2-1-2008

A monocular color vision system for road intersection detection

Michael Scott Kurdziel

A Monocular Color Vision System for Road Intersection Detection

by

Michael Scott Kurdziel

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Supervised by

Associate Professor, Department of Computer Engineering Dr. Juan C. Cockburn
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
February 2008

Approved By:

Dr. Juan C. Cockburn
Associate Professor, Department of Computer Engineering
Primary Adviser

Dr. Marcin Łukowiak
Assistant Professor, Department of Computer Engineering

Raymond Ptucha
Adjunct Faculty

Thesis Release Permission Form

Rochester Institute of Technology

Kate Gleason College of Engineering

Title: A Monocular Color Vision System for Road Intersection Detection

I, Michael Scott Kurdziel, hereby grant permission to the Wallace Memorial Library reporduce my thesis in whole or part.

Michael Scott Kurdziel

Date

Dedication

I dedicate this project to my parents for their endless support.

Acknowledgments

I would like thank the Cornell DARPA Urban Challenge team. My work with them inspired this project and gave me the knowledge to make it possible. I would also like to acknowledge Professor Juan C. Cockburn from the Department of Computer Engineering for his help completing this thesis.

Abstract

Urban driving has become the focus of autonomous robotics in recent years. Many groups seek to benefit from the research in this field including the military, who hopes to deploy autonomous rescue forces to battle-torn cities, and consumers, who will benefit from the safety and convenience resulting from new technologies finding purpose in consumer automobiles. One key aspect of autonomous urban driving is localization, or the ability of the robot to determine its position on a road network. Any information that can be obtained for the surrounding area including stop signs, road lines, and intersecting roads can aid this localization. The work here attempts to combine some previously established computer vision methods to identify roads and develop a new method that can identify both the road and any possible intersecting roads present in front of a vehicle using a single color camera.

Computer vision systems rely on a few basic methods to understand and identify what they are looking at. Two valuable methods are the detection of edges that are present in the image and analysis of the colors that compose the image. The method described here attempts to utilize edge information to find road lines and color information to find the road area and any similarly colored intersecting roads. This work demonstrates that combining edge detection and color analysis methods utilizes their strengths and accommodates for their weaknesses and allows for a method that can successfully detect road lanes and intersecting roads at speeds fast enough for use with autonomous urban driving.

Contents

List of Figures

List of Tables

Glossary

C

CDD Charged-Coupled Device, p. 6.

D

DARPA Defense Advanced Research Projects Agency, p. 1.

G

GPS Global Positioning System, p. 2.

GUI Graphical User Interface, p. 57.

I

IMU Inertial Measurement Unit, p. 2.

IPM Inverse Perspective Mapping, p. 11.

S

SCARF Supervised Classification Applied to Road Following, p. 13.

SVM Support Vector Machine, p. 16.

U

UNSCARF Unsupervised Classification Applied to Road Following, p. 15.

Chapter 1

Introduction

1.1 Problem Introduction

1.1.1 Introduction

Autonomous robotics has been the subject of research for many years. Recently, this technology has made its way into the automotive field. The military is actively looking to develop autonomous urban vehicles to rescue soldiers in battle-torn cities and automotive companies are integrating certain aspects of this technology into consumer cars for safety and comfort. The military's interest in autonomous robotics is seen most prevalently in a series of challenges set forth by the Defense Advanced Research Project Agency (DARPA). These challenges include the 2004 and 2005 DARPA Grand Challenges, where teams were invited to develop autonomous vehicles to travel over 70 miles through an off-road desert course. The most recent competition was the 2007 DARPA Urban Challenge, where teams developed robots capable of traveling autonomously through a 60-mile urban course while negotiating urban traffic, stop signs, intersections, passing lanes, and various other difficult urban scenarios.

To complete the challenging task of autonomous urban driving, the robots had to recognize and understand many different actions involved with driving, many of which humans perform with little thought. The most important of these tasks is

localization. Localization refers to the ability of a robot or person to know its position not only between the lanes of the road being traveled on, but also its position on the road network its attempting to navigate. Most current autonomous vehicles rely heavily on sophisticated systems combining high-accuracy Global Positioning Systems (GPS) and expensive Inertial Measurement Units (IMU) to find their exact location on the earth. This position is then compared to a map of surveyed road networks. This allows vehicles to find the road closest to itself on the map, thereby telling the robot which road its on and where on that road its currently located. Despite the extreme sophistication of these localization systems, the possibility for error still remains. GPS suffers from a myriad of possible errors including satellite geometry, multipath, clock inaccuracies, and atmospheric effects, any of which can cause an inaccurate reading or even a complete lack of GPS information. IMU-based localization works by accumulating acceleration changes in all directions, allowing the robot to know how far and how fast it has moved by integrating the acceleration. The error in this type of system is introduced when even the smallest of inaccuracies is present in the sensor. This inaccuracy accumulates over time and can lead to a significant error given enough time without correction. During the 2007 DARPA Urban Challenge, some teams like that from Cornell University, used cues from the robot's surroundings as landmarks to help reinforce the robot's assumption of its location. These environmental cues included stop lines and turns in the road. When the robot approached a stop in the road and identified its associated stop line, it was able to place itself next to that stop line on a map of the road network, which identified each stop line location. The robot could also use turns in the road to better localize itself. When the robot is driving along a straight road, there may be some uncertainty as to where along that road the it is actually located. When the robot traverses a turn in the road, it knows its exact location since that turn is located in only one location on the map. While these two additions were greatly beneficial in correcting the robots localization system, they

were unable to correct the robots position on a long, straight road. The work in this paper attempts to solve that problem by identifying intersecting roads in front of the vehicle. Using a single color camera, the robot gains the ability to not only detect the lane location and geometry that its driving on, but also the boundaries of the road and the existence and location of any visible intersecting roads in front of the robot. With this information, the robot gains the ability to better estimate its position along a road by understanding its location in relation to other visible intersecting roads.

1.1.2 Problem Definition

The main objective of this work is to develop a method for detecting intersecting roads using a single color camera mounted to the top of a moving vehicle. The system was developed with the following constraints:

- The vehicle must be traveling on roads with discernable boundaries like painted road lines.
- Objects like snow, water, or other vehicles must not heavily obstruct the road.
- The intersecting road must be of similar color to the road where the vehicle is traveling.
- The system must be able to identify the location and geometry of the lane in front of the vehicle.
- The system must be able to identify any intersecting roads on either the driver or passenger side of the vehicle and determine its distance from the camera.
- The system must run fast enough for use in an autonomous vehicle traveling at urban speeds under 40 mph. Ideally, the speed of the system should be greater than 1 Hz.

- The system must perform with a single color camera mounted to the top of a vehicle.

Figure 1.1: Typical Road Intersection Image

Figure 1.1 shows a scenario where an intersection would ideally be detected. The application of camera-based road and intersection detection systems go beyond events like the DARPA Urban Challenge. Consumer automobiles could benefit from similar systems to detect situations where the car drifts outside its lane on a highway or to correct consumer GPS navigation systems, which traditionally rely on less accurate GPS information then that typically used by autonomous robots.

1.2 Computer Vision Systems

1.2.1 Image Basics

A basic computer vision system consists of a camera attached to a computer. The camera can be either color or grayscale depending on the intended use. Many

vision systems utilize high-quality specialty cameras with either very high resolution, very low image noise in the image, or a very high frame rate. A high-resolution image has a more dense pixel concentration in the image it produces, thereby capturing more detailed information. A camera with low noise produces a cleaner image, where all or most information in the image accurately represents the information in the world that it captured. A high frame rate camera is able to capture images at a very fast rate, often greater than 30 times per second.

The camera of a typical computer vision system captures an image and transfers it to the computer as a series of pixels. Each pixel has a red, green, and blue value each ranging between 0 and 255.

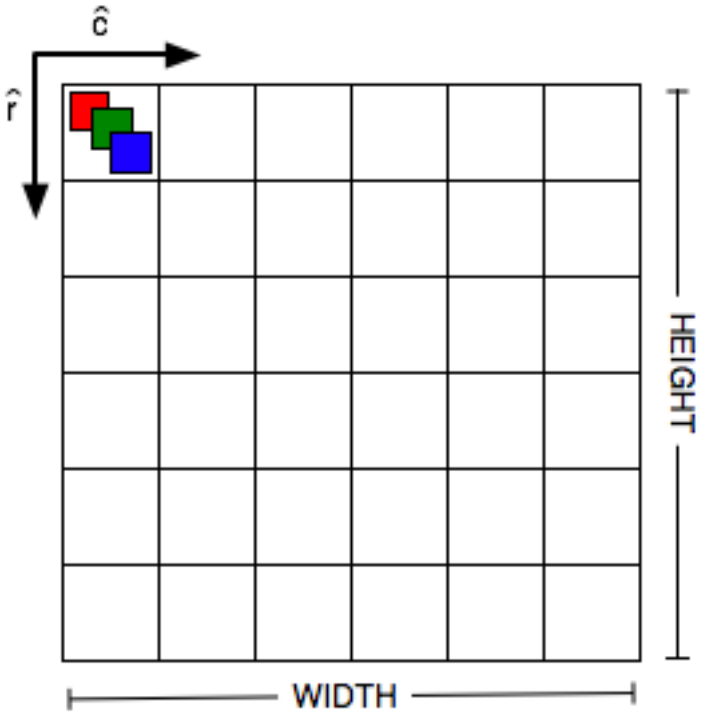


Figure 1.2: Image Frame

Figure 1.2 shows an image frame from a camera. $\hat{u}$ and $\hat{v}$ are the axes of the image frame centered at the optical axis I_0. $\hat{r}$ and $\hat{c}$ are the axes of the image frame originating at the upper left corner of the image. Pixels in the image are most

commonly indexed along the $\hat{r}$ and $\hat{c}$ axes corresponding to the row and column of the image.

Cameras have both extrinsic and intrinsic parameters that factor into their usage with a computer vision system.

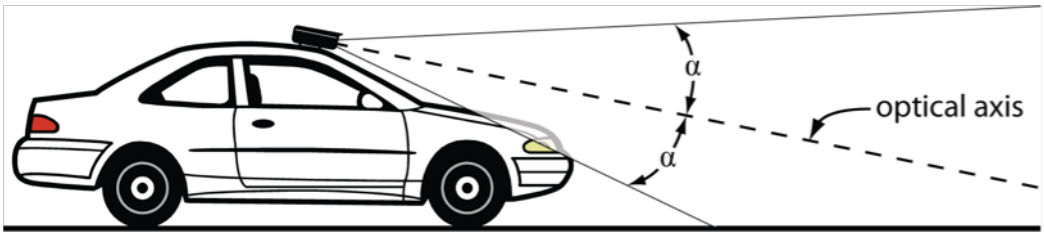


Figure 1.3: Camera on Vehicle

Figure 1.3 shows the camera positioned on top of the vehicle. The optical axis of the camera represents an imaginary line from center of the camera. α is the angular aperture of the camera, also referred to as one half of the cameras viewing angle. Other intrinsic camera parameters are its focal length f, the size of the CDD sensor in the camera, and distortion parameters that occur due to imperfections in the camera.

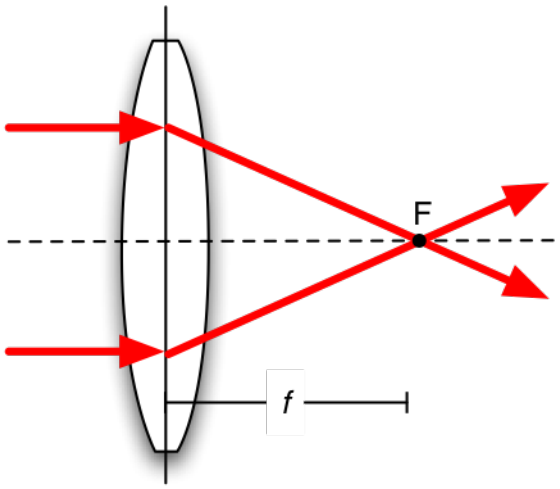
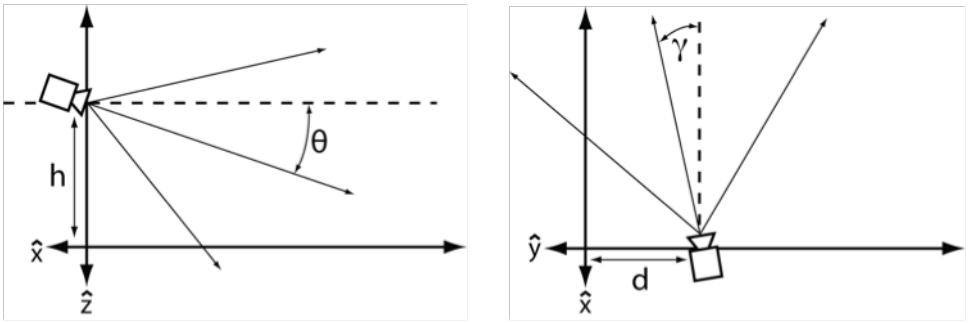


Figure 1.4: Focal Length

The focal length of the camera is the distance from the center of the lens to the

focal point of the lens. The focal length, barrel distortion parameters, and pincushion distortion parameters can all be found using calibration techniques in [33] and [21]

Extrinsic parameters of the camera consist of the camera's position on the vehicle along with its orientation in relation to the vehicle.



(a) Camera Top View (b) Camera Side View

Figure 1.5: Extrinsic Camera Parameters

Figure 1.5a and Figure 1.5b show the top and side views of the cameras location in the world frame. The $\hat{x}$ and $\hat{y}$ axes represent those extending in front of the vehicle and spanning out to the sides of the vehicle. θ represents the downward pitch of the camera, which is the angle between the camera's optical axis and the vector $\hat{x}$. γ represents the yaw of the camera, which is the angle between the optical axis of the camera and the vector $\hat{y}$. These make up the extrinsic parameters of the camera.

Extrinsic and intrinsic camera parameters are important to any computer vision system because they allow the system to recover real-world information from the images obtained by the camera. Without these parameters, the system would not be able to output offsets and distances in real-world measurements.

1.2.2 Edge Detection

A primary method used in many computer vision systems is edge detection. Edges in an image consist of either change in luminosity or change in color. Edge detection is important because it greatly reduces the amount of data for a computer vision system to interpret and preserves only the more important structural properties winthin an image.

One consideration when using edge detection is its susceptibility to image noise. Noisy pixels often create hard edges which introduce false information into the system. This problem is commonly solved using noise reduction filters and smoothing filters before the edge detection is performed. This helps to reduce the image noise, thereby reducing any false edge information.

A simple edge-detection operator is based on a first derivative calculation on the pixel values of the image.

$$I'(x) = -\frac{1 \times I(x-1)}{2} + \frac{0 \times I(x)}{2} + \frac{1 \times I(x+1)}{2} \tag{1.1}$$

Equation 1.1 performs a first derivative on image I using the following kernel:

k = | -1/2 | 0 | +1/2 |

Figure 1.6: First Derivative Edge kernel

A more drastic change in a shorter distance creates a larger edge value. Different edge operators can be used to perform more specific edge detection. One common edge operator is the Sobel operator [31]. This operator is capable of using two different 3x3 kernels, one for horizontal changes and one for vertical changes.

$$G_x = \begin{array}{|c|c|c|}\hline +1 & 0 & -1 \\ \hline +2 & 0 & -2 \\ \hline +1 & 0 & -1 \\ \hline \end{array}$$

(a) Sobel X Kernel

$$G_y = \begin{array}{|c|c|c|}\hline +1 & +2 & +1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

(b) Sobel Y Kernel

Figure 1.7: Sobel Kernels

The results from using the kernels in Figure 1.7a and Figure 1.7b can be used either separately or together to find the edges in either the X direction, Y direction, or both directions together. There are other common edge detectors like the Prewitt edge detector and the Canny edge detector. Each method has its own strengths and weaknesses and must be selected appropriately for the required task.

Edge detection is an important method among computer vision systems. Edges created by sharp intensity changes often reflect structures in the real world, allowing them to be easily interpreted by a computer vision system.

Chapter 2

Related Work

2.1 Road Detection

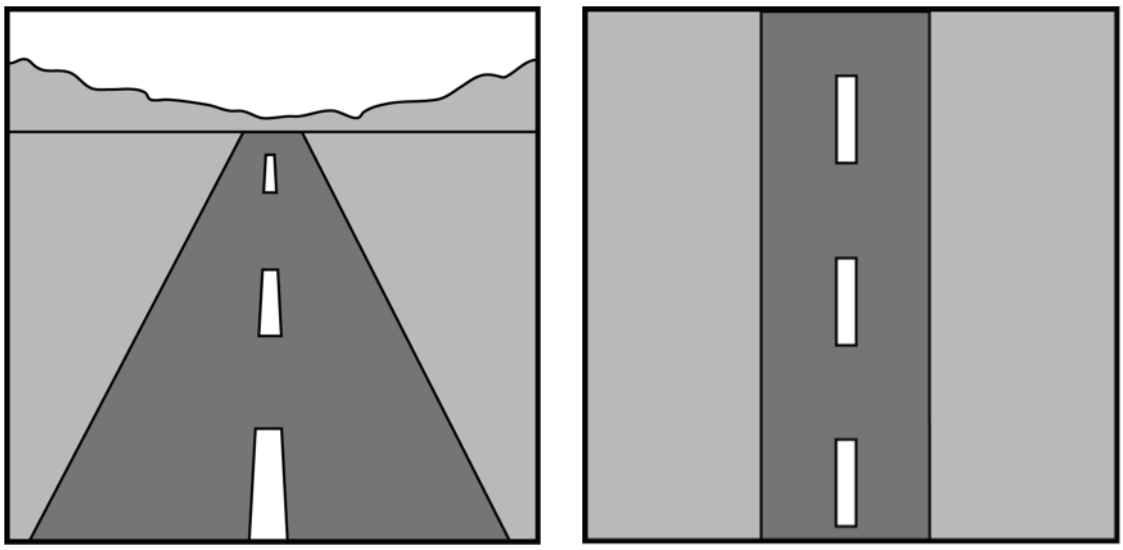
There are many methods to detect a road using a vehicle-mounted camera. Two common approaches are edge detection and color analysis.

2.1.1 Edge Based Methods

Edge detection for detecting roads rely on the roads having defined visual edges, either through painted road lines or physical road edges created by concrete boarders or changing terrain. Many road detection methods utilize this fact to help discern road boundaries from other information in an image. While the types of edge detection used are usually Sobel, Canny, or Prewitt edge detectors, the process of interpreting the resulting edge information is what varies from method to method. Gonzalez [15] devised a method of using histograms and decision trees to find lane candidates and then group them into a decision tree, creating a very fast lane detection system with very low computational cost. The TFALDA algorithm [32] utilizes an evolutionary algorithm to find an ideal tradeoff between starting position, orientation, and gray-level intensities to find the best of many possible lane boundary candidates. The LOIS lane detection algorithm [25] uses a deformable template model of lane structures to locate lane boundaries without thresholding

the edge information. By not thresholding the edges, LOIS tries to be impervious to image clutter like shadows, puddles, and road imperfections, which often make proper thresholding a more difficult task. Some methods determine information about the road by searching for straight lines in the edge result. Schreiber [30] uses the Hough transform to search for straight lines created by edges. The resulting lines are then voted on to determine how likely they are to be road lines. This method is however only able to detect and track straight roads. Chapuis [6] developed a method to recover the three-dimensional shape of a road by using a recursively updating statistical model of the lane obtained by an off-line training phase. This model-driven method is able to very precisely detect lanes and is fairly tolerant of noise.

Many edge-based road detection methods rely on a method known as inverse perspective mapping (IPM) [18] [27] [21]. IPM was originally introduced in [26] and further refined by Bertozzi [2] [3] [5]. IPM is a geometrical transform that attempts to remove the perspective effect present in an image captured by a camera pitched down towards a road. The resulting image without the perspective effect appears as a top-down view of the road.



(a) Road Image with Perspective (b) Road with perspective removed

Figure 2.1: Inverse Perspective Mapping on Synthesized Road

Figure 2.1a shows a synthesized image captured by a camera mounted on a vehicle and pitched downward towards the road. The perspective effect is easily apparent as the road becomes narrower towards the top of the image. Figure 2.1b shows the same synthesized image, but the perspective has been removed using IPM. The remapped image transforms straight roads into vertical lines that can be analyzed much more easily than the untransformed version. IPM relies on intrinsic and extrinsic parameters of the camera and is often adapted to stereovision systems for more complex road determination, like that of non-flat roads. Once the road image has been remapped, the process of extracting the road position and road shape vary between methods.

Road detection through the use of edges is stable on well-defined roads and highways where hard edges are most commonly found. Edge detection for detecting road intersections is difficult for a variety of reasons. Because of the nature and wide variety of road intersections, edge information is not always available or applicable. Road lines break at some road intersections, causing the edges to disappear. Other intersections, like that between a road and a driveway, do not have

line breaks and therefore a gap in road edges is not an applicable characteristic to search for. Searching for horizontal lines is also difficult for detecting intersecting roads. If the intersecting road is perpendicular to the road in front of the vehicle, a horizontal edge may be formed, but roads do not always intersect in this manner. Other horizontal features like house outlines, scene horizons, and other vehicles interfere in the search for horizontal edges. These factors all interfere with the use of edges for detecting road intersections.

