CALCULATING STAIRCASE SLOPE FROM A SINGLE IMAGE

A Thesis

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Nicholas Clarke

June 2015

COMMITTEE MEMBERSHIP

TITLE:	Calculating Staircase Slope from a Single Image
AUTHOR:	Nicholas Clarke
DATE SUBMITTED:	June 2015
COMMITTEE CHAIR:	John Seng, Ph.D. Associate Professor of Computer Science
COMMITTEE MEMBER:	Jane Zhang, Ph.D. Professor of Electrical Engineering
COMMITTEE MEMBER:	Hisham Assal, Ph.D. Associate Professor of Computer Science

ABSTRACT

Calculating Staircase Slope from a Single Image

Nicholas Clarke

Realistic modeling of a 3D environment has grown in popularity due to the increasing realm of practical applications. Whether for practical navigation purposes, entertainment value, or architectural standardization, the ability to determine the dimensions of a room is becoming more and more important. One of the trickier, but critical, features within any multistory environment is the staircase. Staircases are difficult to model because of their uneven surface and various depth aspects. Coupling this need is a variety of ways to reach this goal. Unfortunately, many such methods rely upon specialized sensory equipment, multiple calibrated cameras, or other such impractical setups. Here, we propose a simpler approach.

This paper outlines a method for extracting the slope dimensions of a staircase using a single monocular image. By relying on only a single image, we negate the need for extraneous accessories and glean as much information from common pictures. We do not hope to achieve the high level of accuracy seen from laser scanning methods but seek to produce a viable result that can both be helpful for current applications and serve as a building block that contributes to later development.

When constructing our pipeline, we take into account several options. Each step can be achieved with different techniques which we evaluate and compare on either a qualitative or quantitative level. This leads to our final result which can accurately determine the slope of a staircase with an error rate of 31.1%. With a small amount of previous knowledge or preprocessing, this drops down to an average of 18.7% Overall, we deem this an acceptable and optimal result given the limited information and processing resources which the program was allowed to utilize.

ACKNOWLEDGMENTS

They say it takes a village to raise a child. No matter how great a single man is, there are heights he can not reach without the help from others. There are a lot of people that have helped me along this path, people who have provided me with advice and connected with me both on a technical level and an emotional level. Here is my list of mentors and friends, supporters and advocates, inspirations and dedications.

- First, I want to thank God for guiding my life, giving purpose and joy.
- My parents for their understanding and input, for knowing when to help me, when to give me space, and when to push me. My brothers and sisters, because after God, family is first. When everybody is absent, family is there.
- My grandparents for giving me food, lodging, love, and constantly praying.
- My advisor and committee members: Dr. Seng, Dr. Assal, and Dr. Zhang.
- My friends who gave me the education outside the classroom: Luke, Brandon, Antonio, Matt, Tyler, Dan, Luke, David, Jenna, Merry, Jared, Derek
- The guys who thought I was leading them but probably never realized how much they taught me: Dannie, David, Jack, Jon, John, Wes, Spencer, Nathan
- The beautiful men who traded their time for the solutions to my problems: Lucas, Griffin, Curtis, Tim, Bradley, David
- My csc and cpe friends who could talk in code but also liked kicking it on weekends: Kyle, Kristen, Kaitlyn, Stephen, Andrew, Nico
- The teams who let me play soccer with them and put up with my antics on the field: Bakery #3, Chiller Whales, Speed Bus, Team America

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Depth detection is a significant part of vision perception. An individual human eye is limited by its ability to only relay information in the 2-dimensional spectrum, leading humans to require a second eye for triangulation. Humans also remember the depths of previously seen objects as well as seamlessly detect critical occlusion features. When trying to mimic this ability, a computer system has a few options. The prominent option is to use an additional sensor, such as a laser or sonar system, which can interact with the environment in ways outside of human capabilities. The other main option is to use multiple cameras and calculate the location geometrically in a similar fashion to human vision. A third variant will use any number of cameras and record motion, stitching together several angles to simulate additional cameras. Of course these methods can be combined or extra cameras can be added to give a wider field of view. However, one method that is often overlooked is depth detection from a single image produced by single monocular camera.

1.1 Advantages of Single Image Depth Detection

Despite its initial shortcomings, there are advantages to using only a single image in determining depth. The foremost reason is memory space. Storing and processing a single image is magnitudes simpler than the resources required for video. It is also simpler than saving the extra sensor data and does not require the positional overhead seen in a multiple camera system. A second underrated advantage in single image depth detection deals with focus. Methods involving multiple cameras or video will have a specific point that is perfectly clear while all other parts of the image will gradually be out of focus as you

radiate away from that focal point. This affects the accuracy of triangulation as images are stitched together. It can also lead to knowledge gaps and blank areas. A single image will rarely suffer from this unless huge portions are trying to be captured. Another advantage provided by monocular images is their prevalence in today's society. With the improvements in cellphones and mobile devices that contain embedded cameras, more and more pictures are being taken. Since all the necessary information is already being stored, these images can be retroactively used to determine depth within a scene. Scholars agree that there is indeed benefits provided only by monocular images and thus research into this area has continued.

1.2 Motivation for Determining Depth

We can see a series of circumstances where it would be useful to accurately determine the slope of a staircase. The first of these is to determine the best way to traverse through an environment by creating a realistic layout. Within the realm of movement a common problem is ascending stairs. Being able to accurately estimate the slope of a staircase will greatly help in reducing accidents, such as falling or slipping. This is true whether in the case of an autonomous robot or a visually handicapped human. There is a second application also related to virtual environments. With an increasing tourism industry and entertainment industry, it is becoming more common to model a real space into the virtual space. If there was a simple way to determine the dimensions of any architectural aspect, it would lead to a reduction in cost and effort. Nevertheless, both of these use cases already have solutions. It is fruitful to look into areas where our program would be able to solve a currently unsolvable problem. The first of these is in building planning and inspection. After the construction phase, an inspector must manually travel to the building's location to ensure all standards are being upheld. Our program would introduce a way to do this remotely, a very efficient improvement. The final application is where our program truly shines. Because we place almost no restrictions on the inputted images, it allows the program to use images from the past that were taken without the intention of any depth investigation.

This retroactive analysis is especially useful in criminal investigations. As we already noted, staircases can be dangerous areas, especially during large disasters when exit routes must flow quickly and smoothly. From this we can see how useful it would be in determining fault if the slope of the staircase could be guaranteed to have met all safety requirements. As we have enumerated, there are several applications, not just where our program can improve upon the current methods but also introduce solutions that did not exist prior to our work.

1.3 Breakdown of Proposed Method

We will separate our proposed implementation into three stages for easier explanation. The first stage, diagrammed in Figure 1.1, will focus on determining the depth values within the image. Because this section requires minimal knowledge of the objects within the scene, it can be done before any image analyses.

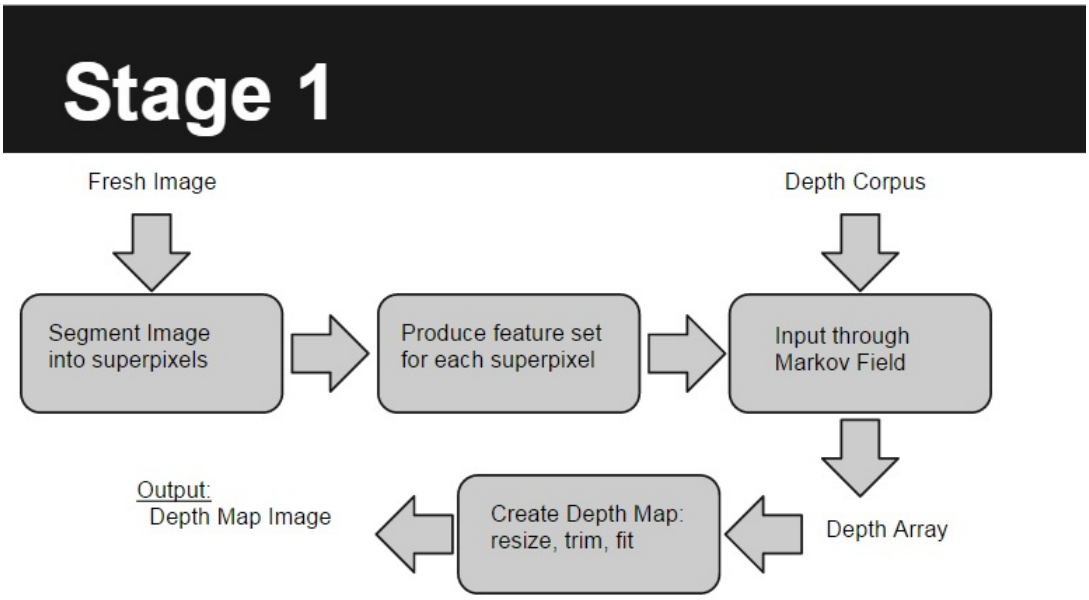


Figure 1.1: Stage 1 of Implementation

The input of this stage will be the fresh image which will produce a depth map detailing the distance of each of the image's pixels from the camera point.

Our main tool in this process will be a Random Markov Field. This is a common artificial intelligence implementation that uses the information from a training set to build a network. When a new instance comes along, the field makes use of the known probabilities of the dependent variables to form predictions. A sample input image can be seen below in Figure 1.2 with the output depth image viewable in Figure 1.3.

Figure 1.2: Sample Fresh Image as Input to Stage 1

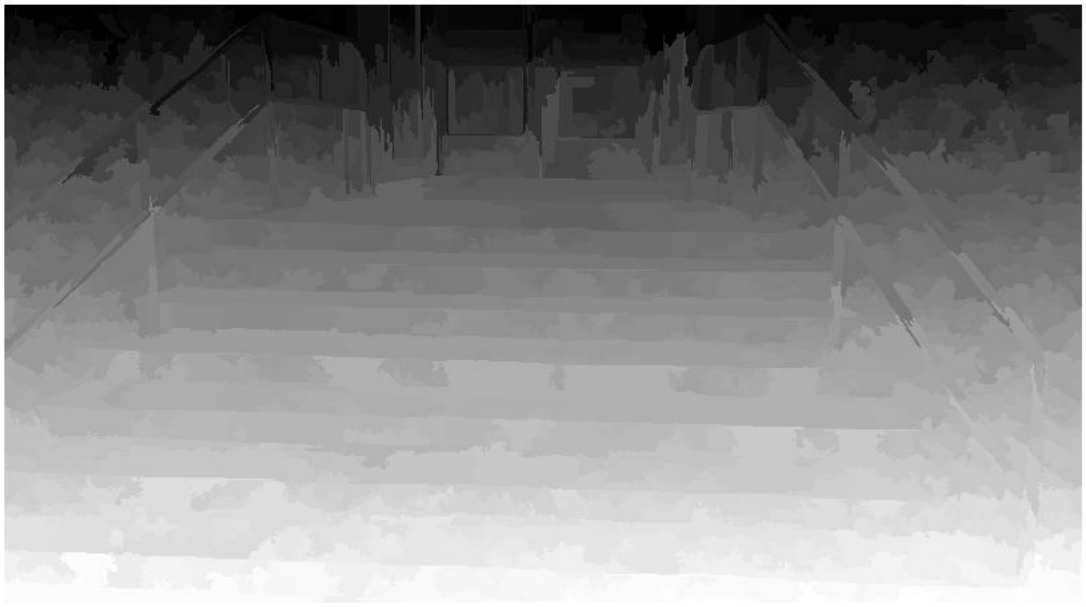


Figure 1.3: Sample Depth Map Output from Stage 1

Stage 2, diagrammed in Figure 1.4, will dive into the image analyses. We will use a series of filters and feature detection algorithms to isolate the relevant regions and begin to build a model of the area under inspection. An important part of this stage is detecting false positives. The edge detectors will inevitable locate more lines than those singularly involved with stairs. Our program will be tasked with the challenge to discard those points that are most likely not included in the staircase. The model will join with the depth map from stage 1 to provide another dimension to the analysis. The output will be a list of points across the 3D space.

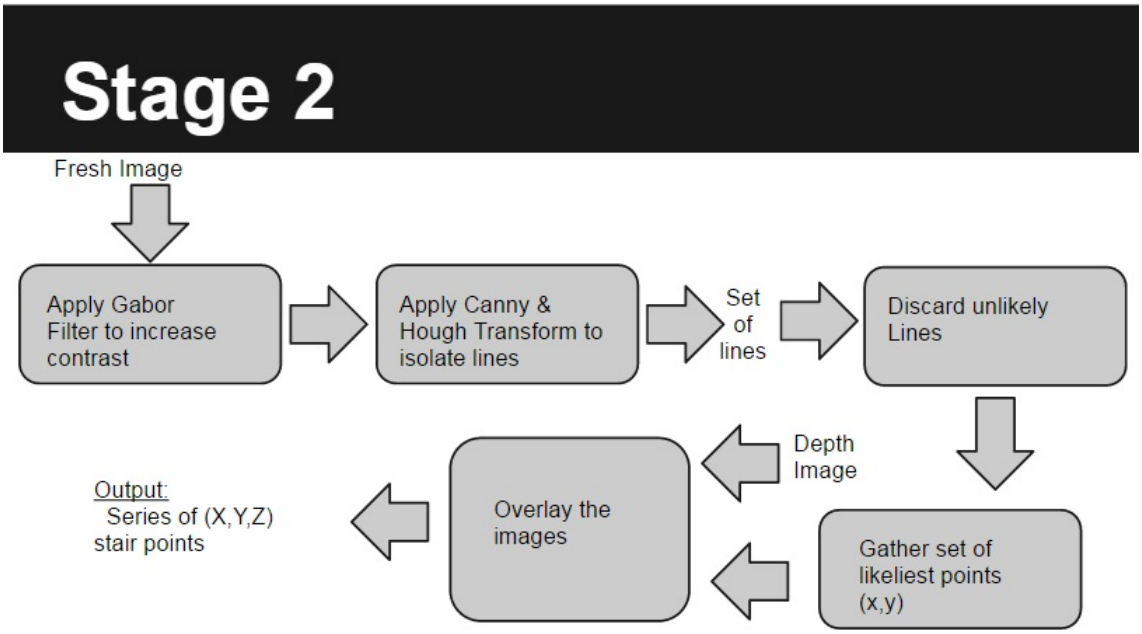


Figure 1.4: Stage 2 of Implementation

Finally, in stage 3, diagrammed in Figure 1.5, the program takes on a more statistical approach. Using regression techniques and a similar methods of tracking down outliers, the 3D model is trimmed and configured. A plane equation is found that produces the best fit for the overall data-set. From this plane, the slope and dimensions can be extracted. Thus the resulting output of the program is achieved.

