Image Dependent Relative Formation Navigation for Autonomous Aerial Refueling

THESIS

James Michael Howard, Major, USAF

AFIT/GE/ENG/11-16

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/11-16

Image Dependent Relative Formation Navigation for Autonomous Aerial Refueling

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 Electrical Engineering

James Michael Howard, B.S.E.E., M.A.S., M.S.F.T.E.

Major, USAF

March 2011

AFIT/GE/ENG/11-16

Image Dependent Relative Formation Navigation for Autonomous Aerial Refueling

James Michael Howard, B.S.E.E., M.A.S., M.S.F.T.E.
Major, USAF

Approved:

/signed/	7 Mar 2011
Lt Col M. Veth, PhD (Chairman)	date
/signed/	7 Mar 2011
Dr. J. Raquet (Member)	date
/signed/	7 Mar 2011
Dr. M. Pachter (Member)	date

Abstract

A definite certainty in the future of Unmanned Aircraft Systems is demand. The potential capabilities of these systems far exceed those of manned aircraft. Without a human on-board, aircraft limitations are diminished and endurance potential increased. An endurance limitation that persists is a trait inherent to all aircraft: the balance between performance and fuel capacity. The ability of an unmanned aircraft to refuel in flight will limit the impact of that balance.

Refueling an unmanned aircraft in flight is an engineering challenge that has demanded the better part of a decade. Some successful approaches have used Differential Global Positioning System (DGPS) between aircraft. Optical sensor tracking has shown potential as a viable alternative, or augmentation, to DGPS for refueling unmanned systems.

This research investigates the feasibility, accuracy, and reliability of a predictive rendering and holistic comparison algorithm with the use of an optical sensor to provide relative distance and position behind a lead or tanker aircraft. Using an accurate model of a tanker, an algorithm renders image(s) for comparison with collected images by a camera installed on the receiver aircraft. Based on this comparison, information used to create the rendered image(s) is used to provide the relative navigation solution required for autonomous air refueling.

Building on previous work, this research reduced the number of required rendered images to 15 or less for each collected image while requiring no modification to the tanker aircraft. The accuracy of this research is considered good enough for autonomous operations. The average error was two *feet* or less at distances of 62.5 *feet* and closer. A remaining limitation of this approach is the length of time to calculate a measurement, which can take up to four seconds. Although improvements are warranted, the methods presented are viable for autonomous air refueling.

Acknowledgements

A thank you to my ladies, without you this research would have no laughter, enjoyment, or fulfillment. I can never repay the time away, only enrich the years ahead.

Lt Col Veth and the extended Air Force Institute of Technology staff, thank you for the opportunities, education, and trust. Above all else, I finally learned how to learn.

United States Air Force Test Pilot School, Class 10 Alpha, and the members of the HAVE GAS TMP, thank you for the best year I never want to repeat; to be the first class was amazing.

Extended family and friends, you will never know how much I enjoy our time together and how much I learn from you; thanks for the job!

Mike

Table of Contents

List of Figures

List of Tables

List of Symbols

Symbol		Page
$\mathbf{X}_{frame}$	the defined axis that determines x translations	8
$\mathbf{Y}_{frame}$	the defined axis that determines y translations	8
$\mathbf{Z}_{frame}$	the defined axis that determines z translations	8
x	a defined translation along the $\mathbf{X}$ axis	8
y	a defined translation along the $\mathbf{Y}$ axis	8
z	a defined translation along the $\mathbf{Z}$ axis	8
$\mathbf{I}$	an image	13
$\mathbf{I}_r$	rendered image	13
$\mathbf{I}_c$	collected image	13
$\mathbf{p}^n_{CAM}$	camera position in n-frame	16
$\mathbf{C}^b_n$	body to navigation frame direction cosine matrix	16
ψ	heading angle	18
θ	pitch angle	18
ϕ	roll angle	18
$\mathbf{p}^e_{NAV}$	n-frame origin in e-frame	20
$\mathbf{T}$	a transformation	20
$\mathbf{K}$	camera matrix	22
f	focal length	22
s	skew	30
$\mathbf{I}_t$	template image	49
$\mathbf{I}_m$	matching image	49
$\mathbf{R}$	a matrix of matching values, a result of cvMatchTemplate	49
$\mathbf{r}$	location of the best match value in $\mathbf{R}$	51
σ	standard deviation	56
$\dot{\psi}$	yaw rate	61

Symbol		Page
$\mathbf{p}_{1NM}^{b_L}$	rendezvous position of the wing aircraft	76
$\mathbf{p}_{PRE}^{b_L}$	pre-contact position of the wing aircraft	76
$\mathbf{p}_{CONTACT}^{b_L}$	contact position of the wing aircraft	77
$\mathbf{C}_{GLcam}^{b_L}$	attitude of the lead aircraft with respect to $GLcam$ - frame	83
$\mathbf{p}_L^{GLcam}$	position of Lead in the $GLcam$ - frame	84
$\mathbf{p}_o$	the true position of an object	96
$\mathbf{p}_r$	the rendered, predicted position of an object	96

List of Abbreviations

Abbreviation		Page
GL-frame	OpenGL world frame	33
GLcam	OpenGL camera frame	33
FOV	Field of View	34
NDC-frame	OpenGL normalized device coordinate frame	36
CVV	OpenGL canonical viewing volume	36
GLimage	OpenGL image frame	43
GLimage′	intermediary OpenGL image frame	43
WGN	white, Gaussian, zero-mean noise	56
DOF	degrees of freedom	60
fps	*feet per second*	65
PSD	Power Spectral Density	67
FOGM	first-order Gauss-Markov process	70
NM	*nautical mile*	76
UARRSI	Universal Air Refueling Receptacle Slipway Installation	77
INS	Inertial Navigation System	82
EO	electro-optic	88
EKF	Extended Kalman Filter	89
UKF	Unscented Kalman Filter	91
RIPE	rendered image position and orientation estimation	93
IFOV	instantaneous field of view (one pixel FOV)	99
ROI	region of interest	99
GAINR	GPS Aided Inertial Navigation Reference	134
C2B	configuration 2B	134
GUI	graphical user interface	157

Image Dependent Relative Formation Navigation for Autonomous Aerial Refueling

I. Introduction

The motivation driving this research is to expand the utility of Unmanned Aircraft Systems (UAS) by incorporating passive, vision-based sensors that enable the system to conduct autonomous aerial refueling (AAR). The Department of Defense (DOD), who operates more than 6,800 UAS, has come to rely on their capabilities in everyday operations [26]. The use of a vision-based sensor to AAR is also motivated by the DOD's emphasis on both non-emissive equipment and system redundancy (to other on-board equipment that can enable AAR), ensuring continued operations in combat environments.

Combat commanders use UAS on the battlefield, because they significantly reduce human exposure to risk, they do not require expenditures for life support equipment, and they are not limited to tolerances of the human body.

Removing human physiological needs from on-board the aircraft eliminates the endurance limitations previously imposed by manned flight. The UAS can potentially loiter (remain airborne) indefinitely, significantly reducing the fuel and time it costs to transit from an operating field to a point of interest and back. The fuel carried on the aircraft is currently the limitation of loiter time. A UAS aerial refueling (AR) capability will increase their endurance, a necessity to expanding their utility. Transmission delays, limited control fidelity, and inadequate feedback between the system operator and the system aircraft currently limit safe AR operations with UAS. Accomplishing AR autonomously, using on-board sensors, is a more probable and desirable alternative method to a system-operator, manually-controlled UAS AR.

Currently the Air Force Research Laboratory (AFRL) is researching and advancing the science of AAR. The bulk of that research is aimed at utilizing Differential Global Positioning System (DGPS) to determine the relative position between a refueling platform (tanker) and the UAS. This research is ongoing and testing with the actual transfer of fuel is forthcoming. DGPS is a proven navigational tool and provides adequate precision to AAR, but could be subject to limitations during wartime conditions, such as Global Positioning System (GPS) jamming or spoofing. As a result, AFRL is currently exploring other sensors to augment and provide dissimilar system redundancy, leading to the secondary motivation for this research.

The DOD has a vested interest to make certain its systems have an inherent redundancy to ensure continued operation despite degradation to the system's sensors or equipment. A combination of solutions including DGPS, integrated navigation solutions, and other measurement devices provide the desired redundancy and ensure both the survivability of the UAS and its ability to conduct AAR. Additionally, the DOD places an emphasis on passive sensors (sensors that do not emit any electromagnetic energy) to reduce the likelihood of detection when operating in unfriendly areas of interest. This would hinder information sharing between the aircraft unless transmissions could be limited in scope, duration, or power. Finally, minimizing the number of modifications to the existing tanker fleet, such as targets, lights, or transmission devices, would dramatically reduce the fielding costs as well as long term operational and maintenance costs. A low emissive, cheap alternative, that is dissimilar to DGPS, characterizes a vision-based AAR approach.

The ability of an UAS to achieve AAR using passive on-board sensors will enhance the UAS' ability to operate for longer periods of time independent of external signals (such as GPS or broadcasted information from other airborne systems or platforms). AAR using vision-based capabilities and other position and orientation estimation (pose) equipment is a capability that will aid the DOD.

1.1 Problem Definition

Even with the current knowledge base and computer processing capabilities, it is difficult to attain precise navigation using vision information alone. The information provided from a typical, non-modified picture is limited to two-dimensions. It is possible to determine in what direction a specific reference point is located, but without additional information, range to that point is not immediately available. Mathematically based algorithms can process collected images and infer a relative position of the point. The science of image-aided navigation is ever expanding and the required precision is currently possible with enough time, information, and processing of the image(s).

The critical problem is a method of real-time, accurate pose. Real-time estimation allows incorporation of a navigation solution into an autopilot response. Accurate estimation introduces fewer errors in the solution and ultimately in the autopilot response. Both real-time and accurate estimations are necessary to conduct safe UAS AAR operations.

There are currently two methods employed by the DOD to accomplish AR. The U.S. Air Force preferred method uses a refueling boom attached to the rear of the tanker aircraft. The boom is controlled with flight control surfaces attached to it. To refuel using this method, the receiver pilot flies the aircraft to within a defined refueling envelope surrounding the refueling boom. A boom operator on the tanker aircraft then flies the boom into a receiver port on the receiver aircraft. Both pilot and boom operator are responsible for the refueling connection.

The U.S. Navy prefers the probe and drogue method of air refueling. Instead of a boom, this method uses a drogue: a basket at the end of a fuel hose that extends behind the tanker aircraft. The receiver aircraft has a probe that extends into the wind stream and mates with the drogue. The tanker aircraft crew has no visibility of the location and movements of the receiver aircraft. The pilot on the receiver aircraft is completely responsible for making the refueling connection. The basket on the end

of the drogue allows the pilot some margin for error and guides the probe into the drogue.

Both methods have strengths and weaknesses, and some aircraft have both a receiver port for a boom as well as a probe for a drogue. Heavy aircraft (aircraft certified to take off weighing more than 255,000 *pounds*) do not normally refuel with the probe and drogue method because of their lack of maneuverability. The U.S. Navy does not operate large tanker aircraft, or many large receiver aircraft. Their aircraft are generally too small to accommodate the bulky and heavy equipment required for boom refueling.

Demonstrations and research of AAR with UAS vary between the two methods, and currently some UAS are being built with equipment for both [28]. As of this writing, no tanker has yet to transfer fuel to an unmanned aircraft.

Flying aircraft in close proximity to each other is an inherently dangerous operation. Human operators require considerable training to perform adequate AR formation maneuvers and incidents between a tanker and receiver continue to occur. An AAR capability for an UAS will necessarily be complex to ensure safe operations at all times.

Even when fully implemented, human operators will still be integral to the AAR process, just as they are for all UAS activities. They will have oversight of the refueling and provide safety measures until the reliability of the AAR system process is adequately determined.

To limit the focus of this research, the author has assumed a boom-refueling method and that a receiver aircraft maintaining a position within the envelope with less than five feet of error will be able to perform AAR. This research will not investigate autopilots or controllers to fly a receiver aircraft. Instead, the research will focus on the navigation solution that will eventually allow an autopilot to maneuver the aircraft into the envelope.

At a minimum, it is assumed that at least one camera is on-board the receiver aircraft and it has an unobstructed view of the tanker (minor discrepancies from limited dirt and grime on the lenses and coverings are acceptable). Estimating an initial relative position between the UAS and refueling platform will be possible with the method researched here, but is not a focus area. To demonstrate the AAR capability, the research assumes that an initial, relative position between the aircraft is known. Finally, no computations will be completed on-board the aircraft and all determinations of potential solutions will be post-processed.

1.2 Research Solution

This preliminary background has demonstrated that a need exists for a dissimilar approach to AAR to augment the DGPS approach or to serve as a backup. The method presented in this thesis will use collected images of a tanker as well as computer-rendered simulated images. The camera, installed on the receiver aircraft looking up at the tanker aircraft, collects images throughout the refueling process.

Multiple renderings of a three-dimensional tanker model permit a comparison between a collected image and rendered images. This comparison determines a matching likelihood of each rendered image. The information used to create the most likely image updates a Kalman filter that tracks the position of the tanker. This approach builds on many successful methods [5, 15, 18, 23, 27, 29, 30] that use similar vision-dependent techniques to estimate the position of a vehicle. The method outlined in this thesis has four specific goals to overcome some of the problems discovered in the previous efforts.

The first goal is to decrease the average time required to determine a solution. Solutions presented in the cited research range from hundredths of a second to 30 seconds for each pose determination. A real-time approach during AR would need updates consistently less than one second apart.

Increasing the accuracy of this approach with the use of inertial navigation solutions and Kalman-filtering is the second goal. An accurate solution requires average pose errors of five feet or less [16].

The third goal is implement the solution with an efficient programming language (e.g., C) as quickly as possible, requiring the use of open-source libraries. The use of open-source libraries dramatically reduces the time to develop a solution. The libraries are efficient and provide many tools that would otherwise require extensive development and validating.

A final goal to prove this approach as a viable alternative is to accomplish the three previous goals without the need for any modification to the tanker aircraft. Achieving this last goal will increase the utility of this solution by minimizing the cost required for implementation.

1.3 Thesis Outline

This thesis is broken into six chapters. The first chapter details the background motivating the research, the problem definition, and the general approach taken to solving the problem. Chapter 2 will introduce the mathematical description of key terms and relationships used throughout the paper. Nomenclature and equations will explain some of the basics of navigation, lenses, and cameras. Chapter 2 also discusses the programming language, the open-source libraries, and the Kalman filtering used in the research. Chapter 3 outlines the nature of AR and characterizes it not only for this research but future projects and explains position realization for formation navigation. Chapter 4 introduces the concept of pose, previous work addressing pose, and this research's approach to AAR. The experimental portion of the research is presented in Chapter 5, including laboratory work at the Air Force Institute of Technology (AFIT) and flight research at the United States Air Force Test Pilot School (TPS) at Edwards AFB, CA. This includes setup, collection, and validation of the research's approach and the results of these tests. Finally, Chapter 6 contains conclusions and recommendations for future research.

II. Navigation, Programming, and Mathematical Background

This thesis relies on accurate navigational relationships, programming that creates real-world representations and interpretations, and mathematical relationships. This chapter introduces the background of these concepts in three sections. The first presents a discussion of the appropriately used nomenclature, assumptions, and concepts used in navigation. The second section describes the two C programming libraries (collections of programming resources) used, how they are used, and what information they provide. The final section provides an overview of Kalman filtering.

The thesis uses the following mathematical notation:

- **Scalars:** Italic type (e.g., x) represents scalars.
- **Vectors:** Bold font, lower case letters (e.g., $\mathbf{x}$) represent vectors.
- **Matrices:** Bold font, upper case letters (e.g., $\mathbf{T}$) represent matrices, except for $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ which represent the axes of coordinate frames.
- **Estimated Variables:** The *hat* character (e.g., $\hat{\mathbf{x}}$) denotes an estimate.
- **Computed Variables:** The *tilde* character (e.g., $\tilde{\mathbf{x}}$) denotes computed variables.
- **Homogenous Coordinates:** An *underline* (e.g., $\underline{\mathbf{x}}$) denotes homogeneous coordinates.

2.1 Navigation

This section presents many aspects of navigation and wherever possible, explains them with images and represents them with symbols. This is done to help the reader understand key relationships applicable to other scientific disciplines. Section 2.1.1 presents the fundamentals of position, followed by the reference frames used to define those positions in Section 2.1.2. Section 2.1.3 expands those fundamentals

to conversions and transformations between frames and a discussion on lenses and cameras.

2.1.1 Position. In navigation, an object's location relative to a coordinate system and reference frame determines its position, denoted as column vector $\mathbf{p}$. There are many different reference frames based on their origin and rotation, which are explained in further detail in Section 2.1.2. Each frame has two or three axes, denoted as $\mathbf{X}_{frame}$, $\mathbf{Y}_{frame}$, and as necessary, $\mathbf{Z}_{frame}$. For these three symbols only, a subscript following the axis denotes the *frame* it is associated with. A position's translation, measured along each one of these axes, determines the location of the position referenced in that specific frame. This distance along a coordinate axis is denoted as x, y, and as necessary, z. Together they are the *coordinates* of that position. Positions in this research require coordinates that are annotated in both Cartesian and homogeneous representation.

Cartesian coordinates are unique; the translations along a reference-frame axis are fixed for the instant of time they are referenced. The following notation abbreviates a position referenced with Cartesian coordinates:

$$\mathbf{p}^{frame}_{identifier} = \begin{bmatrix} x & y & z \end{bmatrix}^{\mathbf{T}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} p^{frame}_{x,\,identifier} \\ p^{frame}_{y,\,identifier} \\ p^{frame}_{z,\,identifier} \end{bmatrix} \tag{2.1}$$

where the superscript *frame* denotes the coordinate frame of reference used to define the position, *identifier* denotes the name of the position, and $^{\mathbf{T}}$ denotes a transpose of any array or matrix. When obvious which point is being referenced, or for non-specific points, the *identifier* subscript is dropped.

Homogenous coordinates contain an additional scaling term, k. The basis of homogenous coordinates allows scaling the coordinates for projective geometry operations. A scaling that reduces the k coordinate to a value of one is referred to as *normalizing* it and is accomplished by dividing all the coordinates by the value of k.

The following notation abbreviates a three-dimensional position in homogeneous coordinates:

$$\mathbf{p}_{identifier}^{frame} = \begin{bmatrix} kx \\ ky \\ kz \\ k \end{bmatrix} = \begin{bmatrix} p_{x,\,identifier}^{frame} \\ p_{y,\,identifier}^{frame} \\ p_{z,\,identifier}^{frame} \\ p_{k,\,identifier}^{frame} \end{bmatrix} \tag{2.2}$$

for any non-zero value of k.

Positions referenced in two-dimensional coordinate frames are similar, but do not include the z or kz term.

2.1.2 Navigational Reference Frames. A reference frame introduces the idea of an observer of a position. Standing at the origin of a reference frame and locating an object based on its distance along the axes of the reference frame defines the position of that object in that frame. For simplicity, the frames in this thesis are orthogonal (all axes of the frame are perpendicular). As needed, introductions of other frames that are not in this section occur throughout the thesis.

2.1.2.1 Inertial Frames. The universally true inertial-frame (I-frame) is the only non-accelerating frame discussed. Using the I-frame would unnecessarily complicate aircraft navigation that, by definition, is limited to altitudes relatively close to the surface of the Earth. As such, a locally defined inertial frame, Earth-centered inertial frame (i-frame), approximates the I-frame for the short durations of time associated with aircraft navigation. This frame is defined with an origin at the Earth's center of mass, the **Z** axis through the North Pole, the **X** axis pointing at the vernal equinox, and the **Y** axis completing the right-handed orthogonal system along the equatorial line. The i-frame does not rotate with the Earth, but its origin translates with the Earth as it orbits the Sun. In contrast, the I-frame never accelerates or rotates.

2.1.2.2 Earth Frame. A reference frame anchored permanently to ground-based locations on the Earth permits navigation with respect to the surface of the Earth. This frame, known as the Earth-centered Earth-fixed frame (e-frame), shares its **Z** axis with the i-frame while its **X** and **Y** axes rotate about the **Z** axis at the same rate as the Earth. The location of these axes are fixed to the surface of the Earth, with the **X** axis extending from the center of the Earth through the intersection of the Prime Meridian and the Earth's equatorial line [13], and the **Y** axis completing the right-handed orthogonal system. The e-frame components used were those of the World Geodetic System - created in 1984 (WGS 84), as defined by the National Geospatial-Intelligence Agency (NGA) [13], further discussed below. Positions referenced in this frame are realized in spherical coordinates as latitude, longitude, and height above ellipsoid (HAE) or in rectangular coordinates as x, y, and z translations. In the e-frame these are more commonly referred to as u, v, and w.

Many regional and local variations of the Earth's surface make a mathematical model difficult to create and use. The definition of an equipotential surface partially compensates for these variations. This surface is perpendicular to the local gravity vector, with equal gravity magnitudes throughout [25]. The shape is called the *geoid* and approximates the mean sea level (MSL) across its surface. The definition of a geometric surface called an ellipsoid, or oblate spheroid, approximates the geoid and permits mathematical computations that define position and positional relationship with respect to the actual surface of the Earth or the geoid.

A *datum* defines a reference ellipsoidal surface for geographic regions of the Earth. Defining the ellipsoid regionally makes use of an average MSL that minimizes the altitude deviations from the reference geoid surface. It is important to note that when discussing an object's position on the Earth it is impossible to tell where the precise location is without information about the datum used to define that location.

In order to make world-wide navigation seamless across geographic boundaries and multiple datum, a geodetic datum exists. This global datum, known as WGS 84,

serves as a best fit for many local systems, enabling reliable world-wide navigation. The NGA is responsible for maintaining the model of the ellipsoid and the gravitational model needed to create this datum. The NGA determines and tracks precise position of several world-wide locations that influence the precision of the model [13]. This tracking allows updates to the model based on changes of the Earth from tectonic motion, tides, etc. [13]

2.1.2.3 Navigation Frame. The navigation frame (n-frame) has its origin located on a platform of interest (aircraft) and is defined with the **X** axis pointing toward true north. The **Z** axis points in the direction of the gravitational pull of the Earth and the **Y** axis completes the right-handed orthogonal system. This is also referred to as north, east, down or NED orientation. This frame does not rotate with the platform. Its orientation is dependent on the platform's location with respect to the e-frame. The i-frame, e-frame, and n-frame are shown in Figure 2.1.

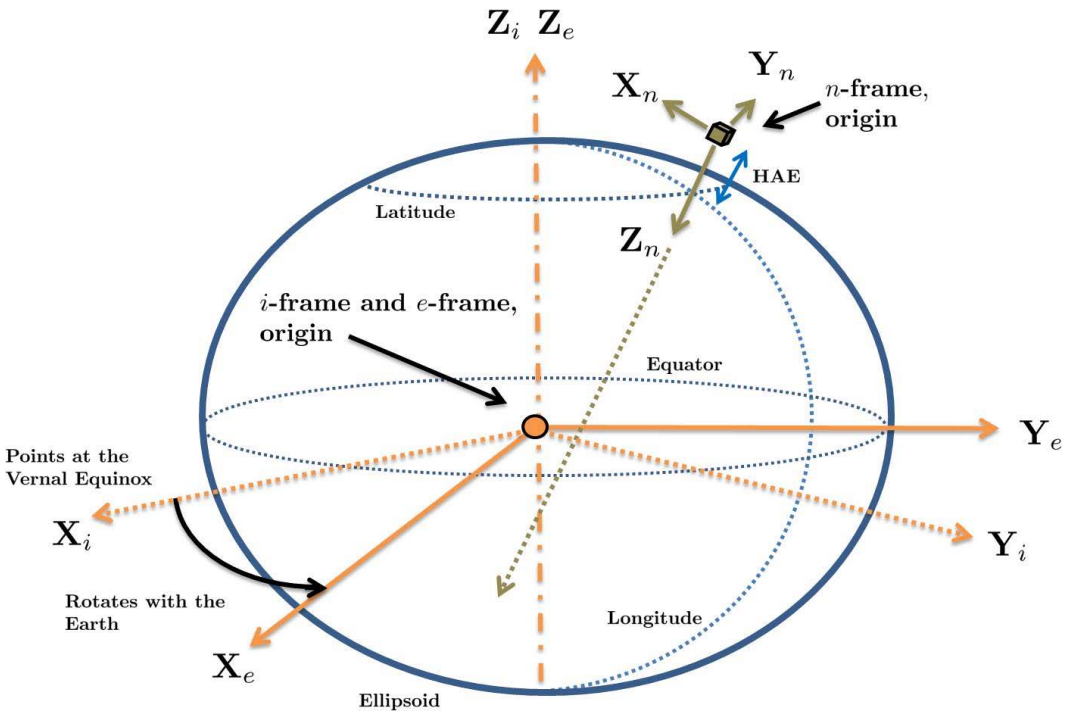


Figure 2.1: The i-frame, e-frame, and n-frame. The i-frame and e-frame both have their origin at the center of the Earth and the n-frame has its origin on the platform of interest [27].

2.1.2.4 Body Frame. Defining the orientation of an airborne platform using heading, pitch, and roll requires the use of the body frame (b-frame). For an aircraft, the b-frame is defined with the **X** axis projected through the nose of the aircraft, the **Z** axis through the bottom of the aircraft and the **Y** axis completes the right-handed orthogonal system, generally assumed to be from the origin out the right wing. The origin can be arbitrarily chosen. Typical choices include: the center of gravity of the aircraft, a truth collection device, or an inertial sensing unit. This frame rotates in conjunction with the platform on which it is defined and is shown in Figure 2.2, on an aircraft in a formation. The subscript W on the frame identifier denotes the aircraft's position in the formation, the *wing* aircraft, further explained in Section 2.1.4. The cube in the figure denotes the origin of this aircraft's b-frame and n-frame.

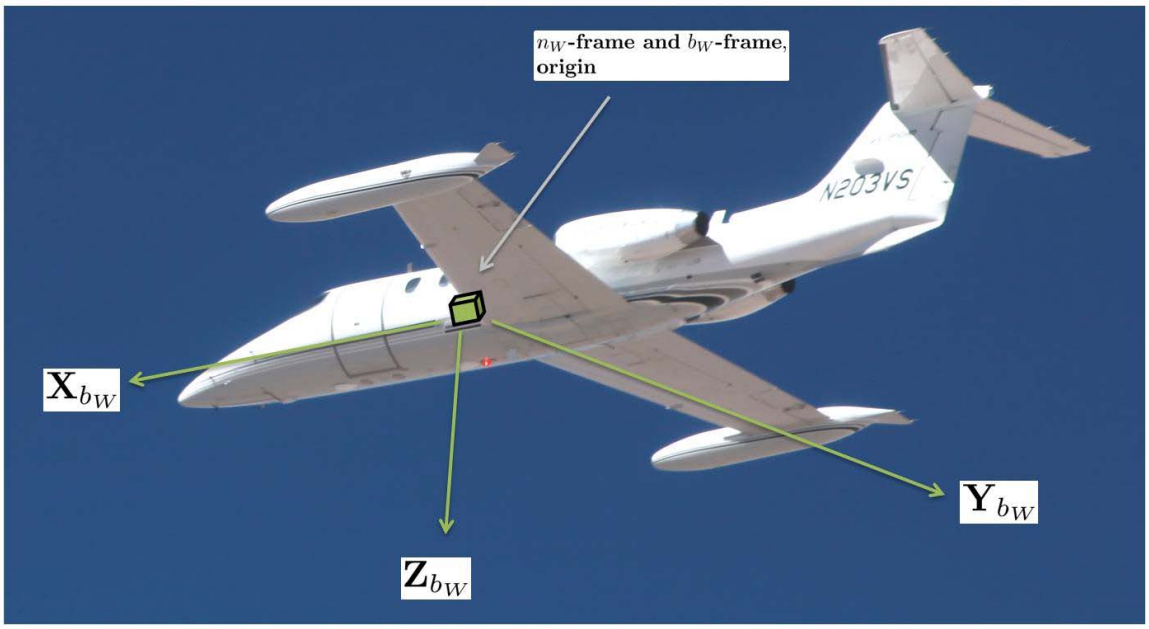


Figure 2.2: The b-frame of a wing aircraft. Denoted as b_W-frame, the b_W-frame and n_W-frame share a common origin.

2.1.2.5 Camera Frame. Referencing a position with respect to a camera requires the camera frame (*cam*-frame) as shown in Figure 2.3. The origin of the frame is located at the optical center of the camera, denoted with a circle in the

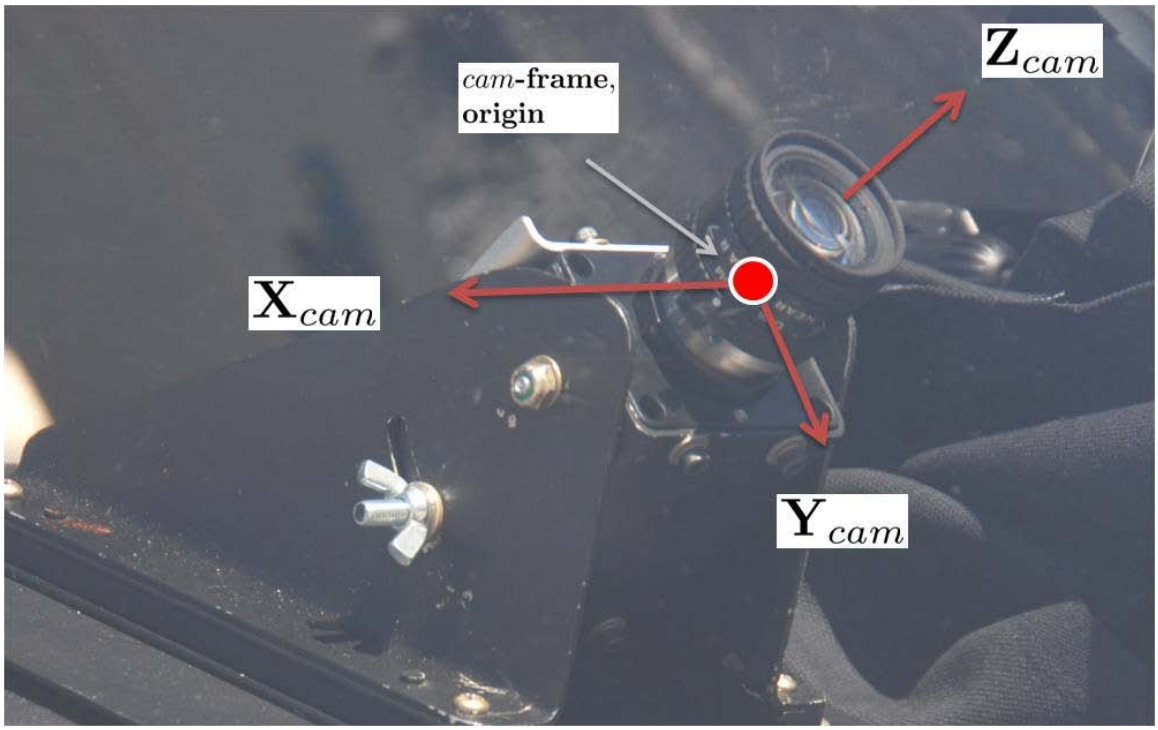


Figure 2.3: The *cam*-frame. The camera and the *cam*-frame are shown as installed on an aircraft's dashboard.

figure. The frame is defined with the **Z** axis out of the front of the lens. The **X** axis projects out of the right of the lens and the **Y** axis projects out of the bottom of the lens. This frame references the position of objects in the camera's field of view that are projected onto images. This process is further discussed in Section 2.1.3.5.

2.1.2.6 Image Frame. There are three different two-dimensional frames associated with images used in this research and two different types of images. The three frames are the *image*-frame, the *GLimage*-frame, and the *CVimage*-frame; the latter two are further discussed in Sections 2.2.1 and 2.2.2. The symbol **I** represents an image and the two types of images are *rendered* and *collected*. The symbol $\mathbf{I}_r$ refers to images that are rendered with the use of the OpenGL library, further detailed in Section 2.2.1. The symbol $\mathbf{I}_c$ refers to images that are collected with the camera.

The optical center of the camera is shown in Figure 2.3 and is the origin of the *cam*-frame. The *cam*-frame origin projected onto an $\mathbf{I}_c$ (along the **Z** axis of the

cam-frame) is typically close to, but not always, the center of the $\mathbf{I}_c$. Referenced to the *image*-frame, translations x_o and y_o denote the location of this projection, known as the *principal point*. The origin and axes of the *image*-frame, the location of the principal point, and their relation to a generic $\mathbf{I}_c$ are shown in Figure 2.4.

Additionally, $\mathbf{I}_c$ images and potentially $\mathbf{I}_r$ images are not necessarily square. Denoted in Figure 2.4 as θ_s [11], a skew angle is the angle between the top and sides of these images and is a concern when it is not 90°. [1]

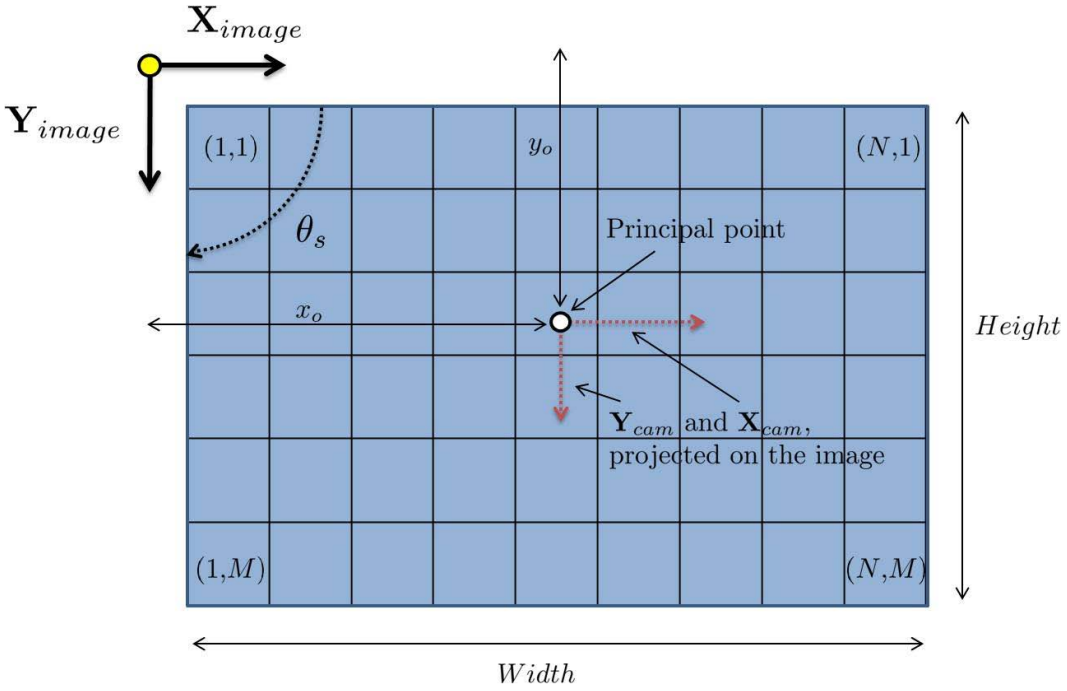


Figure 2.4: The *image*-frame. The origin and axes of the frame for a generic $\mathbf{I}_c$ are shown with the principal point (projection of the *cam*-frame $\mathbf{X}$ and $\mathbf{Y}$ axes on the $\mathbf{I}_c$). The generic $\mathbf{I}_c$ has a width of N *pixels* and a height of M *pixels*. For a non-square $\mathbf{I}_c$, θ_s represents the angle between the top and left side of the $\mathbf{I}_c$.

The origin of the *image*-frame is in the top-left corner of the $\mathbf{I}_c$, eliminating negative translation values (along the $\mathbf{X}$ and $\mathbf{Y}$ axes) and the need to have access to the coordinates of the principal point when referencing positions on the $\mathbf{I}_c$. The origin is actually 0.5 *pixels* up and 0.5 *pixels* to the left of the top left corner of the $\mathbf{I}_c$ [27], allowing the center of the first pixel to be annotated as shown in Figure 2.4.

[1] The subscript s is added to avoid confusion with θ associated with the roll angle of an aircraft.

The **X** axis of the *image*-frame defines the pixel location to the right of the origin and the **Y** axis defines the pixel location below the origin. An $\mathbf{I}_c$ is typically determined by dimensions measured in both a standardized unit of measurement (*centimeter*, *millimeter*, *inches*, etc) as well as camera-dependent dimensions measured in *pixels*. The terms W and H denote standardized units of measurement for width and height respectively, the terms N and M denote the *pixel* units of width and height respectively.

This section covered many of the frames used throughout this thesis. The next section explains the relationship between frames.

2.1.3 Reference Frame Conversions. There is a limit to the utility of understanding an object's position in a single frame of reference if that position cannot be referenced in other frames for further analysis and computation. This section presents various reference-frame conversions to increase the utility of both the reference frames and positions.

This thesis references points in both three-dimensional and two-dimensional space, and in both Cartesian and homogeneous coordinates. This requires the ability to convert between them all. The reference-frame conversion between Cartesian coordinate systems can require a *translation* and a *rotation*. Translation accounts for differences in the origins of the frames. Rotation accounts for the difference in orientation between the frames. The reference-frame conversion between homogenous coordinates requires a *transformation* or *camera* matrix accounting for both origin and orientation differences. *Mapping* or *projecting* describes these transformations and they are detailed in Section 2.1.3.5.

Direction Cosine Matrices (DCMs, Section 2.1.3.1) and Euler Angles (Section 2.1.3.2) convert Cartesian coordinates of a position referenced in one frame to Cartesian coordinates of the same point referenced in another frame. As shown in Figure 2.5, the actual position of an object (a camera) does not move during the conversion. The difference in translations along each axes is evident by orienting the

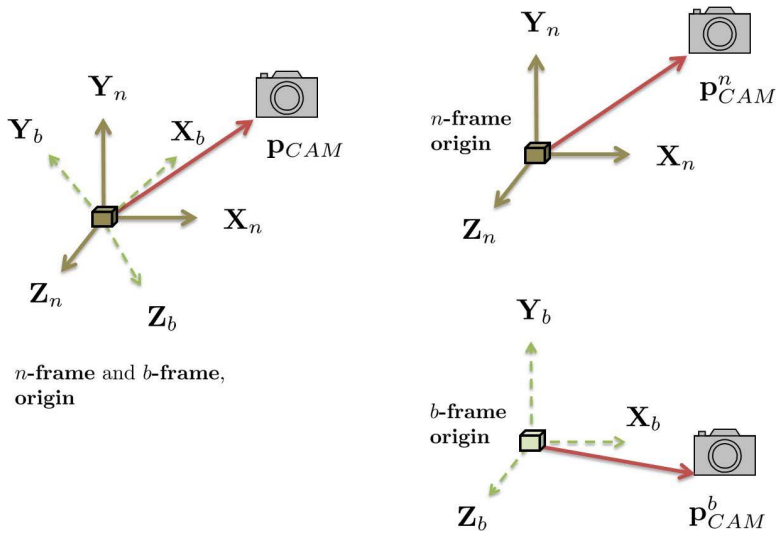


Figure 2.5: Reference frame conversion. Referencing a point in a different frame, does not change the location of the point. The *b*-frame is rotated to have the same orientation as the *n*-frame, in the right side of the image.

two frames in the same manner (right side of the image).

2.1.3.1 Direction Cosine Matrix. A DCM is a 3x3 matrix that simplifies the rotations from one reference frame to another. The matrix is created by expressing each axis unit-vector of one frame as vectors with respect to the other frame. Generally, the nomenclature for a DCM uses the $\mathbf{C}^{to}_{from}$ symbol, where the superscript immediately following designates the reference frame being converted *to* and a subscript letter immediately following designates the reference frame being converted *from*.

As an example, consider a camera with a known position ($[-1,+1,-1]^{\mathbf{T}}$) in the *n*-frame, $\mathbf{p}^n_{CAM}$ shown in Figure 2.6. This figure, shows the position of the camera referenced in both frames.

The symbol $\mathbf{C}^b_n$ is a DCM that rotates a position in the *n*-frame to a position in the *b*-frame (assuming collocated origins). $\mathbf{C}^b_n$ multiplied by the position $\mathbf{p}^n_{CAM}$ determines the location of the camera in the *b*-frame, or $\mathbf{p}^b_{CAM} = \mathbf{C}^b_n \mathbf{p}^n_{CAM}$.

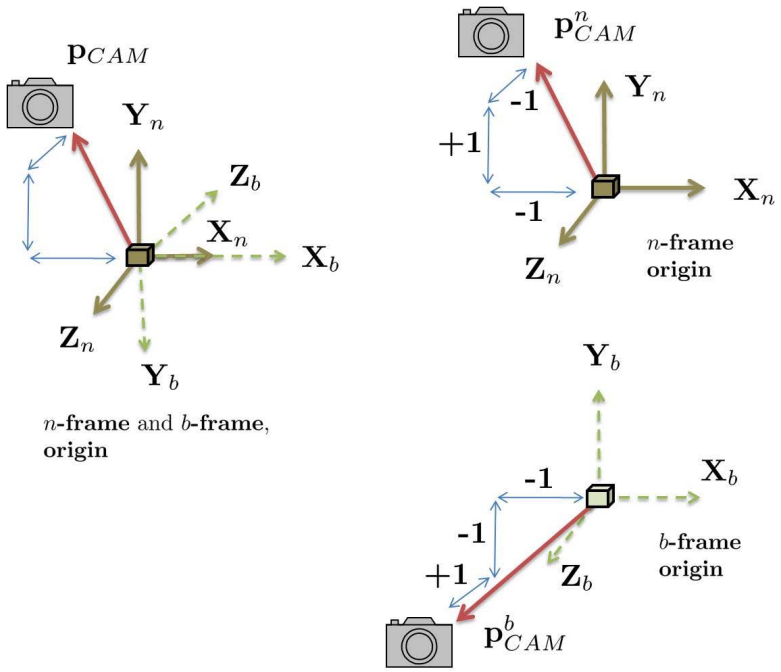


Figure 2.6: Reference frame conversion, DCM. A point with a known location in the n-frame is rotated to be defined in the b-frame. The b-frame is rotated to have the same orientation as the n-frame, in the right side of the image.

Equations (2.3) and (2.4) expand this relationship for the defined frames in Figure 2.6:

$$\mathbf{p}^b_{CAM} = \mathbf{C}^b_n \cdot \mathbf{p}^n_{CAM}, \quad \mathbf{C}^b_n = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{2.3}$$

$$\mathbf{p}^b_{CAM} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \tag{2.4}$$

DCMs are by definition orthonormal and non-singular and have the following properties:

$$Det(\mathbf{C}^n_b) \leftrightarrow |\mathbf{C}^n_b| = 1 \tag{2.5}$$

$$\mathbf{C}^b_n = (\mathbf{C}^n_b)^{\mathbf{T}} = (\mathbf{C}^n_b)^{-1} \tag{2.6}$$

$$\mathbf{C}^b_e = \mathbf{C}^b_n \mathbf{C}^n_e \tag{2.7}$$

2.1.3.2 Euler Angles. Euler Angles are a second method to describe the rotation between reference frames by specifying an angle of rotation in a single two-dimensional plane at a time. By projecting a three-dimensional frame onto a two-dimensional plane of another reference frame, the XY plane for example, the rotation between the two frames is a single angle. This angle is the first rotation, shown as the ψ rotation in Figure 2.7. Projection onto a second plane defines another angle and then again, such that three angles (ψ,θ,ϕ) are attained.

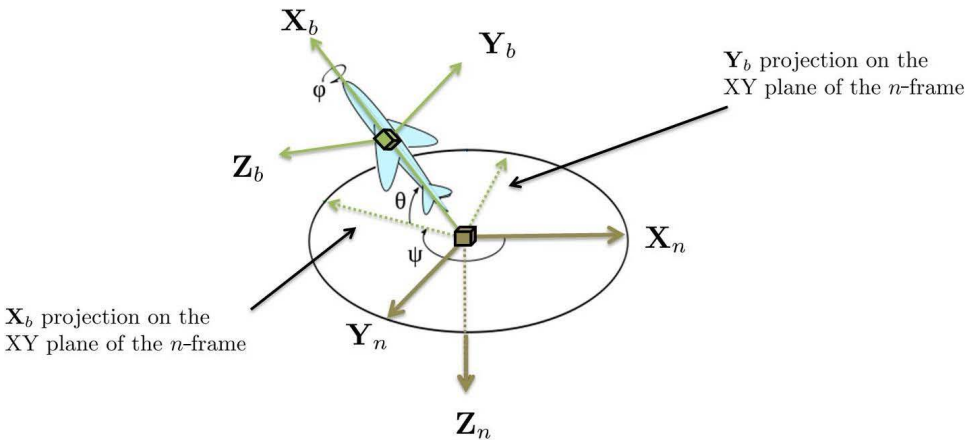


Figure 2.7: Rotational Euler angles for a n-frame to b-frame conversion. The n-frame has been offset from the b-frame intentionally. The angles are determined and applied in series [20].

