CORRIDOR NAVIGATION FOR MONOCULAR VISION MOBILE ROBOTS

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 Electrical Engineering

by

Matthew James Ng

June 2018

COMMITTEE MEMBERSHIP

TITLE:	Corridor Navigation for Monocular Vision Mobile Robots
AUTHOR:	Matthew James Ng
DATE SUBMITTED:	June 2018
COMMITTEE CHAIR:	John Seng, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Jane Zhang, Ph.D. Professor of Electrical Engineering
COMMITTEE MEMBER:	John Oliver, Ph.D. Professor of Electrical Engineering

ABSTRACT

Corridor Navigation for Monocular Vision Mobile Robots

Matthew James Ng

Monocular vision robots use a single camera to process information about its environment. By analyzing this scene, the robot can determine the best navigation direction. Many modern approaches to robot hallway navigation involve using a plethora of sensors to detect certain features in the environment. This can be laser range finders, inertial measurement units, motor encoders, and cameras.

By combining all these sensors, there is unused data which could be useful for navigation. To draw back and develop a baseline approach, this thesis explores the reliability and capability of solely using a camera for navigation. The basic navigation structure begins by taking frames from the camera and breaking them down to find the most prominent lines. The location where these lines intersect determine the forward direction to drive the robot. To improve the accuracy of navigation, algorithm improvements and additional features from the camera frames are used. This includes line intersection weighting to reduce noise from extraneous lines, floor segmentation to improve rotational stability, and person detection.

The addition of all the improvements were compared to the baseline approach. The complete algorithm improved the vanishing point stability by an average of 42.85% on simulated tests and improved real-time robot driving stability by 4%.

ACKNOWLEDGMENTS

Thanks to:

- Dr. John Seng for advising me on this project.
- My family for their support throughout.
- The Cal Poly IEEE Student Branch for allowing me to claim an entire table for the whole year.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

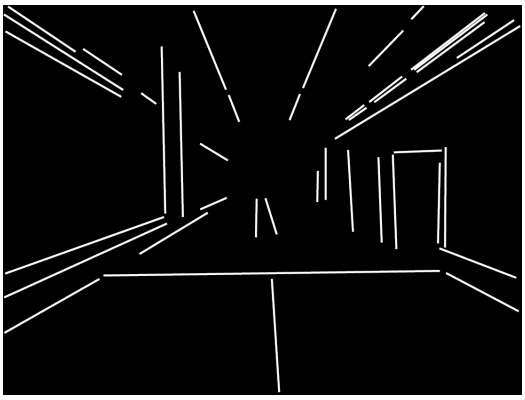
INTRODUCTION

Autonomous hallway navigation has been around for many years and is simple in an ideal environment: no pedestrians, solid colored contrasting walls and floors, and no intersecting corridors. However, navigation becomes increasingly challenging when real world scenarios are encountered such as trash cans, doors, pedestrians, and many more. Many of these challenges can be reduced by integrating multiple sensors such as rangefinders, gyroscopes, accelerometers, compasses, vision sensors, or even providing a map of an area. Compared to other sensors, the camera takes in an enormous amount of data at once. This has an advantage because scenes can change dynamically with various obstacles and the more data the robot receives, the better it can perform. The image captured can be broken down to extract important information. Even though many other sensors are readily available, there is still much to be explored by starting with the simplest method of using only a single camera.

1.1 Statement of Research Question

With this in mind, can a single simple web camera provide enough information about its scene to the robot to determine the best navigation path down a corridor? In particular, what impacts do pedestrians and other obstacles play in hallway navigation? How does the laptop's processor affect the real-time application of the hallway navigation algorithms?

(a) Hallway Image from Camera



(b) Example of a Feature Extracted Hallway Image

Figure 1.1: One Example of Feature Extraction

1.2 Assumptions and Issues

By limiting the robot to indoor corridor navigation, there is an assumption that the most prominent lines will be linear and aimed towards the direction of navigation. This is because most corridors are straight and have differing floor and wall colors. These assumptions are necessary when creating an algorithm to navigate a hallway. The other assumption made is that pedestrians will not be purposefully attempting to interfere with the robot driving. If the robot detects a person on one side of the hallway and attempts to drive around, that person will not move to the other side to get in the way. The pedestrian will follow a predetermined linear path at constant speed and will not alter its path due to robot behavior.

1.3 General Approach

Hallway navigation using a single camera is a problem that can be approached in a variety of ways. One of the most common methods consists of Canny edge detection to extract the most prominent edges in an image followed with a Hough line transform

to turn those edges into lines. After the lines are extracted, intersections between the lines are collected and averaged to calculate the vanishing point which determines the best direction for navigation. This thesis improves on the method above by integrating the line detection algorithm with floor tracking and segmentation.

To run in a realistic scenario, pedestrian detection must be included. A useful person detection algorithm must run in real-time, detect pedestrians with few false positives, and return a bounding box around them. The bounding box will then help determine the location relative to the robot and therefore the navigation path.

The next chapter will cover background information related to this thesis. The following chapters will expand in detail the design of this project, how it is implemented, and the evaluation of the testing.

Chapter 2

BACKGROUND AND RELATED WORK

Because vision based hallway navigation systems are not new, there has been plenty of research in this area. This chapter begins by covering background definitions that will be useful to the overall understanding of this thesis and follows up with various approaches other researchers have developed to solve the challenges of hallway navigation and person detection.

2.1 Background and Definitions

This section includes terms mentioned in the following chapters and the functionality of key algorithms used.

2.1.1 Sensors

For a robot to understand its surroundings, it must receive information from sensors and break that information down into numbers that the robot understands. From this data, the robot can make decisions. Although this thesis is using a monocular camera to determine navigation, two additional sensors will be used to verify how stable the navigation is.

2.1.1.1 Camera

Cameras take in light from a scene and capture that on an image sensor: an array of receivers. The sensor then passes the array of data to the computer. Most cameras take around 24 images per second. The images they capture can be processed and specific features can be extracted from them. By extracting features, only the useful

information is retained, and all the extraneous information is discarded. Such extraneous information can be color, detail, or other irrelevant information. This results in an image that may be more difficult to decipher for the human eye, but assists tremendously in speed and accuracy for the algorithm.

2.1.1.2 Laser Rangefinder

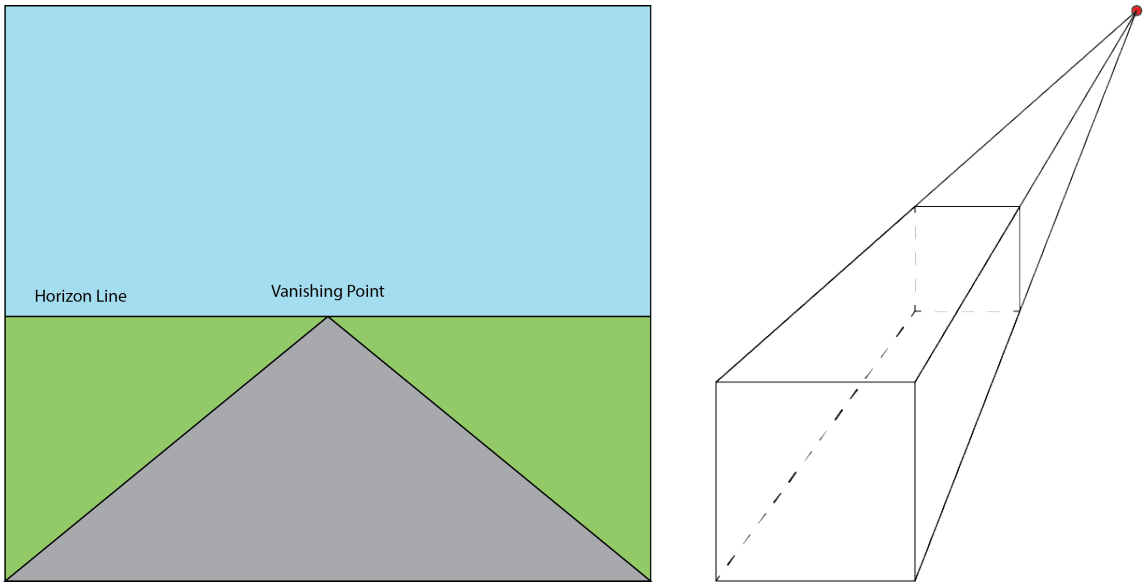
The Laser rangefinder (LIDAR) emits a laser pulse and waits for that laser pulse to be reflected and picked up by the sensor which is located adjacent to the emitter. The distance between the laser and the target can be calculated by timing the duration until the receiver picks up the signal. This sensor has high accuracy when measuring distances and will be used to calculate how centered the robot is while driving in the hallway. The LIDAR used in this thesis is the Lidar-Lite v3 [1].

2.1.1.3 Inertial Measurement Unit

The Inertial Measurment Unit (IMU) has many sensors integrated, but the one used on this robot is the gyroscope. Gyroscopes measure angular velocity by converting vibrations into small capacitance changes that are amplified and picked up by the controller. This sensor in conjunction with the LIDAR will be used to determine how centered the robot driving is. The particular IMU used in this thesis is the BNO055 [2].

2.1.2 Vanishing Point

The vanishing point is the location on the horizon line at which receding parallel lines appear to meet. This can be seen in Figure 2.1a, where the border between the gray road and green grass form two lines lines that meet at the vanishing point. For hallway scenarios, there are many lines formed by the surrounding environment that,



(a) Simple Road Perspective

(b) 3D Perspective

Figure 2.1: Vanishing Point Examples

if extended, will intersect at the vanishing point. In Figure 2.1b, this shows a 3D perspective of how a hallway can project its lines towards the vanishing point.

2.1.3 Canny Edge Detection

The Canny Edge algorithm is a first order derivative function which takes in a grayscale image and calculates the gradient of that image in the X and Y directions. This results in a gradient magnitude and gradient angle image. The gradient magnitude shows spikes in the areas of sharp intensity change such as a change from white to black. The gradient angle points in the direction of the largest intensity pixel values. Non-maximum suppression (NMS) uses this gradient angle to determine local maximums and reduce all other pixels. NMS returns single pixel wide lines that are thresholded to create a binary image and isolate the most prominent edges. The OpenCV Python function can be seen in code 2.1(line 5) where it takes in a grayscale image (line 4).

2.1.4 Hough Line Transform

The Hough Line transform algorithm takes a binary image in which the edges have been detected. This is commonly the result from the Canny Edge Detector. The result of this algorithm is a list of two points specifying the line extracted.

There are two methods of calculating the lines using this transform. The one used involves converting the image space of $y = m * x + b$ into a space in terms of ρ and θ. All of the points from the Canny Edge algorithm goes through this process and results in multiple sinusoidal curves. From these curves, the intersection points in the (ρ, θ) space correspond to lines in the original $y = m * x + b$. This method results in few false line positives. The Hough Line transform takes in a couple parameters: a threshold, a minimum line length, and a maximum line gap. The threshold determines the number of intersections in the (ρ, θ) space it takes for a point to be qualified as a line. Too low of a threshold and too many lines are extracted resulting in noise creating lines. Too high of a threshold will result in too few lines. The minimum line length requires a certain length to prevent short lines from affecting the algorithm as much. Finally, the last parameter is maximum line gap. If there are discontinuities between lines, this parameter sets a limit to how wide a gap to determine if it is a line. This also helps account for noise. The OpenCV Python function can be seen in code 2.1(line 6) where it takes in a Canny edge image (line 5).

```
1 cap = cv2.VideoCapture(0) #open the camera
2 while cap.isOpened():
3     ret, image = cap.read() #capture an image
4     gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
5     edge = cv2.Canny(gray, ...) #Canny Edge Detection
6     lines = cv2.HoughLinesP(edge, ...) #Hough Line Transform
7 cap.release()
```

Listing 2.1: Line Extraction Sample Code

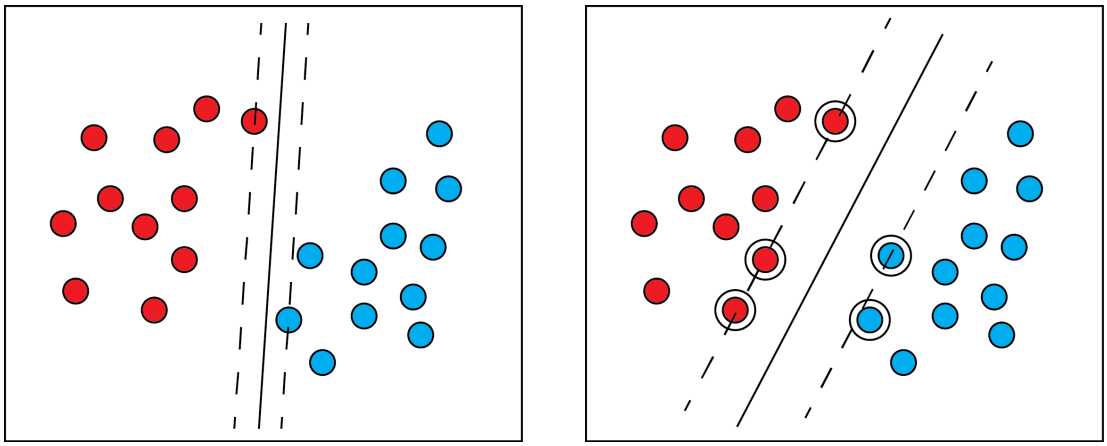
2.1.5 K-Means Segmentation

K-Means segmentation is the K-Means clustering algorithm applied to pixel values. It can be used to split an image into multiple regions based on color values. It does this by observing the histogram extracted from all the pixels in the image. The next step is to then randomly place centroids on the histogram. With these centroids, find the values that are nearest to the centroids. Once the values are assigned to a centroid, update each centroid's mean using the values in each cluster. Repeat this procedure until the means no longer change or after a set number of iterations.

Although the K-Means segmentation algorithm can be used to segment colors in an image, there are a few drawbacks. One drawback is it requires the selection of the number of regions (k) beforehand. Too many or too few regions will cause incorrect segmentation. Another drawback is the random initialization. This can result in incorrect segmentation, but can be alleviated with multiple passes of the algorithm on the image. Other segmentation algorithms that could be used include expectation-maximization or the mean shift algorithm.

2.1.6 Linear Support Vector Machine

Linear SVM is a supervised machine learning algorithm. It finds the best linear decision boundary between sets of data. The best decision boundary according to SVMs should be as far away from the data of both classes. This can be seen in 2.2, where it finds the boundary by finding the maximum margin separating the two classes. The circled points in 2.2b are the support vectors which are the data points that lie closest to the boundary and are the most difficult to classify.



(a) Linear Decision Boundary

(b) Linear Decision Boundary Determined with SVM

Figure 2.2: Linear Decision Boundary Separating Two Classes

2.1.7 Histogram of Oriented Gradients for Person Detection

The Histogram of Oriented Gradients (HOG) breaks down an image by finding the gradient magnitude and angle of the image. This is similar to Canny Edge detection. Once this is extracted, the feature space is then grouped together by clustering pixels. This reduces noise and makes the classification quicker. Classification is done using a Linear SVM described in section 2.1.6. Once the SVM is trained, the HOG person detection algorithm can be run by sliding a window that compares a region of interest in an image to the SVM trained person detector.

