3-1-2021

Using Motion Capture and Augmented Reality to Test AAR with Boom Occlusion

Vincent J. Bownes

Using Motion Capture and Augmented Reality to Test AAR with Boom Occlusion

THESIS

Vincent J. Bownes, 2d Lt, USAF

AFIT-ENG-MS-21-M-016

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-21-M-016

Using Motion Capture and Augmented Reality to Test AAR with Boom Occlusion

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Vincent J. Bownes, B.S.C.S.

2d Lt, USAF

March 25, 2021

AFIT-ENG-MS-21-M-016

Using Motion Capture and Augmented Reality to Test AAR with Boom Occlusion

THESIS

Vincent J. Bownes, B.S.C.S.
2d Lt, USAF

Committee Membership:

Scott L. Nykl, Ph.D
Chair

Clark N. Taylor, Ph.D
Member

Douglas D. Hodson, Ph.D
Member

AFIT-ENG-MS-21-M-016

Abstract

The operational capability of drones is limited by their inability to perform aerial refueling. This can be overcome by automating the process with a computer vision solution. Previous work has demonstrated the feasibility of automated aerial refueling (AAR) in simulation. To progress this technique to the real world, this thesis conducts experiments using real images of a physical aircraft replica and a motion capture system for truth data. It also compares the error between the real and virtual experiments to validate the fidelity of the simulation. Results indicate that the current technique is effective on real images and that the simulation can predict errors in real world experiments.

Acknowledgements

I would like to thank my advisor, whose unwavering dedication to the project made working on my thesis an enjoyable process. I would like to thank my committee, without whose guidance I would not have accomplished nearly as much. Finally, I would like to thank my peers, who worked on the project with me. Their work went hand in hand with mine, and I couldn't have done this without them.

Vincent J. Bownes

Table of Contents

Page

List of Figures

Figure Page

Figure Page

Using Motion Capture and Augmented Reality to Test AAR with Boom Occlusion

I. Introduction

1.1 Problem Background

Drones are currently unable to perform aerial refueling due to the latency between the pilot and the aircraft's response to the control input. The inability to refuel in flight reduces the drone's range and operational capability. Automated aerial refueling (AAR) using computer vision is close to becoming a reality, but further testing is required. Real flight testing is expensive and not practical to be performed with the frequency required to validate current AAR techniques.

Previous works have attempted to quantify the error between the 3D virtual world and the real world. One researcher used motion capture truth data to compare real and virtual AAR pose estimation, but boom occlusion was not taken into account [1]. Another also attempted this but was limited by the size of the motion capture area in which he had to work [2]. Others have demonstrated methods for mitigating the effects of boom occlusion but only in the virtual world [3]. This research will leverage augmented reality and the truth data from motion capture to superimpose a virtual boom onto real images to test boom occlusion mitigation techniques, as well as the accuracy of the AAR pose estimation on real images.

1.2 Research Objectives

The questions this research aims to answer are:

- Is the 3D virtual world simulation an accurate model of the AAR process in the

real world?

- Can the current AAR techniques accurately estimate the pose of a receiver in real time on real images?
- What are the effects of boom occlusion on real images as opposed to virtual?

Additionally, this research will construct a framework for future testing that can be used to test boom occlusion mitigation techniques and approaches with real images.

1.3 Document Overview

Chapter II presents the prerequisite background knowledge, as well as recent work on the AAR project. Chapter III describes how the experiments are constructed and provides detailed explanation of how data is collected. Chapter IV shows the comparison between the experiments done in the real and virtual environments. It demonstrates that AAR has acceptable error even when using real images. To conclude, chapter V summarizes what this thesis has accomplished and how it benefits future research.

II. Background and Literature Review

This chapter provides the background for the methodology and results that follow. Beginning with computer vision, the theory behind the research, this paper will explore the technical details of how features are matched between pairs of images and then re-projected into 3D space. Augmented reality is used to enhance imagery for the purpose of testing and evaluation; a section dedicated to each of those topics follows. Finally, there are three sections that cover the previous work done in this area. The first is a general review of the related works, followed by two sections that discuss the most relevant papers to this thesis.

2.1 Geometric Computer Vision

This section gives a high level overview of the theory that makes geometric computer vision possible. It first discusses the pinhole camera model, which is the model that applies to most modern cameras. That is followed by camera calibration techniques and, finally, epipolar geometry, which makes 3D pose estimation through stereo vision possible.

The pinhole camera model is a simple, ideal model which describes the mathematical relationships from points in 3-dimensional space to their corresponding 2-dimensional points on an image plane [4]. A pinhole plane with a small hole in its center is placed in front of an image plane. The rays of light from objects in the world pass through the central hole in the pinhole plane and are projected upside down onto the image plane. The distance between these two planes is known as the focal length and will determine the size of the projected object on the image plane. This model is representative of a real pinhole camera and can be constructed; however, to simplify the math, an equivalent form of this model is more commonly used, one which cannot

be physically constructed.

In the equivalent model, shown in figure 1, the image plane is in front, and the pinhole plane becomes known as the center of projection. All rays of light pass through the image plane on their way to the center of projection behind it. For example, the light from the point in 3D space $Q = (X, Y, Z)$ passes through the image plane at $q = (x, y, f)$. Therefore, this ray of light is projected onto the image plane at the x, y pixel coordinate, and f represents the focal length.

Along with the focal length, the principal point is the other component that makes up the camera's intrinsic matrix. The principal point is the intersection between the optical axis and the image plane and, counterintuitively, will not always pass through the exact center of the image plane. These parameters for the intrinsic matrix can be found by performing camera calibration.

The method for camera calibration that is still most prominently utilized is described in [5]. Prior to this study, accurate camera calibration was possible, but it required prohibitively expensive equipment and setups. [5] created an accurate and accessible method for camera calibration using a planar pattern. The first step in this

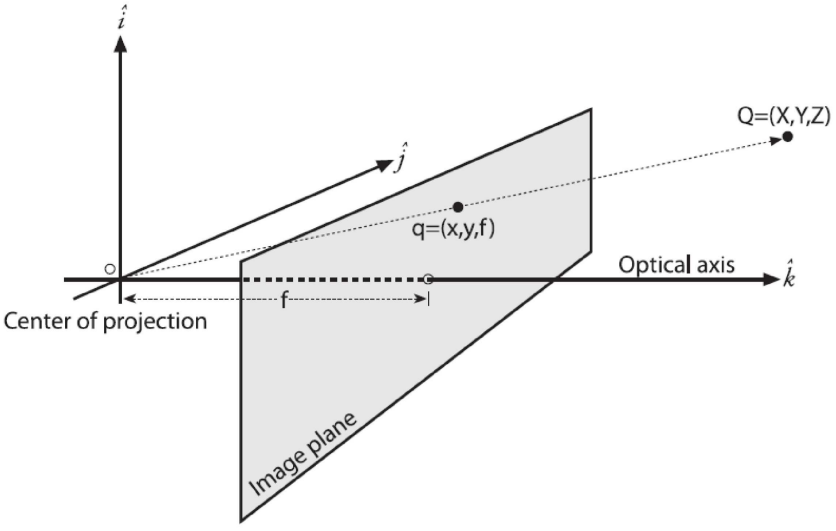

Figure 1: The Pinhole Camera Model

method is to take images of the calibration plane from various orientations. In the majority of cases the calibration plane is a checkerboard. This is followed by finding the feature points in the image (Figure 2). The corners of the checkerboard are the features used for calibration. All of the intrinsic and extrinsic parameters are initially estimated using a closed form solution. Finally, all of these values are fine-tuned by minimizing the equation

$$\sum_{i=1}^{n}\sum_{j=1}^{m} ||m_{ij} - \breve{m}(A, k_1, k_2, R_i, t_i, M_j)||^2 \tag{1}$$

where A is the intrinsic matrix, consisting of the focal lengths and principal point, the k values represent the radial distortion, R and t make up the extrinsic matrix for a particular image, which is the rotation and translation from the world coordinate

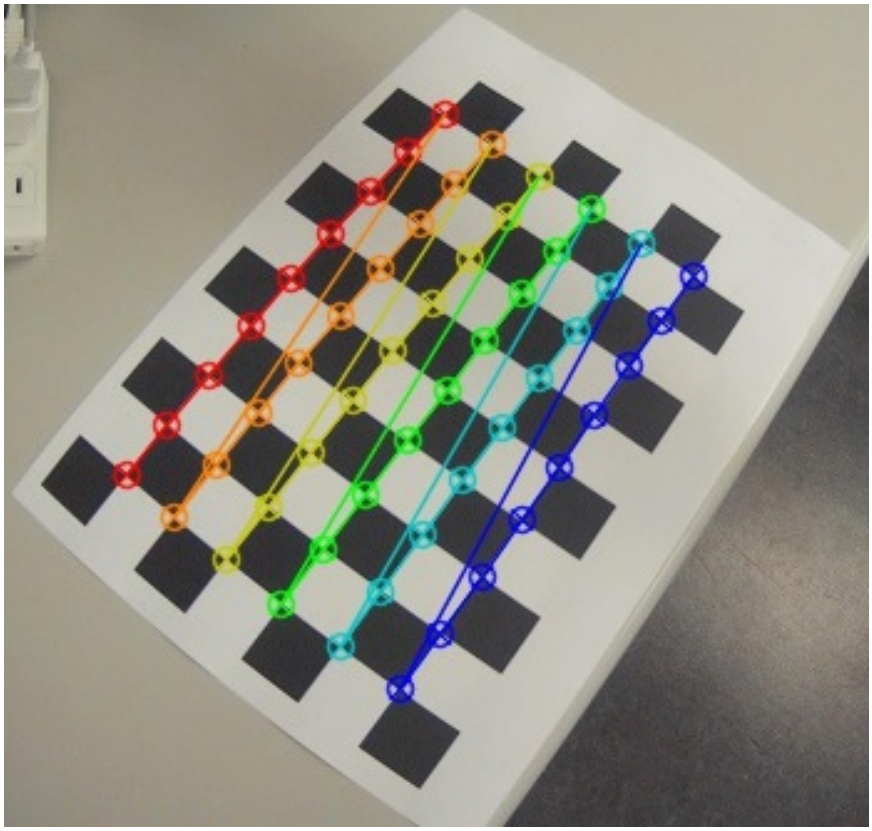

Figure 2: Corner Features Found with OpenCV

system to the camera coordinate system, M_j is a corner on the checkerboard and $\breve{m}(A, k_1, k_2, R_i, t_i, M_j)$ is the projection of that corner into image coordinates. This value is subtracted from the observed location of the corner m_{ij} for every corner in every image where i represents images and j selects corners on the calibration plane. By minimizing the difference between the observed corner coordinates, m_{ij}, and the estimated corner coordinates, $\breve{m}$, the accuracy of the parameters $(A, k_1, k_2, R_i, t_i, M_j)$ is iteratively improved. Camera calibration functions are provided in OpenCV as described in [4]. The most commonly used technique relies on a checkerboard as the planar surface and the corners as the feature points. The number of corners on the checkerboard and the length of the squares are input to the function to help with the initial estimation of the parameters. Figure 3 shows an example of a checkerboard pattern that has been successfully detected by OpenCV.

Epipolar geometry provides the means to localize points in 3D space using a stereo camera system [4, 6]. When a point is projected onto an image plane, the distance

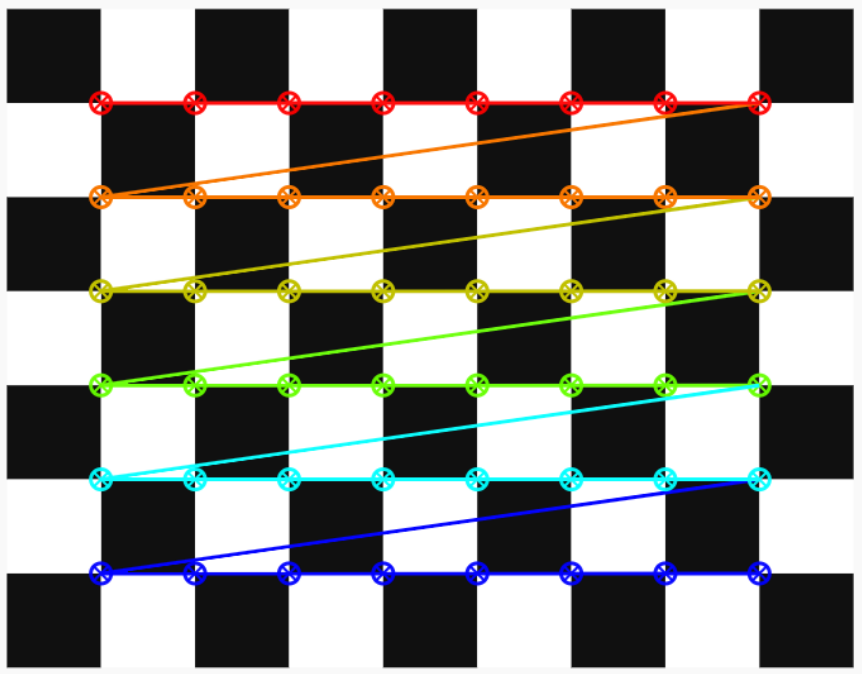

Figure 3: Checkerboard Corner Detection Using OpenCV

information is lost; however, it can be determined if two well calibrated cameras view that same point. In figure 5 two pinhole camera models are combined. Each camera observes the point X in 3D space. O_L and O_R represent the optical centers of the left and right cameras, respectively. As this section discussed earlier, the intersection between the line $\overline{XO_R}$ or $\overline{XO_L}$ and the image plane gives the pixel location of X. In the left camera's image plane, this is X_L; for the right camera, it is X_R. Once the 3D point X has been found in one image, it would require a 2-dimensional search across the other image to attempt to find the matching point. This would be time consuming and error prone. By introducing the epipolar constraint, this is reduced to a 1-dimensional search. The projection of the optical center O_R onto the image plane of the left camera gives us the left camera's epipolar point e_L and vice versa. The line between the epipolar point and the point X on the image plane is the epipolar line. By searching only along this line for a matching point, the epipolar constraint is satisfied and the search process has been improved.

To accurately perform stereo block matching while satisfying the epipolar constraint row, aligned image planes are necessary. This requires the cameras' optical axes to be parallel. Unfortunately, a real stereo camera system is not likely to have this property. Stereo rectification is used to reproject the image planes such that their rows become aligned and their optical axes intersect at infinity. OpenCV provides two algorithms for accomplishing this, one which requires a calibration and one that does not. Figure 4 shows an example of a pair of images before and after stereo rectification.

Figure 4: Image Rectification

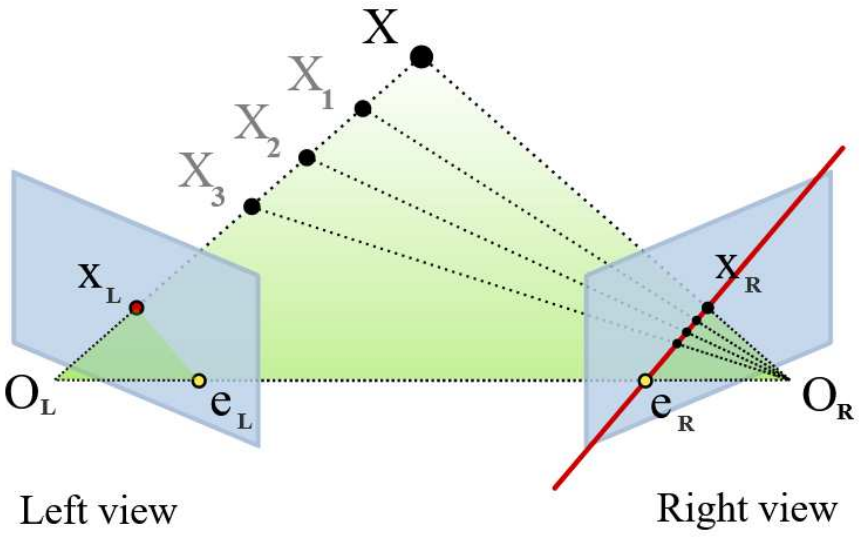

Figure 5: Epipolar Geometry

2.2 Augmented Reality

Augmented reality (AR) is the mixture of our real world with a 3D virtual world (3DVW) through a combination of hardware and software. The applications of AR

range from training and education to medical care and construction. Several popular applications recently have even included video games such as Pokemon Go. The game utilizes a smart phone and its camera as the hardware and the installed app as the software. This section will discuss the different types of AR technologies and the current challenges in this field of study.

Over the last 20 years, a handful of AR research areas have remained in the forefront due to their continued relevance and challenge. These areas include tracking, interaction, and calibration. Research into using AR for evaluation/testing is a more recent development [7]. Tracking is the means by which the user's point of view is registered so that the 3D models can be drawn in the correct location and orientation on the screen. For example, the direction the user is holding his/her smartphone or the way a user wearing a head mounted display (HMD) has turned his/her head is measured by tracking technologies. Interaction is how and what the user is able to manipulate in the augmented scene being rendered. Calibration is how the hardware, such as the camera or inertial measurement unit (IMU), is adjusted so that it will accurately align the real world with the virtual world [8].

There are two types of tracking: sensor based and vision based. Sensor based tracking consists of magnetic, inertial, optical, and acoustic tracking techniques. Magnetic tracking utilizes a magnetic transmitter and receiver, the latter of which is attached to the viewer. The transmitter is considered the origin of the virtual 3D coordinate system, and the receiver's pose can be estimated based on the magnetic signals it receives. Inertial tracking is based on measurements from an IMU, such as an accelerometer and a gyroscope. Modern virtual reality (VR) headsets make use of this type to track the viewer's head movements. Optical tracking uses two or more cameras and 3D-localization technology to find the position of potentially multiple objects in a predefined measurement space. The requirement for multiple

cameras and a predefined measurement space is what differentiates sensor based optical tracking from vision based tracking. Acoustic tracking relies on how long it takes an acoustic signal to reach its intended receivers. This is the technology that motion capture systems employ when finding the position of the motion capture markers.

Vision based tracking uses one of the following: an infrared sensor, visible light sensor, or a 3D-structure sensor to determine the position of the camera. This type of tracking has two sub-categories: markerless and marker-based tracking. The latter uses specialized fiducial markers, such as aruco markers, to identify where to draw the computer generated models and at what orientation [9].

The most common types of hardware used for augmented reality are computers with webcams, head mounted displays, glasses, and smartphones, the latter being the most widely used platform for AR today. The HMD is the most immersive piece of hardware since the user's entire field of view is taken up by the built-in screens. It is able to track head movements in 6DoF, so whichever way the user turns, the IMU will be able to relay that to the software and to alter the augmented reality accordingly. AR-glasses are a less common hardware platform but will likely become more prevalent in the near future. Unlike the HMD, the AR-glasses do not take up the user's entire field of view and are, therefore, less immersive. They resemble a pair of reading glasses and display information as an overlay to the real world.

A number of different software development kits (SDKs) are commonly used to develop AR applications. The most popular are Metaio, Vuforia, and ARToolkit. This paper will not go into more detail regarding these since they are all designed to be used with game engines such as Unity or Unreal, but all development on this project is done within the AftrBurner engine [10].

AR has been used in a wide variety of training and education applications. In [11] augmented reality is used to help computer science students learn about the OpenGL

computer graphics library. Since some of the commands may not be intuitive, especially to novice programmers, the ability to visualize what they are doing in the real world enhances the learning process. Using ARMarkers, QR codes, and a webcam, students can see a 3D model displayed on their desks at the location and orientation of the ARMarker. Each different QR code represents a different OpenGL operation; by showing the QR code to the webcam, the students can witness the effect of that operation in real time. Soldiers and surgeons have made use of AR for training purposes, as well. It has been used to teach soldiers how to repair their weapons and even how to navigate using virtual maps. Surgeon trainees can have virtual 3D organs superimposed into their view for practicing an operation [7].

2.3 Augmented Reality for Testing/Evaluation

Researchers are often constrained by safety or budgetary obstacles when seeking to collect data. This section will discuss examples of augmented reality being used to overcome such obstacles. The first example is a direct precursor to this thesis using the AftrBurner engine [10] to augment flight imagery. The subsequent two are self-driving car studies that use augmented reality to overcome the current infeasibility of testing self-driving cars on busy roads.

Test flights were conducted without a boom present. To examine the effects a boom would have on the collected images, [12] developed an augmented reality technique for displaying a perspective correct virtual boom into the image. A rectangular quad, texture mapped with real images from a camera, is placed at the end of the virtual EO camera's viewing frustum. A boom is attached to the virtual tanker aircraft in its correct location relative to the cameras. The resulting image from these virtual EO cameras is the real image with a boom overlaid in the perspective correct position and orientation as seen in figure 6. This perspective correct geometry is the

key enabler for AR applications.

Augmented reality can be used for testing various sensors, not just EO cameras. In[13], the authors are testing self-driving cars that are equipped with LiDAR, radar, EO cameras, and a high precision GPS. The test vehicle is driven on roads in a closed testing facility that is representative of a realistic urban environment, but there is neither enough traffic nor pedestrians to test all scenarios. To overcome this, a simulation environment is created which mimics the test facility. The test vehicle transmits data to roadside processing units; these control an identical car with identical sensors in the simulation. By adding additional objects into the simulation, this information is fed back to the test vehicle via the roadside units, thereby creating a virtually augmented environment with which the vehicle interacts. By performing these tests in the real and virtual world simultaneously, failed scenarios can be replayed and examined more closely in the simulation to help fine-tune the algorithms involved.