2.1.2 Color Analysis

Color extraction and analysis is another common method of detecting roads. Features like color ratios, histogram shapes, and color intensities are often used to distinguish roads from its surroundings. There are also methods to find non-structured roads using color, like those found on dirt paths and non-paved areas. Some methods involve building color models of the road and evolving it as the road color changes [7] [8] [9] [10] [17] [29]. Many of these use a single color model to extract the road pixels from the image. This color model is often represented by a single Gaussian distribution. Some use two models, one for road color and one for non-road color, and classify pixels based on the model they are closest to. These methods often suffer from the fact that roads are comprised of many colors and using a single model to define these many colors either excludes some road pixels or falsely includes non-road pixels. Methods like SCARF [8] and UNSCARF [9] cluster pixels within the roads to create multiple models to represent the road. This clustering method often has more precise and accurate results at a higher computation cost.

These methods can often successfully extract the road surface from an image, but also come with drawbacks. Initialization of color-based road extraction method must be done in either a supervised manner, by initial guesses, or by off-line training with data sets. Its only through these methods that the initial

color models can be decided and then further updated with images obtained from the camera. Using color extraction alone is difficult for road intersection detection. Distinguishing intersections from noise or misclassified pixels can be difficult without knowing exactly where the boundaries of the road can be found. It also may be difficult to differentiate a large intersection from a curved road.

2.1.3 Combination Methods

To overcome the problems encountered using either edge-based lane detection or color-based road extraction exclusively, Yinghua He [16] developed a method combining both the stability of edges and the adaptability of color-based extraction. This method utilizes inverse perspective mapping and edge detection to determine both the lane boundaries and the road geometry. Once the perspective is removed and the edges are extracted, various pre-defined road curvatures are referenced to find the most applicable curvature of the road in the image frame. The locations of the left and right lane boundaries are found using probability histograms. The resulting road model and road curvature is then used to extract a multivariate Gaussian distribution to represent the color of the road within the determined boundaries. This assumes the road area satisfies the following equation at given frame n.

$$P(x) = \frac{1}{(2\pi)^{(\frac{d}{2})}|\Sigma_n|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_n)^T \Sigma_n^{-1}(x-\mu_n)} \tag{2.1}$$

Here, x is any pixel in the image frame n. μ_n and Σ_n represent the mean value and variance of the Gaussian distribution. If a point in the image belongs to the road area, its color will be similar to the mean of the extracted road color model. The pixels in the image are all analyzed and their probabilities are compared to a pre-defined threshold. Pixels that meet the threshold are determined to be road and pixels that do not meet the threshold are determined to be the area surrounding the road. From this color-based extraction, candidate boundaries from the

edge-based lane extraction are compared using Bayes rule to find the best fit between the two. The result is a fast method that can perform color extraction without supervised training and runs faster than cluster-based methods like SCARF or UNSCARF. This method is the basis for the intersection detection method described in this paper.

2.2 Intersection Detection

The detection of intersections is a topic much less explored when compared with road and lane detection. Duchow [12] attempted to use road markings for detecting the presence of intersections.

Figure 2.2: Marking Based Intersection Candidate

This method assumes that intersections are heavily marked in the direction of traffic. The SCARF system [8] uses a brute force analysis on a color extraction of an image. Using a supervised color classification, the road area is extracted from the

image. SCARF then uses binary masks of road configurations with various curvatures to find one best fits the road in the image. These binary masks search for a main branch of the road, and then a secondary branch that could possibly represent an intersection. The road model that contains the highest probability of road pixels and smallest probability of non-road pixels is determined to be the most accurate model of the road and its intersecting branches. To limit the number of different road configurations that must be searched, SCARF evaluates candidates in a certain sequence so that some configurations can be eliminated. While this is faster than searching all possible configurations, SCARF still compares the entire image to the binary mask of the road model to find a proper probability. This seems unnecessary because after the main branch of the road is already found, calculations no longer need to be done on those pixels while trying to find intersecting roads.

Rasmussen [29] developed a method for detecting roads and intersections by using a three-camera configuration. Each camera is oriented at a different yaw angle in relation to the vehicle heading. One camera is yawed slightly to the left, one camera is centered, and one camera is yawed slightly to the right. The resulting images from these cameras are combined to build a mosaic image with a very large field of view. A snake-based road edge-tracking algorithm [11] is used to find the main road in front of the vehicle. A support vector machine (SVM) is trained to classify road pixels and the resulting road pixels are fit with polygons to model the shape of the road. The downside of this method is the fact that intersection detector is configured to a certain distance ahead of the vehicle, limiting what it can detect. It's also a difficult method to implement because if its intricate three-camera configuration.

Foedisch and Takeuchi [13] [14] attempted to use a neural network for road detection. Jochem [22] [23] used a similar neural network, called ALVINN VC, to detect intersections. Using a virtual camera which is simply a portion of the

image produced by a primary camera projected onto some real world model, the ALVINN VC system attempts to find intersecting roads in these virtual cameras.

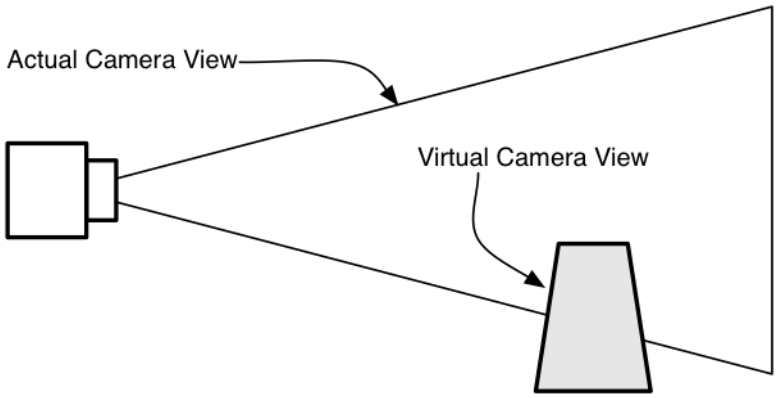


Figure 2.3: Virtual Camera Layout

Using the virtual camera in 2.3 allows the ALVINN VC system to use the same road-trained neural network to search for the main road and for intersecting roads. The virtual cameras are set up so that on an ideal road, the intersecting road will be within the boundaries of that virtual camera. If the neural network attached to that virtual camera finds a road contained in the virtual camera image, the ALVINN VC method determines an intersection is approaching. The downside of this method is that the intersections are only searched for at predefined locations. If the intersections occur outside of these locations, they will go undetected. Also, if the main road happens to intrude into one of these locations, they will be falsely detected as intersections.

These methods are among the few published methods for detecting road intersections.

Chapter 3

Proposed Method

3.1 Introduction

The following computer vision algorithm was developed for the purpose of detecting roads and intersections to assist with the localization of an autonomous urban robot. The algorithm was developed with the following goals.

- It should be able to detect the road lane in front of the vehicle on both straight and curved roads so the robot can be properly localized between road lanes.
- It should provide offsets from the center of the camera to the lane boundaries on the left and right of the vehicle in real-world measurements of meters or centimeters.
- It should not require a training phase and can adapt to roads of varying color or luminosity.
- It should be unaffected by minor shadows or clutter commonly found on urban roads.
- It should be able to detect intersections and provide a forward offset in real-world measurements of meters or centimeters.

- The must must run fast enough for use in an autonomous vehicle traveling at urban speeds under 40 mph. Ideally this speed should be greater than 1 Hz.

- It should work with a single RGB color camera.

Since the system cannot be trained to any specific set of training images, a predefined neural network or support vector machine are not acceptable. Using edge information alone for intersection detection seems unreasonable since intersecting roads have no obvious edge patterns that can be distinguished from the surrounding environment. Color analysis is a much more reasonable method for detecting intersecting roads. And since the system must work reliably on urban roads with acceptable speeds, combining edge-based lane detection with color-based intersection detection is ideal.

The method described here is a combination of three previously developed methods with an adaptation for detecting intersecting roads.

Figure 3.1 describes how the different methods interact. The section outlined in green is the inverse perspective mapping technique outlined in [26] [4]. The section outlined in blue utilizes the edge-based lane detection and road model fitting described by He in [16]. The section outlined in red consists of a method for learning and scoring the color models of the road outlined by Dahlkamp in [10]. The section in orange is an addition that attempts to interpret the results of the preceding methods and extract any intersecting roads to determine their position.

3.2 Image Capture

3.2.1 Automatic Gain

The first step of any computer vision system is obtaining the image from the camera. To accommodate for varying degrees of brightness that occur when capturing

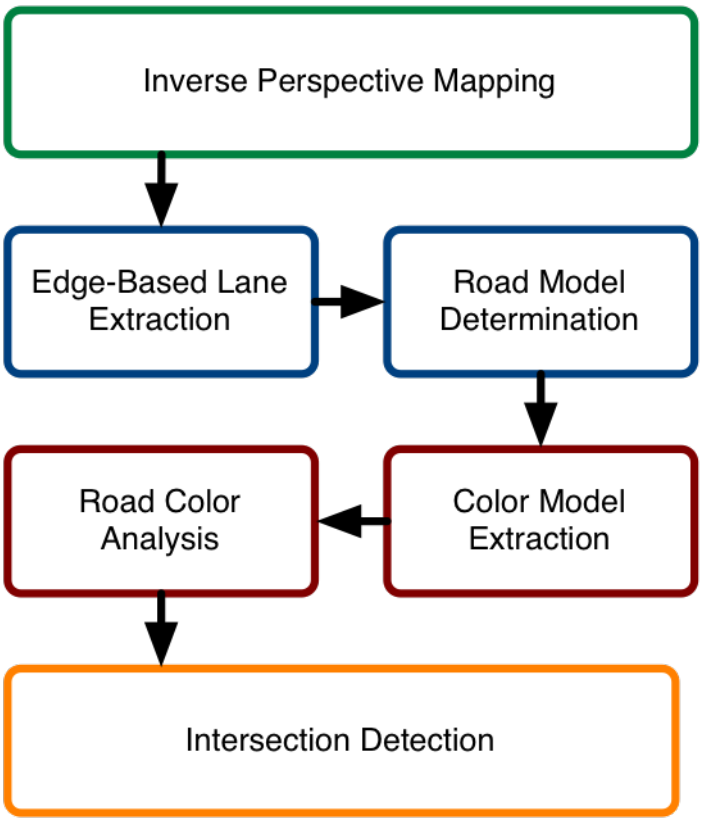


Figure 3.1: Method Flowchart

images in an outdoor environment, a simple automatic gain algorithm is implemented to correct the brightness of the captured image. When the camera captures an image I, it is converted from a color RGB image to an intensity image using the following formula

$$I(u,v) = \frac{1}{3}\left[R(u,v) + G(u,v) + B(u,v)\right] \tag{3.1}$$

Every pixel in the resulting intensity image is traversed and averaged to find a single average intensity for the entire image. This average intensity can either be compared to a pre-defined intensity value or the initialized average intensity of the first few frames on startup, assuming they are taken at an ideal brightness level. Once an ideal average intensity is determined, the gain of the camera can be corrected for every image frame that is captured.

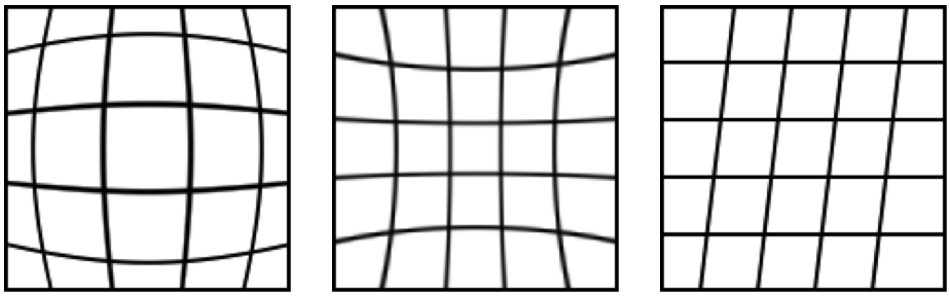
Algorithm 3.1 Automatic Gain Adjustment

G = current camera gain
I = ideal average image intensity
A = average intensity of current image frame
if $A > I$ **then**
  $G = G + 1$
else
  if $A <$I **then**
    $G = G - 1$
  end if
end if

Algorithm 3.1 creates images frames that automatically adjust toward an ideal intensity level. While the adjustment is not instantaneous and takes multiple frames to completely adjust, the result is far better than a fixed gain camera whose images usually end up being either too dark or too bright.

3.2.2 Lens Distortion Removal

Most camera lenses suffer from imperfections that unfortunately manifest as slightly distorted images. To correct this problem and prevent it from affecting results, each image frame captured by the camera must be corrected to remove these distortions. The most common distortions are barrel distortions, pincushion distortions, and skew distortions.



(a) Barrel Distortion (b) Pincushion Distortion (c) Skew Distortion

Figure 3.2: Common Lens Distortions

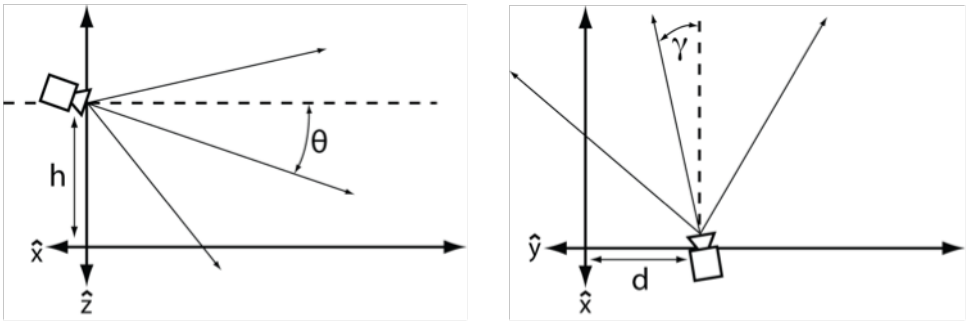
The amount of each of these types of distortion can be found offline using various camera calibration techniques outlined in [20] and [33]. The camera calibration process will determine the value of certain intrinsic parameters of the camera such as the focal length $f_c = [f_x, f_y]$, the principal foci $c_c = [c_x, c_y]$, skew angle a_c, and distortion coefficients $k_c = [k_x, k_y]$. Using these camera calibration parameters it is possible to correct each camera frame for a more accurate representation of the world frame.

(a) Image Before Lens Distortion Removal (b) Image After Lens Distortion Removal

Figure 3.3: Lens Distortion Removal

3.3 Inverse Perspective Mapping

Inverse perspective mapping is the process of removing the perspective present in an image captured by a camera mounted to a vehicle and pitched down towards the road. The process removes this perspective to create a birds-eye view of the road area captured in the image. This method assumes that intrinsic and extrinsic camera parameters are known and also make the assumption of a planar road surface in front of the vehicle. Since roads within the visual range of a camera usually conform to a planar or near-planar model, this assumption causes errors in only a small number of situations.



(a) Bertozzi Camera Top View (b) Bertozzi Camera Side View

Figure 3.4: Extrinsic Camera Parameters

The IPM method outlined by Bertozzi [4] transforms between three-dimensional Euclidean space $W = (x, y, z) \in E^3$ and two-dimensional Euclidean space $I = (u, v) \in E^2$ where the three-dimensional scene is projected. Here, $\hat{x}$,$\hat{y}$,and $\hat{z}$ coordinates in the world frame W are expressed in meters. u and v are coordinates in the image frame I are expressed in pixel location. The image acquired by the camera is represented in I space while the remapped image belongs to the $z = 0$ plane of W space. To remap the image, the following parameters from Figure 3.4b and Figure 3.4a must be known:

- (d, h):the cameras position in 3D space
- γ: camera yaw
- θ: camera pitch
- 2α: angular aperture
- $n \times n$: camera resolution

The transform $W \rightarrow I$ is outlined below.

$$u(x, y, 0) = \frac{atan\left\{h \sin\left[\arctan\left(\frac{y-d}{x-l}\right)\right]\right\} - (\theta - \alpha)}{\frac{2\alpha}{n-1}} \tag{3.2}$$

$$v(x, y, 0) = \frac{\arctan\left[\frac{y-d}{x-l}\right] - (\gamma - \alpha)}{\frac{2\alpha}{n-1}} \tag{3.3}$$

The inverse transform $I \rightarrow W$ is used to map pixels from the image frame to the world frame.

$$x(u, v) = h \cot\left[(\theta - \alpha) + u\frac{2\alpha}{n-1}\right] \sin\left[(\gamma - \alpha) + v\frac{2\alpha}{n-1}\right] + l \tag{3.4}$$

$$y(u, v) = h \cot\left[(\theta - \alpha) + u\frac{2\alpha}{n-1}\right] \cos\left[(\gamma - \alpha) + v\frac{2\alpha}{n-1}\right] + d \tag{3.5}$$

The IPM equations put forth by Bertozzi produced somewhat unexpected results. While the resulting transformed points did appear to have the perspective effect of the camera removed, they also displayed some form of distortion. Straight, horizontal lines became curved lines, which is an effect that should not be caused by a perspective transform.

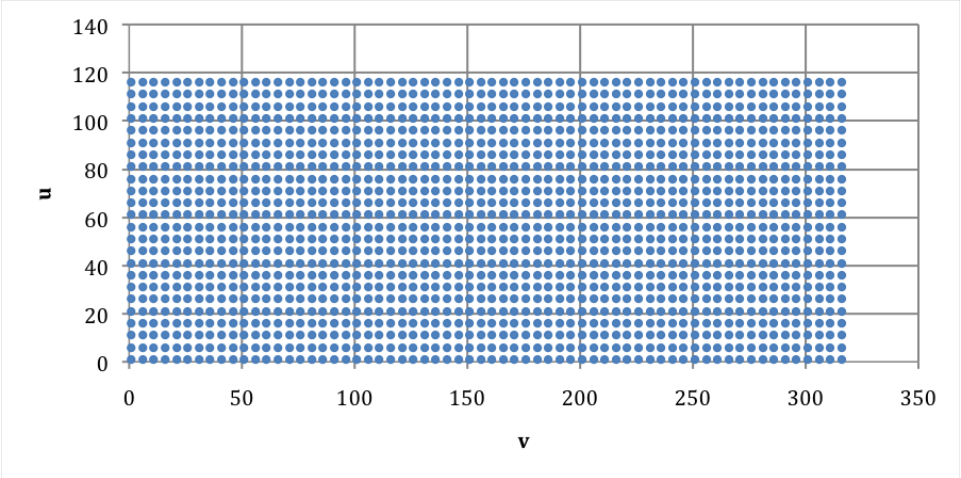


Figure 3.5: Bertozzi Image Frame Points

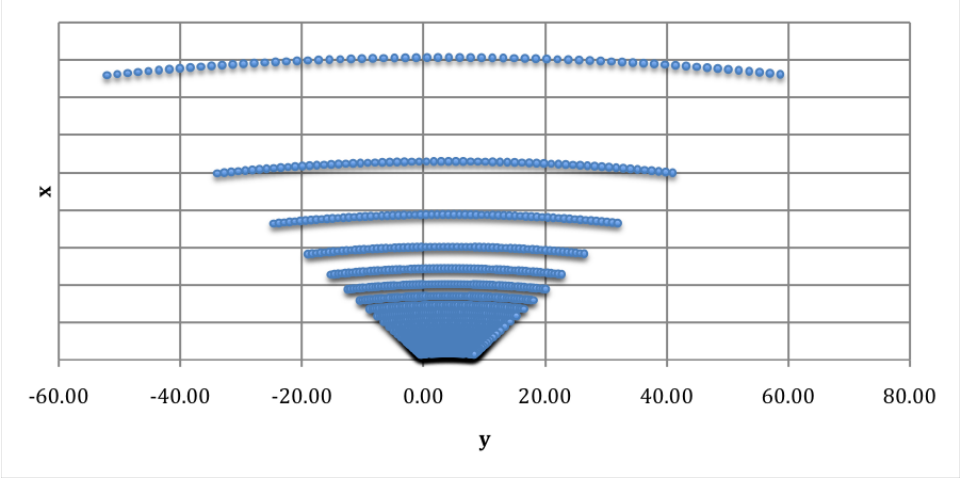
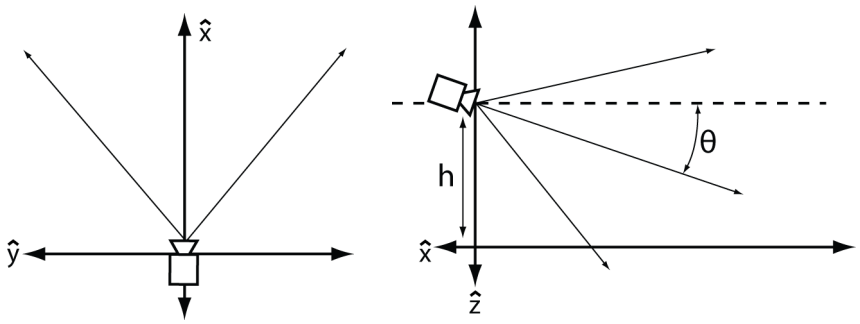


Figure 3.6: Bertozzi Remapped World Frame Points

Figure 3.6 shows the remapped points from Figure 3.5. The horizontal rows of points become curved lines in Figure 3.6. While the curvature introduced may be minimal, it still shows that there may be an error introduced in the derivation.