The HOG algorithm that runs the person detection takes in a couple parameters, most importantly, image, window stride, and scale. The location of the image that HOG will be parsing through is necessary. The next two parameters, window stride and scale, directly affect the detection rate and the speed. Window stride is how many pixels to slide the window by when comparing the SVM trained person detector to the actual image. The scale parameter is how to shrink the image before running the detection sliding window again. This is useful because in the case that a person is

too large compared to the person detector, when the image is scaled, it will be more likely the person will be detected. For both of these parameters, the smaller they are the more likely to detect a person. However, the drawback is a significant decrease in speed and increase in false positive rate. The OpenCV functions can be seen in the code snippet 2.2.

```
cap = cv2.VideoCapture(0) #open the camera
hog = cv2.HOGDescriptor() #initialize the HOG descriptor
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
while(cap.isOpened()):
    ret, image = cap.read() #capture an image
  (rects, weights) = hog.detectMultiScale(image)
  #Bounding boxes are "rects"
  #Confidence in each box is "weights"
cap.release()
```

Listing 2.2: HOG Person Detection Code

2.1.8 Optical Flow

The Optical flow algorithm is used to track motion between multiple consecutive frames. This algorithm assumes that the objects in each frame are not moving very quickly and there are no sudden changes of intensity. The two most common methods used are the Lucas-Kanade (LK) method by Lucas and Kanade (1981) [18] and the Dense method developed by Farneback (2003) [10]. The LK method uses corner detection to track movement of specific points in the video by finding the nearest corner detected in the current frame and comparing it to the previous image. This can result in error if a corner point is obstructed. The corner point might be "attached" to that obstruction, whetether that is a person or another obstacle. The Farneback method calculates the optical flow for all the points in the frame. This is slower than the LK method, however it is more reliable for person detection since this method

does a quick estimation of all points. This method is not perfect and the drawback can be seen in Figure 2.3. All three of these images were taken by driving straight down the hallwaw and the colors represent the motion angles determined. The left image is the Farneback result without any preprocessing, the middle image has a blur of 10 pixels, and the right image has a blur of 20. Even though the rightmost image has the least amount of noise, it is very blurred at this point.

This algorithm is run by passing in two sequential image frames. Besides this, the two most important parameters in this algorithm are window size and neighborhood. Window size averages the values and will make the algorithm more noise resistant, but increases the blurriness. This was seen in Figure 2.3. The neighborhood parameter gives the region that each pixel could move to. The larger this parameter, the more robust the algorithm is because it results in a more blurred motion field. Sample initiation code can be seen in the code snippet 2.3.

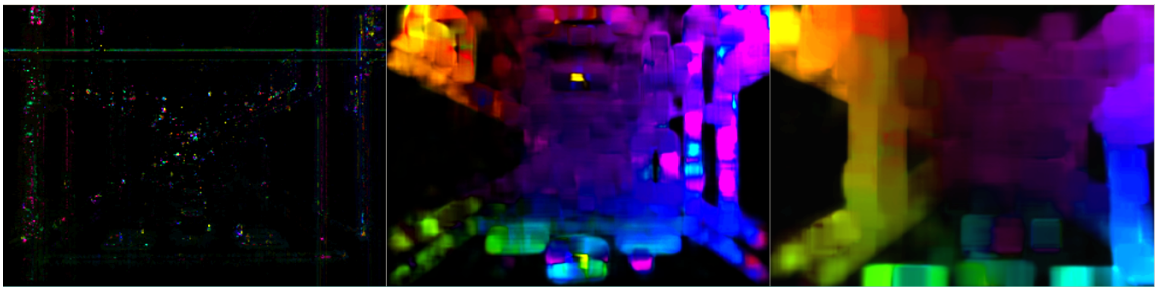


Figure 2.3: Hallway Optical Flow Comparison

```
cap = cv2.VideoCapture(0) #open the camera
ret, frame1 = cap.read()
prvs = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)
while(cap.isOpened()):
    ret, image = cap.read() #capture an image
    nxt = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    flow = cv2.calcOpticalFlowFarneback(prvs, nxt)
```

```
8       #X direction flow can be extracted with flow[..., 0]
9       #Y direction flow can be extracted with flow[..., 0]
10      prvs = nxt #set next frame
11  cap.release()
```

Listing 2.3: Optical Flow Code

2.2 Related Work

As mentioned earlier, there has been much work done with vision based robot navigation. The following sections will highlight the papers which are most relevant to this thesis as well as other approaches that researches have taken.

2.2.1 Corridor Navigation

The basic premise of corridor navigation with monocular vision is to determine the orientation and direction for the robot by analyzing the scene. This can be done using computer vision and machine learning algorithms.

2.2.1.1 Monocular Vision Image Processing Methods

The monocular vision approach uses Canny Edge detection and Hough line transform to determine the forward direction. Many of these papers cover empty hallways with no obstacles.

In the work done by Zhou, Chen, Wu, and Yu (2011) [23], image captured are reduced in size from 320x240 to 80x60 to speed up the processes and to filter out noise. Once reduced, the image is then grayscaled and passed through the Canny Edge detector and Hough line transform to extract lines from the image. These lines have multiple intersections. The final vanishing point is calculated by obtaining the

average vanishing point and calculating the Euclidean distance between that vanishing point and nearby pixels. Pixels outside of a certain boundary are eliminated. This process is repeated several times with smaller threshold boundaries and updating the mean with all the points in the estimated circle.

The approach to monocular vision hallway navigation done by Saitoh, Tada, and Konishi (2009) [21] aims the camera at the floor. From here, the main method of navigation is to use the two floor lines to determine the forward direction. This floor isolation is particularly effective since there is a much darker baseboard compared to the wall and floor.

The work by Chang, Siagian, and Itti (2012) [8] merges path segmentation with the Hough Line transform. This paper shows how path segmentation can fail by segmenting the surroundings instead of just the path. When the Hough line transform is not aligned with the road it can also fail. But with the combination of the two, the driving stability can be improved.

2.2.1.2 Machine Learning Navigation

Because this problem space is broad, many alternative approaches to hallway navigation do not involve extracting features. The other commonly used method for navigation involves using neural networks trained on hallway images.

An example of this was done by Konam (2017) [15] where a monocular vision quadcopter is used as the robot. This robot is connected to a host machine over WiFi and all the processing is done remotely. During the training, a human pilot controls the quadcopter using the six commands: forward, stop, turn left, turn right, spin left, spin right. The images are then set as the input to the neural network and the six commands set as the outputs.

Another unmanned aerial vehicle (UAV) which uses neural network for indoor navigation was done by Kim and Chan (2015) [14]. Similar to the previous UAV, the processing is completed on a host system. The difference in this method is using neural networks to estimate depth in monocular images. Once the depth is determined, the quadcopter can determine the best method of travel where there is more space available.

In the work by Souza, Pessin, Shinzato, Osorio, and Wolf (2011) [22], an artificial neural network analyzes an image to determine the regions in the image classified as a road. Although this is an outside scenario, this road detection can be modified for hallway floor training. Once trained, the classification is passed to 7 different road templates. This breaks down into one for straight forward, two for non-centered straight roads, two for soft turns, and two for hard turns.

2.2.2 Person Detection

Hallway navigation is insufficient if there is no way to identify objects or pedestrians. This thesis uses a combination of various implementations to help identify pedestrians in the robot's path. Similarly to hallway navigation, there are different approaches. Some methods accomplish this by thresholding the image to isolate objects. Other methods attempt to identify and classify objects in the scene.

The You Only Look Once (YOLO) person detection algorithm was designed by Redmon, Divvala, Girshick, and Farhadi (2016) [20]. It used a convolutional neural network used to run real time person detection. The hardware used in this paper was a Titan X GPU allowing the algorithm to run at 45 to 150 frames per second. It is able to predict all bounding boxes across all classes in the training library simultaneously. It does this by overlaying a grid pattern on the image, where each cell is responsible for being the center of an expected object.

The paper by Dalal and Triggs (2005) [9] compares how different person detection algorithms perform versus HOG person detection. Not only did this paper show that HOG out-performs wavelets and other methods, it also mentions that any degree of smoothing before calculating the gradients damages the HOG results by losing the the abrupt edges at fine scales. Even when pyramid scaling the image down to detect different sized people, maintaining the finest available scale will help in overall person detection.

As previously mentioned, Farneback (2003) [10] developed the optical flow algorithm. It was designed for motion estimation when a camera was mounted on a helicopter and was severely affected from the high frequency vibrations. By using the method of polynomial expansion to estimate translation, it is possible to estimate the motion field and compensate for the background motion. This is done for each individual pixel in an image frame and when subsampled can provide a robust solution for motion estimation.

In the work described by Ramzan, Fatima, Shahid, Ziauddin, and Safi (2016) [19], optical flow and HOG person detection are combined. The first step is to calculate the optical flow motion vectors across the entire image. Once this is done, regions of motion become regions of interest. Bounding boxes are formed around these regions of interest and subsequently, HOG features are extracted. These features are given to an SVM classifier to discriminate human from non-human objects. In this paper, it is mentioned that this methodology could not work for a moving robot since a moving vehicle provides a continually dynamic background.

Gavrila and Munder (2006) [11] developed a multi-class classifier for motion detection. One for if the pedestrian is facing forward or away, one for facing left, and one for facing right. This paper however does not use tracking to find a person like in the paper by Ramzan et al. (2016) [19]. Instead, the purpose of tracking is to "overcome

gaps in detection". Having a consistent tracking algorithm helps the computer vision system recognize that once someone has been detected, the next frame should have that person there. This increases the reliability of the driving system.

The work done by Zhou et al. (2011) [23] also attempts obstacle avoidance. Obstacle detection is introduced and distance is calculated by selecting a threshold for the image and finding differences between the floor and objects. This method does not use any classifiers. This is simple to do for this test case because the floor is a different solid color than than the walls. On top of this, the baseboard contrasts sharply with the wall and floor. The object to be detected can be isolated from the background simply based on color. Once the object is detected, a bounding box is drawn around the object. The distance can be estimated by combining prior knowledge of the width of the hallway with the base of the detected object.

Chapter 3

SYSTEM DESIGN

The first step in creating this hallway navigating robot is to put together the basic hardware and software necessary to drive. The hardware covers the physical components, including the main processor, a laptop. The software running on this laptop will be processing the information the camera captures and sending data to the Arduino which controls the motors.

3.1 Hardware Design

The robot used in this thesis is made of a Parallax MadeUSA chassis kit, a Lenovo Ideapad Z580, and two Arduino Megas. The chassis and laptop are pictured in Figure 3.1. The connections between the electronic components can be seen in Figure 3.2. The Lenovo Ideapad Z580, which contains a 2.5GHz Intel i5 processor takes images from the built in webcam at 720x480 resolution, processes those images, determines the direction to drive, and sends a message to an Arduino controlling the motors. There are two Arduino Mega boards attached to the laptop. One Arduino Mega is responsible for the robot drive system described in section 3.1.1. The other is only used for testing. The red dashed lines denote the testing system described in section 3.1.2. The additional sensors on the stability system are not used to drive, but for extracting data from its surroundings to verify the success of the robot's hallway navigation.

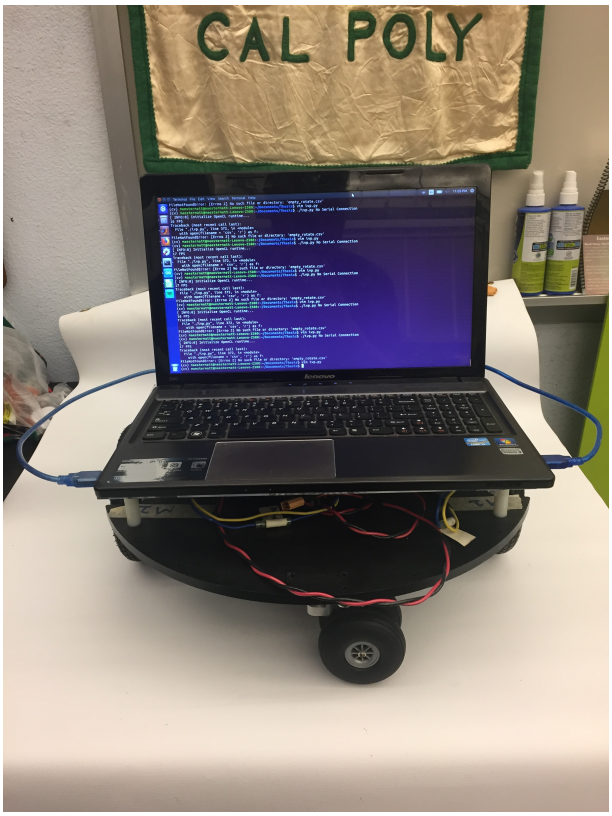


Figure 3.1: Robot used for testing

3.1.1 Robot Drive System

For this system, a Pololu VNH5019 Motor Shield is used. Previously, two L298N H-Bridges were used to drive the motors. Because each motor draws over one amperage of current, this exceeded the H-Bridge motor driver limit causing them to burn out. In Figure 3.3, the blue and yellow wires go to both terminals of the DC motors and the red and black wires go to the battery. There are no other peripherals attached to this main driving Arduino as seen in Figure 3.2.

3.1.2 Stability Measurement System

This system is denoted in Figure 3.2 with the red dashed line. Figure 3.4 shows the LIDAR and IMU are the two sensors that make up this system. These sensors are only used for testing to extract environmental information such as distance from

