IRIS: Intelligent Roadway Image Segmentation using an Adaptive Region of Interest

Ryan Charles Brown

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

Master of Science

In

Mechanical Engineering

6/16/2014

Blacksburg, Virginia

<u>Keywords:</u>

Unmanned Ground Vehicles, Robotic Perception, Roadway Perception, Image Segmentation, Lane Modeling

IRIS:
Intelligent Roadway Image Segmentation using an Adaptive Region of Interest

Ryan Charles Brown

Abstract

The problem of roadway navigation and obstacle avoidance for unmanned ground vehicles has typically needed very expensive sensing to operate properly. To reduce the cost of sensing, it is proposed that an algorithm be developed that uses a single visual camera to image the roadway, determine where the lane of travel is in the image, and segment that lane. The algorithm would need to be as accurate as current lane finding algorithms as well as faster than a standard k-means segmentation across the entire image.

This algorithm, named IRIS, was developed and tested on several sets of roadway images. The algorithm was tested for its accuracy and speed, and was found to be better than 86% accurate across all data sets for an optimal choice of algorithm parameters. IRIS was also found to be faster than a k-means segmentation across the entire image. IRIS was found to be adequate for fulfilling the design goals for the algorithm. IRIS is a feasible system for lane identification and segmentation, but it is not currently a viable system. More work to increase the speed of the algorithm and the accuracy of lane detection and to extend the inherent lane model to more complex road types is needed. IRIS represents a significant step forward in the single camera roadway perception field.

Acknowledgments

I would like to take this opportunity to thank people who were critical to the success of my academic career, and also in my life, to this point:

My parents, Carol and Kevin Brown, whose support was invaluable to my success and whose training was what made me who I am today

My high school teachers, who were my parents

Dr. Al Wicks, who supported my research and gave me excellent advice and help, both in academics and life

Lauren Gibboney, my fiancée, who has been incredibly caring, supportive and helpful in this process

Finally, I would like to take this opportunity to thank God for the resources, opportunity and ability he has blessed me with to be able to undertake an effort like this.

Table of Contents

i List of Figures

ii List of Tables

iii List of Acronyms and Abbreviations

Acronym/Abbreviation	**Definition**
IRIS	Intelligent Roadway Image Segmentation
UGV	Unmanned Ground Vehicle
ALV	Autonomous Land Vehicle
LIDAR	Light Detection and Ranging
GUSS	Ground Unmanned Support Surrogate
LOE	Limited Objective Experiments
NIR	Near-Infrared
FPS	Frames Per Second
USB	Universal Serial Bus
CCD	Charge Coupled Device
CMOS	Complementary Metal–Oxide–Semiconductor
AURORA	Automotive Run-Off-Road Avoidance
YARF	Yet Another Road Follower
ALM	Active Line Model
TFALDA	Three-Feature Based Automatic Lane Detection Algorithm
CHEVP	Canny Hough Estimation of Vanishing Points
RANSAC	Random Sample Consensus
GPU	Graphics Processing Unit
B-Spline	Basis Spline

1 Background and Motivation

1.1 Thesis Overview

The following chapters of this thesis will discuss the relevant literature in the field of work, the theory and implantation of the Intelligent Roadway Image Segmentation (IRIS) algorithm, the experiments performed to characterize its operation, and the conclusions that can be draw from the results of the experiments.

Section 2 contains a review of the relevant literature in the fields of visual lane detection as well as image segmentation. The topics covered will inform the reader of the state-of-the-art and history of each field, as well as the previous research that has been incorporated into IRIS. The theory and method of operation of the IRIS algorithm is discussed in Section 3, as well as necessary information about the geometric formation of an image.

The theory of operation discussed in Section 3 is implemented in Section 4, with an emphasis on the programming structures and choices made when implementing the IRIS algorithm on a computer. Section 5 explains the data collection process as well as the two experiments that were performed to characterize the operation of IRIS. The results of these experiments are presented in Section 6 and the conclusions and future work that were gathered from the results of the experiments are presented in Section 7. References can be found in the bibliography (Section 0).

1.2 Thesis Summary

This thesis will describe the development process and testing of an intelligent roadway image segmentation algorithm. To provide a context for this work, a history of unmanned ground vehicles and the current state-of-the-art will be discussed below in Section 1.3. The typical unmanned ground vehicle, for roadway or off-road use, will use very expensive, active sensing for perceiving its environment [1], [2]. These sensors are not ideal due to their high cost and the fact that an active sensor inherently broadcasts its position in the sensing medium. Active sensors are the state-of-the-art in unmanned ground vehicles because passive sensors, such as cameras, are not currently viable to replace them. The viability of a camera as a sensor is not in the sensor itself, but the ability of the autonomous system to interpret the data into useful information. A discussion of the various technologies related to active and passive sensing will follow in Section 1.4. This discussion will include LIDAR (Light Detection and Ranging), camera basics and current technologies as well as imaging. After an image is acquired by a camera, it must be processed for the relevant information in the image. Image processing techniques for clustering and segmentation of an image will be discussed in Section 1.5 and 1.6.

The algorithm that was developed during this thesis project will support the use of a single visual camera to detect the presence and location of a roadway lane in an image. This information will then be used to create an intelligent region of interest around the lane, which will be segmented into potential objects. The algorithm, called IRIS for Intelligent Roadway Image Segmentation, must be at least as accurate as the current algorithms used in single camera lane detection efforts and must operate on an image faster than a k-means segmentation across the entire image.

To meet these criteria in a cost-effective approach, IRIS was developed in C++ using open source computer vision and machine learning libraries and tested on a diverse data set composed of real road images collected from a vehicle. IRIS was found to be similar accuracy to the current technology as well as faster than a k-means segmentation of the entire image. This finding is encouraging in the validation of the IRIS framework as a feasible method of unmanned lane navigation and obstacle avoidance.

1.3 Unmanned Ground Vehicles

Unmanned Ground Vehicles (UGVs) are a growing field of applied research, incorporating elements of robotic design, computer sensing and data processing. These vehicles are designed for various environments, incorporating multiple types of locomotion and sensing to achieve their intended function. Typical uses of UGVs are military and rescue operations, as well as newer advances in automobile technology. UGVs are designed to operate in either structured or unstructured environments. Structured environments are areas that are designed for human or vehicle interaction. Examples of this type of environment include roadways, buildings, cities, and other human-designed areas. Unstructured environments encompass all areas that do not fit this description. Natural scenes are the prime example of an unstructured environment.

1.3.1 History of UGVs

This section will give a brief history of the overall development of UGVs, looking at important events in the field as well as giving short descriptions of significant UGVs. The latter part of the section is focused on UGV research at Virginia Tech to provide more context for the motivation behind this thesis.

The development of UGVs starts in the late 1960's with the development of "Shakey", a wheeled platform with sensing capabilities and a radio link to an off-board computer [3]. The platform responded to high-level operator commands to navigate around a known environment. Starting in 1973 Stanford University's AI Lab constructed and worked with the "Stanford Cart"[4]. This system, primarily developed by Hans Moravec, was one of the first to successfully implement autonomous obstacle avoidance. The cart could move at most 1 meter every 10 minutes due to the slow processing time of the obstacle detection and avoidance algorithm [4]. The Carnegie Mellon University (CMU) Rover was Moravec's next project, building upon the experience gained from the Stanford Cart. It was a small, cylindrical body with an onboard camera, infrared proximity sensors and contact switches to sense its environment [5]. This system was constructed to allow upgrades and inclusion of a manipulator for the Rover to interact with its environment.

One of the next notable UGVs was DARPA Autonomous Land Vehicle (ALV) [6]. ALV was an eight wheeled vehicle platform retrofitted to include sensing and computing. This vehicle was capable of driving along a roadway using color visual data paired with laser range-finding at 14.5 km/h by 1988 [3]. It was capable of speeds of 0.6 km/h on off-road terrain. ALV is shown below in Figure 1.

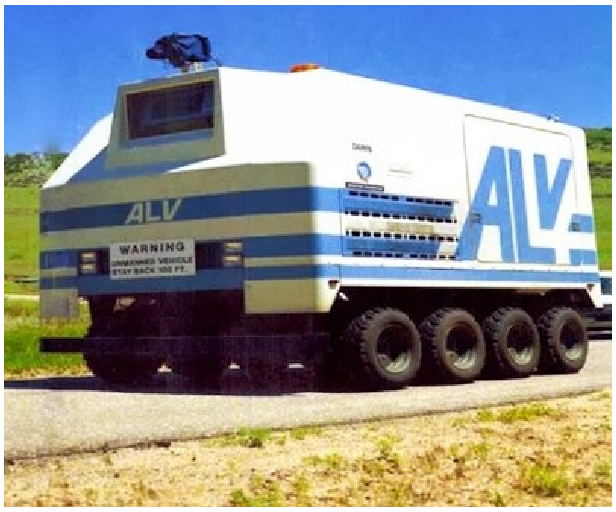


Figure 1: ALV (Autonomous Land Vehicle). https://plus.google.com/+DARPA/posts/XjuyGnuqjXQ [Fair Use], 2014

ALVINN (Autonomous Land Vehicle In a Neural Network) was the next progression from ALV [7]. ALVINN used TV cameras and a laser rangefinder to navigate roadways. The behaviors of the vehicle were determined by a neural network, a type of machine learning algorithm that "learned" how a human operator drove the vehicle and then used that information to associate the images of the road with movements of the steering wheel. It was successful in navigating on single-lane and two lane highways with approximately 3 minutes of training time [7].

The DARPA ALV project and its children were focused on constructing an autonomous framework for defense vehicles. In Europe, the PROMETHEUS (Program for a European Traffic of Highest Efficiency and Unprecedented Safety) project focused on civilian vehicles in more structured environments, such as highways and interstate-like roads [8]. 11 European countries contributed €749 million to universities and companies to create solutions towards safer vehicles and road systems with a focus on autonomy. PROMETHUES funding was responsible for many advances in obstacle avoidance, roadway imaging and high level behavioral planning [8][9][10].

More modern examples of UGVs include vehicles that competed in the DARPA Grand and Urban Challenges. The Mechatronics Lab at Virginia Tech formed team VictorTango and competed in the DARPA Urban Challenge with the vehicle "Odin"[2]. The vehicle was intended for use on roadways in a closed environment and was used to perform a series of tests to determine the best UGV design. Team VictorTango finished 3rd overall in the competition [11]. The sensor suite on Odin was a modern version of the sensors on ALV. LIDAR (Light Detection and Ranging) sensors were used along with cameras for the navigation. Odin is shown below in Figure 2.

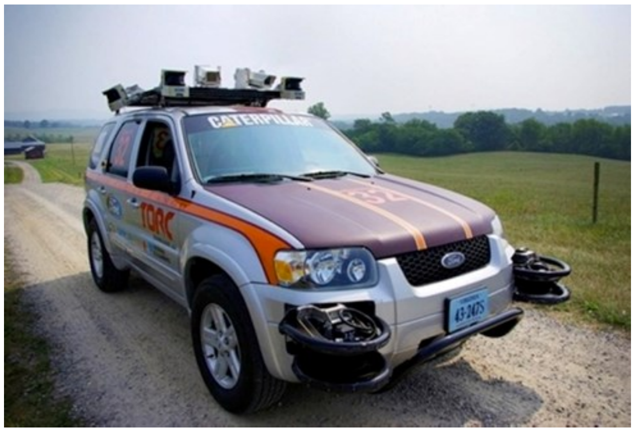


Figure 2: ODIN in DARPA Urban Challenge. http://www.mechatronic.me.vt.edu/About/aboutPage.html [Fair Use], 2014

1.3.2 GUSS

GUSS(Ground Unmanned Support Surrogate) is a vehicle designed to assist a squad of Marines in various operations by providing support and logistic capabilities [12]. GUSS is intended to integrate into a squad of dismounted Marines and assist with remote resupply, offloading some weight from the Marines and providing limited casualty evacuation [13]. GUSS was tested at various limited objective experiments (LOE) at different locations around the United States. GUSS represents a state of the art system for autonomously navigating an unstructured environment. LIDAR is the main sensing component on GUSS, allowing the UGV to navigate through its environment with little to no operator intervention.

1.4 Active vs. Passive Sensing

UGVs capable of reacting to their operating environment employ one or more sensors to gather information about the environment. These sensors typically fall into one of two categories: active sensors and passive sensors. Active sensors emit some form of energy and then collect the return signal from that energy to sense the environment [14]. LIDAR sensors are an example of a common active sensor used on UGVs. Passive sensors detect energy emitted naturally by the environment and translate that energy into data [14]. UGV sensing systems usually include cameras, which are typically structured as a passive sensor. Cameras themselves are passive sensors, but when combined with vehicle-mounted light sources, the sensing system becomes active. The differences between LIDAR and cameras are detailed below.

Active sensing systems generate energy, impart that energy to the target object or phenomena and measure the response of the target to the generated energy. This allows for a better characterization of the target because the input energy is known. An example of this is the radio wave generated by radar. A very specific radio wave at a known frequency with a known integrated power distribution as a function of solid angle to the antenna is generated and the reflected wave is detected, which allows the radar system to detect object of a particular size and in a particular area. The output of the radar system can be more easily processed and understood due to the complete knowledge of the emitted radio wave

Active sensor can be less than ideal in certain situations. If the object or phenomena to be sensed is sensitive to the type of excitation provided by the active sensor, the behavior of the phenomena could be changed or the integrity of the object being sensed could be compromised. Using a high power radar to image a human at close range could be harmful and compromise the integrity of the human being imaged. Also, if the sensor application requires a degree of stealth, active sensing may be a poor choice. Actively imparting energy to the surroundings can potentially be sensed by another similar sensor, causing the sensor operator or computer detect to the presence of the human and, potentially, his or her location. Passive sensors address some of these drawbacks.

Passive sensors sense the ambient energy in the environment. There is no generated input to the environment from the sensor to excite a response by that sensor or any others in the area. This allows for the sensor to be stealthy. It is not potentially as reliable or precise as an active sensor, due to the lack of known input, but there are examples of well-known successful passive sensors that are very accurate and have high reliability.

1.4.1 LIDAR

GUSS is an example of a very capable UGV system, able to navigate through many types of unstructured environments and perform a set of functions to assist the Marine squad. However, it suffers from a major drawback based on the choice of sensing devices. As mentioned in Section 1.3.2, GUSS uses a LIDAR sensing system. LIDAR is an active sensor, supplying a source of light and measuring the time of flight of the reflections [15]. This allows for sensing independent of lighting conditions, but also compromises the stealth of the vehicle. LIDAR typically uses a near-infrared (NIR) laser array to project light onto the environment [16]. The typical wavelengths are 930 nanometers and 1550 nanometers. These wavelengths are typical of an eye-safe laser. While the these wavelengths are suitable for the system to be undetected in the visible spectrum by humans or filtered visible spectrum cameras, as well as eye-safe, most night vision systems operate in the NIR wavelength [17]. This allows the vehicle to be easily detected by any system capable of sensing the NIR wavelength band, which can include unfiltered silicon based camera sensors, such as CCDs, as well as thermal imaging systems. This compromises the ability of the GUSS-embedded dismounted Marine squad to perform any mission requiring stealth.

LIDAR systems are varied in their performance and cost. The least expensive systems can be constructed for under \$50, but have a limited accuracy and range. The Revo LDS costs \$30 to build, but only measures between 0.2 and 6 meters from the sensor in a single horizontal plane, and can only resolve objects with a 3 cm accuracy [18]. It is used in the Neato XV series of robotic vacuums. The opposite end of the cost spectrum is the Velodyne HDL-64E, which can sense objects out to 120 meters away with a 2.5 cm accuracy in 64 planes from -24.5 degrees below horizontal to +2 degrees above. Velodyne does not release information about the cost of the sensor, but according to Hizook.com, who performed a Velodyne-sanctioned disassembly of the device, the approximate cost is \$75,000 [19]. The Velodyne sensor is a typical choice for UGVs that are intended to operate in unstructured outdoor environments [1]. Thus, the expected cost for a state-of-the-art LIDAR sensing system on a UGV for outdoor unstructured environments is well above $70,000 dollars.

1.4.2 Cameras

Cameras are passive sensors, which collect reflected and radiated energy in a particular range of the electromagnetic spectrum and convert that energy into information. The information collected is dependent upon the amount and type of energy present in the scene, the optical system design, the sensitivity of the camera imager and its spectral range. Because cameras do not require an active light source to collect information, a camera-based autonomous system is advantageous for applications where stealth is a high priority.

Cameras range widely in their cost, specifications and size, but almost all are inherently passive. A simple webcam, such as the Microsoft HD-3000 costs $40 and provides up to 30 frames per second (fps) of color images over a Universal Serial Bus (USB) cable [20]. This is a very low cost camera designed for web video communication and not designed to capture a highly accurate representation of a scene. More expensive machine vision cameras, such as the JAI AD-080GE can sell for $2500 or more, depending on the configuration and types of optical spectrum that is desired [21]. There is not a “typical” camera in terms of a UGV, apart from one operating in the visible spectrum. Rather, UGVs will choose cameras that have desired properties within their particular budget [3], [11], [22], [23].

1.4.2.1 Camera Models

Cameras can be mathematically modeled in order to understand the translation from 3D information in the scene to 2D information in the image plane. There are two basic categories of models that are used: global models and local models [24]. Global models describe the whole camera with a single set of parameters, while local models have different sets of parameters for different locations in the image plane. Local models are a subset of a more general class called discrete models, where every pixel in the image plane has an individual set of parameters describing it. Global models will be further discussed below.

The most basic general camera model is the pinhole model. This model will be discussed in detail in the Method and Theory chapter to form an understanding of how 3D information can be inferred from a 2D scene. Briefly, the pinhole camera, or perspective projection model, assumes that all rays pass through a single point, the optical center of the camera, and there is a linear relationship between the point on the image plane and the direction of the ray that created it [25]. A “calibration matrix”, typically represented by K, expresses the linear relationship between the ray and corresponding point in the image plane [25]. An example of the pinhole camera model is shown below in Figure 3 [18].

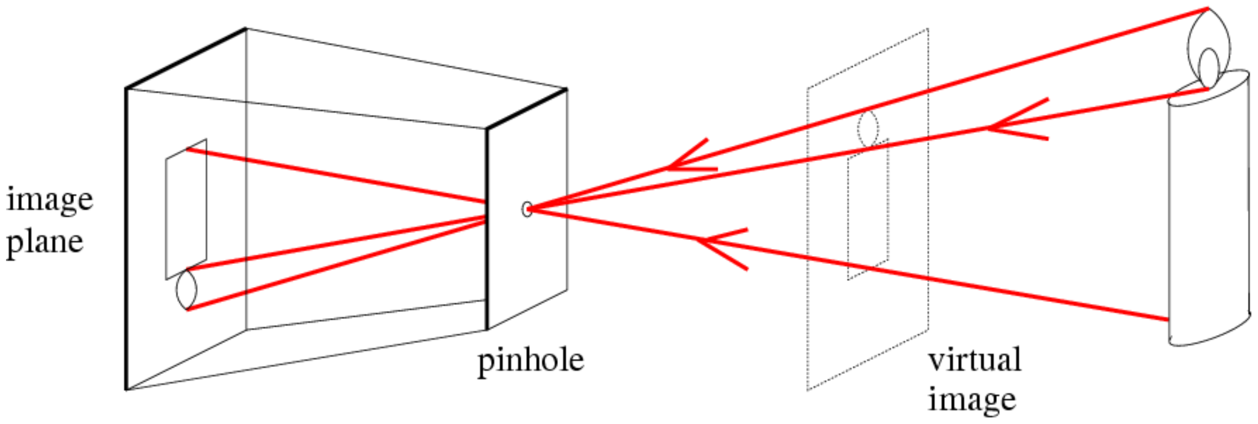


Figure 3: Pinhole Camera Model. Ponce J, Forsyth D, Willow E. Computer vision: a modern approach. *Boston: Prentice Hall; 2011. Available at: http://www.inria.fr/centre/paris-rocquencourt/actualites/computer-vision-a-modern-approach. Accessed February 3, 2014. [Fair Use], 2014*

Other global camera models attempt to improve on the pinhole model so that a camera can be more accurately modeled. The affine model states that the optical camera center is at infinity. This gives a good approximation of real cameras when the scene is far from the camera and the scene depth varies much less than the camera focal length [24]. Other models include polynomial and trigonometric models, where the camera parameters are mapped to functions that attempt to include information about distortion in camera lenses and other sources of error [24]. The intrinsic error present in the camera can be modeled using the more complex models, but at the cost of more involved transformation between real-world coordinates and image coordinates. For most computer vision applications, a standard pinhole camera model is used [24]. Due to the structure of IRIS and the assumptions involved, the camera model is estimated as a part of the iterative process and does not need to be explicitly declared

1.4.3 Digital Camera Sensors

Digital cameras contain an imaging sensor array that reacts to the incoming light and converts the radiation to electrical information. For a certain period of time, radiation is allowed to interact with the sensor array. After the time period has passed, the data gathered is integrated over the time period and output as a set of values that can be interpreted into an image [26]. There are two main types of imaging sensors: Charge Coupled Devices (CCD) and Complementary Metal–Oxide–semiconductor (CMOS) sensors. Each will be discussed further below.

1.4.3.1 CCD Sensors

CCD’s are a dedicated imaging sensor manufactured to exhibit a high efficiency and low output image noise [27]. They were first developed in the early 1970’s at Bell Labs and were recognized for their high resolution, quantum efficiency, small size and low power requirements [28]. The operation of a CCD is often explained by the analogy of measuring rainfall using an array of equally sized buckets in a field. After the rain is finished, the buckets are emptied one at a time to determine how much water was in each bucket, and the results are recorded [29]. The CCD actually measures the spatial distribution of radiation across the sensor’s surface. The sensor is exposed to light for a certain amount of time, usually with a physical shutter. The interaction between the photon and silicon free an electron from the silicon crystal lattice. These electrons are collected and the total charge at the end of the time period is collected and sent to be digitized into the output image.

CCD sensors can be either large or small, are accurate, have a wide spectral response, exhibit low noise and are durable. [29]. Figure 4 shows a close image of a typical CCD sensor. All of these characteristics are why they were the dominant sensor in machine vision and consumer electronics until the CMOS resurgence in the 1990's [30]. They do exhibit some negative behaviors, however, particularly the effect known as "blooming". This occurs when a collection site for the photon-generated electrons in a particular area of the sensor becomes saturated, but more electrons are still being freed in that area. These electrons find their way into neighboring collection sites and cause an artificial increase in the measured charge in that site. This translates into the intensity of the output image and creates the image artifact called bleeding.

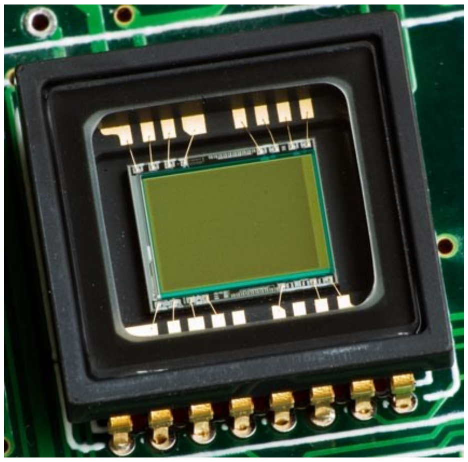


Figure 4: CCD sensor. http://commons.wikimedia.org/wiki/File:CCD_in_camera.jpg Permission from Wikimedia Commons..

1.4.3.2 CMOS Sensors

CMOS sensors were originally developed in the 1960's as a potential imaging sensor. Several advances were made in the field in terms of development and characterization, but a problem of fixed-pattern noise was unable to be solved and focus switched to the CCD in the 1970's. Little work in the CMOS field was done until the 1990's. The next generation CMOS sensor began to be seriously developed in the early 1990's as a lower cost, lower power and smaller alternative to the CCD.[30] CMOS sensors are fabricated using the same manufacturing technology as standard silicon microchip [30]. One specific advantage of this manufacturing technique is the capability to include digitization, amplification and other support circuitry on the same silicon chip as the imager, reducing cost and package size. This allows for highly integrated imaging devices for use in embedded laptop cameras, automotive stereo vision, micro-mobile robotics and other applications [31],[32],[33].

CMOS sensors can be constructed in a passive or active sense. Passive pixel CMOS sensors simply integrate charge over a photodiode pixel and output the total charge at a certain rate [30]. Active CMOS sensors add an amplifier into the pixel. The amplifier is only active during the readout, so it still exhibits the low power qualities of the passive pixel, but requires additional space to include the amplifier stage. The current state of the art in CMOS technology includes an amplifier stage as well as an analog-to-digital conversion (ADC) stage [34]. Figure 5 shows a typical CMOS sensor.

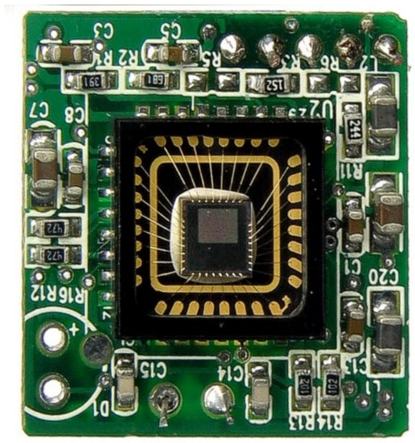


Figure 5: CMOS imaging sensor. http://commons.wikimedia.org/wiki/File:Sweex_USB_webcam_PCB_with_without_lens_close_up.jpg Permission from Wikimedia Commons

1.4.4 Color Imaging

The two major sensor types covered above, CCD and CMOS, both have two common failings: They can only convert light into intensity and, when constructed from silicon, are responsive to light outside the visible spectrum. The first failing means that there is only one value for the image per location. Images formed in this way can only be grayscale, representative of incident light intensity. The second failing means that intensities of images might be different than a human eye would detect.

This second problem can be solved by introducing a filter over the imager that removes radiation with a wavelength higher than visible light, or approximately 700-750 nm. Silicon imagers are most sensitive to incident radiation at approximately 800 nm wavelength with response up to 1000 nm wavelength [35]. An IR cut filter, as it is commonly known, absorbs radiation with a wavelength higher than 700-750 nm. This removes the “extra” information present on the sensor and more closely approximates the scene as a human eye would interpret it.

The first problem was solved in 1976 with the invention of the Bayer filter, by Bryce Bayer of Eastman Kodak [36]. It placed an array of colored elements that only allow specific light frequencies to pass. The luminance element in the Bayer filter is placed in the array such that in each direction, it will occur every other element, with the chrominance elements interlaid in an alternating pattern in between them [36]. This is necessary as the human eye is more sensitive to the luminance element of light, and, thus, humans expect a higher predominance of that element when interpreting the data presented in an image [36].

This arrangement typically is implemented in a red-green-blue fashion, with either a bare pixel or a green filter as the luminance element and red and blue filters as the chrominance elements. The typical Bayer filter arrangement is shown below in Figure 6 [37]. This invention allowed the color of the images in the scene to be recreated digitally, though at a loss of image resolution as four pixels are used to interpolate one pixel's worth of information in the reconstructed image.

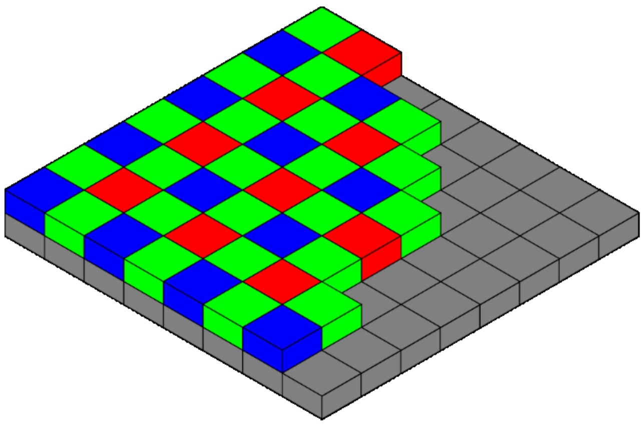


Figure 6: Typical Bayer Filter Arrangement. C. M. L. Burnett, "Bayer pattern on sensor.svg," Wikimedia Commons*, 2006. [Online]. Available: http://commons.wikimedia.org/wiki/File:Bayer_pattern_on_sensor.svg. Used under Creative Commons license, 2014*

1.5 Machine Learning

Machine learning is a growing area of research in computing and robotic systems. It was defined by Ethem Alpaydin in his book "Introduction to Machine Learning" as "programming a computer to optimize a performance criterion using example data or past experience"[38]. The key element in the definition is the optimization of performance. A machine learning algorithm can use information to change the way it interacts with data. As more data becomes available, the parameters of the algorithm change causing the behavior to shift towards more desirable performance.

There are two main groups of machine learning algorithms: supervised and unsupervised. Supervised algorithms are trained to classify data based on feedback from a "supervisor"[39]. The training data is typically labeled with the correct classification and provides feedback to the algorithm about the quality of the predictor. The training of a supervised machine learning algorithm is dependent on the quality and quantity of training dataset examples. Too few examples can lead to a poor predictor of class, while too many can lead to an algorithm that does not follow robust general principles and instead classifies its training data on very specialized cases.

An example of a supervised machine learning algorithm is a neural network. Neural networks are constructed of nodes that have multiple inputs and one output [40]. Every node uses the same function, typically referred to as an activation function, to output a value based on the inputs. The system is trained by passing a known example through the network and determining the error between the known correct answer and the output. The network is then updated to minimize that error and the process is repeated with another training example. ALVINN, the self-driving vehicle discussed in Section 1.3.1, used a neural network to decide what direction to drive [7]. The training examples were images taken of the roadway, the prediction was the output by the network with respect to those images, and the known answer was the steering angle that a human driver chooses. The system trained on these examples for 2 or 3 minutes and then was capable of driving itself on roadways similar to the training road, but not limited to the training road itself [7].

Unsupervised machine learning algorithms are typically focused on finding patterns in data that exist above the noise floor of the data set [41]. Examples of unsupervised learning algorithms are clustering and dimension reduction. Training is not a concern in a typical unsupervised learning algorithm, as the process does not require feedback, either positive or negative, from the user. K-means clustering is an example of an unsupervised learning algorithm commonly used in image processing for image segmentation [42]. K-means uses a number of centroids in a data set to create spheres of data. The clusters are randomly initialized at points in the data set, and each data point is assigned by distance to a cluster centroid. The centroids are then moved to the centroid of all the points assigned to them and the data points are assigned to clusters again. This process repeats until the movement of the centroids falls below some threshold, or a certain number of iterations.

In the context of UGVs, these two types of algorithms are used for different tasks. Supervised learning algorithms are used in classification of objects from LIDAR and image data [43]. Features are inferred from the LIDAR and image data, such as shape, intensity of response in a color band, or a combined higher level feature generated from the raw data [44]. Different types of supervised algorithms are used either as a stand-alone algorithm or together in a topology known as “boosting” [43]. Unsupervised learning tools are used to create inferences within sets of data, or to learn parameters of some model. One example is an operator inference scheme using Hidden Markov Models (HMMs). An HMM can be used to predict the probability of an operator performing an action given some previous states and sensor inputs [45]. Learning the parameters of the HMM from as set of data was performed using unsupervised machine learning [45].

Machine learning is a widely used tool in many fields of study, from biology and bioinformatics to economics to world modeling for gaming environments to homeland security and energy applications [46]. As computer processors grow in power and speed, machine learning algorithms can be faster and more powerful as well.

1.6 Image Segmentation

Images are, in their most basic digital form, a series of numbers that correspond to an intensity of light integrated over a defined spectral region within a specified surface area, usually called a pixel [26]. There are typically three values associated with each pixel for a color image and one value for a monochrome image. These values are defined in terms of their size in bits. This is very different from the human perception of an image. Humans perceive objects in an image, while a computer only perceives intensity of one or more color planes. This creates a need for digital image segmentation. Image segmentation is the process of subdividing an image into uniform and homogenous regions with respect to one or more features, while minimizing ragged edges and holes within regions [47]. Image segmentation, in its most basic form, is a clustering problem on the spatial domain of an image.

Image segmentation was investigated as early as 1970, by Brice and Fennema [48]. They discussed a technique for segmenting an image into atomic regions of uniform intensity, and then used heuristics to join similar regions together. The result was an algorithm that could identify distinct objects in an image. As computing technology evolved, new techniques and algorithms were created. From 1995 to 2000, the number of papers in English concerning image segmentation was nearly constant at approximately 250, but from 2000 to 2005 the number doubled to approximately 500 papers [49]. This correlates with the rising need for image segmentation in the broader field of image processing.

According to Zuva et al. there are three basic types of image segmentation: threshold-based segmentation, edge-based segmentation and region-based segmentation [50]. Threshold-based segmentation is the simplest of the methods. A function is performed on each pixel of the image and the output value is compared with a set threshold value. The pixel is then assigned a class based on what side of the threshold its value falls. This results in an output image containing two segments, one that passes the threshold and one that does not. This method struggles to identify more than two classes of regions in an image. It still remains useful for simple segmentation tasks in structured environments, but in terms of UGVs is of questionable use.

Edge-based segmentation attempts to subdivide an image based on the existence of edges in the image. An edge in an image can be defined as a discontinuity in a number of pixels in a collinear direction. These edges define objects in the image and can be found using several methods, including the Canny, Sobel and Laplacian of Gaussian algorithms [51]. This method can fail if the image contains objects with graded boundaries, or in very noisy images. Figure shows an example of edge based segmentation using active contours, which are discussed later in Section 3.4.1 [52].

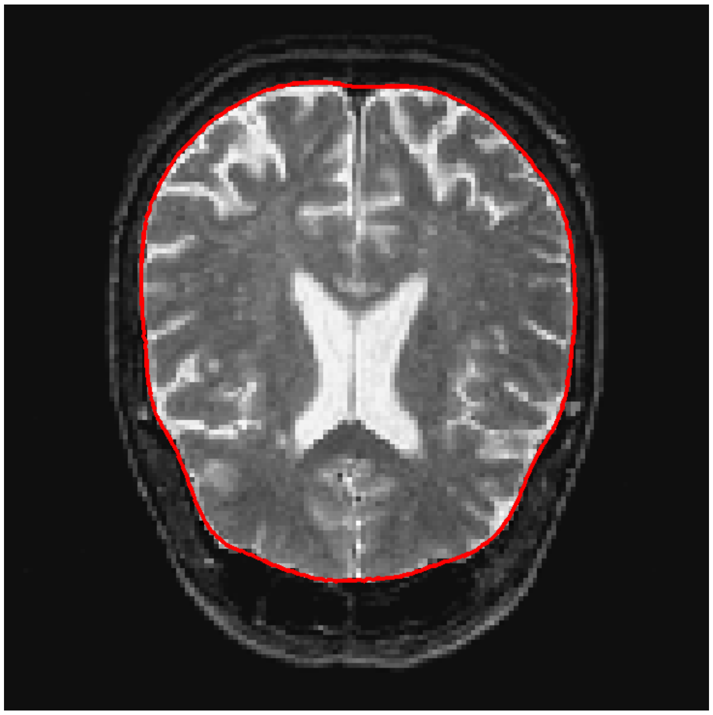


Figure 7: Edge Based Segmentation Example. G. Peyré, "Lecture 1 - Active Contours and Level Sets," ceremade.dauphine.fr*, 2006. [Online]. Available: https://www.ceremade.dauphine.fr/~peyre/teaching/manifold/tp1.html. [Accessed: 01-May-2014]. [Fair Use], 2014*

Region-based segmentation attempts to partition an image into similar regions through a criteria or series of criterion based on their similarity or homogeneity. Some examples of the measures used are texture, color, and intensity. Some conditions are typically enforced on the regions: the mean criteria values are different for each region and the regions are typically characterized by a Gaussian distribution of the pixel criteria within them. Some examples of region-based segmentation algorithms are histogram thresholding, graph-based methods, and surface-based segmentation [53]. This method can experience problems with segments if the criteria are too loose or too strict, which can result, respectively, with segments that encompass more than one object or multiple segments inside one object. Figure 8 shows an example of segmentation by texture, where the colors in the image are similar, but the texture separates the object from the background [54].

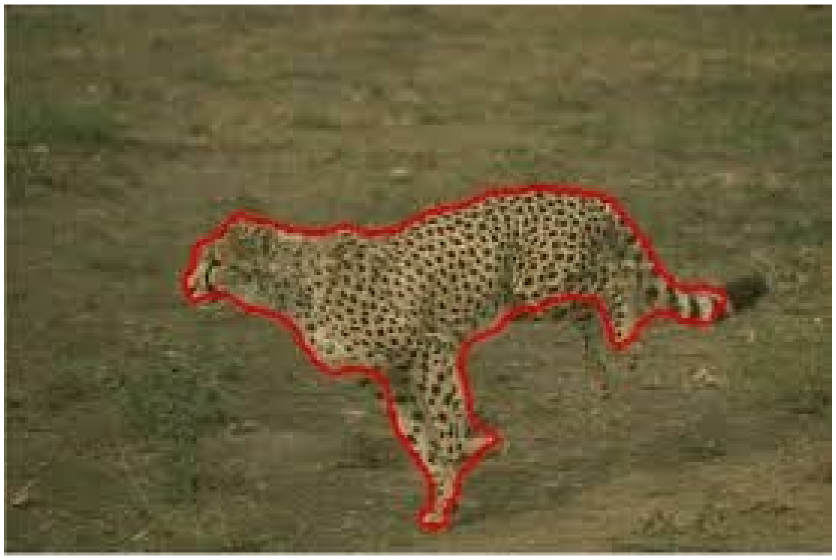


Figure 8: Texture Based Image Segmentation. Y. Kee, M. Souiai, D. Cremers, and J. Kim, "Sequential Convex Relaxation for Mutual-Information-Based Unsupervised Figure-Ground Segmentation," IEEE Conf. onf Comput. Vis. Pattern Recognit.*, 2014. [Fair Use], 2014*

For UGV navigation, image segmentation can be used to identify the presence of objects, and begin the process of identifying them. Vehicle detection, as well as lane line detection using image segmentation has been studied [55],[56]. In general, segmentation of an image will result in some separation of objects for later classification using different techniques, possibly the machine learning techniques discussed in Section 1.5.

To accomplish the goal stated in Section 1.2 of creating an algorithm that will be able to use visual data to reconstruct a road model and perform segmentation using that model, the background information relevant to that effort was discussed. A brief history of UGVs was presented to create context for the state of the research field. Active and passive sensors were discussed to provide reasoning for why a UGV might use passive sensing instead of active. A discussion of cameras models and sensor types was necessary to understand the image geometry and electronic components present in IRIS. Machine learning and image segmentation was discussed to inform the readers about the state of the art and the decisions made in the implementation of IRIS.

2 Literature Review

There are two general topics that will be discussed in this thesis, detecting a lane in a roadway and segmenting objects within images. Both of these topics are necessary for the creation of an intelligent roadway segmentation algorithm, hereafter referred to as IRIS. First, the history and state-of-the-art in both fields will be shown through a review of the relevant literature. Specific algorithms that will be integrated into IRIS will be detailed in the theory section immediately following.

2.1 Lane Detection in Images

Detecting lanes in a roadway has been studied for the past 25 years, most notably beginning with ALV in 1985 [6]. ALV used a laser rangefinder combined with a color camera to identify the roadway in an image. This information was used to perform a simple road following task. ALV was later expanded into ALVINN, a machine learning-based lane follower [7]. ALVINN was based around an active neural network that was fed data from a visual camera and output a prediction of the appropriate steering angle for the vehicle. The output was compared to the human operator driving the vehicle and the neural network was tuned to properly predict a steering angle based on the color image. This formed a relationship between the image of the roadway and the steering angle that was "learned" instead of pre-programmed.

Lane detection algorithms were becoming more developed after the introduction of ALV. Kenue at General Motors Research Laboratories applied a Hough transform algorithm, a region-tracing algorithm, and a vehicle-tracking algorithm on a set of roadway images to test their performance [57]. Lane boundaries were able to be determined with the Hough transform and region tracing algorithms, but the processing time for the algorithms was up to 30 seconds per frame [57]. One of the key research areas in lane detection is speed and efficiency of an algorithm. There has not been much research performed in the area of relating computing speed, sensor update rates and the resulting vehicle speed due to the varying nature of computers used, algorithms used and data transfer protocols. For IRIS to operate in "real-time", where operations on the current camera frame are completed before another frame is acquired, the speed of the computations in IRIS must be below the camera frame rate. Previous work has been done with vehicles operating at approximately 10 mph with a sensor update rate of 10 Hz, thus an eventual goal for IRIS is to operate at these speeds [2].