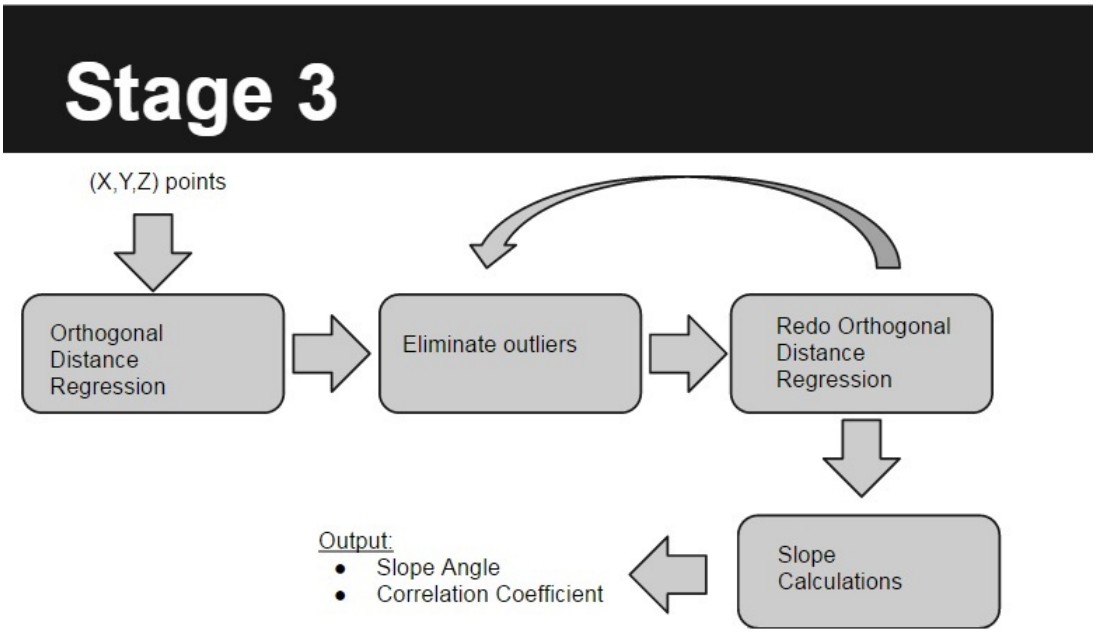


Figure 1.5: Stage 3 of Implementation

1.4 Chapter Overview

In chapter 1 we have introduced the topic and explained why we are pursuing this idea. Chapter 2 explains the related work and inspirations for our research. In chapter 3 we outline the purpose and explain in more detail the usage we envision for our system. Chapter 4 explains the implementation of our system. Chapter 5 shows the results and compares our outcomes to other works. In chapter 6 we review the basic reasoning behind our methodology and discuss avenues of future work. In chapter 7 we conclude and remark upon our contributions within this paper.

Chapter 2

Background

This chapter presents some general information that may be useful for completely understanding the later chapters.

2.1 Image Segmentation

The field of computer vision has been developed much in the past years, elaborating upon the way images are analyzed. The first act to studying almost any image is segmenting the image into regions. One of the most common ways is to cluster together similar pixels. Other common methods include region-growing, thresholding, and expectation-maximization. We researched a few lesser-known techniques to see if they could be modified for this specific problem. In 1998, Hiroshi Ishikawa and Davi Geiger wrote an applicable paper[9]. As opposed to many of the methods focusing on intensity gradients, this work identifies major junctions throughout the image and calculates the smallest set of regions to maintain distinctions among those junctions. It works especially well in junctions where several objects come together. In 2008, Michael Maire et al proposed a similar method of segmentation[14]. Trying to overcome the difficulty in correctly identifying junctions, they combine local features with the image's overall global features to label all the contours. A third notable paper on segmentation comes from Pedro Felzenszwalb, published in 2004[7]. It describes a way to organize the image into a graph and compare neighbors, subsequently merging nodes and forming larger regions. This method will be discussed in detail later as it get implemented for this project. There are numerous other segmentation methods, but none had an influence over the outcome of this project.

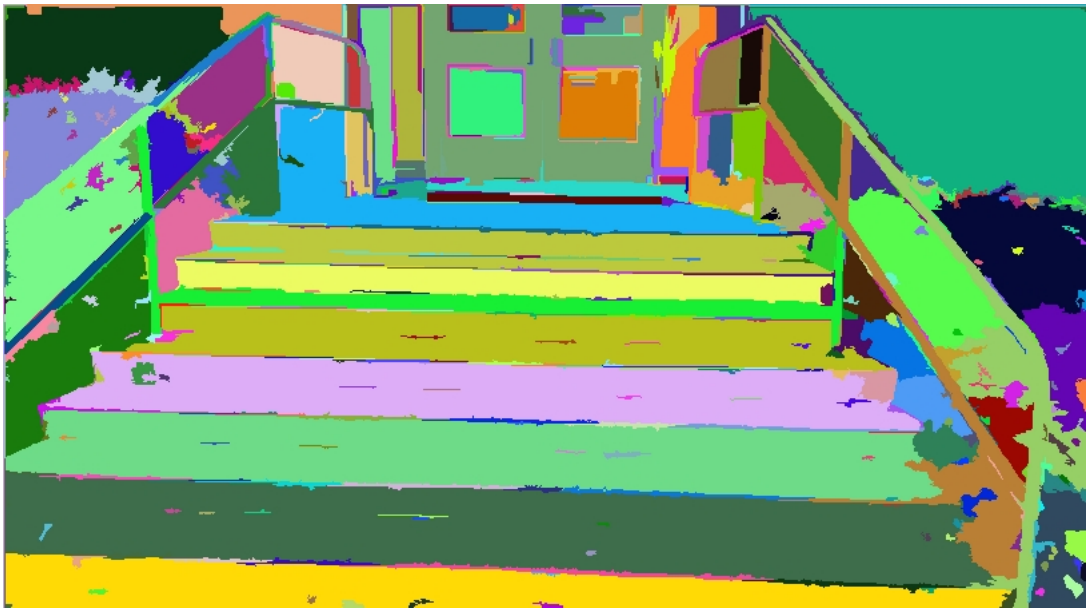


Figure 2.1: A Sample of a Segmented Image

2.2 Region Occlusion in Depth Detection

After the difficulty of image segmentation comes occlusion. An occlusion occurs when one region in the image overlaps another region. Figure 2.2 shows an example of image occlusion illustrated with four balls. As can be seen, occlusion can be a challenge when ordering images based on their relative depth tot he user. There were two main papers that influenced the design of this project, both being published in 2011. The first was written by Guillem Palou and Philippe Salembier[17]. Their proposed method uses a binary tree structure to iteratively merge regions, paying special attention to T-junctions. It compares the color, area, and contour of adjoining regions as the main elements in determining distance between them, in order to resolve merging order. Our program also makes use of a binary tree to store regions but not for occlusion purposes. The other paper is written by Derek Hoiem et al[4]. This paper applies the theory behind grouping principles and pattern finding to form a hierarchical segmentation model. They employ a conditional random field model to iteratively estimate the proper occlusion depths. This is similar to the Markov Random Field that we will implement later.

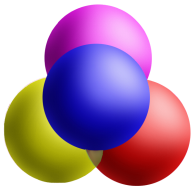


Figure 2.2: Image Occlusion Illustrated with Balls

2.3 Overall Method for Determining Depth

Although segmentation and occlusion are important parts in depth detection, there can still be differences in the overall method. An early example of this is a paper from 2007 by Erick Delage, Honglak Lee, and Andrew Ng[6]. This paper discusses the certain 3D information that can be inferred from an image if there is a small amount of prior knowledge. In that case, they discussed the instance of a "Manhattan World Scene" which is constricted to an indoor environment where the walls are 90 degrees in the corners and the camera is parallel to the ground. The ideas these authors presented were developed, and in 2008 Ng teamed up with Ashutosh Saxena and Min Sun to publish[1]. The researchers built a learning program that could be trained to identify depth in a picture called Make3D. In addition to their written ideas, they also made a vast amount of their code source available for free download[1]. Unfortunately, they admit that "The code will not run out of the box. Therefore, unless you have read the ICCV-3dRR paper, it would be hard to make it run.[1]" This was to reflect the fact that they wanted the code to be versatile enough for several projects so they posted a block of code that could be modified and fit to a user's specific desires. The first half of the Make3D code is used to build the feature set for each region. The feature set is based around three properties. The first is texture variations. To quantize this, they apply Laws' masks which gives the amount of intensity energy from each texture. The second is texture gradients. This is through an implementation of edge filters where each each filter is spaced at intervals of 30 degrees. The third property is color. Color

[1]. The project is currently hosted on the Cornell website located at http://make3d.cs.cornell.edu/

qualities are extracted by applying another local averaging filter to each of the three separated color channels. Additionally, the algorithm takes into account the global position of the pixel. This helps to identify the size of each object. It is also useful in distinguishing the distant sky and the ground plane from perhaps a blue building or green backdrop. The feature set is calculated on three different scales in order to account for distance haze and local effects, a process outlined in more detail in the related paper. This method will be revisited during the implementation section of this paper as it forms the basis on which much of our work is built. In 2014, a paper focusing more on the statistical aspect was published by Youngjung Kim et al[20]. It describes a method of depth estimation within a single image using a more statistical approach. They find similarities in the scenes within the image and compare them to the depths of known scenes using weighted median statistics. An additional reason this paper is interesting, comes from the fact that they use the same training set that is employed by our algorithm which was collected by Ng's team for the Make3D project. Another paper was written in 2014, by Lizhi Zhang et al[12]. The methodology they lay forth in this paper has many similarities to the method we are attempting to create. They glean the image's texture features using Laws' filters then input those features into a trained least-square depth estimation program. The program will then find the best solution to encompass as much of the given data as possible. The feature identification is not only calculated on a pixel basis but scaled up three different proportions to account for global features and overall object sizes. Seeing the benefits, this was a characteristic we wanted to include in our program. These papers went to formulate a solid basis to build our learning algorithm.

2.4 Alternative Methods

Alongside the development of monocular image depth identification, researchers attempted to solve the problem of staircase modeling. As mentioned previously, a common way to provide depth was through the use of sensors. A recent paper, published in 2014 by Alejandro Perez-Yus, is a prime example[13]. The

goal of the paper is similar in that they are trying to model the staircase for easier navigation. They must overcome similar problems of occlusion and feature classification. However, through the use of a head mounted range-finding depth sensor, the result is able to contain more detail, such as the number of steps and specific dimensions. It is important to analyze the results found here and in subsequent papers, as they provide a benchmark to which our program can be compared. A related paper, written by Titus Tang et al. in 2012[19], uses a comparable methodology of gathering information but has different processing. They collect images and depths of staircases using a helmet equipped with a Kinect, a camera, and accelerometers. Once the information has been obtained, the program uses a preemptive random sample consensus (RANSAC) to try fitting parallel planes throughout the model. They wisely admit that "applications related to the detection and navigation of staircases involve a balance of trade offs between system robustness and quality". This means they sacrifice a higher quality for wider adaptability, a consideration that would be applied in our program as well. The next paper is published in 2013 by Jeffrey Delmerico et al.[11]. This paper incorporates the stairway estimation with path planning and traversal. Mounting a Kinect sensor on a mobile robot lets the robot detect geometric cues of the staircase and calculate its dimensions. It also uses the multiple viewpoints from the motion to stitch together the 3D model. A similar paper was published by Joel Hesch et al. in 2010[10]. Descending is difficult because of the limited appropriate field of view. The proposed system does not use a depth sensor but instead uses a camera mounted on top of a robotic autonomous vehicle. The camera detects features such as texture, geometry, and optical flow to create a model from motion as the vehicle approaches the stairway. Related by the modeling-from-motion concept is a paper written in 2011 by Florin Oniga and Sergiu Nedevchsi[16]. Although the goal of finding a curb from a moving car may seem different from the previous staircase detectors, it shares many similarities which is why it is fruitful to scrutinize their methods. Several of the techniques they presented will be modified to be translated to our solution including Hough transforms, random sample consensus, and least square fitting. The final relevant paper that was investigated to prepare the background

for this program was written by Stephen Se and Michael Brady in 2000[18]. Despite this research being fifteen years old, it is still highly significant to our task. Their data consists of images taken as a user is approaching a potential staircase. The algorithm does a combination comparison of the images and combines results from Gabor filters with edge detection to outline the staircase. The depth relies on knowledge of the camera position and the viewing angle to provide triangulation between the images.

The background research done beyond the sources mentioned was used to fill in the gaps but did not add any major contributions to the outcome. A later section will analyze potential methods and solutions that were not implemented and the reasoning behind those decisions. Nevertheless, these related works have provided a solid framework from which to tackle the outlined task.

Chapter 3

Project Motivation

3.1 Purpose and Envisioned Usage

The purpose of this project, as it was alluded to earlier, involves the correct identification and dimensional analysis of a staircase. Figure 3.1 shows an ideal staircase that has been labelled. In the picture, A is the staircase slope, C is the step height, D is the step width, and E is the step depth. The value of A is what we are attempting to find with our system. Navigation can be a difficult task, both for humans with limited eyesight and autonomously operated vehicles. Having the knowledge of the staircase slope makes traversing it much more feasible. It also allows the user to determine if traversing that staircase is unmanageable because it is too steep, avoiding a potential fall. Another function for this program would be the application of building codes. By submitting a simple picture of a staircase, the property manager could prove that the building meets the proper requirements. This saves the times of a building inspector and eases the approval process. A third application involves virtual tours. As virtual environments are becoming more popular, there is an increasing interest in translating real world spaces into a graphical computer space that could be explored and remotely. The transitions provided by stairs are a critical part of any multi-story space, which perfectly folds into the functionality of our program. A major advantage to our system is the lack of special equipment and the post-capture decision. This is means that it could be applied to any image and that the decision to apply our program does not affect how the picture is taken and thus does not need to be determined at the moment of capture. This leads into a fourth application, related to crime scene investigations. After accidents or disasters, the architecture of

the building is often analyzed to determine if any fault lies with the building. As staircases can be a critical component in the exit route, it is scrutinized closely. However, it is often not feasible to visit the stairs if they have been altered or destroyed. By using pictures or segments from security footage, the investigators can determine if the staircase was being properly maintained and if all safety precautions were being observed. This ability to look at past pictures is a large contribution and factor that differentiates our work from previous studies.

