a particular value to each pixel belonging to a particle. After labeling a binary image, it looks like a grayscale image. However, each particle has a unique number assigned to it – the value of each pixel in that particle.

6.3 The PCW Array

To analyze a particle based on the definition of a line, it is necessary to measure the width of the particle at a variety of points. At this point in the program, the problem of line recognition is a matter of gathering particle data and analyzing it to decide on locations of lines. This was achieved by converting the image data into a dataset of numbers called the PCW array. The PCW array (Particle, Center, Width Array) is initially a 2-dimensional array with 4 columns. The image is scanned row-wise and particle data is entered into the PCW array. The formation of the PCW array is best described with an example.

Figure 15a shows a sample image of size 10 X 10 pixels. The zeros indicate background pixels and the non-zero pixels are particles. Two particles – particle number 100 and particle number 200 – are visible in the image. The image is scanned row-wise and the PCW array constructed. The first column indicates particle number, the second and third columns indicate the center of the particle in a particular row (rounded to an integer value), and the fourth number indicates the width of the particle in that row. The PCW array generated from the image is shown in Figure 15b. A scan through all the rows of an image would result in a PCW array that contained information about each particle in the image. The component of this information that interests us is the width of the particle.

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**200**	**200**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**200**	**200**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**100**	**100**	**100**	**0**	**0**	**0**
0	**0**	**0**	**0**	**100**	**100**	**100**	**0**	**0**	**0**
0	**0**	**0**	**100**	**100**	**100**	**100**	**100**	**0**	**0**

(a)

Particle Number	Row	Column	Width
200	2	4	2
200	3	4	2
100	8	6	3
100	9	6	3
100	10	6	5

(b)

Figure 15. Generation of PCW array. Data in the image array (a) is converted into the PCW array format (b)

The number of rows in the PCW array varies depending on the number, shape and size of particles in the binary picture. For most images, the memory taken up by the PCW array is smaller than the memory taken up by image data, because the PCW array does not explicitly store locations of the background pixels. Also, in the PCW array, groups of continuous pixels are represented by a location and a width. For example, the image data in Figure 15 has 100 elements while the same data in PCW array form has only 20 elements. The PCW array contains complete information about the binary image in a compressed form convenient for analysis.

Recalling the dataflow philosophy of LabVIEW, we see that data that entered the program as a color image is now data in the form of a PCW array. The next few sections describe the processing of the PCW array only, since at this point the image is not needed for further processing.

6.4 Processing the PCW Array

Perspective Correction of Image Data

The data in the first 4 columns of the PCW array represent shapes that are distorted by perspective. The Figure 16 shows the perspective projection model. Since the shape in the image plane is different from the shape on the ground plane, and because we are attempting to recognize a shape that exists in the ground plane, it becomes necessary to correct the image for perspective. Perspective correction could have been applied to the original image obtained from the camera. It has been intentionally applied this late in the process in order to reduce the computation time. This reduction in computation time is because the PCW array data is typically at least a couple of orders of magnitude lesser than the data in a color image.

Figure 16 shows the side view of a camera looking down at the ground in front of it [Conner, 2000.] Perspective correction from pixel coordinates to real-world coordinates on the ground plane is a 2-step process – conversion in the image plane from pixel units to real-world units, and then conversion from real world units in the image plane to real-world units in the ground plane. To simplify the process of conversion, it is assumed that the image plane passes through the point where the focal axis intersects the ground plane. This assumption makes it possible to switch cameras or lenses on the camera without having to update focal length values on the software.