[14] uses augmented reality to create training images for a deep neural network that controls self-driving cars. Training images that are generated in a 3D virtual world lack sufficient detail for these researchers' purposes, and gathering enough real imagery is too time consuming. The solution is to overlay highly realistic car models

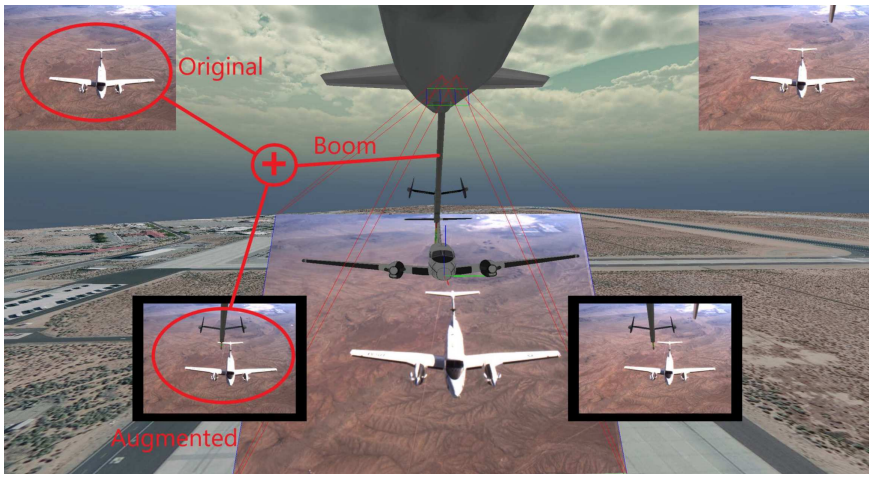

Figure 6: 3-Dimensional Boom Model Over Real Imagery

onto real images. They found that the most effective method for choosing the pose of the car model was to first use the homography between the ground plane and the image plane to transform the image into a top down view, then to manually annotate the paths a car could take and to place the vertical axis of the model along one of those paths. The results using these augmented images were better than using a fewer number of real non-augmented images. This demonstrates that computer vision based machine learning will work equally as well with augmented imagery as with real imagery, as long as the models have sufficient detail and the pose is chosen in a plausible, realistic way.

2.4 Related work on AAR

Early efforts toward automated aerial refueling (AAR) attempted to solve the problem using GPS [15, 16, 17, 18]. A common problem discovered in these studies was that the refueling tanker occludes the area above the receiver, which makes it more difficult for the receiver to be within the view of enough GPS satellites. Additionally, GPS is an asset that could be degraded or denied by an enemy, and this fact must be taken into account during research. Some of the aforementioned studies, such as [16] and [18], combined GPS with other techniques to compensate for the shortcomings of GPS alone. The authors in [16] combined GPS with other sensors, such as an inertial measurement unit, using an extended Kalman filter. Those in [18] combined GPS with machine vision.

The use of electro-optical (EO) and infrared sensors without GPS was explored in [19, 1, 20, 21] . Lidar was studied in [19]. In [22], dual EO cameras, also known as stereo vision, observed a small rotor-craft drone with markers at the refueling point. Computer vision algorithms found the markers and the boom did point toward the receptacle. Stability and accuracy, however, were not precise enough for AAR. [21]

combined the techniques of EO and infrared. Many of these techiques demontrated accurate results but only in experiments not representative of a real flight test.

Flight testing is prohibitively expensive, so it is crucial to this field of research to have high fidelity testing techniques that can be accomplished virtually. Early examples of simulations are proposed in [23, 24, 25]. The simulation in [23] was an environment designed specifically for testing AAR with mathematical models for the atmospheric turbulence and UAV docking controls. The simulation used for all testing in this thesis is the AftrBurner engine [10], which is a high fidelity 3-dimensional virtual world (3DVW). Despite the increasing prevalance of AR in testing and evaluation, it has yet to be applied to AAR. This thesis uses AR to allow the aformentioned techniques to be tested in the real world with the addition of boom occlusion.

2.5 AftrBurner vs. Real World

A goal of this work is to verify the fidelity of the AftrBurner engine simulation by comparing real and virtual experiments. The experiments are made as similar as possible to each other to enable a valid comparison of the results. This is an evolution upon the research in [2, 1], which was conducted with a similar goal.

A previous study conducted at AFIT used the truth data from a small motion capture lab, approximately 10x10 square feet, to compare the error of the stereovision pose estimation pipeline between the real and virtual worlds [2]. A virtual replica was created in the Aftrburner engine of the motion capture lab and a quadcopter drone using texture mapping that matched the real world environment. The truth data from the motion capture lab was fed into the virtual world to recreate the exact path of the drone within 2mm of accuracy. To imitate the real world as closely as possible, the virtual cameras in the 3DVW used the same calibration as the cameras

from the real world. The stereovision pose estimation pipeline was then run on both sets of images to create a point cloud, and the amount of error was compared.

Although the difference in error was small, less than 0.6% for the x,y, and z components, this study was limited in how closely it could replicate aerial refueling. It was primarily limited by the amount of space that was available to operate; this resulted in the stereo cameras having to be placed closer together than they usually would be. Normally, they would have a baseline of 0.5m between them. For this study, that distance had to be reduced to 63.6mm, which alters the disparity maps that are generated. Additionally, the viewing frustum of the cameras was not able to accurately represent a scaled refueling envelope, again due to the size of the experiment's environment (Fig. 7).

A similar study was conducted at AFIT, which was not limited by the size of the testing environment [1]. This experiment was set up in a large motion capture lab approximately 60 square feet in size. A 1:7 scale model of a receiver aircraft was

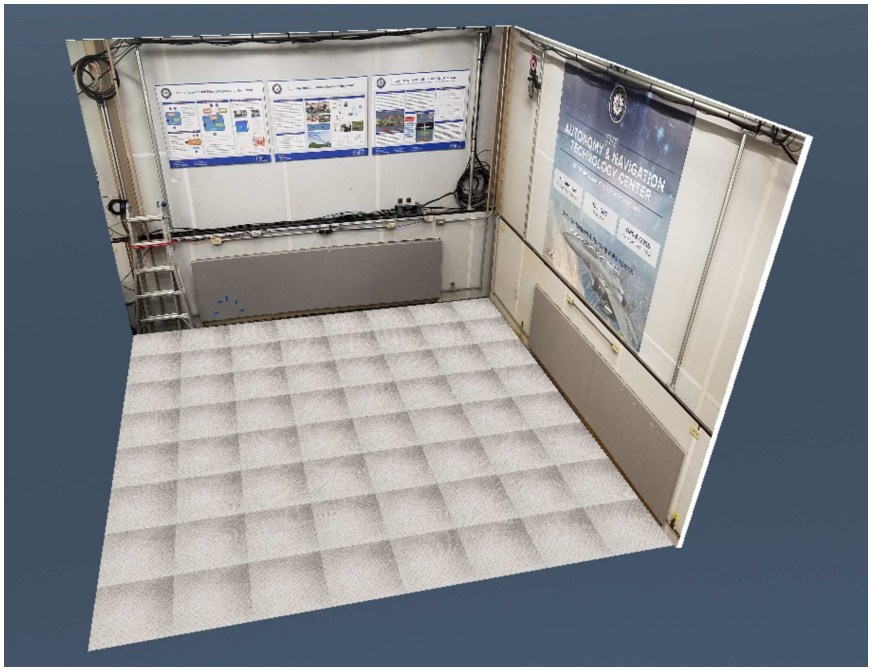

Figure 7: Virtual Model of Motion Capture Lab

placed near the center of the room with a large truss system erected overhead to support the cameras (Fig. 8). The stereo cameras could glide along a cable from one truss to the other and create the illusion that the receiver aircraft was flying up towards the cameras. This provided a more accurately scaled model of the refueling envelope, but the goals of this paper were less concerned with direct comparison between the real world and 3DVW than the work in [2]. The focus here was on using real images to see if the pipeline could still produce accurate results in real time. The motion capture lab provided the truth data against which to compare the results of the pipeline. Some comparison between error trends in simulation and the real world was done, but it was not the primary focus of the paper. To most accurately compare the error in the real world study to the 3DWV, a 1-1 model would have to have been constructed in the AftrBurner engine as it was in [2]. This paper did demonstrate that the pipeline could still produce accurate results in real time, even on real world imagery. Boom occlusion, however, was not taken into account in this paper.

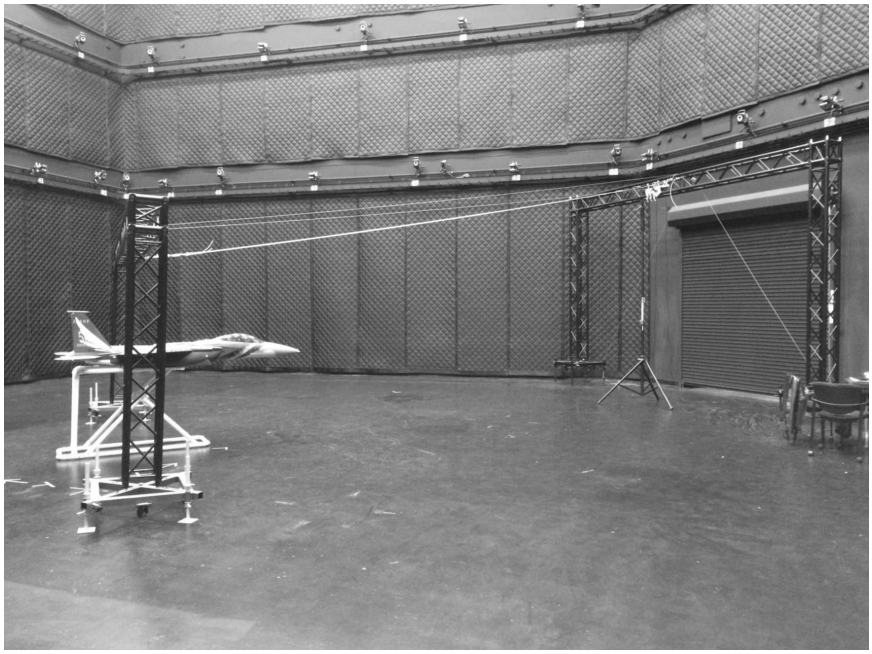

Figure 8: Large Motion Capture Lab Experiment Setup

2.6 Boom Occlusion Mitigation

The issue of boom occlusion is addressed in [3] by using ray-plane collision detection and shadow volumes. Once the receiving aircraft is within the refueling envelope, the boom begins to occlude large portions of the aircraft. The stereo block matching will produce points on the boom since the algorithm cannot differentiate between it and the receiving aircraft. ICP is performed to attempt to match the 3D point cloud to the shelled reference model. During this process, the points on the boom will pull the model towards the boom and produce incorrect pose estimation. To improve the accuracy of this process, the 3D point cloud must have outliers removed, including those on the boom. Additionally, the shelled reference model is altered to more closely match what the cameras see once the boom is occluding the image.

To address the erroneous points in the 3D point cloud, ray-plane collision detection is used. This technique casts a ray through each of the sensed points to check whether or not it collides with the boom model, in which case it is removed. Points on the edges of the boom are not filtered out since the ray cast through them does not collide with the boom. This is fixed by extruding, or enlarging, the boom's geometric model by 10 cm in all directions, after which 99% of the points on the boom are successfully filtered out. The downside to this approach is that even when using an octree for collision detection, it is still a slow process, O(n) for the approximately 8,000 points over 1,600 faces of the boom model.

Shadow volumes are leveraged to dynamically adjust the reference model to match the parts of the aircraft that the cameras can see. Initially, the shelled reference model was statically altered prior to the pose estimation process to resemble the sections of the aircraft that are visible during the most common occlusion phases. The closer the reference model can resemble what the cameras should see, the less error there will be when performing the ICP step. Shadow volumes are created by treating the cameras

as a point light source. The light volume is represented by a set of triangles, which all share the camera as a point (Fig. 9). The remaining vertices of the triangles extend to the edges of the boom silhouette. All triangles that would lie in a shadow for a semi-infinite hull of darkness define the shadow volume. Point inclusion is performed to check which points on the shelled reference model are being occluded from which camera's point of view and then are removed from the model.

The results showed significant improvement over approaches that made no attempt to mitigate boom occlusion. Between the naive and final boom mitigation techniques, the error was reduced by 80-90% in the various regions of the refueling approach. These techiques, however, have only been implemented and tested in the 3DVW. A framework is required which will allow for these techniques, as well as new ones, to be tested on real images in the real world. That is what this work aims to achieve.

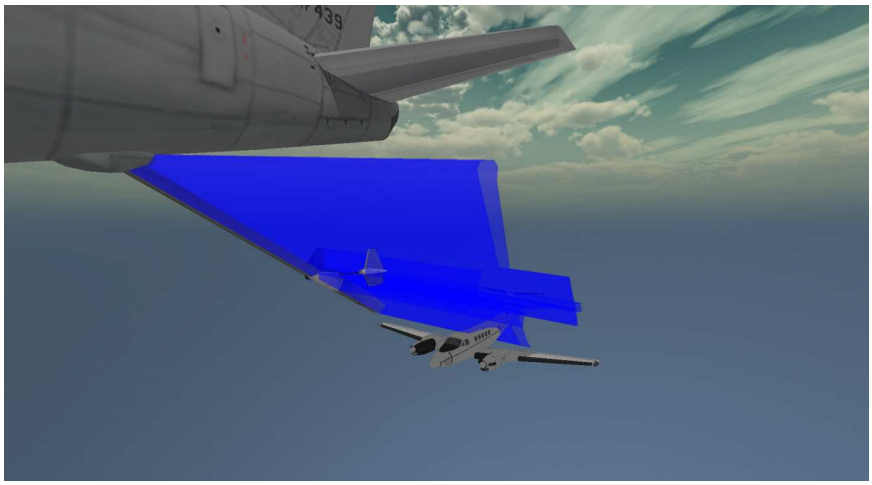

Figure 9: Shadow Volumes

III. Methodology

3.1 Preamble

The primary goals of this work are as follows:

- Create a reusable framework for rapidly generating AAR test data from real images with and without boom occlusion
- Demonstrate that SBM and ICP can achieve desired levels of accuracy with real images in a full scale environment
- Examine error trends between the AftrBurner engine simulation and the real world

To reach these goals, this paper conducts mock aerial refueling approaches in real and virtual environments at near full-scale. Real images are taken of an aircraft replica in a motion capture lab. The current state of the art AAR techniques are employed to find the pose of the aircraft relative to the camera. The motion capture system provides pose data for the tracked aircraft object, which serves as the truth. This experiment is performed in the real world, as well as an equivelant version in the virtual world for comparison. Both real and virtual environments are tested with a geometrically accurate refueling boom imposed over the images. In order for the testing framework to be useful to future researchers, it must demonstrate sufficient accuracy in non-occluded imagery. This baseline will be the control against which future experiments are compared. Also, to compare the error trends between approaches conducted in the real and virtual worlds, the virtual environment must mimic the real one as closely as possible.

This chapter details how the pose of the moving aircraft replica is accurately estimated. It begins by outlining the experiment being performed, a set of mock aerial

refueling approaches. Following that, it explains the techniques used to implement augmented reality. This is critical because it enables the addition of the full-scale 3D boom model in the imagery. An explanation is provided for how the real and virtual worlds are made as indistinguishable as possible. The source of truth data, which is the information produced by the motion capture system, is then detailed and justified. Finally, the potential sources of error are enumerated, and an explanation is provided for how each of them is ruled out or corrected.

3.2 Experiment Overview

While the overall goal of this work is to provide a testing framework for future researchers, the goal of the experiment is to track a moving aircraft in space, using the current computer vision AAR techniques as accurately as possible in real images. By doing so, it will not only demonstrate the feasibility of computer vision based AAR, but will also prove the fidelity of the testing framework. This is achieved by performing stereo block matching (SBM) and iterative closest point (ICP) on real images in real time, using the motion capture data as the truth for the aircraft's pose. The same experiment is conducted in both the real and 3D virtual worlds to compare the error trends. The scale of the experiment mimics the scale of a realistic refueling scenario (Fig. 10). A set of 4K stereo cameras are mounted approximately 8 meters high and view a receiving aircraft as the refueling tanker would. The slant distance from the cameras to the receiver at the far corner of our laboratory is 23.47 meters. This distance mimics a receiver's position relative to the tanker prior to contact. The receiving aircraft is represented by a 1/7th scale replica of a receiver in both the real and 3DVW. In order to make the real and virtual experiments as similar as possible, a full scale model of the motion capture lab was created in the virtual world. Using the data provided by the motion capture system, the objects in

Figure 10: Aerial Refueling

the 3DVW move in real time and match the position and orientation of their physical counterparts. The primary tracked object is the receiving aircraft replica. Its pose is the truth data against which ICP results are compared.

3.3 Virtual Environment

In order to visualize the truth data from the motion capture system, as well as the results of the computer vision AAR pipeline, a virtual environment representative of the real laboratory was created in the AftrBurner engine (Fig. 11). In figure 11 the virtual aircraft is the visualization of the motion capture system's pose data for the physical aircraft replica. The visualization of the AAR pipeline results appear as the yellow and red point clouds in figure 26. This to-scale 3D visualization enables the recreation of equivalent experiments in the virtual world. That is, error trends can be compared between the real and virtual experiments. There are 4 sets of stereo cameras in total, 3 virtual and 1 physical, which are referred to in this section as

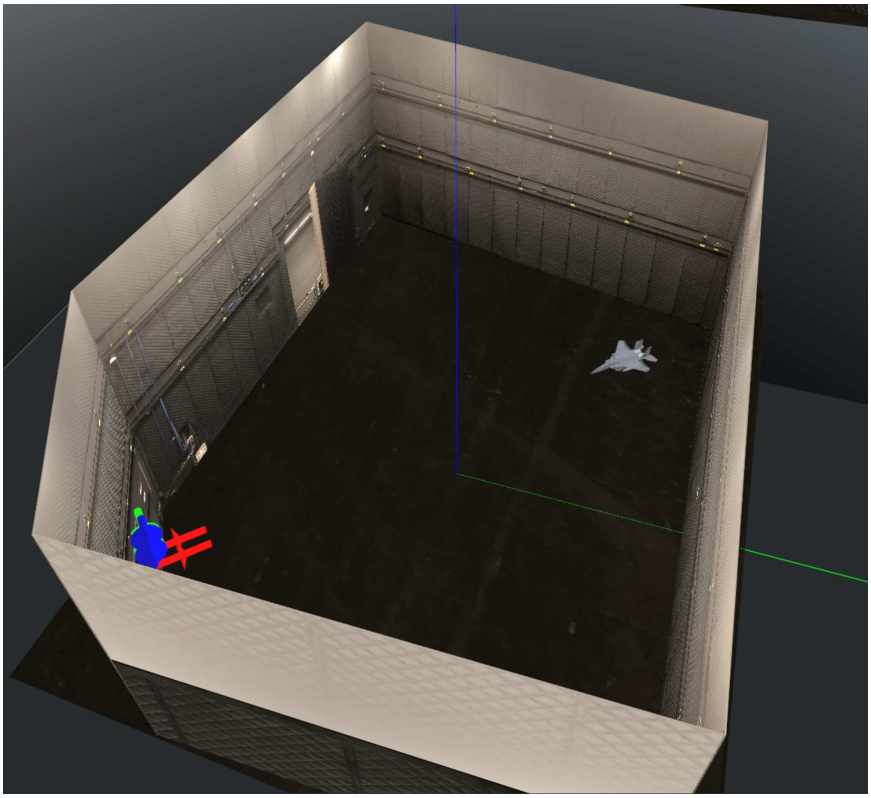

Figure 11: 3D Virtual Scale Model of the Motion Capture Laboratory

the real stereo cameras, virtual stereo cameras, ARReal and ARVirtual. The real stereo cameras (Fig. 12) are the physical cameras mounted in the lab that take real images of the physical aircraft replica. The virtual stereo cameras act as the virtual representation of the real stereo cameras. They are mounted in the virtual lab in the same position and orientation as their physical counterparts. In figure 11 the virtual stereo cameras are shown in the lower left quadrant of the image as a set of red, green and blue axes. These take virtual images of the experiments performed in the virtual motion capture lab that parallel those performed in the real lab. The ARReal and ARVirtual cameras are used to achieve augmented reality.