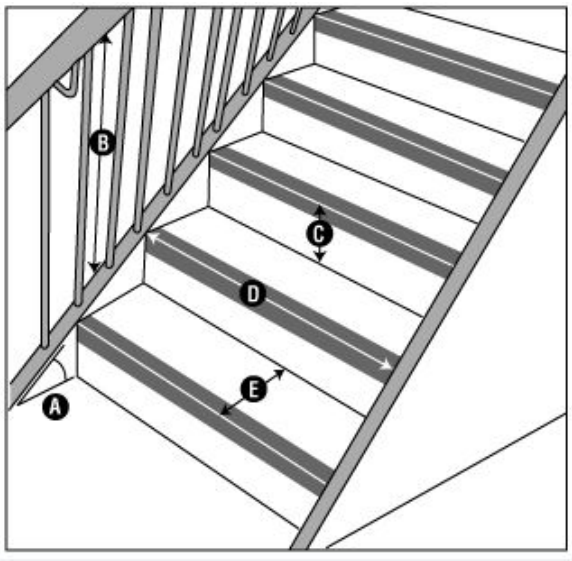


Figure 3.1: An Illustration of a Labelled Staircase

3.2 Image Input Guidelines

We place a few minor restrictions on the images we expect to receive as input. Because part of the underlying goal is to mimic human perception, most of the requirements are those that would also restrict a human. The image should be of RGB format taken approximately parallel to the ground plane. If it is grayscale, it is much more difficult to determine lighting conditions. If the image is taken at a large slant or from an uncommon perspective, it would even be difficult for a human to perceive the correct dimensions. This can potentially be fixed through cropping or extra context cues but will most

likely still produce an erroneous result. We have not placed any limitations on the images size. One of the first steps within the second stage of the program is to resize the image to something manageable. If the image is too distant or blurry, it may be impossible to pick out the specific details of the staircase. These factors follow basic image guidelines and so include most images a user would normally be capturing.

Chapter 4

Implementation

The process of determining the slope of the staircase can be broken into three stages for easier explanation. The first stage is the depth stage where the program will determine each pixel's depth. The second stage is the stair stage where the program will identify which points in the image belong to the staircase. The third stage is the regression stage where the information of the previous two stages are combined and the resulting best-fit slope is calculated.

4.1 Stage 1: Depth Extraction

The first of these stages is the depth stage. The goal is to glean a map of the entire image that characterizes the true depth of each pixel. This is done through the implementation of a learning algorithm that gathers and compares the features of every region of the image to a known dataset.

The first step is to segment the image. Every image is made up of a series of identically shaped pixels. However, each image can also be broken into larger unique non-overlapping regions based on their similarity levels. There exists a multitude of methods used to segment an image. After researching several, including everything from a basic watershed segmentation to the junction based grouping mentioned above, we decided to use a graph based approach.

This is based on the research done by Pedro F. Felzenszwalb from the Massachusetts Institute of Technology and Daniel P. Huttenlocher from Cornell University described in their paper [7]. This method returns a newly created graph where each node is a tree representing a region within the image. It begins by assuming each pixel is a element within the graph connected to its eight neighboring pixels (expect for edge pixels which will have less connections).

Each element is assigned a threshold value and each edge is assigned a weight. The algorithm then iterates through each edge and compares neighboring elements. If the combined threshold value is below the internally calculated threshold value then the combination is deemed acceptable and the elements are combined. If instead the threshold value is too high, then the segments are too different and thus are not combined. The final product will be the top level graph where each node encompasses multiple pixels. Each of these nodes are labeled as superpixels. It is important to note that this method attempts to create an over-segmented result. This means that an object in the image may be composed of several superpixels but there will never be a superpixel containing pixels from multiple objects. Since the threshold function includes no probability, this process is not stochastic, meaning an image will generate an identical graph every time the algorithm is executed.

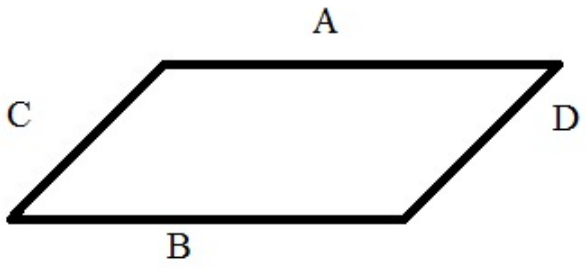


Figure 4.1: The Typically Envision Parallelogram with Labeled Sides

In order to better apply Felzenszwalb's graph-based segmentation [7] to the stated problem of staircase identification, the way the threshold value was calculated had to be adjusted. The algorithm initially computed the threshold based on the size of the components being merged, where the size was simply the number of pixels the component contained. A typical stair step is a four-sided polygon whose width is usually larger than its height. The shapes most often encountered are parallelograms and trapezoids, with other shapes being encountered in more artistic instances, such as spiral or curved staircases. Since trapezoidal steps are usually only encountered at the beginning of a staircase and since they can be approximated to rectangles, the algorithm was

modified to promote the forming of parallelogram shapes when segments were combined. This was also necessary as a rectangular step viewed from the side will similarly look like a parallelogram. A typical parallelogram can be seen in Figure 4.1.

Algorithm 1 Segmentation algorithm

```
1: procedure GRAPH BASED SEGMENTATION
2:     elements ← list of pixels
3:     edges ← starts empty
4:     for each element do
5:         add neighboring edges
6:     if using Size Threshold then
7:         Threshold ← (numberofpixelsinelement)
8:     else if using Simple Threshold then
9:         Threshold ← (equivalenceofsidelengths)
10:        Threshold ← (length − height)
11:        Threshold ← (approximatearea)
12:    else if using Diagonal Angles Threshold then
13:        Threshold ← (equivalenceofsidelengths)
14:        Threshold ← (length − height)
15:        Threshold ← (approximatearea)
16:        Threshold ← (differenceindiagonalangles)
17:    for each edge do
18:        if segmentation_constant > threshold(edge) then
19:            combineelements
20:            add neighboring edges
21:    return ← list of edges to make up graph
```

As can be seen, both pairs of opposite sides, (C and D) as well as (A and

B), are of equal length. This leads to the first value in calculating the new threshold, equivalence of length of sides. Within a staircase, it can be expected that the breadth of a stair is larger than the height. Thus the next value to contribute to the threshold is how much larger the length is to the height. The four points of a parallelogram can be used to calculate the area of the shape. This value was able to replace the previous notion of size. Finally, it can be derived that opposite angles across the diagonal are equal. In the figure, the angle AD is equal to CB and BD is equal to AC. The criterion of matching angles underlines the fundamental definition because if the angle are equal, the shape is guaranteed to be a parallelogram. This angle equivalence becomes the final value to build the threshold. Because the angles can be difficult to calculate within an image and the points were not exact representations, the induced error of the algorithm was compared both when the angle feature was included and when it was excluded. The entire algorithm is detailed in Algorithm 1.

Focusing on different features and providing different weighting factors were able to produce a variety of different segmentations. An example of this is seen in Figure 4.2 below. The rightmost image is the original image; the left 4 images show different segmentations. Not all these segmentation strategies were analyzed. For example, the fourth left image is poorly segmented and would most likely produce an inaccurate result so it was discarded. The segmentations we chose to analyze are explained later in the results section.

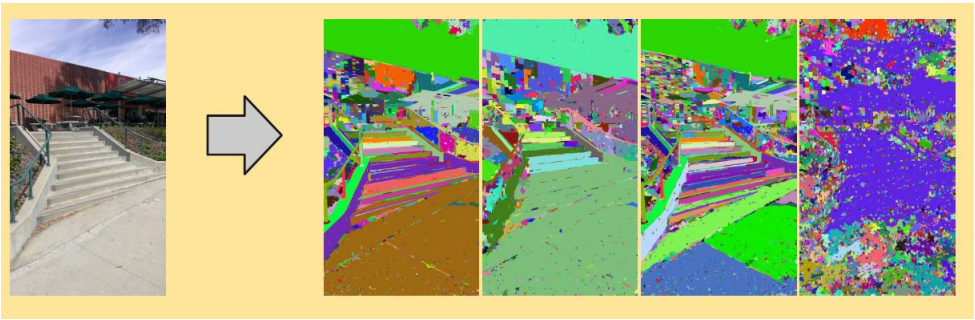


Figure 4.2: Different Threshold Values Producing Different Segmentations

Besides the specific locations of each superpixel, using different threshold values would produce a different overall number of superpixels. This provides another comparison to determine if there is any correlation between the number of superpixels provided and the accuracy of the slope calculation.

Once the image has been fully segmented, the next step in the stage is to produce a feature set for each superpixel. This is necessary in the classification of each segment for the depth comparison analysis.

This step and the next borrow much of the methodology and code used by a series of researchers at Stanford University. In 2007, Ashutoush Saxena, Sung H. Chung, and Andrew Y Ng published the paper "3-D Depth Reconstruction from a single still image"[2]. A year later in 2008, Saxena and Ng collaborated in conjunction with Min Sun to publish "Make3D: Depth Perception from a Single Still Image" which was briefly mentioned in the related works section[1]. After a series of modifications, the code has been customized to our needs and now produces a usable result. The first part of the program that deals with building the feature set for each region, remained basically the same in our program as in the original research. Detecting special additional features within an image was discussed, but the idea was discarded based on depth of our project and the accuracy of our results. It is a potential avenue for future work.

With the segmented image and each pixel's corresponding feature set, the next step of the process was to incorporate a Random Markov Field (RMF)[15]. The RMF is used to predict a representing depth for each pixel.

An RMF is an undirected graphical model where each element within the graph includes a Markov property. An element with a Markov property will have its next state determined without regard to its previous state, thus disregarding need for memory. It is a tool often used within artificial intelligence applications to represent dependencies outside the scope of a Bayesian network. Our program implements a typical RMF already coded into the Make3D program. The RMF needs to be trained and formed from a large sample data. We accomplish this using the same data set included with Make3D.

The Make3D training data set is an openly available series of images with their corresponding depths. The image resolution of each is 2272x1704. The depth component is found using a laser range finder, and is measured in meters. The set contains 534 images. Further details on how the dataset was gathered and the implementation of the RMF are contained in the Make3D documentation.

Before the superpixel graph is inputted through the RMF, a few steps of preparation are required. This includes aligning the superpixels onto the straight line boundary to better prepare for each plane's parametrization. The program will then use a multiple segmentation heuristic to determine occlusion decisions. The RMF will take this as a starting point and begin assigning weights and creating neighboring relationships. The output from the RMF is an array that relates each pixel in the image to a corresponding depth value.

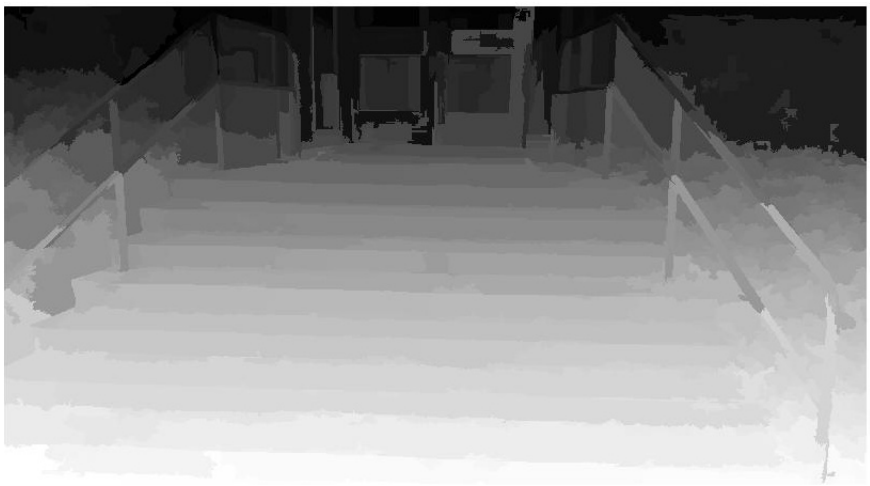


Figure 4.3: Depth Map of Iimage 1 with Size Segmentation

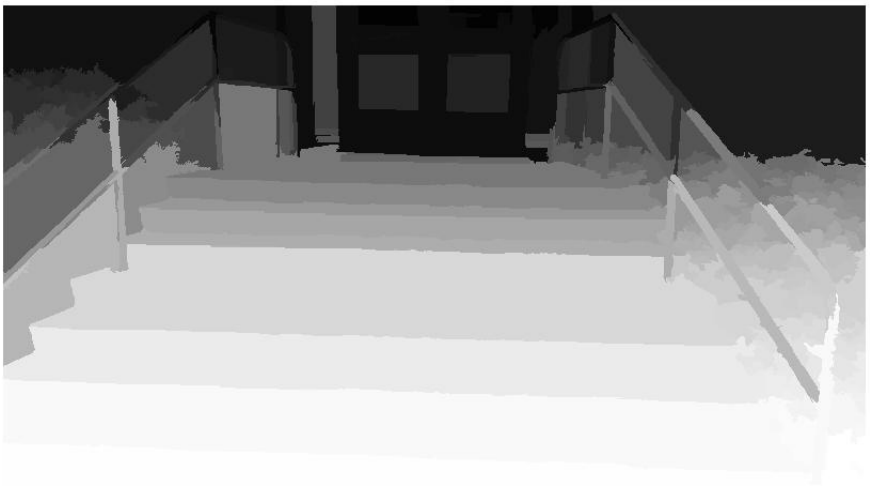


Figure 4.4: Depth Map of Image 1 with Non-Size Segmentation

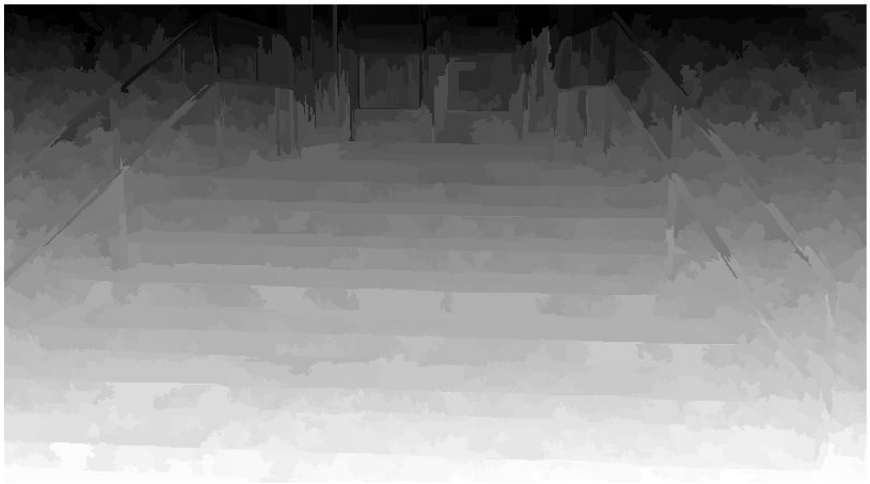


Figure 4.5: Depth Map of Image 1 with Customized Diagonal Segmentation

The final step within this first stage is to transform the array into a more usable framework. This is done by trimming the image down and resizing the result. The initial outcome from the RMF has a border of zero values which must be eliminated. The resulting array is then transformed into a depth map image that can be overlayed on top of the original image. Three different depth maps are shown in Figure 4.2, Figure 4.4, and Figure 4.5, each showing the output from the first image in our test set. The three segmentation

options show the difference between using various threshold functions. Figure 4.3 uses the threshold that only takes into account the number of pixels. Figure 4.4 uses the threshold that considers features of a parallelogram without including anything to determine size. Figure 4.5 shows the output from using a threshold with our suggested feature set that includes the diagonal angles of the approximate shape.