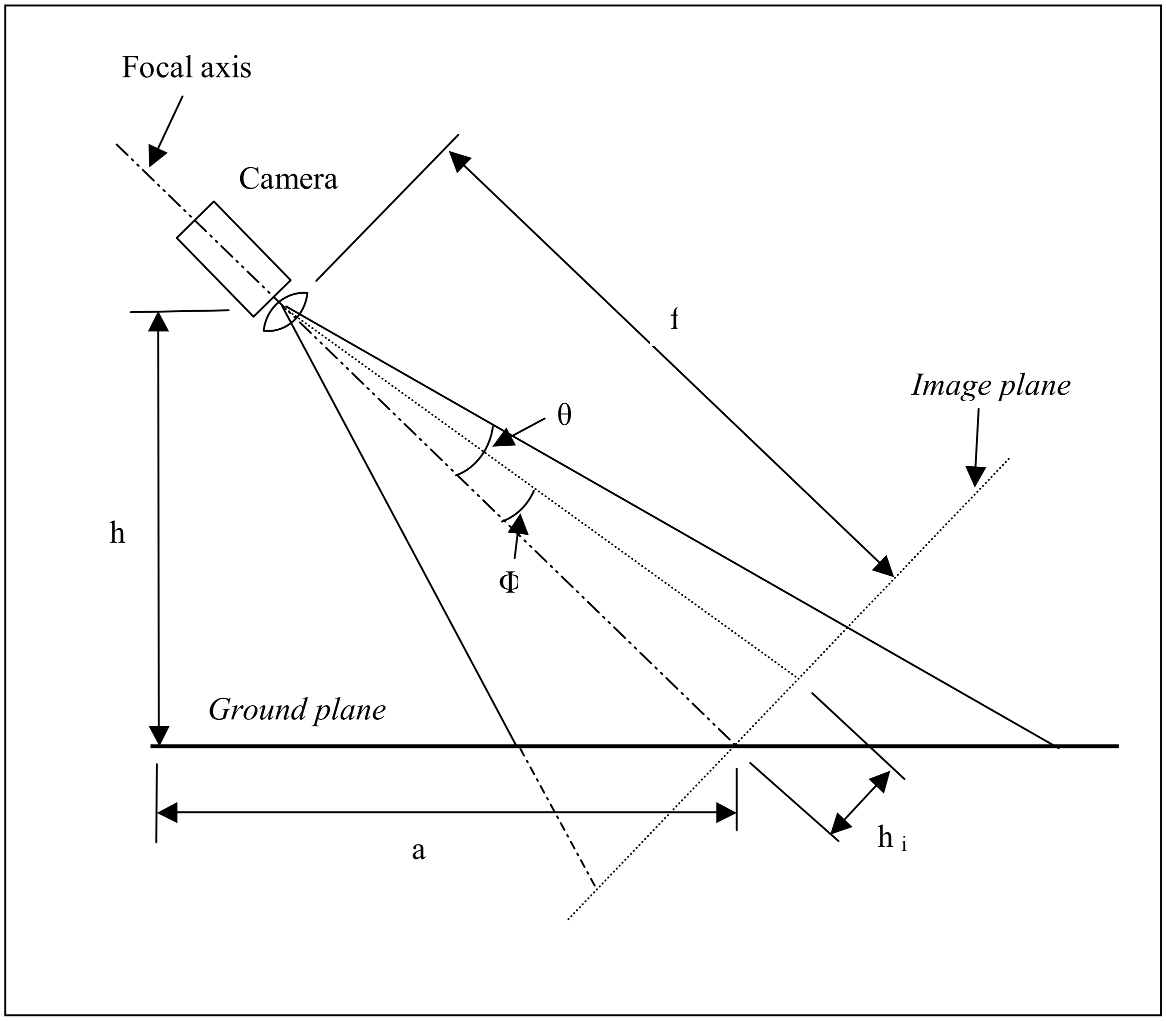


Figure 16. The perspective projection model for a camera on an autonomous vehicle looking down at the ground in front of the vehicle.

The actual width of a particle in mm is given by

$$W_A = \frac{hWWp}{I_w \tan\left(\theta - \tan^{-1}\left(\frac{h_i}{f}\right)\right) f}$$

Since each row in the PCW array contains data about a particle that is 1 pixel height, vertical image correction was applied to 1 vertical pixel height. The height of 1 pixel is given by the equation

$$\frac{W}{I_W}\left[\cos(90-\theta)+\sin(90-\theta)\left[\frac{h_p+1}{\tan(\theta-\phi_2)}-\frac{h_p}{\tan(\theta-\phi_1)}\right]\right]$$

where h = height of camera in mm (input by user)

a = distance on ground between camera and image plane (input by user)

θ = declination angle of the camera (calculated from a and h)

Iw = Image width in pixels (obtained from image data)

hp = height in pixels in the image plane (obtained from row column of PCW array)

hi = height in mm in the image plane (calculated from hp)

f = focal length in mm (calculated from a and h)

W = length in mm of a horizontal line drawn on the ground at center of the image and extending till the left and right edge of the image (input by user)

Wp = Width of particle in pixels (obtained from width column of PCW array)

Calibration of a camera on an autonomous vehicle involves updating the software with information about camera parameters such as location and orientation on the vehicle, and focal length. To simplify the process of calibration, the software is designed for camera calibration using just 3 values: h, A, and W. Since measurement of camera declination angle (θ) has been physically inconvenient in the past, this has been avoided. All three parameters necessary for calibration are distances and can be measured using a tape measure. By using the value W and assuming the focal plane as shown in Figure 16, it has become unnecessary to know the focal length or CCD density of the specific camera.

After perspective correction of the PCW array, the PCW array looks like the array shown in Figure 17.

Particle Number	Row	Column	Width in pixels	Width in mm	Pixel height in mm
200	2	4	2	24	4
200	3	4	2	22	4
100	8	6	3	7	3
100	9	6	3	6	3
100	10	6	5	10	3

Figure 17. The PCW array after perspective correction contains particle information in real-world coordinates.

We see from the new PCW array that particle widths in pixels in the lower regions of the image correspond to smaller actual widths than for similar particle widths in the upper regions of the image. This matches with the expectation that particles closer to us appear wider than those farther away. Looking at the pixel height column, we see that a single row in the lower part of the image corresponds to a lesser height than a single row in the upper part of the image. This corresponds to the expectation that a shape painted on the ground would appear more vertically compressed the farther away it is.

Analysis of PCW Array

Information about each particle in the image is present as a subset of the PCW array. Each particle was separated out and the widths analyzed. A plot of the widths of a sample particle is shown in Figure 18a. Before attempting to select parts of the particle that could be lines, a box filter (or averaging filter) was applied to this plot. The window width was selected to be equal to the width of the expected line. A plot of the widths of the particle, after application of the box filter, is shown in Figure 18b.