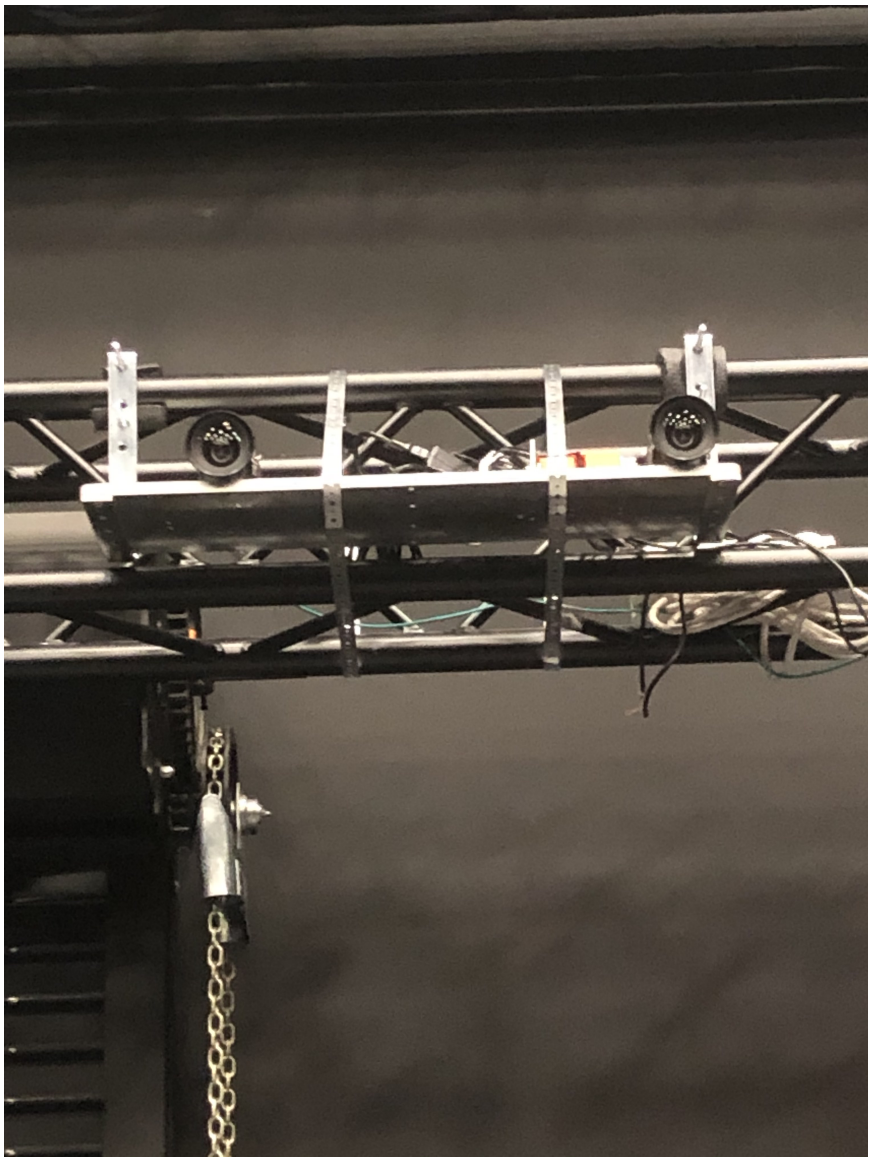

Figure 12: Real Stereo Cameras Mounted in Motion Capture Lab

3.4 Augmented Reality

The virtual environment enables the addition of an augmented reality boom to the receiver approaches. This is accomplished by adding 2 sets of additional stereo cameras, ARReal and ARVirtual, to the virtual environment; both of these observe objects in the distance that act as *green screens* for approach imagery (Fig. 13). These *green screens* are flat rectangular objects, which are texture mapped with

the images from the real and virtual cameras. The ARReal and ARVirtual cameras use the concept of a green screen to generate augmented imagery by placing a 3D refueling boom in front of their respective screens. Similar to green screens used in film, these rectangular models display sets of previously captured images while objects are filmed in front of them. The green screens that the ARReal and ARVirtual stereo cameras observe are displaying imagery sourced from either the real stereo cameras or the virtual stereo cameras. The real stereo cameras are in an elevated position in the corner of the motion capture laboratory to provide a distance from the receiver comparable to that of a realistic aerial refueling approach. The virtual stereo cameras are placed in the same position and orientation as their physical counterparts. In a non-augmented reality setup, stereo block matching would be directly run on the frames observing the receiver's approach. In this environment, however, the frames captured by the real and virtual stereo cameras are sent directly to the green screens,

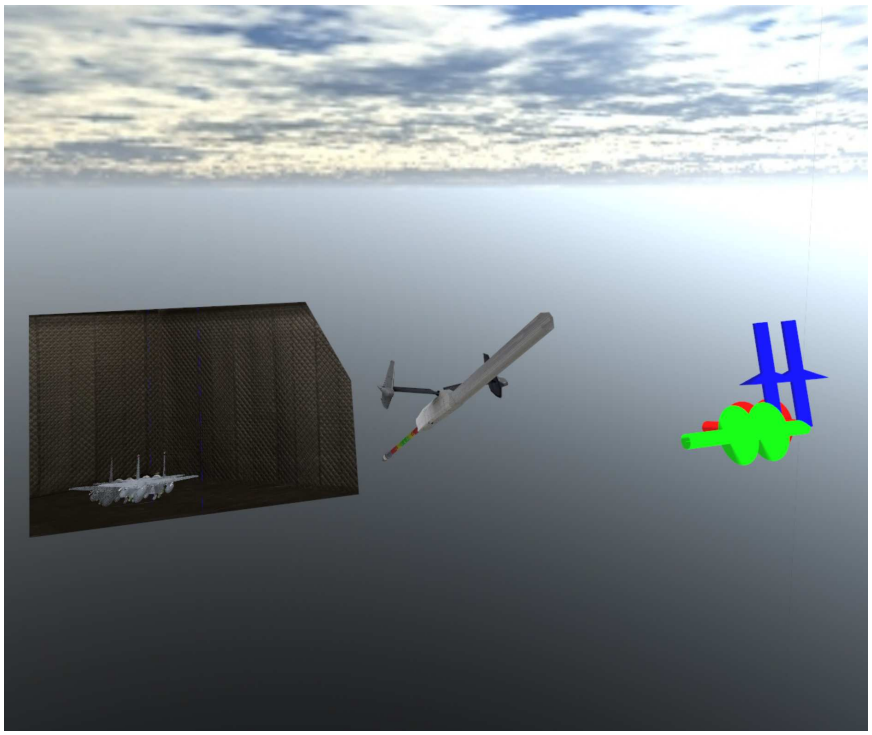

Figure 13: AR Green Screen

which perfectly fill the viewing frustum of the ARReal and ARVirtual cameras that observe them. This is accomplished using the following equations:

$$W = tan(\frac{FOV_H}{2}) \tag{2}$$

$$D = F * 0.99 \tag{3}$$

$$X = 2 * (W * D) \tag{4}$$

$$Y = \frac{X}{A} \tag{5}$$

Where W is the width of the camera's far clipping plane, FOV_H is the camera's horizontal field of view, D is the distance the green screen will be placed from the camera, F is the camera's far clipping plane, X is the x dimension of the green screen, Y is the y dimension of the green screen and A is the camera's aspect ratio.

A full scale virtual model of a boom is placed between these green screens and their observers (Fig. 14). The two booms are placed in the ARReal and ARVirtual coordinate frames using the equation:

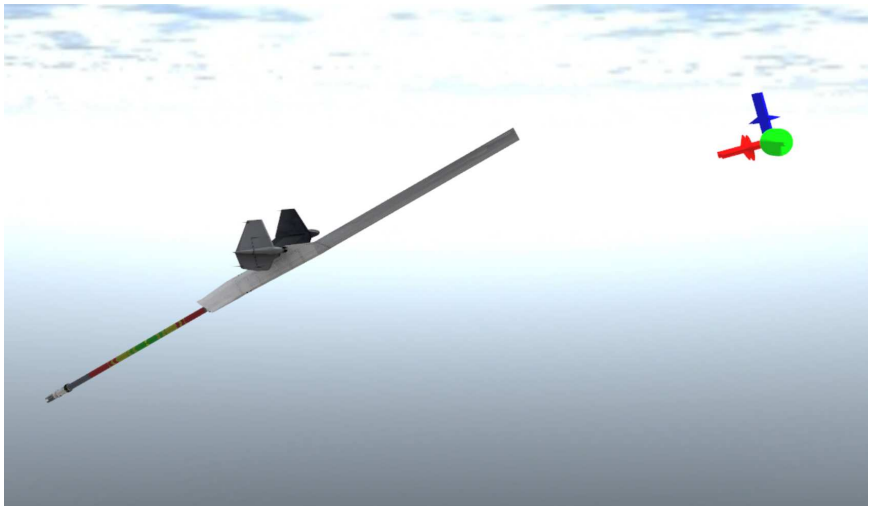

Figure 14: AR Boom

$$BoomPosition = CamDCM \cdot \begin{bmatrix} 13.0 \\ -0.25 \\ -2.0 \end{bmatrix} + CamPosition \tag{6}$$

for each set of stereo cameras. The AR booms are 3D accurate geometry that is virtually rendered on top of real imagery in a perspective correct manner. An observer viewing this virtual boom would realistically believe that it was in the original image if it were lit and colored properly, similar to special effects in a movie (Fig. 15). The boom's pose relative to the stereo cameras mimics that of a refueling tanker. Therefore, the portion of the image that the boom occludes will be the exact same pixels that a real boom would occlude. This provides an accurate and realistic method to test the effects of boom occlusion (Fig. 44).

In order for the comparison between real and virtual experiments to be valid, the virtual environment must mimic the real one as closely as possible. The goal is for the real and virtual cameras to see identical imagery during an approach (Fig. 17). The first step in achieving this is to create a full scale replica of the motion capture lab (Fig. 18). The room is measured and the corner angles calculated (Fig. 19). Images of the walls are texture mapped onto the virtual model of the room. A 1/7th scale replica of a realistic receiver is used as the physical model in the laboratory. The length to wingspan ratio of the physical model is within 0.006 of a full scale receiver (Fig. 20). The virtual model is a full scale replica of a receiver with identical dimensions and a length to wingspan ratio of 0.671. The physical model has a length to wingspan ratio of 0.677. In order to maintain consistency between the real and virtual environments, the virtual model is scaled down by 0.142 to match the size of its physical counterpart.

Figure 15: Virtually Rendered Geometry on Real Images

Figure 16: Left: Augmented Reality Right: Augmented Virtual Reality

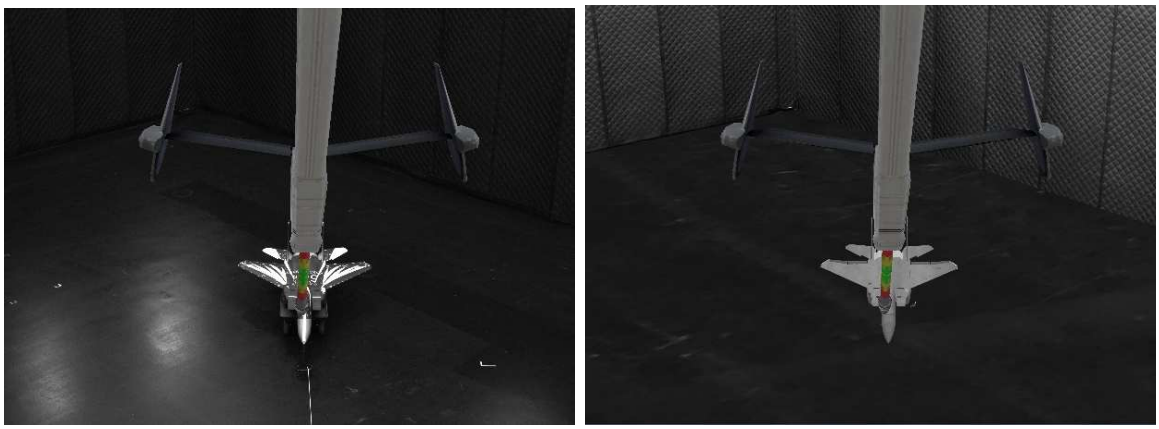

Figure 17: Left: Augmented Reality Top Down Right: Augmented Virtual Reality Top Down

Figure 18: Real vs Virtual

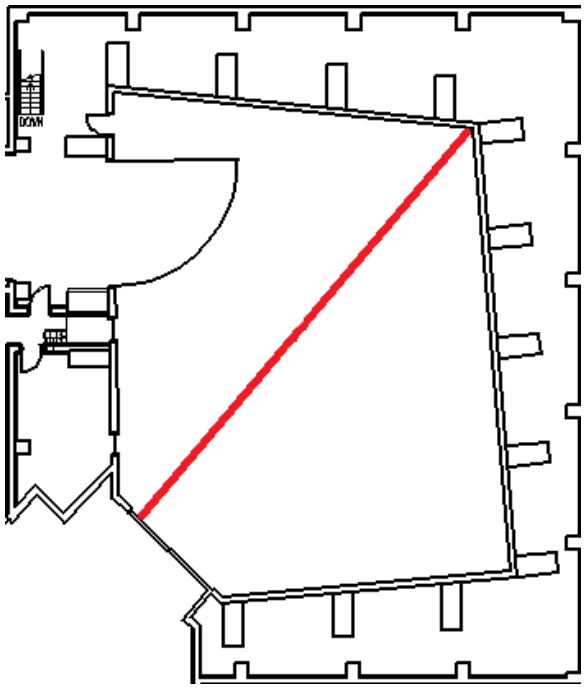

Figure 19: Motion Capture Room Diagram

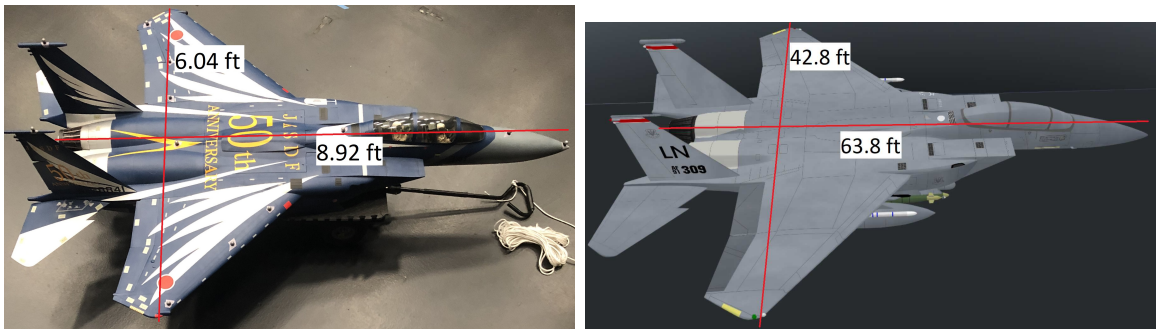

Figure 20: Model Receiver Size Ratio

3.5 Truth Data

The laboratory is equipped with an IR camera based motion capture system (MCS), which provides the truth data for the mock refueling approaches. The motion capture system outputs 75Hz, millimeter accurate position and orientation for any tracked objects within its bounds.

3.5.1 Communications

Motion capture data and images must be simultaneously acquired and processed in order to associate them properly. Vicon Bridge, a ROS application, is used to listen for the data coming in from the MCS. The data collection machine is running Windows 10, but Vicon Bridge and ROS only work in Linux. The Windows Subsystem for Linux (WSL) is used on the data collection machine to run the ROS application. Vicon Bridge listens for the data and publishes it as ROS topics. A ROS application is written to listen for the data from Vicon Bridge and to convert it into an AftrBurner Engine NetMessage that is sent to the 3DVW over the loopback address (Fig. 21). On Windows 10, an AftrBurner Engine module asynchronously acquires stereo image pairs, as well as the corresponding MCS truth pose data. These disparate data streams are aligned and saved to file and can be replayed deterministically.

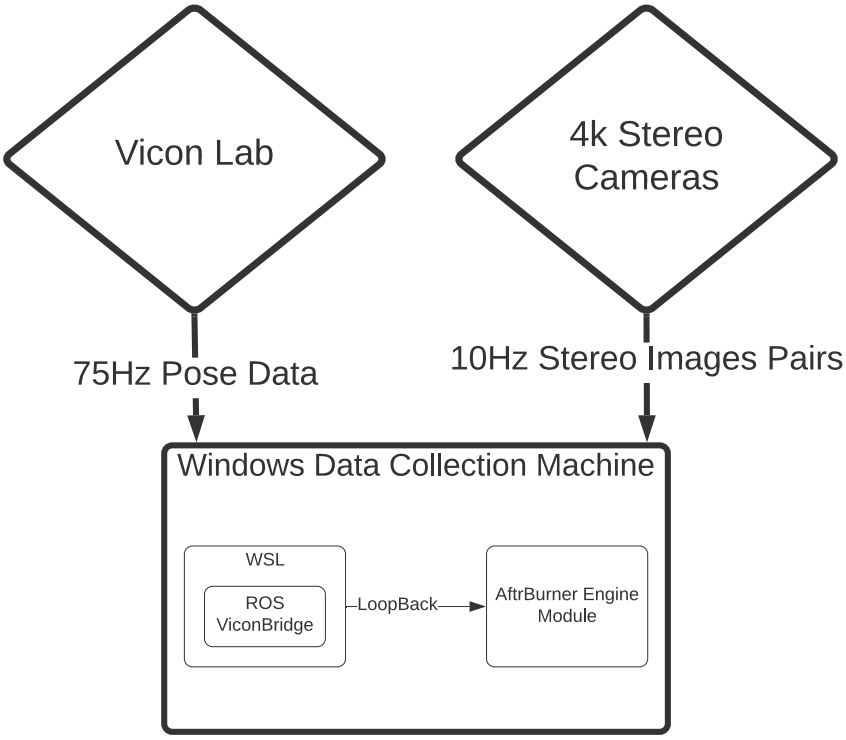

Figure 21: Communications Diagram

3.6 Camera Configurations

Camera configurations, in the context of this paper, are describing the position and orientation of the real stereo cameras in the lab, as well as their settings. Multiple stereo camera configurations were used before determining which works best. Initially, a pair of 4k Prosilica cameras were mounted in the corner of the motion capture lab, approximately 3 meters off the ground. The lighting in the lab is dim compared to outdoors. In order for the Prosilicas to capture images, the exposure setting had to be increased. The exposure setting determines how long the cameras' aperture is open, thus letting in more or less light. Images need to be captured at 10Hz to perform AAR in real time, but the exposure settings required to capture enough light in the dimly lit lab prohibited the cameras from capturing that quickly. The result was that the exposure was set to the point immediately before the cameras could no longer

capture at 10Hz, causing the images to be darker than desired. Additionally, many of the images captured by the cameras with these settings came out blurry, likely due to the aperture being open for so long. The blurry image quality made doing camera calibrations infeasible.

A pair of grayscale 4k FLIR cameras alleviated the unusual exposure problem. The FLIRs captured clear, bright images at 60Hz, even in the dimly lit laboratory. These cameras initially were placed in the same location as the Prosilicas. With the FLIRs, a good camera calibration was obtained. A reasonable sensed point cloud was generated using imagery from this configuration, but the accuracy was not sufficient. The cameras in this position were viewing the receiver from a head-on perspective. In a real refueling scenario the cameras would be above the receiving aircraft viewing it from a top down perspective. It was suspected that the head-on perspective reduced the accuracy of SBM and ICP.

The third and final configuration places the FLIR cameras in a more elevated position, approximately 8 meters high, and pitched down (Fig. 22). This creates a viewing angle more similar to a real refueling scenario. Additionally, it creates a larger area to conduct a mock approach before reaching the contact point at 20 meters (Fig. 10).

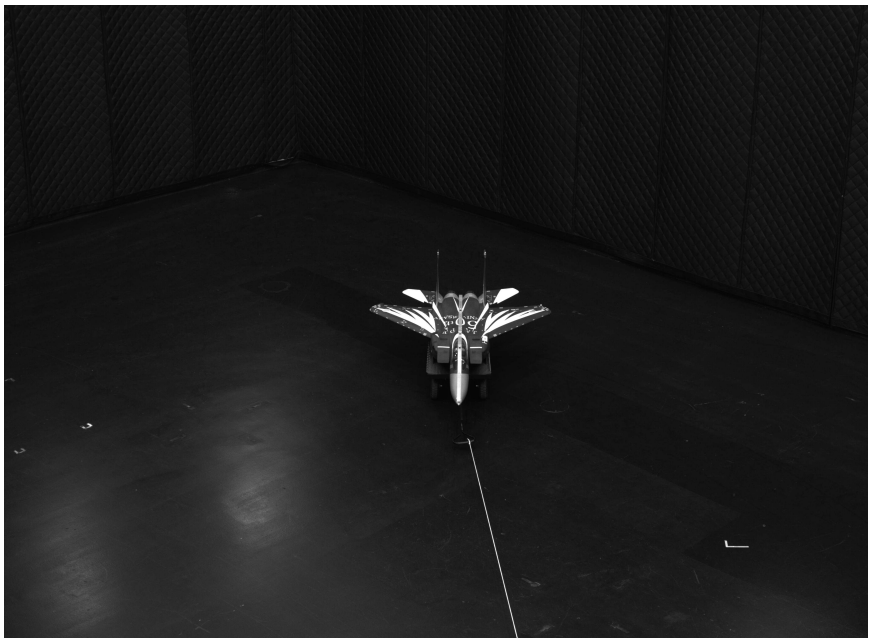

Figure 22: Top Down Camera POV

3.7 Sources of Error

Even with the real stereo cameras in an ideal location, the results still did not meet the accuracy requirements. A reasonable sensed point cloud was obtained, which resembled the receiving aircraft, but was slightly offset in one direction or another (Fig. 23). Before attributing this error to factors such as a poor calibration or SBM being inadequate, all sources of error within the confines of this thesis's testing framework must be examined. This section will systematically rule out or remedy each of those possible sources of error.