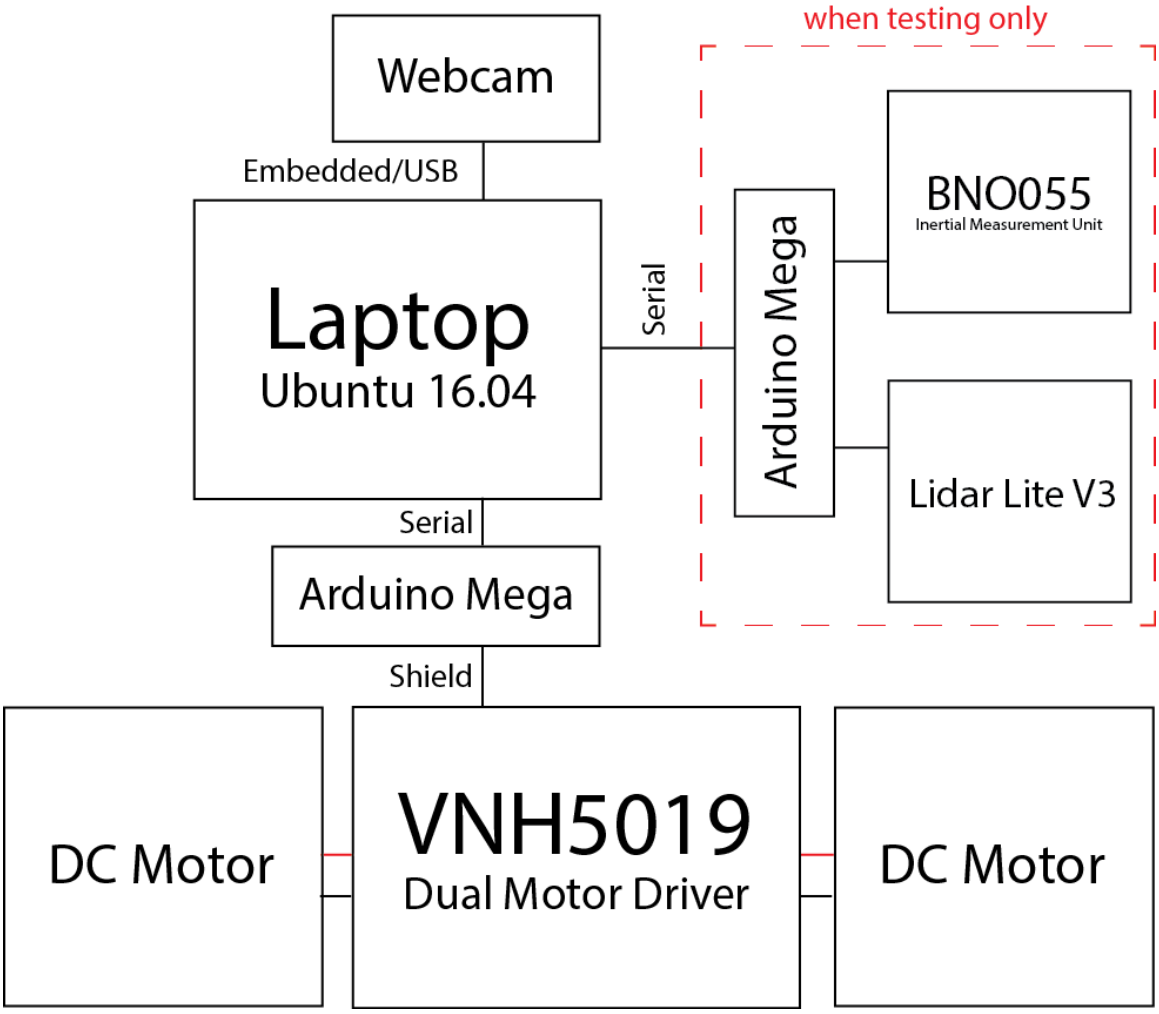


Figure 3.2: Hardware Structure

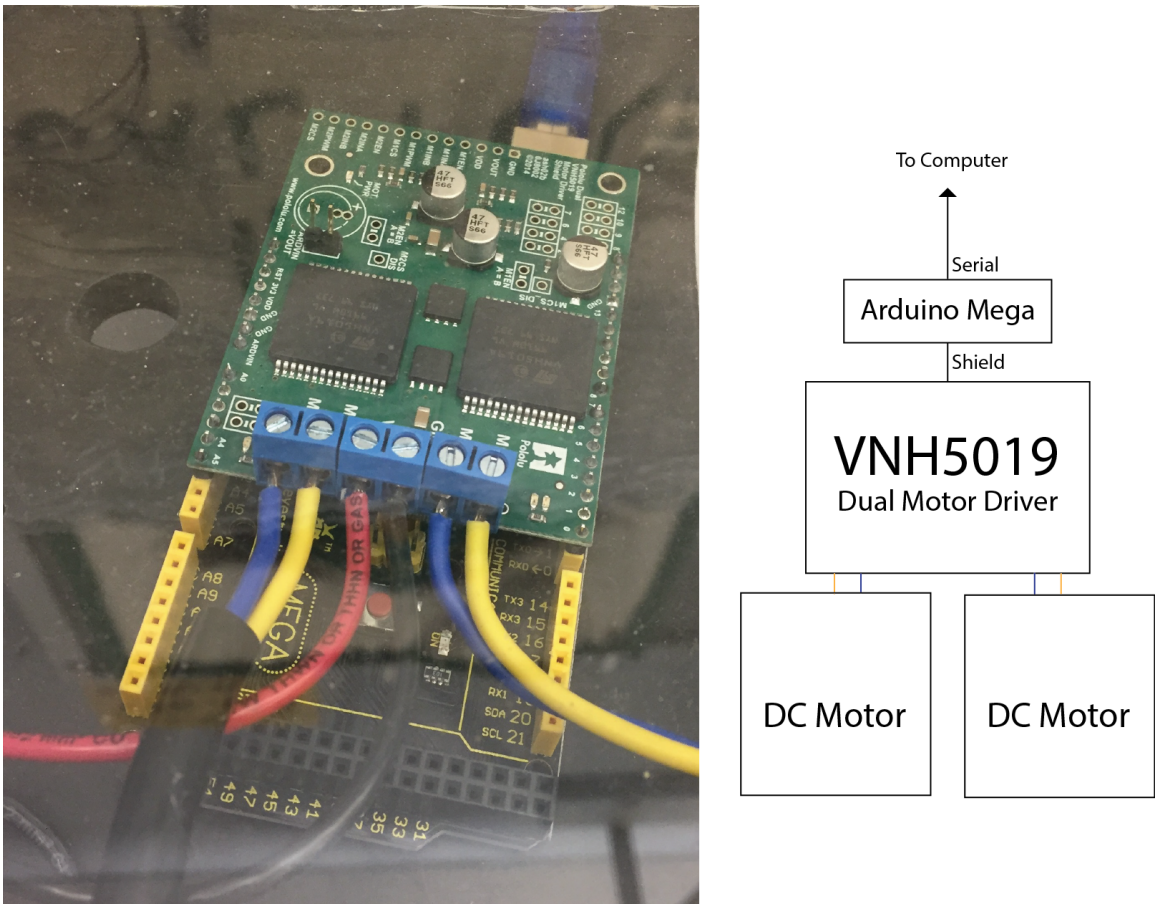


Figure 3.3: Motor Drive System for Robot

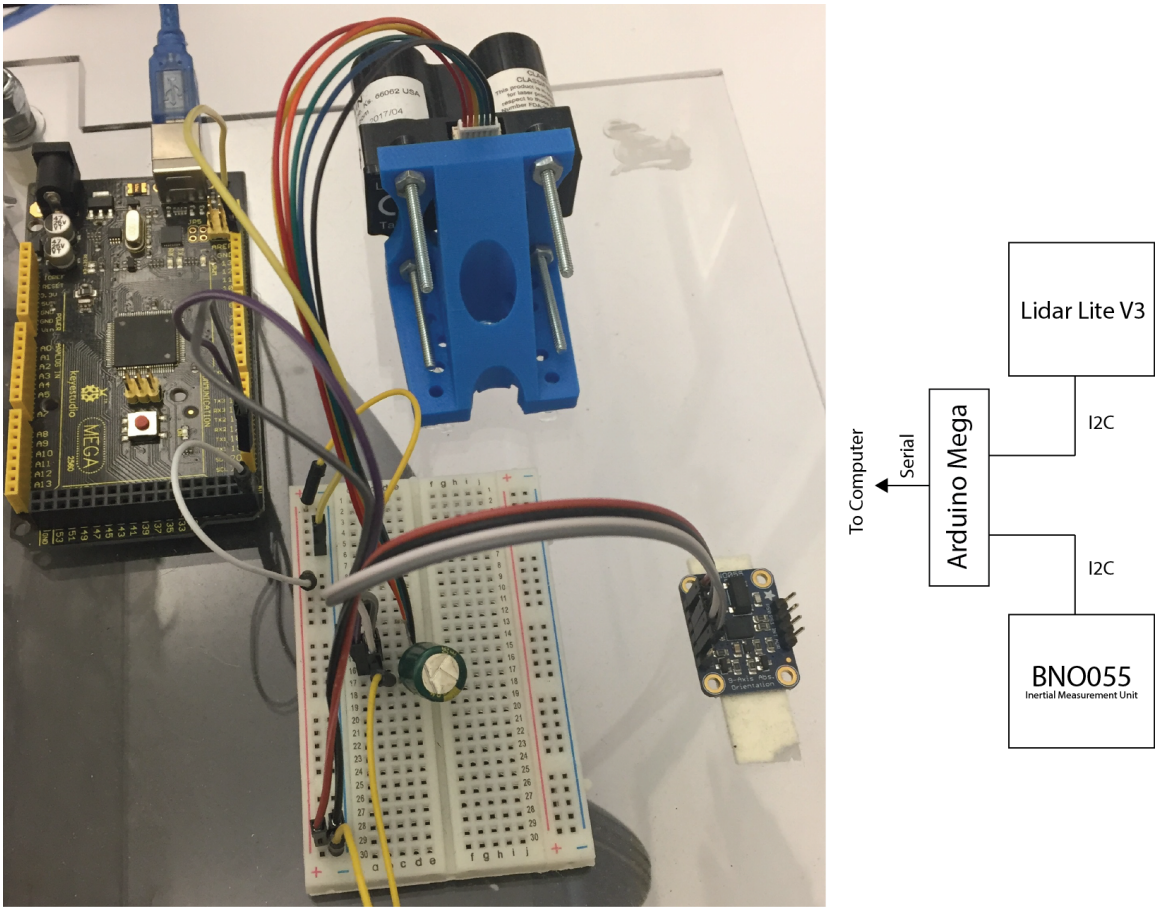


Figure 3.4: Testing Hardware System

one side of the hallway and the angle the robot is facing. Both of these values can determine the actual distance from the side of a hallway. This calculation can be visualized in Figure 3.5.

3.2 Software Design

The operating system run on the laptop is Ubuntu version 16.04. This operating system was chosen because of the simplicity of integrating everything with Python 3. The Arduino Mega communication can be set up quickly using PySerial and interfacing with the port /dev/tty/ACM0 or ACM1 depending on the drive system or testing system respectively.

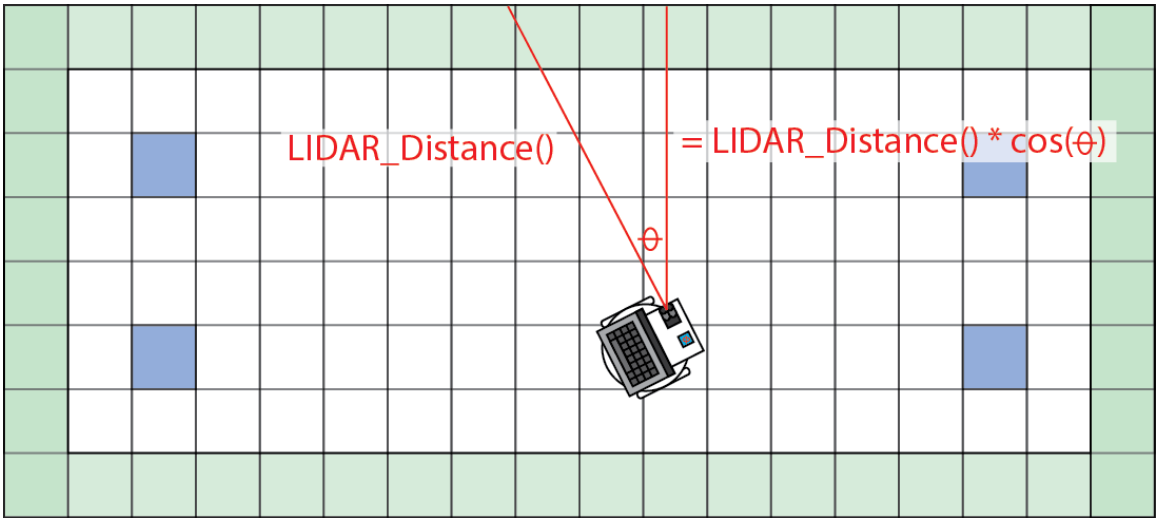


Figure 3.5: Distance Calculation using LIDAR and IMU Data

3.2.1 Libraries

The vision libraries implemented in this thesis were taken from the OpenCV library [13]. Many of the transforms and algorithms were developed from that open source library.

The hardware libraries were integrated with the microcontroller which is the core between the software and hardware system. Two Arduino Megas are used because they are quick to interface with and can easily be adapted. The role of the first Arduino seen in Figure 3.3 is to control the motors. This Arduino receives one byte over serial and then splits this byte into 2 messages, 4 bits per motor. The messages are sent to the motor shield library [5] which outputs the specified voltage to control the motor. Because the robot is not moving quickly, having 16 speed levels is sufficient for driving. The Arduino controlling the measurement sensors, as seen in Figure 3.4, receives and stores I2C communication from the IMU and the LIDAR using the Adafruit BNO055 library [3] and the Garmin LidarLite v3 library [6].

Chapter 4

APPROACHES AND IMPROVEMENTS

The main method for navigation is determining the vanishing point using Canny Edge and Hough Line transforms. These algorithms can be extremely noisy, especially when pedestrians are involved. To improve the stability and reliability, modifications can be made to the base navigation algorithm. This chapter will begin by covering the baseline approach used to navigate a hallway. It will then be followed by the various approaches taken to improve hallway navigation. This includes improvements to the vanishing point calculation and integration of person detection.

4.1 Baseline Approach

The baseline approach is the bare minimum to process captured frames from the camera and determine which direction to travel based on the vanishing point. The next step after determining the location of the "forward" direction is to update the values sent to the motors and control them to ensure smooth driving.

The vanishing point calculation is the basic navigation method. The overview of this approach is seen in Figure 4.1. This algorithm runs quickly and can be used alone

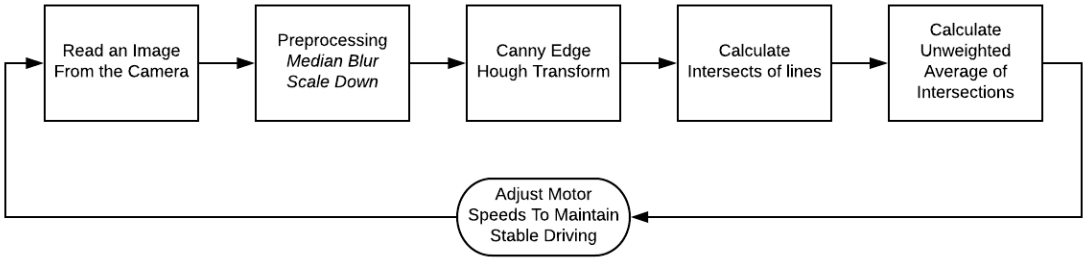


Figure 4.1: Baseline Vanishing Point Algorithm

Figure 4.2: Empty Hallway

to navigate a hallway. However, this algorithm is susceptible to noise, and there are many modifications to improve this algorithm or even supplement it.

The first step in the vanishing point calculation begins with a capture of an empty hallway shown in Figure 4.2. This image is then scaled down to increase processing speed and accuracy since the amount of detail to parse through will be reduced and the extra detail could add noise to the line calculations.