Lane detection was later applied in a roadway departure warning system called Automotive Run-Off-Road Avoidance (AURORA) in 1995 [58]. This system, developed at Carnegie Mellon, imaged a section of road adjacent to the vehicle using a visual camera and measured the distance between the lane marker or side of road and the vehicle body. This information could be used to detect a car drifting off the road and provide corrective measures or alert the driver. Also in 1995, a vision-based road following system called YARF (Yet Another Road Follower) was developed [59]. YARF extracted road features from a color image to provide the information needed to drive a vehicle. YARF defined the model of the road as a series of concentric circles in the ground plane. The YARF system assumed that the yellow lane markers would be hue invariant for a broad range of daytime weather conditions. Using that information, a lane model could be constructed based on the concentric circle assumption. YARF was implemented on the NAVLAB I and NAVLAB II vehicles at Carnegie Mellon University. YARF was able to identify a roadway using local features, but needed sophisticated statistical algorithms to handle intermittent road markings and variations in the roadway [60].

In 1996, Kang et al. developed a method to create a lane model using an active line model (ALM), more commonly called "line snakes" [60]. Line snakes are splines that attempt to minimize energy supplied by some external force while maintaining a shape constraint [61]. In the case of lane detection, the edges identified in the image as lanes provide the energy to the snakes to correct and update their model. The shape constraint is the assumption that the roadway exists in a plane. The method developed by Kang separated the problem of lane detection from lane tracking. Lane detection was accomplished with an initial estimation by a Hough transform to find the vanishing point of the roadway. The initial estimation was used to create a line snake, which would accomplish the lane tracking, by minimizing a pre-defined energy function that compared the model and the edges in the image. The typical energy function in line snakes is discussed later in Section 3.4.

Lane detection moved to more sophisticated techniques by 2003 with the Three-Feature Based Automatic Lane Detection Algorithm (TFALDA), which used the starting position of a lane, the orientation of the lane, and the grayscale intensity image to identify a three-dimensional lane model [62]. In 2004, McCall et al. proposed a method of lane detection using steerable filters [63]. Steerable filters allowed for a finite set of rotation angles to form a basis set for all angles of the steerable filter. This method reduces computation while still allowing any given angle of the filter to be observed. Wang et al. proposed an improved version of the 1996 Kang line snake method using B-snakes, a line snake created from a B-spline [23]. B-splines are spline features defined by control points, with the complexity of the curve related to the number and position of the control points [64]. Using control points to adjust the spline reduces the number of computations that must be performed to update the model. Wang was able to perform the B-Snake tracking computation at 2 frames/second.

The initial lane estimate is performed with Canny/Hough Estimate of Vanishing Points (CHEVP) [23]. In CHEVP, images are divided into horizontal strips, reducing in thickness as they reach the top of the image. Hough Transforms are used to identify the vanishing point in each of the image segments and the spline connecting each vanishing point is estimated to represent the curvature of the roadway. The B-snake is initialized with the CHEVP estimate, and then minimizes the image edge energy to track the lane. In 2006, Heij et al. implemented the 2004 Wang B-snake method and published their findings [65]. This method was shown to be robust against most image noise as well as operate for many different kinds of roadways. However, the method is sensitive to its parameters and the initialization time is long. The authors did not implement the lane tracking portion, only the lane detection algorithm, so the computation time should be reduced once the tracker is initialized.

Kim presented in 2008 a robust lane detection method that used a random sample consensus (RANSAC) algorithm to generate hypotheses for the direction and presence of lane markers [66]. Hypotheses are then compared with a probabilistic framework to determine the best hypothesis. The algorithm was able to run at 60 ms/frame, compared to the algorithm developed by Wang et al. which required 4 s/frame detection time and 2 s/frame tracking time. In the test set, 8.3% of the frames were incorrectly detected, either a lane was not detecting or a lane was detected that did not exist. Kim determined that the major source of the error was low resolution image data, as all the images were 320 x 240 pixels. He hypothesized that a higher resolution camera would mitigate these errors [66]. Kim implemented a version of the CHEVP algorithm to test his method, but it only achieved 52% accuracy. Kim did not implement the B-Snake tracking portion of the Wang algorithm. Based on the previous implementation of CHEVP in Heij et al. which achieved 86% accuracy, Kim's implementation of CHEVP is not correct, and his comparison of accuracy is not accurate.

Bar Hillel et al. authored a survey of modern lane detection techniques, with a goal of identifying the underlying system structure of all road and lane detectors, and discussed future steps in this field [67]. They hypothesized that nearly all road detection or lane detection algorithms followed the same steps in this order: image preprocessing, feature extraction, model fitting, and time integration and image-to-world correspondence. This structure was based on the idea that lane markers are designed to be visually identifiable. Therefore, the most relevant data would come from a visual sensor. The authors also suggested that, while visual data is the most relevant for lane detection, LIDAR sensing is very useful for road segmentation and Global Positioning System/Inertial Measurement Unit (GPS/IMU) data is useful to bound lane tracking algorithms to real world motion possibilities. Machine learning was discussed as a future step that could yield significant improvements in accuracy and speed. Finally, a public benchmark test was discussed as a necessity for cross-publication comparison and reduction of evaluation costs.

Similar work to visual lane detection is visual runway detection for autonomous landing of aircraft. While differences exist in the difficulties encountered between lane detection and runway detection, the underlying problem is similar. Trisiripisal, Parks and Abbott at Virginia Tech and Fleming at NASA Langley experimented with stereo vision based runway localization for autonomous landings [68]. Their findings showed that even with stereo vision data, runway detection algorithms are needed. The types of features and extraction algorithms matches closely with the lane detection findings. The speed of the vehicles and acceptable error in an airborne setting is different than in a UGV, thus the focus of the algorithms is on speed and accuracy [68]. Also, runways do not deform like lanes. A runway is expected to be straight and of a certain width. This allows stricter assumptions about the underlying runway model without reducing the success of the detector [69].

This literature review shows the history and progress in the field of lane detection. Lane detection an essential step in the development of the IRIS algorithm later in this thesis. CHEVP from Wang et al. will be used to find the lane, as it is a proven system that has solid analysis showing its value. A more complete explanation of how CHEVP works can be found in Section 3.3. The B-Snake tracking method proposed by Wang et al. will also be used to track the lane frame-to-frame. A more complete explanation of the B-Snake tracking method can be found in Section 3.4.

2.2 Image Segmentation

Image segmentation is the task of identifying areas of similar properties in an image and differentiating those areas from other such areas in the image. Zhang describes image segmentation as "…the process that subdivides an image into its constituent parts and extracts those parts of interest (objects)" [49]. Historically, the first set of techniques developed were edge detectors, with the first notable such technique the Roberts operator. The Roberts operator takes the square root of the intensity of each pixel, and creates a difference operator between it and its right and lower neighbors [70]. The next major improvement in edge detection was created by Canny in 1986 [71]. The Canny algorithm adds some pre and post processing steps to the edge detection, allowing for a more intelligent identification of edges. A blur operation is used to remove some of the high frequency information in the image that can cause unwanted line segments to appear in the output. A threshold can be applied to the edges to remove low strength responses, retaining only the edges that fall above the user specified threshold. This algorithm is still a standard choice for edge detection operations in modern algorithms [65],[66]. Figure 9 and Figure 10 show an example of Canny edge detection process.

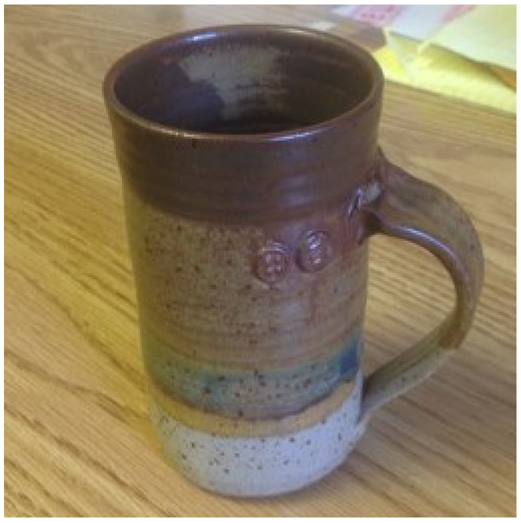


Figure 9: Input to Canny Edge Detector.

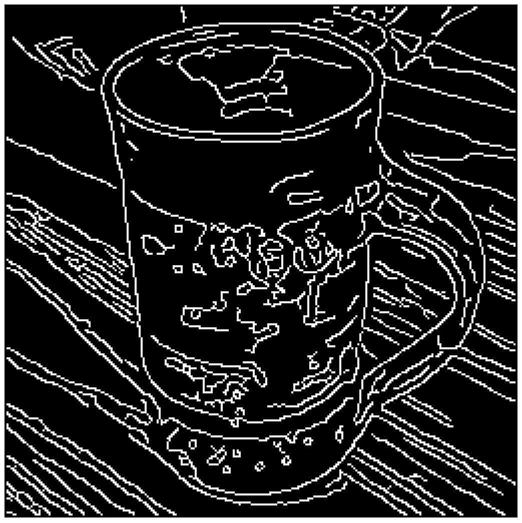


Figure 10: Output of Canny Edge Detector.

Thresholding is a simple example of image segmentation. A typical thresholding algorithm will divide an image into a binary set of either foreground or background. This is analogous to a classification problem between two classes [53]. The simplest example is a gray level intensity threshold, where every pixel with an intensity value below the threshold level is background and every pixel above is foreground. Thresholding can also be based on the image intensity histogram. Finding local maxima in the histogram of a gray image can segment an image into multiple classes [72]. Different assumptions about the distribution of intensities in the histogram can lead to different classification schemes, but all of these only incorporate the histogram information, with no regard for spatial information about the image. Thresholding can fail in images with high noise or an uneven background. Insufficient illumination can also create poor segmentation results [53].

Iterative pixel-based techniques attempt to classify each pixel into a set or class based on a set of information about that pixel. An example of this is described by Jain et al. where a set of Gabor filters are applied to each pixel and computes a response based on itself and its neighbors [73]. Gabor filter responses indicate different textures in the image, and similar textures can be grouped to form image segments. Some other examples of features used in iterative pixel-based techniques are color, pixel location, temporal shift in the case of video, image histogram information, to name a few [56],[49]. An example of texture based segmentation is shown in Figure 8 (shown again below) in the previous chapter.

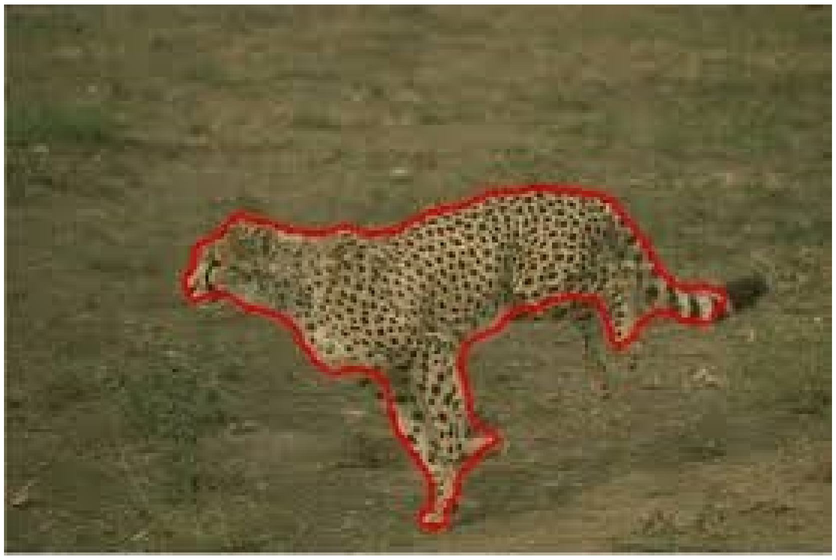


Figure 11: Texture Based Image Segmentation. Y. Kee, M. Souiai, D. Cremers, and J. Kim, "Sequential Convex Relaxation for Mutual-Information-Based Unsupervised Figure-Ground Segmentation," IEEE Conf. onf Comput. Vis. Pattern Recognit.*, 2014. [Fair Use]*

The introduction of Graphics Processing Units (GPUs) into the scientific world has opened significant possibilities in the area of iterative pixel-based segmentation techniques [74]. As long as operations on pixels do not depend on a value computed for another pixel in the image, all of the operations can be performed independently and in parallel. This allows for slower, more complex algorithms to be implemented in time-critical situations without affecting the computational speed of the overall program, as long as these algorithms can be implemented in a parallel manner.

Graph-based segmentation techniques attempt to use previously developed efficient algorithms for finding minimum normalized cuts in a graph on an image [75]. Pixels are represented by nodes in a graph. Edges are added in between nodes with weights based on the similarities in the features of interest. Edges can be added between all nodes, neighboring nodes, or some arbitrary subset of pixels. Once the edges are defined and assigned weights, a minimum cut algorithm can be used to subdivide the graph into two connected components. This cutting process can be repeated until the graph is subdivided into a number of components, and these components are synonymous with segments of an image.

Traditional minimum cut algorithms can result in bad partitions where the minimum cut of a graph segments one node from all other nodes. A normalized cut algorithm, proposed by Shi et al. in 2000, defines a different cost function to minimize that includes the weights of the connected components [76]. This drives the algorithm to favor larger segments over very small ones. The computational cost of this algorithm is higher than that of a traditional minimum cut algorithm, but it gives better results in the context of image segmentation.

Graphs can also be used in a region growing context [47]. Pixels are again assigned nodes and, if the pixel is similar enough to one of its neighbors in the selected feature space, an edge is added between them. Connected regions are considered homogenous segments of the image. This technique can be extended to require more than one edge to link a pixel to a segment or can be applied with other techniques for a hybrid decision process to attach pixels with edges.

Image segmentation can be accomplished with a construct called a “snake.” Snakes are defined by Kass et al. as “an energy-minimizing spline guided by external constraint forces and influenced by image forces that pull it toward features such as lines and edges” [61]. Snakes can be used to accurately define a region with a rough initial estimate of its position and shape. They iteratively move towards the boundary of the object to be segmented. The process of iteration is a minimization of an energy function that contains two parts: internal and external energy. Internal energy contains an elastic term that attempts to make all the points on the spline equidistant, a stiffness term that attempts to reduce the curvature of the spline, and information about a shape that the spline is attempting to model. External energy is based on a feature of the image that the spline should be attracted towards. For example, an external energy function that seeks high gradients would attract the snakes toward edges in the image. Figure 12 (shown again below) from the previous chapter shows an example of a converged snake.

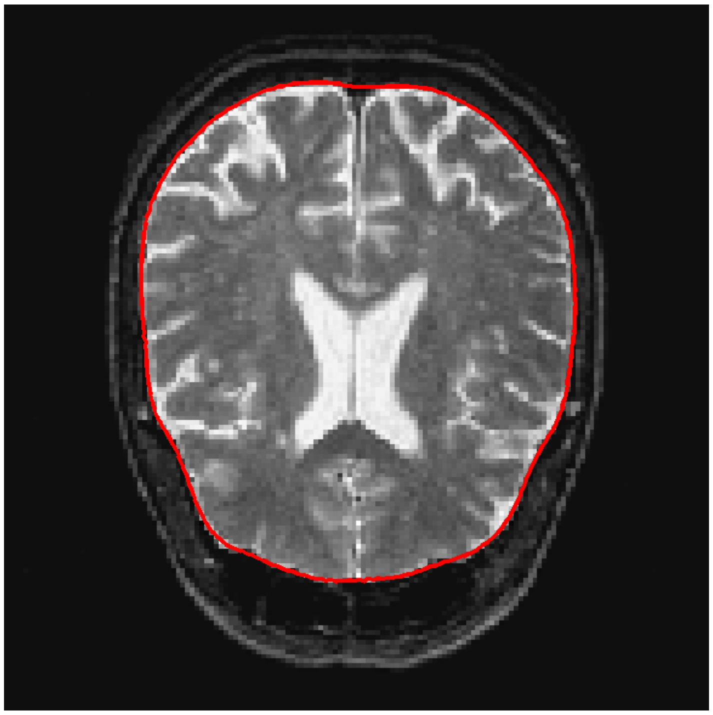


Figure 12: Edge Based Segmentation Example. G. Peyré, "Lecture 1 - Active Contours and Level Sets," ceremade.dauphine.fr*, 2006. [Online]. Available: https://www.ceremade.dauphine.fr/~peyre/teaching/manifold/tp1.html. [Accessed: 01-May-2014]. [Fair Use], 2014*

Clustering algorithms are a popular method of segmenting an image. K-means is a clustering algorithm first proposed over 50 years ago by several different authors in different scientific studies [77]–[79]. K-means operates by initializing random cluster centroids, assigning all points to their nearest centroid, and then moving the centroids to the mean position of all the points assigned to it. The algorithm is iterated until there is no meaningful shift in centroid locations, or until a specified number of iterations passes. This algorithm is very simple to implement, but can be limited in its utility, depending on the structure of the data. It heavily favors spherical clusters in the feature space, and can be computationally expensive. Also, in its most basic implementation, it requires prior knowledge of how many clusters to initialize. If this information is not available to the researcher, it is simply a blind guess. An example of color based k-means clustering is shown below in Figure 13 and Figure 14. The input image is segmented by color and pixels are assigned to their nearest cluster center value.

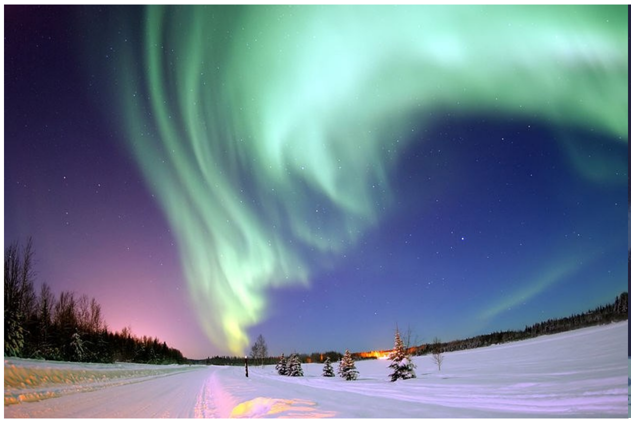


Figure 13: Input Image to K-Means. http://commons.wikimedia.org/wiki/Image_segmentation. Public Domain, 2014

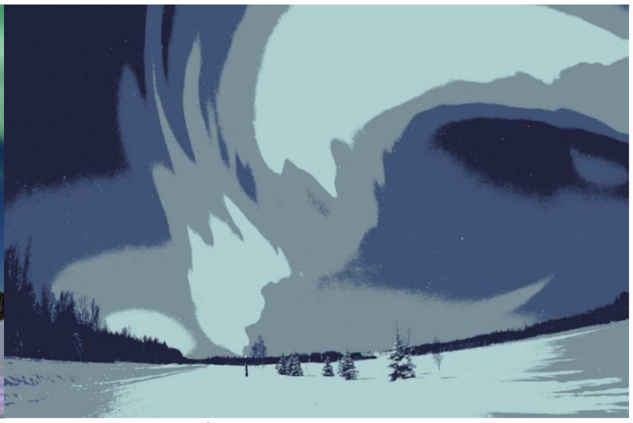


Figure 14: Output of K-Means. http://commons.wikimedia.org/wiki/Image_segmentation. Public Domain, 2014

A method of clustering that avoids the spherical cluster pitfall is called mean shift [80]. Mean shift operates by initializing a specific window at each point in the data, identifies the mean location of all the points in the window and moves the window to that point. This process is repeated until the windows that started at each data point all converge to a local minima. Every point that converged to the same local minima is assumed to belong to the same cluster. This allows for arbitrary cluster shapes, only requiring that the cluster center is a local minima. This operation is limited by the dimensionality of the feature space to be clustered. High dimensional feature spaces are usually sparse, and the density of points is very similar between "high" and "low" density regions [81].

Image segmentation is a key element in the development of the IRIS algorithm in this thesis. The choice of image segmentation method is not necessarily driven by performance in IRIS, as the goal is to compare the speed of segmenting the entire image with the speed of IRIS. The choice of segmentation algorithms must be the same and implemented in the same manner, however. For this implementation of IRIS, k-means segmentation was chosen, as it is well understood and has much literature describing its operation and performance.

3 Method and Theory

This section will discuss the methods incorporated into IRIS and the necessary theory for its development. The driving mathematics of image formation, with respect to a geometric model will be first discussed. After this, the IRIS method will be discussed in detail.

There are three components of IRIS that will be discussed: Lane detection with CHEVP, lane tracking with active contours, and image segmentation using k-means. The fundamental principles of these components, along with the mathematics that drive them will be discussed.

3.1 Image Formation and Geometry

Image formation is the process of projecting 3D information onto a 2D image plane. This process is governed by a projection matrix [82]. Recall from Figure 3 the pinhole camera model, where every ray of light that impacts the image plane passes through the camera optical center. Based on this model, the camera center is in front of the image plane, relative to the object being imaged. The camera center can be assumed as the 3D coordinate origin. The projection of the camera center onto the image plane is the optical center, *C'*. A point *P* with coordinates (*x,y,z*) on the imaged object is projected along a ray passing through the camera center onto the image plane at a point *P'* with coordinates (u,v). For reference, see Figure 15 below. To write the conversion through a projection matrix in a linear form, the image coordinates must be represented in homogeneous coordinates. Homogenous coordinates are a scale invariant transformation where there is a third coordinate assigned to an image location, with the transformation governed by Equation *(1)*.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} \leftrightarrow \begin{bmatrix} \frac{u}{w} \\ \frac{v}{w} \end{bmatrix} \quad (1)$$

When converting a Cartesian point on the image plane to a homogenous point, $w = 1$. This transformation to homogeneous coordinates allows for a transformation to mathematically exist between the object reference frame and the image plane. Let

$$x = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, and\ X = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (2)$$

where *x* is the image homogenous coordinate and *X* is the real world coordinate. Based on these definitions, Equation *(3)* can be formed.

$$x = K[R\ t]X \quad (3)$$

K is a 3 x 3 matrix containing the intrinsic camera parameters, typically called the intrinsic matrix, *R* is the 3 x 3 rotation matrix governing the rotation from the object reference frame to the camera reference frame, and *t* is the 3 x 1 translation matrix governing translation from the object reference frame to the camera frame. The *K* matrix can be further defined in terms of its components, as is shown below in Equation (4).

$$K = \begin{bmatrix} fx & sk & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

In Equation (4), *fx* and *fy* are the focal lengths of the camera expressed in terms of pixels, *cx* and *cy* make the coordinates of the optical center, and *sk* represents the "skewness" of each pixel, or how much it deviates from a square pixel. These parameters are unique to all cameras, and can be approximated through several different processes known as camera calibration [83].

If *K, R,* and *t* are all known, the equation can be solved for either *x* or *X*, given the other. This allows a transformation between a 3D point and a 2D image point generated with a particular camera.

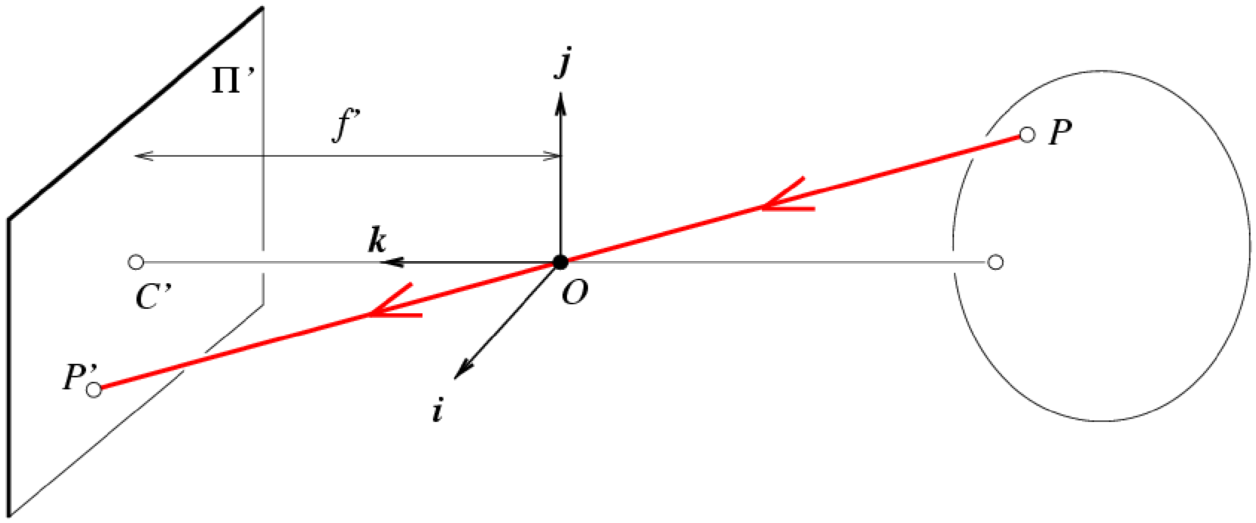


Figure 15: Projective Camera Geometry. J. Ponce, D. Forsyth, and E. Willow, Computer vision: a modern approach. *Boston: Prentice Hall, 2011. [Fair Use], 2014*

3.2 IRIS Overview

The IRIS algorithm attempts to perform lane segmentation in an intelligent manner using information present in the image to create a lane prediction, and then segment only the area within the prediction. IRIS is designed to provide 2 main features:

- IRIS must display similar or better accuracy than the original CHEVP algorithm results
- IRIS must provide faster operation time than standard k-means segmentation algorithms

Based on these criteria, the IRIS algorithm is developed below.

3.3 IRIS Lane Detection with CHEVP

Lane detection is the process of using information present in an image to determine the presence and approximate structure of a roadway lane. The basic assumptions and lane model that is based on those assumptions will be discussed in the following section. Immediately following will be a step-by-step description and mathematical explanation of each step in the CHEVP algorithm. The basic purpose of the algorithm is to create an estimation of the structure of a lane in an image, independent of the parameters of the camera used. By removing the camera parameter dependence to recover 3D information from the 2D image, the algorithm can be applied robustly on many different vehicle/camera combinations with little to no parameter changes. The authors list 5 main goals that this algorithm attempts to achieve:

- The lane detection quality should be invariant to shadows on the roadway
- The algorithm should be capable of processing both painted and unpainted roadways
- The algorithm should be equally capable of detecting curved and straight roadways
- The algorithm should use the parallel lane constraint to reduce the effect of image noise or occlusion of sections of the roadway
- The algorithm should produce a quantitative measurement of the quality of the results

3.3.1 Lane Model and Assumptions

In order to determine the structure of a roadway lane from an image, certain assumptions must be made. The choice of lane model will be important in determining what kinds of roadway features can be detected, as well as the accuracy of the lane detector. The authors of the CHEVP algorithm developed a lane model based on several assumptions. First, the images received from the camera are a 2D representation of a 3D world. In order to recover 3D information about the lane, the lane model must provide constraints about one of the dimensions. The first such assumption is that the two sides of the road are parallel on the ground plane. Extending on this assumption, the right side of the road is assumed to be a horizontally shifted copy of the left side at some distance D where $D = (x_r - x_l)$. In this equation, x_r and x_l are the x coordinates of two points on the right and left lane edges in the ground plane, $P_r(x_r,y)$ and $P_l(x_l,y)$.This can be seen below in Figure 16.

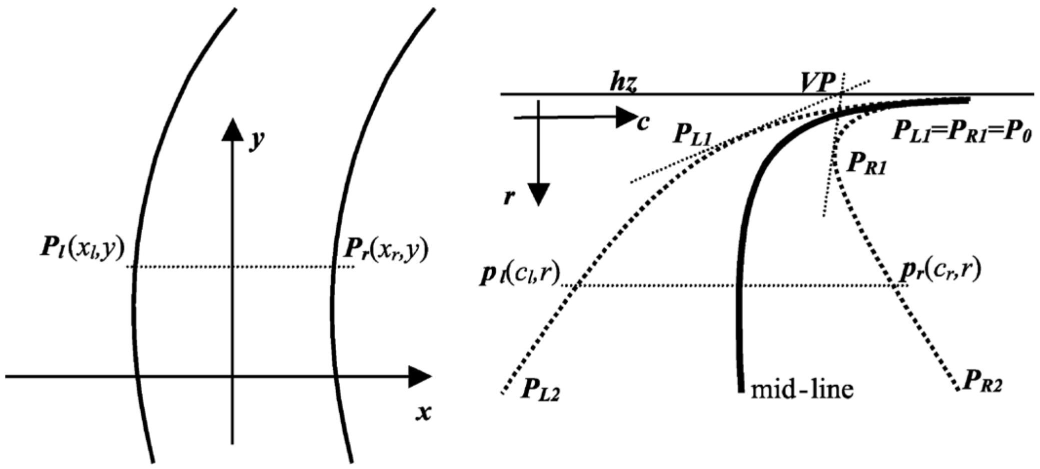


Figure 16: Lane Model used in CHEVP. Reproduced from Y. Wang, E. Teoh, and D. Shen, "Lane detection and tracking using B-Snake," Image Vis. Comput.*, 2004. [Fair Use], 2014*

This defines the assumed 3D lane model. Projecting that onto the image plane, $P_r(x_r,y)$ and $P_l(x_l,y)$ transform into $p_r(c_r,r)$ and $p_l(c_l,r)$, where r is the vertical image plane coordinate and c_r and c_l are the horizontal image plane coordinates. The horizontal distance $d = (c_r - c_l)$ can be determined as a function of camera parameters, camera position, image vanishing point, and vertical coordinate in the image. Equation 1 shows the relationship between d and these variables.

$$d = \frac{\lambda^2 D(r - hz)}{H(\lambda^2 + hz^2)} \tag{5}$$

In Equation 1, λ is the focal length of the camera, *H* is the height of the camera above the ground plane, *hz* is the vertical position of the vanishing point of the lane in the image and r is the vertical coordinate in the image plane. Refer to Figure 16 for reference. Equation 1 can be represented as

$$d = k(r - hz) \tag{6}$$

where k is defined as

$$k = \frac{\lambda^2 D}{H(\lambda^2 + hz^2)} \tag{7}$$

Letting $L_{mid} = (c_m, r_m)$ be the midline of the lane in the image plane, we can define $L_l = (c_l, r_r)$ and $L_r = (c_r, r_r)$ where

$$c_l = c_m - \frac{1}{2}d = c_m - \frac{1}{2}k(r - hz) \quad where\ r = r_m \tag{8}$$

Similarly,

$$c_r = c_m + \frac{1}{2}d = c_m + \frac{1}{2}k(r - hz) \quad where\ r = r_m \tag{9}$$

Based upon Equations 3 and 4, it can be seen that the problem of finding the lane edges in the image plane can be reduced to finding a single lane midline in the image plane. Using these assumptions as the lane model, CHEVP can be used to calculate the midline of the lane, find *hz* and approximate k. The following sections will address each of the five steps in the CHEVP algorithm and present their purpose and mathematical formulation.

3.3.2 Edge Detection

The CHEVP algorithm begins with an edge detection step. Canny Edge detection is a well understood, widely used edge detector that provides a better edge map than a standard gradient operation. The Canny detector was developed in 1985 by J. Canny as an improvement on current edge detection methods [71]. The detector performs 4 discrete steps across the entire image:

- Image noise reduction with a Gaussian kernel convolution.
- Calculation of image gradients
- Non-maximum edge suppression
- Hysteresis edge marking

The input to the Canny edge detector is a grayscale image, which is considered to be noisy. In order to reduce this noise, the image is convolved with a Gaussian kernel. This is mathematically equivalent to smoothing the image. The choice of support and standard deviation for the Gaussian kernel controls the amount of smoothing that occurs in the image. A typical implementation of Canny edge detection will use a 3 pixel by 3 pixel Gaussian kernel with a standard deviation of 0.5, but the choice of values is implementation dependent. If only strong edges are desired, a larger Gaussian kernel with a higher standard deviation can be used to blur the image more. If more edges need to be detected, a smaller Gaussian kernel with a lower standard deviation can be used.

The calculation of image gradients results in two matrices, one with image gradient magnitude, and one with image gradient direction. The image gradients in the X (G_x) and Y(G_y) directions are calculated first, and then used to determine the magnitude and direction of the image gradient using standard vector magnitude and direction.

A non-maximum edge suppression algorithm iterates over all found pixels with a gradient magnitude above a threshold. A pixel is marked as an edge if it has at least one adjacent pixel in its edge direction with a gradient magnitude high enough to be considered an edge. Only pixels that are marked as edges in this way are considered in the next step.

Canny edge detection concludes with a hysteresis based edge marking method. High and low thresholds for this method are supplied by the user. A pixel is marked as an edge in the final map if its gradient magnitude is above the high threshold. Once all of these pixels are marked, each pixel connected to the edge pixels are marked as edges if their edge strength is above the low threshold. This is repeated until no more pixels are marked.

Canny edge detection will result in an edge map where the edges are placed at the center of the edge in the original image due to the non-maximum suppression step, with little to no streaking or dashed edges due to the hysteresis step. A typical output of the Canny edge detector along with its corresponding input image is shown below in Figure 17 and Figure 18, respectively.

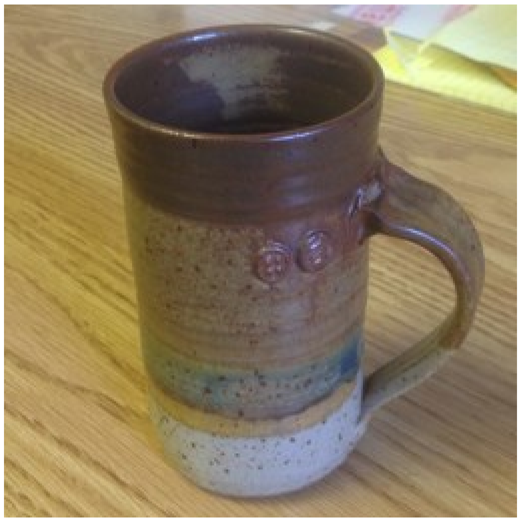


Figure 17: Input to Canny Edge Detector.

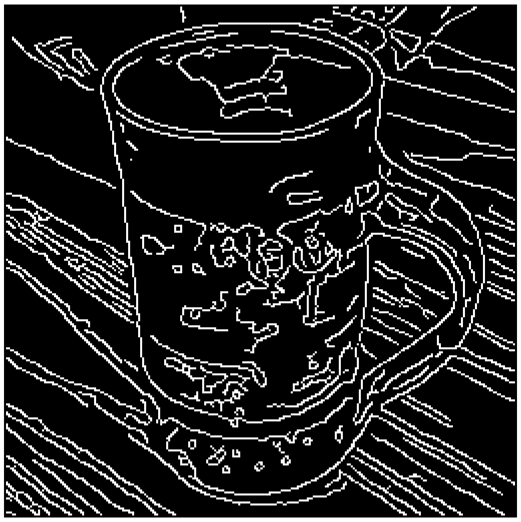


Figure 18: Output of Canny Edge Detector.

Canny edge detection is used in the implementation of CHEVP contained in IRIS. This edge detection scheme was chosen both for its well-understood nature and to maintain consistency with other implementations of CHEVP.

3.3.3 Hough Line Detection

The CHEVP algorithm includes the Hough Transform to detect lines in the edge map output by the Canny edge detector. The Hough Transform is a method of shape matching in which a shape model is projected onto a feature in the image and all possible permutations of its position are marked in a "vote space." This process is repeated for all features in the image, and the point in the vote space with the highest value indicates the most likely position of the shape model in the original image. This is a description of what can be called the Generalized Hough Transform, but for this application it is used for line detection [84].

Lines can be represented parametrically using the equation

$$x\,sin\theta + y\,sin\theta = \rho \tag{10}$$

where x and y are the coordinates of the point on the line, θ is the angle between the line and the positive x axis, and ρ is the distance between the origin and the point (x,y). θ is restricted to $(0 \leq \theta \leq 2\pi)$. For any single line, ρ and θ are constant. This line can also be considered in a (ρ,θ) space instead of the Cartesian (x,y) space. In the (ρ,θ) space, a point fully represents a line in the (x,y) space. The Hough Transform takes advantage of this fact. The (ρ,θ) space is quantized into a matrix with a specified number of cells per dimension. The θ dimension is quantized according to the $0 \leq \theta \leq 2\pi$ restriction by a certain step size, typically 180 or 360 bins along this dimension. The ρ space is quantized from 1 to the maximum distance between the image origin and any image pixel. This quantization is called the accumulator matrix.

The structure of the transform is as follows. The input to the transform is a binary image with the pixels to be considered marked as foreground. The algorithm iterates over each pixel, and for each bin in the θ dimension of the accumulator matrix, the ρ value for that combination of x, y and θ is calculated. The corresponding (ρ,θ) bin in the accumulator matrix is incremented. Once every foreground pixel is considered, the bins in the accumulator matrix with high values correspond to the most likely line parameters. After the accumulator matrix has been completed, either the N highest strength bins are chosen, or all bins with strength greater than some threshold are returned as likely parameter pairs for lines in the input edge map. A typical Hough Transform accumulator matrix is shown below in Figure 19. This accumulator was generated from the edge detection output in Figure 18. The red spots in Figure 19 at (45,300), (45, 365) and (30, 280) show likely parameter sets for lines in Figure 18. These are the accumulator bins that received the most votes, and the (ρ,θ) coordinates of these bins indicate the location of the predicted line(s) in Figure 18.

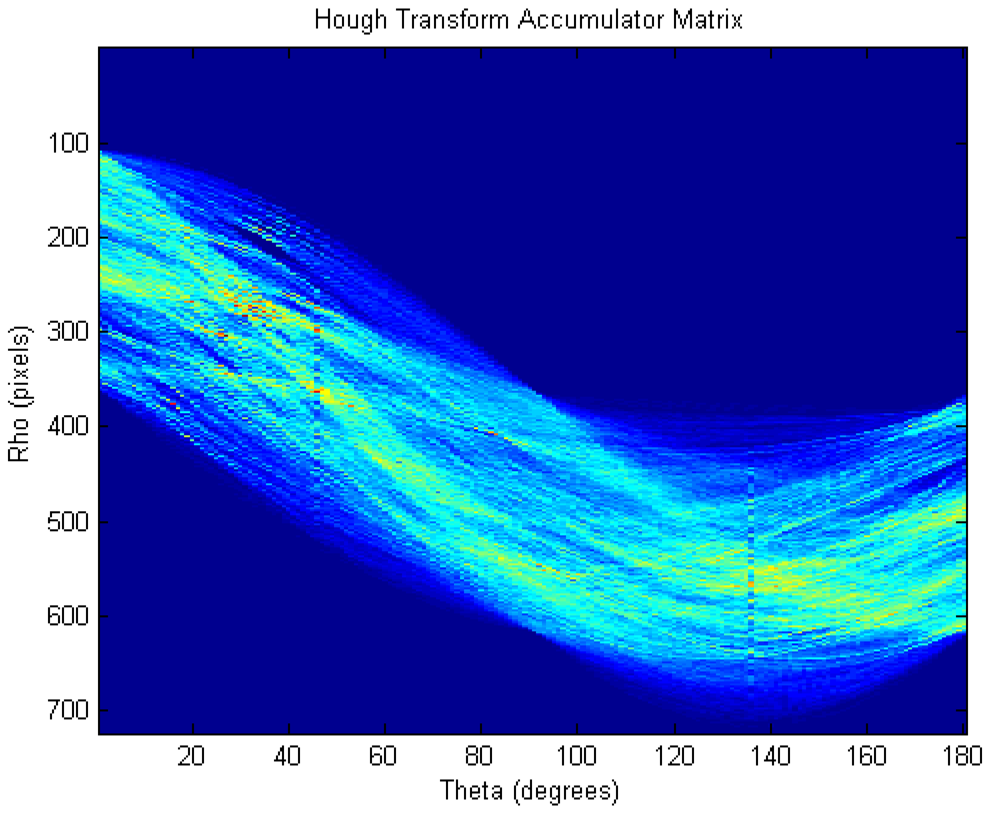


Figure 19: Typical Hough Transform Accumulator Matrix. Matrix generated from Figure 18.