3.7.1 Timing Test

While a mock refueling approach is being conducted, the images from the cameras and the pose data from the MCS are being recorded asynchronously into a single log file. The image from a camera must be correctly associated with the MCS pose data from the instant that image was taken; otherwise, the image shows the receiving

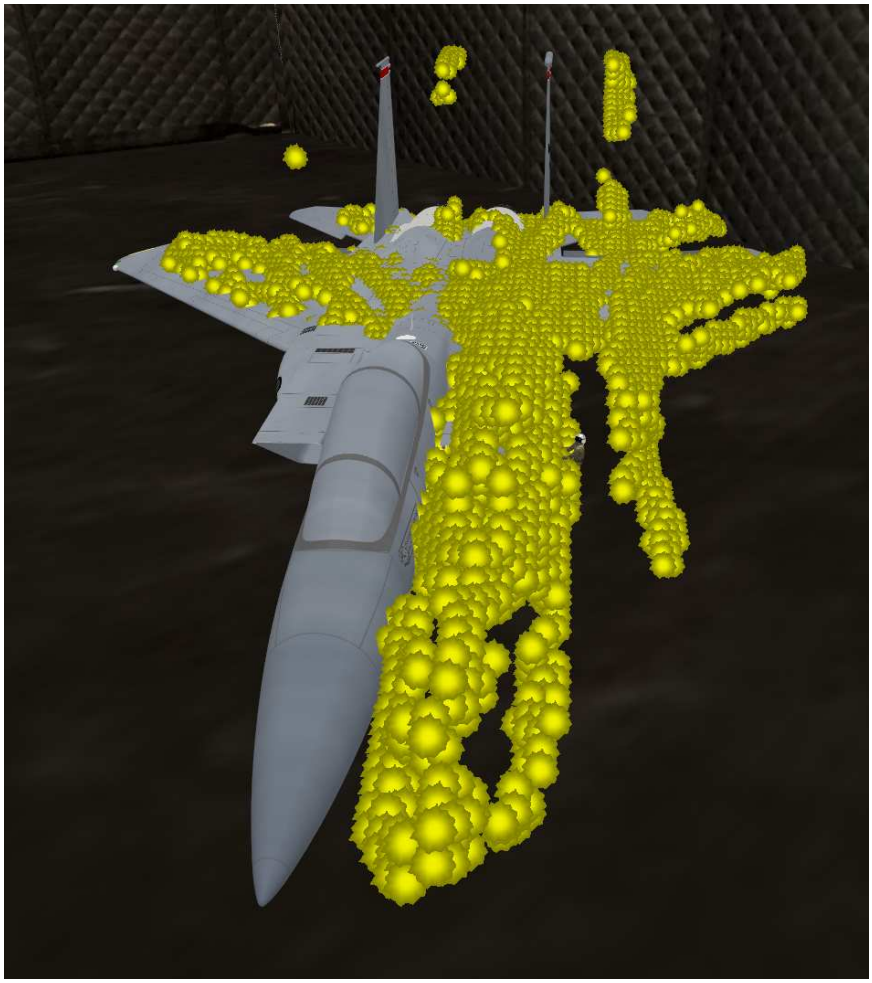

Figure 23: Offset Point Cloud

aircraft in one location, but the MCS truth data claims it is in another. This will cause accurate results to appear to be incorrect.

A timing test was performed to rule out a timing offset between MCS pose data and images as a possible source of error. To conduct the timing test, a MCS tracked checkerboard object was used. The checkerboard was moved directly up and down while being viewed by the cameras and tracked by the MCS. Images and motion capture data were recorded at 30Hz. The Z component in the MCS coordinate frame was plotted, along with the Y component in the left camera's image frame. These are the vertical components in the respective coordinate frames, so a successful timing

test should depict the points of inflection on the graph occurring at the same data points. This was observed as seen in Fig. 24, which indicates that, if there is a timing offset, it is less than 1/30th of a second. Zooming into the graph reveals that all points of inflection occur at the same data points (Fig. 25). Any error caused by an offset that size is negligible due to how slowly the receiver is moving.

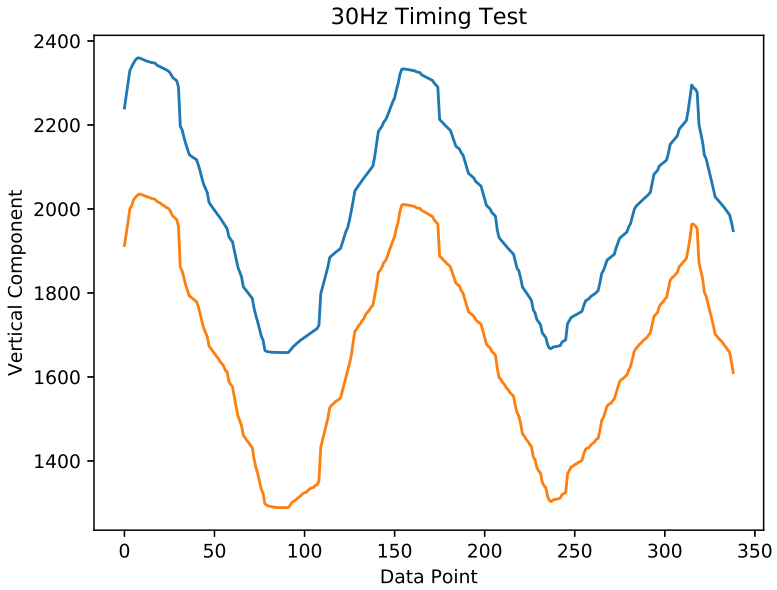

Figure 24: Timing Test

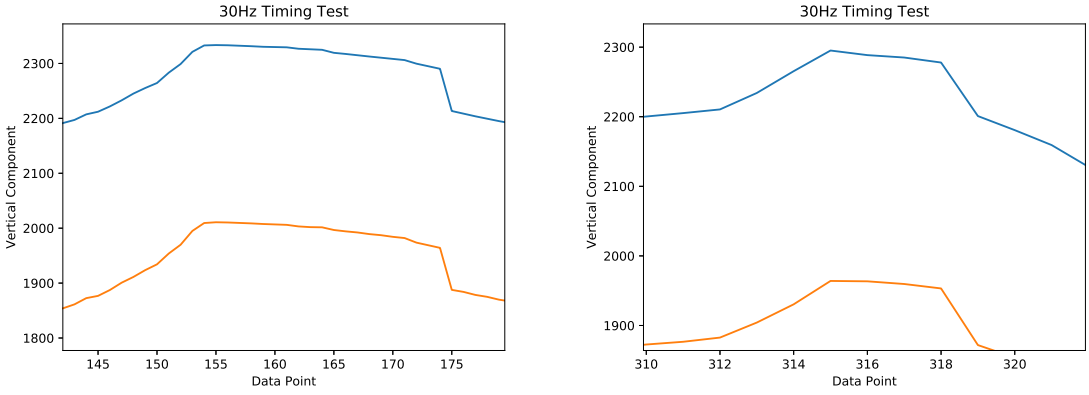

Figure 25: Timing Graph Close Up

3.7.2 Physical Receiver Pose vs Virtual Receiver Pose

The final step in the AAR pipeline, iterative closest point (ICP), is used to match the shelled reference model of the aircraft onto the sensed points (Fig. 26). The error value that this produces is based on the difference between the origin of the shelled reference model and the origin of the virtual receiver. The origins of these two virtual objects coincide when the shelled reference model lies perfectly on top of the virtual receiver.

The position and orientation of the virtual receiver is based on the data from the MCS and must precisely match the physical model in order for the truth data to be meaningful. SBM is being performed on real images of the receiver to create the sensed point cloud; therefore, if the virtual receiver is not in the correct location, an accurate set of projected points will appear incorrect. This problem exists because the MCS data for the physical model's position and orientation is based on an origin that is different from the virtual receiver. The position of the origin for the physical model is the center of mass of the motion capture markers (Fig. 27) that are used to define the object in the MCS tracker software. The orientation of this object's coordinate

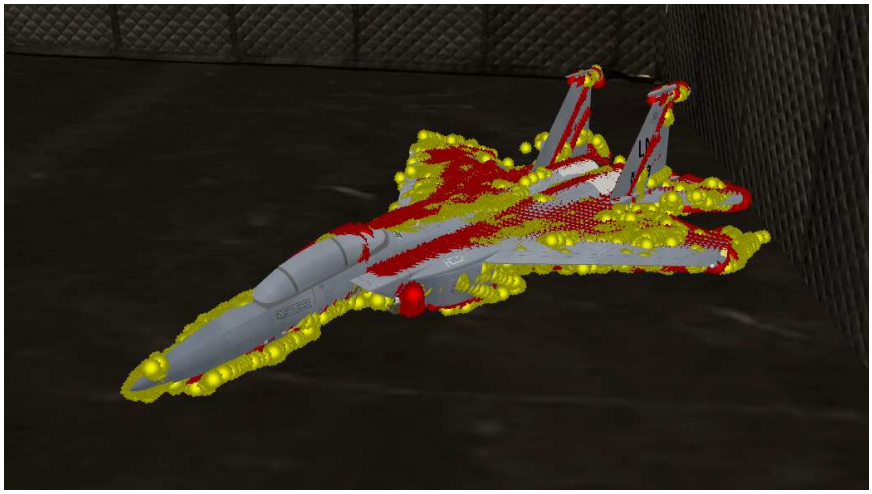

Figure 26: Shelled Reference Model (Red) Matched to Sensed Points (Yellow) MCS Truth Data is the Textured Aircraft Model

Figure 27: Approximate Location of Receiver's Center of Mass According to MCS

frame is defined from the object's orientation when it is created in the MCS tracker software. For example, if the right wing were facing the positive x direction and the nose were pointing in the positive y direction, then the positive y direction for the aircraft's coordinate frame is toward the nose and the positive x toward the right wing.

To find the vector offset from the physical model's origin to the virtual model's origin, a combination of physical measurements and 3DVW information is used. First, the tip of the nose of the physical model is placed directly over the origin with the body of the aircraft along the negative x axis. The accuracy of this is improved by ensuring that the object's motion capture markers are aligned along the axis in the MCS tracker software and that the nose marker is at the origin. The distance from the floor to the tip of the nose is measured then in the real world. In the virtual world, the XYZ offset from the tip of the nose of the virtual receiver to the origin and known height is recorded (Fig. 28). This translation vector is in the receiver's coordinate frame, but, in order to apply it to the receiver's position, it must be transformed into

the MCS coordinate frame.

$$Offset = \mathbf{R}_{Recv}^{MCS} \cdot \begin{bmatrix} 0.271 \\ 0.032 \\ 0.051 \end{bmatrix} \tag{7}$$

The vector on the right side of equation 7 is the measured distance from the virtual aircraft's nose to the MCS origin. The physical aircraft's nose is directly over the MCS origin; therefore, this vector is also the offset from the virtual aircraft origin to the physical aircraft origin. The resulting vector from equation 7 is applied to the virtual receiver every iteration of the simulation.

In order for the rotation to match between the physical and virtual receiver models, the physical model must be placed at the exact identity rotation of the virtual receiver when the object is created in the MCS tracker software. This is infeasible and adjustments are made in the virtual world to compensate. Physical measurements are taken from the ground, which is the z plane in both the real and virtual environments, to the tip of each wing, as well as to the tip of the nose and tail. The heights of the corresponding points on the virtual model are recorded. The rotations to move the virtual model into the same orientation as the physical model are applied. The physical and virtual models now have the same origin and rotation matrix and can be trusted as truth data in the real and virtual environments.

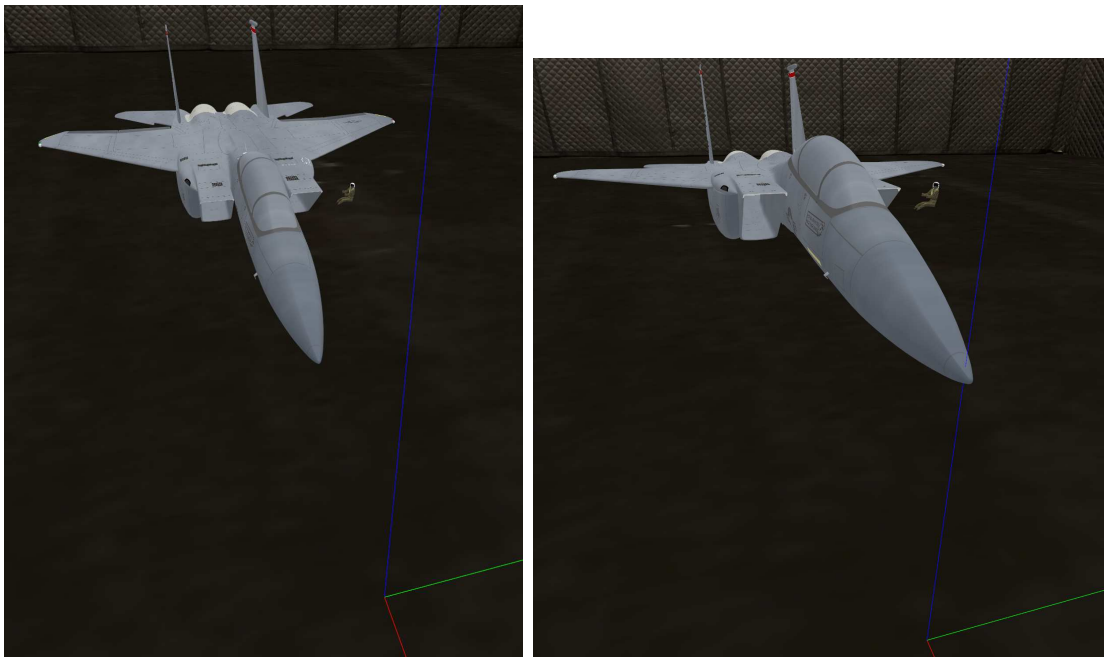

Figure 28: Virtual Receiver Before and After Offset Applied

3.8 Real Stereo Camera Pose vs Virtual Stereo Camera Pose

The MCS is capablable of outputting accurate pose data, but this comes with the caveat that the tracked object must be large enough to accommodate a sufficient number of motion capture markers. A small object, such as a single camera (Fig. 29), does not provide a stable enough orientation to be used as truth data in the experiments.

A critical step in validating the truth data is to ensure the position and orientation of the virtual stereo cameras match that of the real stereo cameras. Similar to the issue with the virtual receiver not being in the correct location, the virtual cameras not being placed correctly cause accurate sensed points to appear incorrect. After SBM is performed, the sensed points are reprojected in the virtual world based on the virtual stereo cameras' position and orientation. Even small errors in rotation, of approximately 1-2 degrees, cause large error in the sensed point cloud position at 20m. Initially, motion capture markers were placed on the cameras and the MCS

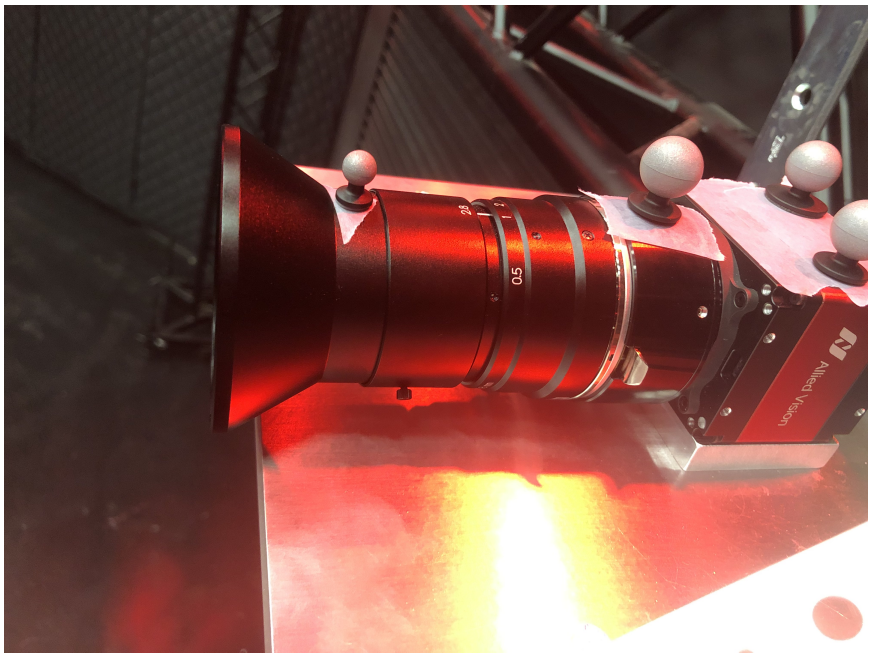

Figure 29: Tracked Camera

was adjusted and recalibrated in order to capture pose data for these objects. It was determined that the accuracy of the camera object was not sufficient, likely due to the close proximity of the markers and how close the object was to the boundaries of the motion capture area (Fig. 29). Additionally, the camera object's origin in the MCS tracker software is offset from the camera's center of projection from which the sensed points should be reprojected. These issues for the camera object prove more difficult to solve than those encountered with the receiver.

3.8.1 Gauss-Newton Optimization

A Gauss-Newton optimization is used to find the position and orientation of the real stereo cameras in the MCS coordinate frame. A slightly modified technique to the one used in [26] is adapted for this problem. In [26], the authors' goal was to find the rotation and translation from the camera to the mount to which it was attached. This was necessary because the motion capture data for the camera was not accurate

enough for image reprojection, but it was accurate enough to yield the pose of the larger mounting object. This paper requires the rotation and translation from the motion capture system (MCS) coordinate frame to the camera (C) coordinate frame. The virtual environment coordinate system is coincident with the MCS coordinate frame, so finding the pose of the camera in the MCS coordinate frame dictates where to place it in the virtual environment for accurate reprojection.

The Gauss-Newton attempts to minimize the error between the measured locations of corners on a chessboard in captured images and where these corners are estimated to be based on the initial guess for camera position and orientation. The measured values are the image pixel coordinates of each corner. These are obtained using OpenCV's findChessboardCorners function, which returns the pixel coordinates for every corner in each image. The estimated corner locations are calculated using equation

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} x^{im} \\ y^{im} \\ z^{im} \end{bmatrix} = \mathbf{K}\mathbf{T}^{C}_{MCS}\mathbf{T}^{MCS}_{CB}\mathbf{T}^{CB}_{FB}\mathbf{x}^{FB} \tag{8}$$

Where $\mathbf{T}$ represents a 4x4 transformation matrix of the form

$$\mathbf{T}^{a}_{b} = \begin{bmatrix} \mathbf{R}^{a}_{b} & \mathbf{0}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3} & -\mathbf{t}^{b}_{b,a} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \tag{9}$$

The subscripts of the transformation (T) and rotation (R) matrices describe the coordinate system they are going from. The super scripts describe which coordinate system they are converting into. The translation vector $-\mathbf{t}^{b}_{b,a}$ denotes a vector from coordinate system b's origin to coordinate system a's origin in b's coordinate frame. In the previous equation, the corner coordinates $\mathbf{x}$, which are in the flatboard co-

ordinate (FB) system, are being transformed from flatboard to checkerboard (CB), checkerboard to motion capture system (MCS), motion capture system to camera (C), and, finally, into image coordinates using the calibration matrix $\mathbf{K}$. In this case $\mathbf{K}$ is represented as a 3×4 matrix.

The corners start in the flatboard coordinate system. The origin of this 2-dimensional coordinate system is the top left corner; subsequent corners are represented by their distance away from the origin in millimeters. For example, the top left corner is [0,0], and the next corner to the right is [0,60]. The checkerboard has motion capture markers on it and is being tracked by the MCS. The origin of this object is not the top left of the board; rather, it is toward the center and slightly displaced in the z direction (away from the front of the board). This coordinate frame is referred to as checkerboard. The rotation and translation from FB to CB ($\mathbf{T}_{FB}^{CB}$) is one of the transformations being optimized for; however, the authors of [26] provided their results for this step, which are used as the initial guess. Since it is the same checkerboard with the same markers, it proved accurate and showed negligible change with optimization.

The next step is to transform from the CB coordinate frame to the MCS coordinate frame. The rotation and translation are provided by the MCS since the checkerboard is a tracked object. Finally, the corners are transformed from the MCS coordinate frame to the camera coordinate frame using the initial guess for camera rotation and translation. These are the primary values being optimized. Once the corners are in the camera coordinate frame, the calibration matrix K is used to reproject them into image coordinates. Once expressed as image coordinates, the estimated points can be compared against the measured corners' locations.

The Jacobian matrix has 2 rows and 12 columns for every corner from each image. The rows are representing the u and v pixel coordinates, and the columns are the

derivatives of equation 8 for the 12 values that are being optimized. These include the translations from the FB to CB with respect to x, y and z, the rotations from FB to CB with respect to the x, y and z rotation axes, as well as those six values for the MCS to C.

