Scene Generation and Target Detection for Hardware-in-the-Loop Simulation

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Ryan E. Sherrill

Certificate of Approval:

John E. Cochran Jr. Professor and Head Aerospace Engineering	Andrew J. Sinclair, Chair Assistant Professor Aerospace Engineering
Brian S. Thurow Assistant Professor Aerospace Engineering	George T. Flowers Dean Graduate School

Scene Generation and Target Detection for Hardware-in-the-Loop Simulation

Ryan E. Sherrill

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
May 9, 2009

Scene Generation and Target Detection for Hardware-in-the-Loop Simulation

Ryan E. Sherrill

Permission is granted to Auburn University to make copies of this thesis at its discretion, upon the request of individuals or institutions and at their expense. The author reserves all publication rights.

Signature of Author

Date of Graduation

VITA

Ryan Edward Sherrill, son of Robert Edward Sherrill Jr. and Isabelle Kathryn Montoya, was born May 4, 1985 in Farmington, New Mexico. He graduated with honors from Farmington High School, and entered Auburn University in the fall of 2003. He received his Bachelor of Aerospace Engineering degree in May of 2007, and entered the Graduate School the following semester.

THESIS ABSTRACT

SCENE GENERATION AND TARGET DETECTION FOR HARDWARE-IN-THE-LOOP SIMULATION

Ryan E. Sherrill

Master of Science, May 9, 2009
(B.A.E., Auburn University, 2007)

74 Typed Pages

Directed by Andrew J. Sinclair

Hardware-in-the-Loop simulations are useful in developing and testing missile components at a lower cost than experimental tests. Accurate results require the missile's optical sensor be stimulated by an artificial environment that represents the physical world. As part of Auburn University's development of a Hardware-in-the-Loop lab, several software modules have been created that generate a simulated infrared engagement scene, and emulate the target detection and tracking which occurs onboard a missile. These scene generation and target detection tools allow pure-digital and static Hardware-in-the-Loop simulations to be performed.

ACKNOWLEDGMENTS

The author is grateful to Christian Bruccoleri and Puneet Singla for use of their Matlab code to generate pseudo-uniform points on a sphere. The author also acknowledges the California Institute of Technology Computational Vision Group for maintaining a collection of articles related to image calibration. The author thanks Dr. Andrew Sinclair for his knowledge and invaluable support, and Dr. John Cochran Jr. for having faith in the author's abilities and inviting him to work on this research project.

The author would like to thank the Aerospace Engineering Department for its generous financial assistance. The author is eternally grateful to his parents for their unwavering support and continuous encouragement.

Finally, the author dedicates this thesis to the memory of two teachers in his life, Mr. James Michael DeField (Oct. 30, 1952 - Aug. 19, 2005), my physics teacher at Farmington High School, and Dr. Alan Scottedward Hodel (April 6, 1962 - Jan. 9, 2009), Associate professor of Electrical and Computer Engineering at Auburn, who both passed away after valiant fights with cancer. These gentlemen were those almost magical teachers that students hear about, people who teach for the sheer joy of teaching, and who unselfishly put their students before their own work. Even though he knew Mr. DeField and Dr. Hodel for a short time, each of them had a profound impact on my life. As he continues his education toward the goal of someday becoming a teacher himself, he will look back on the memories they provided, and for my future students' sake, hope that he becomes half the teacher each of them was.

Style manual or journal used: Bibtex

Computer software used: Matlab 2007a, National Instruments Vision Builder, Microsoft Word 2007, Microsoft Paint, WinEdt, Latex

TABLE OF CONTENTS

LIST OF FIGURES x

1 INTRODUCTION 1
  1.1 Description of TigerSim 2
  1.2 Proposed Additions 5

2 SCENE GENERATION 7
  2.1 Creating an Artificial Environment 7
  2.2 Modeling the Target Characteristics 9
  2.3 Rendering Process in Matlab 11
  2.4 Display Setup 14
  2.5 Examples of Generated Scenes 15

3 SEEKER MODEL: SOFTWARE-IN-THE-LOOP 17
  3.1 Target Detection 17
  3.2 Derivation of the LOS vector 20
  3.3 Seeker Steering 22

4 SEEKER MODEL: STATIC HARDWARE-IN-THE-LOOP 24
  4.1 Hardware Setup 24
  4.2 Target Detection Process 26
  4.3 Coordinate Mapping 29
  4.4 Camera Error Sources 31
    4.4.1 Resolution Downsampling 31
    4.4.2 Bounding Box 31
    4.4.3 Lens Distortion 33
    4.4.4 Other Sources 33
  4.5 Graphical Error Analysis 34

5 RESULTS 38
  5.1 LOS Errors 38
  5.2 Seeker Steering Errors 44
  5.3 Intercept Success Rate 44
  5.4 Simulation Run Time 46

6 Conclusion and Future Recommendations 47

Bibliography 49

Appendix A: Transformation Matrices 50

Appendix B: Matlab Code for Scene Generation 52

Appendix C: Matlab code for SWIL Target Detection 55

Appendix D: Matlab code for SHWIL Target Detection 58

Appendix E: Matlab code for SHWIL Calibration 60

Appendix F: Matlab code for Lens Distortion Determination 62

List of Figures

1.1 Example of a HWIL simulation modeling an actual engagement. . . . 2

1.2 Flow chart showing the TigerSim subroutines. 3

1.3 The LOS vector between the interceptor and target. 4

2.1 The spherical star field used to represent the night sky. 8

2.2 Colormap developed in Matlab. 10

2.3 Visual target model. 10

2.4 The field of view is the angle the seeker is able to observe. 11

2.5 Missile coordinate frame (X, Y, Z) and seeker coordinate frame (x, y, z). The gimbaled sensor inside the missile seeker is oriented through the azimuth (ϕ) and elevation (θ) angles. 12

2.6 The geometric center of the target (red circle) and the point on the body representing the simulation target location (blue circle). Units are in meters. 13

2.7 Monitor resolution of 1280 by 1024 pixels and figure window resolution of 900 by 900 pixels. 14

2.8 The rendered target scene ten seconds prior to intercept. 15

2.9 The rendered target scene one second prior to intercept. 15

2.10 The rendered target scene one half of a second prior to intercept. . . 16

2.11 The rendered target scene one tenth of a second prior to intercept. . . 16

3.1 Raster scan pattern. 18

3.2 Intercept scene with target centroid marked as determined by the target detection algorithm. 19

3.3 Reduction of scan area. 19

3.4 Pinhole camera model. 20

3.5 Relationship between geometric and physical coordinates. 21

4.1 Static HWIL laboratory setup 25

4.2 Alignment of the television and camera a)horizontal b)vertical c)alignment pattern on the screen 27

4.3 Rendered scene captured by the camera showing the target (red box) and the bounding box (green box). 28

4.4 SHWIL example showing the detected target (red box) and the transmitted centroid location (red cross) 28

4.5 Converting generated scene to camera pixel coordinates. 30

4.6 Mapping between monitor and camera coordinates. 30

4.7 Generated image with a resolution of 900 pixels by 900 pixels. 32

4.8 Captured image of 373 pixels by 373 pixels. 32

4.9 Bounding box selects the portion of the camera pixels that contain the figure window. 33

4.10 Radial lens distortion will alter the original grid (black) to a distorted image (red). 34

4.11 Control points (blue cross) and imaged points (red square) for different points on the figure window. 36

4.12 Histogram of X error of the control points. 37

4.13 Histogram of Y error of the control points. 37

5.1 Sphere of possible intercepts along with a nominal target and interceptor path. 39

5.2 Average SWIL LOS error over 100 trials. 40

5.3 Average SHWIL LOS error over 100 trials. 41

5.4 Average SHWIL seeker steering error over 100 trials. 42

5.5 Average SHWIL seeker steering error over 100 trials. 43

5.6 Location of SWIL probable hit and miss intercepts. 45

Chapter 1

Introduction

The purpose of a Hardware-in-the-Loop (HWIL) simulation is to model as accurately as possible the response of physical components to a simulated environment. This technology has multiple uses for military missiles, including the testing of new systems and components, quality control for manufacturing processes, and reliability assurance for stockpiled components. In a typical HWIL test, a missile seeker is mounted to a flight motion table facing a projected scene. The missile's seeker responds to the scene as if in a real flight, passing information to the rest of the missile's systems [1]. Figure 1.1 shows an example of a HWIL simulation recreating an actual missile flight. Numerous simulated engagements can be presented to each missile, and its performance evaluated, offering a wider range of data than would be possible in a live-fire test, at a much lower cost. With collaboration from the US Army, Auburn University is developing a HWIL facility for educational and research purposes. Once operational, it will provide training for students on complex simulations, model development, and the testing of unclassified hardware. As part of this effort, a 6 degree-of-freedom computer simulation was developed that models the flight of a missile as it intercepts a ballistic target in the upper atmosphere. The goal of the research described herein was to increase the simulation capability by augmenting the current computer simulation, TigerSim, with scene generation and target detection programs.

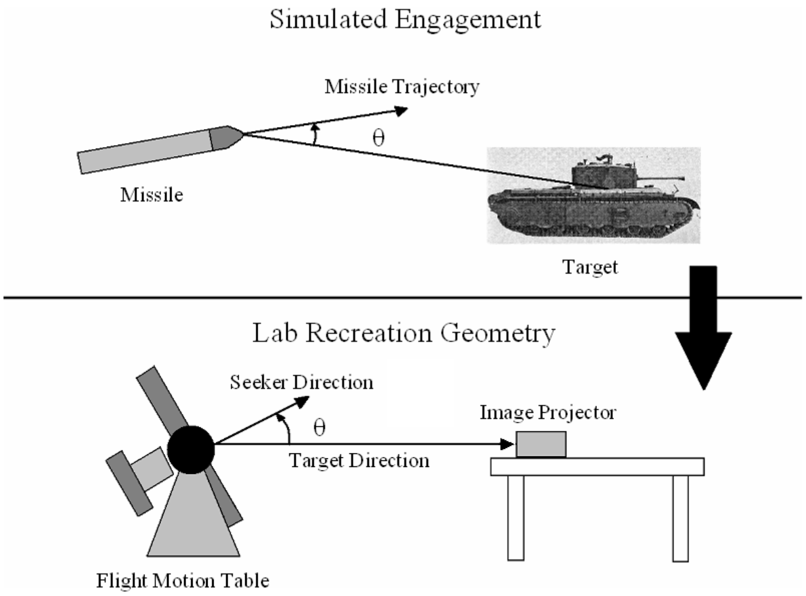

Figure 1.1: Example of a HWIL simulation modeling an actual engagement.

1.1 Description of TigerSim

Auburn University began to develop TigerSim in the fall of 2006. It consists of a series of subroutines that are modeled from the physical systems of an interceptor missile, as shown in Figure 1.2. A short description of each subroutine follows the figure. The simulation code was developed in Matlab, a computer programming language used frequently in science and engineering. It was a pure-digital simulation that possessed no hardware interfaces or graphical outputs.

TigerSim contains twelve state variables, which completely describe the state of the interceptor. The state variables are: the missile position in inertial coordinates, (X, Y, Z); the missile orientation angles, (ϕ, θ, ψ); missile velocity, (u, v, w); and the angular rotation rates, (p, q, r). The transformation matrix from inertial to missile coordinates is given in Appendix A. Note that the state variables ϕ, θ, u, v, and p differ from the image variables u, v, and $\mathbf{p}$ and the seeker orientation angles ϕ and θ both discussed in Chapter 3.

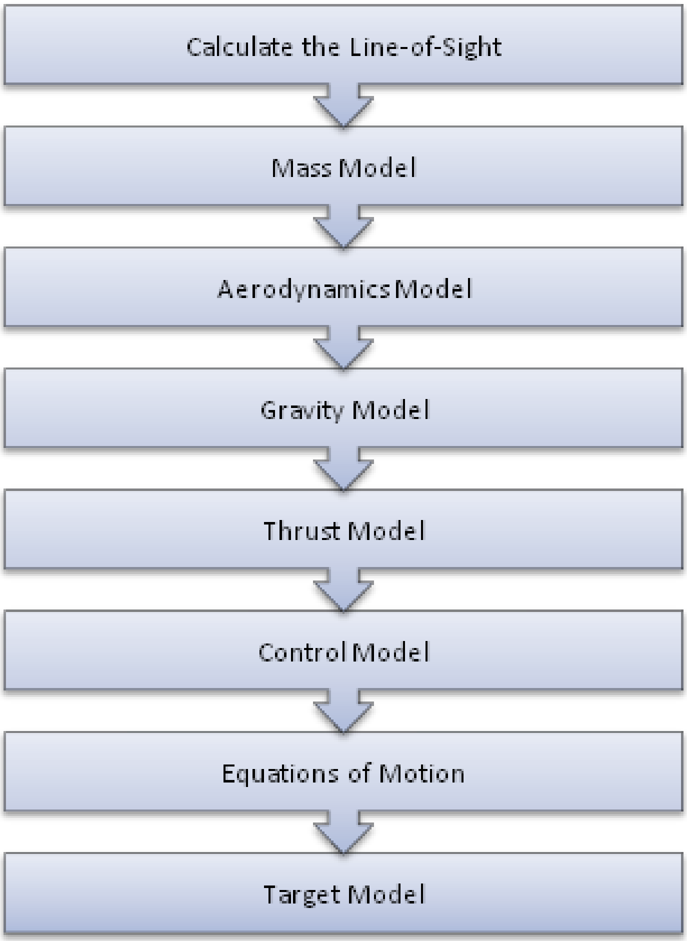

Figure 1.2: Flow chart showing the TigerSim subroutines.

Calculate the Line-of-Sight: This subroutine computed the Line-of-Sight (LOS) vector, and relied on perfect knowledge of the location of the target and interceptor. It served as a simple idealized seeker model in the TigerSim simulation. The LOS is the unit vector aligned with the position vector from the interceptor to the target as shown in Figure 1.3. The LOS vector is used by the missile's guidance system to adjust the interceptor's trajectory, directing the interceptor to strike the target.

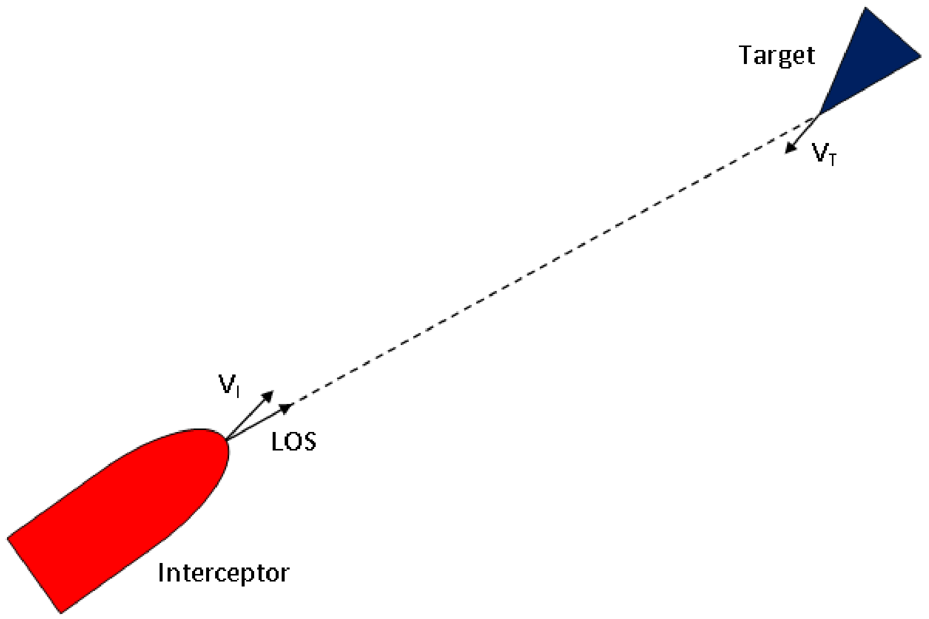

Figure 1.3: The LOS vector between the interceptor and target.