Thus, each pixel gains a z-coordinate to go along with its (x,y) coordinate. This completes the first stage of the implementation.

4.2 Stage 2: Image Analyses

Stage 2 deals with image analyses. The goal of this second stage is to gather a series of (x,y,z) points that belong to a staircase. This involves applying computer vision techniques to the inputted raw image and utilizing the depth output from the previous stage.

The first step in finding the stairs is to increase the image's contrast. This system of feature extraction helps to bring out the specifically designated details, which in our case, is the emphasis of horizontal lines. There are several ways to apply filters and masking combinations. For this project, a Gabor filter was selected. The primary advantage of Gabor filters leading to this decision was their resemblance to a normal human vision system[8]. The Gabor filter takes a Gaussian kernel function and modulates it based on a sinusoidal wave whose attributes are specified by the parameters. The function takes a total of five different parameters. The first is the kernel size which translates to the size of the filter mask to be applied. The next value is sigma, a small real number which determines the standard deviation of the size of the envelope in the Gaussian filter. The theta value determines the orientation of the stripes displayed in the kernel and can vary from 0 to 180 degrees. The lambda value is the wavelength of the sinusoidal factor, containing a value between the real numbers 2 and 256. The last parameter is the psi value which forms the symmetric phase offset of the cosine factor between -180 and 180 degrees. Together these values determine the exact shape and variation of the

Gaussian kernel applied to the image and consequently, the outputted filtered image.

Through a series of trials, we were able to determine the best set of values to parametrize the Gabor Kernel. One of the main difficulties arose in emphasizing the staircase slope without the knowledge concerning the viewing angle of the camera. The program had to account for the image to be from either side resulting in steps that were slightly positive or slightly negative with respect to the x component. In other words, does the staircase appear to be slanting toward the viewer or away. We did not program our algorithm to accept descending stairs but assumed that the viewer was always at the base step. This is a more common scenario and represents a more realistic goal. As is typical of this problem and mentioned in the above section, in order to accept a wider range of inputs, the solution could not be specialized to the smaller range of specific slope values. This was an example of sacrificing accuracy to increase range of applicability. Several rounds of trials and comparisons brought us to select the values for the Gabor filter seen in Table 4.1, as they produced a good result in the wide range of input images selected. The 2D model of the specified Gaussian Filter is pictured in Figure 4.3. Figure 4.4 and Figure 4.5 show the effects of the Gabor kernel applied to an image. It can be seen how the horizontal edges of the staircase are highlighted and smoothly shown in comparison to the rest of the image, especially the slanted guard rails and the background door.

Kernel Size	21
Sigma (Standard Deviation)	8
Theta (Orientation)	32
Lambda (Wavelength)	111
Theta (Orientation)	32
PSI (Phase Offset)	110

Table 4.1: Gabor Filter Values

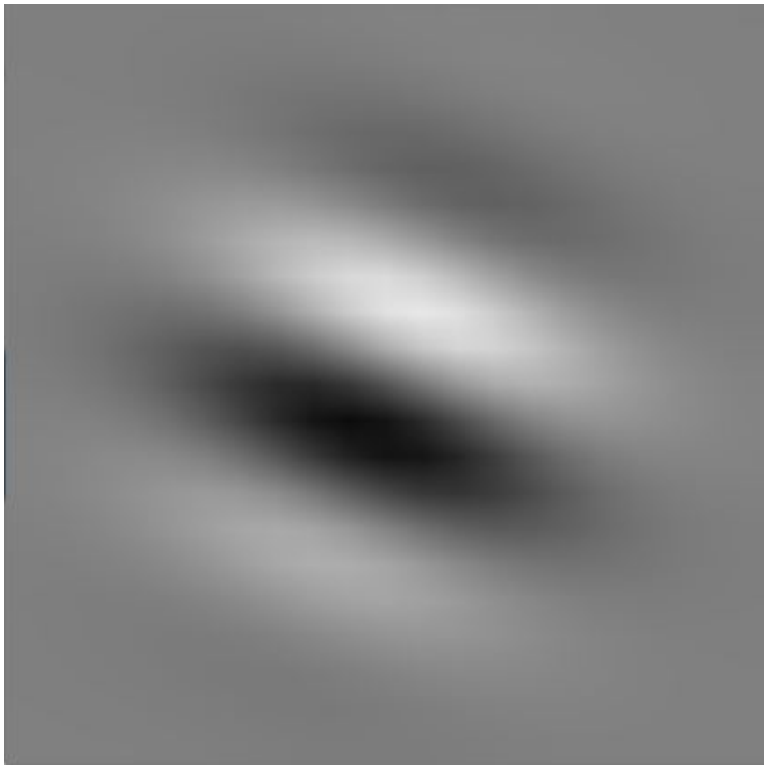


Figure 4.6: Visulization of Gabor Kernel

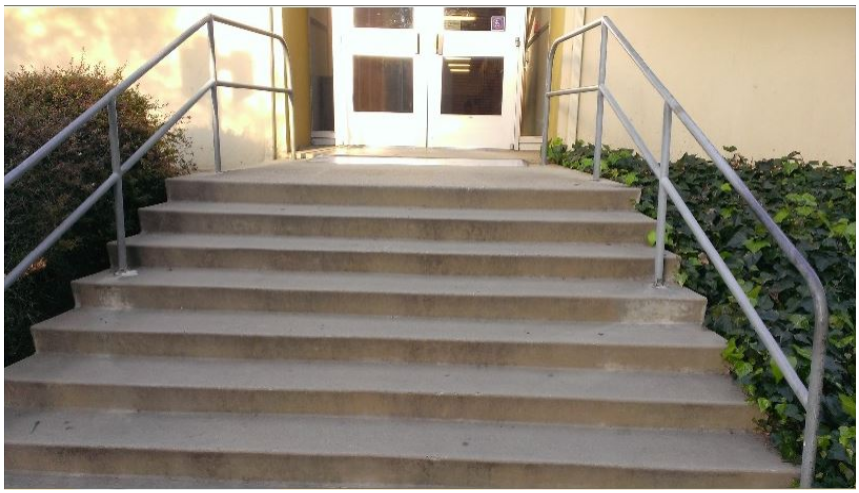


Figure 4.7: Example of an Unfiltered Original Image

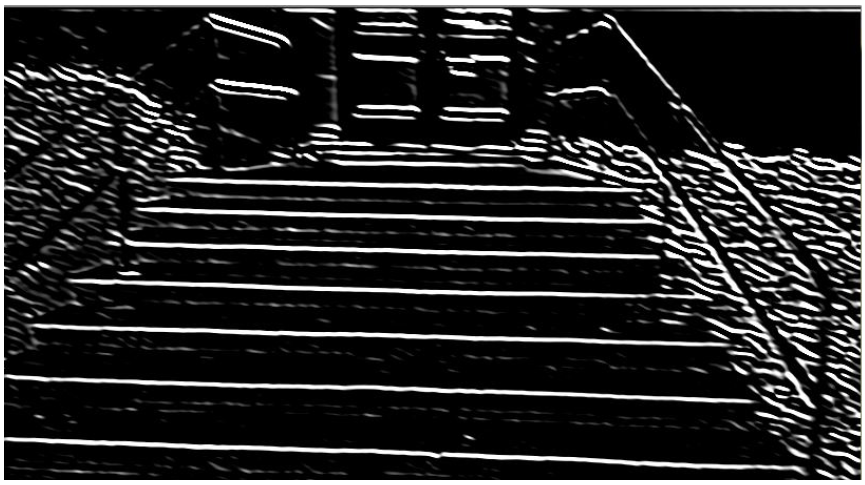


Figure 4.8: Example of a Filtered Image with Gabor Kernel Applied

The output of the Gabor filter is then transferred to the next step. This step furthers the features extraction by applying a combination of a Canny edge detection and a series of Hough transformations.

There are several methods of edge detection, but the most popular by far is Canny[3], despite it being discovered over 30 years ago. The Canny process goes through each pixel and identifies connected lines. After an initial smoothing, the algorithm finds the intensity gradients and determines potential edges within the image through a hysteresis threshold comparison. This leads to the first argument being the aperture size which relays the size of the Sobel filter used to smooth the image. A higher value results in a larger blur effect and a smaller value picks up more detail. Our program used the standard aperture size of 3. Subsequently, a minimum threshold is used to reject obvious non-lines while a maximum threshold specifies lines that are clearly edges. When lines fall between these two thresholds, they require a second round of evaluation to determine if they can be classified as edges. In that case, their surrounding connections are evaluated to see if they fit above or below the threshold values. Our application used a minimum threshold of 50 pixels and a maximum threshold of 150 pixels, which was experimentally found to allow the major edge lines to be detected while canceling most all minor noise that arose from strange textures, such as leaves or tiles. These

values are summarized briefly in Table 4.2. The output of the Canny edge detector is a series of edges that can fed into the Hough Transforms.

Similar to the Canny edge detection, the Hough Transform is a feature extraction method that has been around for decades and is popularly used throughout the computer vision community[5]. Although it can be modified to identify any arbitrary shape that can be mathematically expressed, we need only use the simple version that identifies lines. The Hough Transform was chosen because it excels at identifying the global feature given only knowledge of local measurements. It also deals well with the noise levels expected in our dataset. Line detection is achieved through the use of an accumulator and voting process, taking advantage of polar coordinates to identify line segments. The accumulator scans through the image and gathers all points that could potentially be edges. It then creates a series of lines intersecting the point at different angles, keeping track of the angle and distance from origin. The neighboring points then get to contribute votes to decide which angle is best and which points form the strongest edge. The Hough Transform takes in a series of parameters to alter it to the best fit possible. The first of these is the theta for the accumulator step. This determines the granularity when searching for potential edges with a smaller step leading to better results but at a higher cost. The next two parameters are the maximum gap length and minimum line length. There will inevitably be gaps within each line segment but setting the gap length too large will result in false positives. Likewise, a larger minimum length will reduce the noise of shorter lines but might fail to detect true edges that are less defined. This leads into the last parameter which is the threshold number of votes. By requiring a larger number of votes, the algorithm requires more evidence that the line is a correct edge. Similar to the minimum length and maximum gap parameters, this variable must be set to reduce noisy lines within the image without failing to identify the strong edges.

Our application used an implementation of the Probabilistic Hough Transform. The previously described algorithm can be lengthy to run as each point must be compared to all others, ensuring the best fit is found. The probabilis-

tic version compares the point against only a subset of the other points, enough to ensure statistical relevance. This ensures a very good fit is found even if it is not the same discovered by the original algorithm. An outcome of this method is that a line will be broken up into segments, instead of extending throughout the whole image. This is clearly what we desire as a stair step will usually end in the image instead of continuing out of frame. Unfortunately, this does mean our program will have to merge line segments down the road, but the overhead of this is negligible. The probabilistic Hough also differs from the original in the data it returns. Instead of responding with every point along the line, the probabilistic will only return the two endpoints. This helps reduce memory space and simplify the overhead, but will need to be rectified at the end of the program when the final set of (x,y,z) points are generated. As in the Canny edge detection, the variable parameters were found by experimentally determining which set was best. The accumulator was set to the lowest value of 1 degree. Although this increases time cost, as stated above, the probabilistic attitude of our Hough Transform version helps diminish this and the additional scrutiny was deemed worth it. The minimum line length was set to 100 pixels with the maximum gap length being 10 pixels. The image size, which was discussed above, helps to determine how large the expected staircase step should be, also relying on the assumption that the stairs are a large part of the image. The final parameter, the threshold number for the voting process, was more difficult to set. An original Hough Transform will require a high threshold since more points must be inspected and have the opportunity to vote. However, since the probabilistic Hough Transform checks much fewer points, a much lower threshold is needed. For an image in the size range our program expects, an original threshold would be around 300, but we set our probabilistic threshold to be 100. These values are summarized briefly in Table 4.2. An example image showing the effect of the Canny edge detection and Hough Transform is shown in Figure 4.9.