Euler angles have some limitations. The angles are computed in series and must be utilized in the same order. A common convention and the one used in this research is that of *3-2-1* - first the XY plane (yaw), then the XZ plane (pitch), and finally the YZ plane (roll). If the same order is not maintained throughout, attitude errors will occur.

For n-frame to b-frame conversions, the Euler Angles are defined as (in order of 3-2-1):

- ψ = rotation about the $\mathbf{Z}_b$ axis (heading or yaw)
- θ = rotation about the $\mathbf{Y}_b$ axis (pitch)
- ϕ = rotation about the $\mathbf{X}_b$ axis (roll)

In this thesis, the terms *heading* and *yaw* are interchangeable. A positive Euler angle corresponds to a positive rotation about that axis defined by the right-hand rule. The rotation described by the example in Section 2.1.3.1 was a roll rotation of 180°.

Conversions also exist between DCMs and Euler angles. The following in-matrix computation uses the Euler angles to create the b-frame to n-frame DCM, $\mathbf{C}_b^n$ [25].

$$\mathbf{C}_b^n = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{2.8}$$

The substitutions, $\psi = 0°$, $\theta = 0°$, and $\phi = 180°$ result in the same DCM ($\mathbf{C}_n^b$ or $\mathbf{C}_b^n$, equivalent in that example through the relationship in Equation (2.6)) shown in Equation (2.3). With a known DCM, the following computations determine the Euler angles:

$$\theta = -\sin^{-1}\left(\mathbf{C}_b^n(3,1)\right) \tag{2.9}$$

$$\phi = \sin^{-1}\left(\frac{\mathbf{C}_b^n(3,2)}{\cos(\theta)}\right) \tag{2.10}$$

$$\psi = \sin^{-1}\left(\frac{\mathbf{C}_b^n(2,1)}{\cos(\theta)}\right) \tag{2.11}$$

where $\mathbf{C}_b^n(i,j)$ represents the coefficient in the ith-row and jth-column of the DCM.

From these equations, a singularity occurs when θ (pitch) is close to or equal to $\pm 90^o$ ($cos(\pm 90^o) = 0$, causing Equations (2.10) and (2.11) to be undefined). This is not a concern in this research because an aircraft at this attitude would be completely vertical; the test flights did not put the aircraft in this condition.

When the origins of the reference frames are not collocated, the translation between frames is performed prior to the rotations previously described.

The following equation represents an e-frame to n-frame conversion:

$$\mathbf{p}^n_{CAM} = \mathbf{C}^n_e[\mathbf{p}^e_{CAM} - \mathbf{p}^e_{NAV}] \tag{2.12}$$

where $\mathbf{p}^e_{NAV}$ is the location of the new coordinate system with respect to the former, in this case the origin of the n-frame as it is located in the e-frame.

The transformations and projections of homogenous coordinates expand on the rotation and translation operations presented between Cartesian coordinates. The next section presents the transformation matrices describing this process.

2.1.3.3 Transformation Matrix. A transformation between coordinates is a linear relationship, represented in this thesis by a matrix, $\mathbf{T}$. The DCMs presented in Section 2.1.3.1 are a special type of 3×3 transformation.

Transformation matrices permit general mapping and projections of points between coordinate frames. Beyond the specific rotational transformation of a DCM, a general transformation matrix can scale or shear points in addition to rotating them. The term *perspective projection* is a specific type of transformation. This transformation projects three-dimensional positions onto a two-dimensional plane along lines that emanate from a single location, or the center of the projection.

The effects of homogeneous coordinates in a perspective projection transformation are best illustrated with an example. To illustrate, the origin of the *cam*-frame is designated as the center of projection. Figure 2.8 depicts two distinct points in three-dimensional space, with the Cartesian coordinates shown in the image, ($\mathbf{p}^{cam}_1 = [x,y,x]^\mathbf{T}$ and $\mathbf{p}^{cam}_2 = [2x,2y,2x]^\mathbf{T}$). The plane the points are projected onto is parallel to the $\mathbf{Y}$ and $\mathbf{X}$ axes at a distance of one unit. Through a perspective projection, both points project to the same position on this two dimensional plane. Because the coordinates of these points all share the same ratio, 2:1, the points represent two of an infinite number of points, along the same line, that will also project to the same point on this plane. The scaling term, $k = 1$, included into the homogenous

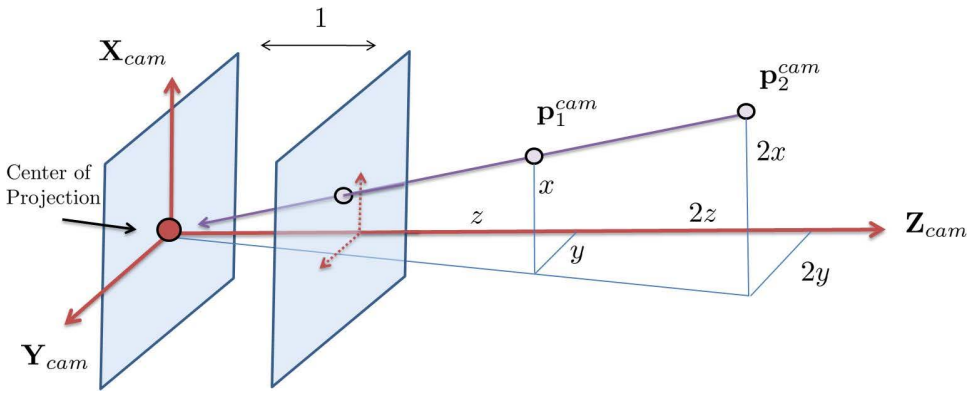


Figure 2.8: Projecting three-dimensional points onto a plane. All the points along a line emanating from the center of the projection (the origin of the *cam*-frame) project to the same position using a perspective projection transformation.

coordinates permits the scaling needed to project these point to the same position on the plane using a single matrix. The correct homogenous coordinates to use in this projection are $\mathbf{p}_1^{cam} = [x,y,x,1]^{\mathbf{T}}$ and $\mathbf{p}_2^{cam} = [2x,2y,2x,1]^{\mathbf{T}}$.

This projection of points is shown mathematically for the example points in Figure 2.8. Consider the transformation matrix, $\mathbf{T}$:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.13}$$

Projecting these points onto this plane with the use of the transformation has the following results:

$$\mathbf{p}_{1*}^{cam} = \mathbf{T} \cdot \mathbf{p}_1^{cam} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z \end{bmatrix} \tag{2.14}$$

$$\mathbf{p}_{2*}^{cam} = \mathbf{T} \cdot \mathbf{p}_2^{cam} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2x \\ 2y \\ 2z \\ 1 \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \\ 2z \\ 2z \end{bmatrix} \tag{2.15}$$

where the * represents the new location of the point.

These two points can be normalized by the value of their k scaling term; in other words, scaled by $\frac{1}{\underline{p}_{k,\,1*}^{plane}} = \frac{1}{z}$ and $\frac{1}{\underline{p}_{k,\,2*}^{plane}} = \frac{1}{2z}$, respectively. The resulting points $\mathbf{p}_{1*}^{cam}$ $= \mathbf{p}_{2*}^{cam} = [\frac{x}{z}, \frac{y}{z}, 1, 1]^{\mathbf{T}}$ occupy the same position, located on the plane.

A camera can be represented in a similar manner. A camera uses a lens to capture visible points along lines emanating from the origin of the *cam*-frame onto a two-dimensional plane, or $\mathbf{I}_c$. This information is a two-dimensional representation of the three-dimensional world where this camera exists. A 3×4 transformation matrix, referred to as the camera matrix $\mathbf{K}$, represents a perspective projection transformation specific to a given camera and lens. Mathematically, $\mathbf{K}$ describes the mapping of three-dimensional positions onto two-dimensional images attained through the lens [8]. Before fully introducing the camera matrix, the next section presents a basic understanding of lenses.

2.1.3.4 Pinhole Camera Model. The understanding of projective geometry is the basis for the projection of a three-dimensional world onto a two dimensional image by a lens. Understanding the geometries involved with lenses helps develop precise mathematical relationships between an object and an image. These mathematical relationships permit real-world determinations of positions based on information contained in an image. A pinhole camera model approximates the relationship between the real world and an image collected by an ideal pinhole camera. Using this model, the necessary relationships are developed for a generic camera.

A typical lens used by a camera (biconvex) alters parallel light incident on its surface towards a point that is a fixed distance away, known as the focal length (f), behind the lens. An example biconvex lens is shown in Figure 2.9.

The lens also alters non-parallel light, but in a different manner that is better understood by examining the *fundamental equation of the thin lens* from [12], as acquired from [27] and shown in Figure 2.10.

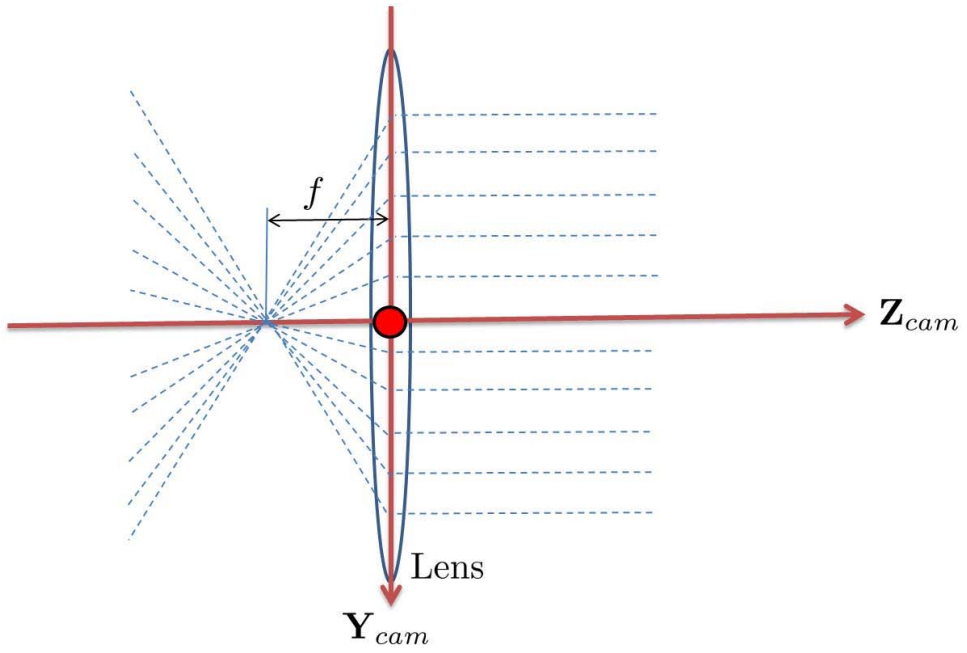


Figure 2.9: A biconvex lens. Parallel light hitting a biconvex lens focuses at a single point that is a fixed distance behind the lens. This distance is the focal length of the lens.

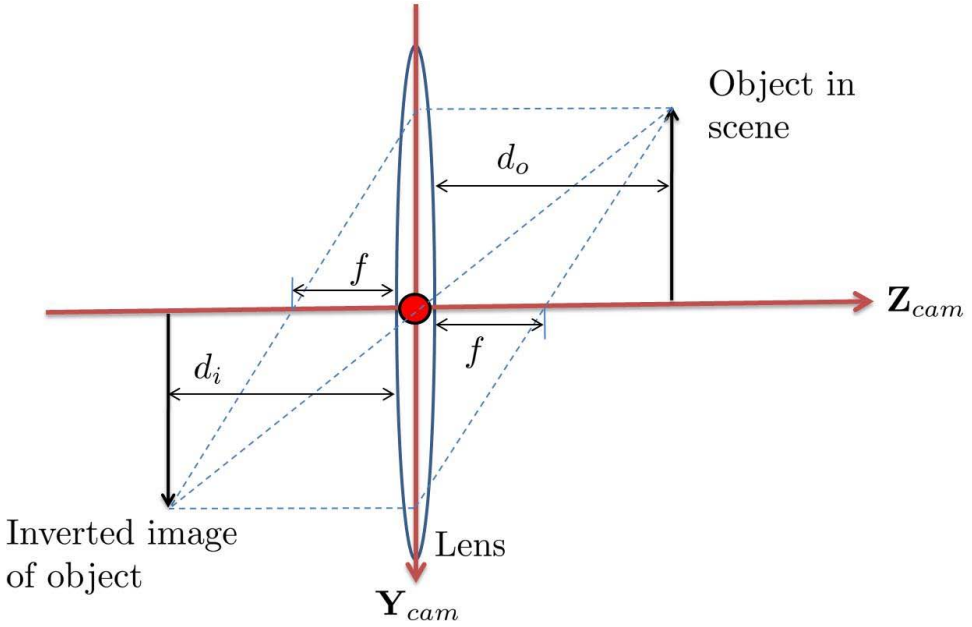


Figure 2.10: Thin lens model. The *fundamental equation of the thin lens* explains that light incident on the lens from a point source (on the top of the arrow) a distance (d_o) in front of the lens arrives at the same point a distance (d_i) behind the lens.

From the thin lens theory, light that is incident on the lens and not parallel does so in a predictable manner, such that all light that irradiates from a point source (the tip of the arrow in the figure) arrives at the same point a certain distance behind the lens.

The following relationship between the distances in Figure 2.10 can be shown:

$$\frac{1}{d_i} + \frac{1}{d_o} = \frac{1}{f} \tag{2.16}$$

where d_o is the distance from the object in the scene to the lens, d_i is the distance from the lens to the image, and f is the focal length of the lens. Moving the object farther away (d_o increases), from a lens with a fixed focal length (f constant), the distance to the image (d_i) decreases and its relative image size (or the image translation in the $\mathbf{Y}$ axis) decreases.

The pinhole model reduces the size of the lens in Figure 2.10 to the size of a tip of a pin. All of the light from the scene that is incident on the lens passes through the optical center of the lens and is projected on to an image plane located a focal-length, f, behind the lens. This is shown in Figure 2.11, where $\mathbf{p}_1$ is an arbitrary point source at the tip of the arrow.

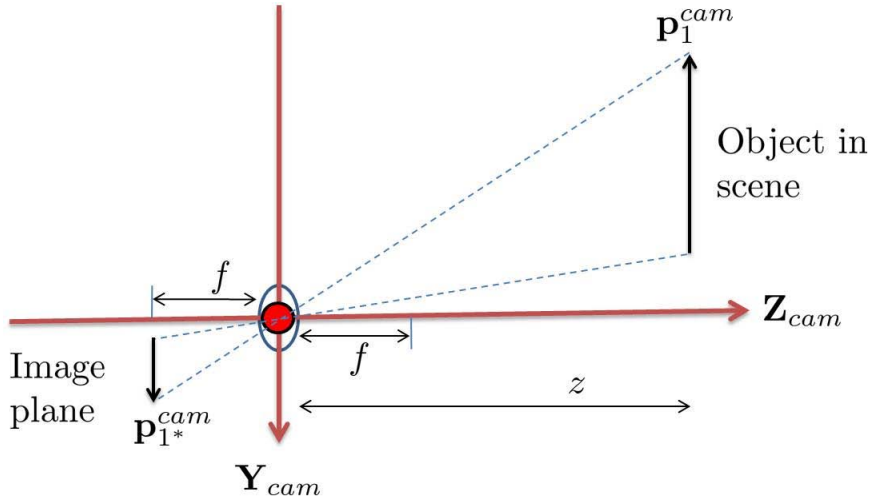


Figure 2.11: Pinhole model. Light from the scene, incident on the lens, is projected to an image a distance f behind the lens.

The physical location of $\mathbf{p}_1$ does not actually move to the image plane. Annotating the subscript *identifier* with an * differentiates this visual representation of a point from the physical location of a point.

From the pinhole model depiction in Figure 2.11, the two triangles created by dotted lines on either side of the lens are geometrically similar. The angles in the triangles are the same, the ratio between the sides are the same, and the locations of the triangles' vertices are related by the negative of that same ratio, or $-\frac{f}{z}$. This analogy shows that the translation of the visual presentation along the $\mathbf{Z}$ axis, $p^{cam}_{z,1*}$, is equal to the ratio $-\frac{f}{z}$ times the translation of the original point in the scene $p^{cam}_{z,1}$, or $p^{cam}_{z,1*} = -\frac{f}{z} \cdot p^{cam}_{z,1}$. The following equation expands this relationship to all the position coordinates of the points [27] [12]:

$$\mathbf{p}^{cam}_{1*} = -\frac{f}{z}\mathbf{p}^{cam}_{1} \tag{2.17}$$

Individually, all the scalar translations of $\mathbf{p}^{cam}_1$ are scaled by the same ratio and negated. It is typical, for visual simplicity and to minimize the mathematical conversions required, to place the image plane in front of the camera frame (such that $p^{cam}_{z,\,1*} = +f$) [8], as shown in subsequent figures. This also removes the need for the negative sign in Equation (2.17).

The physical effects of a lens create a perspective projection transformation matrix, detailed in Section 2.1.3.3, described as a linear mapping of homogenous points in [8]. The following equation expands the relationship of Equation (2.17) into a matrix that transforms three-dimensional homogenous coordinates to the two-dimensional homogenous coordinates:

$$\mathbf{p}^{\,cam}_{1*} = \begin{bmatrix} f_x x \\ f_y y \\ z \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathbf{T} \cdot \mathbf{p}_1^{\,cam} \tag{2.18}$$

where f_x and f_y are the focal lengths in the $\mathbf{X}_{cam}$ axis and the $\mathbf{Y}_{cam}$ axis respectively.

The normalization of the resulting two-coordinate homogenous point (divide the coordinates by the k value, which is z) in Equation (2.18), is the same as the division by z in Equation (2.17). Expanding the two axis pinhole model to show all three axes shows the relationship between the *cam*-frame and the *image*-frame in Figure 2.12.

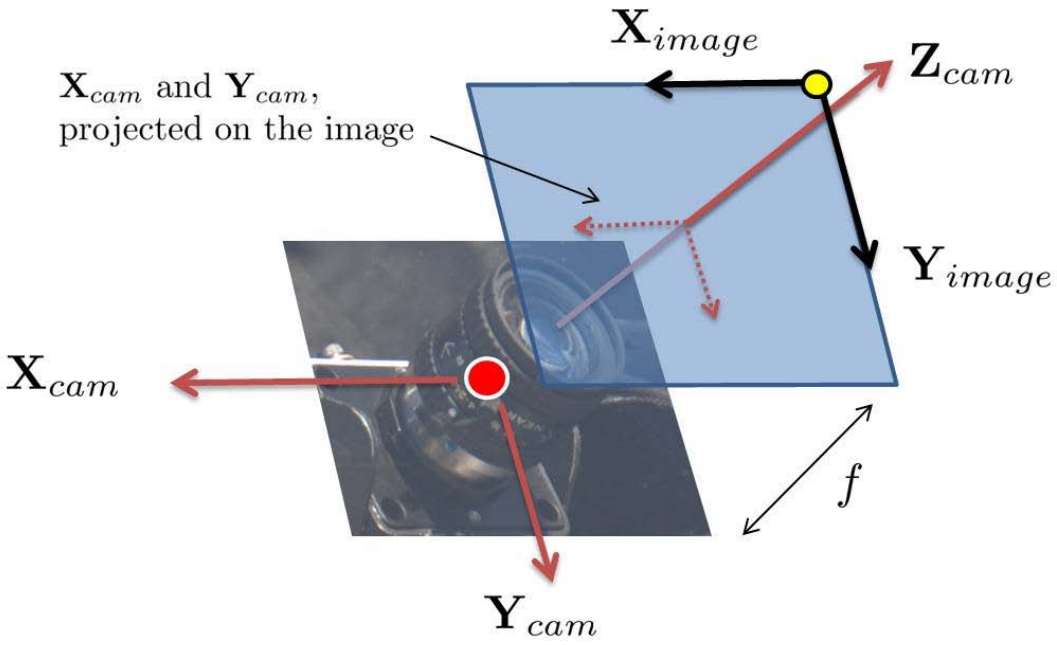


Figure 2.12: The *cam*-frame and *image*-frame relationship. A three-dimensional pinhole camera model demonstrates this relationship.

This relationship is valid for the pinhole model; however, real-world lenses are not ideal pinhole lenses. Real lenses distort the image [9]. Distortion effects of a lens, specifically a camera lens, are examined in the next section. In addition, the transformation between the real-world and images collected by a camera ($\mathbf{I}_c$ images) are expanded to include the conversion to the *image*-frame.

2.1.3.5 Camera Model and Camera Matrix. Locating an object in an image involves three characteristics of a camera. The first set of characteristics is called the *extrinsic parameters* of the camera. These parameters locate the object in the *external* reference frame of the camera, or the *cam*-frame. The second characteristics set is the distortion effects from a non-pinhole lens. The third characteristics

set is called the *intrinsic parameters* of the camera. These parameters locate the object in the *internal* reference frame of the camera, or the *image*-frame. This section details the parameters used to create this entire transformation.

The external relationship between the camera and other navigation frames uses the same rotations and translations as described in Section 2.1.3. The following equation expands the relationship of Equation (2.12), but not the specific values, to account for homogenous coordinates [8]:

$$\underline{\mathbf{p}}^{cam} = \begin{bmatrix} \mathbf{C}_e^{cam} & -\mathbf{C}_e^{cam}\mathbf{p}_{CAM}^e \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \underline{\mathbf{p}}^e \tag{2.19}$$

where $\underline{\mathbf{p}}^e$ is any point referenced in the e-frame, $\mathbf{p}_{CAM}^e$ is the location of the camera in the e-frame, and $\mathbf{0}_{1\times 3}$ is a row vector of three zeros. This matrix is the same as translating and then rotating the location of the point into a new reference frame. This transformation is the extrinsic relationship of the camera to other reference frames. The next section presents the distortion effects of the camera.

A simple model for the distortion effects of the lens approximates the transformation of the normalized points in the *cam*-frame to distorted positions in the *cam*-frame. A calibration process for a specific camera determines these distortion effects [2]. Two components, radial and tangential, comprise the total distortion model. The first component, radial, affects both the x translation and the y translation in the same manner, as a function of their distance from the $\mathbf{Z}_{cam}$ axis [9]:

$$x_r = x\left(1 + c_1 r^2 + c_2 r^4 + \dots\right) \tag{2.20}$$

$$y_r = y\left(1 + c_1 r^2 + c_2 r^4 + \dots\right) \tag{2.21}$$

with the temporary substitutions: x for p_x^{cam} and y for p_y^{cam}, x_r and y_r are the positions of the point in the *cam*-frame, accounting for radial distortion; c_1 and c_2 are arbitrary coefficients that a calibration of the camera and the lens determines. This

distortion model estimate of the actual distortion of the lens depends on the number of coefficients used. For this thesis, three coefficients were sufficient.

The tangential distortion model affects the x translation and the y translation differently. This part of the distortion model accounts for non-symmetric distortion between the two axes, the tangential distortion in each axis is dependent on both translations:

$$\delta x_t = 2c_3xy + c_4(r^2 + 2x^2) + \dots) \tag{2.22}$$

$$\delta y_t = 2c_4xy + c_3(r^2 + 2y^2) + \dots) \tag{2.23}$$

where δx_t and δy_t are the change in position of the point in the *cam*-frame, accounting for tangential distortion. Common practice is to limit these coefficients to two.

One overall distortion model combines the effects of the two distortion models, with the addition of the third radial-distortion coefficient, c_5 [9]:

$$x_d = x + c_1r^2x + c_2r^4x + c_5r^6x + 2c_3xy + c_4(r^2 + 2x^2) \tag{2.24}$$

$$y_d = y + c_1r^2y + c_2r^4y + c_5r^6y + 2c_4xy + c_3(r^2 + 2y^2) \tag{2.25}$$

where x_d and y_d are the normalized positions of the point, accounting for distortion. Through a calibration process, the determination of these coefficients creates the distortion model for a particular camera and lens combination at a fixed focal length. With a defined distortion model a common practice is to un-distort the image. An un-distorted image approximates the image a pinhole camera would have collected. This is accomplished by moving individual pixels in an $\mathbf{I}_c$ through the reverse of the above equations. The result is a better estimate of the normalized locations of the points in the *cam*-frame. Because the intrinsic transformation of the camera (presented next) is a linear transformation, this un-distortion process can be applied directly to the $\mathbf{I}_c$. For the remainder of this thesis, the theory presented assumes un-distorted images.

The final characteristics of a camera, the intrinsic parameters, determine the transformation on a position in the *cam*-frame to its representation on the $\mathbf{I}_c$ as referenced in the *image*-frame. Equation (2.18) detailed the projection from the scene to the image plane, with both points referenced in the *cam*-frame. To convert the visual representation into the *image*-frame, an additional scaling and translation occur. This transformation can be seen in Figure 2.13.

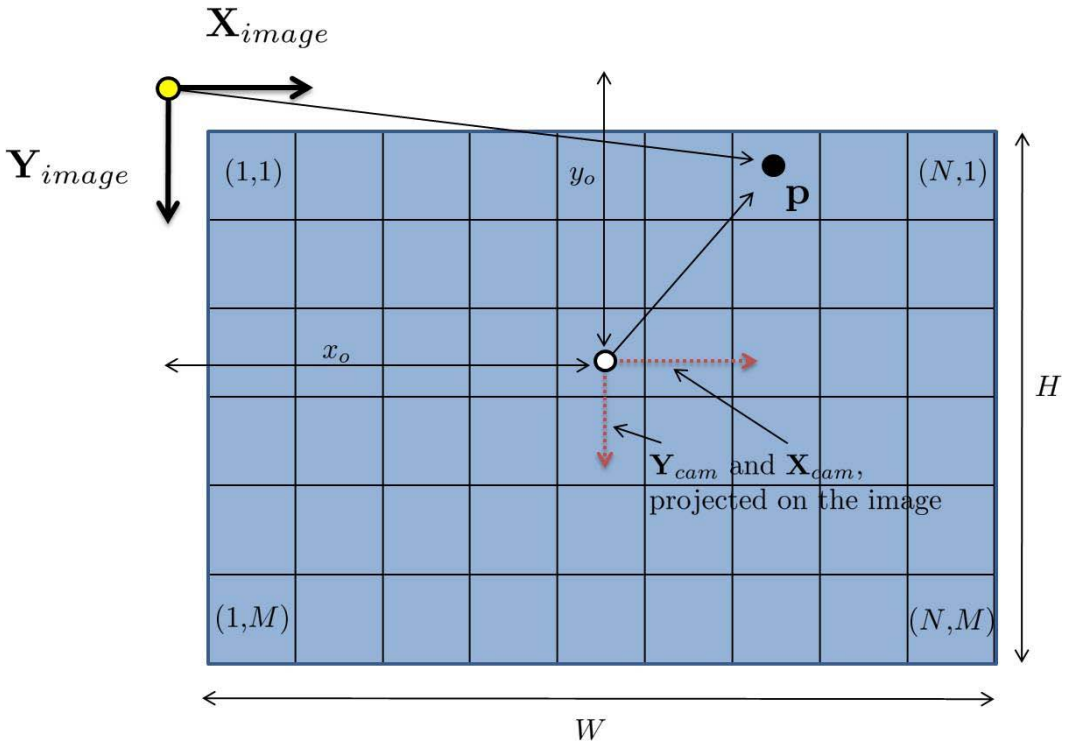


Figure 2.13: Mapping *cam*-frame to *image*-frame. The intrinsic camera parameters determine the projection of an object, located physically in the *cam*-frame to a visual represented location in the *image*-frame.

The scaling terms account for the difference in the units of measurement. Frames external to the camera denote translations in a standardized measurement unit (*feet, meter, miles*, etc.), the *image*-frame denotes translations in *pixels*. To scale the locations, two ratios are used, one for each the $\mathbf{X}$ and $\mathbf{Y}$ axis. The ratio of the width in *pixels* over the width in standardized measurements, or $\frac{N}{W}$, scales the $\mathbf{X}$ axis translation, similarly $\frac{M}{H}$ scales the $\mathbf{Y}$ axis translation. Additionally, the term *skew*,

or s, accounts for the possibility of a non-orthogonal $\mathbf{I}_c$, defined as:

$$s = -\frac{\frac{N}{W}}{\frac{M}{H}} \cot(\theta_s) \tag{2.26}$$

The translation between frames accounts for the location of the principal point in the *image*-frame, $[x_o, y_o]^{\mathbf{T}}$.

The addition of the scaling and translation to Equation (2.18), provides the complete transformation from *cam*-frame to *image*-frame:

$$\underline{\mathbf{p}}^{image} = \begin{bmatrix} f_x\frac{N}{W} & s & x_o & 0 \\ 0 & f_y\frac{M}{H} & y_o & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \underline{\mathbf{p}}^{cam} \tag{2.27}$$

The $f_x\frac{N}{W}$ term represents the focal length of the lens in the $\mathbf{X}_{cam}$ axis, in *pixels*; it is replaced by a single symbol α. The $f_y\frac{M}{H}$ term represents the focal length of the lens in the $\mathbf{Y}_{cam}$ axis, in *pixels*; it is replaced by β.

The camera matrix $\mathbf{K}$ is defined:

$$\mathbf{K} = \begin{bmatrix} \alpha & s & x_o & 0 \\ 0 & \beta & y_o & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.28}$$

Combining the intrinsic and extrinsic transformations of the camera, and assuming the removal of distortion effects, the following complete transformation relates real-world locations to their location in an image:

$$\underline{\mathbf{p}}^{image} = \mathbf{K} \begin{bmatrix} \mathbf{C}_e^{cam} & -\mathbf{C}_e^{cam}\mathbf{p}_{CAM}^e \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \underline{\mathbf{p}}^{e} \tag{2.29}$$

This section demonstrated the relationship between Cartesian and homogenous coordinates, including conversions, rotations, translations, and transformations between frames. This information is the basis for the navigational relationship used

throughout the thesis. The next section expands on some of these concepts as they relate to aircraft formation.

2.1.4 Formation Coordinate Frames. Two aircraft known as the *lead* aircraft and the *wing* aircraft define a basic formation. Multi-element and multi-ship can define extended formations that involve any number of additional aircraft. Navigation within a formation is always a complex endeavor that involves extensive planning and discussion between all involved operators to ensure safe operations.

Formation also complicates the aircraft's navigation. Position and orientation with respect to an Earth-referenced navigation frame is now coupled with position and orientation with respect to single or multiple formation aircraft frames. Location in the formation dictates when reference to one frame will have priority over the other, though neither is used completely independent of the other. For a two-ship formation, the lead aircraft is usually the only additional navigational reference point for the wing aircraft.

The lead aircraft has two separate coordinate systems. The first is that of lead navigation frame or n_L-frame. This frame is independent of aircraft orientation, and was further described in Section 2.1.2.3.

The second coordinate system defined with respect to the lead aircraft is that of lead body frame, or b_L-frame. This frame is rigidly attached to lead, and was further described in Section 2.1.2.4.

Other aircraft within the formation define their location based on one of the frames of the lead aircraft. While there is not an established rule on which frame is preferred, in practice it is dependent on the distance away from the lead aircraft. The farther an aircraft is away, the harder it becomes to visually determine the attitude of the lead aircraft and appropriately, the n_L-frame is used. When close and the lead aircraft's attitude is readily apparent, navigation is typically done with the use of the b_L-frame. For the purposes of AAR, the aircraft will be close, and normally the b_L-frame is used.

The other aircraft in the formation also have coordinate frames associated with them. The wing aircraft have both a b-frame, or b_W-frame, and a n-frame, or n_W-frame. They are both similarly defined as the coordinate frames of the lead aircraft.

The aspects of formation navigation, as they relate to AAR will be further discussed in Chapter 3. The next section details the background information on the programming, including the open-source libraries used in the research.

2.2 Programming Libraries

A large portion of this research is facilitated by the programming code that ultimately provides the navigational solution. The programming for this research used the C programming language.

Within the C language, two powerful image rendering and image processing libraries exist, OpenGL and OpenCV. Both libraries provide critical capabilities to their respective areas, and ultimately to this research. OpenGL is a graphics rendering library that is an industry standard and provides fast, accurate, and flexible images. OpenCV is a computer vision library that utilizes quick operations on matrices and allows real-time analysis of images. Many terms, especially the names of reference frames, are not industry standard but were chosen to minimize confusion between the different disciplines covered in this research. To limit the depth of this section, many important steps and processes are not presented. The intent of this section is to provide a general understanding of these libraries for readers unfamiliar with them and to demonstrate relationships with physical cameras and images. This section looks at the basics of both libraries and their interactions. More in-depth explanations throughout the thesis further expand on the libraries interaction with the research.

2.2.1 OpenGL. OpenGL is a programming library that interfaces with a platform's graphics hardware. The "GL" portion of the library name stands for Graphics Library. Because of its platform-independence, ease of use, and rendering

accuracy and speed, OpenGL is used throughout the computer graphics industry. Using OpenGL to render a virtual image ($\mathbf{I}_r$) of objects involves setting up a scene with objects defined by collections of small polygons and applying appropriate lighting conditions in a customizable viewing-volume. In other words, it involves defining a volume of space (called a frustum) and placing objects inside the frustum to see them on the screen.

The main assumption behind the OpenGL library is that a collection of polygons can approximate any object in a scene. As the number of polygons increase, and their size decrease, the smoothness of textures and surfaces increase towards a representation where individual polygons are not recognizable because of their minute size. Various editing programs exist to create the objects in a polygon representation. Coloring individual polygons to match the actual or desired texture of the object adds realism to the scene.

Orientation in the OpenGL world requires coordination frames, similar to those described in Section 2.1.2. The user determines the location of the OpenGL world frame (GL-frame), anywhere in the OpenGL world, in whatever units are required by the user. It is simply defined by referencing other items in its coordinate frame.

Likewise, the user is free to determine other frames. A camera in the OpenGL world can exist wherever the user determines and typically the optical center of the camera is co-located with the origin of the GL-frame, but this is not required. The camera has a frame associated with it: the *GLcam*-frame. In contrast to the camera defined in Section 2.1.2.5, the *GLcam*-frame has the negative $\mathbf{Z}$ axis projecting into the viewing area, the $\mathbf{Y}$ axis defines the vertical axis in the up direction, and the $\mathbf{X}$ axis defines the horizontal axis to the right. More OpenGL-specific frames will be introduced when needed.

Using the parameters of a camera, detailed in Section 2.1.3.5, customizes the $\mathbf{I}_r$ produced by OpenGL to mimic the $\mathbf{I}_c$ collected by the camera. By modifying the OpenGL frustum to approximate the viewing characteristics of a camera and lens,

renderings of an object can approximate what the object would look like if a camera collected the images. The bulk of this research relies on the correct creation of an $\mathbf{I}_r$ for comparison with an $\mathbf{I}_c$ of the same object. The following sections describe a basic OpenGL setup.

2.2.1.1 OpenGL setup. There are two different types of views available in OpenGL, orthographic and projective. An orthographic view maintains the aspect of parallel lines. For example, an image of railroad tracks would never show the two rails intersecting. Orthographic views are used when a fixed relative scale is required, regardless of the distance from an object. For example, engineering diagrams of a building would use an orthographic view to keep the floors parallel and the intersection of floors and walls perpendicular.

A projective view is what lenses, including the human eye, capture. A projective view does not maintain parallel lines, instead the view introduces the concept of a vanishing point. Those same railroad tracks eventually intersect at a theoretical distance of infinity in a projective view. Humans understand this concept naturally; as an example train conductors do not slow down trains because the tracks appear to narrow. Although both views are available in OpenGL, a projective view produces images that represent a real world scene.

Defining the projective view in OpenGL requires the creation of a projective frustum, best understood visually in Figure 2.14. The definition of a frustum requires defining a near and far plane ($zNear$ and $zFar$), a Field of View (FOV) (typically the FOV in the Y direction, FOV_Y, in degrees or radians), and an aspect ratio between FOV_X and FOV_Y. Alternatively, through trigonometric relationships of the truncated four-sided pyramid, defining *left*, *right*, *top*, and *bottom* of the near *clipping plane* can also define the frustum. Both methods are similar; however, the latter allows a little more customization of the scene. The term *clipping planes* describes the imaginary walls of the frustum because they *clip* objects from the eventual $\mathbf{I}_r$ of the scene. Figure 2.14 shows these terms visually with the orientation of the $GLcam$-frame.

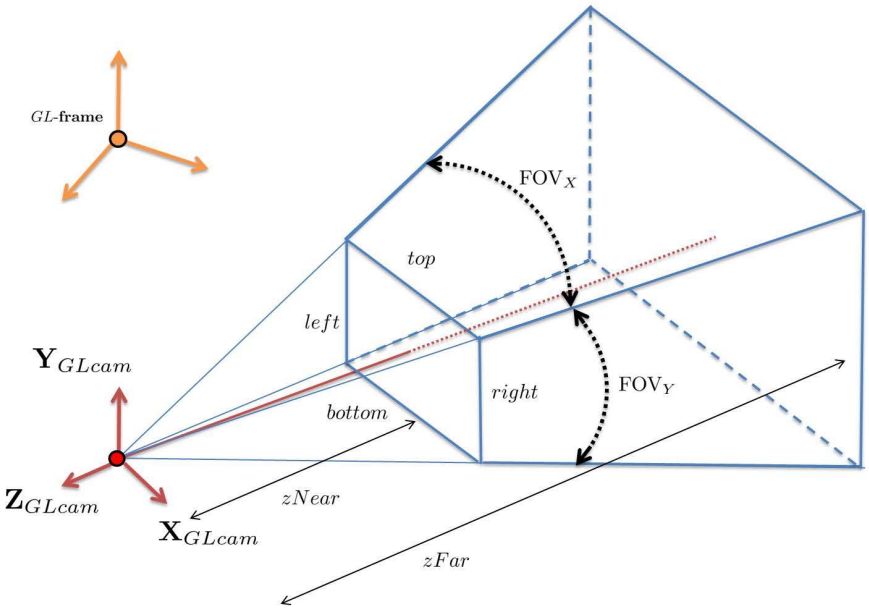


Figure 2.14: OpenGL viewing volume [21]. Using the respective FOVs or the definition of the $zNear$ plane defines the space in the OpenGL world that the camera can potentially see.

OpenGL contains a few different functions that create a projective frustum. A simple view would utilize the glFrustum( ) command as shown in Listing 2.1 [19,21]. It is important to note that the values, $zNear$ and $zFar$, are distances, not translations.

Listing 2.1: glFrustum ( ) function declaration

```
void glFrustum(GLdouble left, GLdouble right, GLdouble bottom,
    GLdouble top, GLdouble zNear, GLdouble zFar);
```

To simulate the view as though a lens of a camera created it, the following solves for the respective FOVs:

$$\mathrm{FOV_Y} = 2 \cdot \tan^{-1}\left(\frac{\frac{H}{2}}{f_y}\right) = 2 \cdot \tan^{-1}\left(\frac{\frac{M}{2}}{\beta}\right) \tag{2.30}$$

$$\mathrm{FOV_X} = 2 \cdot \tan^{-1}\left(\frac{\frac{W}{2}}{f_x}\right) = 2 \cdot \tan^{-1}\left(\frac{\frac{N}{2}}{\alpha}\right) \tag{2.31}$$

which can be used to solve for the parameters in Listing 2.1.

To understand how an object in the frustum appears on the screen, another important frame critical to the use of OpenGL is introduced: the normalized device coordinate frame. For consistent notation, it is represented in this thesis as the NDC-frame. The OpenGL engine uses this frame to determine what, where, and how to place objects on the eventual $\mathbf{I}_r$. Located in this frame is a cube termed the canonical viewing volume (CVV). It has a size of two units by two units by two units with the NDC-frame origin at its center. This frame has the unfortunate characteristic of being left handed with the **Y** axis projected out the top of the viewing volume, the **X** axis out the right, and the **Z** axis into the frustum. Every part of a scene transforms into this frame for at least two simple determinations.

As an over-simplification of this process, the NDC-frame is shown co-located with the $GLcam$-frame in Figure 2.15. Because of the involved transformations, from the *cam*-frame to NDC-frame, the conversion is not as simple as negating the **Z** axis of the two frames.

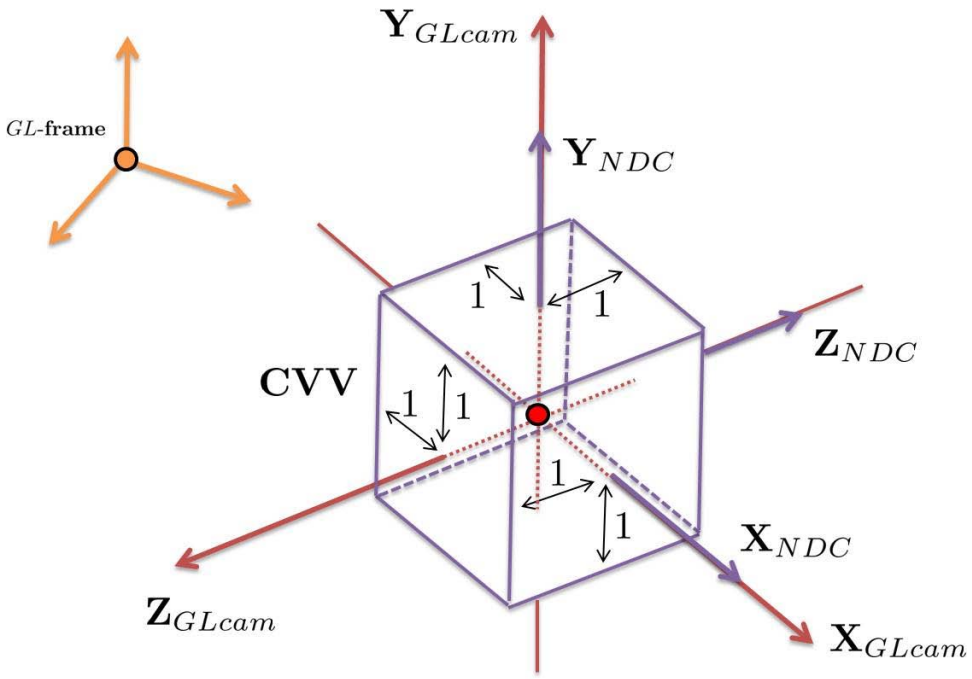


Figure 2.15: OpenGL canonical viewing volume. The OpenGL engine uses this volume to determine what objects in the scene can be placed on the $\mathbf{I}_r$. At the center of the cube is the origin of the NDC-frame.

OpenGL first uses the CVV to determine what objects, or parts of an object, should appear on the screen. If the transformed position falls within the cube, it has the potential to be seen on the screen. Second, OpenGL determines which objects within the cube obscure other objects. If two objects, or more precisely if two pixels from the scene, have the same x and y translation in this frame then the pixel with the lower z translation will be shown (those with z translations less than negative one were already discarded in the first step.)

As a simplified overview of the complete OpenGL rendering process, the following steps occur. First, the projective GL world is transformed such that the projective viewing volume shown in Figure 2.14 becomes a rectangular shape that is then scaled to the shape of the CVV shown in Figure 2.15. Second, collapsing the contents of the cube onto the rear wall (located at z = -1 in the NDC-frame) creates a two unit by two unit image of the scene. Third, scaling that image to the $\mathbf{I}_r$ size required by the user (nominally, the on-screen window size) completes the process. Figure 2.16 details the entire transformation that is presented in the following sections. The first two transformations $\mathbf{T}_1$ and $\mathbf{T}_2$ from the figure are modified slightly in Chapter 5.

2.2.1.2 GLcam-frame to CVV. The transformation from *GLcam*-frame to the NDC-frame and the projective frustum to the CVV is a two-step process. The first step transforms the lines emanating from the *GLcam*-frame origin to lines that are parallel ($\mathbf{T}_1$) and then scales the perspective to the scale of the CVV ($\mathbf{T}_2$). For simplicity, an intermediate coordinate frame is not defined for the intermediate step between *GLcam*-frame to NDC-frame; instead both are referenced in the NDC-frame with an understanding that a secondary transformation occurs.

This first step does not create an orthographic view of the scene; rather it allows the projective view to be scaled such that it becomes rectangular in shape. Since the computer does not have a lens to create this view, it does it digitally by increasing the scale of objects closer to the zNear clipping frame while decreasing the scale of those closer to the zFar clipping frame.

