The Foundations of a Guided Walking System for the Visually Impaired

A thesis submitted in partial fulfillment of the requirement
for the degree of Bachelor of Arts / Sciences in Physics from
The College of William and Mary

by

Kyle Brubaker, Patrick Rice, Amit Verma

Advisor: William Cooke

Senior Research Coordinator: Irina Novikova

Williamsburg, VA

May 9, 2018

Acknowledgements

Each one of us would like to thank Dr. William Cooke for his assistance, guidance, and incredible patience throughout the entirety of this project.

Abstract

The objective of this research project was to create an affordable and unobtrusive guidance system for the visually impaired. The project was split into three distinct functions necessary for safe self-navigation: 1) path detection, 2) object detection, and 3) elevation change detection. Path detection required the use of a camera and image processing tools found in the MATLAB software. Information gathered from lines plotted over detected edges were used to provided auditory signals that warn the user when they drift toward the side of the path. Programs used for object detection focused on two main features: an object's color and its horizontal shift within two stereoscopic images. Based solely on color, the program was able to locate multiple objects within an image, and output information such as location and size to the user. A separate program designed to analyze stereoscopic images was able to detect a single object within a path and convey how far the object was from the cameras. For elevation detection, a Time-of-Flight sensor was used with an Arduino to detect changes in height found in stairs or curbs. This sensor gathered frequent measurements so any major deviation in distance was attributed to an elevation change.

Table of Contents

1 Introduction

1.1 The Goal for the Experiment

According to the National Federation of the Blind, the United States is the home of 10 million blind or visually impaired individuals. Statistics show that another 75,000 Americans are added to this list every year [1]. This means there is a growing need for assistive technologies to help these people get around and perform daily tasks. Traditionally, "white canes" have served as the primary tool for navigation. Those who are visually impaired tap and swing the cane in front of them to detect obstacles in their path. White canes are popular for a variety of reasons. They are affordable, ranging anywhere from free to $40 and they provide tactile information from the environment [2]. They are also customizable. Different tips can be attached to the end of the cane for more specific functions. For example, the "pencil tip" provides good tactile feedback but can get stuck in cracks inhibiting forward motion [3]. The "roller tip" and "marshmallow tip" can easily roll and bounce over cracks respectively, but are heavier which causes arm fatigue more quickly [3]. Despite the advantages of a white cane, there are some disadvantages also. In crowded places, canes can be cumbersome and awkward. Also, cane users face increased interference from good-hearted Samaritans wanting to help even when they are not needed [2]. Others choose to get a guide-dog as a faster, more elegant way of getting around. This method has its own set of problems such as being expensive and requiring a great deal of time and responsibility to train and care for the dog [2]. Higher tech options include the ARIANNA mobile app which is used indoors. Colored tape is laid down beforehand to mark out certain routes. Users then point their cell phone towards the ground and swing their phone to try and detect the lines. The phone will vibrate every time it detects a straight line, gently guiding users to their destination [4].

We want to provide a subtler, cheaper method of travel for people with visually impairments without the need of preexisting markers. The end goal of this project is to successfully create a wearable piece of technology that will gently and safely guide visually impaired persons using auditory or tactile outputs. Chapter 2 describes the hardware and software used in each section of the project. In order to do accomplish this goal, the project was broken up into three parts. Chapter 3 explores path detection and auditory guidance based on the user's relative position on the path. Chapter 4 focuses on detecting obstacles in that path and determining the distance from the user. Chapter 5 develops a procedure for detecting changes in elevation in the path such as stairs or curbs.

2 Technology and Equipment

2.1 The Hardware and Software

Since the goal is to create an affordable and easy guided walking system, the price of the equipment used will also be included in the description. The system contains two ELP 720p Super Mini USB cameras which cost about $40 each. A set of stereo headphones, which may vary in price, is also needed to properly hear the auditory output. The system also requires an Arduino Uno and a VL53L0X IR sensor, which are about $20 and $15 respectively.

The software used for the image processing methods was MATLAB R2015b. This software was chosen because it contains numerous built-in image analysis functions that were used throughout the project. However, the downside is that it is a large program that needs to be run on a laptop. It is conceivable that only the functions that are needed can be coded to fit on a smaller, inexpensive computer model such as a Raspberry Pi. This would be ideal so the user would not have to carry around a large computer and the system could be smaller, lighter, and more self-contained. The IR sensor requires the use of Arduino IDE on a computer to program and use the UNO and sensor. This software is available for free online.

3 Detecting Pathways *by Kyle Brubaker*

3.1 Introduction

The goal for this section of the project is to identify the edges of viable pathways and then provide auditory prompts to the user, allowing them to continue down the path without stepping off. The program takes a snapshot, analyzes the image to detect edges, and outputs a sound to the user, alerting them if they are wandering too close to either side of the path. The program assumes that the user is already on a path when it starts.

3.2 Picture Analysis

The path-detection code takes a picture every 0.1 seconds using only one of the ELP 720p Super Mini USB cameras. The picture is cropped, converted to grayscale, subjected to a 2D Gaussian filter, run through the Canny edge detection, and finally undergoes a Hough transform which plots lines over the original image. Figure 3.1 outlines the process. The program repeats this procedure infinitely thereby simulating a live video feed. The following sections focus on the image processing methods that lead to the detection of the path's edges.

Figure 3.1: This flowchart summarizes the order of imaging methods applied to each picture.

3.2.1 Cropping the Image and Converting to Grayscale

The image produced by the Super Mini USB camera is 720x1280 pixels and displayed in Figure 3.1(a). Most of the area in Figure 3.1 shows details other than the path such as trees, the sky, and fields of grass. Since the focus is on the sidewalk, everything else can be cropped out. The first and last quarters of the horizontal axis, each 320p wide, are removed as well as the first 410p from the top of the vertical axis. The remaining 310x640p image is seen in Figure 3.1(b) with the path clearly isolated.

(a)  (b) 

Figure 3.2: (a) The full 720x1280p image taken by the ELP 720p Super Mini USB camera. (b) The 310x640p cropped image with nearly everything removed except the path.

The Canny edge detection method is used to find the boundaries of the path but can only be applied to a two-dimensional matrix. Color images are stored as three-dimensional matrices: one matrix for the red values, one matrix for the green values, and one matrix for the blue values. Each entry in the two-dimensional matrix stores an intensity value, between 0 and 255, representing the degree to which that individual color is expressed. A pixel is simply a combination of these intensity values. For example, a pixel with the values (255, 0, 0) would be pure red, (0, 255, 0) pure green, and (0, 0, 255) pure blue. When MATLAB performs the conversion from RGB to grayscale, it averages all three intensity values of the pixel into a single value [5]. This creates a single two-dimensional image matrix which, without the third-dimension, can only be seen in shades of gray.

Figure 3.3: The image from Figure 3.2 (b) is converted to grayscale. Each pixel has a certain intensity value which determines how light or dark it appears.

3.2.2 Gaussian Filtering

Currently, the image resolution is too high. If the Canny edge detection method was applied now, it would pick out the edges of bricks, rocks, and even individual blades of grass instead of focusing on the physical boundaries of the path. The solution is to blur the image by applying a 2D Gaussian filter. The Gaussian filter works by creating a small square grid, or mask, that covers the original image matrix. The pixel values of the mask matrix are weighted using the Gaussian function,

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (3.1)$$

where x is the distance from the origin on the horizontal axis, y is the distance from the origin on the vertical axis, and σ is the standard deviation. The pixel in the middle of the mask is given the highest weighted value while pixels farther from the origin are weighted less. A larger value of σ creates a bigger mask, allowing for more entries to be included in the "neighborhood," expanding the effect of the blur [6]. During the filtering process, the Gaussian mask is centered on each pixel in the original image. The intensity value of every pixel under the filter is multiplied by the corresponding mask value. Those numbers are then averaged together and produce a singular mean value. The program creates a new matrix that stores the averaged value at the same coordinates as the original pixel. This allows the mask to create a new image without altering the values of the original [7]. Once the information is stored, the mask then moves to the next pixel and repeats the procedure storing the averaged values in the new matrix. After the process is complete, the original image matrix is completely replaced by the values in the filtered one. Pixels close together with similar intensity values will generally appear as one shade of gray. This breaks up the image into three sections: the dark gray grass on the left, the lighter path in the middle, and the dark gray grass on the right. The program uses a Gaussian filter where $\sigma = 7$. This value was used because it produces a wide mask that does not excessively blur the image. The effect of the 2D Gaussian blur on Figure 3.3 can be seen in Figure 3.4.

Figure 3.4: The image from Figure 3.3 is blurred using a 2D Gaussian filter. Notice the distinction between the dark sections of ground and lighter section of the path.

3.2.3 Edge Detection

Now that the image is finally prepared, the Canny edge detection method can be applied. MATLAB has several built-in edge detection functions but the Canny method was determined to produce the best results. The Canny method first applies the Sobel edge detection method to the image. The Sobel method creates a mask that scans across the image. Every time it detects a noticeable change in intensity, it plots a point. The stronger the change in intensity, the brighter the point. This produces an image where the detected edges appear as gradients, lines with bright pixels in the middle that gradually fade to black. The Canny method then runs another mask over this filtered image, looking for the local maxima [8]. The mask compares the intensity values of all pixels under the mask to the value in the center. If the center pixel is higher than its neighbors, it is recognized as an edge and a 1 is plotted at those coordinates, in a new matrix as before [9]. If no edge is detected, a 0 is plotted instead. In a binary image, the 1 represents a white pixel and the 0 represents a black pixel. The Canny method has two thresholds, one upper and one lower, set by the programmer to help distinguish strong and weak edges. A detected

maximum with a value greater than the upper threshold is automatically counted as an edge. If the detected maximum value is less than the lower threshold, it is not counted as an edge. If the maximum value falls between the two thresholds, it is counted as an edge only if it is adjacent to a previously detected edge site [9]. Setting a high upper threshold decreases the number of potential edges since most of the maxima fall below the limit. A small upper threshold does not provide enough discretion and creates too many potential edges. Through trial and error, it was determined that an upper threshold of 0.275 works best. MATLAB automatically sets the lower threshold as 40% of the upper threshold. In this case, the lower threshold is 0.108. Running this Canny edge detection method over Figure 3.4 yields the binary image in Figure 3.5.

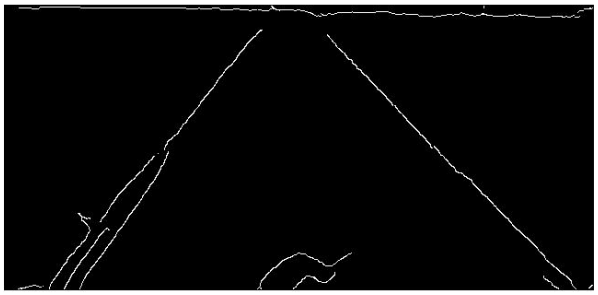


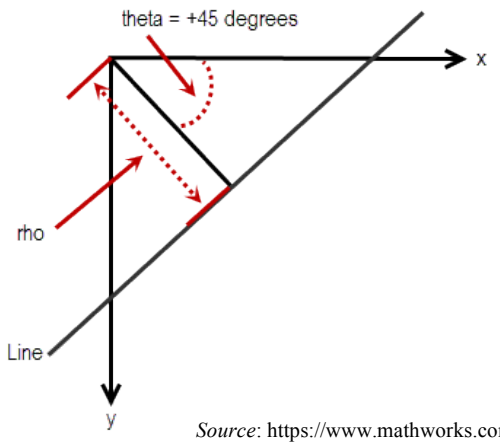
Figure 3.5: A binary image produced by running the image in Figure 3.4 through the Canny edge detection method with a lower threshold of 0.108 and upper threshold of 0.275.

3.2.4 Hough Transform

While it may seem like we have successfully detected the boundaries of the path, Figure 3.5 is still just a binary image. The white lines seen in the image are not actually lines at all. ; there are no endpoints or parameters attributed to them. Actual lines segments can be plotted over these edges using the Hough transform. These lines are conveyed in MATLAB in terms of endpoints, angle, and length. This data is crucial in determining the user's location relative to the path.

The Hough transform works by scanning each row of the binary matrix. When it comes across an edge, that is, when the value in the matrix is a 1, the hough method plots a large number of lines through that point. These lines are represented using only two variables: θ which represents the angle between the line and the x-axis, and ρ which represents the perpendicular distance from the line to the top left corner of the image [10]. The parameterization of these lines is depicted in Figure 3.6. A line described by θ and ρ in x-y space looks like a line. However, in θ-ρ space, the line appears as a single point. To be clear, a line in x-y space appears as a point in θ-ρ space [11]. The next line passing through the original point in the binary image has slightly different values for θ and ρ than the previous one. This means the new point in θ-ρ space is plotted slightly away from the last point. The process continues with each new line being slightly different from its predecessor. The points in θ-ρ space start to produce a trend. The sheer number of lines passing through the original matrix point creates a sinusoidal curve in θ-ρ space dictated by the equation

$$\rho = x_i \cos\theta + y_i \sin\theta \qquad (3.2)$$



Source: https://www.mathworks.com/help/images/ref/hough.html

Figure 3.6: A simple depiction of how a line can be parameterized using rho and theta.