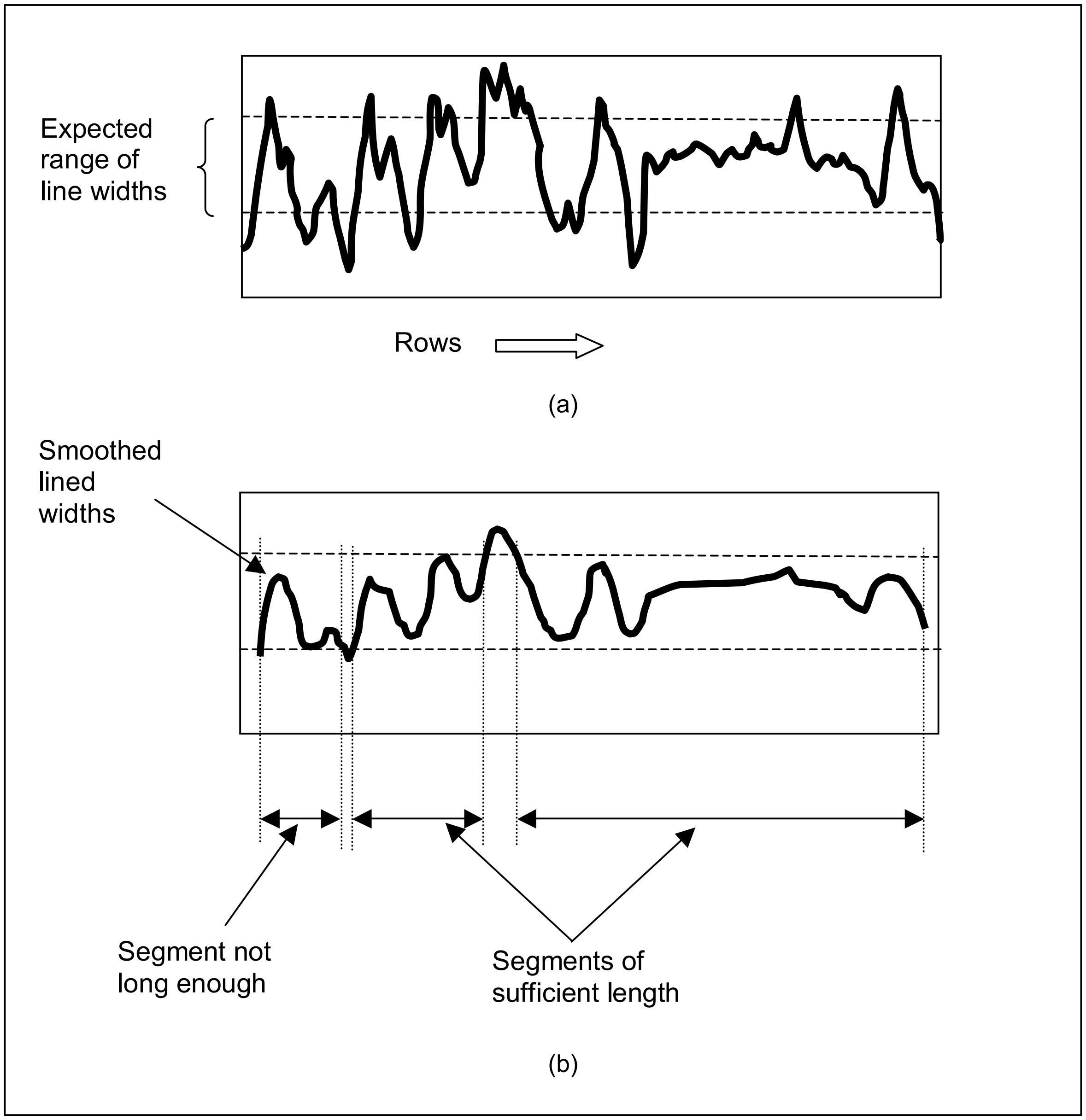


Figure 18. Selection of lines based on widths and lengths

The data in the PCW array is then thresholded by width. In other words, all sections of the particle that do not fall within a certain range around the expected line width are discarded. The values of expected line width and upper and lower limits for the line width are variables that are entered by the user before running the program. All parts of the particle that remain after this step have widths that qualify them to be parts of a line. The next step involves thresholding the particles for length. All particles that are not of a certain minimum length are discarded in the PCW array. This minimum length is also a variable that is input by the user before running the program. A minimum length of thrice the expected width is recommended to qualify a shape as a line. Figure 18b shows the selection of 3 regions that lie within the width range. The second and the third of these regions qualify as lines because they satisfy the length criterion.

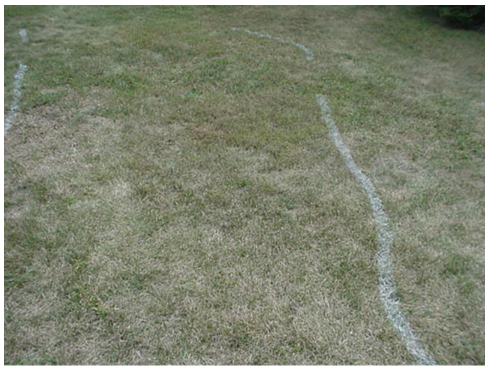
At this point the PCW array consists of a list of points that indicate locations of lines. The results of the image-processing block are discussed in the next section.

Chapter 7: Results

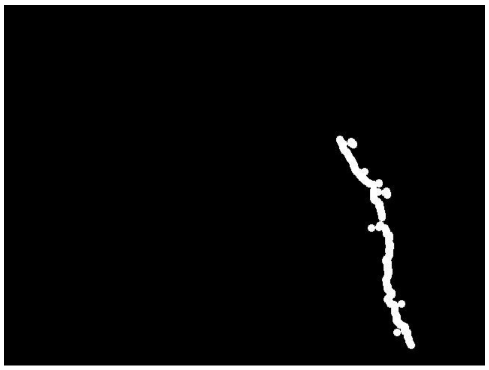
The performance of the image-processing block was tested on sample images of a course painted on a field at the Virginia Tech campus. To enable testing of the image-processing module, it was modified to analyze images stored on file. The final PCW array was used to redraw a binary image. For display purposes, white circles of diameter 10 pixels were drawn at all locations where lines were detected in the image.

7.1 Selection of Parameters

The performance of the software depends on the selection of parameter values such as acceptable range of values for line width, and expected minimum line length. The lines painted on the sample course were approximately 40 mm wide. Figure 19 shows a sample image and the effect of varying parameter values. The image in Figure 19a was processed with a width range from 20 to 40 mm, and a minimum length threshold of 120 mm. The image in Figure 19b shows the result of the operation. Reducing the length threshold to 100 mm and 80 mm, we obtain the results shown in Figures 19c and 19d respectively. We see that reducing the length threshold increases the possibility of incorrectly selecting particles that are not long enough to be lines. The effect of reducing the width range is shown in Figure 19e.