PARAMETER	VALUE
Canny Hysteresis Threshold 1	50 pixels
Canny Hysteresis Threshold 2	150 pixels
Canny aperture size	3
Hough accumulator step size	1 degree
Hough minimum Line Length	100 pixels
Hough maximum Gap length	10 pixels
Hough voting threshold	100 votes

Table 4.2: Canny and Hough Parameters

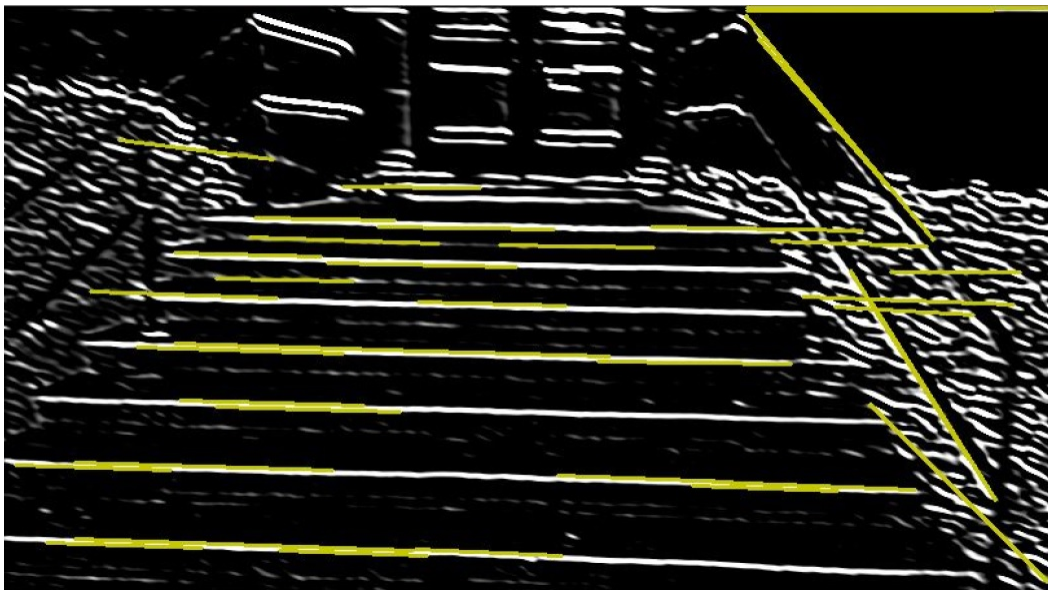


Figure 4.9: Effects of Canny and Hough applied to the Gabor Image

The image seen in Figure 4.9 shows the several lines within the image identified. The front lip of every step has been located along with the back valley in multiple cases. Unfortunately, there are several lines that do not represent steps in the staircase, specifically in this image, the lines caused by the handrail and building window. The next step in the second stage is to evaluate each of these lines and discard unlikely candidates. This is done by looking at the line's position, slope, and comparison to neighbors. However, this must be prefaced by a few assumptions. As stated above, we assume the staircase occupies a large portion of the image. Based on this assumption we can hypothesise that the majority of the near horizontal lines detected by the

processes above are actual correct lines. We also assume that the staircase is fairly normal and able to be ascended by a normal human. The last assumption we make is that each step is roughly parallel to the step above and below. The importance and application for each of these assumptions will be referenced when they become significant.

The initial output from the Hough Transform shows every line that falls within the contrast and length requirements. The first test applied to this set is overall slope. Each line is translated into a standard slope intercept form of $(y = m * x + b)$. From there they can be sorted in order based on their 'm' slope component. As we have assumed no staircase's slope will be too steep, we can reject every line whose 'm' is higher than 1. This value was based on the American building codes which state an optimal slope is between 30 and 40 degrees, while allowing a large buffer to account for steep staircases and odd image angles. The next step utilizes each line's position. The center of each line is found, creating a scatter plot of points which are analyzed for mean, median, and standard deviation. In a statistically normal set of points, the mean and median should be very close together. However, the presence of outliers skews this data, affecting the mean more than the median. This justifies the common practice of using the median in conjunction with the standard deviation to find the bulk of the relevant data and ferret out the outliers, which is a practice our program utilizes. The program compares each line's center point from the scatter plot to the median of the dataset. If that distance is greater than the $1.75 * standarddeviation$, the line is discarded. Although it is more common to use $(x = 2 * standarddeviation)$, we found better results by restricting the requirement down to $(x = 1.75 * standarddeviation)$. After this was completed, another set of calculations were done to find the new mean, median, and standard deviation. The image in Figure 4.10 below shows this process. Each line is represented by it's middle point. The yellow circle shows the initial calculation of $(x = median + 1.75 * standarddeviation)$. The discarded lines whose middle point lie outside that region are shown in turquoise while acceptable lines remain in yellow. The two other circles show the second level of calculations. The pink circle is $(x^2 + y^2 = newMean + 1.75 * standarddeviation)$ and

the red circle is ($x^2 + y^2 = newMedian + 1.75 * standarddeviation$). As can be seen, the red and pink circles cover similar ground showing that another round of discarding based on position would be difficult and not necessarily increase accuracy.

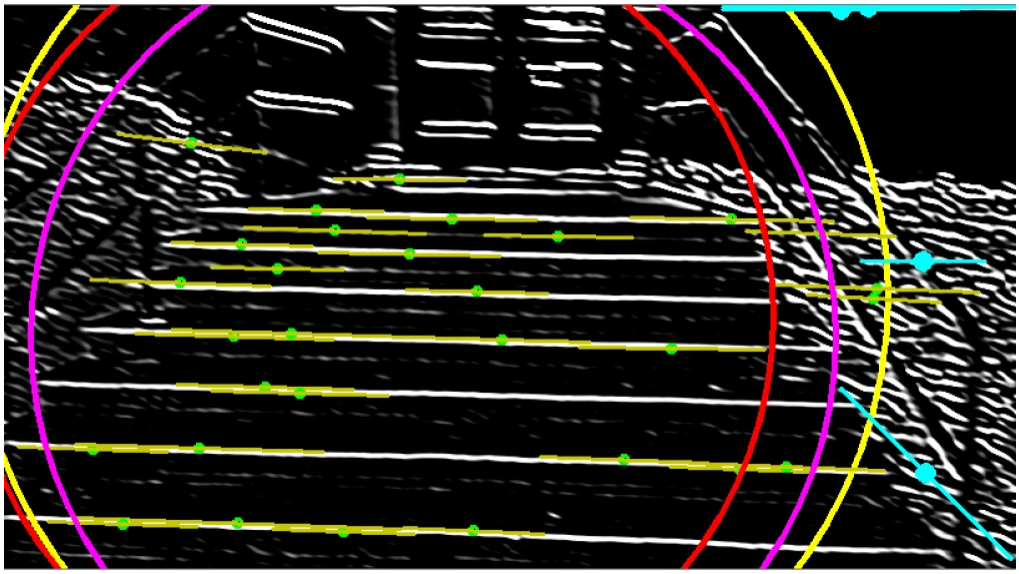


Figure 4.10: Discarding Lines Based on Position

The next round of eliminations apply a similar statistical analysis to the slope parameter. First all the slopes of the remaining lines are gathered and the mean, median, and standard deviation are calculated. Because we assumed all the lines would be nearly parallel, the program can eliminate lines that are not part of the staircase be judging if their slope is an outlier. We again opt to use the median instead of the mean and discard lines whose slope is greater than a standard deviation from that value. This can be seen in Figure 4.11.

The final round of discarding compared lines to create parallels. The list of remaining lines are sorted again based on the 'b' intercept value from their ($y = m * x + b$) form. This means that if two lines are neighboring in the list, they will have the closest intercepts to each other and are most likely to be part of the same step. We again use statistical analysis to calculate the mean, median, and standard deviation of all the intercept values. The program then iterates through the list of lines and calculates the difference in

intercept value between the current line and the previous line. Because we are assuming nearly parallel stairs while only loosely defining the camera angle, the range of intercepts may vary globally but the local differences between steps should be small. This allows us to discard lines who appear to be an outlier from this criteria. This also allows combining line segments. As discussed above, by choosing to use the probabilistic Hough Transform instead of the original, the output was line segments instead of connected lines. As the program iterates through this step it can combine lines with similar slopes and intercepts, essentially filling in gaps. This is shown in Figure 4.12. It can be seen on several steps that segments are connected. This helps by reducing the number of overall line segments to store while simultaneously increasing the area covered by each line.

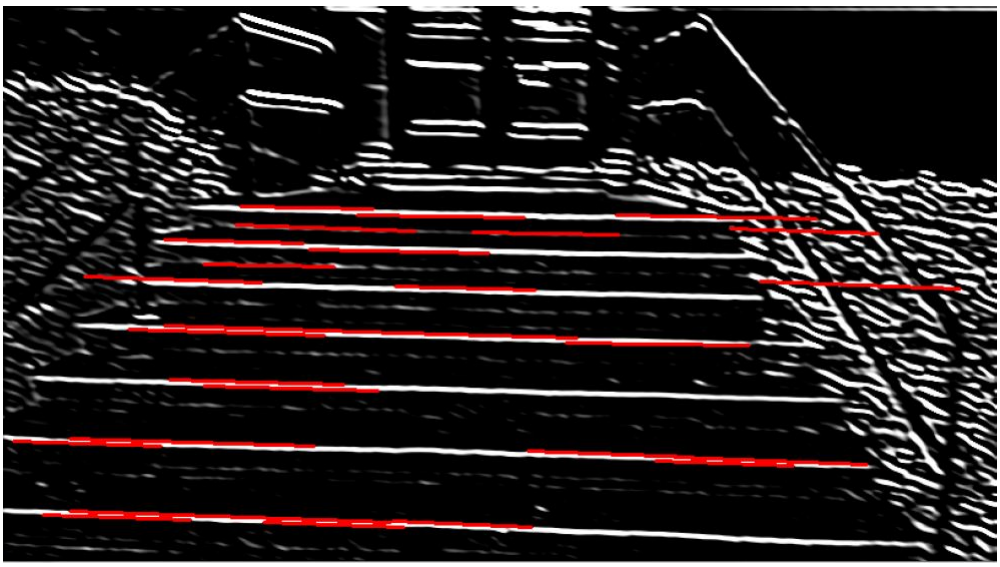


Figure 4.11: Discarding Outlier Slopes

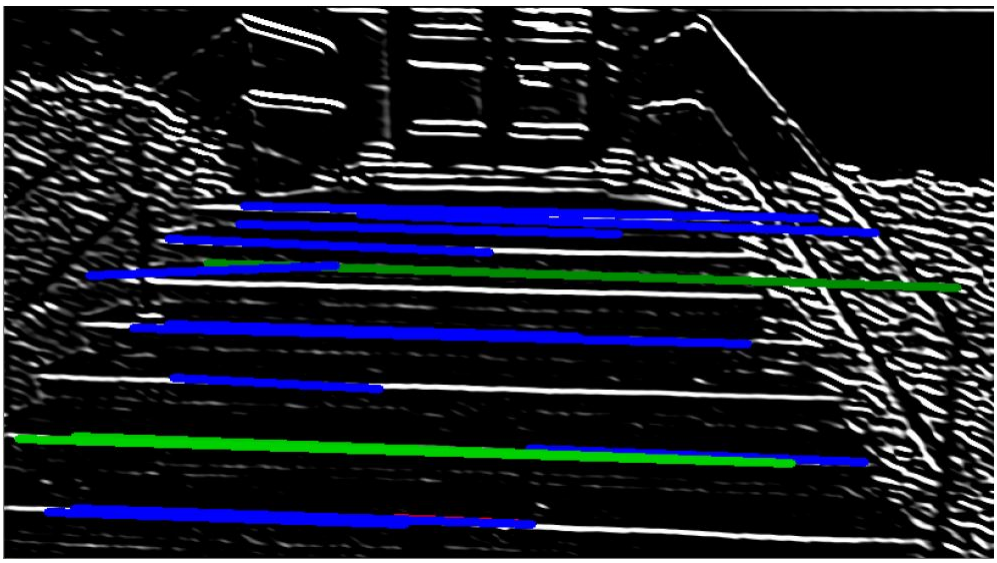


Figure 4.12: Outcome of Parallel Line Merging

Algorithm 2 Processing Image Analysis Algorithm

```
    procedure DETERMINING STAIRS
2:      image ← freshly inputted image
        function APPLY GABOR FILTER(image)
4:          Find the Best Combination of Parameters for Filter
            Apply generic parameters
6:          return Filtered Image
        function APPLY CANNY EDGE DETECTION(image)
8:          Find Edges within the Image
            return List of Edges in Image
10:     function APPLY HOUGH TRANSFORM(image, edge list)
            Find lines from image and Edge List
12:         return List of Lines in Image
```

Algorithm 3 Discarding Outlier Lines Algorithm

```
1: procedure DETERMINING STAIRS
2:     List of Lines ← input from Image Processing
3:     for each line in List of Lines do
4:         if slope > 1 then
5:             delete line
6:         global Average Center ← median of all center points
7:         if (globalAverageCenter − center) > Standard Deviation * 1.75
   then
8:             delete line
9:     sort list of remaining lines by intercept
10:    for each line in List of Remaining Lines do
11:        intercept difference ← current intercept - previous intercept
12:        global Average Intercept ← median of all intercepts
13:        if (interceptdifference + globalAverageIntercept) >
   Standard Deviation * 2 then
14:            delete line
15:    create List of Points
16:    Depth Map ← inputted depth image from stage 1
17:    for each point within List of Remaining Lines do
18:        List of Points ← (x,y) of point, (z) from Depth Map
```

This moves into the final step in stage 2, gathering (x,y,z) coordinates. At this moment, we have stored the endpoints to a series of lines which are all believed to be steps. This set is small as it only includes two or four points for each step in the staircase, usually resulting in less than 50. Thus we need to increase it to provide enough data for the statistics that will take place in the next stage. In order to do this, the program iteratively steps along, sampling 100 new points from each line. However, these are only (x,y) points and the

final stage requires each point to have a depth component. This is where the output from stage 1 comes into play. Each image has a corresponding image which relates depth using gray scale. This file is now opened and layered on top of the original image. Every (x,y) point samples the new depth map and stores the value as the pixel's depth. The final output from this stage is a series of over 1000 (x,y,z) points which lie on the staircase. These are stored by being written to file that can subsequently be analyzed in the final stage.