This means that a point in x-y space corresponds to a sinusoidal curve in θ-ρ space [11]. Repeating these steps for every point in the binary image produces the images seen in Figure 3.7.

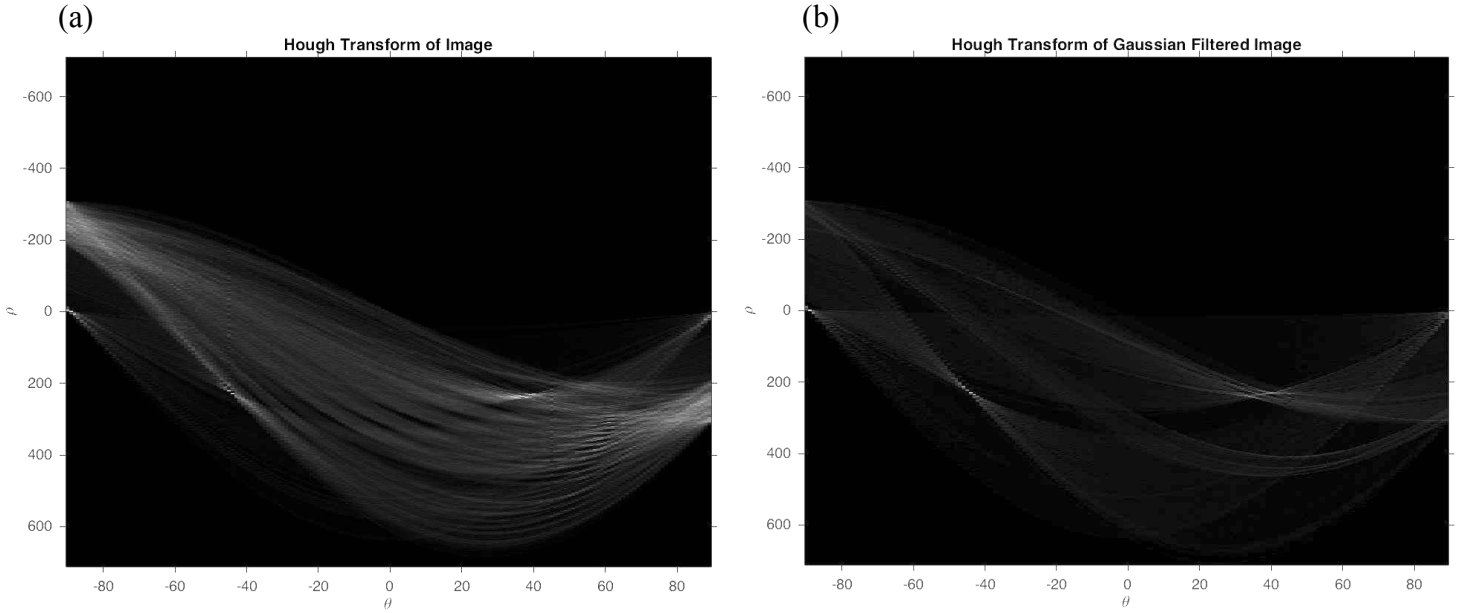


Figure 3.7: The graph depicts points on an x-y plane as sinusoidal curves on the θ-ρ plane. The intersections of the sinusoidal curves represent lines that connect those points in the original image. (a) The left graph is the Hough transform of Figure 3.2 without Gaussian filtering. (b) The right graph is the Hough Transform of Figure 3.2 with Gaussian filtering.

Since points in θ-ρ space represent straight lines in x-y space, the intersection of sinusoidal curves represents a line in x-y space that pass through all of the corresponding points[11]. The point with the most intersections in θ-ρ space is the same as the line that connects the most points in x-y space. The more points in the Canny edge detection image, the more curves in the Hough transform. Figure 3.7 (a) shows the number of curves in the image without the 2D

Gaussian filter while Figure 3.7 (b) shows the number of curves in the image with the 2D Gaussian filter. Reducing the number of curves in the Hough transform reduces the number of unnecessary intersections and lines, emphasizing the stronger peaks. The graphs in Figure 3.7 span all angles of θ between -90 and 90 degrees. A large number of intersections occur on the fringes of the graph, between -80 to -90 degrees and between 80 to 90 degrees, indicating the presence of several straight, horizontal lines. Since the focus of this section is to remain on a straight path, the range of θ was limited to values between -65 and 65 degrees. This eliminates any horizontal or near horizontal line from being examined, keeping the focus on the angled boundaries of the path at hand. The Hough transform looks for the strongest peaks in the θ-ρ image representing the longest connective line segments which appear to be at -40 and 40 degrees. The function then outputs this information as an array that contains the values of both endpoints in the x-y plane as well as their values for θ and ρ [12]. Figure 3.8 depicts the information in the line arrays plotted on top of the cropped image from Figure 3.1(b), showing that the lines do in fact follow the edges of the path. For this specific application, the output number of lines was limited to two since there are only two edges of the path.

Figure 3.8: The line data retrieved from the Hough transform is plotted over the cropped image from Figure 3.1(b) demonstrating the edges of the path were successfully found.

Lines with θ values greater than zero, representing the left edge, are plotted in green and lines with θ values less than zero, representing the right edge, are plotted in red. The endpoints of both lines are plotted yellow. The colors have no bearing on the program itself but makes it easier for us to see what is happening. Overlaying the Hough transform on the original cropped image seen in Figure 3.9, shows that the lines effectively follow the edges of the path.

3.3 Output of Information

Now that there are actual data structures, it is possible to use this information to prevent the user from wandering too far left or right. The user would be wearing stereo headphones and the information would be conveyed to them through sounds in each individual ear. This is accomplished by looking at the slope of the lines. If the user drifts too close to the left side of the path, the left line, depicted in green, appears more vertical, and if they drift too close to the right side, the right line, depicted in red, also appears vertical. These effects can be seen in both

images of Figure 3.9. This means that as the user drifts to either side, the slope of that line approaches infinity. Therefore, the program institutes a threshold for the slope of the left and right lines. If the absolute value of the left slope is greater than 1.5, an electronic chime sound is played in the left headphone, signaling to the user that they are approaching the left edge of the path. If the absolute value of the right slope is greater than 1.5, an electronic chime sound is played in the right headphone, signaling to the user that they are approaching the right edge of the path. The user would then correct their course based on which side of the headphones played the sound, allowing them to safely walk down the rest of the path.

(a) (b)

Figure 3.9: (a) The Hough transformed overlaid on an image where the user is standing on the left edge of the path. Notice how vertical the green line appears. (b) The Hough transformed overlaid on an image where the user is standing on the right edge of the path. Notice how vertical the red line appears.

3.4 Chapter 3 Conclusion

3.4.1 Key Findings

1) The program can successfully detect the boundaries of pathways using a combination of the 2D Gaussian filter, Canny edge detection, and the Hough transform. The program can also provide auditory feedback that prompts the user to stay on course. Overall, the program seems to be a satisfactory foundation for a future guiding system.

2) The edge detection can be finicky if the distinction between the path and surrounding area is not well defined. Abnormally large, narrow, or round paths can also cause confusion in the edge detection process and yield unusable results.

3.4.2 Suggestions for Further Research

1) Re-explore the thresholds in the 2D Gaussian filter, Canny edge detection, and Hough transform functions. There are many combinations of these thresholds that could affect each other in different ways. Re-examining the application of these thresholds might yield better images than the values I used.

2) Experiment using both of the ELP 720p Super Mini USB cameras and determine if stereoscopic vision has the potential to increase the accuracy of path detection.

3) Find a better solution to determine the at what point the user drifts too far left or too far right. Currently, there is an absolute slope threshold of 1.5 on both sides that triggers the auditory output when exceeded. Since paths take on a variety of shapes, this absolute value is ineffective in some cases. I believe there is a relation between the slope of the lines and the distance between the bottom endpoints that could provide the solution.

4) The output to the user should not be auditory in nature. The sounds from the system became irritating after a short period of time. Investigate tactile feedback systems such as vibrations or maybe determine what works best for a majority of the visually impaired population.

4 Detecting Objects *by Patrick Rice*

4.1 Introduction

Using images captured from two ELP 720p USB Camera Modules, I experimented with different techniques to find the best method of locating potential hazards within the user's path. Objects such as: people, benches, trees, etc. all pose a potential danger to the visually impaired. These objects often have unique characteristics that can be utilized to extract them from a larger image. An object's color, edges, and position relative to the camera are all features that can be used in the detection process. The following sections of this chapter explore two different method of detecting objects. The first explains a method for detecting objects solely based on their color while the latter gives a more in-depth process that utilizes stereoscopic imaging.

All the techniques mentioned in the sections of this chapter are carried out using MATLAB software. MATLAB stores image data in m by n by 3 arrays, where m and n represent the images resolution. Each pixel of the image has a corresponding red, green, and blue value. These values are unsigned 8-bit integer values that range from 0 to 255 that are recorded inside the array. [13] Once the image is stored, one can mathematically manipulate the arrays to further isolate important characteristic from the image data.

4.2 Detecting Objects of a Single Color

One method of object detection is to differentiate objects based on color. The goal of this experiment was to create a program that can identify red objects within an image. At first glance one might be tempted to isolate pixels that meet a certain threshold in the red portion of the image array. Their reasoning being all pixels with a high red value must be part of a red object. However, colors such as purple, brown, and while all have equally large red values and so the problem becomes finding a color space where different colors are represented in such a way that they can be filtered easily. Therefore, the program starts by converting the image data from the typical red, green, blue color space (RGB) to hue, saturation, value color space (HSV). In HSV color space hue refers to the angular position of the color on a color wheel. Because each color

has a distinct angular position it is possible to isolate the red section of color wheel without fear of including any unwanted colors. [14] Saturation describes how white the color appears to the naked eye. Holding all else equal, pixels with low saturation appear less vivid and often have a gray tinge. Value on the other hand describes a pixel's brightness. Pixels with a value of zero appear black and become lighter as value is increased. By placing thresholds on a pixel's value and saturation, it is possible to filter out certain shades of colors as well as any pixels that entirely black or white. [14]

Figure 4.1: Original image (left). Binary Image created using specified HSV thresholds (right).

To demonstrate the effectiveness of isolating objects based on color, consider the left image in Figure 4.1. The image contains several red balloons mixed in with balloons of various colors that the program is trying to differentiate between. The right image in Figure 4.1 shows how the program creates a binary array with entries of either 1 or 0 to mark which pixels are within specified hue, saturation, and value thresholds. Once the program processes the image into a binary format, one can locate the red objects using MATLAB's Image Processing Toolbox, which denotes large clusters of white pixels in the binary image as distinct objects. The result of this process is shown in Figure 4.2 where the program has placed green bounding boxes around each of the red balloons. It is important to note that while the program succeeds in identifying where all the red balloons are in the image, it mistakenly identifies some balloons as two separate objects. This is a flaw in the program as it has trouble when objects are partially obscured. Detecting monochromatic objects may be an overly simplified version of our expressed goal for this research, however the ability to differentiate between objects of varying color is an important step in object classification that could be utilized later.

Figure 4.2 The final output from the color detection program. Each green bounding box represents what the program has identified as a distinct object.

4.3 Utilizing the Difference in Stereoscopic Images

4.3.1 Theory

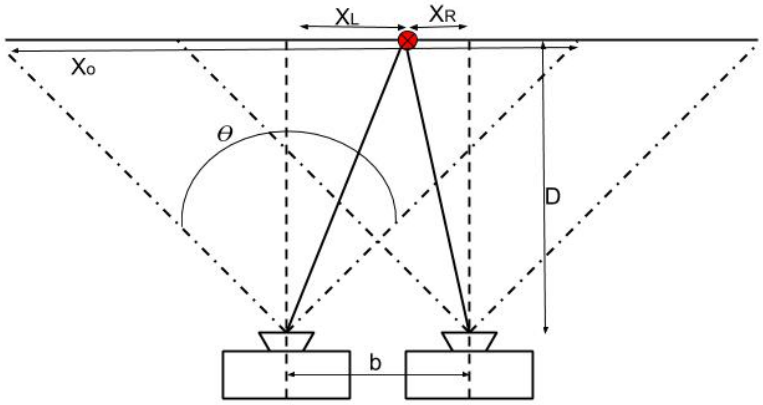


Figure 4.3 Diagram for stereoscopic imaging set up

Stereoscopic imaging is the process of taking images from two cameras that are horizontally offset from one another to create the illusion of depth. This is demonstrated in Figure 4.3 which shows the two cameras are separated by a distance b and there is an object a distance D from the cameras. The object appears x_L pixels from the center of the image taken by the left camera and x_R pixels from the center of the image taken by the right camera. Using basic trigonometry, we find: [15]

$$D = \frac{bx_o}{2\tan\left(\frac{\theta}{2}\right)(x_L - x_R)} \quad (4.1)$$

This function shows that the distance an object is from the cameras is inversely proportional to the disparity (in pixels) between its position in both images. To verify this

relationship and obtain a value for the constants in equation 4.1, I took several pairs of images containing a target and recorded the disparity between the targets location in each. The graph in Figure 4.4 shows that the relation is indeed as described in equation 4.1 and gives a value of 14.22 meters times pixels for the relevant constants.