To prevent the distortion present in Bertozzi's version of IPM, the perspective mapping formulas were re-derived. Figure 3.7a and Figure 3.7b show the layout of the world frame.

Figure 3.7a and Figure 3.7b show the top and side views of the world frame.



(a) World Frame Top View (b) World Frame Side View

Figure 3.7: World Frame Diagram

The camera is mounted on top of a vehicle and pitched downwards towards the road. The downward pitch of the camera is represented by the angle θ. The height of the camera is the distance from the ground plane to the center of the camera's sensor and is represented by the distance h. This height is measured in meters. The world frame is in three-dimentional space where the $\hat{x}$ axis extends from the ground plane below the camera's sensor outwards in front of the vehicle. The $\hat{y}$ axis extends from the ground plane below the camera's sensor to either side of the vehicle. The $\hat{y}$ axis originates at the camera, therefore the $\hat{y}$ is positive to the right of the camera and negative to the left of the camera when facing down the road. The $\hat{z}$ axis extends from the ground plane upwards through the camera's sensor. Since a planar road is assumed, everything captured in the image frame is from the $z = 0$ plane of the world frame.

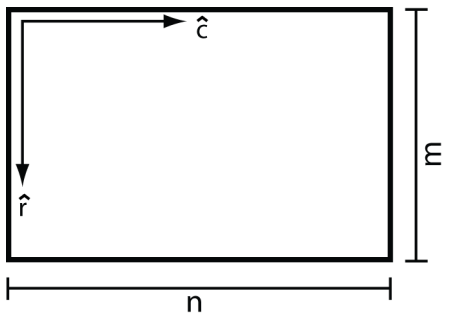


Figure 3.8: Image Frame Diagram

Figure 3.8 shows the image frame captured by the camera. The image is indexed by column and by row originating from the upper left corner of the image. The width of the image is n pixels and the height of the image is m pixels.

When the image frame $I(c, r)$ is captured by the camera, it is remapped to the $z = 0$ plane of the world frame $W(x, y, 0)$ using the transform $I \leftarrow W$.

$$x(r) = h \left(\frac{1 + \left[1 - 2\frac{r-1}{m-1}\right] \tan(\alpha_x) \tan(\theta)}{\tan(\theta) - \left[1 - 2\frac{r-1}{m-1}\right] \tan(\alpha_y)} \right) \tag{3.6}$$

$$y(r, c) = h \left(\frac{\left[1 - 2\frac{c-1}{n-1}\right] \tan(\alpha_x)}{\sin(\theta) - \left[1 - 2\frac{r-1}{m-1}\right] \tan(\alpha_y) \cos(\theta)} \right) \tag{3.7}$$

The reverse transform $W \leftarrow I$ is shown below.

$$r(x) = \frac{m-1}{2} \left(1 + \frac{h - x \tan(\theta)}{h \tan(\theta) + x} \cot(\alpha_x) \right) \tag{3.8}$$

$$c(x, y) = \frac{n-1}{2} \left(1 - \frac{y}{h \sin(\theta) + x \cos(\theta)} \cot(\alpha_y) \right) \tag{3.9}$$

Each pixel in the image frame $I(c, r)$ is transformed to a $(x, y, 0) \in W$ value. Since the world frame is in meters originating from the camera outwards in front and to the right of the vehicle, these values need to transformed and scaled so they can be represented as a transformed image. The image is taken from the world frame W to the remapped image frame $I_{remapped}$ using the following transform where $SCALE$ is a value to scale up or scale down the size of the remapped image.

$$r_{remapped}(x) = m - (x \times SCALE) \tag{3.10}$$

$$c_{remapped}(y) = \frac{n}{2} + (y \times SCALE) \tag{3.11}$$

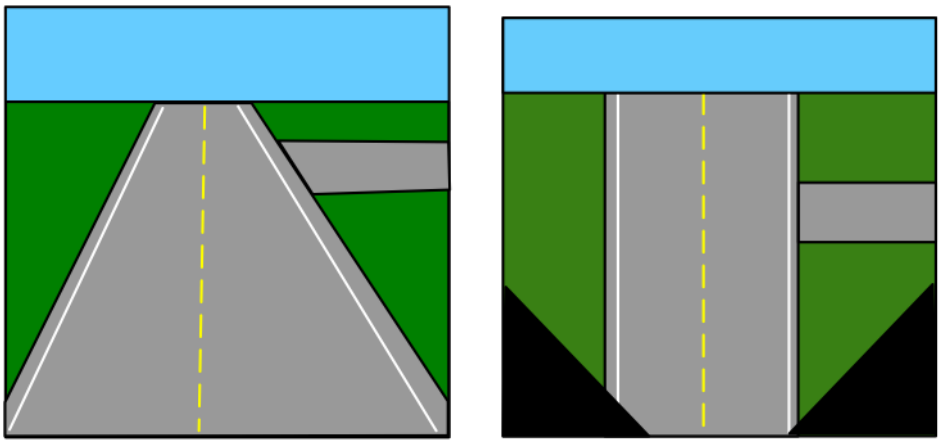
To increase the speed of IPM, the remapping is only calculated once and stored in a lookup table indexed by the row and column of the original image. Due to the

large number of floating point calculations required by the IPM method, preprocessing this table saves significant time.

$$x_{remapped}[c, r] = ipmTable[c, r].x \tag{3.12}$$

$$y_{remapped}[c, r] = ipmTable[c, r].y \tag{3.13}$$

The resulting remapped image contains portions that can be discarded. The area above the horizon is assumed to be non-road and therefore insignificant to this process. The location at which the horizon lies is represented by the row of the original image I where the road intersects the sky.



(a) Synthetic Road Image (b) Remapped Synthetic Road Image

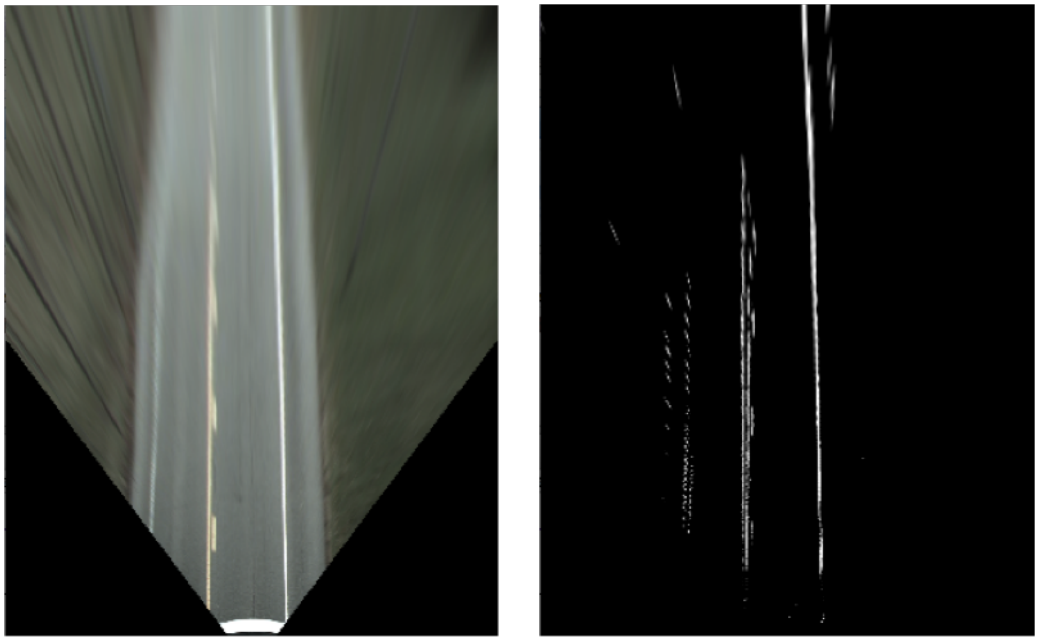
Figure 3.9: Inverse Perspective Mapping Example

The $horizon$ value depends on the angular aperture α of the camera lens as well as the camera mounting height h and downward pitch θ. Since assuming a planar road model means the horizon line should not change, the $horizon$ value can be pre-defined based on the camera configuration. Figure 3.9b also shows how empty pixels are created from the bottom of the image slanting outward. Because of this, the remapped image requires cropping to isolate only the useful pixels and prevent unnecessary calculations.

3.4 Lane Detection

3.4.1 Lane Boundary Extraction

To extract the painted road lines and other road boundaries, edge detection is performed on the remapped image $I_{remapped}$. Sobel edge detection was chosen over Canny edge detection because the thinning and suppression performed by the Canny algorithm make the proceeding steps of this method less effective. Sobel edge detection was also chosen for the optimized implementation of the Sobel detector in the libraries used for implementation. Once edge detection is performed, the edge image is thresholded using a predefined edge threshold $edge_{thresh}$. All edge intensities above $edge_{thresh}$ are set to a value of 1 and all edge intensities below $edge_{thresh}$ are set to 0. The resulting image is a binary map I_{thesh}, that represents the significant image edges in the world frame.



(a) Remapped Color Image (b) Thresholded Edge image

Figure 3.10: Remapped Color Image

Once the image is remapped and the edge gradient is thresholded, the binary

image I_{thresh} can be searched for road lane boundaries. As seen in Figure 3.10b, a straight road transforms into a series of vertical lines at the locations of the road lane edges. This relies on the assumption that the road in the image is straight, the lane center is collinear with the vertical image center, and the road in front of the vehicle is planar.

To extract the lane boundaries from I_{thresh}, the image is analyzed in a column-wise manner. The number of edge pixels present in each column determines the probability that a lane boundary is present. Traversing the rows of each column allows the probability of a lane boundary at each column to be determined,

$$p_{lane}(c) = \sum_{r=1}^{m} I_{thresh}\,(c, r) \tag{3.14}$$

Determining the lane probability of each column creates the histogram $hist_{lane}(c)$ of lane probabilities, indexed by column.

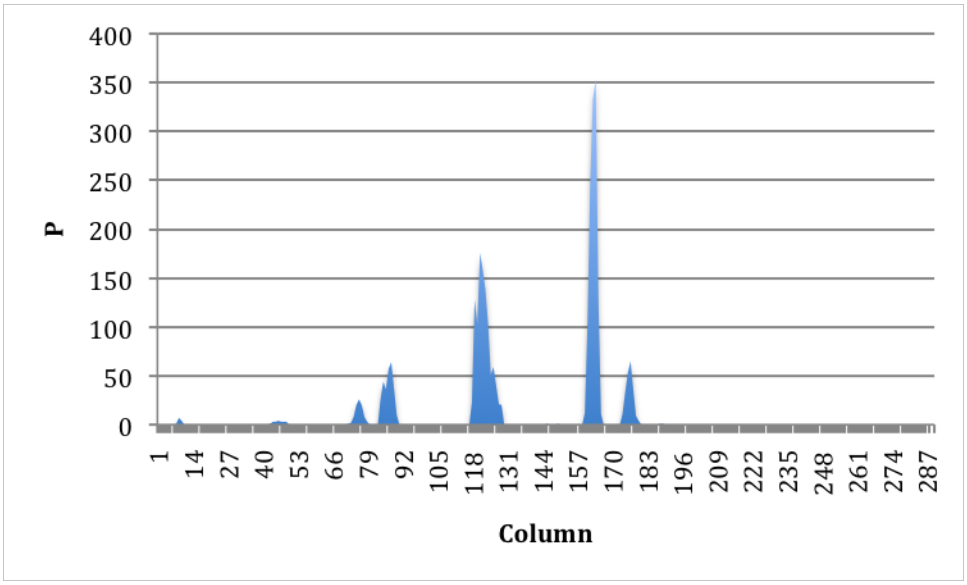


Figure 3.11: Lane Boundary Histogram Example

Figure 3.11 shows the histogram resulting from the edge image in Figure 3.10b. The two large peaks in the histogram can be correlated with the two well-defined lane boundaries in the image. The smaller peaks may represent noise, other lane

lines, or possibly the outer road edge. Using a global maximum would prove unsuccessful due to the fact that the rightmost lane line in Figure 3.11 contains two columns with the highest overall probabilities. To successfully find the columns with the highest probabilities that are not correlated to the same lane boundary, local maxima are found across the histogram using the following algorithm

Algorithm 3.2 Histogram Local Maxima

$inGroup = false$
for c=0 to n-1 **do**
  if $h_{lane}(c) > 0$ **then**
    if $inGroup == false$ **then**
      $localMaxValue = h_{lane}(c)$
      $localMaxIndex = c$
      $inGroup = true$
    else
      if $h_{lane}(c) > localMaxValue$ **then**
        $localMaxVale = h_{lane}(c)$
        $localMaxIndex = c$
      end if
    end if
  else
    if $h_{lane}(c) == 0$ or $c == n - 1$ **then**
      if $inGroup == true$ **then**
        $localMaxList.add(localMaxIndex, localMaxValue)$
        $inGroup == false$
      end if
    end if
  end if
end for

The result of Algorithm 2 is a variable number of lane boundary candidates depending upon the number of peaks in h_{lane}. This result can be limited using a probability threshold to limit the lane boundary candidates to only those above an acceptable probability value.

Once the local maxima are found, they must be analyzed to find the two lines most likely to be the real lane boundaries of the road. Since it is entirely possible that the two most prevalent lane boundaries in the image could be the center lane

and the leftmost lane, or the rightmost lane and the leftmost lane, or even a single lane boundary and en edge created by road clutter, the lane boundaries must be considered by their relation to various road characteristics. Since it is better suited to analyze pairs of lane boundaries as real world lanes rather than single lane boundaries by themselves, each lane boundary candidate is paired with each other lane boundary candidate to create a list of all possible lane boundary pairs. If there are n lane boundary candidates and two boundaries for each lane, there are $\frac{n!}{2(n-2)!}$ possible lane candidates. Each lane candidate consists of the two lane boundaries, where the boundary with the smaller column index is set to the left lane boundary and the larger column index is set to the right lane boundary.

To rank the lane candidates according the probability that they represent the lane being driven on, the following weights are considered:

- the width of the lane created by the lane boundaries
- the distance between the center of the lane and the optical axis of the camera
- the distance of the lane found to the lane found in the previous image frame

The width of the lane is judged based on a pre-defined ideal lane width. This could be either the width of the lane found in the previous image frame or based on measurements of the lanes intended to be driven on, if available. These weights are modeled using Gaussian distributions. The probability of the lane boundary is found using the following formulas.

$$p_{lane} = p_{boundary,left} + p_{boundary,right} + p_{lane,width} + p_{lane,center} + p_{lane,previous} \tag{3.15}$$

$$p_{width} = \left[\frac{a_{width}}{\sigma_{width}\sqrt{2\pi}} e^{\frac{(x-\mu_{width})^2}{2\sigma^2_{width}}} \right] \tag{3.16}$$

$$p_{center} = \left[\frac{a_{center}}{\sigma_{center}\sqrt{2\pi}} e^{\frac{(x-\mu_{center})^2}{2\sigma^2_{center}}} \right] \tag{3.17}$$

$$p_{previous} = \left[\frac{a_{previous}}{\sigma_{previous}\sqrt{2\pi}} e^{\frac{(x-\mu_{previous})^2}{2\sigma^2_{previous}}} \right] \tag{3.18}$$

Here, μ_k is the mean, σ_k^2 is the variance, and a_k is the weight for each determining factor. $width$ is defined as the column of the left boundary subtracted from the column of the right boundary and $center$ is defined as the left boundary column plus half the image width. The list of lane candidates is sorted by its calculated lane probability and the lane candidate with the highest probability is the best candidate for the image frame. This method of weighting lane candidates helps to discard invalid lane possibilities and leads to a more stable lane finding system.

3.4.2 Road Model Determination

While the method described above can quickly and reliably detect the lane boundaries of straight roads, it is unsuccessful on roads that are not perfectly straight. He described a method [16] to utilize pre-defined road models to detect various road curvatures. The method by He allows the same lane finding algorithm used to find straight roads to find roads of various different curvatures. To do this, a set of road model curvatures must first be defined.

Assuming that the road lane boundaries are parallel lines allows the curvature of the road to be defined as a set of offsets based on the image row. Road curvatures for this method were described using the following formula

$$\text{offset}(r) = \left(\frac{m - r}{\left(\frac{n}{4.0 \times s}\right)} \right)^2 \tag{3.19}$$

Here, m is the image height, n is the image width, r is a specific row of the image, and s is a predefined curvature scalar.

Figure 3.12 shows how different curvature scalars create different curvature values. These curvatures can range from slight curves in the road to sharp turns in either direction.

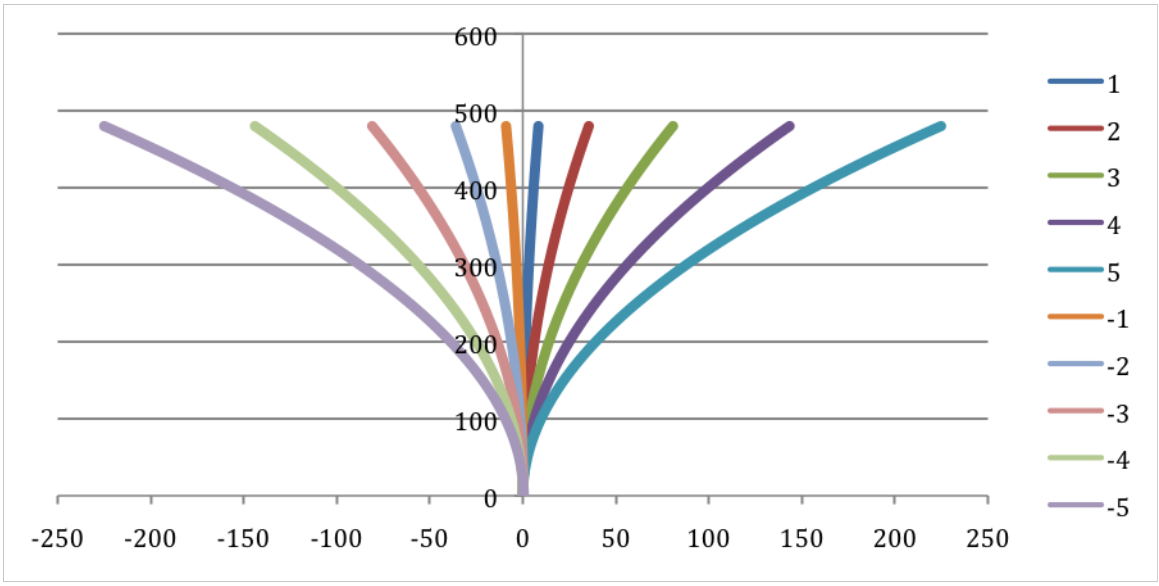
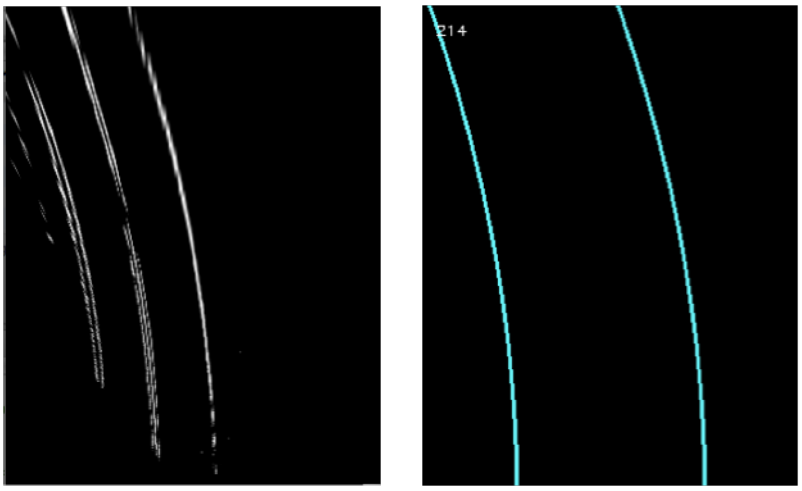


Figure 3.12: Possible Road Curvature Models

To utilize these road curvature models with the lane boundary detection method outlined above, the pixels from the remapped, thresholded binary edge image is shifted before the column-based histogram is determined.



