RECONSTRUCTION TECHNIQUES FOR FIXED 3-D LINES AND FIXED 3-D POINTS USING THE RELATIVE POSE OF ONE OR TWO CAMERAS

A Thesis
Presented to
The Academic Faculty

by

Roshan Satish Kalghatgi

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
G.W. Woodruff School of Mechanical Engineering

Georgia Institute of Technology
May 2012

RECONSTRUCTION TECHNIQUES FOR FIXED 3-D LINES AND FIXED 3-D POINTS USING THE RELATIVE POSE OF ONE OR TWO CAMERAS

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Nader Sadegh for giving me the freedom to develop my own Master's thesis project. His brilliant insights and guidance were fundamental to the timely completion of this work. I would also like to thank my parents, especially my mother Manisha and my sister Reena for raising me, encouraging me and supporting me over the years as I chose to follow my ambitions and dreams. I would also like to thank my friends Geetanjali Ningappa, Myles Akin, Alexander Pitt, Alexander Merritt and others that I have met along the way for making my years at Georgia Tech some of the best years of my life. I would also like to acknowledge the field of science and science fiction for inspiring me and motivating me to continue to follow this path that I have taken.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

Stereovision is a common computer/machine vision technique used to extract three dimensional information from a set of images. These images can be acquired with two or more cameras, or a combination of cameras and IR sensors placed in a known pose relative to a world coordinate frame. Typical applications use two cameras each with a known pose relative to a world coordinate frame.

In general, stereovision can be defined as a two part problem. The first is the *correspondence problem*. This involves determining the image point in each image of a set of images that correspond to the same physical point *P*. We will call this set of image points, *N*. The second problem is the *reconstruction problem*. Once a set of image points, *N*, that correspond to point *P* has been determined, *N* is then used to extract three dimensional information about point *P*.

This master's thesis presents three novel solutions to the reconstruction problem. Two of the techniques presented are for detecting the location of a 3-D point and one for detecting a line expressed in a three dimensional coordinate system. These techniques are tested and validated through point detection or a finger gesturing application. The techniques presented are unique because of their simplicity and because they do not require the cameras to be placed in specific locations, orientations or have specific alignments. On the contrary, it will be shown that the techniques presented in this thesis allow the two cameras used to assume almost any relative pose provided that the object of interest is within their field of view.

The relative pose of the cameras at a given instant in time, along with basic equations from the perspective image model are used to form a system of equations that when solved, reveal the 3-D coordinates of a particular *fixed* point of interest or the three dimensional equation of a *fixed* line of interest. Finally, it will be shown that a *single* moving camera can successfully perform the same line and point detection accomplished by two cameras by altering the pose of the camera.

The results presented in this work are beneficial to any typical stereovision application because of the computational ease in comparison to other reconstruction techniques for points and lines. But more importantly, this work allows for a single moving camera to perceive three-dimensional position information, which effectively removes the two camera constraint for a stereo vision system. When used with other monocular cues such as texture or color, the work presented in this thesis could be as accurate as binocular stereo vision at interpreting three dimensional information. Thus, this work could potentially increase the three dimensional perception of a robot that normally uses one camera, such as an eye-in-hand robot or a snake like robot. Furthermore, this type of work would bring robots closer to having visual perception similar to the human eye, which can observe depth using just a single eye.

1 INTRODUCTION

1.1 The Limitations of Stereovision

Stereovision is a common computer/machine vision technique used to extract 3-D information from a set of images. These images can be acquired with two or more cameras, or a combination of cameras and IR sensors (such as the X-BOX Kinect) placed in a known pose relative to a world coordinate frame [1]. Once the set of images are obtained, traditional stereovision algorithms are used to determine any desired 3-D characteristics such as the 3-D location of an object of interest [1]. Typical applications use two cameras each with a known pose relative to a world coordinate frame.

In general, stereovision image analysis can be defined as a two part problem. The first is the *correspondence problem*. This involves determining the image point in each image of a set of images that correspond to the same physical point P. We will call this set of image points, N. The second problem is the *reconstruction problem*. Once a set of image points, N, that correspond to point P has been determined, the set of points N is then used to extract three dimensional information about point P [2]. Generally, this is accomplished using a probabilistic method or more commonly Epipolar geometry, which uses triangulation and 2-D information from each camera to determine the coordinates of a fixed point in three dimensional space [1] [3][4][5][6].

Although Stereovision has been used successfully in the field of Robotics for many years, it has been repeatedly shown to have several drawbacks that limit its use during actual applications. First, Stereovision relies on the use of two or more cameras for interpreting three-dimensional information. If one of the cameras is destroyed or damaged during use, the reconstruction problem becomes impossible because the 2-D information from both cameras is no longer available. Second, the use of two or more cameras introduces additional complexities and uncertainties from each camera due to the processing needs of the camera and, the high level of precision necessary for camera calibration and placement. Each camera also introduces an additional level of sensitivity to correspondence errors which can result from the physical limitations of the camera, errors from the Epipolar analysis or any additional image pre-processing required for the application.

But more importantly, stereovision is inherently limited by the baseline distance between the two cameras used. As the point of interest moves further away from the cameras, the depth estimation of the point becomes increasingly inaccurate due to small correspondence and triangulation errors that are compounded over time [7] [8]. In addition, the baseline distance between the cameras becomes unperceivable as the point of interest moves further away. Essentially, this means that the point of interest shows no noticeable change in location in the either image which results in a collapse of the Epipolar analysis altogether [2][1].

Lastly, from a philosophical standpoint it is evident that the field of Robotics is motivated by the need to replace humans for applications no longer desired by humans to perform. Therefore, in order for robots to perform successfully in their application, they must be given the same capabilities of a human being or better. For humans, the eye can perceive depth and other 3-D information without the need of information from the other eye [9]. Humans unconsciously use geometric techniques as well as monocular cues such as texture, color, shading and haze to determine 3-D information all from just a single eye[10][7]. Therefore, the advancement of Robotic vision is dependent on the development of novel solutions that allow robots to perceive 3-D information using a *single* camera, just as a human can perceive 3-D information using a *single* eye [10].

To overcome the shortcomings of Stereovision, much work has been done in the area of *Single Camera Stereovision*, more generally referred to as *Monocular Vision*. Monocular Vision is used to determine depth and other 3-D information using a *single* camera and either a *single* or *multiple* images that are related through some type of correspondence algorithm. Both approaches are complementary and are more effective in determining 3-D information from an environment rather than traditional stereovision. The focus of this thesis work is to present novel techniques in Monocular Vision. Therefore, in this chapter various approaches to depth estimation through Monocular vision will be reviewed to allow for a better understanding of the context for the work presented in this thesis. First, single camera mapping and SLAM techniques will be discussed followed by Omnidirectional and catadioptric vision systems. Finally, single image depth estimation techniques and structure from motion will be discussed. Lastly, the overview and implications of this thesis work will be presented.

1.2 Single Camera SLAM and other Single Camera 3-D Mapping Techniques

Simultaneous Localization and Mapping or SLAM is a widely used technique that allows robots to map an unknown environment and simultaneously track their position without using any previous knowledge of the environment. Typically, SLAM techniques acquire information about the environment using a single sensor or combine information from several sensors such as laser range finders, sonar sensors and cameras. In particular, the use of cameras in SLAM applications has become important because they are compact, noninvasive, ubiquitous and becoming more affordable [11] [12] [13]. The aforementioned limitations of stereovision have been recognized by the SLAM community and as a result, they have proceeded to develop and explore various types of single camera SLAM techniques.

Some of the earliest work was done by Harris and Pike [14] in 1987. Although their work was not officially called SLAM, it was similar because it used images taken in succession from a single camera to build 3-D visual maps. Feature points of interest are extracted and tracked from each image and used along with Kalman filtering to determine their actual physical 3D locations. Their work was successful in achieving accurate 3-D maps and real time implementation. However, serious drawbacks related to their initial assumptions cause concern in regards to the reliability of their technique. For example, the common camera motion was ignored when determining the locations of each of the mapped visual features [13].

In 1988, another basic 3-D mapping method similar to template matching was developed and used with a single camera attached to a mobile robot to identify its location in a room [15]. Vertical edge detection was performed and compared to a known room map of vertical edges that is acquired beforehand. The authors craft their algorithm around the practical assumption of imperfect edge detection and achieve accurate and significant results. However, they fail to address the tediousness of creating and using known vertical edge detection maps. Furthermore, problems from unknown objects or other random scene changes are completely unaddressed.

More recently, Davison and Reid have developed a real time algorithm that can determine the 3-D trajectory of a single mobile camera moving through an unknown environment [13]. They have dubbed their system *MonoSLAM* because it uses a single camera to determine 3-D information. They accomplish this by using fundamental probabilistic SLAM techniques, motion modeling and the measurement and mapping of visual landmarks found on planar surfaces within the environment. MonoSLAM has broad applications in robotics and wearable computing. In particular, Davison and Reid have successfully used MonoSLAM to command a humanoid robot to walk in circles with a high level of accuracy and precision. Other interesting applications include the successful use of MonoSLAM with an automobile driven through an urban environment [16]. However, MonoSLAM is still in its infantile stages and requires further work to deal with issues such as changing lighting, significant occlusions from objects and the ability to operate in larger indoor/outdoor environments [13].

Other related works have utilized Kalman filtering with SLAM to map a 3-D environment. In general, single camera SLAM systems based on Kalman filtering have been successful, but are limited due to the computational complexity of the techniques and inaccuracy due to linearization[13][14].

Indeed, these works reveal that single camera 3-D mapping techniques, specifically SLAM techniques are successful. However, they are complex and must account for unknown scene changes during practical applications.

1.3 Other 3-D Line and 3-D Point Detection Algorithms

The goal of this work is to not only expand on the area of Monocular Vision but to present reconstruction techniques for fixed 3-D lines and fixed 3-D points. Therefore, it is important to discuss the various methods that currently exist for line and point reconstruction. Traditional 3-D point reconstruction algorithms are known as Triangulation and the Trifocal tensor. Triangulation is typically used with two images, while the Trifocal tensor is used with three images [5].

Triangulation works by first solving the correspondence problem using Epipolar geometry. Next the fundamental matrix (also known as the bifocal tensor) or essential matrix is determined and used along with image points from at least two images of the same physical point of interest [5]. The resulting prediction of the physical point of interest is with respect to a world coordinate frame, which is in contrast to the techniques in this thesis which provide 3-D point coordinate estimates relative to the camera frame [17].

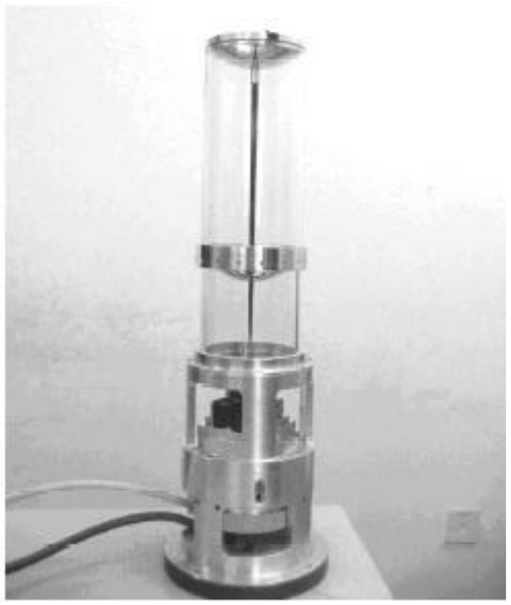
The Trifocal tensor relates correspondence information from three images to create what is known as the trifocal relationship which basically states that the 3-D coordinates of a point can be found by analyzing the relationship between four intersecting planes. The trifocal tensor can also be used to identify the equation of a line using three or more intersecting planes. Typically, the trifocal tensor is used with multiple views for line, point and plane reconstruction.

Additionally, there are many methods in literature that currently exist for point and line detection. This is because point and line detection is an older computer vision problem, and thus many unique solutions have been developed for it. For example, [18] presents a method for 3-D reconstruction of points, planes and lines based on user inputed coplanarity, perpendicularity and parallelism constraints. These techniques were designed for single view and were shown to be successful for reconstruction and calibration purposes. Another interesting example is[19], which presents a novel linear, non-iterative reconstruction technique for points and lines from correspondence measurements subject to noise. Through rigorous experimental results, this method was shown to be useful with one or multiple image viewpoints. Many other solutions exist in literature for point and line detection. They take advantage of a wide range of techniques and assumptions and must be considered carefully before they are used for the application.

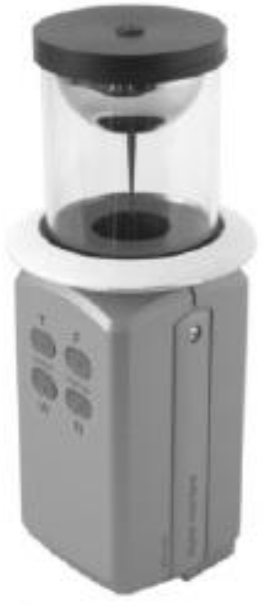
1.4 Omnidirectional Stereovision and Catadioptric Vision Systems

Omnidirectional stereovision is another example of Monocular Vision. Essentially, the goal of this approach is to image an entire 360 degree panorama from a specific viewpoint. This panorama can be used to obtain various types of 3-D information, with the most important being depth estimation [20]. Several interesting methods have been developed and tested over the years. A simple approach uses a single, off-center rotating camera to image a particular environment [21][22]. Other methods are more complicated and require the use of multiple cameras and curved mirrors to obtain 3-D information from a panoramic viewpoint in a single image [21] [23] [24]. The use of cameras and curved mirrors to achieve panoramic images, often referred to as catadioptric imaging systems can seem unnecessary, but they have been shown to be extremely useful [25].

In particular, catadioptric systems have found a place amongst the security systems of most shopping malls and stores. Typically, cameras are used in conjunction with convex or concave mirrors to provide an observer with a larger viewpoint of any area of the store, thus eliminating any blind spots outside a normal camera's field of view that are taken advantage of by criminals [26]. Typical catadioptric vision systems are commercially available and consist of a hyperbolic mirror placed in front of a camera. During use, light is reflected of the hyperbolic mirror and into the camera to generate a wide view round image of the environment, as seen in Figure 1.



(a)



(b)

Figure 1: (a) An experimental Catadioptic Camera System using two hyperbolic mirrors (b) A commercially available Catadioptic Camera system using one mirror **[27][28]**

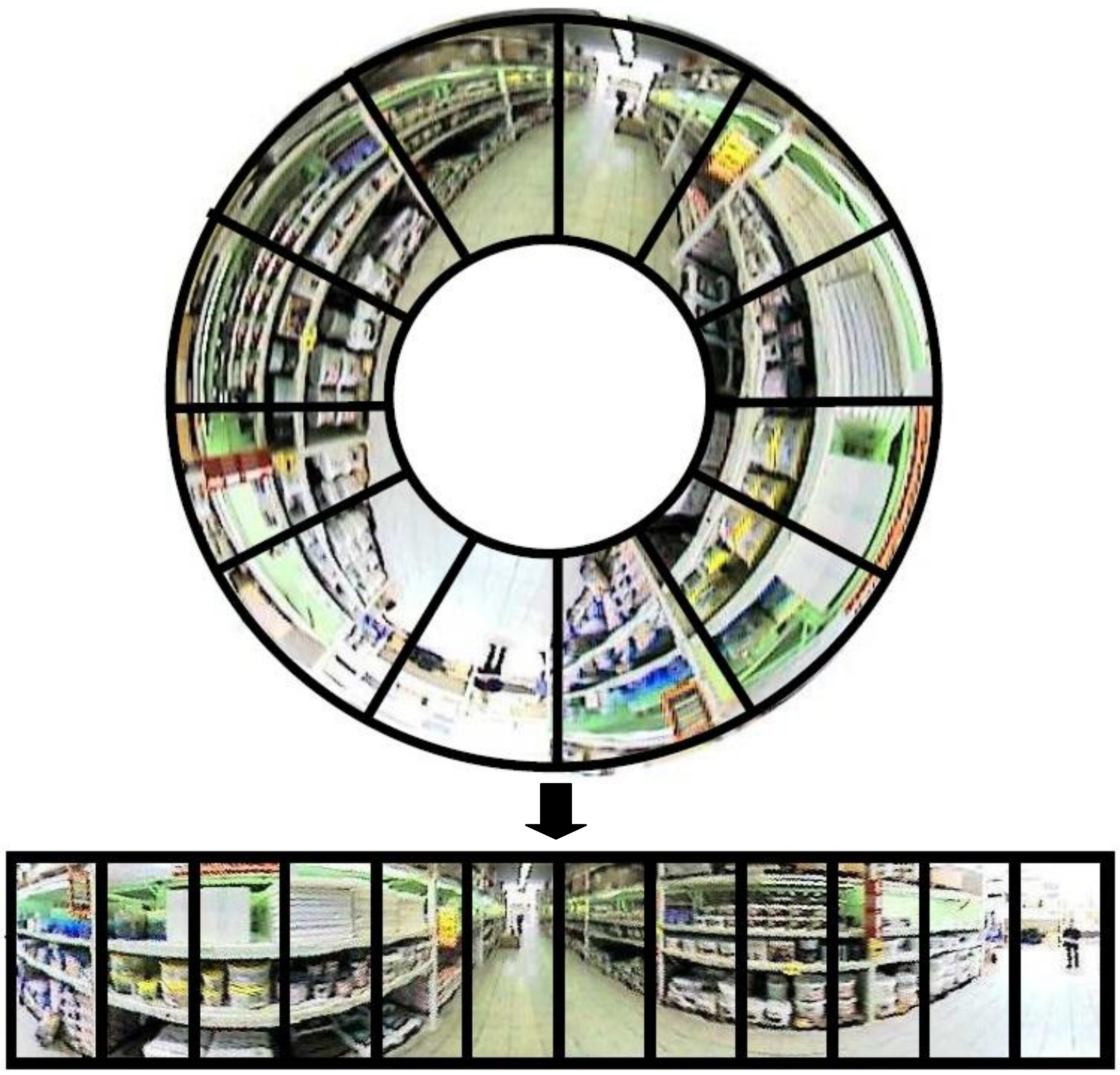


Figure 2: Captured Catadioptric stereo image and the unwrapped panoramic image **[28]**

The round image (as seen in Figure 2) is then unwrapped to obtain panoramic images for analysis [20][28]. Catadioptric imaging systems have also been successfully used on mobile robot platforms for navigation purposes. Novel robots using cameras and hyperbolic mirrors have been designed for assisting humans during tasks such as grocery shopping and as contestants in the RoboCup, which is a Robot soccer competition [20][29][28].