Mass Model: As the rocket burns fuel and the boost stage separates from the kill vehicle, the mass properties of the rocket change during flight. This subroutine computes the total mass of the interceptor, the location of the center of gravity, and the moments of inertia.

Aerodynamics Model: The aerodynamics model relies on a series of lookup tables generated by Missile DatCom, a computer program written by the US Air Force. Atmospheric properties such as temperature and density are a function of flight conditions, and are used to determine the aerodynamic forces and moments acting on the missile.

Gravity Model: This subroutine determines the gravitational forces that act on the missile during flight.

Thrust Model: The thrust produced from the interceptor's booster motors consist of three linear segments over the first 20 seconds of flight. The thrust, initially at zero increases to 4,000 N in one second and further increases to 6,000 N over the next 16 seconds. The thrust then decreases to zero over a three second span.

Control Model: The LOS vector is passed to the control subroutine which uses proportional-navigation and attitude-control algorithms to direct the divert thrusters that steer the interceptor toward the target.

Equations of Motion: This subroutine uses the parameters from the above algorithms in a series of equations to update the interceptor state forward in time during the flight.

Target Model: The position of the target is updated as it follows a pre-determined path through the atmosphere.

1.2 Proposed Additions

There were two goals of this research project. The first was to increase the simulation capability by augmenting the system with a scene generation program. This would produce a visual representation of what the missile's seeker would observe during an intercept scenario.

The second goal was to develop a target detection program that simulates a missile seeker. Instead of relying on an idealized seeker model, the target detection program would calculate the observed LOS based on the image produced by the scene generation program. This would allow for a digital simulation of a missile intercept. A similar target detection program was installed on a commercially available smart camera to allow for static HWIL simulations. This allows for the integration of physical components into the digital simulation, as an intermediate step to full HWIL simulations. The following sections describe the development of the scene generation system along with the physical and mathematical aspects of the target detection program.

Chapter 2

Scene Generation

The purpose of a scene generation program is to stimulate a missile's optical sensor. Therefore, it is vital to model the intercept environment as accurately as possible. The simulation must be able to match the seeker's physical parameters such as field of view and resolution, in addition to displaying the target and its background. This chapter describes the development of the scene generation program and concludes with several images of the scene generation program during flight.

2.1 Creating an Artificial Environment

In the TigerSim simulation, the intercept takes place at an altitude of approximately 78,000 meters. While not having reached the boundary of space, most optical effects from the earth's surface and atmosphere are negligible at such an altitude [2]. Therefore, the simulation did not include a representation of the earth's surface or a scattering model from light transversing the atmosphere. In addition, there were no exogenous sources of light from the sun, moon, or other spacecraft. The scene is developed, however, from visual models of the target and background environment of the particular missile application. In this work, a ballistic-warhead target and night-sky background were incorporated.

The missile's seeker was assumed to detect light in infrared frequencies between 0.3 and 0.5 μm. This corresponds to an "optical window," specifically, a frequency of light with high atmospheric permeability. Military infrared sensors commonly use this band for airborne target acquisition of missiles and aircraft. The output from the scene generation program is a gray-scale image, which can be displayed on a hardware device, such as an infrared projector. The generated image represents the intensity of light received in each part of the image.

Objects modeled in the simulation include the stars of the night sky set against the void of space. Using Matlab's rendering tools, a black background was created. Next, an artificial field of stars was added to model stellar infrared emissions. The generated stars will force the target detection program to distinguish between the target and its background, as discussed in the following chapters.

The stars were modeled as a pseudo-uniform distribution of 200 points on a sphere. The distribution was created by modeling each point as a positive charge, and using the law of repulsion, the positive charges distributed themselves on the sphere's surface. The resulting celestial sphere is shown in Figure 2.1. The camera is located essentially at the center of the sphere and looks outward; therefore, at any instant only a small section of stars can be seen.

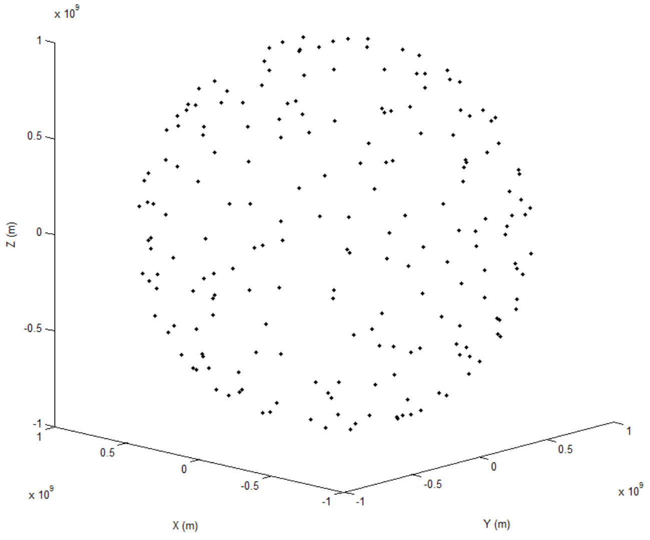

Figure 2.1: The spherical star field used to represent the night sky.

2.2 Modeling the Target Characteristics

The target is the next and final portion of the scene to be generated. The target dimensions were modeled on the warhead of the United States' LGM-118A Peacekeeper Intercontinental Ballistic Missiles (ICBMs) which entered service in 1986. This ICBM contains 10 re-entry warheads, each approximately 1.0 meters in diameter and 2.25 meters tall [3]. The target's position in the TigerSim simulation is controlled by the Target Model algorithm.

Objects entering the Earth's atmosphere typically experience temperatures between 900 to 1200 degrees Celsius [5]. Objects this hot emit strongly in the infrared spectrum of light, making them easier to discriminate against the atmosphere. The nose of the target would experience the hottest temperatures, due to the location of the stagnation point. Points on the cone's surface further from the stagnation point would experience cooler temperatures. This temperature gradient produces a corresponding infrared gradient, which was modeled in Matlab.

A custom colormap was developed, to model the target's infrared characteristics. The intensity at the beginning and end of the colormap is defined. Matlab then interpolates between those values at every point on the surface, to produce the desired color distribution, as shown in Figure 2.2. White at the tip and a medium gray at the base were chosen to produce a representative distribution of infrared intensity of the target. The gray color that was chosen to terminate the colormap was the same gray color used as the stars. The complete target is shown in Figure 2.3.

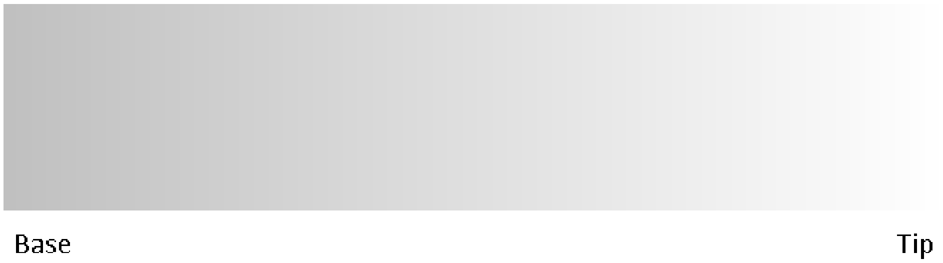

Figure 2.2: Colormap developed in Matlab.

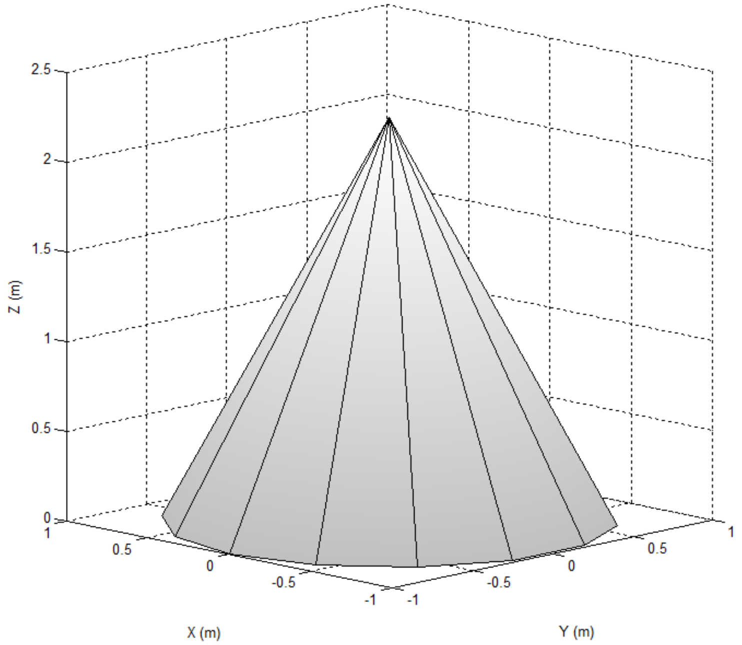

Figure 2.3: Visual target model.

2.3 Rendering Process in Matlab

The portion of the artificial environment which is displayed is determined by several factors, including the missile position and orientation, and the seeker gimbal angles and the field of view (FOV). The missile seeker has a gimbaled sensor that is actively steered by the seeker gimbal angles. The FOV is the angular extent of the outside world that can be observed at a single time, as shown in Figure 2.4. In HWIL testing, the FOV of the generated scene needs to be matched to the FOV of the seeker. In TigerSim, the scene generation process was made adjustable to allow for the testing of different seekers. The experiments performed in this report used a FOV of 20 degrees, to match the physical properties of the camera, which is discussed in more detail in Chapter 4.

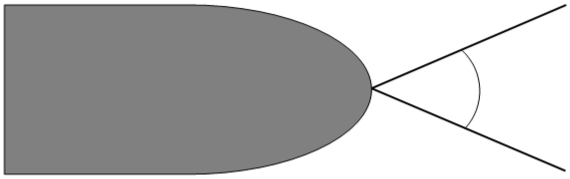

Figure 2.4: The field of view is the angle the seeker is able to observe.

Figure 2.5 shows the missile seeker and the gimbaled sensor. In the figure, the missile coordinate frame is given by (X, Y, Z), and is fixed to the missile. The gimbaled sensor is free to rotate within the seeker and is actively steered to keep the sensor boresight pointed at the target. The seeker frame (x,y,z) rotates with the sensor. The sensor steering is dictated by the azimuth ϕ and elevation θ angles. These angles are determined by the seeker-steering algorithm as described in Section 3.3.

The inputs to the scene generation program are the locations of the target and interceptor in three dimensional space, the interceptor orientation, and seeker gimbal angles. The scene generation program then builds the observed target around the

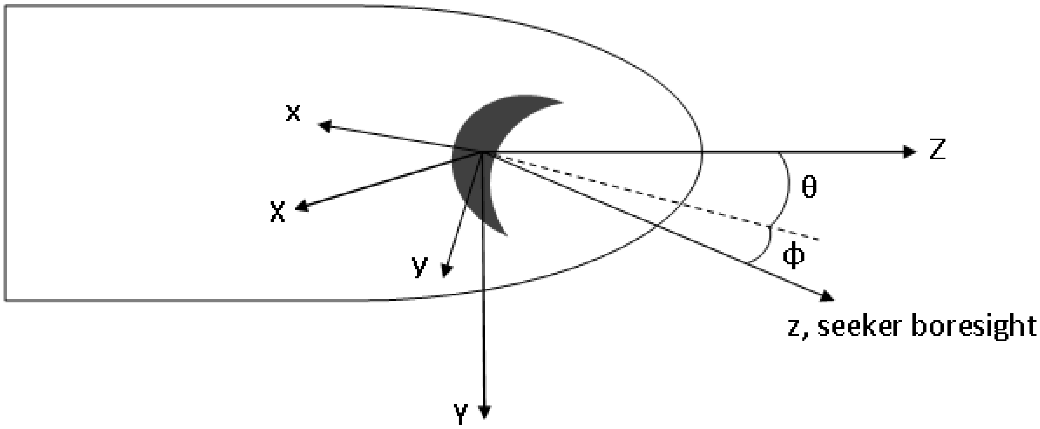

Figure 2.5: Missile coordinate frame (X, Y, Z) and seeker coordinate frame (x, y, z). The gimbaled sensor inside the missile seeker is oriented through the azimuth (ϕ) and elevation (θ) angles.

point in space. The target is centered horizontally and vertically as shown in Figure 2.6. The cone representing the target is constructed out of 14 triangles projecting radially from the vertex. The base of the cone is a 14-sided polygon.

At great distances, the size of the target may be no larger than a pixel. Also, the target location may not be centered on a screen pixel, causing the target intensity to be spread over several pixels. When Matlab renders the scene, the target may be too faint to be detected. This problem cannot be easily corrected by adjusting the Matlab rendering process. Instead, a small fiduciary marker was plotted on top of the target whenever the range between the target and the interceptor was greater than 600 m. This ensures that the target is visible at large distances. For ranges less than 600 m, the target is sufficiently large that this step is not necessary.

In the TigerSim simulation, the target and the interceptor move independently of each other in three-dimensional space. For the greatest simplicity, the scene generation program would plot the scene features in inertial coordinates and place a virtual camera at the seeker's position and orientation. However, the graphical ability of

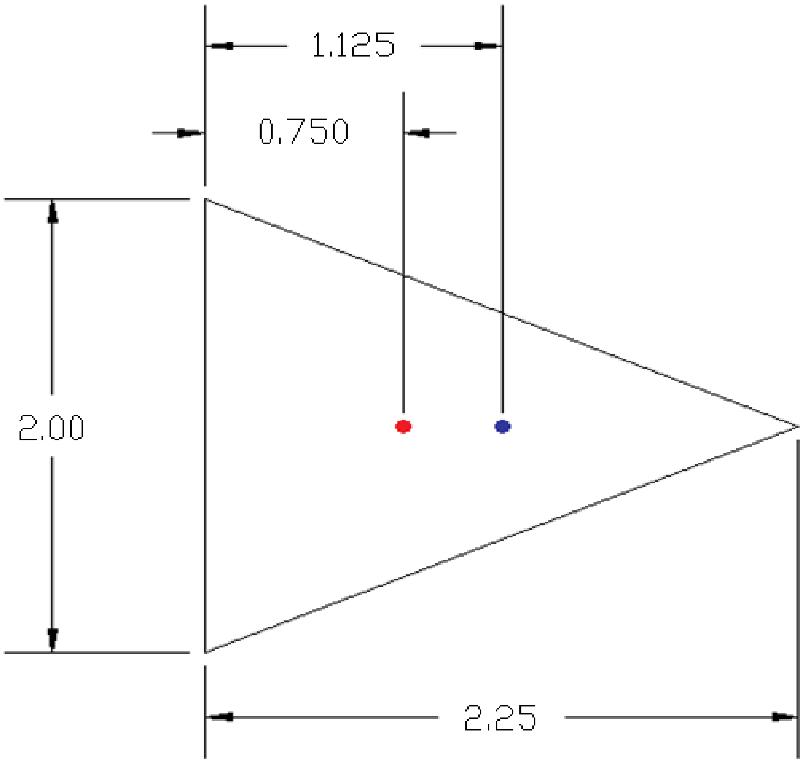

Figure 2.6: The geometric center of the target (red circle) and the point on the body representing the simulation target location (blue circle). Units are in meters.