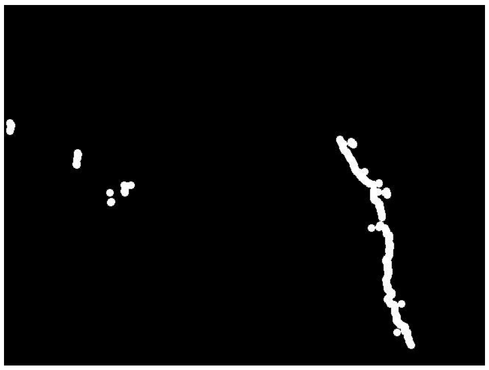


(a) Original Image



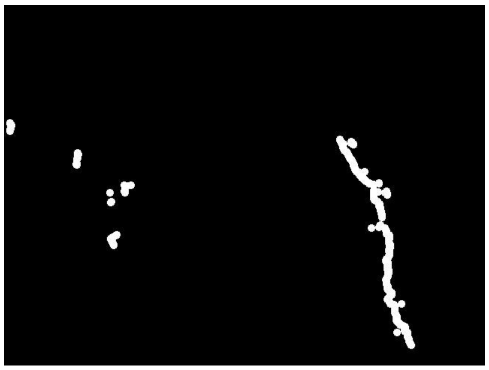
(b) Range: 20 – 60 mm

Length: >120 mm



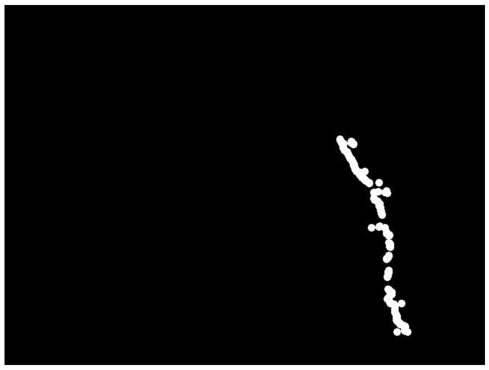
(c) Range: 20 – 60 mm

Length: >100 mm



(d) Range: 20 – 60 mm

Length: >80 mm



(e) Range: 30 – 50 mm

Length: >120 mm

Figure 19. Resulting images using different parameter values

7.2 Effectiveness

The effectiveness of the software is best described with sample images. Figure 20a shows a sample image that was successfully processed to obtain the image shown in Figure 20b. Figures 20c & 20d illustrates one of the most significant achievements of the work presented in this thesis. Notice that the software successfully separated and extracted the line from the connected white circle. Figure 20e shows another sample image. When processed, Figure 20f was obtained. An interesting phenomenon we observe in this result is the software's tendency to fail in the upper regions of the image, just like humans would. Appendix B shows the results of the image processing on a few more sample images.