(a) Thresholded Edge Image of Curved Road

(b) Road Curvature Model

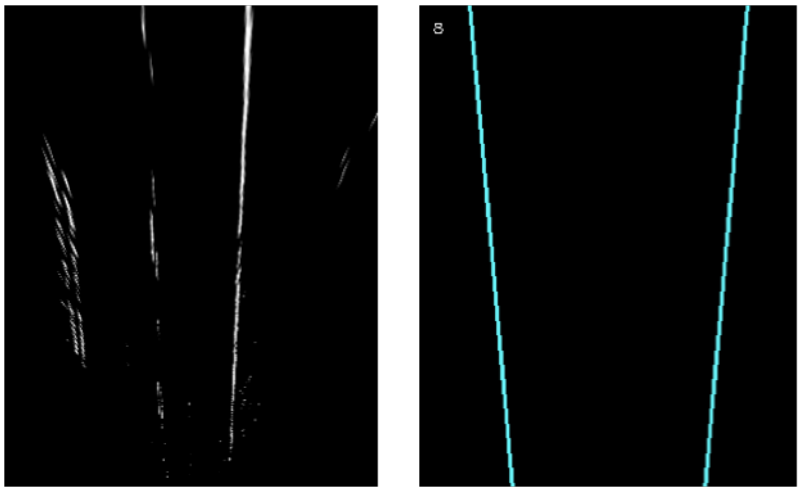
Figure 3.13: Road Model on Curved Road

Figure 3.13a shows a road image after inverse perspective mapping, edge detection, and thresholding. The resulting image is a binary map of the image edges. The curve in the road prevents the vertical line based-histogram analysis being effective in this situation. Figure 3.13b shows a graphical representation of a road curvature model created by Equation 3.19 and a pre-defined curvature scalar. Figure 3.4.2 shows the image from Figure 3.13a with each row shifted by the offset generated by road curvature model in Figure 3.13b. The edges in this image are now in straight vertical lines and can be analyzed using the vertical line-based histogram analysis. The location of the left and right lane boundaries can be found and combined with the curvature scalar value of the road curvature model to provide an accurate description of the detected road model.

The method described by He [16] for utilizing these pre-defined road models uses them only to extract roads with a single curve. The method, however, can be expanded to detect other road shapes and distortions. One distortion that can be corrected is the perspective change that occurs when a vehicle encounters a bump in the road and changes pitch. The change in vehicle pitch also changes the pitch

Figure 3.14: Curved Road Shifted with Road Curvature Model

of the camera, effectively changing the parameters required to correctly perform inverse perspective mapping. The result is a remapped image that appears to still have a perspective effect, thereby preventing vertical line analysis from being effective



(a) Remapped Edge Image with Perspective Distortion

(b) Road Perspective Transform Model

Figure 3.15: Road Model on Perspective Road

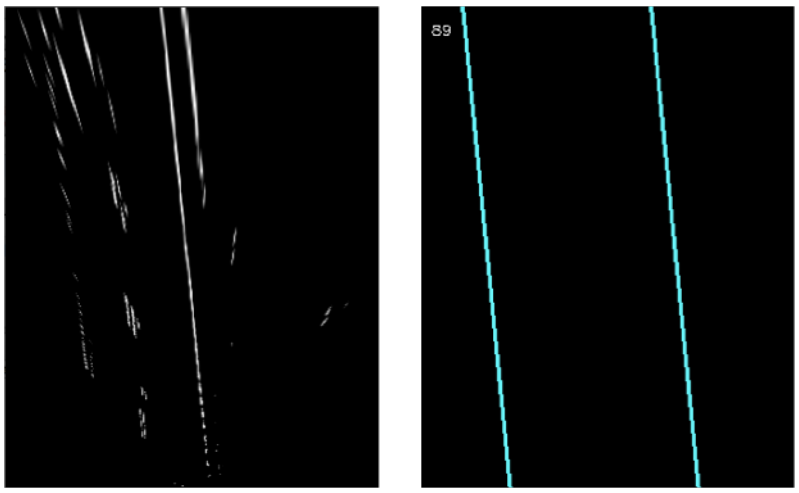
Figure 3.15a shows an image that has been distorted by an unplanned change in

vehicle pitch. The perspective distortion created by this unplanned pitch is enough to prevent the vertical line-based histogram analysis from working accurately because the lane boundaries slant slightly outward from the optical axis of the camera. This distortion can be removed using the same model-driven analysis used to find road curvature.

$$\text{offset}(r) = p \times r \tag{3.20}$$

Figure 3.15b shows a graphical representation of the perspective distortion model affecting the image in Figure 3.15a and the road perspective model in Equation 3.20, where p is a predefined perspective scalar and r is any row index in the image. Like the road curvature models, this road perspective model is simply a set of offsets indexed by image row. However, the road perspective model differs such that the offset is different on either side of the cameras optical axis. So to remove the perspective distortion, the image has to be shifted by a positive offset on the left side of the optical axis and shifted by a negative offset on the right side of the optical axis, which runs vertically through the center of the image. The pixels must also be checked before shifting to prevent pixels from shifting across the optical axis, which could prevent duplicate pixels and unnecessary distortion.

Another possible distortion with inverse perspective mapping is a lateral offset from the center of the lane caused by the vehicle drifting too far away from the center of the road lane. Since the inverse perspective mapping technique assumes the subject of the image is found at the cameras optical axis, moving the subject away from this axis can cause the remapped image to appear skewed.



(a) Remapped Edge Image with Lateral Distortion

(b) Road Skew Transform Model

Figure 3.16: Road Model on Skewed Road

Figure 3.16a shows how this distortion affects the remapped image, again making vertical line-based histogram analysis less effective. Figure 3.16b shows a graphical representation of this skew distortion model, which is also based on Equation 3.20 except the offset is the same on either side of the cameras optical axis. This transform skews the remapped image, restoring the straight vertical lines.

The road transformation models mentioned above are expressed in the form $t_k = (type, value)$ where t_k is an individual transformation model, $type$ is the type of distortion it represents, being either curve, perspective, or skew, and $value$ is scalar value for the model, either positive or negative. Since these models may use computationally expensive formulas, such as the raised power of the curvature model, to determine the offset value, these values are all pre-calculated and stored as an offset vector for that transform. For a given remapped image frame, the transformed version can be quickly found by taking each pixel and simply adding the integer offset from the corresponding transforms offset vector. This creates an effective way of constructing the histogram for any given road transformation

model, which can then be analyzed to find possible lane boundaries.

3.4.3 Road Model Interpolation

The problem with model-based road detection methods is the tradeoff that occurs between speed and accuracy. To gain better accuracy, more pre-defined road models must be used. To run the system at a higher speed, fewer models can be used, but a larger margin of error is introduced into the system. Generally, the tradeoff in a system like this comes down to a design decision, where speed or accuracy is chosen based on the systems intended application, or extensive testing allows for the best speed to be achieved within a decidedly acceptable margin of error.

To avoid this inherent problem with model-driven road detection, a different approach was taken that attempts to achieve a high level of accuracy while maintaining an acceptably fast speed. An initial set of road curvature models is defined with an increasing curvature value $C_{intial} = \{-5, -4, -3...3, 4, 5\}$. These initial curvatures define possible curvatures in the road, but not enough to be completely accurate. A secondary set of road curvature models $C_{interpolated}$ is defined with a more extensive set of curvatures, excluding those defined in $C_{initial}$.

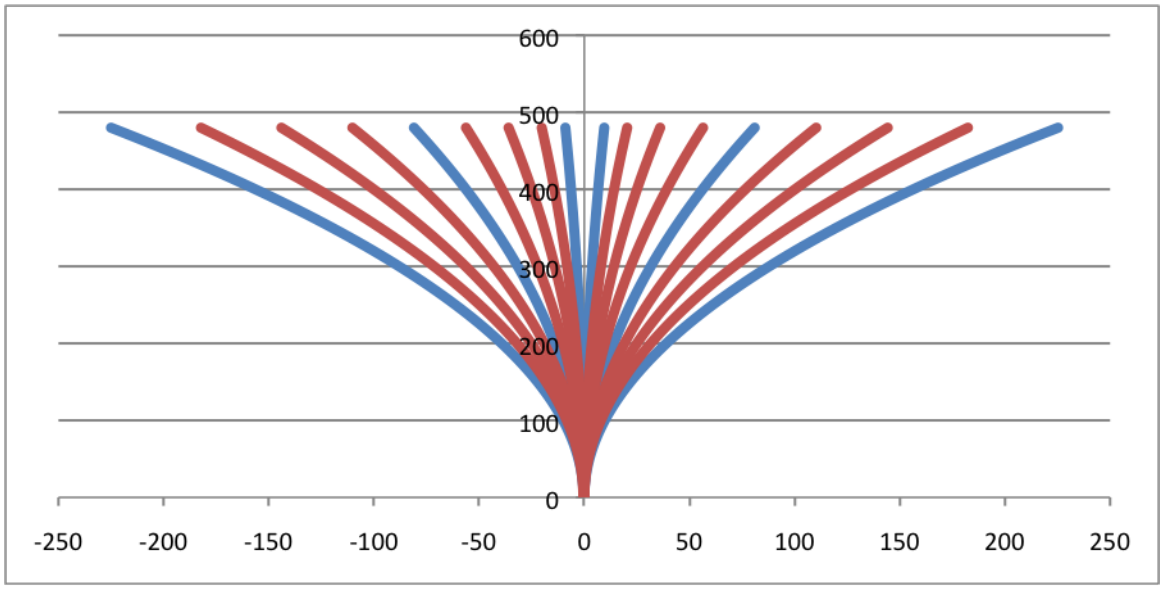


Figure 3.17: Graph of Initial and Interpolated Curvature Models

In Figure 3.17, blue lines are curvatures from $C_{initial}$ and red lines are curvatures from $C_{interpolated}$. For each pair of curvatures $C_{initial}(k)$, $C_{initial}(k+1)$, there are i curvature models between them in $C_{interpolated}$. To associate a given set of curvatures from $C_{interpolated}$ with the two corresponding curvature values from $C_{initial}$, a lookup table is defined that maps any pair of curvatures from $C_{initial}$ to a set of curvatures in $C_{interpolated}$.

$$map(c_a, c_b) = \{c_0, c_1, ...c+n\} \in C_{interpolated} \tag{3.21}$$

For example, $map(1,5)$ returns $\{2,3,4\}$. The idea behind this curvature map is that if two curvatures are a close match to the road curvature present in $I_{remapped}$, there is a high probability that there is an even better match between those curvatures. Therefore, to find the road curvature model that correctly fits the road lane boundaries in remapped image $I_{remapped}$, the following process is followed.

1. For the n curvatures in $C_{initial}$, the remapped input image $I_{threshold}$ is shifted to create a series of transformed images $I_{transformed}[i], i \in \{0, 1..n\}$.

2. For each transformed image $I_{transformed}[i]$ where $i \in \{0, 1, ...n\}$, compute the column-wise histogram $hist[i]$ of edge values.

3. For each histogram $hist[i]$, calculate the probability p that the histogram contains a road edge boundary candidate.

4. Sort the array $hist$ by probability p to create $hist_{ordered}$.

5. Set t_1 and t_2 to the histograms with the highest probability p from $hist_{ordered}$.

6. Perform a lookup $map(t_1, t_2)$ to find any curvature models that lie between the top two curvature models with the highest probability.

7. If a series of models from $C_{interpolated}$ is found, perform steps 1-6 with these interpolated models to find a new set of histograms, $hist_{ordered,interpolated}$

8. Compare the probabilities in $hist_{ordered,interpolated}$ to the probabilities in $hist_{ordered}$ to find the highest probability.

9. Take the histogram with the highest probability and set it as the best-fit curvature model for input image I

This method is used for not only the road curvature models, but also the road perspective models and road skew models. Because each model is used to perform a transformation on $I_{remapped}$, they are called transformation models. Using this method allows a large number of transformation models to be considered, but only searched when necessary. For example, if there are 20 initial transformation models and 10 models between each of those initial models, there are a total of 200 possible transformation models. To use all 200 models without this interpolation method, each image frame $I_{remapped}$ would have calculate all 200 shifted images, histograms, and probabilities. Using this interpolation method, there need only be 20 initial searches and 10 interpolated searches, leaving only 30 searches for each frame. This allows the road model fitting to perform at a much more accurate

level while still maintaing an acceptable speed. Once this processes completes, the result is a model of the road lane boundaries in front of the vehicle.

$$Road = (l, r, t) \tag{3.22}$$

In Equation **??**, l is the column index of $I_{remapped}$ of the left lane boundary, r is the column index of the right lane boundary, and t is the road transformation model with the highest probability. Since $I_{remapped}$ is directly related to the world frame, the following lateral offsets in relation to the vehicle can be determined in meters.

$$\text{left offset} = \frac{\left(\frac{WIDTH}{2}\right) - l}{SCALE} \tag{3.23}$$

$$\text{right offset} = \frac{r - \left(\frac{WIDTH}{2}\right)}{SCALE} \tag{3.24}$$

Here, $WIDTH$ is the width of $I_{remapped}$, $HEIGHT$ is the height of $I_{remapped}$, and $SCALE$ is the pre-defined scalar used to scale meters in the world frame to pixels in $I_{remapped}$. $Road = (l, r, t)$ can also be used to project the detected lane boundaries back onto the original image using the inverse perspective mapping transform $I \rightarrow W$.

3.5 Road Color

3.5.1 Color Extraction

Once the road lane boundaries are found, it can be assumed that the area within those lane boundaries are road surface. In order to correctly identify intersecting roads, color is used as an identifier to determine what surfaces outside of the lane boundaries are also road. Dahlkamp [10] outlined a method for determining color models of dirt roads for the 2005 DARPA Grand Challenge. They first used a laser range finder to detect an area in front of the vehicle that could be considered safe to drive and without obstacles. Then they projected this area onto the image frame

captured by their camera and used that area to extract a color model. The color model extracted from that detected area was then used in a search of the entire image frame, finding other surfaces in the image with a similar color. They assume that if the color of additional areas in the image match the color of a safe drivable area found with the laser scanner, those matching areas are also safe and drivable. A similar idea is applied here to roads. Once the road lane boundaries are found, the area within those boundaries can be considered drivable road. The color extracted from the lane is used to search the entire image, finding other surfaces that are also assumed to be road. Since dirt roads have a wider variation of colors than most urban roads, the statistical color model used by Dahklamp is better able to find dissimilarly colored intersecting roads than using a single color model like that used by He in [16].

Many color-based road extraction methods use a multivariate Gaussian distribution to model the color of the road. In some of these methods, the color is defined using only mean color values. For a given $m \times n$ road image I_{road}, the mean is calculated for each color channel.

$$\mu = \frac{1}{m-1}\frac{1}{n-1}\sum_{r=0}^{m}\sum_{c=0}^{n} I_{road}(c,r) \tag{3.25}$$

This allows for the color of the road to be represented by the following.

$$color = \begin{bmatrix} \mu_r \\ \mu_g \\ \mu_b \end{bmatrix} \tag{3.26}$$

Other color-based methods use both the mean and variance of each color channel to more accurately model the road color.

$$\sigma^2 = \frac{1}{m-1}\frac{1}{n-1}\sum_{r=0}^{m}\sum_{c=0}^{n} \left(I_{road}(c,r) - \mu\right)^2 \tag{3.27}$$

The road surface can then be compared to a multivariate Gaussian distribution. The problem with these methods is that they consider each color channel

separately.

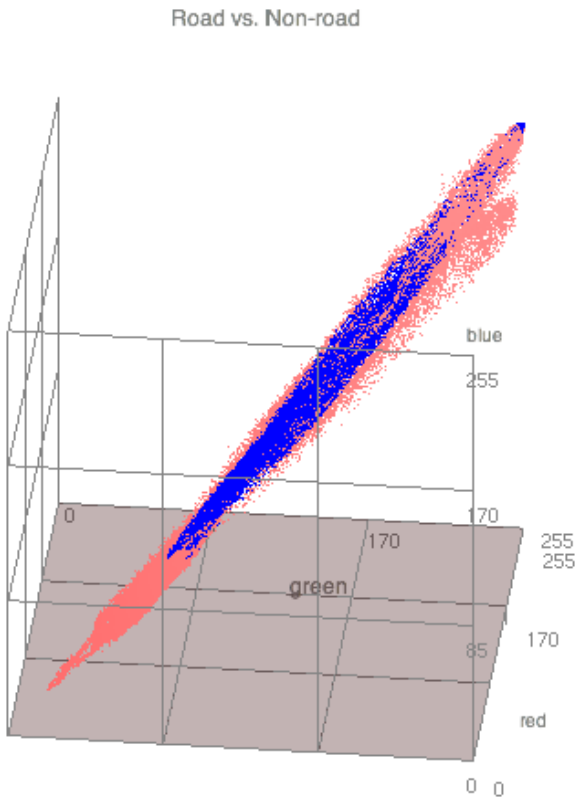


Figure 3.18: Road vs Non-road Color Distribution

Figure 3.18 shows the distribution of road vs. non-road pixels over 10 image frames plotted in RGB space. Blue pixels represent road pixels and red pixels represent non-road pixels. Modeling this distribution is difficult because of its odd shape. Since the red, green, and blue values of a color pixel might all have different relations to each other, it is more statistically accurate to model the means and the covariance for the pixels. This maintains the correlation between the red, green, and blue channels. The covariance of n pairs of two variables is found with the following equation.

$$C(x,y) = \frac{1}{n}\sum_{i=0}^{n-1}(x-\mu_x)(y-\mu_y) \tag{3.28}$$

Using Equation 3.28, the covariance of n pixels is found using the following equation.

$$Cov(r,g,b) = \begin{pmatrix} Cov(r,r) & Cov(r,g) & Cov(r,b) \\ Cov(r,g) & Cov(g,g) & Cov(g,b) \\ Cov(r,b) & Cov(g,b) & Cov(b,b) \end{pmatrix} \tag{3.29}$$

While a single color model can accurately represent roads with little variation, roads captured by a downward pitched camera often exhibit a color variation between the bottom of the image closest to the camera and the top of the image further down the road. To compensate for this and develop a more robust method of determining the road color, multiple color models can be extracted from each image frame to develop a more accurate color model.

To build multiple color models for a given image frame, a k-means clustering algorithm is used. K-means is a clustering algorithm that attempts to cluster n objects into k separate partitions where $k < n$. The image frame $I_{color,remapped}$ with lane boundary model $m = (l, r, t)$ is clustered using the following k-means algorithm.

Algorithm 3.3 K-means pixel clustering

$C = \{c_1, c_2, ..., c_n\}$ (cluster centroids)
for $col = l$ to r **do**
  for $row = 0$ to $HEIGHT$ **do**
    pixels.add($I_{color,remapped}(c, r)$)
  end for
end for
Set $C_i, i \in \{0, 1, ..k\}$ to random pixel from $pixels$
while C has changed **do**
  for each $pixel$ in $pixels$ **do**
    $pixel.cluster = min\,[distance(pixel, C)]$ (find closest cluster to pixel)
  end for
  for $i = 0$ to k **do**
    $C_i.mean$ = $mean$(pixels with cluster=i) (update centroid means)
  end for
end while
for $i = 0$ to k **do**
  $C_i.covariance$ = $covariance$(pixels with cluster=i) (update centroid covariance)
end for

When Algorithm 3.3 has completed, the result is k color models, each with mean μ_i and covariance Σ_i. Using these clusters alone can have acceptable results, but the problem arises when an inaccurate lane boundary model is used or when momentary road clutter occurs. This causes non-road colors to become their own clusters, thereby making them appear as valid road pixels. If the lane boundary model accidently causes pixels out of the road to be clustered, the resulting cluster will be invalid.

To deal with the problem of invalid pixels being introduced, clusters are weighted by pixel mass and learned over time. For each image frame, Algorithm 3.3 is used to find k training color models. These training models are then used to update l learned models based on the following conditions.

1. If trained model T overlaps a learned model L based on

$$(\mu_L - \mu_T)^T(\Sigma_L + \Sigma_T)^{-1}(\mu_L - \mu_T) \leq 1 \tag{3.30}$$

the models are considered similar enough that the learned model L can be updated with the training model T according to the following

$$\mu_L \leftarrow (m_L\mu_L + m_T\mu_T)/(m_L + m_T) \tag{3.31}$$

$$\Sigma_L \leftarrow (m_L\Sigma_L + m_T\Sigma_T)/(m_L + m_T) \tag{3.32}$$

$$m_L \leftarrow (m_L + m_T) \tag{3.33}$$

where m_i is the mass of the model set to the number of pixels associated with that model.

2. If trained model T overlaps no learned model and there are learned models still set to null, replace that model with T

3. If trained model T overlaps no learned model and no learned models are null, use T to replace the learned model L that has the lowest mass

By managing the trained and learned models in this way, the system can adapt to quickly changing road color while reoccurring color models are updated and gain more mass. If non-road pixels are found within the lane boundaries, the training model it creates will be introduced with a small mass, therefore being ignored when compared to the learned clusters with higher mass.