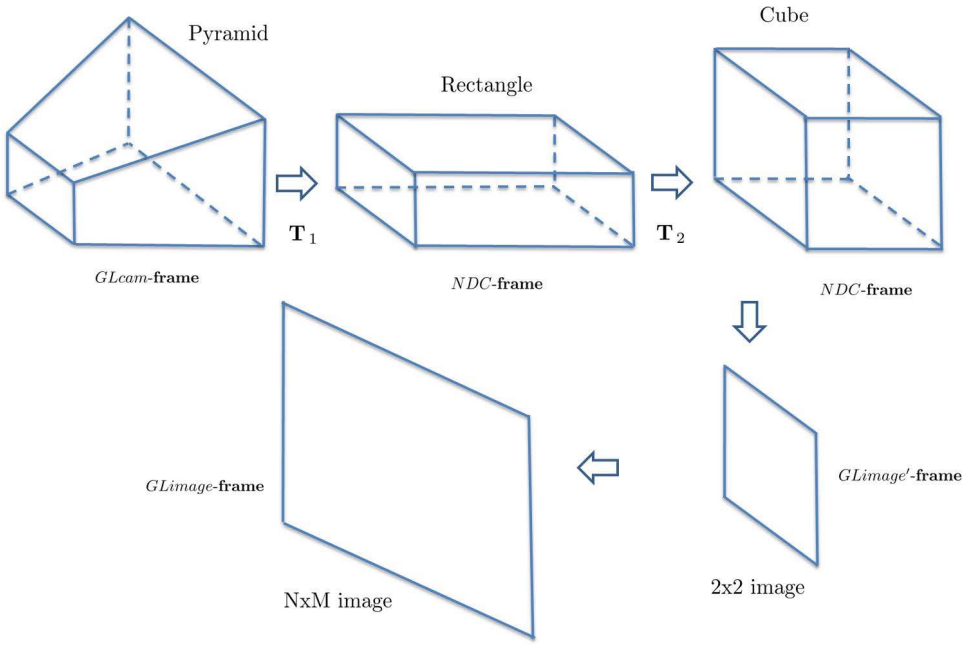


Figure 2.16: Overview OpenGL process. The OpenGL process starts with a defined viewing frustum and transforms it into a 2x2x2 cube, the contents of the cube are placed on a 2x2 image that is expanded to the necessary size.

As an example, compare the relative size of the two arrows in Figures 2.17 and 2.18. In the projective view, they are both located close to the zNear and zFar clipping planes, but within the defined projective frustum. Additionally, both have the same height (visually, the length in the y translation). Because Arrow 1 is closer to the camera frame it spans a larger angle of the frustum's FOV_Y. In contrast Arrow 2 is farther away and spans a smaller angle. The first transformation converts all the lines emanating from the center of projection (the *GLcam*-frame origin) into parallel lines.

This transformation appropriately scales the entire scene in the *GLcam*-frame such that the four sided, truncated pyramid-shaped frustum becomes rectangular in shape. A transformation matrix ($\mathbf{T}_1$) represents the first transformation:

$$\mathbf{T}_1 = \begin{bmatrix} zNear & 0 & 0 & 0 \\ 0 & zNear & 0 & 0 \\ 0 & 0 & zNear + zFar & zNear * zFar \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{2.32}$$

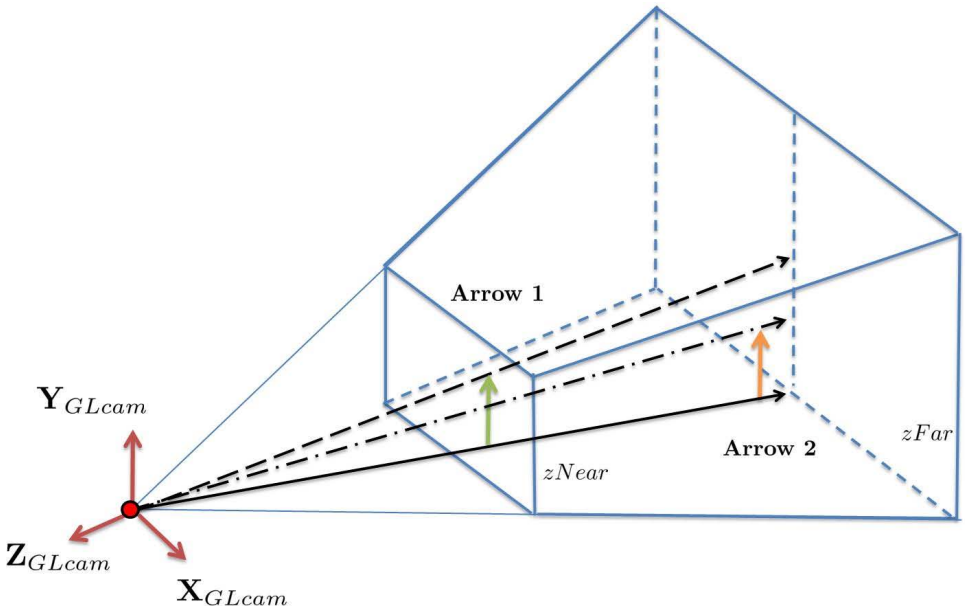


Figure 2.17: Two arrow example, projective frustum. In this example, the two arrows referenced in the *GLcam*-frame appear to have the same height.

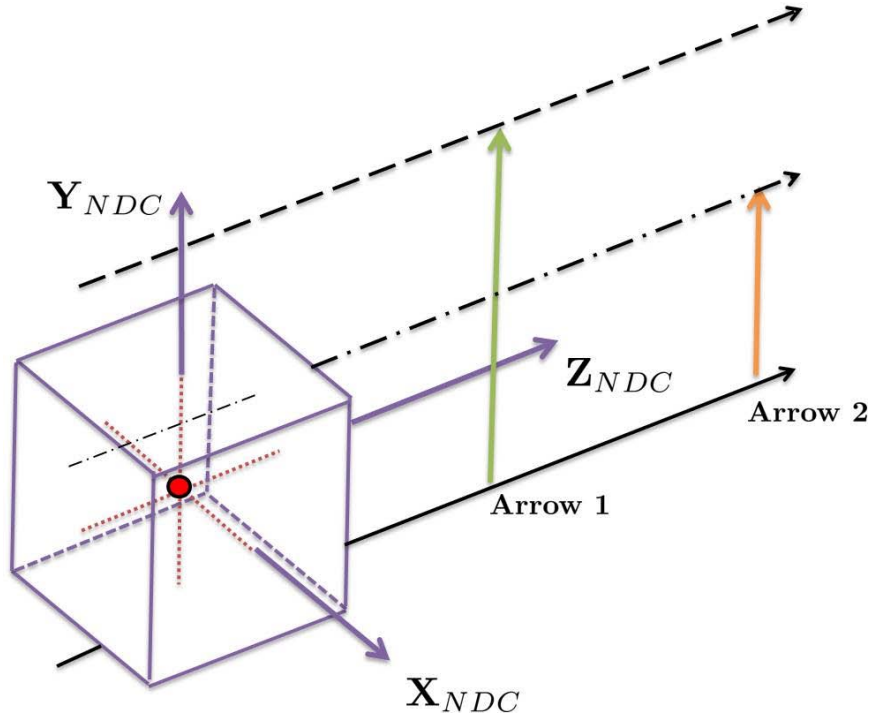


Figure 2.18: Two arrow example, *NDC*-frame. The example scene from Figure 2.17 is transformed, such that the lines emanating from the origin of the *GLcam*-frame become parallel. Arrow 1 is now scaled larger than Arrow 2.

This first transformation does not account for scaling (the projective view frustum scale to the cube scale). This is illustrated in Figure 2.18 as the arrows do not reside within the CVV. The second step ($\mathbf{T}_2$) scales the transformation accomplished in the first step appropriately; in other words the rectangular-shape frustum becomes cube-shaped. For completeness, there is a final step that normalizes the homogenous coordinates. In the second step, shown in Figure 2.19, the two arrows both fall within the CVV and both have the potential to be seen on the $\mathbf{I}_r$. However, in this example arrow 1 would most likely cover up arrow 2 and arrow 2 would not be seen on the $\mathbf{I}_r$.

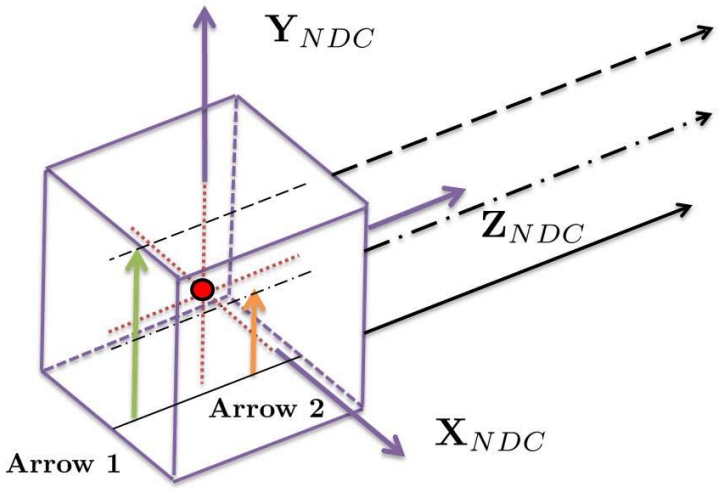


Figure 2.19: Two arrow example, CVV. The second step in the transformation process scales the scene in the NDC-frame, such that the projective viewing volume defined by glFrustum ( ) is transformed to the CVV.

The second transformation is represented as a transformation matrix ($\mathbf{T}_2$):

$$\mathbf{T}_2 = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{zNear-zFar} & \frac{zFar+zNear}{zNear-zFar} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.33}$$

This transformation ($\mathbf{T}_2$) is the same transformation required if the original scene was an orthographic view and not projective. It can be created with a call to the OpenGL library function glOrtho( ), shown in Listing 2.2 [22].

Listing 2.2: glOrtho( ) function declaration

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,
    GLdouble top, GLdouble zNear, GLdouble zFar);
```

OpenGL also creates a similar transformation with the call to glFrustum( ). The transformation created by glFrustum( ) is the same as the combination $\mathbf{T}_2 \cdot \mathbf{T}_1$ (placed in this order because the original points are pre-multiplied by the transformations).

The transformation created by glFrustum( ) is shown in Equation (2.34). This transformation accomplishes the two steps of the *GLcam*-frame to CVV transformation.

$$\text{glFrustum} = \begin{bmatrix} \frac{2*zNear}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2*zNear}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & \frac{zFar+zNear}{zNear-zFar} & \frac{2*zFar*zNear}{zNear-zFar} \\ 0 & 0 & -1 & 0 \end{bmatrix} = \mathbf{T}_2\mathbf{T}_1 \qquad (2.34)$$

This one step transformation is seen in Figure 2.20, with the inclusion of the lines emanating from the center of projection in each viewing volume.

For an example of this transformation, consider a projective viewing-volume that is symmetric both horizontally and vertically (equal viewing area on both sides of the **Z** axis). The function call terms can be replaced with width (W) and height (H) values, such that $left = -\frac{W}{2}$, $right = \frac{W}{2}$, $bottom = -\frac{H}{2}$, and $top = \frac{H}{2}$, causing the first two values in the third column in Equation (2.34) to be zero. With that setup, consider a point in the *GLcam*-frame with the following translations:

$$\mathbf{p}_1^{\text{GLcam}} = \begin{bmatrix} -\frac{W}{2} & \frac{H}{2} & -zNear & 1 \end{bmatrix}^{\mathbf{T}} \qquad (2.35)$$

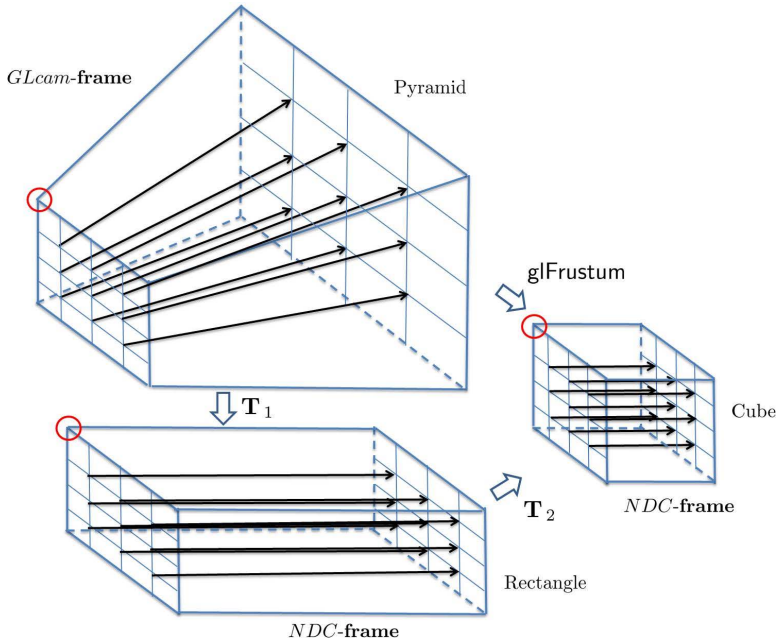


Figure 2.20: The glFrustum( ) transformation. The function creates a transformation matrix that maps the eight corners of the projective viewing volume into the shape of a cube.

This point is located in the projective-view frustum on the near clipping plane in the top left corner, shown as a circle on the pyramid in Figure 2.20. Pre-multiplying this point by the transformation created by glFrustum( ), the intermediate point results in:

$$\mathbf{p}_1^{NDC} = \begin{bmatrix} \frac{2*zNear}{W} & 0 & 0 & 0 \\ 0 & \frac{2*zNear}{H} & 0 & 0 \\ 0 & 0 & \frac{zFar+zNear}{zNear-zFar} & \frac{2*zFar*zNear}{zNear-zFar} \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} -\frac{W}{2} \\ \frac{H}{2} \\ -zNear \\ 1 \end{bmatrix} = \begin{bmatrix} -zNear \\ zNear \\ -zNear \\ zNear \end{bmatrix} \tag{2.36}$$

which normalizes to:

$$\mathbf{p}_1^{NDC} = \begin{bmatrix} -1 & 1 & -1 & 1 \end{bmatrix}^{\mathbf{T}} \tag{2.37}$$

This position is located on the top, left, rear corner of the eight unit^3 CVV cube and would be seen on the image. Similarly, the other seven coordinates that make

up the pyramid would transform to the other seven corners of the cube. This section covered the first step the next section details the second step of the OpenGL process.

2.2.1.3 CVV to GLimage-frame. The final transformation of an object to an $\mathbf{I}_r$ is now presented. Locations on an $\mathbf{I}_r$ are referenced in the *GLimage*-frame. This frame is located with the origin at the bottom-left corner of the resulting $\mathbf{I}_r$. The $\mathbf{X}$ axis projects out the right side of the image, parallel to the $\mathbf{X}$ axis of the *GLcam*-frame and the $\mathbf{Y}$ axis projects out the top of the image, parallel to the $\mathbf{Y}$ axis of the *GLcam*-frame.

To demonstrate the transformation first requires the $GLimage'$-frame as an intermediary frame. This frame is similarly defined as the NDC-frame but is located at the bottom, left, rear corner of the CVV cube, or one unit below, one unit to the left, and one unit behind the NDC-frame origin as shown in Figure 2.21. The transformation of a point located in the NDC-frame to the same point referenced in the $GLimage'$-frame requires adding one unit to all three coordinates.

Similar to the *cam*-frame to *image*-frame transformation (Section 2.1.3.5), the projection of the NDC-frame on to the $\mathbf{I}_r$ is shown in Figure 2.21 to visualize the conversion process. To make the transformation into the *GLimage*-frame, the image is scaled to the size of the user-defined window, typically not accomplished using the OpenGL library (however, various utility wrappers exists, such as OpenGL utility toolbox (GLUT) that will create these windows). As shown in Figure 2.22 the scaling for the $\mathbf{X}$ axis uses the ratio of the desired width in *pixels*, N, over the current width, two, or $(\frac{N}{2})$, with similar scaling used for the $\mathbf{Y}$ axis $(\frac{M}{2})$. This is the default transformation for OpenGL, while additional customization can be used to move the origin anywhere in the image and to define a new width and height for a smaller viewing window.

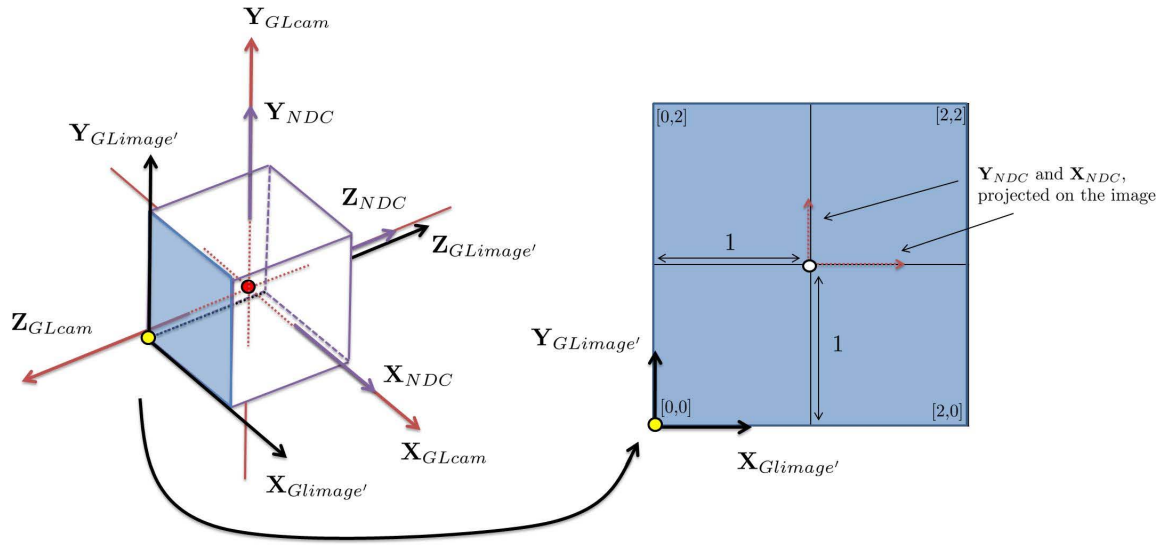


Figure 2.21: The *GLimage′*-frame. The frame is located at the bottom, left, rear corner of the CVV cube. The projection of the *NDC*-frame is shown projected on to the image.

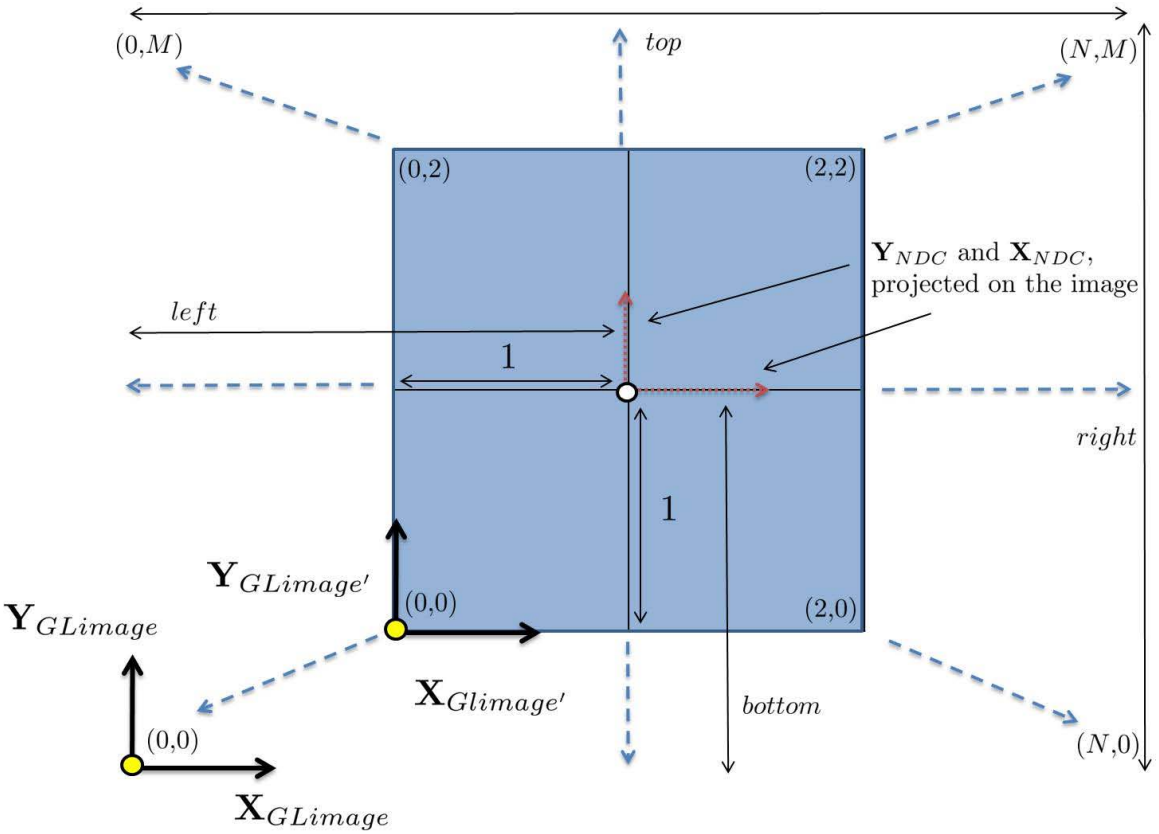


Figure 2.22: The *GLimage′*-frame to *GLimage*-frame. The image in the *GLimage′*-frame is stretched to the user's need to create the $\mathbf{I}_r$.

The mathematical transformation from CVV to *GLimage*-frame is shown:

$$\underline{p}_x^{GLimage} = \frac{N}{2}(\underline{p}_x^{NDC} + 1) \tag{2.38}$$

$$\underline{p}_y^{GLimage} = \frac{M}{2}(\underline{p}_y^{NDC} + 1) \tag{2.39}$$

The total transformation from an object in the GL world to an image combines the two transformation, *GLcam*-frame to CVV with CVV to *GLimage*-frame.

To render a scene in OpenGL, the typical process starts with the origin of the model located at the origin of the *GLcam*-frame, shown in Figure 2.14 as the intersection of the three axes. The process then translates and rotates the model into the frustum, at which point the conversion process presented in this section occurs. If portions of the model, in its final position prior to rendering, fall outside of the frustum, the process clips it from the resulting rendered image, $\mathbf{I}_r$, detailed further in the next section.

2.2.1.4 OpenGL Image. An advantage of OpenGL rendering is that it is efficient. The OpenGL rendering engine creates and displays an $\mathbf{I}_r$ quickly, typically without extensive modification, and then discards it (clearing memory space for the next image). To efficiently operate, OpenGL references an $\mathbf{I}_r$ as a 1-dimensional array [1]. An array for a simple $\mathbf{I}_r$ with a height of six *pixels* and a width of nine *pixels* is shown in Figure 2.23.

As a drawback to OpenGL accessing images in this manner, the ability to access a single pixel, or the color value of a single pixel, is not as efficient as it is with an OpenCV image (shown in the next section). To access a pixel in the OpenGL image structure requires some knowledge of the image, additional programming to access it, and again to modify it. A simplistic example is increasing the value of the green component for every pixel in the image represented in Figure 2.23. The value in the second array storage location (G in pixel one) and the value in every third array storage location thereafter would be accessed directly, modified, and then the image

Figure 2.23: OpenGL image storage configuration [1]. OpenGL accesses pixel information from an image using a single array with a length equal to the total number of image pixels x three.

could be displayed. Additionally, the rendering process must be complete and the image located on the visual producing hardware before access to pixels is available.

Of note, the bottom left of an image (the origin of the *GLimage*-frame) is the location of the first pixel in an OpenGL image, contrary to the location of the first pixel in an OpenCV image, which is in the top left.

This completes the introduction to OpenGL, more specifics of the library are presented throughout the thesis. The next section presents the details of the OpenCV library, followed by the interaction between the two libraries.

2.2.2 OpenCV. OpenCV is a programming library that provides tools for analyzing images; typically an $\mathbf{I}_c$ acquired from an optical-type sensor, such as a camera, infrared sensor, radar, etc. The "CV" portion of the library's name stands for Computer Vision. The tools the library provides extract quality information from the images.

The basis behind the OpenCV's analysis is creating access to an image as a matrix with a width and height equal to the number of pixels. For color, the matrix has an additional dimension with additional layers for color components as shown in Figure 2.24. Each value in the matrix corresponds to the color or grayscale intensity of the corresponding pixel. By realizing images in this way, quick, effective, and accurate analysis is possible. As an example, to increase the value of the green component in every pixel in an image, the entire memory for the green component (a single level of the matrix) can be accessed at one time and modified collectively, a more efficient method than OpenGL would implement. The language also allows

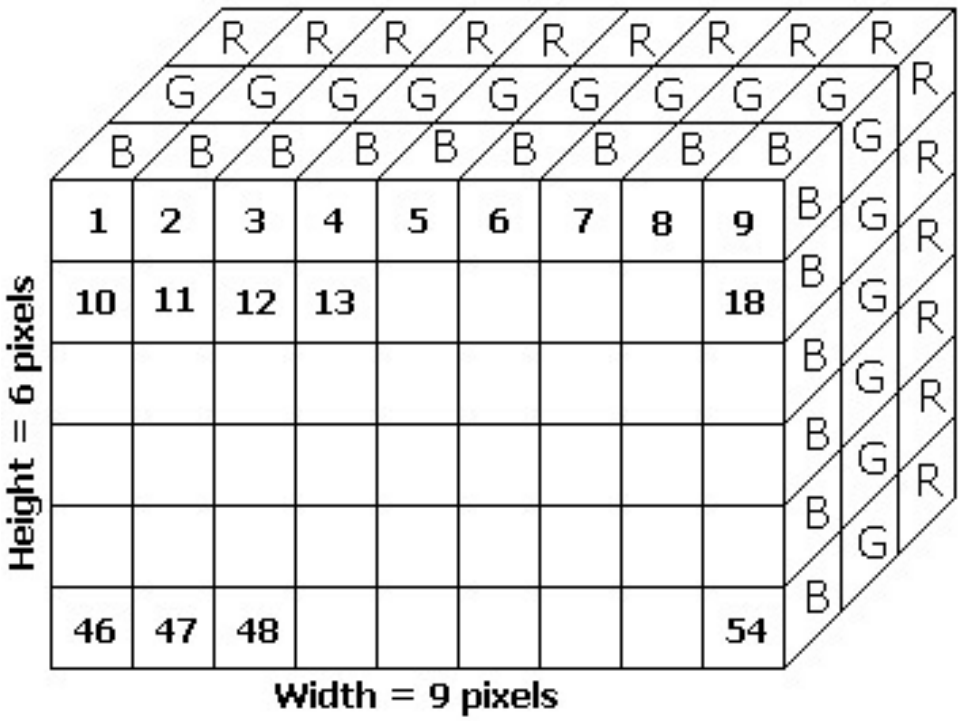


Figure 2.24: OpenCV image storage configuration [1]. OpenCV accesses pixel information from an image using a single or multiple level matrix.

for efficient modification and analysis of the images in other ways. By modifying the values of the pixels, images can be sharpened, smoothed, darken, lightened, and more.

Storage of an image in the OpenCV library is done with the use of a data structure. The structure is referenced as IplImage, originally defined as part of Intel's Image Processing Library (the Ipl in IplImage) [3]. This structure is shown in Listing 2.3.

The actual memory storage locations of the pixels of an image are contained in the imageData portion of the IplImage (a pointer to the first pixel of image data). The other variables of the structure provide useful information about the image, most of which are intuitive. Values in some of the variables determine the most efficient memory allocation and storage. A black and white image uses fewer layers of the storage matrix and generally less memory than a color image.

The images accessed by OpenCV are referenced in the two-axis $CVimage$-frame. The origin of the frame is in the top left corner of the image, with the **X** axis denoting

Listing 2.3: OpenCV image structure

```
typedef struct _IplImage
{
    int       nSize;
    int       ID;
    int       nChannels;
    int       alphaChannel;
    int       depth;
    char      colorModel[4];
    char      channelSeq[4];
    int       dataOrder;
    int       origin;
    int       align;
    int       width;
    int       height;
    struct _IplROI        *roi;
    struct _IplImage      *maskROI;
    void*     imageId;
    struct _IplTileInfo   *tileInfo;
    int       imageSize;
    char*     imageData;
    int       widthStep;
    int       BorderMode[4];
    int       BorderConst[4];
    char*     imageDataOrigin;
}
IplImage;
```

horizontal pixel location to the right and the **Y** axis denoting vertical pixel location below. The first pixel in the top left corner is at position [0, 0] in the image.

The benefits of the OpenCV image structure are evident in the speed at which the library processes images. Some of the useful functions used by this research are presented in the next portions of this section.

2.2.2.1 OpenCV Find Contour. As the name of the function implies, cvFindContours( ) accomplishes just that. A collection of points found in an image that somehow appear connected, such as a line or curve, define a *contour*. Often

an outline of an object or identifiable marks on the object define these curves. This function searches through an image and returns a collection of contours.

As an example, the code snippet in Listing 2.4 demonstrates the use of cvFindContours( ). The snippet takes an IplImage (defined as image), thresholds the image (effectively reduces the effects of strong illumination and reflection), finds the contours, and then draws the contours on a new IplImage (contour_image) which can be displayed as needed. The results of this function call can be seen in Figure 2.25.

2.2.2.2 OpenCV Match Template. Another important function that OpenCV provides is a template matching function. This function accepts two IplImages regardless of size. If the images are different sizes, the smaller IplImage is the template, and the larger IplImage is what the template is matched against. If the images are the same size they are simply matched against each other. The symbol $\mathbf{I}_t$ represents the *template* image, $\mathbf{I}_m$ represents the larger image to *match* against. Using a matching algorithm (detailed next), the function systematically compares $\mathbf{I}_t$ with every possible portion of $\mathbf{I}_m$. The function returns to the user a matrix of values that result from the matching. The dimensions of the returned matrix are equal to the difference in dimensions between the two original IplImages plus one. If the two images were the same size, the returned matrix is a single value. As an example, the two images in Figure 2.26 are exactly 100 *pixels* different in size (scaled to fit on the page). The resulting 101×101 matrix would return 10,201 values for every possible location of $\mathbf{I}_t$ in $\mathbf{I}_m$. The symbol $\mathbf{R}$ represents the resulting matrix, and $\mathbf{R}(i, j)$ represents the value in the matrix at the ith-row and the jth-column. A subscript following the $\mathbf{R}$ denotes the method used to compute it (such as $\mathbf{R}_{corr}$, subscript notation matches the methods presented in [3].)

Typically, information gathered from cvMatchTemplate( ) is the location in the $\mathbf{I}_m$ where the $\mathbf{I}_t$ most likely matches and the result at that location of the chosen matching function. The returned locations of the match are referenced in the

Listing 2.4: OpenCV cvFindContours( ) function pseudocode

```
    // An OpenCV IplImage already exists named image
IplImage contour_image (width x height) // Create an OpenCV image
Memory g_storage (width x height x 3)   // Create contour storage

    // Threshold the image and place the results in contour_image
cvAdaptiveThreshold(image, contour_image)

    // Find contours and place in storage
cvFindContours(contour_image, g_storage)

    // Clear the created image
cvZero(contour_image)

    // Draw the contours on contour_image
cvDrawContours(contour_image, g_storage)
```

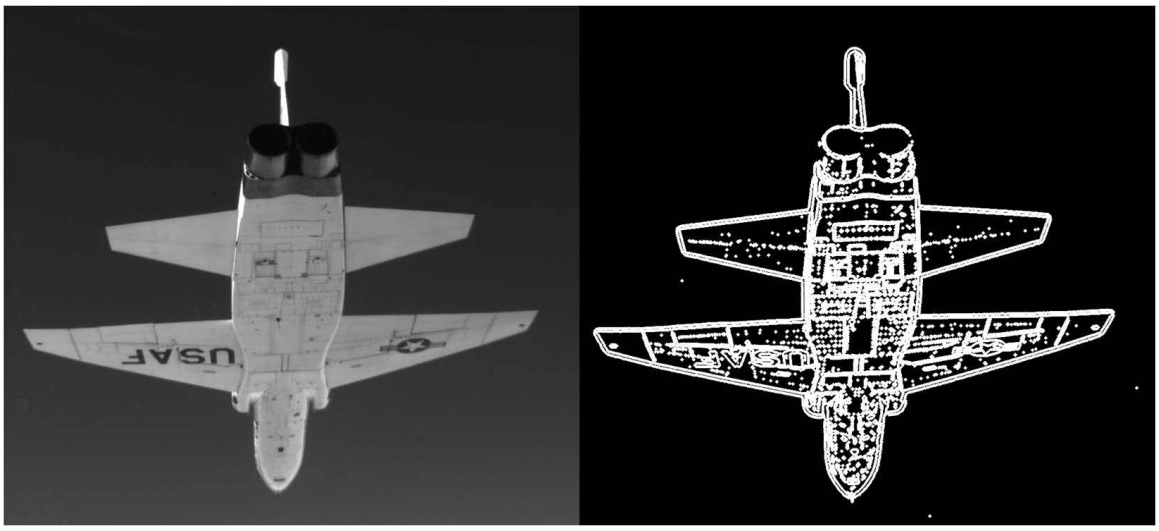


Figure 2.25: OpenCV cvFindContours( ) function. This function finds the contours of an image (left side), and in this application of it, places the contours in another image (right side).

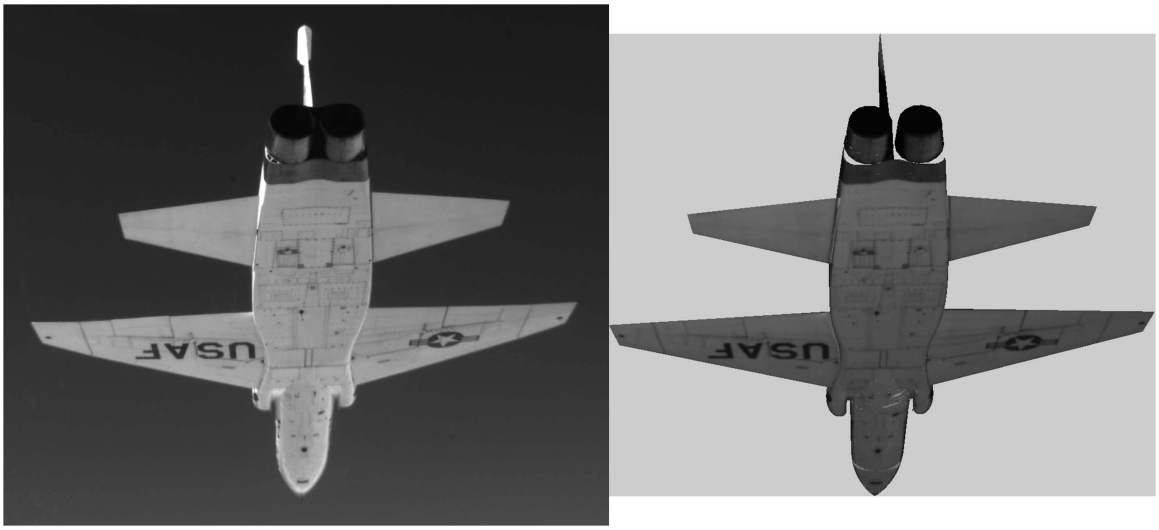


Figure 2.26: OpenCV cvMatchTemplate function. This function matches the template on the right side $\mathbf{I}_t$ with the image on the left $\mathbf{I}_m$.

CVimage-frame. The symbol **r** denotes a two by one position vector of the location of the most likely match.

There are six different matching functions available in cvMatchTemplate( ) [3]; however, three of them are normalized versions of the other three. These normalized versions help reduce the effects of lighting and shading on the images. Two of the six were used in this research, both of which were normalized and the details of those two are presented. The following nomenclature is used in this section: W_t and H_t are the width and height of the template, W_m and H_m are the width and height of the image to match against. x and y represent the translations of specific pixels in $\mathbf{R}$, and x', y' and x'', y'' represent the translations of specific pixels in $\mathbf{I}_t$ (referenced twice in one equation). All the translations are referenced in the *CVimage*-frame.

The first matching method is the *correlation matching method* [3]. This method determines a match by multiplying the values in the images together and then squaring them. This is similar to a sum squared difference, with a multiplication instead of a difference. A perfect match will be large, and bad matches will be small or zero:

$$\mathbf{R}_{corr}(x,\ y) = \sum_{x',\ y'} \left(\mathbf{I}_t(x',\ y') \cdot \mathbf{I}_m(x + x',\ y + y')\right)^2 \tag{2.40}$$

The second method is the *correlation coefficient matching* method [3]. This method matches the $\mathbf{I}_t$ relative to its mean against the $\mathbf{I}_m$ relative to its mean. A perfect match would be one and a perfect mismatch would be negative one. The process determines the mean value of $\mathbf{I}_t$ and $\mathbf{I}_m$ and subtracts the respective mean from the pixel values of each image. This creates new images denoted as $\mathbf{I}_t^{'}$ and $\mathbf{I}_m^{'}$, detail in the following equations:

$$\mathbf{I}_t^{'}(x',\ y') = \mathbf{I}_t(x',\ y') - \frac{1}{(W_t \cdot H_t) \sum_{x'',\ y''} \mathbf{I}_t(x'',\ y'')} \tag{2.41}$$

$$\mathbf{I}_m^{'}(x + x',\ y + y') = \mathbf{I}_m(x + x',\ y + y') - \frac{1}{(W_m \cdot H_m) \sum_{x'',\ y''} \mathbf{I}_m(x + x'',\ y + y'')} \tag{2.42}$$

These intermediate images are then multiplied, squared, and summed in the same manner as the first method to provide a value between negative one and positive one for each location in the $\mathbf{R}$ matrix:

$$\mathbf{R}_{coeff}(x,\ y) = \sum_{x',\ y'} \left(\mathbf{I}_t^{'}(x',\ y') \cdot \mathbf{I}_m^{'}(x + x',\ y + y') \right)^2 \tag{2.43}$$

The normalized versions of these matching methods divide the resulting matrix by a normalized coefficient. The resulting $\mathbf{R}$s are shown:

$$\mathbf{R}_{corr\_normed}(x,\ y) = \frac{\mathbf{R}_{corr}(x,\ y)}{\sqrt{\sum_{x',\ y'} \left(\mathbf{I}_t(x',\ y')\right)^2 \cdot \sum_{x',\ y'} \left(\mathbf{I}_m(x + x',\ y + y')\right)^2}} \tag{2.44}$$

$$\mathbf{R}_{coeff\_normed}(x,\ y) = \frac{\mathbf{R}_{coeff}(x,\ y)}{\sqrt{\sum_{x',\ y'} \left(\mathbf{I}_t(x',\ y')\right)^2 \cdot \sum_{x',\ y'} \left(\mathbf{I}_m(x + x',\ y + y')\right)^2}} \tag{2.45}$$

Figure 2.27 shows an example of the correlation coefficient matching method applied to the two sample images shown in Figure 2.26.

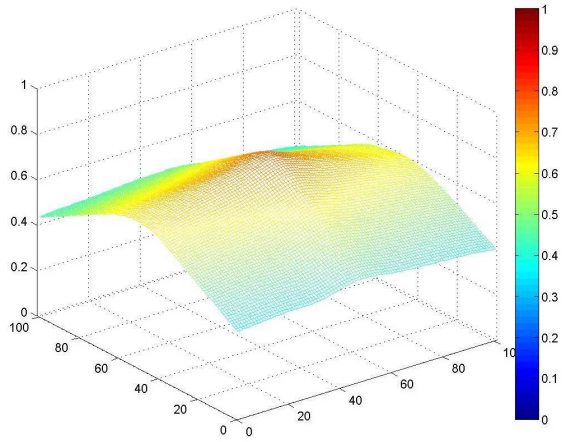


Figure 2.27: OpenCV cvMatchTemplate( ) result. The result of the cvMatchTemplate( ) applied to the images in Figure 2.26 is a matrix of dimensions equal to the difference in dimensions between the $\mathbf{I}_t$ and $\mathbf{I}_m$ plus one. The negated values of the matrix, from the correlation coefficient matching method, are shown here as height in the plot.

The values of $\mathbf{R}_{coeff\_normed}(x,\ y)$ are plotted as negative heights, the images are closer to exact opposites (values close to negative one). From both Figures (2.26 and 2.27), it appears that the best match (or best mismatch) of the two images is near the middle of $\mathbf{I}_m$. The next section introduces the interaction between the libraries.

2.2.3 OpenGL to OpenCV. The benefits of both libraries are evident as they were designed to be efficient at what they do best. Fortunately, both can accomplish portions of the other's capabilities when needed, but not as efficiently as the other. To utilize the rendering power of OpenGL in combination with the image manipulation and comparison power of OpenCV it is necessary to pass information between them. In this research, the interaction between the two libraries was limited to passing images. It was required that images rendered by OpenGL be compared with images collected from the camera. To make the comparison, the OpenGL images had to be converted to OpenCV images. Because of the different storage methods of the two libraries, and the lack of available applications requiring both OpenGL and OpenCV, an open-source conversion process was not available.

According to [1], a possible solution requires accessing the color value of each pixel in the OpenGL image and arranging it in the proper sequence of a blank OpenCV image of the same size, as shown in Listing 2.5. In this example, an OpenGL image is rendered to the screen (not shown in the listing), and an OpenCV image (CVimage) is created and memory allocated with the same memory size as the OpenGL image. The OpenGL image is then *read* into the memory location with the glReadPixels( ) function. The for loop cycles through all the pixels of the OpenGL image, now stored in the memory location GLimage, and copies them to the correct location in CVimage. The memory location GLimage(0) would access the *R* component of the first pixel. Similarly, the CVimage−>imageData (0,0,0) variable would point to the memory location of the first pixel of the initially blank OpenCV image.

The pixel transfer does not account for the difference in first pixel location between an OpenGL and OpenCV image, a flip of the image is accomplished with cvFlip( ). The other variables in the listing are used to systematically move through

Listing 2.5: OpenGL to OpenCV image conversion pseudocode [1]

```
IplImage CVimage (width x height)       // Create an OpenCV image
Memory   GLimage (width x height x 3)   // Create memory storage
glReadPixels(width, height, GLimage)    // Screen image into storage
wIndex, hIndex = 0
// wIndex is the width index, hIndex is the height index

for i from 0 to width*height*3 by 3
// Cycle through all the pixels in the image
    IF wIndex >= width
    wIndex = 0
    hIndex = hIndex + 1