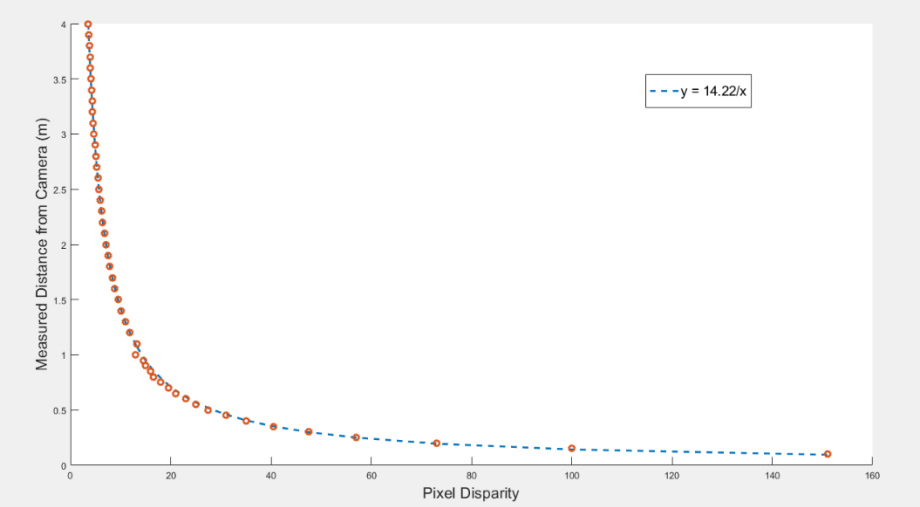


Figure 4.4: Graph showing the relationship between the difference (in pixels) of a target's location within two images vs. measured distance away from the cameras

4.3.2 Picture Analysis

The program created for object detection works using images captured by two ELP 720p USB Camera Modules set up in the configuration shown in Figure 4.3. The technique for detecting objects in the images focuses mainly on the array created by taking the difference between the two images. Based on equation 4.1, as an object's distance from the cameras increases the difference between its location in each picture approaches zero. This means that objects in the background of the images share the same position in both images, while objects closer to the cameras display a horizontal shift. The steps the program takes to identify these shifts are laid out in the following sections.

4.3.3 Loading Images

The program loads image data similarly to the color detection technique but in addition converts the unsigned 8-bit entries to double precision values. This simply means that every integer in the original image arrays are converted to floating-point values capable of being either positive or negative. Figure 4.5 shows the type of images the program uses as input data. From these images, it's possible to see that the person within the path has shifted from one image to the other, but it hard to determine by exactly how much. In order to see this shift more clearly the program subtracts the two images to create a difference array. The easiest way to visualize this array is as a heat map (shown in figure 4.6b). Pixels with a light green color represent a value of zero while positive and negative values are displayed as red and blue respectively. Values closer to zero in the difference array represent pixels in each of the respective images that share the same physical point in space. In contrast, the portions of the image that are a dark blue or red denote an object's horizontal shift between the two original images.

Figure 4.5: A sample pair of stereoscopic images with an object inside the path

4.3.4 Adjusting for Vertical Misalignment

A necessary requirement for using equation 4.1 to estimate the depth of object in a set of images is that the stereoscopic cameras are aligned vertically. To adjust for any vertical misalignment between the two images, the program examines the total difference in pixels as one image is translated downwards. The point where the total difference between the two images is at a minimum represents when the two images are vertically aligned. Due to small differences between the two cameras the program determined the right image was shifted 35 pixels up compared to the left. To correct for this the top 35 rows of the right image and bottom 35 rows of the left image are cropped out. The result (shown in in Figure 4.6b) is a difference array of two vertically aligned images.

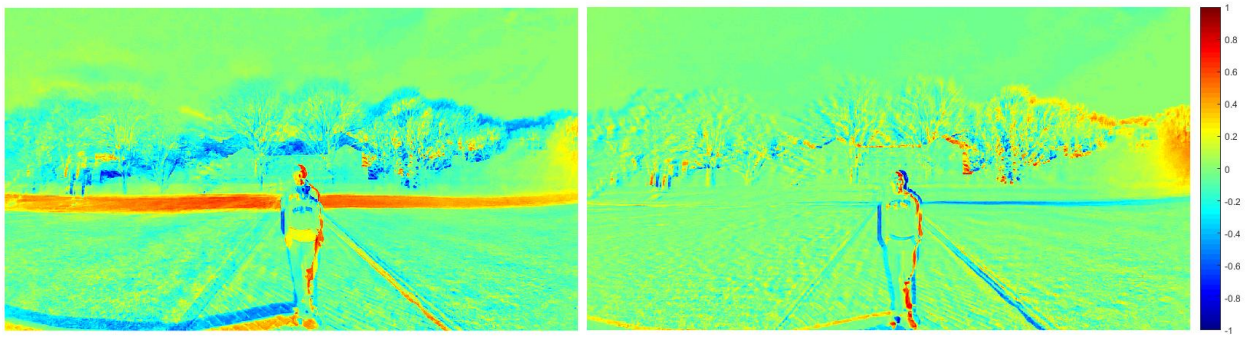


Figure 4.6a: Original difference array
Figure 4.6b: Difference array after correcting for unwanted vertical shift.

4.3.5 Determining the Boundaries of the Object

To better pick out the portions of the array that represent the object, the program first crops the image to focus on the middle 800 pixels of the image as is demonstrated in Figure 4.7. This does not affect the final outcome of the program as it is assumed that the user is in the middle of his/her path and does not need to worry about objects outside of it. After cropping the image, the program creates a "coarse-grained" array by setting entries of the difference array that meet a certain threshold to 1 and entries that do not to 0. It does this separately for the "cold" (negative) and "hot" (positive) portions of the array. The resulting binary images are shown in Figure 4.8. From this figure, it is easy to see the difference between what the program has determined to be unnecessary background information and the object it is trying to detect.

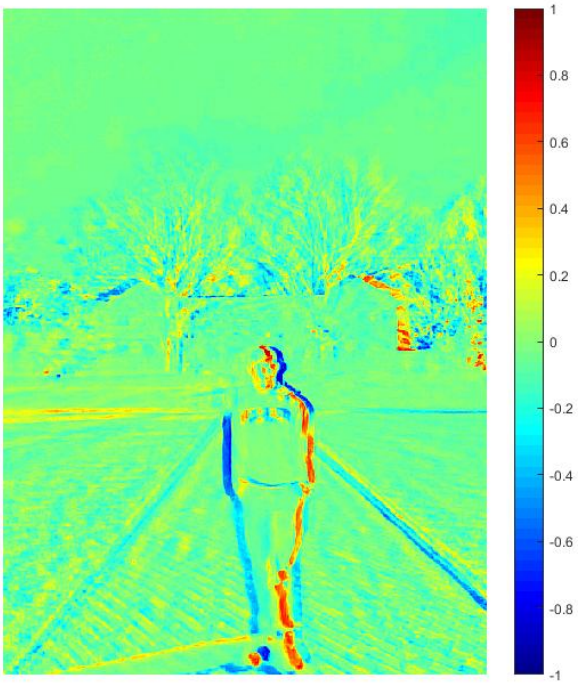


Figure 4.7: Heatmap image of the difference between the two stereoscopic images in Figure 4.5.

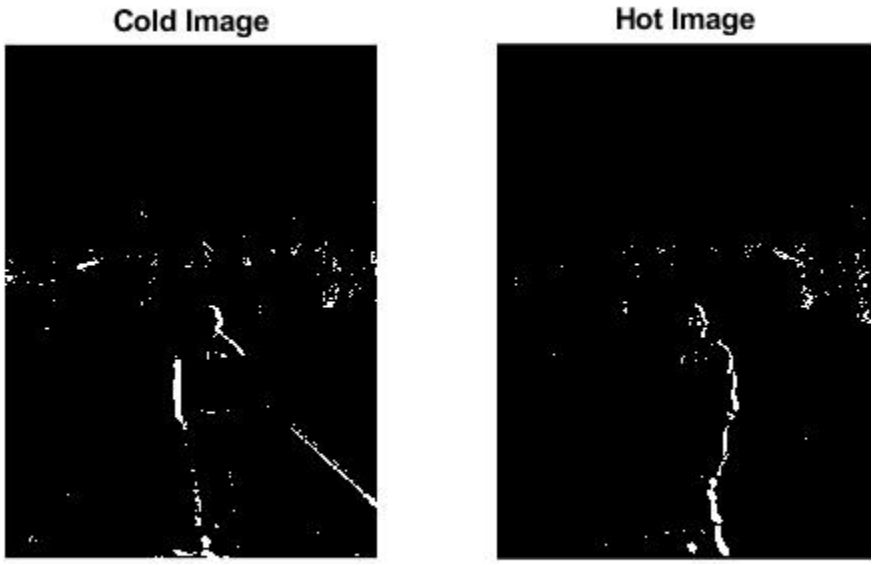


Figure 4.8: Coarse-grained images based on a preset threshold.

The next step in program is to determine the left and right boundaries of the object. Examining the column sum of the entries in the coarse-grained arrays, one expects to see larger values in areas that correspond to the portion of image containing the object. The graphs in Figure 4.9 show just that; the large spike in values in the left graph corresponds to the “cold” side of the object. Similarly, the peak in the right graph demonstrates where in the difference array the “hot” side of the object is located.

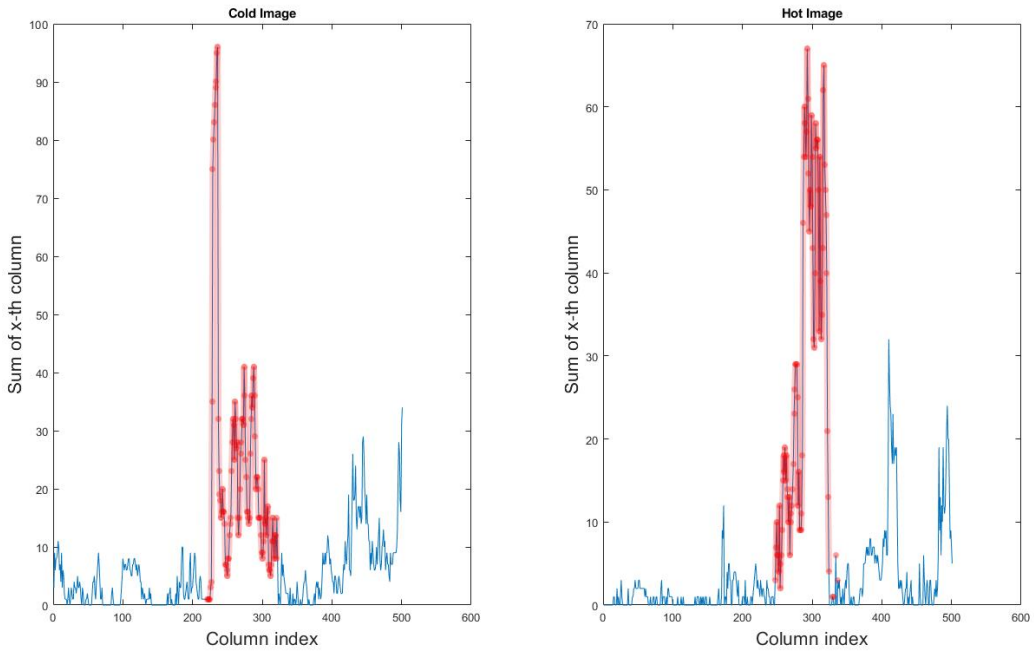


Figure 4.9: Plot of the column sums vs column indices for both "hot" and "cold" arrays

By differentiating the functions in Figure 4.9 it is possible to get a value for where the object's shift begins and ends. Figure 4.10 shows the result from this differentiating. The sharp spikes refer to points of significant change in the column sum values. Consequently, the maximum and minimum values in these plots signify the edges of the object in both of the original stereoscopic images. For example, in Figure 4.10, the global maximum in the left graph at x = 228 represents the object's left boundary in the left image. The global minimum at x = 236 represents the left boundary in the right image.