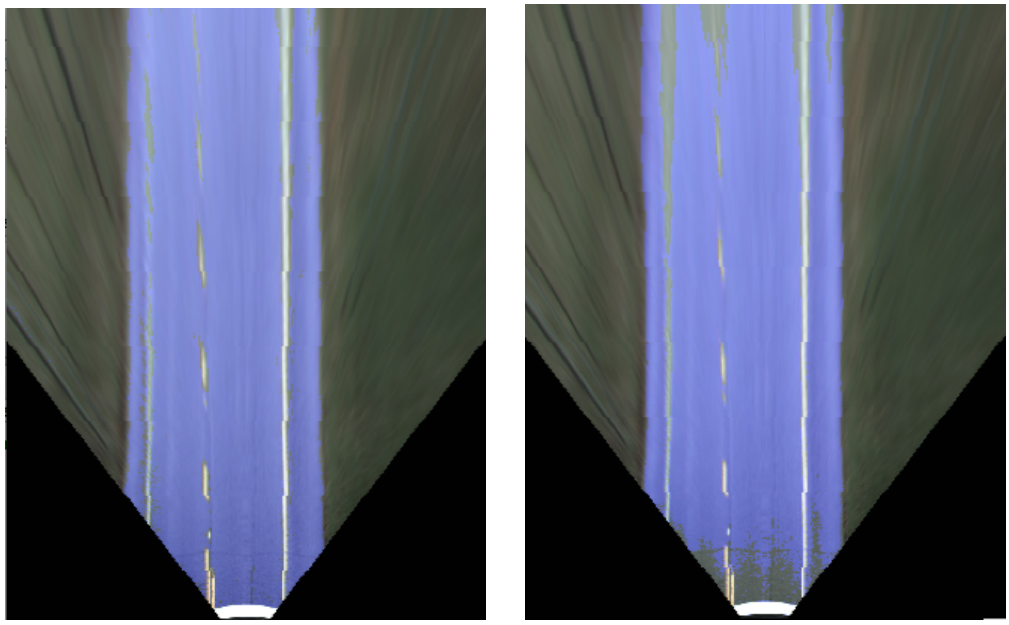
3.5.2 Color Analysis

Once the training models are found and the learned models are updated, the entire remapped color image $I_{color,remapped}$ is analyzed using the learned color models. Since each learned model L is characterized by mean values μ_i and covariance matrix Σ_i, a Mahalanobis distance is used as a comparison between a given pixel $I_{color,remapped}(c, r)$ and learned model L_i. The Mahalanobis distance was chosen because of its ability to consider correlations between variables which can identify

patterns. The Mahalanobis distance between two values is defined as

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T P^{-1} (\vec{x} - \vec{y})} \tag{3.34}$$

where P is the covariance matrix. To increase speed and reduce the consideration of spurious invalid models, learned models are only considered if their mass is above a certain percentage of the learned model with the largest mass. This ensures that constantly updated learned models are always considered and models that appear for only a single frame are not considered. The masses of learned color models are also degraded by a pre-defined degradation rate before each update to prevent any single model from growing a mass large enough that no new values can be introduced.



(a) Remapped Image with Clustered Models

(b) Remapped Image with Non-Clustered Model

Figure 3.19: Color Extraction Comparison

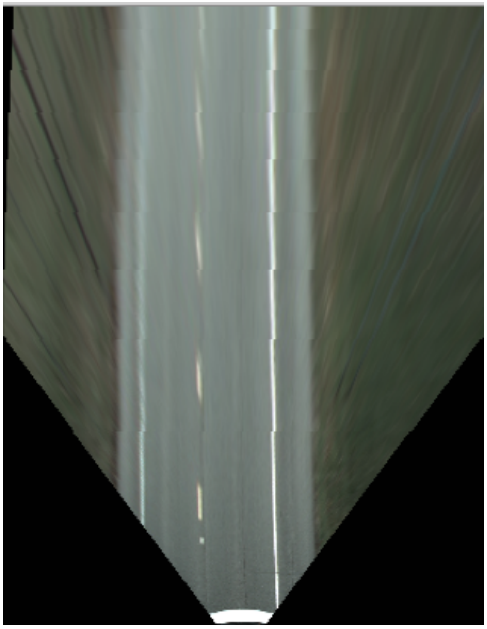
Figure 3.19b shows color extraction done with a single multivariate Gaussian color model. The Gaussian model leaves pixels incorrectly classified both near the vehicle and near the horizon. Figure 3.19a shows the color extraction using

the learned color model method outlined above. All pixels are correctly classified as road and no pixels are classified incorrectly. The use of covariances and correlations also benefits detection of many intersecting roads that are a dissimilar color to the main road. A driveway, for example, may have darker pavement but also maintain the same correlation between its color channels. Because the Mahalanobis distance considers these correlations, there is a better chance of that darker pavement being extracted than when using a multivariate Gaussian model alone.

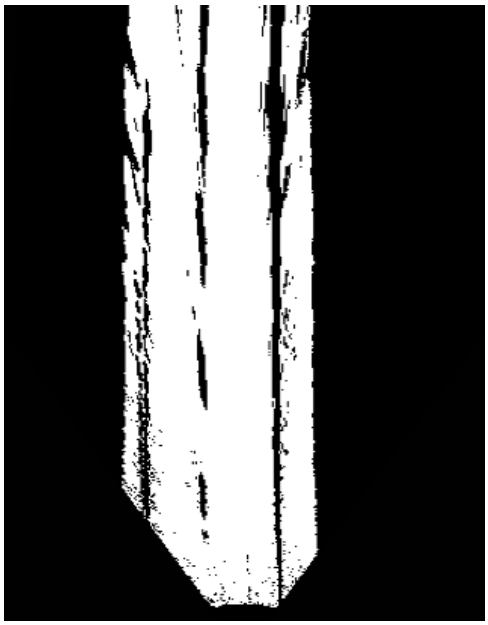
3.6 Outer Boundary Extraction

To successfully detect intersecting roads, it is necessary to know the location of both the road lane boundary and the outer boundary of the road. Assuming that the road lane boundaries and the boundary between the road and its surrounding are all parallel lines, both the lane and the outer road boundaries can be described using the same road transformation model.

The first step is to take the image resulting from the color analysis step, which is a binary thresholded image representing the extracted road area.



(a) Color Remapped Image Before Thresholding



(b) Color Remapped Image After Thresholding

Figure 3.20: Color Binary Extraction

As Figure 3.20b shows, the binary version of the extracted color pixels exhibits gaps at the location of the road lines. Since the color extracted in the initial steps was taken from between the lane boundaries, it is expected that these lane boundaries are excluded from the color models. To make the process of finding the outer road boundaries easier, these gaps are filled using a morphological process called dilation. This process attempts to grow the road area until smaller non-road gaps are filled in. Another morphological process called erosion is then used to contract the outer edges not filled in by the dilation. This assures that the outer road boundaries remain the same while the inner holes are filled.

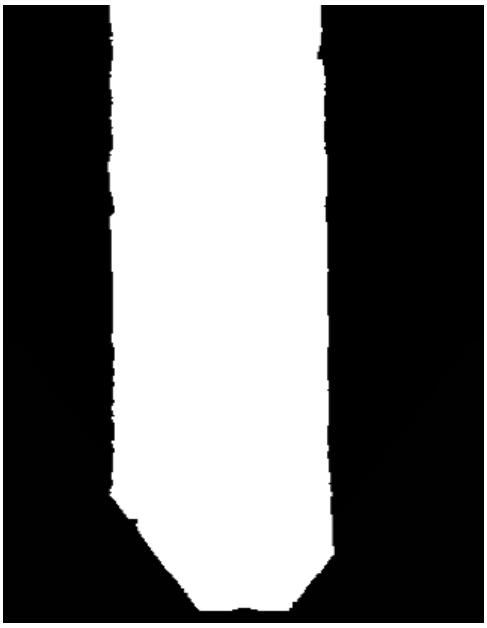


Figure 3.21: Binary Road Image After Dilation and Erosion

Figure 3.21 shows the binary road image after the morphological processes are performed. The gaps created by the road lines are now filled in and the result is a solid binary image where the white pixels make up the entire road area and the black pixels represent pixels that do not belong to the road.

To find the outer road boundaries in Figure 3.20b, the left and right boundaries are searched independently. Since the road lane boundaries are already known and the binary road image in Figure 3.20b has already been shifted to undo the road transformation model, the binary road image in Figure 3.20b can be searched in a column-wise manner.

Algorithm 3.4 Outer Boundary Determination

$left$ = column of left lane boundary
$right$ = column of right lane boundary
(look for left outer boundary)
for $c = left$ to 0 **do**
  for $r = 0$ to $HEIGHT$ **do**
    $columnPixels+ = I_{road,remapped}$
  end for
  if $columnPixels/HEIGHT < outerThreshold$ **then**
    $leftOuterBoundary = c$
    break
  end if
end for
(look for right outer boundary)
for $c = right$ to $WIDTH$ **do**
  for $r = 0$ to $HEIGHT$ **do**
    $columnPixels+ = I_{road,remapped}$
  end for
  if $columnPixels/HEIGHT < outerThreshold$ **then**
    $leftOuterBoundary = c$
    break
  end if
end for

Algorithm 3.4 shows how the outer boundaries are searched for. This algorithm relies on a set threshold $outerThreshold$ to decide when there are too few road pixels in a given column to consider that column as part of the road. $outerThreshold$ can be set as a percentage, making it independent of image scale. It can be set very conservatively so that over 90% of the column must be road pixels to consider that column part of the road, or it could be set to a smaller percentage to account for any noise in the binary road image. The result of this process is another road model, $outerRoad = (l, r, t)$ where l is the column index of the remapped image of the left outer boundary, r is the column index of the right outer boundary, and t is the road transformation model found during the road lane boundary extraction process.

3.7 Intersection Detection

The purpose of the intersection detection step is to take the road transformation model learned from the lane boundary determination, the road area learned from the color extraction and analysis step, and the location of the outer road boundaries from the preceding step, and use this knowledge to find any possible intersections. In this step, an intersection is defined by a section of road area that extends from the outer boundary of the road to the outer boundary of the image frame. For this step, all that need be known is the remapped, transformed, color extracted, binary road image $I_{road,remapped}$ and the outer boundary model $outerRoad$.

(a) Color Intersection Image (b) Binary Intersection Image

Figure 3.22: Road Intersection Image

Figure 3.22b shows the road extraction of the color image in Figure 3.22a. As seen in Figure 3.22b, the intersecting road is clearly seen as the road area located beyond the outer boundary of the main road on the right side of the road. Since only the road area outside of the outer road boundaries need be analyzed, only the area to the left of the leftmost outer road boundary and to the right of the right

most outer road boundary need be searched.

The intersection area is analyzed using a similar histogram-based method to that used to analyze road lane boundaries. Two histograms are first defined, $hist_{left}$ and $hist_{right}$. These histograms have their width set to the height of $I_{road,remapped}$. The histograms are then populated with the area around the outer road boundaries using Algorithm 3.5.

Algorithm 3.5 Intersection Histogram

$left$ = column of left outer boundary
$right$ = column of right outer boundary
(populate left intersection histogram)
for $row = 0$ to $I_{road,remapped}.height$ **do**
  for $col = 0$ to $left$ **do**
    $colValues+ = I_{road,remapped}(c, r)$
  end for
  $hist_{left}(row) = colValues/(right - 0)$
end for
(populate right intersection histogram)
for $row = 0$ to $I_{road,remapped}.height$ **do**
  for $col = right$ to $I_{road,remapped}.width$ **do**
    $colValues+ = I_{road,remapped}(c, r)$
  end for
  $hist_{right}(row) = colValues/(I_{road,remapped}.width - right)$
end for

The result is a normalized histogram for each side of the road, indexed by row, representing the percentage of that row which is made up of road pixels.

Figure 3.23 shows the histogram resulting from the binary road image in Figure 3.22b. The location of the intersection can be distinguished easily by the rows where the road area extends from the outer road boundary to the outer image boundary. The location of the intersection is determined using a rectangular search area that searches for rows where the road area percentage is higher than a predefined threshold $intersectionThreshold$.

Algorithm 3.6 shows the how an intersection histogram is searched to find a

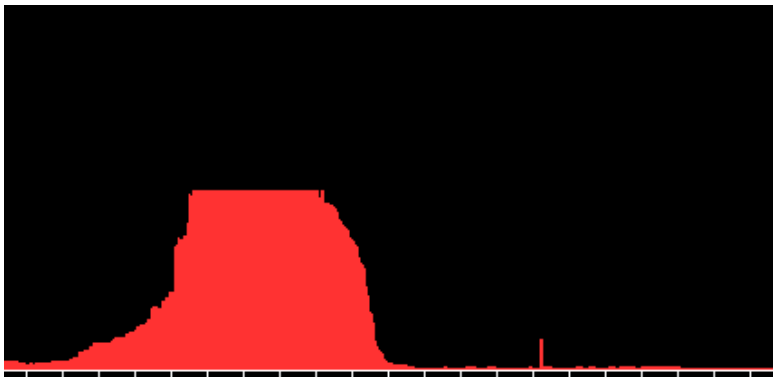


Figure 3.23: Road Intersection Histogram

Algorithm 3.6 Intersection Extraction

$intersectionTop = null$
$intersectionBottom = null$
for $row = 0$ to $I_{road,remapped}.height$ **do**
  if $hist_i(row) > intersectionThreshold$ **then**
    if $intersectionTop == null$ **then**
      $intersectionTop = row$
    else
      $intersectionBottom = row$
    end if
  end if
end for

rectangular area that defines an intersection. While this type of search favors intersections that intersect the main road at a perpendicular angle, the limited viewing angle of most single camera configurations cause intersecting roads to span from the outer road boundaries to the outer image frame, allowing them to be extracted using this method.

The intersection extraction is performed on both the left and right intersection histograms separately. The result is two intersection models, int_{right} and int_{left}, where

$$int_i = (rowTop, rowBottom, colLeft, colRight) \tag{3.35}$$

In the intersection model, $rowTop$ and $rowBottom$ are the row index of the remapped image for the top and bottom of the detected intersection. $colLeft$ and $colRight$ are

the column index of the remapped image for the left and right boundaries of the detected intersection. To convert these back to world frame coordinates, the following equations are used

$$\text{left intersection forward offset} = int_{left}.rowBottom/SCALE \tag{3.36}$$

$$\text{left intersection lateral offset} = int_{left}.colRight/SCALE \tag{3.37}$$

$$\text{right intersection forward offset} = int_{right}.rowBottom/SCALE \tag{3.38}$$

$$\text{right intersection lateral offset} = int_{right}.colLeft/SCALE \tag{3.39}$$

After the left or right intersections are transformed back into the world frame, the offsets are in meters and can be used for localization purposes.

Chapter 4

Testing and Results

4.1 Equipment

4.1.1 Computer

The intersection detection method outlined here was implemented in C++ using the Intel Open Source Computer Vision Libraries (OpenCV)[19]. These libraries provided a framework for many basic computer vision operations and functions required to develop this application. OpenCV provided functions for camera distortion correction, edge detection, and morphological processing that are optimized for use on an Intel processor. OpenCV also allowed for easy viewing of images and drawing capabilities for presenting results in a visual manner. OpenCV is commonly used in the computer vision and robotics field due to its optimized functions and easy GUI capabilities.

The algorithm was implemented for use on a single computer. The machine utilized was a laptop capable of capturing images directly from a camera so the system could be used in real time from a moving vehicle. The development was done in C++ using the Xcode development environment on Mac OS X 10.5. All functions used were from standard C++ libraries. Because of this and the cross-platform nature of OpenCV, the application can easily be ported to either a Windows or Linux computer. This was an important factor since there is no standard

platform for robotics development. The Table 4.1 summarizes the test machine and its relevant specifications.

Table 4.1: Development and Test Parameters

Parameters	Value
CPU Architecture	Intel Core 2 Duo
CPU Clock Rate	2.4 GHz
System RAM	4.0 GB 667 MHz DDR2
Operating System	Mac OS X 10.5.2
Kernel	Darwin Kernel 9.2.0
Development Environment	Xcode 3.0
Compiler	GCC 4.0.1

4.1.2 Camera

The algorithm was developed and tested using a Basler A311fc. This camera was used because it is considerably higher quality then a standard computer webcam and provides color images with very little noise. This was important because noisy images provide poor results when used with edge detection. A noisy pixel is one that appears to be a random value, completely independent of what the pixel value should be. When using edge detection, this noise may appear as a hard edge, introducing invalid information into the system. This problem is usually solved by using a smoothing or median filter on the image. A higher-quality camera removes the need for this additional filtering and provides a truer image.

The camera exposure on the Basler A311fc camera is configurable via the firewire bus. This allows the application to control how dark or bright the image is and attempt to maintain an ideal level of brightness. This is more desirable than most standard webcams that attempt to perform built-in automatic exposure control because they are often not able to be configured as accurately.

The Basler A311fc camera is also desirable for this application because it uses a C-type lens mount, allowing it to be mated with a variety of different camera

lenses. These lenses provide different focal lengths and viewing angles that provide different types of information to the system. A lens with a wider viewing angle can see further out to either side of the vehicle, but can see less far in front of the vehicle. The lens can be selected depending on the application desired usage or the expected roads to be traveled. The algorithm was tested using two different camera lenses to assure that it was functional independently of the camera. The first lens is Pentax 4.2 mm 1:1.6 wide lens with a viewing angle of $74^\circ \times 60^\circ$. The other lens was a Pentax 8.5 mm 1:1.5 lens with a viewing angle of $40^\circ \times 30^\circ$.

4.2 outlines the specifications of the camera.

Table 4.2: Camera Parameters

Parameters	Value
Sensor Size (HxV Pixels)	658x492
Sensor Type	Progressive CCD
Pixel Size (in μm)	9.9x9.9
Max Frame Rate at Full Resolution	73 fps
Video Output Format	Raw16 (12bpp)
Interface	1394 Firewire bus
Exposure Control	Programmable via 1394 bus
Lens Mount	C-Mount

The camera was interfaced using the CMU 1394 Digital Camera Driver [1]. These drivers provided a stable interface to the camera with functions available for configuring camera parameters over the Firewire 1394 bus.

4.2 Calibration

4.2.1 Intrinsic Parameters

Both lenses were calibrated using the Camera Calibration Toolbox for Matlab [24]. This toolbox is capable of determining the intrinsic camera parameters from Table 4.3.

Table 4.3: Intrinsic Camera Parameters

Parameter	Description
Focal Length	Distance from the center of the lens to the principal focal point
Principal Point	The point at which light rays from a lens converge
Skew Coefficient	The angle the x and y pixel axes
Distortions	The radial and tangential distortion coefficients

The calibration toolbox works by using the un-calibrated camera to capture images of a black and white checkerboard pattern from a variety of different angles and distances. The outer edges of the checkerboard pattern in each image are selected by the user and the toolbox must then be told the number of squares in the pattern and the size of each square. With this information alone, the camera calibration toolbox is able to determine each parameter from Table 4.3

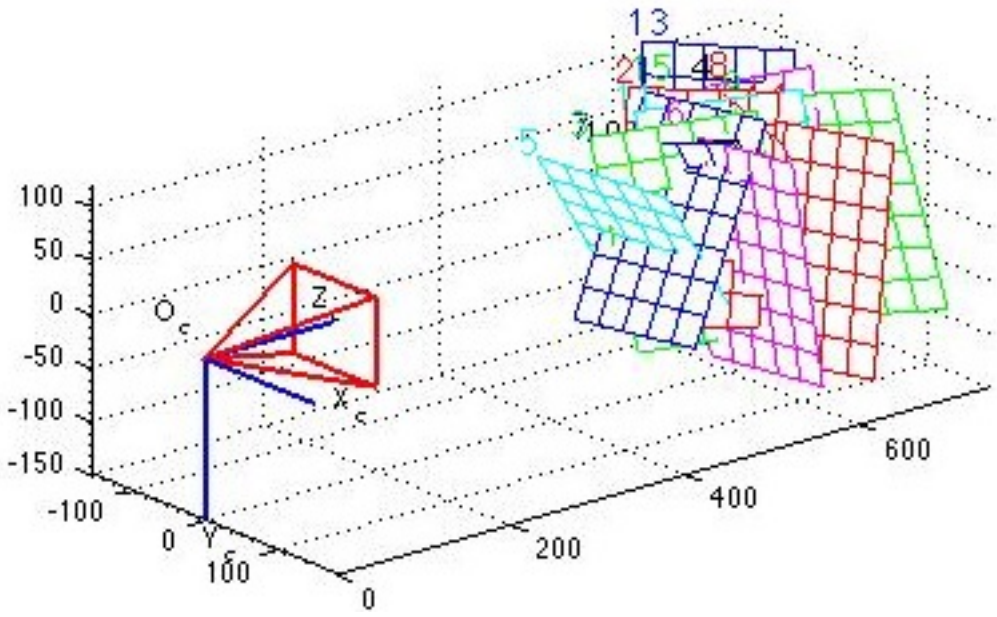


Figure 4.1: Camera Calibration Toolbox Results

Figure 4.1 shows how the camera calibration toolbox is able to determine the position of each test pattern in relation to the camera and use that information to understand lens parameters and distortions.

The calibration process was done separately for each lens since they each have their own unique distortions. The calibration provides the focal length in both the x and y direction as (f_x, f_y). With the focal length known from calibration and the camera sensor information known from the manufacturer, the angular aperture (viewing angles) can be determined using the following formulas where $pixel$ is the number of pixels in the sensor and $sensorsize_x$ is the size of the pixel in meters.

$$d_x = pixel_x \times sensorsize_x \tag{4.1}$$

$$d_y = pixel_y \times sensorsize_y \tag{4.2}$$

$$a_x = 2 \times \arctan\left(\frac{d_x}{2 \times f_x}\right) \tag{4.3}$$

$$a_y = 2 \times \arctan\left(\frac{d_y}{2 \times f_y}\right) \tag{4.4}$$

The angular aperture of each lens is necessary to perform inverse perspective mapping.

4.2.2 Extrinsic Parameters

For the inverse perspective mapping done in this method, two extrinsic camera parameters had to be determined. These parameters are the camera height h and the camera downward pitch θ. These parameters are shown in Figure 4.2.