Matlab prevents this. Instead, a “scene frame” is created. The scene frame is aligned with the seeker frame, but the origin is located at the target. The virtual camera translates and rotates around the target, mimicking the relative position and orientation of the target relative to the interceptor, as determined by the Target Model and Target Detection subroutines. Because of this, the scene generation program computes a transformation matrix from inertial coordinates to seeker coordinates, which are used in the scene generation and target detection subroutines. The transformation matrix is the product of the matrices given in Appendix A.

2.4 Display Setup

The scene generation program was intended to model an infrared scene. For lower cost and complexity the actual hardware used here for HWIL simulations operated in the visual region. For these simulations, the generated scene was displayed on a 18.1 inch liquid crystal display (LCD) monitor. The native resolution of the monitor is the SXGA standard of 1280 horizontal pixels by 1024 vertical pixels. All experiments were performed with the monitor set to its native resolution. The generated scene had a resolution of $n_x = n_y = 900$ pixels. This was done to ensure the displayed pixels remained square. The display pixels have coordinates of (u, v), with the origin in the upper left corner of the scene. Pixel coordinates are discussed in more detail in Section 3.2. Figure 2.7 shows the display figure window on the monitor screen.

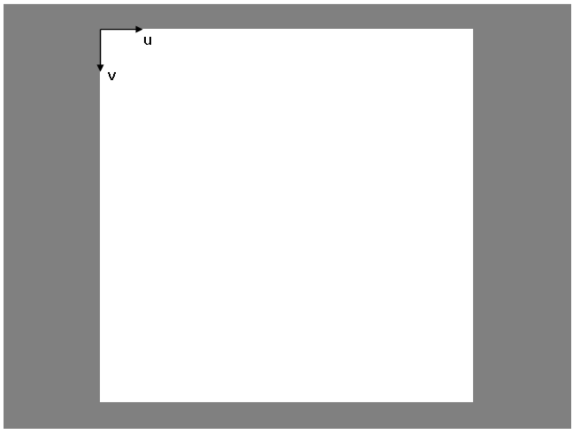

Figure 2.7: Monitor resolution of 1280 by 1024 pixels and figure window resolution of 900 by 900 pixels.

2.5 Examples of Generated Scenes

The following figures provide an example of the scene generation capabilities of TigerSim. All figures have a resolution of 900 pixels by 900 pixels. The target is located in the center of each image, with a portion of the artificial celestial sphere in the background.

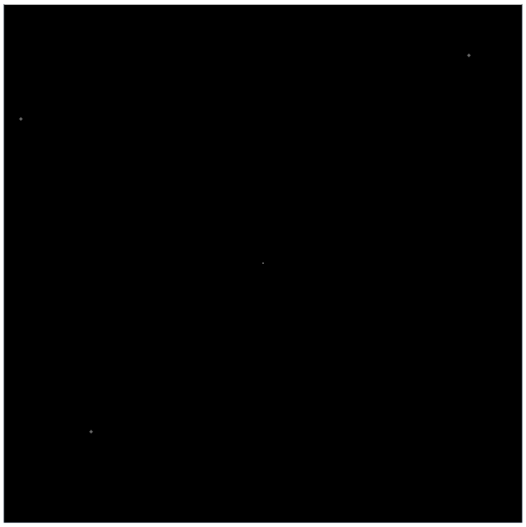

Figure 2.8: The rendered target scene ten seconds prior to intercept.

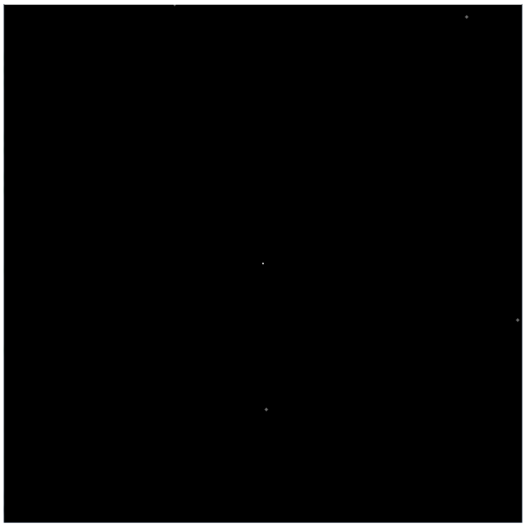

Figure 2.9: The rendered target scene one second prior to intercept.

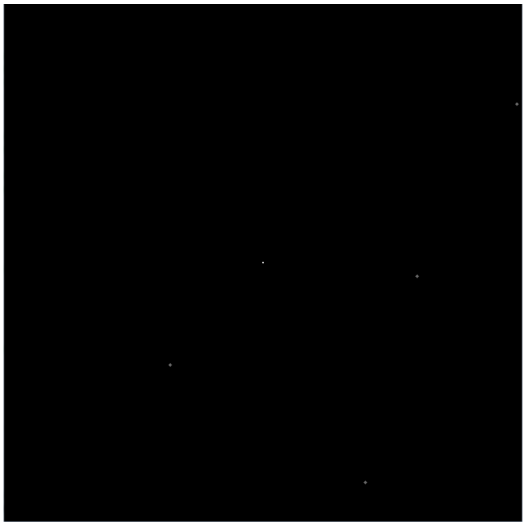

Figure 2.10: The rendered target scene one half of a second prior to intercept.

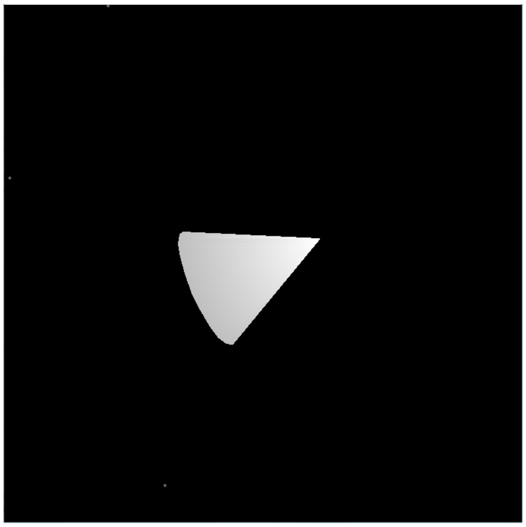

Figure 2.11: The rendered target scene one tenth of a second prior to intercept.

CHAPTER 3

SEEKER MODEL: SOFTWARE-IN-THE-LOOP

Two different methods of target detection were developed in the Auburn Hardware-in-the-Loop lab to replace the LOS calculation subroutine found in previous versions of TigerSim. This chapter describes the pure digital simulation, while the next chapter describes the addition of a smart camera into the target detection program.

3.1 Target Detection

A Software-in-the-Loop (SWIL) target-detection module was created to simulate the behavior of a missile seeker. This allowed for digital simulation by directly capturing the rendered scene. In developing the target detection algorithm, it was assumed that the engagement scenario involved a single interceptor and target, and that the target could not deploy decoys or other countermeasures. Therefore, the target detection subroutine can locate the target by determining the value of the highest intensity pixel, and locating all pixels with that intensity value.

The target was located by analyzing each pixel in a two-dimensional rectilinear pattern, also called a raster scan [2], as shown in Figure 3.1. The intensities of all pixels were compared to find the highest intensity. All pixels with that intensity were identified as the target and their pixel locations (x_i, y_i) were extracted. The centroid of the target is determined from the pixel coordinates by Equation (3.1). Here n is the number of pixels with the brightest intensity and u and v represent the pixel locations of the target centroid.

$$
\begin{aligned}
u &= \frac{1}{n}\sum_{i=1}^{n} x_i \\
v &= \frac{1}{n}\sum_{i=1}^{n} y_i
\end{aligned}
\tag{3.1}
$$

Figure 3.2 shows a sample intercept scene with the target centroid marked as a red cross as determined by the target detection program. As can be seen, this detection algorithm focuses on the brightest part of the target, the nose.

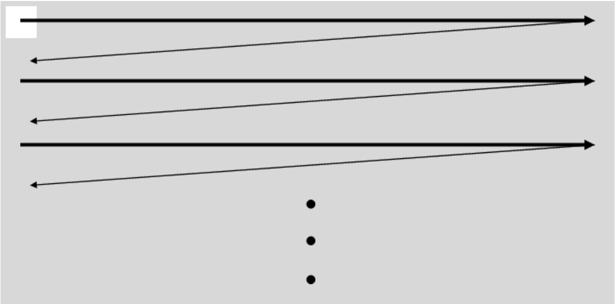

Figure 3.1: Raster scan pattern.

The complete Raster scan of an image is computationally expensive, greatly slowing down the simulation. Therefore, a modified procedure was implemented after the initial image detection, as shown in Figure 3.3. Since the seeker is actively steered to keep the target in the center of the field of view, the target will be located near the center of the image. Using this fact, the 100 pixels in the middle of the image are initially scanned. If an object is detected that has the same or higher intensity value as the initial scan, it is assumed that the target has been located. If no object is found, the entire image is analyzed to re-locate the target. This method reduces the search time by approximately two orders of magnitude.

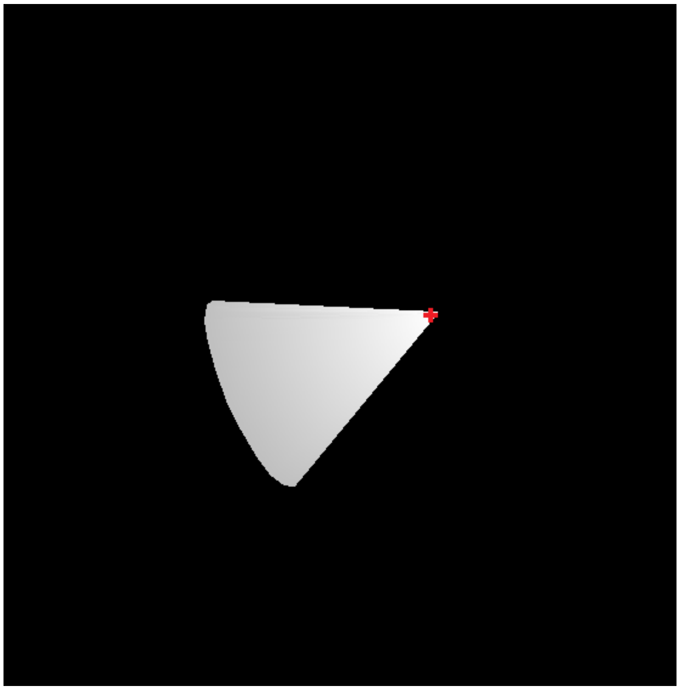

Figure 3.2: Intercept scene with target centroid marked as determined by the target detection algorithm.

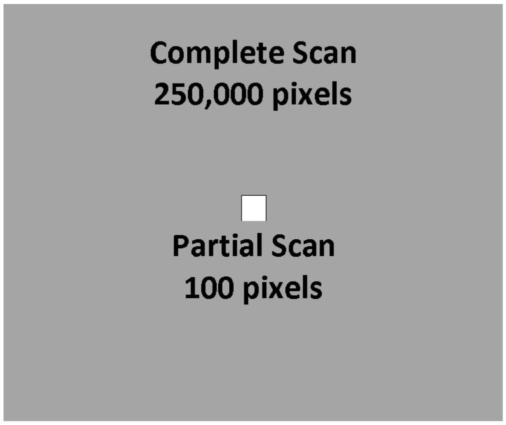

Figure 3.3: Reduction of scan area.

3.2 Derivation of the LOS vector

Previous subsections outlined the methods for scanning the image to locate the pixels that represent the target, in pixel coordinates. The guidance system used in the missile simulation requires the LOS vector to be re-defined in inertial coordinates, in order to steer the missile toward the target. In addition, the azimuth and elevation of the target are required to orient the missile seeker. This section outlines the method used to extract the LOS vector based on the sensor model.

Figure 3.4 shows a pinhole camera. Light from the imaged object (far right) enters the camera at the projection point and is collected by the camera, forming the image. The object is inverted on the image plane, therefore it is common to consider an equivalent image, as shown in the figure. This gives the advantage that the image and the imaged object have the same orientation. The equivalent image will be used in this work.

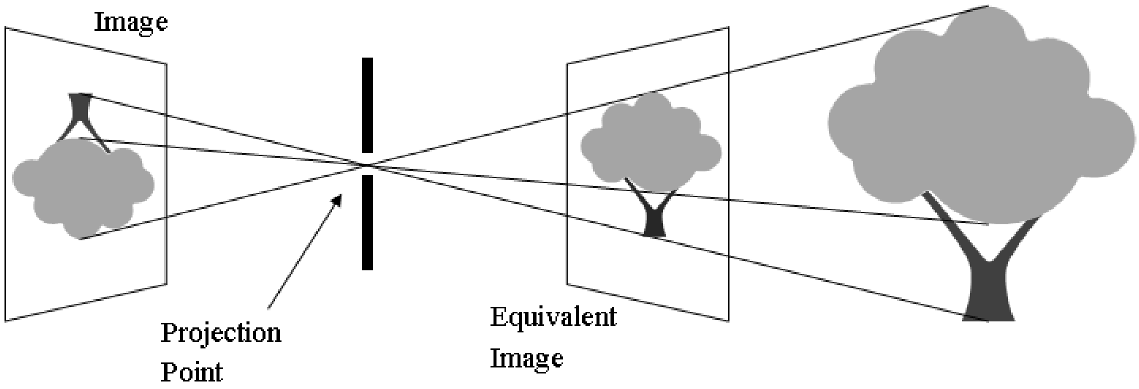

Figure 3.4: Pinhole camera model.

To determine the LOS vector, geometric coordinates of the image must be defined. To describe the position of the image, a set of physical coordinates (x, y, z) are defined as well as the pixel coordinates (u, v), as shown in Figure 3.5. The coordinates u_o and v_o denote the center of the image in pixel coordinates. A target can be described in either pixel coordinates, **m**, or camera coordinates, **p**. Because of the range ambiguity from the image, both sets of coordinates are normalized so that the third coordinate equals one.

$$\mathbf{m} = \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \end{bmatrix} \tag{3.2}$$

$$\mathbf{p} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3.3}$$

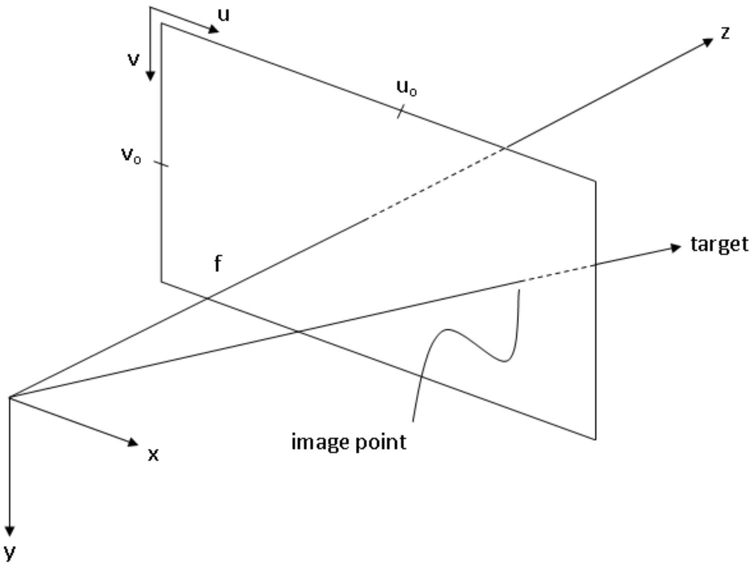

Figure 3.5: Relationship between geometric and physical coordinates.