4.3 Stage 3: Statistical Assessment

The final stage steers away from the previous techniques in computer vision to dwell more on mathematics and statistics. This stage is also shorter than the previous two. The input is the text file containing hundreds of (x,y,z) points. The goal is to sort through each set, remove outliers, and produce a model that can formulaically include each point with low error. The first step is to read in the points and sort them. The inputted file must be scanned through and each point divided to make three lists of (x), (y), and (z). We included a few error checking expressions to help with odd characters, but there is minimal need as the file format is specified explicitly in the previous stage. Once the points have been brought together they can be sent into the Orthogonal Distance Regression.

Orthogonal Distance Regression (ODR) is a regression model similar to a least-squares method. As in any regression method, it is a mathematical way to estimate the relationship among the variables given there could be errors within the original dataset of independent variables. ODR is used here primarily because it can easily be expanded into the third dimension while still returning an explicit equation. In addition to having the provided points, the ODR needs a model on which to try fitting the variables. Applying it to our staircase problem, we simplify the jagged rise/run of a staircase into the smooth slope of a ramp. It is both unnecessary and impractical to dissect the resulting slope into its rise/run components. Therefore, the ODR is provided

with the mathematical linear plane model of

$$z = a * x + b * y + c$$

. The relevant output from the ODR is the error and the best fit for the three constants 'a', 'b', and 'c', which can be placed back in the formula to glean a standard plane equation.

The newly found plane equation is then evaluated with each point. A new list is generated with the distance of each point from the plane, which is essentially the error for each point. Running statistical analysis on this list generates the mean, median, and standard deviation. Within this dataset, there are still lines that might potentially not belong to the staircase. An even more common phenomenon occurs when the line extends to far and includes a few points that are then falsely labeled. These points are much easier to identify as outliers using their (x,y,z) in 3D space compared to the previous stage which only searched through the (x,y) 2D space. Therefore, each point is tested and if the normal distance from the standard plane found by the ODR is greater than ($x = mean + standarddeviation$), that point is discarded. This forms a new set of points which is slightly smaller than the previous set but more statistically relevant. Another regression is run on this new dataset. The new equation is compared to the old equation, if they are similar enough, the new equation will pass through.

The final step of the stage is to calculate the slope angle. Because the plane may be viewed from a slanted angle, the program can't assume very many characteristics. Thus, it must find the plane's normal and calculate the angle based on the distance from the origin and the y-axis. This will return the angle of the slope of the discovered staircase.

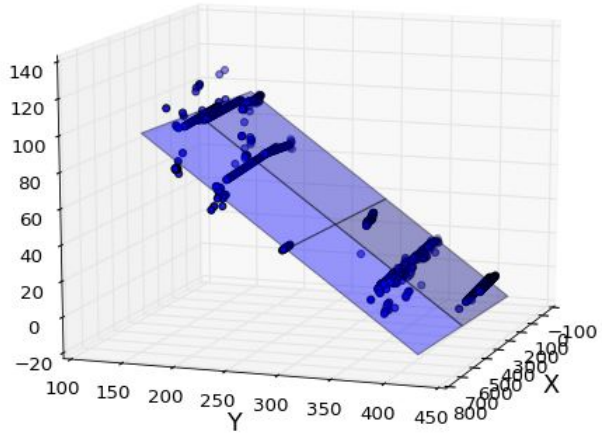


Figure 4.13: Example of the Output from an Orthogonal Distance Regression

Chapter 5

Results

5.1 Generating a set of Test Images

We collected our own test set to determine the accuracy of our algorithm over widely differing environmental factors. It contained a series of fourteen different images. Two of those images were of the same staircase from different angles, to determine the size of the error that was based on the perspective. Two other images were of the same staircase from different times of day and thus under different lighting conditions. This was hypothesized to be critical because so much of the algorithm relied on colors and texture contrasts that shadows could easily wreak havoc. The other ten images were unique. They expressed a variety of materials, backgrounds, sizes and viewpoints. Although this sample set may seem small, we deem it statistically relevant because it explored every facet of the attributes our program is able to handle. Later discussion will elaborate on what contributed most in each image's error.

After the images were taken, the dimensions of the staircase needed to be obtained. This was achieved manually by measuring the rise and run of each stair step. A standard tape measure was used with 1/8 inch accuracy. Unfortunately there were a few issues that introduced error in this step. In several cases, the individual steps in a staircase were not uniform. This was to produce a style where the start of the staircase was more gradual with a longer run that shortened up to being steeper at the end. This would cause the slope to take on more of a quadratic nature than the assumed linear. This phenomenon is slight but is noted as a source of error and is visualized in Figure 5.1. A larger issue was steps wearing down. Steps are meant to be walked upon and so are intended to experience repeated contact and forces.

Yet, resilient as they may be built, there will always be part that rubs and wears off. Because people tend to walk more in the center of the staircase (except in special situations), this middle will get more worn than the sides. This means that the depth of an individual step is no longer linear but slightly parabolic as well. This came into consideration as we measured primarily in the center of the stair step but assumed it was constant depth all along. It is hard to measure how significant a factor this is but we noted it as a way our model deviated from reality; it is pictured in Figure 5.2. A third instance where steps deviated from our expectations is in the top lip causing a back slope. An ideally modeled stair will have right angles for both its front point and back point. Realistically, it is common of a higher step to extend beyond the start of a lower step. Figure 5.3 shows ideal steps in black compared to the realistic version in red overlay. This creates trouble in measuring as well as difficulty in correctly modeling the situation. Because it is no longer a right triangle, the plan trigonometry will not work. This means the run measurement must be modified to take into account the diminished distance. In all these cases, a human measuring the staircase with a tape measure introduces intrinsic error as edges and ridges are approximated as close as possible.

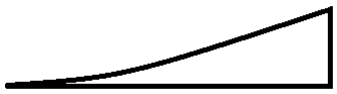


Figure 5.1: Parabolic Staircase Slope instead of Linear, Viewed from Side

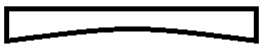


Figure 5.2: Worn Down Step, Viewed from Top

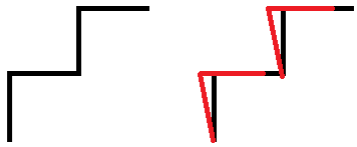


Figure 5.3: Modeling the Top lip of the Step, Viewed from the Side

Once the measurements were recorded, simple trigonometry was used to determine the appropriate angles. Being the case that measurements were known to contain error, the highest error in the hypotenuse, we took an average of the sine and tangent values. A sample measurement is shown in Figure 5.4 below. It was taken from image 3 at the front of the EE office Building 21.

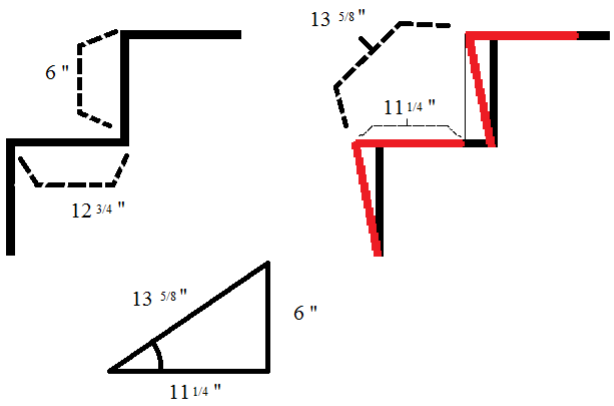


Figure 5.4: Example of How the Stair Measurements were Recorded

From this we can gather a measurement of the slope of the staircase in every image. Below in Table 5.1, each image is listed. The first column show the image identification number, which is chosen arbitrarily but will be useful throughout the rest of the paper when refering to certain images. The second column gives a brief description of the image in relation to its location around the Cal Poly Campus. The third column is the image size when it is originally taken. We tried to gurantee multiple image dimensions to test the ruggedness

of our program. The fourth column holds the measured angle of the slope of the staircase. We explained above, how this number was measured. The final column contains a thumbnail of the image. A larger instance of each image is displayed in the appendix.

ID	Description	Image Size (pixels)	Angle (radians)	Thumbnail
1	Large EE	812 x 452	0.447	
2	Ave	918 x 1632	0.480	
3	EE office	918 x 1632	0.473	
4	front UU	3456 x 4608	0.358	
5	new science	4608 x 3456	0.591	
6	back EE	4608 x 3456	0.431	
7	NM parking lot	4608 x 3456	0.448	
8	Science North 53	4608 x 3456	0.303	
9	Fisher	4608 x 3456	0.492	

Table 5.1: Data test set Part 1

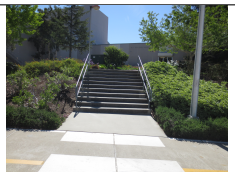
ID	Description	Image Size (pixels)	Angle (radians)	Thumbnail
10	Admin Bldg	4608 x 3456	0.326	
11	Music Bldg	4608 x 3456	0.468	
12	Large EE tilted	760 x 374	0.447	
13	Graphic Arts (dark)	918 x 1632	0.627	
14	Graphic Arts (light)	464 x 658	0.627	

Table 5.2: Data test set Part 2

5.2 Quantitative Result of Angle Detections with Various techniques

These images were each inputted into our program. We ran the program with the three different segmentation feature sets explained above: just size, the whole feature set without angle calculations and without number of pixels, and all new features described without the original size variable. The difference between these three lie in how the threshold is calculated to determine the region joining. This is outlined in the enumerated list below. We will refer to these three segmentations throughout the rest of the paper as size, waws (without angles, without size), and all features, respectively. The output of the calculated slope is shown in Table 5.3.

1. The Size Threshold:
    - Number of pixels
2. The WAWS Threshold:
    - Equivalence of Side Lengths
    - Length - Height
    - Approximate Area
3. The All Features Threshold:
    - Equivalence of Side Lengths
    - Length - Height
    - Approximate Area
    - Difference in Diagonal Angles

By comparing the resulting angles to the correct slope, we can generate the percent error of each calculation. This is displayed in Table 5.4 below. Graphing this information produces the chart in Figure 5.5.

This chart shows the comparison between the different segmentation options. There are a few interesting trends that can be deduced from the table

ID	Size	WAWS	All Features
1	0.4798	0.4634	0.4686
2	0.4029	0.5725	0.344
3	0.4773	0.6189	0.3484
4	0.3159	0.3815	0.24
5	0.3009	0.2009	0.2801
6	0.2959	0.2742	0.288
7	0.2475	0.3041	0.2868
8	0.3077	0.3798	0.2895
9	0.254	0.0674	0.303
10	0.2976	0.2731	0.2812
11	0.3298	0.3217	0.2799
12	0.6172	0.6799	0.6143
13	0.311	0.4716	0.2927
14	0.4245	1.0057	0.4142

Table 5.3: Output of Slope Detection (measured in radians)

ID	Size	WAWS	All Features
1	7.385	3.698	4.879
2	16.085	19.233	28.364
3	0.905	30.85	26.344
4	11.808	6.508	32.985
5	49.078	66.001	52.595
6	31.315	36.342	33.138
7	44.69	32.049	35.905
8	1.584	25.385	4.41
9	48.351	86.286	38.401
10	8.803	16.319	13.848
11	29.468	31.186	40.124
12	38.128	52.165	37.466
13	50.421	24.818	53.344
14	32.326	60.314	33.972

Table 5.4: Percent Error

and corresponding chart. The first thing to notice is the sizable variations in the accuracy measurements among different images. If we take a side detour and compare the maximum, minimum, and average results, we gain the graph in Figure 5.6.

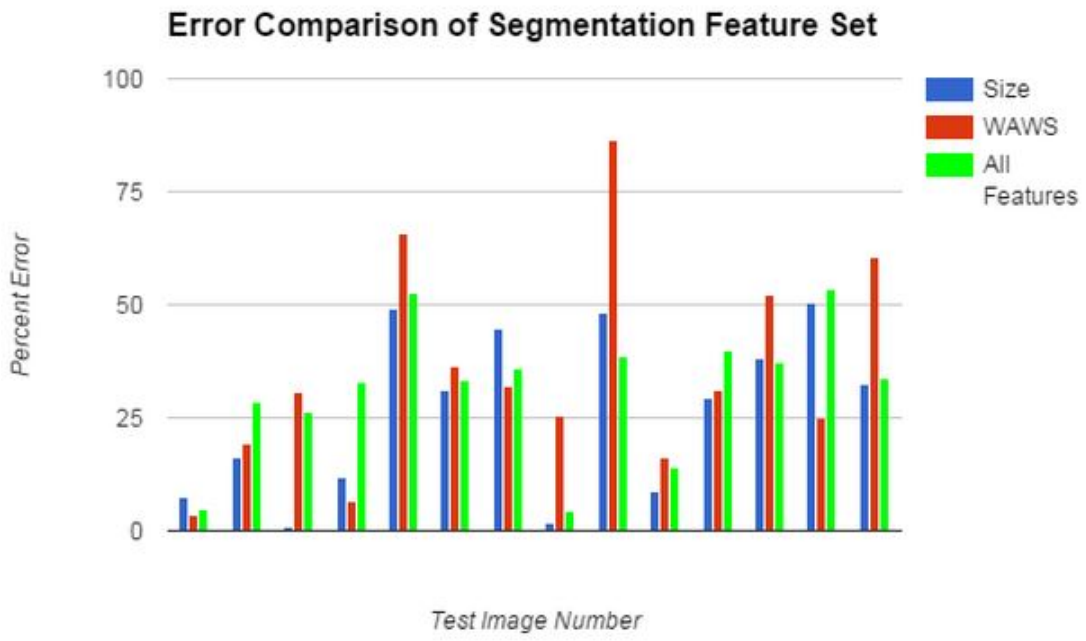


Figure 5.5: Chart Comparing Different Segmentations

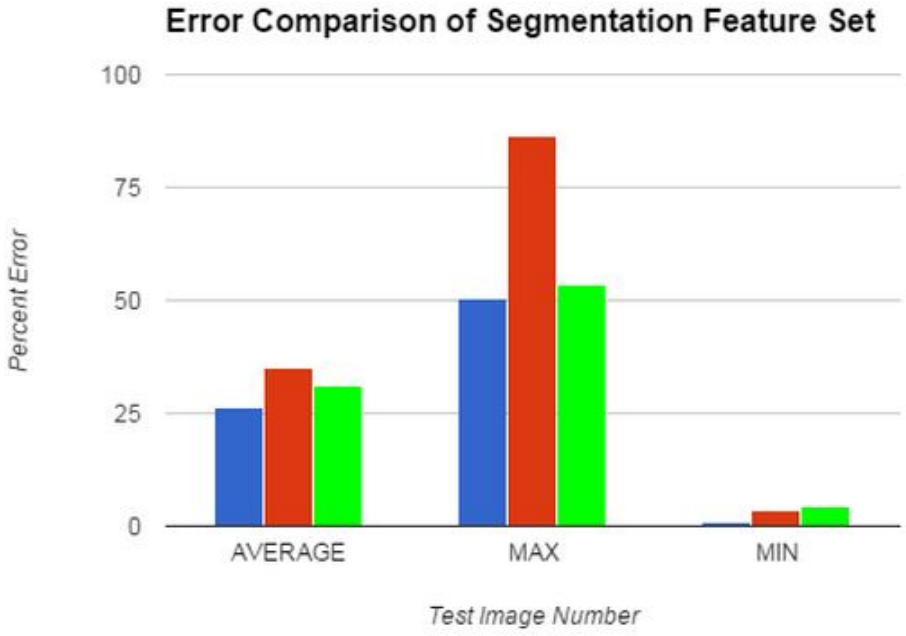


Figure 5.6: Comparison of Average, Max, and Min for Segmentation Algorithms

From this we see that even in the best case version, the size category, there is a 49.5% swing between maximum and minimum accuracy ratings. We will delve into this a little later on by analyzing trends in certain images.