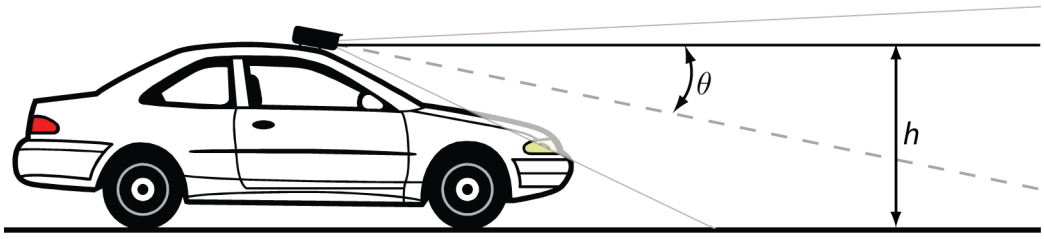


Figure 4.2: Extrinsic Camera Parameters

To measure these parameters, the following calibration setup was used.

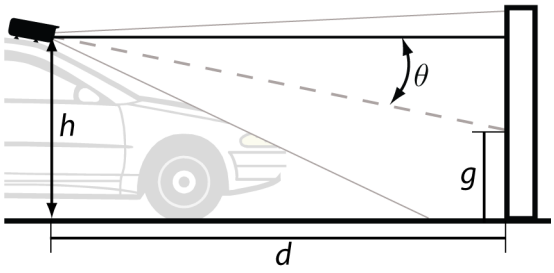


Figure 4.3: Extrinsic Camera Parameter Measurement

In Figure 4.3, there are three required measurements shown. The vehicle was first placed on a planar surface in front of a vertical wall. The height of the camera, h, was measured as the distance from the ground to the center of the camera lens. Using the image output from the camera, a horizontal and vertical line were overlaid on the image that passed through the center of the horizontal and vertical axes. The point at which these two lines met determined the camera's optical axis. The point where the optical axis was projected to the wall in front of the vehicle was marked on the wall, and the distance between that point and the ground was measured as distance g. The distance from the point where the camera lens to the wall was measured as distance d. The downward pitch of the camera θ was then calculated using the following formula.

$$\theta = tan^{-1}\left(\frac{h-g}{d}\right) \tag{4.5}$$

4.3 Test Results

To develop and test this algorithm, various data sets were gathered. These data sets consist of a series of images recorded in JPEG format at the highest frame rate available from the camera. In the case of the Basler A311fc, the images were

recorded at 7 frames per second in 658x492 RGB color pixels. Data sets were gathered with both the normal and the wide-angled lens on a variety of different two lane roads on multiple days. The data sets were gathered on roads around Ithaca, NY, Seneca, NY, and on the testing site of the Cornell DARPA Urban Challenge Team at the Seneca Army Depot in Ovid, NY.

The following images show various key results obtained by running the algorithm on various different data sets using the same program parameters for all sets, with the exception of different camera parameters for the different lenses. Each result consists of the original image , the remapped edge image, and the detected road model. Overlaid on each image are the visual indicators shown in Table 4.4.

Table 4.4: Visual Indicators

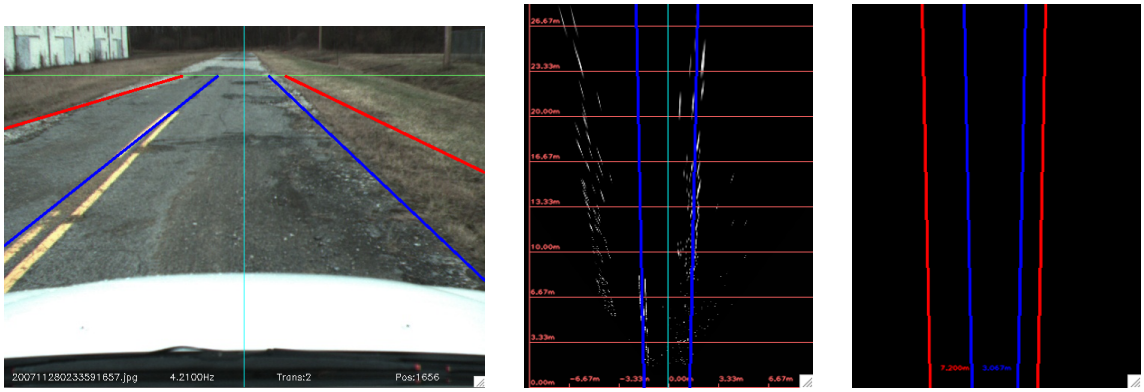
Visual Indicator	Description
thin green line	horizon line
light blue lines	centerline of the camera
dark blue lines	road lane boundaries
red lines	outer road boundaries
green lines	intersection indicator
light red lines	world frame scale in meters

4.3.1 Road Lane Boundary Detection

Straight Roads

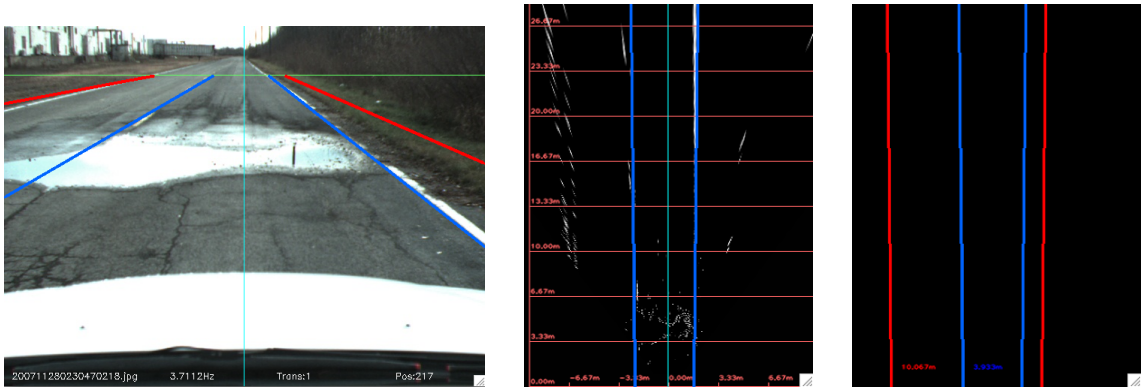
The following images show the results of the road lane boundary detection on various straight roads.

The results show the ability of the road lane boundary algorithm to detect various straight roads. Figure 4.4a shows it successfully detecting a road with no right lane line. The edge detection was able to detect the boundary between the paved road and gravel edge. This image also shows a road transformation model being used to correct for a perspective distortion. The lines in 4.4c move inward towards



(a) Original Image (b) Edge Image (c) Road Model

Figure 4.4: Straight Road Example 1

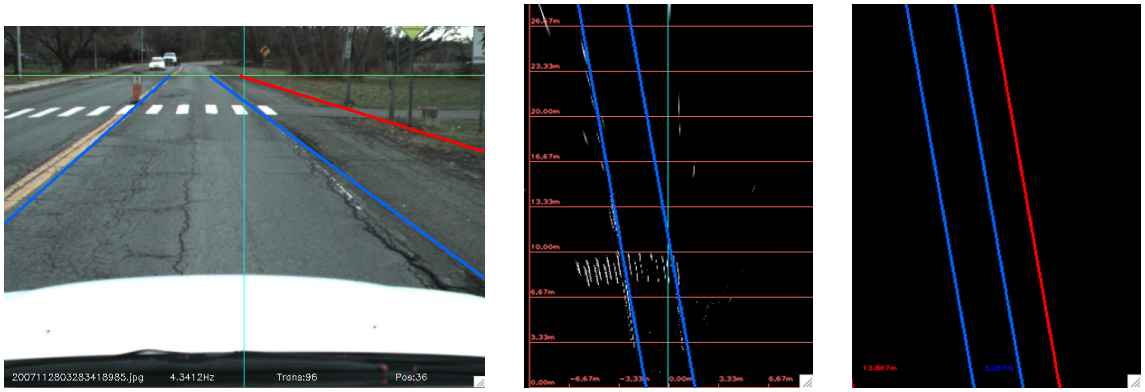


(a) Original Image (b) Edge Image (c) Road Model

Figure 4.5: Straight Road Example 2

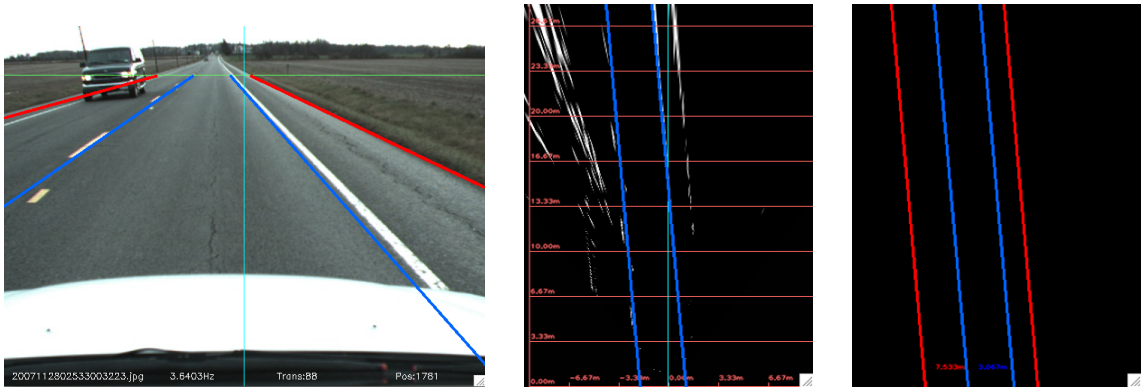
the bottom of the image resulting from a perspective distortion it has corrected for. Figure 4.5a shows the algorithm successfully detecting a road with a large area of water occluding the road and parts of the lane boundary. Figure 4.6a shows the algorithm successfully detecting a road with a painted pedestrian crosswalk through the middle. Figure 4.6c also shows how the road transformation model is correcting for a skew distortion because the car is not pointing directly towards the center of the road. Figure 4.7a shows the algorithm successfully detecting a two lane road with a broken centerline.

This algorithm was designed for use on urban roads, which are traditionally



(a) Original Image (b) Edge Image (c) Road Model

Figure 4.6: Straight Road Example 3



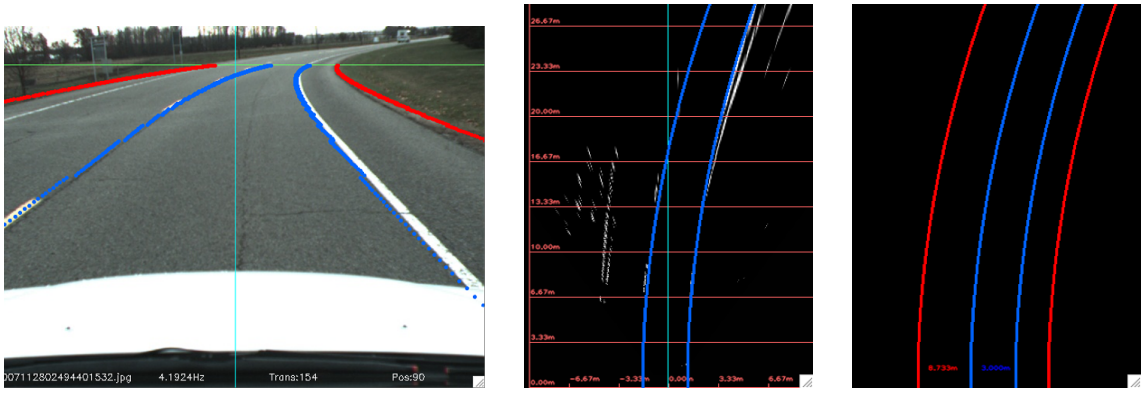
(a) Original Image (b) Edge Image (c) Road Model

Figure 4.7: Straight Road Example 4

two-lane roads with painted lane boundaries and lanes approximately 3 meters wide. These results show its ability to detect these types of lanes while tolerating common variations.

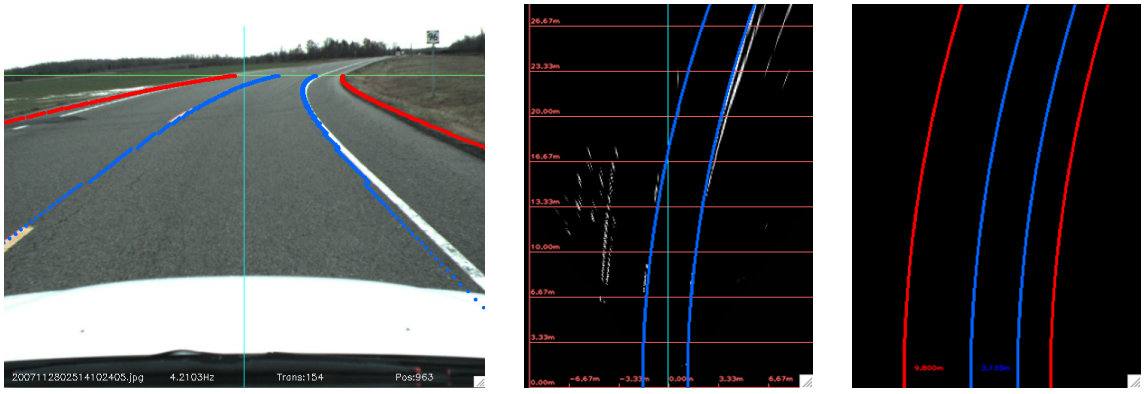
Curved Roads

While intersections appear more commonly on straight roads then on curved roads, the ability to detect curving roads is important to any useful road detection algorithm. These results show the lane boundary algorithm detecting lanes of various curvatures.



(a) Original Image (b) Edge Image (c) Road Model

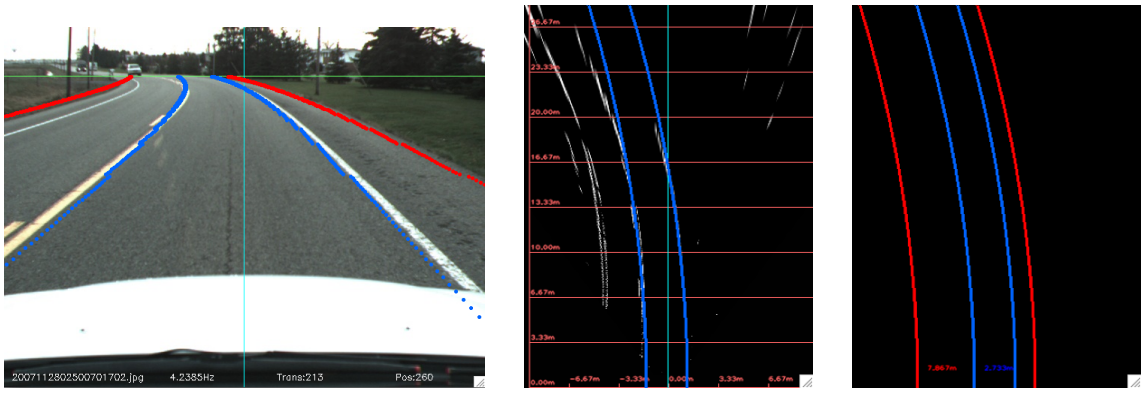
Figure 4.8: Curved Road Example 1



(a) Original Image (b) Edge Image (c) Road Model

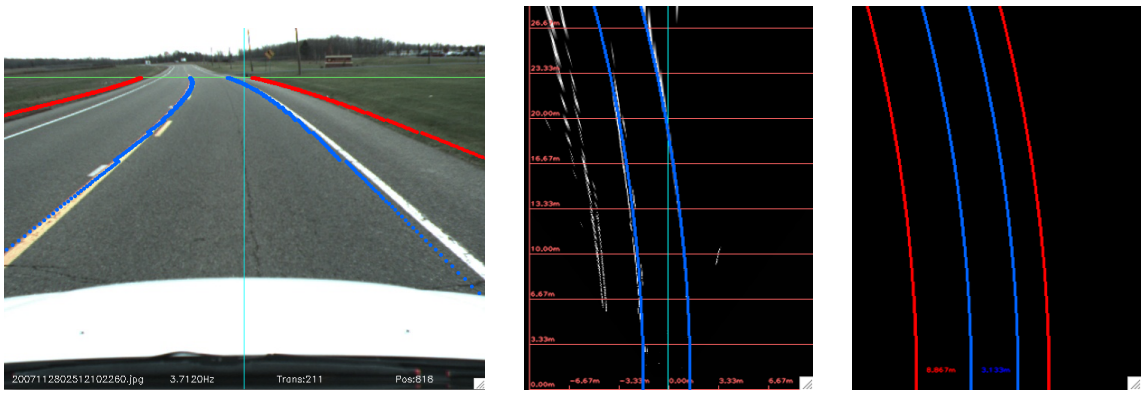
Figure 4.9: Curved Road Example 2

Figure 4.8a and Figure 4.9a show the lane boundary algorithm successfully detecting lanes curving to the right. Figure 4.10a and Figure 4.11a show the lane boundary algorithm successfully detecting lanes curving to the left.



(a) Original Image (b) Edge Image (c) Road Model

Figure 4.10: Curved Road Example 3

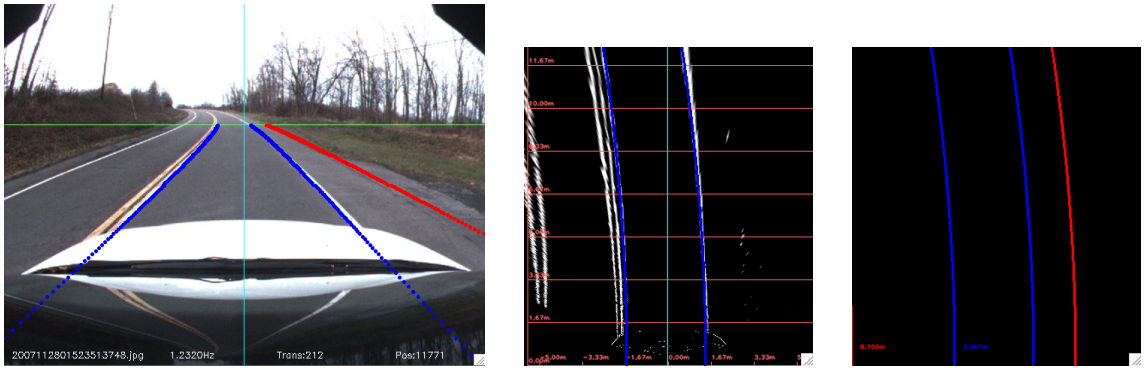


(a) Original Image (b) Edge Image (c) Road Model

Figure 4.11: Curved Road Example 4

Wide Lens

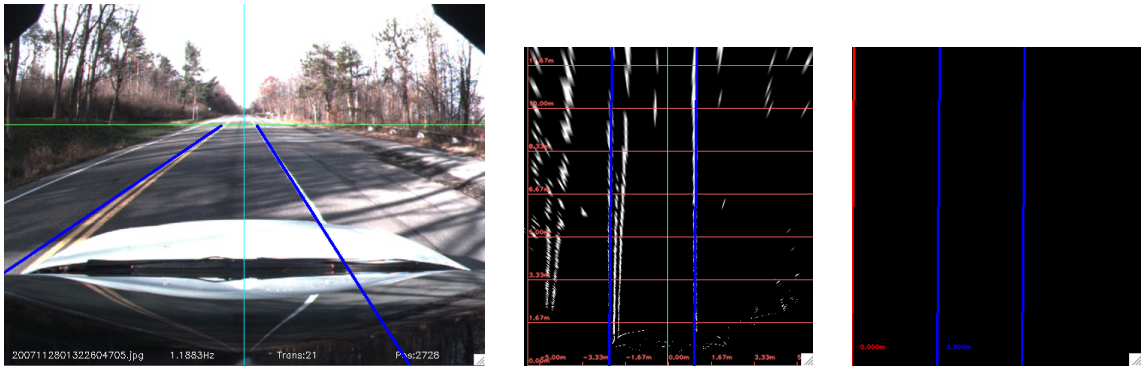
The following results show the algorithm detecting various types of road lane boundaries using the wide angle lens. Due to the very large viewing angle of this lens, fewer road pixels were available for analysis since a large area of the vehicle was captured. The same algorithm parameters were used with this lens that were used with the normal lens. These parameters include thresholds and algorithm variables. The only parameters changed were those specific to the lens such as viewing angle and focal length. The results show the ability of the algorithm to function with a lens that completely changes the view of the road.



(a) Original Image (b) Edge Image (c) Road Model

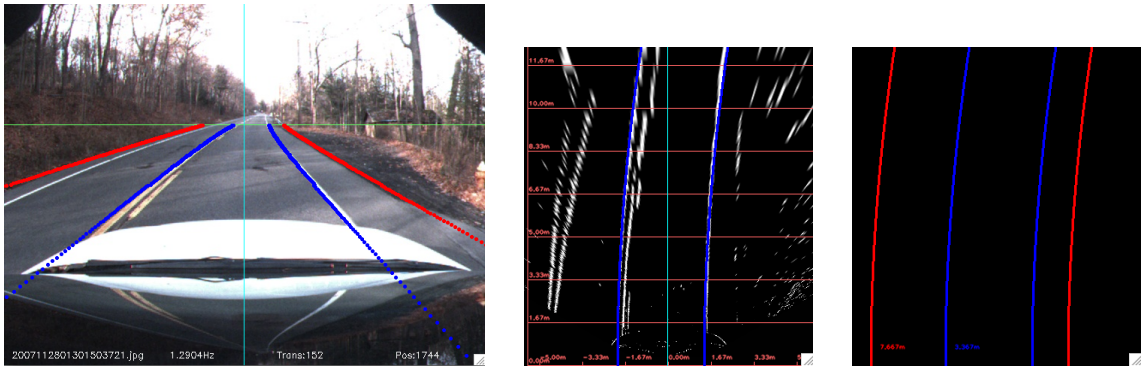
Figure 4.12: Wide Lens Example 1

Figure 4.12a shows the road lane boundary algorithm successfully detecting a road curved slightly to the left. Figure 4.13a shows the lane boundary algorithm successfully detecting a road covered in shadows caused by trees on the roadside. Figure 4.14a shows the algorithm detecting another curved road and 4.15a shows the algorithm successfully detecting a road with a horizontal crosswalk painted on the road.