The target detection program provides the target centroid, u and v in pixel coordinates, $\mathbf{m}$, while the LOS vector is written in physical coordinates, $\mathbf{p}$. These two values are related through the intrinsic matrix. First, the FOV, u_o, and v_o, mentioned in previous sections are used to determine the number of pixels per focal length in Eq. (3.4).

$$\begin{aligned} \sigma_x &= \frac{u_o}{\tan(\frac{fov_x}{2})} \\ \sigma_y &= \frac{v_o}{\tan(\frac{fov_y}{2})} \end{aligned} \tag{3.4}$$

The following transformation matrix, known as the intrinsic matrix, relates the two sets of coordinates [4].

$$\begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \end{bmatrix} = \begin{bmatrix} \sigma_x & 0 & u_o \\ 0 & \sigma_y & v_o \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3.5}$$

The LOS unit vector can be constructed as shown in seeker coordinates.

$$LOS_{seeker} = \frac{1}{\sqrt{(\frac{x}{z})^2 + (\frac{y}{z})^2 + 1}} \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \end{bmatrix} \tag{3.6}$$

Using the transformation matrices found in Appendix A, the LOS vector in Equation (3.6) is converted into inertial coordinates to steer the missile.

3.3 Seeker Steering

As mentioned, the seeker contains a gimbaled sensor. During a missile's flight, it is important that the sensor remains pointed at the target in order to provide proper guidance commands to the missile. A seeker steering algorithm needed to be developed to model this aspect of missile behavior.

Steering was performed by pointing the z-axis of the seeker frame (the boresight of the sensor) along the LOS direction from the previous time step. The LOS vector calculated in the previous section points toward the target. Azimuth and elevation angles can be used to relate the LOS vector to the missile coordinate frame, shown in Figure 2.5, as they represent the horizontal and vertical angle (ϕ and θ respectively) between the center of the image and the target location. Because the azimuth and elevation angles locate the target in the image, they are used to orient the seeker.

After the target detection algorithm transforms the LOS vector to the missile frame, it performs the additional step of calculating the azimuth and elevation angles, shown in Equation (3.7).

$$\begin{aligned} \phi &= \sin^{-1}\left(LOS_{missile}(3)\right) \\ \theta &= \tan^{-1}\left(\frac{LOS_{missile}(2)}{LOS_{missile}(1)}\right) \end{aligned} \tag{3.7}$$

The azimuth and elevation angles are then stored by TigerSim and used to orient the seeker frame in the next time step. This steering approach does not attempt to anticipate future motions of the target, but its effectivemess is investigated through simulation trials described in Chapter 5.

Chapter 4

Seeker Model: Static Hardware-in-the-Loop

The second series of experiments involved the addition of a camera into the simulation. Static HWIL (SHWIL) simulations were conducted to investigate distributing the simulation over various software and hardware components. For development purposes, a smart camera was used to model a missile seeker head, combining the required optics, detector, and electronics. The inclusion of hardware components provided a higher fidelity HWIL simulation, and is an important stepping stone to full HWIL dynamic simulations.

4.1 Hardware Setup

A Sony XCI-V3 smart camera was used as a seeker model. The XCI-V3 combines a 640 pixel by 480 pixel still camera and a computer using a 400 MHz AMD processor with Embedded Microsoft Windows XP. The camera was used with a Tamron 12mm lens. The generated scene was displayed on an LCD monitor, facing the camera as shown in Figure 4.1. For simulations, the camera was tripod mounted and located so that the rendered scene nearly filled the entire camera FOV.

An important aspect for accurate simulations was to ensure that the camera was level, and that the camera and LCD monitor were properly aligned. A construction level was used to adjust the tripod so that the camera was level, and to ensure the monitor screen was vertical. A laser level was then placed on the camera. By inspecting the cross pattern the laser level displayed on the monitor, as illustrated in

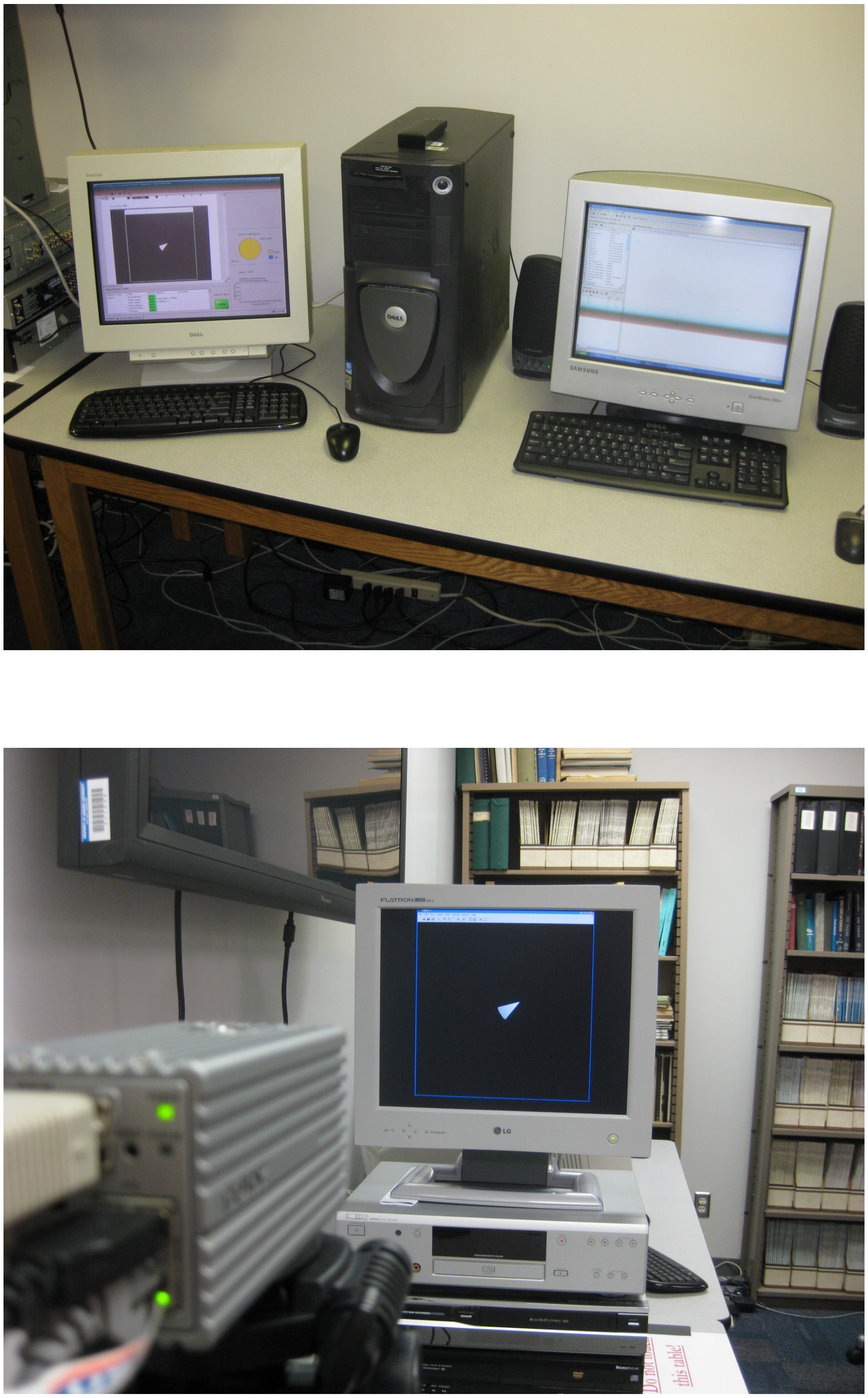

Figure 4.1: Static HWIL laboratory setup

Figure 4.2, the position of the camera was adjusted to be centered on the monitor. The alignment process was iterative, and needed to be repeated several times to ensure accuracy.

4.2 Target Detection Process

After the scene was generated, the simulation PC used a TCP/IP connection to send a trigger to the camera. This started an inspection program written in National Instruments' Vision Builder software. The program acquired and analyzed an image of the generated scene to determine the target centroid.

The SHWIL target detection algorithm used a different technique than the SWIL target detection to determine the target centroid. The inspection program scanned the entire image and located the object which had the greatest intensity. It then constructed a box around the object. The center of the box was used as the target centroid. Once the inspection program was complete, the pixel coordinates of the target, m and n, were transmitted back to the simulation PC by the TCP/IP connection. Figure 4.3 shows the target located in a image captured by the camera. The steps outlined in Section 3.2 to calculate the LOS were carried out on the simulation PC.

The SWIL mainly determined that the nose of the target was the target centroid, as the nose contains the pixels with the highest intensity. The advantage of the SHWIL method is that the transmitted target location is closer to the geometric center of the target. Figure 4.4 shows the transmitted pixel coordinates of the target m and n is the center of the red detection box. The main disadvantage of the SHWIL method is that the orientation of the target relative to the interceptor directly effects the shape of the box, and consequently the transmitted target location.

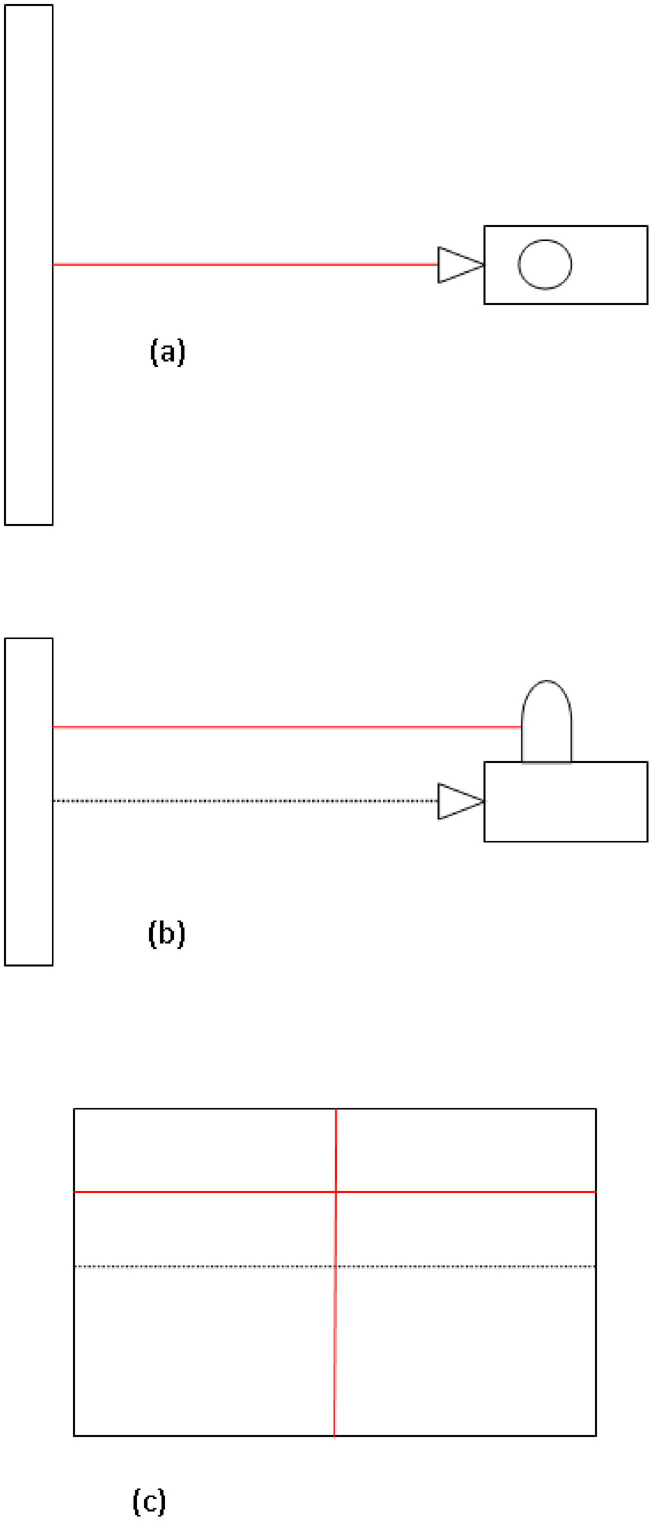

Figure 4.2: Alignment of the television and camera a)horizontal b)vertical c)alignment pattern on the screen

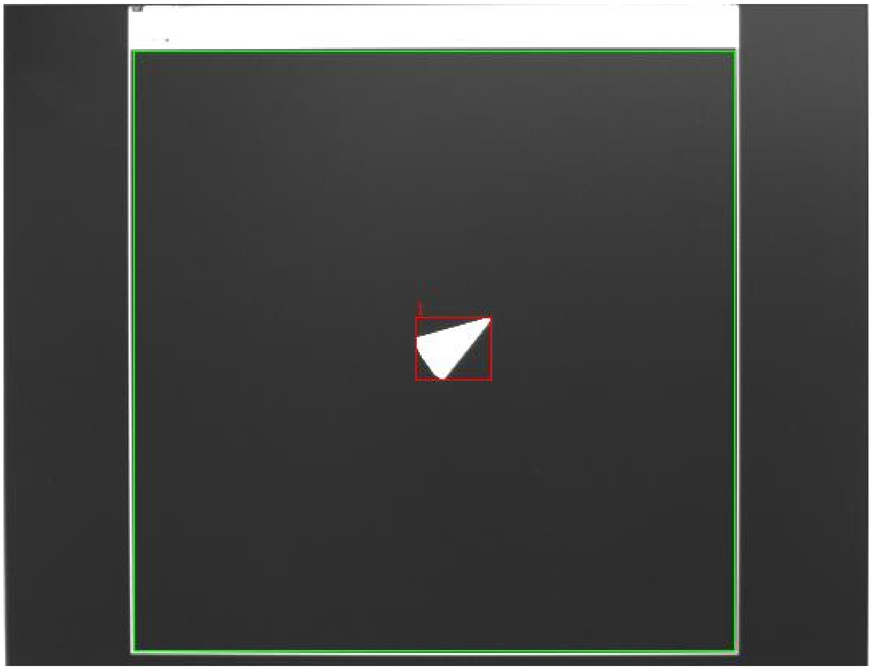

Figure 4.3: Rendered scene captured by the camera showing the target (red box) and the bounding box (green box).

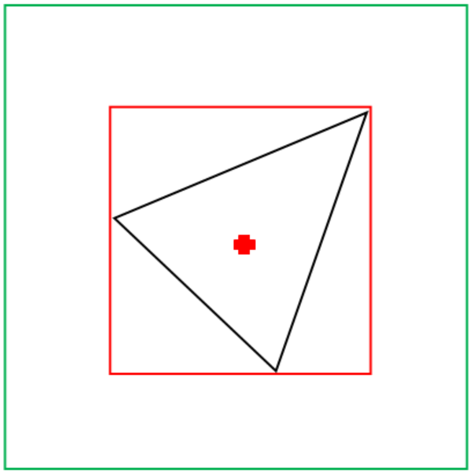

Figure 4.4: SHWIL example showing the detected target (red box) and the transmitted centroid location (red cross)

4.3 Coordinate Mapping

For the static HWIL simulation, the target detection provides the target centroid in camera pixel coordinates. These coordinates have an origin at the upper left corner of a bounding box that was manually placed in the image around the rendered scene. In these simulations, a camera calibration must be used to convert from camera pixel coordinates, (m, n), to screen pixel coordinates, (u, v) as shown in Figure 4.5. Figure 4.6 shows that u coordinates and m coordinates are related through a scaling factor, α as shown in Equation (4.1).

$$u - u_o = \alpha \left(m - m_o \right) \tag{4.1}$$

This scaling factor is computed by first plotting a fiduciary point on the LCD monitor at a known pixel coordinate (u', v'), and measuring the corresponding camera coordinate of that marker, (m', n'). The complete mapping between the two points is given by Equation (4.2). The calibration process was performed with four different markers, and the average calibration value was used.

$$\begin{aligned} u - u_o &= \left(\frac{u' - u_o}{m' - m_o} \right) (m - m_o) \\ v - v_o &= \left(\frac{v' - v_o}{n' - n_o} \right) (n - n_o) \end{aligned} \tag{4.2}$$

After solving Equation (4.2) for u and v, Equation (3.5) was used as in the SWIL simulation.