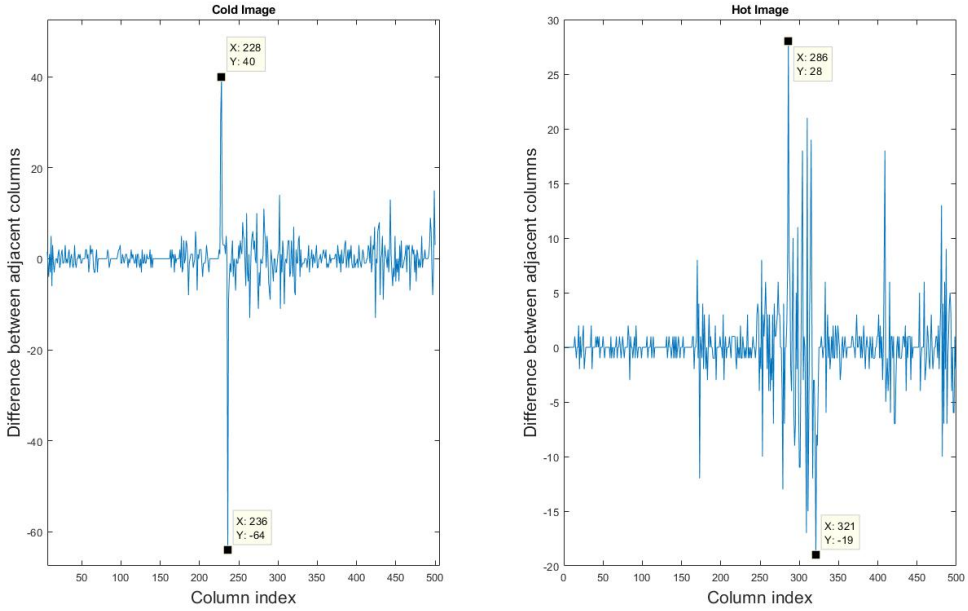


Figure 4.10: Differentiation of the column sum plot with respect to column indices

Once the program determines the object's left and right boundaries it corps the original image further to only include the columns that have the object inside them. The program then finds the top and bottom boundaries in a similar fashion as the left and right by taking the row sum of the cropped image. Once the program determines all the boundaries associated with the object it outputs an image with a bounding box around the region of interest. This output image is shown in Figure 4.11. Furthermore, objects horizontal shift can be used in equation 4.1 to conclude how far the object is from the user. This information is valuable as it lets the user know how far before he/she has before a possible collision.

Figure 4.11: Final image with bounding box around the object

4.4 Chapter 4 Conclusion

4.4.1 Key Findings

1) Detecting an object based on its color is a viable method when the object is relatively monochromatic.

2) The program designed to analyze sets of stereoscopic images was able to identify potential hazards within a path based solely on the difference between the two images. Objects approximately 1 to 2 meters in height with straight edges are detected once they are within 2 meters of the user. However smaller objects with less defined edges were difficult of detect.

3) Once an object has been detecting accurately in both images the horizontal shift is measurable and correlates to how far away the object is to the user. This information can be relayed as an output of the program and gives the user an idea of how far he/she can travel before a collision is likely.

4.4.2 Suggestions for Future Research

1) Explore the use of computer learning algorithms as a means of identifying and classifying objects within an image. These types of programs require a large amount of time to train but result in a more versatile method of object detection.

2) Consider using the images acquired from the two webcams to create a three-dimensional representation of the user's surroundings.

3) Build a self-contained unit with a microprocessor capable of running the different programs described in the paper. If this project is to one day be a wearable device to assist the visually impaired more research is required on the necessary hardware.

5 Detecting Elevation Changes *by Amit Verma*

5.1 Introduction

Following the detection of paths and objects, the detection of sudden elevation changes is equally vital to a person being able to walk safely. Changes in altitude while walking are common, and an overwhelming majority of these changes do not present themselves as gradual changes. The most frequent instance of a sudden elevation change would be stairs, and these can pose as both an inconvenience and danger to the visually impaired. Figure 5.1 illuminates the necessity of an elevation detection system. For the figure, if path detection alone was used for walking assistance, then the individual would be kept on the sidewalk, but there would be no warning of the upcoming steps.

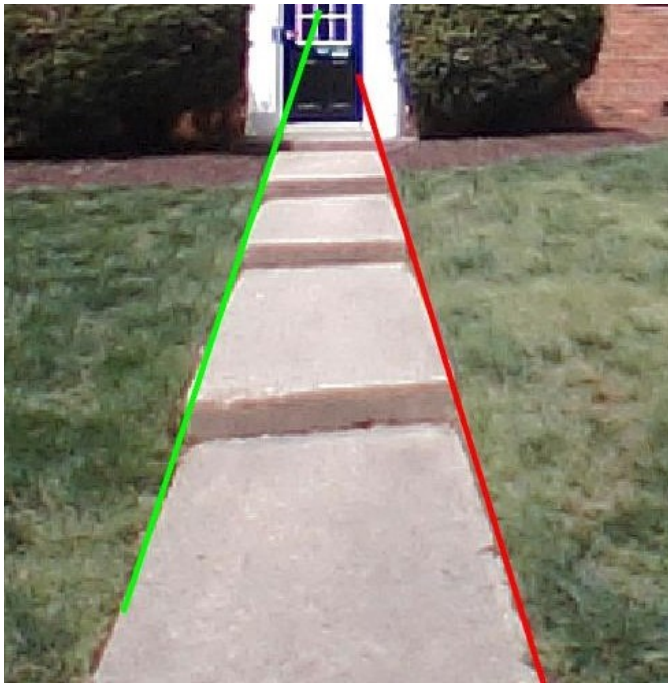


Figure 5.1: Steps leading up to house passed through path detection system, with no indication of the stairs.

The picture above demonstrates the need for a complete guidance system to have a method to detect elevation changes. The proposed solution is to use a distance sensor mounted on the body, that scans the ground immediately in front of the individual walking for significant deviations in elevation. When a change in elevation is detected, the device relays this to the person and instructs them on how to adjust their actions. More specifically, the primary concern of this elevation system is to detect the first step going up or down, as once a person manages the first step, it is relatively easy to continue taking steps without needing to know their exact position due to the consistent size of stairs. Similarly, gradual changes in elevation such as slopes and ramps are not necessary to detect because they are possible to navigate solely through walk and feel, although if the change is significant enough the sensor would detect it.

5.1.1 Overview of Chapter

Section 5.2 and 5.3 discuss the sensor used, including its functionality and tests of its limitations. Section 5.4 discusses tests with the rolling board, with the rocking motion from walking was eliminated. Section 5.5 explains the issues associated with walking, as well as tests with the sensor on the shoe and on the shin. Section 5.6 concludes the chapter with a summary of findings and suggestions for further research.

5.2 The Sensor

The sensor used for the project is a Time of Flight (ToF) sensor. Specifically, it is the VL53L0X by Adafruit. The sensor is meant to be used with a microcontroller, and for this

project it is used with an Arduino Uno connected to a laptop to read distance measurements off a serial monitor. The sensor costs about $15 and along with its housing and circuit board is smaller than a quarter. Figure 5.2 is a pinout diagram for the sensor and an Arduino board.

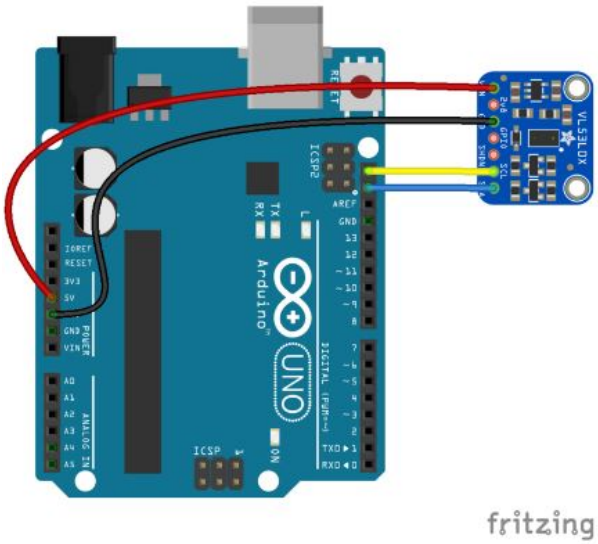


Figure 5.2: Pinout diagram of Arduino Uno and VL53L0X sensor [16].

This ToF sensor is an infrared sensor, and it works by sending out an IR laser at 940 nm perpendicular to the sensor and then timing how long it takes for the light to reflect off a surface and return to the sensor. Based on the time it takes for the light to return, the sensor uses the speed of light and calculates the distance to the surface [17], and the measurement is then displayed on a serial monitor. The sensor is marketed with a range of 30 mm to 1000 mm and a field of view of 25°. A maximum range of ~1 meter means that the sensor needs to be attached below a person's waist to detect the ground, and the consequences of this limit on positioning will be revisited later. The delay on how often the sensor takes measurements is configurable within the code used, however there is an internal delay of 35 milliseconds for the sensor to carry out a reading. Another limitation of the sensor is its inability to take readings from all surfaces; darker surfaces pose a problem because of their lower reflectance, and this can result in an error code if the sensor is detecting these surfaces. The code used for all the tests is a modified version of the sample code given for testing the sensor online.

The other common devices used for distance measurements are cameras and ultrasonic sensors. The IR sensor was chosen over these other options for a couple of reasons. Namely, it was cheaper, smaller, and more precise, which were the pertinent choices to be made. Cameras are too slow at taking distance measurements compared to IR sensors, and while slower measurements are acceptable for path and object detection because those are things that can be detected in the distance and stay relatively constant with motion, stairs need to be detected quickly, and within a certain range to ensure a warning can be given in time. Ultrasonic sensors were not used because they need to have a larger detecting area, and this makes errant measurements that could result in inaccurate directions for guidance more likely.

5.3 Sensor Tests

Two accuracy tests had to be carried out to test the VL53L0X's limitations. The purpose of the first test was to test the distance accuracy for the sensor as well as the reported min/max range of 30 mm to 1000 mm. The second test was designed to measure the sensing cone against the field of view given in the datasheet of 25°.

5.3.1 Distance Accuracy Test

To test the accuracy, the sensor was set up at the end of a table, with a measuring tape set out along the table. A flat object was then moved down the tape with measurements taken intermittently. A picture of the set up for the test is below.

Figure 5.3: Image of accuracy test set-up.

Measurements were taken by setting the sensor delay to one second and taking ten readings. The ten readings were then averaged and plotted on Figure 5.4. More measurements were taken towards the ends of the sensor's range as these were important positions. The first graph below is of the entire range tested. The second is of the section from 0 mm to 100 mm to show how the accuracy tails off at short distances. The dotted line represents how a perfect sensor would function.

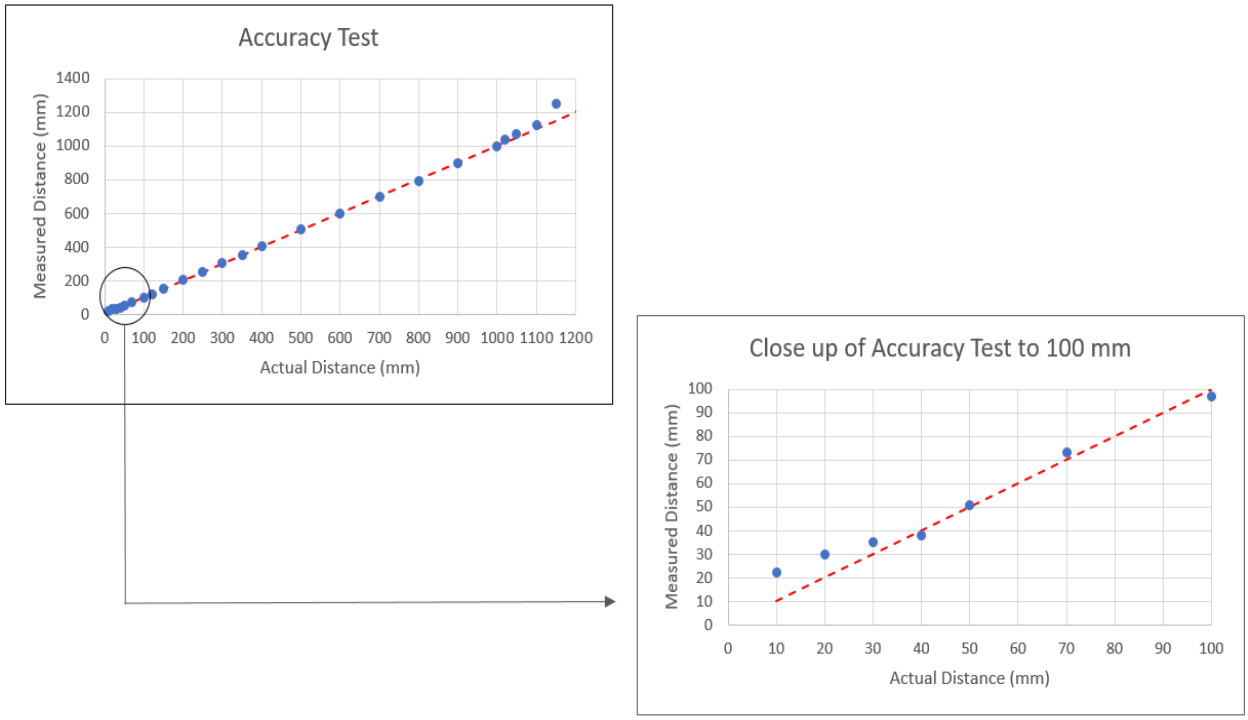


Figure 5.4: Graph of accuracy test for sensor, tested from 10mm to 1150mm. Second graph is a close-up of measurements from 10mm to 100mm. Note that the accuracy tails off before 40 mm and past 1000 mm.