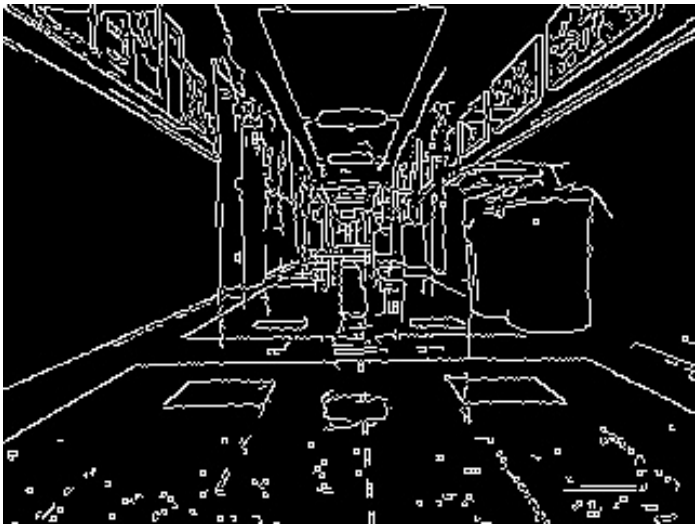


Figure 4.3: Canny Edge Detected Empty Hallway

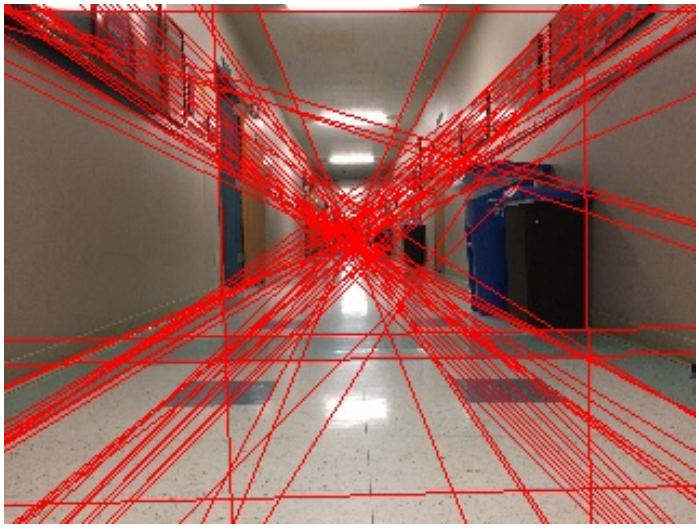
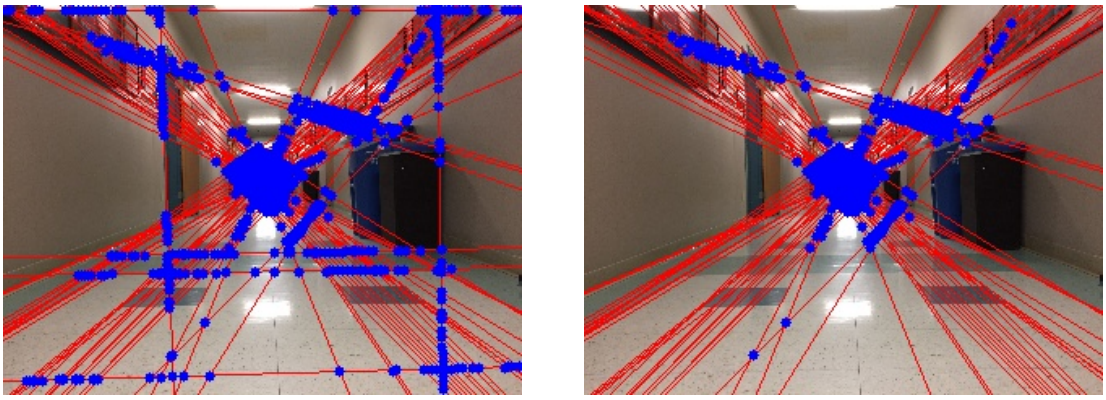


Figure 4.4: Hough Line Transform of Canny Edge Detected Image

The Canny Edge algorithm described in section 2.1.3 is then run on the image. This function returns the binary image seen in Figure 4.3 with all the edges represented as white pixels. This algorithm finds lines based on strong intensity or color change in an image. Referring back to the image, the desired lines from the floor and wall transition are extracted, but noisy edges appear from the images on the walls of the hallway.

This binary image is then passed to the Hough line transform described in section 2.1.4 which calculates the most likely lines from the Canny Edge image. The result is overlaid in red on the hallway image as seen in Figure 4.4. There are a few observations that can be made of this figure. The vertical and horizontal lines do not help with the vanishing point detection. They can be formed from the floor tiles, trash cans, doors, and many other features. Another observation is that there are a few lines originating from similar regions of the image. This results in many random intersections that do not provide usefulness to the vanishing point detection. The most important lines are those that intersect with vastly different slopes.



(a) Removing Similar Line Intersections (b) Removing Horizontal and Vertical Lines

Figure 4.5: Hough Line Transform Intersections

Once these lines are calculated, intersections are found. To resolve the issue that arises from multiple lines originating from the same location, if the slopes of the lines are similar, the resulting intersections are ignored. The other issue comes from horizontal and vertical lines as seen in Figure 4.5a. By removing these horizontal and vertical lines, the image is much less noisy as a result in Figure 4.5b.

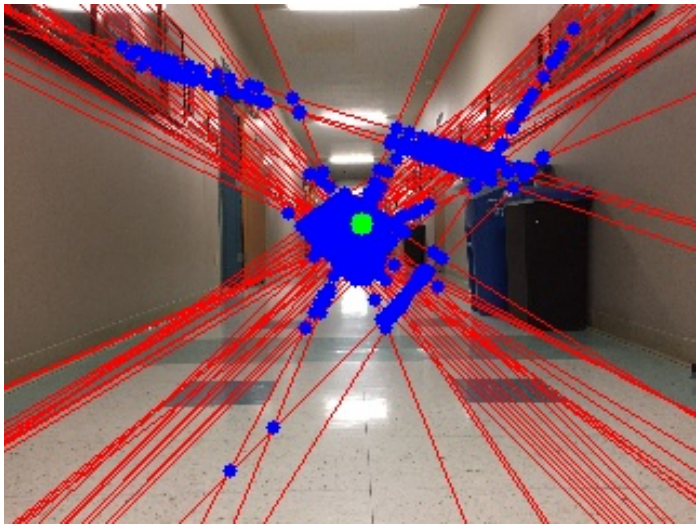


Figure 4.6: Intersection Centerpoint Averaged from all Intersections

The remaining intersections after filtering can be averaged by summing together the x and y components separately and dividing each sum by the total number of

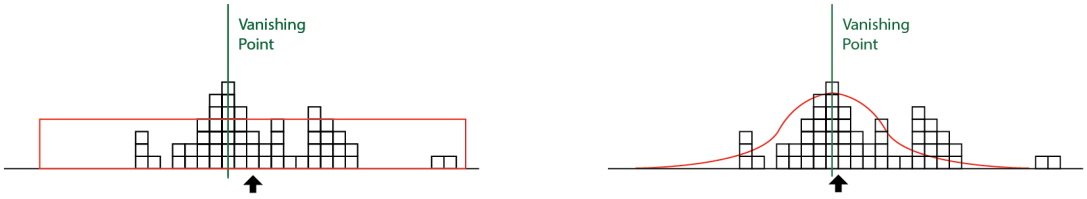
intersections. In Figure 4.6, the center point is visualized in green and it appears to be slightly shifted towards the top right. This is due to some noisy lines. This problem is approached in the remaining sections of this chapter. The steps mentioned above form the baseline approach that will be used for the remainder of this thesis. Although it is not simply running Canny Edge and Hough Line transform, those two algorithms are the core components of determing the vanishing point.

4.2 Vanishing Point Calculation Adjustments

A problem occurs when all the intersections are weighted the same towards calculating the vanishing point described in section Figure 4.1. This method is easily susceptible to noise since a single line can significantly pull the vanishing point to one side or the other. Two methods were applied to attempt and fix this issue. Both methods rely on previous knowledge.

4.2.1 Gaussian Weighting

The problem where a single line shifts the vanishing point can be reduced by weighting the intersection points by assuming a Gaussian distribution of intersection points. Unlike the average sum method, the return value from Gaussian weighting is not the vanishing point. Instead it returns a change in position from the previous vanishing point. This previous vanishing point is set as the mean of the Gaussian function. The variance for the Gaussian function is set to be the width of the image frame since it is possible for intersections to end up anywhere. A 1D example can be seen in Figure 4.7. This data is pulled from the intersections of Figure 4.13.The "vanishing point" marked in green on the graph is the actual vanishing point determined visually. In 4.7a, this is using the basic averaging method. Since it weights all the values equally, the line created by the pedestrian skews the result towards the right. The Gaussian



(a) Baseline Averaging Method (b) Using Gaussian Weighting

Figure 4.7: Gaussian Weighting Example Using Results from Figure 4.13a

weighting almost negates the spike and results in a more accurate response of data. This shows that by implementing Gaussian weighting, the image is less susceptible to noise.

This method is not perfect because if the robot is turning and the vanishing point does not change quick enough, the intersections will point at a new location too far from the mean of the old vanishing point and result in the change indicated by the Gaussian weighting close to zero. This results in the vanishing point getting "stuck" if the robot turns too sharply.

4.2.2 Image Splitting for Floor Isolation

The purpose of image splitting is to simplify the image by only viewing the lower half. By doing this, the amount of information the algorithm must go through to parse data is decreased. In addition, the lower half of hallway images contain less distracting information. Typically the lower half of hallways only contain the walls and floor. Figure 4.8 shows that the lower half of the hallway does not contain any display cases, signs, pictures, and flyers. These items cause noisy edges when run through the Canny Edge detector in Figure 4.3. The following processes build off of image splitting to attempt and segment the floor.

Figure 4.8: The Lower Half of the Electrical Engineering Hallway

4.2.2.1 Floor Lines

Being able to find the two lines where the wall meets the floor not only assists in vanishing point calculations, but it can help the robot to determine its location respective to both walls. As mentioned before, the paper written by Saitoh et al. (2009)[21] achieves this by aiming the camera towards the floor and only looking for the floor lines. However this method is not as effective when pedestrians are introduced. Instead, to develop floor line tracking, all the lines that intersect the vanishing point are preserved in the bottom half of the image. By eliminating stray lines, this helps narrow the search for the two main floor lines. To determine which lines correspond to the floor, this process finds the nearest slope to the current one. The premise of this design is that the floor lines are not changing rapidly. If the robot is driving down a hallway, it should maintain two floor lines in approximately the same location. Even if the robot rotates, the floor lines should be the same angle but translated towards the new vanishing point. To maintain continuity of floor lines in frames, if a line is not detected for a frame the last detected line is held. This creates a more stable and consistent floor detection algorithm.

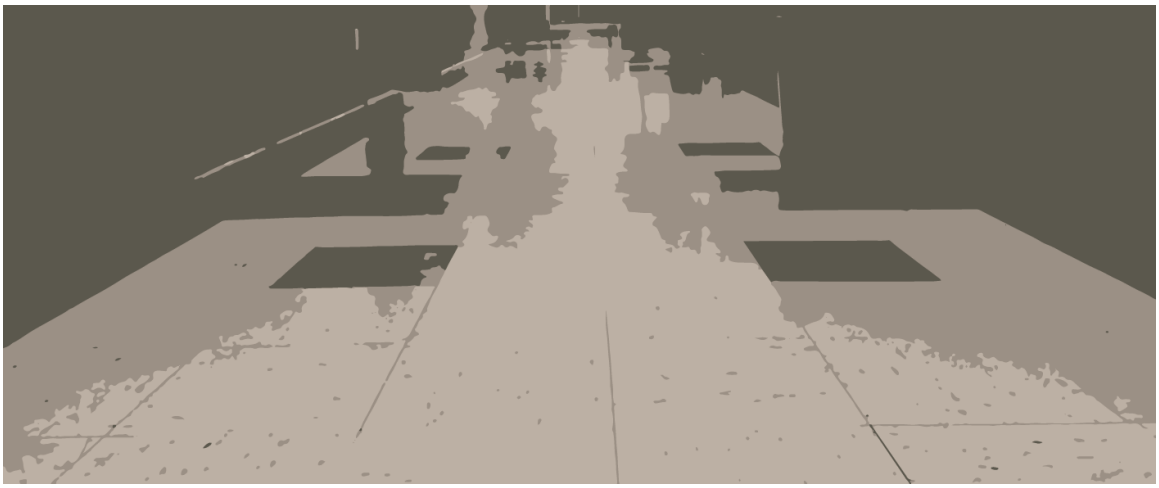


Figure 4.9: K-Means Segmentation of the EE hallway

4.2.2.2 K-Means

The Hough Line vanishing point calculation does poorly when the robot rotates. This can be from simple navigation or obstacle avoidance. Regardless, the camera frame is subject to motion blur and makes it increasingly difficult for the Canny Edge detector to capture the hallway edge lines properly and find the vanishing point. The method introduced by Chang et al. (2012) [8] shows how merging K-Means can help to determine the center point for navigation. This is because K-Means requires blurring of the image to succeed and Canny Edge and Hough Line transforms fail in this aspect.

Using K-Means to determine the vanishing point requires knowledge of the previous vanishing point. This is because the segmentation result splits the image into K number of regions which results in Figure 4.9 and the algorithm must decide which segmented region is the correct one. It uses the previous vanishing point to find the region which contains the point $(x, max(y)$. Once the region shown in Figure 4.10 is extracted, the peak of this region is found. The peak's x location is the vanishing point determined by the K-Means algorithm shown in 1.

Figure 4.10: Visualization of the Center Segment masked from the image

This method however is not perfect. Because the K-Means segmentation algorithm initializes centroids randomly, there is no guarantee that the algorithm will segment properly and therefore accidentally segment the ground and walls together. This will result in an impulse-like change in the K-means algorithm. Understanding this drawback however can be adapted to improve the reliability of this algorithm.

Algorithm 1: K-Means Algorithm

Result: Calculate Vanishing Point Using K-Means Segmentation

Capture a frame from the camera;

Isolate the lower half of the frame;

Run large median blur on the half frame;

Run K-Means Segmentation on the half frame to split into K regions;

Isolate the region that contains the pixel located at (x = Previous Vanishing Point x, y = Image Height);

Extract contours from selected region;

Calculate the peak of the contour region;

Return the X value of the peak;

4.3 Software Flow of Complete Algorithm

The adjustment methods described in the previous section help stabilize the vanishing point. These algorithms are less affected by noise compared to the average sum vanishing point calculation described in the baseline approach. However, because they rely on past information, there is an issue when the actual vanishing point changes drastically. This is when the average sum method is the best to find the new vanishing point since it relies only on current values. Because all these applications provide improvements, the best software approach would be to merge all of the algorithms together.

Combining both algorithms should perform the best and tackle many different scenarios. The vanishing point calculation software flow can be seen in Figure 4.11. This software flow is also described in the algorithm 2. It is much more complex now than the original simple vanishing point calculation in Figure 4.1. By adding additional features, the processing time will increase. In the next chapter, the performance will be compared across multiple test simulations and test runs.