Critical aspects for the success of this technique are accurate initial guesses, as well as quantity and quality of images. If the camera is in a location where the MCS can track it, the provided pose is a suitable initial guess. The final physical camera position this paper uses was too high to be tracked, so an alternate method was required. First, a plumb line was dropped to the ground from the camera. This measurement provides the approximate z coordinate for the camera in the MCS coordinate frame. A tracked object was placed on the ground with its center of mass directly below the plumb line. This provided the approximate x and y. For the rotation approximation, a recorded mock refueling approach was required. When the mock refueling approach was replayed in the virtual environment, the receiver's sensed point cloud was reprojected into the virtual world. Small, manual adjustments were made to the virtual camera's rotation until the sensed point cloud lay on the virtual receiver and the error was minimized. This orientation was then recorded and used as the initial guess for the optimization. A sufficient number of images must be used, and the images must be taken at varying distances from the camera, ideally along the entire length of the receiver approach path. Additional images will improve accuracy up to a point; however, beyond that point there are massive increases in execution time for negligible improvement in error. A few hundred images are ideal and the optimization takes approximately 20 minutes with 10 iterations.

The following images in Fig. 30 show where the corner points were estimated to be based on the initial guesses. While those in Fig. 31 show the estimated points after the optimization is complete. Before the optimization, the average distance between

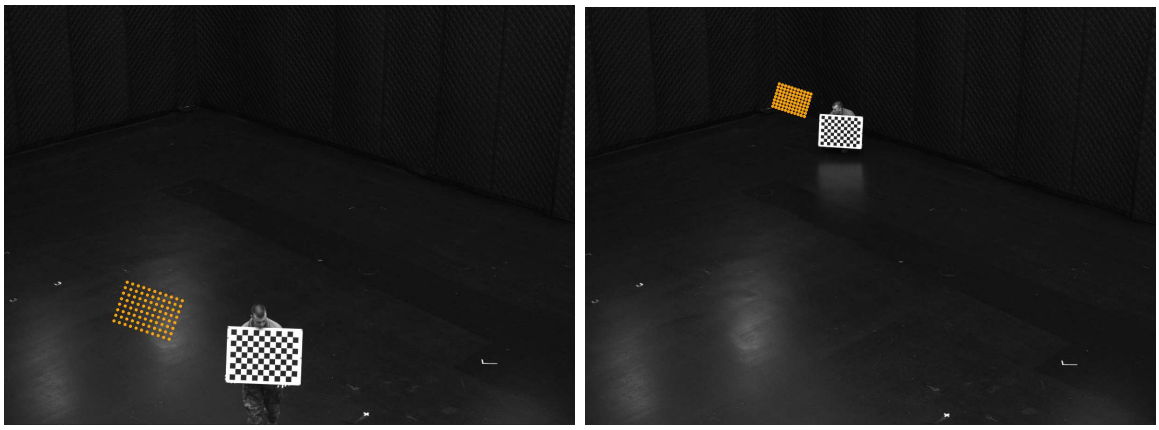

Figure 30: Before Gauss-Newton Optimization

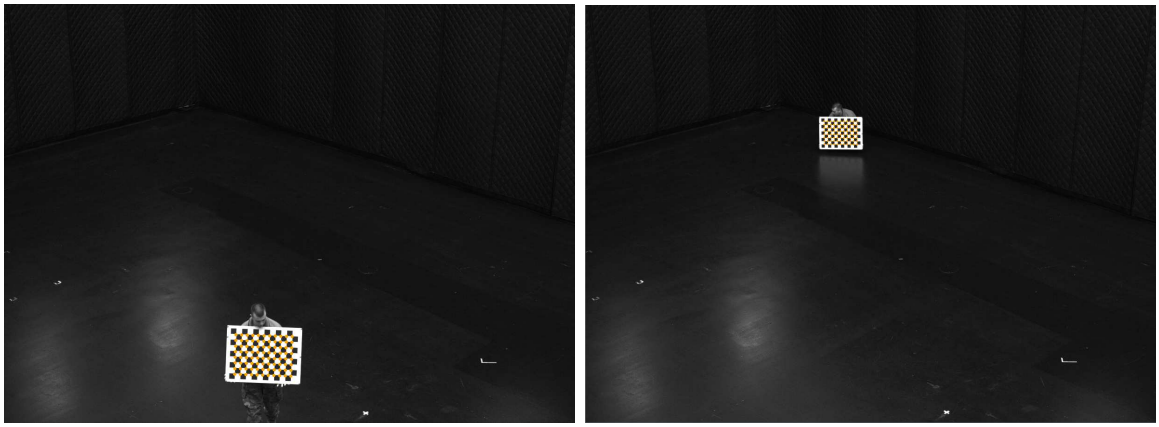

Figure 31: After Gauss-Newton Optimization

the estimated corner coordinates and the measured corner coordinates was about 280 pixels. After optimization the average offset was about 3 pixels. This demonstrated that an accurate pose estimate for the cameras had been obtained.

3.8.2 Reprojection Filters

The final source of error this section will discuss is the filtering of reprojected points. Once an accurate sensed point cloud is obtained, ICP matches the red truth point cloud to it as closely as possible. If the sensed point cloud is a perfect shelled model of the aircraft and is in the correct position, but there are sensed points elsewhere in the world, the truth point cloud will be pulled toward them and results will

appear inaccurate.

The proximity of the receiver to the floor and walls creates a unique problem for filtering reprojected points. In a real refueling scenario, a less restrictive reprojection filter would be sufficient since there is nothing near the receiving aircraft. In this experiment, however, a very restrictive filter must be implemented to avoid including points from the floor and walls in the sensed point cloud. The filter is typically in the stereo cameras' coordinate frame. Due to the angle at which the cameras are viewing the aircraft, this precludes the ability to perfectly filter points outside of the aircraft. Since having surfaces so close to the receiver is unrealistic, the typical technique of filtering in the camera's coordinate frame is bypassed. Instead, a filter that relies on the information from the 3DVW is used to perfectly remove any reprojected points that lie outside the bounding box of the virtual receiver (Fig. 32).

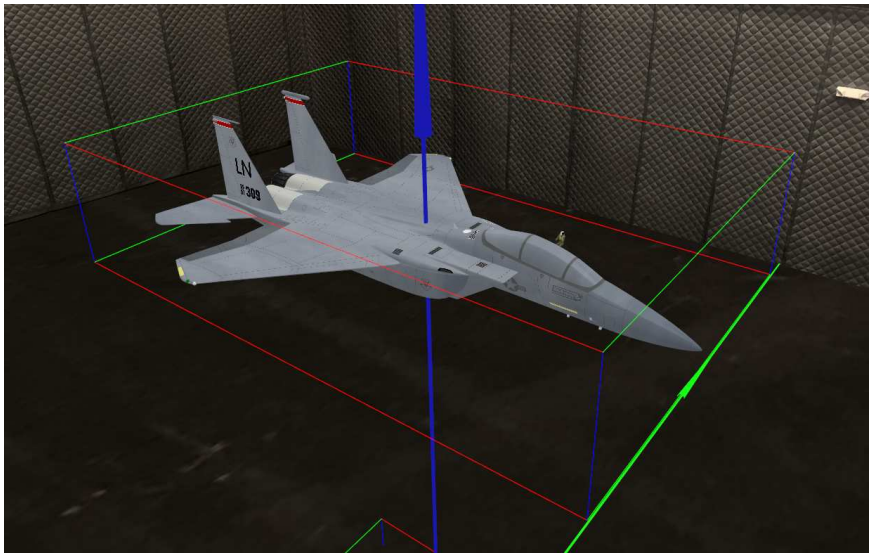

Figure 32: Receiver Bounding Box

IV. Results and Analysis

4.1 Preamble

This chapter presents the results from multiple experiments, which are explained in further detail at the beginning of each section. There are five variations shown, each performed in the real and virtual environments. These include a straight in approach, a serpentine pattern, an occluded straight in approach, a diagonal approach and, finally, a maximum distance approach. Each experiment will include the translation error in one graph and the rotation error in another.

To preface the results, it is helpful to understand the difference between the disparity maps created by the real and virtual images. A real camera introduces randomness when capturing an image; however, the virtual cameras do not. This results in SBM finding more features in the images than it normally would (Fig. 33). The dense virtual disparity map creates a noisier point cloud than the real disparity map, which pulls the red truth model away from the correct position and orientation in the virtual approaches.

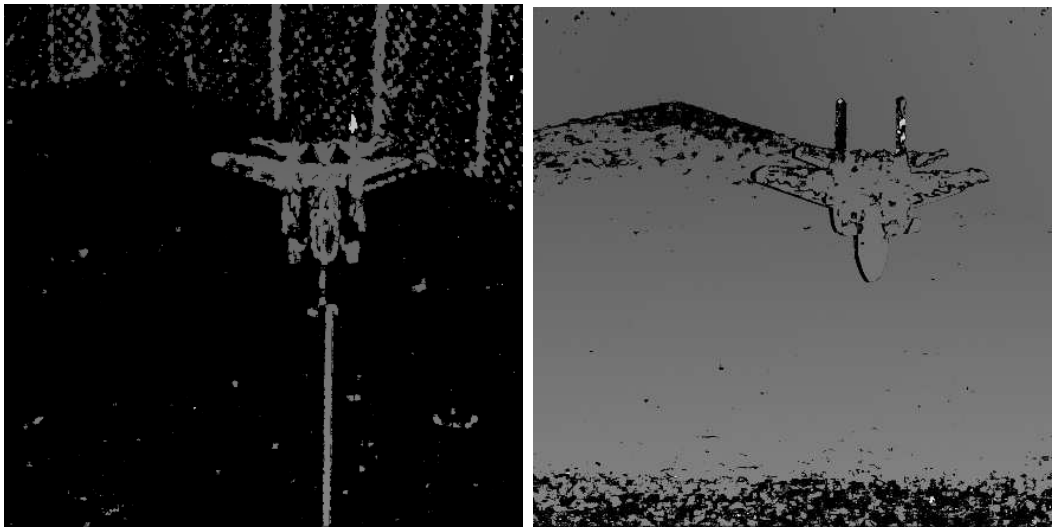

Figure 33: Left: Real image disparity map Right: Virtual image disparity map

4.2 Straight in Approach

The path of the receiver in this set of images is representative of how the tanker would view the aircraft during a refueling approach. It approaches from the top center of the image frame and moves towards the bottom, with the nose facing the cameras (Fig. 22). Figures 34 and 36 show the translation and rotation error for the approach done in the real environment, while figures 35 and 37 show the error for the same, exact approach performed in the virtual environment with the virtual cameras.

4.2.1 Translation Graphs

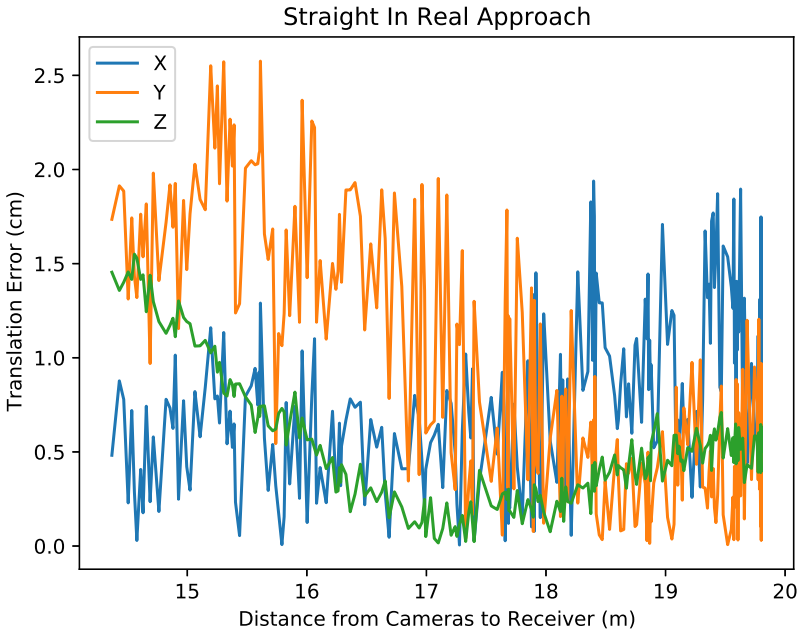

Figure 34: Straight In Real Approach Translation Error Graph

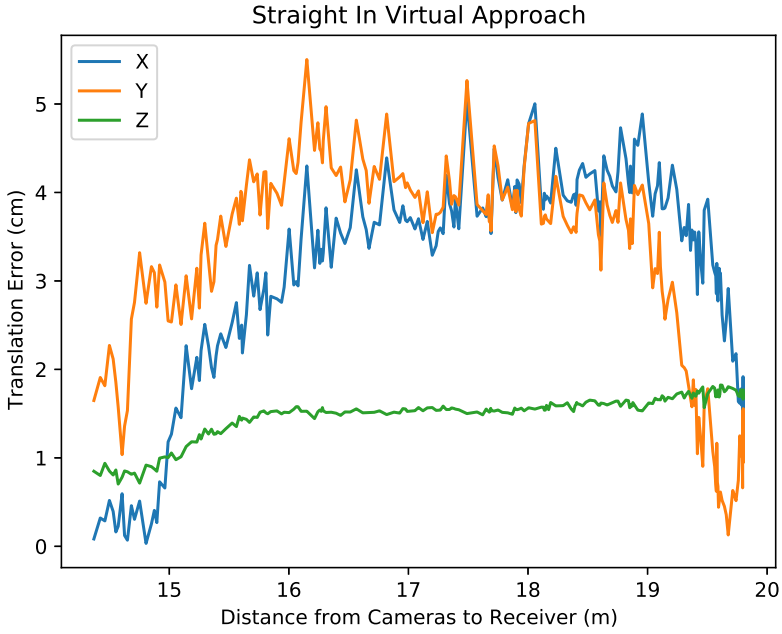

Figure 35: Straight In Virtual Approach Translation Error Graph

The only similarity observed between the real and virtual approaches, in this case, is that the z component is the most stable and experiences the least amount of error. Overall, both approaches remain within 6cm of error.

4.2.2 Rotation Graphs

The rotation error degrades as the receiver moves closer to the cameras. This is primarily observed in the roll component. An imperfect extrinsic calibration is suspected as the cause, with the error becoming more obvious as the receiver approaches the cameras. The increase in roll error is observed in both the real and virtual approaches.

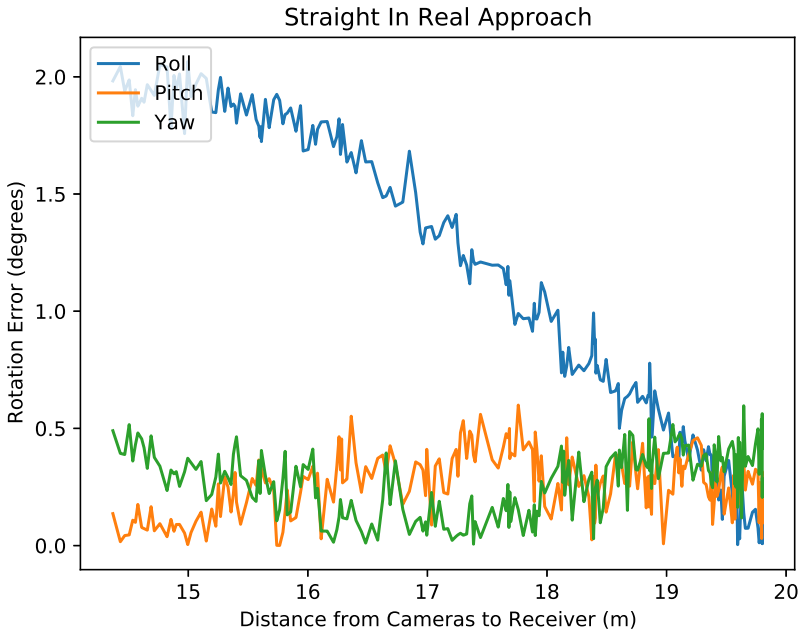

Figure 36: Straight In Real Approach Rotation Error Graph

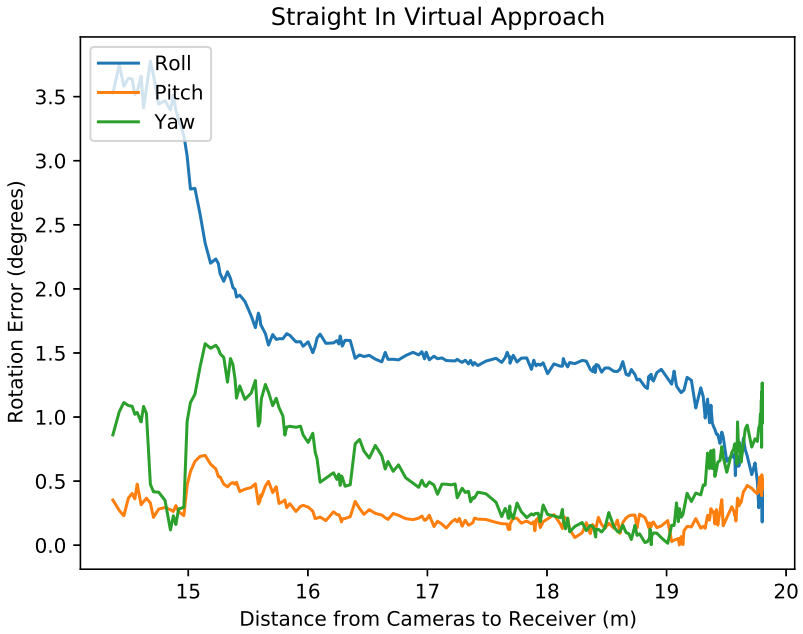

Figure 37: Straight In Virtual Approach Rotation Error Graph

4.3 Zig Zag Approach

To thoroughly test the employed algorithms, the receiver is dragged in a serpentine pattern. This would not be done during a real refueling approach, but it helps to reveal weak spots in the current AAR process. Towards the end of the approach path, the receiver makes a sharp turn to move from the right side of the frame across to the left side (Fig. 38). At this point the ICP algorithm converges incorrectly, causing the translation and rotation error to spike until the receiver is turned back towards the cameras. The truth model point cloud is created from the perspective of the cameras with the aircraft facing them; therefore, the majority of the points in that shelled reference model are on the front and top of the receiver. A more complete truth model could improve the results of ICP when viewing the receiver from the side.

This approach is valuable because it demonstrates that the 3DVW can predict errors in the real world. The sudden spike in error occurs in both the real and virtual

environments at approximately 17 meters from the cameras and is corrected at about 14.5 meters.