Other interesting design approaches for Omnidirectional Stereovision exist as well. For example, camera lenses, from spheres to bio-inspired design, have been developed and used in conjunction with mirrors to achieve Omnidirectional Stereovision[23] [26]. One attempt requires the camera to image an environment while traversing a spherical track encapsulating the environment of interest [22]. Although accurate depth information was obtained, it is clearly not a practical solution because a spherical track would need to be constructed for each application. This will be very difficult especially if the environment is very large, such as a stadium or a large room. Besides the spherical track attempt, specific omnidirectional cameras and sensors have also been developed to achieve a panoramic view from a single image [22].

In general, it is clear that omnidirectional stereovision can be successfully used in many different ways but, it is limited to a specific application which is of course, attaining a panoramic viewpoint of an environment. If this is the desired application, then implementation of this type of stereovision can be complex because specially designed mirrors must be used and maintained. Furthermore, these mirrors will require accurate and precise placement, which is in addition to camera calibration and placement. However, aside from these drawbacks, Omnidirectional Vision or more specifically catadioptric systems, can provide accurate depth and 3-D scene information for most applications. However, 3-D interpretation for complex and practical applications such as human gesture recognition or facial recognition has yet to be achieved with Omnidirectional or Catadioptric systems.

1.5 Depth Estimation from a Single Image

Depth estimation from a single image has been investigated and accomplished by the use of a probabilistic map along with a supervised learning algorithm. Earlier works in this area were successful but impractical because they relied on unrealistic assumptions about the environment. However, Saxena et al. impressively solves the problem of estimating detailed 3-D structures of an unknown environment using a single still image. Rather than using triangulation techniques as in normal stereovision, a Markov Random Field (MRF) is utilized along with supervised learning techniques to obtain the “depth maps” as shown in Figure 3. In these depth maps, a single color pertains to a specific physical distance from the camera itself. Saxena et al. uses the MRF to model the depth at different resolutions by combining various methods from computer vision such as feature point identification and multiscale representation of images. Furthermore, monocular cues such as haze, color, motion parallax and texture are used along with the MRF to create depths similar to those in Figure 3 [7] [8]. Absolute and relative image features are also taken advantage of to produce accurate depth estimation from a single image.

Saxena et al. applied these techniques to real world applications such as static environments containing trees, buildings or other people. Furthermore, these techniques were successfully applied to a small RC truck to perform obstacle avoidance at high speeds [30]. In addition, these techniques have been combined into free software called Make3D, which is free and can be downloaded from the internet [31]. Make3D allows you to create 3D panoramic models from a single static image as seen in Figure 4.

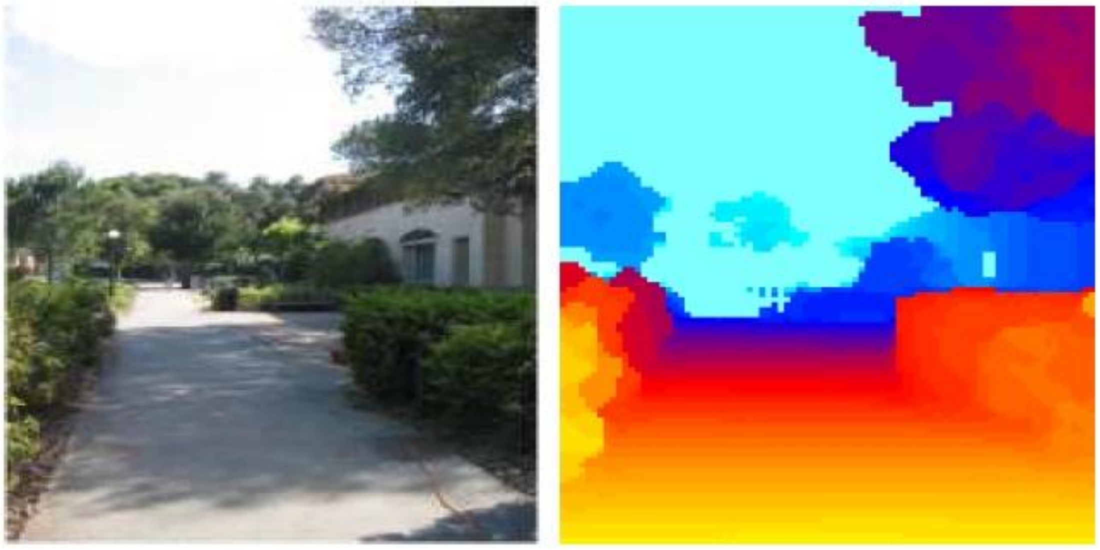


Figure 3: A single still image and the corresponding depth map. Each color corresponds to a different depth **[7]**

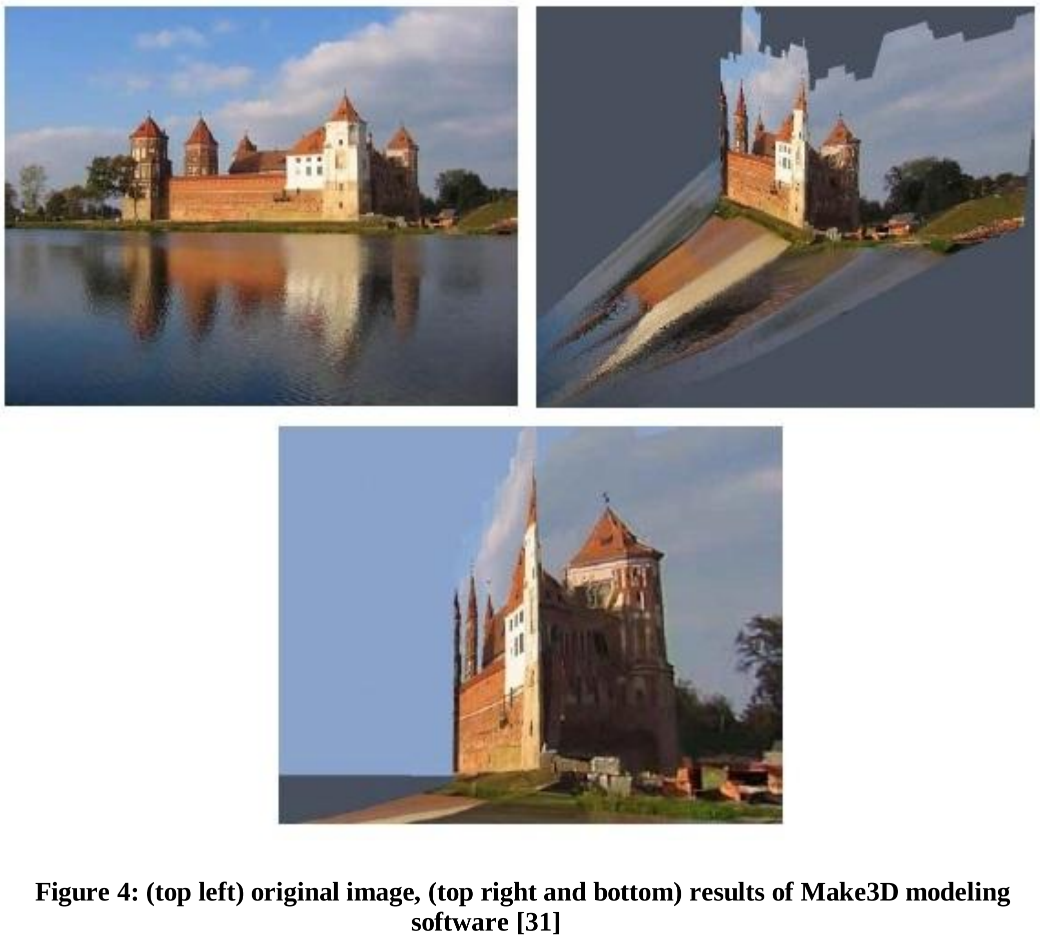


Figure 4: (top left) original image, (top right and bottom) results of Make3D modeling software [31]

Although these techniques are very successful and impressive, they do have issues that must be addressed. In particular, these techniques have difficulty predicting the 3-D information of the environment behind an object without using information about the object itself. Furthermore, depth estimation becomes nearly impossible if the image is largely homogenous such as in the case of a wall of uniform color or even a blue sky. These environments lack the diversity needed for the analysis techniques proposed by Saxena et al. In general, these techniques require a heavy amount of training images and prior knowledge about the environment to estimate depth from a single image. This is largely due to the fact that monocular 3-D reconstruction is an inherently difficult and ambiguous problem [7][32][33].

1.6 Structure from Motion

The area of Structure from Motion (SfM) is another attempt to infer 3-D geometry from 2-D image projections using one or multiple cameras. The 3-D information is traditionally determined by using 2-D projections of the *motion trajectory or motion signals* of an object or environment occurring in 3-D space [34]. In fact, SfM is very similar to the techniques presented in this thesis hence, a thorough discussion of SfM is extremely relevant to this literature review. In general, SfM assumes that either the contents of the environment are moving or the camera itself is moving [35]. Furthermore, another assumption is that there exists a correspondence algorithm that identifies, extracts and labels 2-D image features such as corners, curves and Centroids. These 2-D features are then related to their corresponding instances in each image. The end result of this process is used by the SfM algorithm for determining 3-D information [36]. The reconstruction portion of SfM involves techniques similar to stereo vision. Depending on the number of images used, the camera poses along with Epipolar analysis or affine transformations are used to determine the 3-D information from the set of images. The resulting information is summarized and compiled into motion data for an object of interest within the environment. Clearly, SfM appears to be very similar to traditional stereovision however, SfM requires only one camera while Stereovision requires two cameras.

Several applications exist for SfM, some such as 3D model reconstruction and 3D motion matching, computer animation, camera calibration and 3-D vision for Robotics. More recent work has been done in reconstructing the 3-D trajectory of a moving point from its correspondence from a set of 2-D images, provided that the 3-D spatial pose and time of capture from each camera is known [37]. These researchers were able to track hand movement and other body movement trajectories using a RANSAC correspondence algorithm and SfM techniques during activities such as rock climbing and dancing [37]. Although SfM techniques are very successful they are fundamentally limited by the relationship between the 3-D trajectory of a point and the 3-D trajectory of the center of the cameras used. Furthermore, SfM are computationally intensive and subject to noise, which is typical of most 3-D Computer Vision algorithms.

1.7 Conclusions

Clearly the breadth of existing Monocular Vision research is large and varied. Many different applications have been explored in an effort to showcase the practicality and usefulness of Monocular Vision. Individually, these techniques were shown to be very precise and accurate. However, it became apparent either directly or indirectly, that the best vision system for a robot is one that combines both Monocular and Stereovision techniques. This is because the vision system inherent in human beings operates exactly the same way. The human eye and brain together take advantage of Monocular and Stereoscopic cues to interpret the scene surrounding the human being. Therefore, while this thesis will present experimental data to support the techniques presented in this work, it must be noted that the work presented here will be more effective when combined with other 3-D scene interpretation techniques.

2 RESEARCH OVERVIEW

This thesis attempts to introduce three novel Monocular Vision techniques that can be used with multiple cameras or a *single* moving camera. These techniques do not address the problem of correspondence; in fact these techniques assume that the correspondence problem is solved beforehand and that the results have correspondence error. The new approaches presented in this paper are fresh solutions to the reconstruction problem. Two of the techniques presented are for detecting the location of a point and one for detecting a line expressed in a 3-D coordinate system. The methods presented are unique because of their simplicity and because they do not require the cameras to be placed in specific locations, orientations or have specific alignments. On the contrary, the techniques presented in this paper will show that the two cameras used can assume almost any relative pose provided that the object of interest is within their field of view.

The relative pose of the cameras at a given instant in time, along with basic equations from the perspective image model will be used to form a system of equations that will reveal the 3-D coordinates of a particular *fixed* point of interest or the three dimensional equation of a *fixed* line of interest. Finally, it will be shown that a *single* moving camera can successfully perform the same line and point detection accomplished by two cameras by altering the pose of the camera.

The results presented in this work are beneficial to any typical stereovision application because of the computational ease in comparison to Epipolar geometry.

But more importantly, this work allows for a single moving camera to perceive 3-D position information, which effectively removes the two camera constraint for a stereo vision system. When used with other monocular cues such as texture or color, the work presented in this paper could be as accurate as binocular stereo vision or human vision at interpreting 3-D information [6]. Thus, this work could potentially increase the 3-D perception of a robot that normally uses one camera, such as an eye-in-hand robot or a snake like robot.

In the next chapters, the basic perspective imaging equations will be discussed followed by homogeneous transformation equations. Later these equations will be combined to form a system of equations that will yield either the 3-D coordinate of a point or the 3-D equation of a line. A rigorous derivation of the degenerate cases will follow and finally experimental verification of the techniques presented in this work will be provided.

3 FIXED 3-D POINT RECONSTRUCTION METHOD 1

3.1 Perspective Imaging Equations

The camera pinhole model defines the relationship between the image coordinates and 3-D coordinates of a point **p** as follows,

$$\frac{f}{z} = \frac{u}{x} \qquad (1) \qquad\qquad \frac{f}{z} = \frac{v}{y} \qquad (2)$$

where *x*, *y* and *z* are the 3-D coordinates of point **p** relative to the optical center of the camera, *u* and *v* are the image coordinates of point **p** relative to the image principal point and f is the focal parameter for the camera. The focal parameter is defined as the focal length divided by the pixel length in either the *x* or *y* direction. For our purposes we will assume that the focal parameter is the same for both *x* and *y* directions [1] [2].

3.2 Homogenous Transformations Between Coordinate Systems

A point **p** can be expressed in different Cartesian coordinate reference frames (CRF) using a homogenous transformation. If point **p** is represented in the *i*-th CRF by

$$\mathbf{p}^i = \begin{bmatrix} p_x^i & p_y^i & p_z^i \end{bmatrix}$$

then its representation $\mathbf{p}^j$ relative to the *j*-th CRF is related to $\mathbf{p}^i$ through the following equation,

$$\mathbf{p}^i = \mathbf{R}_j^i \mathbf{p}^j + \mathbf{d}^i \qquad (3)$$

where $\mathbf{R}_j^i$ is a 3 x 3 rotation matrix ($R^T R = I$) whose columns are the 3x1 vector representation of the *j*-th CRF basis unit vectors relative to the to *i*-th CRF, and the *displacement* vector

$$\mathbf{d}^i = \begin{bmatrix} d_x^i & d_y^i & d_z^i \end{bmatrix} \qquad (4)$$

is the origin of the *j*-th CRF relative to the *i*-th CRF. In the next sections, the indices of the displacement vector and rotation matrix (i.e., $\mathbf{d}=\mathbf{d}^i$ and $\mathbf{R}= \mathbf{R}_j^i$) will be omitted when dealing with only two CRFs [38].

3.3 Combining Homogenous Equations and Perspective Equations

Suppose two cameras, camera 1 and camera 2, each with their own CRF are viewing a fixed point **p**. The pose of camera 2 relative to camera 1 is known. The coordinates of point **p** relative to each camera can be solved for by using equations (1) and (2) and the image coordinates, *u* and *v*, of point **p** from each camera. Let (u_i, v_i) denote the image coordinates of **p** relative to camera *i* and define the image vector as

$$\bar{\mathbf{p}}^i = \begin{bmatrix} u_i & v_i & f_i \end{bmatrix}^T \qquad (5)$$

Then (1) and (2) can be used to relate $\mathbf{p}^1$ to the image vector in camera 2:

$$\mathbf{p}^1 = \alpha_2 \mathbf{R}\bar{\mathbf{p}}^2 + \mathbf{d} = \alpha_1 \bar{\mathbf{p}}^1 \qquad (6)$$

where $\alpha_i = p_z^i / f$. (6) can be rearranged as a system of linear equations in terms of $\boldsymbol{\alpha} = [\alpha_1 \ \alpha_2]$ whose solution will yield point $\mathbf{p}$:

$$\underbrace{\begin{bmatrix} \bar{\mathbf{p}}^1 & -\mathbf{R}\bar{\mathbf{p}}^2 \end{bmatrix}}_{\mathbf{A}_p} \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}}_{\boldsymbol{\alpha}} = \mathbf{d} \qquad (7)$$

The unknown vector $\boldsymbol{\alpha}$ can be solved for as follows provided that matrix $\mathbf{A}_p$ is full rank,

$$\begin{aligned} \boldsymbol{\alpha} &= (\mathbf{A}_p^T \mathbf{A}_p)^{-1} \mathbf{A}_p^T \mathbf{d} \\ \mathbf{p}^i &= \alpha_i \bar{\mathbf{p}}^i, \; i = 1,2 \end{aligned} \qquad (8)$$

From equations (6) and (7), it can be seen that matrix $\mathbf{A}_p$ is full rank as long as $\mathbf{p}^1$ and $\mathbf{d}$ are linearly independent or equivalently point $\mathbf{p}$ does not lie on the line joining the origins of the cameras' CRF. Once these conditions are satisfied, the 3-D coordinates of a fixed point $\mathbf{p}$ can be determined using only, the image coordinates of point $\mathbf{p}$ from each camera and the relative pose between camera 1 and camera 2. Any pose is permitted provided that the displacement vector between the cameras does not go through point $\mathbf{p}$. A change in orientation of the camera will be insufficient and will result in $\mathbf{A}_p$ being singular (which will be discussed in detail in the next section).