The linear portion of the graph, from 40 mm to 1000 mm shows that the sensor is accurate, with a small error of 1.49%, for this range. Inaccuracy in measurements below 40 mm is acceptable, as once an elevation change is within 4 cm it is likely too close for any warning to be effective. To set a standard practice for data-taking, any measurements that were given as out of range by the sensor were set to be reported as 2.5 m. This was outside the reported range for the sensor, and ensured that the data was still usable and shown as well out of range.

5.3.2 Field of View Test

To test the field of view, the sensor was set on a swivel in line with the corner of a doorway. This was set as the angle of 0° for the sensor, and the sensor was then rotated away from the doorway. The goal was to rotate the sensor by a few degrees at a time and to take readings until the sensor completely lost sight of the doorway. These measurements were averaged from 10 readings at a one second delay. The three diagrams in Figure 5.5 illustrate stages of the test.

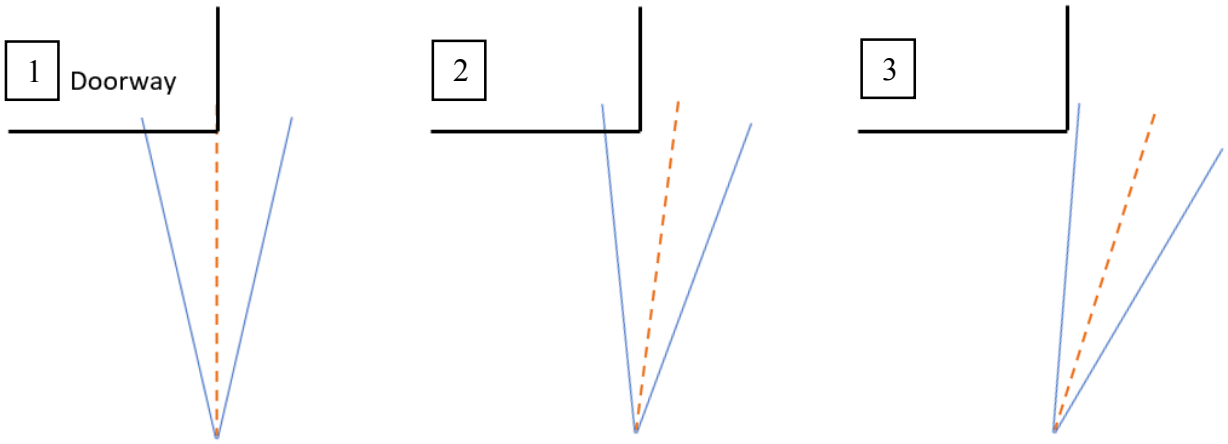


Figure 5.5: 1) the 90° line from the sensor is at the edge of the doorway, 2) sensor is rotated slightly so that middle of laser is away from the doorway, but the cone still detects the doorway, 3) the cone is entirely clear of the doorway and nothing is detected.

This only tested half of the field of view, so the test was replicated on the other side of the sensor, rotating the other way. The results are below, and although not incredibly precise, they are consistent enough with the value for the field of view given in the datasheet that we feel comfortable using 25° as the value. The results from this test at 500 mm were consistent for other distances tested as well.

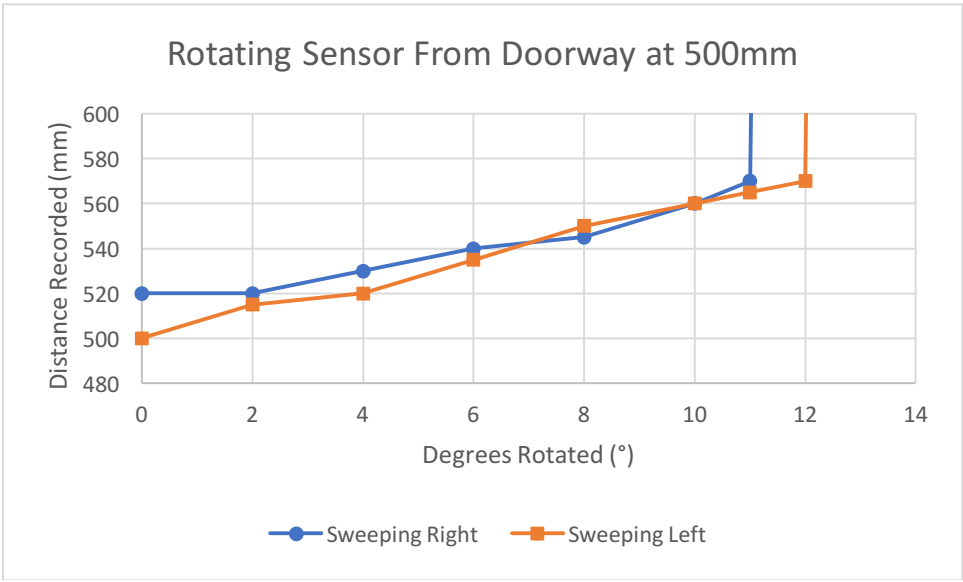


Figure 5.6: Graph of the two rotation tests, for both sides the tests were accurate until 11°-12°. Past these angles, readings became inconsistent and at 13°, all measurements were out of range.

Understanding the field of view is important, because it dictates how large of an area the sensor detects. The light from the sensor does not travel in a single line, it spreads from the sensor, and the further the sensor is from the ground, the wider the spread is. Using trigonometry,

and based off 1) the field of view, 2) the height of the sensor from the ground, and 3) the angle of the sensor with respect to the ground, the equations below can be derived for the length of ground the sensor "sees". The variables in the equations correspond to the diagram, with x_1 being the distance behind the middle line and x_2 being the distance in front. This is relevant for understanding exactly when the sensor should begin detecting a step as it approaches the stairs once it is set at a certain angle.

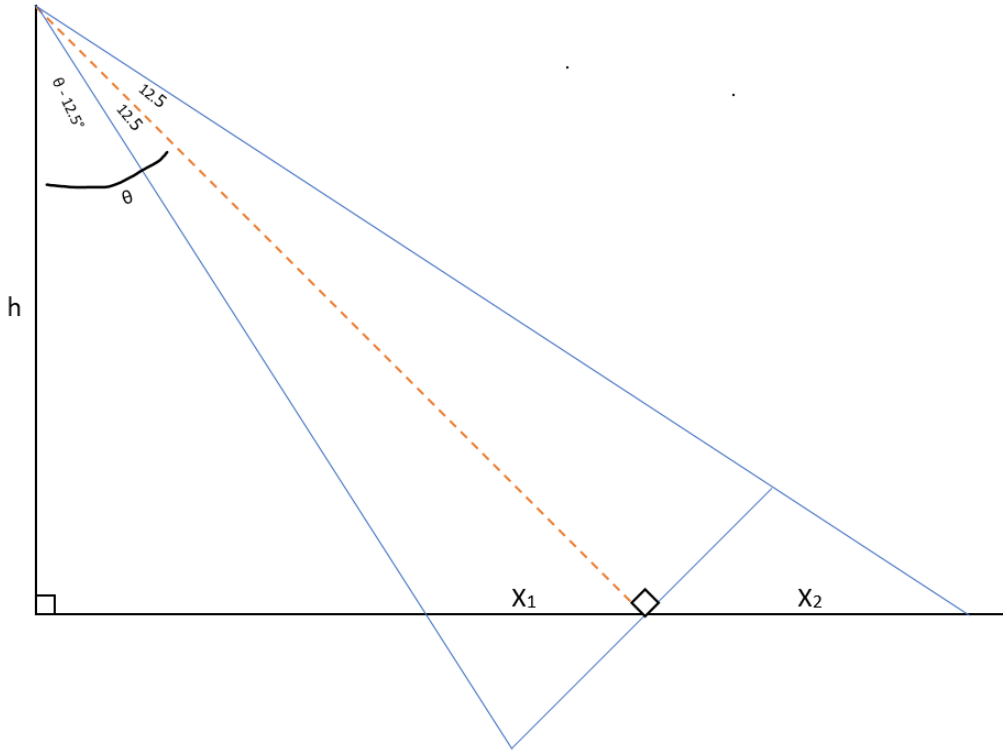


Figure 5.7: Illustration of sensing cone at angle θ to the ground. Note that on flat ground, x_2 will always be larger, but there will be a higher light intensity in x_1.

$$x_1 = \frac{h*\sin(12.5°)}{\cos(\theta)\sin(77.5°+\theta)} \qquad x_2 = \frac{h*\sin(12.5°)}{\cos(\theta)\sin(77.5°-\theta)}$$

The cone, and corresponding x_1 and x_2 also determine what the distance that will be given is weighted towards. That is, if there is significantly more surface to be reflected off in x_1 or if the surface is closer in x_1 (if it were elevated for instance), then the measurements given will be shorter than the straight-line distance from the sensor to the ground. This will be relevant in the next section.

5.4 Rolling Board

The rolling board consisted of attaching the sensor with the angle setter to a board with wheels. This group of tests had two sections and an image of the board is given in Figure 5.8. The sensor was set to 340 mm off the ground because this replicated the height of the sensor affixed to mid-shin, but the angle for the sensor was changed between the two parts.

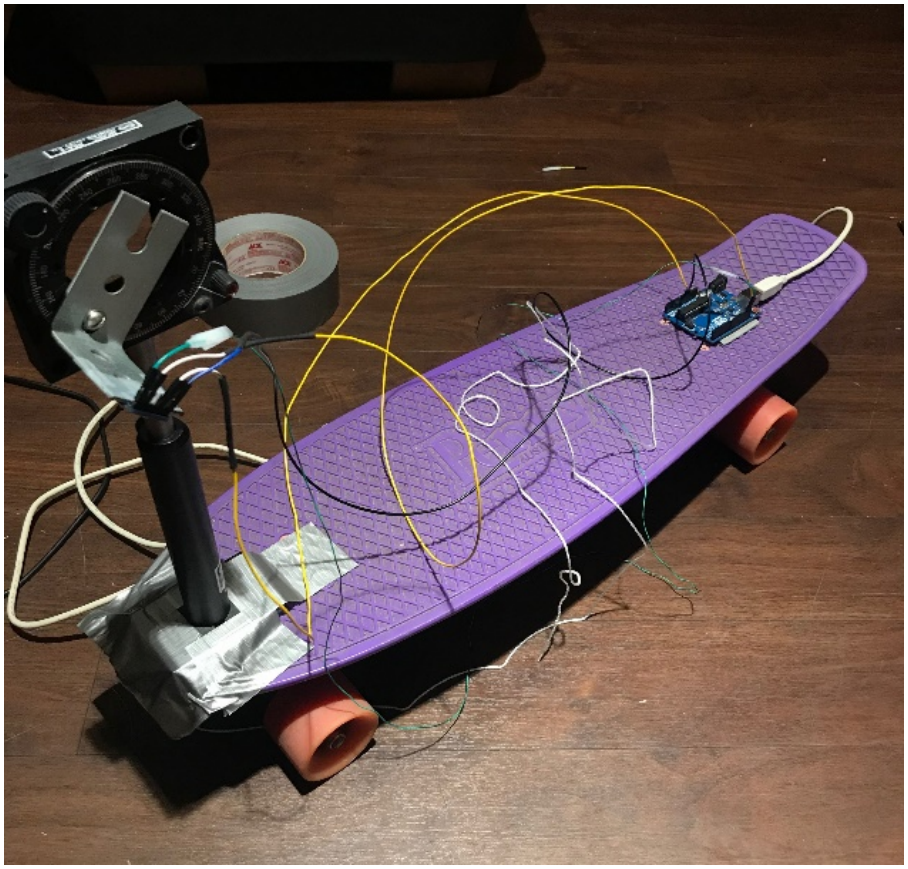


Figure 5.8: Image of board, sensor is on angle setter to the left and connected to Arduino Uno at the end of the board.