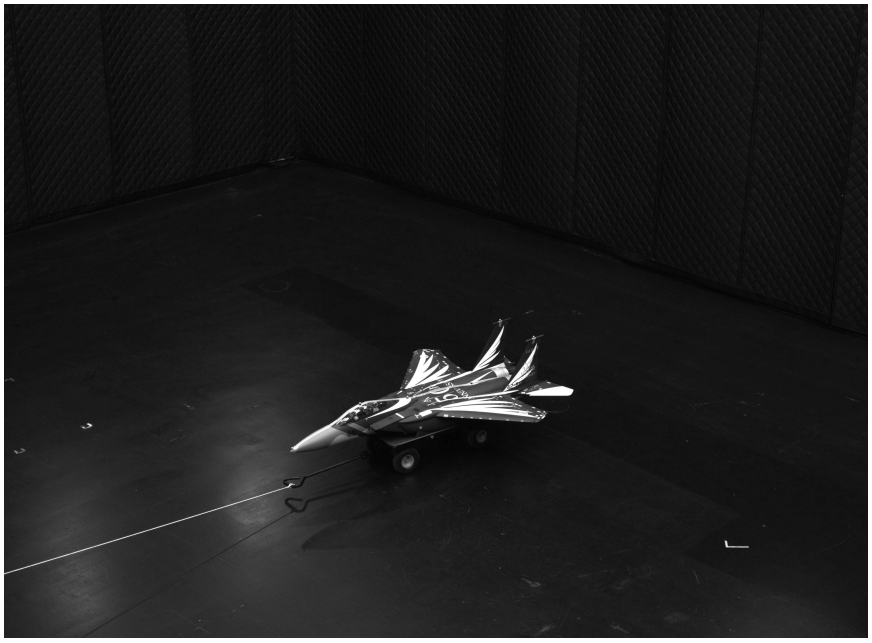

Figure 38: Receiver Side View

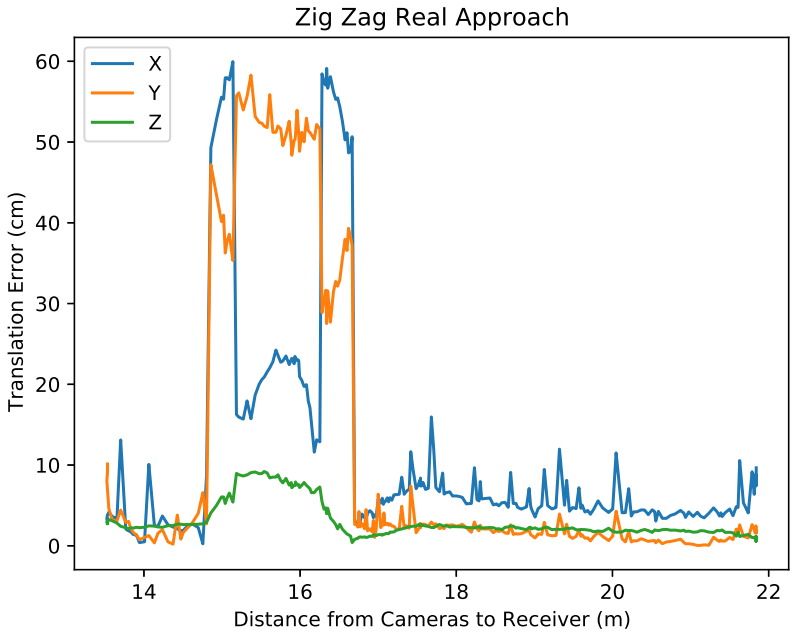

Figure 39: Zig Zag Real Approach Translation Error Graph

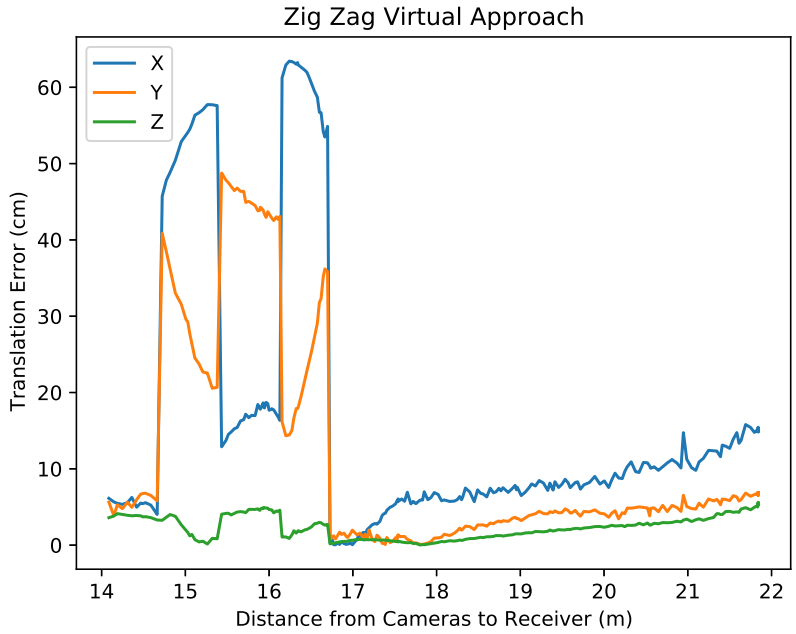

Figure 40: Zig Zag Virtual Approach Translation Error Graph

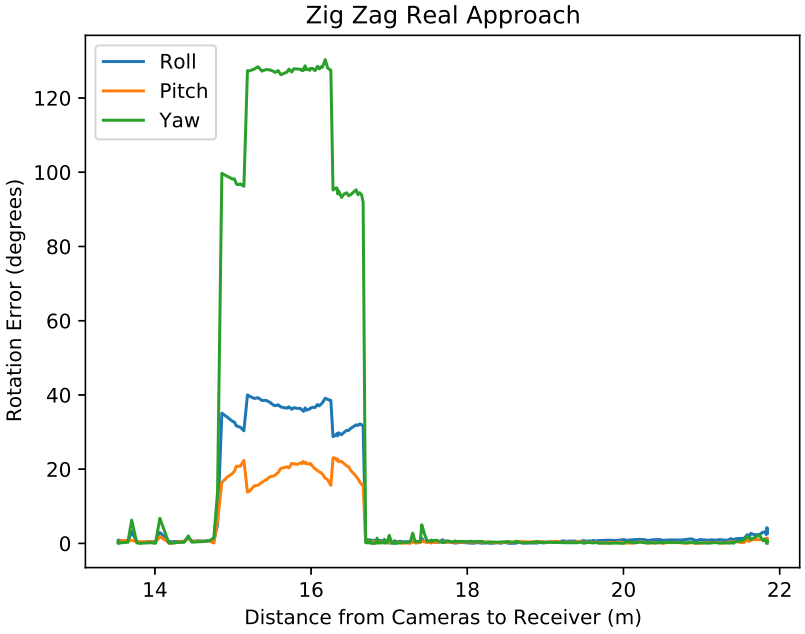

Figure 41: Zig Zag Real Approach Rotation Error Graph

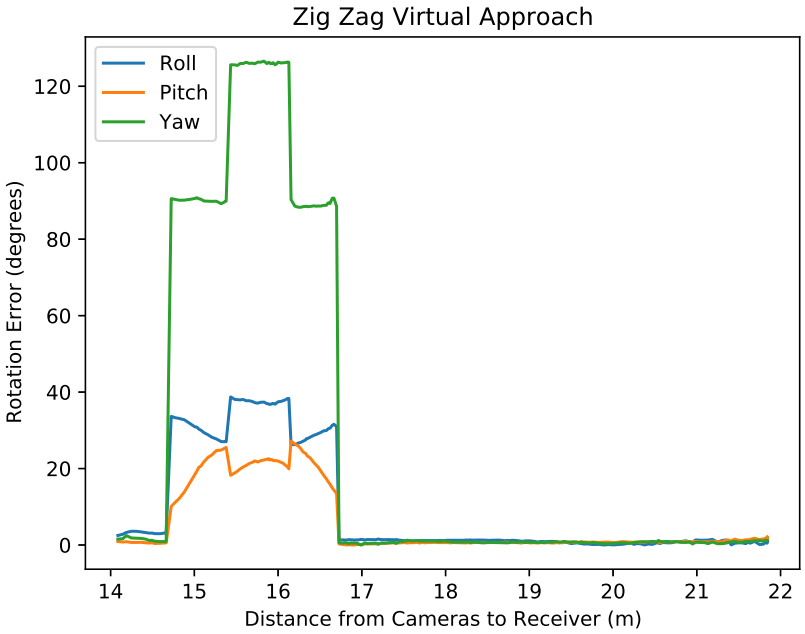

Figure 42: Zig Zag Virtual Approach Rotation Error Graph

4.4 Boom Occluded Approach

A full scale, perspective correct refueling boom is placed over the straight in approach for this experiment. The receiver model is only a 1/7th scale, so behind the full scale boom, it is mostly occluded (Fig. **??**). During this phase of the approach, the overall error is high, but once the receiver's wings become more visible, the translation error stays below 10cm. This is the only approach where the virtual environment outperformed the real one, providing an upper bound for experiments done in the real world.

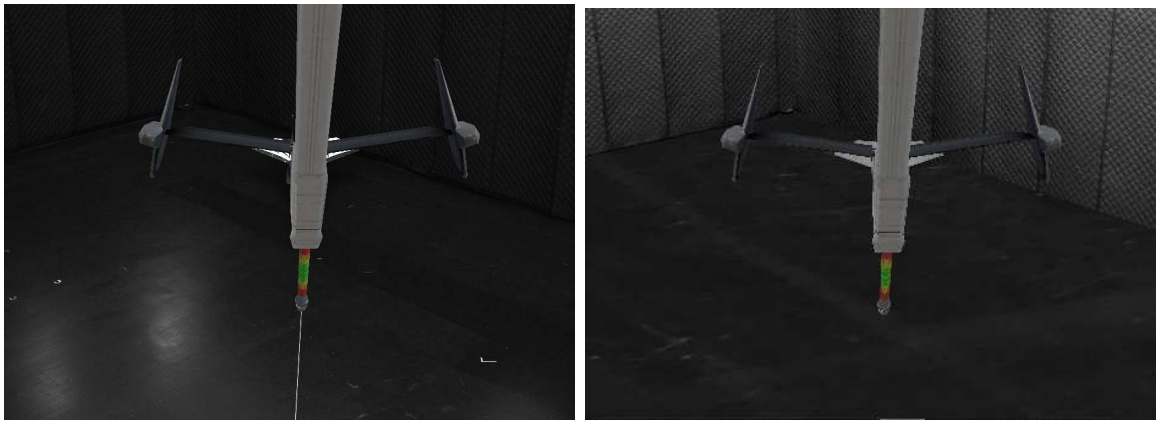

Figure 43: Left: Boom Occluded Receiver at 19 Meters Right: Boom Occluded Virtual Receiver at 19 Meters

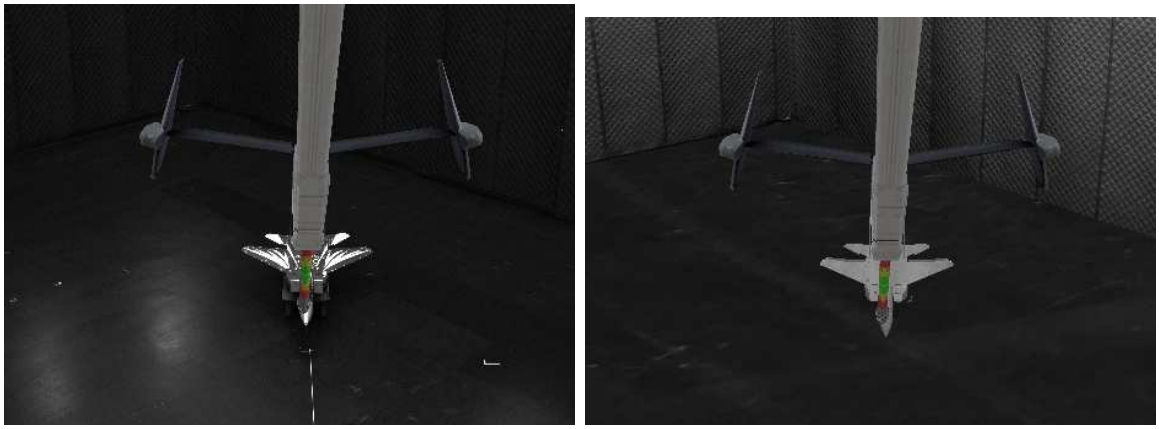

Figure 44: Left: Boom Occluded Receiver at 16 Meters Right: Boom Occluded Virtual Receiver at 16 Meters

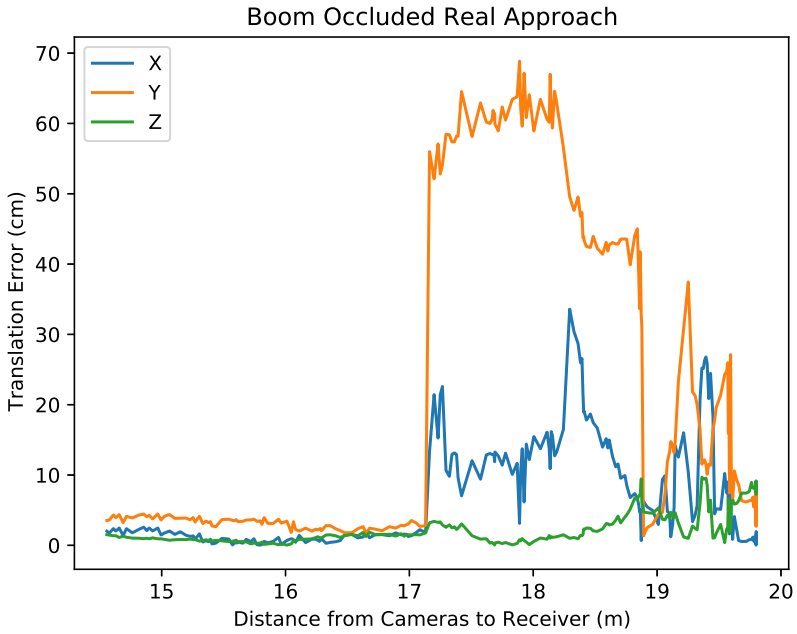

Figure 45: Boom Occluded Real Approach Translation Error Graph

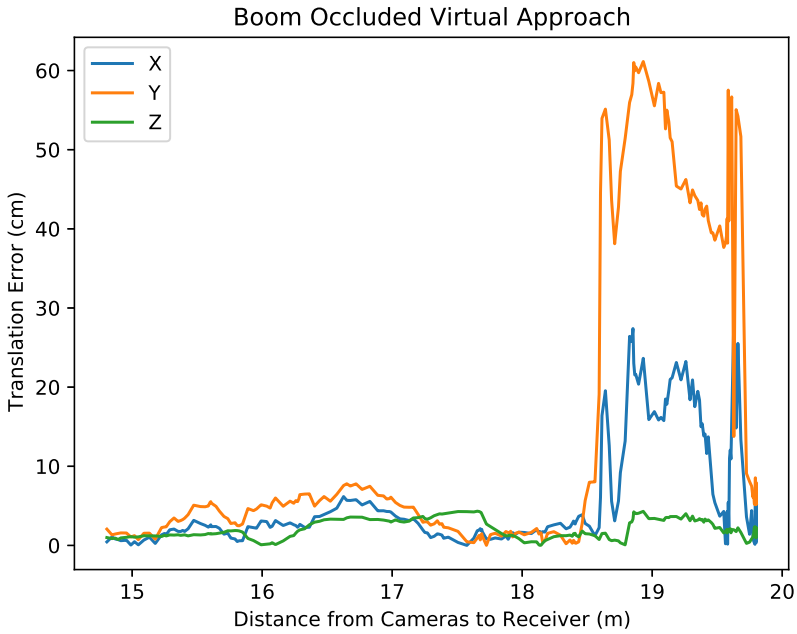

Figure 46: Boom Occluded Virtual Approach Translation Error Graph

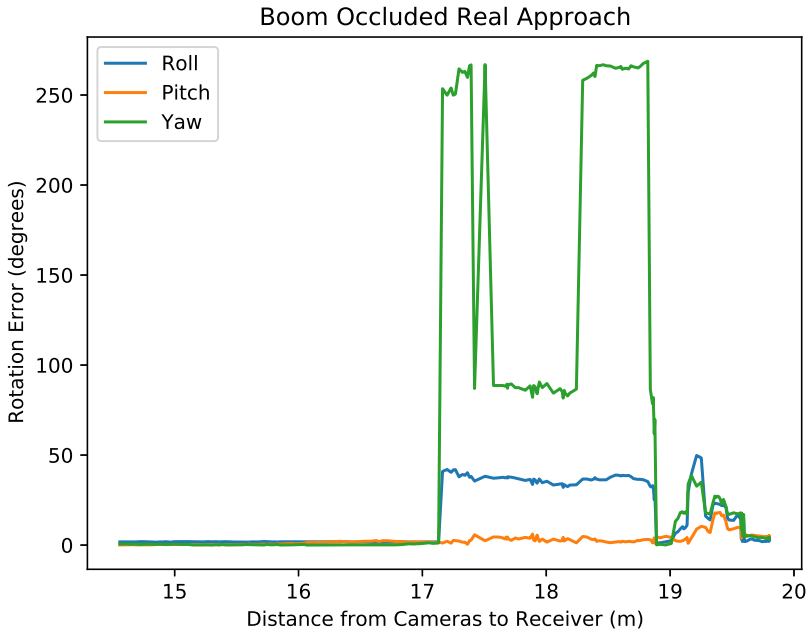

Figure 47: Boom Occluded Real Approach Rotation Error Graph

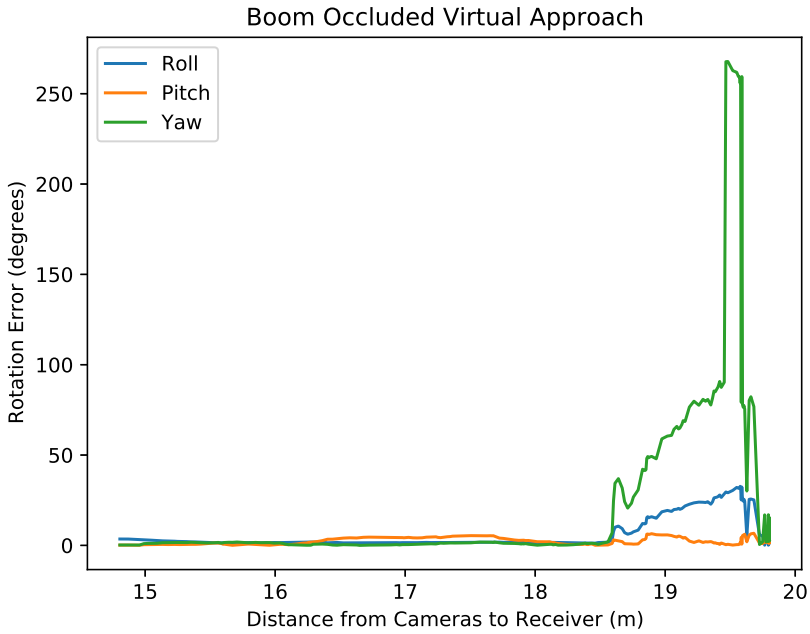

Figure 48: Boom Occluded Virtual Approach Rotation Error Graph

4.5 Diagonal Approach

This approach, similar to the zig zag approach, is designed to test the robustness of the current AAR algorithms. Unlike the zig zag approach, there is no turning involved here. The receiver is dragged from the left side of the frame across the the right side of the frame, while the cameras are primarily viewing the side of the aircraft. The real results demonstrate acceptable levels of accuracy, less than 7cm of transnational error throughout. The only similarity between real and virtual environments in this case is that the z component is the most stable.