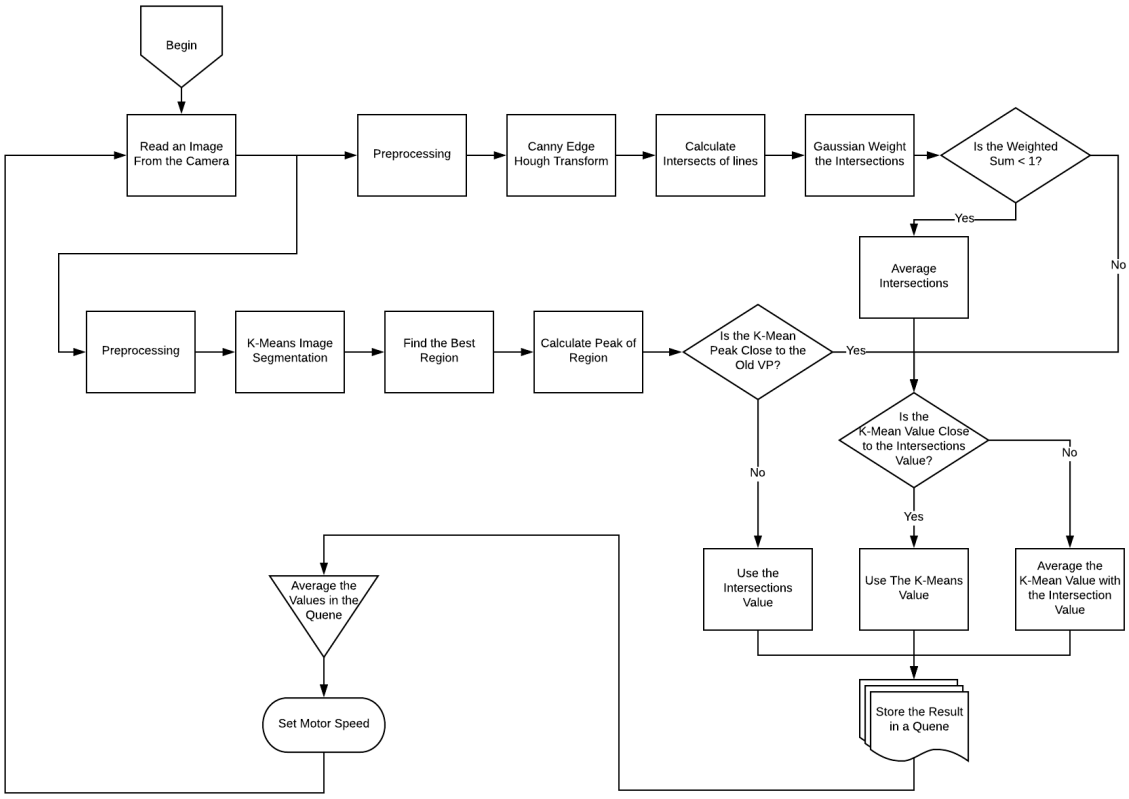


Figure 4.11: Vanishing Point Software Flow

Algorithm 2: Software Flow

Result: Determine Vanishing Point

Capture a frame from the camera and preprocess to remove noise;

Run Canny Edge and Hough Line transform to extract lines from the frame;

Extract floor lines as mentioned in section 4.2.2.1;

Calculate all the intersections from the lines;

Gaussian weight all the intersections as mentioned in section 4.2.1;

if *If Sum of Gaussian Intersections <1* **then**

    Replace Gaussian calculated center with average sum;

end

Calculate the K-Mean vanishing point value using algorithm 1;

if *The K-Mean Value is close to the previous Vanishing Point* **then**

    if *The K-Mean Value is close to the Intersection Value* **then**
        Use the K-Mean value

    else

        Average the K-Mean value with the intersection value;

    end

else
    Use the intersection value

end

Store the calculated vanishing point into a queue;

Average the values in the queue to provide a moving average;

4.4 Person Detection

Hallway navigation is incomplete without person detection. Pedestrians can cause noise in an image by creating extraneous lines picked up by the Hough Line transform. They can also get in the way of the robot's navigation path.

4.4.1 Person Detection Algorithm

To solve this issue, a bounding box needs to be detected around a pedestrian in which false positives cannot occur. Previously built person detection algorithms were implemented to see if a bounding box can be created around the person. These person detection algorithms output the bounding box as well as the confidence that the object detected was a person. Two object detection algorithms were tested to see which algorithm ran quicker and was more accurate.

4.4.1.1 You Only Look Once Object Detection

Since the laptop used in this thesis does not include an NVIDIA GPU, the person detection algorithms were slow. This affected certain algorithms more than others. An example of this is the You Only Look Once (YOLO) person detector (Redmon et al., 2016)[20]. This process takes a single shot of the entire image and breaks down the possibility of being different objects. YOLO is capable of detecting a variety of objects such as pedestrians, trash cans, doors, and many other objects. Having this algorithm run for this robot would be able to navigate a variety of obstacles. The downside with this algorithm is the processing speed. As mentioned in this paper, YOLO runs between 45 to 150 frames per second on the Nvidia Titan X GPU. Running this algorithm on the laptop however resulted in approximately 2 frames per second video.

4.4.1.2 Histogram of Oriented Gradients Person Detection

The less computationally intensive algorithm was the Histogram of Oriented Gradients (HOG) person detector at around 20 frames per second. It was trained to detect people and would struggle occasionally, especially when the pedestrians are too close

and only legs are visible or too far away. This was mentioned by Dalal and Triggs (2005) [9] where gradients should be detected at the finest available scale and should have good contrast with its surroundings. The problem with far away pedestrians is that the detector does not receive a sharp image at that small a scale.

Although HOG performed quicker, the clear problem with HOG person detection was that it would miss a frame where a person was in. This can be a problem if the robot had just detected a person and attempted to move out of the way, but did not move enough. If the detection algorithm misses that the pedestrian is still in the path of the robot, it can result in a collision. One way this could be fixed is by changing the HOG window stride or scale parameters so it would run a more intensive search process. Unfortunately, by increasing the intensity of the search process, it drastically increased the time to process the video. To increase the detection rate, optical flow could be integrated with the HOG person detection. This is similar to the work done by Ramzan et al. (2016) [19]. In this paper, HOG person detection was performed after detecting motion using optical flow. This was done with a static camera. In the paper, the authors mention that this procedure would not work with a moving camera since the background would be dynamic. Instead, the problem of person tracking and frame continuity would be be better suited to the work done by Gavrila and Munder (2006) [11], to complete the gaps in person detection. The process is as follows: once HOG successfully detects a person, this bounding box becomes a region of interest. The optical flow takes the average motion in that bounding box and computes the region of interest's translation in the camera space. On top of the existing HOG person detection, a more intensive search is applied to this translated region of interest. The procedure above increases the frame connectivity for the detection by combining tracking capabilities with HOG.

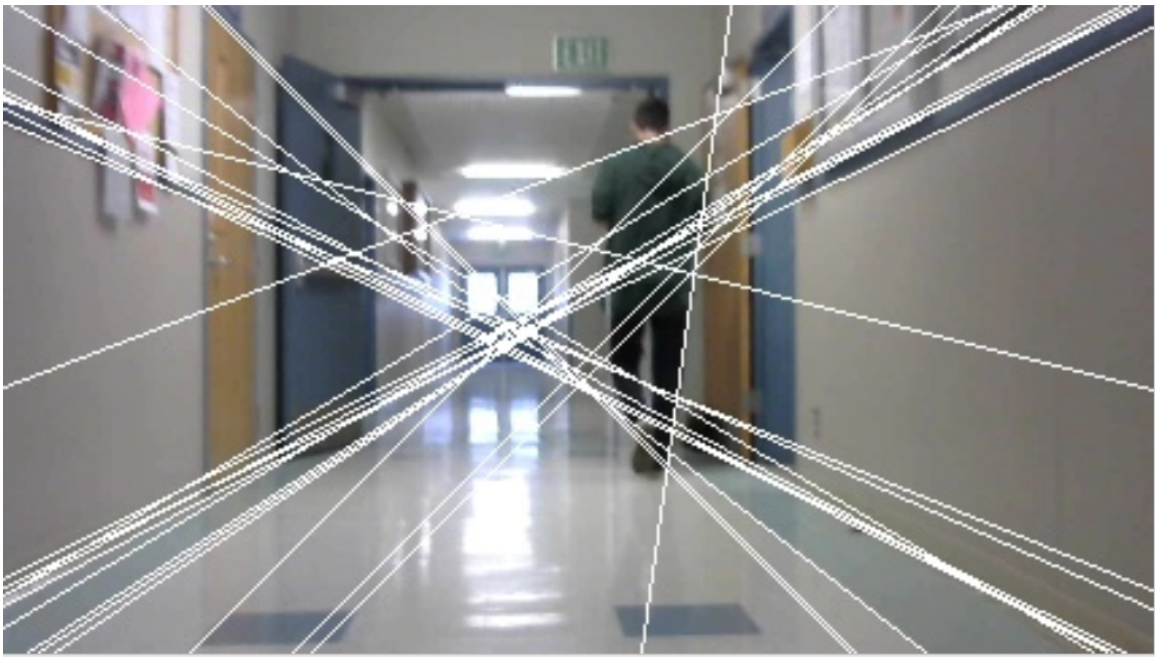
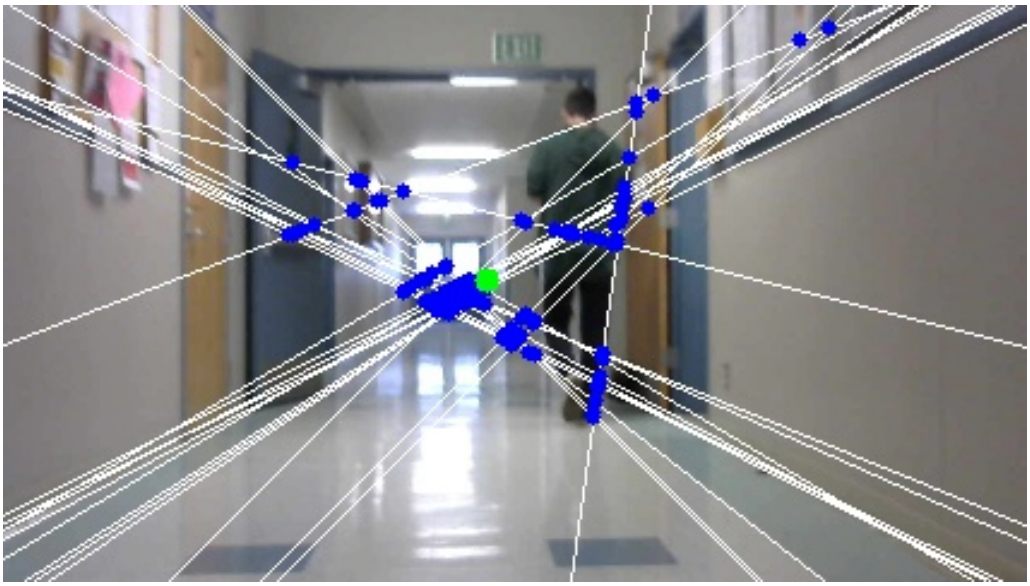


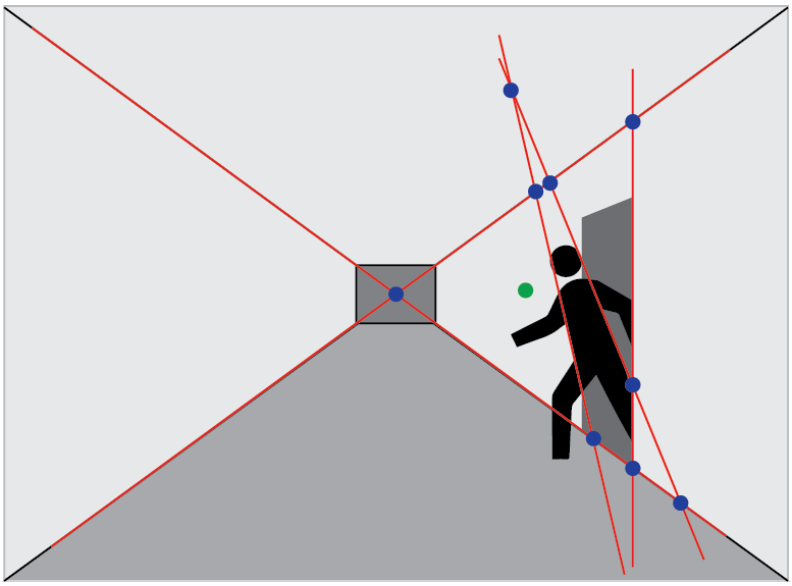
Figure 4.12: Hough Lines from an Image with a Pedestrian

4.4.2 Person Extraction

By observing the visualized Hough lines from Figure 4.12, it is clear that a line from the Hough transform is created by the person in the frame. This can have an affect on the vanishing point as seen in Figure 4.13 where the green circle is denoting the average value of the center point when the actual vanishing point is more likely the center of the two doors. One approach to fixing this error is by removing lines created by the pedestrian. To do this, once the HOG person detection algorithm returns a bounding box around the person, the lines that are calculated inside this bounding box are removed. This should fix the shifted vanishing point visualized as the green dot caused by the pedestrian in this frame.



(a) Shifted Vanishing Point Caused by the Pedestrian



(b) Simple Vanishing Point Skew Example

Figure 4.13: Skewed Vanishing Point Examples

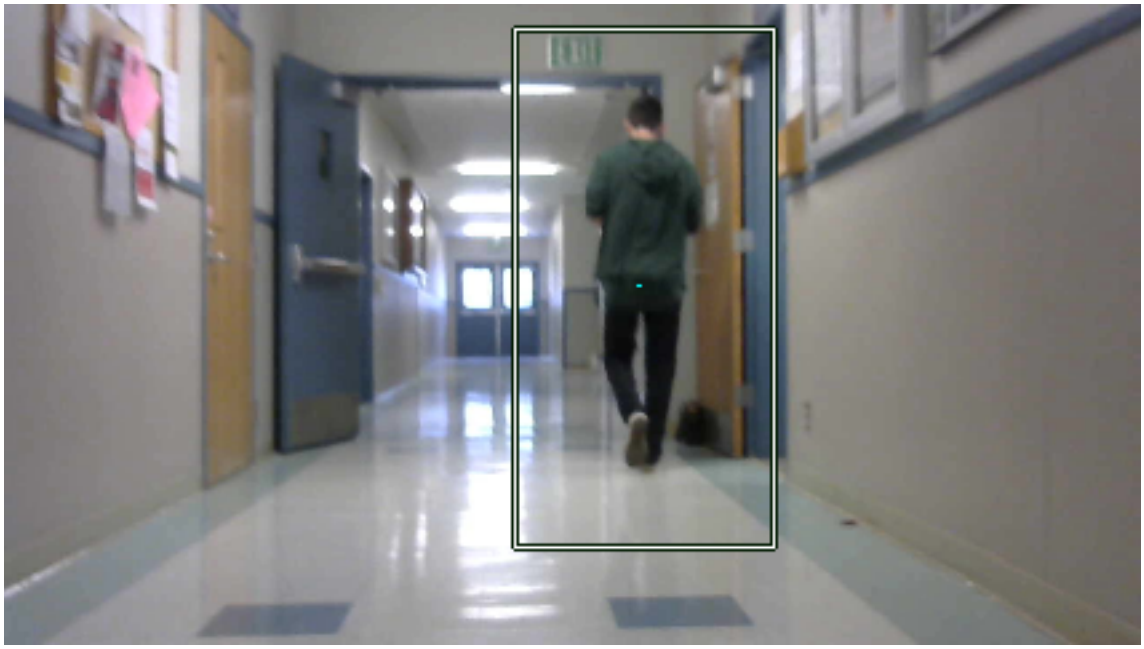


Figure 4.14: Person Detected Using HOG Person Detection Algorithm

4.4.3 Person Avoidance

Person extraction only removes lines created by pedestrians. However, if the pedestrian is in the path of the robot, the robot must move around them to prevent a collision. By finding the center of the bounding box and comparing that center to the vanishing point, the algorithm is capable of determining which side to move towards. For example, if the person is located to the left of the vanishing point, the robot should move towards the right to avoid them. The bounding box also impacts how severe the robot should turn in the direction specified. By taking the lower y coordinate, it signifies the distance that the person is from the robot. The closer the person is to the robot, the sharper the turn required to avoid them. This is computed linearly.

Algorithm 3: Avoiding Pedestrians

Result: Avoid Pedestrians

Search for Person using HOG and Linear SVM;

if *Person Found and N counter is 0* **then**

Determine the center of bounding box to extract location of pedestrian relative to vanishing point;

Determine the bottom of bounding box to extract distance of pedestrian from robot;

if *Bounding Box Center is to the Left of Current Vanishing Point* **then**

Use the distance between the pedestrian and the robot to adjust the sharpness of the turn;

Turn right ;

Drive Straight;

Turn left;

else

Turn left;

Drive Straight;

Turn right;

end

Stabilize and re-calibrate to find the vanishing point;

Set counter N;

else

Follow Vanishing Point Algorithm;

end

4.5 Arduino Driving

The purpose of the Arduino is to receive serial commands and drive the motor. No processing should be taking place on the Arduino since it processes at around 16MHz. This also centralizes all the software and algorithmic processes.