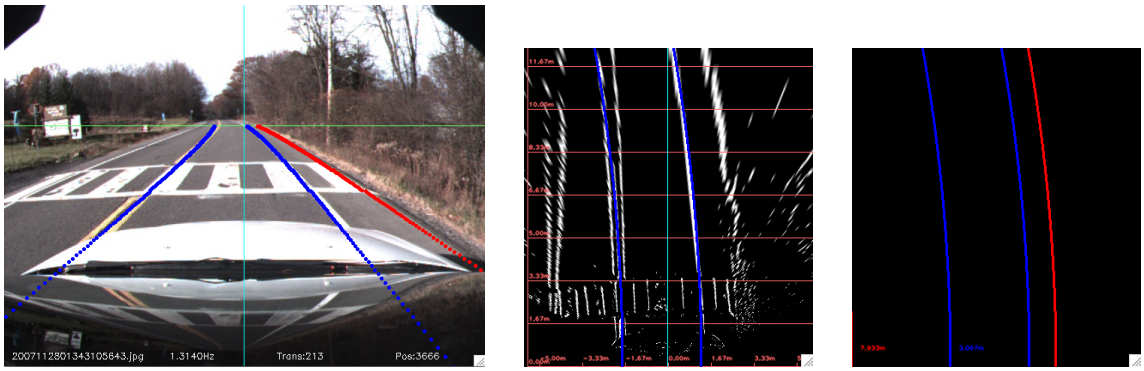
(a) Original Image (b) Edge Image (c) Road Model

Figure 4.13: Wide Lens Example 2



(a) Original Image (b) Edge Image (c) Road Model

Figure 4.14: Wide Lens Example 3

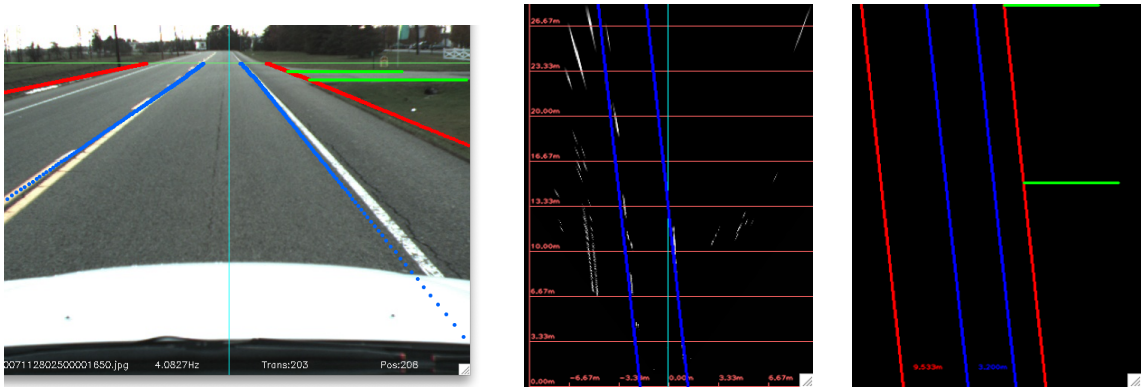


(a) Original Image (b) Edge Image (c) Road Model

Figure 4.15: Wide Lens Example 4

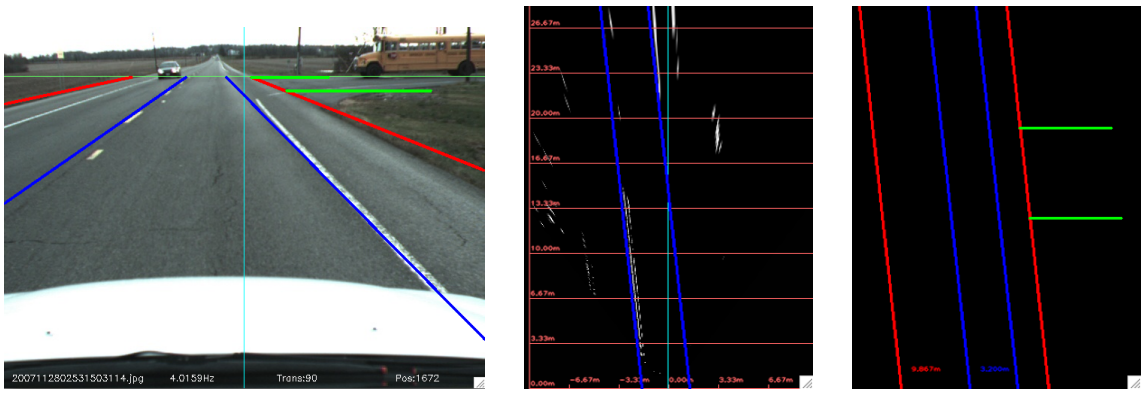
4.4 Intersections

The following results show all aspects of the algorithm working together to successfully detect intersections. When an intersection is detected, it is outlined on top and bottom with green lines. The detected intersection is also displayed on the road model view.



(a) Original Image (b) Edge Image (c) Road Model

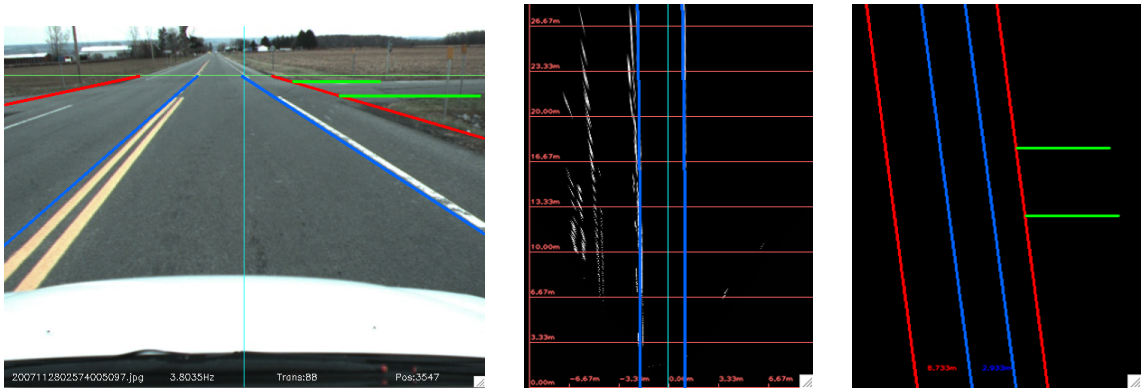
Figure 4.16: Intersection Example 1



(a) Original Image (b) Edge Image (c) Road Model

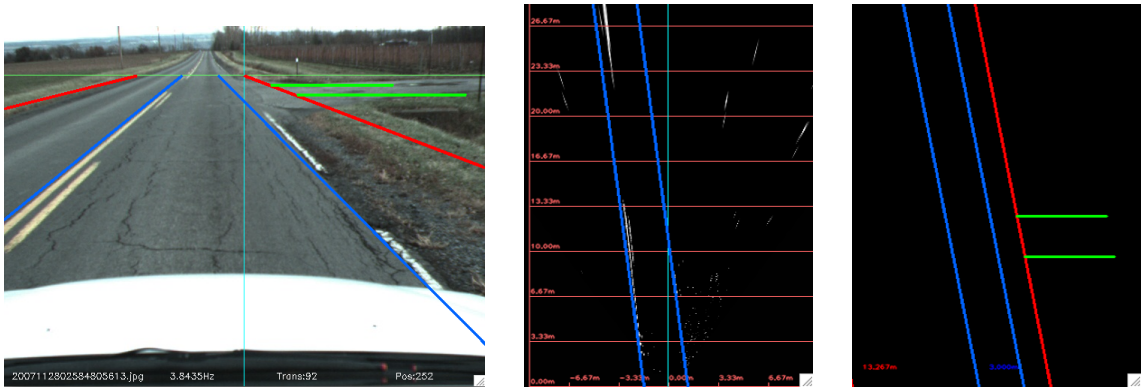
Figure 4.17: Intersection Example 2

Figure 4.16a and Figure 4.17a show intersections being successfully detected on the right side of the road. Figure 4.17a and Figure 4.20a shows the intersection successfully being detected even with a vehicle approaching the intersecting road.



(a) Original Image (b) Edge Image (c) Road Model

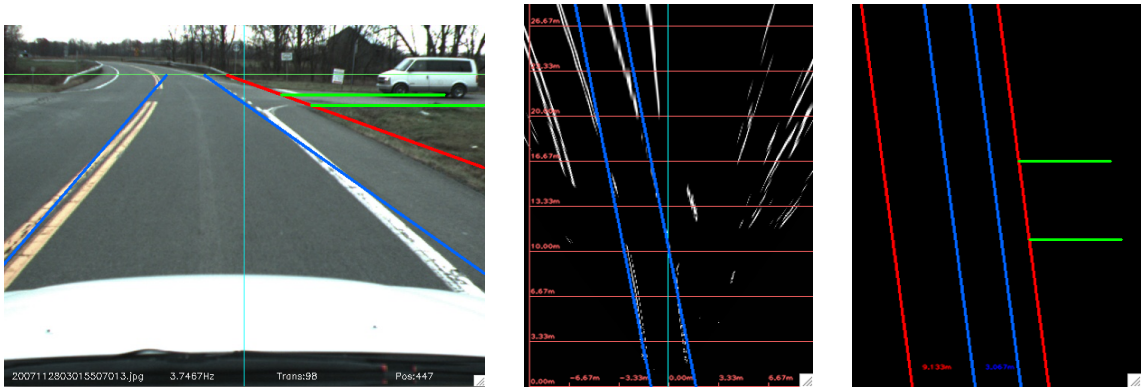
Figure 4.18: Intersection Example 3



(a) Original Image (b) Edge Image (c) Road Model

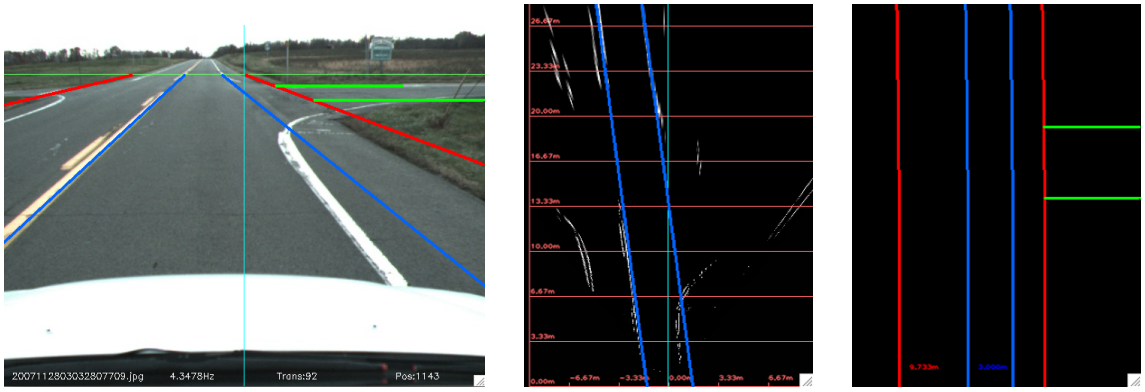
Figure 4.19: Intersection Example 4

Figure 4.22a, Figure 4.23a, 4.24a show intersections being successfully detected on both sides of the road simultaneously. Figure 4.25a show an intersection being detected on the left side of the road.



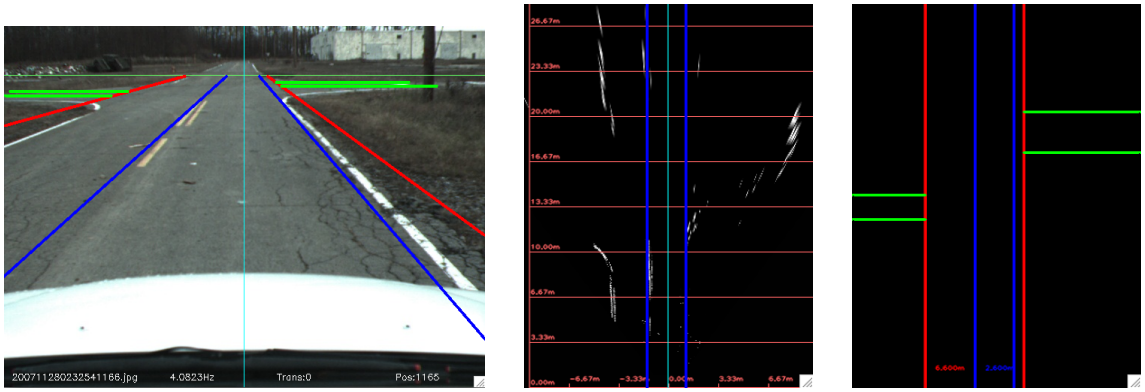
(a) Original Image (b) Edge Image (c) Road Model

Figure 4.20: Intersection Example 5



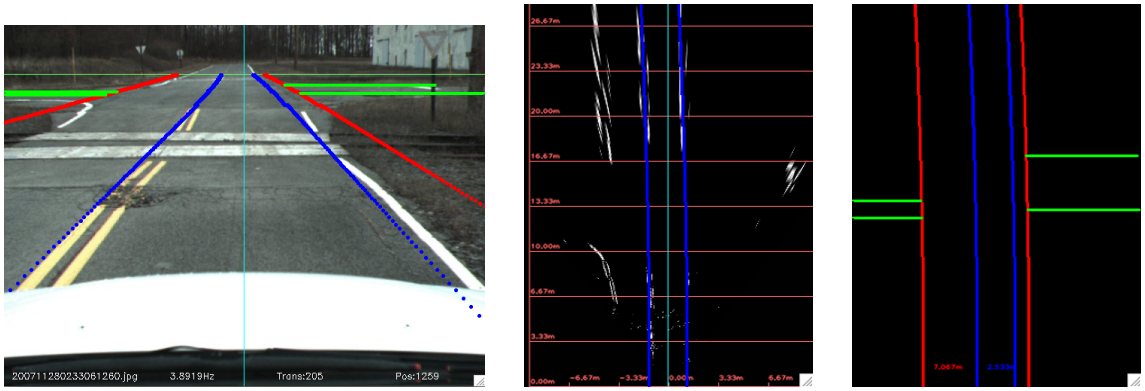
(a) Original Image (b) Edge Image (c) Road Model

Figure 4.21: Intersection Example 6



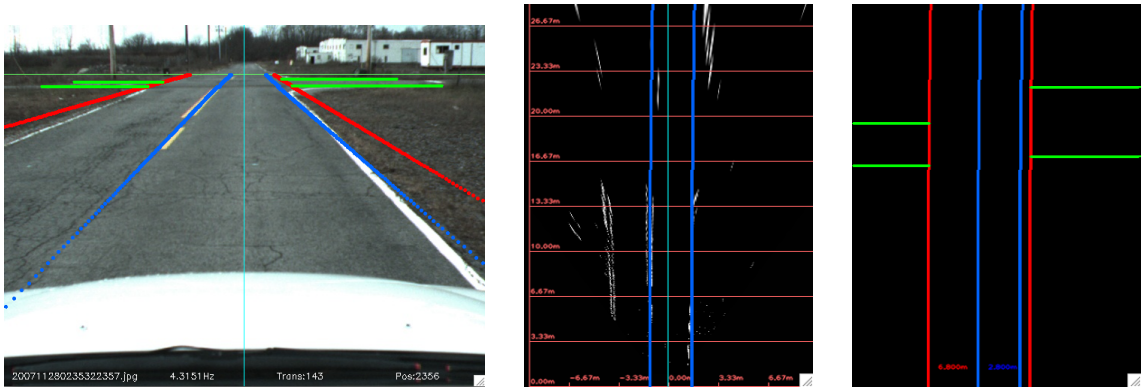
(a) Original Image (b) Edge Image (c) Road Model

Figure 4.22: Intersection Example 7



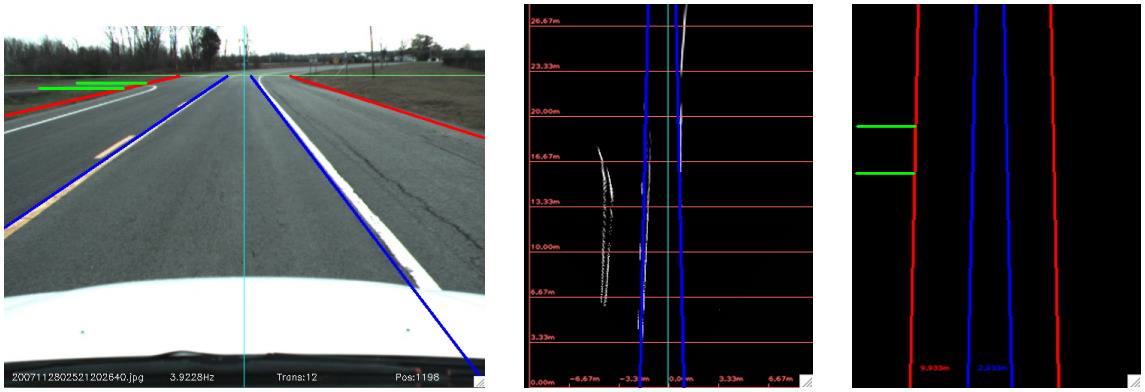
(a) Original Image (b) Edge Image (c) Road Model

Figure 4.23: Intersection Example 8



(a) Original Image (b) Edge Image (c) Road Model

Figure 4.24: Intersection Example 9



(a) Original Image (b) Edge Image (c) Road Model

Figure 4.25: Intersection Example 10

Chapter 5

Result Analysis

Because computer vision is such a difficult task that often requires a level of comprehension beyond the capabilities of current computing, results of a computer vision algorithm are most accurately judged by visual inspection from a human observer. Since the algorithm has parameters that can be changed to alter the results of the algorithm, automated testing was required.

Automated testing was done by selecting a specific data set and creating a truth file. The truth file contained information about each image frame in the data set. For each image frame, there was an indication of whether a left or right intersection was present in that image frame, the location of the left and right lane boundaries, the location of the left and right outer road boundaries, and the best-fit road transformation model. The truth file was constructed using human supervision to specify the presence of intersections and verify the road lane boundary information for each image frame. The intersection detection application was outfitted with the ability to log all information about its results for each image frame, recording lane position, transformation model, intersection presence, and speed. A script was written to analyze these results and compare them to the truth file, providing information about how the results from each changed parameter varied from the truth file and how they compared to each other. The algorithm was then run with varied parameters to produce different results. While this method does not provide an ideal set of parameters for the system and simply shows the algorithm's

response to a particular data set, it does however provide information about how the parameters effect different aspects of the system. Each significant algorithm parameter will be explained separately to highlight its impact on the system.

5.1 Lane Transformation Model Interpolation

The lane model interpolation affects how the road lane transformation models are considered. The transformation model selected determines the shape of the road detected. With interpolation, only an initial set of road transformation models are considered and then a second pass with more specific models are considered. Without interpolation, all transformation models are considered for every single image frame. There are two important effects this process has on the algorithm.

The first effect interpolation has on the algorithm is the resulting transformation model chosen and the location of the lanes. In an ideal situation, there would be no difference between the lane model selected with interpolation and without interpolation and the location of the lanes would be the same. The most important factor is the difference in the lane location. If the transformation model varies slightly but the lanes are still in the same location, the error will be less than if the model is the same and the lane location differs.

Figure 5.1 shows a graph of the transformation model variation. The blue bar represents the test with all 260 transformation models tested for every frame. The red bar shows the test with 43 initial transformation models and 260 interpolation transformation models. The green bar shows the test with with only the 43 initial transformation models and no interpolation models. The graph charts the percentage of variation between the detected transformation model and the truth file with all 260 transformation models. The important aspect of this result is that using the 43 initial models with the 260 models produces a different model than the truth file about 22% of the time. Upon analysis of the resulting images, it seems that in

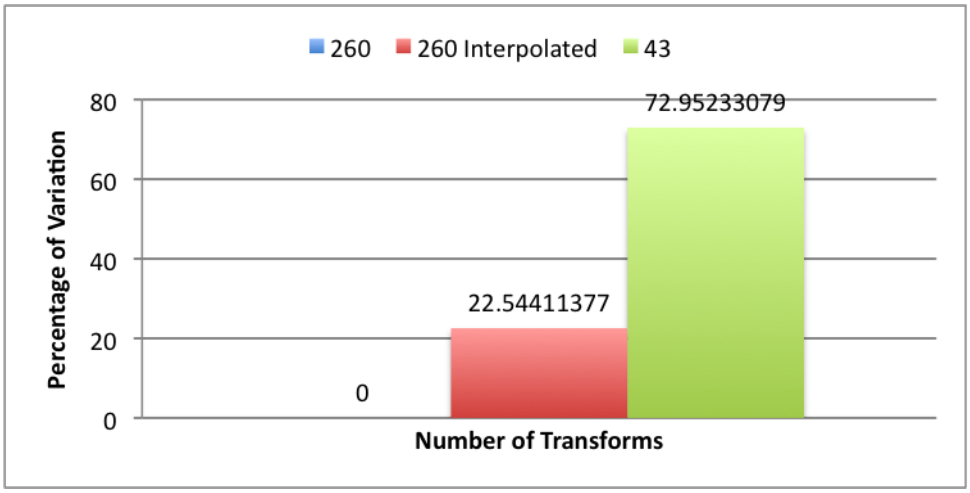


Figure 5.1: Interpolation Lane Model Variation

some cases, slight curvature models were substituted with slight skew distortion models. A more useful analysis is to determine the difference between the detected lane model boundaries.