4.5.1 Translation Graphs

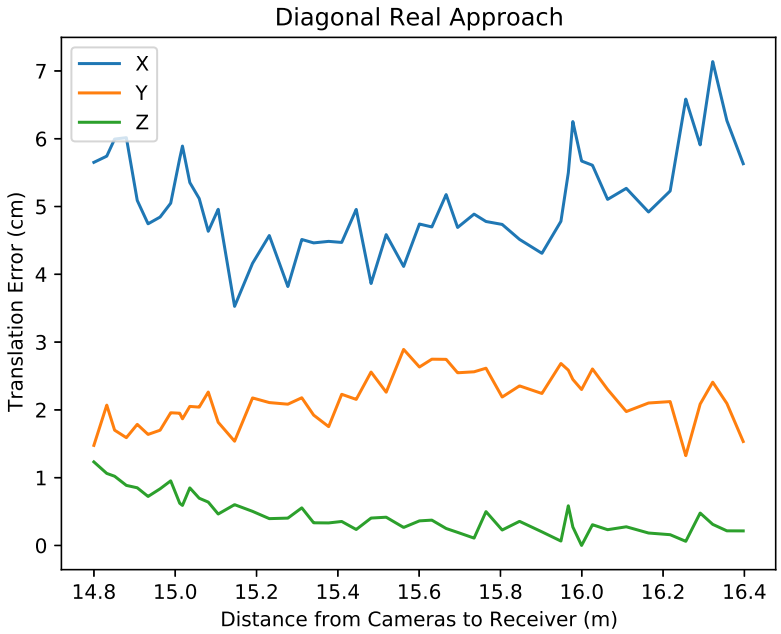

Figure 49: Diagonal Real Approach Translation Error Graph

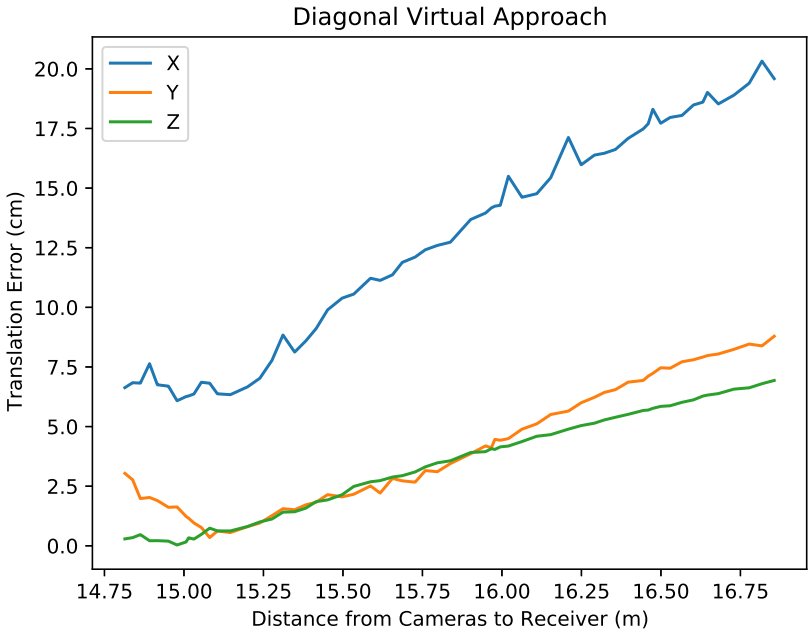

Figure 50: Diagonal Virtual Approach Translation Error Graph

4.5.2 Rotation Graphs

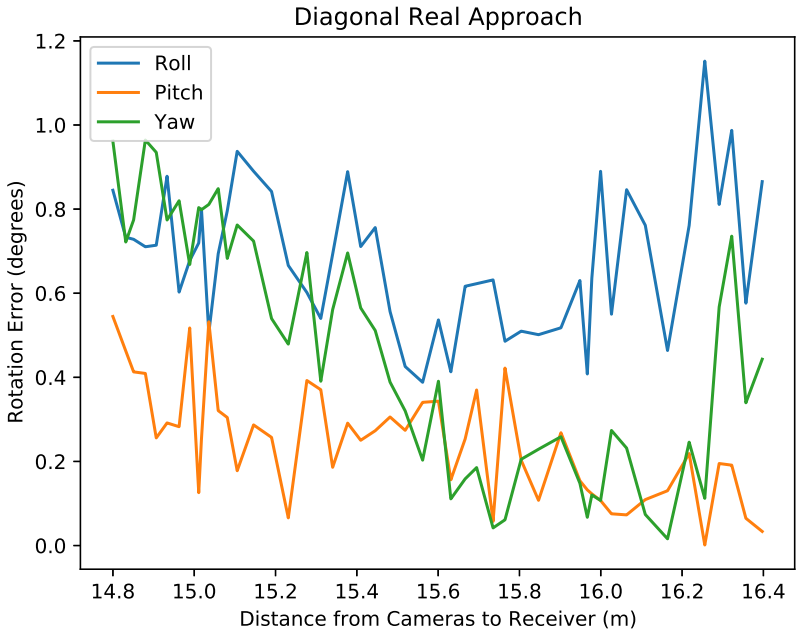

Figure 51: Diagonal Real Approach Rotation Error Graph

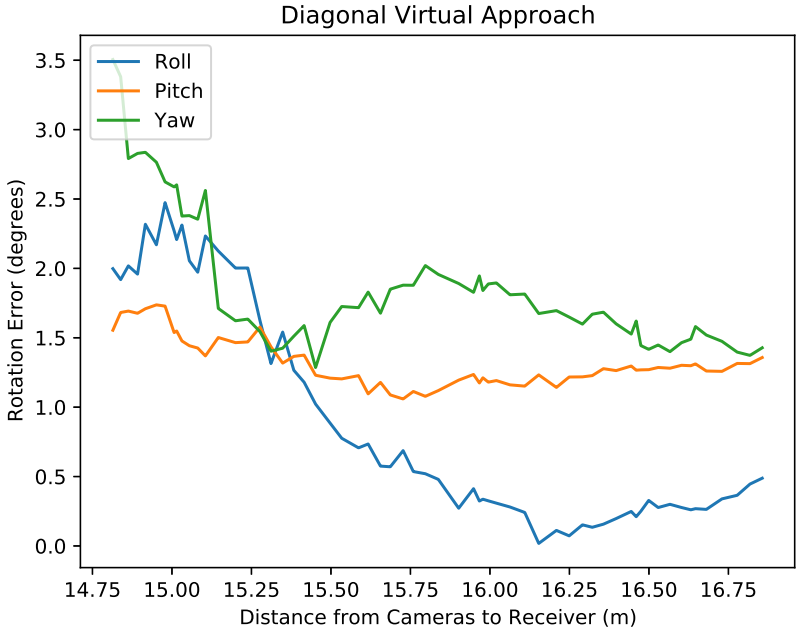

Figure 52: Diagonal Virtual Approach Rotation Error Graph

4.6 Max Distance Approach

It is assumed that the refueling point is at approximately 20 meters from the cameras, but all approaches so far have started at 19 meters or closer. This approach is designed to simulate the aircraft being just outside the contact point at 22 meters and moving forward to refuel at 20 meters. The difference between this approach and the straight in approach is that, in this case, the aircraft is closer to the left side of the image frame, since the farthest corner in the lab was near the left edge of the cameras' viewing frustum. Once again, it is observed that the z component is the most stable in both the real and virtual experiments. The spike in error at 22 meters is due to the aircraft being too close to the wall to filter out points on it, which pulls the truth model toward the rear wall in the lab.

4.6.1 Translation Graphs

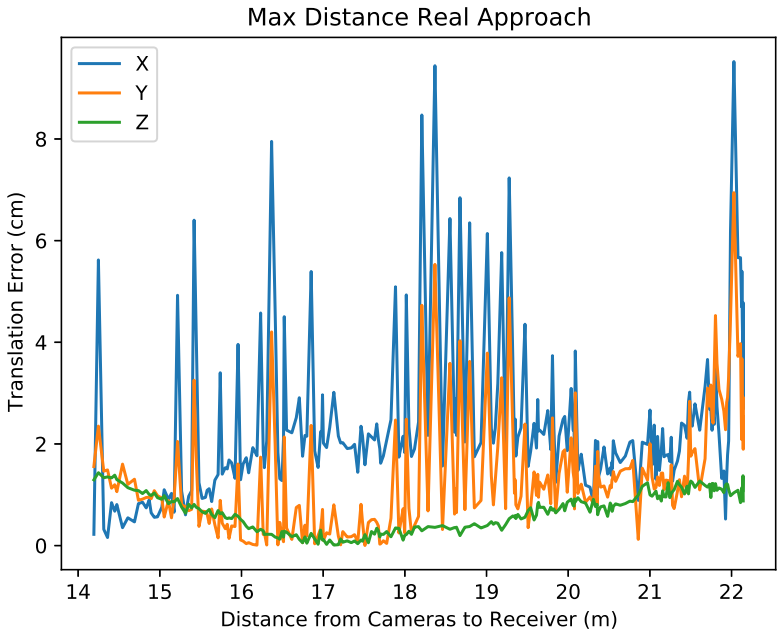

Figure 53: Max Distance Real Approach Translation Error Graph

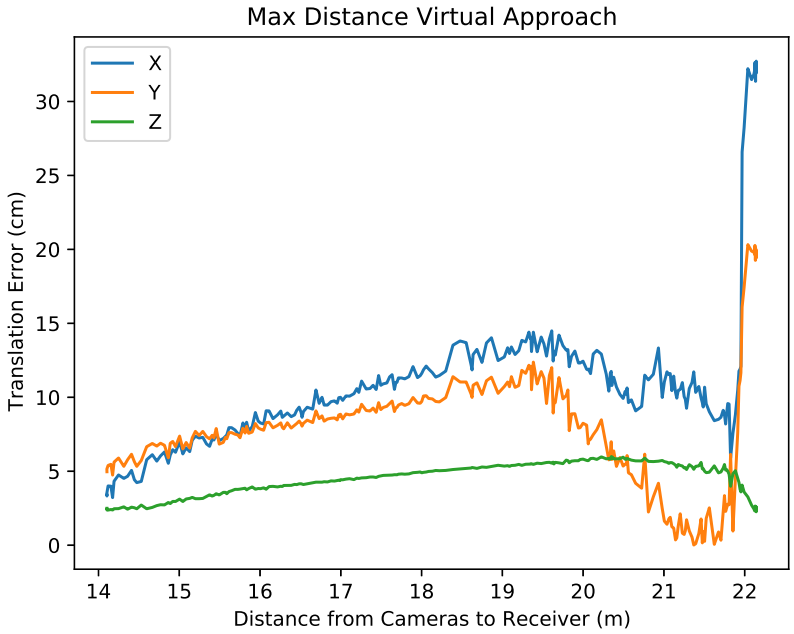

Figure 54: Max Distance Virtual Approach Translation Error Graph

4.6.2 Rotation Graphs

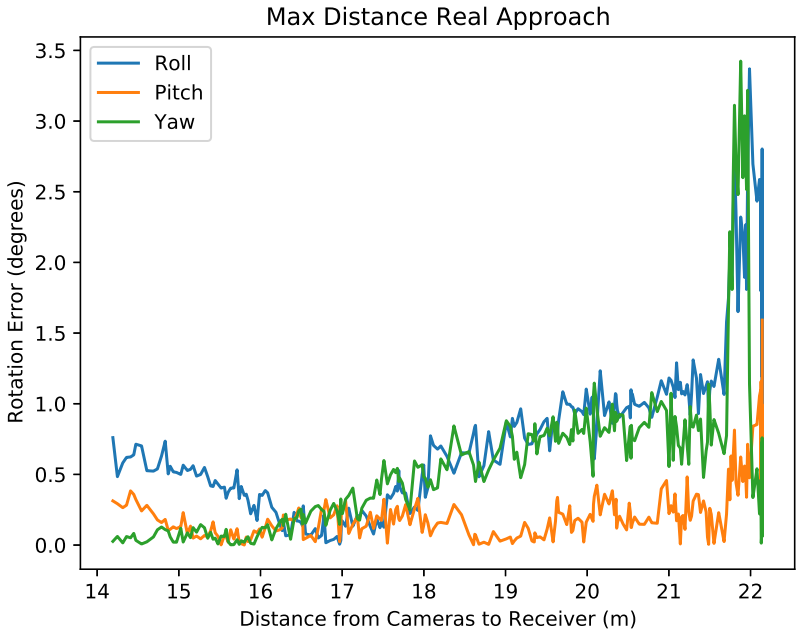

Figure 55: Max Distance Real Approach Rotation Error Graph

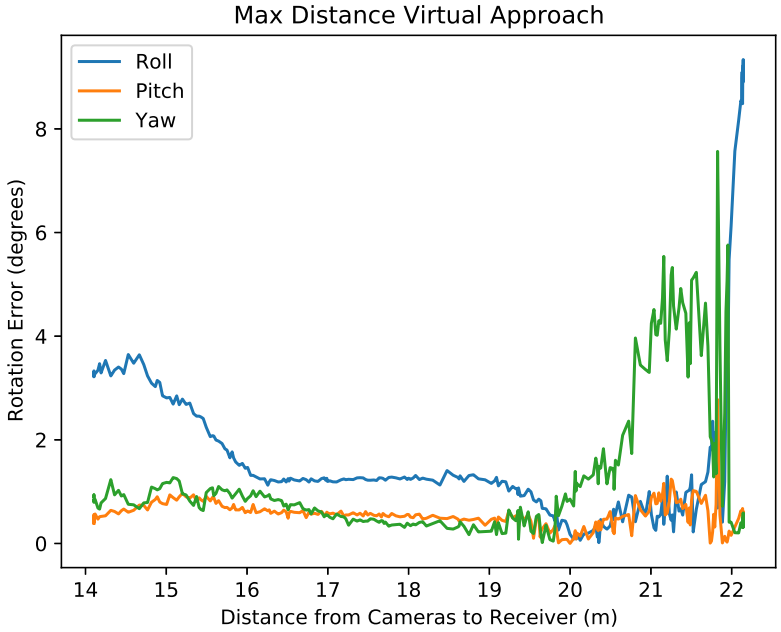

Figure 56: Max Distance Virtual Approach Rotation Error Graph

V. Conclusions

In this paper we have analyzed the differences between the real and virtual AAR testing enviroments, the accuracy of AAR techniques on real images, and the effects of unmitigated boom occlusion on both real and virtual images. This was made possible by the testing framework designed for this thesis, which allows indentical experiments to be performed in both the real and virtual enviroments. The testing framework demonstrated acceptable levels of robustness and can, therefore, be used by future researchers to perform experiments in the real world with accurate, reliable truth data. This will be critical in developing new boom occlusion mitigation techniques for use outside of the simulation.

The comparisons between real and virtual experiments show some similar error trends; however, the real environment is likely a better indication of what to expect during real flight testing. The virtual enviroment does act as an oracle in some cases and indicates where the current AAR techniques will fail.

The straight in approach without boom occlusion proves that AAR can be done on real images, in real time. The 1/7th scale receiver model begins at a distance of approxmiately 20m from the cameras, the refueling point, and shows error acceptable to perform AAR. A full scale aircraft at this distance would be an easier problem, as it would take up a larger portion of the image frame.

5.1 Future Work

The next steps that build upon this work are as follows:

- Boom occlusion mitigation will be the most important step moving forward. Boom occlusion mitigation has been done only in the virtual environment using

information that would not be available in the real world. New techniques for overcoming the occlusion on real images will need to be researched.

- Incorporate faster ICP algorithm techniques to speed up that stage of the pipeline. Work on that topic has been done concurrently with this paper and shows promising results. Now, it needs only to be integrated into this framework.

- Using 4k images slows down the process of generating disparity maps. An AI was designed by a previous AFIT student [27] which automatically sets a region of interest around the receiver aircraft. This will need to be incorporated going forward to ensure real time AAR is possible.

Appendix A. Gauss-Newton Optimization Code

https://git.nykl.net/aar/ViconAugRel.git

```
import numpy as np
import numpy.linalg as la
from scipy.linalg import expm
from scipy.linalg import lstsq
import GNSupportingFunctions as gnsf
import cv2
import glob

#contains the initial guess for the flat board to checker board transformation
FBtoCBInitialGuessFile = 'C:/Users/Vincent/Desktop/Corrected_Variables.npz'
#contains the precomputed corner locations
CornerFile = 'C:/Users/Vincent/Desktop/calibration_in5.npy'
#a csv file generated by the AftrBurner engine from a vicon log file
#contains the pose of the checkerboard for each dat point and
#associates it with the corresponding image
viconPoseCSV = 'C:/Users/Vincent/Desktop/24ftGaussNewtonOptimization.csv'

est_img_pts = np.array([])
img_pts = np.array([])

# Hard coded pose for the camera in the vicon coord frame
#converts from camera to vicon but we need other way around
v_R_cam = np.array([ [-0.725, -0.685, -0.076] ,
                     [0.683, -0.728,  0.059] ,
                     [-0.095, -0.009,  0.995]  ])



Rot25Deg = np.array([  [0.9063078,  0.0000000,  -0.4226183],
   [0.0000000,  1.0000000,  0.0000000],
  [0.4226183,  0.0000000,  0.9063078] ])
#rotate this by 25 degrees about the y since we know that is approximately
#how much the cameras are angled downward. This would change if the cameras are
#repositioned
v_R_cam = Rot25Deg @ v_R_cam

#convert meters to millimeters
cam_T_v = np.array([ 7.729, -6.581, 7.315]) * 1000

#get the inverse of the rotation matrix
cam_R_v = np.transpose(v_R_cam)

# then rotate because the camera in the lab has x coming out of the lens
# but we need z coming out so it can be mupltiped by K to give us the image coord
YawRot = gnsf.createRot(np.array([-90,0,0]),True)
PitchRot = gnsf.createRot(np.array([0,0,90]),True)
cam_R_v = YawRot @ PitchRot @ cam_R_v

# this loads our initial guess for the transformation from flatboard coords to checkerboard object
    coords
```

```
variables = np.load(FBtoCBInitialGuessFile)
cb_T_fb = variables['cb_T_fb']
cb_R_fb = np.array([[1,0,0],[0,1,0.],[0,0,1.]])

# load in image corner points. Easier this was than trying to find them everytime
openCVfindCornersStuff = np.load(CornerFile, allow_pickle = True)
badImageIndexArr = []
for img in range(len(openCVfindCornersStuff)):
    #corner finding AI gives corners starting from bottom left then going up
    #this codes needs them to start at the top left and go across. This swaps them
    for corner in gnsf.swapCorners(openCVfindCornersStuff[img][4]):
        img_pts = np.append(img_pts, corner)

img_pts = img_pts.reshape(len(img_pts), 1)

# Create checkerboard corners in the checkerboards coordinate system
# If a different checkerboard is being used than the 12x9 board with 60mm sqaures
# this will need to be changed
objp = gnsf.create_corners((11,8), 60)
# Read in Vicon information from corresponding CSV files
# this loads in vicon pose data which gives us the transformation from checkerboard object coords to vicon coords
# (we need to take the transpose of the rotation matrix though.. this is done in Big_Sim_functions.py, project_in_image() )
v_T_cb, v_R_cb, imagePaths = gnsf.grab_Rotations_and_Translations_and_imagePaths(viconPoseCSV)

# Hard coded instrinsic calibration matrix for the left EO camera in the lab
K = np.array( [ [ 8.4078562532560827e+03, 0., 2.0452493168634646e+03], [0.,
       8.4239635709810391e+03, 1.5026734947879536e+03], [0., 0., 1.] ] )

# ----------------Find estimated image points over a whole set of images----------------

# Find the estimated image locations based on vicon pose data and initial guesses
print("Calculating estimated image points\n")
for i in range(len(v_R_cb)):
    pts = gnsf.project_in_image(K, cam_R_v, v_R_cb[i], cb_R_fb, cb_T_fb, v_T_cb[i], cam_T_v, objp)
    est_img_pts = np.append(est_img_pts, pts)

    #draw images for a sanity check in the beginning
    # if(i==0):
    #     img = cv2.imread("Your image path here")  #the image path will have to correspond to index i
    #     for pt in pts:
    #         img = cv2.circle(img, (int(pt[0]),int(pt[1])), radius=3, color=(0, 165, 255), thickness=6)
    #     scale_percent = 30
    #     #calculate the 30 percent of original dimensions
    #     width = int(img.shape[1] * scale_percent / 100)
    #     height = int(img.shape[0] * scale_percent / 100)
    #     # dsize
    #     dsize = (width, height)
    #     out = cv2.resize(img, dsize)
    #     cv2.imshow('img', out)
    #     cv2.waitKey(0)
```

```
100     #     cv2.destroyAllWindows()
101     # elif(i==651):
102     #     img = cv2.imread("Your image path here")  #the image path will have to correspond to
         index i
103     #     for pt in pts:
104     #         img = cv2.circle(img, (int(pt[0]),int(pt[1])), radius=3, color=(0, 165, 255),
         thickness=6)
105     #     scale_percent = 30
106     #     #calculate the 30 percent of original dimensions
107     #     width = int(img.shape[1] * scale_percent / 100)
108     #     height = int(img.shape[0] * scale_percent / 100)
109     #     # dsize
110     #     dsize = (width, height)
111     #     out = cv2.resize(img, dsize)
112     #     cv2.imshow('img', out)
113     #     cv2.waitKey(0)
114     #     cv2.destroyAllWindows()
115
116 est_img_pts = est_img_pts.reshape(len(est_img_pts), 1)
117
118 # ------Beginning of Gauss Newton------
119 iter_count = 0
120 iterate = True
121 num_of_iterations = 10
122
123 while iterate is True:
124     print("Beginning iteration " + str(iter_count))
125     Y = np.array([])
126     J = np.array([])
127     check_values = np.array([])
128     print("Calculating the difference Y between estimated image points and actual image points\n")
129     Y = np.subtract(img_pts, est_img_pts)
130     print('At the beginning of iteration, mag Y^2 is ',np.linalg.norm(Y)**2)
131     # ------Derivatives for a set of images------
132     print("\nCalculating Jacobian matrix\n")
133     for i in range(len(v_T_cb)):
134         for j in range(len(objp)):
135             deriv = gnsf.find_Deriv(K, cam_R_v, v_R_cb[i], cb_R_fb, objp[j], cb_T_fb, v_T_cb[i],
         cam_T_v)
136             J = np.append(J, deriv)
137     J = J.reshape(len(img_pts), 12)
138     #J[:,:3]=np.zeros((len(img_pts),3))
139     #J[:,6:9]=np.zeros((len(img_pts),3))
140     db = la.lstsq(J,Y, rcond=None)[0]
141     # ------Begining of scaled Gauss Newton------
142     delta_y_predicted = np.dot(J, db)
143
144     error = 5
145     k = 1
146     while((error < 0.25 or error > 4) and k < 10000):
147
148         cb_T_fb_prop  = cb_T_fb.copy()
149         cb_T_fb_prop[0] += db[0]/k
150         cb_T_fb_prop[1] += db[1]/k
```

```
151         cb_T_fb_prop[2] += db[2]/k
152 
153         cb_R_fb_prop = cb_R_fb.copy()
154         cb_dr1 = db[3]/k
155         cb_dr2 = db[4]/k
156         cb_dr3 = db[5]/k
157         cb_skew = expm(np.array([[0., -cb_dr3, cb_dr2],
158                                  [cb_dr3, 0., -cb_dr1],
159                                  [-cb_dr2, cb_dr1, 0.]]))
160         cb_R_fb_prop = np.dot(cb_skew, cb_R_fb_prop )
161 
162         cam_T_v_prop = cam_T_v.copy()
163         cam_T_v_prop[0] += db[6]/k
164         cam_T_v_prop[1] += db[7]/k
165         cam_T_v_prop[2] += db[8]/k
166 
167         cam_R_v_prop = cam_R_v.copy()
168         cam_dr1 = db[9]/k
169         cam_dr2 = db[10]/k
170         cam_dr3 = db[11]/k
171         cam_skew = expm(np.array([[0., -cam_dr3, cam_dr2],
172                                  [cam_dr3, 0., -cam_dr1],
173                                  [-cam_dr2, cam_dr1, 0.]]))
174         cam_R_v_prop = np.dot(cam_skew, cam_R_v_prop )
175 
176         Y_prop = np.array([])
177         est_img_pts_prop = np.array([])
178 
179         # Find the estimated image locations again based on new pose values
180         for i in range(len(v_R_cb)):
181             pts_prop = gnsf.project_in_image(K, cam_R_v_prop, v_R_cb[i], cb_R_fb_prop,
    cb_T_fb_prop, v_T_cb[i], cam_T_v_prop, objp)
182             est_img_pts_prop = np.append(est_img_pts_prop, pts_prop)
183 
184         est_img_pts_prop = est_img_pts_prop.reshape(len(est_img_pts_prop), 1)
185 
186         Y_prop = np.subtract(img_pts, est_img_pts_prop)
187 
188         predicted_mag = np.linalg.norm(Y)**2 - np.linalg.norm(np.subtract(Y,delta_y_predicted/k))
    **2
189         real_mag = np.linalg.norm(Y)**2 - np.linalg.norm(Y_prop)**2
190         print ('At the end of iteration Y_prop mag squared is',np.linalg.norm(Y_prop)**2)
191         error = predicted_mag/real_mag
192         k *= 2.0
193     # ------End of scaled Gauss Newton------
194 
195     est_img_pts = est_img_pts_prop
196     cam_T_v = cam_T_v_prop
197     cam_R_v = cam_R_v_prop
198     cb_T_fb = cb_T_fb_prop
199     cb_R_fb = cb_R_fb_prop
200 
201 
202     print("Finishing iteration " + str(iter_count) + "\n")
```

```
    iter_count += 1
    iterate = iter_count < num_of_iterations

# ------End of Gauss Newton------

print("cam_T_v\n", cam_T_v)
print("cam_R_v\n", cam_R_v)
print("cb_T_fb\n", cb_T_fb)
print("cb_R_fb\n", cb_R_fb)

YawRot2 = gnsf.createRot(np.array([90,0,0]),True)
PitchRot2 = gnsf.createRot(np.array([0,0,-90]),True)
cam_R_v2 = PitchRot2 @ YawRot2 @ cam_R_v

final= gnsf.get_Transformation_inverses_from_single_transformation(cam_R_v2, cam_T_v)
print(final)


# #visual check
# pts2 = gnsf.project_in_image(K, cam_R_v, v_R_cb[0], cb_R_fb, cb_T_fb, v_T_cb[0], cam_T_v, objp)
# img2 = cv2.imread("Your image path here")  #the image path will have to correspond to index used in v_T_cb[0] and v_R_cb[0] above
# for pt in pts2:
#     img2 = cv2.circle(img2, (int(pt[0]),int(pt[1])), radius=3, color=(0, 165, 255), thickness=6)
# scale_percent = 30
# #calculate the 30 percent of original dimensions
# width = int(img2.shape[1] * scale_percent / 100)
# height = int(img2.shape[0] * scale_percent / 100)
# # dsize
# dsize = (width, height)
# out2 = cv2.resize(img2, dsize)
# cv2.imshow('img2', out2)
# cv2.waitKey(0)

# #visual check
# pts3 = gnsf.project_in_image(K, cam_R_v, v_R_cb[651], cb_R_fb, cb_T_fb, v_T_cb[651], cam_T_v, objp)
# img3 = cv2.imread("Your image path here")  #the image path will have to correspond to index used in v_T_cb[651] and v_R_cb[651] above
# for pt in pts3:
#     img3 = cv2.circle(img3, (int(pt[0]),int(pt[1])), radius=3, color=(0, 165, 255), thickness=6)
# # dsize
# dsize = (width, height)
# out3 = cv2.resize(img3, dsize)
# cv2.imshow('img3', out3)
# cv2.waitKey(0)
```