3.3.4 Horizon and Vanishing Point Detection

Vanishing points on the roadway are calculated using the found line parameters in the next step of CHEVP. Every non-parallel line intersects at a point, and these intersections can be accumulated in a new (x,y) voting space to determine the vanishing point of objects in an image. The road is split into horizontal sections, and a vanishing point calculated for each section as well. The combination of the overall vanishing point of the road and the vanishing point of each section can be used to estimate the direction that the road is traveling, and the horizontal component of the overall vanishing point is the constant *hz* in Equation *(6)*.

3.3.5 Lane Midline Estimation

The lane of the road is, as was described in Section 3.3.1, assumed to consist of parallel lines in a horizontal strip that are some distance *d* apart from each other. It was also shown above in Equations *(5)* through *(9)* that the process of finding the lane boundaries can be constituted as solving for the lane midline, the horizon of the roadway and the constant *k*. The horizon was found in the previous step, thus only the midline and k remain to be solved. The camera is assumed to exist between the two lane boundaries that we wish to detect. This assumption is based on typical, and legal, driving patterns. The bottom of the image is used as the horizontal component in the parametric line equation to solve for the x component of the lane boundaries as they pass below the image. The midpoint between these x-components can be assumed as the middle of the lane. A line is drawn piecewise from that midpoint towards the vanishing point of the image. At the border between horizontal image sections, the line direction is adjusted to aim towards the vanishing point of that section, and this process is repeated for all horizontal image sections.

3.3.6 B-Spline Control Point Estimation

The piecewise lane midline estimation generated in the above step is used to create a smooth curve that approximates the lane midline. A Basis Spline (B-Spline) is used to represent this midline. B-Splines are useful in this context because they are described by a small set of parameters relative to the number of interpolated points that can be generated from them.

3.3.6.1 B-Spline Definition

The B-Splines are a set of polynomial piecewise functions that exist in a variable x with some degree k. They are defined over a range from $s_1 \leq x \leq s_m$ where $m = k+2$ [64]. A set of non-decreasing knots defines the behavior of the B-Spline. The B-Spline only exists between s_1 *and* s_m and is continuous at the knot values. For any given knot set, the B-spline is unique, and any spline function of a degree k over a given knot set can be defined as a linear combination of B-Splines. Splines written as a linear combination of B-Splines can be open or closed, and points can be interpolated from the continuous B-Spline combination. Some confusion can occur when discussing B-Splines, as it is not uncommon for curves represented by B-Spline combinations to be simply called "B-Splines". For the purposes of this paper, I will refer to the set of B-Spline functions as B-Spline functions and curves approximated by B-Splines as B-Spline curves.

An open B-Spline curve has a degree k and exists on $s = [0,1]$. It requires $k+1$ control points, $\{Q_0, Q_1 \ldots Q_k\}$ and consists of k-2 connected segments. A B-Spline has u -1 continuous derivatives at $s = 0$ and $v - 1$ continuous derivatives at $s = 1$, where $k = u + v = 1$.For the case of a cubic B-Spline curve, the degree $k = 3$, there are 4 control points, and it consists of one connected segment with a continuous first derivative at each endpoint. The derivative continuity order at a control point is reduced if more control points are placed coincident to it, losing one order of continuity for each repeated control point. Each curve segment is a linear combination of 4 control points by a parameter s, which is bounded $0 \leq s \leq 1$. The curve segment can be expressed as

$$g(s) = \sum_i M_i(s) Q_i \tag{11}$$

where $M_i(s)$ are the B-Spline basis functions. $M_i(s)$ controls the influence of the i-th control point on the segment. In order for a B-Spline curve of degree k to pass through a particular point, there must be k knots at that point. This is caused by the relationship between knot multiplicity, p, and basis functions for the B-Spline curve at that knot. The number of non-zero basis functions is related by k-p+1. If k=p, there is only one possible non-zero basis function, and thus only one control point has a non-zero coefficient, meaning that the interpolated curve is forced to pass through this point. This means that for a curve to pass through two specified endpoints, there must be 7 knots, with the first 3 at one endpoint and the last 3 at the other endpoint. This gives 3 unique knots, corresponding to 3 unique control points: one at each endpoint, and one in the middle controlling curvature.

To represent curves that require more than one B-Spline segment to interpolate, the B-Spline is interpolated for the first $k+1$ control points over s, and then shifted to the next control point and the process is repeated. For example, if there are 7 control points, control points 0, 1, 2 and 3 are first considered, and then points 1, 2, 3 and 4, points 2, 3, 4 and 5 and so on until all control points have been considered. The degree of the spline determines the derivative continuity at the endpoints of each interpolation according to the information given above.

3.3.6.2 *Control Point Estimation*

The beginning and end of the estimated lane midline are chosen as control points. The middle control point is calculated based on the middle knot, which is chosen based on the relative angles between midline segments. This chosen knot, P_1, can be used to calculate the middle control point according to Equation *(12)*.

$$Q_1 = \frac{3}{2}P_1 - \frac{1}{4}(Q_0 + Q_2) \tag{12}$$

Once calculated, the 3 control points are combined with the 7 element knot vector, with the first and last three knots identical, and the middle set to P_1. This creates 4 segments that the B-Spline curve exists on. Points on the spline can be interpolated using Equation *(11)*.

3.4 IRIS Lane Tracking with Active Contours

Once the initial lane estimation is performed using CHEVP, IRIS will track the lane frame to frame using active contours, or “snakes” based on the initialized B-Spline curve. Lane tracking is faster than lane detection using CHEVP, based on the assumption that the lane shape does not typically vary highly from frame to consecutive frame. It can be reasonably assumed that as long as the initialization is near the true lane shape and the time difference between frames is small, lane tracking will converge in a shorter amount of time than CHEVP.

The time complexity of the CHEVP algorithm can be considered using orders of magnitude. The larger complexity operations will dwarf the number of operations of the smaller ones, so the order of complexity will be dominated by the complex operations. These operations are the Canny edge detector and the Hough Line Transform. The Canny edge detector has an order of complexity of $O(mn\log(mn))$ where m and n are the image dimensions, based on the implementation of the FFT-based convolution across the image [85]. The Hough Line Transform has an upper bound of order of complexity of $O((\rho\theta)^2)$, and $\rho\theta \cong mn$ [86]. This indicates that the Hough Transform dominates CHEVP in terms of time used.

For the lane tracking algorithm, the image gradient calculations and the control point update calculations are the most computationally intensive pieces. The image gradient is a FFT based convolution, which was established above to be$O(mn\log(mn))$. The control point update is a loop with 4000 repeated matrix multiplication operations, which can be considered$O(mn)$. This indicates that the image gradient operation is the more time complex of the two main algorithm pieces.

Because the lane tracking algorithm and the Canny edge detector exist in the same order of time complexity, and the Hough Line Transform was determined to have a higher order of time complexity than the Canny edge detector, it can be justified that the lane tracking algorithm will converge in less time than CHEVP.

The following sections will introduce the mathematics of active contours in general, as well as the specific B-spline curve active contour method used. For the purposes of clarity, the B-spline curve active contour will be referred to as a B-snake.

3.4.1 Active Contours

Active contours are continuous spline models that update their position in an image based on a minimization of energy [61]. The total energy function that the active contour, or snake, attempts to iteratively minimize can be represented as

$$E_{snake}^{*} = \int_{0}^{1} E_{snake}\big(v(s)\big)ds \tag{13}$$

where the position of the snake is parametrically represented as $v(s) = \big(x(s), y(s)\big)$. The E_{snake} value can be further split to give Equation *(14)*.

$$E_{snake}\big(v(s)\big) = E_{internal}\big(v(s)\big) + E_{external}\big(v(s)\big) \tag{14}$$

$E_{internal}$ represents the internal energy of the snake and $E_{external}$ represents the external energy supplied by the underlying image. The internal energy is responsible for maintaining smoothness and continuity in the curve, and the external energy is responsible for shifting the snake to the contour in the image that is to be marked. Several types of typical external energy functions exist, and as long as the snake can gather a value for external energy from the function, any image property or high-level feature could be used as the external energy basis.

Active contours are particularly useful in object tracking. Once the snake has been initialized near a contour to be tracked, it iteratively updates its position until it minimizes the energy function. If the contour to be tracked changes shape or moves, the energy function value will increase. This will cause the snake to shift to a lower energy position, which should be the new contour shape or location, assuming the change is not drastic.

3.4.2 Internal Energy

The typical internal energy function for an active contour can be written as follows:

$$E_{internal} = \frac{\alpha(s)|\overrightarrow{v_s}(s)|^2 + \beta(s)|\overrightarrow{v_{ss}}(s)|^2}{2} \tag{15}$$

The first term controlled by $\alpha(s)$ is a first order differential term, and the second term controlled by $\beta(s)$ is a second order differential term. These correspond to a snake elasticity term controlled by $\alpha(s)$ and a snake stiffness term controlled by $\beta(s)$. By controlling the values of $\alpha(s)$ and $\beta(s)$ the behavior of the snake can be controlled.

Internal energy can also contain a third term, which is a shape preference. The relative distances between the points on the snake can be set to match some template, which is useful when attempting to detect specific objects in the image. The previous knowledge of the shape, referred to as a "shape prior", would increase the internal energy if the snake did not follow the template.

Internal energy is not necessary for the snake to find objects in an image, but it allows the snake to maintain a particular shape as it nears the image energy minima. If the underlying curve that the snake updates contains some stiffness and/or elasticity, the internal energy calculation in the snake can be ignored, and the properties of the updated spline used instead. This is one advantage of using the B-snake method, as the B-spline curve itself contains a shape prior (bounded endpoints) and stiffness, based on the degree of the spline [87]. This allows the B-snake algorithm to focus on the external energy minimization alone, and allow the internal energy to be implicitly determined by the B-Spline curve as it is interpolated.

3.4.3 External Energy

The type of external energy to use is an application specific choice. External energy is typically derived from the image gradient, but it can be based on any image feature that can be represented as a numeric value. When calculated from the image gradient, the typical energy function can be represented as a magnitude or as a set of orthogonal vectors. When calculated as a magnitude, it takes the typical form

$$E_{external} = -|\nabla I(x,y)|^2 \tag{16}$$

where *I(x,y)* is the image. This attracts the snake to contours with large gradients. When considered as a set of vectors, the x and y gradients $G_x$and G_y are calculated and stored separately. When computed as a set of vectors, the direction of the gradient can also be considered, but this increases the computational complexity of the snake iteration process.

3.4.4 B-Snake Lane Tracking

The B-Spline curve lane model consists of 4 state variables that will be updated as part of the tracking algorithm, the 3 control points and *k*. Because the lane boundaries can be expressed in terms of *k, hz,* and the lane midline curve, each boundary can be used to calculate an external energy, and the combination of these two energies will deform the lane midline. The intrinsic stiffness of the cubic B-Spline curve will act as the internal energy for the tracking algorithm, and thus only external energy will need to be calculated.

The external energy calculated for each point on the left and right lane boundaries will be represented by $E_L(s)$ and $E_R(s)$. These are vector values, and will sum to form the total internal energy of the midline.

$$E_{midline} = E_L(s) + E_R(s) \tag{17}$$

The vector sum in Equation *(17)* will push the midline in a direction. This updates the 3 control points. When the B-Snake approaches the lane boundaries, $E_{midline} \approx 0$. When $E_{midline} \approx 0$, there is no change in position or shape of the B-Spline curve for the next time step. Based on this, the following equation can be formed.

$$E_{external} = E_{midline} = \gamma\left(L_{mid}(t) - L_{mid}(t-1)\right) \tag{18}$$

Equation *(18)* can be further developed into

$$E_{external} = \gamma M_r(s)\left(Q(t) - Q(t-1)\right) = \gamma M_r(s)\Delta Q(t) \tag{19}$$

where γ is a step size constant, $M_r(s)$ is the basis function for the B-Spline curve and $\Delta Q(t)$ is the difference between the control points of the current and previous iteration.

To update *k*, the difference between the horizontal components at of the lane boundary energies is considered.

$$E^x_{midline} = E^x_L(s) - E^x_R(s) \tag{20}$$

This equation will be used to update *k*. If the value of $E^x_{midline}$ is positive, *k* will increase and the lane model will widen. If $E^x_{midline}$ is negative, k will decrease and the lane model will narrow.

3.5 IRIS Image Segmentation with K-Means

Image segmentation is the process by which an image is subdivided into a discrete set of regions that are similar within the region and distinct from other regions [50]. By segmenting an image into discrete clusters, information about the existence of objects in the image can be deduced. K-means is a standard, well understood method of data clustering, and its applications in image processing have been studied for several years [47]. By reducing the amount of data that needs to be clustered based on knowledge of the lane geometry, the clustering algorithm will operate on the reduced data set in a shorter amount of time than if it operated on the entire image. The lane model can be used to create a region of interest that will be passed to the clustering algorithm, focusing detection on the lane the vehicle is travelling in as well as speeding up segmentation.

3.5.1 K-Means

The k-means algorithm uses a user defined distance function, $f(s_1,s_2)$, to cluster a data set consisting of a number of points, s, into k partitions or clusters [79]. K-means is not guaranteed to converge to a global optimum solution, and likely will not. It will, however always converge to a local optimum solution. The utility of the algorithm comes in the ease of programming and computational economy compared to other clustering functions. The data set can be of any number of dimensions, denoted as N. The function $f(s_1,s_2)$ must operate on all N dimensions of two data points. Typically, $f(s_1,s_2)$ is chosen to be the Euclidian distance function.

To compute the clusters, k data points are initially chosen to be "cluster centers" and assigned a label. For every point in the data set $f(s_1,s_2)$ is computed between the point and each of the cluster centers. The point is assigned the label corresponding to the closest cluster center. Once every point has been assigned a label, the mean of each cluster is calculated and the center reassigned to the mean location. This process is repeated until the movement of each cluster center falls below a user-specified threshold.

This process tends to create "spherical" clusters in the feature space [81]. As the feature space can be N-dimensional, spherical does not have a typical meaning in terms of a 3D shape, but the clusters tend to favor symmetry with the furthest points in any direction approximately equidistant from the cluster center.

3.5.2 Cluster Features

The segmentation occurs in an N dimensional feature space, where each of the N dimensions represents a feature of a data point. These features can be chosen from any image element that can be calculated on a per-pixel basis. This ranges from pixel location, to intensity and hue values, up to a response to a convolved filter set. As long as the result of the operation is a single value, it can be considered a feature in the clustering operation. Each data point is represented as a vector of features, and based on this vector representation, distance, direction and other vector values become useable.

Choosing the features of interest to cluster is implementation dependent. For example, when detecting objects with different textures, a clustering of pixel hue values might not give a distinct segmentation for each object. Rather, a texture-based filter set should be convolved over the image, and the filter set responses chosen as features. For roadway image segmentation, hue, saturation, and image gradients are good choices for features. There are other features that could be useful as well. The hue and saturation will attempt to segment objects of similar color and intensity of color, hopefully identifying the road as one cluster. The image gradients should identify lines and changes in intensity, which should lead to sharp cluster edges where there is a large image gradient.

3.5.3 Lane Based Region of Interest

After the lane model has been acquired from the B-Snake algorithm, it can be segmented. The lane model is used to create a region of interest in the original image, and that region will be segmented. The segmentation should be faster than segmenting the entire image. Time complexity analysis will not be thorough enough to prove this, but in the Results section of this thesis, the comparative speed between segmenting the entire image and lane based intelligent segmentation, as performed in IRIS, will be discussed.

4 Implementation of Method and Theory

This section discusses the specific implementation of the method and theory discussed in the previous chapter. IRIS was implemented in C++ using the Qt, OpenCV, Eigen, and dlib open-source libraries.

Qt is an open source, cross-platform library for developing user interfaces and applications using C++ [88]. It features high level container classes with sophisticated standard functionality, cross-platform GUI and file system interaction, and a standard event-loop driven software model. The cross-platform quality is important, as it allows programs developed on a particular operating system to run in the same manner on a different operating system. This is very useful, as it abstracts the programmer from the specific operating system functions, allowing for cleaner, easier to develop programs. The documentation for Qt is at *qt-project.org/doc.*

OpenCV is an open source computer vision library designed to create a common infrastructure in computer vision applications [89]. It includes a set of standard computer vision algorithms for many different programming languages: C++, C, Python, Java, Matlab as well as integration on Windows, Linux, Mac OS, iOS, and Android. OpenCV is supported by many industry leading companies, and has an active user group with a support page. The online documentation for OpenCV is at *docs.opencv.org* [83].

Eigen is an open source linear algebra library for C++ [90]. It is a header only library, which means that there are no dependencies on external files for the library to function properly [91]. It was first developed in January of 2008 as an element of the KDE Linux distribution. Currently, it is being supported by Inria, and new features are under development for further optimization of the linear algebra functions. It is used in many applications, notably in the development of the Robotic Operating System (ROS) and in the Point Cloud Library (PCL). The online documentation for Eigen is at *eigen.tuxfamily.org/dox/.*

The dlib library is a general purpose cross platform open source library that has an extensive machine learning component [92]. The library has been under development since 2002, but the focus has been on creating a broad set of statistical machine learning tools since 2009 [93]. This includes supervised and unsupervised machine learning algorithms, such as support vector machines, neural networks, k-means clustering, as well as other clustering algorithm implementations. The online documentation for dlib machine learning is at *dlib.net/ml.html.*

The C++ Standard Library, denoted using the std namespace, was also used in the software implementation. The C++ Standard Library is included with every C++ compiler distribution and contains a set of standard classes and algorithms. It is used in this software to create dynamically allocated containers for information, as well as a fast vector sorting method. The library also includes output formats for text and binary files, both for printing information to the system console, as well as writing data to files directly.

The complete code for all parts of IRIS can be seen in the appended documentation and source code. IRIS as a whole is composed of three classes: CHEVPClass, BSnakeClass, and SegmentationClass. Each of these classes is responsible for a different section of IRIS. The classes provide a single function to perform their respective parts of the IRIS algorithm. All three classes are initialized with a variable controlling the verbosity of their operation. If the classes are set to verbose operation, images and text are output from the intermediate steps of the algorithm to show the different step of operation. If the verbosity option is off, the classes run quietly, with no output. This quiet option is the version that would be used in a performance critical system, as well as the version that will be analyzed for timing in the Results chapter. Key design considerations for each of the three classes will be discussed below.

4.1 Lane Detection with CHEVP Implementation (CHEVPClass)

The CHEVP algorithm, as it was described in earlier chapters, consists of five key steps. The implementation of each of these steps will be discussed in their order of operation. A standard input image was used to generate example images for each of the steps detailed below. Figure 20 shows the standard input image.

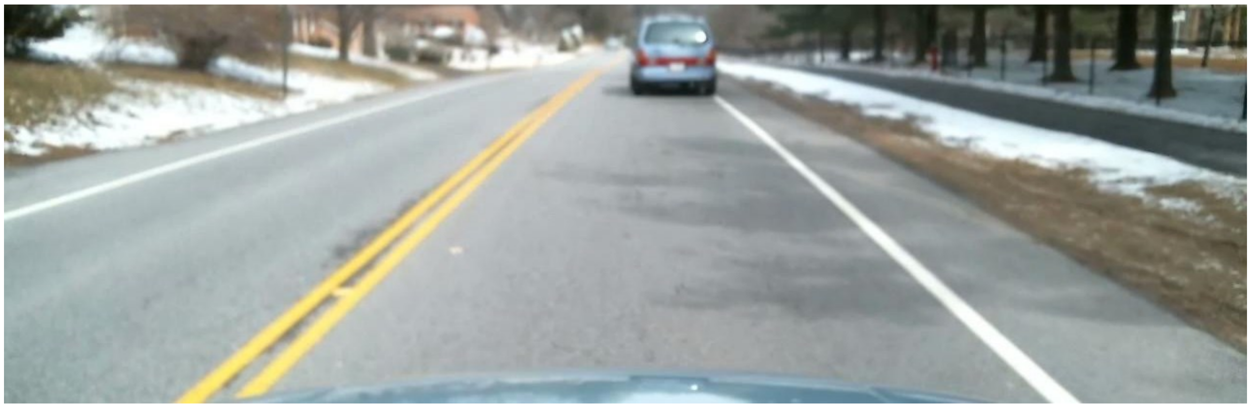


Figure 20: Input Image to CHEVP Algorithms

4.1.1 Edge Detection

The OpenCV implementation of the Canny edge detection algorithm was used in this implementation of CHEVP. The only application-specific parameters of the Canny edge detector are the choices of the Gaussian kernel parameters and the high and low hysteresis thresholds. The Gaussian kernel was left as the typical implementation, a 3 x 3 kernel with a standard deviation of 0.5.

The low and high thresholds are chosen based on the median values in the grayscale version of the input image. This is similar to choosing the mean value of a histogram equalized image, but without the additional step of histogram equalization [94]. Once the median value is calculated, the low and high thresholds are set as $0.66 * median$ and $1.33 * median$ respectively. The output of this step is a binary edge map. Another binary edge map is created using the Canny detector, but from an image where the red color channel has been subtracted from the blue color channel. This gives a high response on the yellow road lines, which can be difficult to distinguish from the roadway in the grayscale image. The grayscale and blue-red channel subtraction images are shown below in Figure 21and Figure 22. These two edge maps are added to create a unified edge map, which will be passed directly into the Hough Transform line detection stage. A typical output of this section of the algorithm is shown below in Figure 23.

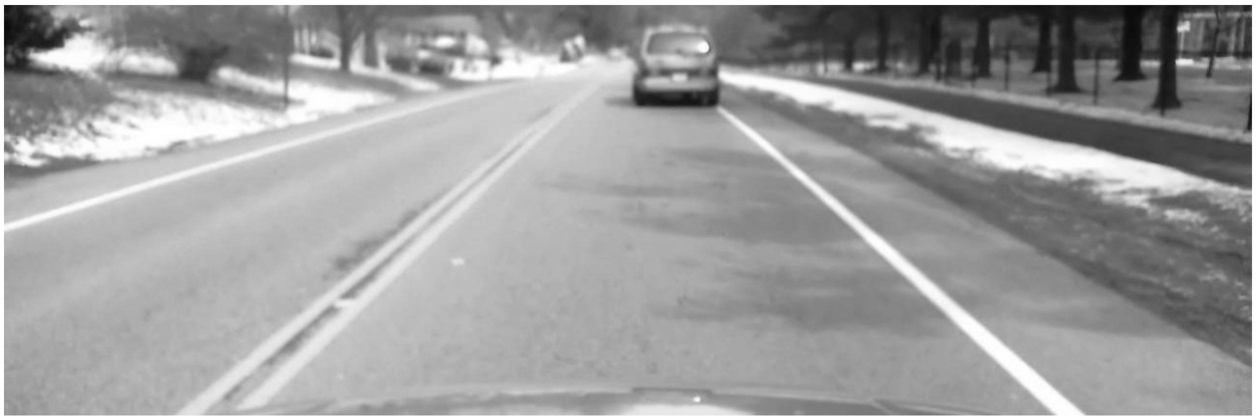


Figure 21: Grayscale Image of Roadway

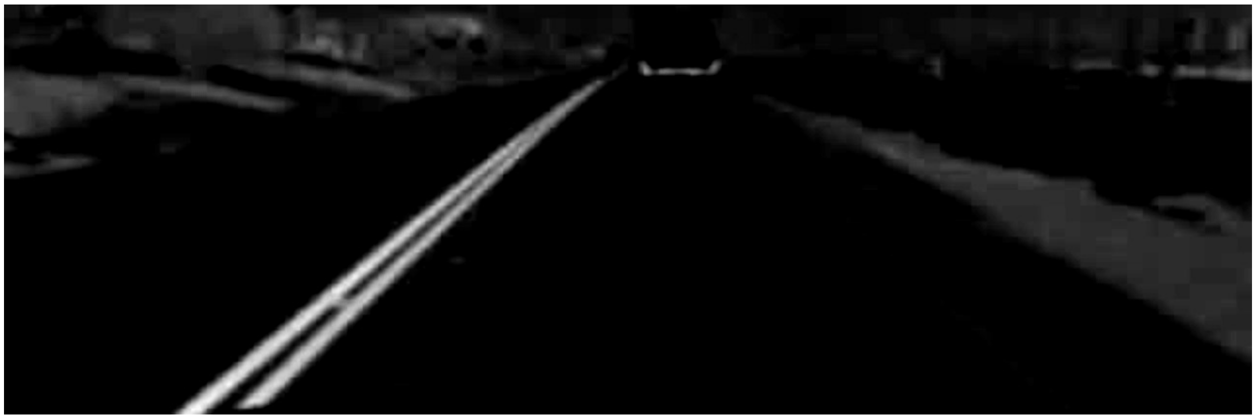


Figure 22: Blue-Red Subtraction Image of Roadway.

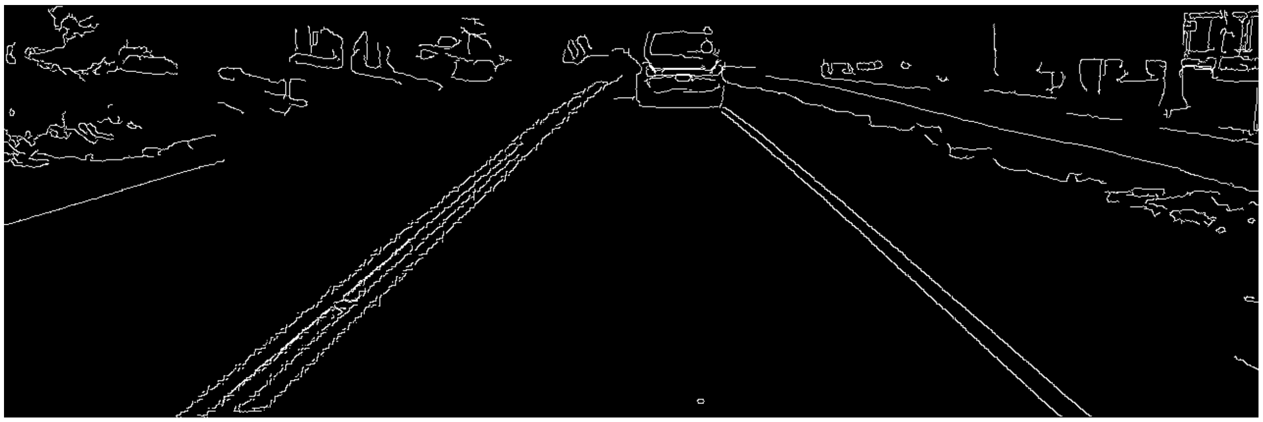


Figure 23: Canny Edge Image

4.1.2 Hough Line Detection

The Hough Transform implementation in the CHEVP algorithm was realized using the OpenCV Hough Transform function. This function returns a set of all the lines found in the binary input image in a vector. The actual values returned in the set are the ρ and θ parameters representing each line.

The accumulator matrix for the transform is not accessible, but if the strength of the line needs to be considered in the algorithm, the order of lines in the returned vector indicates their relative strength to each other. This is used in the later section for vanishing point detection.

The image was divided into 5 sections, decreasing in height from the bottom to the top of the image. The image sections allow different vanishing points to be found for different sections of the roadway. This accounts for curvature in the roadway, as the road will appear to curve when it is further away from the camera. The number of sections was chosen to maintain consistency with the Wang implementation of CHEVP so that comparisons could be made between IRIS and the Wang results.

In order to handle the changing number of lines in each horizontal image section, a dynamically allocated vector data type from the STD library was used. A vector of vectors allowed the lines to be stored by section independent of the number of lines found in each section. This allows each section to be considered individually in later sections of the software. Figure 24 shows the detection of lines in each horizontal section, numbered 1-5 from top to bottom. This naming convention will be adhered to from this point on. Notice how there are different lines detected in each section, but the overall vanishing point of the lines tends toward the same point. Also, no lines were detected in Sections 4 and 5, but CHEVP is capable of detecting the lack of lines and subsequently ignoring that section.

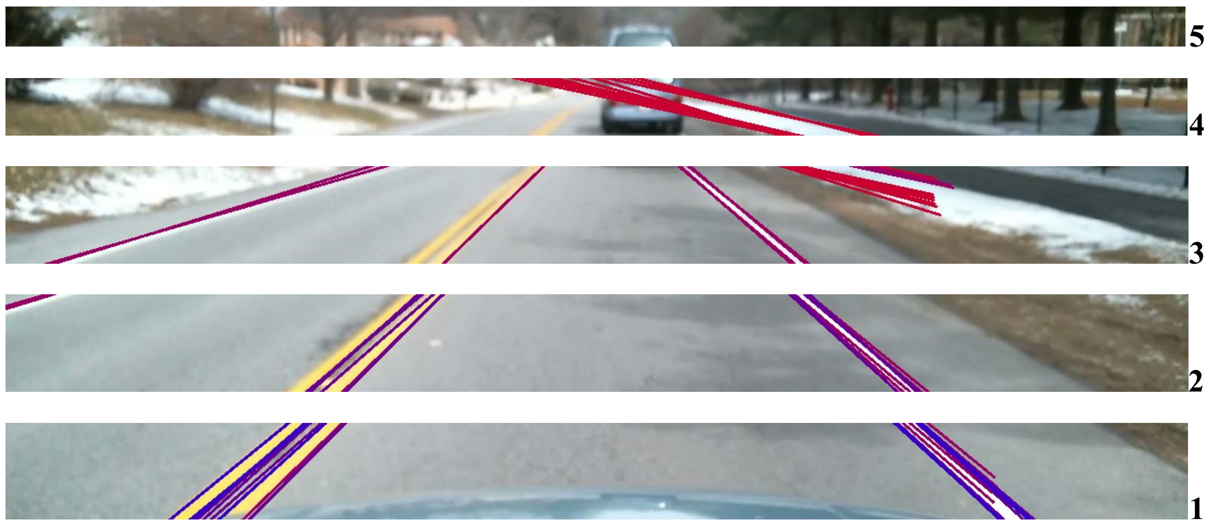


Figure 24: Horizontal Sections of Image with Detected Lines Overlaid.

4.1.3 Horizon and Vanishing Point Detection

The vanishing point detection scheme developed in the Method and Theory chapter was implemented using a Hough-style voting system. Two accumulator matrices larger than the image were initialized with zero values. One matrix serves to find the overall image vanishing point, and the other is reset for each horizontal image section to find the vanishing point per section.

Each image section is considered individually, and all possible line intersection points are found. The points are calculated using the Eigen linear algebra library using a column-pivoting Householder QR decomposition algorithm. The algorithm decomposes a matrix *A* into matrices *P, Q,* and *R* such that

$$AP = QR \tag{21}$$

where *P* is a permutation matrix, *Q* a unitary matrix and *R* an upper triangular matrix. Once this matrix A is decomposed, the results are stored in an Eigen class. This class has a solve function that solves the equation $Ax = B$. The parameters of the two lines are plugged back into Equation *(21)* and two matrices are constructed that take the form

$$A = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ \cos\theta_2 & \sin\theta_2 \end{bmatrix} \tag{22}$$

$$B = \begin{bmatrix} \rho_1 \\ \rho_2 \end{bmatrix} \tag{23}$$

The solution, $x = \begin{bmatrix} x \\ y \end{bmatrix}$ is the intersection point of the two lines characterized by the above parameters.

These points are considered the center of a 10 x 10 pixel square. This square is incremented by a value based on the strength of the intersecting lines found in the Hough Transform in both accumulator matrices. This square was chosen to remove the necessity of the intersections falling on exactly the same point for the proper accumulator bin to be incremented. This removes some accuracy from the calculation, but allows small differences in intersection point to be discounted. The end result of this operation is two identical matrices containing peaks where the vanishing points are most likely to be. The point in the section matrix with the highest value is set as the vanishing point for that section and the section accumulator matrix is reset to zeroes. The overall vanishing point matrix is left with its values, and the next section is then considered in the same way as the first. Figure 25 shows the section of the accumulator matrix with non-zero values. The bright spot in the center of the image corresponds to the overall roadway vanishing point. Figure 26 shows the individual accumulator matrices per image section. It can be seen that there are shared elements between the overall accumulator and the section accumulators, which is expected. The matrix shown below in Figure 25 and Figure 26 exists in the same coordinate space as the image. The bin coordinates are translated back to the image as predicted vanishing points.

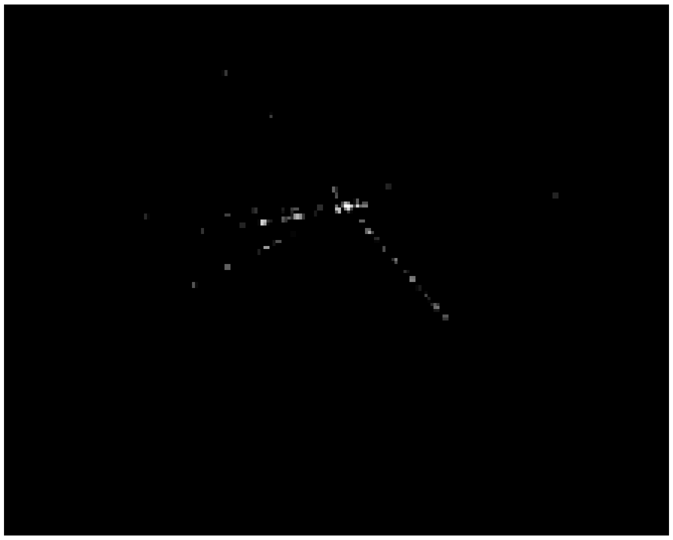


Figure 25: Section of Overall Accumulator Matrix. Section with non-zero values shown

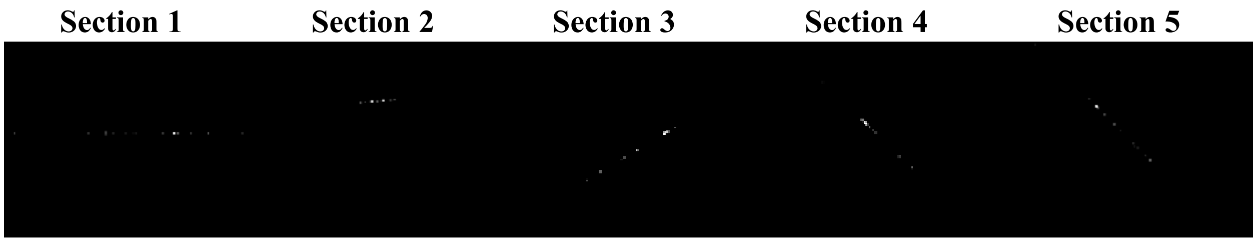


Figure 26: Section Accumulator Matrices. Non-zero sections shown.

The end result after all image sections are considered is a set of vanishing points for each section based on the section accumulator matrices, and an overall accumulator matrix with all lines in every section considered. Each section vanishing point is assumed to exist on the same horizontal image row as the overall vanishing point, based on the planar lane model assumption made in the previous chapter. The overall vanishing point of the roadway can be gathered from this overall accumulator matrix. The height of this point in the image plane will be used as the *hz* value in the next set of calculations. Figure 27 shows the overall vanishing point and section vanishing points overlaid on the input image. The overall vanishing point is in green and the section points are encoded from blue to red, where Section 1 is red and Section 5 is blue. Most of the section vanishing points are very near the overall vanishing point, but one point (Section 4 in this case) is an outlier. CHEVP is also robust to errors of this type, and can correctly identify the lane even when one vanishing point is a significant outlier. This robust behavior is shown in the sections below.

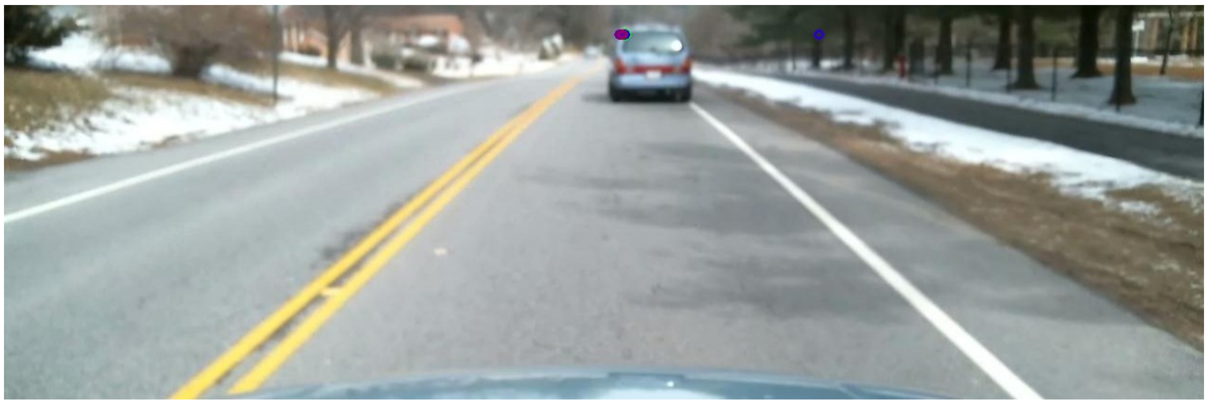


Figure 27: Vanishing Points. The overall vanishing point is in green, and the section vanishing points change from blue to red, with Section 1 as red and Section 5 as blue.

4.1.4 Lane Midline Estimation

After the vanishing points are calculated, the two lines in the lower-most horizontal section that are closest to the middle of the image while being on opposite sides of it are found. These two lines form the inner boundaries of an approximate lane. The intersections of these lines with the bottom of the image are found, and the midpoint between these two points is calculated. This midpoint is the starting point of the piecewise lane estimation. The inner boundaries of the rest of the horizontal image sections are calculated in order to approximate *k*.

The lane boundaries are assumed to be the edges of the lane in the section, and are plugged into Equation *(6)* along with the *hz* value found in the previous section. The equation is rearranged to solve for *k*, and the results for each section stored in a vector. This vector is averaged after it is filled, and the result is the approximation of *k* that will be used to recreate the lane lines.

A line is found that connects the calculated bottom midpoint to the vanishing point of that section, or the overall image vanishing point. The choice between the two depends on what section is currently being evaluated and the consistency of the section vanishing point with the overall trend of the lane. This line is traced until the border of the next section is reached. This point is then marked as the calculated midpoint for the next section and the line tracing process is repeated with a new vanishing point. Figure 28 shows the estimated piecewise midline overlaid on the input image along with the estimated midpoint for each section. Note that the estimations tend to become less reliable when in the upper sections of the image. Also of note is the fact that the estimation fails in the upper sections. This will be considered in the next section as to what endpoint to choose for the B-Spline curve model. The intersection points of the midline and section borders are stored in a vector and passed to the control point estimation phase of the software.

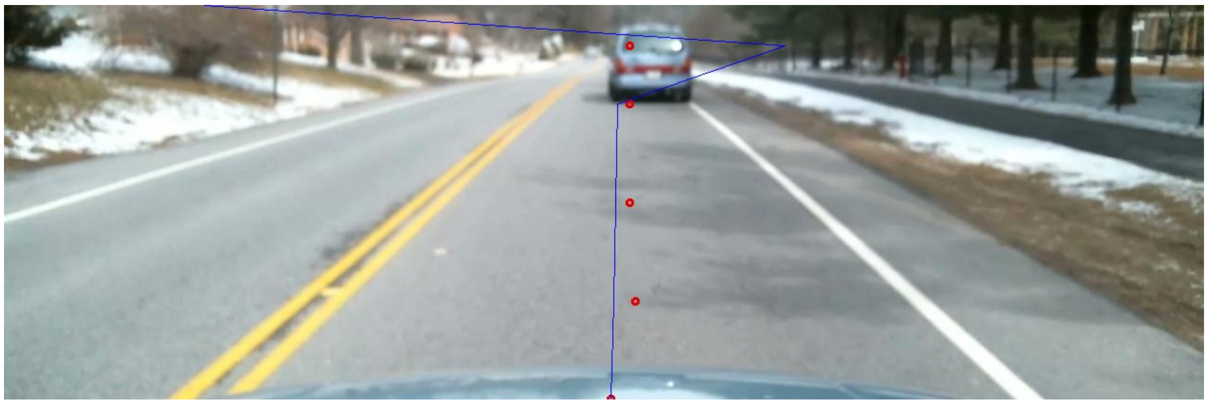


Figure 28: Estimated section midpoints and piecewise lane midline.

4.1.5 B-Spline Control Point Estimation

The final step in CHEVP is estimating the control points for the B-Spline curve that approximates the midline of the lane. The first control point is set to the starting point of the piecewise midline on the bottom of the image. The last control point is set to either the overall vanishing point or the uppermost section vanishing point, depending on the accuracy of the vanishing point detection in the uppermost section. This accuracy is determined based on the overall lane trend and the difference between the overall vanishing point and the upper vanishing point.