// Place the pixels of GLimage into CVimage
CVimage->imageData(hIndex, wIndex, 0) = GLimage(i+2) // B
CVimage->imageData(hIndex, wIndex, 1) = GLimage(i+1) // G
CVimage->imageData(hIndex, wIndex, 2) = GLimage(i+0) // R

wIndex = wIndex + 1
end
cvFlip(CVimage);  // Account for different origins
```

all the pixels in both images.

This is the initial method of image conversion between programming libraries used in this research. This concludes the background section on programming, some of the areas are revisited in Chapter 5. To finalize this chapter, a introduction to Kalman filtering is presented.

2.3 Kalman Filtering

Kalman filtering is a statistically based method to update, propagate and estimate the state (mean and uncertainty) information for a system with noise. By assuming a statistical knowledge of the noise and a model of the system process, the filter estimates state information that tends to be closer to true values than a system without a Kalman filter. The filter also assumes some knowledge of the available measurements, how they relate to the process, and their accuracy. The research presented in this thesis made use of a linear Kalman filter. As an introduction, the following terms are defined:

- $\mathbf{x} \rightarrow$ state vector (nx1 vector)
- $\dot{\mathbf{x}} \rightarrow$ derivative of the state vector (nx1 vector)
- $\mathbf{F} \rightarrow$ homogeneous, continuous-time, system-dynamics matrix (nxn matrix)
- $\Phi \rightarrow$ discrete-time state transition matrix (nxn matrix)
- $\mathbf{B} \rightarrow$ input matrix (nxb matrix)
- $\mathbf{u} \rightarrow$ input (bx1 vector)
- $\mathbf{G} \rightarrow$ noise transformation matrix (nxq matrix)
- $\mathbf{w} \rightarrow$ white noise processes (qx1 vector)
- $\mathbf{Q} \rightarrow$ process covariance (qxq matrix)
- $\mathbf{z} \rightarrow$ measurement vector (mx1 vector)
- $\mathbf{H} \rightarrow$ observation matrix (mxn matrix)

- $\mathbf{v} \rightarrow$ white noise processes (mx1 vector)
- $\mathbf{R} \rightarrow$ measurement covariance (mxm matrix)
- $\mathbf{P} \rightarrow$ state covariance (nxn matrix)
- $\mathbf{K} \rightarrow$ Kalman gain (nxm matrix)

In the introduction of terms, n is the number of states to be tracked, m is the number of measurements available, b is the number of inputs into the system, and q is the number of noise sources in the system model. A subscript k after the above values denotes a discretized version of the term. The noise processes are white, Gaussian, zero-mean noise (WGN) processes. A white noise source has constant power across all frequencies, the WGN is a random process with a mean of zero and standard deviation from zero of σ.

In a Kalman filter, the states are represented as random variables characterized by a Gaussian distribution. Gaussian distributions allow the filter to characterize the state with only two values, their mean ($\mathbf{x}$) and their covariance ($\mathbf{P}$). The state mean is the filter's estimate of the true value of the state, while the covariance is representative of the uncertainty in that estimate.

Without any noise in the system, and no inputs, the basic relationship between $\dot{\mathbf{x}}$ and $\mathbf{x}$ is the $\mathbf{F}$ matrix as expressed in the difference equation $\dot{\mathbf{x}} = \mathbf{F}\mathbf{x}$. Additionally, if deterministic external influences exist in the form of inputs into the system, they relate to $\dot{\mathbf{x}}$ by the $\mathbf{B}$ matrix, or $\dot{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{B}\mathbf{u}$. This relationship assumes zero noise in the system; therefore, this relationship does not introduce uncertainty. With a determined initial condition, this process model would know the state at any instant in time without any uncertainty.

Since most systems have noise of some intensity, the Kalman filter characterizes the noise in the process as a WGN process $\mathbf{w}(t)$ with covariance $\mathbf{Q}(t)$, defined as:

$$\mathrm{E}\{\mathbf{w}(t)\mathbf{w}^{\mathbf{T}}(t+\tau)\} = \mathbf{Q}(t)\delta(\tau) \tag{2.46}$$

where $\delta(\tau)$ is the Dirac delta function, and $\mathrm{E}\{\cdot\}$ is the expectation operator. This represents zero-correlation in time of the noise source; the noise value at any instant is not dependent on the noise value at any other time.

The basic Kalman filter denotes the relationship between these terms in a continuous-time, stochastic differential equation:

$$\dot{\mathbf{x}}(t) = \mathbf{F}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) + \mathbf{G}\mathbf{w}(t) \tag{2.47}$$

This equation models the system dynamics in continuous time. The WGN processes in the model do not actually introduce noise into the states, rather they characterize the noise that is already present as a function of the process.

It is more common, and necessary when implementing in digital computers, to represent a model in discrete-time for implementation in a Kalman filter, such that Equation (2.47) is represented as:

$$\mathbf{x}_k = \Phi_{k-1}\mathbf{x}_{k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \tag{2.48}$$

where k is an instant in time, $k-1$ is one instant before k, and Φ_{k-1} is the state transition matrix for time $k-1$. The state transition matrix is a function of $\mathbf{F}$ and the sampling interval (Δt) of the process, such that:

$$\Phi(\Delta t) = e^{\mathbf{F}\Delta t} \tag{2.49}$$

Implementing this system into a Kalman filter is possible without measurement updates. The filter tracks the mean and covariance of the state, $\mathbf{x}$ and $\mathbf{P}$. The Kalman filter makes the following predictions at every sampling interval:

$$\hat{\mathbf{x}}_k^- = \Phi_{k-1}\hat{\mathbf{x}}_{k-1}^+ + \mathbf{B}_{k-1}\mathbf{u}_{k-1} \tag{2.50}$$

$$\mathbf{P}_k^- = \Phi_{k-1}\mathbf{P}_{k-1}^+\Phi_{k-1}^{\mathbf{T}} + \mathbf{Q}_{k-1} \tag{2.51}$$

where $\hat{x}_k^-$ and $\mathbf{P}_k^-$ indicate the estimation and uncertainty immediately before time k (*a priori*), while $\hat{\mathbf{x}}_{k-1}^+$ and $\mathbf{P}_{k-1}^+$ indicates the estimation and uncertainty immediately after the previous time $k-1$ (only referenced as an *a posteriori* estimate if a measurement update was incorporated.) This is the propagation step of the filter, as time progresses, the uncertainty in the state estimation increases ($\mathbf{P}$ increases).

Decreasing the uncertainty in the state estimation requires the inclusion of measurements into the Kalman filter. The discrete measurement process is modeled:

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{2.52}$$

where $\mathbf{v}$ is a WGN process with covariance $\mathbf{R}(t)$:

$$\mathrm{E}\{\mathbf{v}(t)\mathbf{v}^{\mathbf{T}}(t+\tau)\} = \mathbf{R}(t)\delta(\tau) \tag{2.53}$$

The key to the Kalman filter updating $\mathbf{x}$ and $\mathbf{P}$ with measurement information is determining the Kalman gain at the current time:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^{\mathbf{T}} \left(\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^{\mathbf{T}} + \mathbf{R}_k \right)^{-1} \tag{2.54}$$

The Kalman gain is based on the current covariance of the system, $\mathbf{P}$, and the covariance of the measurement, $\mathbf{R}$. Based on those values the Kalman filter updates the estimate of $\mathbf{x}$ and its uncertainty, $\mathbf{P}$, with the measurement update

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k \left(\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_k^- \right) \tag{2.55}$$

$$\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^- \tag{2.56}$$

where $\mathbf{I}$ is an identity matrix of size $n \times n$. This is the update step of the filter.

These are the basic relationships of the linear Kalman filter. The process continually repeats as needed: propagate then update. Every system is unique, some include measurements at every sampling time, others only have access to measure-

ments periodically and continually propagate until a measurement is available. This general set of equations can be tailored to many different situations and adapted to more advanced filters (the extended Kalman filter and unscented Kalman filter are both based on these basic equations).

To implement the Kalman filter, an analysis of the system is needed to gain some knowledge of the noise in the system process and the measurements. This, in addition to a process and measurement model of the system, permits the Kalman filter to estimate the state information better than not using a filter at all.

This chapter has presented the background needed for this thesis. The following chapter adapts the information presented here to the AAR problem outlined in Chapter 1.

III. The Nature of Air Refueling

The general nature of air refueling is dynamic and the interactions between aircraft are non-deterministic. The dynamics of flight, compounded by aerodynamic influences between aircraft, unpredictable atmospheric, and environment conditions, variations in lighting, and partially predictable human responses, all make AR a stochastic process. In AR, the human operator innately understands the balance between the possible and the probable. He applies this knowledge to render accurate and timely decisions that impart motions to the aircraft based on their estimations of current conditions.

The best way to portray this information to an autonomous system is with models. By modeling the dynamics of a tanker aircraft, the autonomous solution can apply some of the same innate knowledge in an attempt to match the effectiveness of its human counterpart.

This chapter has two sections: the first details some key AR assumptions applied to the development of a process model and the second presents the AR positions and the information needed to determine them for autonomous operations and render them as images in OpenGL. This chapter assumes two aircraft in formation, a lead aircraft and a wing aircraft, and no knowledge of the lead aircraft's position or attitude.

3.1 Air Refueling Dynamics

In a very basic sense, the number of degrees of freedom (DOF) of two individual aircraft is twelve. Each aircraft has six DOF, three translations from a coordinate system origin, and three attitudes with respect to that coordinate system. By placing the reference coordinate system on one of the aircraft, three DOF are eliminated because the aircraft with the coordinate system has a translation of zero in all three axes. Using the b-frame as the coordinate system, the attitude information is zero in all three axes as well. Using the n-frame as the coordinate system, the attitude information is not zero, but the system has access to the attitude of the aircraft with

an INS. Using either frame reduces the total unknown DOFs to six: the attitude of the lead aircraft and the translation of the lead aircraft from the wing aircraft.

The first portion of this section presents the theoretical and empirical motions of a simulated tanker aircraft in these six DOF and how they relate to AR. The second portion builds on this knowledge to create a model of the lead aircraft's dynamics.

3.1.1 Air Refueling Assumptions. Of these six DOF, a few have less variation during air refueling. The first, and most identifiable, is the lead aircraft's yaw attitude, or heading. A typical refueling consists of straight tracks, with very small heading changes and turning tracks with constant roll-angle turns and predictable heading changes. During the straight tracks, verification of the lead aircraft's heading is not required very often.

During turns, verification of the lead aircraft's heading is accomplished more often. With a good estimate of the roll attitude of the lead aircraft, a change in heading (or yaw rate, $\dot{\psi}$) can be predicted [29]. An aircraft's turn rate, often related to standard rate ($3°/second$) or half standard rate ($1.5°/second$), is dependent on the bank angle and true airspeed of the aircraft, V_T:

$$\dot{\psi} = \frac{g\sqrt{\frac{1}{(\cos\phi)^2} - 1}}{V_T} = \frac{g\tan\phi}{V_T}(^{rad}/_{sec}) \tag{3.1}$$

where g is the acceleration due to gravity. This relationship assumes coordinated flight and can be used to update the estimated heading of the lead aircraft, at Δt sampling times, through the following relationship:

$$\psi_k = \psi_{k-1} + \Delta\psi \tag{3.2}$$

where, $\Delta\psi$ is the amount of change between sampling times, such that $\Delta\psi = \dot{\psi} \cdot \Delta t$. With this equation as a predictor of heading, only a slightly higher verification rate of heading is required during turning tracks, as compared to straight tracks.

Empirical truth data, representative of tanker maneuvers, verify the yaw motion of a lead aircraft during AR and the estimation of the heading from Equation (3.2). During the data collection for this thesis the lead aircraft flew these maneuvers up to an approximately 30^o roll angle. Data were collected at 100Hz, and evaluated at 0.1 *second* intervals (Δt). The data collected was over a 250-*second* time segment during which operationally representative refueling maneuvers were performed by the lead and wing aircraft. Further details on data collection are presented in Chapter 5.

The data is shown in Figure 3.1 and demonstrates two things: the benign heading changes of the aircraft and the accuracy of the heading predictions. The aircraft's heading with respect to the n-frame is shown as a continuous line, referenced to the left axis, and the maximum change in heading between Δt is shown in small circles, referenced to the right axis. Additionally, the estimated heading is shown as a dotted line also referenced to the left axis.

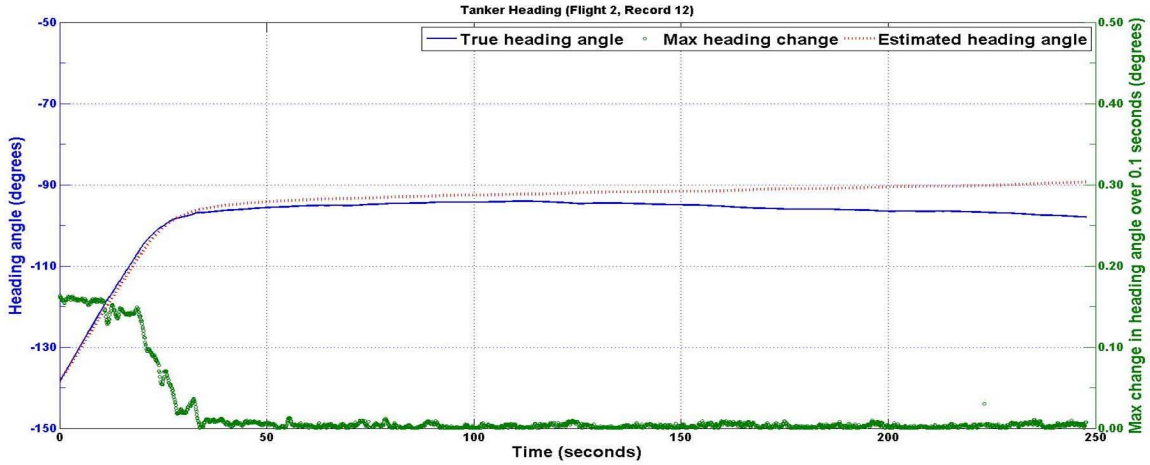


Figure 3.1: Lead aircraft ψ (solid line), $\Delta\psi$ (small circles), and estimated heading (dotted line). Heading, change in heading, and estimated heading during a representative profile of a tanker aircraft during a rendezvous maneuver, including an initial roll angle to roll-out. The max change values are presented as absolute values.

The change in heading is shown as absolute values. The time period of interest includes a 33° roll angle turning track in the first 25 *seconds* leading to a roll-out on heading, followed by a straight track. The maneuvers were hand flown by experienced

pilots and the actual heading of a tanker aircraft will vary depending on the use of an autopilot and pilot skill. Based on 10 years of AR experience, it is the author's opinion that these maneuvers represented actual refueling maneuvers.

Data analysis of this figure determined that the change in heading is approximately 0.02° per Δt ($0.2^\circ/second$) during the straight track. During the turning track the change in heading is approximately 0.15° per Δt ($1.5^\circ/second$), or half standard rate. These values will change with differing bank angles and true airspeeds flown. Additionally, the estimation of heading initially does a good estimation over short durations. Later in the data run, without updates, the estimation drifts away from the actual true heading.

In conclusion, Figure 3.1 demonstrates that ψ verification is not required often, especially during a straight track, and Equation (3.2) predicts ψ as a function of ϕ even during turning tracks. If verifications were accomplished every two to three *seconds*, the lead aircraft's heading change would be less then 0.5° during that time.

Another low-variation DOF, is the lead aircraft's pitch attitude. Tanker aircraft attempt to maintain a constant altitude during AR. A constant altitude requires small periodic motion in the pitch attitude causing a direct influence on the velocity in the down axis in the n-frame. In Figure 3.2 the lead aircraft's pitch attitude, with respect to the n-frame, during the same time period in flight as Figure 3.1, is shown with similar markings.

The tanker pitch range during this entire time period is 2° and the largest change in pitch was 0.04° per Δt ($0.4^\circ/second$). Even during the turn accomplished during the first 25 *seconds*, the pitch did not change dramatically. As an example, two aircraft, one with 7.5° and one with 5.5° pitch with respect to the n-frame, as seen from the *cam*-frame of a wing aircraft, are shown in Figure 3.3. Typically, a human operator is not able to recognize this subtle a difference over the time spans of interest. As a result of these AR assumptions and the empirical data, this thesis

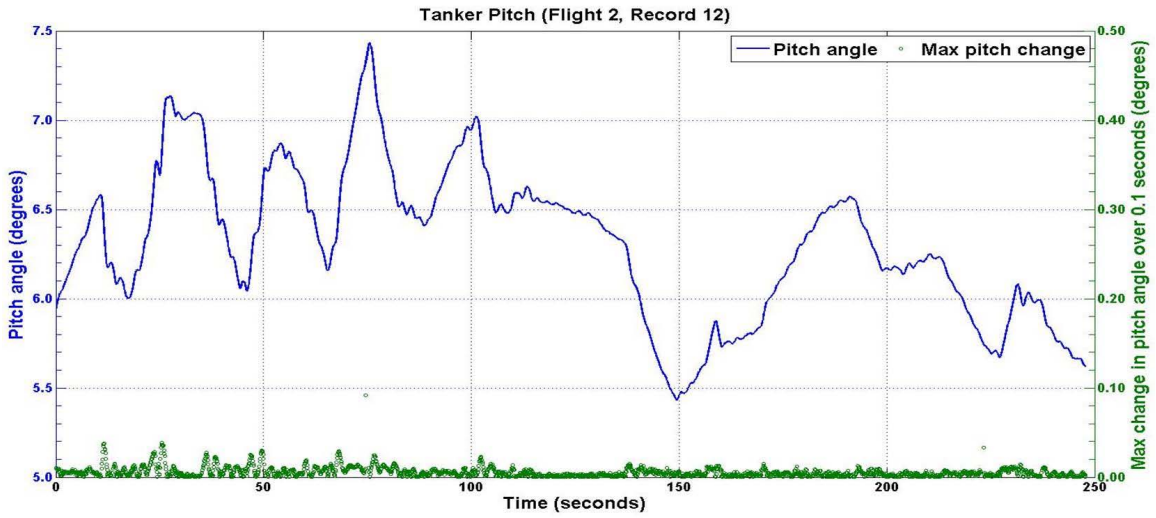


Figure 3.2: Lead aircraft θ (solid line) and $\Delta\theta$ (small circles). Pitch and change in pitch during a representative profile of a tanker aircraft during a rendezvous maneuver, including an initial roll angle to roll-out. The max change values are presented as absolute values.

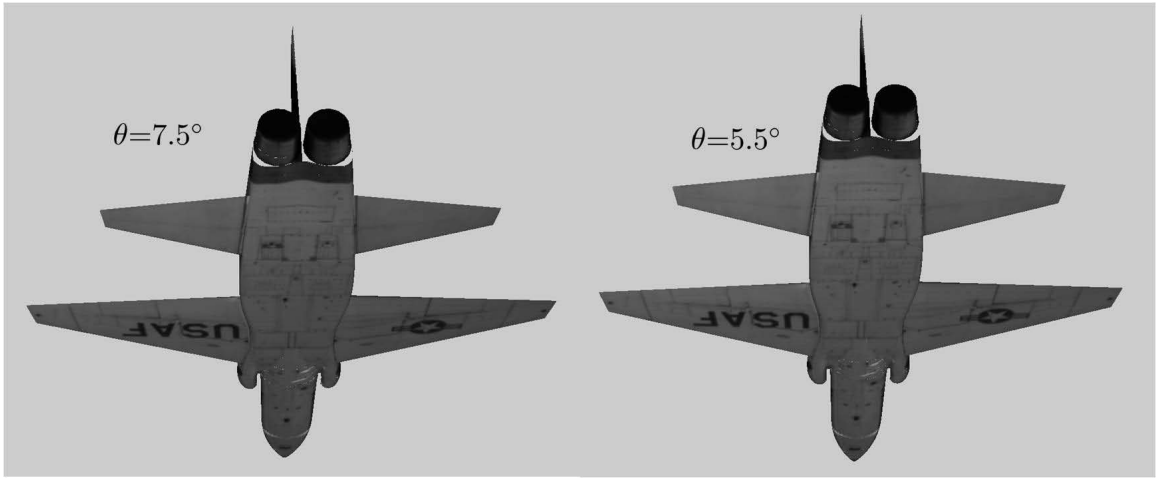


Figure 3.3: Lead aircraft at $\theta = 7.5°$ and $\theta = 5.5°$. The visual difference in an aircraft's pitch as viewed from a camera below, looking up at approximately 30°. The image on the left is at $\theta = 7.5°$, the right is at $\theta = 5.5°$, both with respect to the n-frame.

assumes that θ is constant and ψ is a function of ϕ, and thereby verifies these DOFs less frequently than the other DOFs.

The same run from the previous figures is shown depicting the down velocity of the lead aircraft with respect to the n-frame in Figure 3.4, as a direct result of the pitch inputs by the pilot (witnessed in the similarity between the peaks and values of the two Figures 3.2 and 3.4). The lead aircraft was generally within 10 *feet per second* (fps) of holding constant altitude (or zero fps). Additionally, the change in down velocity was typically less than 0.2 fps per Δt. The relationship between down velocity and θ is non-linear, but over short periods a linear approximation can be made. The algorithm presented in this thesis does not leverage this relationship; however, by accurately predicting one of the two DOFs an estimate of the other is possible, in a similar manner to the $\dot{\psi}$ and ϕ relationship shown earlier.

Of the two remaining translation DOFs, north and east velocities, the same data is shown for east velocity only in Figure 3.5. Both are similar, and the difference between actual values depends only on current heading, ψ. These two DOFs have

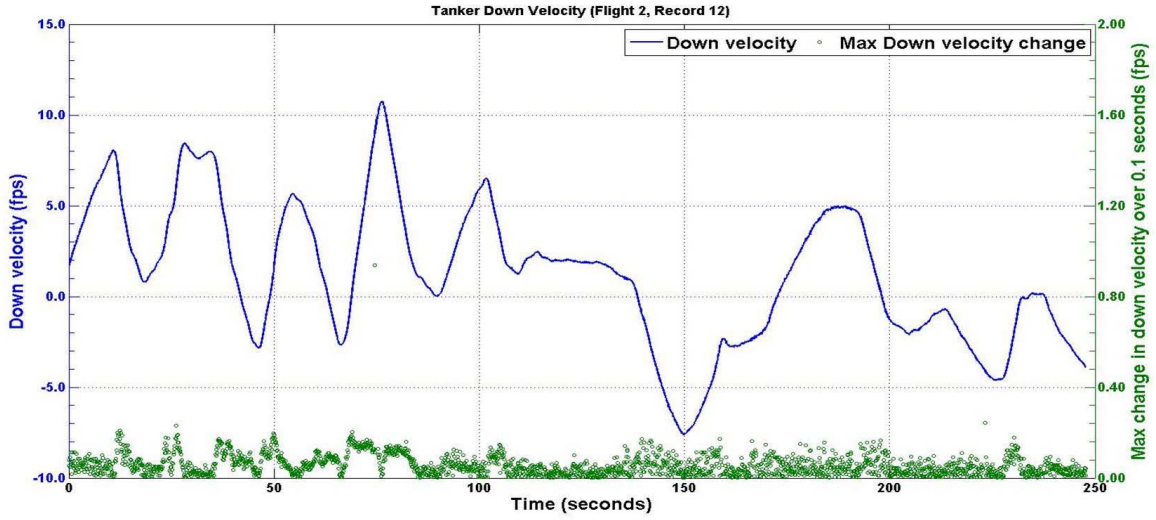


Figure 3.4: Lead aircraft V_{down} (solid line) and ΔV_{down} (small circles). Down velocity and change in down velocity during a representative profile of a tanker aircraft. The max change values are presented as absolute values.

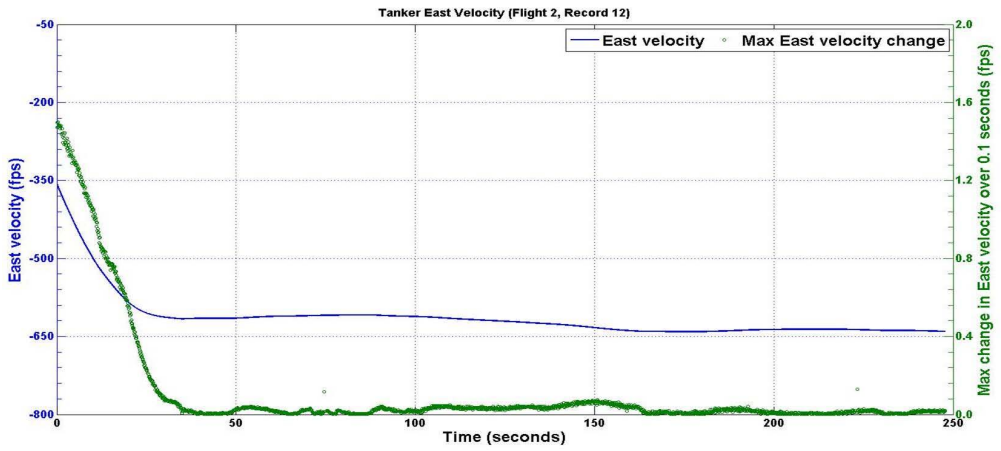


Figure 3.5: Lead aircraft V_{east} (solid line) and ΔV_{east} (small circles). East velocity and change in east velocity during a representative profile of a tanker aircraft. The max change values are presented as absolute values.

larger variation than the down velocity, but together they are constrained by the fairly constant true airspeed of the lead aircraft.

Finally, the most dynamic DOF during AR is roll, shown in Figure 3.6. The majority of the time this attitude stays constant; however, large changes periodically occur that must be verified more frequently. Based on these lead aircraft assumptions and empirical data, the next section presents a dynamic model of the process.

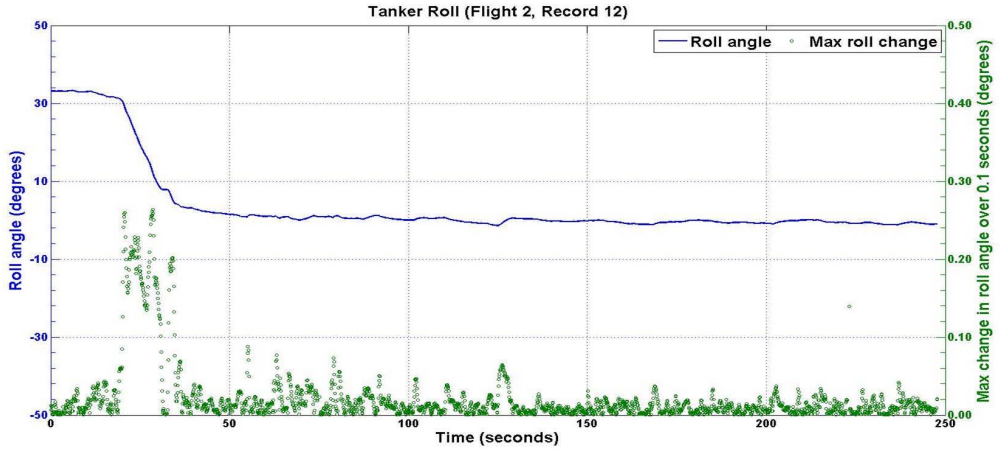


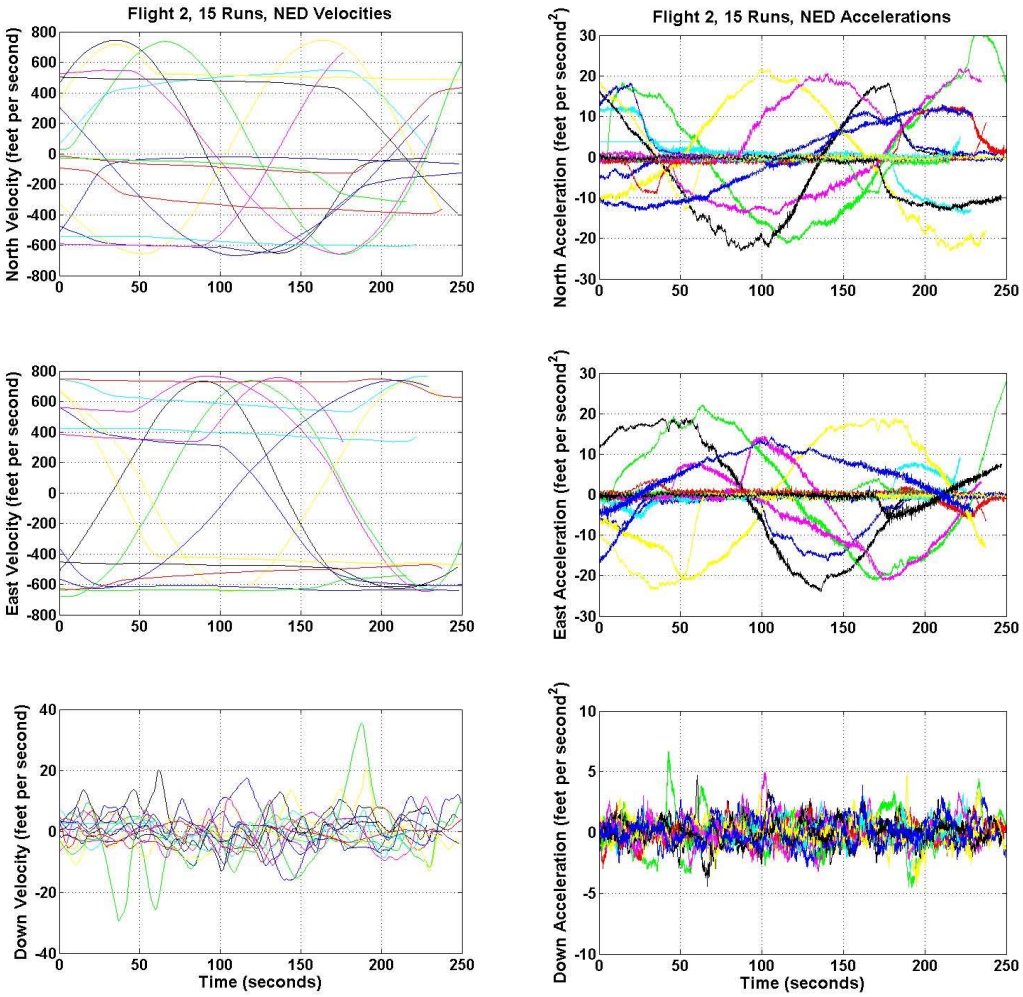
Figure 3.6: Lead aircraft ϕ (solid line) and $\Delta\phi$ (small circles). Roll and change in roll during a representative profile of a tanker aircraft. The max change values are presented as absolute values.

3.1.2 Air Refueling Model. The Kalman filter presented in Section 2.3 requires an estimated model of a system's dynamics. Analyzing the motion of the aircraft determines an approximate model for the filter. This section analyzes the empirical data of a lead aircraft with respect to the n-frame and determines that model. Many of the coordinate frames presented in Chapter 2 are justifiable for this analysis. The n-frame was chosen for its proximity to the aircraft and independence from the motion of the wing aircraft. The created model is based on the dynamics of the lead aircraft in the n-frame and can be tailored to the other frames, based on the knowledge of the wing aircraft's motion.

Analysis of the data in the previous figures along with other similar data runs (15 total), provides the data needed for creation of a model of the lead aircraft in the n-frame. Analysis of the magnitude and frequency characteristics of the data leads to a model that approximates the motion, or potential motion, of the aircraft.

The plots in Figure 3.7 are a culmination of 15 empirical runs. Figure 3.7a shows the velocities in the representative axis. These are the same plots as the previous figures of NED velocity, without the change per Δt shown. Figure 3.7b is a similar plot of the accelerations of the lead aircraft in the n-frame. The acceleration data is more noisy then the velocity. The analysis of this data is shown in Table 3.1.

The north and east velocities show large variation about mean values that are not similar to each other. Down velocity is centered near zero with a low standard deviation. All the accelerations have a near-zero mean. Since the Kalman filter requires the modeling of the noise in the system to be WGN, the accelerations and potentially the down velocity can be modeled as noise sources for the Kalman filter. However, the accelerations for north and east do not actually appear to be *white* noise sources, they appear to have a random, but constrained motion to them, not a characteristic of a white noise source. To help further characterize this motion, the data from the runs were placed in a Power Spectral Density (PSD) plot shown in Figures 3.8 (a) and 3.8 (b).

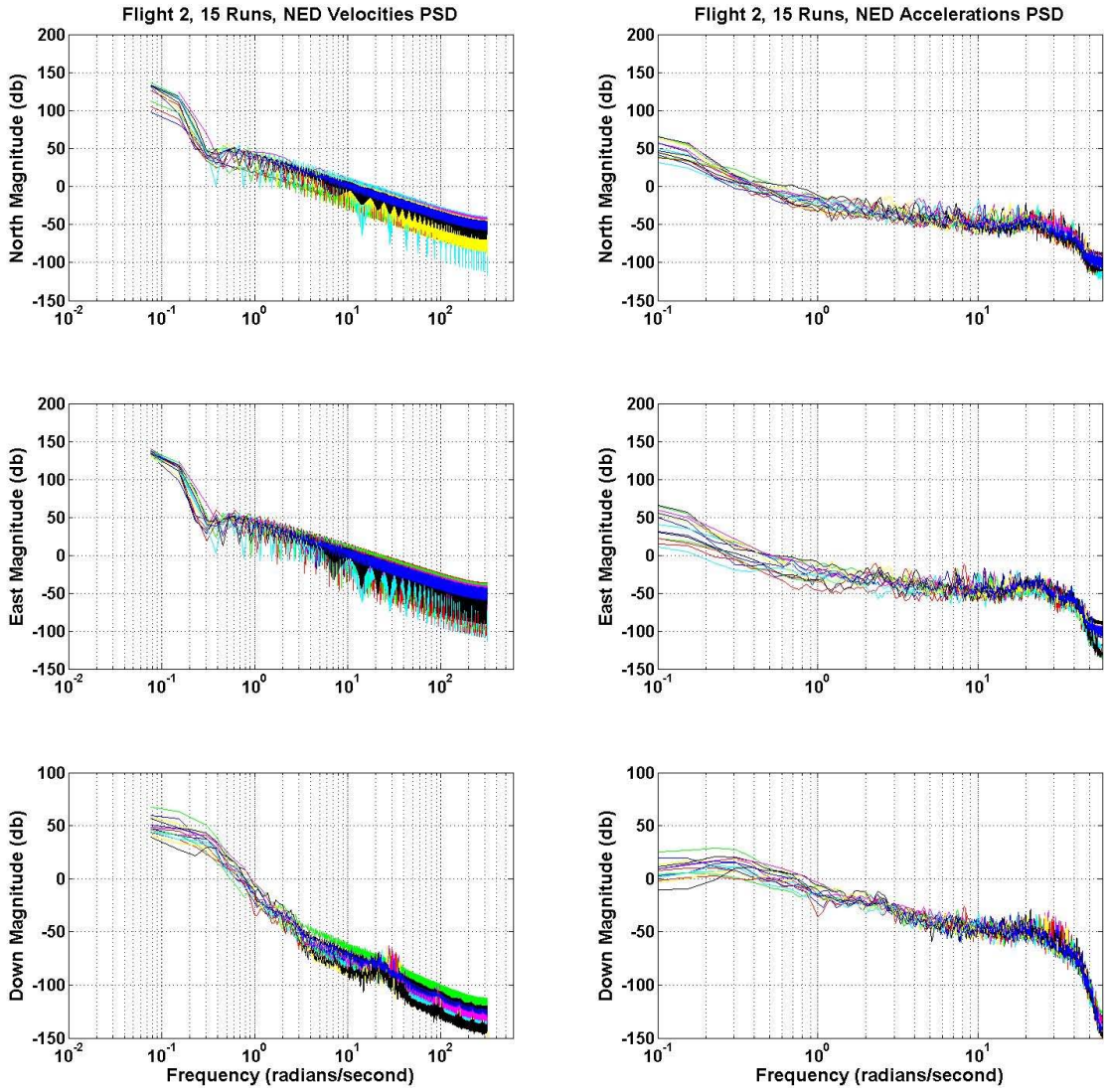


(a) (b)

Figure 3.7: Velocity and acceleration motion of a lead aircraft.
(a) The aircraft's NED velocities for 15 runs; data analysis shown in Table 3.1.
(b) The aircraft's NED accelerations for 15 runs; data analysis shown in Table 3.1.

NED Direction	Mean velocity (fps)	STD velocity (fps)	Mean acceleration (fps^2)	STD acceleration (fps^2)
North	-36.0	186.0	0.3	5.0
East	-6.0	183.0	-0.3	4.0
Down	-0.1	4.0	0.0	0.7

Table 3.1: Analysis of empirical data. STD is the standard deviation of the value. Absolute values greater than one were rounded to the nearest integer.



(a) (b)

Figure 3.8: PSD of the lead-aircraft's velocity and acceleration data. A PSD of the culmination of data runs details the magnitude and frequency of the aircraft's motion.

The velocity PSDs are close to a straight line (similar to a PSD of integration). The acceleration PSDs show a break point common to all three axes around 30 rad/s. This information, coupled with the data from Table 3.1, defines a possible model for the tanker. A first-order Gauss-Markov (FOGM) is a process that can represent the accelerations as a noise source with two values: a time constant and a variance [4]. For a generic acceleration, a, an example FOGM process is shown:

$$\dot{a} = -\frac{1}{T}a + w(t) \tag{3.3}$$

where T is the time constant, and $w(t)$ is a WGN such that:

$$\mathrm{E}\{w(t)w^T(t+\tau)\} = Q(t)\delta(\tau) \tag{3.4}$$

where Q is the variance of the noise. Modeling a system state as a FOGM process represents a time-related statistical limit to the variability of the state. The FOGM introduces the concept of a time-correlated *random walk* best illustrated with a counter example shown in Figure 3.7b. The down acceleration is a process that does not immediately appear to have limited variability in its values. The down acceleration has indiscriminately-sized random motions over short periods of time. In fact, the down acceleration could be modeled directly as a WGN source, with a covariance equal to the standard-deviation, determined from the empirical-data, squared:

$$\dddot{p}_z^n = w(t) \tag{3.5}$$

$$\mathrm{E}\{w(t)w^T(t+\tau)\} = \sigma^2\delta(\tau) \tag{3.6}$$

where $\dddot{p}_z^n$ is the change in down acceleration of lead in the n-frame. This is the equivalent of zero correlation in time between any two values of $\ddot{p}_z^n$. Modeling a state as a noise source in this manner portrays to the Kalman filter that the value of the state will *most likely* (statistically) be anywhere between $\pm\sigma$ at any given instance

in time and the value has no dependance on time; in other words the value at one instance in time is not dependent on the value at any other instance in time.

In a FOGM process, a value in time *is* dependent on other values in time. This is demonstrated in Figure 3.7 (b). Both the north and east accelerations have a time-correlated random walk. The accelerations do not jump to the extremes of the figure, instead they are limited to smaller motions near their previous values. As seen in the figure, over time, they can randomly walk to the extremes of the figure. With a time constant incorporated into the model of the system, the filter assumes a statistically-based limitation to the change in value with respect to time.

Finally, since the PSD of the velocities are similar to an integration, the velocities are modeled as an integration of acceleration and position as a double integration of acceleration:

$$\dot{p}_x^n = \dot{p}_x^n \tag{3.7}$$

$$\ddot{p}_x^n = \ddot{p}_x^n \tag{3.8}$$

$$\dddot{p}_x^n = -\frac{1}{T}\ddot{p}_x^n + w(t) \tag{3.9}$$

$$\begin{bmatrix} \dot{p}_x^n \\ \ddot{p}_x^n \\ \dddot{p}_x^n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -\frac{1}{T} \end{bmatrix} \begin{bmatrix} p_x^n \\ \dot{p}_x^n \\ \ddot{p}_x^n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} w(t) \tag{3.10}$$

where $\dot{p}_x^n$ is the north change in position of lead in the n-frame. Of note, the w(t) noise *is* still a WGN process; it is shaped into a FOGM process with the use of the time constant.

The down acceleration was shown as a process that might be better modeled as a WGN source; however, it has a corner frequency similar to the other accelerations. Additionally, climbs and descents were not accomplished on these runs, those types of motions would be better modeled with a FOGM process as well.

The acceleration PSD plots in Figure 3.8 had a similar corner frequency of approximately 30 rad/s. The time constant of the FOGM is the inverse of the corner frequency, $T = 1/30$. Since there is a non-unity gain shown in the PSD, the following relationship is used for the covariance of the noise:

$$Q = \frac{2\sigma^2}{T} \tag{3.11}$$

where σ is the standard deviation determined in Table 3.1 [4].

As verification, analysis of 30 Monte Carlo runs of this dynamic process model are shown as a PSD of the FOGM in Figure 3.9. Of these 30 runs, the mean acceleration was zero and the standard deviation was five, matching the values from the empirical data. The FOGM has more gain than the empirical data in the one to ten rad/s range. This extra noise, portrays to the filter a dynamic process less accurate than the actual process. The uncertainty in the state estimate will grow faster than the actual accuracy of the process, possibly weighting the measurements more than they should during measurement updates. However, this also portrays to the filter that larger deviations from the state estimate are reasonable, allowing motions, larger than σ, to be accepted with less uncertainty. This extra noise should not be a concern for this process; a higher-order model of the aircraft motion would possibly match the system dynamics better.

Assuming the only difference between north and east velocities is current heading, this FOGM model will be adequate for those two DOFs. The down component shares the same time constant with a lower σ. Finally, a similar analysis was conducted for roll. Unfortunately only the roll attitude was collected empirically. Based upon the empirical data the mean roll angle was 3.5° with a standard deviation of 6.0°. Inferred from empirical data was a mean roll rate of 0.0°/sec with a standard deviation of 0.5°/sec. As a result, the roll rate was modeled as a FOGM as well, but a time constant for the roll was not determined from the empirical data.

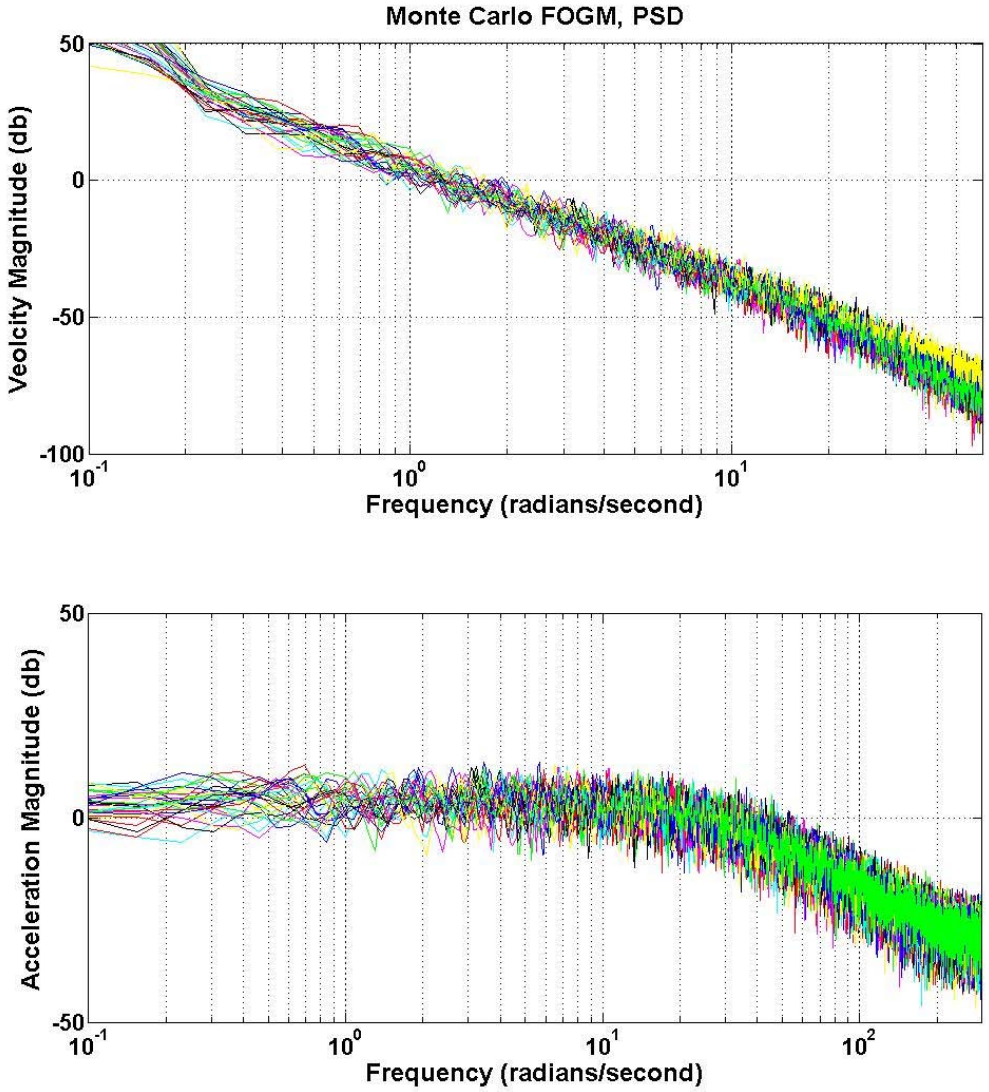


Figure 3.9: PSD of 30 Monte Carlo runs. A model of the dynamics of the tanker approximates the possible motions of the aircraft. The mean acceleration of all the runs was zero, with a standard deviation of five, matching the empirical data.

The following continuous-time process model represents the lead aircraft in the n-frame:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \dddot{x} \\ \dddot{y} \\ \dddot{z} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix}_{11\text{x}1} = \begin{bmatrix} \dot{\mathbf{p}} \\ \ddot{\mathbf{p}} \\ \dddot{\mathbf{p}} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix}_{11\text{x}1} = \mathbf{F}\mathbf{x}(t) + \mathbf{G}\mathbf{w}(t), \quad \mathbf{F} = \left[\begin{array}{c|c|c|c} \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_{3\text{x}2} \\ \hline \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_{3\text{x}2} \\ \hline \mathbf{0}_3 & \mathbf{0}_3 & -\frac{1}{T}\cdot\mathbf{I}_3 & \mathbf{0}_{3\text{x}2} \\ \hline \mathbf{0}_{2\text{x}3} & \mathbf{0}_{2\text{x}3} & \mathbf{0}_{2\text{x}3} & \begin{matrix} 0 & 1 \\ 0 & -\frac{1}{T_\phi} \end{matrix} \end{array}\right]_{11\text{x}11} \tag{3.12}$$

$$\mathbf{G} = \left[\begin{array}{c|c} \mathbf{0}_{6\text{x}3} & \mathbf{0}_{6\text{x}1} \\ \hline \mathbf{I}_3 & \mathbf{0}_{3\text{x}1} \\ \hline \mathbf{0}_{2\text{x}3} & \begin{matrix} 0 \\ 1 \end{matrix} \end{array}\right]_{11\text{x}4} \qquad \mathbf{Q}(t) = \begin{bmatrix} \frac{2\sigma_{ne}^2}{T} & 0 & 0 & 0 \\ 0 & \frac{2\sigma_{ne}^2}{T} & 0 & 0 \\ 0 & 0 & \frac{2\sigma_d^2}{T} & 0 \\ 0 & 0 & 0 & Q_\phi \end{bmatrix}_{4\text{x}4} \tag{3.13}$$

where $\mathbf{0}_M$ is a MxM matrix of zeros, $\mathbf{0}_{M\text{x}N}$ is a MxN matrix of zeros, $\mathbf{I}_M$ is a MxM identity matrix, σ_{ne} is the standard deviation for north and east, σ_d is the standard deviation for down, and T_ϕ and Q_ϕ will be determined through the tuning of the filter. To apply the Kalman filter equations, the matrices are converted to discrete time matrices, to resemble Equation (2.48). This continuous-time, dynamic-process model is valid for tracking a lead aircraft performing tanker-type maneuvers in the n-frame. Accounting for the dynamics of another aircraft updates this model to track in another frame, such as the b_L-frame or *cam*-frame of a wing aircraft. The tracking accomplished in this report, the lead aircraft in the *cam*-frame of the wing aircraft, accounted for the additional motion of the wing aircraft by simply modifying the time constants and σs used in the model.