Chapter 5

TESTING/RESULTS/EVALUATION

This chapter will cover the various experiments run with the system to determine how the changes in the vision algorithm affected the accuracy of the vanishing point as well as the performance of the robot in an actual hallway setting. Having multiple methods of testing is a vital component to validating whether or not changes improve a system or not.

5.1 Vanishing Point Testing

Accuracy of the vanishing point is a difficult measure to test in live scenarios. Random events can occur and these events can not be replicated every time. This is why multiple videos were used to test the results of the vanishing point. With multiple videos, the tests across different algorithms could be run on the same videos and compared.

5.1.1 Electrical Engineering Hallway

To test the vanishing point accuracy, eight different videos were captured. They were all in the Electrical Engineering Hallway on Cal Poly campus. The next step is to find a good measure to compare multiple vanishing point tests to. Even though the camera was manually pushed down the hall, the vanishing point is not perfectly stable. Because of this, the vanishing point needed to be manually chosen for each frame and saved into a CSV file. This file was compared to the vanishing point calculated for each algorithm modification to verify if the mean difference had improved or not.

The 8 videos described in 5.1.1.1 each had over 750 frames and were capped at 750 to provide consistency.

5.1.1.1 Electrical Engineering Hallway Video Testing Descriptions

1. The camera was rotated from side to side in the hallway, moving the actual vanishing point between the left side of the frame to the right side quickly. The rotations varied in speed and in distance.

2. The rotation of the camera attempted to keep the vanishing point in the center of the frame while the camera's position was shifted horizontally left and right between the sides of the hallway.

3. The camera was pushed down the center of the hallway that was empty.

4. The camera was pushed down the center of the hallway with one student who walks towards the camera

5. The camera was pushed down the center of the hallway with one student who walks towards the camera and then two students follow shortly after

6. The camera was pushed down the center of the hallway with multiple pedestrians walking towards the camera, doors open, and some pedestrians standing on the side

7. The camera was pushed down the center of the hallway with a person who is sitting on the right side of the hallway

8. The camera was pushed down the center of the hallway with a student passing in front of the camera and then walking away and then towards the camera with another student following after.

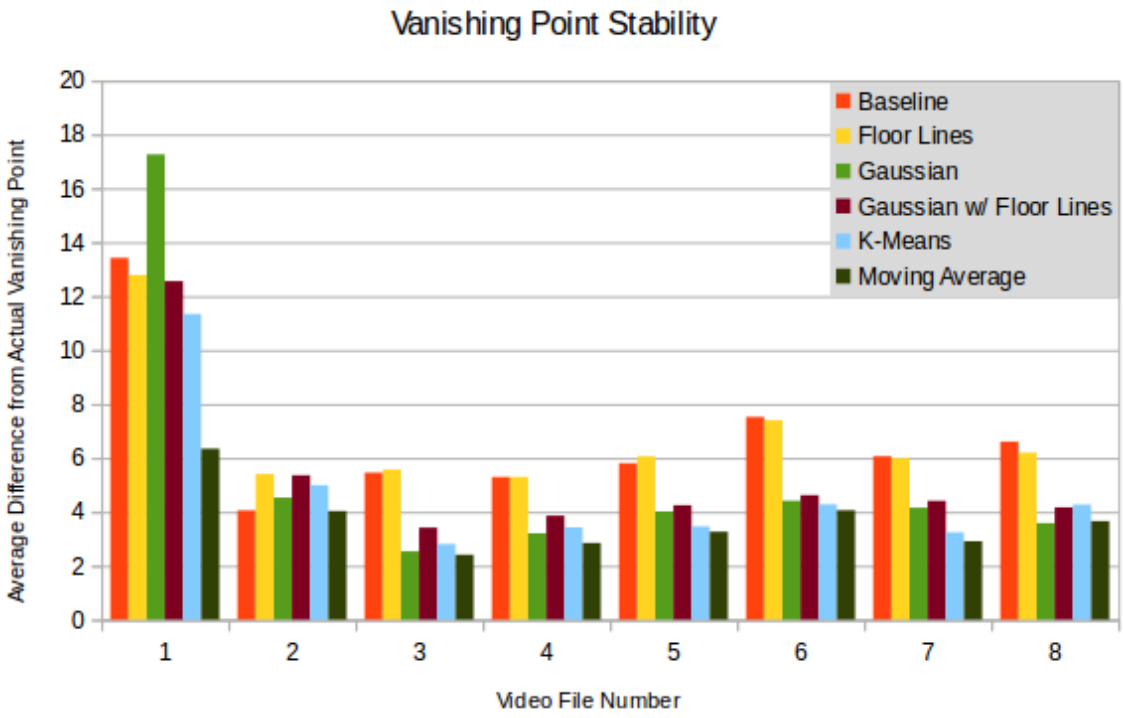


Figure 5.1: Vanishing Point Stability

The orange bar represents nothing changed from the basic vanishing point algorithm described in Figure 4.1. The yellow bar represents floor line tracking as described in section 4.2.2.1. The green bar replaces the uniform averaging method with Gaussian weighting described in section 4.2.1. The maroon bar combines the floor lines with Gaussian averaging. The sky blue bar adds the K-Means algorithm described in section 4.2.2.2 to the Gaussian with floor lines. The final green bar adds a moving average of 5 frames to the algorithm.

Figure 5.1 shows the vanishing point stability. The first video file was significantly higher in the difference. This is because the video feed rapidly rotates which makes it difficult for many algorithms to track the vanishing point with the exception of the K-Means algorithm. There are a few trends that can observed. In most of the videos, the floor lines performed similarly to the baseline test case. This is understandable because floor extraction follows the same Canny and Hough extractions. The best

Table 5.1: Average Vanishing Point Improvements from Complete Algorithm

Baseline	Floor Line	Gaussian	Gaussian w/o Floor Lines	K-Means
42.85%	45.44%	18.23%	26.56%	16.35%

method for vanishing point stability was undoubtedly the complete algorithm. This can be further shown in Table 5.1.

The other statistic observed was the variance of the difference from the actual vanishing point. This can be seen in Figure 5.2. Because the first video was constantly moving the vanishing point, the variance was extremely high. Despite that however, it can be seen that the K-Means is the most precise. In Figure 5.3, the first video file was removed to see how the variance of the vanishing point compared in the other video files. The trend of this graph is that averaging and floor line calculation varies the most. This is exaggerated in video file number 6, which is the extremely crowded hallway. This is because slight variations from students will shift the vanishing point dramatically. The two that performed the best were Gaussian with floor lines and K-Means with moving average.

There is a drawback from the improvements however. In Figure 5.4, the maximum frames per second is shown. Because the camera captures at approximately 24 frames per second, there is no way that the algorithm can run at 140 frames per second. However, this shows that K-Means is computationally intensive and can only run at approximately 10 frames per second on the single core processor.

5.1.2 Hallway Driving Test

In the previous section, the results from testing showed that the complete algorithm does improve vanishing point tracking. However, this needed to be verified using

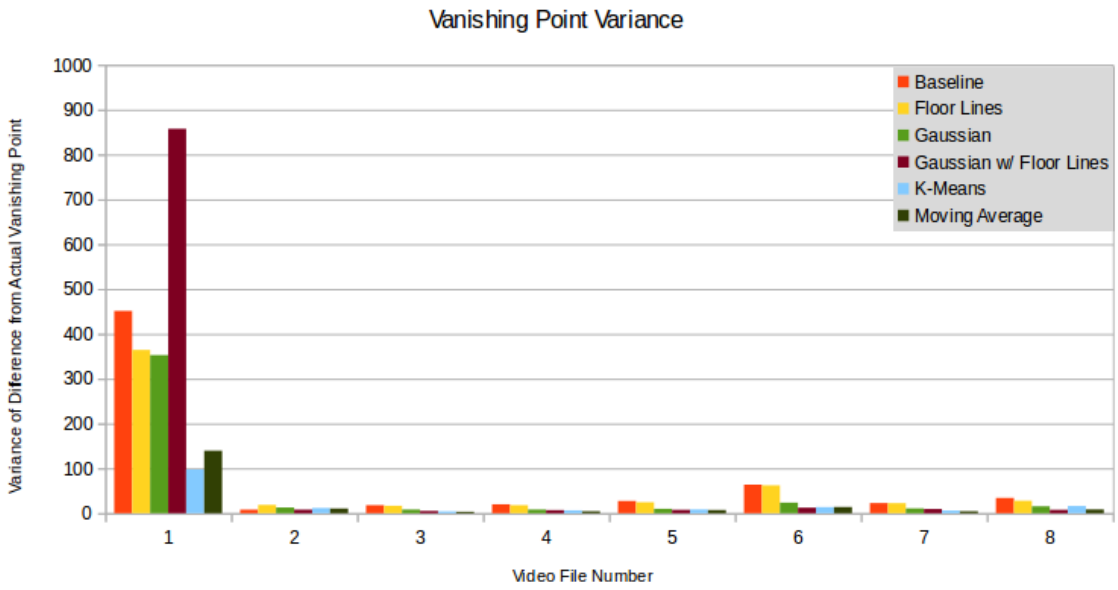


Figure 5.2: Variance of the Vanishing Point

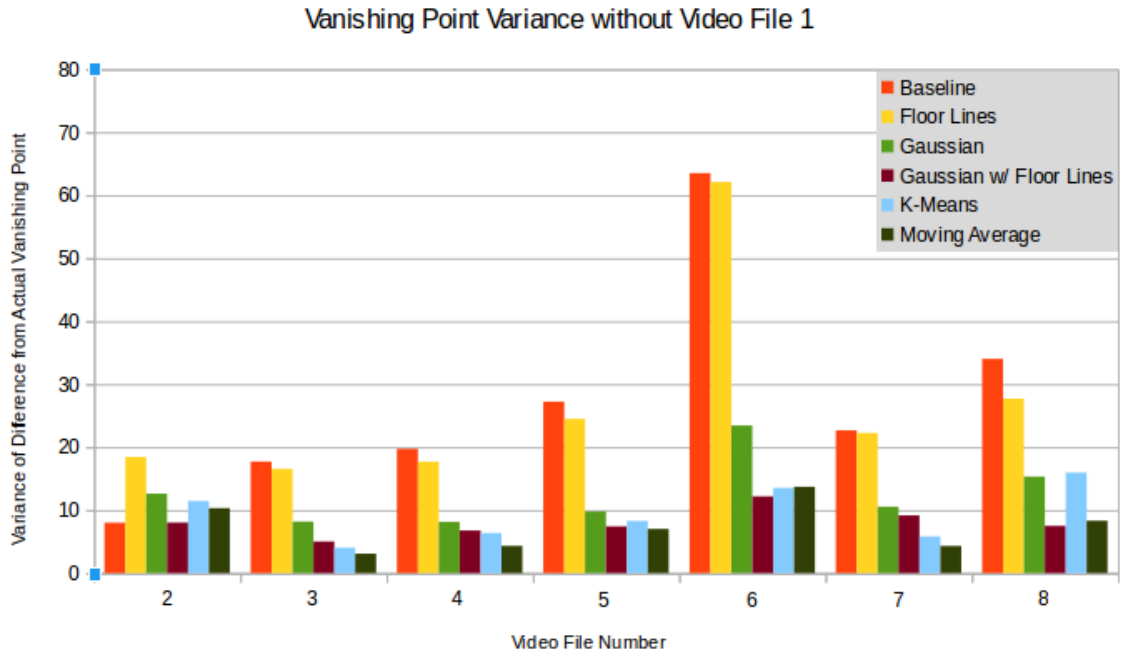


Figure 5.3: Variance of the Vanishing Point Without Video Number 1

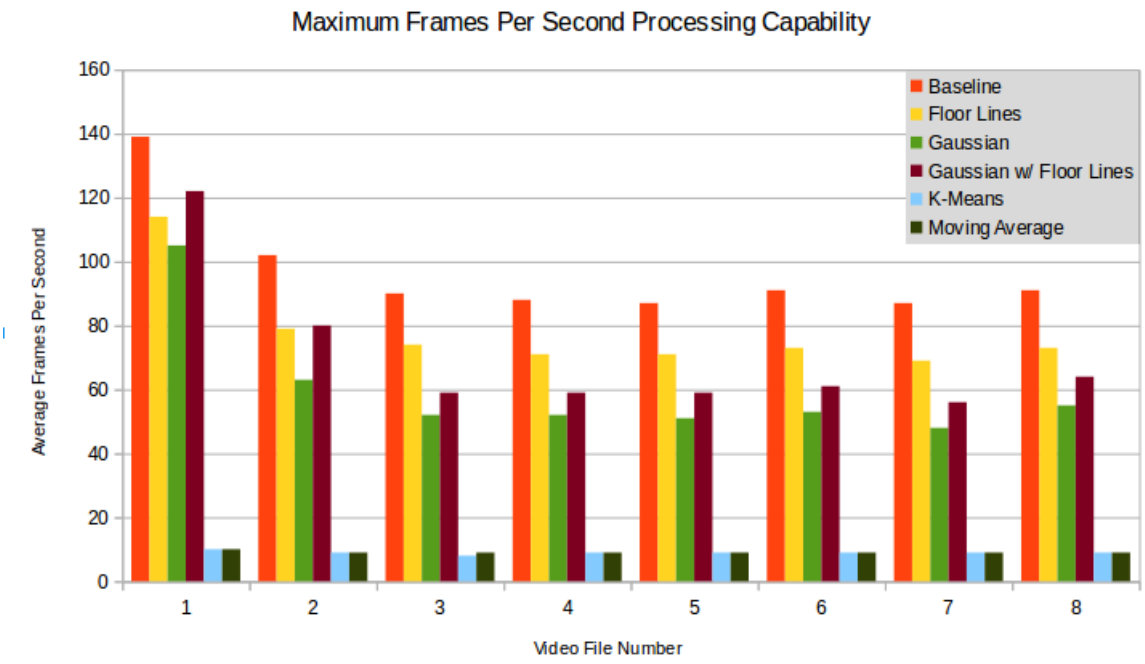


Figure 5.4: Maximum Frames Per Second

the stability testing system described in section 3.1.2 to see if the integrations of all the algorithms improve robot navigation. The robot was placed in the center of the beginning of the hallway and 100 squares were measured to set a standard travel distance for multiple runs. The Figure 5.5 shows the results comparing the baseline algorithm and the completed algorithm. These tests were averaged over three runs each to eliminate some random factors the robot could be affected by. In order to average results that have different number of data points depending on the time the robot took to travel the hallway, a fixed number of points that was less than the minimum number of data points was selected. The data was then sampled to result in the the same number of points selected earlier. Because the LIDAR is pointing at the right side, this graph shows the distance from the right side. When the robot is completely centered in the hallway, the LIDAR reads 44 inches. This can be seen in the reading at the start of the graph. As the robot drives down the hallway, in both cases the robot drifts to the right. Although the completed algorithm does a better