5.4.1 Rolling Board to Stairs

The first part involved rolling the board to a step, and the goal was simply to see what changes in elevation were recorded. This was performed on flat ground as well as for a set of stairs going both up and down. The delay was set such that measurements were taken every 100 ms, and the board was slowly pushed each time. The sensor was set to 45° for all three tests because this made calculating the horizontal range for the sensor simple, and was a middle angle that could be adjusted up and down as necessary for later tests depending on the results.

5.4.1.1 Rolling on Flat Ground

For flat ground, at an angle of 45° and a height of 340 mm, the distance reading for the hypotenuse was expected to be ~480 mm. The graph for flat ground data trial is given in Figure 5.9.

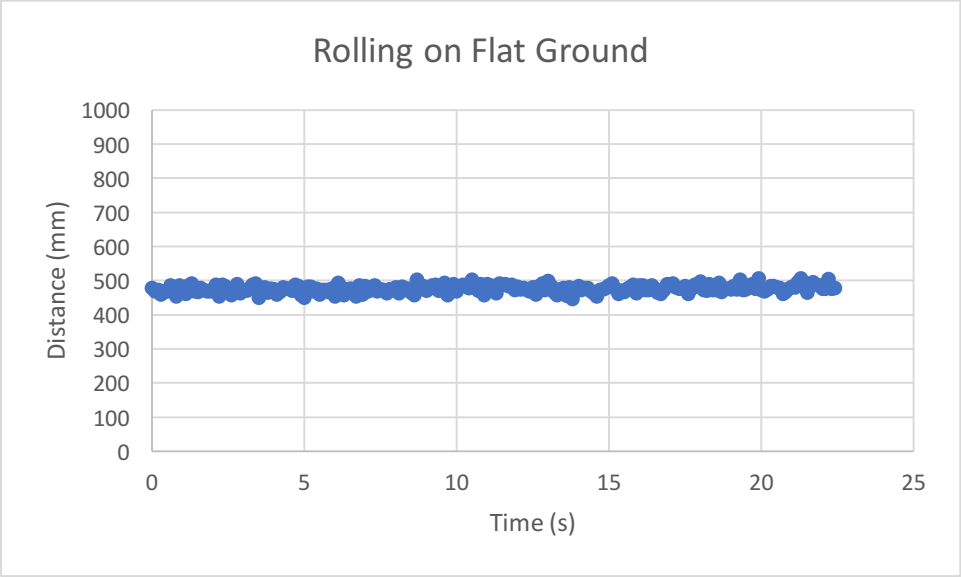


Figure 5.9: Graph of readings given by rolling on flat ground.

The graph shows an almost linear reading with zero slope, at an average distance of 476 mm. The zero slope was expected because the elevation is not changing, and the average measured value is close to the calculated value. There is an error of 1.95% in the readings from the expected reading of 480 mm, but the error is slight, and every deviation from the expected value is within 50 mm which is small in comparison to the height/depth of a step, meaning that it can essentially be disregarded. This trial creates a requirement for the trials going up and down, in that the elevation change they record should be initially zero like this trial and then show a change greater than 50 mm so that we can be sure that elevation changes are recognized by the sensor.

5.4.1.2 Rolling Towards Step Going Up

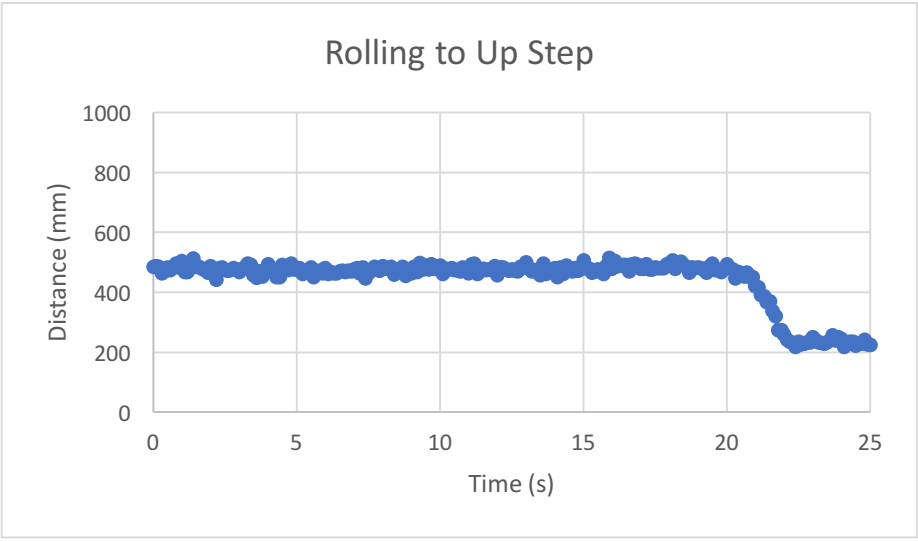


Figure 5.10: Graph of rolling on flat ground to step going up.

This graph shows a flat approach from 0 s to 20 s, and then the distance from the ground to the sensor starts to quickly shorten up from 21 s to 22 s, as the board approaches the first step. At 20s, the board is around 40 cm from the step, as determined from the second equation in 5.3.2 and the sensor being at 340 mm. Past 22 s, the measured distance to the first step has an average reading of 230 mm. The expected value for the distance from the sensor to the step is 226 mm, which is consistent with the measured result and a significant deviation from 480 mm. The diagram below helps to illustrate what the sensor is seeing as well as the calculation of the expected distance for the top of the step. The smooth curve from 21 s to 22 s is due to the sensing cone, as its weighting for where more values are coming from transitions from the bottom of the stair to the top.

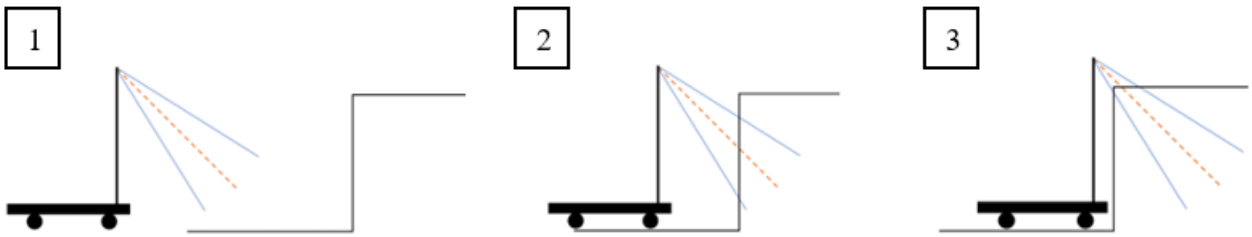


Figure 5.11: 1) Sensor rolling on flat ground towards step, 2) Sensor begins to detect the step and distance smoothly shortens up as it gets closer, 3) Sensor clears the step and forms a new 45-45-90 triangle with the higher ground which allows for the calculation of the expected final distance.

5.4.1.3 Rolling Towards Step Going Down

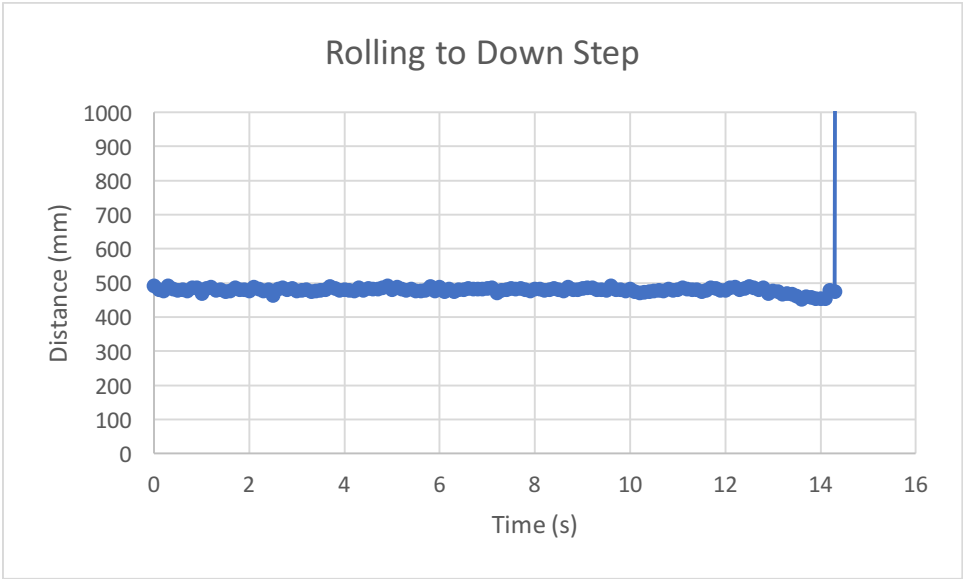


Figure 5.12: Graph of rolling on flat ground to step going down.

This graph shows a flat approach to the down step for 13 s, and then a slight dip in the distance readings until 14 s, where the readings go to infinity as the sensor fails to receive enough reflected light. The reason for the dip and the infinity reading is the field of view of the

sensor moving across the stair. As mentioned at the end of section 5.3.2, when there are significantly more reflected beams back from one side of the cone, the measurement ends up weighted in that direction. In this case, as the cone started to cross the edge of the staircase, the x_1 section behind the middle line dominated the reading. Eventually enough of the cone crosses that the sensor receives no reflection. The reading of no reflection is useful, because the sensor is seeing the down step as if the world is falling away, and this can be easily translated for the user as a major decrease in elevation. Related to this, an important takeaway is that it might not be necessary for the exact elevation change to be known for proper guidance, only that the threshold for what would be considered a severe change has been crossed, as well as its direction. Figure 5.13 is to assist in understanding the reason for the results.

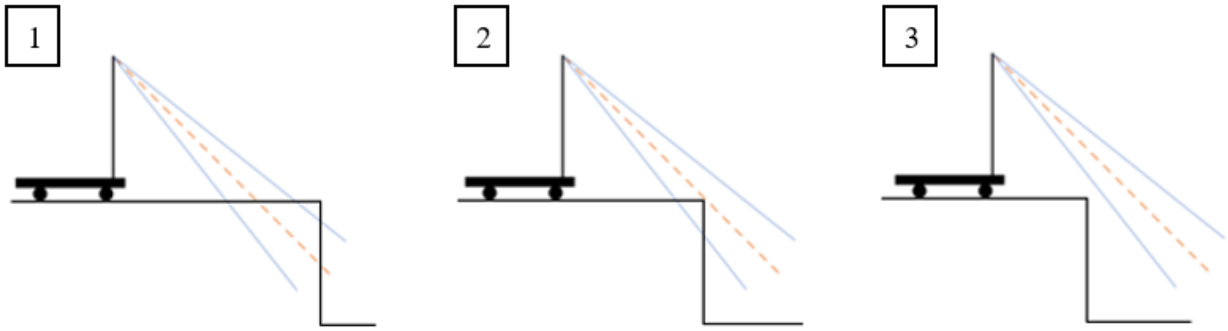


Figure 5.13: 1) The sensor is rolling on flat ground towards the edge of the step, 2) Sensor cone begins rolling past step, and at this point nothing is being reflected back in x_2 so reflected light in x_1 gives a reading that is shorter than the middle line/flat ground reading resulting in the dip in Figure 5.12, 3) The sensor cone completely passes the step and the distance measurements go to infinity as nothing is reflected back.

5.4.2 Predicting Distance for Stair Detection

Following the success of the sensor on the board in being able to recognize elevation changes, the goal of the second experiment with the rolling board became predicting at what distance the steps would be visible in data by converting time to distance. This consisted of connecting the board to a winch, setting it at a specific distance, and then rolling it at a constant speed and predicting at what point the sensor would detect the step. For this test, only the down step was used because it was more manageable physically than the up step, and the results would be cleaner because the distance goes to infinity. The reasoning for this test is that when walking, the necessary capability of the sensor is the functional distance for which it can detect stairs, rather than the time it detects stairs at.

The angle of the sensor for this test was adjusted to match the angle of the stairs. This meant changing the angle from 45° to 60°, and this was done to make the change of when the edge of the staircase was crossed more drastic, as well as to attempt to increase the range the sensor could see ahead.

The speed for the board was set to 21 cm/s with the winch, and the board was set to 208 cm from the edge of the stairs. At 60° and 340 mm, it was calculated that the sensor could see

~60 cm in front of the board. From this, it was predicted that the board would detect the down step first at 148 cm from the starting point (208 cm – 60 cm). Figure 5.14 is the graph of the data, and the time axis from earlier graphs has been converted to distance by multiplying time by the speed of the board. The prediction for when the sensor should detect the stair is on the graph as the solid line, and the dashed line gives the actual position for where the step was detected. The expected line is at 148 cm, whereas the actual distance on the graph where the stair is noticeable is at 135 cm. The sensing cone being at 60°, which tilts far more in favor ahead of the sensor's middle line than behind, may account for the 13 cm difference, which gives the board a detecting range of 73 cm instead of 60 cm. The sensor's readings are weighted slightly ahead because of the cone, and this allows the rolling sensor to detect the step earlier than calculated.