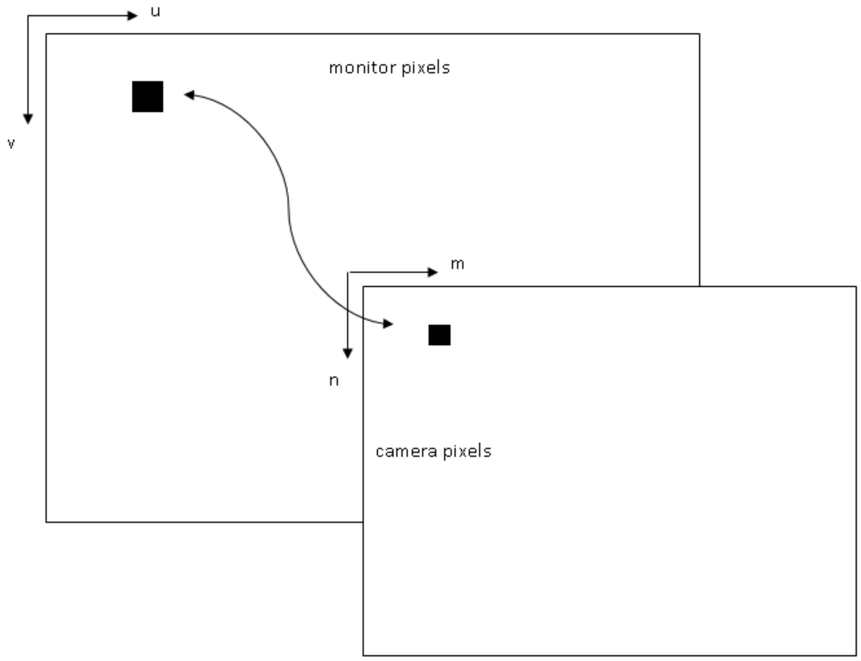

Figure 4.5: Converting generated scene to camera pixel coordinates.

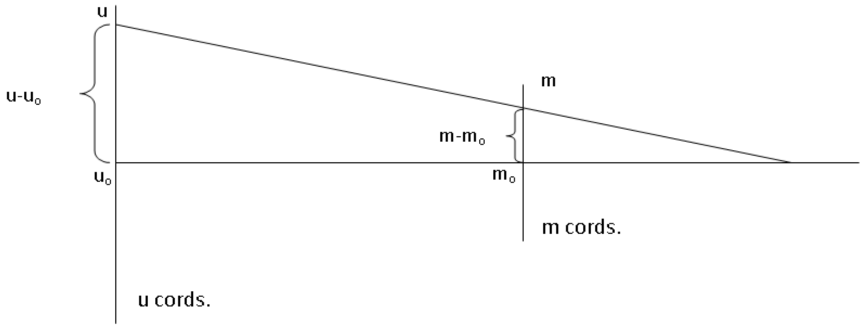

Figure 4.6: Mapping between monitor and camera coordinates.

4.4 Camera Error Sources

Including the camera in the TigerSim simulation introduced errors not present in the digital simulations. While these errors are present in all HWIL simulations, it was necessary to identify and quantify the major error sources in the TigerSim simulation. The major sources of error are discussed below, while the next section discusses a graphical method for analyzing the amount of error present in the HWIL simulation.

4.4.1 Resolution Downsampling

The Matlab figure window, which displays the generated scene, had a resolution of 900 pixels by 900 pixels. The camera captured the displayed scene with a resolution of 373 pixels by 373 pixels. This means that distinct adjoining pixels in (u, v) can correspond to the same pixel in (m, n). Figure 4.7 shows a sample generated scene with a resolution of 900 pixels by 900 pixels, while Figure 4.8 shows the image captured by the camera. Both images are shown to at the same pixels per inch.

4.4.2 Bounding Box

An additional source of error is the selection of the bounding box in Vision Builder. Figure 4.9 shows the origin of the camera pixel coordinates in the upper left corner of the image. This origin does not correspond with the figure window displaying the generated scene. As part of the camera set-up, an inspection area, or bounding box must be selected. This bounding box shifts the origin of the camera pixels so it correlates with the origin of the figure window. The bounding box must be set by hand, and its selection determines the values of m_o, n_o, m, and n in Equation (4.2). Therefore, errors of several pixels can be introduced into the LOS equation.

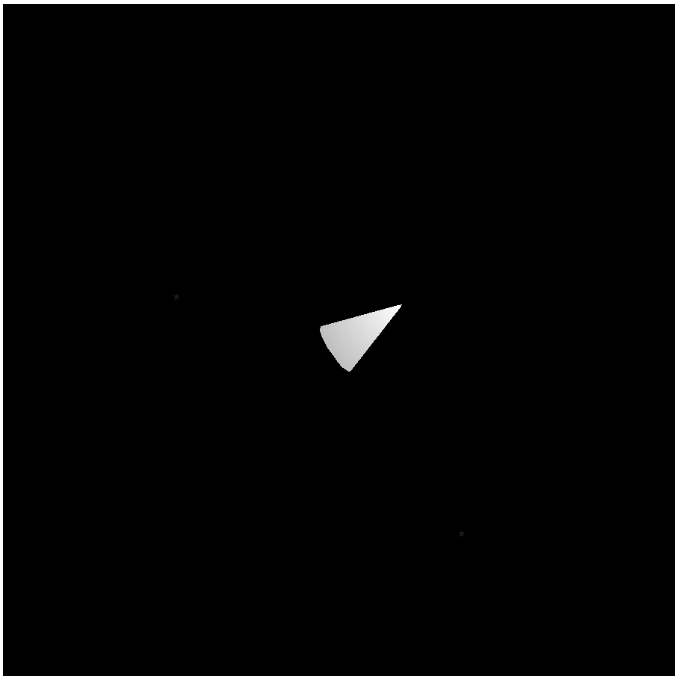

Figure 4.7: Generated image with a resolution of 900 pixels by 900 pixels.

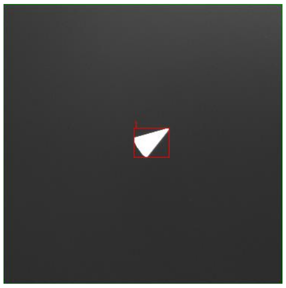

Figure 4.8: Captured image of 373 pixels by 373 pixels.

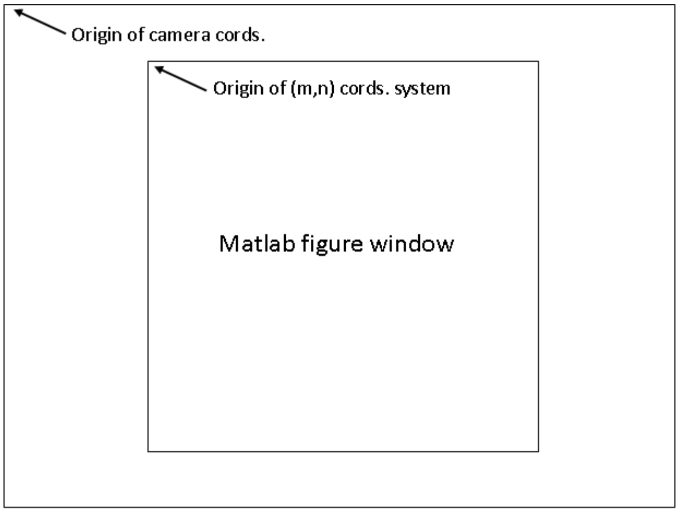

Figure 4.9: Bounding box selects the portion of the camera pixels that contain the figure window.

4.4.3 Lens Distortion

Radial distortion errors, as illustrated in Figure 4.10, is the most significant error source in modern commercial lens. In most applications, the most apparent effect of radial distortion is straight-line objects will appear curved in the captured image [6]. In this experiment, the pixel error in the target location caused by radial distortion is of concern. Several methods to calibrate for radial distortion are available in the literature [7, 8, 9, 10].

4.4.4 Other Sources

The three sources of error mentioned about result from including a camera in the simulation. However, the camera introduces several additional opportunities for error to enter the simulation. The first is an inaccurate alignment of the LCD monitor and the camera. While great care was taken to ensure accurate orientation of the camera, an inadvertent bump could cause either the LCD monitor or the camera to move. This would cause the camera captured image to become slightly skewed.

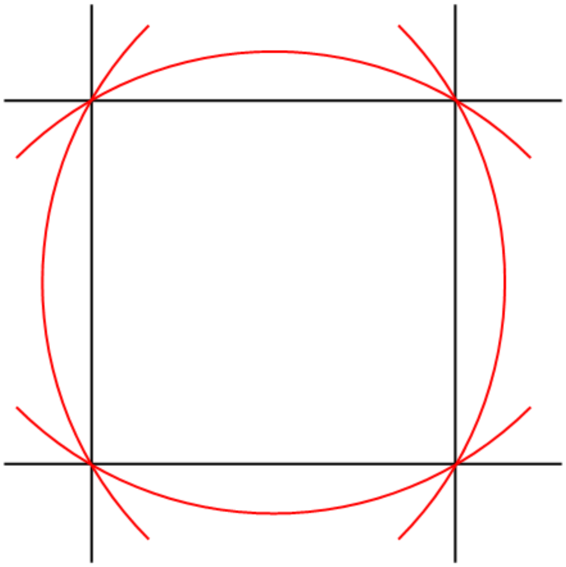

Figure 4.10: Radial lens distortion will alter the original grid (black) to a distorted image (red).

Sources of light not from the LCD monitor could interfere with the camera being able to capture the displayed image. A reflecting from another light source off the monitor could wash out a certain area of the camera. Additionally, non-uniform pixel response from either the LCD monitor or the camera could alter the rendered or captured image. While these other error sources are present, they were mainly judged to be inconsequential in comparison to resolution downsampling, bounding box error and lens distortion.

4.5 Graphical Error Analysis

In order to determine the total amount of error present, a grid was constructed by plotting a control point in 45 pixel increments both horizontally and vertically across the entire image, resulting in 361 points. Using Equation (4.2), the imaged control point from the camera was transformed to screen pixel coordinate and plotted along with the original control point. Figure 4.11 shows the control points as blue crosses and the imaged control point as a red square. In addition, histograms illustrating the amount of error are shown in Figure 4.12 and Figure 4.13. The figures show both an error distribution and pixel bias. The figures did not show a large consistent pixel offset indicating a bounding box error or a large distribution indicating the presents of significant lens distortion. In addition, the error is fairly low near the center of the image, where the target is located for most of its flight. For these reasons, no specific error calibration was performed.

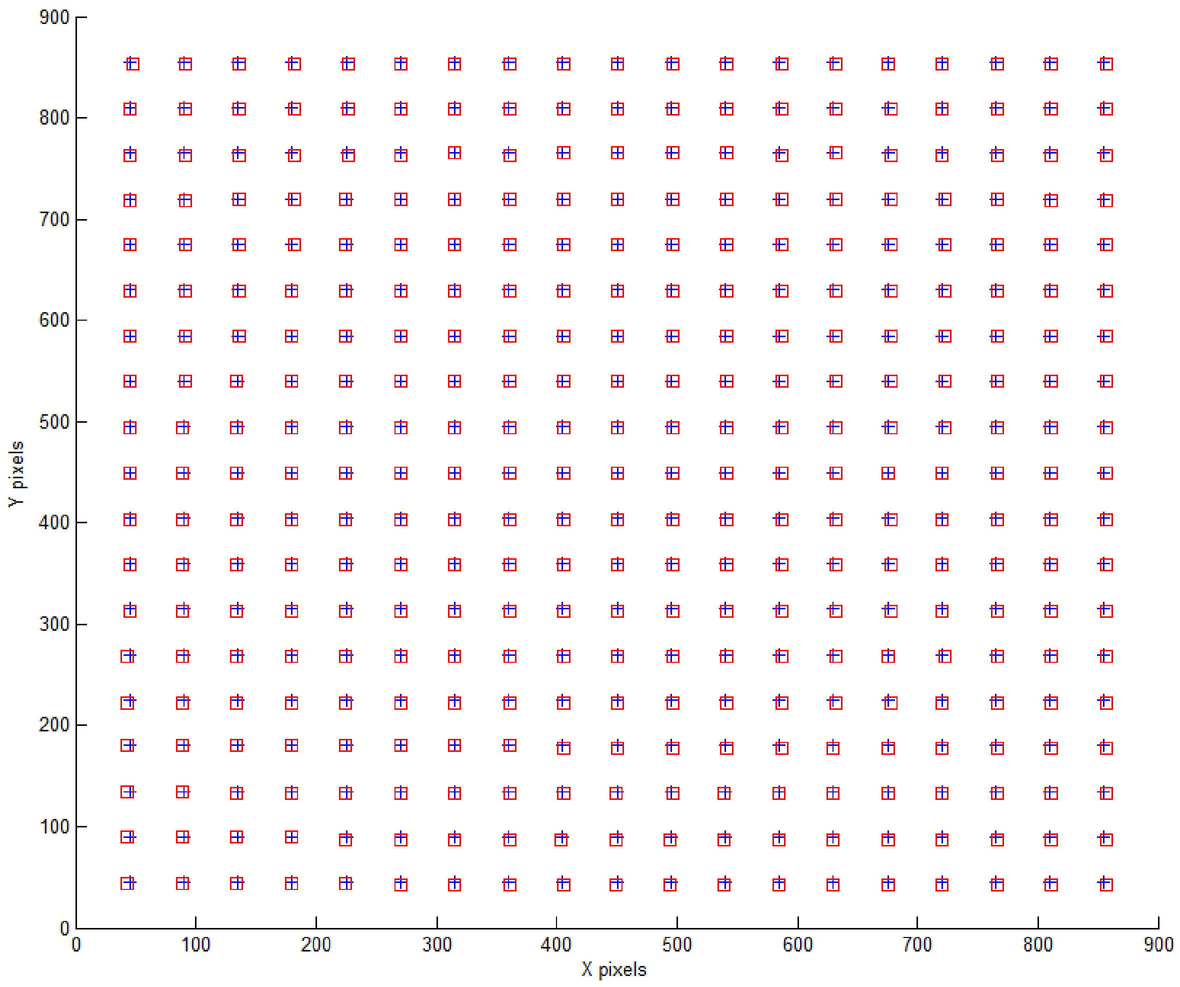

Figure 4.11: Control points (blue cross) and imaged points (red square) for different points on the figure window.

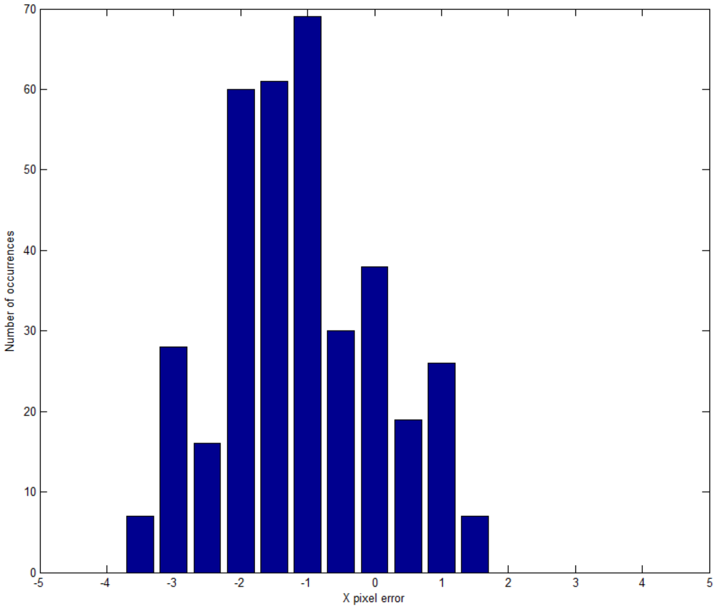

Figure 4.12: Histogram of X error of the control points.