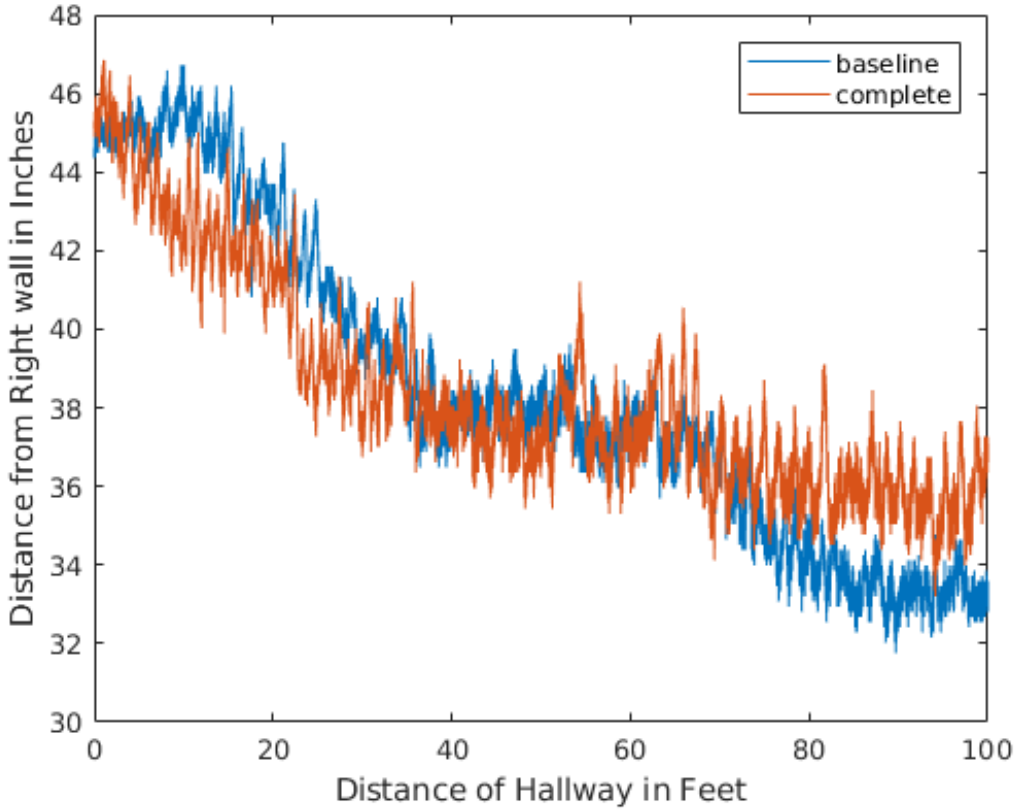


Figure 5.5: Empty Hallway Driving Test

job in staying centered, there is more noise in the LIDAR data. This is because the algorithm is slower, so there is more correction procedures while driving compared to the baseline one.

5.2 Incorporating Person Detection with HOG

The two uses of the HOG person detection algorithm were tested. The first was to see if the vanishing point stability would be improved when the procedure described in section 4.4.2 was used to detect pedestrians and remove lines in the pedestrian's bounding box. The second use was to avoid pedestrians by comparing the vanishing

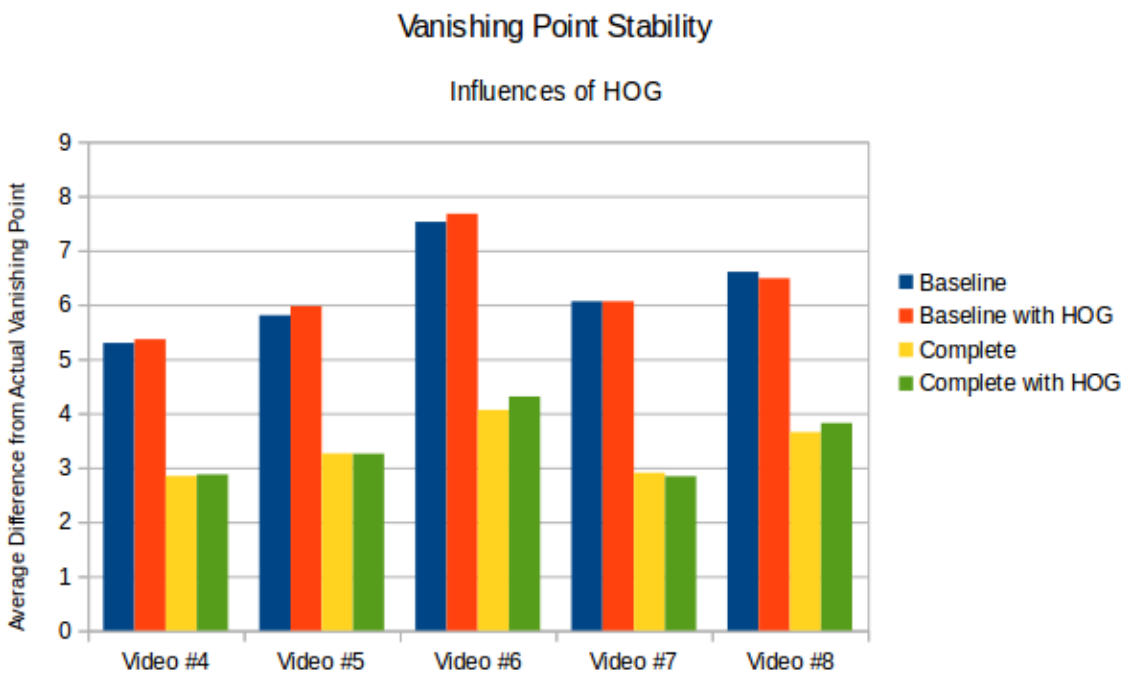


Figure 5.6: Line removal using HOG

point to the bounding box extracted by HOG. This procedure was described in section 4.4.3.

5.2.1 HOG Line Removal

To compare whether or not HOG improved the algorithm detection, both the baseline and the complete algorithms were run with and without HOG on videos 4 through 8. These five videos were the ones that had pedestrians walk by, so if the algorithm was to make an impact, it would on these videos. The results can be seen in Figure 5.6. This graph shows that HOG does not make a significant impact on the entire video navigation. In some cases it performed slightly worse. The average impact from HOG line removal is a 1.52% decrease in accuracy.

The reason that line removal can sometimes hurt the vanishing point algorithm is the pedestrian's bounding box encompasses a buffer space around them. This might

eliminate helpful helpful lines as well as remove the distracting ones. On top of this, the complete algorithm uses a Gaussian distribution to choose the best vanishing point. This reduces the weights that the pedestrians impact the vanishing point.

5.2.2 HOG Person Avoidance

Even though the line removal did not assist with the stability of the vanishing point, HOG is still necessary for pedestrian avoidance in the hallway. By driving and implementing the algorithm described in Figure 3, the robot is successfully able to move away from a person. This was done in the Electrical Engineering hallway of this can be seen in Figure 5.7. The path is shown in green and red. The red segments denote a person has been detected and the robot is attempting to move out of the way. The green segments signify normal navigation.

While the robot can avoid pedestrians in the hallway, the algorithm is not perfect. This can be seen in the second red line segment where the robot actually moves closer to the pedestrian. This is because while the robot is attempting to move away, the camera is not taking in new frames. This means that when the robot takes the next frame, the previous frame's vanishing point might not the same as the current one. The algorithm must then adapt and discover the vanishing point on its own. In the case of the second red line segment, the vanishing point was calculated to be to the right of the pedestrian so the robot moved right. The third red segment corrects this mistake and turns sharply away from the pedestrian since it notices that the pedestrian is close to the robot. For these tests, the pedestrians are not walking down the hallway. This is due to the limitation of the laptop's processor only able to run at approximately 10 frames per second.

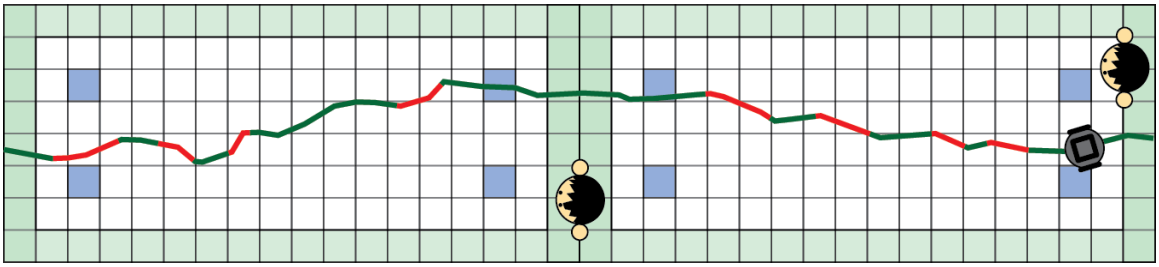


Figure 5.7: Robot Test Drive

5.3 New Scenario Testing

The methods used to develop the modifications to the vanishing point algorithm were built upon the hallways in the Electrical Engineering building. The parameters and thresholds selected for each algorithm have been tuned for this hallway. To determine if these modifications improved driving in multiple situations, different videos were captured by pushing the laptop down the hallway similar to what was done in section 5.1.1.1. The three scenarios used were the Computer Science hallway, the Math hallway, and Via Carta road.

5.3.1 Computer Science Building

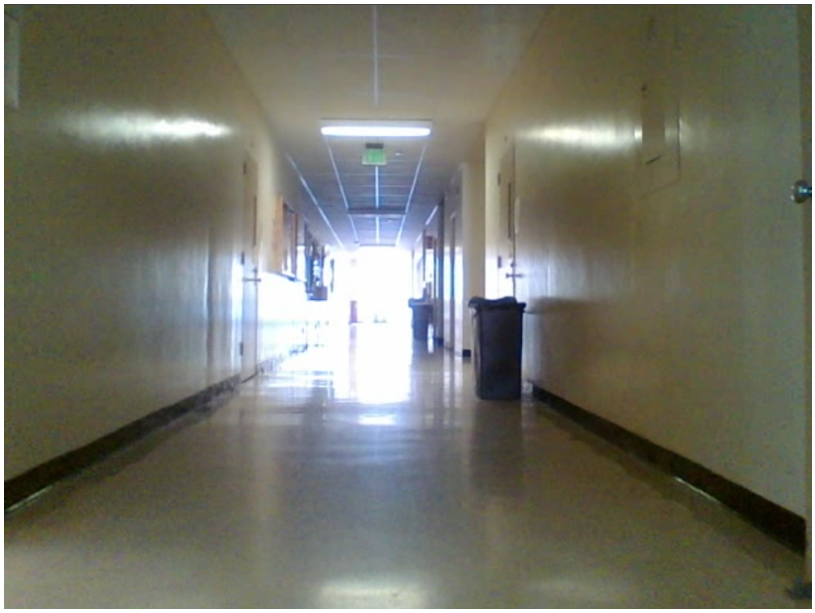


Figure 5.8: Computer Science Hallway

The first case was the Computer Science hallway. The largest difference from the Electrical Engineering building by having hallway sized windows on one side. This allows a significant amount of light to flow into this hallway which creates a bright center which contrasts from the walls. This may make it more difficult to find lines since there may be additional lines that appear from the edges of the light. Another difference is that the baseboard used is black. This may increase the ease of finding floor lines since it is a higher contrast to that of the walls and floors.

Figure 5.9 shows the intersections consolidated at the end of the hallway. This is because there are very little distractions along the walls in the computer science building. The walls and floors are pastel white and the baseboard is black. This makes the floor lines very defined and a major help to calculate the vanishing point. However because the walls and floor are both pastel colors, the K-Mean algorithm is

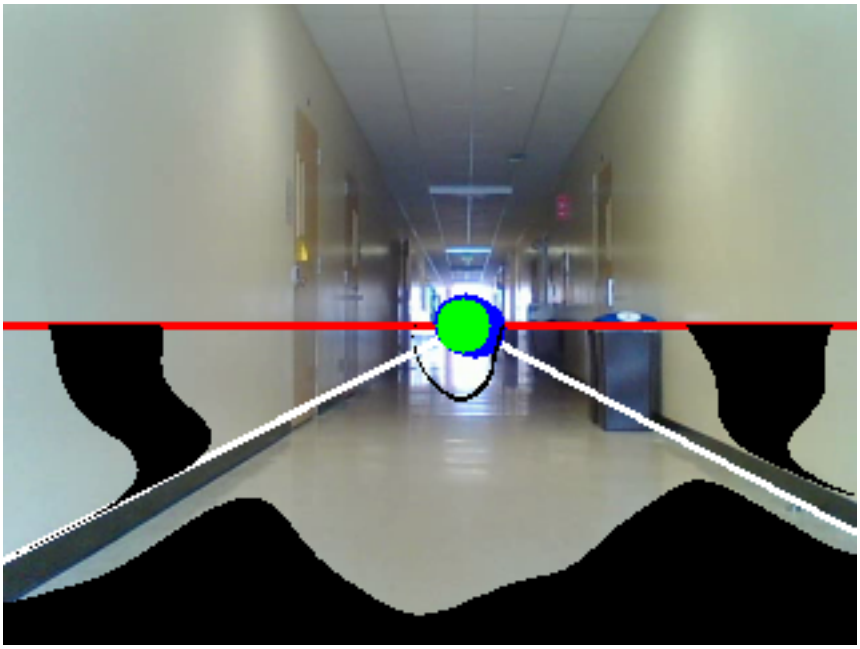


Figure 5.9: Computer Science Hallway Algorithm

not as effective in calculating the vanishing point. As noted earlier, the bright light from the hallway sized window at the end of the hallway causes error as well.

5.3.2 Math Building

Figure 5.10: Math Hallway

The second case was the math building. This was a very simple hallway, even more so than the Electrical Engineering building. The walls were white with hardly any posters, bulletins, or additional content that would add noise to the algorithm. The floor was one solid color and not a mixture of several compared to the EE building. This means that the K-Means algorithm may find it easier to isolate the floor. The baseboard had a sharp contrast with the white walls and floors making it stand out which can result in better floor detection.

Similarly to the computer science building, in Figure 5.11, the vanishing point line calculation performs very well and the intersections are localized. The floor lines are well defined since the baseboard contrasts with the wall and floor. However, K-Means does not perform as well. Since the entire building is very pastel white, it is very easy for the K-Mean algorithm to group the floor white color to the walls.