More importantly, we can easily see that a single moving camera can be used with equations (6) and (7) to identify point $\mathbf{p}$. In this case, several sensors must be used to record the relative pose of the camera from t_i *to* $t_{(i+1)}$. Furthermore, the focal parameter must be known and the image coordinates at t_i and $t_{(i+1)}$ must be determined through a correspondence algorithm. For example, a single camera can identify the 3-D coordinates of a point of interest **p** by simply moving towards it such that **p** is not collinear with the camera's CRF. The movement could be in a straight line or a curved path provided that point **p** is within the camera's field of view. This movement would be similar to a snake like robot or a robot with an eye-in-hand camera.

3.4 Experimental Verification

3.4.1 Overview

Experimental verification of (7) was accomplished by identifying a fixed point of interest using various camera poses. As shown in Figure 5 and Figure 7, the camera used was a 640 x 480 resolution camera (f = 525) built into a Hewlett Packard Pavilion DV6T Selection Edition laptop running Windows 7. The point of interest or target, was a black dot drawn on a white index card attached to a standard laboratory test stand. For each test, the Centroid of the target was approximated using Microsoft Paint from two images taken from different randomly chosen camera configurations. The pose of the target itself remained fixed throughout the experiment. Each camera pose was attained by precise translations of the laptop. However, no changes in the orientation of the laptop were utilized. The pixel locations of the Centroid and the relative pose of the laptop camera were then used with (7) to determine the 3-D location of the target.

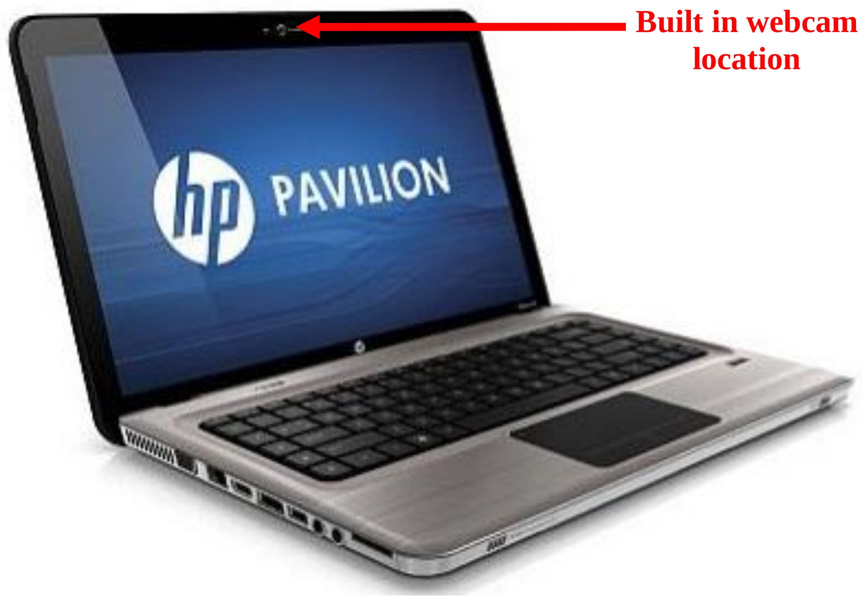


Figure 5: Professional photo of HP DV6T SE Series Laptop used for the experiment **[39]**

3.4.2 Laptop Translation and Pose Estimation Procedures

The experimental setup with an example laptop pose is shown in Figure 7. The precise location of the laptop was determined with the use of four experimental aids. The first aid was a 1/2 inch x 1/2 inch grid poster paper, which was firmly secured and placed underneath the laptop and test stand. A grid system such as this one served as a two dimensional coordinate system that made laptop translation measurements clearer and more defined. The second aid used was standard measuring tape that was used to measure the translation of the laptop during the experiment. The third aid was a foot long ruler that was used to ensure that the laptop screen was kept orthogonal to the horizontal axis. This was done to ensure that the orientation of the laptop remained fixed at all times during the experiment and, that the camera axes were in the same direction as the laptop translation axes, as seen in Figure 8.

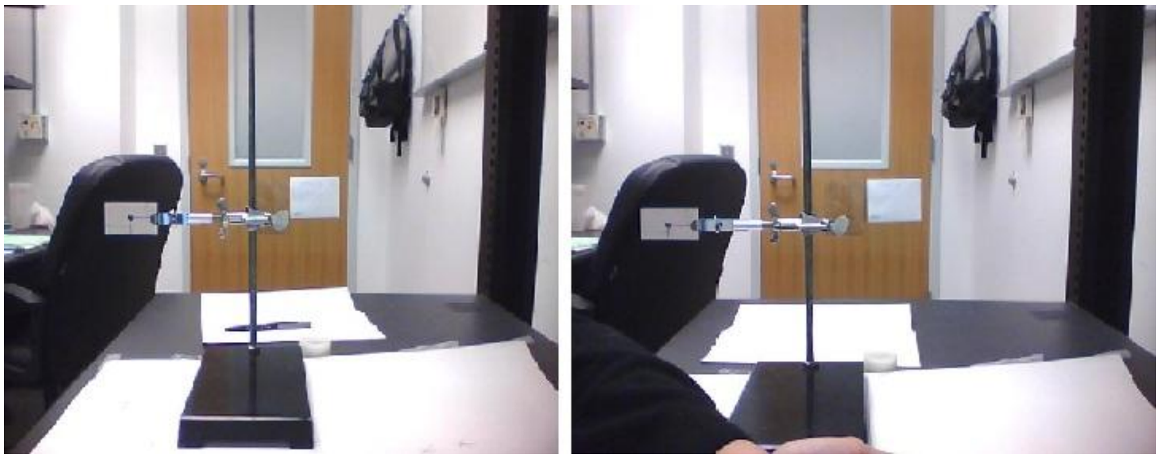


Figure 6: Sample set of images taken. The image on the right was taken after the camera was displaced in the z direction by 2 inches.

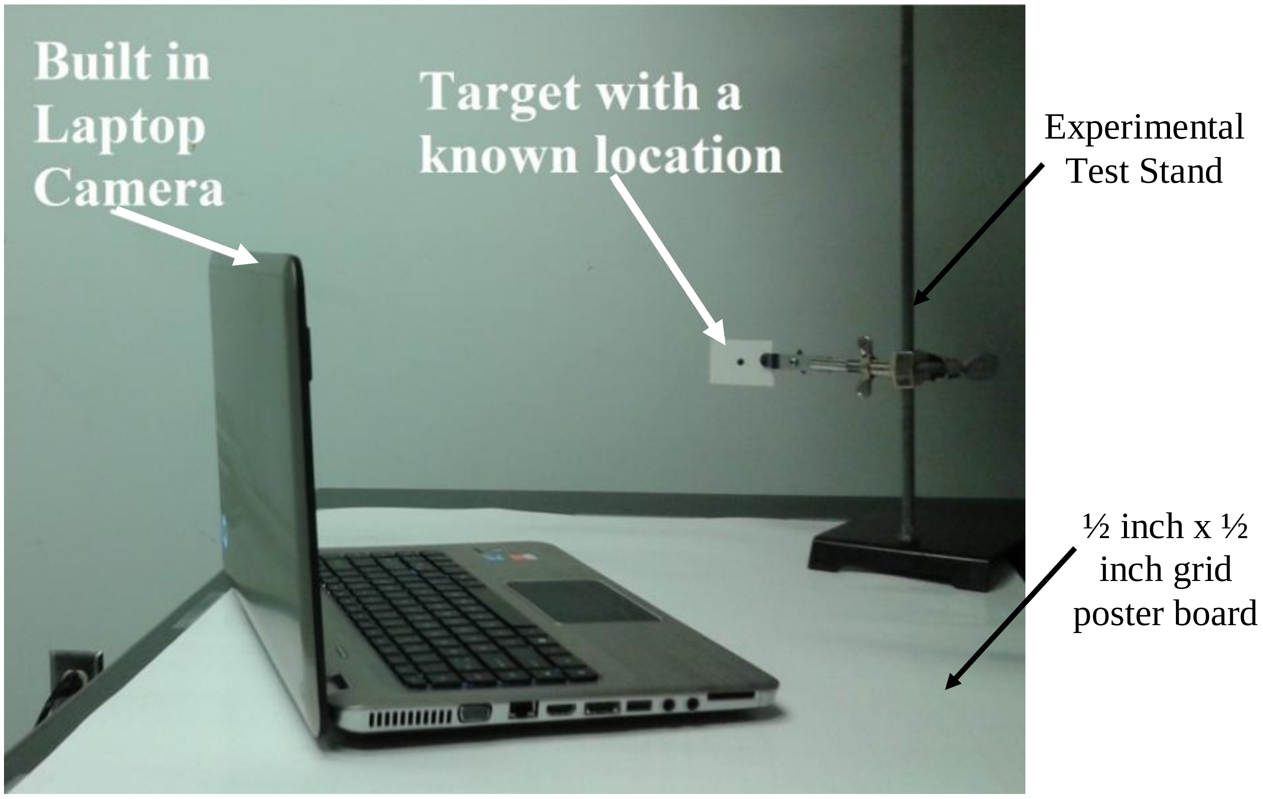

Built in Laptop Camera
Target with a known location
Experimental Test Stand
½ inch x ½ inch grid poster board

Figure 7: An example laptop pose of point detection experimental testing setup

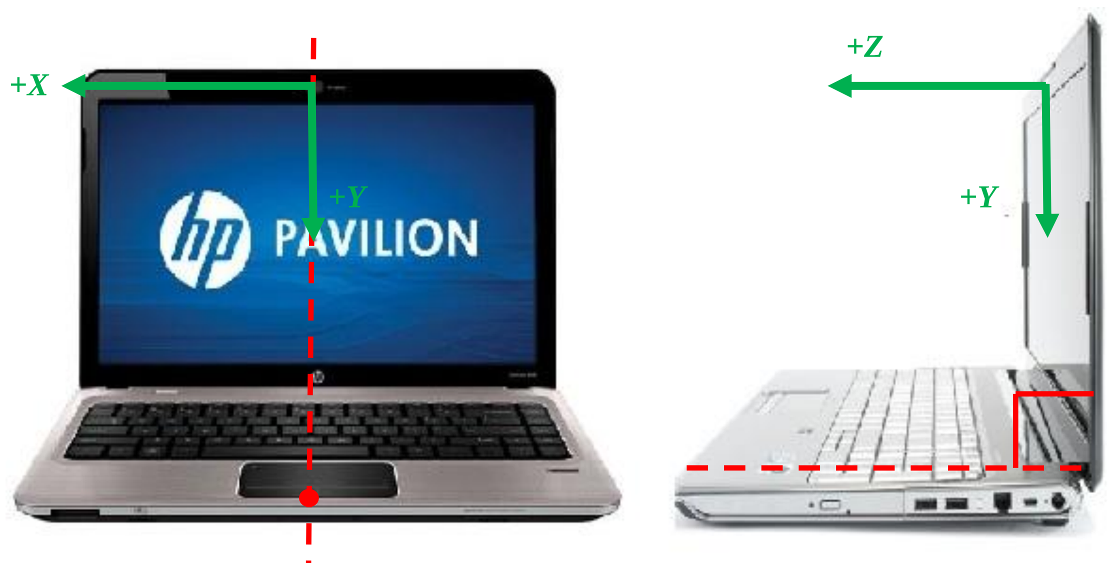


Figure 8: (a) Dotted line is line with camera center, the red dot indicates a reference point of measurement during the experiment. (b) Profile of the laptop. The screen orientation was kept perpendicular to the horizontal at all times. Camera axes and laptop translation axes are in the same direction. **[40]**.

The fourth aid was the reference point on the mouse pad of the laptop. This point, shown as the red dot in Figure 8 was in line with the camera axes and was used as a reference point when measuring laptop displacement during the experiment. Lastly, once both images were taken, Microsoft Paint was used to determine the pixel locations of the Centroid as seen in Figure 9. This was successful because, Microsoft Paint gives you the current pixel coordinates of the cursor. By using this feature in Microsoft Paint and by zooming in to the highest zoom level, an accurate estimation of the target's Centroid was determined.

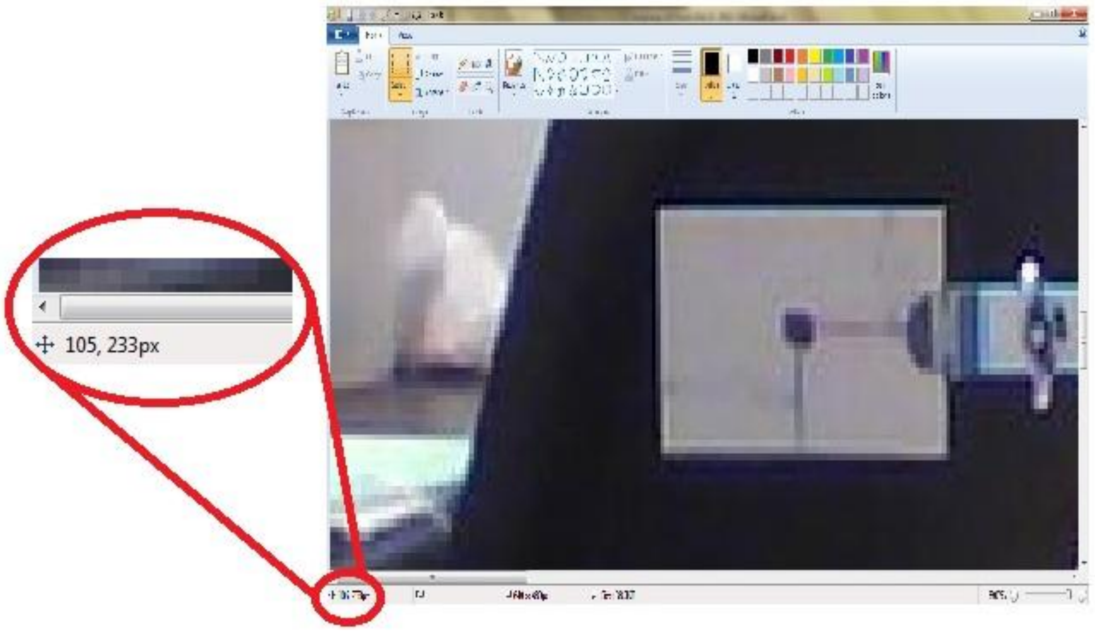


Figure 9: Using Microsoft Paint to determine the pixel coordinates of the Centroid. In this image, maximum zoom has been used. The cursor has been placed over the center of the target, MS Paint displays the pixel coordinates of the cursor in the lower left hand portion of the screen.

The pixel coordinates of the Centroid from each image and the relative pose of the laptop camera were then used with Equation (7) to determine the 3-D location of the target. Approximately fifteen experimental test points and thirty random camera poses (two for each test) were used to validate (7). A flow chart summarizing the experimental procedure has been given in Figure 10. Furthermore, the results of the experiment are shown in Figure 11 in the next subsection.

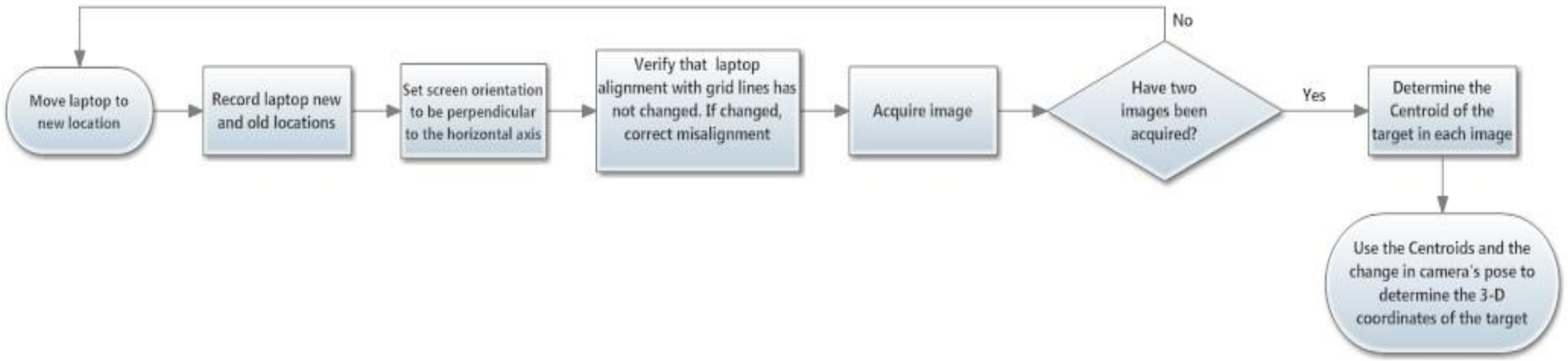

No
Move laptop to new location
Record laptop new and old locations
Set screen orientation to be perpendicular to the horizontal axis
Verify that laptop alignment with grid lines has not changed. If changed, correct misalignment
Acquire image
Have two images been acquired?
Yes
Determine the Centroid of the target in each image
Use the Centroids and the change in camera's pose to determine the 3-D coordinates of the target

Figure 10: Experimental procedure flowchart for testing method 1. This flowchart is assuming that the test stand and target have been setup and will remain fixed throughout the experiment.