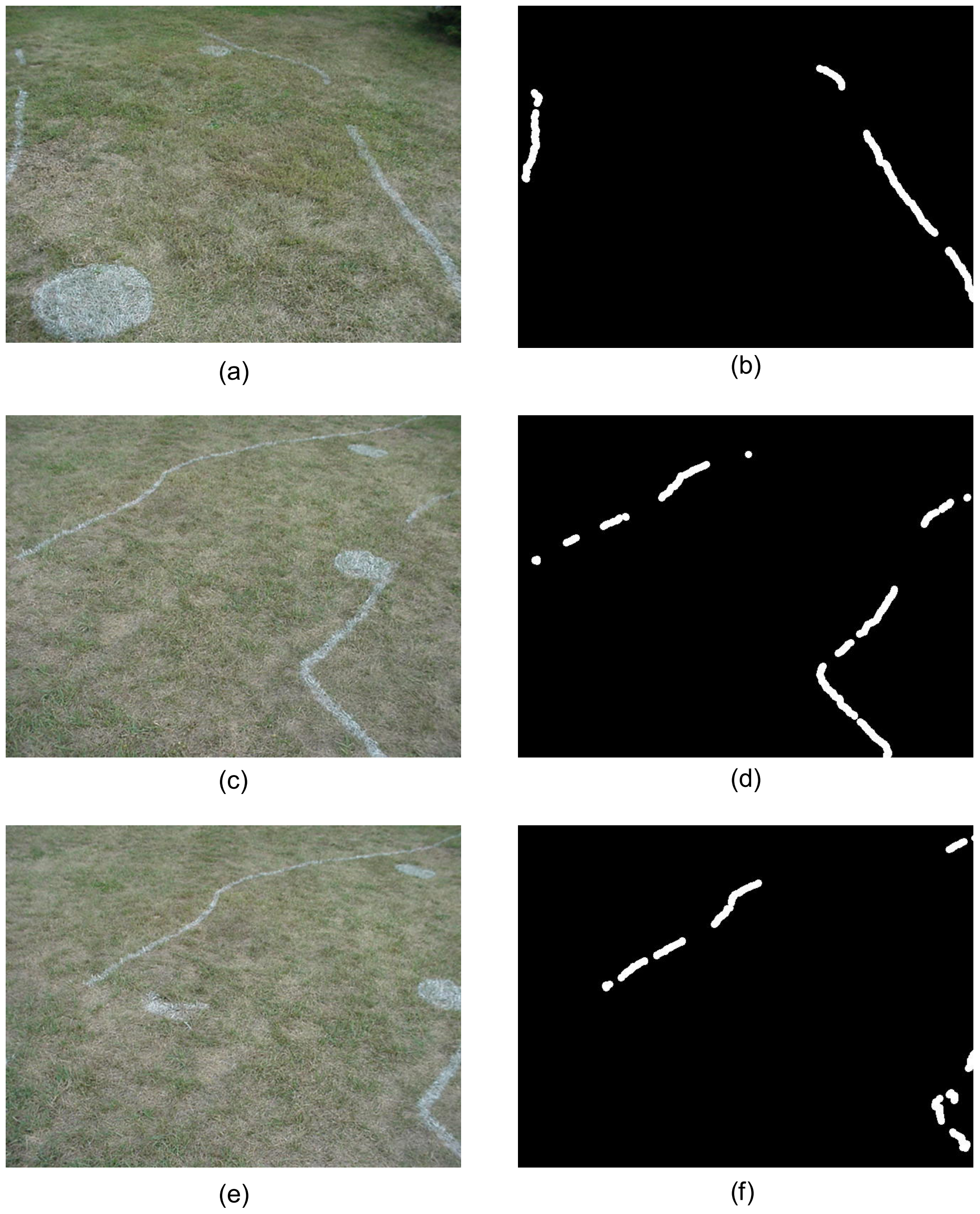


Figure 20. Sample images and results

7.3 Recommendations for Further Research

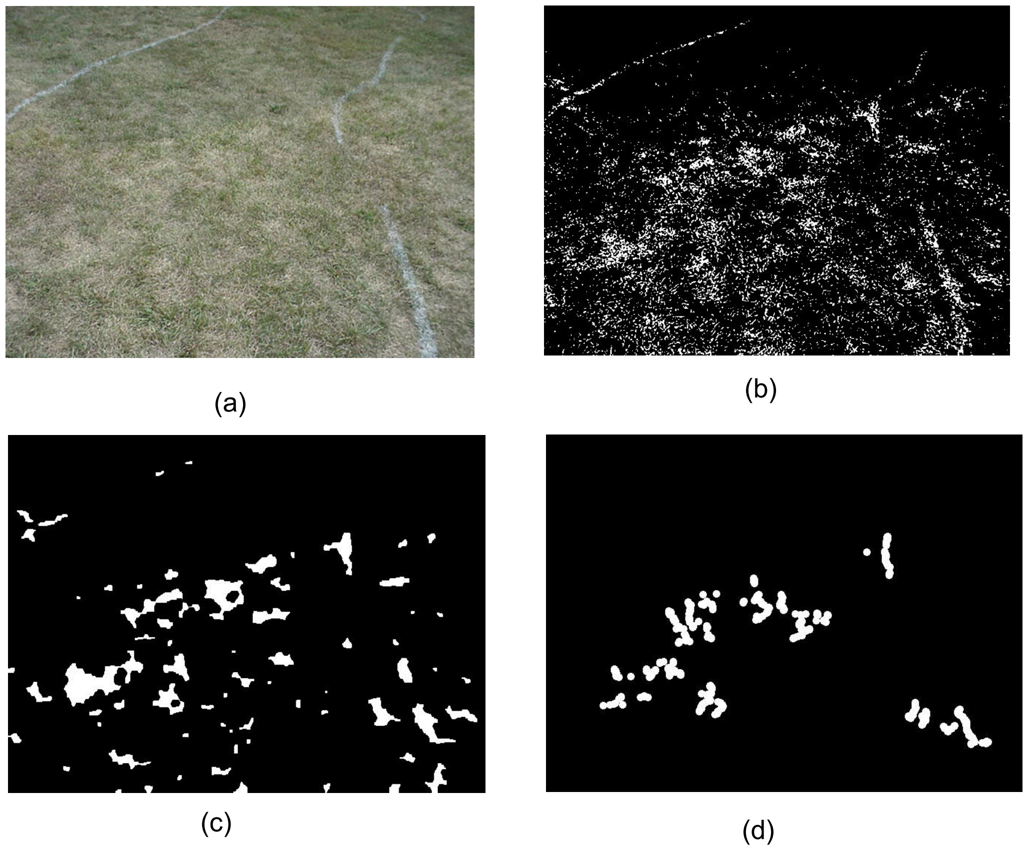
Testing of the image-processing block of the software showed certain cases where the image-processing method developed in this thesis failed to recognize lines. Such failures occurred on about 5% of the images tested. Three of the most common reasons for failure are described below, and recommendations made for further research.

Color Thresholding

Figure 21a shows a sample image. The result of the image-processing block is shown in Figure 21d. To analyze the root of the failure, the image data was recorded at different stages. Figure 21b shows the image after color thresholding. Notice that the selection of white areas has not been very successful. Our present thresholding method involves thresholding between static ranges in every color plane. In other words, in the color space, thresholding is performed by adjusting the location and widths of a rectangular block. Dynamic selection of threshold values would vastly improve the autonomous quality of the algorithm. Another suggestion to future researchers is to explore the possibility of thresholding a 3-dimensional irregular surface instead of being constrained to a rectangular block. Another probable solution is to use a perceptually uniform color space. This would allow using a spherical surface for thresholding.

Further Analysis of the PCW Array

Figure 21c shows the image after binary morphological operations are performed on it. Notice in Figure 21d that many particles have been incorrectly selected as lines. This occurs because lines were selected only on the basis of their widths and lengths. The center locations of the lines along each row are present in the PCW Array, but they are not used for analysis. In an ideal line, the center locations in the PCW Array would vary gradually, if at all, as we moved row-wise in the image. Further analysis of the center locations is recommended for improving the performance of the image processing software.


(a)

(b)

(c)

(d)

Figure 21. Sample image illustrating failure case

Detection of Horizontal Lines

The shape analysis performed on the image would not be able to recognize horizontal lines in the image because only horizontal particle widths were considered. Analysis of particle widths in the vertical direction would help in the recognition of lines that are closer to horizontal than vertical.

References

Borenstein, J., and Koren, Y., "The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots," *IEEE Journal of Robotics and Automation,* Vol 7, No 3 (June 1991).

Brooks, R. A., "New Approaches to Robotics," *Science* (September 1991).

Conner, D. C., *Sensor Fusion, Navigation, and Control of Autonomous Vehicles*, master's thesis (Blacksburg, VA: Virginia Tech, July 2000).

Craig, J. J., *Introduction to Robotics: mechanics and control* (Reading, MA: Addison-Wesley, 1989).