5.2.1 Horizontal vs Vertical Image Analyses

The comparison highlights the importance of including the angles as a feature. As we discussed above, a parallelogram is primarily defined by its angles. The length of the sides are more an outcome of this than a contributor. By failing to take these into account, the error can be increased by 47.885% in the highest case, and 4% in the average case. This is why it is further interesting that in the minimum case, the feature set without the diagonals fared slightly better, even if the difference is less than 1%. This highlights the wide variability and underscores the reasoning behind evaluating the average case as the primary example of the algorithm.

Another point seemingly made evident from this detour is the critical difference between calculating size from the number of pixels and approximating the area from geometric formulas. If you recall, when we determine the size of the regions for merging within the segmentation algorithm, we use the stored series of 4 points. These points are then used as an approximation to the area by the basic quadrilateral formula. This can only be considered an approximation as the actual outline of the region may have a ragged edge or the region may be a more complex polygon. It is only feasible to store a limited number of border points and the extra precision gained by storing more will counter the intent of simplifying each shape to its rough parallelogram. However, it can be seen that the extra precision retained by solely using the number of pixels per region, outweighs the advantages of geometrically encouraging parallelogram shapes to form. By looking at the graph further, we can find some of the reason behind this assessment. We notice that our proposed algorithm of simplifying the area performs noticeably worse it images 2,3,4 and slightly worse in images 13,14. Recalling from Table 5.1, these are the images whose vertical component is larger than its horizontal component. If we separate these into different data sets we can see the trend shown in Table 5.5.

Category	Size	WAWS	All Features
All Images	26.45	35.08	31.13
Vertical Images	22.31	28.34	35
Horizontal Images	28.76	38.83	28.97

Table 5.5: Average Percent Error across Image Orientation

Separating the data in this manner can bring out a new conclusion. Based on all images, it is clear the segmentation method based on the size calculated through the number of pixels is adequate. If we only consider the horizontal images within the test set, we see that the resulting error from that method and the method using all the features is nearly the same. To form some speculation on the matter, we would have to be reminded on the goal of the segmentation algorithm. Depending on how the threshold value is calculated, merging will encourage the outcome to be of differently shaped regions. The size category just encourages the largest regions possible. The all features category attempts to encourage stair step shapes which are parallelograms that have a greater length than height. When the original image is taller than it is wide, it becomes more difficult to segment the regions into long parallelograms. This could potentially lead to the decreased accuracy.

5.2.2 Canny without Gabor

As we are comparing the components in our algorithm, we must regularly justify the existence of each one. This is best done by showing the outcome with and without it being included. This is applied to our utilization of the Gabor filter.

In Figure 5.7, we can see the outcome of applying the Canny Edge Detection and the Hough Transform to an unfiltered image. The outcome of this image with the Gabor filtered applied is shown in Figure 5.8. The main visible differences is in both the number of lines and in their placement. Foremost, it can be seen that by applying the filter, the number of lines decreases dra-

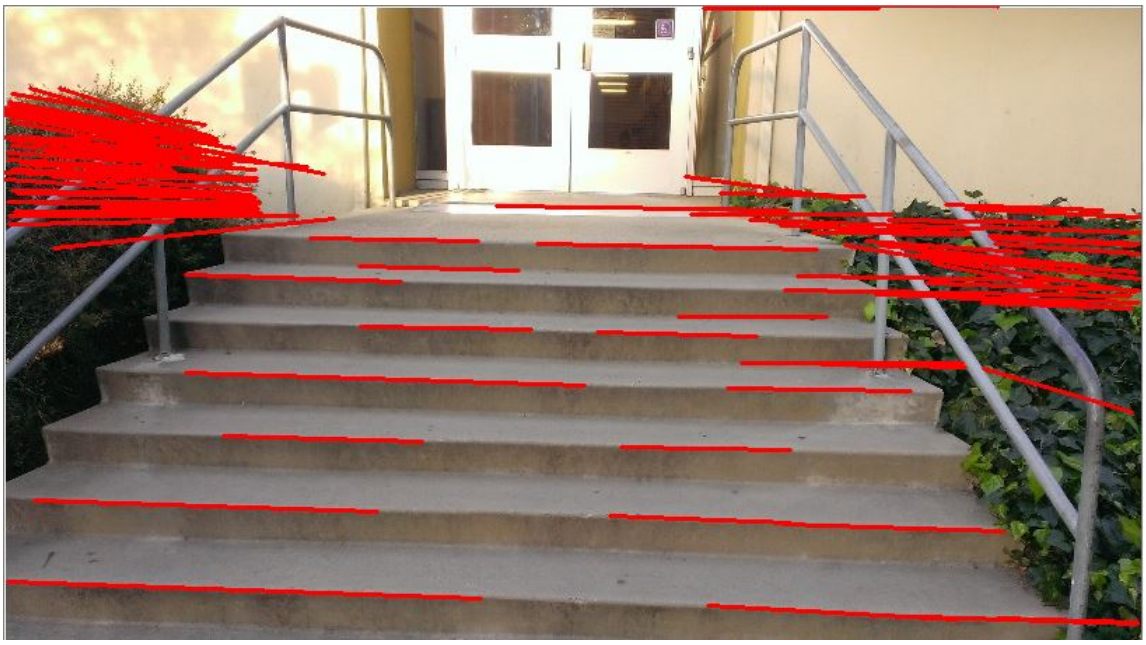


Figure 5.7: Showcasing in Red Lines the Outcome of Canny Edge Detection and Hough Transform without an Initial Gabor Filter

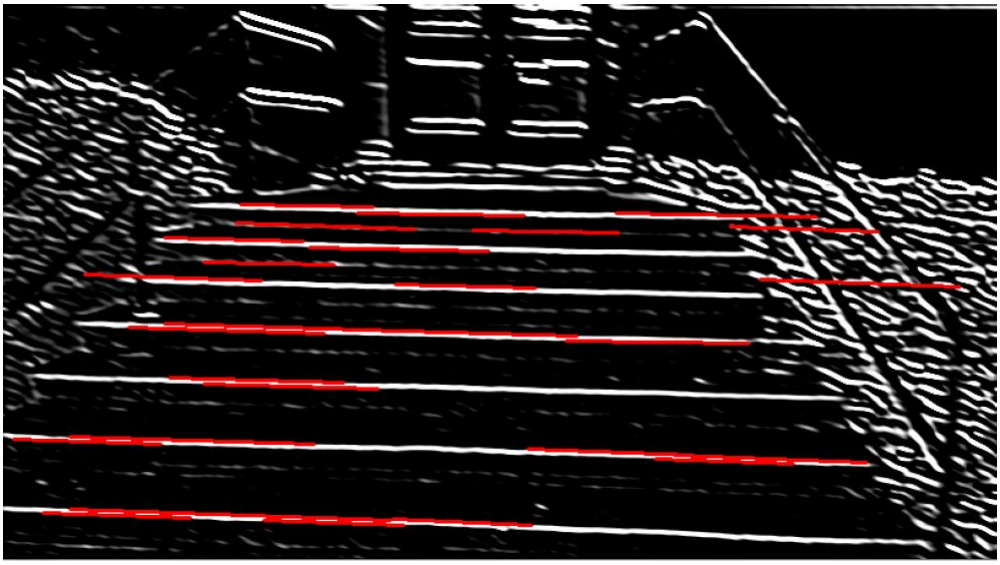


Figure 5.8: Showcasing in Red Lines the Outcome of Canny Edge Detection and Hough Transform with the placement of an Initial Gabor Filter

matically. The quantity can be as much as four times less than without the effects of the Gabor filter. The second factor is in the location of the lines. In the first image, each step is identified but there is also the false identification of much of the bushes in the left and right sideground. In this example, the introduced error is minimal because the bushes are at a similar distance to the actual steps. Therefore when the planar regression is run, it will not deviate as much from the ideal output. Looking at it analytically, we can see that

Metric	Size	WAWS	All Features
AVERAGE	23.09714286	44.65714286	32.08428571
STDDEV	10.69209948	29.49791477	15.82208229

Table 5.6: Statistical Analysis of Stair Detection without Gabor filtering

5.2.3 Regression and Point Analyses

We would like to step back and analyse another factor that can be altered. The outlier detection we previously described that took place in stage 3 after the initial ODR is risky because it attempts to trim data points by evaluating their global position instead of the local differences between points. Although this might have produced a more accurate result, it was deemed unnecessary and too much overhead. Nevertheless, the way outliers were detected and discarded could occasionally eliminate key lines that were underrepresented. The first step in investigating this matter is to see how many points were eliminated between the first and second ODR. This data is displayed in Table 5.7.

Not surprisingly, this doesn't provide a very clear picture unless we know how many point there were originally in each case. Only then can we determine if this is a significant remainder. This gives Table 5.8 below which show the percentage of points that were eliminated based on each run.

In the majority of cases, over 10% of the points were discarded, with some instances reaching closer to 20%. Because this seemed like a large amount of data loss, the algorithm was repeated with the exception that this time,

ID	Size	WAWS	All Features
1	163	272	166
2	228	170	243
3	66	73	61
4	234	443	461
5	439	799	574
6	485	565	357
7	392	369	116
8	742	686	558
9	151	71	146
10	35	501	23
11	407	622	488
12	110	191	228
13	157	135	168
14	156	182	142

Table 5.7: Number of Points Discarded as Outliers in Stage 3

ID	Size	WAWS	All Features
1	11.415	19.048	11.625
2	17.195	12.821	18.326
3	12.941	14.314	11.961
4	8.824	16.704	17.383
5	9.564	17.407	12.505
6	15.85	18.464	11.667
7	7.686	7.235	2.275
8	17.743	16.404	13.343
9	16.449	7.734	15.904
10	1.04	14.884	0.683
11	11.736	17.935	14.072
12	7.703	13.375	15.966
13	15.392	13.235	16.471
14	15.294	17.843	13.922

Table 5.8: Percentage of points Discarded in Stage 3

outliers were not discarded in stage 3. Meaning the program was run under the conditional that it accepted the first plane equation generated and calculated the slope based on that ensuing equation. The results can be seen below in Table 5.9. Yet, this means little until it is compared to the initial data we have seen in Table 5.4. Thus next to each column is another column labeled

"Diff" which subtracts the first round percent error from the second round percent error. When this value is positive, it means the second round of the ODR reduced the percent error whereas if it is negative it means the error increased.

ID	Size	Diff	WAWS	Diff	All Features	Diff
1	7.134	-0.251	4.824	1.126	4.839	-0.04
2	14.159	-1.926	19.057	-0.176	28.017	-0.347
3	2.832	1.927	37.791	6.941	26.809	0.465
4	11.173	-0.635	3.023	-3.485	33.992	1.007
5	48.567	-0.511	67.862	1.861	52.595	0
6	32.544	1.229	40.201	3.859	33.011	-0.127
7	44.46	-0.23	35.164	3.115	35.74	-0.165
8	0.478	-1.106	16.05	-9.335	4.419	0.009
9	47.806	-0.545	82.181	-4.105	38.326	-0.075
10	9.034	0.231	19.979	3.66	13.426	-0.422
11	33.373	3.905	29.733	-1.453	40.333	0.209
12	37.191	-0.937	47.896	-4.269	36.748	-0.718
13	50.815	0.394	16.949	-7.869	53.274	-0.07
14	34.244	1.918	63.524	3.21	32.995	-0.977

Table 5.9: Percent Error

If we take the average error from the first round ODR and separate it based on image orientation as we did to the previous set of data, we obtain a table similar to Table 5.4 that is displayed in Table 5.10.

Image Set	Size (2)	WAWS (2) (2)	All Features (2)	Size (1)	WAWS (1)	All Features (1)
All Images Images	26.45	35.08	31.13	26.70	34.59	31.04
Vertical Images	22.31	28.34	35.00	22.64	28.07	35.02
Horizontal Images	28.76	38.83	28.97	28.95	38.21	28.83

Table 5.10: Average Error based on Image Orientation Compared across ODRs

Interestingly enough, running the second ODR increases the error in every subset of the WAWS and in the Horizontal Images category in All Features. It

performs the expected result of reducing error in the Size category. It is also notable that the trend continues of the Size category achieving better results within the subset of vertical images.

5.3 Comparison of the Results after Altering the Gabor Filter Parameters

Another feature that can be modified and explored is the Gabor filtering. When we initially constructed our feature detection system, we built the Gabor filter with the intent of visually highlighting the most prominent stair steps. This was done by reaching a compromise between the parameters and determining which values provided good results overall. Nevertheless every image has its own set of values that will provide optimal results. This is seen in Table 5.11 which shows the best fit for the (sigma, lambda, theta, psi) values.