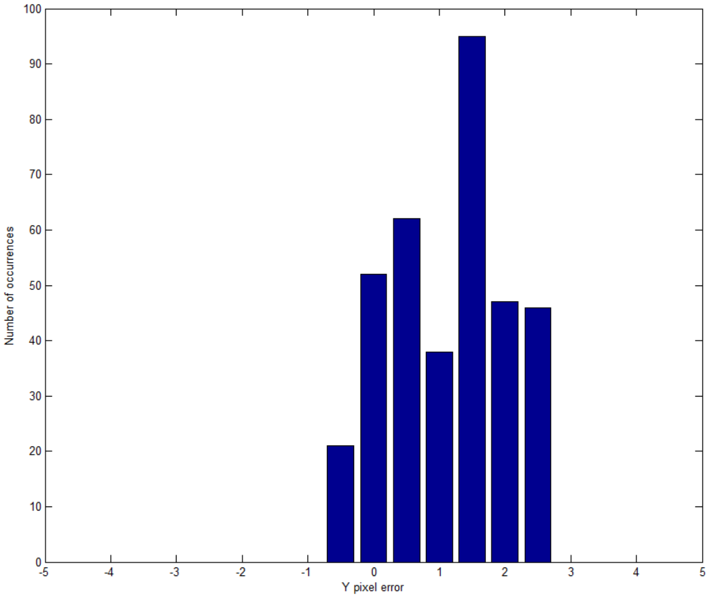

Figure 4.13: Histogram of Y error of the control points.

Chapter 5

Results

In the tested engagement scenario, the target traveled along a constant velocity path, with a constant heading angle. In the target detection program, the point the target passes through after 70 seconds of simulation and the target speed are variables. From this information, the target position is propagated either forward or backward in time. For each trial, variations were introduced to the final intercept location and the target speed. This defined a sphere of possible intercept locations. Variations were sampled from a normal distribution and had a standard deviation of 587 m and 6.05 m/s respectively. The resulting sphere contained locations that were reasonable for the interceptor to reach in 70 seconds. This methodology allowed the interceptor to be launched at t=0 seconds for every trail. Figure 5.1 shows nominal target and interceptor trajectories, as well as a sphere containing all of the intercept locations. The same deviations were used for the SWIL and SHWIL simulations.

5.1 LOS Errors

The following figures present the averaged results of the experiment over 100 trials. Figure 5.2 and Fig. 5.3 show the error in radians between the true LOS vector and the observed LOS vector from the target detection algorithm. Only the final 5 seconds prior to intercept are shown, as the error rate did not significantly change up to that point. Sources of error prevalent throughout the SWIL simulation include finite pixel resolution and the fact that the image centroid is not the same as

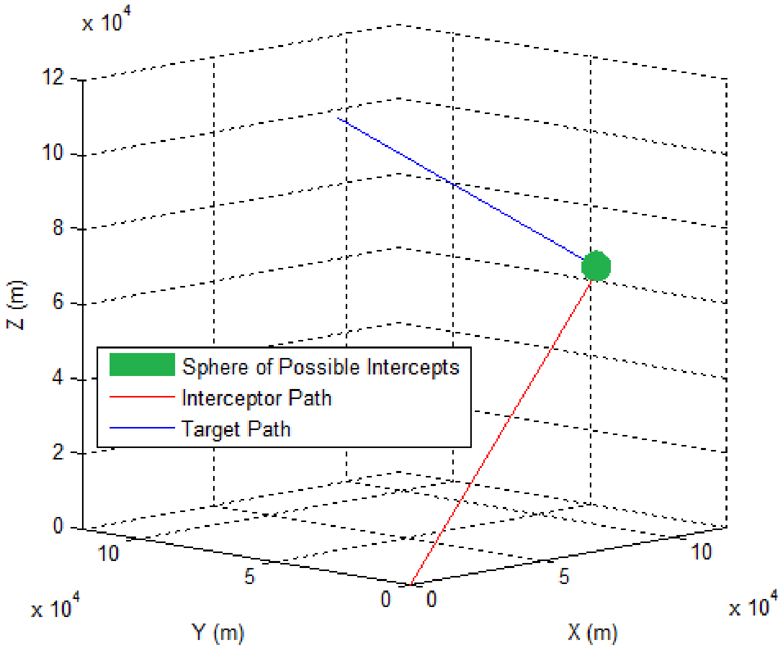

Figure 5.1: Sphere of possible intercepts along with a nominal target and interceptor path.

the geometric centroid. Toward the end of the flight, the angle between the target centroid and the location of the highest intensity pixels toward the nose of target increases, contributing to increase in error over the final 0.5 seconds. The SHWIL simulation had slightly larger error than the SWIL simulation, on the order of a one pixel increase. This increase was expected as the inclusion of hardware into the simulation increases error, and also the fidelity of the simulation. As the increase was small, the additional sources of camera error discussed in Section 4.4 do not have a significant effect on the simulation, further validating the decision not to specifically calibrate for camera error sources.

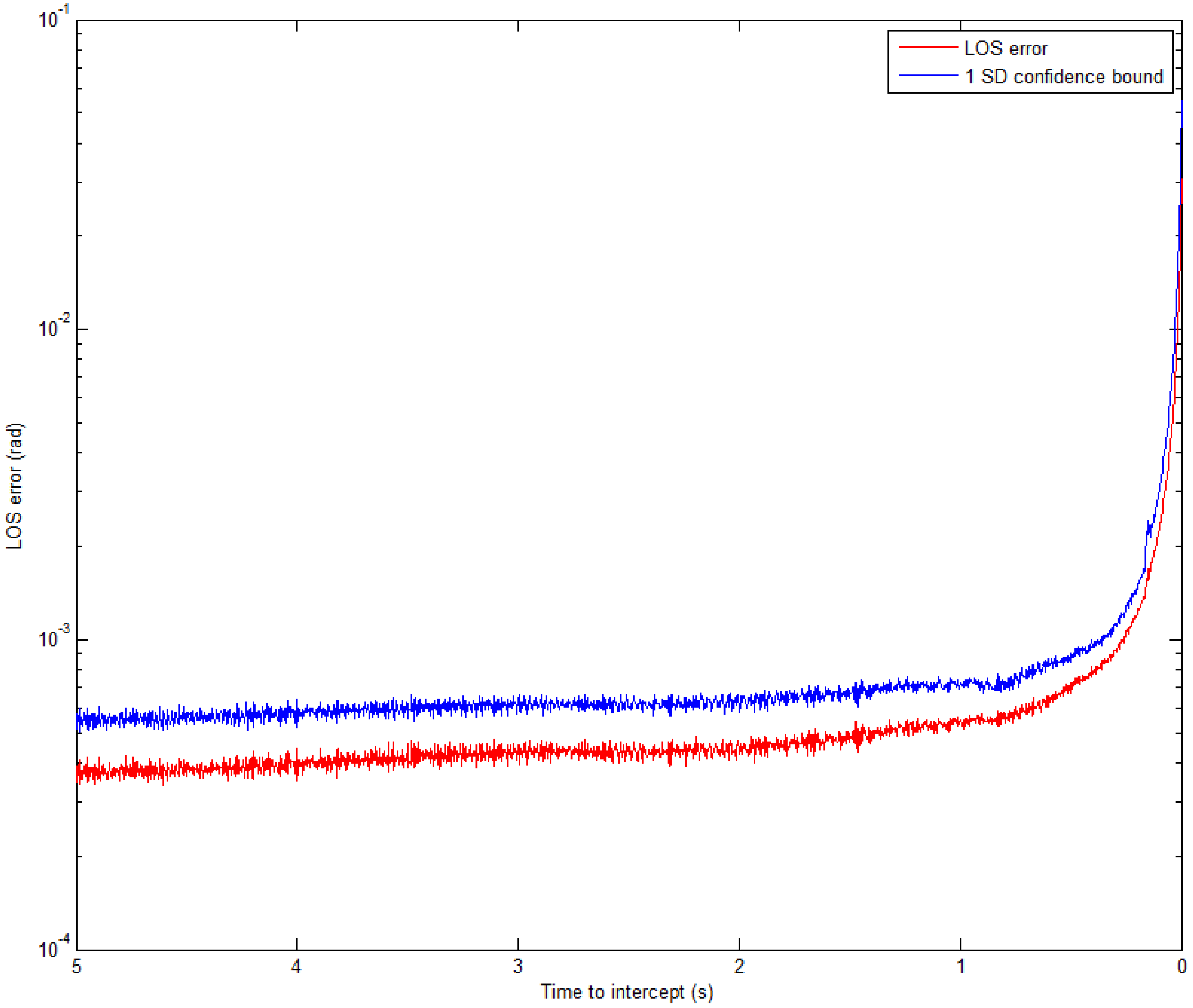

Figure 5.2: Average SWIL LOS error over 100 trials.

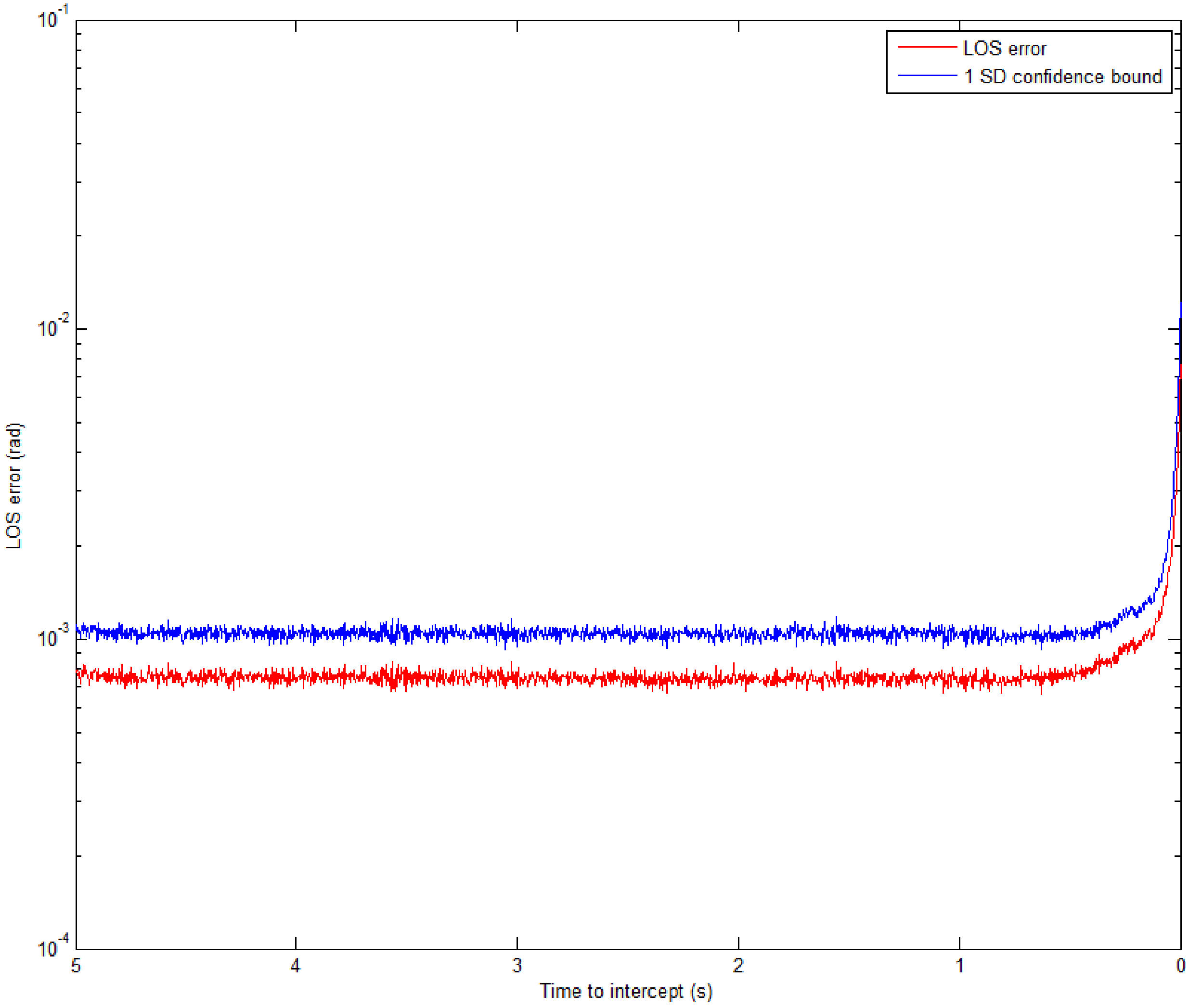

Figure 5.3: Average SHWIL LOS error over 100 trials.

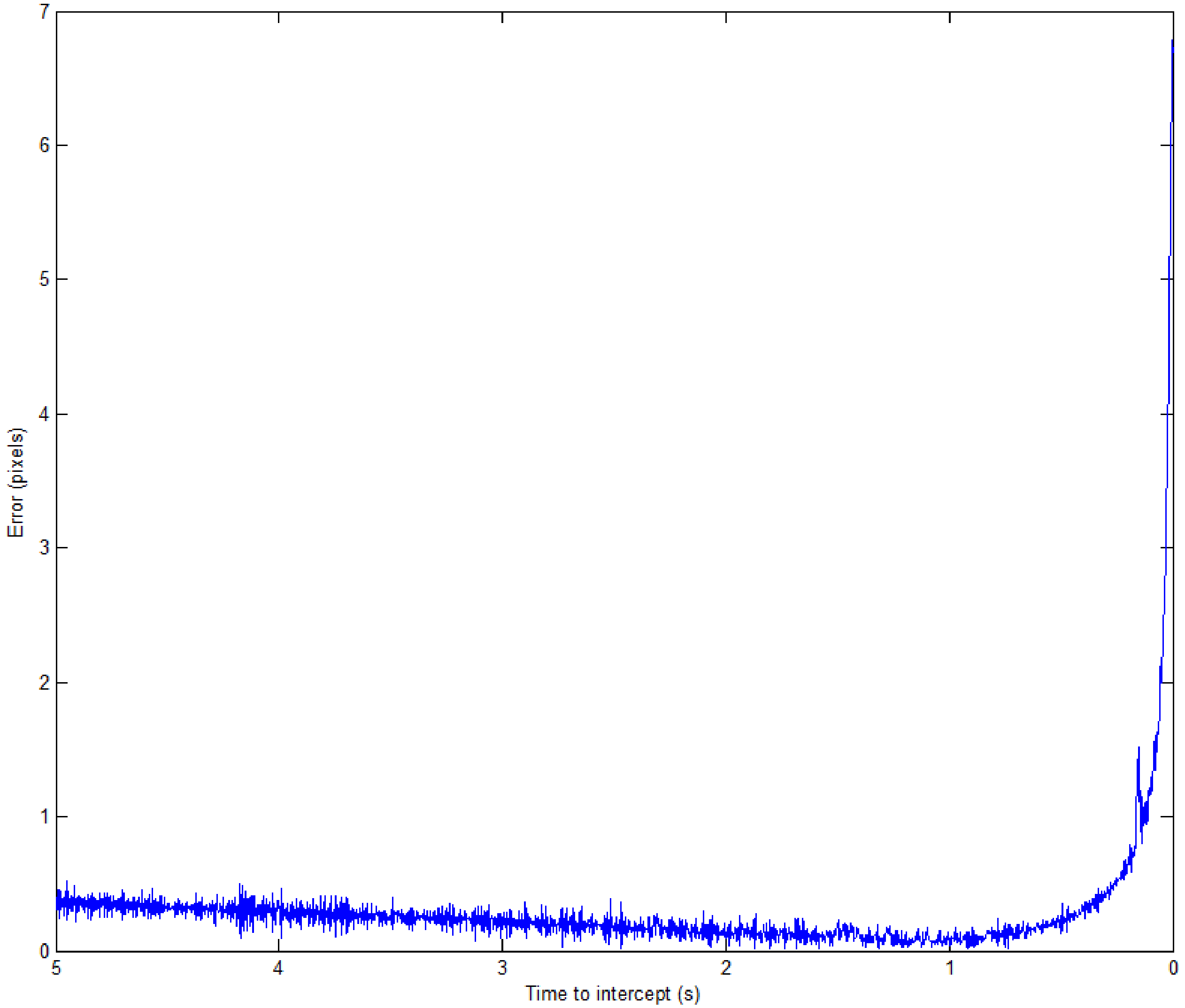

Figure 5.4: Average SHWIL seeker steering error over 100 trials.