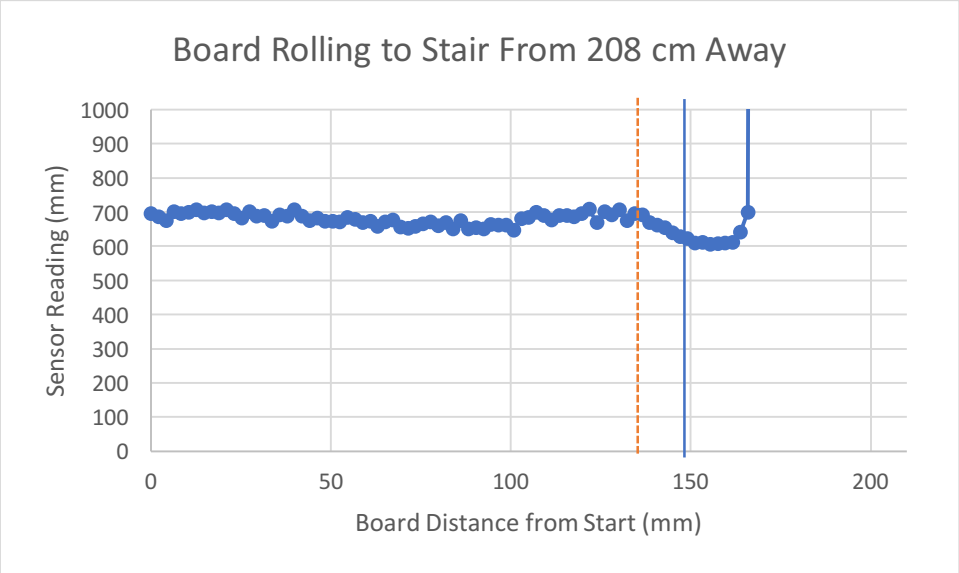


Figure 5.14: Graph of rolling to down step, similar to Figure 5.14 but with a larger flat ground distance and dip because of the shallower angle of the sensor.

As mentioned earlier, this test was not replicated for rolling to an up step, because it was difficult to maintain a constant speed, but given the results, it is reasonable to assume that the same method would work to estimate the distance to detect a step up.

5.4.3 Summary of Rolling Board Tests

This grouping of tests with the rolling board demonstrates that the behavior of the sensor with detecting elevation changes can be predicted and is consistent with what one would expect from the sensor. Stairs going up are visible in the data as a sudden shortening of distance readings, stairs going down are visible as the sensor going to infinity once the staircase begins, and flat ground is visible in the data as a level line with no slope. Additionally, if the distance from the stairs and the speed of the board are known quantities, then it can be reasonably estimated for the distance at which the step is visible to the sensor. This sets the stage for moving the sensor from a rolling board to a person, as a person can walk with regular speed and the sensor can be set to a specific angle/height on the body, but the crucial difference is how the movement associated with walking affects the readings from the sensor.

5.5 Walk Cycle

Due to the short maximum range of the sensor, the sensor must be placed somewhere below the knee to be effectively used. This portion of the leg is one of the parts of the body that moves the most while walking, and this adds an extra cyclical vertical movement when moving forward that was not present with the rolling board. Figure 5.17 is a drawing from an animator's website to illustrate this cycle, and shows that with each step the foot/leg swings through many angles.

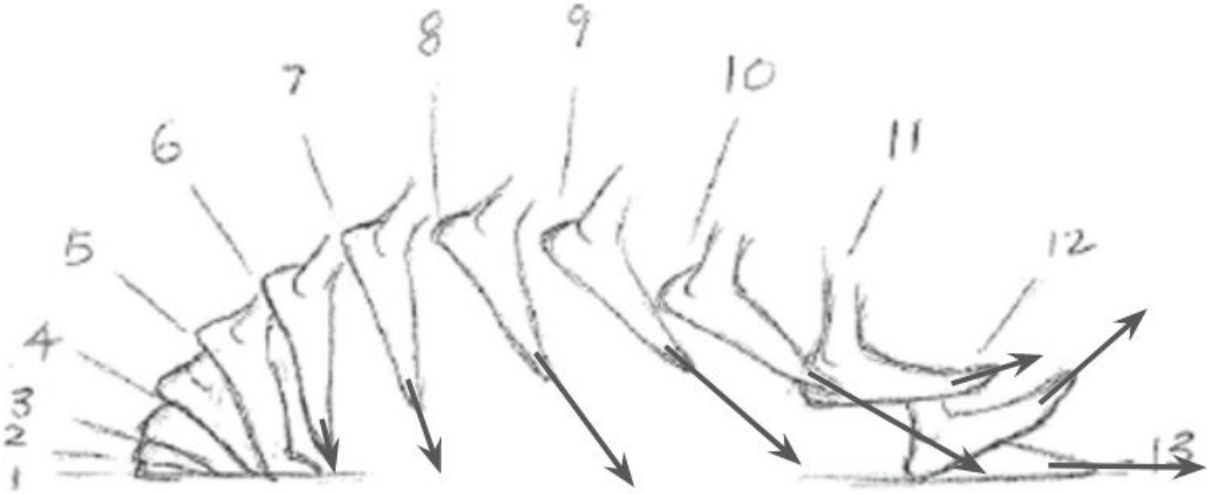


Figure 5.15: Walk cycle of foot [18], arrows are added to show the change in angles.

Consequently, it was believed that since the foot is at the end of the knee-leg lever system, the motion due to the walk cycle would be the most exaggerated with the sensor on the foot. To test this, the sensor was mounted on the front of a shoe, as shown in Figure 5.16.

Figure 5.16: Sensor on shoe, angle was set by fastening the sensor to a piece of metal.

The graph below shows the effect of the walk cycle, by giving sensor readings from walking on flat ground with the sensor at 90°.

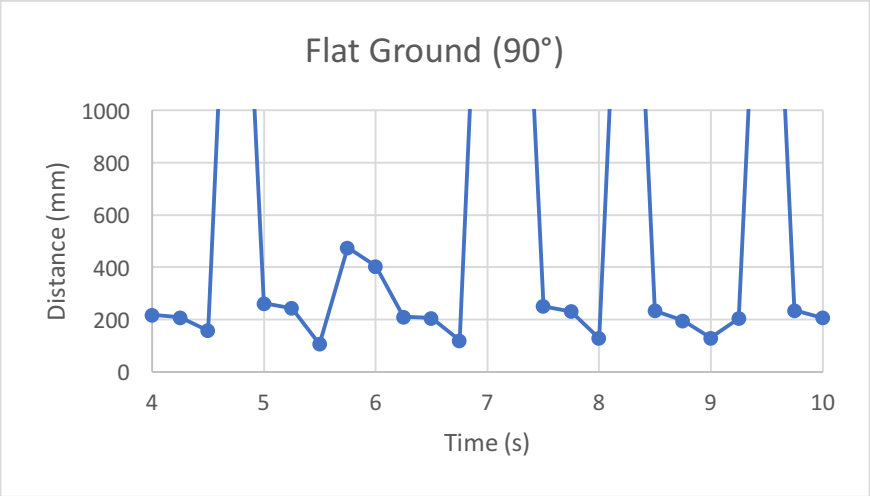


Figure 5.17: Graph of walking on flat ground with the sensor oriented at 90° on the front of a shoe.

The data is cyclic as can be seen by looking at the portions at 5 s, 7.5 s, and 8.5 s, but the deviations for each step are extreme, because for every cycle of the step there is a portion where the foot points away from the ground (label 12 in Figure 5.15), and the sensor goes to infinity. This arrangement would not work even if it was on the rolling board, because at 90° parallel to the ground the sensor should never detect anything, however the purpose of this test was to show the extra vertical motion detected by the sensor due to the walk cycle. The logical progression is to tilt the sensor down, so that the sensor stays oriented on the ground throughout the foot moving. So, as was done for the board, the sensor was set to 45°. The reasoning behind 45° is that it worked well with the field of view for the sensor on the board, in that it did not weigh the cone too heavily to the front, unlike 60°, and that it makes it simple to determine how far the sensor sees because the legs of the triangle are equivalent. Figure 5.18 is of the sensor on the front of the shoe at 45° walking on flat ground.

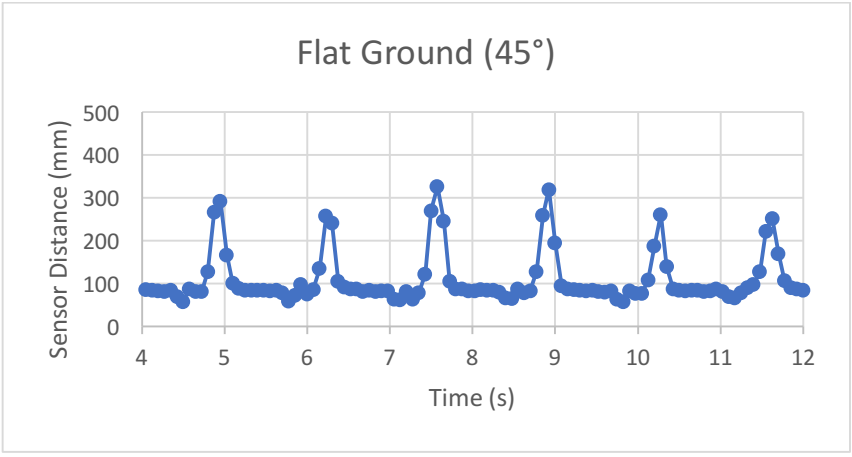


Figure 5.18: Graph of walking on flat ground with sensor at 45°.

For all the walking tests, the sensor is on the right shoe, so the graphs only show half of total movement, meaning that there would be another graph shifted half a period from these graphs for the left foot. As compared to the rolling board on flat ground (Figure 5.9), there is far more regular deviation in the sensor's measurements. The deviation is a considerable amount, at near 200 mm, which is close to the same deviation seen for when a step was detected by the sensor on the board. The effect of this is that if a distance threshold was set for identifying the step as mentioned at the end of the rolling board section, it would likely be errantly triggered during the step cycle. Nevertheless, for this graph the walk cycle due to the steps is much clearer than for 90°. The flat periods are when the right foot is stationary as the left foot is moving. The slight dips following the flat sections are the parts of the cycle (Figure 5.15) labelled 4-10 when the foot is pointed towards the ground and is moving forward. The rise and the peaks past the dips are from 11 and 12 when the foot swings away from the ground but the sensor at 45° manages to stay oriented towards the ground. The return to the ground are labels 13 and 1 where the foot comes back to a flat position on the ground to wait for the left foot to move.

Ideally, with a consistent cycle for the shoe it would perhaps be possible to have a running average and then trigger a threshold for when the average gets higher for a stair going down or lower for a step going up. For this sensor position though, there are a few issues that make it unrealistic for stair detection. For one, the height of the sensor for a distance reading of 85 mm at 45° means that the sensor is only at 60 mm, and can only see 60 mm ahead. This is not nearly enough of a detection zone to give time for a warning that can be acted upon to be relayed to the user. Additionally, because the sensor is on the foot, it is at the furthest point in the swing of the leg. The logical step is to move the sensor up the leg, to increase the distance the sensor is detecting over, and to reduce how much swing the sensor experiences with each step.

To mount the sensor higher, it was placed on a wooden brace fastened to the shin. The sensor was attached at mid-shin, 34 cm from the ground, equivalent to the height for the sensor on the rolling board. Then, like the rolling test in section 5.4.1, we walked to a staircase from below and above. Figures 5.18 and 5.19 are graphs of the two tests.

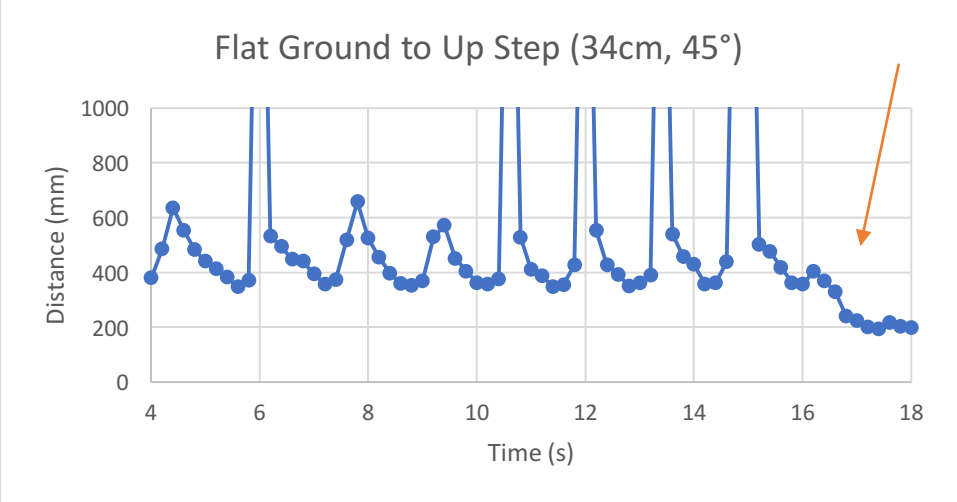


Figure 5.18: Graph of walking towards stair going up with sensor on brace. The arrow indicates the location of the step, and 16 s to 18 s should be compared to Figure 5.11.