3.4.3 Experimental Results

Figure 11 displays the target location at the origin and each additional point represents a predicted location from camera 1 or camera 2 with respect to the target location. From Figure 11, it is evident that the final estimated position of the target from each camera was very accurate, with the results evenly distributed around the target location. The resulting error for each coordinate was on average less than one inch, which is acceptable for this type of basic experiment. The error present in the results is largely due to correspondence error. The pixel locations were chosen manually and are subject to human interpretation of the images containing the target. A small mistake in determining the location of same pixel in each image would result in errors in the final estimation.

Furthermore, the use of higher resolution cameras with a larger focal parameter would allow for more accurate predictions over a larger distance from the camera. Specifically, a higher resolution camera would yield a sharper image with smaller divisions that would allow for a better estimate for the Centroid of the target. Aside from some minor experimental errors, in the end it was shown that by simply moving the camera, (7) can be used to accurately determine the 3-D location of a point of interest relative to the camera.

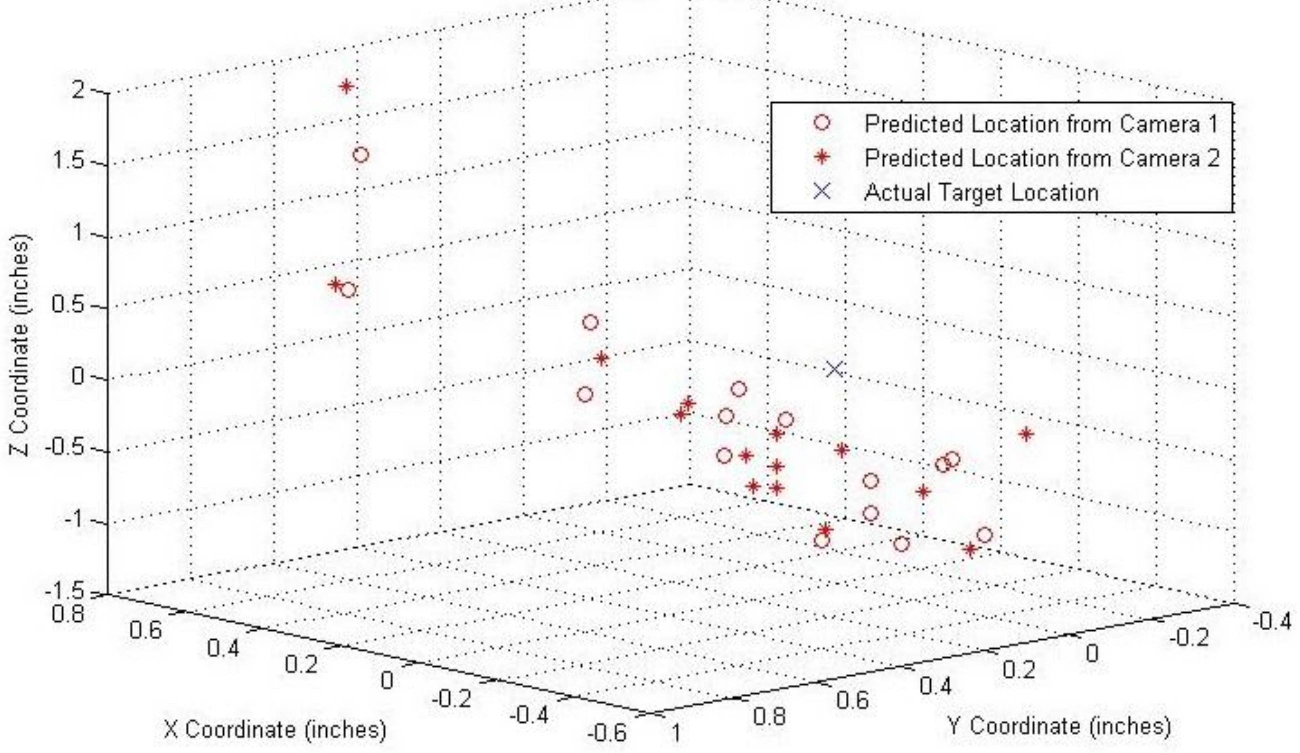


Figure 11: Predicted target locations relative to the actual target location (at the origin) for both cameras

4 FIXED 3-D LINE RECONSTRUCTION

4.1 Using Intersecting Planes To Define a Line

A line L in a 3-D space can be defined using a point $\mathbf{p}_0 \in L$ and unit vector $\mathbf{u}$ representing the line direction. Any $\mathbf{p} \in L$ can be expressed as,

$$\mathbf{p} = \mathbf{p_0} + \alpha\mathbf{u}, \forall \alpha \in \mathbf{R} \qquad (8)$$

The image $\tilde{L}$ of line L in the *uv*-plane of a camera, assuming the line does not intersect the camera center, can be described using its normal vector $\mathbf{n}$=[-sinθ cosθ]^T and point $\tilde{\mathbf{p}}_0 \in \tilde{L}$ where θ is the orientation of $\tilde{L}$ as shown in Figure 12,

$$\mathbf{n}^T(\tilde{\mathbf{p}}_0 - \tilde{\mathbf{p}}) = u\sin\theta - v\cos\theta + \tilde{\mathbf{n}}^T\tilde{\mathbf{p}}_0 = 0 \qquad (9)$$

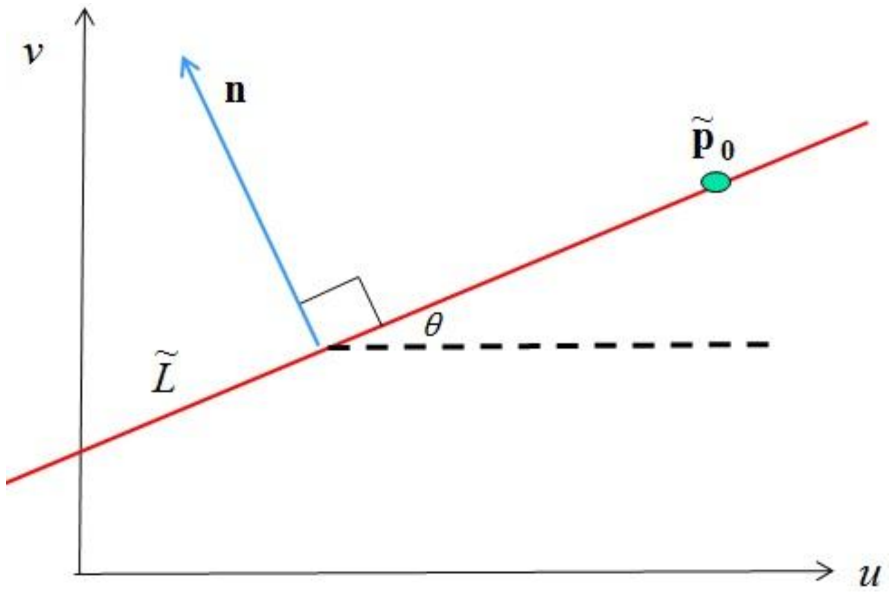


Figure 12: A point p0 on line L in as viewed by a camera

If we substitute the perspective equations (1) and (2) into equation (9) we arrive at the plane equation containing point $\mathbf{p}_i$ and line L_i imaged in the i^{th} camera,

$$(f\sin\theta)x - (f\cos\theta)y + (\tilde{\mathbf{n}}^T\tilde{\mathbf{p}}_0)z = 0 \qquad (10)$$

Or equivalently $\mathbf{n}^T\mathbf{p}=0$ for all $\mathbf{p}\in L$ where

$$\mathbf{n} = \begin{bmatrix} f\sin\theta & -f\cos\theta & \tilde{\mathbf{n}}^T\tilde{\mathbf{p}}_0 \end{bmatrix}^T \qquad (11)$$

Note that the normal vector $\mathbf{n}$ in (11) is perpendicular to any $\mathbf{p}\in L$ hence by (8), $\mathbf{n}^T\mathbf{u}=0$ and $\mathbf{p}_0\times\mathbf{u}=\beta\mathbf{n}$ for some scalar $\beta\neq 0$ or equivalently the triple cross product

$$\mathbf{n} \times (\mathbf{p_0} \times \mathbf{u}) = 0 \qquad (12)$$

If two cameras are now used, we can write plane equations using (10) of the same imaged line with respect to each camera. Next, we can write the plane equation in camera 1 with respect to camera 2 using the coordinate transformation described in (2),

$$\begin{aligned} &\mathbf{n}_2^T \mathbf{p}^2 = 0 \\ &\mathbf{n}_1^T \mathbf{p}^1 = \mathbf{n}_1^T \mathbf{R} \mathbf{p}^2 + \mathbf{n}_1^T \mathbf{d} = 0 \end{aligned} \qquad (13)$$

where $\mathbf{n}_i$ is the normal vector in (11) resulting from the *i*-th image. Using (8) in (13) also implies that $\mathbf{n_2}^T\mathbf{u_2} = \mathbf{n_1}^T\mathbf{R}\mathbf{u_2} = 0$ or equivalently,

$$\mathbf{u}_2 = \frac{\mathbf{n}_2 \times \mathbf{n}_1^2}{\left\| \mathbf{n}_2 \times \mathbf{n}_1^2 \right\|} \qquad (14)$$

provided that $n_2xn_1^2 \neq 0$ where $\mathbf{u}_i$ is the vector representation of unit vector u relative to the *i*-th CRF and $\mathbf{n}_1^2 = \mathbf{R}^T\mathbf{n}_1$. Point $\mathbf{p}_0$ closest to the origin satisfies (8) as well as resulting in the following system of linear equations,

$$\underbrace{\begin{bmatrix} \mathbf{n}_2^T \\ \mathbf{n}_1^T \mathbf{R} \\ \mathbf{u}_2^T \end{bmatrix}}_{\mathbf{A}_0} \underbrace{\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}}_{\mathbf{p}_0^2} = \underbrace{\begin{bmatrix} 0 \\ -\mathbf{n}_1^T \mathbf{d} \\ 0 \end{bmatrix}}_{\mathbf{b}_0} \qquad (15)$$

Solving equation (15) will yield the vector equation of the line L containing point $\mathbf{p}_0$ with respect to the second camera. Note that the matrix $\mathbf{A}_0$ is nonsingular provided that $\mathbf{n}_2 x \mathbf{n}_1^2 \neq 0$ since $|\mathbf{A}_0| = \mathbf{u}_2^T(\mathbf{n}_2 \times \mathbf{n}_1^2) = \|\mathbf{n}_2 \times \mathbf{n}_1^2\|$.

4.2 Singularity Analysis

In the preceding section the necessary and sufficient conditions for singularity of the line identification problem was derived in terms of the normal vectors rising from each image. To gain more insight into the singularity condition, a more explicit condition in terms of line L and the locations of the two cameras will be derived. To this end let point $\mathbf{p_0}$ be an arbitrary point on line L. Throughout the paper we assume that line L contains neither of the two cameras' CRF origins so that $\mathbf{p} x \mathbf{u} \neq 0$, $\forall p \in L$, where $\mathbf{u}$ is the line's directional unit vector. Applying the coordinate transformation equation (3) to $\mathbf{p_0}^i$, i=1,2 yields the following,

$$\mathbf{p}_0^1 = \mathbf{R}\mathbf{p}_0^2 + \mathbf{d}$$

and evaluating its cross product with $\mathbf{u}_1$ from the right and $\mathbf{n}_1$ (as previously defined) from the left using (12) we arrive at,

$$\begin{gathered} \mathbf{n}_1 \times (\mathbf{p}_0^1 \times \mathbf{u}_1) = \\ \mathbf{n}_1 \times (\tilde{\mathbf{p}}_0^1 \times \mathbf{u}_1) + \mathbf{n}_1 \times (\mathbf{d} \times \mathbf{u}_1) = 0 \end{gathered} \qquad (16)$$

where $\tilde{\mathbf{p}}_0^1 = \mathbf{R}\mathbf{p}_0^2$. Using $\mathbf{u}_1 = \mathbf{R}\mathbf{u}_2$ and $\mathbf{p}_0^2 \times \mathbf{u}_2 = \beta \mathbf{n}_2$ for some scalar $\beta \neq 0$, and the identity $\mathbf{Ra} x \mathbf{Rb} = \mathbf{R}(\mathbf{a} x b)$ the preceding equation yields,

$$\beta\mathbf{R}\left(\mathbf{n}_1^2 \times \mathbf{n}_2\right) + \mathbf{n}_1 \times (\mathbf{d} \times \mathbf{u}_1) = 0 \qquad (17)$$

By the triple cross product identity ax(bxc)=b(a^Tc)-c(a^Tb) and $\mathbf{n_1}^T\mathbf{u_1}$=0, (17) is equivalent to

$$\beta\left(\mathbf{n}_1^2 \times \mathbf{n}_2\right) = (\mathbf{n}_1^T\mathbf{d})\mathbf{u}_2 \qquad \textbf{(18)}$$

Thus $\mathbf{n}_2 \times \mathbf{n}_1^2 = 0$ if and only if $\mathbf{n}_1^T\mathbf{d} = 0$. In summary, *the line identification problem is singular if and only if line L and the line joining the origins of the cameras' CRFs are coplanar.*

5 FIXED 3-D POINT RECONSTRUCTION METHOD 2

5.1 Adding a third plane

The results of the line identification presented in the preceding section can be extended so that 3-D points can be identifying using three intersecting planes. If a third plane is included in (16), then the 3-D coordinates of a fixed point **p** can be determined. This third plane could be a plane that describes the ground or some other bounding surface that includes point **p**. Letting vector $\mathbf{n}_3$ represent the normal vector to the aforementioned plane containing **p** and $h \in \mathbf{R}$, then (18) combined with $\mathbf{n}_3^T\mathbf{p}^2 = h$ can be presented in equation (19).

$$\underbrace{\begin{bmatrix} \mathbf{n}_2^T \\ \mathbf{n}_1^T\mathbf{R} \\ \mathbf{n}_3^T \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x \\ y \\ z \end{bmatrix}}_{\mathbf{p}^2} = \underbrace{\begin{bmatrix} 0 \\ -\mathbf{n}_1^T\mathbf{d} \\ h \end{bmatrix}}_{\mathbf{b}} \qquad \textbf{(19)}$$

A simple case where the third plane represents a flat ground surface was used in (19). In this case, $\mathbf{n}_3$=[0 1 0] and h is the constant height of the ground relative to the pose of camera 2. Equation (19) has a unique solution $\mathbf{p}^2=\mathbf{A}^{-1}\mathbf{b}$ provided that matrix **A** is nonsingular.

The determinant of matrix **A** similarly to that is given by

$$|\mathbf{A}| = \mathbf{n}_3^T(\mathbf{n}_2 \times \mathbf{n}_1^2) = \|\mathbf{n}_2 \times \mathbf{n}_1^2\|\mathbf{n}_3^T\mathbf{u}_2$$

Thus |A|=0 if and only if $\mathbf{n}_2 \times \mathbf{n}_1^2 = 0$ or equivalently $\mathbf{n}_1^T\mathbf{d} = 0$ or $\mathbf{n}_3^T\mathbf{u}_2 = 0$. In other words, *|**A**|=0 if and only if line L and the line joining the origins of the cameras' CRFs are coplanar or line L is parallel to the third plane.*

5.2 Experimental Verification Through Finger Gesturing

5.2.1 Overview

An increasingly popular application of stereovision is the identification of human movement. In general, movement is a versatile form of communication that can be used to send all types of information [41] [42] [43]. For practical purposes, human movement or gesturing can be interpreted for robot control or for entertainment purposes such as with the XBOX Kinect. Therefore, verification of the techniques presented in this thesis was attempted through gesturing to illustrate the advantages of the techniques in this thesis and, to highlight their applicability to current and popular stereovision trends. In our case, a novel MATLAB GUI was developed to identify the 3-D point on the floor that a static gloved finger was pointing towards, as seen in Figure 13 and Figure 14.