The dynamics of the lead aircraft, including a model to represent those dynamics has been presented. The next section details the formation positions of AR and the necessary information required to determine those positions without knowledge of the lead aircraft's position or attitude.

3.2 Formation Position

The premise of the algorithm in this thesis is to determine the location of the lead aircraft in the *cam*-frame on a wing aircraft in order to determine where the receiver is in relation to the b_L-frame as presented in Section 2.1.4. A few of the various AR positions are introduced as positions in the b_L-frame to understand the requirements to effect AAR and to quantify the precision needed at the various positions.

For this research, the lead aircraft (a T-38 as a simulated tanker) had a prominent rotating beacon underneath the aircraft that was visible in the images collected by the wing aircraft (an LJ-24 as a simulated receiver). This beacon was located approximately 17.0 *feet* from the nose of the aircraft, or 29.0 *feet* from the tail of the aircraft. The beacon was designated as the origin for both the b_L-frame and n_L-frame, denoted as a cylinder (approximate shape of the beacon) in Figure 3.10.

The wing aircraft had a length of approximately 43.0 *feet* and the coordinate frames, b_W and n_W, were originated at the center of the truth data collection device (for simple determination of errors in the algorithm), denoted as a cube in Figure 3.10. This device was located approximately 23.7 *feet* from the nose of the wing aircraft and one *foot* left of centerline. With a coordinate system defined, aircraft within the formation can navigate with respect to predefined positions in the b_L-frame.

3.2.1 Rendezvous Position. When aircraft are not in visual contact with each other or otherwise knowledgeable about the other aircraft's position, the wing aircraft proceed to and/or maintain a position that provides both lateral and vertical separation between aircraft. This position allows for safe maneuvering until the wing aircraft can attain visual contact with lead and initiate a rejoin or rendezvous. The

definition of this position is 1,000 *feet* below and 1 *nautical mile* (NM) in trail of the lead aircraft. Typically, this is a transient position and held constant only when the lead aircraft is not in sight. The position translations are *no closer than* values (that a receiver aircraft can not proceed closer than without visual contact with the tanker). In the b_L-frame, this position can be represented as a position vector $\mathbf{p}_{1NM}^{b_L}$ with the following coordinates in *feet*:

$$\mathbf{p}_{1NM}^{b_L} = \begin{bmatrix} -6076 & 0 & 1000 \end{bmatrix}^{\mathbf{T}} \tag{3.14}$$

At this distance, the reference frame origin locations of each aircraft are not a significant influence on this position.

3.2.2 Pre-Contact Position. After passing the rendezvous position with the lead aircraft in sight, the wing aircraft proceeds to a position known as pre-contact. The wing aircraft reduces the separation between the aircraft both vertically and horizontally in a straightforward maneuver. The wing aircraft initiates the maneuver with 20 *knots* greater airspeed than the lead aircraft and reduces the closure rate to zero as a climb to lead's altitude is accomplished. The definition of this position is 50 *feet* aft of lead and slightly below. The position is dependent on the size of the aircraft and where the appropriate coordination frames are located; 50 *feet* describes the distance from the nose of the wing aircraft to the tail of the lead aircraft.

The down component is approximated by using a 30° aspect angle measured from the b_L-frame's negative $\mathbf{X}$ axis in the direction of its positive $\mathbf{Z}$ axis. This position can be represented as a position vector $\mathbf{p}_{PRE}^{b_L}$ with the following coordinates:

$$\mathbf{p}_{PRE}^{b_L} = \begin{bmatrix} -96 & -1 & 25 \end{bmatrix}^{\mathbf{T}} \tag{3.15}$$

At this distance the origin locations are accounted for in the position definition. The negative one *foot* offset in the $\mathbf{Y}$ axis of the b_L-frame accounts for the off-centerline location of the two coordinate frames' origin on the wing aircraft.

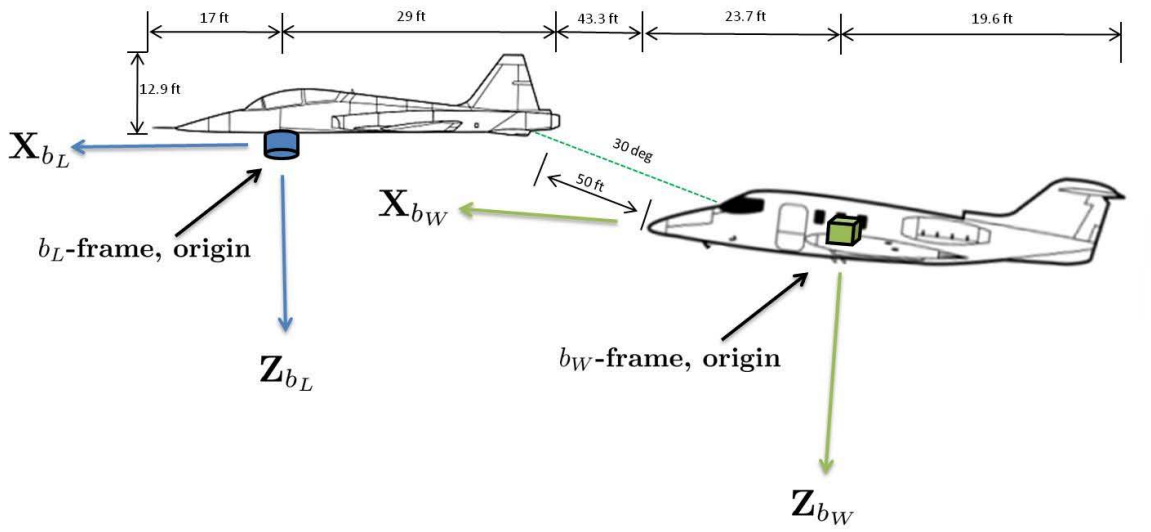


Figure 3.10: AAR coordinate system. The coordinate systems used for AAR, demonstrated in the pre-contact position ($\mathbf{p}^{b_L}_{\text{PRE}}$) (Note: not to scale)

The pre-contact position is a static position that a wing aircraft maintains for a period of time and is used for a few tasks. First, it allows the wing aircraft to stabilize while matching the lead aircraft's airspeed. Second, it demonstrates to the boom operator on the lead aircraft that the pilot of the wing aircraft is under control and it is safe to approach. Third, it allows the pilot of the wing aircraft to prepare for the impending maneuver. The position translations are *no closer than* values.

3.2.3 Contact Position. Once the wing aircraft sustains a stable pre-contact position, the wing aircraft is cleared to a contact position, denoted as $\mathbf{p}^{b_L}_{CONTACT}$. With a desired one *foot per second* closure rate, maneuvering from pre-contact to contact should last 30-60 *seconds*. The contact position is defined as a position on a 30° aspect angle measured from the b_L-frame's negative $\mathbf{X}$ axis in the direction of its positive $\mathbf{Z}$ axis. For a KC-135 aircraft, this position is denoted as 12 *feet* slant range from the end of the refueling boom (before extension) of the lead aircraft to the Universal Air Refueling Receptacle Slipway Installation (UARRSI) of the wing aircraft. This is the fuel port where the boom of the lead aircraft connects to transfer fuel. For this research, the UARRSI is approximated with the origin of the *cam*-frame and the

end of the refueling boom is approximated with the tail of the lead aircraft; however, the position is still defined to the coordinate-frame origins of the wing aircraft:

$$\mathbf{p}_{\text{CONTACT}}^{b_L} = \begin{bmatrix} -63 & -1 & 6 \end{bmatrix}^{\mathbf{T}} \tag{3.16}$$

This position is intended to be held statically, but for human operators, it is often challenging to do so. Changing conditions in airspeed, altitude, attitude, wind, visibility, turbulence, etc., require constant inputs to the controls of the wing aircraft, requiring constant attention and focus. A maneuvering envelope about the position permits fluctuations, inherent in this position. For a KC-135 aircraft the envelope allows the aspect angle to vary from 20° to 40°. The distance envelope is 6 to 18 *feet* and the azimuth angle (degrees left and right from center) is allowed to vary up to 10°. An experienced pilot does not require the full envelope; however, initial-training student-pilots will approach and often exceed these limits causing a disconnect (either manually or automatically initiated) of the boom from the UARRSI. Exceeding these limits can and has led to severe aircraft damage and loss of life.

Figure 3.11 shows the envelope with a B-52 aircraft in the contact position. The distance from the base of the boom to the end of the refueling boom (before the extension) is approximately 27 *feet*, 7 *inches*, and the slant range envelope starts 6 *feet*, 1 *inch* aft of that or the tip of the extension (33 *feet*, 8 *inches* total). The envelope extends 12 *feet* 3 *inches*, allowing a full envelope of 6 to 18 *feet* slant range.

3.3 Position Realization

It is possible to train a human pilot to recognize the above formation positions and determine deviations from those positions using both visual and aircraft instrumentation cues. From the author's AR instructing experience, experienced aviators can typically estimate the contact position within 1 to 2 *feet* of accuracy and the pre-contact position within 10 to 20 *feet* of accuracy. Pilots often rely on aircraft instrumentation (such as radar) for distances further than 3,000 *feet* range, with a

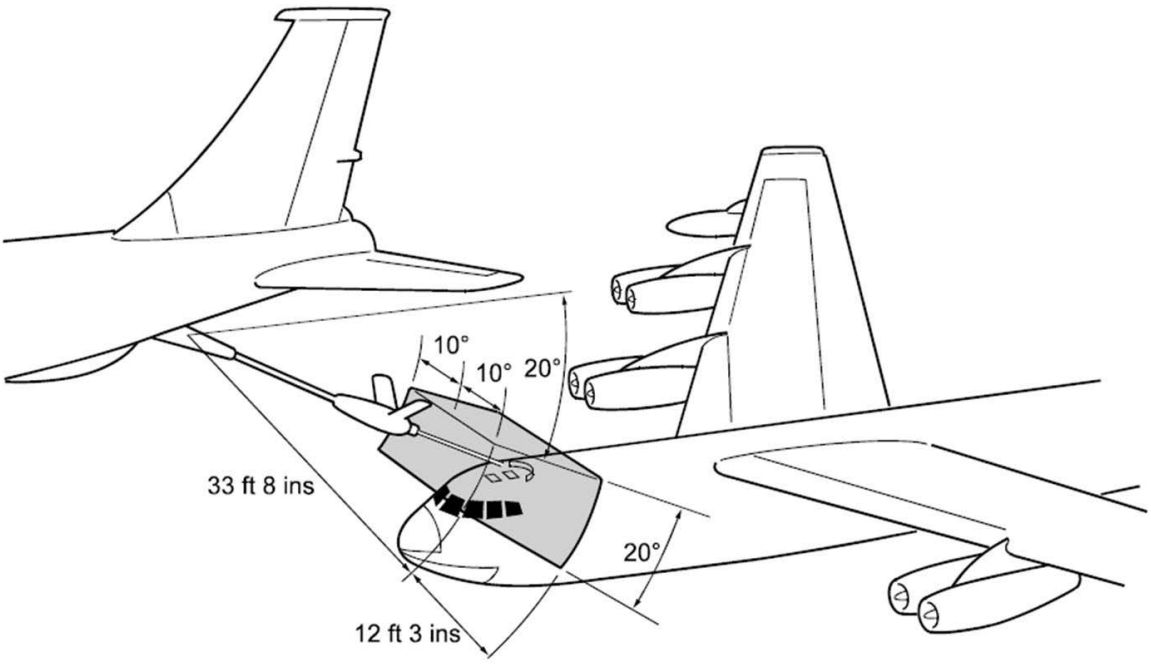


Figure 3.11: KC-135 refueling envelope [14]. This research assumes this envelope.

transition to visual-only references at closer distances. For a computer-vision solution, determining these positions is more involved. Starting with the most general difference equation between two navigating bodies, this section presents a derivation that details the information needed to determine the position of the wing aircraft in the b_L-frame:

$$\mathbf{p}^i_{L\Rightarrow W} = \mathbf{p}^i_W - \mathbf{p}^i_L \tag{3.17}$$

where the symbol, $\mathbf{p}^i_{L\Rightarrow W}$ denotes the position vector from the lead aircraft to the wing aircraft, referenced in the i-frame. This equation relates the difference in inertial position of the aircraft in the i-frame. It is generally accepted that navigation on the Earth's surface and sub-orbital atmosphere can be accomplished in the e-frame. This is the navigation frame used by aircraft with the aid of GPS and is the frame used in this research:

$$\mathbf{p}^e_{L\Rightarrow W} = \mathbf{p}^e_W - \mathbf{p}^e_L \tag{3.18}$$

If the rotation between the e-frame and b_L-frame is known, the difference can be rotated into the b_L-frame, $\mathbf{p}^{b_L}_{L\Rightarrow W} = \mathbf{C}^{b_L}_{e}\mathbf{p}^{e}_{L\Rightarrow W}$.

$$\mathbf{p}^{b_L}_{L\Rightarrow W} = \mathbf{C}^{b_L}_{e}(\mathbf{p}^{e}_{W} - \mathbf{p}^{e}_{L}) \tag{3.19}$$

The DCM $\mathbf{C}^{b_L}_{e}$ is separated into realizable transformations with the following results:

$$\mathbf{p}^{b_L}_{L\Rightarrow W} = \mathbf{C}^{b_L}_{b_W}\mathbf{C}^{b_W}_{n_W}\mathbf{C}^{n_W}_{e}\mathbf{p}^{e}_{W} - \mathbf{C}^{b_L}_{n_L}\mathbf{C}^{n_L}_{e}\mathbf{p}^{e}_{L}. \tag{3.20}$$

Critical navigation will only be required when the aircraft are relatively close to each other, so the e-frame to n-frame conversion of both aircraft is assumed to be equal ($\mathbf{C}^{n_L}_{e} = \mathbf{C}^{n_W}_{e}$). Furthermore, the position errors resulting from this assumption will not add significantly to the findings in this research. If, in the future, the approach presented here can reduce the error below an appropriate threshold then this assumption can be re-evaluated. To reduce confusion, these two frames have been replaced with a general n-frame:

$$\mathbf{C}^{n}_{e} \triangleq \mathbf{C}^{n_L}_{e} = \mathbf{C}^{n_W}_{e} \tag{3.21}$$

$$\mathbf{p}^{b_L}_{L\Rightarrow W} = \mathbf{C}^{b_L}_{n}\mathbf{C}^{n}_{e}(\mathbf{p}^{e}_{W} - \mathbf{p}^{e}_{L}) \tag{3.22}$$

Dealing with a frame located at the origin of the lead aircraft allows the subscript L to be dropped, it is assumed that the position vector is from the frame origin to the *identifier* subscript:

$$\mathbf{p}^{b_L}_{W} = \mathbf{C}^{b_L}_{n}\mathbf{C}^{n}_{e}(\mathbf{p}^{e}_{W} - \mathbf{p}^{e}_{L}) \tag{3.23}$$

This is the basic equation needed for a DGPS approach and requires the orientation of the lead aircraft to compute the n-frame to b_L-frame conversion ($\mathbf{C}^{b_L}_{n}$). The wing aircraft has access to its own position in the e-frame, $\mathbf{p}^{e}_{W}$, and also the rotation $\mathbf{C}^{n}_{e}$ as a function of $\mathbf{p}^{e}_{W}$. By broadcasting the lead aircraft's navigation position ($\mathbf{p}^{e}_{L}$)

and attitude ($\mathbf{C}_n^{b_L}$) to the wing aircraft, the wing aircraft can compute its position relative to the lead aircraft and navigate successfully.

For justifications addressed in Chapter 1, it would be useful if a camera and its associated reference frame, *cam*-frame, could provide an alternative approach to this equation. The following equations determine the necessary information needed for this alternate approach. In a similar manner to the DGPS equation derivation shown above, the following equations are presented:

$$\mathbf{p}_{CAM\Rightarrow L}^{cam} = \mathbf{C}_n^{cam}\mathbf{C}_e^n(\mathbf{p}_L^e - \mathbf{p}_{CAM}^e) \tag{3.24}$$

$$\mathbf{p}_{W\Rightarrow CAM}^{b_W} = \mathbf{C}_n^{b_W}\mathbf{C}_e^n(\mathbf{p}_{CAM}^e - \mathbf{p}_W^e). \tag{3.25}$$

Solving for $\mathbf{p}_{CAM}^e$ in Equation (3.24):

$$\mathbf{p}_{CAM}^e = \mathbf{p}_L^e - \mathbf{C}_n^e\mathbf{C}_{cam}^n\mathbf{p}_{CAM\Rightarrow L}^{cam} \tag{3.26}$$

Substituting Equation (3.26) into Equation (3.25) and solving for $\mathbf{p}_W^e$, results in:

$$\mathbf{p}_W^e = \mathbf{p}_L^e - \mathbf{C}_n^e\mathbf{C}_{cam}^n\mathbf{p}_{CAM\Rightarrow L}^{cam} - \mathbf{C}_n^e\mathbf{C}_{b_W}^n\mathbf{p}_{W\Rightarrow CAM}^{b_W} \tag{3.27}$$

Substituting $\mathbf{p}_W^e$ from Equation (3.27) into Equation (3.22), results in:

$$\mathbf{p}_{L\Rightarrow W}^{b_L} = -\mathbf{C}_n^{b_L}(\mathbf{C}_{cam}^n\mathbf{p}_{CAM\Rightarrow L}^{cam} + \mathbf{C}_{b_W}^n\mathbf{p}_{W\Rightarrow CAM}^{b_W}) \tag{3.28}$$

The DCM $\mathbf{C}_{cam}^n$ is separated into realizable transformations:

$$\mathbf{p}_{L\Rightarrow W}^{b_L} = -\mathbf{C}_n^{b_L}\mathbf{C}_{b_W}^n(\mathbf{C}_{cam}^{b_W}\mathbf{p}_{CAM\Rightarrow L}^{cam} + \mathbf{p}_{W\Rightarrow CAM}^{b_W}) \tag{3.29}$$

Again, the subscripts can be dropped:

$$\mathbf{p}_W^{b_L} = -\mathbf{C}_n^{b_L}\mathbf{C}_{b_W}^n(\mathbf{C}_{cam}^{b_W}\mathbf{p}_L^{cam} + \mathbf{p}_{CAM}^{b_W}) \tag{3.30}$$

Instead of using the e-frame locations of the aircraft, this approach makes use of the position of the camera in the b_W-frame ($\mathbf{p}_{CAM}^{b_W}$) and the estimated position of lead in the cam-frame ($\mathbf{p}_L^{cam}$). These positions are converted into a common reference frame, summed, and converted into the b_L-frame to determine the estimated position of the wing aircraft in the b_L-frame, ($\mathbf{p}_W^{b_L}$). Critical components of Equations (3.23) and (3.30) are shown graphically in Figure 3.12.

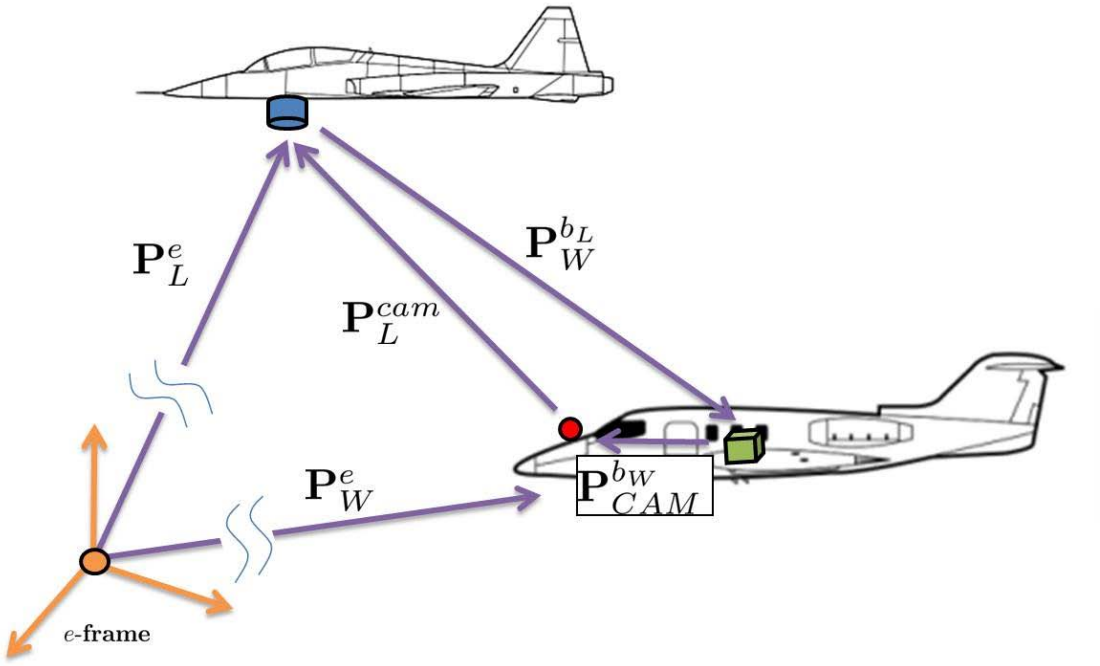


Figure 3.12: The critical components required for autonomous air refueling navigation. $\mathbf{p}_W^{b_L}$ can be determined from the difference of $\mathbf{p}_W^e$ and $\mathbf{p}_L^e$ through the use of differential GPS or the summing of $\mathbf{p}_L^{cam}$ and $\mathbf{p}_{CAM}^{b_W}$ through the use of image-aided relative-formation navigation.

In Equation (3.30), $\mathbf{p}_{CAM}^{b_W}$ and $\mathbf{C}_{cam}^{b_W}$ are assumed constant and known before flight. In contrast to the information needed in Equation (3.23), the DCM $\mathbf{C}_{b_W}^n$ is needed and will be known from an on-board Inertial Navigation System (INS). However, without a broadcast transmission from the lead aircraft, a wing aircraft will not have access to two components of this equation: the three Euler angles of the lead

aircraft to compute $\mathbf{C}_n^{b_L}$, or $\mathbf{p}_L^{cam}$. Estimating these two components using the images provided from the on-board camera and the INS of the wing aircraft is the focus of this research.

The final section outlines transferring these position realizations into the OpenGL world. The algorithm in this thesis creates $\mathbf{I}_r$ images based on the Kalman filter's estimation of the two required components of Equation (3.29) and the following section demonstrates how potential positions and attitudes of the tanker are created as $\mathbf{I}_r$ images.

3.3.1 Rendering Positions. At its simplest, the OpenGL rendering process requires six parameters: the position of an object in the *GLcam*-frame (three) and their rotation with respect to that frame (three). OpenGL allows the object to be rotated first and then translated or translated first then rotated, resulting in much different scenes and images. To adhere to the coordinate-reference-frame translation and rotation process detailed in Section 2.1.3, the rendering process shown here will always translate first and then rotate. The rotation between the *GLcam*-frame and the *b*-frame of an object is accomplished in the following manner: rotation about the $\mathbf{Y}_{GLcam}$ axis, rotation about the $\mathbf{X}_{GLcam}$ axis, and finally rotation about the $\mathbf{Z}_{GLcam}$ axis. Additionally, since this process is intended to represent actual images collected by the camera, the *cam*-frame and *GLcam*-frame share the same origin as demonstrated in Section 2.2.1.2.

The necessary attitude information of the lead aircraft is different then that required for the navigation solution in Section 3.3. Instead, the attitude of the lead aircraft with respect to the *GLcam*-frame, or $\mathbf{C}_{GLcam}^{b_L}$, is needed. This DCM can be computed by combining four different DCMs; two created from the Euler angles of each aircraft ($\mathbf{C}_n^{b_L}$ and $\mathbf{C}_{b_W}^n$), one determined before flight (the rotation of the wing aircraft b_W-frame origin to camera, $\mathbf{C}_{cam}^{b_W}$), and the transformation between the

cam-frame and *GLcam*-frame ($\mathbf{C}^{cam}_{GLcam}$) as shown:

$$\mathbf{C}^{b_L}_{GLcam} = \mathbf{C}^{b_L}_{n}\mathbf{C}^{n}_{b_W}\mathbf{C}^{b_W}_{cam}\mathbf{C}^{cam}_{GLcam} \tag{3.31}$$

Of these four DCMs, only one will be unknown to the algorithm, $\mathbf{C}^{b_L}_{n}$.

The translation of the lead aircraft will be referred to as $\mathbf{p}^{GLcam}_{L}$, similar to that described in Section 3.3 and accounting for the rotational difference between frames. Equation (3.29) is modified to:

$$\mathbf{p}^{b_L}_{W} = -\mathbf{C}^{b_L}_{n}\mathbf{C}^{n}_{b_W}(\mathbf{C}^{b_W}_{cam}\mathbf{C}^{cam}_{GLcam}\mathbf{p}^{GLcam}_{L} + \mathbf{p}^{b_W}_{CAM}) \tag{3.32}$$

and solving for $\mathbf{p}^{GLcam}_{L}$ yields:

$$\mathbf{p}^{GLcam}_{L} = -\mathbf{C}^{GLcam}_{cam}\mathbf{C}^{cam}_{b_W}(\mathbf{C}^{b_W}_{n}\mathbf{C}^{n}_{b_L}\mathbf{p}^{b_L}_{W} + \mathbf{p}^{b_W}_{CAM}) \tag{3.33}$$

Of these parameters, only six will be unknown to the algorithm, $\mathbf{C}^{n}_{b_L}$ and $\mathbf{p}^{b_L}_{W}$.

The vector $\mathbf{p}^{GLcam}_{L}$ and matrix $\mathbf{C}^{b_L}_{GLcam}$ with six total parameters (or six DOFs) determine where to place the aircraft in the frustum, and at what attitude. The algorithm predicts these six parameters from a combination of state estimations from the Kalman filter and their final values at the previous time instant, then verifies them with a measurement process detailed in the next chapter.

This chapter introduced the dynamics of AR, the model that describes those dynamics, and the position requirements of AR including the OpenGL representation of those positions. The next chapter presents this thesis' solution to Autonomous Aerial Refueling.

IV. Pose and Air Refueling

Building on the knowledge of the previous chapters and incorporating the research accomplished by others, this chapter presents a novel approach to AAR. The chapter is broken into three sections: the first details the position and orientation estimation, commonly referred to as pose estimation: a process alluded to in previous chapters of this thesis. The next section is a background investigation into other approaches taken to solve AAR, and those using a rendered model approach to pose. The final section presents the works and techniques of this thesis.

4.1 Pose

In computer vision disciplines, it is often necessary to determine an object's position and orientation with respect to a specified coordinate system. Estimating an object's attitude and translation in the *cam*-frame ($\mathbf{C}^{b}_{cam}$ and $\mathbf{P}^{cam}$) is this thesis' interpretation of pose estimation. The pose of an object allows the system to make determinations about the object, interact with the object, or in the case of AAR, track the object. This field of study encompasses many different methods to determine the pose of an object with the desired accuracy. The pose process incorporates single or multiple image collection devices and may incorporate other sensors (such as distance measuring or three-dimensional scanning), which are real-time or post-processed for various lengths of time.

The research presented in this thesis focuses on finding the pose of a lead aircraft based on images alone to determine a relative navigation solution between the two aircraft. The inherent difficulty with pose from images is estimating depth, or translation of the object along the $\mathbf{Z}_{cam}$ axis, as described in Section 2.1.3.5 (the camera matrix, $\mathbf{K}$, discards this information). To aid the process, a system often has some predetermined knowledge of the camera, environment, and object (typically the object's scale).

With this *a priori* knowledge, a common vision-based approach (not used in this research) determines the pose of an object with trigonometric calculations based on a

specific number of surveyed features of the object that are located in an image. This approach usually involves matching, from image to image, these determined features on the object. This is typically referred to as point or feature tracking.

Tracking individual points visually on an object presents a few challenges. The first difficulty is in accounting for features departing the FOV of the camera. If not handled appropriately, the system will match the features to similar but incorrect features on the object, causing residual error in the estimation. The second difficulty concerns the lighting and shading on an object. Features that are easy to determine in one lighting condition might not be as easy to detect in other conditions. Changes in lighting can also introduce confusion between similar but incorrectly matched features. A final difficulty is in updating the visual appearance of the feature with changes in orientation. When tracking an antenna on the bottom of the aircraft, the antenna's appearance changes depending on which side of the aircraft the camera is on.

These difficulties are not insurmountable and this approach works for many applications. The next section details a small sampling of visual-based pose in addition to differing methods to conduct AAR.

4.2 Current Science

This section presents various approaches to AAR and the vision based approach to pose. AFRL has done and continues to do extensive research in the area of AAR, and many projects sponsored by AFRL have aided this research immensely.

4.2.1 AFIT. Research at AFIT began in the early 1990s and concentrated mainly on formation flight controllers [16]. Starting with simple controllers and limited freedoms by the lead and wing aircraft, investigations focused on real-time autonomous controllers.

4.2.1.1 Spinelli and Ross. AFIT research into AAR reached a milestone with thesis work done by Spinelli and Ross in 2006 [16, 24]. Both worked

independently to create a combined DGPS-dependent formation controller. Spinelli's research focused on a combination transmitter and receiver from the lead aircraft broadcasting real-time GPS data to the receiver aircraft. This information was processed by a computer on-board the receiver aircraft. Algorithms devised by Ross plotted a relative navigation solution. This solution provided control inputs to the receiver's autopilot controller, resulting in autonomous formation flight at ranges from 10 to 100 *feet* and up to 30 *degrees* of bank.

The research was demonstrated using a TPS C-12 Huron as the simulated tanker and a Calspan LJ-24 Learjet as the receiver, shown in Figure 4.1. The wing aircraft maneuvered through three positions, based off the lead aircraft: contact, pre-contact, and awaiting AR (wing aircraft immediately off the right wing of lead). Average error was calculated to be approximately one to two *feet* with a maximum of approximately four to five *feet* [16]. These test flights represented a tremendous step towards achieving AAR.

Figure 4.1: Differential GPS AAR between a C-12 and a LJ-24 [16]. This successful demonstration proved the capability of AAR and led to many future projects.

While this work became a benchmark for AAR, many improvements to the approach were desired. Specifically, the incorporation of a vision-based sensor to aid the navigation solution.

4.2.1.2 Spencer. Following the success of Spinelli and Ross, Spencer also used a C-12 and LJ-24 to conduct test flights in 2008 using optical tracking of the lead aircraft [23]. The test flights were not autonomous, but accomplished real-time, with on-board processing and data collection for additional post flight analysis.

Spencer utilized an electro-optic (EO) sensor in conjunction with a Harris corner detector algorithm to track multiple feature points of the lead aircraft (more than 12 per frame, as shown in Figure 4.2). The EO sensor was able to collect real-time images for processing by an on-board computer. By making predictions of where the corners should be with a Kalman filter, point tracking of the corners used a *gating* technique. This technique limited the field of view around each tracked point that a feature could match to, so antennae on the rear of the aircraft could not match to antennae on the front of the aircraft. An initial position and attitude was provided to the algorithm via a non-passive sensor [23].

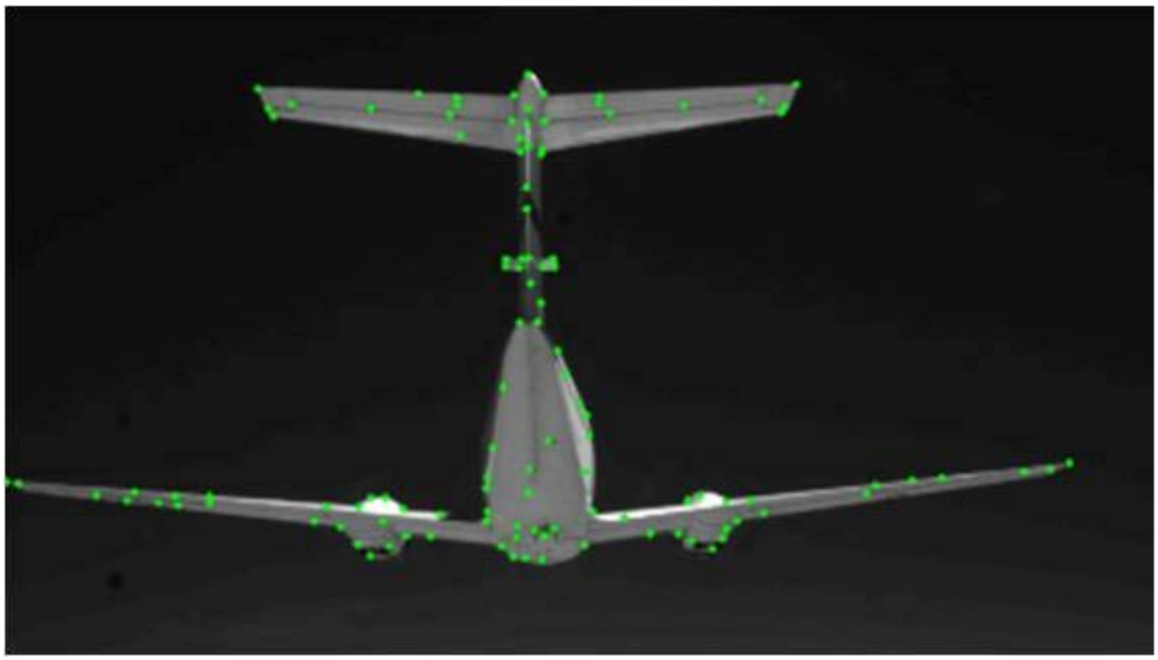


Figure 4.2: Visual AAR using feature tracking [23]. A visual approach to determining the relative position between aircraft by tracking individual points, surveyed to known locations, of a lead aircraft.

This process found and updated a relative position of the wing aircraft with respect to the lead aircraft using known distance and direction between the tracked points (a three-dimensional model of surveyed points on the aircraft). Spencer's research demonstrated the feasibility of the vision-based navigation concept. As a result, future research can now address some of the problems discovered in his research. Specific errors relating to the vision-based approach to AAR are: the loss of tracking features due to poor visibility, delays in position estimates because of image processing time, errors related to camera calibration, and pose errors from incorrect corner tracking.

Perhaps one of the most important lessons learned concerned camera attitude (ψ, θ, ϕ). The receiver aircraft's attitude is very important in determining the receiver's position in relation to the tanker. Failing to account for yaw, pitch, and roll of the receiver aircraft dramatically detracts from the algorithm's ability to identify the correct relative position.

Spencer suggested, as a potential solution to a few of the problems, to apply group tracking instead of individual points. While individual points are prone to misinterpretation by the detection algorithm, tracking groups of points can potentially limit those errors. He discovered that multiple point errors were not common in an individual frame. The *group of points* recommendation was expanded to include the entire tanker (designated here as the *whole aircraft* approach) in subsequent research efforts.

4.2.1.3 Weaver. As a follow up to Spencer's work, Weaver pursued an alternative approach to the aircraft tracking problem [29]. Weaver's work utilized a long-wave infrared video representation of a KC-135R (acquired from an AFRL research initiative) in conjunction with a three-dimensional rendered model of the same aircraft type (obtained commercially). Using an extended Kalman filter (EKF), Weaver was able to make *a priori* predictions about the KC-135R's position relative to the receiver. These predictions became images produced through the rendering

of the three-dimensional computer model. A sum-squared difference between the pixel intensities of the rendered and collected image determined an error between the predicted position and the actual position. However, the calculation only provided the magnitude of the error. Determination of the relative direction of the error required an iterative process to correct the image for the next update. This process continually perturbed (change in magnitude and direction) the estimated position and orientation of the KC-135R. This created a new image for each perturbation to compare with the true collected image of the tanker. An example of a collected image with the predicted image overlaid (intentionally offset) is shown in Figure 4.3.

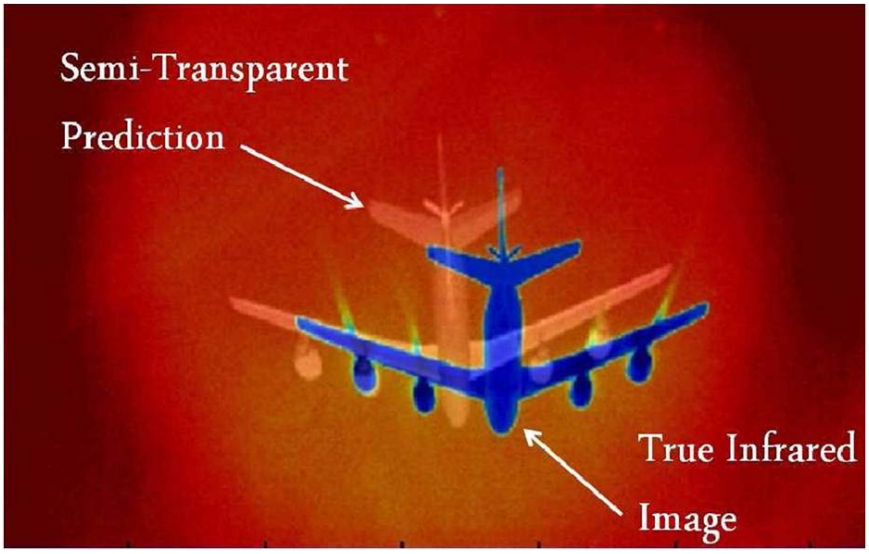


Figure 4.3: AAR using whole aircraft tracking with infrared images [29]. A vision based approach determines the relative position between aircraft by comparing a collected image with rendered images of the same aircraft. The prediction image is intentionally offset from the true image for visual clarity.

Weaver's research was a departure from the feature and point tracking methods found in many research efforts. By forging this new direction he discovered new sets of problems including errors in tanker-body **X**-axis, roll errors, and length image processing time.

Images associated with a pitching movement by the tanker were difficult to distinguish from images representing a relative acceleration forward. This resulted

in Weaver's system having position estimates with 1-meter accuracy in the **Y** and **Z** directions, but 2-meters in the **X** direction. In addition, this system did not respond well to turns by the tanker, potentially because of the values predicted by the Kalman filter for the tanker's roll position. When a turn occurs, an incorrectly modeled filter wants to return to, or maintain, a nominal wings level state instead of accepting the change in roll measurement. The majority of the time, the tanker maintains constant speed, heading, and roll angle. Eventually, however, the aircraft does make large roll angle changes. If the Kalman filter permits large variations to the state (a large covariance in the dynamic-model process-noise), accurate solutions are difficult. Conversely, allowing only small variations can minimize the effects of measurements on the Kalman state.

Suggestions by Weaver for future research included additional sensors such as ranging or GPS data link (as introduced by Spinelli and Ross) and potentially an Unscented Kalman Filter (UKF). The UKF could enhance the speed of the updates to the navigation solution by potentially reducing the processing time to determine the direction of error between the two images.

4.2.2 Georgia Institute of Technology. Researchers at Georgia Tech have done considerable work in the area of vision-based navigation [5, 15, 30]. In 2005, Wu, Johnson, and Proctor used images of an object seen by an unmanned helicopter to aid in the helicopter's navigational solution. Information determined from the images consisted of a center point and total area. Using an EKF, these measurements provided updates to the state and covariance of the navigation solution. This system assumed that the pose algorithm knows the position, size, and orientation of the object before the flight. Using an object (a window of 36 ft^2) the system was able to track its position within a mean error of about six feet [30].