Image ID	sigma	lambda	theta	psi
1	5	36	97	308
2	7	36	88	250
3	4	31	81	308
4	4	35	115	308
5	1	32	116	313
6	1	34	116	312
7	1	33	113	302
8	1	31	97	302
9	2	33	85	270
10	11	28	75	321
11	14	25	86	317
12	12	28	86	158
13	6	34	65	112
14	8	36	105	123

Table 5.11: Best parameters for Gabor Filter by Image

By applying the specific filter values to our program we can glean the much better results. The percent error of each segment for each image is shown in Table 5.12 and visually displayed in a chart below that. The last line in the table calculates the average error over the whole data set. It only compares the

two feature sets of the original segmentation and our specified segmentation because the other option was noticeable worse and seemed to skew the data. Thus the presented graph highlights the difference that we are trying to show with a higher resolution.

ID	Size	WAWS	All Features
1	2.33	20.47	0.03
2	4.12	49.66	1.28
3	30.25	120.87	14.36
4	24.64	248.26	18.27
5	25.51	19.23	49.09
6	13.31	3.70	31.92
7	29.17	12.04	29.69
8	18.47	56.14	0.01
9	45.73	90.06	35.23
10	43.03	99.23	11.3
11	33.44	31.28	35.18
12	10.15	43.85	24.38
13	32.65	47.48	7.97
14	11.99	38.49	3.00
AVE	23.20	62.91	18.69

Table 5.12: Percent Error with Specific Gabor Values

It can be seen that the error rate is significantly less than previously seen. However, because each image has a different optimal point in the four-parameter Gabor kernel equation, it is extremely difficult to find a global optimization. This was what we were attempting to find earlier. Taking the average of each parameter would lead to the quadruple (6, 32, 95, 265). Unfortunately this would provide an average accuracy of [28.99, 40.76, 33.20] for [Size, WAWS, All Features] respectively, which is worse than our previously found optimal point.

The American Building Association dictates the standard angle of staircases to be .598 rad. By setting our program to assume this is the slope and finding the parameters for the Gabor filter that best converge to that point, we can come up with an good results. However, only initial investigations were

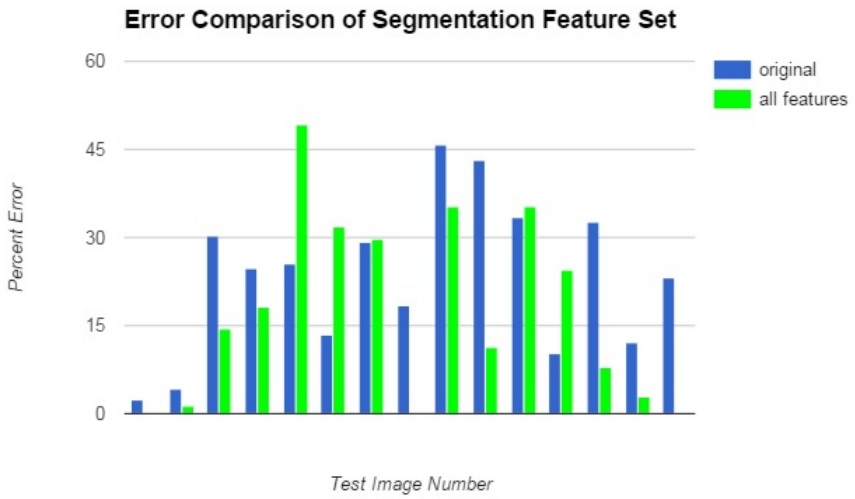


Figure 5.9: Error Comparison from Specific Gabor filters for each Image

made into this method, so we leave it as a promising avenue of future research work.

5.4 Comparison to other Research paper Methods

There are several papers attempting to solve similar problems that were discussed in the Related Works section which give a good baseline for comparison. Nevertheless, it is critical to make notice of the main difference. The input the program receives contains vastly different amount of information. Our program attempts to find the slope of a staircase within an image given only the single RGB photo. No other research to date attempts to solve this exact problem with such limited information. Regardless of how the error compares to other methods, our program is a contribution that can be used and built upon by other researchers because it explores a previously undeveloped area. For comparison purposes, it is still fruitful to evaluate the error rate of our method in light of these others, in order to see how it lines up. In the Table 5.13 below, we compare four of the top ranking papers who display a high level of relevance.

From this we can see that our method with a minimum error of 4.42%, a

Paper Title	Author	Smallest Error	Largest Error
Ascending Stairway Modeling from Dense Depth Imagery for Traversability Analysis	Jeffrey Delmerico et al	2.3%	4.2%
Plane-based detection of staircase using Inverse Depth	Titus Tang et al	3.62%	6.09%
Detection and Modeling of Staircases Using a Wearable Depth Sensor	Alejandro Perez-Yus et al	2.03%	9.41%
Curb Detection for Driving Assistance Systems: A Cubic Spline-based Approach	Florin Oniga and Sergiu Nedevschi	95%	95%
Vision-based Detection of Staircases	Stephen Se et al	8.7%	124%

Table 5.13: Comparison of Error Values

maximum error of 52.60%, and an average error of 28.83%, compares reasonably to the other papers. The minimum error rate seems to fit accordingly with the best other methods. It does not perform as well as those with additional equipment but improves upon the modeling-from-motion method of Se. The worst case lands somewhat in the middle, performing noticeably worse than the sensor-based methods, but better than the camera-based methods. Unfortunately, it is nearly impossible to get a completely accurate comparison. This is because we cannot use the same test set on the other programs. By recalling the reason we had to gather our own test set was to get the additional measure of stair case slope, our program would not be able to be tested on a generic data set. Likewise, we do not have the implementation of the other programs to test on our gathered data set. Therefore, we can do the comparison to gain a general relationship but to truly determine the related error rates is not practically possible.

Chapter 6

Future Work

Evidently, there were several methods or techniques that researchers have used both in the arena of depth perception and that of staircase identification. We chose the method at each stage for a specific reason. We opted to use the random Markov field for its exemplary performance in representing certain dependencies and its ability to be easily trained. RMFs are also widely used within artificial intelligence applications, thus producing a known and tested commodity. The graph-based segmentation we implemented was tailored for images, easily scalable, and could be restructured for staircase identification without increasing speed or memory capacity too greatly. Our application of Gabor filters, as previously mentioned, was largely due to their similarity to human perception. They can also be applied very easily to raw images and can be tuned to highlight a specific shape as necessary. The Canny edge detection followed from the same reasoning that it is a commonly used and well tested method. The high-contrast image that followed from the Gabor filter would have lent itself graciously to any kind of edge detection. The Hough Transforms were similarly an industry recognized approach and clear choice for line detection.

There were also many techniques that we did not implement. The first was a color or texture matching for specific stairs. We rejected this idea as we wanted a more versatile program that would identify stairs based less on their appearance, which can change, and more on the attributes that makes them a staircase. This leads into the next idea of using a training set specified with images of staircases. This would be a good example of potential further work, but we decided that using a set of data that has previously been verified is more credible and complete. It allows us to compare our results

with more applicability and saves the time and equipment necessary to create a better version. Another potential idea was to use other visual cues within the image, such as doorways, or architectural cues, such as vanishing points. The explanation for not pursuing this ideas after initial research was a lack of versatility and improvements. Similar to the color/texture explanation, we wanted the image to be as unspecified as possible, which is evident in our wide range of test images. There is also not much evidence to support the idea that implementing these would bring about improvements. Se et al. [18] remarked in their paper "Vision based Detection of stairs" that the effect of finding vanishing points on the accuracy level "stays roughly the same as the number of lines drops from 100 down to 20, but degrades notably from 20 down to 5. In fact, any convergent group consisting of relatively small number of lines will be left undetected with this approach." Because our algorithm was consistently detecting less than 20 correct lines on the staircase, we decided not to implement vanishing point analyses.

As any project can be indefinitely extended or improved, we decided to pursue the problem with the methodology outlined above. Further improvements and specializations can then use our work as a platform to improve upon our findings and reach a more accurate result. As the field grows and the unique benefits of monocular vision are perpetuated, different methods may be combined to provide the best of all outcomes.

Chapter 7

Conclusion

Our initial goal was to identify the dimensions of a staircase exclusively using the information that can be gathered from a single monocular image.

We have identified and implemented a viable method that achieves this goal. Along the way, we described the choices that instituted each step of the pipeline. We have compared the results of using different segmentation method, different outlier detection tactics, and different image variations. We achieved a fairly accurate result with an error rate around 29%, reaching below 5% in some cases. Because our goal is unique and our methodology built from the theoretical work of several sources, we cannot make a straight comparison. Nevertheless, in relation to more sophisticated and expensive methods, our results are good but not the best. Now that this research has contributed to the known field, it can be expanded for more applications.

BIBLIOGRAPHY

[1] A. Y. N. Ashutosh Saxena, Min Sun. Make3d: Depth perception from a single still image. *In AAAI (Nectar track)*, 2008.

[2] A. Y. N. Ashutosh Saxena, Sung H. Chung. 3-d depth reconstruction from a single still image. *International Journal of Computer Vision (IJCV)*, 2007.

[3] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986.

[4] A. E. D. Hoiem and M. Hebert. Recovering occlusion boundaries from an image. *IJCV*, 2011.

[5] R. O. Duda and P. E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, Jan. 1972.

[6] H. L. Erick Delage and A. Y. Ng. Automatic single-image 3d reconstructions of indoor manhattan world scenes. *International Symposium Robotics Research ISRR*, 2007.

[7] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 2004.

[8] D. Gabor. Theory of communication. *In J. IEE, vol. 93, London*, 1946.

[9] D. G. H Ishikawa. Segmentation by grouping junctions. *IEEE Conference on Computer Vision and Pattern Recognition*, 1998.

[10] J. Hesch, G. Mariottini, and S. Roumeliotis. Descending-stair detection, approach, and traversal with an autonomous tracked vehicle. *Intelligent Robots and Systems*, 2010.

[11] P. D. J. R. Jeffrey A. Delmerico, David Baran and J. J. Corso. Ascending stairway modeling from dense depth imagery for traversability analysis. *International Conference on Robotics and Automation (ICRA)*, 2013.

[12] T. C. . H. S. . Z. Z. . X. J. . J. H. Lizhi Zhang. The image depth estimation based on multi-scale texture features and least-square method. *12th International Conference on Signal Processing (ICSP)*, 2014.

[13] A. P.-Y. G. Lopez-Nicolas and J. J. Guerrero. *Detection and Modelling of Staircases Using a Wearable Depth Sensor*. Springer International Publishing, 2014.

[14] M. Maire, P. A. C. Fowlkes, and J. Malik. Using contours to detect and localize junctions in natural images. *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[15] A. Markov. Extension of the law of large numbers to dependent quantities. *Izvestiia Fiz.-Matem. Obsch. Kazan Univ., (2nd Ser.)*, 1906.

[16] F. Oniga and S. Nedevschi. Curb detection for driving assistance systems: A cubic splinebased approach. *Intelligent Vehicles Symposium (IV)*, 2011.

[17] G. Palou and P. Salembier. Occlusion-based depth ordering on monocular images with binary partition tree. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011.

[18] S. Se and M. Brady. Vision-based detection of stair-cases, 2000.

[19] W. L. TJJ Tang, WLD Lui. Plane based detection of staircases using inverse depth. *Australasian Conference on Robotics and Automation*, 2012.

[20] S. C. . K. S. Youngjung Kim. Data-driven single image depth estimation using weighted median statistics. *IEEE International Conference on Image Processing (ICIP)*, 2014.

APPENDIX A

List of Images Used

Here we identify the test set of 14 images used to analyse the program. The reader can clearly see the diversified inputs that lead to this set being statistically relevant on which to test our implementation.

Figure A.1: Image 1

Figure A.2: Image 2

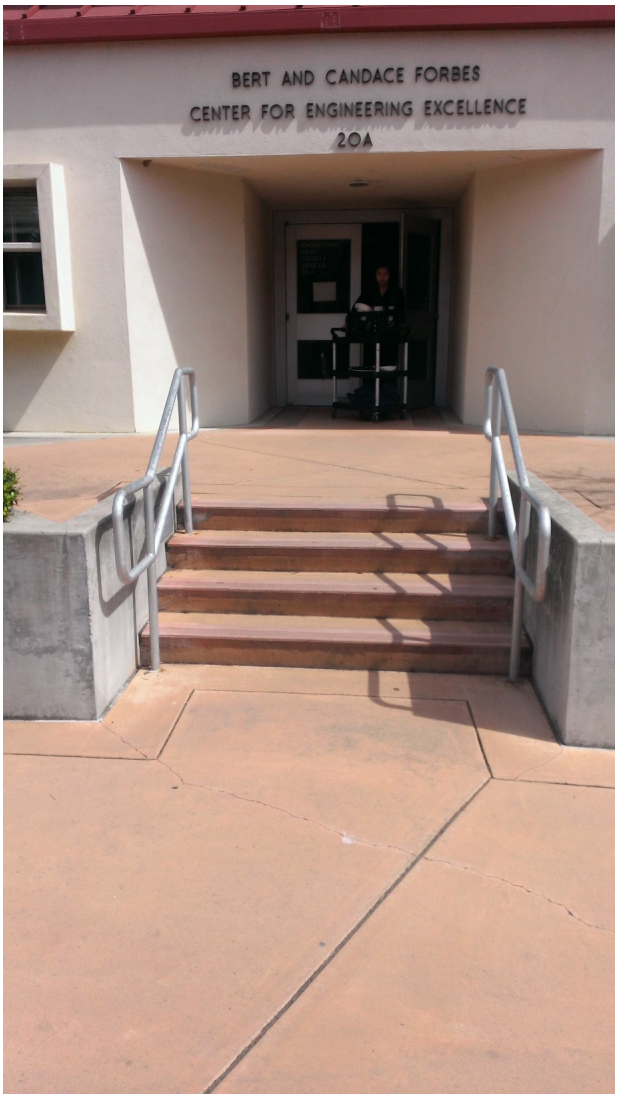


Figure A.3: Image 3

Figure A.4: Image 4

Figure A.5: Image 5

Figure A.6: Image 6

Figure A.7: Image 7

Figure A.8: Image 8

Figure A.9: Image 9

Figure A.10: Image 10

Figure A.11: Image 11

Figure A.12: Image 12

Figure A.13: Image 13

Figure A.14: Image 14