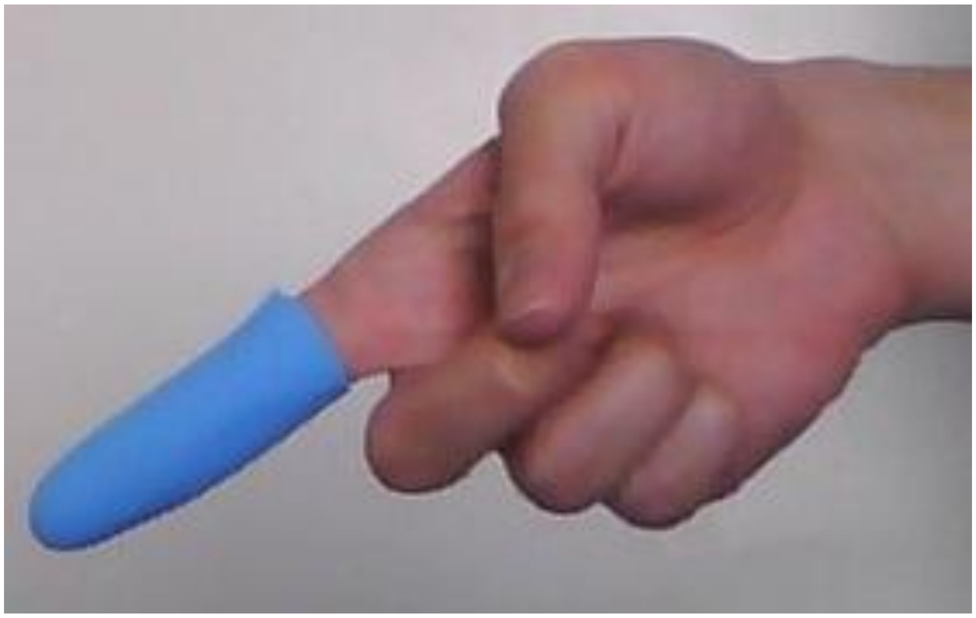
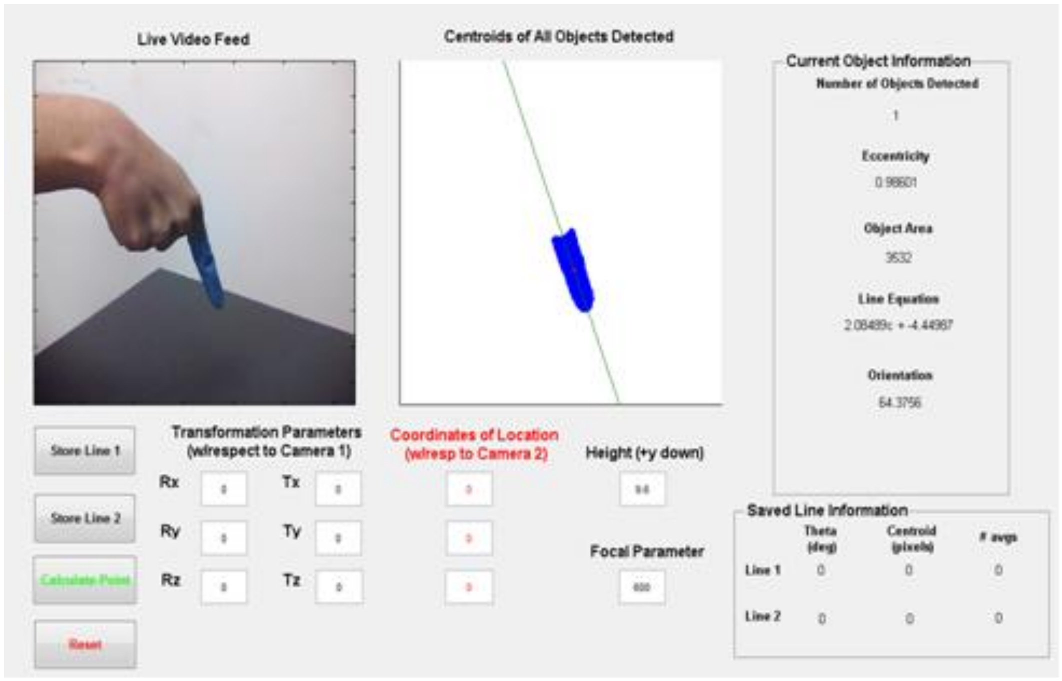


Figure 13: Finger glove used for finger gesturing experiment. The finger glove was cut from a Clorox cleaning glove purchased at a Target store.



Transformation Parameters (w/respect to Camera 1)

Rx 0 Tx 0
Ry 0 Ty 0
Rz 0 Tz 0

Coordinates of Location (w/resp to Camera 2)

0
0
0

Height (+y down) 9.6

Focal Parameter 600

Current Object Information

Number of Objects Detected
1

Eccentricity
0.98601

Object Area
3532

Line Equation
2.08489x + -4.44987

Orientation
64.3756

Saved Line Information

	Theta (deg)	Centroid (pixels)	# avgs
Line 1	0	0	0
Line 2	0	0	0

Figure 14: MATLAB GUI developed for finger detection application

In particular, a *single* camera was used to determine the 3-D location of the point of interest. First, a color filter and area filter algorithm were used to isolate objects resembling a static gloved pointing finger. The gloved finger was assumed to have an elliptical shape, and as such, properties of an ellipse, such as eccentricity, were used to isolate the gloved finger. Next, five images were taken in succession and for each image, the orientation and Centroid of the finger (assumed to be on the line defining the finger) was found. These results were averaged to yield the Centroid and orientation of the finger for a distinct camera pose. This process was repeated for another distinct camera pose. This information was used in conjunction with (9) to calculate the 3-D point of interest with respect to the second camera position.

The finger glove was kept in a fixed location by attaching it to a test stand, while thirty random camera poses were used (fifteen trials). The camera used was the same as described in Chapter 3.4 and as seen in Figure 5. As with the experimental results in Chapter 3.4, the pose of the laptop was altered through translation only. The results of the experiment are shown in Figure 21. In the next few subsections, the design process behind the GUI will be outlined and discussed followed by a discussion of the experimental results.

5.2.2 GUI Design Overview

The finger gesturing experiment required the development of a GUI with the following characteristics:

- Ability to detect a gloved finger in any environment (i.e. regardless of lighting, other objects in the environment, etc.)
- Able to calculate and store information related to the gloved finger
- Manipulate stored information through averaging or other techniques
- Accurately calculate the 3-D location of a point using the stored information from the detected gloved finger.

This was accomplished by first using a color filter to identify all objects that have a similar color as the glove itself and by approximating the finger as an elliptical object. Lastly, the orientation and Centroid of the finger were acquired from two images and used with (**9**) to determine the 3-D point of interest.

5.2.3 Object Area Filter and Color Filter Overview

The color filter used was a standard normalized RGB filter. First, the RGB values for each pixel in the image matrix were found and normalized with respect to the magnitude of the RGB vector. Next, pixels within the image matrix that did not fall within the following normalized RGB parameters were removed:

$$R_{\text{normalized}} > 0.45$$

$$G_{\text{normalized}} < 0.40$$

$$B_{\text{normalized}} < 0.60$$

These parameters were found experimentally using various photos of the finger glove in various lighting conditions. The above parameters represent an average approximation of the RGB values for a pixel containing a portion of the finger glove. Overall, this scheme for color filtering was very fast and accurate.

The resulting image matrix was then converted to black and white and MATLAB was used to perform object detection. Next, a simple object area filter was used to remove objects that were less than 200 pixels^2. This was done to limit the focus of the GUI to objects within a specific distance of the camera itself and to reduce random background noise.

5.2.3.1 Elliptical Approximation

A profile view of the finger, as seen in Figure 15, can be viewed as an ellipse existing in a 2-D environment. This approximation not only matches the shape of the finger well, but it also takes advantage of the many image processing tools in MATLAB which makes a similar approximation to any object detected.

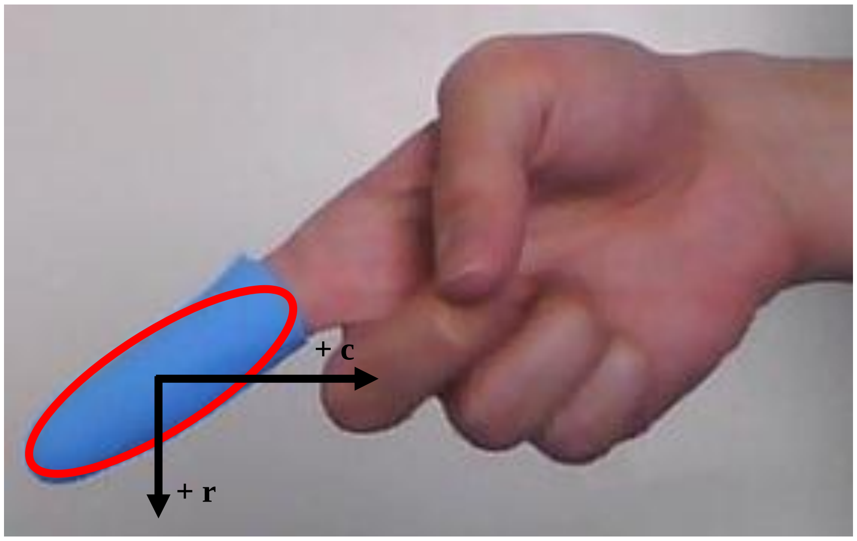


Figure 15: Finger glove approximated as a 2-D ellipse

For example, MATLAB's image processing toolbox returns the eccentricity and orientation of an ellipse with the same second polar moment of inertia of the object of interest. Thus, once the color and area filter were applied, the object eccentricity filter was used to identify objects that were elliptical in shape. A perfect eccentricity has a value of 0, which is that of a circle. Therefore, the eccentricity of an object is a non-dimensional measure of an object's shape relative to a circle. Specifically, ellipses have an eccentricity that is greater than zero but less than 1 [44]. For the finger glove in question, experimental testing revealed that the eccentricity of the glove was larger than 0.86 but less than 1 during any given experiment. Therefore, the GUI was designed to identify objects with an eccentricity greater than 0.86 but less than 1.

In addition to the above filters, a "π filter" was used on the remaining objects to isolate the desired elliptical object. This was accomplished by using the area of an ellipse which can be written as,

$$A = \pi ab \qquad \textbf{(20)}$$

where *a* and *b* are the semi- major and semi-minor axes of the ellipse. For each object, MATLAB calculates the object area, major and minor axes. This information was used along with (20) to experimentally estimate the value of π. Since the area of the finger glove is not exactly an ellipse, the calculated value of π will be close to the known value of 3.14. In our case, it was determined that for the finger glove π was estimated to be between 2.90 and 3.08. Therefore, objects with geometric characteristics that do not yield a value of π within this range were ignored.

5.2.3.2 GUI Gloved Finger Filtering Summary

The combination of normalized color, area, eccentricity and π filters was successfully used to identify a gloved finger in any 3-D environment (aside from an environment absent from lighting of course!). The filtering process has been summarized in the flow chart shown in Figure 16. The final algorithm is essentially a loop that continues as long as the program is running.

5.2.4 GUI Interface Design Summary

The GUI interface was designed such that the user could see the live video feed from the camera and the final interpreted result after the filtering algorithm was finished. Furthermore, the GUI was designed so that the user could store line information from two different camera poses and average them if needed. This stored information could then be used by the user to calculate the intersection point of the line with the ground. The ground itself was also a parameter that could be entered by the user. Figure 17 depicts the interface in detail.

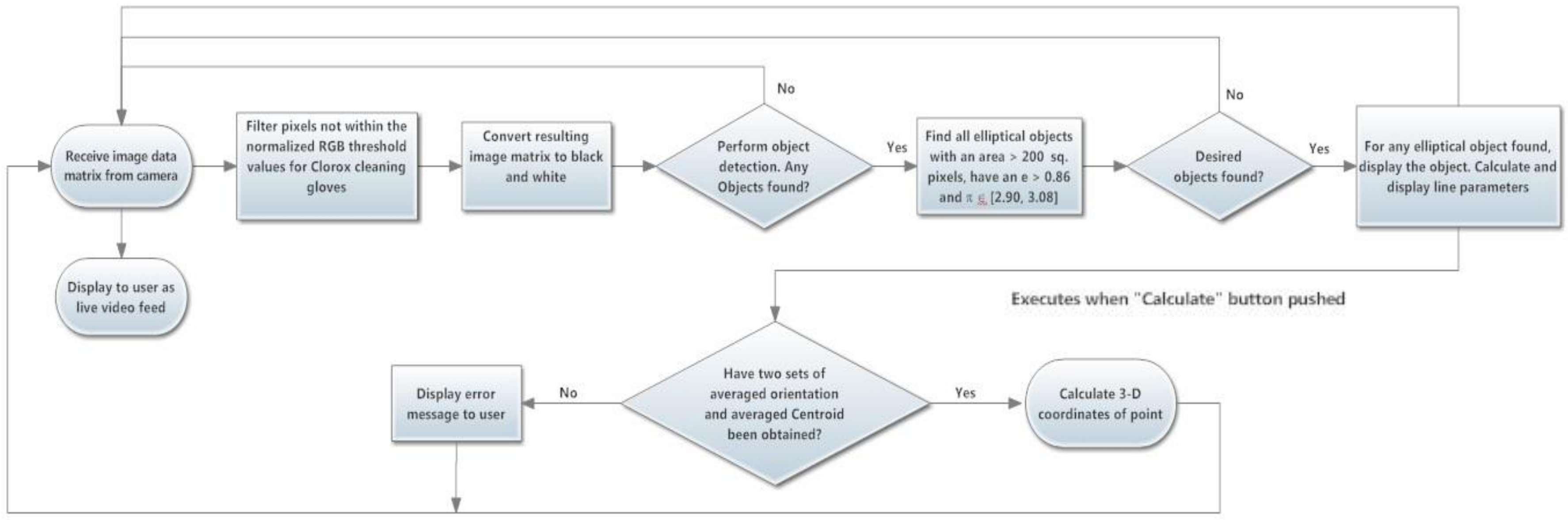

Receive image data matrix from camera
Filter pixels not within the normalized RGB threshold values for Clorox cleaning gloves
Convert resulting image matrix to black and white
Perform object detection. Any Objects found?
No
Yes
Find all elliptical objects with an area > 200 sq. pixels, have an e > 0.86 and π ∈ [2.90, 3.08]
Desired objects found?
No
Yes
For any elliptical object found, display the object. Calculate and display line parameters
Display to user as live video feed
Executes when "Calculate" button pushed
Have two sets of averaged orientation and averaged Centroid been obtained?
No
Yes
Display error message to user
Calculate 3-D coordinates of point

Figure 16: Flow chart of filtering algorithm used in MATLAB GUI for gloved finger detection

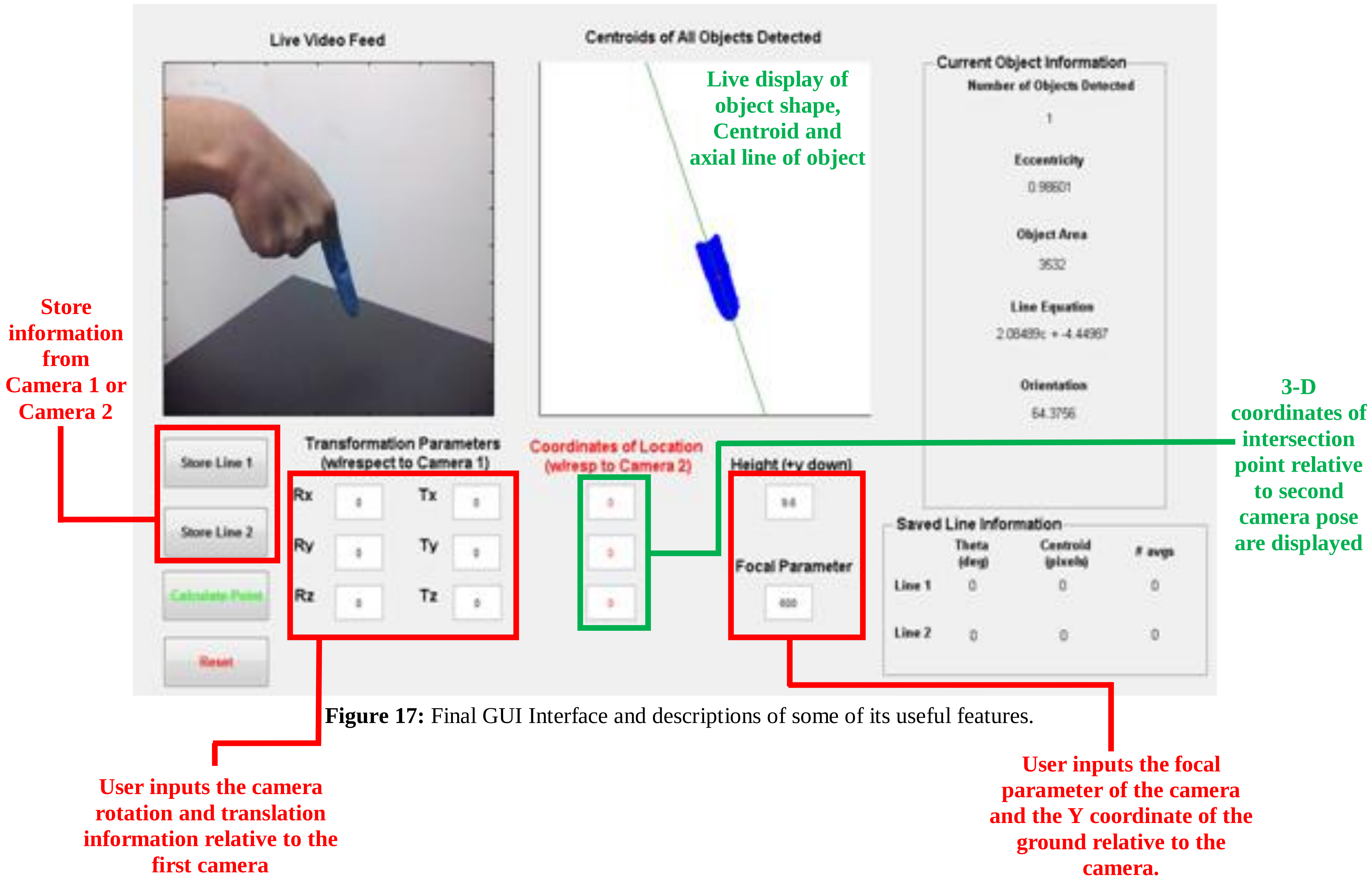

Store information from Camera 1 or Camera 2
Live Video Feed
Centroids of All Objects Detected
Live display of object shape, Centroid and axial line of object
Current Object Information
Number of Objects Detected
1
Eccentricity
0.98601
Object Area
3632
Line Equation
2.08489x + -4.44987
Orientation
64.3756
3-D coordinates of intersection point relative to second camera pose are displayed
Store Line 1
Store Line 2
Calculate Point
Reset
Transformation Parameters (w/respect to Camera 1)
Rx
Ry
Rz
Tx
Ty
Tz
Coordinates of Location (w/resp to Camera 2)
Height (+y down)
Focal Parameter
Saved Line Information
Theta (deg)
Centroid (pixels)
# avgs
Line 1
Line 2
User inputs the camera rotation and translation information relative to the first camera
User inputs the focal parameter of the camera and the Y coordinate of the ground relative to the camera.

Figure 17: Final GUI Interface and descriptions of some of its useful features.