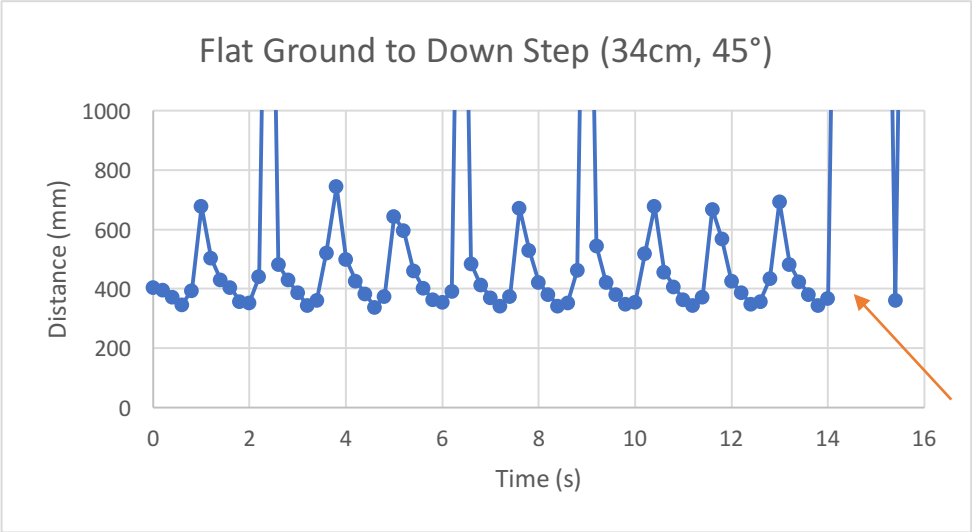


Figure 5.19: Graph of walking towards stair going down with sensor on brace. The arrow indicates the location of the stair, and should be compared to Figure 5.12.

There a few important things to note from these graphs. Crucially, for both tests, the stair at the end was detected. In the first graph, it is shown by the decrease in elevation at 16 s, where the stair is close to the sensor just as it was for the rolling board (Compare to Figure 5.10). In the second graph, it is shown by the sustained measurement at infinity at 14 s (Compare to Figure 5.12). Notice however that there are other measurements at infinity on the flat ground prior to the stairs for both tests. This was not due to the sensor swinging away from the ground as it was for the shoe test, but instead must be due to the sensing cone becoming too large at some points during the swing due to height. Somewhat similarly, the range of measurements per each step cycle was greater than it was for the sensor on the shoe. For the shoe, the range was from 85 mm-290 mm, but for the sensor at the shin it was from 380 mm-680 mm. This was counter to the reasoning for moving the sensor up the leg, and a different approach is needed to allow for walking and detecting stairs simultaneously. The issue is clearly not that 34 cm is too high, because it worked for the board, but that the movement of the leg is too severe to allow for the sensor to work properly.

5.6 Chapter 5 Conclusion

5.6.1 Summary of Findings

1) The VL53L0X sensor is accurate between 40 mm-1000mm with an error of 1.49%, and has a field of view of 25°.
2) In the absence of the rocking associated with walking, the sensor can identify both upward and downward elevation changes from 340 mm away, and these can be relayed to the user by setting an upper and lower distance threshold.
3) When using the sensor while walking, the step cycle due to leg movement can be tracked consistently, but the cycle poses a problem for readily detecting elevation changes.

4) Moving the sensor higher up the leg gives a larger range for detection but also makes stair detection more difficult by exacerbating the effect of the step cycle.

5.6.2 Suggestions for Further Research

1) Move the sensor above the waist, because there is less vertical movement associated with walking above the legs. This will require changing the sensor's current configuration or finding a sensor with greater range. Moving the sensor to the chest would allow the entire guidance system to be consolidated into one location.
2) If the sensor is to stay on the leg, try a different method from setting a distance threshold for detecting elevation changes. A rolling average might work, but it would require tracking the walk cycle exactly and taking extremely quick measurements.
3) Use a different microcontroller. The Arduino is cheap and easy to operate as far as coding goes, but requiring a laptop for all data collection is slow and inconvenient.
4) Create a single housing unit for the microcontroller and sensor to avoid the hassle of dealing with exposed wires.

6 Conclusions

In conclusion, the objective of this research project was to create a guidance system for the visually impaired. The project was split into three functions: 1) path detection, 2) object detection, and 3) elevation detection. The programed designed to detect the boundaries of pathways is most effective for straight, well defined sidewalks. Relying on Canny edge detection and the Hough transform, the program is able to identify both edges of a sidewalk and plot lines over these edges in real time. The slope of the plotted lines tells the user their proximity to the path's edge. A sound emitted by the headphones from the side they are approaching signals to the user to change course. While the sound is effective in guiding the user, it can be tiresome to listen to. Research into tactile outputs such as vibrations will most likely provide a less irritating response. Other difficulties occur if there is not a clear distinction between the sidewalk and the rest of the ground. Narrow walkways, very large walkways, and curving sidewalks all pose challenges to the current program. Possible resolutions for these problems involve going back to the image processing methods and re-examining their thresholds and parameters. The use of stereoscopic vision to detect edges is another avenue of potential research that would fit in nicely with the next function. The program created to perform object detection is able to correctly identify the location and outer boundaries of potential hazards within the user's path. The program functions best when presented with objects that have sharp, defined edges that are no further than 2 meters from the user. Once an object has been detecting in both images the program can accurately estimate how far the user can travel before having to change their course. Currently it is not possible to detect multiple objects using stereoscopic imaging, because the program has no way of differentiating between one object's horizontal shift from another's. Future research focused on being able to better identify and track multiple objects at one time would greatly alleviate this problem. For elevation detection, the IR sensor used can detect elevation changes when vertical motion due to movement is minimized, but the short range of the sensor makes using the sensor while walking a challenge. Suggestions for future research could involve using a sensor with a larger range or changing the method by which elevation changes are detected. Separately, our devices perform as expected, although some minor alterations might be beneficial. A challenge for future developers is compressing three separate devices and methods into one apparatus. These researchers need to be aware of how to deal with three different sensory inputs from three different sources. Overall, we believe that our individual devices provide a solid foundation for a future project that would more completely assist the visually impaired.

Bibliography

[1] Fact Sheet Blindness and Low Vision. (n.d.). Retrieved November 18, 2017, from https://nfb.org/fact-sheet-blindness-and-low-vision

[2] Austin, K. (2016, September 13). White Cane vs. Guide Dog. Retrieved November 18, 2017, from https://www.second-sense.org/2016/09/white-cane-vs-guide-dog-why-or-why-not/

[3] What Type of Cane Should I Use? (n.d.). Retrieved November 19, 2017, from http://www.visionaware.org/info/essential-skills-2/an-introduction-to-orientation-andmobility-skills/what-type-of-cane-should-i-use/235

[4] Hill, S. (2014, November 19). 5 amazing gadgets that are helping the blind see. Retrieved November 21, 2017, from https://www.digitaltrends.com/mobile/blind-technologies/

[5] MathWorks. (2018). Convert RGB image or colormap to grayscale - MATLAB rgb2gray. Retrieved April 18, 2018, from https://www.mathworks.com/help/matlab/ref/rgb2gray.html

[6] Shapiro, L. G., & Stockman, G. C. (2001). *Computer Vision*. Upper Saddle River: Prentice Hall.

[7] Haran, B. [Computerphile]. (2015, October 2). *How Blurs & Filters Work – Computerphile* [Video File]. Retrieved from https://www.youtube.com/watch?v=C_zFhWdM4ic

[8] MathWorks. (2018). Find edges in intensity image - MATLAB edge. Retrieved April 18, 2018, from https://www.mathworks.com/help/images/ref/edge.html

[9] Haran, B. [Computerphile]. (2015, November 11). *Canny Edge Detector – Computerphile* [Video File]. Retrieved from https://www.youtube.com/watch?v=sRFM5IEqR2w

[10] MathWorks. (2018). Hough transform - MATLAB hough. Retrieved April 18, 2018, from https://www.mathworks.com/help/images/ref/hough.html

[11] Duda, R. O., & Hart, P. E. (1972). Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Communications of the ACM,15*(1), 11-15. doi:10.1145/361237.361242

[12] MathWorks. (2018). Extract line segments based on Hough transform - MATLAB houghlines. Retrieved April 18, 2018, from https://www.mathworks.com/help/images/ref/houghlines.html

[13] MathWorks. (2018). Read images from graphic file - MATLAB imread. Retrieved April 18, 2018, from https://www.mathworks.com/help/matlab/ref/imread.html

[14] MathWorks. (2018). Convert RGB colors to HSV - MATLAB rgb2hsv. Retrieved April 18, 2018, from https://www.mathworks.com/help/matlab/ref/rgb2hsv.html

[15] Mrovlje, J., & Vrancic, D. (2008). Distance measuring based on stereoscopic pictures. *9th International PhD Workshop on Systems and Control: Young Generation Viewpoint.* Retrieved from http://dsc.ijs.si/files/papers/s101 mrovlje.pdf

[16] Adafruit VL53L0X Time of Flight Micro-LIDAR Distance Sensor Breakout. (n.d.). Retrieved November 29, 2017, from https://learn.adafruit.com/adafruit-vl53l0x-micro-lidar-distance-sensor-breakout/wiring-test

[17] Adafruit VL53L0X Time of Flight Distance Sensor. (n.d.). Retrieved November 29, 2017, from https://www.adafruit.com/product/3317

[18] Animation Research: Walk Cycle. (2016, January 08). Retrieved November 29, 2017, from https://cloytoons.wordpress.com/2015/12/01/animation-research-walk-cycle/

Appendix A: Path Detection MATLAB Code

```
clearvars
[chime,Fs] = audioread('electricchime.mp3');
left = chime(:,1)';
right = chime(:,2)';
out_left = [left; zeros(size(left))];
out_right = [zeros(size(right));right];
cam_1 = webcam('Stereo Vision 1');
talk_left = 0;
talk_right = 0;
for k=1:inf;
   posk = 1;
   negk = 1;
   pause(.01);
   cla;
   x1_coord = (length(ans)/4);
   x2_coord = (length(ans)/4*3);
   y_coord = 411;
   cropped = ans(y_coord:end,x1_coord:x2_coord,:);
   crop_gr = (rgb2gray(cropped));
   I = imgaussfilt(crop_gr, 7);
   BW = edge(I,'canny',[.108 .275]);
   [H,T,R] = hough(BW,'Theta', -65:65);
   P = houghpeaks(H,10,'threshold',ceil(0.3*max(H(:))));
   x = T(P(:,2));y = R(P(:,1));
   lines = houghlines(BW, T, R, P, 'FillGap', 1000, 'MinLength',100);
   imshow(ans(y_coord:end,x1_coord:x2_coord,:)), hold on;
   pos = 0;
   neg = 0;
   for k = 1:length(lines);
      if pos == 0 & lines(k).theta > 0;
         xyright = [lines(k).point1; lines(k).point2];
         plot1 = plot(xyright(:,1),xyright(:,2),'LineWidth',2,'Color','green');
         plot(xyright(1,1),xyright(1,2),'x','LineWidth',2, 'Color','yellow');
         plot(xyright(2,1),xyright(2,2),'x','LineWidth',2, 'Color','yellow');
         pos = 1;
         posk = k;
      end
      if neg == 0 & lines(k).theta < 0;
         xyleft = [lines(k).point1; lines(k).point2];
         plot2 = plot(xyleft(:,1),xyleft(:,2),'LineWidth',2,'Color','red');
         plot(xyleft(1,1),xyleft(1,2),'x','LineWidth',2, 'Color','yellow');
         plot(xyleft(2,1),xyleft(2,2),'x','LineWidth',2, 'Color','yellow');
         neg = 1;
         negk = k;
      end
   end

   rise1 = lines(posk).point2(2) - lines(posk).point1(2);
   run1 = lines(posk).point2(1) - lines(posk).point1(1);
   slope1 = rise1/run1;
   rise2 = lines(negk).point2(2) - lines(negk).point1(2);
   run2 = lines(negk).point2(1) - lines(negk).point1(1);
   slope2 = rise2/run2;

   if abs(slope1) > 1.5
      if talk_right == 0
         sound(out_left,Fs);
         talk_right = 1;
      end
   else
      talk_right = 0;
   end
   if abs(slope2) > 1.5
      if talk_left == 0
         sound(out_right,Fs);
         talk_left = 1;
      end
   else
      talk_left = 0;
   end
end
```