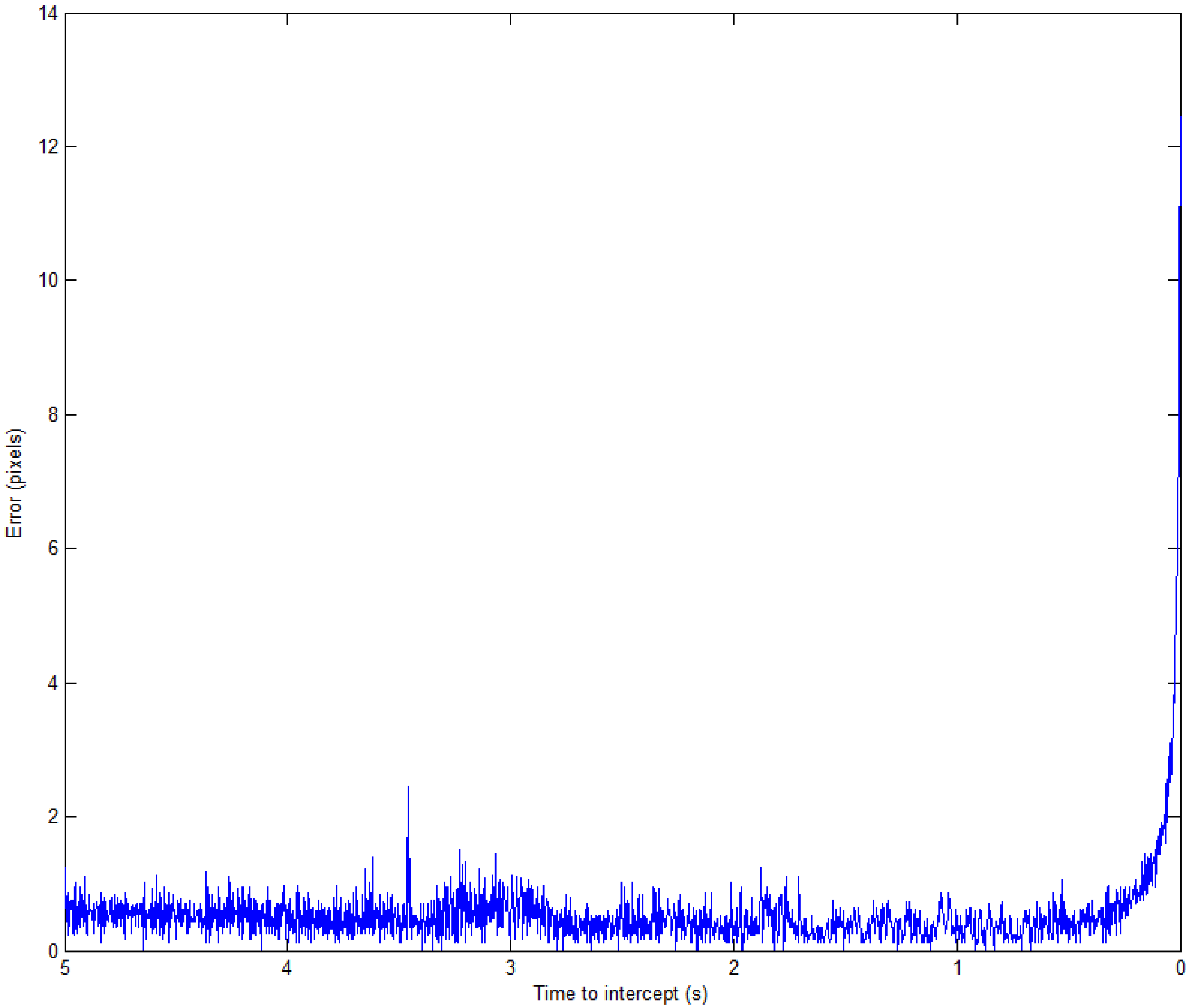

Figure 5.5: Average SHWIL seeker steering error over 100 trials.

5.2 Seeker Steering Errors

Figure 5.4 and Figure 5.5 show the seeker steering error in pixels over the final 5 seconds of the simulation. The seeker is actively steered to keep the target in the center of the image. The plots show the difference between the center of the figure window and the center of the target over the 100 trials. The error is noticeably small for the majority of the intercept, indicating that the seeker steering algorithm is able to keep the target located at the center of the image. At the very end of the simulation, the seeker steering algorithm is unable to compensate for the drastic changes per time step in the target's location. The noticeable increase in error in the SHWIL simulation can be primarily contributed to upscaling the image from camera to screen resolution.

5.3 Intercept Success Rate

Figure 5.6 shows which SWIL intercept simulations were successful in hitting the target. A precise "hit" or "miss" was indeterminable with the data collected, because the interceptor is traveling about 6 m per time step. For each simulation, the interceptor's velocity and the true range between the interceptor and the target was recorded. Using the velocity vector, it was possible to determine the distance traveled by the interceptor per time step. If the minimum range between the interceptor and target was less than the distance traveled per time step, a probable hit was determined. From the recorded data, 96 of the 100 simulations resulted in a probable hit.

For the SHWIL simulations, the target detection software required that the target never touch the border of the bounding box. Therefore, each SHWIL simulation was stopped if the range between the target and the interceptor was less than 8 m,

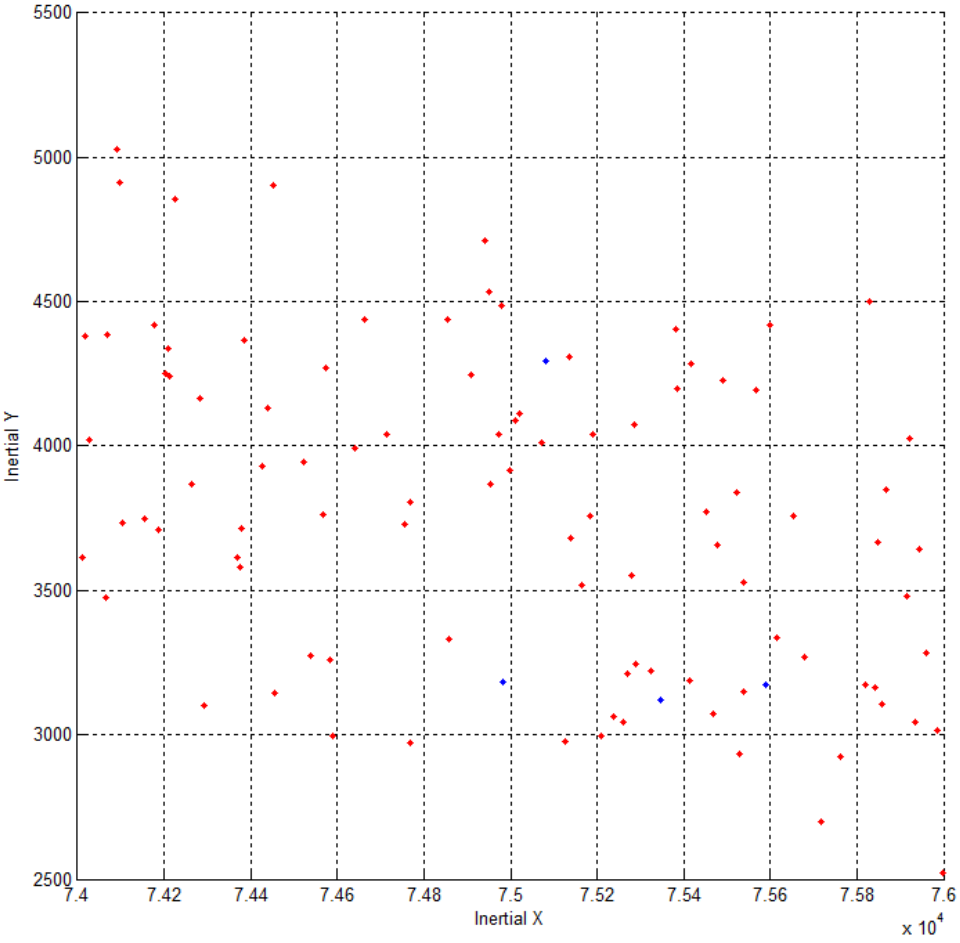

Figure 5.6: Location of SWIL probable hit and miss intercepts.

causing each SHWIL simulation to stop one time step before the SWIL simulations. This premature stop of the simulation prevented probable hit or miss from being determined.

5.4 Simulation Run Time

A final method of comparison is the run time for an approximately 70 second simulation. The SWIL simulations took 26.15 hours to run, for an average time of 15.69 minutes per simulation. The SHWIL simulations took slightly longer, at 28.73 hours total time and 17.24 minutes per simulation. While the SHWIL simulation distributes computing resources over additional hardware, the longer run times can be attributed to the additional steps of converting camera coordinates to screen coordinates and the communication delay between the simulation computer and the camera. This indicates that HWIL simulations are extremely computationally expensive, over seventeen minutes to run a approximately 70 second simulation. For the simulation to run in real-time, purpose-specific hardware and software would need to be developed to significantly reduce run time.

CHAPTER 6

CONCLUSION AND FUTURE RECOMMENDATIONS

The long-term goal in the Auburn University HWIL laboratory has been to develop a missile simulation that can be used for the testing of unclassified hardware. The work presented here described the development of a scene generation and target detection capability for HWIL simulation of missile engagements. These capabilities were developed using widely available software packages and commercial hardware. The success rate of the simulation indicates that the scene generation and target detection algorithms that were developed can be confidently used to replace the previous TigerSim algorithm, which used perfect knowledge of the target and interceptor's location to determine the LOS. The results also demonstrated that the guidance system implemented in the missile simulation can generate successful target interceptions even in the presence of hardware imperfections. While this new capability brings the Auburn HWIL lab closer to accurately simulating a missile intercept, significant work is still required before high-fidelity simulation is possible.

Using the work presented in this thesis as a foundation for future students, the author would like to make several suggestions for possible research topics. One area of important work could be to focus on the modeling of the target's appearance. Currently, the target is given an artificially constant intensity throughout the flight. In a real-life interception, the targets intensity would vary with range and atmospheric conditions. Also, even though the rendered image of the target may be one pixel in size, it is realistic that the target's intensity could be split over several pixels. Therefore, a visual scattering model is proposed for future development.

The author also suggests two improvements to the target detection program. Currently, the SWIL target detection program scans the image to locate the highest intensity pixel value. All the pixels containing that intensity are classified as the target. The shortcoming of this method is that the interceptor is steered toward to nose of the target, instead of the target's center. An improved target detection program would scan the entire image and determine the range of values that represent the target. By locating the entire target in the image, the interceptor would be able to steer toward the target's geometric center. This new target detection method should be used for both the SWIL and SHWIL simulations, allowing for a more accurate comparison of target detection methods.

A second suggestion would be the history of the LOS and seeker steering angles be stored by TigerSim. If the target detection algorithm is unable to locate the target in the image, then based on the targets last known position and heading, the targets current position is estimated. This estimate is then used to calculate the LOS and seeker steering for the current time step. The following time step, the target detection algorithm would again scan the image to locate the target. This would allow for the inclusion of exogenous sources of light into the simulation.

BIBLIOGRAPHY

[1] United States Army Redstone Technical Test Center. `http://www.rttc.army.mil/whatwedo/primary_ser/modeling/hwil.htm`.

[2] Ronald Driggers, Paul Cox, and Timothy Edwards. *Introduction to Infrared and Electro-Optical Systems*. Artech House, Boston, 1999.

[3] United States Strategic Command Intercontinental Ballistic Missiles Fact Sheet. `http://www.stratcom.mil/FactSheetshtml/ballistic_missiles.htm`.

[4] Yi Ma, Stefano Soatto, Jana Kosecha, and S. Shanker Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer, 2005.

[5] J. Martin. *Atmospheric Reentry: an Introduction to its Science and Engineering*. Prentice-Hall, Englewood Cliffs, N.J., 1966.

[6] Duane Brown. *Close-Range Camera Calibration*. Symposium on Close-Range Photogrammetry, January 1971.

[7] B. Prescott and G.F. McLean. Line-based correction of radial lens distortion. *Graphical Models and Image Processing*, 59(1):39–47, January 1997.

[8] Moumen Taha El-Melegy and Aly A. Farag. Statistically robust approach to lens distortion calibration with model selection. *IEEE 1063-6919/03*, 2003.

[9] Vitaliy Leonidovich Orekhov. A full scale camera calibration technique with automatic model selection-extension and validation. Master's thesis, The University of Tennessee, 2007.

[10] Christopher Paul Broaddus. Universal geometric camera calibration with statistical model selection. Master's thesis, The University of Tennessee, 2005.