5.2.5 Experimental Procedure

This experiment was carried out similarly to the experiment for point detection method 1. The finger glove was attached to the test stand in a fixed position, while new images were obtained from the laptop that was moved around. The point that the finger glove was pointing towards was recorded on the grid paper for comparison against experimental results. The experimental setup from the laptop point of view is shown in Figure 18. The finger glove was attached to the test stand to ensure a static configuration was maintained throughout the experiment. Using an actual finger for multiple measurements would be difficult since an actual finger is prone to random movement and

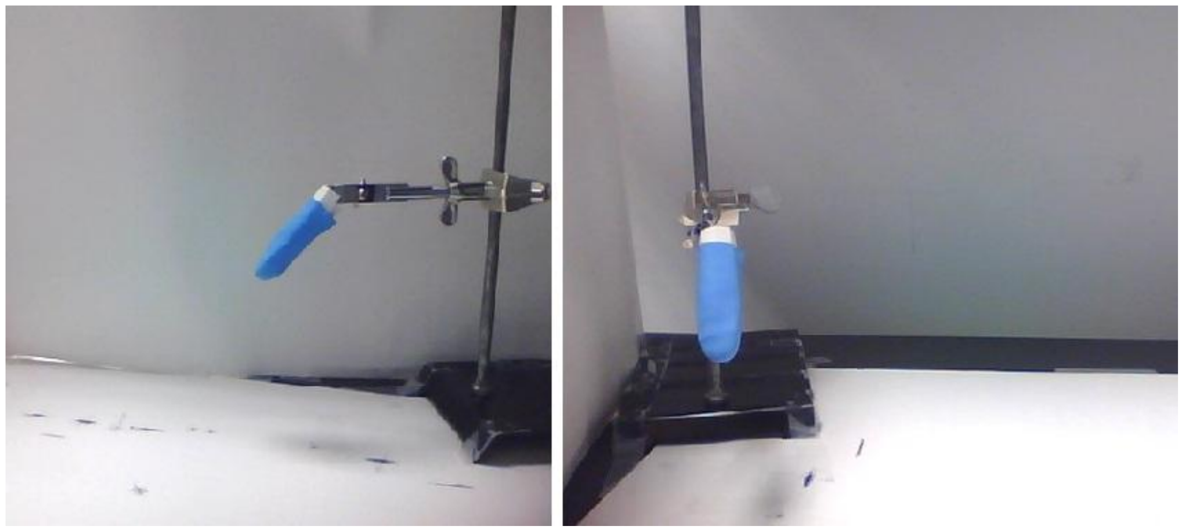


Figure 18: Experimental setup for gesture application from laptop POV. Finger glove placed in non-planar orientation

physical exhaustion.

The precise location of the laptop was determined with the use of four experimental aids. The first aid was a 1/2 inch x 1/2 inch grid poster paper, which was firmly secured and placed underneath the laptop and test stand. A grid system such as this one served as a two dimensional coordinate system that made laptop translation measurements clearer and more defined. The second aid used was standard measuring tape that was used to measure the translation of the laptop during the experiment. The third aid was a foot long ruler that was used to ensure that the laptop screen was kept orthogonal to the horizontal axis. This was done to ensure that the orientation of the laptop remained fixed at all times during the experiment and, that the camera axes were in the same direction as the laptop translation axes, as seen in Figure 19.

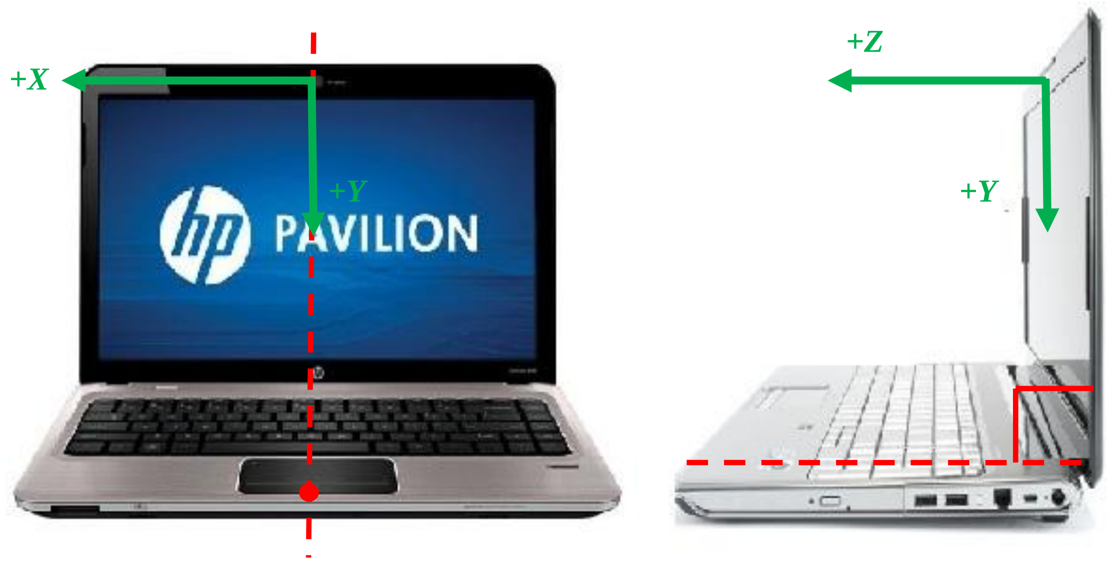


Figure 19: (a) Dotted line is line with camera center, the red dot indicates a reference point of measurement during the experiment. (b) Profile of the laptop. The screen orientation was kept perpendicular to the horizontal at all times. Camera axes and laptop translation axes are in the same direction **[37]**.

The fourth aid was the reference point on the mouse pad of the laptop. This point shown as a red dot in Figure 19 was in line with the camera axes and was used as a reference point when measuring laptop displacement during the experiment. It is important to note that in this case, the positive *y* axis has been assumed to be down.

For each camera pose, the average orientation and average Centroid of the gloved finger were determined from <u>five consecutive images</u>, provided that first the pose of the laptop was known and correct. The pose of the laptop was altered and the average orientation and average Centroid of the gloved finger were again determined using five consecutive images. The averaged Centroid and averaged orientation of the gloved finger from two different camera poses are saved and used along with the +*Y* coordinate of the ground, the camera focal parameter and homogenous transformation parameters of the camera to obtain the 3-D location of the point on the ground where the gloved finger was pointing towards.

For this experiment, this process was repeated fifteen times (30 different camera poses) to obtain the results presented in Figure 5. It must be noted that the orientation of the laptop remained fixed and the laptop was translated in either the *x* or *z* directions or a combination of both *x* and *z* directions. A summary of this experimental procedure is given in Figure 20.

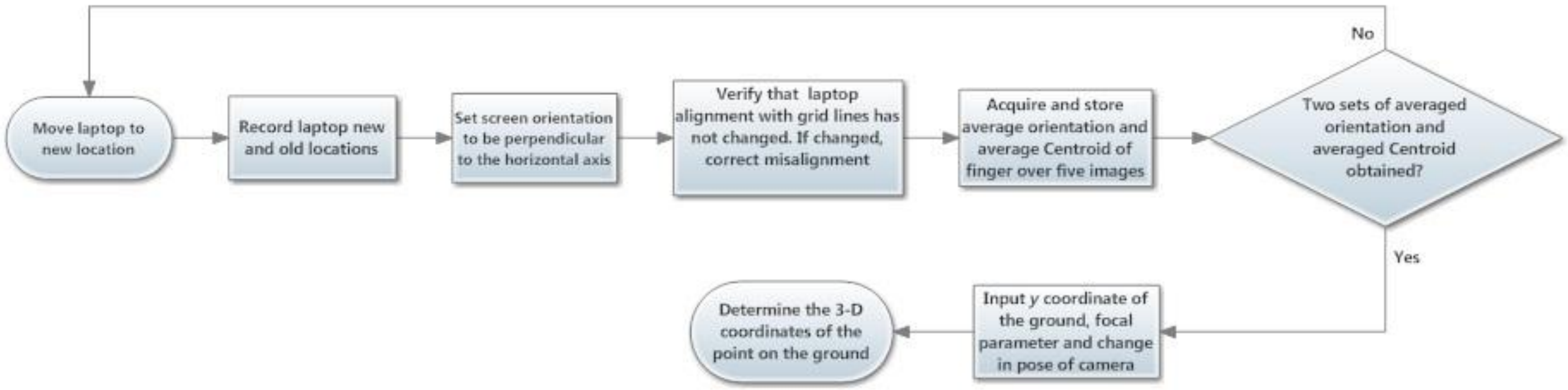

Move laptop to new location
Record laptop new and old locations
Set screen orientation to be perpendicular to the horizontal axis
Verify that laptop alignment with grid lines has not changed. If changed, correct misalignment
Acquire and store average orientation and average Centroid of finger over five images
Two sets of averaged orientation and averaged Centroid obtained?
No
Yes
Input *y* coordinate of the ground, focal parameter and change in pose of camera
Determine the 3-D coordinates of the point on the ground

Figure 20: Experimental procedure flowchart for testing method 2. This flowchart is assuming that the test stand and finger glove have been setup and will remain fixed throughout the experiment.

5.3 Experimental Results

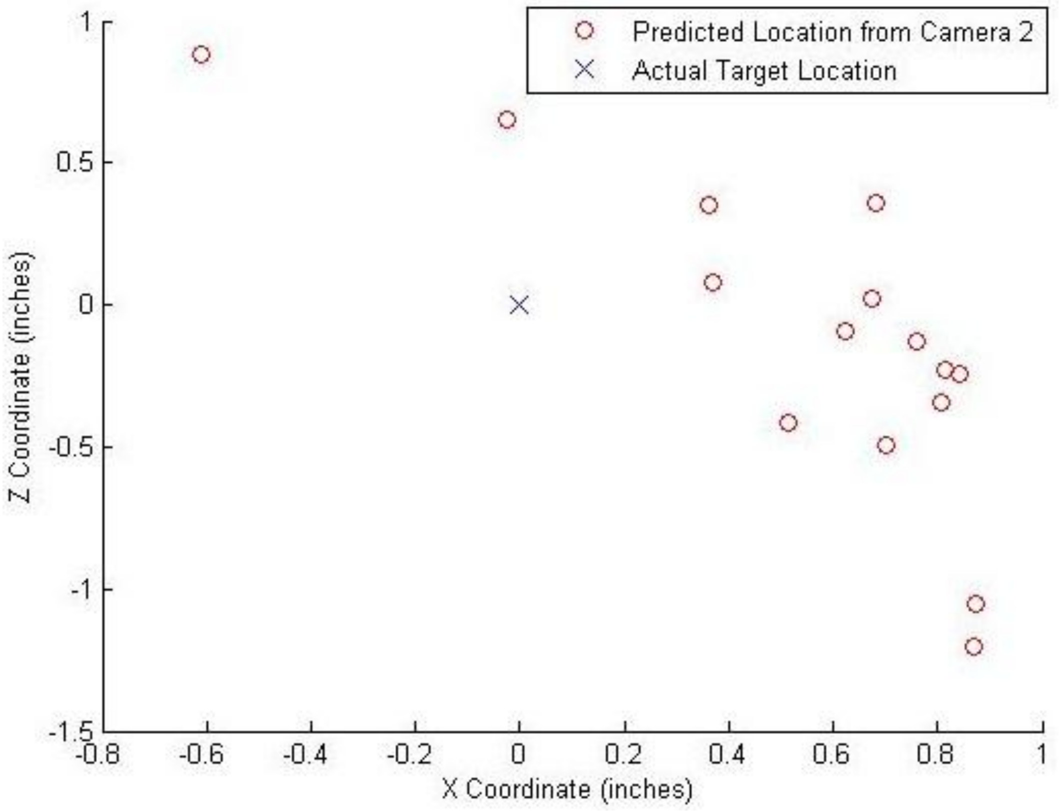


Figure 21: Predicted target locations relative to the Actual Target Location (at the origin) from the second camera. Ground location is assumed to be fixed at *y* = 9.625 inches.

Similar to Figure 11, Figure 21 shows the fixed target location at the origin and the predicted locations relative to the actual location. Only the *x* and *z* coordinates have been shown since *y* is assumed to be *h* = 9.625 inches (camera center to table top). From Figure 21, it is evident that the predictions were accurate to within ± 1 inch of the target. This translates to an average error of -15% for the *x* coordinate prediction and a +1% error for the *z* coordinate prediction, which is acceptable for this type of basic experiment.

From Figure 21, it is clear that while the *z* coordinate predictions were evenly distributed between ± 1 inch of the target *z* coordinate, most of the *x* coordinate predictions were overestimated, indicating an experimental bias. This bias was investigated and it was determined to be the result of *x* axis translation measurement error and/or issues specific to the placement of the finger glove. *x* axis translation measurement error as small as ± 0.1 inches can cause the final *x* coordinate to be overestimated by as much as 0.20 inches. Eleven of the experimental trials consisted of either pure translation along the *x* axis or a combination of translation along both the *x* and *z* axes, with the *x* axis translation always being the largest. It is entirely possible that for a few of these trials a measurement error as small as 0.1 inches could have occurred.

Additionally, the finger glove was placed in a non-planar position with respect to the camera. As a result, estimation of the area was more susceptible to variation which can result in errors in the Centroid estimation from the second image. Better lighting and better equipment (especially the camera) could have reduced the error even further. Nevertheless, if we consider how small the magnitude of the error is in the predictions, it becomes clear that (19) is an accurate and novel method for identifying points expressed in a 3-D environment.

5.4 The Effects of Averaging the Centroid and Orientation of the Finger Glove

During the experiment, the Centroid and orientation of the finger glove at each pose were measured five times and averaged before the final 3-D point estimate was determined. Five averages were chosen based on an experiment designed to determine the running average estimation of the fixed 3-D point that the finger glove was pointing towards. Figure 22 and Figure 23 depict the results of the experiment. From each plot it can be seen that the accuracy of the prediction increases as the number of averages increases over time. Therefore, averaging the Centroid and the orientation of the finger glove does have a positive impact on the final 3-D estimate. Choosing the number of averages seems obvious by examining the trend in Figure 22 and Figure 23. However, if the difference between five and 50 averages is considered, we can see that the accuracy improves by 1% on average. Therefore, based on these calculations it is apparent that five averages is more than enough to achieve accurate experimental results. Any additional averaging would most likely not have a higher benefit on the final results. Additionally, performing more than five averages may not be practical for real applications because it could take a long period of time than is desired and effectively slow down the robot during its use.

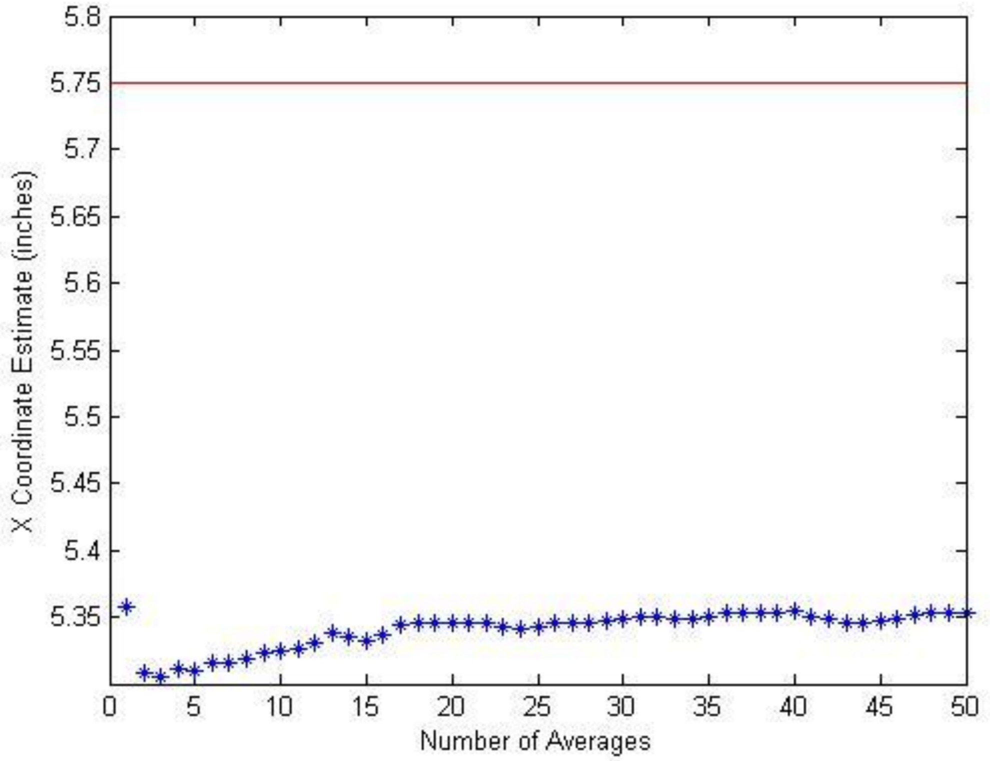


Figure 22: Running Average for *x* coordinate prediction for finger glove intersection point. Ground location is assumed to be fixed at y = 9.625 inches.