The choice of middle control point is more involved. The control point cannot be chosen directly, instead it must be calculated with Equation *(12)*. The knot P_1 can be directly chosen, and exists on the piecewise midline. The choice for P_1 is based on the angles between the piecewise midline segments in the two uppermost horizontal image sections, referred to for clarity as section 4 (lower) and section 5 (top). If the angles are both not zero, P_1 is calculated as the midpoint of the line in section 4. If the angles are the same, the bottommost point on the midline segment in section 4 is used. If the angles are on different sides of vertical, the bottommost point on the midline segment in section 4 is used. If the line is vertical, the topmost point on the midline segment in section 4 is used. Table 1 shows the information discussed in this paragraph.

Table 1: Choice of Control Point Based on Angles

Angle in Section 5	Angle in Section 4	Control Point (P_1)
$\neq 0$ AND $\neq$ Section 4	$\neq 0$ AND $\neq$ Section 5	Midpoint of segment in Section 4
$= 0$	Any value	Lowermost point of segment in Section 5
Anything else	Anything else	Lowermost point of segment in Section 4

Figure 29shows the output of the B-Spline curve initialization phase. The overall vanishing point is used due to the poor vanishing point estimate in Section 5, determined by the lack of detected lines. This provides a good estimate of the lane trajectory, while choosing a lane boundary with a small error.

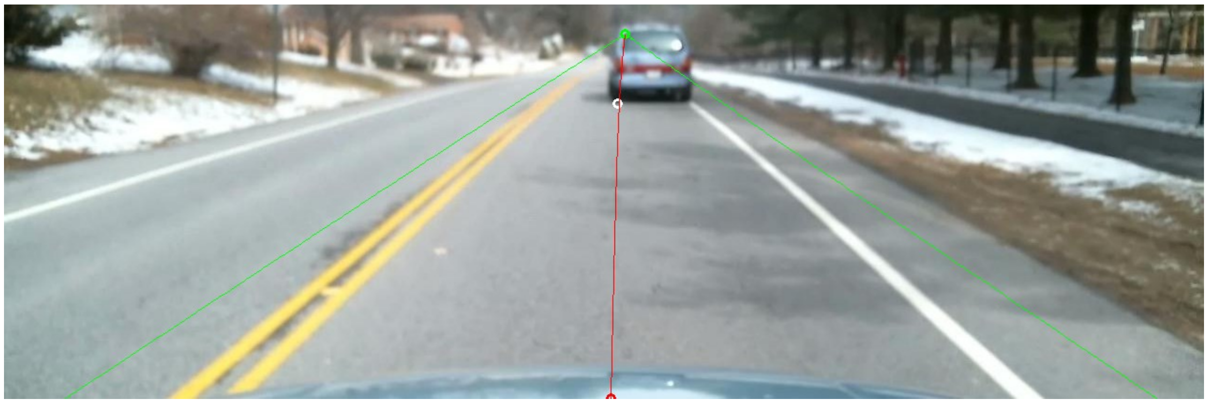


Figure 29: B-Spline Curve Lane Estimate from CHEVP.

4.2 Lane Tracking with Active Contours (BSnakeClass)

After the initial B-Spline curve that approximates the roadway is found, the lane tracking algorithm begins. The initialization for the tracking stage can be either the output from the CHEVPClass runCHEVP() function or the output from a previous iteration of the BSnakeClass updateBSnake() function. The first image in the data set will always run through the CHEVP algorithm, but if the B-Snake converges in a set number of iterations, the B-Snake values will be used to initialize the B-Snake on the next image. If it does not converge, CHEVP will be performed on the next image.

There are two parts to the B-Snake lane tracking algorithm: Computing the external energy function on the input image, and Control point estimation. These are discussed further below.

4.2.1 External Energy Implementation

The external energy function defined as $E_{external}$ in the previous chapter was chosen to be at vector composed of the image gradients in the X and Y directions.

$$E_{external} = \langle G_x, G_y \rangle \tag{24}$$

Based on this definition, a direction and magnitude of $E_{external}$ can be calculated for all pixels in the image. This is performed using the OpenCV implementation of the Scharr filter. The Scharr filter is a specific filter that is convolved across an image to form the image gradient. It takes the form

$$\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$$

when computing the X derivative. The matrix is transposed to compute the Y derivative. The output from this portion of the lane tracking algorithm is two images, one containing $G_x$and the other G_y. These images are normalized to exist in the range $-2 \leq G \leq 2$. This allows for simpler computations in the control point estimation stage of the algorithm. Figure 30 shows the X and Y gradient images calculated on the input image in Figure 20. Any strong edges in the image are highlighted, and the roadway can be clearly seen.

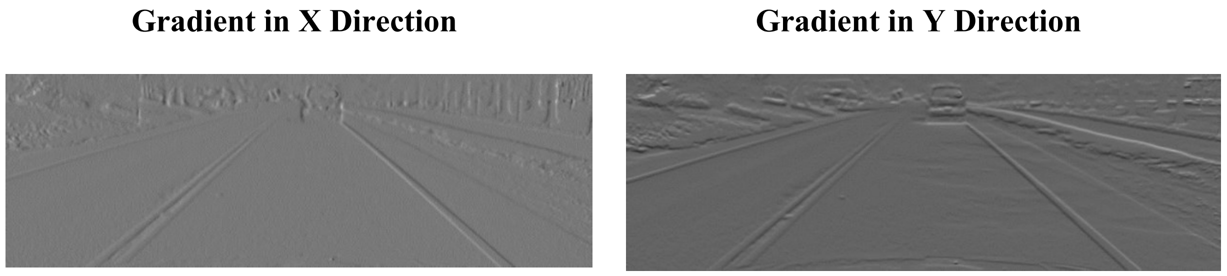


Figure 30: Image Gradients in the X and Y directions.

4.2.2 New Control Point Estimation

After the $E_{external}$ has been calculated, new control points are updated. Recalling Equation (19) from the previous chapter, $E_{external}$ is related by a set of constants to the difference between the control points of this iteration and the previous iterations. $E_{external}$ is also composed of the sum of vectors from the left lane and right lane according to Equation (25).$E_{external}$ can then be sampled at points along the B-Spline curve giving an approximation of the external energy at each chosen point on the midline. This allows an approximate *ΔQ(t)* to be computed using the following equation.

$$\Delta Q(t) = \gamma^{-1}[M^T M]^{-1} M E_{ext}^{-1} \tag{25}$$

M is defined as

$$M = \begin{bmatrix} M_{-1} & 0 & \cdots & \cdots & 0 \\ 0 & M_0 & 0 & \cdots & 0 \\ \vdots & \cdots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & M_{n-1} & 0 \\ 0 & \cdots & \cdots & 0 & M_n \end{bmatrix} \tag{26}$$

where M_i is defined as

$$M_i = \begin{bmatrix} s_{i,1}^3 & s_{i,1}^2 & s_{i,1} & 1 \\ s_{i,2}^3 & s_{i,2}^2 & s_{i,2} & 1 \\ \cdots & \cdots & \cdots & \cdots \\ s_{i,m}^3 & s_{i,m}^2 & s_{i,m} & 1 \end{bmatrix} \begin{bmatrix} -\frac{1}{6} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{6} \\ \frac{1}{2} & -1 & \frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 \end{bmatrix} \tag{27}$$

The first matrix in Equation (27), referred to as *S,* is composed of the powers of the current interpolated point on the B-Spline curve. Because the same number of points are interpolated per section, *S* is constant for all intervals on the B-Spline curve. The second matrix in Equation (27), referred to as *A,* is the basis function for a cubic B-Spline curve. *A* is constant across all cubic B-Spline curves. Due also to the cubic nature of the B-Spline curve, $n = 2$. This allows$\gamma^{-1}[M^T M]^{-1}M$ to be pre-calculated once for all iterations of the B-Snake.

The dimensions of the value$\gamma^{-1}[M^T M]^{-1}M$ are dependent on the number of interpolated points used in the calculation. In this implementation, 100 points per segment were used, giving a total of 400 points. This means that the dimensions of *S* are 100 x 4 and the dimensions of *A* are 4 x 4. This means that M_i is a 100 x 4 matrix and *M* is a 400 x 16 matrix. The result $\gamma^{-1}[M^T M]^{-1}M$ is then a matrix of size 16 x 400. $E_{external}$ is a matrix of size 400 x 2, containing the x and y gradient at each of the 400 chosen points on the B-Spline curve. Thus the size of *ΔQ(t)* is 16 x 2, with the 4 x-y value pairs per segment. Eigen linear algebra functions are used to compute these matrix multiplications. The resulting x-y value pairs are separated by the control point they are associated with and then averaged to get an x-y pair corresponding to the movement of each of the three control points.

Computing a new value for k is also performed in this algorithm section. Based on Equation (28) the following is true:

$$E_k = E_{midline}^x = E_{midline}^x = \tau\big(k(t) - k(t-1)\big) \tag{28}$$

where τ is a step-size constant. Equation (28) can be rearranged such that

$$\Delta k(t) = \frac{E_k}{\tau} \tag{29}$$

Equation (29) governs the shift in k to better approximate the lane.

After *ΔQ(t)* and *Δk(t)* are calculated, they are compared against thresholds to determine if the B-Snake has converged to the lane or not. If *ΔQ(t) < QThreshold* and *Δk(t) < kThreshold* for any given iteration, the B-Snake has converged and is assumed to approximate the lane. If the B-

Snake does not converge after 250 iterations, it is assumed to incorrectly represent the lane, and the CHEVP algorithm is used to initialize the next frame.

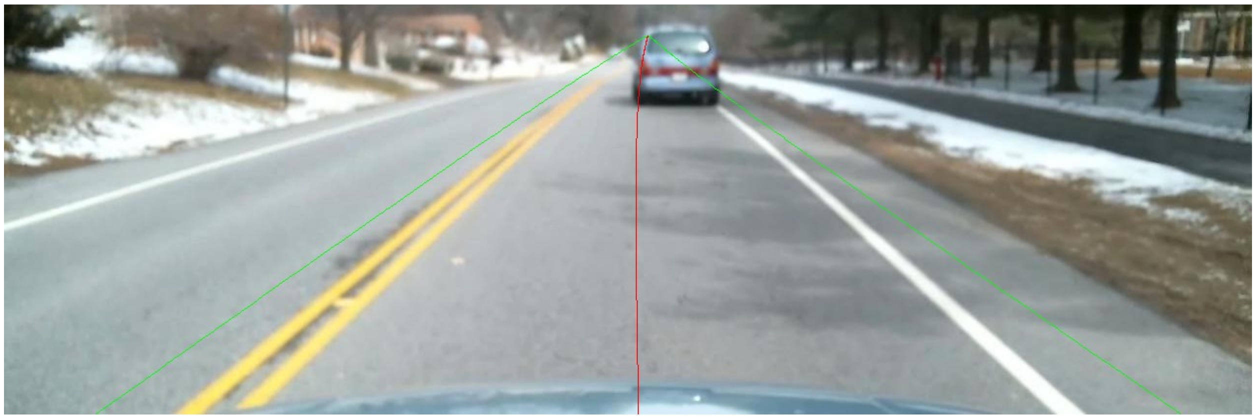


Figure 31: Final B-Snake Location

Figure 31 shows the final B-Snake location. Notice how the lane boundaries have tightened to better approximate the lane shape. It is of note that, for this image, the Lane Tracking algorithm converged in 15 iterations. This means that the *ΔQ(t)* and *Δk(t)* fell below the thresholds for convergence. Example iterations for another image set are shown in Figure 32 to illustrate the effect of convergence over multiple iterations.

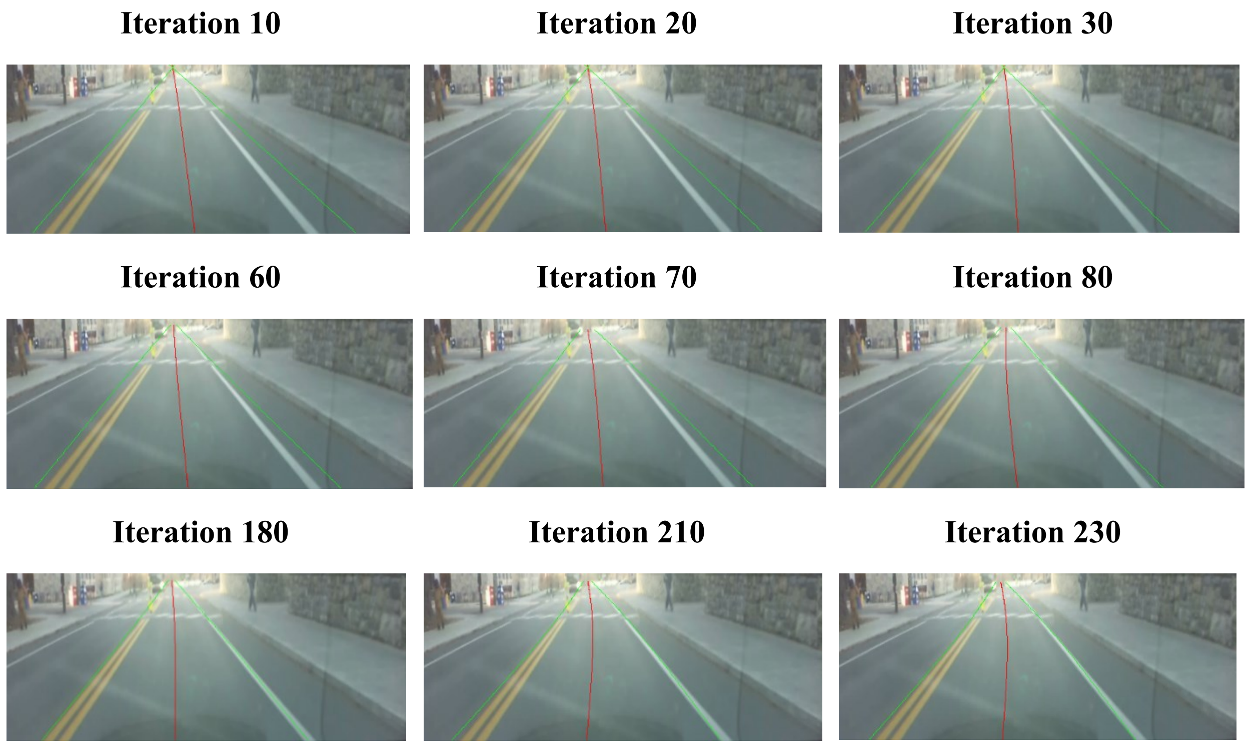


Figure 32: Selected Iterations of Lane Tracking Algorithm.

4.3 Image Segmentation (SegmentationClass)

Once the lane model has been updated, it is passed into the SegmentationClass to begin the image segmentation process. This requires two distinct steps. First, the lane model is used to create an ROI mask that is the same shape as the lane model, and then that mask is applied to the image. Only pixels that fall within that mask are segmented in the second step. The second step is the image segmentation. The image is segmented according to a number of features into a constant number of clusters. The two steps will be discussed further below.

4.3.1 ROI Creation

To create the mask that defines the lane based ROI, two OpenCV functions are used. The first, approxPolyDP() is given a set of sparse points that define a polygon and outputs a dense set of points that fully define it. This is used to transform the left and right lane boundaries into a dense mask that will be used for segmentation. The mask is a binary image where the lane is marked as foreground and everything else is marked as background. If any other portion of the image were to be segmented, the mask would simply need to be updated to contain foreground over the new ROI. The mask created by this function is shown below in Figure 33.

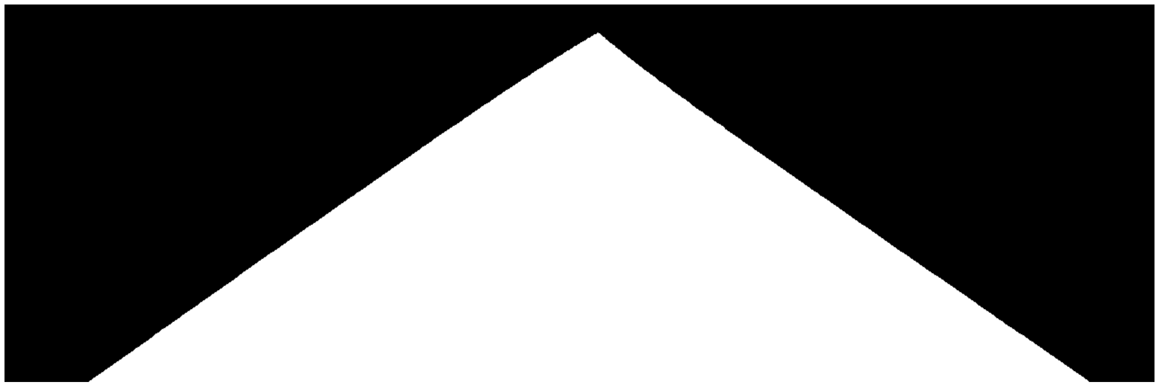


Figure 33: Mask Generated from Lane Model

The second function is boundingRect(). This function is given a set of points and computes the minimum bounding rectangle for those points. The input image is cut down to the size of the minimum bounding rectangle and the mask found above is applied to that image to create the smallest possible rectangular input to the segmentation function. The modified image with the bounding rectangle and mask applied is shown below in Figure 34. This will be used in conjunction with the segmentation code in the following algorithm section.

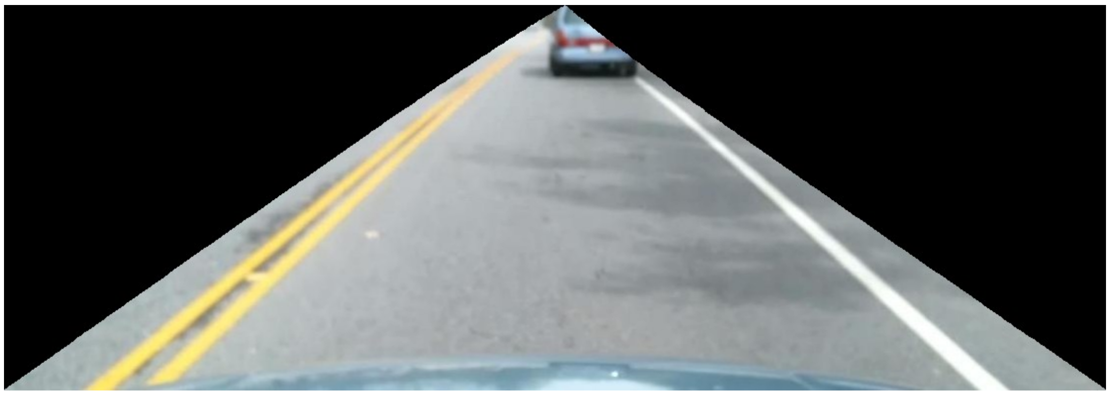


Figure 34: Masked ROI Containing only the Lane.

4.3.2 Image Segmentation

Image segmentation on the ROI is performed using the dlib implementation of k-means. The features that the image is segmented over are supplied to the k-means function, and every pixel is assigned a cluster. The features chosen are basic image features that were readily available in IRIS. The x and y location of each pixel are features, the hue, saturation and value of each pixel are all features. Other features can be included in the clustering if desired. The software separates most of the feature creation from the clustering, allowing for modularity. This particular set of features makes a 5-value feature vector for every pixel, and the pixels are clustered based on these 5 features.

The clustering output is shown below in Figure 35. The lane can be seen clearly in the clustering output, especially the cluster that approximates the double yellow line. This is a marked by a light grey band on the left side of the lane. There are artifact clusters caused by the OpenCV implementation of k-means needing a rectangular region as an input, but when the image is output to the user, it can be masked again using the binary mask image shown in Figure 33. The final output of the program is a clustered lane superimposed over the original input image. This can be seen in Figure 36. The roadway is separated into two clusters, which is not an ideal output, but can be caused by lighting conditions or shadows on the road. The "goodness" of the segmentation is unimportant to this set of experiments as the feature selection and k-means parameter settings are well understood and are left for the particular implementation of IRIS. IRIS will be used to perform a set of experiments that are discussed in the following chapters.

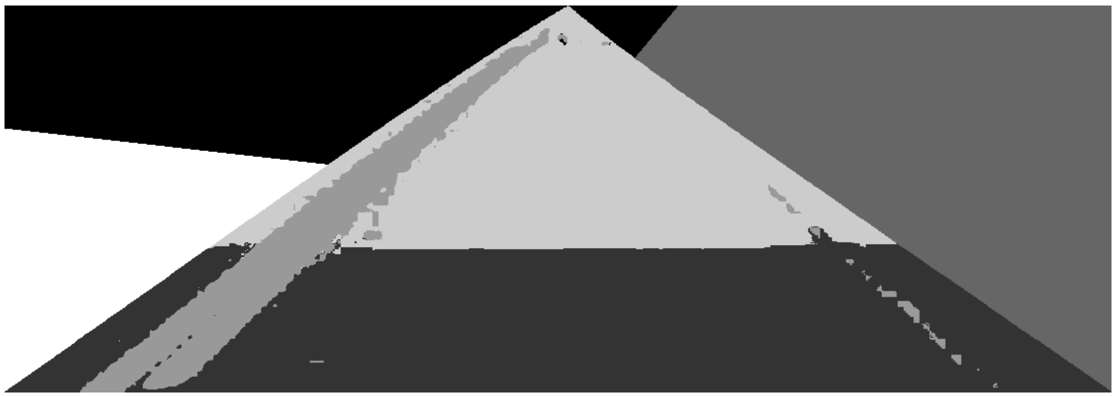


Figure 35: Clustered ROI using 5 Features.

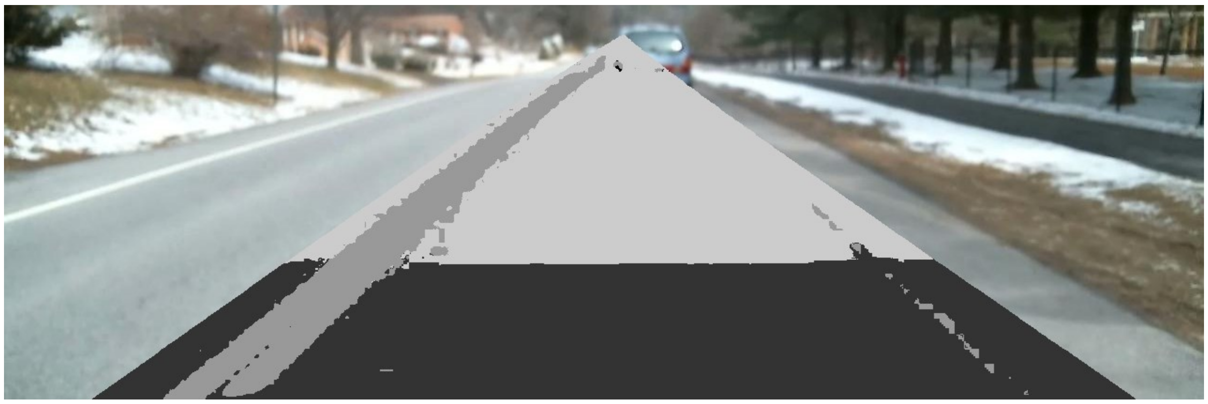


Figure 36: Final Output of IRIS showing segmented lane.

5 Experiment

To test the operation of the IRIS algorithm, two major claims must be evaluated. First, the accuracy of the lane model generated will be calculated. A "ground truth" will be created for each test image, consisting of a hand drawn lane model. This will be compared to the algorithm-generated lane, with the assumption that the hand drawn lane model is accurate.

Second, the claim that the image segmentation performed through the developed algorithm is faster than image segmentation across the entire image will be evaluated. A consistent image set will be segmented using both methods and the time for each image, as well as average time will be calculated. The error in timing associated with multi-tasking operating systems will be explored and integrated into the results of the timing experiment.

To evaluate these claims, a consistent and repeatable data collection method was employed. A camera was mounted to a vehicle and data taken over a wired USB connection. The data collection hardware and method is discussed further below.

The results from these experiments will be tabulated and discussed in the Results chapter. Metrics and expected results will be discussed below.

5.1 Data Collection

The creation of the test data set is described in this section. First, the camera mounting and acquisition method will be discussed. A magnetic mount was used to attach a high definition webcam to the roof of a vehicle, and data was captured via wired connection. Second, the intentional variation within the data set will be discussed. The data set contains images of different road types in different lighting conditions.

5.1.1 Camera Setup and Mounting

The camera mount is a RigWheels RMH1, a magnetic mount with a ¼" threaded bolt for tripod adapted cameras. This mount has holding force in vertical pull of up to 35 lbs. It has a ball mounting head that allows the camera to be angled, and held in place with a set screw. An image of the RMH1 is shown below in Figure 37.

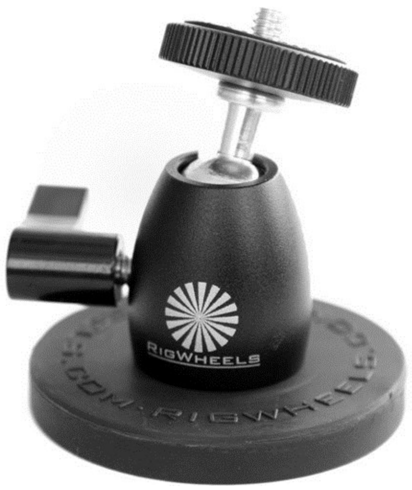


Figure 37: RigWheels RigMount RMH1. http://www.bhphotovideo.com/bnh/controller/home?O=&sku=900569&gclid=CNbR4qid_74CFXQOOgodDysAEA&Q=&is=REG&A=details [Fair Use], 2014

The camera used is a Logitech HD Pro C615 webcam. This webcam is capable of saving 720p video streams at up to 30 frames per second. The camera weighs 0.55 lbs, and is connected to a computer via a USB connection. An image of the Logitech webcam is shown below in Figure 38.

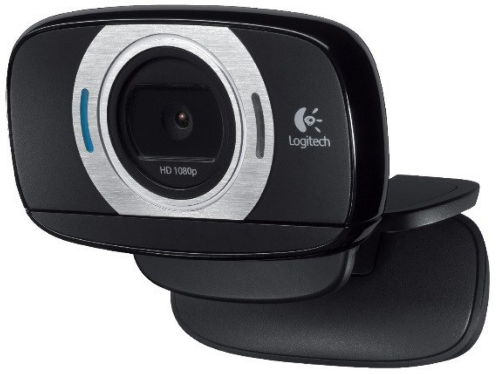


Figure 38: Logitech HD Pro Webcam. http://www.amazon.com/dp/B004YW7WCY. [Fair Use], 2014

The camera and mount assembly was attached to a vehicle using the magnetic mount. The camera was aligned so that it was in the center of the vehicle, as far forward above the windshield as possible. The goal of this alignment is to have the horizon as high as possible in the output image without excluding it from the image, and reducing the amount of occlusion from the vehicle hood. Having more of the lane in the image closer to the vehicle should allow for a better estimate of the lane. Figure 39 and Figure 40 show the vehicle with the camera mounted in the configuration used to take the image data.

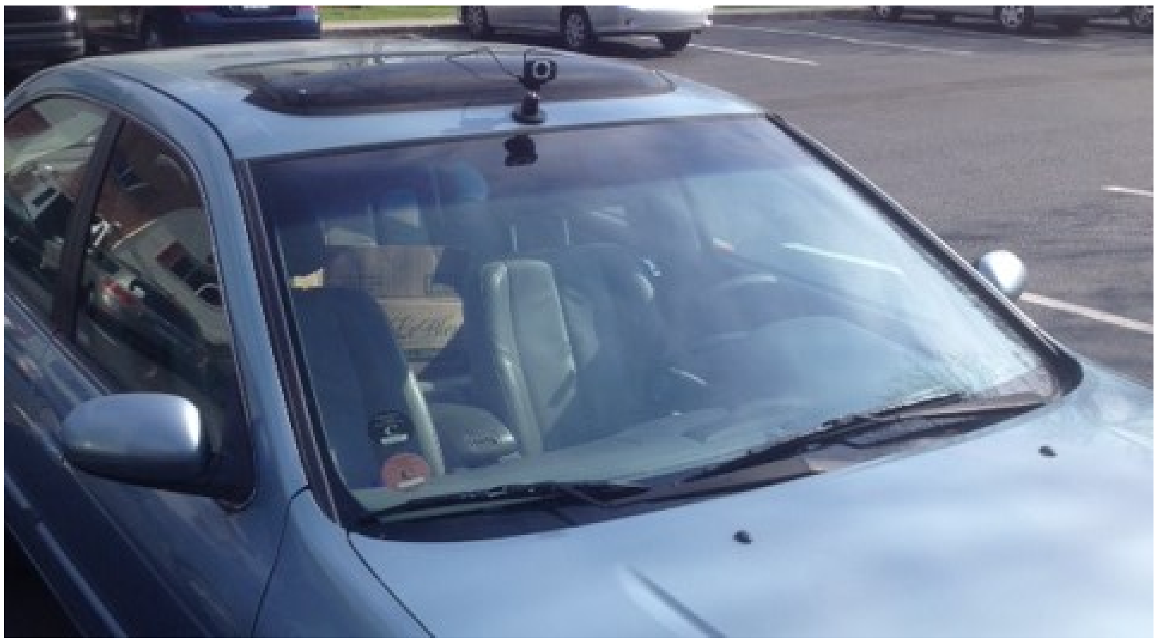


Figure 39: Front view of Camera on Vehicle.

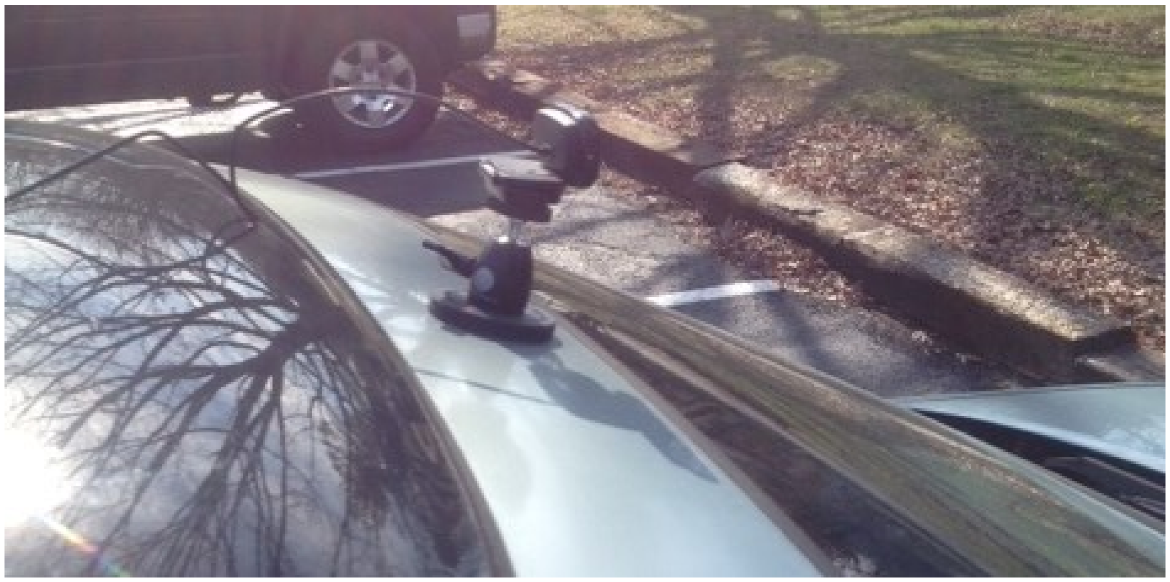


Figure 40: Side view of Camera on Vehicle.

5.1.2 Data Variation

Data sets were generated from recorded video taken from the camera rig. The camera angle was slightly varied between data sets to include different amounts of the vehicle hood in the image. Data sets were taken at different times of day, specifically 10:00 AM, 2:00 PM and 5:00 PM. The weather at 10:00 AM was partly cloudy, and the weather at 2:00 and 5:00 PM was clear. Data sets of overcast or rainy days were not considered due to the unprotected nature of the camera rig.

Different road conditions and types were included in the data set. The two major road types considered were single lane, multi-lane/interstate. Sets of each of these road types were generated. Different conditions on each road type was also considered. Solid and dashed line roads were included in the sets, as well as roads that had faded vs. bright lane markings. Shadows on roadways were present in the data sets, as well as roads without any shadows. Example images of all of these road types can be seen in Appendix A.

5.1.3 Data Down-Selection and Image Set Creation

The data collection resulted in a set of videos at a constant 30 frames per second. These videos included different road types and were taken at different times of day. The videos included redundant frames, as well as unusable frames. The unusable frames included frames that were very over or under exposed, such as from inside a tunnel, or when the sun caused glare on the camera lens. Frames where the car was not moving were also not considered.

After these frames were removed from consideration, 10 representative data sets were formed from the rest of the data. Each data set was chosen based on its uniqueness relative to the other data sets according to the road and lighting types listed above in Section 5.1.2. Each data set combined a road type, line type and lighting. Not all combinations were considered as not all combinations exist in the testing area, for example, interstate type roads in the area do not have yellow lines and thus that combination could not be considered.

To create representative data sets from these videos, a program was written in Matlab that allowed the user to choose a section of a video and extract frames from that video as images. The images were cropped to a uniform size that removed the vehicle and excess sky from the image. The goal was to reduce all images to a uniform size that maximized the percentage of the image that included roadway. These images were then saved to a folder for use in testing IRIS. The 10 created data sets were of differing numbers of images, ranging from 50 to 600 for a total of 1400 images.

5.2 Accuracy Experiment

The accuracy of the lane model generated by IRIS is an important part of the evaluation of the algorithm. To test this, a hand drawn lane model is compared against the algorithm-generated lane model for every test image. The hand drawn models were created using a custom C++ program based on OpenCV and Qt. Figure 41 shows an example of a hand drawn lane model.

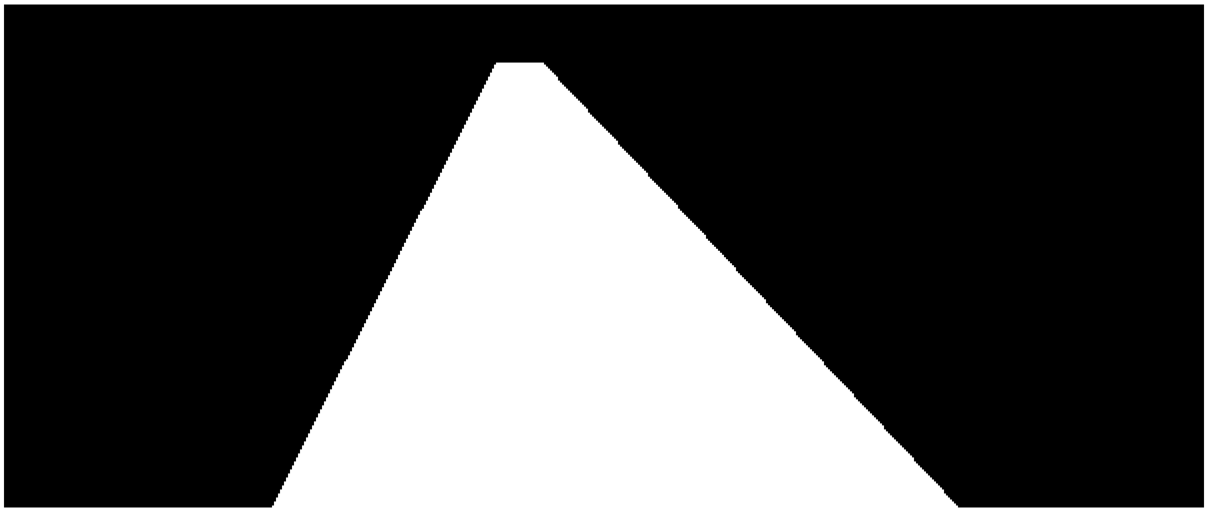


Figure 41: Hand Drawn Lane Ground Truth

5.2.1 Pixel-Based Accuracy Metrics

Accuracy is a loosely defined term in classification problems. A more specific measure is necessary to better represent the results of the experiment. For this particular classification problem, "lane" vs. "not lane," a confusion matrix is useful. A confusion matrix is an N x N matrix where N is the number of possible classes. In this particular classification experiment, $N = 2$. Table 2 shows a generic 2 x 2, or binary, confusion matrix, along with generated metrics.

There are several metrics that can be derived from a confusion matrix that better characterize the performance of the classifier. The binary confusion matrix contains four elements, true positives, true negatives, false positives and false negatives. True positives (TP) are defined as elements where the true class was positive and the predicted class was positive. True negatives (TN) are the opposite, where the true class was negative and the predicted class was negative. False positives (FP) and negatives (FN) predict a class opposite the true class.

Using these four elements, classification metrics can be derived. Precision and Recall, two classic metrics in classification problems, are defined by the following equations [95].

$$Recall = \frac{TP}{TP + FN} \tag{30}$$

$$Precision = \frac{TP}{TP + FP} \tag{31}$$

Recall compares the true positives classifications to the sum of all positive examples in the data set. This is essentially a metric of what percentage of all true examples were classified as true. Precision is a comparison of true positives to all classified positive examples, giving a metric of how relevant the returned positive results are. One definition for accuracy born from this matrix is as follows

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{32}$$

which compares the number of correct classifications to the total number of examples.

Two other useful metrics that can be generated are the "inverse" of the Precision and Recall metrics. True Negative Rate measures the recall in a negative example sense, and Negative Precision measures the precision in a negative example sense. The usefulness of these metrics is dependent on the application and the cost of false positives vs. false negatives. All of the discussed metrics are shown below in Table 2.

Table 2: Confusion Matrix and Generated Metrics

	True Condition	False Condition	
True Prediction	True Positive (TP)	False Positive (FP)	$Precision = \frac{TP}{TP + FP}$
False Prediction	False Negative (FN)	True Negative (TN)	$Neg.Precision = \frac{TN}{TN + FN}$
	$Recall = \frac{TP}{TP + FP}$	$True\ Neg.Rate = \frac{TN}{TN + FP}$	$Acc. = \frac{TP + TN}{TP + TN + FP + FN}$

In the context of this experiment the classification is defined on the pixel level as "lane" vs. "not lane." A confusion matrix and associated metrics will be generated for each image, and then can be averaged across the tested images to generate metrics for the entire data set.

5.2.2 Shape-Based Accuracy Metrics

The above pixel-based metrics capture the classification performance of the algorithm, but do not directly indicate the similarity of shape between the ground truth lane model and the predicted lane model. Thus the second measure of accuracy is the difference in shape of the predicted lane mask vs. the ground truth lane mask. Shape similarity metrics are a current area of research with new measures still being developed [96],[97],[98]. There are two major categories of shape similarity metrics: Point based metrics and global metrics.

Point based metrics focus on discretizing a shape into a series of points, and computing a distance between corresponding points using some function. Some examples of functions are pure Euclidian distance, Fréchet distance, where the points are considered as a parametric set on the same parameter and distance is computed point to point, and keypoint matching techniques using image properties or descriptors, like SIFT [99].

Global metrics compute properties of the entire image and compare them. Image moments are the most popular global metric, and the standard set of image moments is the Hu moment set [100],[101]. Hu moments are scale, rotation and translation invariant properties of an image generated from a combination of certain weighted averages of the image intensity. The Hu moments are used for shape matching in many applications, including human action classification studies [100].

Image moments can be written in terms of Reimann integrals as

$$\mathrm{m}_{p,q} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q \rho(x,y)\, dx\, dy \qquad where\ p,q = 0,1,2 \ldots \tag{33}$$

where $m_{p,q}$ is the image moment of order p,q, (x,y) is the position in the image and ρ is the intensity value at that point in the image. When applied to a discrete image, the integrals are finite and take place across the image boundaries. Central image moments are simply an image moment with a zero mean. This can be seen below.

$$\eta_{p,q} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x-\bar{x})^p (y-\bar{y})^q \rho(x,y)\, d(x-\bar{x})\, d(y-\bar{y}) \tag{34}$$

The set of central image moments, denoted by $\eta_{p,q}$, is translation invariant, according to Hu [101]. The set of 7 Hu moments is a linear combination of different order central moments. These Hu moments are combined to be rotation and scale invariant, as central moments are already translation invariant.