Further research at Georgia Tech involved whole aircraft tracking. Aircraft were tracked using their center and wingtips (designated in their research as the *center tips* approach) [15]. The information about the aircraft's center estimated

the target's relative azimuth and elevation, while its tips estimated the aircraft's distance. The UKF, as implemented, tracked the target's position, velocity, size, and acceleration using measurements attained from the image. Results from this research were satisfactory and one of the future research suggestions was to incorporate a more sophisticated representation of the target, such as a predicted rendering of the whole target, as opposed to estimation of just the three tracked points.

4.2.3 Visual-Model Based POSE. Researchers de Ruiter and Benhabib expanded the whole aircraft approach to include visual or textual models of the target [17, 18]. By utilizing *a priori*, visual, and an adaptive model of a rigid body, the researchers were able to track the body and determine the orientation and position of the object. A comparison between the rendered predictive images, utilizing the model in OpenGL, with the actual image collected by the camera, determines the accuracy of the prediction. The difference between the two images was calculated and combined with the information used to render the initial predictive image to determine an accurate pose of the target.

Image-based tracking is computationally demanding and difficult to use in order to attain accurate navigation information in real-time. However, de Ruiter and Benhabib were able to attain rates of 80-100 *frames per second* and sub-pixel accuracy by reducing unnecessary computations. Reduced gradient computation time was possible by setting a region of interest around the target in the image and reducing three color gradient images to single color gradient images [18]. The target used in this research was a simple cube with a simple texture/image on each side.

The background research provides a baseline of the current approaches to pose and AAR. The method of this thesis builds on these techniques, combining the benefits of some while addressing the limitations of others. The next section presents a generic rendered-image approach to vision-base pose, followed by the specific implementation in this research.

4.3 RIPE

A few of the researchers in the previous section and some others are independently developing a sub-discipline of pose that proposes a *whole object* method of tracking. This type of approach addresses some of the difficulties of point tracking alluded to in the background research by tracking an object as a whole instead of individual features. Weaver's approach to whole object tracking made use of rendered images of the object. The premise of a rendered image position and orientation estimation, RIPE[2], approach to navigation requires a three-dimensional computer model of an object (*a priori* or created real-time) that can be rendered as a two-dimensional $\mathbf{I}_r$. This notional approach can involve one or more $\mathbf{I}_r$ images. The time allotted for pose and the required accuracy determines the desired number of images.

The method employed by Weaver [29] in his research consisted of creating multiple $\mathbf{I}_r$ images, comparing each with a single $\mathbf{I}_c$ for every measurement update. The six DOF information to create each $\mathbf{I}_r$ initially came from the EKF's state estimation in the n-frame (rotated and translated into the cam-frame) of the tanker aircraft's attitude and translation from the camera, $\mathbf{C}_{cam}^{b_L}$ and $\mathbf{p}_L^{cam}$ respectively.

The Kalman filter's *a priori* state estimation was rendered in addition to several $\mathbf{I}_r$ images created with perturbations in each DOF about the estimate. Computations on the uncertainty in the state estimation (a Cholesky decomposition of the covariance, $\sqrt[c]{\mathbf{P}}$) determined the necessary magnitude of the perturbations about the $\mathbf{C}_{cam}^{b_L}$ and $\mathbf{p}_L^{cam}$ estimates.

A sum-squared-difference calculation on the pixel intensities of each $\mathbf{I}_r$ and the $\mathbf{I}_c$ found an error value for each resulting perturbation image as well as the original state-estimated image. Combined, a set of the errors created a gradient of likelihood for each of the six DOFs. The DOF gradient with the most potential improvement in matching likelihood updated the estimate. This update replaced the DOF's value

[2]The term RIPE was introduced as part of the data collection project facilitating this research and is used in this thesis as a label for this newly developing approach to pose.

in the original estimate with the more-likely value (based on the gradient likelihood), creating a new estimate. From this new estimate, a new set of perturbation were determined and new set of $\mathbf{I}_r$ images were rendered. The process iterated until no more improvements were possible, such that no perturbation $\mathbf{I}_r$ images appeared more likely than the estimate's $\mathbf{I}_r$.

Next, a more precise, or fine-pose measurement determination was initiated. The fine process mimicked the coarse process with perturbations one tenth the size, until no further improvements were possible. This final estimate represented a global, most-likely pose of the tanker aircraft. The information required to create the final, most-likely $\mathbf{I}_r$ provided a measurement update to the filter.

The approach was iterative; large perturbations along one DOF at a time were rendered to determine coarse estimations of the object and then smaller perturbations were rendered for fine estimation [29]. An assumption made in that research was the decoupling of the six DOFs. Without this assumption, the matching process would require 729 $\mathbf{I}_r$ images per iteration; the state estimate plus two perturbations in each DOF coupled ($3^n, n =$ number of coupled DOFs). This important assumption reduced the required $\mathbf{I}_r$ images to 13 for each iteration, the initial estimate plus a single positive and negative perturbation in each DOF (2^*m+1, $m =$ number of decoupled DOFs). Unfortunately this assumption also limits the precision of the estimation.

Decoupling the DOFs assumes the motion of an object in any DOF is independent of the other five. Visually, a decoupling assumes that motion in one DOF can be determined without concern of other motions. For example, if an object translates left and up in an image, the decoupling assumes the left translation is distinguishable in the image without accounting for the translation up.

This assumption is valid when the required perturbations about the estimated position to make an image match are small. When large motions occur between collected images this assumption introduces additional error. Large visual matching errors do not necessarily relate to position errors in the linear manner necessary

for this assumption. Weaver's work relied on a single $\mathbf{I}_r$ resembling $\mathbf{I}_c$ more than other $\mathbf{I}_r$ images and that the iterated perturbations would eventually find a global maximum likelihood $\mathbf{I}_r$ as a match to the true $\mathbf{I}_c$. However, with large motions and decoupled DOFs, local maximum-likelihoods can occur at locations removed from the true location. At these local maximums, all other potential $\mathbf{I}_r$ images appear less likely and the perturbation and iteration process stops, accepting the local maximum as the incorrect position and orientation of the tanker. These errant positions became incorrect measurements that disrupted the estimation of the filter. Fortunately, in his approach the perturbations about future state estimations grew because of the bad measurements (larger covariance entailed larger perturbations) allowing the filter to eventually recover. The cost of the assumption was both processing time and temporary inaccuracies in the state estimate.

The next section presents an efficient rendered image approach to pose with a focus on minimizing the required number of $\mathbf{I}_r$ images to as few as possible while attempting to maintain the accuracy of a coupled DOF approach.

4.3.1 Quick-RIPE. To address some of the problems encountered in previous efforts, the quick-RIPE approach uses template matching to reduce the number of $\mathbf{I}_r$ images necessary for a pose estimation of an object while not completely decoupling the DOFs for faster and accurate estimations. Determined from the background research, and laboratory experiments, there is a discernable balance between speed and accuracy. With more images rendered, the pose accuracy increases but so does the time required for determination. The first section details a general quick-RIPE approach, while the second section tailors this approach to AAR.

4.3.1.1 Quick-RIPE Methodology. With the use of a template matching function, such as the one described in Section 2.2.2.2, the total number of $\mathbf{I}_r$ images needed to determine the *image location* of an object in the $\mathbf{I}_c$ reduces to one, while effectively coupling two or more DOFs. For the rest of this thesis, the term *image location* refers to the translation of the object in the $\mathbf{X}_{image}$ and $\mathbf{Y}_{image}$ axes

(the object's coordinates in the *image*-frame). Through the linear relationship of the camera matrix ($\mathbf{K}$, Section 2.1.3.5) the object's translations in $\mathbf{X}_{cam}$, $\mathbf{Y}_{cam}$, and $\mathbf{Z}_{cam}$ determine the image location of the object in the $\mathbf{I}_c$. The term *position* still refers to the actual position of the object in the desired reference frame. This section references two positions, the real-world position of the object, $\mathbf{p}_o$, and the algorithm's predicted position of the object, $\mathbf{p}_r$. The process involves estimating $\mathbf{p}_r$ close enough that a template match corrects it to $\mathbf{p}_o$. This section details the template matching process.

Quick-RIPE template matching separates the DOFs of an object into the two following groups, which are broken down into their components:

- **Attitude plus image location:** Rotation in all three *cam*-frame axes plus translation in the $\mathbf{X}_{image}$ and $\mathbf{Y}_{image}$ axes.
- **Size plus image location:** Translation in the $\mathbf{Z}_{cam}$, in addition to $\mathbf{X}_{image}$ and $\mathbf{Y}_{image}$ axes.

In this grouping, *size* denotes the object's translation in the $\mathbf{Z}_{cam}$ axis. Effectively, this grouping creates a group of five DOFs and a group of three respectively. At first glance, this does not appear to be much of a decoupling. However, with template matching it reduces the rendering cost to an equivalent grouping of three DOFs coupled and one DOF uncoupled respectively. The grouping reduces the number of required $\mathbf{I}_r$ images while permitting partial coupling of the DOFs.

As a general overview, the quick-RIPE template matching renders an object approximately close to and with approximately the same attitude as the actual object in the camera's FOV. First, the coupled attitude-DOFs of the object are perturbed and each rendered as an $\mathbf{I}_r$. Each $\mathbf{I}_r$ is then template matched to the $\mathbf{I}_c$ determining the most likely combination of attitudes. Second, the process is repeated with perturbations along the $\mathbf{Z}_{cam}$ axis, determining the most likely size of the object and, as a result of the template matching process, the most likely image location of the object. The combination of the translation in the $\mathbf{Z}_{cam}$ axis, $p_{z,o}^{cam}$, and the image location of

the object determines the translation in the $\mathbf{X}_{cam}$ axis and $\mathbf{Y}_{cam}$ axis, $p_{x,o}^{cam}$ and $p_{y,o}^{cam}$, through an approximated, linear, trigonometric relationship.

4.3.1.2 Quick-Ripe Template Matching. To help introduce the template matching approach the following, incorrect but temporary, assumption is made: the attitude and size of the object are known. The reason this assumption is incorrect and the compensation for it being incorrect is addressed in Section 4.3.1.3. Additionally, as a necessary assumption for the template matching process to work, it is assumed that the actual image location of the object is *near* the currently rendered object image location ($\mathbf{p}_r \approx \mathbf{p}_o$ from an initial condition or from a recent update). As a result of these assumptions, within a region around $\mathbf{p}_o$, or with small $\mathbf{X}_{cam}$ and $\mathbf{Y}_{cam}$ translational errors in the estimate, $\mathbf{p}_r$, the rendered object looks similar to the actual object.

This can be seen in the sample images of Figure 4.4, the top of the image is an $\mathbf{I}_c$ and the bottom two images are perturbation $\mathbf{I}_r$ images, along the $\mathbf{X}_{cam}$ axis. With such noise-free backgrounds, both will match with the aircraft in the $\mathbf{I}_c$. However, with the better image location estimate (the $\mathbf{I}_r$ on the left), the rendered image comes closer to resembling the actual object. Visually, this is distinguishable in the figure, by the difference in appearance of the two $\mathbf{I}_r$ images. With the $\mathbf{I}_r$ closely resembling the $\mathbf{I}_c$, the template matching function's coordinates of the most-likely match ($\mathbf{r}$) is an estimator of $p_{x,o}^{cam}$ and $p_{y,o}^{cam}$ through an approximated, linear, trigonometric relationship.

Presented next are the necessary components of this relationship, which include the pixel FOV in both the $\mathbf{X}_{cam}$ and $\mathbf{Y}_{cam}$ axis, the approximate $p_{z,o}^{cam}$, and the difference in size between the template image and the image it is matched against. Presented after these three components is an overview of the template matching process and the complete equation demonstrating the relationship.

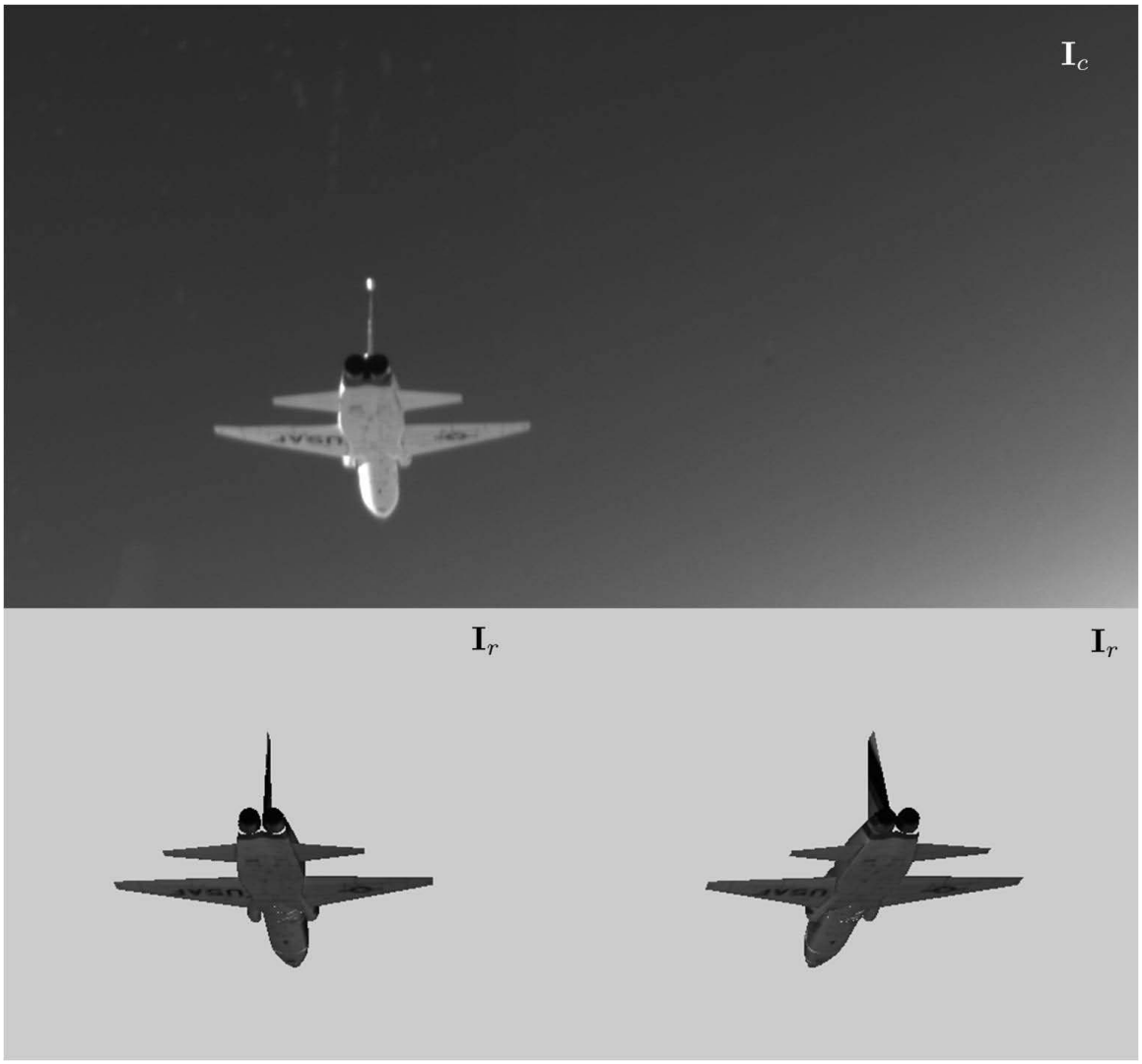


Figure 4.4: An example $\mathbf{I}_c$ and two perturbation $\mathbf{I}_r$ images. The template matching approach requires the initial estimate to be relatively close to the true estimate.

The first necessary component is the pixel FOV, or instantaneous FOV (IFOV), of the camera and lens in both the $\mathbf{X}_{cam}$ and $\mathbf{Y}_{cam}$ axis. Dividing the entire FOV (Equation 2.31) by the number of pixels in the array, determines the pixel IFOV.

$$\mathrm{IFOV_Y} = \frac{\mathrm{FOV_Y}}{M} \tag{4.1}$$

$$\mathrm{IFOV_X} = \frac{\mathrm{FOV_X}}{N} \tag{4.2}$$

The second item required, $p_{z,o}^{cam}$, is approximated with the -$\mathbf{Z}_{GLcam}$ translation of the rendered object, -$p_{z,r}^{GLcam}$.

Finally, the difference in size between the template image and the associated matched image is known because they both are user defined sizes. The method applied for this research was to create a template based on the contours of the rendered object in the $\mathbf{I}_r$. Because the image is clean, with no background or foreground noise, the contour program finds the contours of the object only. The extreme contours in both the $\mathbf{X}_{CVimage}$ and $\mathbf{Y}_{CVimage}$ axes create a bounding box around the rendered object. The bounding box defines a region of interest (ROI) in $\mathbf{I}_r$ labeled the *template* and denoted as $\mathbf{I}_t$. The template is shown as the rectangle around the aircraft in the left side image of Figure 4.5. The center of $\mathbf{I}_t$ is shown as a white circle in the figure.

The center of $\mathbf{I}_t$, found in the $\mathbf{I}_r$ (left image of Figure 4.5), but now placed on the $\mathbf{I}_c$, defines the center of a second bounding box, shown as the small white square in the right image. This second bounding box is labeled the *match* ROI, denoted as $\mathbf{I}_m$. The $\mathbf{I}_m$ is shown as the larger rectangle around the aircraft in the right image of Figure 4.5. The smaller rectangle in the right image has the same center and size as $\mathbf{I}_t$, simply copied onto the $\mathbf{I}_c$. From this setup, the two aircraft do not occupy the same position in their respective images; the rendered aircraft is half a wing length to the left of the actual aircraft location.

The size of the $\mathbf{I}_m$ is arbitrary. The approach in this thesis increased all four sides equally. The symbol Δ_{box} denotes an integer number of pixels added twice to

both dimensions of the $\mathbf{I}_t$, increasing the size of the $\mathbf{I}_m$. Passing these two images to the matching program creates an $\mathbf{R}$ matrix with dimensions: $(2\Delta_{box}+1) \times (2\Delta_{box}+1)$.

The template matching process, detailed in Section 2.2.2.2, compares the $\mathbf{I}_t$ to the $\mathbf{I}_m$ by starting in the top left corner. The $\mathbf{I}_t$ is then slewed, one pixel at a time until reaching the right edge of the $\mathbf{I}_m$. It then returns to the top left corner, one pixel down, and repeats. A comparison between the two images is made at every possible location, determining the location of the most likely match.

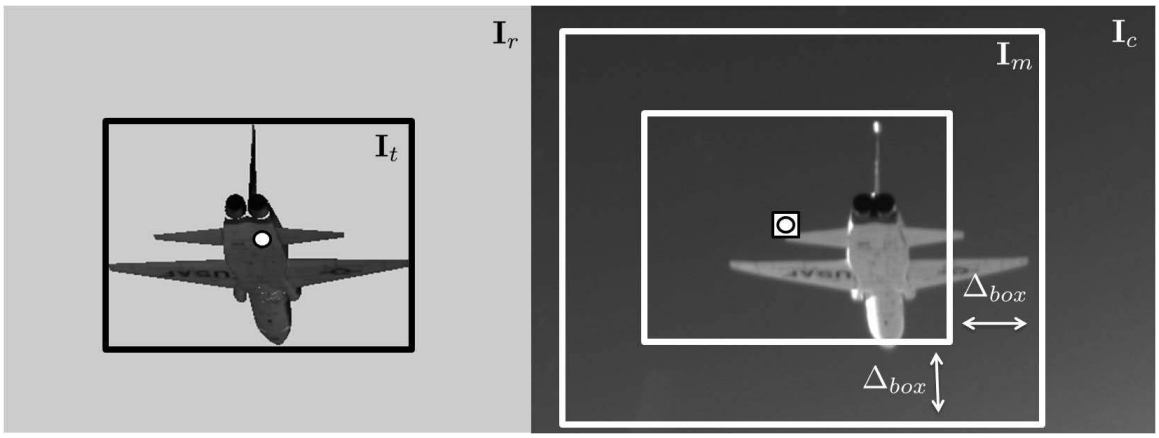


Figure 4.5: Template ($\mathbf{I}_t$) and match ROI ($\mathbf{I}_m$) creation. The center of the $\mathbf{I}_t$ in the $\mathbf{I}_r$, white dot, left image, is placed in the same coordinates on the $\mathbf{I}_c$ to determine the center of the $\mathbf{I}_m$ in the right image, a white square. Each dimension of the $\mathbf{I}_m$ is $2\Delta_{box}$ greater than $\mathbf{I}_t$.

The result of the template match is $\mathbf{R}$. When accessing its values, the top left-most coordinate of $\mathbf{R}$ is referenced as $\mathbf{R}(0,0)$. This value in $\mathbf{R}$ represents the value of the matching method (Section 2.2.2.2) applied to the $\mathbf{I}_t$ and the top, left-most portion of $\mathbf{I}_m$. Similarly, the bottom right-most position in $\mathbf{R}$, referenced as $\mathbf{R}(2\Delta_{box}+1,\ 2\Delta_{box}+1)$ represents the value of the matching method applied to the $\mathbf{I}_t$ and the bottom, right-most portion of $\mathbf{I}_m$. The center of $\mathbf{R}$, referenced as $\mathbf{R}(\Delta_{box}+1,\Delta_{box}+1)$ represents the center of the $\mathbf{I}_m$ and the original center of the $\mathbf{I}_t$. *Important relationship*: if the most-likely match of the entire $\mathbf{R}$ is at the center position, $\mathbf{r}_{match}=(\Delta_{box}+1,\Delta_{box}+1)^{\mathbf{T}}$, the translations along the $\mathbf{X}_{cam}$ and $\mathbf{Y}_{cam}$ axes of

the object are the same as the translations used to create the rendered object, or $p_{x,o}^{cam} = p_{x,r}^{GLcam}$ and $p_{y,o}^{cam} = \text{-}p_{y,r}^{GLcam}$.

Finally, combining the necessary components into an approximate trigonometric relationship, an estimation for $p_{x,o}^{cam}$ and $p_{x,o}^{cam}$ is determined:

$$\tilde{p}_{x,o}^{cam} = p_{x,r}^{cam} - p_{z,o}^{cam} \cdot \tan((\Delta_{box} + 1 - r_x) \cdot \text{IFOV}_\text{X}) \tag{4.3}$$

$$\tilde{p}_{y,o}^{cam} = p_{y,r}^{cam} - p_{z,o}^{cam} \cdot \tan((\Delta_{box} + 1 - r_y) \cdot \text{IFOV}_\text{Y}) \tag{4.4}$$

where, r_x is the x coordinate of the most likely match ($\mathbf{r}$) and r_y is the y coordinate. The conversion from *GLcam*-frame (required to render the image) to *cam*-frame has already occurred. To implement in OpenGL:

$$\tilde{p}_{x,o}^{cam} = p_{x,r}^{GLcam} + p_{z,r}^{GLcam} \cdot \tan((\Delta_{box} + 1 - r_x) \cdot \text{IFOV}_\text{X}) \tag{4.5}$$

$$\tilde{p}_{y,o}^{cam} = -p_{y,r}^{GLcam} + p_{z,r}^{GLcam} \cdot \tan((\Delta_{box} + 1 - r_y) \cdot \text{IFOV}_\text{Y}) \tag{4.6}$$

This relationship depends on a key principle, the linear mapping of $\mathbf{K}$. This linear relationship is essential, because the center of $\mathbf{I}_t$ and $\mathbf{I}_m$ are arbitrarily chosen. The center of the chosen template can be anywhere on the rendered object or not on it at all. However, it should overlay the same relative position of the real object in the most-likely match. The position of the template center referenced to the *b*-frame of the object remains constant during the matching. Because of this, the linearity of the $\mathbf{K}$ matrix allows the translation difference between the center of the template and the center of the most likely match to be the same as if the physical origin of the object had been matched in the images.

As an example, it is assumed the center of $\mathbf{I}_t$ in the $\mathbf{I}_r$ was the wingtip of an aircraft. The position of the wingtip is a fixed translational distance in all three axes from the defined origin of the object (in the *b*-frame). When matching the $\mathbf{I}_t$ to an $\mathbf{I}_m$, the wingtips will match. The difference in translation (between the $\mathbf{I}_t$ and the $\mathbf{I}_m$)

in the $\mathbf{X}_{cam}$ and $\mathbf{Y}_{cam}$ axes of the wingtip is the same as the difference in translation of the origin of the object.

Equations (4.3 - 4.6) are simply trigonometric relationships, relating the change in translation in pixels to a unit of measurement based on the distance an object is from the origin of the *cam*-frame. The completion of the example, from Figure 4.5, is shown to demonstrate.

Figure 4.6 shows the $\mathbf{I}_t$ box overlayed on the most likely matching portion of the $\mathbf{I}_m$ (right image of the figure). The aircraft in the $\mathbf{I}_r$ needed to move to the right and down slightly to match the actual aircraft in the $\mathbf{I}_c$. The star in the right image denotes the center of the most likely ROI in $\mathbf{I}_m$ that matches $\mathbf{I}_t$, the square denotes the center of the $\mathbf{I}_m$. Both of these symbols are placed on their related coordinates on the $\mathbf{R}$ matrix in the bottom image of Figure 4.6. The distance in pixels between them (in both the $\mathbf{I}_c$ and the $\mathbf{R}$ matrix) represents the angular difference between $\mathbf{p}_o$ and $\mathbf{p}_r$.

This relationship is an approximation because the distance ratios, $p^{cam}_{z,r}/p^{cam}_{x,r}$ and $p^{cam}_{z,r}/p^{cam}_{y,r}$, determine the accuracy of the calculation. The larger the ratios, the more accurate it becomes. This is best seen in Figure 4.7 which continues the previous example with only a $\mathbf{X}_{cam}$ translation difference between the positions. The square and the star represent the same change in angular positions. Because the triangle, created with the line of sight lines emanating from the origin of the *cam*-frame to both the square and star, is oblique, the tangent function used in the equation is an approximation. Cameras with a larger FOV that are locating objects relatively close along the $\mathbf{Z}_{cam}$ axis must use an expanded trigonometric relationship.

As a result of this relationship, the matching function effectively couples two DOFs, the translations in the $\mathbf{X}_{cam}$ and $\mathbf{Y}_{cam}$ axes. One $\mathbf{I}_r$ accomplishes the equivalent of $(\Delta_{box}+1)^2$ perturbation $\mathbf{I}_r$ images. The next section revisits the assumptions of this section and details the effects of this coupling on the attitude plus image location coupled group.

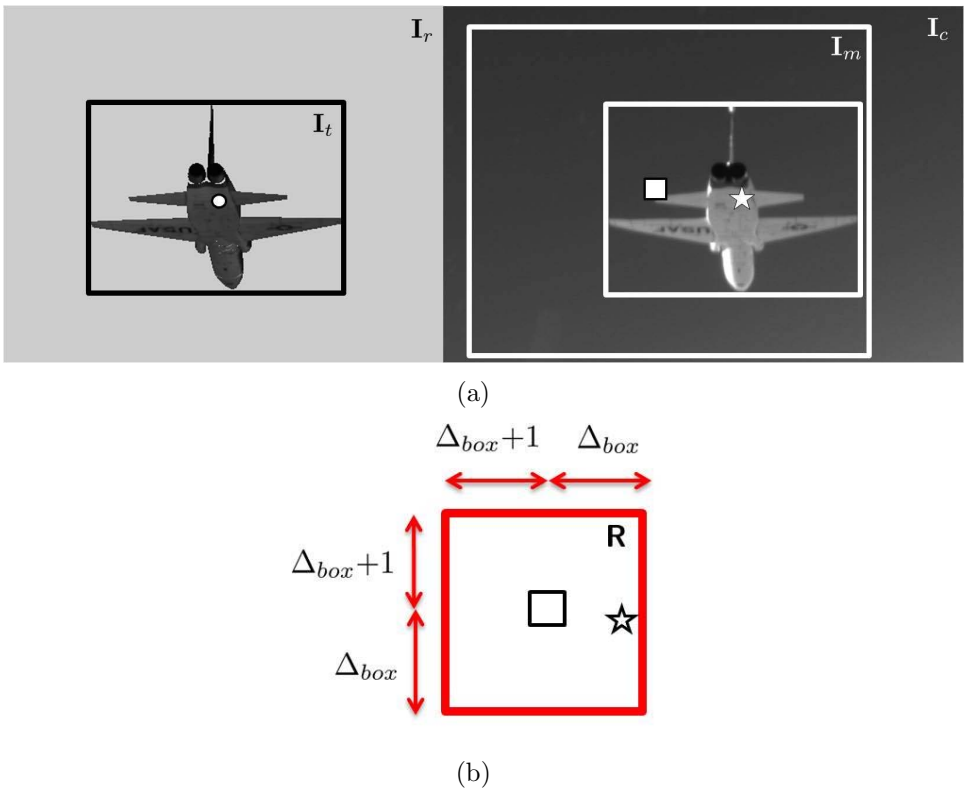


Figure 4.6: Results of template matching.
(a) The template from $\mathbf{I}_r$, left image, is matched to every possible position in $\mathbf{I}_m$, right image.
(b) The results of the matching values are returned in the **R** matrix. The star denotes the most likely match position, $\mathbf{r}_{match}$. The square denotes the center of the $\mathbf{I}_m$ and the center of **R**.

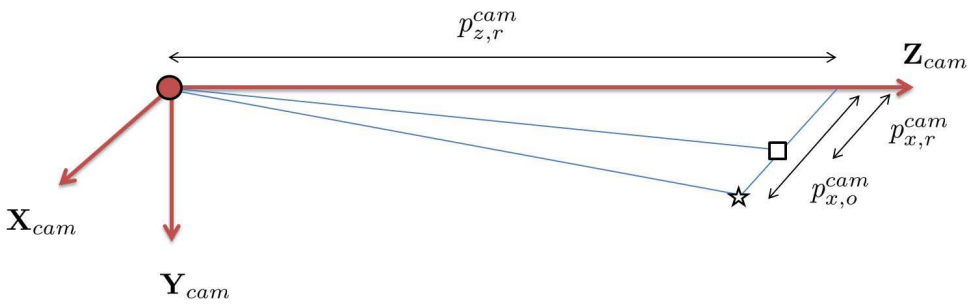


Figure 4.7: Template match accuracy. The template match approximation, requires a large ratio between $p^{cam}_{z,r}$ and both $p^{cam}_{x,r}$ and $p^{cam}_{y,r}$, to be accurate. The triangle created between the origin of the *cam*-frame and the square and star is oblique, the tangent function is a close approximation for larger ratios.

4.3.1.3 Attitude Plus Image Location. This section revokes the assumption of correct initial attitude of the object. Instead it is assumed that the rendered object's attitude is close to the actual object's attitude. If the attitude of the predicted object is grossly inaccurate in comparison to the actual object, the visual appearance of the $\mathbf{I}_r$ will most likely be different from the object in the $\mathbf{I}_c$.

This does not apply if the object is highly symmetric in some manner, like a sphere, where the attitude or portions of the attitude do not influence the visual appearance. Assuming a non-symmetric object, visually inaccurate templates will not correctly identify the image location of the object with template matching.

For small motions along the $\mathbf{Z}_{cam}$ axis (the only DOF not addressed in this group), the relative size of the object is of less importance to the visual appearance of the object. It is possible to get an accurate image location in a template match with correct attitude information and small errors in the size of the object. Determining the size of an object is presented in the next section.

This attitude coupling reduces the perturbations required to three DOFs. With small attitude motions of the object between $\mathbf{I}_c$ images and the attitude DOFs allowed to remain coupled, the accuracy of the attitude estimation increases, iterations are eliminated, and the process requires fewer $\mathbf{I}_r$ images than coupling all six DOFs. Two perturbations, plus the initial state estimate, for the three DOFs equates to 27 $\mathbf{I}_r$ images (3^3) compared to the 729 required to perturb all six DOFs with the same number of perturbations in each DOF (or 243 to perturb five DOFs, 3^5).

The complete process renders each possible combination of attitude, the estimate plus two perturbations, for each of the three DOFs. Each resulting $\mathbf{I}_r$ is then template matched against a suitable ROI of the $\mathbf{I}_c$. The attitude values associated with the most-likely match become the attitude measurement update. It is also possible to attain the image location with this most-likely attitude match at this point. Although, if an update requires the translation in the $\mathbf{Z}_{cam}$ axis, determination of the size of an object should occur before determination of the image location of the

object. This will permit a more accurate image location. Hence, this decoupled group might more appropriately be titled *attitude*; however, all together the process is the equivalent of coupling five DOFs at the rendering cost of coupling three.

A box diagram of this step in the process is shown in Figure 4.8 for two perturbations in each attitude DOF. The inputs to the system are shown on the left side of the diagram and consist of the initial estimate in orientation ($\mathbf{C}^r_{cam}$), the initial estimate in position ($\mathbf{p}^{cam}_r$), and the image collected by the camera ($\mathbf{I}_c$). The later two are unchanged by this step in the process. The most likely *attitude plus image location* results are the 6×1 vector used to render the $\mathbf{I}_{ri}$ ($i \in \{1, 2, ..., 27\}$) image

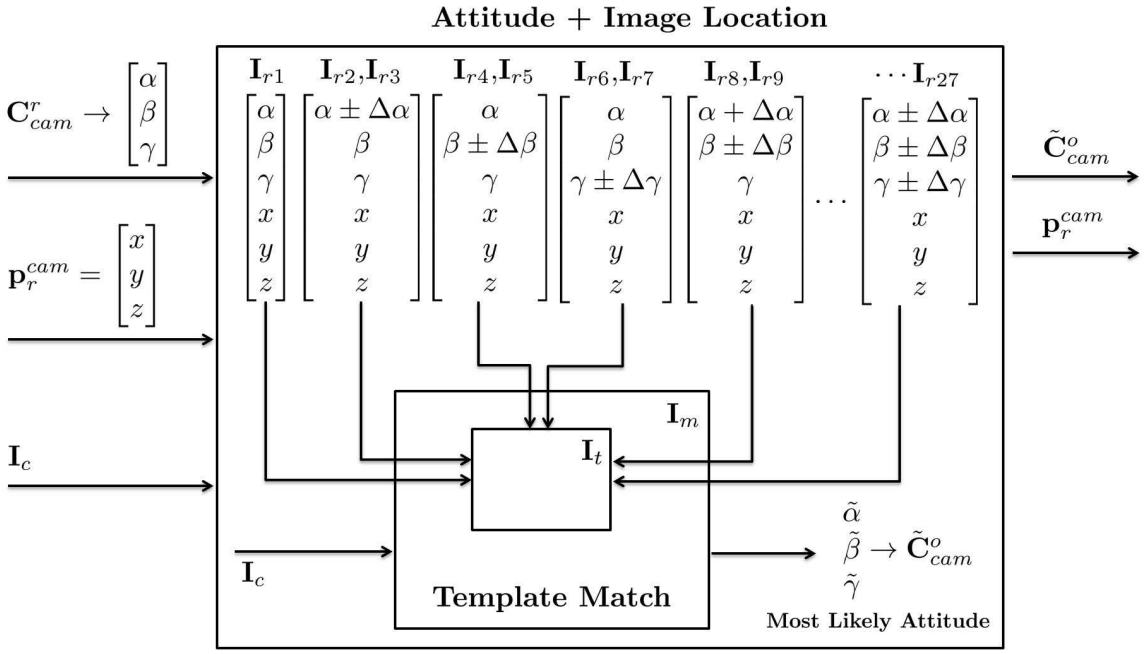


Figure 4.8: RIPE: attitude plus image location. The orientation of the rendered object in the *cam*-frame ($\mathbf{C}^r_{cam}$) is annotated with three euler angles between the *cam*-frame and the *b*-frame of the rendered object: α, β, and γ. The position of the rendered object in the *cam*-frame ($\mathbf{p}^{cam}_r$) is annotated with three individual translations in the *cam*-frame: x, y, and z. The contents of the six DOF array below each $\mathbf{I}_{ri}$ ($i \in \{1, 2, ..., 27\}$) are the parameters required to create the respective image. The perturbations are denoted as a Δ for each of the respective DOF. Each possible combination of attitude is rendered and a ROI of each becomes an $\mathbf{I}_t$ to match against the ROI of $\mathbf{I}_c$ ($\mathbf{I}_m$.) The attitude values used to render the most likely $\mathbf{I}_t$ are recombined to create the attitude measurement of the object, ($\tilde{\mathbf{C}}^o_{cam}$).

that produced the highest matching value with $\mathbf{I}_c$ (such as the correlation coefficient from Section 2.2.2.2.) Because the translations in position are the same for all the $\mathbf{I}_{ri}$ images, that portion of the vector does not add any new information. The attitude measurement of the object, ($\tilde{\mathbf{C}}^o_{cam}$), is the unique output from this step. As a reference, rendering and comparing only the first seven $\mathbf{I}_{ri}$ ($i \in \{1, 2, ..., 7\}$) images without template matching is an example of a decoupled approach.

Gross inaccuracy in size does diminish the accuracy of this approach. Fortunately, the inaccurate size affects the matching of all the $\mathbf{I}_r$ images equally, diminishing the overall effect of the error as a result of decoupling the single DOF. The next section presents the other decoupled group.

4.3.1.4 Size Plus Image Location. With correct attitude of an object, solving for the size and image location of the object remains. This process simply repeats the attitude plus image location process with only translational perturbations along the $\mathbf{Z}_{cam}$ axis. Because motions along this axis are typically challenging to determine in an image, decoupling the DOFs in this manner permits more, total perturbations about this DOF without an exponential increase in the required number of $\mathbf{I}_r$ images. The final step of this process determines the size and image location of the object ($\tilde{\mathbf{p}}^{cam}_o$) based on the most-likely value of all the templates matched, and the position in $\mathbf{R}$ of that most-likely match. The benefit of this process effectively couples the three translation DOFs.

A box diagram of this step in the process is shown in Figure 4.9 for four perturbations in the $\mathbf{Z}_{cam}$ axis DOF. With the better orientation estimation from the previous step, a rendered image should have a closer resemblance to the collected image. The most likely *size plus image location* results are the 6×1 vector used to render the $\mathbf{I}_{ri}$ (i=1:5) image that produced the highest matching value with $\mathbf{I}_c$. The $\mathbf{r}_{match}$ of the most likely match are recombined to create the position measurement of the object, ($\tilde{\mathbf{p}}^{cam}_o$).

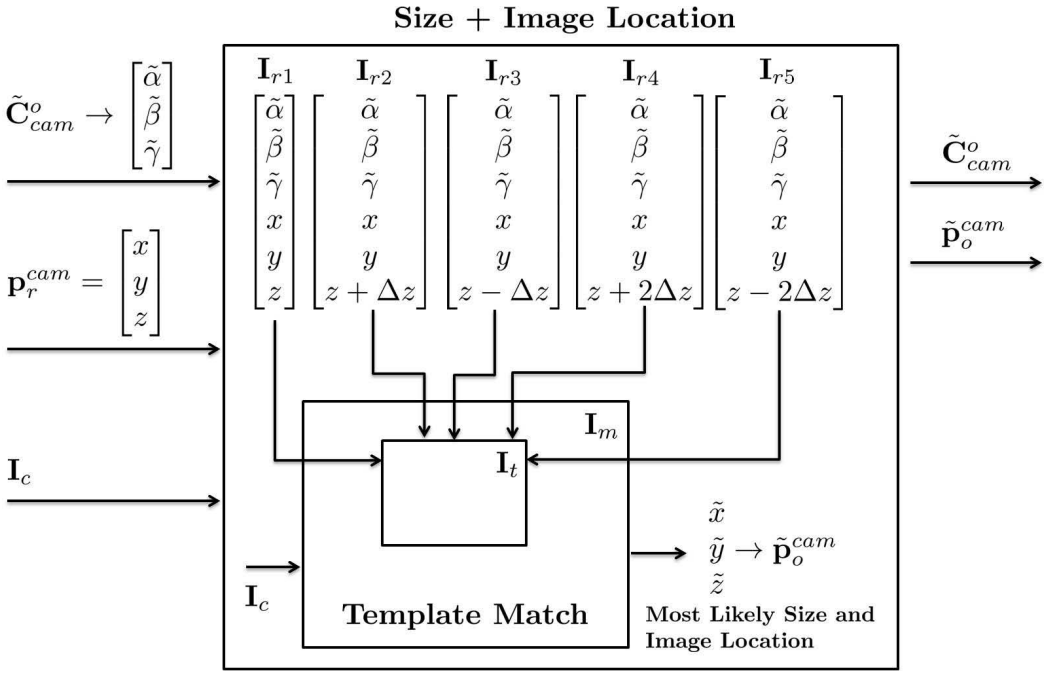