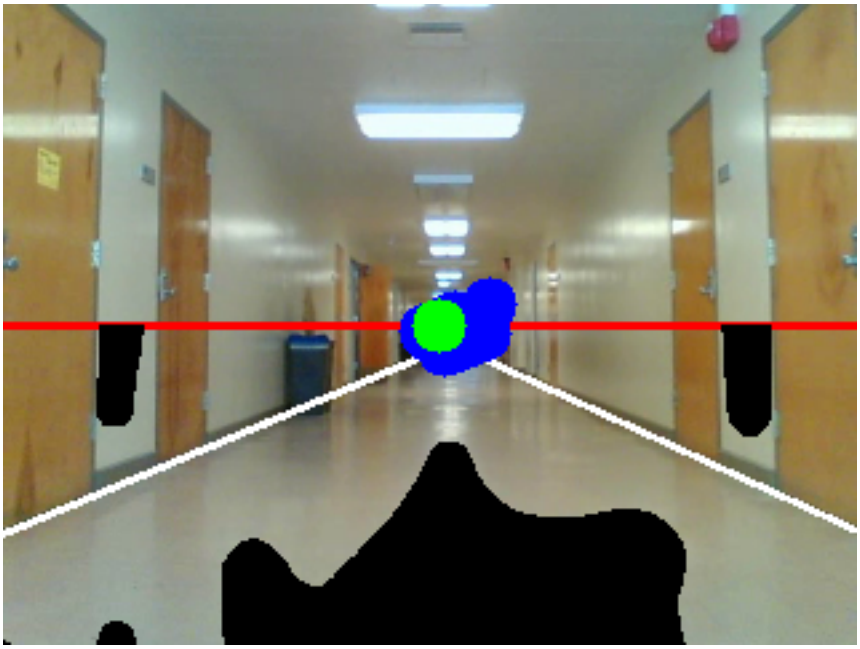


Figure 5.11: Math Hallway Algorithm

5.3.3 Outside Road

Figure 5.12: Outside Road

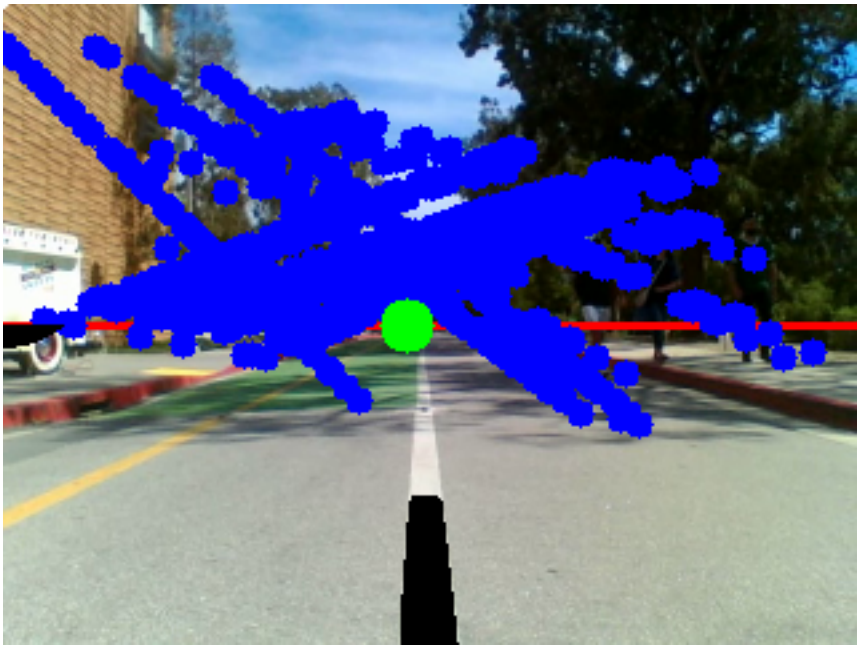


Figure 5.13: Outside Road Algorithm

The last case was an outdoor setting on Via Carta road between the EE building and the Baker Science building. There are no walls to assist with vanishing point calculation, but the curb is painted red and the pavement is black which is completely different from the natural surroundings. Hopefully with this, the K-Means algorithm in this case should be more successful than vanishing point alone.

From Figure 5.13, this is clearly the noisiest in terms of vanishing point intersections. However, the K-Mean algorithm is able to help out by tracking the white line in the center of the road. This method does not fix the difficulty at navigating outside because if the vanishing point center is off and begins to track the gray surroundings, it will have varying peaks since the trees create darker shadows. The hypothesis from earlier that the red pavement will help define the floor lines was inaccurate since the lines rarely showed up in this test. The cause could also be the trees casting a shadow on the red curb which distorts the lines.

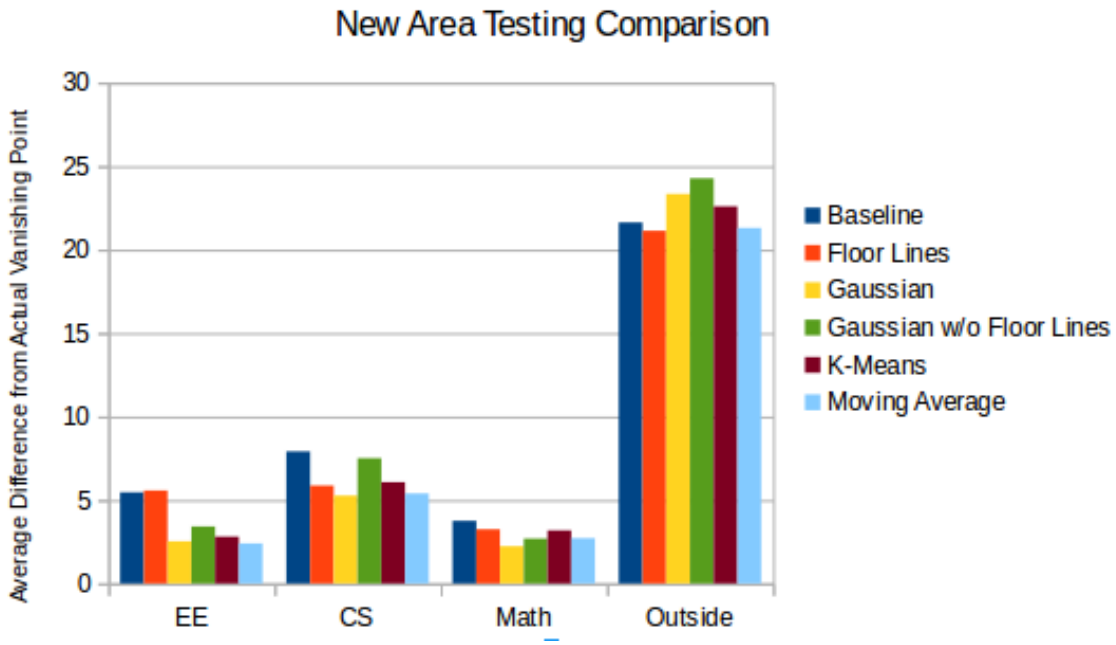


Figure 5.14: New Scenario Testing Results

5.3.4 New Scenario Vanishing Point Testing Results

The same steps run in the EE hallway were run on the three test cases above. Figure 5.14 shows that the complete algorithm performed on the better side than the other cases. The graph also shows that this algorithm is better suited for indoor runs. This makes sense since the thesis was aimed towards an indoor environment and was not built for outdoor situations.

Another observation is that for the Computer Science and Math building hallways, the best run was Gaussian with Floor lines. The success of this was because the baseboard contrasted heavily to the wall and floor making floor tracking much easier compared to that of the Electrical Engineering building. Especially since the K-Means algorithm captured the walls when segmenting the floor, using this method provided useless and probably more detrimental to the vanishing point calculation than beneficial.

Table 5.2: Average Vanishing Point Improvements from Complete Algorithm Run on Different Scenarios

Baseline	Floor Line	Gaussian	Gaussian w/o Floor Lines	K-Means
7.64%	2.96%	-1.89%	4.92%	3.95%

Table 5.3: Average Number of Hough Lines

EE	CS	Math	Outside
4.25	23.02	3.86	236.23

As seen in Table 5.2, the complete algorithm did improve the vanishing point stability for most of the values. When it improved the stability, the percentage it did improve was not as large as the training tests in the Electrical Engineering hallway. The completed algorithm even performed worse on average than the Gaussian and Floor Lines case by 1.89%. This was due to the K-Means algorithm segmenting the walls when trying to segment the floor as well. Overall however, the baseline improvement from the different scenario runs was an average of 7.64%. This is compared to the 42.85% average improvement seen in the EE hallway. Since this algorithm was not built for these new situations, the improvement could not be as drastic. Many of the parameters were selected to best fit the EE hallway.

To investigate the reason behind the noisiness of the vanishing point of the outside path, the number of Hough lines were extracted and averaged over each frame. As seen in Table 5.3, the number of Hough lines was significantly higher than the others. This is a clear indicator that the results will be noisy because the Hough line transform combines them into one line if there are lines close in angle or proximity. This is why for the Math building, there is only an average of 3.86 lines after removing the vertical

Table 5.4: Mean and Variance Results Comparing the Addition of K-Means with Floor Line Restraint

	EE	CS	Math	Outside
Mean Before	2.40	5.39	2.71	21.29
Mean After	2.37	5.94	4.19	22.01
Variance Before	4.25	23.02	3.86	236.23
Variance After	2.63	21.91	3.31	204.66

and horizontal lines, but the stability was the most accurate out of all the new area tests.

5.3.4.1 Implementing K-Means Line Restraint

From the real world testing data above, the CS and Math building excelled using floor lines and basic vanishing point calculations. The K-Means algorithm was less helpful since the walls and floor could appear similar in color and therefore get segmented together. The slight modification to the algorithm completed at this point is to use the floor lines to help determine the vanishing point with K-Means. If the contour line peak detection from 1 was above the floor lines, it was considered a wall and not part of the segmentation. The results of this can be seen in 5.4. For the EE hallway, both the average mean difference and the variance improved. This could not be said for the other navigation methods. Variance decreased across all of them, but the mean increased as well. This means that the vanishing point calculations with the floor line restraint are more stable but slightly misaligned.

Chapter 6

FUTURE WORK

This thesis provides enormous opportunity to expand. On top of the work done, there are a number of ways monocular vision for hallway navigation can be improved.

6.1 Automatic Adjustments

The algorithm could determine which method is most reliable in its situation. By having the robot sit and process before driving, the algorithm might be able to determine in certain situations that K-Means segmentation will not be a reliable method of navigation as seen in Figure 5.10.

Another way the robot could determine the best navigation method is by automatically choose threshold values for the different algorithms. Canny Edge, Hough Transform, and K-Means algorithms all have thresholds that were selected and built in the EE hallway. An example of this would be to limit the number of lines are found by the Canny Edge algorithm. The threshold value would be set to best adapt to the current environment. This would prevent the large number of edges which resulted in a cluster of intersections found in Figure 5.12.

6.2 Robot Improvements

Although the robot chassis drove smoothly, the robot used has significant drawbacks when it navigates. The main drawback is the two forward drive wheels. This is good for on point turns, but when avoiding a person or simply navigating, it has to turn away from looking forward. This could be improved by having omniwheel driving. This would let the robot strafe while keeping the vision aimed down the hallway.

Another downside to the chassis is the low perspective. This makes it more difficult to isolate the ground compared to if the camera was located higher up.

Since this is the baseline method using only one camera, this could be improved by adding more sensors. The next step would be to add another sensor and implement stereo vision. This could improve the person navigation by increasing the accuracy of person detection, but also assist in navigation by detecting depth of field.

Another sensor to add on would be a LIDAR. This could help with the navigation, but also for obstacle detection. The camera does this with object recognition, but in a monocular vision case, the depth of the object is limited. However, if the camera detects a person, the LIDAR could be used to verify the positioning of the person respective to the robot and area around it to determine the best possible path of travel.

This thesis provides a foundation to argue the benefits of the camera as the main sensor in navigation. However, there are still many improvements that can be made.

Chapter 7

CONCLUSION

From the work presented in this thesis, a single web camera is able to provide enough information to navigate a corridor. On top of this, improvements on the vanishing point algorithm can be made to negate the impact pedestrians and obstacles play in hallway navigation. These improvements also increased the accuracy of vanishing point algorithm, outperforming each individual modification when run on videos in the Electrical Engineering hallway on Cal Poly's campus. Compared to the baseline algorithm, the complete algorithm improved the stability by 42.85%. When tested on an actual robot averaged over multiple runs in the same hallway, the complete algorithm is 4% more stable. The other addition made to the algorithm was the integration of person detection. The navigating robot was successful in pedestrian detection and avoidance in a simulated environment.

With this work, there is still much room for further optimization in the future. After running tests in different scenarios, the complete algorithm in its current form is not robust and achieved only a 7.64% improvement in stability compared to the baseline algorithm. This is much smaller improvement than the 42.85% gained when implementing the complete algorithm in the EE hallway. It also performed worse by 1.89% compared to the algorithm implementation using Gaussian weighting and floor line finding. To improve robustness, the algorithm could be trained on multiple hallways.

One of the greatest limiters to the accuracy and speed of the robot was the hardware. There is significant processing cost when integrating each algorithm on this laptop. With a GPU and faster processing speed, it would be easier to compare the two algorithms equally since the robot could run in real-time. This new hardware

could be integrated with a powerful camera. Newer cameras are capable of taking images faster. This thesis shows that one single webcam alone can provide enough information to navigate a hallway. While maintaining the camera-centric approach in hallway navigation, hopefully this thesis will lead to tests and developments that consider real world scenarios.

BIBLIOGRAPHY

[1] https://buy.garmin.com/en-US/US/p/557294.

[2] Adafruit BNO055 Absolute Orientation Sensor. https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview.

[3] Adafruit Unified BNO055 Driver. https://github.com/adafruit/Adafruit_BNO055.

[4] Cal Poly Github. http://www.github.com/CalPoly.

[5] Dual VNH5019 Motor Driver Shield Library. https://github.com/pololu/dual-vnh5019-motor-shield.

[6] LIDAR-Lite v3 Arduino Library. https://github.com/garmin/LIDARLite_v3_Arduino_Library.

[7] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, NO 6*, 1986.

[8] C.-K. Chang, C. Siagian, and L. Itti. Mobile robot monocular vision navigation based on road region and boundary estimation. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[9] N. Dalal and B. Triggs. Histogram of oriented gradients for human detection. *International Conference on Computer Vision and Pattern Recognition*, 2005.

[10] G. Farneback. Two-frame motion estimation based on polynomial expansion. *Lecture Notes in Computer Science, vol 2749*, 2003.

[11] D. Gavrila and S. Munder. Multi-cue pedestrian detection and tracking from a moving vehicle. *International Journal of Computer Vision 73, pp. 41-59*, 2006.

[12] Itseez. *The OpenCV Reference Manual*, 2.4.9.0 edition, April 2014.

[13] Itseez. Open source computer vision library. `https://github.com/itseez/opencv`, 2015.

[14] D. K. Kim and T. Chen. Deep neural network for real-time autonomous indoor navigation. *Cornel University*, 2015.

[15] S. Konam. Vision-based navigation and deep-learning explanation for autonomy. *Carnegie Mellon University*, 2017.

[16] H. Kong, J.-Y. Audibert, and J. Ponce. Vanishing point detection for road detection. *Internation Conference on Computer Vision and Pattern Recognition*, 2009.

[17] A. Kosaka and J. Pan. Purdue experiments in model-based vision for hallway navigation. *Proceedings of Workshop on Vision for Robots in IROS'95 Conference*, 1995.

[18] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. 1981.

[19] H. Ramzan, B. Fatima, A. Shahid, S. Ziauddin, and A. Safi. Intelligent pedestrian detection using optical flow and hog. *International Journal of Advanced Computer Science and Applications*, 2016.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *International Conference on Computer Vision and Pattern Recognition*, 2016.

[21] T. Saitoh, N. Tada, and R. Konishi. Indoor mobile robot navigation by center following based on monocular vision. *IEEE Transactions on Electronics, Information and Systems, Volume 129, Issue 8, pp. 1576-1584*, 2009.

[22] J. Souza, G. Pessin, P. Shinzato, F. Osorio, and D. Wolf. Vision-based autonomous navigation using neural networks and templates in urban environments. *Mobile Robotics Laboratory, University of Sao Paulo*, 2011.

[23] Z. Zhou, T. Chen, D. Wu, and C. Yu. Corridor navigation and obstacle distance estimation for monocular vision mobile robots. *International Journal of Digital Content Technology and its Applications. Volume 5, Number 3*, 2011.