The 7 Hu moments, shown below in Equation (35), exist on different ranges from each other, and must be normalized for a Euclidian distant calculation to be meaningful. Each moment is moved to zero mean and normalized based on its range which moves the moments to exist in a range between -1 to 1. Computing the Euclidian distance between two moment vectors will result in a number that exists in a range between 0 and 14. This number can be scaled to exist between 0 and 1, and averaged across all the results. Equations *(35)* through *(38)* show this normalization operation.

$$Hu = \langle \psi_0, \psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6 \rangle \tag{35}$$

$$\Psi_n = \frac{\psi_n - \overline{\psi_n}}{{\psi_n}^{max} - {\psi_n}^{min}} \Psi_n \qquad \Psi_n \in [-1,1] \tag{36}$$

$$\Delta\Psi_n = \sqrt{(\Psi_{n,1} - \Psi_{n,2})^2} \qquad \Delta\Psi_n \in [0,2] \tag{37}$$

$$\Delta Hu = \frac{\sum_{n=0}^{6} \Delta\Psi_n}{14} \qquad \Delta Hu \in [0,1] \tag{38}$$

The similarity between Hu moments of two binary images will show the similarity in mask shape in a scale, rotation and translation invariant manner. The accuracy of the lane shape is variant in scale, rotation and translation, but because the shifts in scale, rotation and translation will affect the pixel based accuracy metric, the Hu moment metric can be used to determine if the predicted lane mask is a similar shape to the ground truth lane mask. Combining these metrics together will give a better understanding of the accuracy of the predicted lane model.

5.2.3 Experimental Implementation

To apply these metrics to the IRIS algorithm, tests must be implemented in software. The ground truth lane mask, shown above in Figure 41, is a binary image with the lane as foreground. IRIS generates a binary mask over the predicted lane locations. These masks can be compared to generate the confusion matrix derived metrics, as well as the Hu moments.

The OpenCV library includes a set of bitwise matrix operations for operating on binary images. When the bitwise AND operation is performed with the ground truth mask and the predicted mask, the pixels where they agree will remain foreground, creating a true positive result. To create the false positive and false negative result, two steps are needed. First, an XOR operation between the ground truth and predicted masks, and then the AND operation between the ground truth mask for the false negatives, or the predicted mask for the false positives. To get the true negative result, the NOT operation is used on the result of an OR operation between the ground truth and the predicted masks. The logical operations used to make these results are shown below in Table 3.

Table 3: Bitwise Operations for Confusion Matrix

Ground Truth = GD, Predicted = PR	
True Positive = GD AND PR	False Positive = PR AND (GD XOR PR)
False Negative = GD AND (GD XOR PR)	True Negative = NOT(GD OR PR)

After the bitwise operations, the pixels in each of the four images will be summed to give four scalars. These are the values for each quadrant in the confusion matrix. Simple addition and division of these scalars will result in the desired metrics, as shown above in Table 2.

To implement the shape-based metrics, the OpenCV functions to create Hu moments were used on both mask images. Following that, the normalized Euclidian distance between the two vectors was computed and the resulting value is the accuracy of the shape metric. The shape metric was scaled to exist between 0 and 1, with 1 being perfect accuracy.

Both of the accuracy values, the pixel-based and the shape based, are averaged across all tested images and output to the user. All of the intermediate and unused metrics are also saved to a file for future use.

5.2.4 Expected Results of Accuracy Experiment

The accuracy experiment focuses on two metrics, pixel based accuracy and shape based accuracy. The confusion matrix based accuracy metric exists on a range from 0 to 1, therefore a percentage will exist from 0% to 100%. It is claimed in Wang et al. that the CHEVP algorithm has an average accuracy of 95%, but in Heij et al. it is claimed that the algorithm only exhibits 86% accuracy. Based on these two papers, is expected that the IRIS algorithm will exhibit a pixel based accuracy of at least 86% at the optimal set of BSnake convergence parameters [23],[65].

The shape based metric is the normalized difference between Hu moments for the two lane mask images, one ground truth and one predicted. The normalized Euclidian distance between the Hu moments exists on a range from 0 to 1, and can be inverted such that 1 is a perfect match and 0 is the worst possible match. This metric can be thought of as a “percent similar”, for example vector A is 85% similar to vector B. The expected value of the shape accuracy is not known, the metric falls within a 0-100% range, so it is reasonable to assume similar accuracy to the pixel based metric.

5.3 Speed Experiment

The second claim to evaluate is the speed of IRIS compared to full image k-means segmentation. It is expected that the IRIS algorithm will be faster than segmenting the entire image. It is also worthwhile to note that even without a speed increase from full image segmentation, IRIS results in a lane model that can be adapted for autonomous driving purposes. It does not invalidate the usefulness of the system if it is not faster, but it would require continued research and efficient programming to implement IRIS on a system.

5.3.1 Experimental Considerations

The timing of an algorithm on a computer is not a trivial task. The typical operating system (OS) is a multi-tasking environment that has application priority and is interrupt driven [102]. Because of the variability inherent in the OS, timing operations’ accuracy can be compromised by a higher priority interrupt occurring at a critical point in the timing operation. Apart from the interrupt driven nature of the OS, modern CPU architectures have allowed thread-based software development, where the CPU can perform multiple tasks at the same time. This also causes timing inconsistencies, as tasks can be moved between CPU cores, resulting in a delay.

In order to characterize the typical timing operations in a multi-tasking OS, a small program was created that measured the time between the start and finish of a certain millisecond delay. This measurement was taken 10 times, and the difference between the true delay and the measured delay was stored. The average and standard deviation of these stored values was calculated. This process was repeated 10 times for each time delay value and the averages and standard deviations were averaged across those 10 iterations. This resulted in an average mean and an average standard deviation for each considered delay interval calculated from 100 total samples. The average mean can be considered the bias error, and the average standard deviation can be considered the measurement error.

The delay is performed using the Windows API Sleep(int ms) function. This function has an input argument, *ms*, of a number of milliseconds to pause thread execution. The Sleep function has been shown to be accurate to within 7 milliseconds when run in a single thread at delay times at or above 200 milliseconds [103]. The IRIS algorithm is expected to converge in greater than 200 milliseconds on average, therefore the Sleep function is a viable error characterization technique. The bias error and measurement error for the delay time measurement is shown below in Figure 42. Based on the method of data collection outlined above, the averaging of the Sleep function measurements should reduce the noise present in the measurement.

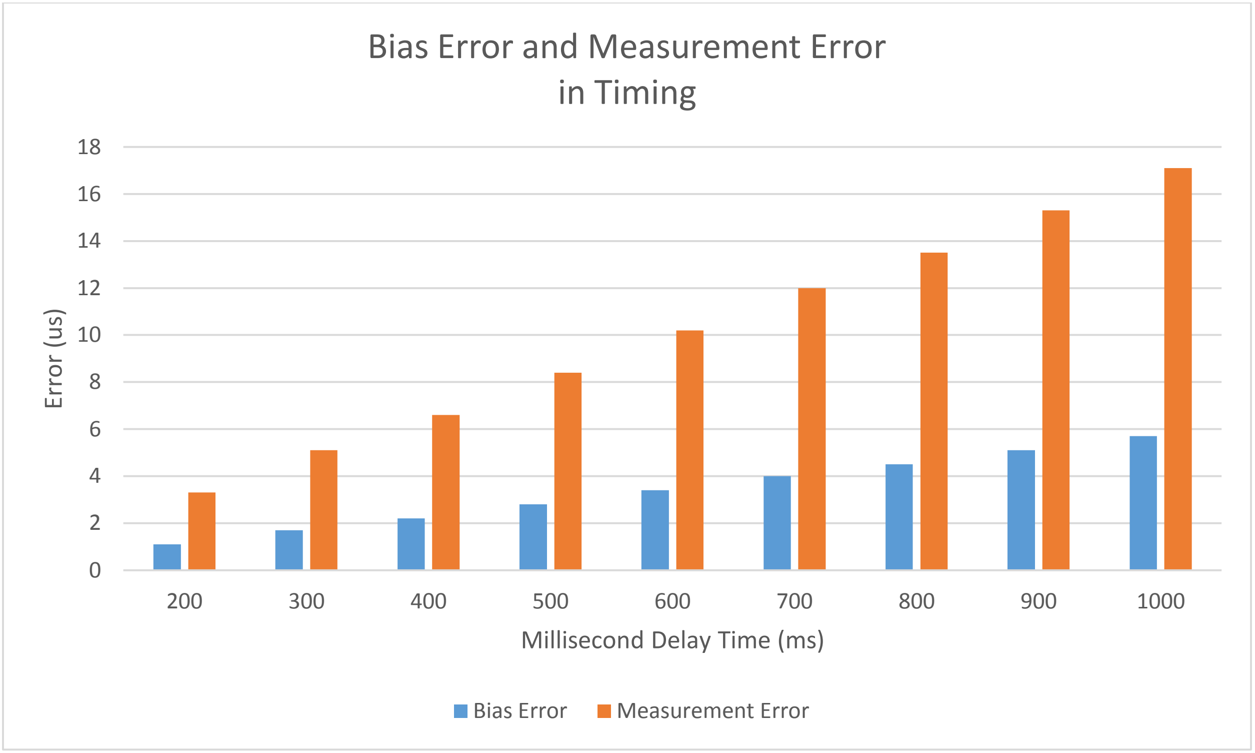


Figure 42: Bias and Measurement Error in Timing

Figure 42 shows a linearly increasing trend in both bias error and measurement error. To help identify this trend, the same timing data is plotted below in Figure 43 as a percent of delay time, instead of in microseconds. It can be seen in Figure 43 that the error as a percent of delay is constant. This will allow extrapolation of bias and measurement error to times of greater than 1 second based on a percent of the measured time. Using this information, as well as the above information that the Sleep function is accurate to within 7 milliseconds in the desired range, the error in the timing system can be characterized.

The error present in timing is within the tens of microseconds, even up to a 1 second delay. It can then be assumed that, based on the data in Figure 43, the bias error is 0.0006% of the measured time in milliseconds, and the measurement error is 0.0017% of the measured time in milliseconds.

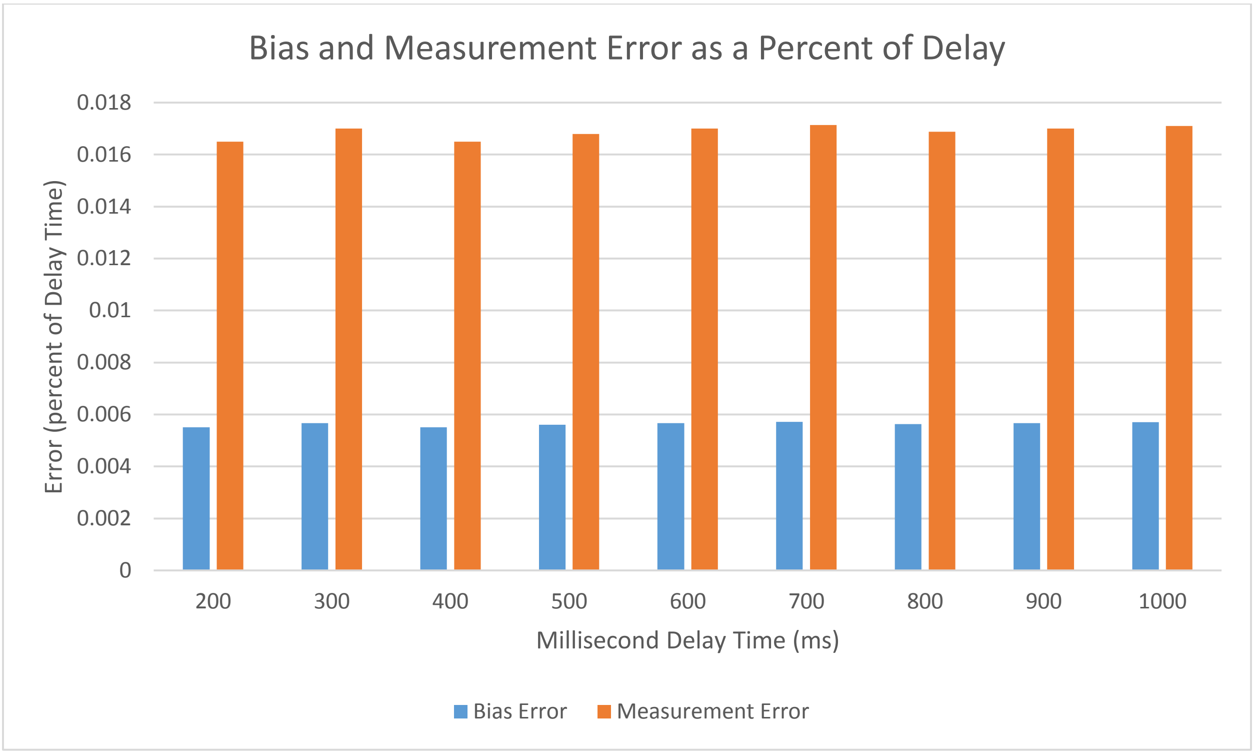


Figure 43: Bias and Measurement Error as Percent of Delay

5.3.2 Experimental Implementation

To test the speed of IRIS, the STL chrono library was used to get a microsecond level accuracy based on the clock in the test computer [104]. Because the speed of an algorithm is dependent on computer specifications, the same computer with the same configuration was used to test IRIS and full image segmentation. The test computer is an Apple MacBook Pro Retina 2014 model with an Intel Core i7 2.30 GHz processor, 16.00 GB of RAM, a NVIDIA GT750M graphics card, and a 512 GB solid state hard drive. The operating system is Windows 7 64-bit Service Pack 1, run through the Boot Camp dual boot utility. The Windows 7 hard drive partition is 300 GB with approximately 200 GB of free space.

The chrono library uses two main constructs to measure time. A Clock object is used to count time and report the current time, and a Period object is used to hold a certain time point reported by the Clock. The Clock object accuracy is dependent on the system clock of the computer. The Clock object was tested using the method described above in Experimental Considerations. The results of that error testing were used to validate the results from the algorithm speed testing.

The test is run on a set of consecutive images. First, each image is segmented using the IRIS algorithm at a certain number of clusters with a basic feature set. Second, a traditional k-means algorithm is used to segment for the same features and number of clusters as in the IRIS implementation. The segmentation operations are timed and the result is placed into a data set. The convergence of the B-Snake algorithm in IRIS is noted for each image. The converged and not-converged image times are separated into two data sets, which are also averaged. This test is repeated for different numbers of segmentation clusters.

5.3.3 Expected Results

It is expected that IRIS will be faster than traditional k-means segmentation. The effect of number of clusters on the speed of operation is unknown and will be tested. It is also expected that images starting from a converged previous state will be the fastest in operation, based on the analysis in IRIS Lane Tracking with Active Contours.

5.3.4 Parameter Variation

The accuracy and timing tests as described above are for a single data set with a single set of BSnake convergence parameters, *QThreshold* and *kThreshold*. The tests are run many times with different combinations of data sets, *QThreshold* and *kThreshold*. As the BSnake parameters change, the number of converged images will change, as well as the accuracy of the lane model. Determining the behavior of the algorithm with changing parameters is critical to characterizing its accuracy. It is expected that for a parameter set with higher values, more images will converge, as the thresholds are larger.

The effect on the accuracy testing of this higher convergence is not known. If the algorithm has more converged lane models, it could mean that the accuracy is very high, but it could also mean that the algorithm has converged to an incorrect local minima and is continuing to converge to that minima over successive images. Charting the accuracy of the testing at different parameter sets will allow characterization of this behavior and provide a better set of threshold values for typical operation.

The effect on the timing testing of the higher convergence should be faster operation. If more images converge before the 250 BSnake iterations have completed, fewer operations will be performed, resulting in a lower time of operation. In regards to timing, a higher threshold is always better, up until every image converges in 1 BSnake iteration. This is most likely a poor parameter set for accuracy, but would result in the fastest possible algorithm operation. Timing testing over several parameter sets should validate the faster operation claims.

6 Results

This section outlines the results generated from the performed testing. First, a full parameter sweep was performed on the accuracy test using a single data set to determine if the IRIS's accuracy was independent of either BSnake parameter. IRIS's accuracy was found to be independent of the *kThreshold* parameter for all tested values. This was important because it simplified further testing and analysis of IRIS.

Based on the *kThreshold* independence finding from the parameter sweep test, all further testing was performed with a constant *kThreshold*. Accuracy testing was performed on all data sets, comprised of 1433 images to establish the accuracy of the lane detection and tracking, so that IRIS could be compared to other implementations of the CHEVP algorithm. The test was performed several times on each data set with different *QThreshold* values to determine the relationship between accuracy and convergence, as well as the optimal choice for *QThreshold*. This was done to establish a set of parameters that would be suggested for operation on a vehicle.

Timing testing was performed to test the claim that IRIS is faster than performing k-means segmentation on an entire roadway image. The test was performed on all data sets for different number of k-means clusters. This finding is important to the viability and feasibility of implementing IRIS in a real-time setting on a vehicle. If IRIS is slower than k-means, it is not viable for implementation. If it is faster, but not fast enough to operate in real-time, it is not feasible for implementation at this time on a computer similar to the test computer.

6.1.1 Parameter Independence

To determine if the algorithm is independent of either of the BSnake parameters, accuracy testing was performed on a single data set over a range of *QThreshold* and *kThreshold* values. The resulting data is shown below in Figure 44. The data was swept over parameter values suggested in Wang et al. and Heij et al. [23], [65]. *QThreshold* was swept from 0.2 to 0.9 in the inner loop and *kThreshold* was swept from 0.01 to 0.08. The *QThreshold* and *kThreshold* parameter sets are plotted on the x-axis of Figure 44, with the *kThreshold* values listed. The *QThreshold* values sweep between every labeled value.

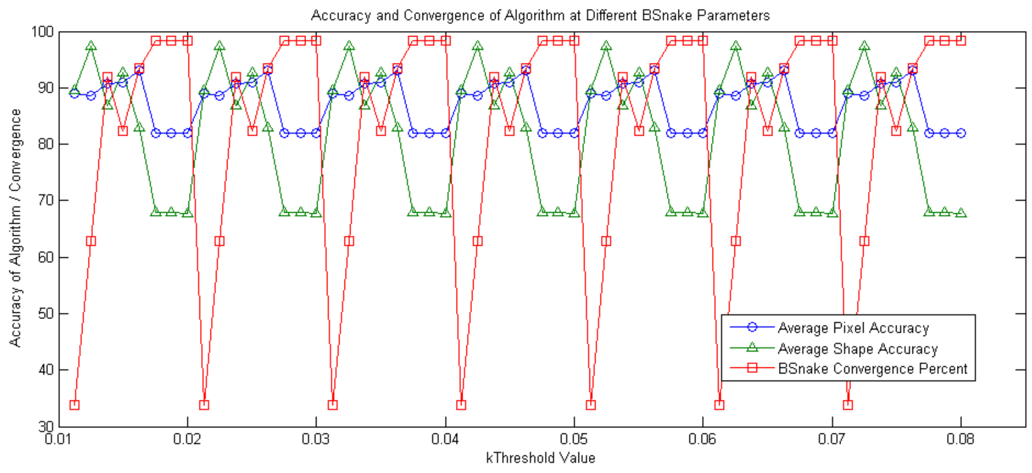


Figure 44: Accuracy Test Data Parameter Sweep

This data seems to be periodic in nature over every 8 data points, indicating that the algorithm is independent of the *kThreshold* value. This assertion can be tested by separating an 8 sample period of the data set into a small vector and convolving it across the data set. If the response every 8 samples is the same, the data is periodic in nature. Shown below in Figure 45, Figure 46and Figure 47are the convolution vector and response for each data vector in Figure 44. The convolution response shows that the data is exactly periodic over the 8 sample set. This proves the independence of the algorithm accuracy and convergence from the parameter *kThreshold*. Because of this independence, all future testing was performed with a constant arbitrary *kThreshold* of 0.04.

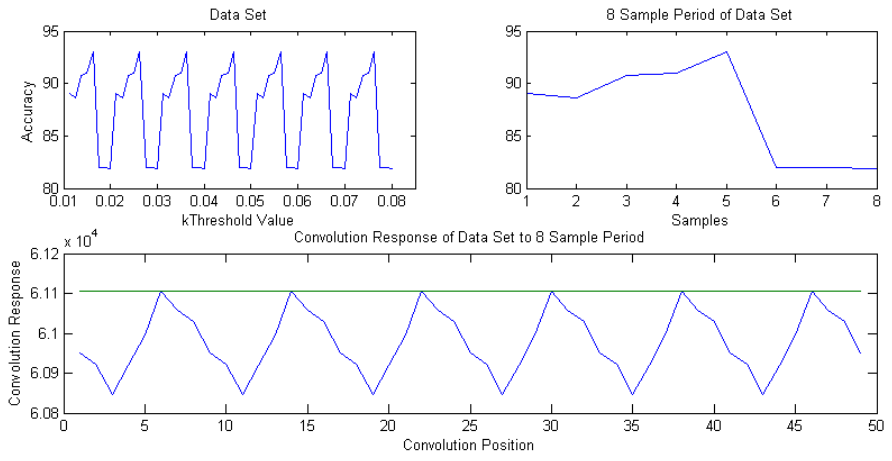


Figure 45: Pixel Accuracy Convolution

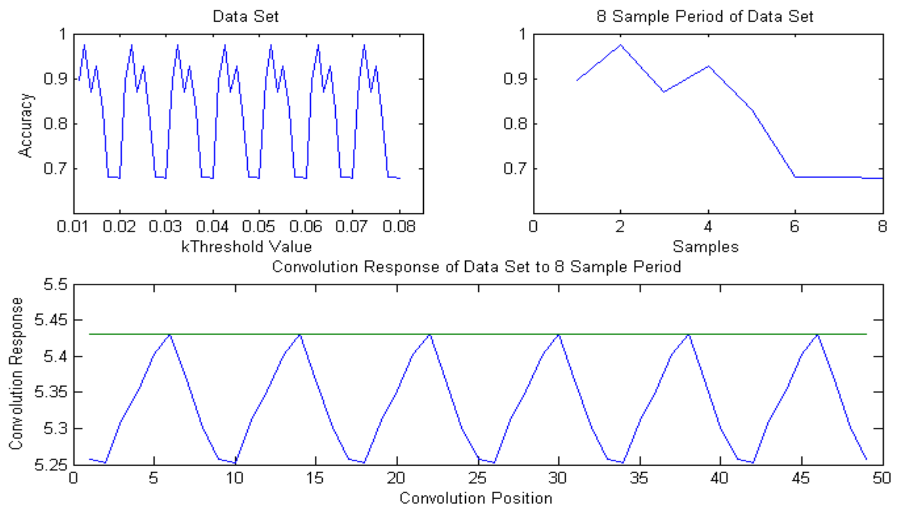


Figure 46: Shape Accuracy Convolution

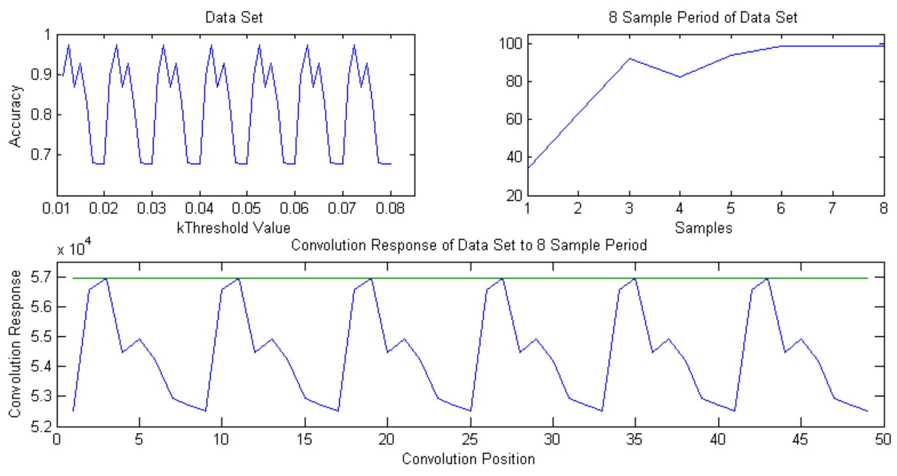


Figure 47: BSnake Convergence Convolution

6.1.2 Timing Baseline Testing

The timing testing for the algorithm requires the creation of a baseline time based on the segmentation of an entire image into different number of clusters. Before performing the test on the full data set, a representative sample was processed using only the standard k-means segmentation algorithm 6 times to determine if the k-means algorithm time was dependent on the processed image. The data was averaged and the standard deviation computed for each cluster number and the results are presented below in Figure 48.

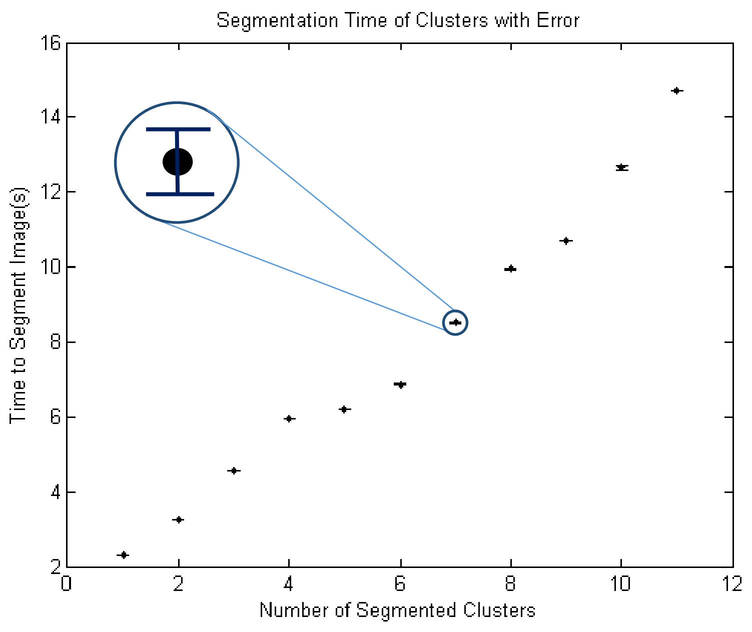


Figure 48: Average Segmentation Time

The error bars are difficult to see in Figure 48, and this is due to the standard deviation of the timing data being 3 orders of magnitude less than the mean of the timing data. A data point is blown up in Figure 48 to clarify the arrangement and type of error indication used. Based upon how small the standard deviation of the k-means timing data is compared to the mean, it can be asserted that the k-means timing does not depend on the image and is constant for a given number of clusters. This both creates a baseline time and reduces the amount of computation necessary for testing each image.

6.1.3 Full Accuracy Testing

After the algorithm independence to *kThreshold* was established in the previous section, full accuracy testing on all data sets was performed with constant *kThreshold* of 0.04 and a swept *QThreshold* from 0.2 to 0.8. The pixel and shape accuracies, as well as the BSnake convergence data were recorded for each parameter set in each image set. The tables containing all of the accuracy data is in Appendix B. Figure 49shows a graphical representation of the accuracy data charted at each value of *QThreshold*.

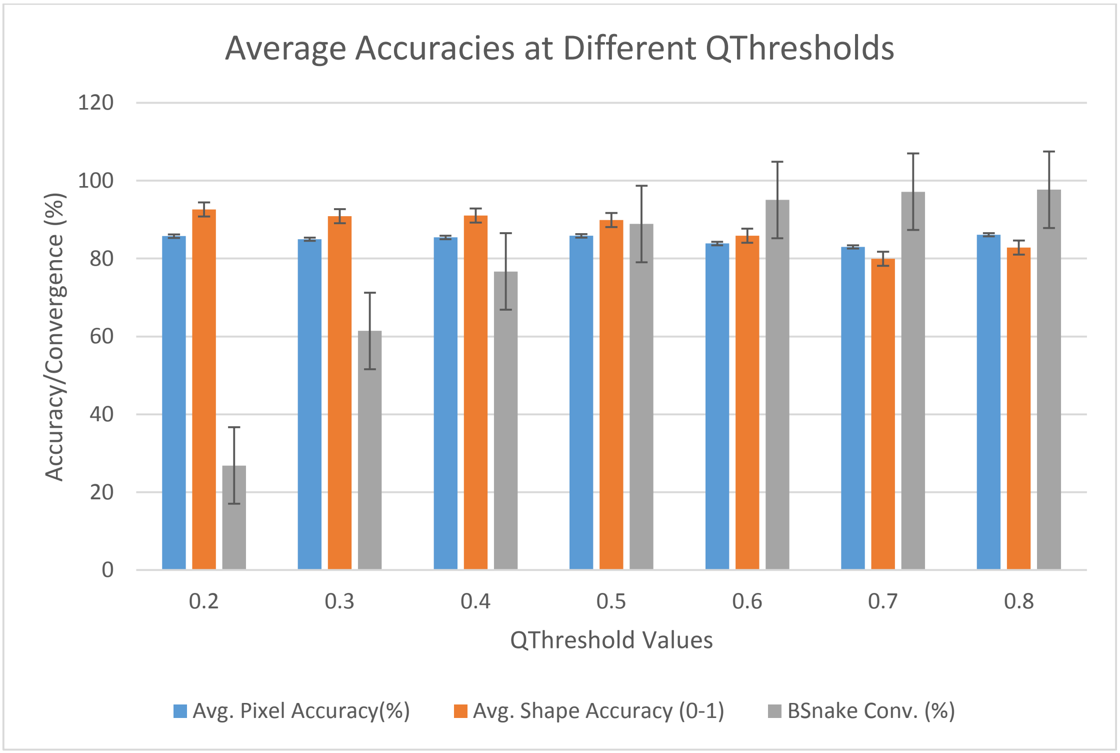


Figure 49: Average Accuracy Means at Different QThreshold Values

As expected, the BSnake convergence increases with the *QThreshold* value. The pixel accuracy value is approximately constant across all *QThreshold* values and the shape accuracy value decreases with increasing *QThreshold*. These values are averaged on multiple tests across all data sets. This information is useful for generating an optimal value for *QThreshold*. Qualitatively, *QThreshold* values of 0.4 and 0.5 appear to be optimal in accuracy as well as amount of converged images. This claim can be evaluated by observing the sum of the three charted values for each *QThreshold* value. The solution where the pixel accuracy, shape accuracy and B-Snake convergence are maximized is the optimal solution. A simple way to determine if there is a clear solution among the selected *QThreshold* values is to sum the accuracies and convergence data, as they all exist on a range from 0 to 100. The sums of the data are shown below in Figure 50.

Figure 50 shows that the sum of accuracies and convergence for a *QThreshold* value of 0.5 is the best choice out of the tested values. This value provides the highest combined accuracy and convergence, indicating that choosing 0.5 for the *QThreshold* value will result in algorithm performance that has high accuracy as well as converges a higher percentage of the time, which should result in a faster algorithm. The speed results will be discussed in a later section.

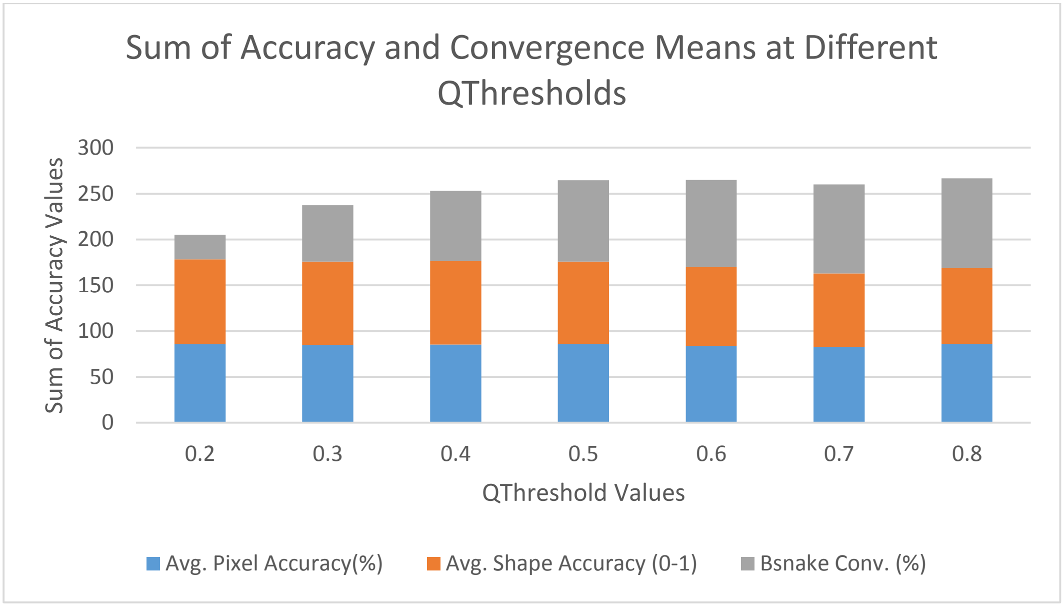


Figure 50: Sums of Accuracy and Convergence

For this particular *QThreshold* value the pixel accuracy is 85.8%, the shape accuracy is 89.9% and the B-Snake algorithm converges 88.9% of the time. For the tested *QThreshold* values, this is optimal, but further exploration was required to determine if there was a more accurate optimal value. *QThreshold* was swept from 0.45 to 0.55 by 0.01 to find a good approximation of the optimal value to within ±0.005. Figure 51 shows averages of three measured values with the standard error plotted.

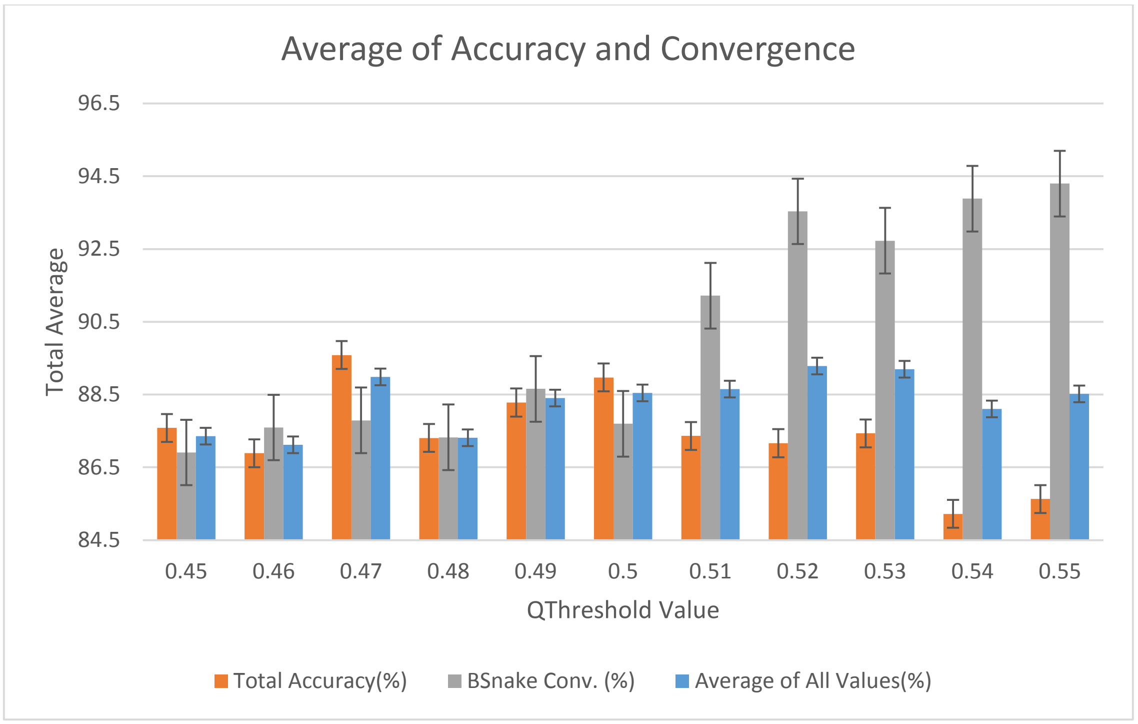


Figure 51: Average of Accuracy and Convergence Data

The two accuracy values are averaged and shown in the Total Accuracy series. The B-Snake convergence is shown in the BSnake Conv. Series and the Average of All Values series is the average of the convergence and the accuracy metrics.

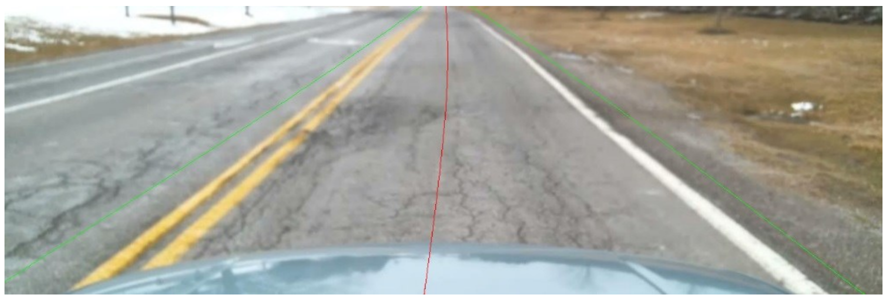
Based on the data shown in Figure 51, there are two choices for the optimal point based on the desired operation of the algorithm. If the highest accuracy is the goal, *QThreshold* should be set to 0.47. If highest B-Snake convergence is the goal, 0.52 is the value that maximizes the convergence without a significant drop in accuracy. Either of these choices will result in acceptable performance of the algorithm, but the tradeoff between speed and accuracy is left to the specific implementation of IRIS. Table 4shows the accuracy and convergence values for the two chosen *QThreshold* parameters.

Table 4: Chosen QThresholds and Associated Accuracies

QThreshold	Average Pixel Accuracy (%)	Average Shape Accuracy (%)	B-Snake Convergence (%)
0.47	86.75 ± 1	92.42 ± 1.5	87.79 ± 2
0.52	85.64 ± 1	88.69 ± 1.5	93.54 ± 2

Comparing the experimental results to the expected accuracy results from the Experiment section, the pixel accuracy values fall in the expected range claimed by Heij et al. The shape accuracies are within 2.5 of 90%, and fall in the expected acceptable range based on the baseline testing in the Experiment section. The B-Snake convergence percentage indicates that >87% of the images converge before 250 iterations with either of the chosen values. The difference in speed between the two QThreshold values will be discussed in Timing Results section below.

All of the analysis up to this point has discussed accuracy in terms of a global average. These averages are useful for characterization, but it is difficult to visualize the difference between a 92% shape accurate image and an 85% shape accurate image. To help with this disconnect between data and real images, some examples of different accuracy segmentations are shown below in Figure 52through Figure 55. These are typical examples of image accuracies that might be helpful in placing meaning with two accuracy metrics.

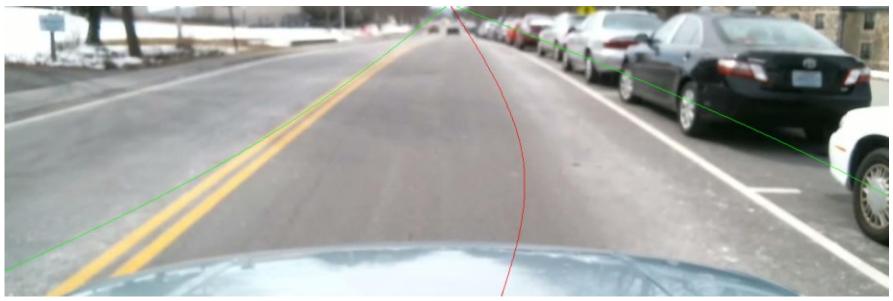


Shape Accuracy: 99%

Pixel Accuracy: 91%

Figure 52: Image with Shape and Pixel Acc. (high)

Figure 52 shows a very accurate lane prediction, where the shape accuracy is nearly a perfect match, and the pixel accuracy is very high. The shape accuracy is high based on the correct lane centerline and mostly correct lane boundaries. The pixel accuracy is slightly lower due to the consistently wide predicted lane boundaries.