Figure 4.9: RIPE: size plus image location. The orientation of the rendered object in the *cam*-frame ($\mathbf{C}^r_{cam}$) is now closer to the actual orientation of the object. Each perturbation about the z translation (Δz) is rendered and a ROI of each becomes an $\mathbf{I}_t$ to match against the ROI of $\mathbf{I}_c$ ($\mathbf{I}_m$.) The z value used to render the most likely $\mathbf{I}_t$ and the x and y values attained through the $\mathbf{r}_{match}$ of the most likely match are recombined to create the position measurement of the object, ($\tilde{\mathbf{p}}^{cam}_o$).

By decoupling the DOFs in this manner, five DOFs are coupled at the rendering cost of three, and three DOFs are coupled at the rendering cost of one. Overall, the entire six DOF estimate is completed at the cost of three coupled and one uncoupled DOF with an increase of precision in the $\mathbf{X}_{cam}$ and $\mathbf{Y}_{cam}$ axes. The equivalent perturbations about these axes are much smaller than the perturbations of the other four. The effective size and number of the perturbations in these axes depend on the IFOV and the size of Δ_{box} respectively.

An example of a complete RIPE process diagram is shown in Figure 4.10 with the incorporation of a Kalman filter and INS updates (which are detailed further in Section 4.4.)

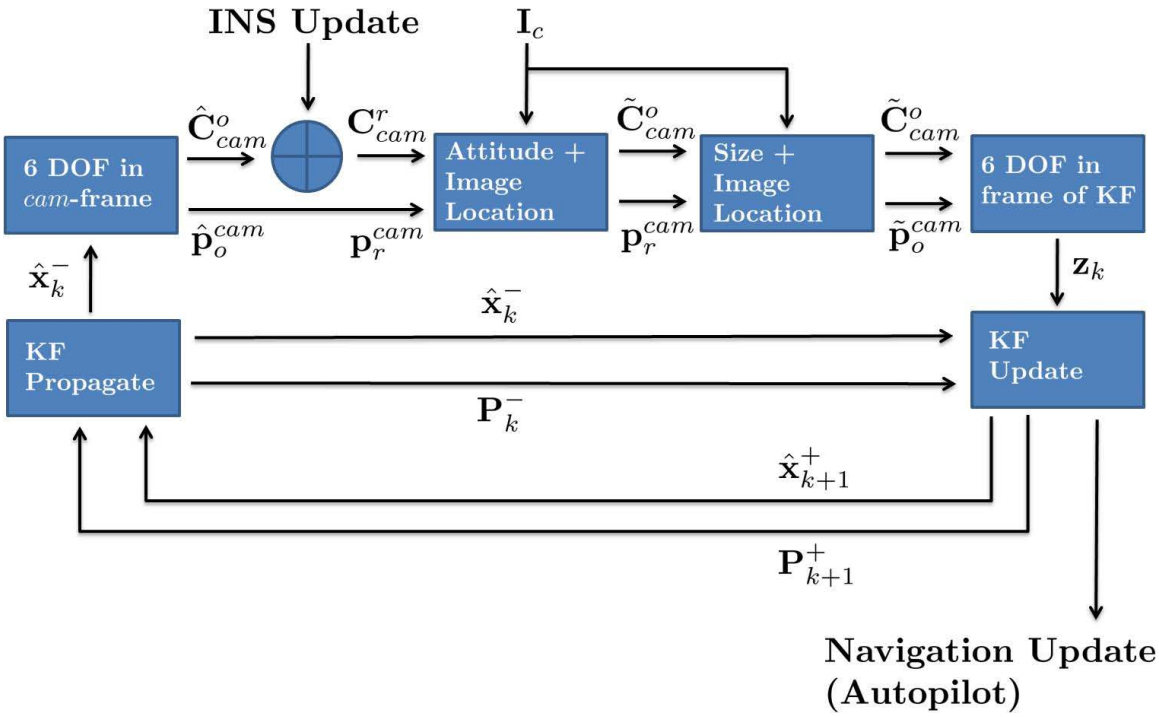


Figure 4.10: The RIPE process. The complete process with incorporation of a Kalman filter and INS updates. The process uses an *a priori* estimate converted into the *cam*-frame and incorporates the INS update to create an initial estimate of orientation and position. This initial estimate ($\mathbf{C}^r_{cam}$ and $\mathbf{p}^{cam}_r$) is rendered and template matched with $\mathbf{I}_c$ in addition to perturbations about the initial estimate. The most likely attitude, size, and image location of the object is combined into a measurement update ($\mathbf{z}_k$) for the Kalman filter to determine the relative position of the object.

Two other alternative template matching options are presented as examples of additional coupling possibilities. Because of time limitations, they were not further explored for this research.

A digital zoom of the $\mathbf{I}_r$ images (expanding and contracting the size of the object) can create a three-dimensional template matching, with a known relationship between the zoom and perturbations in the $\mathbf{Z}_{cam}$ axis. A scale-invariant template matching process would reduce the entire six DOF estimation to three DOFs while increasing the precision in the $\mathbf{Z}_{cam}$ axis.

Second, the first rotation of the object in the scene could be a rotation about the $\mathbf{Z}_{GLcam}$ axis, instead of the $\mathbf{Y}_{GLcam}$ axis used in this research. Digitally rotating the

template about its center, and creating a circular match-ROI, removes an additional DOF rendering cost and increases the precision in that rotation.

As a last note on the quick-RIPE process, it also accounts for objects near the edge of the $\mathbf{I}_c$ by artificially clipping the four sides of the $\mathbf{I}_r$ by the size difference Δ_{box}. This partially limits the usefulness of the $\mathbf{I}_r$, but allows the $\mathbf{I}_m$ to always be $2\Delta_{box}$ larger than the $\mathbf{I}_t$ in both dimensions. This also allows the matching process to determine translations in both axes equally, instead of artificially limiting the matching to directions toward the center of the image. For example, if the edge of the $\mathbf{I}_t$ was allowed to be the edge of the $\mathbf{I}_r$, the $\mathbf{I}_m$ would be on the edge of the $\mathbf{I}_c$ as well (since there is no image beyond this point to create a match-ROI around) and the template matching could only match the current location or translational motion towards the center of the $\mathbf{I}_c$.

After determining the entire six DOFs, the pose of the object in the *cam*-frame is complete. The next section applies this process to AR, further reducing the number of $\mathbf{I}_r$ images required.

4.3.2 RIPE Tailored to AR. The knowledge of a tanker aircraft's relatively benign and predictable motion during AR allows further tailoring of the RIPE approach. The analyzed statistical motions of the aircraft determine the necessary range, direction, and magnitude of the perturbations about the nominal state. Decreasing the number of perturbations about one or more states permits more perturbations in other more dynamic states with the same cost to rendering, processing, and matching time.

In AR, the states with the most motion as determined through empirical data are roll and translation in the $\mathbf{X}_{b_L}$ axis, as presented in Chapter 3. Even when the frame of motion is rotated to the *cam*-frame, the motions are shown to be related.

The forward motion of the aircraft (a combination of north and east motion, depending on the heading of the aircraft) rotates to a combination of motion in the $\mathbf{Z}_{cam}$ and $\mathbf{Y}_{cam}$ axes. For this research, the angle between the $\mathbf{Z}_{cam}$ of the camera on

the wing aircraft and the $\mathbf{X}_{b_L}$ axis was approximately 30°. At this angle, a majority of the change in forward motion is rotated into the $\mathbf{Z}_{cam}$ axis ($\cos(30°) = 0.866$). The motion in this axis of the camera is modeled the same as the north or east motion in the n-frame.

Fifty percent of the change in forward motion is rotated into the $\mathbf{Y}_{cam}$ axis ($\sin(30°) = 0.5$); however, motion in both this axis and $\mathbf{X}_{cam}$ axis are accounted for with the template matching, as long as Δ_{box} is large enough to cover the potential motion in these axes.

The roll motion of the lead aircraft is accounted for in the *cam*-frame through the following relationship: $\mathbf{C}_{cam}^{b_L} = \mathbf{C}_{n}^{b_L}\mathbf{C}_{b_W}^{n}\mathbf{C}_{cam}^{b_W}$. Separating the rotations in this manner, allows the known dynamics of $\mathbf{C}_{n}^{b_L}$ to be realized in $\mathbf{C}_{cam}^{b_L}$. This separation also allows the inclusion of INS data, detailed in the next section. Since $\mathbf{C}_{cam}^{b_W}$ is constant and known and $\mathbf{C}_{b_W}^{n}$ is known from the INS, the remaining DCM, $\mathbf{C}_{n}^{b_L}$, can be modeled as presented in Chapter 3. This relationship allows the lead aircraft to be tracked in the *cam*-frame using the dynamics demonstrated in the n-frame. In other words, the actual values of $\mathbf{C}_{n}^{b_L}$ and $\mathbf{C}_{cam}^{b_L}$ will be different, but the unknown variability of $\mathbf{C}_{cam}^{b_L}$ will be the same as $\mathbf{C}_{n}^{b_L}$, with the knowledge of the other two DCMs.

Modeling the lead aircraft in this manner tailors the quick-RIPE process. The attitude plus image location reduces to perturbations in the roll DOF only. With this reduction to one DOF, increasing the number of perturbations, incrementally increases the number of $\mathbf{I}_r$ images required. The algorithm's approach used four perturbations about the state estimate, $\pm 1°$ and $\pm 2°$. The other two rotations are checked less often when necessary. The algorithm presented in this research updated yaw every *second* during straight and level flight and every half *second* during turns, pitch was updated every two *seconds*. These were added as an additional *attitude plus image location* group and not included in the initial group with the roll DOF. This required, at some cost to precision, only five additional $\mathbf{I}_r$ instead of 20 additional to

couple the rotations with roll (roll already required five $\mathbf{I}_r$, two DOFs coupled requires 25 $\mathbf{I}_r$ total)

The size plus image location group retains the translation about the $\mathbf{Z}_{cam}$ axis perturbations. The RIPE algorithm's approach used four perturbations about the state estimate, ± 5 *inches* and ± 10 *inches*. This can eventually be tailored as a function of distance between the camera and aircraft. Perturbing by such a small amount for aircraft farther away, might not change the visual appearance of the aircraft. In fact, for aircraft farther away, the algorithm can be reduced to attitude plus image location, with only periodic updates to size.

The DOFs used and their perturbation amounts partially account for the potential motions of the wing aircraft, which can vary depending on the platform and pilot. Without other sources of data, proof of this process and the values chosen is limited to the system it was designed for.

Through this approach, the entire quick-RIPE process requires ten $\mathbf{I}_r$ images per single $\mathbf{I}_c$ with extras required periodically and at an increased rate during turns. Thus, the process is reduced to *attitude*, *size*, and *location*. The final portion of this chapter, details the implementation of this process with a Kalman filter.

4.4 Integrated RIPE

Applying the RIPE process to tracking a tanker can include an interaction with a Kalman filter designed to track the aircraft. The Kalman filter dynamics in Chapter 3, modified appropriately for tracking the lead aircraft in the *cam*-frame, uses the output of the modified-RIPE algorithm, three translational DOF, and the roll DOF, as measurement updates, $\mathbf{z}$. The remaining items of the filter, not already presented, consist of INS attitude updates and the noise of the measurement, $\mathbf{R}$.

4.4.1 INS Update. The INS interaction with the filter included attitude updates only. Other useful information is available from the INS; however, additional mechanization in the filter is necessary to incorporate it. To ensure the statistical

continuity of the Kalman filter, the INS does not modify the state information directly, rather it uses the *a priori* state of the filter, $\hat{\mathbf{x}}_k^-$ to create the initial estimate for the RIPE process. The state is updated to $\hat{\mathbf{x}}_k^+$ after the RIPE measurement and does not directly include the updated INS information. This effectively separates the INS update from the statistical accuracy of the filter, while using the information as a better initial estimate for the RIPE process.

The INS information updates the estimate by separating $\mathbf{C}_{cam}^{b_L}$ into $\mathbf{C}_n^{b_L}\mathbf{C}_{b_W}^n\mathbf{C}_{cam}^{b_W}$. The INS information updates the $\mathbf{C}_{b_W}^n$ rotations. The other values ($\mathbf{C}_n^{b_L}$ and $\mathbf{C}_{cam}^{b_W}$) are kept constant, and then recombined to create the initial attitude estimates for the next RIPE measurement. Other state information from $\hat{\mathbf{x}}_k^-$ is used to create $\mathbf{p}_L^{cam}$ for the initial translational estimate in the RIPE process.

Incorporation of the INS attitude information accounts for some of the attitude motion of the wing aircraft in the tracking process. Without the inclusion of the INS information, the movement of the tanker aircraft in the *cam*-frame would not as closely resemble the predicted motion from the analysis as in the *n*-frame. If INS information were not available, increasing the noise dynamic model of the filter ($\mathbf{Q}$) could partially account for this unpredicted motion of the lead aircraft.

4.4.2 Kalman Filter Measurement Noise. The last remaining requirement of the Kalman filter is the noise of the measurement ($\mathbf{R}$). The value of $\mathbf{R}$ depends on the independent accuracy of the RIPE algorithm. To determine the accuracy of RIPE, a sample run of AR collected images was processed by RIPE with no filtering, producing relative position estimates. Without a filter, the estimate of position and attitude from a previous time epoch became the initial six DOF estimates for the next time epoch. Yaw and pitch were estimated every two and three *seconds* respectively. The data had minimal noise in the image and minimal extraneous movements of both aircraft.

The results of the sample run are shown in Figure 4.11. The run lasted 77.6 *seconds*, and accomplished 776 measurement updates at ranges from -1500 to -740

inches in the $\mathbf{X}_{b_L}$ axis. As a reference, the pre-contact position is approximately -1150 *inches* and the contact position is in the range of -830 to -690 *inches*. The wing aircraft maintained a near constant $\mathbf{Y}_{b_L}$ axis translation at approximately 13 *feet*. The run was accomplished at 30° aspect angle, typical for AR. Portions of the aircraft were occluded from the camera FOV from approximately 46 *seconds* to 51 *seconds*. There was a loss of truth data at 10 *seconds*, denoted with a spike in the graphs. The data were rotated into the b_L-frame for determination of the AR positions presented in Chapter 3. The analysis of the data is shown in Table 4.1.

4.4.3 RIPE Errors. First the errors in $\mathbf{X}_{b_L}$ and $\mathbf{Z}_{b_L}$ both decrease with decreasing range between the aircraft. The errors in $\mathbf{X}_{b_L}$ are approximately 3-4% of the range, 24 *inches* in error at 700 *inch* range (two feet error in contact position), and 50 *inches* in error at 1500 *inch* range. The $\mathbf{Y}_{b_L}$ can be seen to have the same trend, *if* the removal of the bias placed the errors on the positive side of the error scale (the $\mathbf{Y}_{b_L}$ error would start with a higher *positive* error, reducing as the aircraft closed). This is a function of resolvable distance at further distances where the blurring of the aircraft in the $\mathbf{I}_c$ accounts for a larger percentage of pixels of the aircraft. This blurring area of pixels makes it harder to make a precise match compared to closer distances. This is also witnessed in the noisiness in error at those further distances. The error in $\mathbf{Y}_{b_L}$ is smoother because motion in this axis was minimal.

Second, the error in roll has a definite discrete aspect to it. This is attributed to the discrete perturbation allowed in roll (±1° and ±2°). This error can be decreased with smaller perturbation values. Further discussion on the viability of the RIPE approach will be discussed in Chapter 5.

From the error analysis, only errors in roll met the Kalman filter requirement of zero mean. The origins of the biases in the errors presented, and the errors to be presented in Chapter 5, were not fully determined. The standard deviation of the accuracy of the truth data was 18 *inches*, the biases shown are, at least partially, truth data biases. Visually, the $\mathbf{I}_c$ and $\mathbf{I}_r$ are similar, as shown in Figure 4.12. The

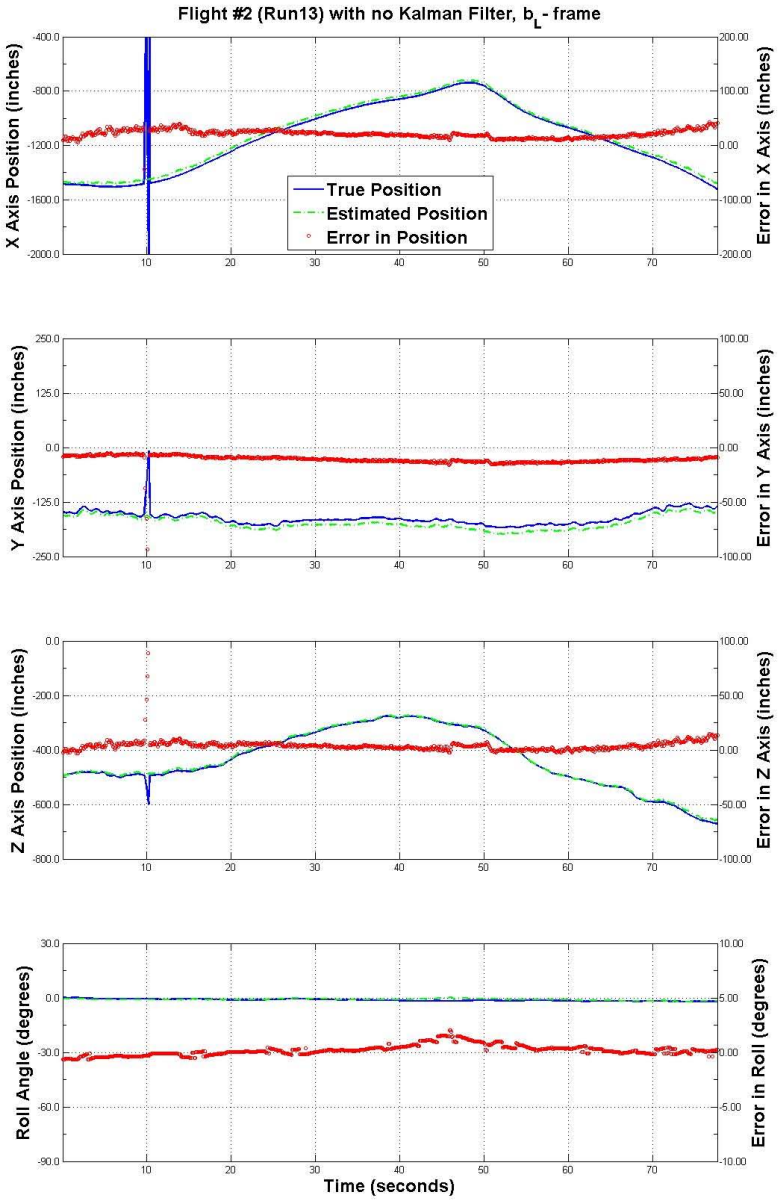


Figure 4.11: Accuracy of RIPE algorithm. A test run of the RIPE algorithm, with no filtering of the measurements, a statistical analysis of the performance, shown in Table 4.1, determines the values of $\mathbf{R}$ in the Kalman filter. The top chart is the position of the wing aircraft in the $\mathbf{X}_{b_L}$ axis, as the solid and dotted lines move towards the top of the chart, (time = 45 through 50 *seconds*), the wing aircraft is closer to lead, similar for the third chart, the position of the receiver in the $\mathbf{Z}_{b_L}$ axis. The second chart is the wing aircraft in the $\mathbf{Y}_{b_L}$ axis, or translations left and right. The final plot is the roll position of the lead aircraft.

DOF	Max (-) Error (*inches*)	Max (+) Error (*inches*)	Mean Error (*inches*)	Standard Deviation (*inches*)
$\mathbf{X}_{b_L}$ axis	6.0	42.0	21.0	7.0
$\mathbf{Y}_{b_L}$ axis	-16.0	-5.0	-10.0	2.5
$\mathbf{Z}_{b_L}$ axis	-15.0	3.0	-3.5	3.0
Roll	-2.0	0.5	0.0	0.5

Table 4.1: Data analysis of RIPE measurement error. Data was rounded to the nearest half *inch*.

standard deviations shown in Table 4.1, squared, determined the appropriate values in $\mathbf{R}$. From experimental experience the best match possible, $\mathbf{r}_{coeff\_normed}$, was approximately 0.85 (unit-less) and the worst value that appeared (visually) to still match was approximately 0.55, a sliding scale between these values increased the value of $\mathbf{R}$. This accounted for measurements with increased visual noise.

This concludes the chapter on pose, background information on AAR including pose applied to AAR, and finally this thesis' approach to AAR using RIPE. The next chapter presents the application of RIPE to real-world collected data and error analysis.

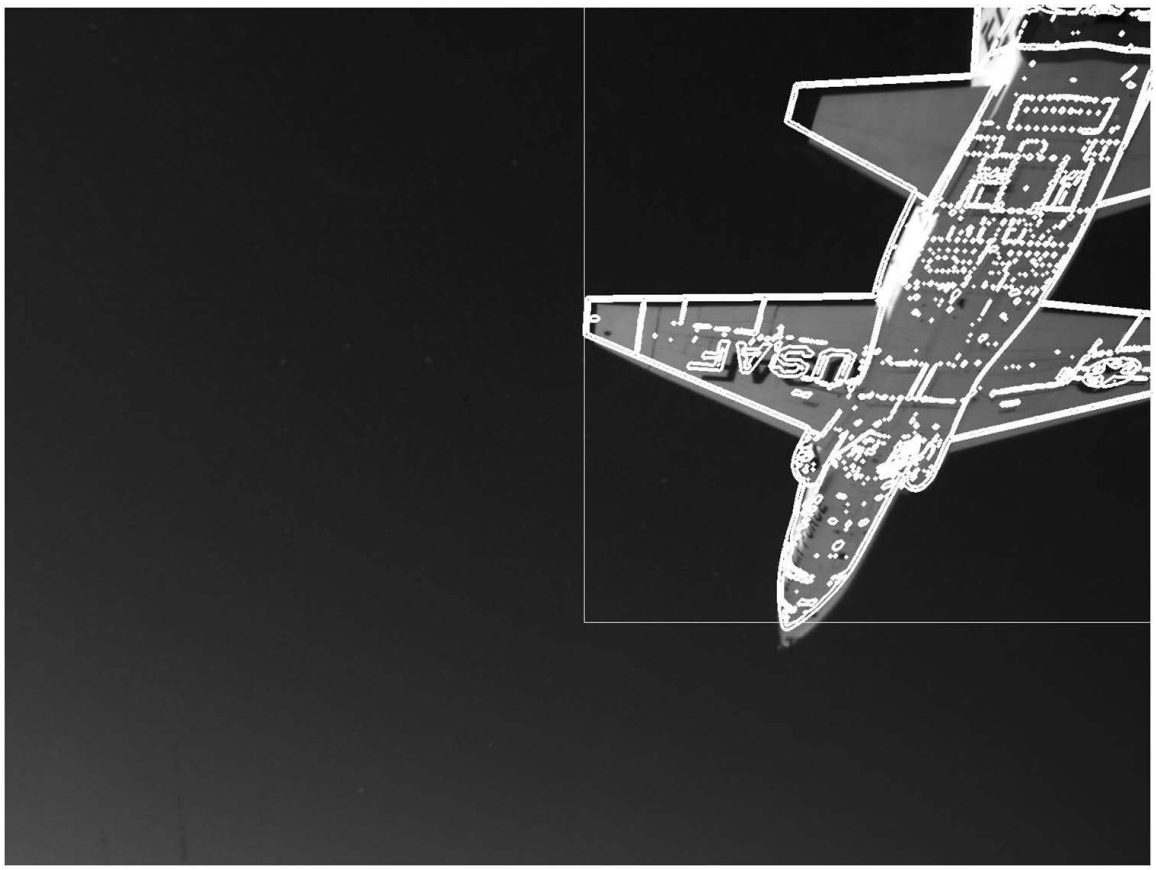


Figure 4.12: The RIPE algorithm tested with no filtering. The contour of the I_r is shown overlayed on the $\mathbf{I}_c$ at around 48 *seconds* in the data run shown in Figure 4.11. The images are close enough for a measurement update, but not perfect.

V. Experimental Results

Relative navigation using images for the purposes of air refueling an aircraft autonomously, involved every aspect of the previous four chapters. The algorithm as presented in Chapter 4 estimated the position of a lead aircraft in the camera's field of view enabling the accurate determination of a relative position of the wing aircraft for AAR. With a Kalman filter and INS updates, the research algorithm successfully met three of the four goals of this research; it increased the accuracy of the RIPE approach with open source libraries and did not require modification to the lead aircraft. However, the time required to implement the solution exceeds what the author believes is acceptable for an autopilot response. This chapter covers the creation of the models, laboratory and field work, experimental data collection, estimation errors, and analysis of the process.

The first step in both experiments involved creating a model of the intended target. While it is possible to apply the RIPE process without an *a priori* model, an attempt was made to ensure that the errors in this approach to pose were not from modeling. The first section covers the creation of models for both the laboratory and field work.

The laboratory work consisted of a basic box-shape aircraft in a Vicon® environment. The work was a risk management step to ensure a rendered image approach to pose was possible and ultimately the work led to the development of the quick-RIPE algorithm. The Vicon® system provided both a very controlled environment and a precise truth collection process. Images of the box-aircraft were collected simultaneously with its true position and the true position of a camera relative to a locally defined Vicon® navigation frame. While collecting the images, the camera was randomly moved at different angles and distances to the box-aircraft. The work evaluated the RIPE method applied to two independently moving objects without the use of INS data or a Kalman filter and with partially occlusion of the box-aircraft (up to half of the object out of the FOV) in the images. Because of background and foreground noise, the early version RIPE algorithm used contour images of both the

$\mathbf{I}_r$ and $\mathbf{I}_c$ for template matching. Lessons learned in the lab improved the algorithm for the field work.

The field work mimicked the laboratory work in a real-world stochastic, non-deterministic environment. This work focused on the incorporation of differing data sources, INS data, and a Kalman filter into the navigation solution. This portion of the process entailed flights with a TPS T-38A Talon aircraft as a simulated tanker and a Calspan LJ-25 Learjet as a receiver. The aircraft flew maneuvers representing those flown by actual tanker and receiver combinations in the operational realm. The aircraft were flown by experienced pilots enrolled in TPS. A camera mounted on the dash of the LJ-25 captured images of the the T-38A while flying representative refueling maneuvers. Multiple truth collection sources provided an initial position for the RIPE algorithm and a validation of the process and its accuracy. The RIPE algorithm was then executed using the recorded data after landing. The algorithm was causal, but slowed to allow processing.

5.1 Model Creation

The models used in this research are simply a collection of coordinates for each individual polygon that make up the entire model. When a program renders the model, the OpenGL library produces a visual representation on screen, as detailed in Section 2.2.1. In modeling the aircraft for the predictive rendering portion of the algorithm, absolute precision is not a requirement. However, accuracy and scale *are* important [18].

Accuracy is necessary to provide quantitative comparisons between the collected and rendered images because incorrectly placed items on the aircraft will negatively affect the solution. The approach presented in this research depends on determining which rendered image out of a certain number most likely resembles the collected image. If an engine pod is modeled at the wrong position on the aircraft, images of this incorrect model, rendered at incorrect locations relative to the camera may appear more likely to the algorithm than those rendered at the correct location.

Scale is very important when determining the distance between aircraft. Using an aircraft modeled with a shorter wingspan than the actual aircraft will result in a shorter estimate than the actual solution. Incorrect scaling of other parts of the aircraft will affect the solution similarly. As shown in [18], an error of 20% in the model resulted in a four-fold increase in the tracking error.

Creating a model from engineering diagrams is not always possible. Assuming such diagrams exist, they would not necessarily include modifications, alterations, or even paint schemes. Completing the model requires determining and including these items. Some of the options to create three dimensional models of aircraft include laser scanning, photogrammetry, and collecting point measurements.

Laser scanning is very accurate but currently requires expensive equipment and training. As the costs continue to drop and equipment evolves to become more user-friendly, this option may become viable in the future. Models created using laser scanning still need textures or images added to produce a rendered image representative of the actual aircraft.

At the cost of some of the precision of laser scanning, advances in photogrammetry software allow very accurate three-dimensional modeling with the use of images from a known calibrated camera. Photogrammetry utilizes the science of multi-view perspective geometry. This science is similar to triangulating a navigation position based on distances from three or more known reference points. Photogrammetry requires combining several images of an object, each with an image location of a characteristic feature or defined point of the object. Enough features identified in multiple images determines the relative positions of all the characteristic points in three-dimensional space. Combining the relative position of the characteristic points creates a three-dimensional model. Correct scaling of the entire model only requires the addition of a single verified distance between any two points on the object being modeled.

This software imports the images of the object to be modeled and the user references characteristic points in as many images as possible. When referenced across a few images, the software automatically computes epipolar lines drawn on unreferenced images. These lines help the user visually see where the point should be in the image and determine how close the points match up with the model the software is creating.

A benefit of photogrammetry is the automatic inclusion of images with the model creation. Multiple images texture mapped to the object creates a more realistic model.

The laboratory work of this research utilized PhotoModeler®, a commercial software suite, to build a basic three-dimensional model of a simple wooden airplane. Such a process was not necessary for such a basic shape, but was a proof of concept for potential use on an actual aircraft. An overview of the PhotoModeler® process is shown in Figure 5.1.

A problem encountered with the use of photogrammetry is the difficulty in locating specific points to reference between photos, a problem of correspondence. Various textures, dots, or symbols, applied to an object, can make the process easier, and in some cases automated. Covering an entire aircraft with these textures is difficult and time consuming.

Because of the limited downtime in the aircraft's flight schedule, the field work modeling effort attempted to use PhotoModeler® without the use of the dots or symbols. Unfortunately, the lack of distinguishable features on the aircraft (almost completely white) made the process impossible without considerable effort to locate individual rivets and joints in multiple portions of the aircraft. This method was abandoned because of the failure to place characteristic points to reference on the aircraft.

Eventually, two other modeling methods created two different types of models, one of which was used for the RIPE algorithm. Both models began with a wire frame

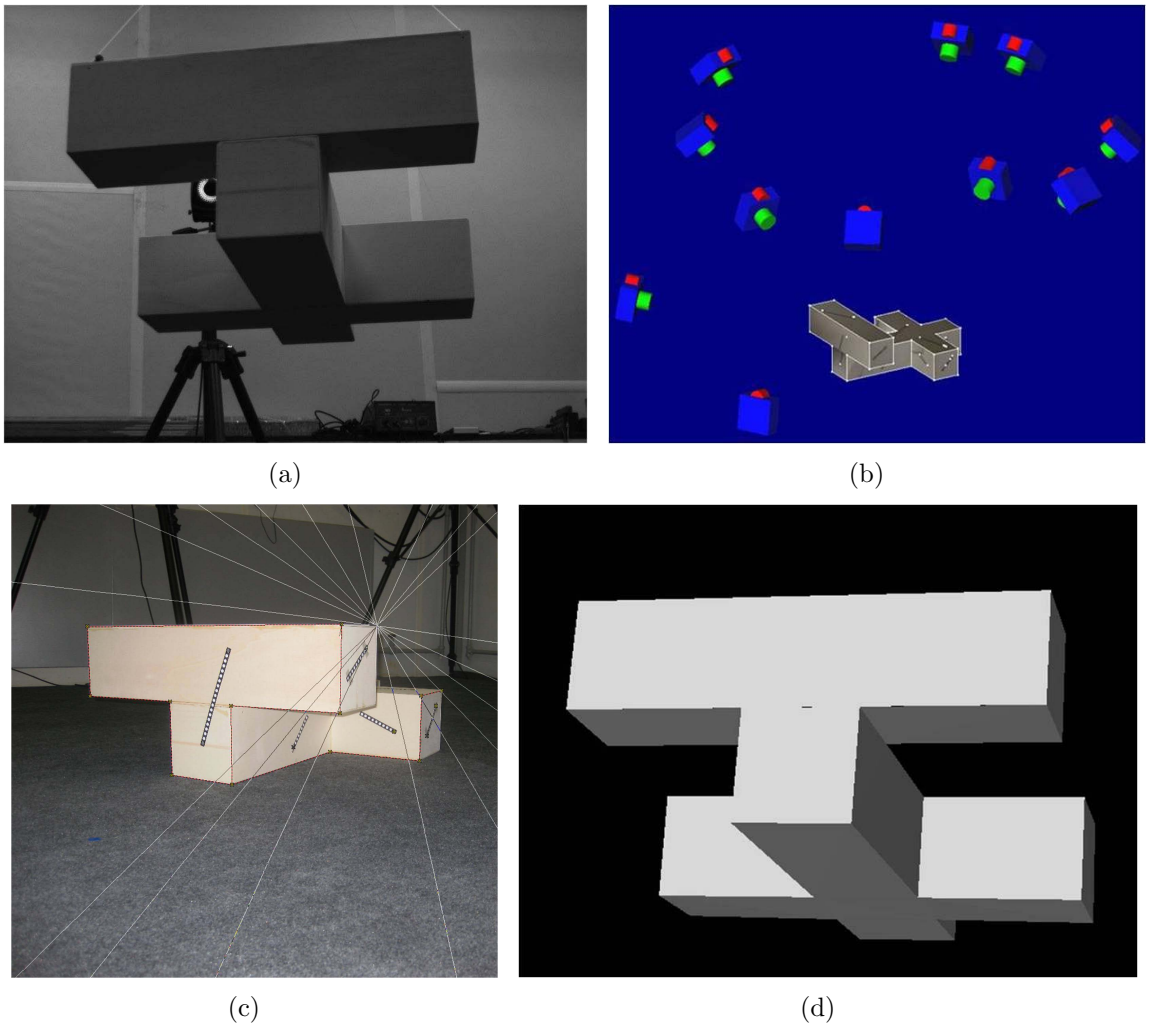


Figure 5.1: Creating a 3-D model with PhotoModeler®.
(a) To minimize estimation errors associated with an object, a simple box frame aircraft was constructed out of spruce plywood.
(b) Pictures of the box-aircraft were taken from various angles and distinguishable features were cross-referenced in PhotoModeler®. Including, but not shown, pictures taken from underneath.
(c) After referencing the same feature in multiple images, PhotoModeler® presents the user with epi-polar estimation-lines for reference-point determinations.
(d) After exporting the model as a collection of polygons, the aircraft can be rendered using the OpenGL library.

diagram built with the original specifications from Northrop Grumman, shown in Figure 5.2. Over its lifespan, the aircraft was altered with one particular alteration visible in some of the rendered images. A flat disk around the top of the vertical stabilizer at the rear of the aircraft was not in the original design. This could possibly contribute to some errors in roll when highlighted by the sun during flight. The error in the model was not noticed initially and was never corrected.

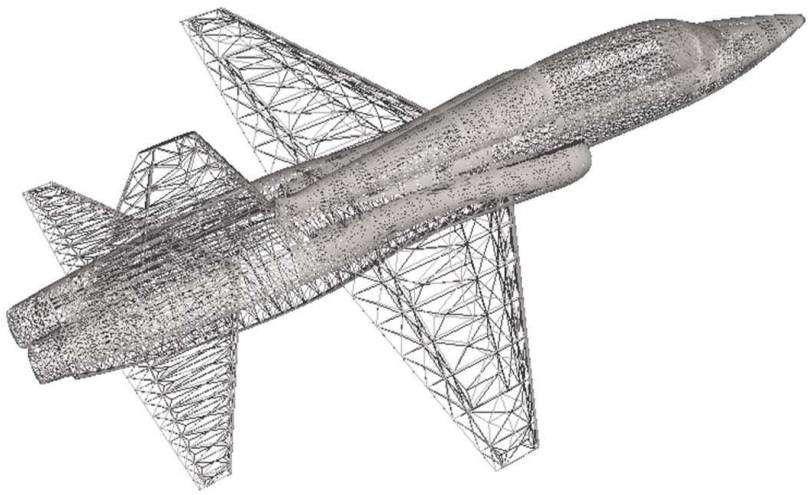


Figure 5.2: Initial wire frame model. This was the beginning model used in the research, created using the original aircraft specifications from Northrop Grumman.

A model created and used for this research projected and then attached an underside photo of the aircraft to the bottom surface of the wireframe model. This only necessitated a single photo collected during flight. The photo was applied in individual sections so it did not appear as a flat image on the bottom of the aircraft. This is important during the rendering process because flat surfaces will not change appearance in the same manner as contoured surfaces. By maintaining the shape of the wire frame model with the photo texture, realistic lighting and shading can appear on the aircraft. This model, with a close up view from underneath the aircraft, is shown in Figure 5.4 (a) and (b).

There are a few caveats with this approach. First, and probably most importantly, the process requires a picture of the bottom side of the aircraft. Unless a hoist

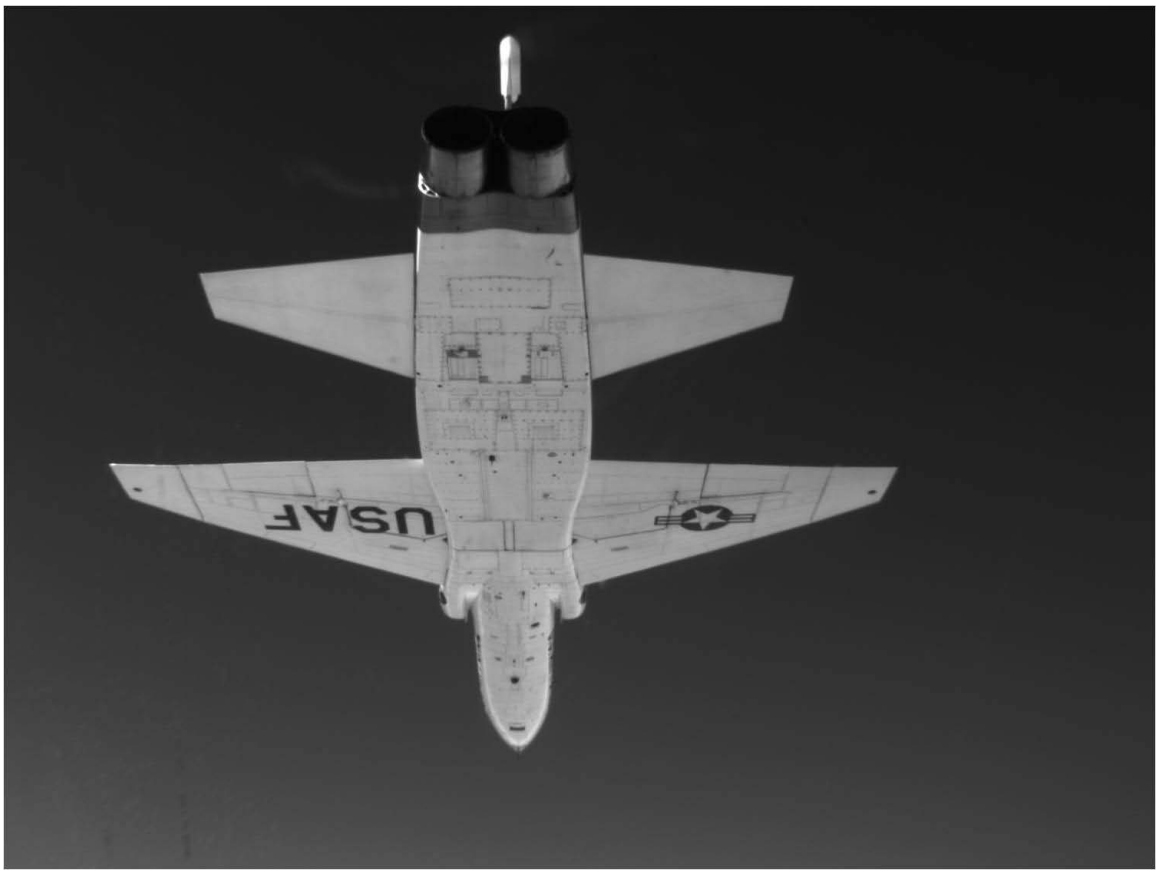
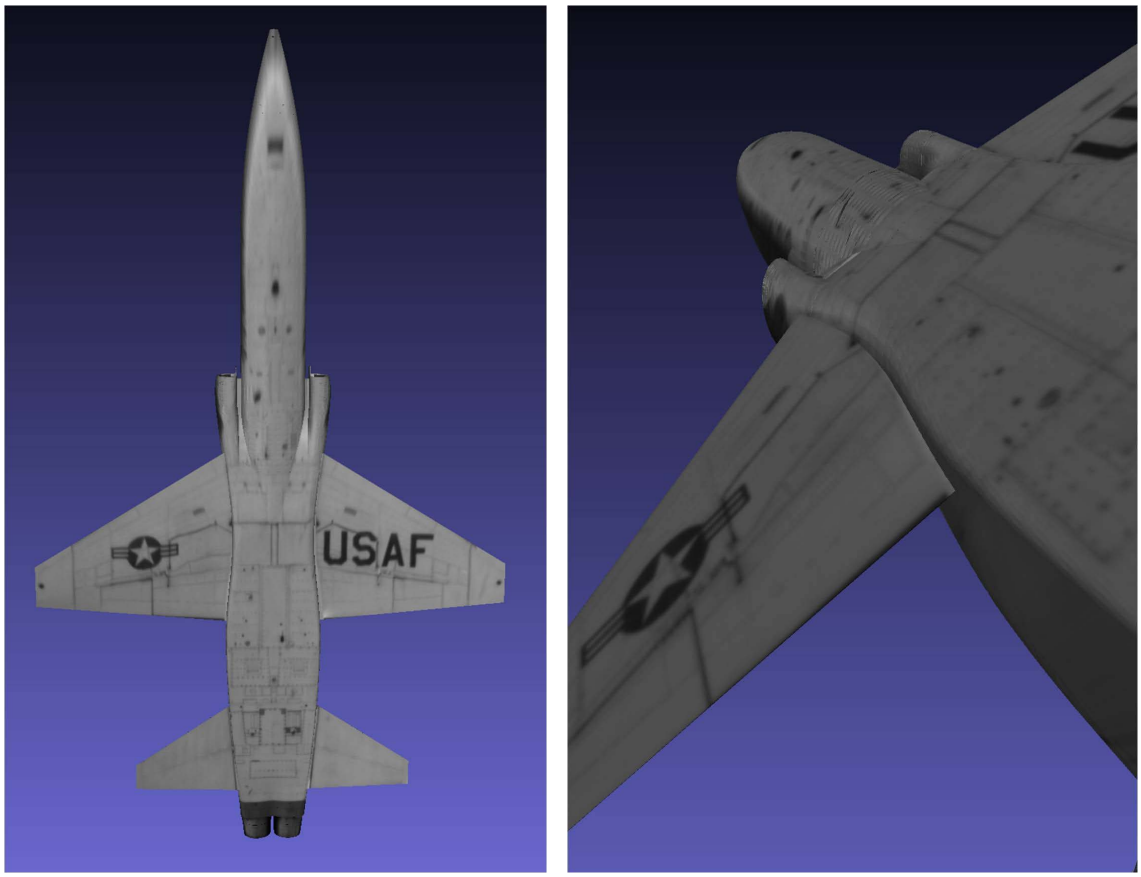


Figure 5.3: Texture image. This image, taken in flight, was projected on and applied to the underside of the wireframe model of the same aircraft (shown in Figure 5.2) to create the model shown in Figure 5.4 (a) and (b).



(a) (b)

Figure 5.4: 3-D model with texture mapping.
(a) A photo taken of the bottom of the aircraft in flight was applied to the wire frame model shown in Figure 5.2.
(b) A close up view of the texture-mapped model, showing the changes in contour of the aircraft.

or crane can lift the aircraft (with the gear retracted) or stitching together multiple individual-pictures from underneath the aircraft is possible, the process involves collecting the image(s) in flight. Even with photos taken on the ground, an aircraft looks different on the ground (wings bend down) than it does in flight (wings bend up) due to aerodynamic effects, this was minimized in this research but still noticeable. In-flight photography is expensive and it's difficult to determine the distortion effects of the windscreen that the photographer will use. As a solution, the same camera used in the research with a known distortion model also collected the image for the texture mapping.

Second, attaining a good perspective image of the aircraft is difficult. In Figure 5.4 (a), the image appears smeared near the nose of the aircraft. This *smearing* occurs, because it would be difficult to fly directly below the aircraft and take a picture from underneath; therefore, the image was taken at an angle from behind. The rotated model shows the projective distortion in the photo applied to the bottom surface. In Figure 5.4 (b) the perspective of the aircraft is similar to the perspective when it was taken, reducing the effect of the distortion.

Another modeling technique facilitated the creation of a backup model. For this model, the entire wire frame was colored white and specific visual-textures were applied to the aircraft and colored black. For accurate truth collection in the research, it was necessary to know the location of the truth-data and image collection equipment by *boresighting* their location and orientation. The Faro® arm equipment used to boresight those devices also precisely mapped some of the paint schemes of the aircraft. Applying the determined positions of the paint schemes to the model and coloring them appropriately created a more accurate textured-mapped model than other techniques. The completed model is shown in Figure 5.5.

This process also has problems. First, it's expensive; the equipment, technician, and the aircraft down-time is almost as costly as flying. Second, the paint scheme locations were precise but not absolute and minor errors in their locations were dis-

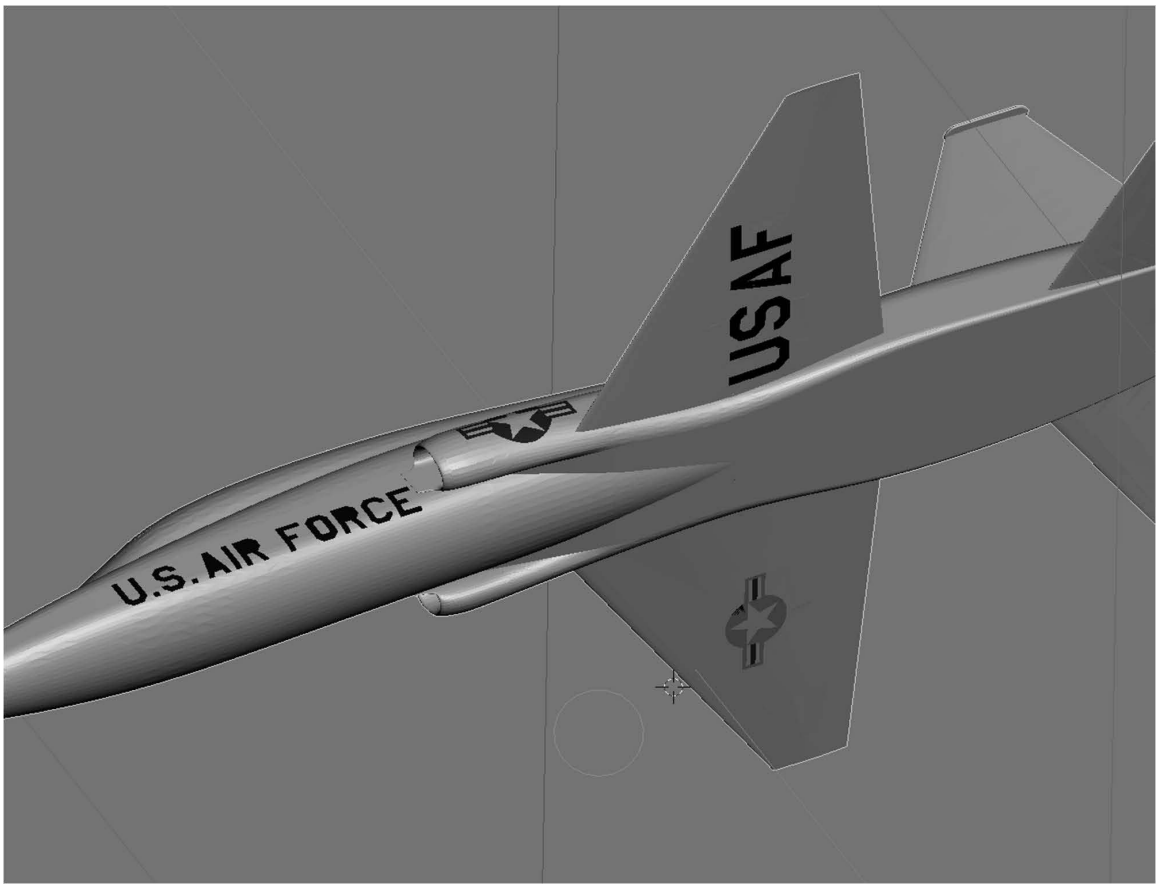


Figure 5.5: A model created with precise paint scheme points. This model was not ultimately used for this research, but is a better representation of the aircraft for future research.

covered. This presented issues when applying the color to the skin of the aircraft. If a coordinate is incorrect towards the inside of the skin, the color is not visible when rendered (the white skin of the wire model covers it). Similarly, if a coordinate is incorrect to the outside of the skin, the color appears to float, unattached to the body of the aircraft. To correct this, many of the points had to be translated along a line normal to the surface of the aircraft, so they could be *on* the skin and seen in renderings.