Devijver, P. A., and Kittler, J., *Pattern Recognition: A Statistical Approach* (Englewood Cliffs, NJ: Prentice-Hall International, 1980).

Duda, D. O., and Hart, P. E., *Pattern Classification and Scene Analysis* (New York: Wiley-Interscience, 1973).

M^{c}Kerrow, P.J., *Introduction to Robotics*, (Reading, MA: Addison-Wesley, 1991).

IMAQ Vision Concepts Manual, (Austin, TX: National Instruments Corporation, October 2000).

Jain, R., Kasturi, R., and Schunck, B. *Machine Vision* (New York: McGraw Hill, April 1995).

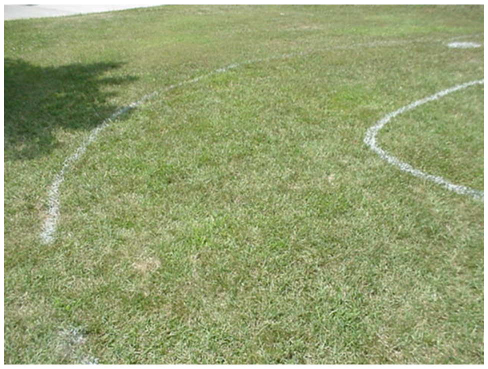
Kedrowski, P. R., *Development and Implementation of a Self-Building Global Map for Autonomous Navigation*, master's thesis (Blacksburg, VA: Virginia Tech, April 2001).

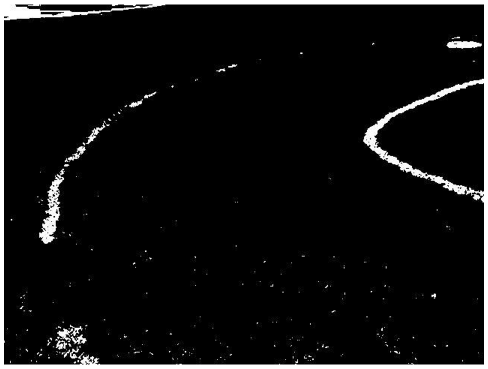
R. Schalkoff, *Pattern Recognition: Statistical, Structural and Neural Approaches* (New York: John Wiley & Sons, 1992).

Appendix A: Comparison of Grayscale and Color Thresholding

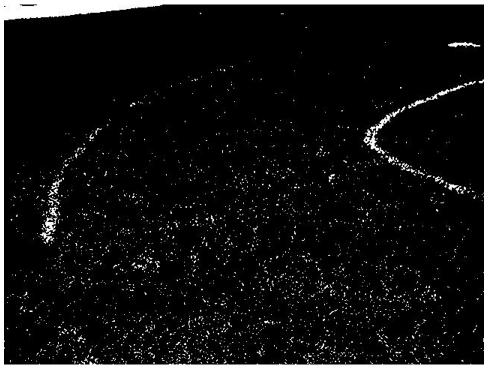
The images shown in this appendix compare the results of color thresholding with the results of grayscale thresholding. Color thresholding was performed using the following threshold values: Hue – 30 to 135; Saturation – 0 to 26; Luminance – 158 to 255. Grayscale thresholding was performed by selecting all pixels greater than 200.