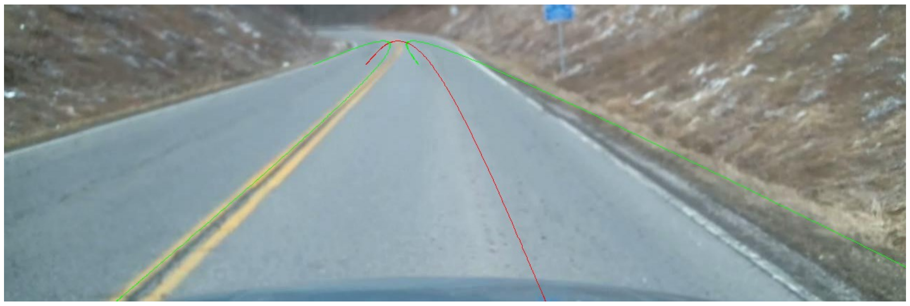


Shape Accuracy: 85%

Pixel Accuracy: 92%

Figure 53: Image with Shape and Pixel Acc. (medium)

Figure 53shows a medium accuracy lane estimate. The pixel accuracy is quite high because the estimated lane covers the entire lane area and is not much wider than the actual lane. The shape accuracy is lower due to the skewed nature of the predicted lane centerline. This is potentially caused by the gradients present in the vehicles on the right side of the road. This is an example of an acceptable but not ideal case, where the lane model is technically incorrect, but in a way that does not impair the segmentation of an appropriate section of the lane. The centerline of the lane is approximately correct, and does not leave the lane boundary. The predicted lane edges encompass the entire true lane and will result in the true lane being segmented, along with some extra boundary information.

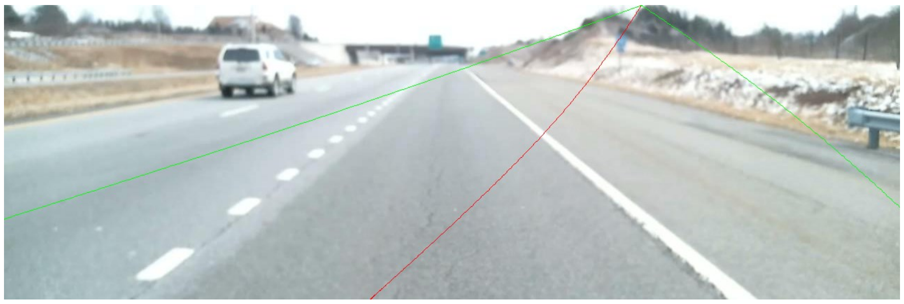


Shape Accuracy: 84%

Pixel Accuracy: 93%

Figure 54: Image with Shape and Pixel Acc. (medium)

Figure 54 shows another acceptable but not ideal case, where the lane curvature is correct until the top of the lane. This prediction correctly identifies the lane centerline until the uppermost part of the image, and is very accurate in determining the width of the lane. A vehicle using this information to drive would be able to stay within the actual lane by following the predicted lane until the next image was taken and analyzed with no ill effects.



Shape Accuracy: 68%

Pixel Accuracy: 79%

Figure 55: Image with Shape and Pixel Acc. (low)

Figure 55 shows an example of a failure case, where the lane is incorrectly predicted. The inaccuracy is reflected in the two accuracy metrics. Note that for this failure case, the pixel based accuracy is still nearly 80%. This is due to the large majority of the actual lane being covered by the incorrect predicted lane model. The shape accuracy is low compared to the values shown above, but it is not much below 70%. This, again, is due to the gross similarities between the actual lane and the predicted model. Both have points at the top of a triangular structure. Both have a centroid in the bottom third of the image. The differences are great when viewed qualitatively, but are not as high when viewed numerically. However, these metrics still give a good separation between a highly accurate lane predictions, medium accuracy lane predictions and low accuracy lane predictions.

To understand the effect that the image type has on the accuracy of IRIS, the data sets were considered individually. Data sets were labeled with their road type, as listed above in Section 5.1.2. Sets with the same parameters were then averaged for each of the road type parameters. This data is presented in Figure 56.

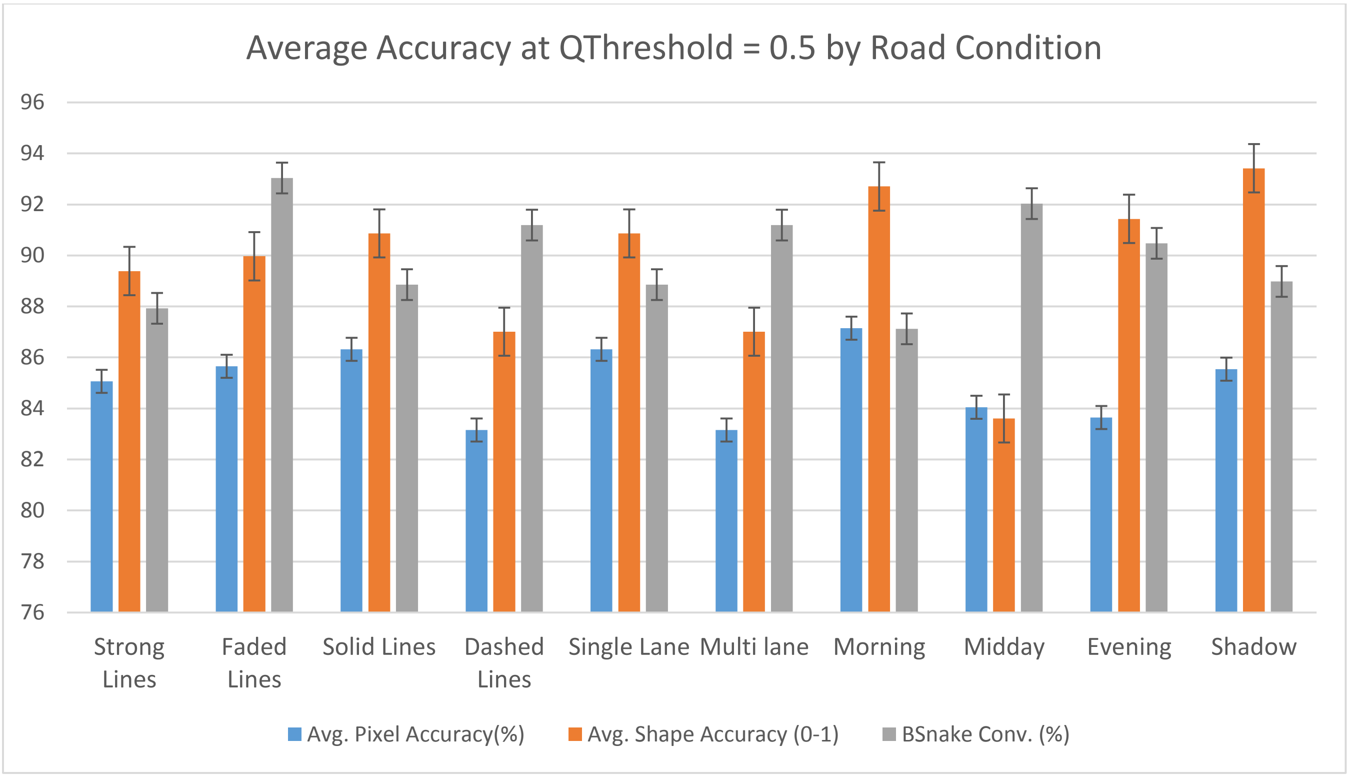


Figure 56: Accuracy by Road Type

The results here show that the algorithm is similarly accurate when predicting roads using images with strong vs. faded lines. The error of the measurement is greater the difference between the two. It is worthwhile to note that faded lines do not include lines that have faded to the point of being no longer visible. Solid lines road predictions are clearly more accurate than dashed line predictions, which is expected based on the dependence on lane lines for IRIS to operate. This also causes single lane roads to be more accurate than multi-lane roads, as lanes are separated by dashed lines. Time of day variations produce a varied response, and based on the type of data collected at different times may be a result of the other parameters, but it is interesting to note that shadows seem to have a negligible impact on the performance of IRIS. This is a positive result, as CHEVP was initially designed to be robust to shadows in the roadway.

The average accuracies for each data set are presented below in Figure 57 to give an understanding of how different combinations of parameters change the accuracy of the data. Table 5 shows the characteristics of each data set for reference.

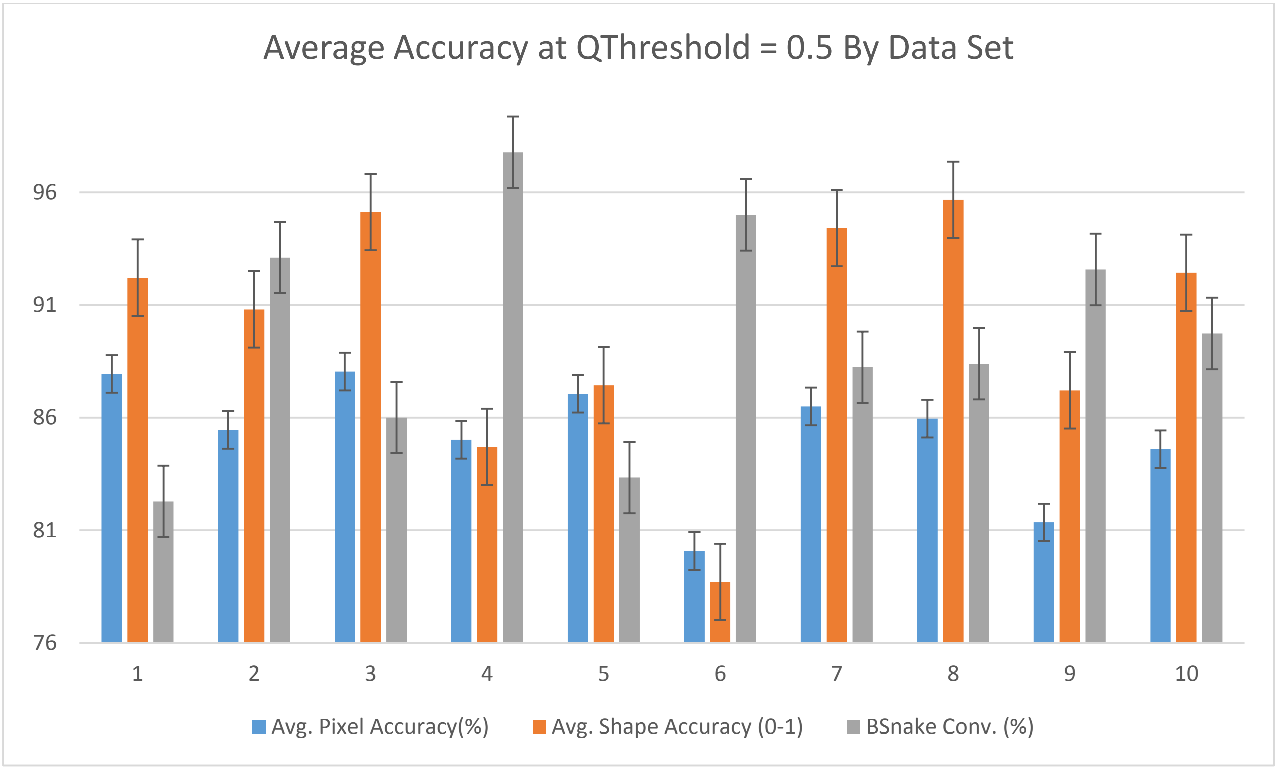


Figure 57: Average Accuracy by Data Set

Table 5: Data Set Labels and Characteristics

Data Set Number	Data Set Characteristics
1	Strong, Solid, Single Lane, Morning
2	Faded, Solid, Single Lane, Morning
3	Strong, Dashed, Multi-lane, Morning
4	Faded, Solid, Single Lane, Midday
5	Strong, Solid, Single Lane, Midday
6	Strong, Dashed, Multi-lane, Midday
7	Faded, Solid, Single Lane, Shadow
8	Strong, Solid, Single Lane, Evening
9	Strong, Dashed, Multi-lane, Evening
10	Strong, Solid, Single Lane, Shadow

IRIS maintains an accuracy in both metrics above 80% for all but one data set, which was taken on an interstate highway. An example of an image from this data set was shown previously in Figure 55. The road parameter dependent results can provide an estimate of the trust a future navigation system could have in IRIS. If it is known that the UGV is travelling on an interstate highway, the lane prediction in IRIS can be assigned a greater degree of uncertainty, and the vehicle could rely on other sensors to augment IRIS.

Based on the metrics shown above in Table 4, the average accuracy of IRIS falls in the range between the medium images shown above and the high accuracy image. This is near or greater than the expected results from the Experiment Section.

6.1.4 Full Timing Testing

To validate the claim that IRIS is faster than segmenting the entire image, the timing experiment discussed in the earlier section was performed across all the data sets at different values for *QThreshold* with different numbers of k-means clusters. The timing data was averaged across the data sets, and then all data set average times with the same *QThreshold* and number of k-means clusters were averaged. This data is presented below in Figure 58.

Figure 58 shows that the k-means segmentation time across the entire image is greater than the IRIS segmentation time for cluster numbers greater than 5, and is greater than converged IRIS segmentation times for all cluster numbers greater than 4. A realistic number of clusters to properly segment the road is greater than 6 (shown above in Figure 36), therefore it can be stated that IRIS, for all practical applications, is faster than pure k-means segmentation. Figure 59shows the averages of all the converged and not converged timing results with the k-means timing results for a clearer picture of the data.

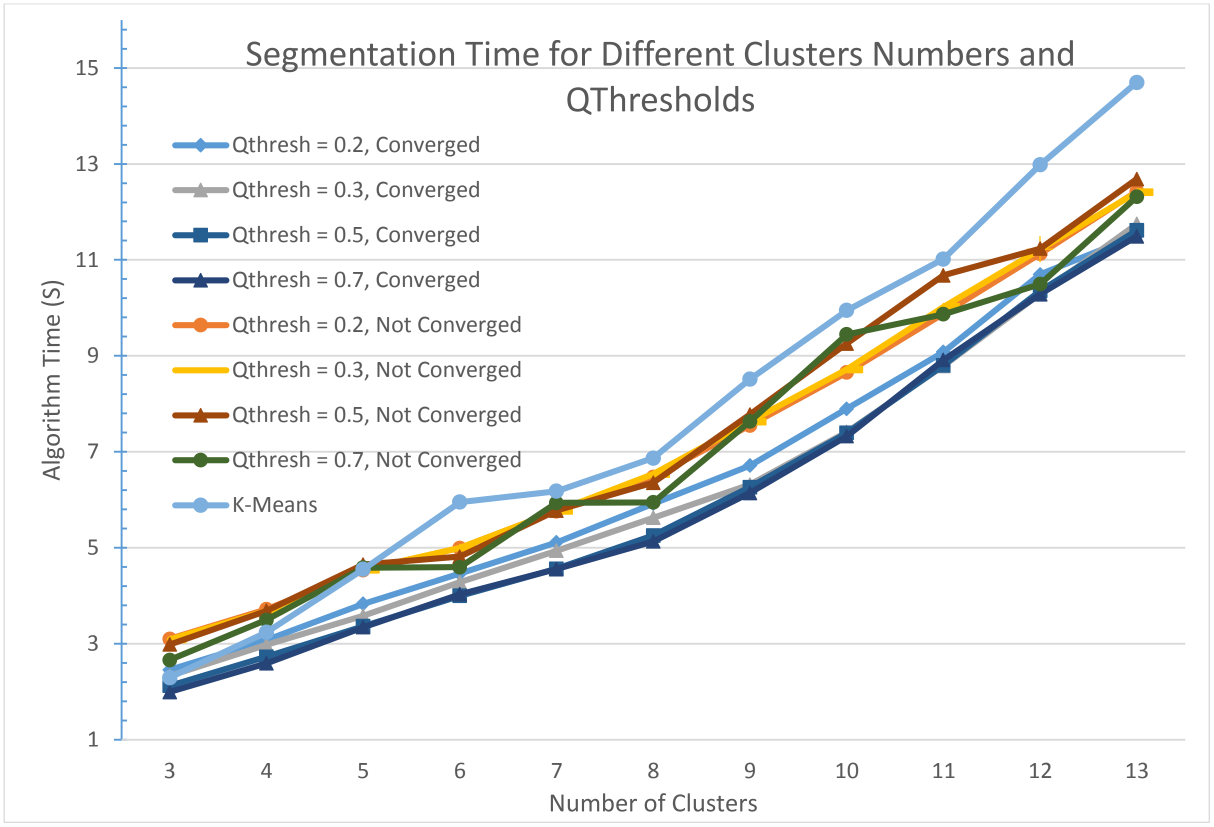


Figure 58: Segmentation Timing Data

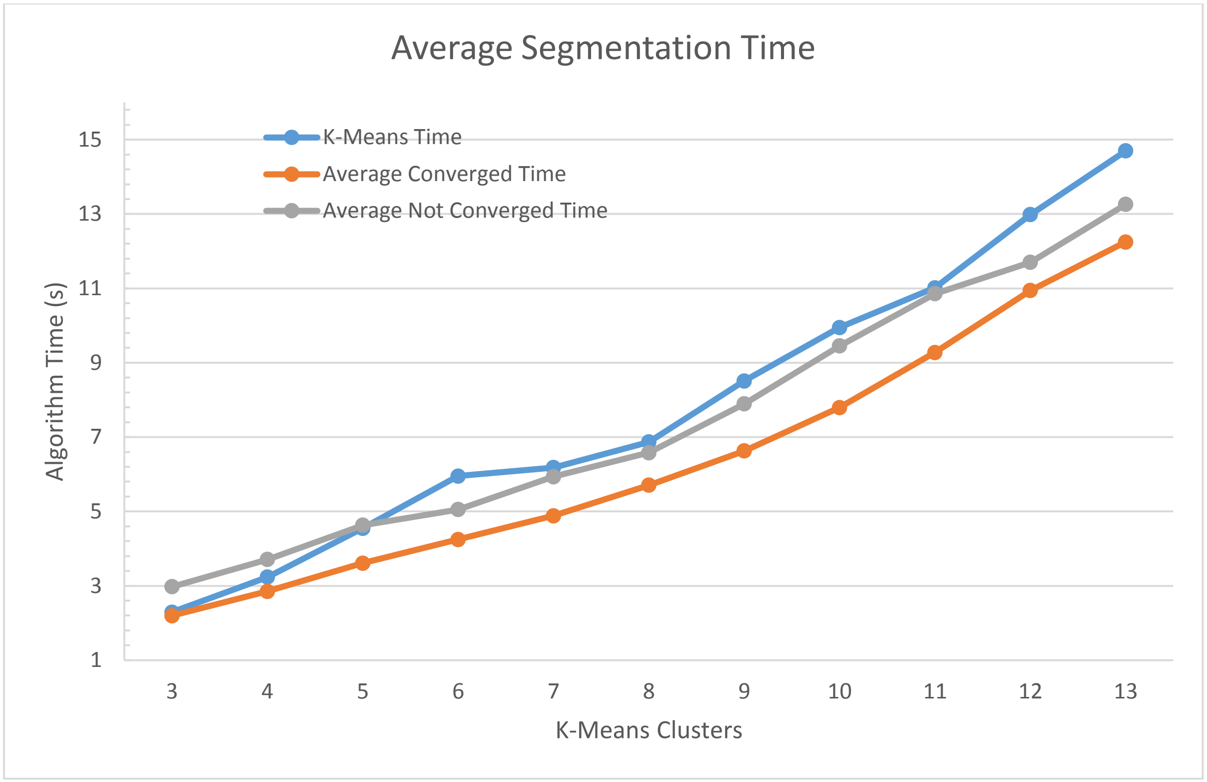


Figure 59: Average Segmentation Timing Data

This statement that IRIS is faster than standard k-means segmentation is clearly true for all the investigated points, but the behavior is unknown once the number of clusters is increased past the highest tested point. To visualize the behavior of each trend, the data points are presented in a split fashion in Figure 60. This allows the slope of each data series to be viewed more easily.

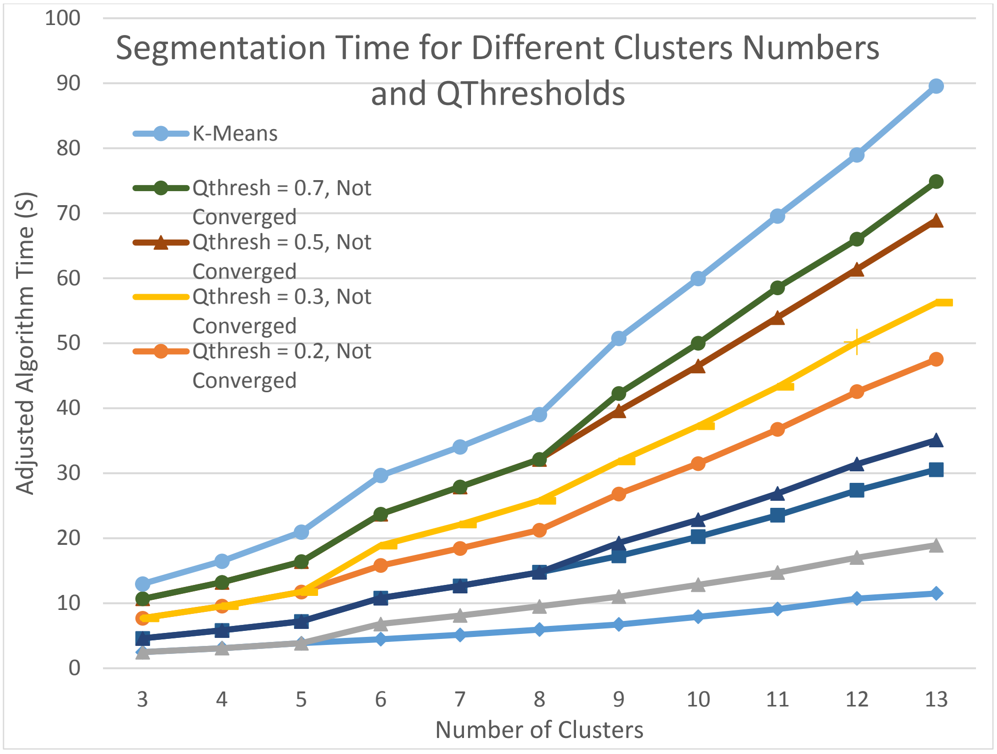


Figure 60: Split-Line Chart of Segmentation Time Data

Figure 60 shows that the slopes of the converged data sets are less than the slopes of the not converged data sets, which are still less than the slope of the k-means data set. The trend shown above indicates that the segmentation time will continue to increase with the number of clusters, and this result verifies the claim that for all cluster numbers greater than 5, IRIS outperforms pure k-means segmentation in terms of computational time.

To determine what impact input image size has on the performance of IRIS, timing testing was performed on 3 different image sizes. These sizes were normalized to the original image size and resulted in a linear scaling factor, which indicates how much each side of the image was scaled. Scaling factors of 0.5, 1 and 1.5 were tested and the results are shown in Figure 61. These scaling factors correspond to a decrease or increase in total pixels of

$$newPixelNum = oldPixelNum * ScalingFactor^2$$

where *ScalingFactor* is one of the three previously described scaling factors.

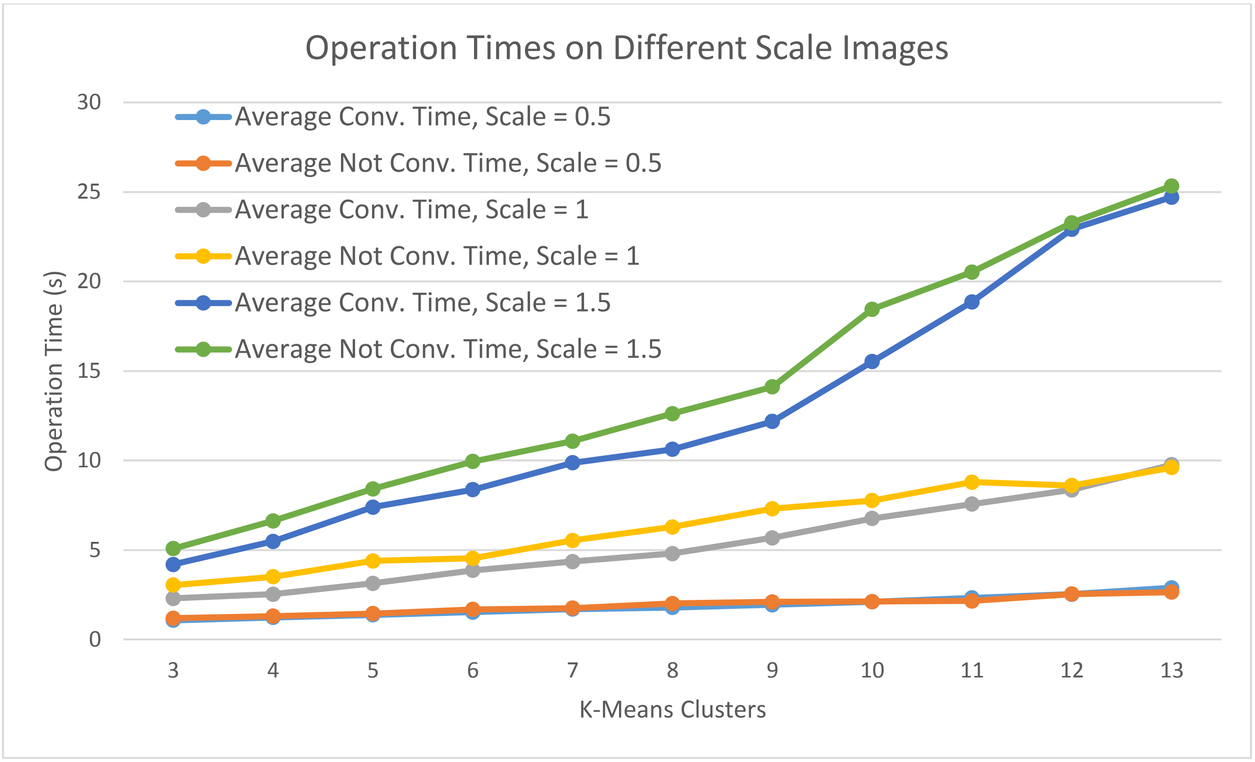


Figure 61: IRIS Speed at Different Images Scales

The data indicates that the processing time increases as the number of pixels increase. To determine the relationship between the scaling factor and the increase in time, the same data is presented again in Figure 62, but normalized to the times found with a scale factor of 1.

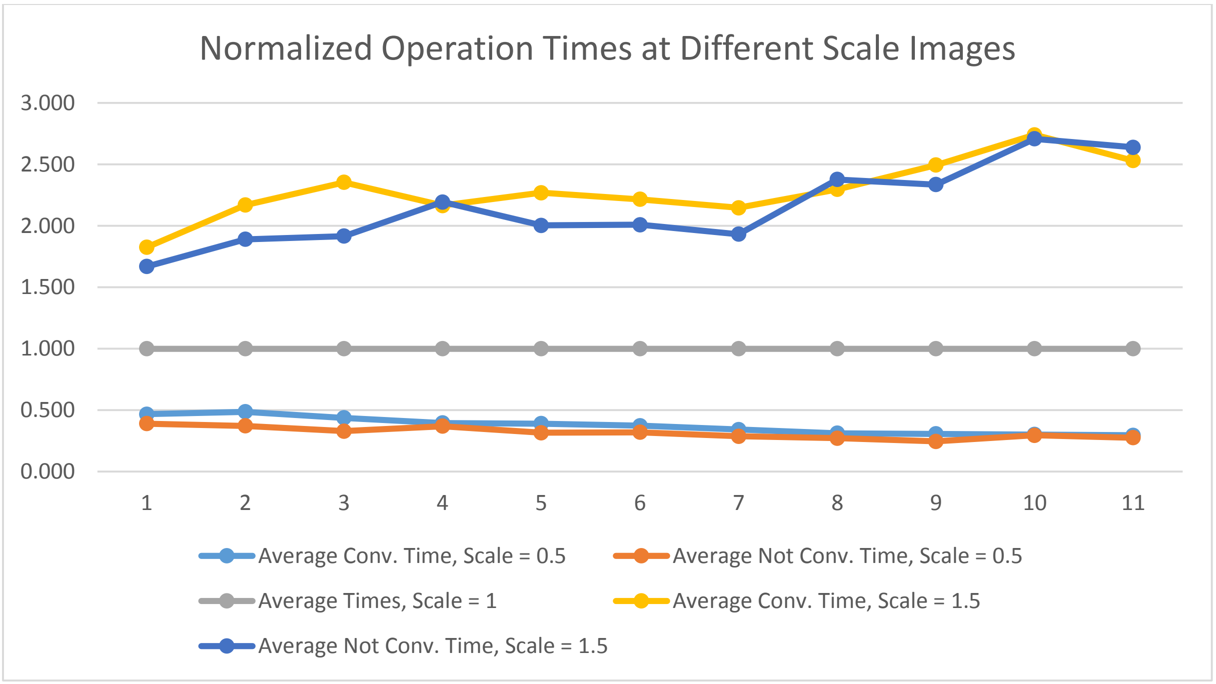


Figure 62: Normalized IRIS Speed at Different Image Scales

Figure 62 shows that the operation time not only increases with scale factor, but also increases in slope with scale factor. To characterize the type of slope increase, the data was plotted again on a lin-log scale, to determine if the increase is exponential. Figure 63 shows that plot and based on the linear nature of the data in that scale, the function that best represents the increase in operation time is near exponential. Exponential trend-lines are plotted on Figure 63, with their respective equations next to the label for the data set on the plot.

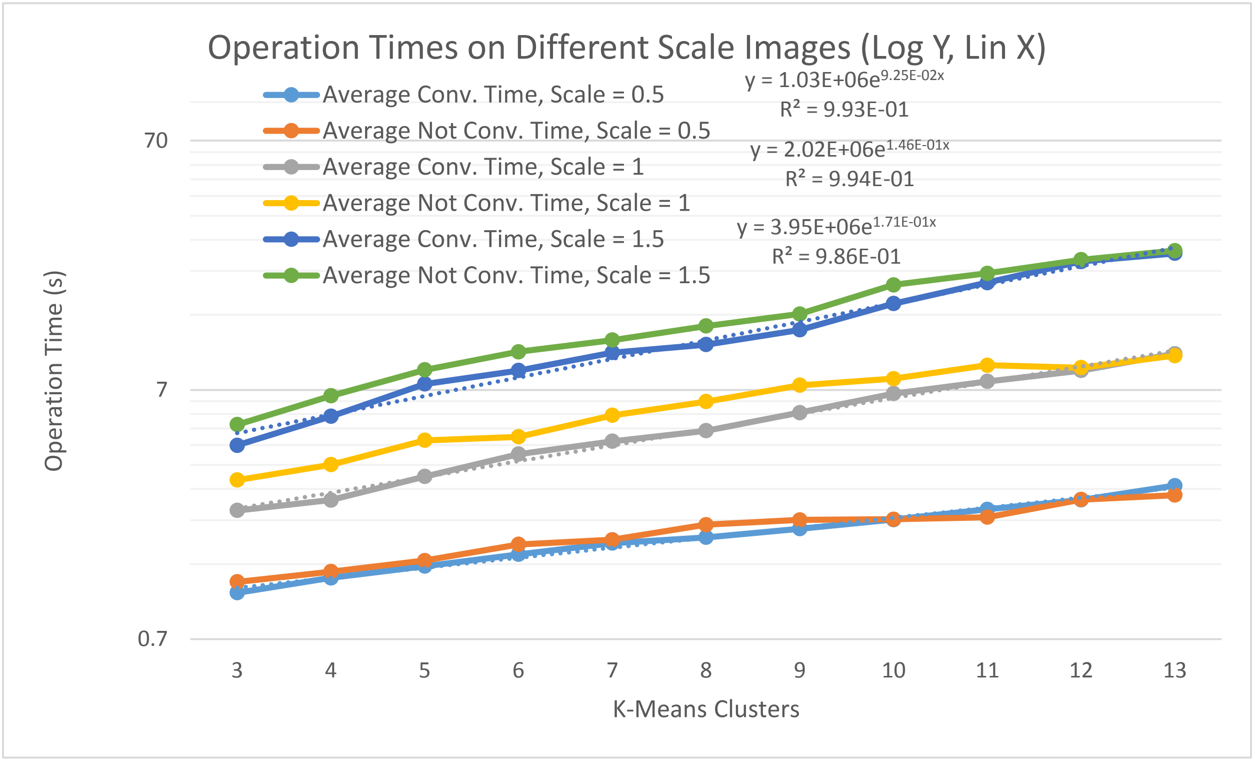


Figure 63: Lin-Log Plot of IRIS Operation Time for Scaled Images

Each of these trend-line equations has an R^2 value greater than 0.985, and can be considered a good representation of the increase in processing time. Higher cluster number times can be inferred from this equation, assuming that nature of the increase remains constant above the tested cluster numbers.

IRIS was found to be faster than full image segmentation at cluster numbers greater than five, and thus met its speed performance goal. The operation time increased exponentially as a function of clusters, and increased in slope as a function of the image scale.

7 Conclusions

The results of the experiments detailed above validate the accuracy and speed claims of IRIS. The algorithm exhibits greater than 86% accuracy in the pixel and shape metrics at one of the chosen optimal parameter sets, and is faster in operation than standard k-means segmentation techniques. Also, as can be seen in Figure 52 through Figure 54, the algorithm can create a qualitatively good lane prediction. There are failure cases, but the large majority of lane predictions in the tested data sets are within acceptable tolerances. Based on the acceptance criteria in the Method section, IRIS is an acceptable solution to the intelligent lane segmentation problem.

7.1 Current Problems with IRIS

There are still problems inherent in the algorithm. While the operation time is faster than standard k-means segmentation, it is still too slow for practical use in vehicles. Dividing an image into 6 clusters will take, at the best case, 4 seconds on the test computer. This is not acceptable for use in a vehicle moving at road speeds. The algorithm can converge to local minima that do not accurately represent the lane if the B-Snake parameter thresholds are too high. The exact numerical value for "too high" is unknown, and most likely varies with the environment as well as the algorithm parameter settings.

The current road model used in this work is overly simplistic for many road types. While the algorithm performs well on defined roadways with few intersections, the underlying road model assumptions are violated in many typical roadway situations. It can be argued that the majority of the roadway fits within this model, and that the assumptions are valid most of the time, but the behavior of the vehicle using IRIS to navigate is currently undefined and more than likely inappropriate for any areas that fail the model assumptions. Undefined behavior for a vehicle on a roadway is not acceptable

The current IRIS lane model does not account for changing width in lanes. This causes a model violation when lanes are added or removed from a roadway, when there are exits on an interstate highway, when parking or fire lanes exist on the side of the road, and in other similar situations. The current lane model does not account for changing elevation on the roadway, and needs flat roads to be accurate. The model does not account for traffic circles or intersections of any kind, and can fail if the roadway lines are not well defined.

IRIS has not been tested in crowded road environments, such as stop-and-go traffic on an interstate highway or gridlock-style traffic in a city. The vanishing points of the roadway in the image could be completely obscured by the vehicles in the image and cause the algorithm to fail. IRIS has not been tested at night, and the low light environment might cause poor lane identification with the lack of visible features, especially as the distance from the camera to the feature increases.

Based on all of these issues, a new lane model that can account for all of these cases is needed. This model, however, would be very complex and need to differentiate many distinct roadway situations from each other reliably. This is a very difficult model to construct and would require a great deal of work and testing.

7.2 Future Work

With all of these problems and limitations in mind, IRIS is not appropriate for vehicular use at this time. It does, however, represent a step forward in the unmanned ground vehicle perception research field. Using a single camera to identify a lane and its 3D structure in an image is not a trivial task, and has not been implemented in a satisfactory manner on a vehicle. IRIS proves the concept is feasible. Future work for IRIS is to prove that it is viable.

Another important result of this work is the concept of region of interest segmentation. It seems a trivial answer that segmenting a smaller portion of the image would result in a faster algorithm operation time, but this result means that as long as the lane finding algorithm is fast enough, it can be “plugged in” place of the lane finding portion of IRIS to take advantage of this finding.

Areas of future improvement to IRIS are many, and it is yet to be determined if the strategy employed by IRIS is the most viable for single camera roadway systems, but improvements can be made that will move the algorithm towards viability. The slow speed of processing can be addressed in different ways. Simply moving to a faster computer will result in faster computational time and, theoretically, a computer can exist that will allow IRIS as it currently exists to operate at speeds useful for roadway driving, though the size, cost and power requirements of such a computer may be prohibitive for use on a vehicle. With the current speed of IRIS on at test computer with a 2.6 GHz processor, it can be extrapolated the speed of a computer required for IRIS to operate at 10 Hz. IRIS operating with 5 clusters operates in approximately 4 seconds per image. To increase this to 10 Hz, a processor operating at approximately 105 GHz is required.

Another way that the speed of IRIS can be improved is through GPU acceleration. Any parallelizable operation can be shifted to the GPU on a computer and will usually result in a speed increase if the operation is performed correctly, according to the GPU architecture and the particular algorithm structure. The Hough Line transform, the Horizon Detection, the BSnake MMSE algorithm and the K-Means algorithm are all areas where GPU acceleration could be used to increase the speed of IRIS. The K-means algorithm operation time could be as much as 25 times faster with the proper implementation on a GPU [105]. This also means that at some point of parallelization, k-means will be faster than IRIS. Because IRIS is designed for processors with few numbers of cores it will operate faster than k-means on a CPU in most cases, as shown in Section 6.1.4, but k-means will outperform IRIS in speed when shifted onto a GPU. This creates a parallel design stream for detection algorithms based on the available hardware. GPU based algorithms may not be compatible with CPU based algorithms and vice versa.

The road model could be improved to handle elevation changes in the roadway by allowing each image section to have an individual vanishing point instead of a common horizon. This would allow the road model to change in elevation, but would require more accuracy in the detection of each section vanishing point. The overall vanishing point detection in the current implementation of IRIS is accurate on a high percentage of images, but the section vanishing point exhibits a higher degree of error, as can be seen in Figure 27 and Figure 28. Work will need to be done to improve the accuracy of the section vanishing points, but if that can be done, a 3D lane model that includes elevation changes is easily implemented.

Also, inclusion of other electromagnetic bands could potentially improve the lane detection. A longwave infrared camera, which is responsive to electromagnetic wavelengths from 8000 nm to 15000 nm, can detect radiated heat. Using the temperature information about the road versus the grass or sidewalk border could improve road detection [106]. Other wavelength ranges could provide other benefits, but at a cost of registering the images to the visual camera image. More research would be needed to determine the benefit of these wavelengths, as well as the associated costs of their inclusion.

Research in the optimal field of view for roadway imaging is also needed. IRIS was tested with a uniform field of view for consistency and repeatability, the widest field of view possible on the camera. Based on experience with IRIS, a goal of future implementations should be to increase the amount of pixels in the input image that are roadway. If more pixels are roadway, then it will be easier to detect edges and lines on the roadway and the vanishing point calculations should be more accurate. This could be performed through zoom or a decreased field of view, or by increasing the height of the camera above the roadway. To fully determine the gains of a change like this, more research is required.

IRIS has shown promise as a potential solution for the problem of single camera lane detection. It is not a complete solution in its current state, but represents an important step in the development process for a viable single camera perception system for an autonomous road vehicle.

Bibliography

[1] Velodyne Acoustics INC, “Velodyne’s HDL-64E: A High Definition LIDAR Sensor for 3-D Applications,” 2007.

[2] P. Currier, J. Hurdus, and A. Bacha, “The VictorTango Architecture for Autonomous Navigation in the DARPA Urban Challenge,” *Exp. from …*, 2012.

[3] D. W. Gage, “Special Issue on Unmanned Ground Vehicles UGV HISTORY 101 : A Brief History of Unmanned Ground Vehicle (UGV) Development Efforts,” vol. 13, no. 3, 1995.

[4] H. Moravec, *The Stanford cart and the CMU rover*. 1990.

[5] H. Moravec, “The CMU Rover.,” *AAAI*, 1982.

[6] J. Lowrie, “The autonomous land vehicle (ALV) preliminary road-following demonstration,” *1985 …*, 1985.

[7] D. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” 1989.

[8] M. Xie and L. Trassoudaine, “Active and intelligent sensing of road obstacles: Application to the European Eureka-PROMETHEUS Project,” *Comput. Vision, …*, 1993.

[9] S. Denasi, “Real-time system for road following and obstacle detection,” *Photonics …*, 1994.

[10] A. Broggi, “Parallel and local feature extraction: a real-time approach to road boundary detection,” *Image Process. IEEE Trans.*, 1995.

[11] A. Bacha and C. Reinholtz, “The DARPA Grand Challenge: overview of the Virginia Tech vehicle and experience,” *… 7th Int. …*, 2004.

[12] “Multi-Mission Autonomous Vehicle Undergoes Two-Week Experimentation by Marines.” [Online]. Available: http://www.torcrobotics.com/blog/multi-mission-autonomous-vehicle-undergoes-two-week-experimentation-by-marines/. [Accessed: 18-Jul-2013].