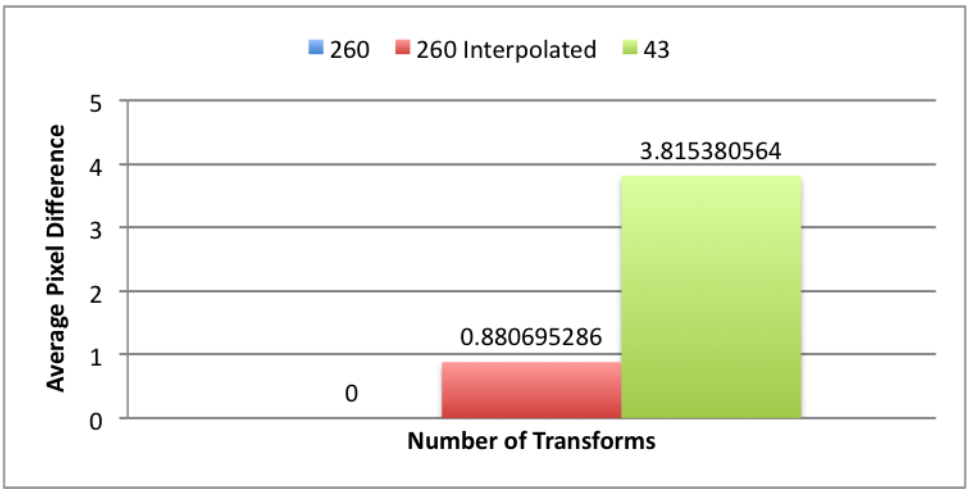


Figure 5.2: Interpolation Average Pixel Difference

Figure 5.2 shows the average pixel variation between image frames for the 43, 260 interpolated, and full 260 transformation models. The pixel difference is measured as the horizontal difference of both the left and right lane boundaries measured at the location clostest to the vehicle. From the results, it appears that using

only the 43 initial transforms produces an average of 8.5 pixels of difference per frame between the road models. This equates to roughly .25 meters in variation between the closest point of the detected road lane boundaries and the vehicle. Using the 260 models with interpolation produces only an average of .88 pixels of difference. This is roughly 0.05 meters in variation, which is far more acceptable. While this is still a larger error margin than checking all 260 transformation models for every frame, but when compared to the speed tradeoff, the error is acceptable.

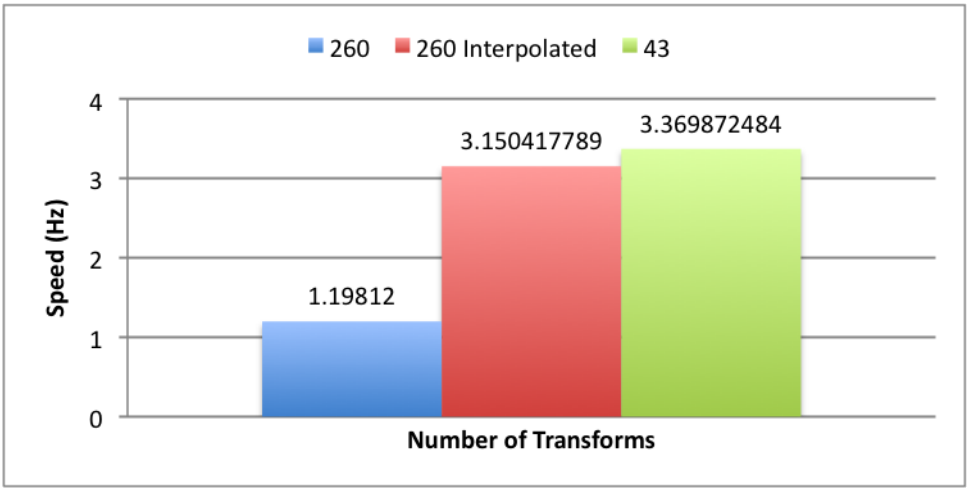


Figure 5.3: Interpolation Speed

Figure 5.3 shows the speed comparison in Hz between the three different test setups. The fastest version of the algorithm used only the 43 initial road transformation models. The second fastest was the 260 models with interpolation. The slowest was the entire set of 260 models. This was to be expected, but the promising conclusion is that using all 260 models with interpolation is only 6% slower then using only the 43 initial models. Using all 260 models was significantly slower than the other results, proving that the model interpolation was an important addition.

5.2 Mahalanobis Distance

The Mahalanobis distance parameter determines how close a color pixel must be to the learned color models to consider it as a road pixel. Increasing the distance allows more pixels to be considered road, and decreasing the distance allows fewer pixels to be considered as road. The benefit of a greater distance is that more intersecting roads may be detected. The advantage of a smaller distance is that less noise and fewer non-road pixels will be considered as road.

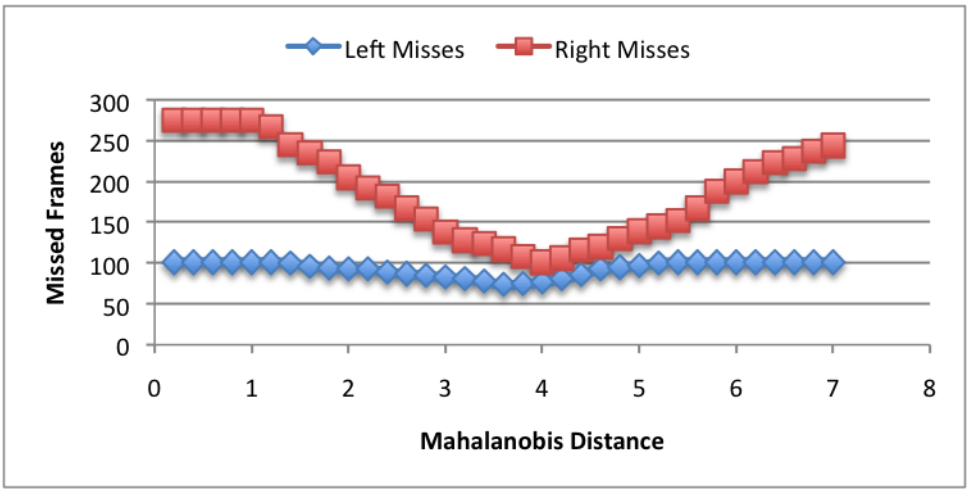


Figure 5.4: Mahalanobis Distance - Misses

Figure 5.4 shows the number of missed intersection frames for the left and right sides of the road plotted against increasing Mahalanobis distances. The results show very clearly that a distance of 4 has the fewest number of missed intersection frames.

Figure 5.5 shows the number of true positives for the left and right sides of the road against increasing Mahalanobis distances. Similar to Figure 5.4, a distance of 4 produces the best results.

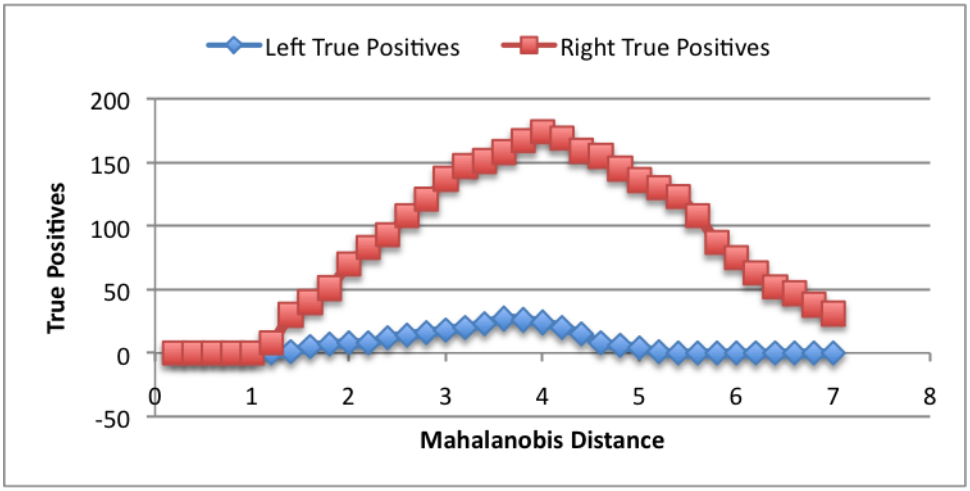


Figure 5.5: Mahalanobis Distance - True Positives

5.3 Clustering

There are two important parameters for the color extraction and analysis part of the algorithm. The first parameter is the number of learned color models the system maintains. The second is the number of training models it learns from clustering in each new frame. The more training models that are allowed, the longer the clustering has to run, therefore reducing the speed of the overall system. Adding more training models create more accurate models of the system and fewer models run at a faster speed.

Figure 5.6 shows that the number of learned models do not effect the speed of the system nearly as much as the number of training models.

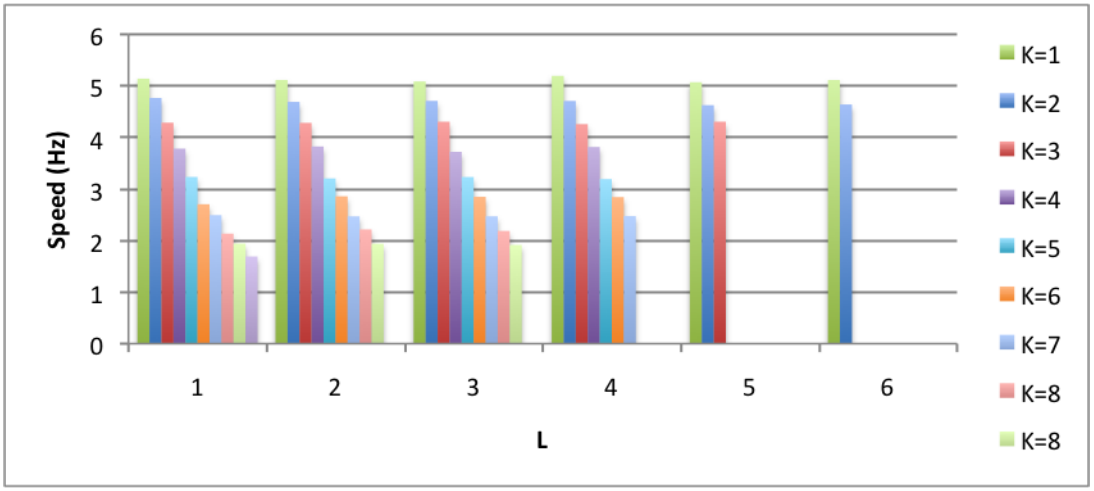


Figure 5.6: Clustering Parameters - Speed

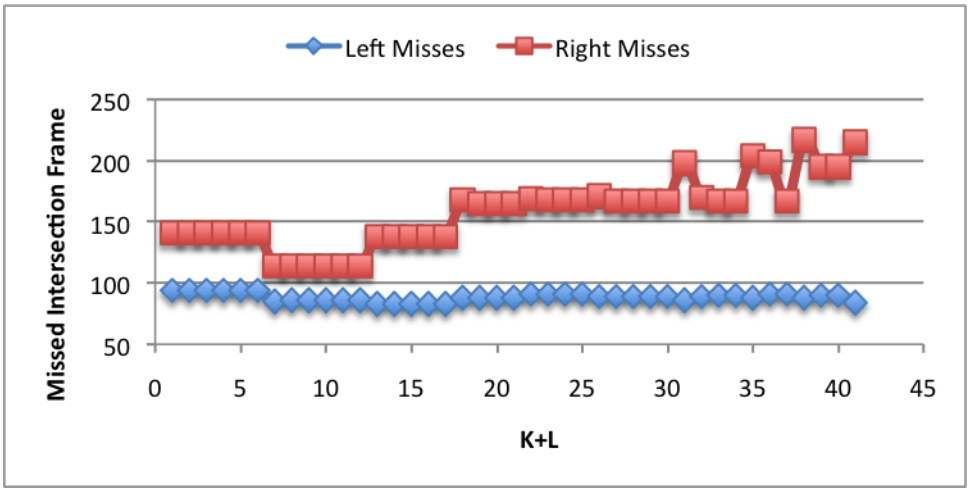


Figure 5.7: Clustering Parameters - Intersection Misses

Figure 5.7 shows that the fewest missed intersection frames occurred with either two or three training models. The number of learned models appear to have little effect on the number of missed intersection frames.

5.4 Failure Points

As with many computer vision algorithms, there are certain failure points to this algorithm. The inverse perspective mapping is a useful tool that when the car is aligned correctly on the road, but when going over a bump or hill that causes the defined horizon line to move above the road area, the mapping equation stops working correctly and the road becomes distorted. Some of the distortions encountered with this method were resolved using the additional transformation models, but there were still situations where it failed. The wider lens did not work as well as expected. Intuitively it would seem that a lens with a wider viewing angle would allow more intersection area to be captured in the image. Unfortunately, due to the low height of the test vehicle and the mounting position of the camera, the image contained a large amount of the vehicles hood and a low horizon. This allowed for fewer road pixels in the image. The wider lens may be beneficial with a different camera setup that was higher off the ground.

Chapter 6

Conclusion and Future Work

The road and intersection detection method proposed here performed reasonably. Since there are a very limited number of published intersection detection methods available, this method proved to be an interesting venture into using edges and color for identifying intersections. The method proved to be simple enough to run at a reasonable speed between 3Hz and 6Hz. It was able to find intersections on both sides of the road, but clearly favored the right side due to the fact that the camera is better positioned to capture the right side of the road. The use of edges for lane boundary detection proved to be very tolerant of broken lines, puddles, and other common urban road debris. The road transformation model interpolation improved upon the accuracy of the road detection while keeping the speed within the desired ranges. The interpolation also introduced only a small error into the system, which is an important factor.

There are many options for future work with this algorithm and in the area of road intersection detection. Since the road lane boundary extraction, color analysis, and intersection detection are all very separable, each area could be developed further independent of the others. One possibility is to use a more advanced lane boundary detection method. The inverse perspective mapping technique has a variety of complications that make it fail when the camera is not correctly aligned in the road. A road lane boundary detection algorithm that searches within the original image frame may produce better results. A road lane boundary detector

that doesn't rely on templates or road models may be more accurate. If the road lane detection was more precise and stable, the color extraction would be more accurate as well. The only problem with a better lane boundary detection method is that it may slow the algorithm down to an unreasonable rate.

Another possibility for future work is to implement some form of tracking on both the lane boundaries and detected intersections. Many computer vision algorithms implement Kalman filters to track any detected object. Tracking the road lane boundaries, outer lane boundaries, and intersections could create a more stable system.

Overall, the intersection detection algorithm was capable of detecting many different intersections on both sides of the road. The system met its goal of acceptable speed and would be beneficial to any map-based localization system.

Bibliography

[1] Christopher Baker. Cmu 1394 digital camera driver.

[2] M. Bertozzi and A. Broggi. GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE Transactions on Image Processing. In press.*, 1997.

[3] M. Bertozzi, A. Broggi, M. Cellario, A. Fascioli, P. Lombardi, and M. Porta. Artificial vision in road vehicles, 2002.

[4] M. Bertozzi, A. Broggi, and A. Fascioli. Stereo inverse perspective mapping: Theory and applications, 1998.

[5] Alberto Broggi, Massimo Bertozzi, and Alessandra Fascioli. Architectural issues on vision-based automatic vehicle guidance: The experience of the argo project. *Real-Time Imaging*, 6(4):313–324, 2000.

[6] R. Chapuis, R. Aufrere, and F. Chausse. Accurate road following and reconstruction by computer vision. *Intelligent Transportation Systems, IEEE Transactions on*, 3(4):261–270, 2002.

[7] Kuo-Yu Chiu and Sheng-Fuu Lin. Lane detection using color-based segmentation. pages 706–711, 2005.

[8] J. Crisman and Chuck Thorpe. Scarf: A color vision system that tracks roads and intersections. *IEEE Trans. on Robotics and Automation*, 9(1):49 – 58, February 1993.

[9] J. D. Crisman and C. E. Thorpe. Unscarf-a color vision system for the detection of unstructured roads. pages 2496–2501 vol.3, 1991.

[10] Hendrik Dahlkamp, Adrian Kaehler, David Stavens, Sebastian Thrun, and Gary R. Bradski. Self-supervised monocular road detection in desert terrain.

In Gaurav S. Sukhatme, Stefan Schaal, Wolfram Burgard, and Dieter Fox, editors, *Robotics: Science and Systems*. The MIT Press, 2006.

[11] Ernst D. Dickmanns. Vehicles capable of dynamic vision: a new breed of technical beings? *Artificial Intelligence*, 103(1-2):49 – 76, 1998. Autonomous vehicles;Autonomous road vehicle guidance;.

[12] C. Duchow. A marking-based, flexible approach to intersection detection. volume 3, pages 60–60, 2005.

[13] M. Foedisch and A. Takeuchi. Adaptive real-time road detection using neural networks. pages 167–172, 2004.

[14] M. Foedisch and A. Takeuchi. Adaptive road detection through continuous environment learning. pages 16–21, 2004.

[15] J. P. Gonzalez and U. Ozguner. Lane detection using histogram-based segmentation and decision trees. pages 346–351, 2000.

[16] Yinghua He, Hong Wang, and Bo Zhang. Color-based road detection in urban traffic scenes. *Intelligent Transportation Systems, IEEE Transactions on*, 5(4):309–318, 2004.

[17] Minghao Hu, Wenjie Yang, Mingwu Ren, and Jingyu Yang. A vision based road detection algorithm. volume 2, pages 846–850 vol.2, 2004.

[18] Kunsoo Huh, Jaehak Park, Daegun Hong, D. Cho, and Jahng Hyon Park. Vision-based lane detection for passenger cars: configuration aspects. *American Control Conference, 2004. Proceedings of the 2004*, 1:792–797 vol.1, 30 June-2 July 2004.

[19] Intel. Open source computer vision libraries.

[20] O. Silven J. Heikkila. A four-step camera calibration procedure with implicit image correction. *Proc. of IEEE Computer Vision and Pattern Recognition*, pages 1106–1112, 1997.

[21] Gang Yi Jiang, Tae Young Choi, Suk Kyo Hong, Jae Wook Bae, and Byung Suk Song. Lane and obstacle detection based on fast inverse perspective mapping algorithm. *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, 4:2969–2974 vol.4, 2000.

[22] T. M. Jochem, D. A. Pomerleau, and C. E. Thorpe. Vision-based neural network road and intersection detection and traversal. volume 3, pages 344–349 vol.3, 1995.

[23] Todd Jochem, Dean Pomerleau, and Chuck Thorpe. Vision based intersection navigation. In *Proceedings of the 1996 IEEE Symposium on Intelligent Vehicles*, pages 391 – 396, September 1996.

[24] Stefan Fuchs Cristian Paredes Klaus Strobl, Wolfgang Sepp and Klaus Arbter. Camera calibration toolbox for matlab.

[25] K. Kluge and S. Lakshmanan. A deformable-template approach to lane detection. pages 54–59, 1995.

[26] H.H.;Little J.J.;Bohrer S. Mallot, H.A.; Bulthoff. Inverse perspective mapping simpliefies optical flow computations and obstacle detection. In *Biological Cybernetics*, volume 64, pages 177–185, 1991.

[27] A.; Samad S.A.; Mustaffa M.M.; Majlis B.Y. Muad, A.M.; Hussain. Implementation of inverse perspective mapping algorithm for the development of an automatic lane tracking system. *TENCON 2004. 2004 IEEE Region 10 Conference*, A:207–210 Vol. 1, 21-24 Nov. 2004.

[28] O. Ramstrom and H. Christensen. A method for following unmarked roads. pages 650–655, 2005.

[29] C. Rasmussen. Road shape classification for detecting and negotiating intersections. pages 422–427, 2003.

[30] D. Schreiber, B. Alefs, and M. Clabian. Single camera lane detection and tracking. pages 302–307, 2005.

[31] Feldman G. Sobel, I. A 3x3 isotropic gradient operator for image processing. presented at a talk at the Stanford Artificial Project, 1968.

[32] Young Uk Yim and Se-Young Oh. Three-feature based automatic lane detection algorithm (tfalda) for autonomous driving. *Intelligent Transportation Systems, IEEE Transactions on*, 4(4):219–225, 2003.

[33] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. *Seventh International Conference on Computer Vision*, 1:666, 1999.