Sample 1

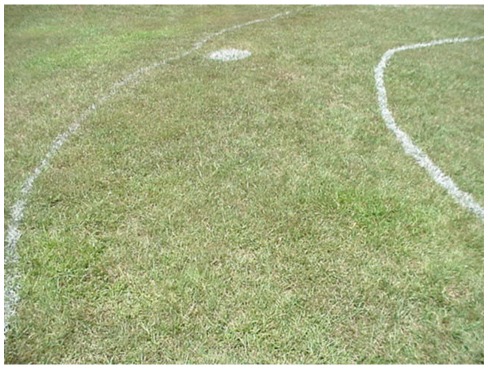

Original Image

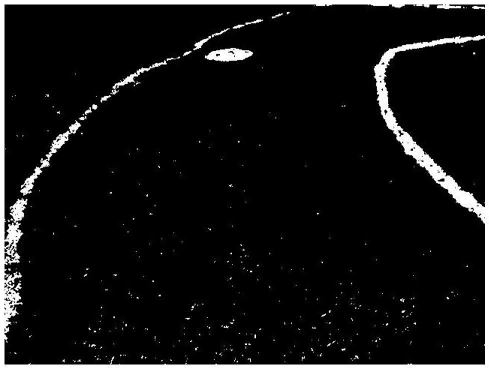

Color Thresholding

Grayscale Image

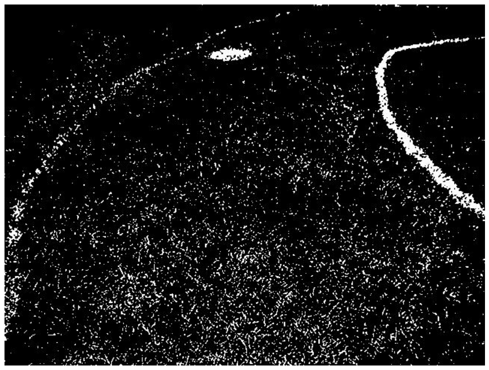

Grayscale Thresholding

Sample 2

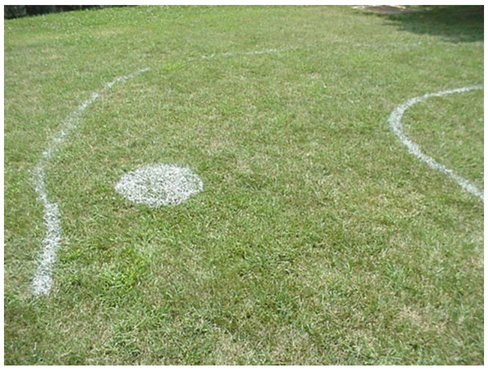

Original Image

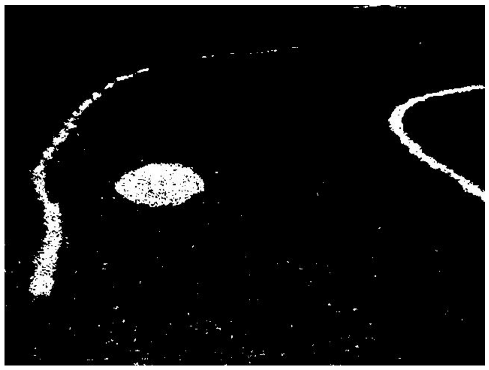

Color Thresholding

Grayscale Image

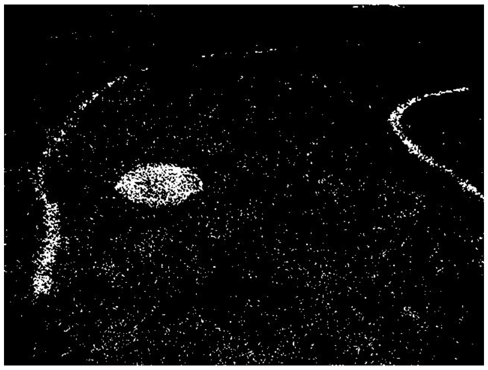

Grayscale Thresholding

Sample 3


Original Image

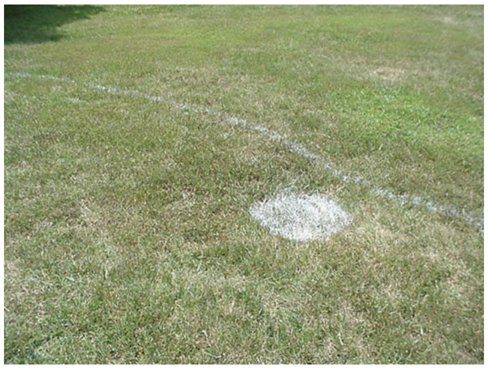

Color Thresholding

Grayscale Image

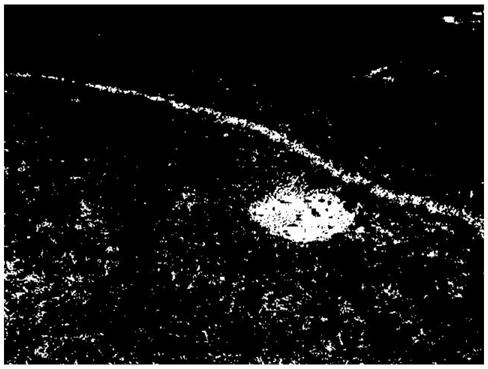


Grayscale Thresholding

Sample 4

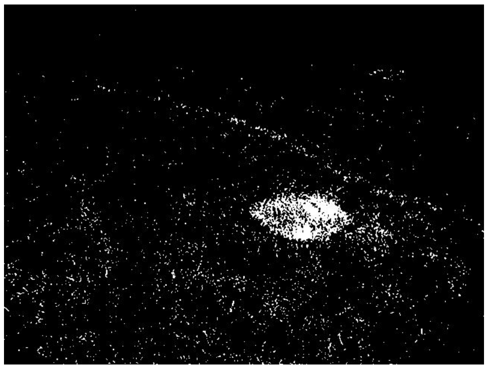


Original Image



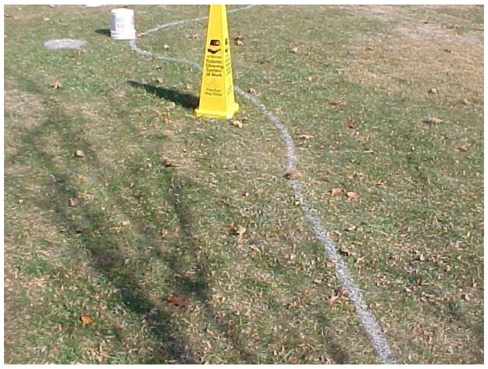
Color Thresholding

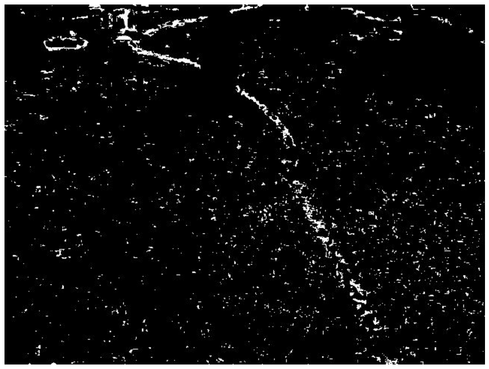
Grayscale Image



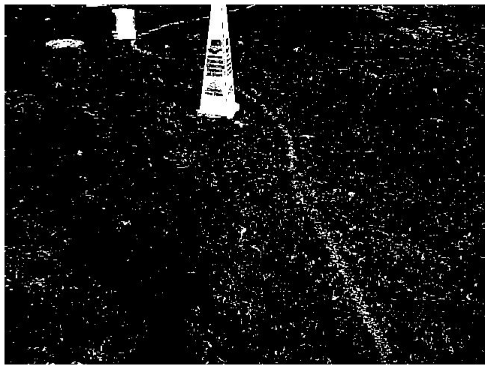
Grayscale Thresholding

Sample 5


Original Image


Color Thresholding

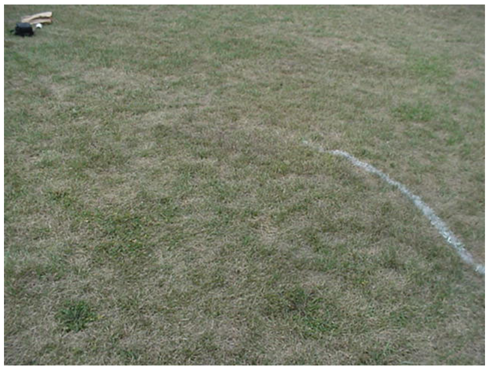
Grayscale Image

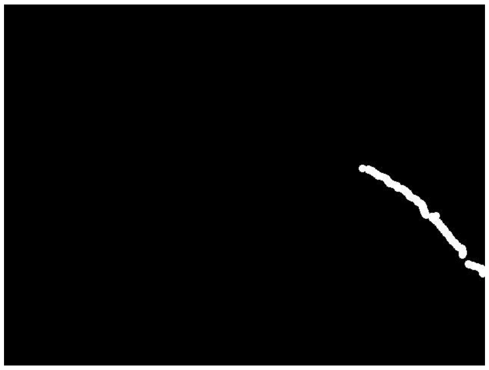

Grayscale Thresholding

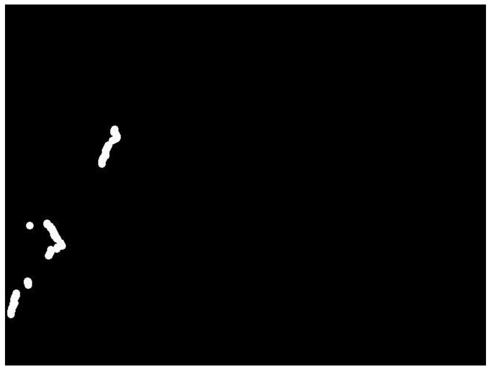
Appendix B: Sample Images and Results

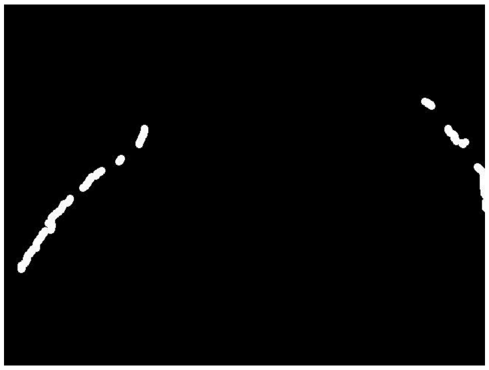
This appendix shows sample images and regions that were selected as lines. Parameter values are listed below.