[13] “Ground Unmanned Support Surrogate.” [Online]. Available: http://www.torcrobotics.com/case-studies/ground-unmanned-support-surrogate. [Accessed: 18-Jul-2013].

[14] “NASA - What are passive and active sensors?” Brian Dunbar.

[15] B. Schwarz, “Mapping the world in 3D,” *Nat. Photonics*, vol. 4, no. July, pp. 3–4, 2010.

[16] Velodyne Acoustics INC, “High Definition LiDAR ™ HDL-32E Datasheet,” 2011.

[17] W. Shawlee, “The Dark and Mysterious World of Night Vision :,” *Avion. News*, no. March, 2008.

[18] K. Konolige and J. Augenbraun, “A low-cost laser distance sensor,” *… , 2008. ICRA 2008. …*, 2008.

[19] T. Deyle, “Velodyne HDL-64E Laser Rangefinder (LIDAR) Pseudo-Disassembled | Hizook.” [Online]. Available: http://www.hizook.com/blog/2009/01/04/velodyne-hdl-64e-laser-rangefinder-lidar-pseudo-disassembled. [Accessed: 22-Jul-2013].

[20] Microsoft, “Microsoft LifeCam HD-3000 Datasheet,” p. 3000, 2012.

[21] “JAI-NIR Products.” [Online]. Available: http://www.jai.com/en/products/nearinfrared. [Accessed: 31-Jan-2014].

[22] R. Kuc and B. Barshan, “Navigating vehicles through an unstructured environment with sonar,” *Robot. Autom. 1989. …*, 1989.

[23] Y. Wang, E. Teoh, and D. Shen, “Lane detection and tracking using B-Snake,” *Image Vis. Comput.*, 2004.

[24] P. Sturm, S. Ramalingam, and J. Tardif, “Camera models and fundamental concepts used in geometric computer vision,” *… Trends® Comput. …*, 2011.

[25] J. Ponce, D. Forsyth, and E. Willow, *Computer vision: a modern approach*. Boston: Prentice Hall, 2011.

[26] L. Shapiro and G. Stockman, *Computer Vision*, 1st ed. 2001, p. 617.

[27] E. Fossum, “Active pixel sensors: are CCDs dinosaurs?,” *… Symp. Electron. Imaging …*, 1993.

[28] W. Boyle and G. Smith, “Charge-coupled devices-a new approach to mis device structures,” *Spectrum, IEEE*, 1971.

[29] G. Healey and R. Kondepudy, “Radiometric CCD camera calibration and noise estimation,” *Pattern Anal. Mach. …*, 1994.

[30] E. Fossum, “CMOS image sensors: Electronic camera-on-a-chip,” *Electron Devices, IEEE Trans.*, 1997.

[31] S. Decker and D. McGrath, “A 256× 256 CMOS imaging array with wide dynamic range pixels and column-parallel digital output,” *Solid-State Circuits, …*, 1998.

[32] J. Kim and R. Hsu, “Embedded CMOS camera in a laptop computer,” *US Pat. 6,496,361*, 2002.

[33] J. Palacin, A. Sanuy, and X. Clua, “Autonomous mobile mini-robot with embedded CMOS vision system,” *IECON 02 [Industrial …*, 2002.

[34] A. El Gamal and H. Eltoukhy, “CMOS image sensors,” *Circuits Devices Mag. IEEE*, 2005.

[35] J. Geist, E. Zalewski, and A. Schaefer, “Spectral response self-calibration and interpolation of silicon photodiodes,” *Appl. Opt.*, 1980.

[36] B. Bayer, “Color imaging array,” *US Pat. 3,971,065*, 1976.

[37] C. M. L. Burnett, “Bayer pattern on sensor.svg,” *Wikimedia Commons*, 2006. [Online]. Available: http://commons.wikimedia.org/wiki/File:Bayer_pattern_on_sensor.svg.

[38] E. Alpaydin, *Introduction To Machine Learning*. MIT Press, 2004, p. 415.

[39] “AI Horizon: Introduction to Machine Learning.” [Online]. Available: http://www.aihorizon.com/essays/generalai/supervised_unsupervised_machine_learning.htm. [Accessed: 26-Jul-2013].

[40] P. Jansson, “Neural networks: An overview,” *Anal. Chem.*, 1991.

[41] Z. Ghahramani, “Unsupervised learning,” *Adv. Lect. Mach. Learn.*, 2004.

[42] A. Joshi and R. Kaur, “A Review: Comparative Study of Various Clustering Techniques in Data Mining,” *Int. J.*, 2013.

[43] P. Gehler and S. Nowozin, “On feature combination for multiclass object classification,” *Comput. Vision, 2009 IEEE 12th …*, 2009.

[44] J. Bagnell and D. Bradley, “Learning for autonomous navigation,” *Robot. …*, 2010.

[45] Y. Boussemart, M. Cummings, J. Las Fargeas, and N. Roy, “Supervised vs Unsupervised Learning for Operator State Modeling in Unmanned Vehicle Settings,” *J. Aerosp. Comput. Inf. Commun.*, vol. 8, 2011.

[46] “ICMLA 2013: Call For Papers.” [Online]. Available: http://icmla-conference.org/icmla13/callpaper.htm. [Accessed: 30-Jul-2013].

[47] R. Haralick and L. Shapiro, “Image segmentation techniques,” *Comput. vision, Graph. image …*, 1985.

[48] C. Brice and C. Fennema, “Scene Analysis Using Regions,” *Artif. Intell.*, 1970.

[49] Y. Zhang, “An overview of image and video segmentation in the last 40 years,” *Adv. Image Video Segmentation*, pp. 1–15, 2006.

[50] T. Zuva and O. Oludayo, “Image segmentation, available techniques, developments and open issues,” *… J. Image …*, 2011.

[51] R. Maini and H. Aggarwal, “Study and comparison of various image edge detection techniques,” *Int. J. Image Process. …*, 2009.

[52] G. Peyré, “Lecture 1 - Active Contours and Level Sets,” *ceremade.dauphine.fr*, 2006. [Online]. Available: https://www.ceremade.dauphine.fr/~peyre/teaching/manifold/tp1.html. [Accessed: 01-May-2014].

[53] N. Pal and S. Pal, “A review on image segmentation techniques,” *Pattern Recognit.*, vol. 26, no. 9, 1993.

[54] Y. Kee, M. Souiai, D. Cremers, and J. Kim, “Sequential Convex Relaxation for Mutual-Information-Based Unsupervised Figure-Ground Segmentation,” *IEEE Conf. onf Comput. Vis. Pattern Recognit.*, 2014.

[55] C. Rotaru, T. Graf, and J. Zhang, “Color image segmentation in hsi space for automotive applications,” *J. Real-Time Image Process.*, 2008.

[56] X. Lin and S. Chen, “Color image segmentation using modified HSI system for road following,” *Robot. Autom. 1991. Proceedings., …*, 1991.

[57] S. Kenue, “Lanelok: Detection Of Lane Boundaries And Vehicle Tracking Using Image-Processing Techniques-Part I: Hough-Transform, Region-Tracing And Correlation,” *1989 Adv. Intell. Robot. …*, 1990.

[58] M. Chen, T. Jochem, and D. Pomerleau, “AURORA: A vision-based roadway departure warning system,” *Intell. Robot. …*, 1995.

[59] K. Kluge and C. Thorpe, “The YARF system for vision-based road following,” *Math. Comput. Model.*, 1995.

[60] D. Kang, J. Choi, and I. Kweon, “Finding and tracking road lanes using ‘line-snakes,’” *Intell. Veh. Symp. …*, 1996.

[61] M. Kass, A. Witkin, and D. Terzopoulos, “Snakes: Active contour models,” *Int. J. Comput. Vis.*, vol. 331, pp. 321–331, 1988.

[62] Y. Yim and S. Oh, “Three-feature based automatic lane detection algorithm (TFALDA) for autonomous driving,” *Intell. Transp. Syst. IEEE …*, 2003.

[63] J. McCall and M. Trivedi, “An integrated, robust approach to lane marking detection and lane tracking,” *Intell. Veh. Symp. 2004 …*, 2004.

[64] C. De Boor, *B (asic)-spline basics*. 1986.

[65] R. Heij, M. Helgers, W. Kockelkorn, and R. Smelik, “Snakes for Lane Detection Lane detection,” no. April, pp. 1–19, 2006.

[66] Z. Kim, “Robust lane detection and tracking in challenging scenarios,” *Intell. Transp. Syst. IEEE Trans. …*, 2008.

[67] A. Bar Hillel, R. Lerner, D. Levi, and G. Raz, “Recent progress in road and lane detection: a survey,” *Mach. Vis. Appl.*, Feb. 2012.

[68] P. Trisiripisal, M. Parks, and A. Abbott, “Stereo analysis for vision-based guidance and control of aircraft landing,” *44th AIAA Aerosp. …*, 2006.

[69] X. Gong, A. Abbott, and G. Fleming, “A survey of techniques for detection and tracking of airport runways,” *44th AIAA Aerosp. Sci. …*, 2006.

[70] L. G. Roberts, “Machine perception of three-dimensional solids,” *Opt. Electro-optical Inf. Process.*, pp. 159–197, 1965.

[71] J. Canny, “A computational approach to edge detection,” *Pattern Anal. Mach. Intell. IEEE …*, 1986.

[72] Y. Nakagawa and A. Rosenfeld, “Some experiments on variable thresholding,” *Pattern Recognit.*, 1979.

[73] A. Jain and F. Farrokhnia, “Unsupervised texture segmentation using Gabor filters,” *Pattern Recognit.*, 1991.

[74] L. Pan, L. Gu, and J. Xu, “Implementation of medical image segmentation in CUDA,” *Inf. Technol. Appl. …*, 2008.

[75] J. Kenney, T. Buckley, and O. Brock, “Interactive segmentation for manipulation in unstructured environments,” *Robot. Autom. 2009. …*, 2009.

[76] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *Pattern Anal. Mach. Intell. IEEE …*, 2000.

[77] H. Steinhaus, “Sur la division des corp materiels en parties,” *Bull. Acad. Pol. Sci*, 1956.

[78] G. Ball and D. Hall, “ISODATA, a novel method of data analysis and pattern classification,” 1965.

[79] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” *Proc. fifth Berkeley Symp. …*, 1967.

[80] Y. Cheng, “Mean shift, mode seeking, and clustering,” *Pattern Anal. Mach. Intell. IEEE …*, 1995.

[81] A. K. Jain, “Data clustering: 50 years beyond K-means,” *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651–666, Jun. 2010.

[82] R. Szelizki, *Computer Vision: Algorithms and Applications*, 1st ed. Ithaca: Springer, 2011, pp. 29–51.

[83] “OpenCV Documentation,” *www.opencv.org*, 2014. [Online]. Available: http://docs.opencv.org/.

[84] D. Ballard, “Generalizing the Hough transform to detect arbitrary shapes,” *Pattern Recognit.*, 1981.

[85] J. VandeKieft, “Computational Improvements to Linear Convolution with Multirate Filtering Methods,” University of Miami, 1998.

[86] M. Albanesi, “Time complexity evaluation of algorithms for the Hough transform on mesh connected computers,” *CompEuro’91. Adv. Comput. Technol. …*, pp. 253–257, 1991.

[87] “Spline Terminology - NX Training,” *www.content.mygetit.com*. [Online]. Available: http://content.myigetit.com/trainingPublic/204/html/110908.asp?free=0.

[88] Digia, “Documentation | Qt Project,” *https://qt-project.org/doc/*, 2013. [Online]. Available: https://qt-project.org/doc/. [Accessed: 06-Feb-2014].

[89] “ABOUT | OpenCV,” *www.opencv.org*, 2014. [Online]. Available: http://opencv.org/about.html.

[90] G. Guennebaud and J. Beno et. al., “Eigen v3.” http://eigen.tuxfamily.org, 2010.

[91] G. Gunnebaud, “Eigen - a C++ Linear Algebra Library,” *Aristote*. Aristotle Association, Palaiseu, 2013.

[92] D. King, “Dlib-ml: A machine learning toolkit,” *J. Mach. Learn. Res.*, 2009.

[93] D. King, “dlib C++ Library - Introduction,” *www.dlib.net*, 2009. [Online]. Available: http://dlib.net/intro.html.

[94] K. Wong, “Kerry D. Wong » Blog Archive » Canny Edge Detection Auto Thresholding,” *www.kerrywong.com*, 2009. [Online]. Available: http://www.kerrywong.com/2009/05/07/canny-edge-detection-auto-thresholding/.

[95] J. Davis and M. Goadrich, “The relationship between Precision-Recall and ROC curves,” *… 23rd Int. Conf. Mach. Learn.*, 2006.

[96] M. Khan, A. Ayob, and T. Ray, “An efficient memetic algorithm for 3D shape matching problems,” *Eng. Optim.*, 2013.

[97] E. Hsiao and M. Hebert, “Gradient networks: Explicit shape matching without extracting edges,” *Proc. AAAI Conf. Artif. …*, 2013.

[98] Z. Lian, A. Godil, B. Bustos, and M. Daoudi, “A comparison of methods for non-rigid 3D shape retrieval,” *Pattern Recognit.*, 2013.

[99] R. Veltkamp, “Shape matching: Similarity measures and algorithms,” *Shape Model. Appl. SMI 2001 …*, 2001.

[100] J. Davis and A. Bobick, “The representation and recognition of human movement using temporal templates,” *Comput. Vis. Pattern …*, 1997.

[101] M. Hu, “Visual pattern recognition by moment invariants,” *Inf. Theory, IRE Trans.*, 1962.

[102] K. Forster and J. Forster, “DMDX: A Windows display program with millisecond accuracy,” *… Res. Methods, Instruments, Comput.*, 2003.

[103] “Quantifying The Accuracy Of Sleep - CodeProject.” [Online]. Available: http://www.codeproject.com/Articles/3831/Quantifying-The-Accuracy-Of-Sleep. [Accessed: 03-Mar-2014].

[104] H. Hinnant, W. Brown, J. Garland, and M. Paterno, “A Foundation to Sleep On: Clocks, Points in Time, and Time Durations,” 2008.

[105] M. Zechner and M. Granitzer, “Accelerating k-means on the graphics processor via cuda,” *Intensive Appl. Serv. …*, 2009.

[106] L. Li, J. Song, and F. Wang, “IVS 05: new developments and research trends for intelligent vehicles,” *IEEE Intell. …*, 2005.

APPENDIX A Road Image Examples

This appendix contains examples of the types of images used for analysis in the above work. The parameters intentionally varied in the collected data were the road type (single-lane, multi-lane), the line color (white, yellow), the line strength (normal, faded) and the light type (bright sun, late day sun, shadows). The captions below each image indicate what its particular classification is.

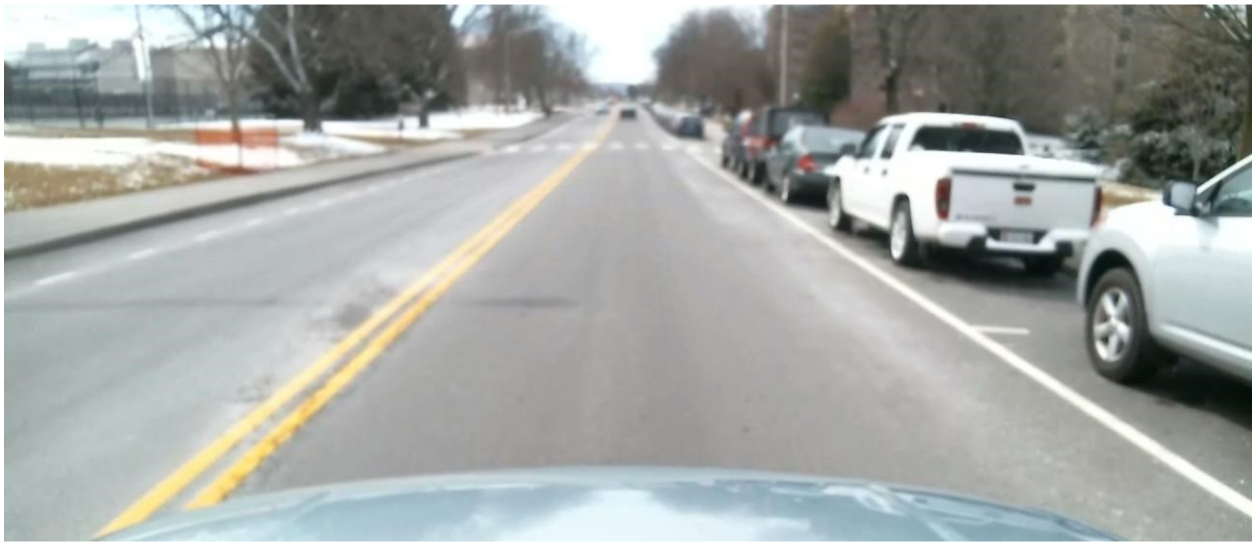


Figure 64: Single lane, Yellow Lined Roadway, Bright Sunlight

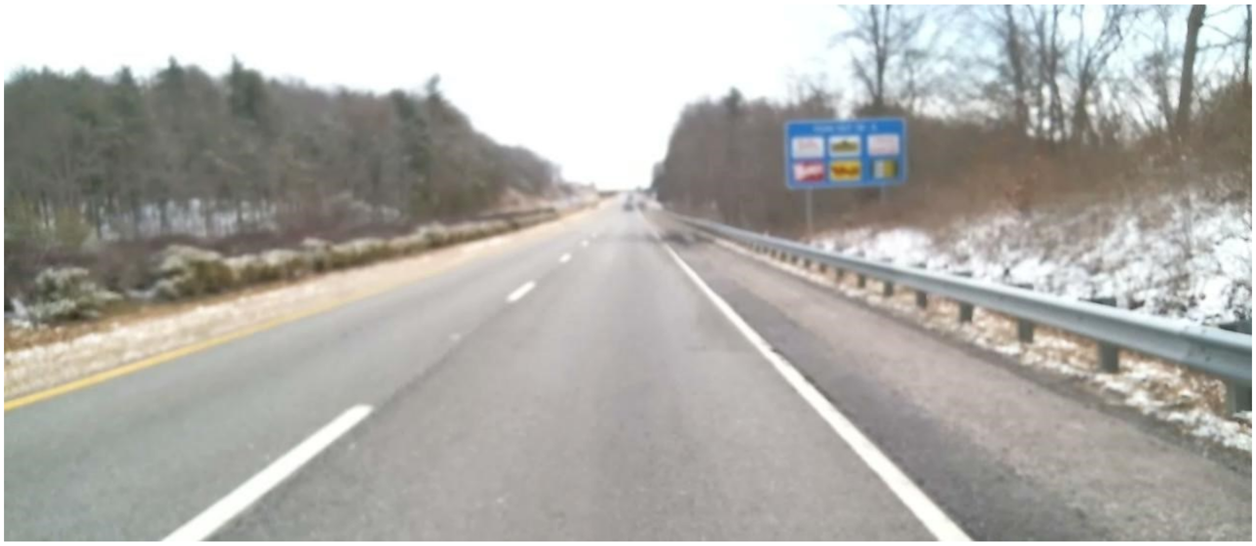


Figure 65: Multi-lane, White Lined Roadway, Bright Sunlight

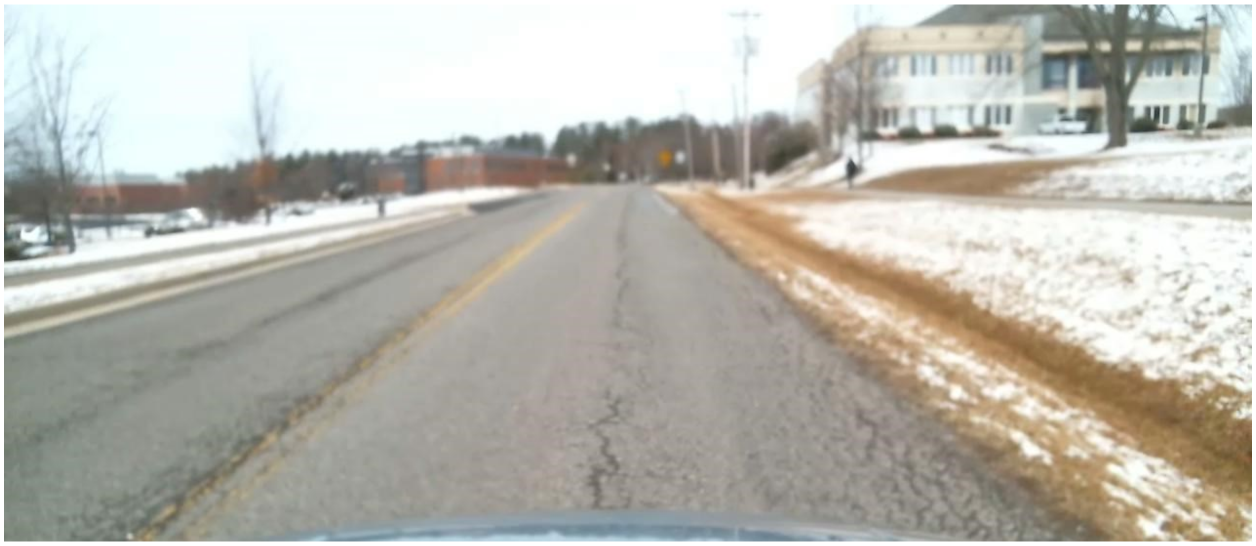


Figure 66: Single Lane, Faded Yellow Line, Bright Sunlight

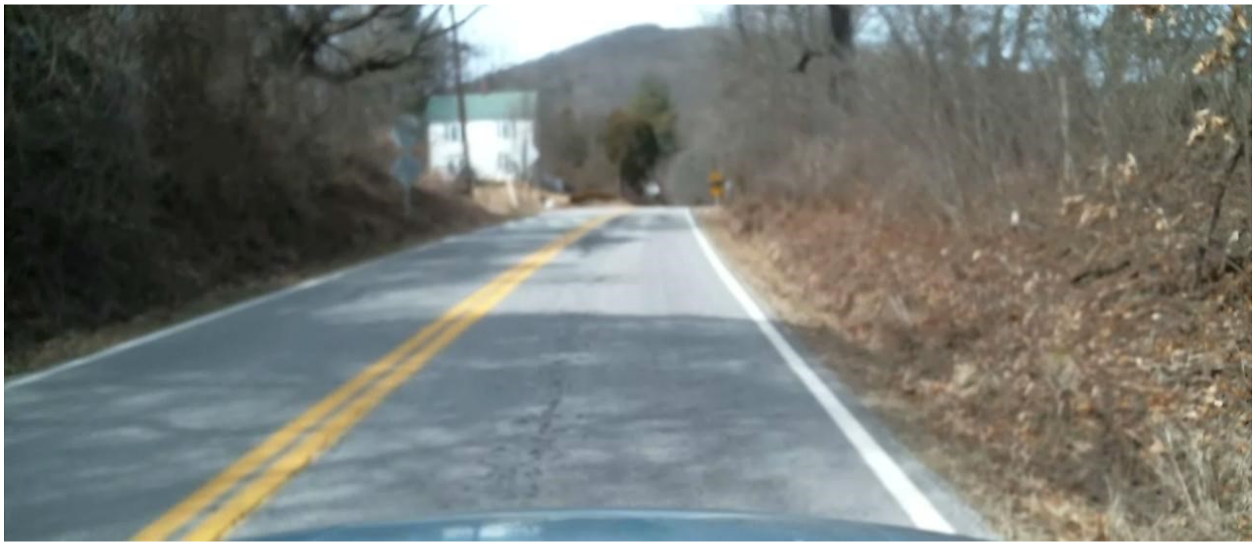


Figure 67: Single Lane, Yellow Line, Shadows

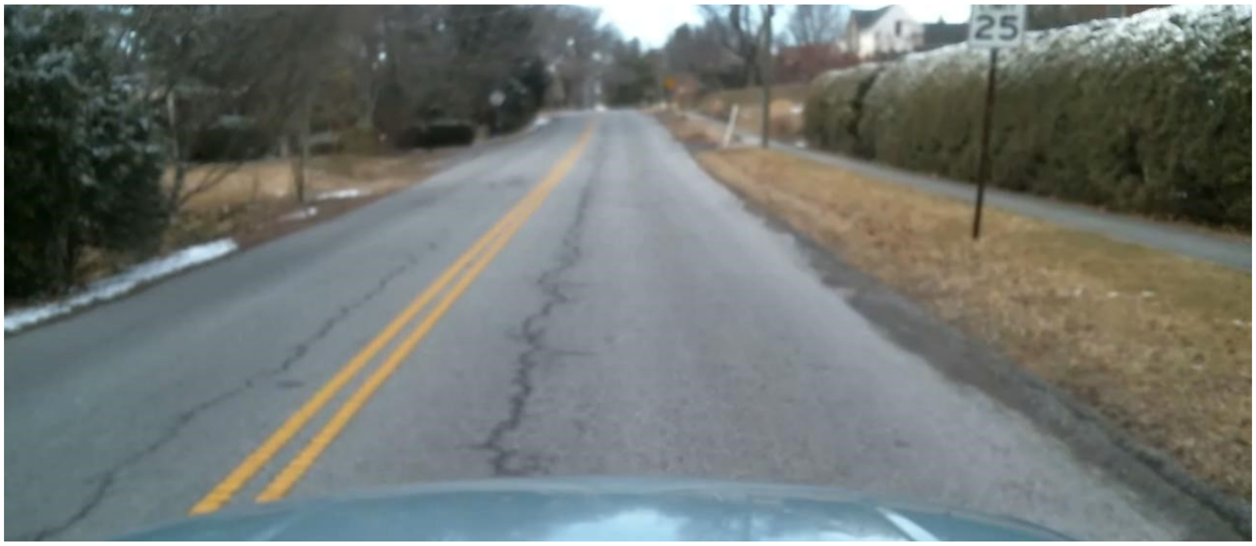


Figure 68: Single Lane, Yellow Line, Late Day Sun

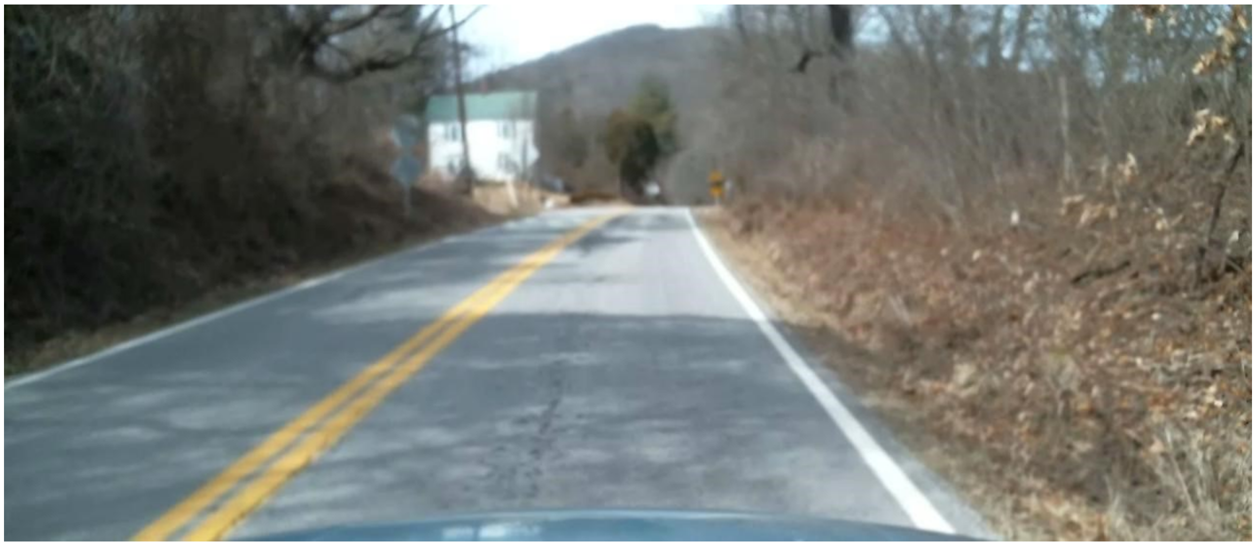


Figure 69: Single Lane, Yellow Line, Shadows

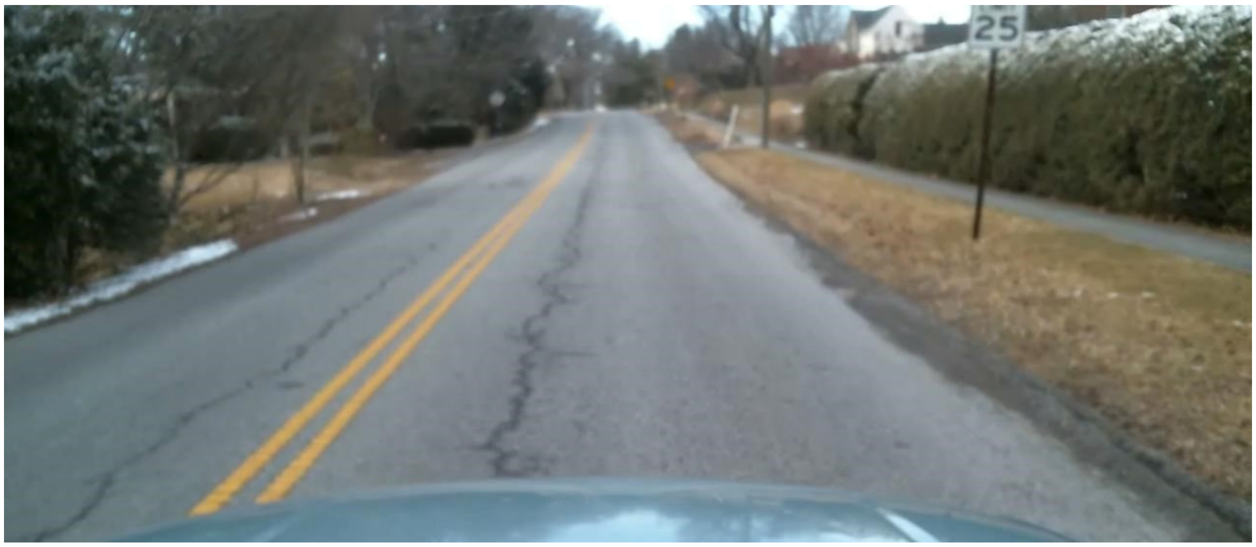


Figure 70: Single Lane, Yellow Line, Late Day Sun

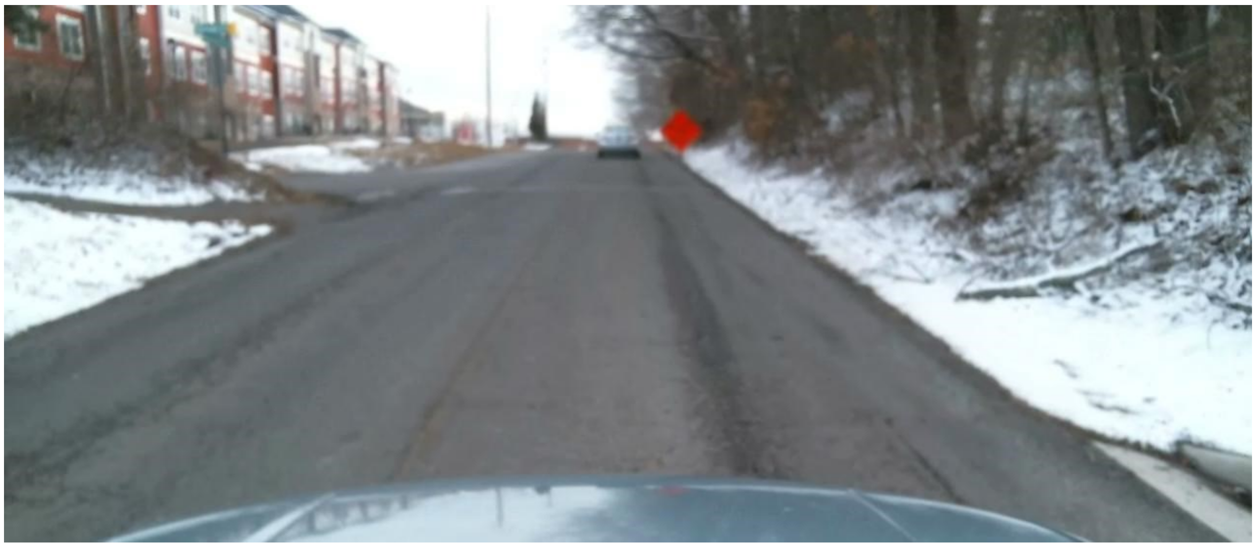


Figure 71: Single Lane, Faded Line, Late Day Sun

APPENDIX B Tables of Accuracy and Convergence Data

Accuracy Data

Data Set	Q Threshold	K Threshold	Avg. Pixel Accuracy(%)	Avg. Shape Accuracy (0-1)	BSnake Conv. (%)
2,0	0.2	0.04	87.8472	0.934658	16.4557
2,0	0.3	0.04	87.9501	0.939315	48.1013
2,0	0.4	0.04	87.5012	0.923096	67.0886
2,0	0.5	0.04	87.9326	0.922031	82.2785
2,0	0.6	0.04	83.5009	0.872478	97.4684
2,0	0.7	0.04	87.251	0.901073	91.1392
2,0	0.8	0.04	85.6572	0.892752	93.6709
2,2	0.2	0.04	86.1838	0.927731	26.7241
2,2	0.3	0.04	84.2292	0.922235	56.8966
2,2	0.4	0.04	84.8571	0.903371	75.8621
2,2	0.5	0.04	85.4558	0.907982	93.1034
2,2	0.6	0.04	83.2471	0.694492	97.4138
2,2	0.7	0.04	81.7034	0.855051	96.5517
2,2	0.8	0.04	85.1384	0.852198	99.1379
2,4	0.2	0.04	85.7864	0.943922	24
2,4	0.3	0.04	86.8984	0.945949	58
2,4	0.4	0.04	86.2113	0.944512	82
2,4	0.5	0.04	88.0403	0.951179	86
2,4	0.6	0.04	82.2282	0.986521	96
2,4	0.7	0.04	79.1071	0.779413	99
2,4	0.8	0.04	94.3173	0.869116	99
2,7	0.2	0.04	90.5157	0.937377	30
2,7	0.3	0.04	90.5005	0.895304	51.1111
2,7	0.4	0.04	88.486	0.891457	82.2222
2,7	0.5	0.04	85.0172	0.846957	97.7778
2,7	0.6	0.04	84.1208	0.838022	96.6667
2,7	0.7	0.04	87.3164	0.771827	98.8889
2,7	0.8	0.04	84.2762	0.752188	98.8889
2,9	0.2	0.04	84.9532	0.901866	13.3333
2,9	0.3	0.04	83.9785	0.867442	40
2,9	0.4	0.04	85.4656	0.952945	48.8889
2,9	0.5	0.04	87.0508	0.874356	83.3333
2,9	0.6	0.04	87.6245	0.897448	90
2,9	0.7	0.04	85.8677	0.863028	97.7778
2,9	0.8	0.04	92.2341	0.900616	98.8889
3,5	0.2	0.04	77.8036	0.916626	40
3,5	0.3	0.04	78.0511	0.83533	75
3,5	0.4	0.04	78.9391	0.878406	77.5
3,5	0.5	0.04	80.0749	0.787003	95
3,5	0.6	0.04	79.9479	0.766762	95

Data Set	Q Threshold	K Threshold	Avg. Pixel Accuracy(%)	Avg. Shape Accuracy (0-1)	BSnake Conv. (%)
3,5	0.7	0.04	79.1081	0.760155	95
3,5	0.8	0.04	78.6233	0.719199	95
3,7	0.2	0.04	87.7702	0.88118	29.4118
3,7	0.3	0.04	85.3077	0.890723	82.3529
3,7	0.4	0.04	84.5176	0.87855	84.0336
3,7	0.5	0.04	86.493	0.944102	88.2353
3,7	0.6	0.04	81.0948	0.866041	94.1176
3,7	0.7	0.04	80.635	0.668974	99.1597
3,7	0.8	0.04	83.9554	0.878079	96.6387
6,1	0.2	0.04	85.5854	0.966941	24.8639
6,1	0.3	0.04	85.2768	0.932961	53.539
6,1	0.4	0.04	86.4126	0.971361	76.225
6,1	0.5	0.04	85.9552	0.956679	88.3848
6,1	0.6	0.04	82.821	0.962456	94.7368
6,1	0.7	0.04	80.7748	0.924923	96.5517
6,1	0.8	0.04	88.9332	0.921975	98.9111
6,4	0.2	0.04	81.9191	0.952338	29.7143
6,4	0.3	0.04	78.7154	0.882434	86.2857
6,4	0.4	0.04	81.0303	0.893664	81.1429
6,4	0.5	0.04	81.3426	0.872044	92.5714
6,4	0.6	0.04	81.0415	0.870486	95.4286
6,4	0.7	0.04	86.0387	0.792091	98.8571

Timing Data

Average Times Across all Data Sets for IRIS and K-Means

For all tests, kThreshold was held constant at 0.04. QThreshold is listed above each table.

QThreshold = 0.2

Clusters	Average Converged Time (us)	Average Not Converged Time (us)
3	2453539	3098870
4	3077268	3717272
5	3829105	4530538
6	4459078	4993983
7	5112711	5762606
8	5917476	6473038
9	6711828	7548347
10	7894633	8652063
11	9079187	9876696
12	10696882	11128986
13	11504143	12428900

QThreshold = 0.3

Clusters	Average Converged Time (us)	Average Not Converged Time (us)
3	2325488	3086227
4	2971676	3681082
5	3580693	4541605
6	4279313	4998874
7	4938519	5765077
8	5625006	6535398
9	6314933	7629268
10	7413356	8714929
11	8788283	10019325
12	10295295	11220114
13	11747493	12407220

QThreshold = 0.5

Clusters	Average Converged Time (us)	Average Not Converged Time (us)
3	2124210	2983071
4	2725627	3682728
5	3364166	4648572
6	3995446	4815799
7	4561251	5773406
8	5254849	6355701
9	6260773	7775875
10	7391013	9256151

11	8804551	10677415
12	10353010	11233870
13	11614670	12686375

	QThreshold = 0.7	
Clusters	**Average Converged Time (us)**	**Average Not Converged Time (us)**
3	1992464	2661132
4	2590087	3494680
5	3344537	4582890
6	4029404	4595643
7	4554031	5934120
8	5131028	5945746
9	6140400	7631320
10	7325129	9447589
11	8913416	9867956
12	10282840	10494140
13	11492246	12313648

Clusters	Average K-means Time
3	2293333.333
4	3236666.667
5	4550000
6	5951666.667
7	6180000
8	6868333.333
9	8511666.667
10	9948333.333
11	11016666.67
12	12983333.33

All Raw Timing Data

All raw timing data is presented below.