Listing A.1: Gauss-Newton Optimization

```
import math as m
import numpy as np
import pandas as pd

def img_deriv(x, y, z, deriv):
```

```
    du = (deriv[0]/z) - (x/z**2)*deriv[2]
    dv = (deriv[1]/z) - (y/z**2)*deriv[2]

    return du, dv

def find_Deriv(K, cam_R_v, v_R_cb, cb_R_fb, pt, cb_T_fb, v_T_cb, cam_T_v, simulation=False):

    v_R_cb = quaternian_to_rotation(v_R_cb)

    #flatboard point in checkerboard object coord frame
    pointChecker = cb_R_fb @  pt + cb_T_fb

    #flatboard point in vicon coord frame
    pointVicon = v_R_cb.T @  pointChecker + v_T_cb - cam_T_v

    #flatboard point in camera coord frame
    pointCamera = cam_R_v @ pointVicon

    #flatboard point in image frame
    pointImage = K @ pointCamera

    x = pointImage[0]
    y = pointImage[1]
    z = pointImage[2]

    ss = []
    ss.append( np.array([[0., 0., 0.],
                         [0., 0., -1.],
                         [0., 1., 0.]] ) )

    ss.append( np.array([[0., 0., 1.],
                         [0., 0., 0.],
                         [-1., 0., 0.]] ) )

    ss.append( np.array([[0., -1., 0.],
                         [1., 0., 0.],
                         [0., 0., 0.]] ) )

    # Derivatives for cb_R_fb
    A = K @ cam_R_v @ v_R_cb @ ss[0] @ cb_R_fb @ pt
    cb_R_fb_dr1 = (K @ cam_R_v @ v_R_cb.T @ ss[0] @ cb_R_fb @ pt).reshape(3,1)
    cb_R_fb_dr2 = (K @ cam_R_v @ v_R_cb.T @ ss[1] @ cb_R_fb @ pt).reshape(3,1)
    cb_R_fb_dr3 = (K @ cam_R_v @ v_R_cb.T @ ss[2] @ cb_R_fb @ pt).reshape(3,1)

    du_cb_R_fb_dr1, dv_cb_R_fb_dr1 = img_deriv(x, y, z, cb_R_fb_dr1)
    du_cb_R_fb_dr2, dv_cb_R_fb_dr2 = img_deriv(x, y, z, cb_R_fb_dr2)
    du_cb_R_fb_dr3, dv_cb_R_fb_dr3 = img_deriv(x, y, z, cb_R_fb_dr3)

    CB_dr1 = np.array([du_cb_R_fb_dr1, dv_cb_R_fb_dr1]).reshape(2,1)
    CB_dr2 = np.array([du_cb_R_fb_dr2, dv_cb_R_fb_dr2]).reshape(2,1)
    CB_dr3 = np.array([du_cb_R_fb_dr3, dv_cb_R_fb_dr3]).reshape(2,1)

  # Derivatives for cam_R_v
    C = np.dot(K, cam_R_v)
```

```
    cam_R_rig_dr1 = (K @ ss[0] @ cam_R_v @ (v_R_cb.T @ (cb_R_fb @ pt + cb_T_fb) + v_T_cb - cam_T_v)).reshape(3,1)
    cam_R_rig_dr2 = (K @ ss[1] @ cam_R_v @ (v_R_cb.T @ (cb_R_fb @ pt + cb_T_fb) + v_T_cb - cam_T_v)).reshape(3,1)
    cam_R_rig_dr3 = (K @ ss[2] @ cam_R_v @ (v_R_cb.T @ (cb_R_fb @ pt + cb_T_fb) + v_T_cb - cam_T_v)).reshape(3,1)

    du_cam_R_rig_r1, dv_cam_R_rig_r1 = img_deriv(x, y, z, cam_R_rig_dr1)
    du_cam_R_rig_r2, dv_cam_R_rig_r2 = img_deriv(x, y, z, cam_R_rig_dr2)
    du_cam_R_rig_r3, dv_cam_R_rig_r3 = img_deriv(x, y, z, cam_R_rig_dr3)

    cam_dr1 = np.array([du_cam_R_rig_r1, dv_cam_R_rig_r1]).reshape(2,1)
    cam_dr2 = np.array([du_cam_R_rig_r2, dv_cam_R_rig_r2]).reshape(2,1)
    cam_dr3 = np.array([du_cam_R_rig_r3, dv_cam_R_rig_r3]).reshape(2,1)

    # Derivatives for cb_T_fb
    dx_cb_T_fb = (K @ cam_R_v @ v_R_cb.T @ np.array([1.,0,0]).reshape(3,1)).reshape(3,1)
    dy_cb_T_fb = (K @ cam_R_v @ v_R_cb.T @ np.array([0.,1,0]).reshape(3,1)).reshape(3,1)
    dz_cb_T_fb = (K @ cam_R_v @ v_R_cb.T @ np.array([0.,0,1]).reshape(3,1)).reshape(3,1)

    du_cb_T_fb_tx, dv_cb_T_fb_tx = img_deriv(x, y, z, dx_cb_T_fb)
    du_cb_T_fb_ty, dv_cb_T_fb_ty = img_deriv(x, y, z, dy_cb_T_fb)
    du_cb_T_fb_tz, dv_cb_T_fb_tz = img_deriv(x, y, z, dz_cb_T_fb)

    CB_tx = np.array([du_cb_T_fb_tx, dv_cb_T_fb_tx]).reshape(2,1)
    CB_ty = np.array([du_cb_T_fb_ty, dv_cb_T_fb_ty]).reshape(2,1)
    CB_tz = np.array([du_cb_T_fb_tz, dv_cb_T_fb_tz]).reshape(2,1)

    # Derivatives for cam_T_v
    K @ cam_R_v @ np.array([-1.,0,0]).reshape(3,1)

    dx_cam_T_rig = (K @ cam_R_v @ np.array([-1.,0,0]).reshape(3,1)).reshape(3,1)
    dy_cam_T_rig = (K @ cam_R_v @ np.array([0,-1.,0]).reshape(3,1)).reshape(3,1)
    dz_cam_T_rig = (K @ cam_R_v @ np.array([0,0,-1.]).reshape(3,1)).reshape(3,1)

    du_cam_T_rig_tx, dv_cam_T_rig_tx = img_deriv(x,y,z, dx_cam_T_rig)
    du_cam_T_rig_ty, dv_cam_T_rig_ty = img_deriv(x,y,z, dy_cam_T_rig)
    du_cam_T_rig_tz, dv_cam_T_rig_tz = img_deriv(x,y,z, dz_cam_T_rig)

    cam_tx = np.array([du_cam_T_rig_tx, dv_cam_T_rig_tx]).reshape(2,1)
    cam_ty = np.array([du_cam_T_rig_ty, dv_cam_T_rig_ty]).reshape(2,1)
    cam_tz = np.array([du_cam_T_rig_tz, dv_cam_T_rig_tz]).reshape(2,1)

    deriv = np.concatenate((CB_tx, CB_ty, CB_tz, CB_dr1, CB_dr2, CB_dr3, cam_tx, cam_ty, cam_tz, cam_dr1, cam_dr2, cam_dr3), axis=1)

    return deriv

def project_in_image(K, cam_R_v, v_R_cb, cb_R_fb, cb_T_fb, v_T_cb, cam_T_v, pts):
    img_pts = []
    v_R_cb = quaternian_to_rotation(v_R_cb)

    for i in range(len(pts)):

```

```
110         #flatboard point
111         point = pts[i]
112 
113         #flatboard point in checkerboard object coord frame
114         pointChecker = cb_R_fb @  point + cb_T_fb
115 
116         #flatboard point in vicon coord frame
117         pointVicon = v_R_cb.T @  pointChecker + v_T_cb - cam_T_v
118 
119         #flatboard point in camera coord frame
120         pointCamera = cam_R_v @ pointVicon
121 
122         #flatboard point in image frame
123         pointImage = K @ pointCamera
124 
125         UVarray = np.array([])
126         UVarray = np.append(UVarray, pointImage[0]/pointImage[2])
127         UVarray = np.append(UVarray, pointImage[1]/pointImage[2])
128         img_pts.append(UVarray)
129 
130     return img_pts
131 
132 def createRot (RPY, degrees=False):
133     if degrees:
134         i_yaw = m.radians(RPY[2])
135         i_pitch = m.radians(RPY[1])
136         i_roll = m.radians(RPY[0])
137     else:
138         i_yaw = RPY[2]
139         i_pitch = RPY[1]
140         i_roll = RPY[0]
141 
142     R_yaw = np.array([[m.cos(i_yaw), 0., -m.sin(i_yaw)],[0., 1., 0.],[m.sin(i_yaw), 0., m.cos(
        i_yaw)]])
143     R_pitch = np.array([[1., 0., 0.],[0., m.cos(i_pitch), m.sin(i_pitch)],[0., -m.sin(i_pitch), m.
        cos(i_pitch)]])
144     R_roll = np.array([[m.cos(i_roll), m.sin(i_roll), 0.],[-m.sin(i_roll), m.cos(i_roll), 0.],[0.,
         0., 1.]])
145     return np.dot(R_roll, np.dot(R_pitch, R_yaw))
146 
147 #corner finding AI gives corners starting from bottom left then going up
148 #this codes needs them to start at the top left and go across. This swaps them
149 def swapCorners(corners):
150     newCorners = corners.reshape(11,8,2)
151     swappedCorners = np.array([])
152     for x in range(7,-1, -1):
153         for y in range(len(newCorners)):
154             swappedCorners = np.append(swappedCorners, newCorners[y][x])
155     return swappedCorners
156 
157 # Creates object points in a checkerboards coordinate system
158 # Dimensions is either a tuple or numpy array of two elements
159 # Element one is the number of rows and element two is the number of columns
160 # Size is the length, in mm, of a checkerboard square's side
```

```
def create_corners(dimensions, size):
    corners = np.zeros((dimensions[0]*dimensions[1], 3), np.float32)
    corners[:, :2] = np.mgrid[0:dimensions[0], 0:dimensions[1]].T.reshape(-1, 2) * size
    corners[:, 1] = -corners[:, 1]

    return corners

def grab_Rotations_and_Translations_and_imagePaths(path):
    translation = np.array([])
    rotation = np.array([])

    obj = pd.read_csv(path)

    tran_x = obj["x"]
    tran_y = obj["y"]
    tran_z = obj["z"]
    rot_w = obj["quatW"]
    rot_x = obj["quatX"]
    rot_y = obj["quatY"]
    rot_z = obj["quatZ"]
    imagePaths = obj["imagePath"]

    for i in range(len(tran_x)):
        tmp = np.array([tran_x[i], tran_y[i], tran_z[i]])
        translation = np.concatenate((translation, tmp), axis=0)
        tmp2 = np.array([rot_w[i], rot_x[i], rot_y[i], rot_z[i]])
        rotation = np.concatenate((rotation, tmp2), axis=0)

    translation = translation.reshape(len(tran_x), 3) * 1000
    rotation = rotation.reshape(len(tran_x), 4)

    return translation, rotation, imagePaths

#wxyz
def quaternian_to_rotation(rotation):
    R = np.array([[1-2*(rotation[2]**2 + rotation[3]**2), 2*(rotation[1]*rotation[2] - rotation
     [3]*rotation[0]), 2*(rotation[0]*rotation[2] + rotation[1]*rotation[3])],
                  [2*(rotation[1]*rotation[2] + rotation[0]*rotation[3]), 1-2*(rotation[1]**2 +
     rotation[3]**2), 2*(rotation[2]*rotation[3] - rotation[0]*rotation[1])],
                  [2*(rotation[1]*rotation[3] - rotation[0]*rotation[2]), 2*(rotation[0]*rotation[1]
      + rotation[2]*rotation[3]), 1-2*(rotation[1]**2 + rotation[2]**2)]])

    return R
```

Listing A.2: Gauss-Newton Optimization Supporting Functions

Bibliography

1. Bradley D Denby. Towards automated aerial refueling: Real time position estimation with stereo vision. Technical report, AIR FORCE INSTITUTE OF TECHNOLOGY WRIGHT-PATTERSON AFB OH WRIGHT-PATTERSON ..., 2016.

2. Nicholas Seydel, Will Dallmann, and Scott Nykl. Visualizing behaviors when using real vs synthetic imagery for computer vision. In *Proceedings of the International Conference on Scientific Computing (CSC)*, pages 3–9. The Steering Committee of The World Congress in Computer Science, Computer ..., 2018.

3. Zachary Paulson, Scott Nykl, John Pecarina, and Brian Woolley. Mitigating the effects of boom occlusion on automated aerial refueling through shadow volumes. *The Journal of Defense Modeling and Simulation*, 16(2):175–189, 2019.

4. Adrian Kaehler and Gary Bradski. *Learning OpenCV 3: computer vision in C++ with the OpenCV library.* " O'Reilly Media, Inc.", 2016.

5. Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.

6. Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision.* Cambridge university press, 2003.

7. Soo Kyun Kim, Shin-Jin Kang, Yoo-Joo Choi, Min-Hyung Choi, and Min Hong. Augmented-reality survey: from concept to application. *KSII Transactions on Internet & Information Systems*, 11(2), 2017.

8. Feng Zhou, Henry Been-Lirn Duh, and Mark Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. In

2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, pages 193–202. IEEE, 2008.

9. Danilo Avola, Luigi Cinque, Gian Luca Foresti, Cristina Mercuri, and Daniele Pannone. A practical framework for the development of augmented reality applications by using aruco markers. In *ICPRAM*, pages 645–654, 2016.

10. Scott Nykl, Chad Mourning, Mitchell Leitch, David Chelberg, Teresa Franklin, and Chang Liu. An overview of the steamie educational game engine. In *2008 38th Annual Frontiers in Education Conference*, pages F3B–21. IEEE, 2008.

11. Chin-Hung Teng and Jr-Yi Chen. An augmented reality environment for learning opengl programming. In *2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing*, pages 996–1001. IEEE, 2012.

12. James D Anderson, Scott Nykl, and Thomas Wischgoll. Augmenting flight imagery from aerial refueling. In *International Symposium on Visual Computing*, pages 154–165. Springer, 2019.

13. Yiheng Feng, Chunhui Yu, Shaobing Xu, Henry X Liu, and Huei Peng. An augmented reality environment for connected and automated vehicle testing and evaluation. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1549–1554. IEEE, 2018.

14. Hassan Abu Alhaija, Siva Karthik Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *International Journal of Computer Vision*, 126(9):961–972, 2018.

15. Ryan Dibley, Michael Allen, and Nassib Nabaa. Autonomous airborne refueling demonstration phase i flight-test results. In *AIAA atmospheric flight mechanics conference and exhibit*, page 6639, 2007.

16. Walton R Williamson, Gregory J Glenn, Vu T Dang, Jason L Speyer, Stephen M Stecko, and John M Takacs. Sensor fusion applied to autonomous aerial refueling. *Journal of Guidance, Control, and Dynamics*, 32(1):262–275, 2009.

17. Samer M Khanafseh and Boris Pervan. Autonomous airborne refueling of unmanned air vehicles using the global positioning system. *Journal of Aircraft*, 44(5):1670–1682, 2007.

18. Marco Mammarella, Giampiero Campa, Marcello R Napolitano, Mario L Fravolini, Yu Gu, and Mario G Perhinschi. Machine vision/gps integration using ekf for the uav aerial refueling problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(6):791–801, 2008.

19. II Curro and A Joseph. Automated aerial refueling position estimation using a scanning lidar. Technical report, AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH GRADUATE SCHOOL OF ..., 2012.

20. Daniel T Johnson, Scott L Nykl, and John F Raquet. Combining stereo vision and inertial navigation for automated aerial refueling. *Journal of Guidance, Control, and Dynamics*, 40(9):2250–2259, 2017.

21. William E Dallmann. Infrared and electro-optical stereo vision for automated aerial refueling. 2019.

22. Yimin Deng, Ning Xian, and Haibin Duan. A binocular vision-based measuring system for uavs autonomous aerial refueling. In *2016 12th IEEE International Conference on Control and Automation (ICCA)*, pages 221–226. IEEE, 2016.

23. Giampiero Campa, Marcello R Napolitano, and Mario L Fravolini. Simulation environment for machine vision based aerial refueling for uavs. *IEEE Transactions on Aerospace and Electronic Systems*, 45(1):138–151, 2009.

24. Richard Burns, Curt Clark, and Ron Ewart. The automated aerial refueling simulation at the avtas laboratory. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, page 6008, 2005.

25. Ba Nguyen and Tong Lin. The use of flight simulation and flight testing in the automated aerial refueling program. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, page 6007, 2005.

26. Erik Rodenburgh and Clark Taylor. A system for evaluating vision-aided navigation uncertainty. In *Proceedings of the 33rd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2020)*, pages 2272–2280, 2020.

27. Andrew T Lee. Object detection with deep learning to accelerate pose estimation for automated aerial refueling. Technical report, AIR FORCE INSTITUTE OF TECHNOLOGY WRIGHT-PATTERSON AFB OH WRIGHT-PATTERSON AFB, 2020.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
25–03–2021	Master's Thesis	Sept 2019 — Mar 2021

4. TITLE AND SUBTITLE

Using Motion Capture and Augmented Reality to Test AAR with Boom Occlusion
to Test AAR with Boom Occlusion

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Vincent J. Bownes

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-21-M-016

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RQQC
Dan Schreiter
WPAFB OH 45433-7765
COMM 937-938-7765
Email: dan.schreiter@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RQQC

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The operational capability of drones is limited by their inability to perform aerial refueling. This can be overcome by automating the process with a computer vision solution. Previous work has demonstrated the feasibility of automated aerial refueling (AAR) in simulation. To progress this technique to the real world, this thesis conducts experiments using real images of a physical aircraft replica and a motion capture system for truth data. It also compares the error between the real and virtual experiments to validate the fidelity of the simulation. Results indicate that the current technique is effective on real images and that the simulation can predict errors in real world experiments.

15. SUBJECT TERMS

Aerial Refueling, Computer Vision, Motion Capture, Augmented Reality

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Scott L. Nykl, AFIT/ENG
U	U	U	UU	89	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x4395 scott.nykl@afit.edu