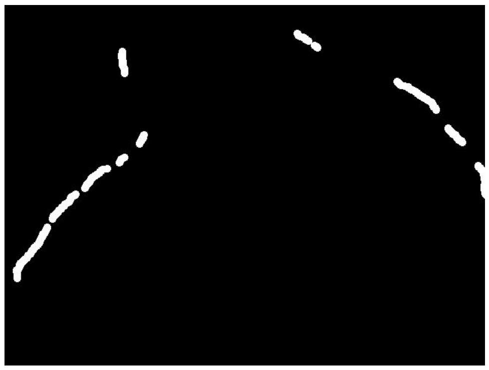
Parameter	Value
Hue Range	75 to 135
Saturation Range	0 to 25
Luminance Range	125 to 190
Number of Erosions and Dilations	4
Expected Line Width	40 mm
Line Width Range	20 mm to 60 mm
Line Length	> 200 mm
h	2000 mm
a	3000 mm
w	2500 mm

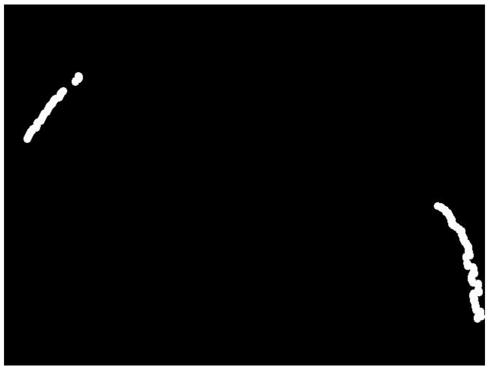


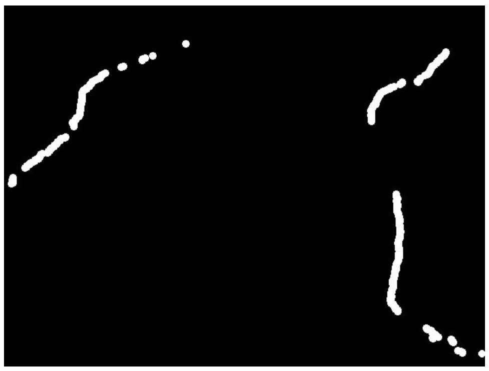


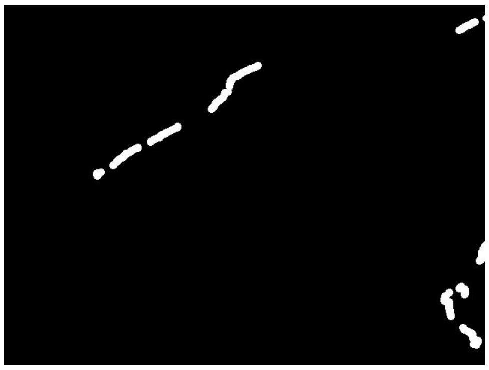












Vita

Sudhir Gopinath was born on May 5th, 1975, to Tara and S. Gopinath, both architects practicing in Bangalore. He completed his basic schooling at National English School, Bangalore in 1991 and his Pre-University from S. Nijalingappa College, Bangalore in 1993. He then got his Bachelor of Engineering degree from Ambedkar Institute of Technology, Bangalore University, in 1999. After working for the Microland group of companies for a little less than a year, he joined the master's program in Mechanical Engineering at Virginia Polytechnic Institute and State University, Blacksburg, Virginia to specialize in robotics. He completed his master's program in 2002, and has been working since early 2003 as a systems engineer at Sensing Systems LLC, San Mateo, California.