APPENDIX A

TRANSFORMATION MATRICES

The transformation matrix from inertial to missile coordinates is given by the following equation.

c11 = cos(θ) cos(ψ)

c12 = cos(θ) sin(ψ)

c13 = -sin(θ)

c21 = -cos(ϕ) sin(ψ) + sin(ϕ) sin(θ) cos(ψ)

c22 = cos(ϕ) cos(ψ) + sin(ϕ) sin(θ) sin(ψ)

c23 = sin(ϕ) cos(θ)

c31 = sin(ϕ) sin(ψ) + cos(ϕ) sin(θ) cos(ψ)

c32 = -sin(ϕ) cos(ψ) + cos(ϕ) sin(θ) sin(ψ)

c33 = cos(ϕ) cos(θ)

$$C = \begin{bmatrix} c11 & c12 & c13 \\ c21 & c22 & c23 \\ c31 & c32 & c33 \end{bmatrix}$$

The transformation matrix from missile to seeker coordinates is given by the following equation.

c11 = cos(El) cos(Az)

c12 = cos(El) sin(Az)

c13 = -sin(El)

c21 = sin(Az)

c22 = cos(Az)

c23 = 0

c31 = sin(El) cos(Az)

c32 = sin(El) sin(Az)

c33 = cos(El)

$$C = \begin{bmatrix} c11 & c12 & c13 \\ c21 & c22 & c23 \\ c31 & c32 & c33 \end{bmatrix}$$

APPENDIX B

MATLAB CODE FOR SCENE GENERATION

```
function [drawany]=SceneGen(X, targetpos, stars, range, time, plotfig)

global aumov dt stepsperframe Cinertial2seeker Cmissile2seeker visstarttime
Az El dt

if and (time>visstarttime, time<visstarttime+dt)
    %Relative Position of target to interceptor
    SeekerPosition=X(1:3,1);
    TargetPosition=targetpos;
    roll_missile=X(4,1);   pitch_missile=X(5,1);   yaw_missile=X(6,1);

    %Necessary State Information
    RelPos=TargetPosition-SeekerPosition;
    rho=norm(RelPos);  % unit vector from seeker to target
    Cinertial2missile=DCM(yaw_missile,pitch_missile,roll_missile);
% orientation of interceptor
    Cinertial2target=DCM(pi,-pi/6,0);  % orientation of target
    LOS_m=Cinertial2missile*RelPos/rho;  % LOS vector in missile coords

    %Compute Seeker Orientation
    El=asin(LOS_m(3));
    Az=atan2(LOS_m(2),LOS_m(1));
    Cmissile2seeker=DCM(Az,-El,0);

    Ctarget2seeker=Cmissile2seeker*Cinertial2missile*Cinertial2target';
    Cinertial2seeker=Cmissile2seeker*Cinertial2missile;
end

if time>visstarttime+dt
    %Relative Position of target to interceptor
    SeekerPosition=X(1:3,1);
    TargetPosition=targetpos;
    roll_missile=X(4,1);   pitch_missile=X(5,1);   yaw_missile=X(6,1);

    %Necessary State Information
    RelPos=TargetPosition-SeekerPosition;
    rho=norm(RelPos);  % distance from seeker to target
    Cinertial2missile=DCM(yaw_missile,pitch_missile,roll_missile);
% orientation of interceptor
```

```
Cinertial2target=DCM(pi,-pi/6,0);  % orientation of target
Cmissile2seeker=DCM(Az,-El,0);

Ctarget2seeker=Cmissile2seeker*Cinertial2missile*Cinertial2target';
Cinertial2seeker=Cmissile2seeker*Cinertial2missile;

% for display purposes
LOS_m=Cinertial2missile*RelPos/rho;  % LOS vector in missile coords
Az_true=atan2(LOS_m(2),LOS_m(1));
El_true=asin(LOS_m(3));
Cmissile2seeker_true=DCM(Az_true,-El_true,0);
Cinertial2seeker_true=Cmissile2seeker_true*Cinertial2missile;

% Target Model
TargetSceneCoords = Cinertial2seeker*RelPos - [rho ; 0 ; 0];
theta=[0:2*pi/20:2*pi]';
radius=.75;  % target characteristics
height=2.25;

% vertex points in target-fixed coords
basez=radius*cos(theta);
basey=radius*sin(theta);
basex=-height/2*ones(length(theta),1);
tip=[height/2 0 0]';

% convert to seeker coords
for k=1:length(basex)
    R=Ctarget2seeker*[basex(k) basey(k) basez(k)]';
    basex_s(k,1)=TargetSceneCoords(1,1)+R(1);
    basey_s(k,1)=TargetSceneCoords(2,1)+R(2);
    basez_s(k,1)=TargetSceneCoords(3,1)+R(3);
end
tip_s=TargetSceneCoords+Ctarget2seeker*tip;

%Star model
for k=1:length(stars)
    stars_s(k,:)=(Cinertial2seeker*stars(k,:)')';
    stars_s(k,1)=stars_s(k,1)-rho;
end

plot=figure(plotfig);
clf('reset')
plotaxes=axes('Position',[0 0 1 1]);  % set axes to fill entire figure
set(plotfig,'Color',[0 0 0])
camproj('perspective')
camva(20) % set the camera field of view

campos([-rho 0 0]); % position of camera
camtarget([0 0 0]); % point the camera at the target
camup([0 0 -1])  % Z points down!
axis equal
```

```
    axis off
    hold on

    %Add stars
    for j=1:length(stars)
        if([1 0 0]*stars_s(j,:)'>0)
            star=plot3(stars_s(j,1),stars_s(j,2),stars_s(j,3),'p');
            set(star,'MarkerEdgeColor',[.1 .1 .1]);
            set(star,'MarkerFaceColor',[.1 .1 .1]);
        end
    end

    %Load the custom colormap and apply to current figure
    load('MyColormaps','targetcolor')
    colormap(targetcolor)

    % plot the target
    fill3(basex_s,basey_s,basez_s,[.6 .6 .6]);  % plot the base
    for k=1:length(basex)-1  % plot the cone
        polygon=fill3([basex_s(k) basex_s(k+1) tip_s(1)],[basey_s(k)
 basey_s(k+1) tip_s(2)],[basez_s(k) basez_s(k+1) tip_s(3)],[0;0;1]);
        set(polygon,'EdgeColor','interp')
        set(polygon,'FaceColor','interp')
    end

    %Plot a single white pixel to identify the target
    if range>600
        targetpoint=plot3((TargetSceneCoords(1)+R(1)),
(TargetSceneCoords(2)+R(2)),(TargetSceneCoords(3)+R(3)),'w.');
        set(targetpoint,'MarkerSize',2);
    end
    drawany=1;

else
    drawany = 0;
end
```

Appendix C

Matlab code for SWIL Target Detection

```
function [LOS, LOStrue, xloc, yloc]=Target_gen_and_detect(X, targetpos,
stars, range, time, plotfig, prevLOS, drawany);

global aumov dt stepsperframe baseline Az El numdetect pixelx pixely
Cinertial2seeker Cmissile2seeker visstarttime

if drawany==1
    % Capture the Image
    TargetImage=getframe(plotfig);
    myimage=TargetImage.cdata;

    %Convert the image to grayscale
    Pic=rgb2gray(myimage);

    %Calculate the location of the target
    num=0;
    targetlocationx=0;
    targetlocationy=0;

    %Scan the image for the target
    baseline=50;
    if time<visstarttime+.003
        scnstart=1;
        scnend=pixelx;
    else
        scnstart=(pixelx/2)-10;
        scnend=(pixelx/2)+10;
    end
    for i=scnstart:scnend
        for j=scnstart:scnend
            pixel=Pic(i,j);
            if pixel>baseline
                targetlocationx=i;
                targetlocationy=j;
                baseline=pixel;
                num=0;
            end
            if pixel==baseline
                num=num+1;
                targetlocationx(num,1)=i;
                targetlocationy(num,1)=j;
            end
        end
    end

    %Verify that the target was located
    if baseline<110;
        for i=1:pixelx
            for j=1:pixelx
                pixel=Pic(i,j);
                if pixel>baseline
                    targetlocationx=i;
                    targetlocationy=j;
                    baseline=pixel;
                    num=0;
                end
                if pixel==baseline
                    num=num+1;
                    targetlocationx(num,1)=i;
                    targetlocationy(num,1)=j;
                end
            end
        end
    end

    %Calculate the center of the target (Pixel Coordinates)
    x=(mean(targetlocationx));
    y=(mean(targetlocationy));

    xloc=x;
    yloc=y;

    %Compute the line of sight vector
    FOVx=30;
    FOVy=30;

    Uo=pixelx/2;
    Vo=pixely/2;

    sigmax=(Uo/(tand(FOVx/2)));
    sigmay=(Vo/(tand(FOVy/2)));

    A=[sigmax 0 Uo;0 sigmay Vo;0 0 1];
    LOSvec=inv(A)*[x;y;1];
    LOSnorm=norm(LOSvec);

    LOS=Cinertial2seeker'*[0 0 1;0 1 0;1 0 0]*(LOSvec/LOSnorm);

    x = X(1);
    y = X(2);
```

```
    z = X(3);
    LOStrue=(targetpos - [x y z]')/norm(targetpos - [x y z]');

    LOSmiss=Cmissile2seeker'*[0 0 1;0 1 0;1 0 0]*(LOSvec/LOSnorm);

    Az=atan(LOSmiss(2)/LOSmiss(1));
    El=asin(LOSmiss(3));

else
    x = X(1);
    y = X(2);
    z = X(3);

    % line of sight in inertial coords
    LOS = (targetpos - [x y z]')/norm(targetpos - [x y z]');
    drawany=1;
    LOStrue=[0;0;0];

    xloc=0;
    yloc=0;
end
```

Appendix D

Matlab code for SHWIL Target Detection

```
function [LOS, LOStrue, xloc, yloc]=Target_gen_and_detect(X, targetpos,
stars, range, time, plotfig, prevLOS, drawany, xtnfm, ytnfm);

global aumov dt stepsperframe baseline Az El numdetect pixelx pixely
Cinertial2seeker Cmissile2seeker visstarttime obj1 leftcameraedge
topcameraedge xsize ysize

if drawany==1
    %Ensre the image has been drawn
    pause(0.1)
    if time<visstarttime+2*dt
        pause(0.5)
    end
    if range<200
        pause(0.5)
    end

    %Communicating with instrument object, obj1.
    fwrite(obj1, '0')
    count=1;
    while count==1
        [data1,count] = fscanf(obj1,'%c',6);
    end

    x=((str2double(data1(1:3))-xsize-leftcameraedge)*xtnfm)+pixelx/2;
    y=((str2double(data1(4:6))-ysize-topcameraedge)*ytnfm)+pixely/2;

    xloc=x;
    yloc=y;

    FOVx=20.0;
    FOVy=20.0;

    Uo=pixelx/2;
    Vo=pixely/2;

    sigmax=(Uo/(tand(FOVx/2)));
    sigmay=(Vo/(tand(FOVy/2)));
    A=[sigmax 0 Uo;0 sigmay Vo;0 0 1];
    LOSvec=inv(A)*[x;y;1];
    LOSnorm=norm(LOSvec);

    LOS=Cinertial2seeker'*[0 0 1;1 0 0;0 1 0]*(LOSvec/LOSnorm);

    x = X(1);
    y = X(2);
    z = X(3);
    LOStrue=(targetpos - [x y z]')/norm(targetpos - [x y z]');

    LOSmiss=Cmissile2seeker'*[0 0 1;1 0 0;0 1 0]*(LOSvec/LOSnorm);

    Az=atan(LOSmiss(2)/LOSmiss(1));
    El=asin(LOSmiss(3));

else
    x = X(1);
    y = X(2);
    z = X(3);

    LOS = (targetpos - [x y z]')/norm(targetpos - [x y z]');

    LOStrue=[0;0;0];
    xloc=0;
    yloc=0;
end
```

APPENDIX E

MATLAB CODE FOR SHWIL CALIBRATION

```
function [xtnfm ytnfm]=SHWIL_Calibration()

global leftcameraedge topcameraedge xsize ysize obj1

pixelx=900;
pixely=900;

scrpxl=zeros(4,2);
campxl=zeros(4,2);
pxltnfmx=zeros(4,1);
pxltnfmy=zeros(4,1);

for ss=1:4

    if ss==1
        x=0.05;
        y=0.05;
    elseif ss==2
        x=0.95;
        y=0.05;
    elseif ss==3
        x=0.05;
        y=0.95;
    elseif ss==4
        x=0.95;
        y=0.95;
    else
        disp('Error in Calibration')
    end


    mntrpos=get(0,'MonitorPosition');
    left=(mntrpos(2,3)+mntrpos(2,1)−1)/2−pixelx/2;
    bottom=(mntrpos(2,4)+mntrpos(2,2)−1)/2−pixely/2+mntrpos(1,4)−
mntrpos(2,4);
    calibr=figure('Position',[left bottom pixelx pixely]);
    set(calibr,'Toolbar','none');

    %Plot the calibration markers
    plotaxes=axes('Position',[0 0 1 1]);
    plot(x,y,'ws', 'markerfacecolor','w')
    axis ([0 1 0 1])
    set(gcf,'Color','k')
    axis off
    pause(0.2)


    %Communicate with the camera
    %Communicating with instrument object, obj1.
    fwrite(obj1, '0')
    count=1;
    while count==1
        [data1,count] = fscanf(obj1,'%c',6);
    end

    %Store pixel locations for calibration
    scrpxl(ss,1)=(x*pixelx);
    scrpxl(ss,2)=(pixely-(y*pixely));
    if ss==1
        campxl(ss,1)=(str2double(data1(1:3))-leftcameraedge);
        campxl(ss,2)=(str2double(data1(4:6))-topcameraedge);
    elseif ss==2
        campxl(ss,1)=(str2double(data1(1:3))-leftcameraedge);
        campxl(ss,2)=(str2double(data1(4:6))-topcameraedge);
    elseif ss==3
        campxl(ss,1)=(str2double(data1(1:3))-leftcameraedge);
        campxl(ss,2)=(str2double(data1(4:6))-topcameraedge);
    elseif ss==4
        campxl(ss,1)=(str2double(data1(1:3))-leftcameraedge);
        campxl(ss,2)=(str2double(data1(4:6))-topcameraedge);
    end

    pxltnfmx(ss,1)=(scrpxl(ss,1)-pixelx/2)/(campxl(ss,1)-xsize);
    pxltnfmy(ss,1)=(scrpxl(ss,2)-pixely/2)/(campxl(ss,2)-ysize);

    %Clean up
    pause(0.1)
    close all
end

%Compute coordinate transformation
xtnfm=mean(pxltnfmx(:,1));
ytnfm=mean(pxltnfmy(:,1));

pause(0.5)
```

Appendix F

Matlab code for Lens Distortion Determination

```
%Program to graphically display the amount of distortion present on a
%camera lens

%Image size: X=640 Y=480
xp=330;
yp=240;
K1=6*10^-7;
K2=-2*10^-12;
pixelx=900;
pixely=900;
leftcameraedge=66;
topcameraedge=95;
xsize=188.5; %Camera inspection window /2
ysize=188.5;

obj1 = instrfind('Type', 'tcpip', 'RemoteHost', '131.204.22.56',
'RemotePort', 502, 'Tag', '');

%Create the tcpip object if it does not exist otherwise use the object
that was found.
if isempty(obj1)
    obj1 = tcpip('131.204.22.56', 502);
else
    fclose(obj1)
    obj1 = obj1(1);
end

mntrpos=get(0,'MonitorPosition');
left=(mntrpos(2,3)+mntrpos(2,1)-1)/2-pixelx/2;
bottom=(mntrpos(2,4)+mntrpos(2,2)-1)/2-pixely/2+mntrpos(1,4)-mntrpos(2,4);
calibr=figure('Position',[left bottom pixelx pixely]);
set(calibr,'Toolbar','none');

%Connect to instrument object, obj1.
fopen(obj1)
i=1;
for x=.05:.05:.95
    for y=.05:.05:.95;
        clf('reset')
        plotaxes=axes('Position',[0 0 1 1]);
        plot(x,y,'ws', 'markerfacecolor','w')
        axis ([0 1 0 1])
        set(gcf,'Color','k')
        axis off
        pause(0.2)

        fwrite(obj1, '0')
        count=1;
        while count==1
            [data1,count] = fscanf(obj1,'%c',6);
        end

        scrpxl(i,1)=(x*pixelx);
        scrpxl(i,2)=(pixely-(y*pixely));

        campxl(i,1)=str2double(data1(1:3));
        campxl(i,2)=str2double(data1(4:6));

        i=i+1;
    end
end
close all

for i=1:length(campxl)
    x(i)=((campxl(i,1)-127-188.5)*2.3859)+450;
    y(i)=((campxl(i,2)-95-188.5)*2.3789)+450;
    xpxlerror(i)=scrpxl(i,1)-x(i);
    ypxlerror(i)=scrpxl(i,2)-y(i);
end

figure(1)
for i=1:length(campxl)
    hold on
    plot(x,y,'rs')
    plot(scrpxl(i,1),(scrpxl(i,2)),'b+','MarkerFaceColor','r')
    xlim([0 900])
    ylim([0 900])
end

spacing=-4:.5:4;

figure(2)
xhist=histc(xpxlerror,spacing);
bar(spacing,xhist);

figure(3)
yhist=histc(ypxlerror,spacing);
bar(spacing,yhist);
```