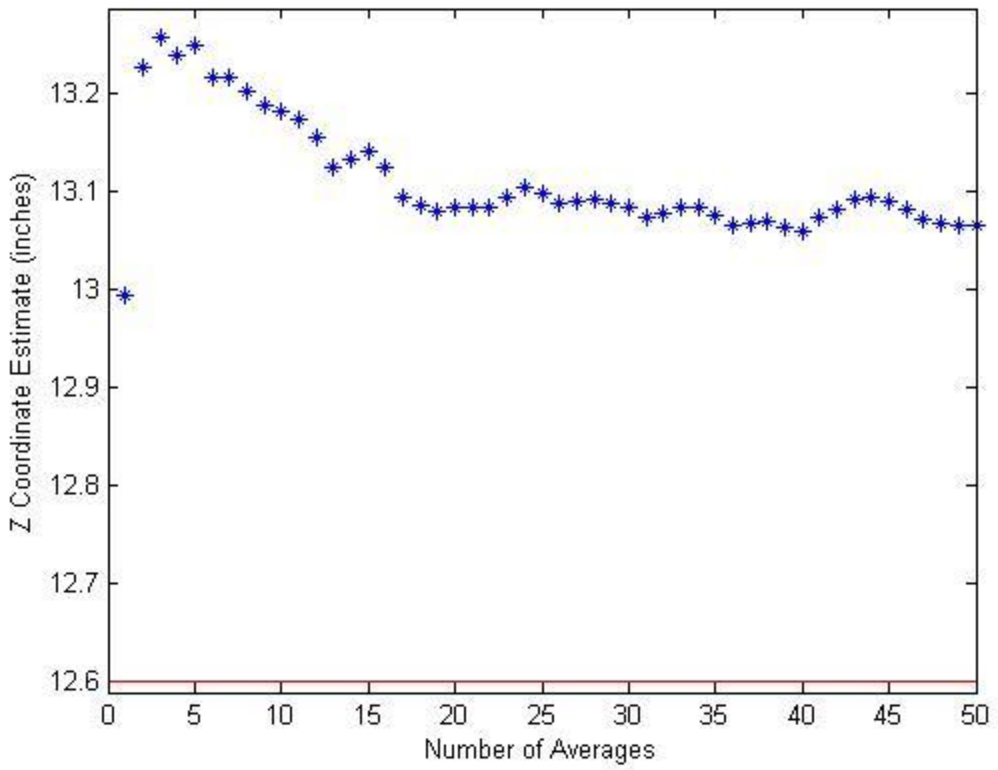


Figure 23: Running Average for *x* coordinate prediction for finger glove intersection point. Ground location is assumed to be fixed at y = 9.625 inches.

It must also be noted that a noticeable bias in the final coordinate estimation does exist in Figure 22 and Figure 23. The bias is most likely due to small translation measurement error or inaccuracies in the camera used. The camera is a webcam built into a laptop and not an expensive machine vision camera normally used for experimental work.

6 Limitations of Fixed 3-D Line and Fixed 3-D Point Reconstruction Techniques 1 and 2

The techniques presented in this work have several drawbacks. First, the techniques presented in this work rely on precise camera pose estimation from on board sensors. Any error in estimation from these sensors could result in error of the estimation of the line equation or 3-D point coordinates. Additionally, there are other 3-D techniques that receive this information using point estimation from each image, which make the techniques in this thesis less attractive because it requires additional hardware for implementation. Second, the approaches presented in this thesis do not provide a complementary correspondence algorithm. The correspondence problem is essentially left to the engineer to solve before using the techniques in this work.

Furthermore, an accurate sensitivity analysis with respect to correspondence and pose estimation errors has not been completed. Therefore, it is largely unknown how sensitive the techniques presented in this thesis are to errors in correspondence and camera pose estimation.

7 CONCLUSIONS AND FUTURE WORK

This work presented in this thesis was motivated by two important observations. First, through this work and other related work, it has been shown that Stereovision has several limitations related to the fundamentals of its design and implementation. Second, in order for robots to be more successful as human replacements, they must have human like visual perception. In particular, they must have the ability to interpret depth using a single camera, similar to how humans can perceive depth using a single eye. Therefore, this thesis has presented three novel techniques for identifying the 3-D coordinates of fixed 3-D points and the 3-D equation of a fixed line in a physical 3-D environment. The techniques presented in this thesis can be successfully used with one moving camera or two stationary cameras provided that a separate correspondence technique is implemented beforehand. The techniques used for identifying 3-D fixed points have been experimentally validated and shown to be very accurate aside from a small experimental bias. Both results suffered from the same inaccuracies and are due to factors such as small correspondence errors, small measurement errors or limitations of the equipment used.

Lastly, the methods presented in this thesis can be expanded on in three ways. First, experimental verification of the line detection method would further validate the work in this thesis. Second, the techniques presented in this thesis can be expanded to account for *n* images, meaning that either *n* cameras or a single camera taking *n* images can be used to determine any desired points and lines in a 3-D environment. Enabling the use of *n* cameras or *n* images allows for a more general approach not limited by the number of cameras used. A general approach *increases* the overall information available and will allow for a better approximation for the location of the 3-D point of interest.

Finally, these techniques should be experimented with actual moving robots to attain a better understanding of the practical uses and implementation requirements necessary for successful depth estimation using these techniques. In general, further advancement of the work presented in this thesis would be greatly beneficial to the vision system of any mobile robot or moving robot such as a robotic arm because it will remove the two camera constraint, thus allowing for a less complex vision system overall.

APPENDIX A

POINT DETECTION METHOD 1 MATLAB CODE

```
%Roshan Kalghatgi
%Date Created 11/6/2011
%This program locates a 3-D point relative to a camera in a two camera
%pose setting.

function [A,B,alpha,p] = stereovision_point(f,uv,th,d)

%Define General Rotation matrix
thx=deg2rad(th(1));
Rx=[1 0 0;0 cos(thx) -sin(thx);0 sin(thx) cos(thx)];
thy=deg2rad(th(2));
Ry=[cos(thy) 0 sin(thy);0 1 0;-sin(thy) 0 cos(thy)];
thz=deg2rad(th(3));
Rz=[cos(thz) -sin(thz) 0;sin(thz) cos(thz) 0;0 0 1];
R=Rx*Ry*Rz



%Shift uv relative to IPP
uv(1) = (uv(1) - 320);
uv(3) = (uv(3) - 320);
uv(2) = (240 - uv(2));
uv(4) = (240 - uv(4));

%Define p_bar
p_bar1 = [uv([1:2]) f]'
p_bar2 = [uv([3:4]) f]';

%Define A and B
A(:,1) = p_bar1;
A(:,2) = -R*p_bar2;

B = d';

%Solve for alpha
alpha = A\B;

p = [alpha(1)*p_bar1 alpha(2)*p_bar2];

end
```

APPENDIX B

POINT DETECTION METHOD 2 MATLAB CODE

```
%Roshan Kalghatgi
%Date Created 11/6/2011
%This program computes the x,y,z coordinates of point that is common to
three planes
function [p,A,B,R]=stereovision_line(a,p,f,th,d,h)
%a,b: angle and one point on each line [a1 a2], [p1 p2], pi=[ri;ci]
%f: camera focal parameter
%th: 3x1 vector of successive rotation about x, y, and z axes to
specify the rotation matrix between the 2 camera
%d: traslation vector between the camers optical centers.
%h: height of the ground from the camera center (defined on the y axis)

%Form the rotaion matrix R
thx=deg2rad(th(1));
Rx=[1 0 0;0 cos(thx) -sin(thx);0 sin(thx) cos(thx)];
thy=deg2rad(th(2));
Ry=[cos(thy) 0 sin(thy);0 1 0;-sin(thy) 0 cos(thy)];
thz=deg2rad(th(3));
Rz=[cos(thz) -sin(thz) 0;sin(thz) cos(thz) 0;0 0 1];
R=Rx*Ry*Rz;

%Line parameters
%unit vector perpendecular to line
%any rc(r,c) on the line satisfies m'*rc=m'*p, where m is the unit
vector
%perpendicular
a1=deg2rad(a(1)); a2=deg2rad(a(2));
m1=[cos(pi/2+a1);sin(pi/2+a1)];
m2=[cos(pi/2+a2);sin(pi/2+a2)];
p1 = p(1,[1:2]); p2=p(1,[3:4]);
b1=m1'*p1';
b2=m2'*p2';

%Form A matrix
A1 = [m2(1)*f m2(2)*f -b2];
A2 = [m1(1)*f*R(1,1)+m1(2)*f*R(2,1)-b1*R(3,1)
m1(1)*f*R(1,2)+m1(2)*f*R(2,2)-b1*R(3,2) m1(1)*f*R(1,3)+m1(2)*f*R(2,3)-
b1*R(3,3)];
A3 = [0 1 0];
A = [A1;A2;A3];

B = [0; (-m1(1)*f*d(1)-m1(2)*f*d(2)+b1*d(3));(h-d(2))];
p=inv(A)*B;

return
end
```

APPENDIX C

FINGER DETECTION GUI MATLAB CODE

```
%Name: Roshan Kalghatgi
%Date Created: 8/3/2011
%Last Updated: 11/12/2011
%File Name: BlueHandFilterLive8.m
%Version 8

function varargout = BlueHandFilterLive(varargin)


% BLUEHANDFILTERLIVE M-file for BlueHandFilterLive.fig
%      BLUEHANDFILTERLIVE, by itself, creates a new BLUEHANDFILTERLIVE
or raises the existing
%      singleton*.
%
%      H = BLUEHANDFILTERLIVE returns the handle to a new
BLUEHANDFILTERLIVE or the handle to
%      the existing singleton*.
%
%      BLUEHANDFILTERLIVE('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in BLUEHANDFILTERLIVE.M with the given
input arguments.
%
%      BLUEHANDFILTERLIVE('Prop erty','Value',...) creates a new
BLUEHANDFILTERLIVE or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before BlueHandFilterLive_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to BlueHandFilterLive_OpeningFcn
via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help BlueHandFilterLive

% Last Modified by GUIDE v2.5 07-Jul-2011 11:58:48

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @BlueHandFilterLive_OpeningFcn, ...
    'gui_OutputFcn',  @BlueHandFilterLive_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
end

% --- Executes just before BlueHandFilterLive is made visible.
function BlueHandFilterLive_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to BlueHandFilterLive (see
VARARGIN)

% Choose default command line output for BlueHandFilterLive
handles.output = hObject;

%Initialize the camera
imaqreset;

global obj;
obj = videoinput('winvideo',1,'YUY2_640x480');
set(obj,'ReturnedColorSpace','RGB','FramesPerTrigger',1);
set(obj,'TriggerRepeat',Inf,'FrameGrabInterval',2,'TimerPeriod',1/2,'Ti
merFcn',...
    @BlueHandFilter);

%set up an images to put pictures in
vidRes = get(obj, 'VideoResolution');
nBands = get(obj, 'NumberOfBands');
hIm1 = image(zeros(vidRes(2), vidRes(1), nBands),'parent',...
    handles.LiveVideoFeed);

start(obj);

    function BlueHandFilter(obj,~)
        global b m theta CentroidStore;
        tic
        data = getdata(obj);
        flushdata(obj);

        %send image data to the image object in LiveVideo Feed
        set(hIm1,'CData',data);
        set(handles.LiveVideoFeed,'xticklabel',[]);
        set(handles.LiveVideoFeed,'yticklabel',[]);
        %---Implement hand filter algorithm---%

        %I_manip is the I matrix that this program will be manipulating
        I = double(data(:,:,:,:));
        I_manip = I;

        %Find the size of I
        [m n q] = size(I_manip);

        %Preallocate space for denom matrix
        denom = zeros(m,n);

        %---Implement Color Filter---%

        %Normalize the image matrix
        denom = sqrt(I_manip([1:m],[1:n],1).^2 +
I_manip([1:m],[1:n],2).^2 + I_manip([1:m],[1:n],3).^2);

        I_manip(:,:,1) = I_manip(:,:,1)./denom;
        I_manip(:,:,2) = I_manip(:,:,2)./denom;
        I_manip(:,:,3) = I_manip(:,:,3)./denom;


        %Find the indices of the pixels that don't meet the RGB
thresholds
        [r c] = find(I_manip(:,:,1) > 0.45); %R

        lindex1 = sub2ind(size(I_manip),r,c); %you can specificy the
layer in sub2ind, third input is the layer array
        lindex2 = sub2ind(size(I_manip),r,c,2*ones(length(r),1));
        lindex3 = sub2ind(size(I_manip),r,c,3*ones(length(r),1));

        I([lindex1' lindex2' lindex3']) = 0;

        [r c] = find(I_manip(:,:,2) < 0.40); %G

        index1 = sub2ind(size(I_manip),r,c); %you can specificy the
layer in sub2ind, third input is the layer array
        lindex2 = sub2ind(size(I_manip),r,c,2*ones(length(r),1));
        lindex3 = sub2ind(size(I_manip),r,c,3*ones(length(r),1));

        I([lindex1' lindex2' lindex3']) = 0;

        [r c] = find(I_manip(:,:,3) < 0.6); %B 50.65

        lindex1 = sub2ind(size(I_manip),r,c); %you can specifiy the
layer in sub2ind, third input is the layer array
        lindex2 = sub2ind(size(I_manip),r,c,2*ones(length(r),1));
        lindex3 = sub2ind(size(I_manip),r,c,3*ones(length(r),1));

        I([lindex1' lindex2' lindex3']) = 0;
        %---END Color Filter---%

        %Convert to bw based on a grayscale threshold
        thresh = graythresh(I);
        I = im2bw(I,thresh);

       % I = bwconncomp(I,26);

        %Find objects in the image and all the properties
        cc1 = 
regionprops(I,'Area','Orientation','Centroid','PixelList','Perimeter','
Eccentricity','MajorAxisLength','MinorAxisLength');

     %---Filter objects by Area, Perimeter^2/Area, and Eccentricity---%

        %Of all objects found find those >500 sq pixels & e >=0.86
        indices1 = intersect(find([cc1(:,:).Area] > 
200),find([cc1(:,:).Eccentricity] >= 0.86));
        %indices1 = find([cc1(:,:).Eccentricity] >= 0.86);

        %Calculate pi using the area, major axis and minor axis
        try
        ab = 
(cc1(indices1,:).MajorAxisLength).*(cc1(indices1,:).MinorAxisLength);

        ab = ab(ab ~=0);
        indices1 = indices1(find(ab ~=0));

        PI_filter = 4*(cc1(indices1,:).Area./ab);


        %Notes: Perimeter^2/Area and max/min are good BUT they are 2-D
        %characteristics, what if you angle your finger. You have the
        %effects of foreshortening and thus the finger will be filtered
        %out. this can work if you have a large
        %range of values. Need a param that is independent of 3-d, also 
need to improve
        %color filter

        %Of all obj found in indices1, determine which have PtoA within 
[15 35]
        %indices2 is the index of indices1! Very important note!
        indices2 = intersect(find(PI_filter > 2.90),find(PI_filter < 
3.08));

        %indices give you the original indices from cc1 of the object 
that
        %meets the criteria
        indices = indices1(indices2);

        %Determine the number of objects
        NumObjects = length(indices);
        %---End object filter---%

        %Preallocating
        AreaofObj = zeros(1,NumObjects)';
        Centroids = zeros(2,NumObjects)';

        catch
            NumObjects = 0;
        end

        if NumObjects > 0
            for k = 1:1:NumObjects

                    %Calculate area, orientation and centroid of object
found
                    AreaofObj(k) = cc1(indices(k),1).Area;
                    cc1(indices(k),:).Eccentricity;
                    Centroids(k,[1:2]) = cc1(indices(k),:).Centroid
%(x,y)

                    % maxtomin =
(cc1(indices(k),:).MajorAxisLength)*(cc1(indices(k),:).MinorAxisLength)
                    %ratio = 4*AreaofObj(k)/maxtomin

%([cc1(indices(k),:).Perimeter].^2)./[cc1(indices(k),:).Area]

                    %Make centroids relative to IPP
     Centroids(k,[1:2]) = [(Centroids(k,1)-320) (Centroids(k,2)-240)];
                    CentroidStore = Centroids(k,[1:2]);

                    %Calculate orientation
                    theta = -cc1(indices(k),:).Orientation

                           %Calculate the normal vector of the plane
                  that the line
                    %lies on
                    N = [cosd(theta+90) sind(theta+90)];

                    %Determine the line parameters
                    D = N*[Centroids(k,[1:2])]';

                    b = D/sind(theta + 90);

                    m = (-1/tand(theta+90));

                    x = [-300:1:300];
                    y = m*x + b;

                    %Find all the pixels that make up the object
                    PList{k} = cc1(indices(k),:).PixelList;
       PList{k}(:,1:2) = [(PList{k}(:,1)-320) (PList{k}(:,2)-240)];

                end
        else
```

```
            Centroids = [0 0];
            PList = [0 0];
            x = 0;
            y = 0;
            m = 0;
            b = 0;

        end
        try
            slope = m;
            yintercept = b;
            thetaTag = num2str(theta);
        catch
            slope = 0;
            yintercept = 0;
            thetaTag = 'No object';

        end

        try
            e = num2str(cc1(1,1).Eccentricity);
            AreaTag = num2str(AreaofObj);
        catch
            e = 'No object';
            AreaTag = 'No object';
        end

        %Send data to textboxes
        set(handles.eccen,'string',e);
        set(handles.lineEq ,'string',sprintf('%gc +
%g',slope,yintercept));
        set(handles.NumObjText,'string',NumObjects);
        set(handles.OrientationTag,'string',thetaTag);
        set(handles.AreaTag,'string',AreaTag);

        %Plot the centroids and the finger area
        if NumObjects > 0
            % plot(handles.FilterResults,Centroids(:,1),-
Centroids(:,2),'r+','MarkerSize',20);
            for z = 1:1:NumObjects
                plot(handles.FilterResults,PList{z}(:,1),-
PList{z}(:,2),'*',x,-y,'-',Centroids(:,1),-
Centroids(:,2),'r+','MarkerSize',5);
            end
        else
            plot(handles.FilterResults,0,0);
        end
         toc
      end%end BlueHandFilter

% Update handles structure
guidata(hObject, handles);

end%end opening function
% UIWAIT makes BlueHandFilterLive wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = BlueHandFilterLive_OutputFcn(~, ~, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
end

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(~, ~, ~)
% hObject    handle to pushbutton1 (see
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

end
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(~, ~, ~)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDA
end



% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, ~, ~)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
imaqreset();
delete(hObject);
end



% --- Store Line 1 info
function strL1_Callback(~, ~, handles)

global theta_line1 Centroid1 theta CentroidStore n1;

%Initialize n1, Centroid 1 and theta_line1
if isempty(n1)
    n1 = 0;
end

if isempty(Centroid1)
    Centroid1 = [0 0];
end
if isempty(theta_line1)
```

```
    theta_line1 = 0;
end

%increment counter
n1 = n1 + 1;

% Store line 1 characteristics and average them
theta_line1 = ((n1-1)*theta_line1 + theta)/n1;
Centroid1 = [(((n1-1)*Centroid1(1) + CentroidStore(1))/n1),(((n1-
1)*Centroid1(2) + CentroidStore(2))/n1)];

%send line 1 info to the textboxes
set(handles.thetaL1,'string',num2str(theta_line1));
set(handles.CentroidL1,'string',sprintf('(%g,%g)',Centroid1(1),Centroid
1(2)));
set(handles.NumavgL1,'string',num2str(n1));
end

% --- Store Line 2 info
function strL2_Callback(~, ~, handles)

global theta_line2 Centroid2 theta CentroidStore n2;

%Initialize variable
if isempty(n2)
    n2 = 0;
end

if isempty(Centroid2)
    Centroid2 = [0 0];
end

if isempty(theta_line2)
    theta_line2 = 0;
end

%increment counter
n2 = n2 + 1;

% Store line 2 characteristics
theta_line2 = ((n2-1)*theta_line2 + theta)/n2;
Centroid2 = [(((n2-1)*Centroid2(1) + CentroidStore(1))/n2),(((n2-
1)*Centroid2(2) + CentroidStore(2))/n2)];

%send line 1 info to the textboxes
set(handles.thetaL2,'string',num2str(theta_line2));
set(handles.CentroidL2,'string',sprintf('(%g,%g)',Centroid2(1),Centroid
2(2)));
set(handles.NumavgL2,'string',num2str(n2));
end

% --- Executes on button press in calcPoint.
function calcPoint_Callback(~, ~, handles)
% hObject    handle to calcPoint (see GCBO)
```

```
global b_line1 m_line1 theta_line1 Centroid1 b_line2 m_line2
theta_line2 Centroid2

%Store Transformation Parameters
th_x = str2num(get(handles.Rx,'string'))
th_y = str2num(get(handles.Ry,'string'))
th_z = str2num(get(handles.Rz,'string'))

d_x = str2num(get(handles.Tx,'string'))
d_y = str2num(get(handles.Ty,'string'))
d_z = str2num(get(handles.Tz,'string'))

%Store Height (relative to the second camera)
ht = str2num(get(handles.height,'string'));

%Set focal parameter
f = str2num(get(handles.focalParam,'string'));

%Calculate point
[p,A,B,R]=stereovision_line([theta_line1 theta_line2],[Centroid1
Centroid2],f,[th_x th_y th_z],[d_x d_y d_z],ht)


%Output point to the gui textboxes
set(handles.px,'string',num2str(p(1)))
set(handles.py,'string',num2str(p(2)))
set(handles.pz,'string',num2str(p(3)))

% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
end


function Rx_Callback(~, ~, ~)
% hObject    handle to Rx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Rx as text
%        str2double(get(hObject,'String')) returns contents of Rx as a
double
end

% --- Executes during object creation, after setting all properties.
function Rx_CreateFcn(hObject, ~, ~)
% hObject    handle to Rx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function Ry_Callback(~, ~, ~)
% hObject    handle to Ry (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Ry as text
%        str2double(get(hObject,'String')) returns contents of Ry as a
double
end

% --- Executes during object creation, after setting all properties.
function Ry_CreateFcn(hObject, ~, ~)
% hObject    handle to Ry (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function Rz_Callback(~, eventdata, handles)
% hObject    handle to Rz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Rz as text
%        str2double(get(hObject,'String')) returns contents of Rz as a
double

end
% --- Executes during object creation, after setting all properties.
function Rz_CreateFcn(hObject, ~, ~)
% hObject    handle to Rz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end

end

function Tx_Callback(hObject, eventdata, handles)
% hObject    handle to Tx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Tx as text
%        str2double(get(hObject,'String')) returns contents of Tx as a
double

end
% --- Executes during object creation, after setting all properties.
function Tx_CreateFcn(hObject, ~, handles)
% hObject    handle to Tx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


end
function Ty_Callback(hObject, eventdata, handles)
% hObject    handle to Ty (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Ty as text
%        str2double(get(hObject,'String')) returns contents of Ty as a
double

end
% --- Executes during object creation, after setting all properties.
function Ty_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Ty (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

end
```

```
function Tz_Callback(hObject, eventdata, handles)
% hObject    handle to Tz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Tz as text
%        str2double(get(hObject,'String')) returns contents of Tz as a
double
end

% --- Executes during object creation, after setting all properties.
function Tz_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Tz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function height_Callback(hObject, eventdata, handles)
% hObject    handle to height (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of height as text
%        str2double(get(hObject,'String')) returns contents of height
as a double

end
% --- Executes during object creation, after setting all properties.
function height_CreateFcn(hObject, eventdata, handles)
% hObject    handle to height (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end
```

```
function pz_Callback(hObject, eventdata, handles)
% hObject    handle to pz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pz as text
%        str2double(get(hObject,'String')) returns contents of pz as a
double
end

% --- Executes during object creation, after setting all properties.
function pz_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

end

function py_Callback(hObject, eventdata, handles)
% hObject    handle to py (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of py as text
%        str2double(get(hObject,'String')) returns contents of py as a
double
end

% --- Executes during object creation, after setting all properties.
function py_CreateFcn(hObject, eventdata, handles)
% hObject    handle to py (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

end
```

```
function px_Callback(hObject, eventdata, handles)
% hObject    handle to px (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of px as text
%        str2double(get(hObject,'String')) returns contents of px as a
double

end
% --- Executes during object creation, after setting all properties.
function px_CreateFcn(hObject, ~, ~)
% hObject    handle to px (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end


function focalParam_Callback(~, eventdata, handles)
% hObject    handle to focalParam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of focalParam as text
%        str2double(get(hObject,'String')) returns contents of
focalParam as a double

end
% --- Executes during object creation, after setting all properties.
function focalParam_CreateFcn(hObject, ~, handles)
% hObject    handle to focalParam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
end
```

```
% --- Executes on button press in Reset.
function Reset_Callback(~, ~, handles)
% hObject    handle to Reset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global theta_line1 Centroid1 theta_line2 Centroid2 n1 n2;


%Reset averages counter
n1 = 0;
set(handles.NumavgL1,'string',num2str(n1));
n2 = 0;
set(handles.NumavgL2,'string',num2str(n2));

%Reset Transformation Parameters
th_x = 0;
set(handles.Rx,'string',num2str(th_x));
th_y = 0;
set(handles.Ry,'string',num2str(th_y));
th_z = 0;
set(handles.Rz,'string',num2str(th_z));

d_x = 0;
set(handles.Tx,'string',num2str(d_x));
d_y = 0;
set(handles.Ty,'string',num2str(d_y));
d_z = 0;
set(handles.Tz,'string',num2str(d_z));

%Store Height (relative to the second camera)
ht = 9.625;
set(handles.height,'string',num2str(ht));

%Reset focal parameter
f = 525;
set(handles.focalParam,'string',num2str(f));

%Reset saved information
%Reset all line parameters and transformation parameters
theta_line1 = 0;
set(handles.thetaL1,'string',num2str(theta_line1));
Centroid1 = [0 0];
set(handles.CentroidL1,'string',sprintf('(%g,%g)',Centroid1(1),Centroid
1(2)));
theta_line2 = 0;
set(handles.thetaL2,'string',num2str(theta_line2));
Centroid2 = [0 0];
set(handles.CentroidL2,'string',sprintf('(%g,%g)',Centroid2(1),Centroid
2(2)));

%Reset display for calculated point
set(handles.px,'string',num2str(0))
set(handles.py,'string',num2str(0))
set(handles.pz,'string',num2str(0))  end
```