Data Set	QThresh	Images	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
3,5	0.2	40	3	2.18E+06	12	2.87E+06	28
3,5	0.2	40	4	2.73E+06	12	3.61E+06	28
3,5	0.2	40	5	3.76E+06	12	4.89E+06	28
3,5	0.2	40	6	3.94E+06	12	4.51E+06	28
3,5	0.2	40	7	4.43E+06	12	4.87E+06	28
3,5	0.2	40	8	6.14E+06	12	5.42E+06	28
3,5	0.2	40	9	5.78E+06	12	7.25E+06	28
3,5	0.2	40	10	7.21E+06	12	8.77E+06	28
3,5	0.2	40	11	9.48E+06	12	9.16E+06	28
3,5	0.2	40	12	1.11E+07	12	9.57E+06	28
3,5	0.2	40	13	1.10E+07	12	1.06E+07	28
3,5	0.3	40	3	2.12E+06	20	2.86E+06	20
3,5	0.3	40	4	2.74E+06	20	3.63E+06	20
3,5	0.3	40	5	3.44E+06	20	5.14E+06	20
3,5	0.3	40	6	3.84E+06	20	4.51E+06	20
3,5	0.3	40	7	4.65E+06	20	4.87E+06	20
3,5	0.3	40	8	5.28E+06	20	5.59E+06	20
3,5	0.3	40	9	5.07E+06	20	7.61E+06	20
3,5	0.3	40	10	6.01E+06	20	9.00E+06	20
3,5	0.3	40	11	7.44E+06	20	9.35E+06	20
3,5	0.3	40	12	8.07E+06	20	9.75E+06	20
3,5	0.3	40	13	9.18E+06	20	1.07E+07	20
3,5	0.5	40	3	1.81E+06	33	2.63E+06	7
3,5	0.5	40	4	3.02E+06	33	3.68E+06	7
3,5	0.5	40	5	3.30E+06	33	5.75E+06	7
3,5	0.5	40	6	3.50E+06	33	4.01E+06	7
3,5	0.5	40	7	4.01E+06	33	4.47E+06	7
3,5	0.5	40	8	4.52E+06	33	5.18E+06	7
3,5	0.5	40	9	5.40E+06	33	6.69E+06	7
3,5	0.5	40	10	7.21E+06	33	7.72E+06	7
3,5	0.5	40	11	8.64E+06	33	9.24E+06	7
3,5	0.5	40	12	1.10E+07	33	1.02E+07	7
3,5	0.5	40	13	1.14E+07	33	1.01E+07	7
3,5	0.7	40	3	1.81E+06	37	2.10E+06	3
3,5	0.7	40	4	2.47E+06	37	3.78E+06	3
3,5	0.7	40	5	3.33E+06	37	7.22E+06	3
3,5	0.7	40	6	3.24E+06	37	3.79E+06	3
3,5	0.7	40	7	3.76E+06	37	4.52E+06	3
3,5	0.7	40	8	4.53E+06	37	5.01E+06	3
3,5	0.7	40	9	5.59E+06	37	5.44E+06	3

Data Set	QThresh	Images	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
3,5	0.7	40	10	6.70E+06	37	6.46E+06	3
3,5	0.7	40	11	8.94E+06	37	8.91E+06	3
3,5	0.7	40	12	9.97E+06	37	1.05E+07	3
3,5	0.7	40	13	1.16E+07	37	9.15E+06	3
2,0	0.2	62	3	2.34E+06	12	3.08E+06	50
2,0	0.2	62	4	2.77E+06	12	3.60E+06	50
2,0	0.2	62	5	3.15E+06	12	4.20E+06	50
2,0	0.2	62	6	3.60E+06	12	5.06E+06	50
2,0	0.2	62	7	4.71E+06	12	6.08E+06	50
2,0	0.2	62	8	6.29E+06	12	6.75E+06	50
2,0	0.2	62	9	5.76E+06	12	8.02E+06	50
2,0	0.2	62	10	6.82E+06	12	7.94E+06	50
2,0	0.2	62	11	7.48E+06	12	8.83E+06	50
2,0	0.2	62	12	7.45E+06	12	9.75E+06	50
2,0	0.2	62	13	8.30E+06	12	1.00E+07	50
2,0	0.3	62	3	2.14E+06	29	3.10E+06	33
2,0	0.3	62	4	2.60E+06	29	3.61E+06	33
2,0	0.3	62	5	2.93E+06	29	4.30E+06	33
2,0	0.3	62	6	3.90E+06	29	5.05E+06	33
2,0	0.3	62	7	4.72E+06	29	5.92E+06	33
2,0	0.3	62	8	5.23E+06	29	6.85E+06	33
2,0	0.3	62	9	5.71E+06	29	8.59E+06	33
2,0	0.3	62	10	6.61E+06	29	8.34E+06	33
2,0	0.3	62	11	7.07E+06	29	9.12E+06	33
2,0	0.3	62	12	7.91E+06	29	1.01E+07	33
2,0	0.3	62	13	1.02E+07	29	1.02E+07	33
2,0	0.5	62	3	2.06E+06	52	2.87E+06	10
2,0	0.5	62	4	2.43E+06	52	3.37E+06	10
2,0	0.5	62	5	2.82E+06	52	4.07E+06	10
2,0	0.5	62	6	4.10E+06	52	4.45E+06	10
2,0	0.5	62	7	4.69E+06	52	5.82E+06	10
2,0	0.5	62	8	5.68E+06	52	6.79E+06	10
2,0	0.5	62	9	6.27E+06	52	1.03E+07	10
2,0	0.5	62	10	6.78E+06	52	8.93E+06	10
2,0	0.5	62	11	7.99E+06	52	9.22E+06	10
2,0	0.5	62	12	8.25E+06	52	1.02E+07	10
2,0	0.5	62	13	8.95E+06	52	9.94E+06	10
2,0	0.7	62	3	1.91E+06	58	2.38E+06	4
2,0	0.7	62	4	2.33E+06	58	3.16E+06	4
2,0	0.7	62	5	2.73E+06	58	4.43E+06	4
2,0	0.7	62	6	4.17E+06	58	4.01E+06	4
2,0	0.7	62	7	5.07E+06	58	5.80E+06	4
2,0	0.7	62	8	5.50E+06	58	7.00E+06	4

Data Set	QThresh	Images	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
2,0	0.7	62	9	6.04E+06	58	1.51E+07	4
2,0	0.7	62	10	6.47E+06	58	1.04E+07	4
2,0	0.7	62	11	7.34E+06	58	9.94E+06	4
2,0	0.7	62	12	8.14E+06	58	1.28E+07	4
2,0	0.7	62	13	8.94E+06	58	1.08E+07	4
2,2	0.47	79	3	2.00E+06	56	2.79E+06	23
2,2	0.47	79	4	2.64E+06	56	3.26E+06	23
2,2	0.47	79	5	3.59E+06	56	4.51E+06	23
2,2	0.47	79	6	4.39E+06	56	5.71E+06	23
2,2	0.47	79	7	5.14E+06	56	6.57E+06	23
2,2	0.47	79	8	6.24E+06	56	7.40E+06	23
2,2	0.47	79	9	6.82E+06	56	8.34E+06	23
2,2	0.47	79	10	8.00E+06	56	8.96E+06	23
2,2	0.47	79	11	9.14E+06	56	1.11E+07	23
2,2	0.47	79	12	1.06E+07	56	1.34E+07	23
2,2	0.47	79	13	1.26E+07	56	1.28E+07	23
2,2	0.52	79	3	1.91E+06	62	2.68E+06	17
2,2	0.52	79	4	2.45E+06	62	3.20E+06	17
2,2	0.52	79	5	3.39E+06	62	4.27E+06	17
2,2	0.52	79	6	4.31E+06	62	5.63E+06	17
2,2	0.52	79	7	4.89E+06	62	6.48E+06	17
2,2	0.52	79	8	6.31E+06	62	7.21E+06	17
2,2	0.52	79	9	6.72E+06	62	8.47E+06	17
2,2	0.52	79	10	7.39E+06	62	9.42E+06	17
2,2	0.52	79	11	8.91E+06	62	1.17E+07	17
2,2	0.52	79	12	1.04E+07	62	1.23E+07	17
2,2	0.52	79	13	1.22E+07	62	1.24E+07	17
2,9	0.2	90	3	2.66E+06	12	3.19E+06	78
2,9	0.2	90	4	3.03E+06	12	3.59E+06	78
2,9	0.2	90	5	4.08E+06	12	4.28E+06	78
2,9	0.2	90	6	5.05E+06	12	4.91E+06	78
2,9	0.2	90	7	5.30E+06	12	5.78E+06	78
2,9	0.2	90	8	5.47E+06	12	6.53E+06	78
2,9	0.2	90	9	6.46E+06	12	7.30E+06	78
2,9	0.2	90	10	7.38E+06	12	7.92E+06	78
2,9	0.2	90	11	6.74E+06	12	9.35E+06	78
2,9	0.2	90	12	7.77E+06	12	1.03E+07	78
2,9	0.2	90	13	9.62E+06	12	1.11E+07	78
2,9	0.2	90	3	2.51E+06	19	3.10E+06	71
2,9	0.2	90	4	3.04E+06	19	3.70E+06	71
2,9	0.2	90	5	3.61E+06	19	4.15E+06	71
2,9	0.2	90	6	3.65E+06	19	4.70E+06	71
2,9	0.2	90	7	4.18E+06	19	5.08E+06	71

Data Set	QThresh	Images	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
2,9	0.2	90	8	5.30E+06	19	5.58E+06	71
2,9	0.2	90	9	5.51E+06	19	6.20E+06	71
2,9	0.2	90	10	6.99E+06	19	7.68E+06	71
2,9	0.2	90	11	8.58E+06	19	8.95E+06	71
2,9	0.2	90	12	9.05E+06	19	1.07E+07	71
2,9	0.2	90	13	1.00E+07	19	1.31E+07	71
3,10	0.2	90	3	2.64E+06	12	3.17E+06	78
3,10	0.2	90	4	3.01E+06	12	3.57E+06	78
3,10	0.2	90	5	4.05E+06	12	4.25E+06	78
3,10	0.2	90	6	5.00E+06	12	4.87E+06	78
3,10	0.2	90	7	5.26E+06	12	5.74E+06	78
3,10	0.2	90	8	5.43E+06	12	6.48E+06	78
3,10	0.2	90	9	6.41E+06	12	7.25E+06	78
3,10	0.2	90	10	7.33E+06	12	7.86E+06	78
3,10	0.2	90	11	6.70E+06	12	9.29E+06	78
3,10	0.2	90	12	7.72E+06	12	1.03E+07	78
3,10	0.2	90	13	9.56E+06	12	1.11E+07	78
2,9	0.3	90	3	2.58E+06	26	3.12E+06	64
2,9	0.3	90	4	2.86E+06	26	3.49E+06	64
2,9	0.3	90	5	3.35E+06	26	4.25E+06	64
2,9	0.3	90	6	4.28E+06	26	4.83E+06	64
2,9	0.3	90	7	5.40E+06	26	5.73E+06	64
2,9	0.3	90	8	6.37E+06	26	6.61E+06	64
2,9	0.3	90	9	6.76E+06	26	7.22E+06	64
2,9	0.3	90	10	6.74E+06	26	7.65E+06	64
2,9	0.3	90	11	7.68E+06	26	9.34E+06	64
2,9	0.3	90	12	8.83E+06	26	1.03E+07	64
2,9	0.3	90	13	9.97E+06	26	1.12E+07	64
2,9	0.3	90	3	2.28E+06	42	3.07E+06	48
2,9	0.3	90	4	3.06E+06	42	3.50E+06	48
2,9	0.3	90	5	3.47E+06	42	4.12E+06	48
2,9	0.3	90	6	4.05E+06	42	4.66E+06	48
2,9	0.3	90	7	4.19E+06	42	4.97E+06	48
2,9	0.3	90	8	4.69E+06	42	5.48E+06	48
2,9	0.3	90	9	5.55E+06	42	6.24E+06	48
2,9	0.3	90	10	7.15E+06	42	7.98E+06	48
2,9	0.3	90	11	8.98E+06	42	9.33E+06	48
2,9	0.3	90	12	1.09E+07	42	1.13E+07	48
2,9	0.3	90	13	1.19E+07	42	1.33E+07	48
3,10	0.3	90	3	2.57E+06	26	3.09E+06	64
3,10	0.3	90	4	2.84E+06	26	3.46E+06	64
3,10	0.3	90	5	3.33E+06	26	4.22E+06	64
3,10	0.3	90	6	4.25E+06	26	4.80E+06	64

Data Set	QThresh	Images	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
3,10	0.3	90	7	5.37E+06	26	5.69E+06	64
3,10	0.3	90	8	6.34E+06	26	6.57E+06	64
3,10	0.3	90	9	6.72E+06	26	7.18E+06	64
3,10	0.3	90	10	6.70E+06	26	7.60E+06	64
3,10	0.3	90	11	7.64E+06	26	9.28E+06	64
3,10	0.3	90	12	8.78E+06	26	1.02E+07	64
3,10	0.3	90	13	9.91E+06	26	1.12E+07	64
2,9	0.5	90	3	2.32E+06	64	3.08E+06	26
2,9	0.5	90	4	2.55E+06	64	3.55E+06	26
2,9	0.5	90	5	3.17E+06	64	4.44E+06	26
2,9	0.5	90	6	3.90E+06	64	4.58E+06	26
2,9	0.5	90	7	4.38E+06	64	5.58E+06	26
2,9	0.5	90	8	4.83E+06	64	6.33E+06	26
2,9	0.5	90	9	5.71E+06	64	7.36E+06	26
2,9	0.5	90	10	6.81E+06	64	7.81E+06	26
2,9	0.5	90	11	7.61E+06	64	8.85E+06	26
2,9	0.5	90	12	8.41E+06	64	8.65E+06	26
2,9	0.5	90	13	9.82E+06	64	9.66E+06	26
2,9	0.5	90	3	2.07E+06	79	3.04E+06	11
2,9	0.5	90	4	2.58E+06	79	3.53E+06	11
2,9	0.5	90	5	3.07E+06	79	4.05E+06	11
2,9	0.5	90	6	3.60E+06	79	4.72E+06	11
2,9	0.5	90	7	3.93E+06	79	5.13E+06	11
2,9	0.5	90	8	4.41E+06	79	5.64E+06	11
2,9	0.5	90	9	5.05E+06	79	6.76E+06	11
2,9	0.5	90	10	6.59E+06	79	8.43E+06	11
2,9	0.5	90	11	9.00E+06	79	1.07E+07	11
2,9	0.5	90	12	1.09E+07	79	1.20E+07	11
2,9	0.5	90	13	1.29E+07	79	1.50E+07	11
3,10	0.5	90	3	2.30E+06	64	3.04E+06	26
3,10	0.5	90	4	2.53E+06	64	3.51E+06	26
3,10	0.5	90	5	3.14E+06	64	4.39E+06	26
3,10	0.5	90	6	3.86E+06	64	4.53E+06	26
3,10	0.5	90	7	4.35E+06	64	5.53E+06	26
3,10	0.5	90	8	4.80E+06	64	6.28E+06	26
3,10	0.5	90	9	5.67E+06	64	7.31E+06	26
3,10	0.5	90	10	6.76E+06	64	7.76E+06	26
3,10	0.5	90	11	7.56E+06	64	8.79E+06	26
3,10	0.5	90	12	8.36E+06	64	8.60E+06	26
3,10	0.5	90	13	9.77E+06	64	9.60E+06	26
2,9	0.7	90	3	2.14E+06	82	2.95E+06	8
2,9	0.7	90	4	2.55E+06	82	3.26E+06	8
2,9	0.7	90	5	3.16E+06	82	3.67E+06	8

Data Set	QThresh	Images	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
2,9	0.7	90	6	4.29E+06	82	4.11E+06	8
2,9	0.7	90	7	4.44E+06	82	4.56E+06	8
2,9	0.7	90	8	4.71E+06	82	4.68E+06	8
2,9	0.7	90	9	5.42E+06	82	6.09E+06	8
2,9	0.7	90	10	6.32E+06	82	6.27E+06	8
2,9	0.7	90	11	7.82E+06	82	7.63E+06	8
2,9	0.7	90	12	8.97E+06	82	7.86E+06	8
2,9	0.7	90	13	1.08E+07	82	9.67E+06	8
2,9	0.7	90	3	1.95E+06	86	2.86E+06	4
2,9	0.7	90	4	2.67E+06	86	3.88E+06	4
2,9	0.7	90	5	3.08E+06	86	3.86E+06	4
2,9	0.7	90	6	3.53E+06	86	4.56E+06	4
2,9	0.7	90	7	3.89E+06	86	4.98E+06	4
2,9	0.7	90	8	4.39E+06	86	6.31E+06	4
2,9	0.7	90	9	5.24E+06	86	5.96E+06	4
2,9	0.7	90	10	6.74E+06	86	8.91E+06	4
2,9	0.7	90	11	8.19E+06	86	1.12E+07	4
2,9	0.7	90	12	1.08E+07	86	1.19E+07	4
2,9	0.7	90	13	1.21E+07	86	1.62E+07	4
3,10	0.7	90	3	2.13E+06	82	2.93E+06	8
3,10	0.7	90	4	2.53E+06	82	3.23E+06	8
3,10	0.7	90	5	3.14E+06	82	3.65E+06	8
3,10	0.7	90	6	4.26E+06	82	4.07E+06	8
3,10	0.7	90	7	4.41E+06	82	4.52E+06	8
3,10	0.7	90	8	4.68E+06	82	4.64E+06	8
3,10	0.7	90	9	5.38E+06	82	6.04E+06	8
3,10	0.7	90	10	6.28E+06	82	6.23E+06	8
3,10	0.7	90	11	7.77E+06	82	7.58E+06	8
3,10	0.7	90	12	8.92E+06	82	7.81E+06	8
3,10	0.7	90	13	1.07E+07	82	9.61E+06	8
2,7	0.2	100	3	2.16E+06	17	2.81E+06	83
2,7	0.2	100	4	2.88E+06	17	3.43E+06	83
2,7	0.2	100	5	3.52E+06	17	4.10E+06	83
2,7	0.2	100	6	4.12E+06	17	4.87E+06	83
2,7	0.2	100	7	5.34E+06	17	5.75E+06	83
2,7	0.2	100	8	5.75E+06	17	6.33E+06	83
2,7	0.2	100	9	6.58E+06	17	6.99E+06	83
2,7	0.2	100	10	6.05E+06	17	7.81E+06	83
2,7	0.2	100	11	7.33E+06	17	8.32E+06	83
2,7	0.2	100	12	8.55E+06	17	8.81E+06	83
2,7	0.2	100	13	1.11E+07	17	1.01E+07	83
2,7	0.3	100	3	2.03E+06	52	2.82E+06	48
2,7	0.3	100	4	2.75E+06	52	3.40E+06	48

Data Set	QThresh	Images	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
2,7	0.3	100	5	3.67E+06	52	3.95E+06	48
2,7	0.3	100	6	4.39E+06	52	4.89E+06	48
2,7	0.3	100	7	4.85E+06	52	5.67E+06	48
2,7	0.3	100	8	5.45E+06	52	6.31E+06	48
2,7	0.3	100	9	6.25E+06	52	6.87E+06	48
2,7	0.3	100	10	7.06E+06	52	7.83E+06	48
2,7	0.3	100	11	7.54E+06	52	8.14E+06	48
2,7	0.3	100	12	8.74E+06	52	9.17E+06	48
2,7	0.3	100	13	8.75E+06	52	9.74E+06	48
2,7	0.5	100	3	1.77E+06	86	2.82E+06	14
2,7	0.5	100	4	2.43E+06	86	3.38E+06	14
2,7	0.5	100	5	3.19E+06	86	4.05E+06	14
2,7	0.5	100	6	3.89E+06	86	4.64E+06	14
2,7	0.5	100	7	4.66E+06	86	6.00E+06	14
2,7	0.5	100	8	5.22E+06	86	5.80E+06	14
2,7	0.5	100	9	6.07E+06	86	6.82E+06	14
2,7	0.5	100	10	6.56E+06	86	7.63E+06	14
2,7	0.5	100	11	7.31E+06	86	8.84E+06	14
2,7	0.5	100	12	8.44E+06	86	1.02E+07	14
2,7	0.5	100	13	1.01E+07	86	1.10E+07	14
2,7	0.7	100	3	1.66E+06	98	2.23E+06	2
2,7	0.7	100	4	2.33E+06	98	2.78E+06	2
2,7	0.7	100	5	3.08E+06	98	3.43E+06	2
2,7	0.7	100	6	3.77E+06	98	4.91E+06	2
2,7	0.7	100	7	4.64E+06	98	9.48E+06	2
2,7	0.7	100	8	5.15E+06	98	4.90E+06	2
2,7	0.7	100	9	5.45E+06	98	5.98E+06	2
2,7	0.7	100	10	6.54E+06	98	6.80E+06	2
2,7	0.7	100	11	7.26E+06	98	6.72E+06	2
2,7	0.7	100	12	7.91E+06	98	7.96E+06	2
2,7	0.7	100	13	9.03E+06	98	9.14E+06	2
2,4	0.2	116	3	2.61E+06	23	3.18E+06	93
2,4	0.2	116	4	3.64E+06	23	4.07E+06	93
2,4	0.2	116	5	4.17E+06	23	4.73E+06	93
2,4	0.2	116	6	5.01E+06	23	5.15E+06	93
2,4	0.2	116	7	5.79E+06	23	6.03E+06	93
2,4	0.2	116	8	6.85E+06	23	7.29E+06	93
2,4	0.2	116	9	8.23E+06	23	8.23E+06	93
2,4	0.2	116	10	9.45E+06	23	9.69E+06	93
2,4	0.2	116	11	1.21E+07	23	1.06E+07	93
2,4	0.2	116	12	1.29E+07	23	1.26E+07	93
2,4	0.2	116	13	1.31E+07	23	1.40E+07	93
2,4	0.3	116	3	2.45E+06	58	3.20E+06	58

Data Set	QThresh	Images	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
2,4	0.3	116	4	3.34E+06	58	4.15E+06	58
2,4	0.3	116	5	4.05E+06	58	4.70E+06	58
2,4	0.3	116	6	4.85E+06	58	5.25E+06	58
2,4	0.3	116	7	5.68E+06	58	5.92E+06	58
2,4	0.3	116	8	6.30E+06	58	7.22E+06	58
2,4	0.3	116	9	6.99E+06	58	8.22E+06	58
2,4	0.3	116	10	8.40E+06	58	1.01E+07	58
2,4	0.3	116	11	1.02E+07	58	1.11E+07	58
2,4	0.3	116	12	1.34E+07	58	1.21E+07	58
2,4	0.3	116	13	1.41E+07	58	1.35E+07	58
2,4	0.47	116	3	2.22E+06	97	3.12E+06	19
2,4	0.47	116	4	3.04E+06	97	4.49E+06	19
2,4	0.47	116	5	3.87E+06	97	4.51E+06	19
2,4	0.47	116	6	4.31E+06	97	5.15E+06	19
2,4	0.47	116	7	4.97E+06	97	5.64E+06	19
2,4	0.47	116	8	6.05E+06	97	6.90E+06	19
2,4	0.47	116	9	7.21E+06	97	8.44E+06	19
2,4	0.47	116	10	8.10E+06	97	1.05E+07	19
2,4	0.47	116	11	9.63E+06	97	1.10E+07	19
2,4	0.47	116	12	1.22E+07	97	1.22E+07	19
2,4	0.47	116	13	1.36E+07	97	1.48E+07	19
2,4	0.5	116	3	2.17E+06	100	3.15E+06	16
2,4	0.5	116	4	3.06E+06	100	4.71E+06	16
2,4	0.5	116	5	3.76E+06	100	4.64E+06	16
2,4	0.5	116	6	4.39E+06	100	5.29E+06	16
2,4	0.5	116	7	5.14E+06	100	5.82E+06	16
2,4	0.5	116	8	5.98E+06	100	6.80E+06	16
2,4	0.5	116	9	7.14E+06	100	8.47E+06	16
2,4	0.5	116	10	8.21E+06	100	1.07E+07	16
2,4	0.5	116	11	1.00E+07	100	1.14E+07	16
2,4	0.5	116	12	1.20E+07	100	1.26E+07	16
2,4	0.5	116	13	1.28E+07	100	1.57E+07	16
2,4	0.52	116	3	2.18E+06	102	3.14E+06	14
2,4	0.52	116	4	3.02E+06	102	4.74E+06	14
2,4	0.52	116	5	3.84E+06	102	4.69E+06	14
2,4	0.52	116	6	4.28E+06	102	5.31E+06	14
2,4	0.52	116	7	5.13E+06	102	5.82E+06	14
2,4	0.52	116	8	6.03E+06	102	6.82E+06	14
2,4	0.52	116	9	7.05E+06	102	8.62E+06	14
2,4	0.52	116	10	8.03E+06	102	1.09E+07	14
2,4	0.52	116	11	1.01E+07	102	1.16E+07	14
2,4	0.52	116	12	1.26E+07	102	1.27E+07	14
2,4	0.52	116	13	1.35E+07	102	1.55E+07	14

Data Set	QThresh	Images	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
2,4	0.7	116	3	2.05E+06	109	2.96E+06	7
2,4	0.7	116	4	2.81E+06	109	4.77E+06	7
2,4	0.7	116	5	3.71E+06	109	4.01E+06	7
2,4	0.7	116	6	4.25E+06	109	5.32E+06	7
2,4	0.7	116	7	4.93E+06	109	6.18E+06	7
2,4	0.7	116	8	5.73E+06	109	6.39E+06	7
2,4	0.7	116	9	6.85E+06	109	7.34E+06	7
2,4	0.7	116	10	8.44E+06	109	1.15E+07	7
2,4	0.7	116	11	9.95E+06	109	8.41E+06	7
2,4	0.7	116	12	1.16E+07	109	9.28E+06	7
2,4	0.7	116	13	1.21E+07	109	1.16E+07	7
3,7	0.2	119	3	2.37E+06	32	3.16E+06	87
3,7	0.2	119	4	2.61E+06	32	3.75E+06	87
3,7	0.2	119	5	3.51E+06	32	4.73E+06	87
3,7	0.2	119	6	4.22E+06	32	4.89E+06	87
3,7	0.2	119	7	4.82E+06	32	5.69E+06	87
3,7	0.2	119	8	5.08E+06	32	5.79E+06	87
3,7	0.2	119	9	5.74E+06	32	6.59E+06	87
3,7	0.2	119	10	7.03E+06	32	8.13E+06	87
3,7	0.2	119	11	8.10E+06	32	9.25E+06	87
3,7	0.2	119	12	1.08E+07	32	1.11E+07	87
3,7	0.2	119	13	1.19E+07	32	1.27E+07	87
3,7	0.3	119	3	2.35E+06	57	3.14E+06	62
3,7	0.3	119	4	2.62E+06	57	3.78E+06	62
3,7	0.3	119	5	3.40E+06	57	4.82E+06	62
3,7	0.3	119	6	4.05E+06	57	4.92E+06	62
3,7	0.3	119	7	4.50E+06	57	5.86E+06	62
3,7	0.3	119	8	4.73E+06	57	5.88E+06	62
3,7	0.3	119	9	5.15E+06	57	6.82E+06	62
3,7	0.3	119	10	7.11E+06	57	8.66E+06	62
3,7	0.3	119	11	8.50E+06	57	9.45E+06	62
3,7	0.3	119	12	9.88E+06	57	1.04E+07	62
3,7	0.3	119	13	1.18E+07	57	1.20E+07	62
3,7	0.5	119	3	2.14E+06	101	2.86E+06	18
3,7	0.5	119	4	2.53E+06	101	3.61E+06	18
3,7	0.5	119	5	3.29E+06	101	4.40E+06	18
3,7	0.5	119	6	3.89E+06	101	4.78E+06	18
3,7	0.5	119	7	4.35E+06	101	6.59E+06	18
3,7	0.5	119	8	4.70E+06	101	6.20E+06	18
3,7	0.5	119	9	5.27E+06	101	8.01E+06	18
3,7	0.5	119	10	6.22E+06	101	1.14E+07	18
3,7	0.5	119	11	8.04E+06	101	1.12E+07	18
3,7	0.5	119	12	9.94E+06	101	1.20E+07	18

Data Set	QThresh	Images	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
3,7	0.5	119	13	1.14E+07	101	1.24E+07	18
3,7	0.7	119	3	1.96E+06	112	2.30E+06	7
3,7	0.7	119	4	2.38E+06	112	3.13E+06	7
3,7	0.7	119	5	3.40E+06	112	3.84E+06	7
3,7	0.7	119	6	3.81E+06	112	4.50E+06	7
3,7	0.7	119	7	4.31E+06	112	7.19E+06	7
3,7	0.7	119	8	4.54E+06	112	6.49E+06	7
3,7	0.7	119	9	5.56E+06	112	7.91E+06	7
3,7	0.7	119	10	6.97E+06	112	1.00E+07	7
3,7	0.7	119	11	8.33E+06	112	9.69E+06	7
3,7	0.7	119	12	9.88E+06	112	1.06E+07	7
3,7	0.7	119	13	1.13E+07	112	1.10E+07	7
6,4	0.2	175	3	2.53E+06	43	3.22E+06	132
6,4	0.2	175	4	3.55E+06	43	3.94E+06	132
6,4	0.2	175	5	4.23E+06	43	5.00E+06	132
6,4	0.2	175	6	5.02E+06	43	5.50E+06	132
6,4	0.2	175	7	5.66E+06	43	6.32E+06	132
6,4	0.2	175	8	6.46E+06	43	7.31E+06	132
6,4	0.2	175	9	8.37E+06	43	8.86E+06	132
6,4	0.2	175	10	1.04E+07	43	1.04E+07	132
6,4	0.2	175	11	1.22E+07	43	1.25E+07	132
6,4	0.2	175	12	1.59E+07	43	1.41E+07	132
6,4	0.2	175	13	1.53E+07	43	1.58E+07	132
6,4	0.2	175	3	2.53E+06	43	3.21E+06	132
6,4	0.2	175	4	3.53E+06	43	3.92E+06	132
6,4	0.2	175	5	4.20E+06	43	4.97E+06	132
6,4	0.2	175	6	4.99E+06	43	5.48E+06	132
6,4	0.2	175	7	5.62E+06	43	6.29E+06	132
6,4	0.2	175	8	6.41E+06	43	7.27E+06	132
6,4	0.2	175	9	8.29E+06	43	8.79E+06	132
6,4	0.2	175	10	1.03E+07	43	1.03E+07	132
6,4	0.2	175	11	1.21E+07	43	1.24E+07	132
6,4	0.2	175	12	1.58E+07	43	1.40E+07	132
6,4	0.2	175	13	1.51E+07	43	1.57E+07	132
6,4	0.3	175	3	2.37E+06	86	3.23E+06	89
6,4	0.3	175	4	3.46E+06	86	3.91E+06	89
6,4	0.3	175	5	4.09E+06	86	4.98E+06	89
6,4	0.3	175	6	4.61E+06	86	5.56E+06	89
6,4	0.3	175	7	5.03E+06	86	6.52E+06	89
6,4	0.3	175	8	5.96E+06	86	7.44E+06	89
6,4	0.3	175	9	7.50E+06	86	8.80E+06	89
6,4	0.3	175	10	9.21E+06	86	1.00E+07	89
6,4	0.3	175	11	1.15E+07	86	1.26E+07	89

Data Set	QThresh	Images	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
6,4	0.3	175	12	1.33E+07	86	1.45E+07	89
6,4	0.3	175	13	1.59E+07	86	1.62E+07	89
6,4	0.3	175	3	2.36E+06	86	3.22E+06	89
6,4	0.3	175	4	3.44E+06	86	3.88E+06	89
6,4	0.3	175	5	4.06E+06	86	4.95E+06	89
6,4	0.3	175	6	4.58E+06	86	5.52E+06	89
6,4	0.3	175	7	5.00E+06	86	6.48E+06	89
6,4	0.3	175	8	5.92E+06	86	7.40E+06	89
6,4	0.3	175	9	7.45E+06	86	8.73E+06	89
6,4	0.3	175	10	9.14E+06	86	9.97E+06	89
6,4	0.3	175	11	1.14E+07	86	1.25E+07	89
6,4	0.3	175	12	1.32E+07	86	1.44E+07	89
6,4	0.3	175	13	1.58E+07	86	1.61E+07	89
6,4	0.47	175	3	2.28E+06	141	3.15E+06	34
6,4	0.47	175	4	3.09E+06	141	3.74E+06	34
6,4	0.47	175	5	3.98E+06	141	5.18E+06	34
6,4	0.47	175	6	4.43E+06	141	5.42E+06	34
6,4	0.47	175	7	5.22E+06	141	6.17E+06	34
6,4	0.47	175	8	6.20E+06	141	7.02E+06	34
6,4	0.47	175	9	7.60E+06	141	8.37E+06	34
6,4	0.47	175	10	9.61E+06	141	1.10E+07	34
6,4	0.47	175	11	1.14E+07	141	1.47E+07	34
6,4	0.47	175	12	1.33E+07	141	1.48E+07	34
6,4	0.47	175	13	1.48E+07	141	1.74E+07	34
6,4	0.5	175	3	2.31E+06	148	3.17E+06	27
6,4	0.5	175	4	3.07E+06	148	3.76E+06	27
6,4	0.5	175	5	3.96E+06	148	5.36E+06	27
6,4	0.5	175	6	4.42E+06	148	5.60E+06	27
6,4	0.5	175	7	5.07E+06	148	6.42E+06	27
6,4	0.5	175	8	6.22E+06	148	7.30E+06	27
6,4	0.5	175	9	8.03E+06	148	8.04E+06	27
6,4	0.5	175	10	9.43E+06	148	1.11E+07	27
6,4	0.5	175	11	1.10E+07	148	1.43E+07	27
6,4	0.5	175	12	1.31E+07	148	1.40E+07	27
6,4	0.5	175	13	1.45E+07	148	1.68E+07	27
6,4	0.5	175	3	2.30E+06	148	3.15E+06	27
6,4	0.5	175	4	3.05E+06	148	3.73E+06	27
6,4	0.5	175	5	3.94E+06	148	5.32E+06	27
6,4	0.5	175	6	4.39E+06	148	5.56E+06	27
6,4	0.5	175	7	5.03E+06	148	6.37E+06	27
6,4	0.5	175	8	6.18E+06	148	7.24E+06	27
6,4	0.5	175	9	7.97E+06	148	7.98E+06	27
6,4	0.5	175	10	9.36E+06	148	1.10E+07	27

Data Set	QThresh	Images	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
6,4	0.5	175	11	1.09E+07	148	1.42E+07	27
6,4	0.5	175	12	1.30E+07	148	1.39E+07	27
6,4	0.5	175	13	1.44E+07	148	1.67E+07	27
6,4	0.52	175	3	2.30E+06	152	3.18E+06	23
6,4	0.52	175	4	3.00E+06	152	3.61E+06	23
6,4	0.52	175	5	3.96E+06	152	5.32E+06	23
6,4	0.52	175	6	4.48E+06	152	5.53E+06	23
6,4	0.52	175	7	5.11E+06	152	6.41E+06	23
6,4	0.52	175	8	6.09E+06	152	7.21E+06	23
6,4	0.52	175	9	7.62E+06	152	8.12E+06	23
6,4	0.52	175	10	9.12E+06	152	1.11E+07	23
6,4	0.52	175	11	1.11E+07	152	1.39E+07	23
6,4	0.52	175	12	1.30E+07	152	1.29E+07	23
6,4	0.52	175	13	1.47E+07	152	1.63E+07	23
6,4	0.7	175	3	2.16E+06	165	2.96E+06	10
6,4	0.7	175	4	2.92E+06	165	3.50E+06	10
6,4	0.7	175	5	3.93E+06	165	5.88E+06	10
6,4	0.7	175	6	4.51E+06	165	5.36E+06	10
6,4	0.7	175	7	5.06E+06	165	6.08E+06	10
6,4	0.7	175	8	6.06E+06	165	7.04E+06	10
6,4	0.7	175	9	7.98E+06	165	8.26E+06	10
6,4	0.7	175	10	9.43E+06	165	1.40E+07	10
6,4	0.7	175	11	1.18E+07	165	1.44E+07	10
6,4	0.7	175	12	1.34E+07	165	1.31E+07	10
6,4	0.7	175	13	1.43E+07	165	1.81E+07	10
6,4	0.7	175	3	2.15E+06	165	2.95E+06	10
6,4	0.7	175	4	2.90E+06	165	3.47E+06	10
6,4	0.7	175	5	3.91E+06	165	5.84E+06	10
6,4	0.7	175	6	4.47E+06	165	5.32E+06	10
6,4	0.7	175	7	5.03E+06	165	6.03E+06	10
6,4	0.7	175	8	6.01E+06	165	7.00E+06	10
6,4	0.7	175	9	7.90E+06	165	8.20E+06	10
6,4	0.7	175	10	9.35E+06	165	1.39E+07	10
6,4	0.7	175	11	1.17E+07	165	1.42E+07	10
6,4	0.7	175	12	1.32E+07	165	1.30E+07	10
6,4	0.7	175	13	1.41E+07	165	1.79E+07	10
2,2	0.2	79	15	1.65E+07	13	1.50E+07	66
2,2	0.2	79	16	1.55E+07	13	1.59E+07	66
2,2	0.2	79	17	1.83E+07	13	1.64E+07	66
2,2	0.2	79	18	1.77E+07	13	1.70E+07	66
2,2	0.2	79	19	1.61E+07	13	1.71E+07	66
2,2	0.3	79	15	1.46E+07	31	1.41E+07	48
2,2	0.3	79	16	1.57E+07	31	1.52E+07	48

Data Set	QThresh	Ima ges	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
2,2	0.3	79	17	1.56E+07	31	1.49E+07	48
2,2	0.3	79	18	1.69E+07	31	1.66E+07	48
2,2	0.3	79	19	1.72E+07	31	1.65E+07	48
2,2	0.5	79	15	1.36E+07	57	1.55E+07	22
2,2	0.5	79	16	1.58E+07	57	1.73E+07	22
2,2	0.5	79	17	1.65E+07	57	1.59E+07	22
2,2	0.5	79	18	1.73E+07	57	1.64E+07	22
2,2	0.5	79	19	1.75E+07	57	1.84E+07	22
2,2	0.7	79	15	1.31E+07	72	1.24E+07	7
2,2	0.7	79	16	1.50E+07	72	1.54E+07	7
2,2	0.7	79	17	1.56E+07	72	1.58E+07	7
2,2	0.7	79	18	1.72E+07	72	1.74E+07	7
2,2	0.7	79	19	1.66E+07	72	1.96E+07	7
2,4	0.2	116	15	1.56E+07	23	1.57E+07	93
2,4	0.2	116	16	1.93E+07	23	1.79E+07	93
2,4	0.2	116	17	2.15E+07	23	1.96E+07	93
2,4	0.2	116	18	2.47E+07	23	2.17E+07	93
2,4	0.2	116	19	1.90E+07	23	2.11E+07	93
2,4	0.3	116	15	1.48E+07	58	1.53E+07	58
2,4	0.3	116	16	1.76E+07	58	1.79E+07	58
2,4	0.3	116	17	1.81E+07	58	1.96E+07	58
2,4	0.3	116	18	2.05E+07	58	2.11E+07	58
2,4	0.3	116	19	2.18E+07	58	2.06E+07	58
2,4	0.5	116	15	1.59E+07	100	1.72E+07	16
2,4	0.5	116	16	1.95E+07	100	2.09E+07	16
2,4	0.5	116	17	1.97E+07	100	2.18E+07	16
2,4	0.5	116	18	2.19E+07	100	2.29E+07	16
2,4	0.5	116	19	2.16E+07	100	2.22E+07	16
2,4	0.7	116	15	1.50E+07	109	1.37E+07	7
2,4	0.7	116	16	1.67E+07	109	2.23E+07	7
2,4	0.7	116	17	1.81E+07	109	1.94E+07	7
2,4	0.7	116	18	2.12E+07	109	2.06E+07	7
2,4	0.7	116	19	2.14E+07	109	2.14E+07	7
6,4	0.2	175	15	2.13E+07	43	1.82E+07	132
6,4	0.2	175	16	2.07E+07	43	2.00E+07	132
6,4	0.2	175	17	2.17E+07	43	2.16E+07	132
6,4	0.2	175	18	2.52E+07	43	2.47E+07	132
6,4	0.2	175	19	2.63E+07	43	2.68E+07	132
6,4	0.3	175	15	1.87E+07	86	1.79E+07	89
6,4	0.3	175	16	1.95E+07	86	1.94E+07	89
6,4	0.3	175	17	2.10E+07	86	2.15E+07	89
6,4	0.3	175	18	2.29E+07	86	2.41E+07	89
6,4	0.3	175	19	2.55E+07	86	2.61E+07	89

Data Set	QThresh	Ima ges	Clusters	Avg Conv. Exec.(us)	Conv. Images	Avg Not Conv. Exec.(us)	Not Conv. Images
6,4	0.5	175	15	1.74E+07	148	1.96E+07	27
6,4	0.5	175	16	1.93E+07	148	2.10E+07	27
6,4	0.5	175	17	1.97E+07	148	2.15E+07	27
6,4	0.5	175	18	2.25E+07	148	2.55E+07	27
6,4	0.5	175	19	2.45E+07	148	2.52E+07	27
6,4	0.7	175	15	1.75E+07	165	1.69E+07	10
6,4	0.7	175	16	1.90E+07	165	1.90E+07	10
6,4	0.7	175	17	2.02E+07	165	2.43E+07	10
6,4	0.7	175	18	2.21E+07	165	2.70E+07	10
6,4	0.7	175	19	2.48E+07	165	2.33E+07	10