APPENDIX D

RUNNING AVERAGE PLOTTING SCRIPT

```
%Averaging test data script
%Roshan Kalghatgi
%12/31/2011
%This code takes thetas and Centroids taken over 50 averages and
computes a running average plot of x and z coordinate predictions.
n = 50;
a(:,[1:2]) = [dummyStore1([1:n],1) dummyStore2([1:n],1)];

p(:,[1:4]) = [dummyStore1([1:n],[2:3]) dummyStore2([1:n],[2:3])];

for i = 1:1:n

prediction(i,[1:3]) = stereovision_line(a(i,:),p(i,:),525,[0 0 0],[1.1
0 0],9.625);
end

numAvgs = [1:1:n];

figure(1)
plot(numAvgs,prediction(:,1),'*',[0 numAvgs], 1.91*ones(n+1,1),'-r');
ylabel('X Coordinate Estimate (inches)');
xlabel('Number of Averages');

figure(2)
plot(numAvgs,prediction(:,3),'*',[0 numAvgs], 8.375*ones(n+1,1),'-r');
xlabel('Number of Averages');
ylabel('Z Coordinate Estimate (inches)');
```

REFERENCES

[1] L.G. Shapiro and G.C. Stockman, *Computer Vision*. Upper Saddle River, NJ: Prentice Hall, 2001.

[2] O. Faugeras, *Three-Dimensional Computer Vision: A Geometric Viewpoint*. Cambridge, , MA: MIT Press, 1993.

[3] D.H. Ballard and C.M. Brown, *Computer Vision*. Englewood Cliffs, NJ: Prentice Hall, 1982.

[4] D. Oram, "Rectification for Any Epipolar Geometry," *The British Machine Vision Association and Society for Pattern Recognition*, pp. 653-662, 2001.

[5] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, United Kingdom: University Press, 2000.

[6] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, United Kingdom: University Press, 2003.

[7] A. Saxena, "Monocular Depth Perception and Robotic Grasping of Novel Objects," Stanford University, Ph.D Thesis 2009.

[8] A. Saxena, S.H. Chung, and A.Y. Ng, "3-D Depth Reconstruction from a Single Still Image," *International Journal of Computer Vision*, vol. 76, no. 1, pp. 53-69, January 2008.

[9] C. Bianco. (2011, December) How Vision Works: Depth Perception. [Online]. http://science.howstuffworks.com/environmental/life/human-biology/eye10.htm

[10] H., Basslich, P. Bassmann, "Monocular Computer Vision," in *3rd Conf. on Image Processing and its Applications*, Warwick, 1989, pp. 107-111.

[11] S., Blas, M. Riisgaard. (2005) SLAM for Dummies. [Online]. http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf

[12] H., Bailey, T. Durrant-Whyte, "Simultaneous Localization and Mapping (SLAM): Part I The Essential Algorithms," *Robotics and Automation Magazine*, vol. 13, no. 2, pp. 99-110, June 2006.

[13] A., Reid, I., Molton, N., Stasse, O. Davison, "MonoSLAM: Real-Time Single Camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052-1067, June 2007.

[14] C.G., Pike, J.M. Harris, "3D Positional Integration From Image Sequences," in *Third Alvey Vision Conference*, 1987, pp. 233-236.

[15] K. Sugihara, "Some Location Problems for Robot Navigation Using a Single Camera," *Computer Vision, Graphics, and Image Processing*, vol. 42, pp. 112-129, 1988.

[16] M., Wyeth, G. Milford, "Single Camera Vision-Only SLAM on a Suburban Road Network," in *IEEE ICRA*, Pasadena, 2008, pp. 3684-3689.

[17] R. Zisserman, A. Hartley. (1999) Multiple View Geometry. Power Point Lecture.

[18] S., Strum P. Maybank, "A Method for Interactive 3-D Reconstruction of Piecewise Planar Objects from Single Images," in *The 10th British Machine Vision Conference*, 1999, pp. 265-274.

[19] B., Maier, D., Manner, R., Yu, M. Liu, "An Efficient and Accurate Method for 3-D Point Reconstruction from Multiple Views," *International Journal of Computer Vision*, January 2002.

[20] J., Hubner, K. Zhang. (2002, June) Using Symmetry as a Feature in Panoramic Images for Mobile Robot Applications. Power Point Presentation.

[21] H., Miura, J., Shirai, Y. Koyasu, "Recognizing Moving Obstacles for Robot Navigation using Real-time Omnidirectional Stereo vision," *Journal of Robotics and Mechatronics*, vol. 14, no. 2, pp. 147-156, 2002.

[22] Z. Zhu, "Omnidirectional Stereo Vision," in *10th IEEE ICAR*, Budapest, 2001.

[23] S., Ahuja, N. Yi, "An Omnidirectional Stereo Vision System Using a Single Camera," in *18th IEEE ICPR*, Hong Kong, 2006.

[24] S., Nayar, S. Baker, "A Theory of Single-Viewpoint Catadioptric Image Formation," *International Journal of Computer Vision*, vol. 35, no. 2, pp. 175-196, 1999.

[25] E., Sagawa, R., Echigo, T., Yagi, Y. Mouaddib, "Stereovision with a Single Camera and Multiple Mirrors," in *IEEE ICRA*, Barcelona, 2005.

[26] T.,Yamaguchi, J. Nishimoto, "Three Dimensional Measurement Using Fisheye Stereo Vision," in *SICE Annual Conference*, Takamatsu, 2007, p. 2007.

[27] L., Luo, C., Feng, Z., Hao, Y. He. (2008) intech.com.

[28] H., Koenig, A., Schroeter, C., Boehme, H. Gross, "Omnivision-based Probabilistic Self-localization for a Mobile Shopping Assitant Continued," in *IROS*, Las Vegas, 2003, pp. 1505-1511.

[29] C., Koenig, A., Boehme, H., Gross, H. Schroeter, "Multi-Sensor Monte-Carlo-Localization Combining Omni-vision and Sonar Range Sensors," in *2nd European Conference on Mobile Robots*, Ancona, 2005, pp. 164-169.

[30] A. Saxena, J. Michels, and A.Y. Ng, "High Speed Obstacle Avoidance using Monocular Vision and Reinforcement Learning," in *22nd International Conference on Machine Learning (ICML)*, Bonn, 2005.

[31] A. Saxena, M. Sun, and A.Y. Ng, "Make3D: Learning 3D Scene Structure from a Single Still Image," *IEEE Transactions of Pattern Analysis and Machine Intelligence (PAMI)*, vol. 30, no. 5, pp. 824-840, 2009.

[32] A. Saxena, S.H. Chung, and A.Y. Ng, "Learning Depth from Single Monocular Images," in *Neural Information Processing Systems (NIPS)*, 2005, p. 18.

[33] A. Saxena, J. Schulte, and A. Ng, "Depth Information Using Monocular and Stereo Cues," in *20th International Joint Conference on Artificial Intelligence*, Hyderabad, India, 2007.

[34] G., Nalpantidis, L., Sirakoulis, G., Gasteratos, A. De Cubber, "Intelligent Robots need Intelligent Vision: Visual 3D Perception ," International Workshop on Robotics for Risky Interventions and Survelillance of the Enviornment, Benicassim, 2008.

[35] Azarbayejani, A., Pentland, A. Jebara T. (1999, May) 3D Structure from 2D Motion. [Online]. http://www.cs.columbia.edu/~jebara/htmlpapers/SFM/sfm.html

[36] S. Lazebnik. (2011, March) Structure from Motion Lecture Slides. Power Point Presentation.

[37] H., Shiratori, T., Matthews, I., Sheikh, Y. Park, "3D Reconstruction of a Moving Point from a Series of 2D Projections," in *ECCV*, Crete, 2010, pp. 158-171.

[38] J.J. Craig, *Introduction to Robotics, Mechanics and Control*. Upper Saddle River, NJ: Prentice Hall, 2005.

[39] Johann. (2010, May) Techfresh.net. [Online]. http://laptops.techfresh.net/hp-pavilion-dv6t-select-edition/

[40] D. Pirvu. (2009, Apr.) GeekSailor.com. [Online]. http://www.geeksailor.com/hp-pavilion-dv6t-1030us-review/

[41] D., Huber E., Bonasso, P. Kortenkamp, "Recognizing and interpreting gestures on a mobile robot," in *AAAI-96*, Portland, OR, 1996, pp. 915-921.

[42] D., Baillieul, J. Raghunathan, "Relative Motion of Robots as a Means of Signaling ," in *International Conference on Intelligent Automation and Robotics*, 2009.

[43] V., Sharma, R., Huang, T. Pavlovic, "Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 677-695, 1997.

[44] Wolfram Research Inc. (2011) Eccentrcity. [Online]. http://mathworld.wolfram.com/Eccentricity.html

[45] Celeste Biever, "Why a robot is better with one eye than two," *New Scientist* , vol. 188, no. 2530, p. 30, December 2005.

[46] T. Breckon. (2006) 3D Recognition from stereovision: Correspondance and Triangulation. Power Point Lecture.

[47] N. Leventon, M., Freeman, W. Howe, "Bayesian Reconstruction of 3D Human Motion from Single-Camera Video," Mitsubishi Electric Research Laboratory, Cambridge, Published Internal Report 1999.

[48] G., Kim S., Kweon I. Jang, "Single Camera Catadioptric Stereo System," in *Workshop on Omnidirectional Vision, Camera Networks and Non-classical cameras*, 2005.

VITA

ROSHAN SATISH KALGHATGI

Roshan was born in Schaumburg, Illinois. He attended public schools in Schaumburg, IL, Ann Arbor, MI and Skillman, NJ. Prior to attending Georgia Tech, he received a B.S. in Aerospace Engineering from Boston University in 2009. While at Boston University he worked as a summer intern at GE Aviation in Lynn, MA and also performed undergraduate research in motion based communication and its application to Robotics. When not working on engineering problems, Roshan enjoys writing short stories and reading anything he can get his hands on. He also enjoys movies, exercise and personal fitness.