Finally, the natural curves of the aircraft makes this process challenging. Because of the sheer number of points required, it's difficult and time consuming to collect every point, of every aspect of the paint scheme. Because of this, the Faro® arm collected only well defined features. Instead of collecting the entire left side of the "U" at the front of the aircraft, the Faro® arm only determined the corners that define the straight lines of the "U". Unfortunately, the "U" is not straight at all, as it curves along the side of the aircraft. When inputting these two points onto the model, the straight line between them would pass through the skin of the aircraft. These lines were hand-curved to the side of the aircraft as well as translated to the surface.

No attempt was made to determine if the quality or type of model affected the results of the navigational approach presented in this thesis. With a model created, the next step involved preliminary evaluation of the process in a laboratory setting, detailed in the next section.

5.2 Laboratory Work

The laboratory work made use of the Vicon® motion capture system to provide the true location of the simulated aircraft. The Vicon® system has an advertised accuracy of approximately 1 *mm*. Images were collected using a Prosilica® GC 1290C camera with an 8 *mm* lens (simulating a wing aircraft). With this setup, representative refueling motions were *hand flown*, by moving the camera only, at distances

ranging from 60 to 160 *inches*. Images of the box aircraft were collected at 10 *Hz* during the simulated flight.

The model was constructed out of spruce plywood and was 24 *inches* wide, 32 *inches* long, and 16 *inches* tall (shown in Figure 5.6(b) and Figure 5.1). To allow the camera to collect images below and aft of the box-aircraft, it was suspended with fishing wire to minimize the visibility of the suspension wires in the collected images. The process discussed in Section 5.1 details the creation of the virtual representation of the model.

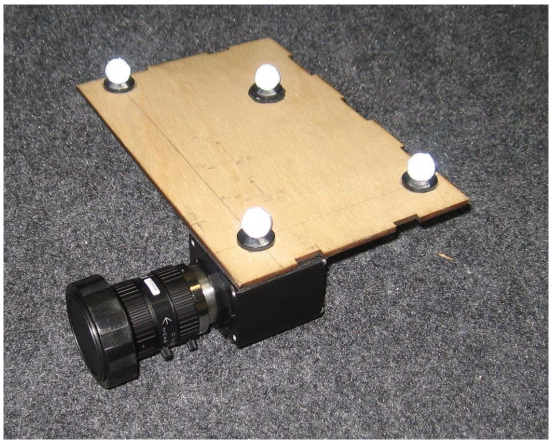
The camera had a 4.8 *mm* by 3.6 *mm* sized sensor with 1280 by 960 *pixel* resolution. The camera and lens combination provided a FOV_X of 33.4° and a FOV_Y of 25.4° for an aspect ratio of 1.317. Images were collected of a standard checkerboard and a distortion model was constructed using the theory demonstrated in [9] and the Camera Calibration Toolbox for Matlab® [2].

The use of the Vicon® system was invaluable for this testing. Small tracking devices (reflective balls), placed on the box-aircraft and the camera, allowed the system to collect accurate position and orientation of both. The system's cameras projected and then detected the reflection of infrared light off the reflective balls on the objects, triangulating their position. Figure 5.6 shows pictures of the equipment used with the reflective balls attached and the Vicon® environment.

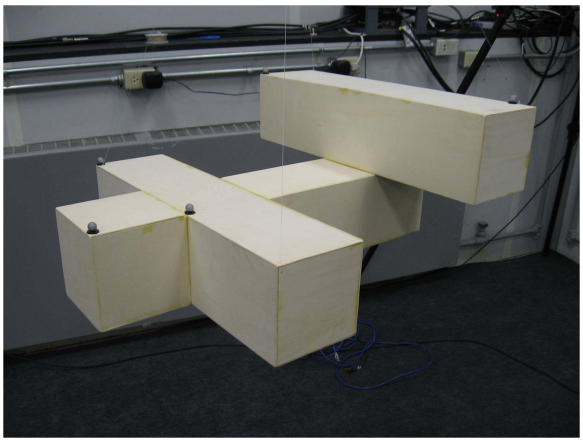
Collecting the Vicon® data simultaneously with the images created a time-stamped data source for each collected image. A single laptop computer with two ethernet connections collected both the Vicon® data and the images.

For various reasons, mainly because of visual noise in the scene, the images were preprocessed prior to matching. A contour function, applied to both the collected and rendered images, created a contour representation before initiation of the template matching. This process was described in Section 2.2.2.1.

The simulation simplified a few aspects of AAR. No b_w-frame was defined, only a *cam*-frame. The origin of the Vicon® system simulated the *e*-frame for relative



(a)



(b)

(c)

Figure 5.6: Vicon®-tracked objects. The cameras of the system detect infrared light reflecting off of the balls on the objects.
(a) The camera did not meet the minimum size requirement for the Vicon® system to track its orientation, requiring the addition of an extension.
(b) The aircraft was hung with fishing wire that was barely visible in the $\mathbf{I}_c$ images.
(c) The Vicon® cameras surround the tracked objects for optimum triangulation.

position error determination and also the n-frame for attitude information and error, which minimized unnecessary translations and rotations. The system as defined and used is shown in Figure 5.7.

Collection of data simulated the receiver at various positions behind and below the box-aircraft and accomplishing various random motions. These motions did not necessarily represent true AR motions, additionally, the box-aircraft had a natural sway motion from a single attachment point for the fishing wire. Because of this natural sway of the aircraft, the RIPE process included both roll and yaw in every matching update. The algorithm periodically updated the pitch of the box, but the suspension of the box limited the pitch motion considerably. An example result of the algorithm's matching is shown in Figure 5.8, with the contour overlay of the $\mathbf{I}_r$ on the $\mathbf{I}_c$.

Post-processed data runs were tracked by the RIPE process. An initial value was given to the algorithm from the truth source. The algorithm did not make use of a filter, the initial estimate of a measurement update was simply the final measurement from the previous update determined by the RIPE process. The error associated with this process without INS and without a Kalman filter is shown in Figure 5.9.

The analysis of the laboratory data is shown in Table 5.1. It was assumed that the addition of an INS and filter should only increase the accuracy of the system. They were excluded from the laboratory work because of time limitations.

For this single run of 520 frames or measurement updates at ranges from 80 to 110 *inches*, the algorithm performed well with an error of approximately $\pm 1\%$, as a function of the range between aircraft, in all three axes. There is an obvious delay in the $\mathbf{Z}$ axis of the *cam*-frame (seen in the top of Figure 5.9) which is attributed to the small range of perturbations allowed to that axis. The algorithm could simply not adjust the translation in that axis fast enough to match the actual motion of the target, it used the maximum perturbation allowed in each frame. The errors in the $\mathbf{Y}$ axis and $\mathbf{X}$ axis are attributed to the contouring process. The contour function

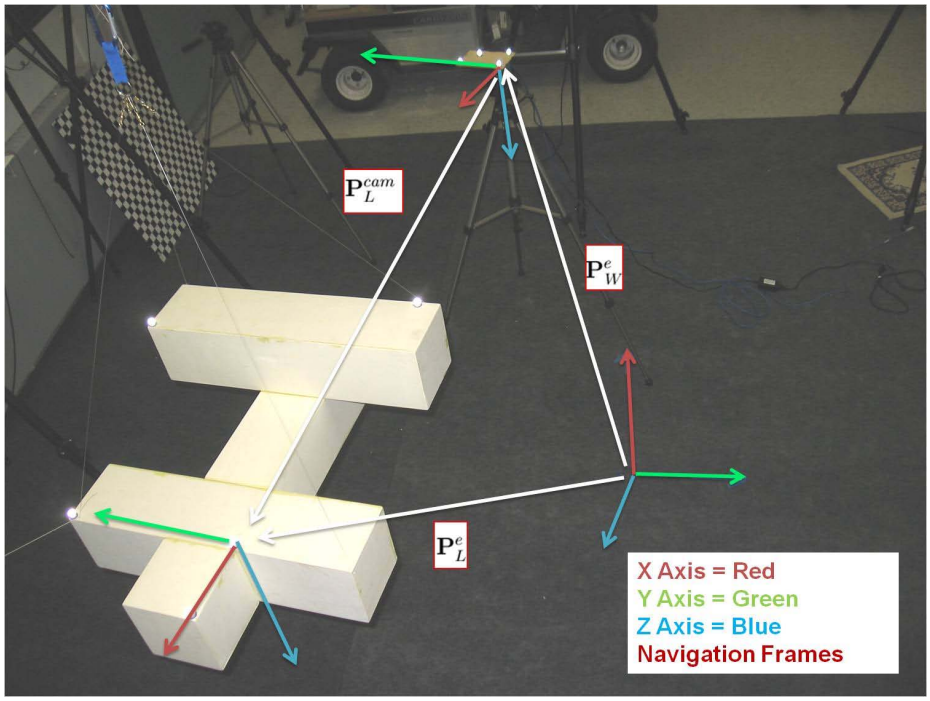


Figure 5.7: Laboratory work navigation setup. The laboratory work environment with the simplified AAR-associated reference-frames.

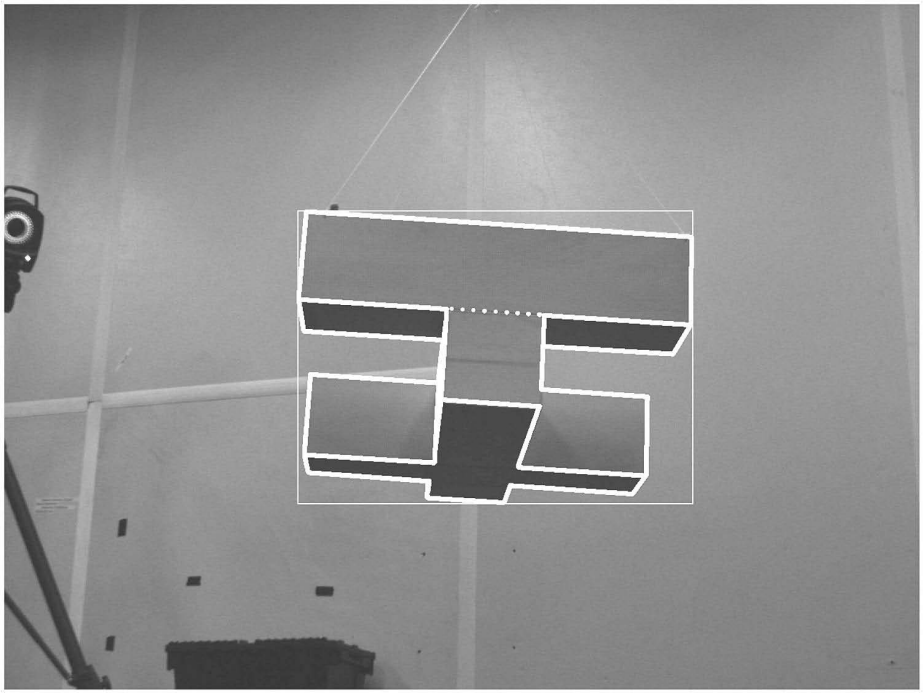


Figure 5.8: The laboratory work example match. The contour overlay of the $\mathbf{I}_r$ is shown on the $\mathbf{I}_c$.

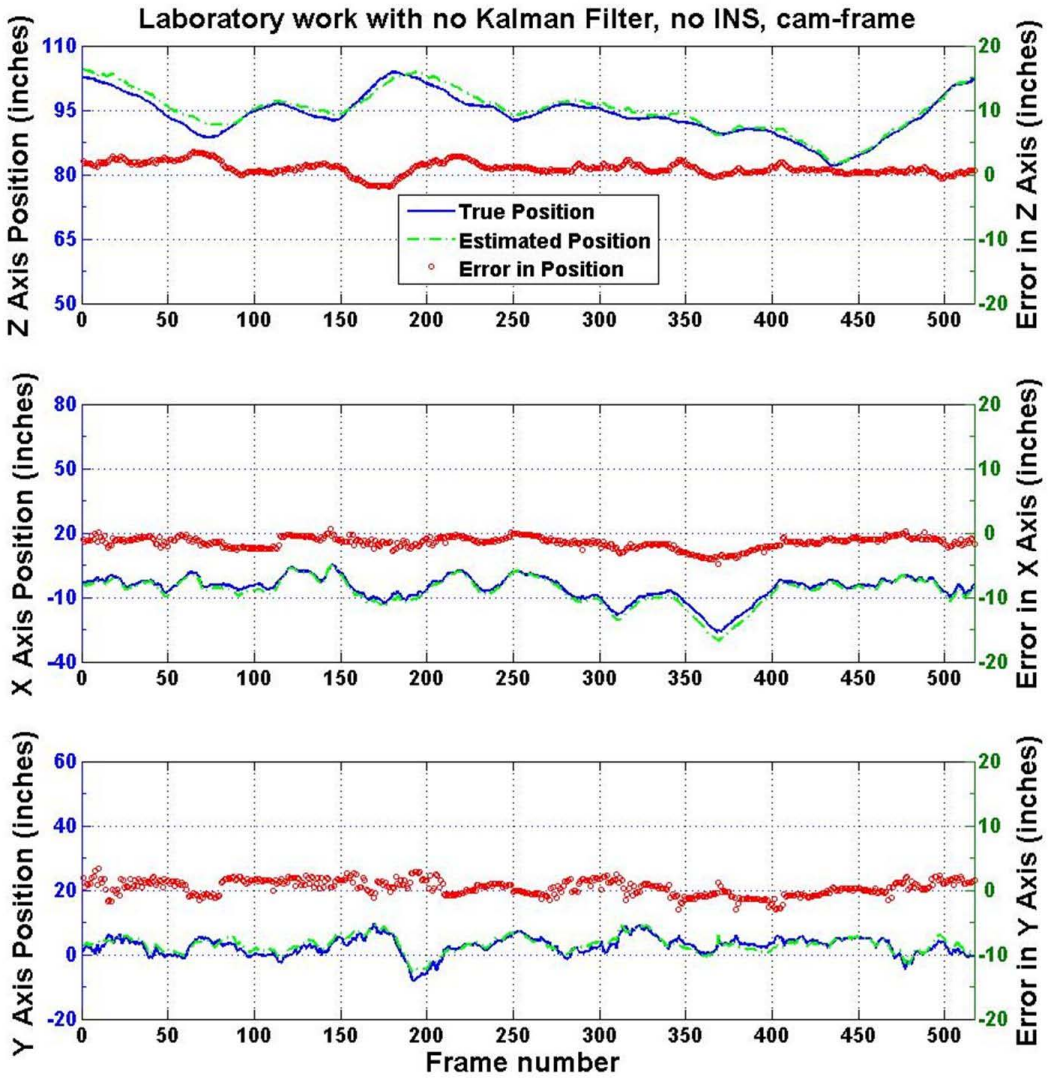


Figure 5.9: The laboratory work error. The work in the lab demonstrated the viability of the RIPE approach. This error is shown in the *cam*-frame because there was no b_W-frame defined nor defined AR positions in this setup.

DOF	Max (-) Error (*inches*)	Max (+) Error (*inches*)	Mean Error (*inches*)	Standard Deviation (*inches*)
$\mathbf{Z}_{cam}$ axis	-2.0	3.5	1.0	1.0
$\mathbf{X}_{cam}$ axis	-5.0	0.5	-1.5	1.0
$\mathbf{Y}_{cam}$ axis	-3.0	3.5	0.5	1.0

Table 5.1: Data analysis of RIPE laboratory error. Data was rounded to the nearest half *inch*.

as implemented found the interior and exterior contours of the box-aircraft. When this contour was displayed, the exterior edges of the box-aircraft had two lines, for both the interior and exterior contours found. To compensate, the line-width used to draw the contours was increased (creating a single line), blurring the true edge of the box-aircraft slightly. This can be seen in Figure 5.8; the contour lines are thicker than the actual edge of the box-aircraft. As a result, the pure black and white (with no grey) contour $\mathbf{I}_c$ and $\mathbf{I}_r$ images had more of a line-width to match, decreasing the precision.

Finally, during the data run shown here, the box-aircraft temporarily and only partially exited the field of view around frame number 365 (witnessed in Figure 5.9 as a large change in the translation in the $\mathbf{X}_{cam}$ axis.) Figure 5.10 shows an image of the farthest occlusion of the box-aircraft. While the box-aircraft was only at this extreme position for a few frames, the algorithm was able to consistently track it throughout.

The laboratory work demonstrated that such a process was viable. Predictions from the errors seen in the lab were estimated to be similar in flight, with an error increase from the additional complexity of flight including motion relative to the ground, more maneuverable objects, and significant increase in range between them. Unfortunately, the process was too slow to be implemented in real-time. Because of the extra time required to create contours of each image in addition to the discovered inefficiency of the OpenGL to OpenCV conversion, every measurement required approximately four *seconds*. Most likely, this is unacceptable for close formation aircraft navigation. A discussion on the OpenGL to OpenCV conversion is presented in Section 5.5.3. The next section details the RIPE approach to the field work.

5.3 Field Work

The field work entailed flights with a TPS T-38A Talon aircraft as a simulated tanker and a Calspan LJ-25 Learjet as a receiver [10]. The data collection was part of a test management project (TMP) for the students in the TPS class 10 Alpha. The *Here's A Visually-Enabled Guided Air refueling System* (HAVE GAS) TMP

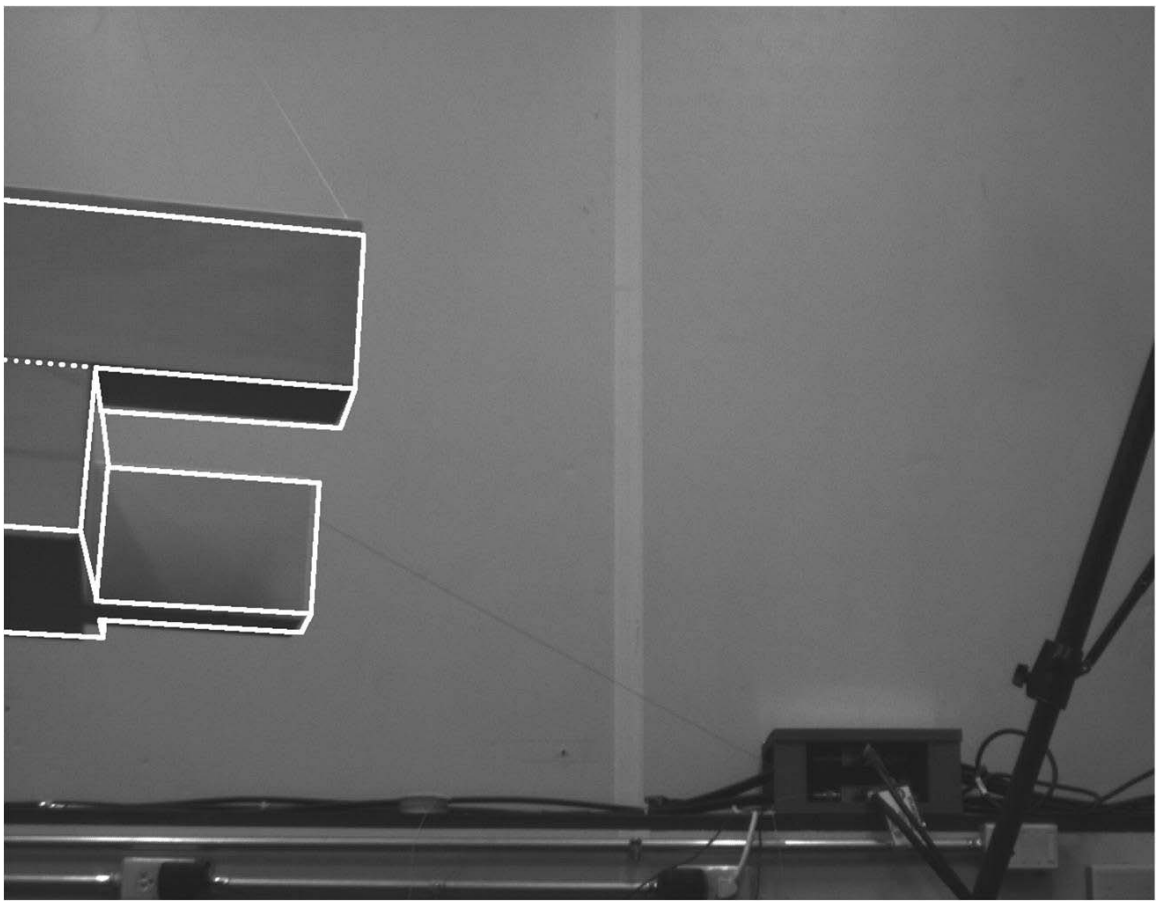


Figure 5.10: Partial aircraft occlusion. To test the algorithm's ability to track with partial occlusion, the camera was moved limiting the amount of the box-aircraft seen in the field of view.

group flew the aircraft through representative refueling formations, rejoins, closures, and separations as demonstrated in Figure 5.11. The LJ-25 had a digital Prosilica® monochrome camera, model GE1660, installed behind the windshield. The camera images were collected by an on-board computer for post-flight download. The aircraft were flown at various distances, offsets from centerline, and aspect angles. Truth data for 36 different parameters were collected at 100 *Hz* and images were collected at 10 *Hz*.

The data collection devices consisted of GPS Aided Inertial Navigation Reference (GAINR) units, configuration 2B (C2B) located on the two different aircraft. These units had an internal IMU (HG-1700) and dual (L1/L2) frequency GPS antenna

Figure 5.11: HAVE GAS TMP data collection flights. The data collection process doubled as a curriculum event for the TPS course.

mounted on the outside of the aircraft. A picture of the installed unit on the LJ-25 is shown in Figure 5.12. The data were filtered post flight and partially corrected with the boresight locations of the devices. The two GAINR units were time-synced with GPS and recorded their data stamped with GPS time. The LJ-25 had an additional truth collection system installed, used as a backup to the GAINR. The data attained from the LJ-25 on-board computer created the INS source for the RIPE algorithm, allowing for independence of the truth source from the algorithm. The truth data had a reported accuracy in position of 18 *inches* and in attitude of 0.1° and were only used for an initial position and a final comparison [7].

The collected images were time-stamped with a file name based on the internal clock of the Prosilica® camera. When power was applied to the camera, the internal clock started counting from zero at a rate of 79,861,111 *Hz*. Because the camera was within the pressurized cabin and the temperature was relatively constant

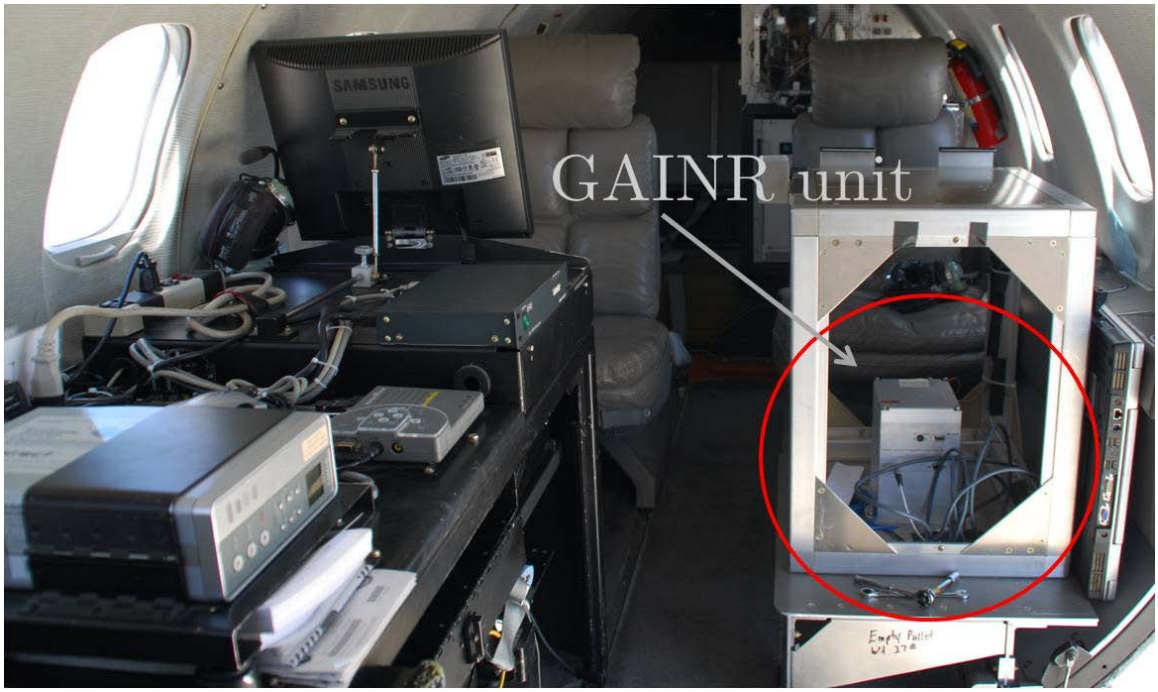


Figure 5.12: Truth collection device installed on the project LJ-25. A similar unit was installed in the nose of the T-38.

($\pm 5°F$), very little environmentally-induced variability influenced the camera timing. Converting the camera timestamps into *GPS seconds*, to match the truth data timestamps, required the number of *counts per second* plus the time the camera was initially powered. It would be challenging to determine the exact time power was applied, so time-sync maneuvers were flown in flight. The maneuvers were visually identifiable in both the $\mathbf{I}_c$ and the truth data and consisted of bank to bank rolls in both aircraft. Ultimately, only the first bank maneuver by the T-38 was required, the other rolls were less crisp then the first T-38 roll. The time sync was accomplished at the beginning and end of the flight to determine if there was any time drift. During these maneuvers the camera collection rate was increased to 30 *Hz* to minimize the time between collections to pinpoint the exact time of maximum bank angle in the images. An example of the T-38 at maximum roll angle with the truth data as a contour overlay is shown in Figure 5.13.

Figure 5.13: Time sync maneuver. Four different data sources were synced together using a few time sync maneuvers through the flight. The contour overlay is from the combination of the two GAINR unit's truth data. A position bias between aircraft in the truth data is visible by the non-intentional offset.

For the second flight (the data used in this thesis), the camera was determined to be turned on 56,204.464 *seconds* after 0000.000 local time with a 95% confidence level of ±72 *milliseconds* [10]. Accounting for this possible error in time correlation between the three data sources resulted in a corresponding range error of ±3.5 *inches* with an assumed maximum closure or separation rate of 4 *feet per second* [10].

The Prosilica® camera used was 1200 by 1600 *pixel* resolution with 5.5 *micrometer* pixel size for a sensor size of 6.6 *millimeter* by 8.8 *millimeter*. The lens was a VS Technology Cooperation Mega Pixel Closed Circuit Television (CCTV) SV-0814MP with a fixed 8.3 *millimeter* focal-length lens. Focus and aperture were set on the ground before flight; however, the pilot could adjust if required in flight. Typically, only one or two adjustments of the aperture were necessary in flight, once the tanker aircraft was in the FOV.

Images of a checkerboard were taken with the system camera as mounted in a fixed location on the windshield. The images combined with the calibration software [2] determined a calibrated camera model and distortion model. The field work

checkerboard collection was slightly more cumbersome and had to be partially repeated. The combination of a wide field of view and high pitch angle of the camera limited the use of a planned lifting device from getting close enough to the camera. Additionally, the checkerboard was too flexible when held from one corner as shown in Figure 5.14. A reinforcement grid of wood was attached to the back for stiffening, which made it heavy to hold and position correctly.

The camera calibration determined the **K** matrix to be:

$$\mathbf{K} = \begin{bmatrix} 1562.95 & -0.0127 & 767.84 & 0 \\ 0 & 1528.77 & 607.709 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.1}$$

where the values (except skew, **K**(1,2), which is dimensionless) are in *pixels*. These values are presented with 99.7% confidence in focal length of ± 5.5 *pixels* and in principal point of ± 3.0 *pixels*, both rounded to the nearest half *pixel*. The skew value was within ± 0.00036 with 99.7% confidence.

It is important to note the distinction between empirical values and manufactured specifications. The specified focal length of the lens was 8.3mm, which should have allowed a $\text{FOV}_Y = 44.83°$ according to Equation (2.30). However, using the empirical values from Equation (5.1) in Equation (2.30), a smaller $\text{FOV}_Y = 43.36°$ results. The focal length in pixels for the **Y** axis was 1528.77 pixels with 5.5μm sized pixels, the empirical focal length was actually 8.4mm (1528.77pixels x $5.5^{\mu\text{m}}/_{\text{pixel}}$) and 8.6mm for the **X** axis. This is an important distinction when using the OpenGL setup and must be accounted for. Additionally, the non-center principal points and skew of the **K** are not accounted for in the normal OpenGL setup and is addressed in Section 5.5.1.

The resulting pixel error from the distortion removal process was approximately one pixel in both axes. This distortion-model pixel error was a result of the difficulty in modeling the visual warping through the windshield. The camera was not initially rotated correctly on the dash and a final correction to rotate the camera to a higher

Figure 5.14: Collecting camera calibration images. This process was more difficult than the average laboratory setting, flexing in the original board caused errors in the model.

pitch angle brought the lens very close to the very curved windshield. The effects of this pixel error are not significant and are not attributed to the errors of this process.

The data were collected in the Edwards AFB, CA R-2508 complex during September 2010. Collection was conducted during the daytime with minimal weather impact (clouds, rain, etc.) Two flight test engineers operated the camera collection laptop and LJ-25 truth collection system. Collections were accomplished in two to four minute segments due to limitations in airspace and to minimize possible corruption or data loss on a single run. Actual AR maneuvers can possibly be longer in duration and most likely, less dynamic.

5.4 Data and Errors

After the data were collected, the RIPE algorithm was tuned for the most difficult run that involved the Sun in the FOV, included a turn and a crossing from one side of the aircraft to the other. This run was non-typical for AR, so more benign representative runs characterize the quality of the RIPE approach, while the others are presented for a determination of its robustness.

The first run demonstrates the effects of the Kalman filter on a turning track AR maneuver. Figure 5.15 shows the error from run ten of the second flight. In this run, the wing aircraft started at 168 *feet* aft of the lead aircraft, closed to 62 *feet* and maintained it for 15 *seconds* before backing out. The run was conducted at a 40° elevation, the bottom of the refueling envelope (witnessed in the more negative $\mathbf{Z}_{b_L}$ axis translation, compared to Figure 4.11) and translated across the $\mathbf{Y}_{b_L}$ axis (witnessed in the $\mathbf{Y}_{b_L}$ axis position crossing zero *feet* translation). Additionally, the lead aircraft maintained above 30° of roll angle throughout the maneuver and the tail of the aircraft was occluded from view for a portion of the run (approximately 43 to 66 *second* marks). This result appears to resemble the errors of the measurement only run in Figure 4.11; however, this maneuver was more dynamic and started farther back. The analysis of the data from Figure 5.15 is shown in Table 5.2.

The errors visible in the figure and table for run number ten have characteristics worth examining. First, the roll errors appear less discrete than the measurement only run in Figure 4.11. Instead, the roll error now has an oscillation to it. This oscillation is still attributed to the discrete capability of the RIPE in the roll axis. The roll angle has to go beyond 0.5° before the next closest integer degree appears more likely (the perturbation amounts in roll were ±1° and ±2°). This affects the roll rate of change in the Kalman filter. Large discrete jumps in roll angle appear as large rates of changes to the filter. The filter updates the roll rate of change state appropriately and predicts a similar rate of change for the next propagation step. For small motions in bank angle, this causes an *overshoot* and the oscillation seen in the roll position.

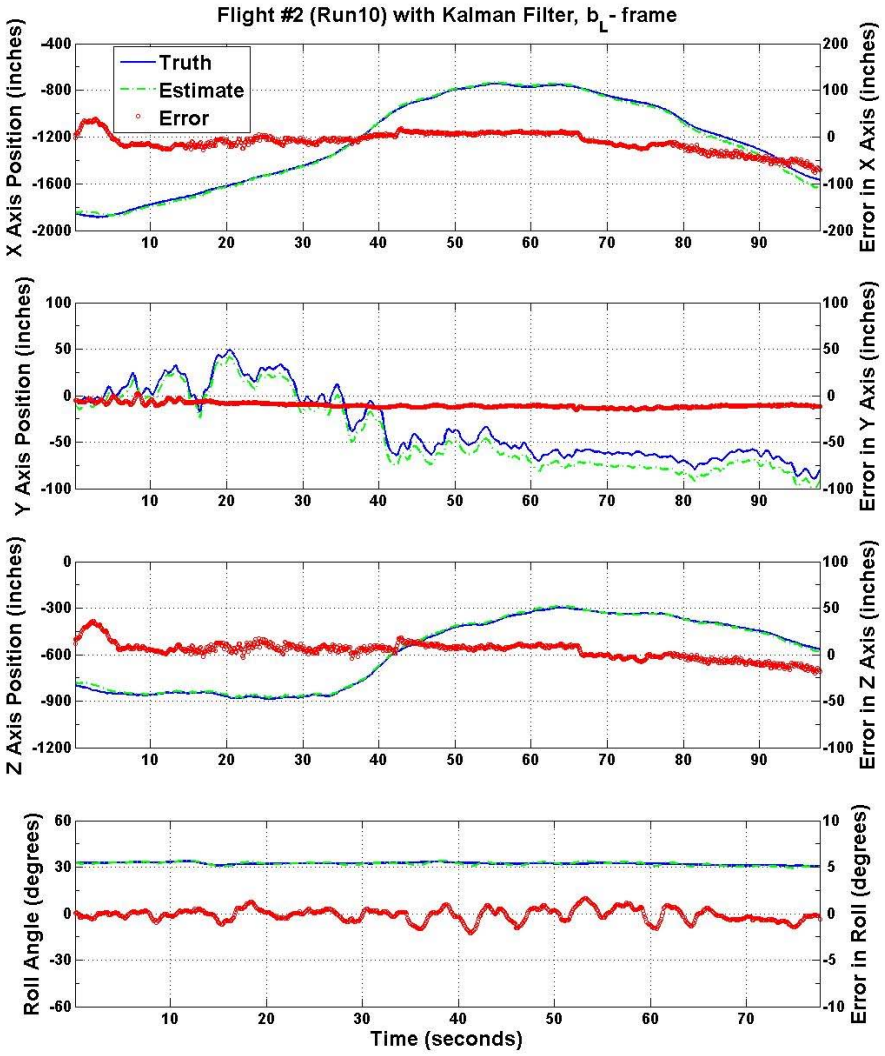


Figure 5.15: Flight two, run ten RIPE error.

DOF	Max (-) Error	Max (+) Error	Mean Error	Standard Deviation
$\mathbf{X}_{b_L}$ axis	-76.0 *inches*	38.0 *inches*	-10.0 *inches*	21.0 *inches*
$\mathbf{Y}_{b_L}$ axis	-16.0 *inches*	2.5 *inches*	-10.0 *inches*	2.5 *inches*
$\mathbf{Z}_{b_L}$ axis	-20.0 *inches*	36.0 *inches*	4.0 *inches*	9.0 *inches*
Roll	-2.0°	1.5°	0.0°	0.5°

Table 5.2: Data analysis of flight two, run ten. Data was rounded to the nearest half *inch* or half *degree* respectively.

The error in the roll is not an important concern, since it is just a means to create a better visual representation of the aircraft to determine the relative position. With such a small standard deviation and limited impact on the relative position no improvement to the Kalman filter's roll estimates was attempted. For brevity, roll errors are left out of the remaining error plots.

Other errors are visible in run number ten. Similar to Figure 4.11, a bias is visible in the $\mathbf{Y}_{b_L}$ axis, attributed to a bias in the truth collection. Also, interaction between $\mathbf{X}_{b_L}$ and $\mathbf{Z}_{b_L}$ axis is apparent. This is a difficult problem in AAR [29] and pose in general, differentiating between motions in the different DOFs. Evident throughout the run, most notably at the beginning, is the compensation of the RIPE algorithm. When an error appears in one axis, a visual compensation to maintain the correct appearance of the aircraft appears in the other axis. This is most pronounced at further distances between aircraft and when partial occlusion of the aircraft occurs. Because of the relatively small changes in the $\mathbf{Y}_{b_L}$ axis, the interaction errors are generally not as evident in this axis, a later example will demonstrate this interaction.

The theory of template matching should reduce the effects of this interaction; if the theory is correctly implemented, it will. It is compromised when implemented in Cartesian coordinates. Unless the object is in the center of the image, any perturbations in the $\mathbf{Z}_{cam}$ axis appear in the image as a change in size and a change in image location. This change affects the visual appearance of the aircraft. This change to the visual appearance for each perturbation in the $\mathbf{Z}_{cam}$ axis changes the template matching likelihood comparison. The theory requires for the *size plus image location* group that the appearance of the aircraft remains the same while allowing a change to its size only. If correctly implemented, the errors witnessed in the $\mathbf{Z}_{b_L}$ axis could resemble those seen in the $\mathbf{Y}_{b_L}$ axis.

Next, two common trends were witnessed in most of the data runs. First a large jump in error was present near the beginning of the runs. Second, a steadily increasing, and uncorrected error was observed as the aircraft separate.

The large jump in error at the beginning is partially from the incorrect initial state provided to the Kalman filter. The initial state provided to the filter are the three translation values and the roll value, the rates of change to these states and their derivatives were set to zero. The filter takes some time to determine those values from the measurement updates provided by the RIPE process. Additionally, the large error at the beginning of the run is attributed to the size of the allotted perturbations at the farther distances. The perturbations in the $\mathbf{Z}_{cam}$ axis were arbitrarily chosen to be constant throughout the run (± 5 and ± 10 *inches*). At the farther distances, the visual appearance of the rendered aircraft does not change dramatically, if at all, with such small perturbations. This allows all five of the possible $\mathbf{I}_r$ images in the *size plus image location* group to appear equally likely. With the Kalman filter determining the rate of change of these states at the same time errant measurements are likely, the error is exacerbated until the aircraft are close enough together that a correction is possible. A worst case example of this error is presented later.

Additionally, affecting the beginning as well as the ending of the runs, the distance between aircraft influences the errors. On average the errors are typically 3-4% of the distance between the aircraft or less, with the maximum error values less than 7% of the distance between the aircraft (both measured in the $\mathbf{X}_{b_L}$ axis). As an example, the maximum error in the $\mathbf{X}_{b_L}$ axis near the beginning of the run in Figure 5.15 is 2% of the distance ($^{38in}/_{1900in}$), and at the end of the run, the error is 4.75% of the distance ($^{76in}/_{1600in}$.) This is a resolvable distance limitation of the images as discussed in Section 4.4.

The second trend in the data runs is the steady, increasing, and uncorrected error at the end of the run. This is attributed to the template matching. For an unknown reason, the algorithm does not match the size of the receding aircraft as accurately as it does when the aircraft is closing. This is a potential limitation to the template matching function.

The error analysis in Table 5.3 characterizes the error in run number ten for just the pre-contact position (1150 *inches*) and closer, when accuracy is most critical (between 38 *seconds* and 82 *seconds* in Figure 5.15.) In general, most of the errors are reduced. The most notable changes are the errors in the $\mathbf{X}_{b_L}$ and $\mathbf{Z}_{b_L}$ axis, both are reduced in max errors and standard deviation of error. The mean error in the $\mathbf{X}_{b_L}$ axis is reduced dramatically as well; however, this is because the interaction with the $\mathbf{Z}_{b_L}$ axis caused the error to transition negative for half the time. The root mean square error is 13 *inches* in the pre-contact position or closer.

DOF	Max (-) Error	Max (+) Error	Mean Error	Standard Deviation
$\mathbf{X}_{b_L}$ axis	-30.0 *inches*	18.0 *inches*	0.0 *inches*	13.0 *inches*
$\mathbf{Y}_{b_L}$ axis	-16.0 *inches*	-10.0 *inches*	-12.0 *inches*	1.0 *inches*
$\mathbf{Z}_{b_L}$ axis	-8.0 *inches*	18.0 *inches*	5.0 *inches*	6.0 *inches*
Roll	-2.0°	1.5°	0.0°	1.0°

Table 5.3: Data analysis of flight two run ten, pre-contact (1,150 *inches*) and closer. Data was rounded to the nearest half *inch* and *degree* respectively.

Figure 5.16 shows run 18, another example run of maneuvers resembling AR. This error plot shows only the run's pre-contact to contact transition and back. Determinations of the error during run 18 are shown in Table 5.5. With a few differences, the errors in this run are similar to the errors in run ten, pre-contact position and closer (Table 5.3). The interaction between the two axes ($\mathbf{X}_{b_L}$, $\mathbf{Z}_{b_L}$) is evident as well as the bias in the $\mathbf{Y}_{b_L}$ axis. The largest differences between the runs is the mean error in the $\mathbf{Z}_{b_L}$ axis, which is now negative and an outlier in the $\mathbf{Y}_{b_L}$ axis affected the maximum (+) error in that axis. The $\mathbf{X}_{b_L}$ axis error has a slightly reduced standard deviation because of a slower separation rate between aircraft.

A final AR representative run, number 12, is shown in Figure 5.17. This run shows a larger error in the beginning (6.5% as a function of $\mathbf{X}_{b_L}$ axis distance between aircraft) that is able to correct before the pre-contact position. This is shown as an example of a worst case for the representative AR maneuvers. The error is a combination of incorrect implementation of the template matching theory, resolvable distance limitation, and incorrect initial conditions to the filter.

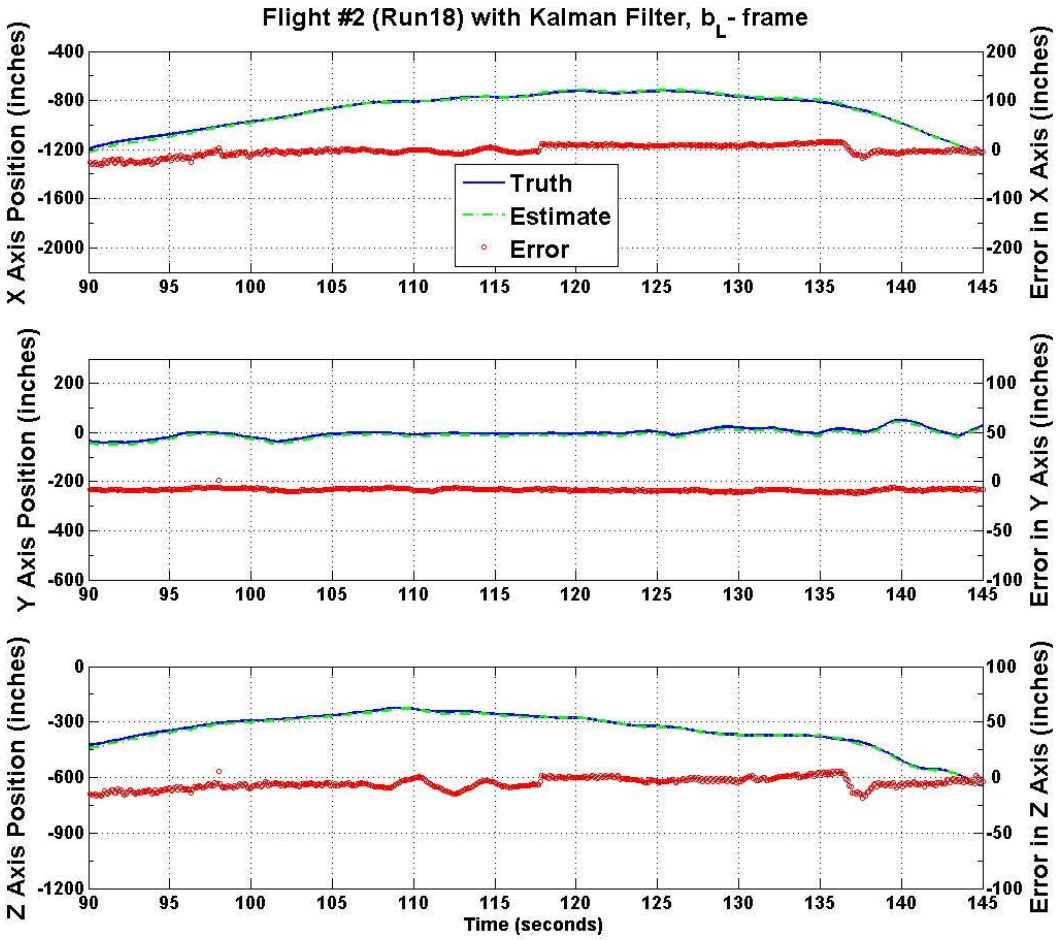


Figure 5.16: Flight two, run 18 RIPE error.

DOF	Max (-) Error (*inches*)	Max (+) Error (*inches*)	Mean Error (*inches*)	Standard Deviation (*inches*)
$\mathbf{X}_{b_L}$ axis	-30.0	17.0	-0.5	9.5
$\mathbf{Y}_{b_L}$ axis	-12.0	1.5	-8.5	1.5
$\mathbf{Z}_{b_L}$ axis	-18.5	6.0	-5.0	5.0

Table 5.4: Data analysis of flight two, run 18 pre-contact position (1,150 *inches*) and closer. Data was rounded to the nearest half *inch*.

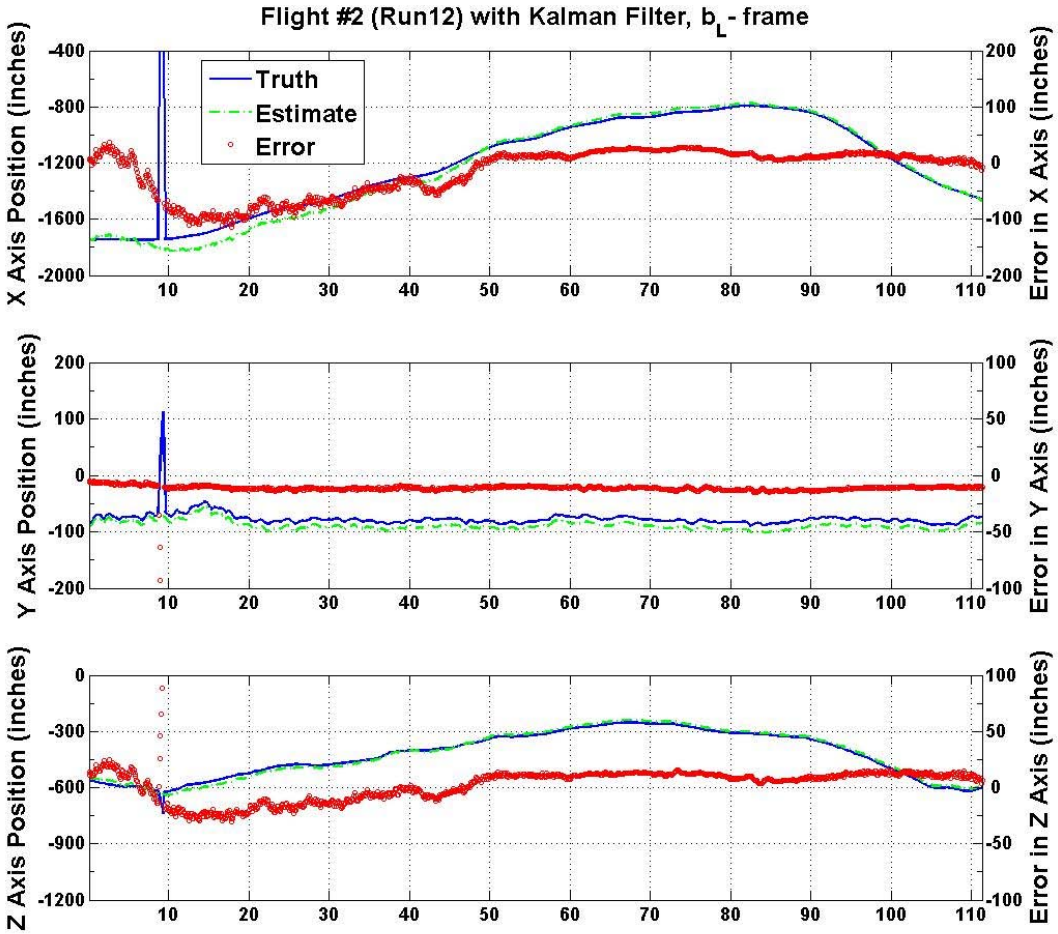


Figure 5.17: Flight two, run 12 RIPE error.

DOF	Max (-) Error (*inches*)	Max (+) Error (*inches*)	Mean Error (*inches*)	Standard Deviation (*inches*)
$\mathbf{X}_{b_L}$ axis	-25.0	29.0	15	8.0
$\mathbf{Y}_{b_L}$ axis	-15.0	-9.0	-12	1.5
$\mathbf{Z}_{b_L}$ axis	0.0	15.0	10	2.5

Table 5.5: Data analysis of flight two, run 12 pre-contact position (1,150 *inches*) and closer. Data was rounded to the nearest half *inch*. Demonstrating a worst case for representative AR maneuvers.

As a challenge to the algorithm, two atypical AR maneuvers were flown. The first tested the algorithm's ability to handle extreme dynamic motions. The second tested the algorithm's ability to handle difficult visual situations in addition to dynamic motions.

The first challenge required the wing aircraft to initially maintain the pre-contact position with approximately 30° roll angle. After stabilizing, the wing aircraft initiated a rapid roll reversal causing the lead aircraft to quickly exit the FOV of the camera. Because of the speed at which the lead aircraft left the FOV, this maneuver was captured and processed by the RIPE algorithm at 30 *Hz*. Because these dynamics were not modeled, the Kalman filter was not used (witnessed in the discrete errors in Figure 5.19).

Figure 5.18 shows a few sample images from the maneuver, with the RIPE estimated position contoured overlayed. Figure 5.19 shows the errors resulting from this maneuver. The errors associated with this dynamic maneuver are mainly realized in the $\mathbf{Z}_{b_L}$ axis.

The second challenge included both visual and dynamic motions in a difficult AR situation. Sample images from this maneuver are shown in Figure 5.20. The maneuver began with the wing aircraft displaced to the left of the lead aircraft with approximately 30° roll angle. The wing aircraft transitioned sides when the sun started to appear in the camera FOV. The sun streaks on the windshield were a challenge to the template matching, and the algorithm adapted the contouring-first method used in the laboratory work. This obviously adds to the time required to compute a measurement and is not optimal; however it does allow the algorithm to continue working even with difficult visual noise. Figure 5.21 shows the errors associated with this visually challenging maneuver. It would be very difficult to navigate with the large errors at the end of the run. A better image processing technique, such as a Local Illumination Normalization Filter, might produce better results with the sun in the FOV [17]. Tracking at all through the sun was very

difficult and the important aspect is the capability of the RIPE process, not the image processing technique. As a note, all the errors presented in Figure 5.21 are 9% or less as a function of distance between the aircraft.

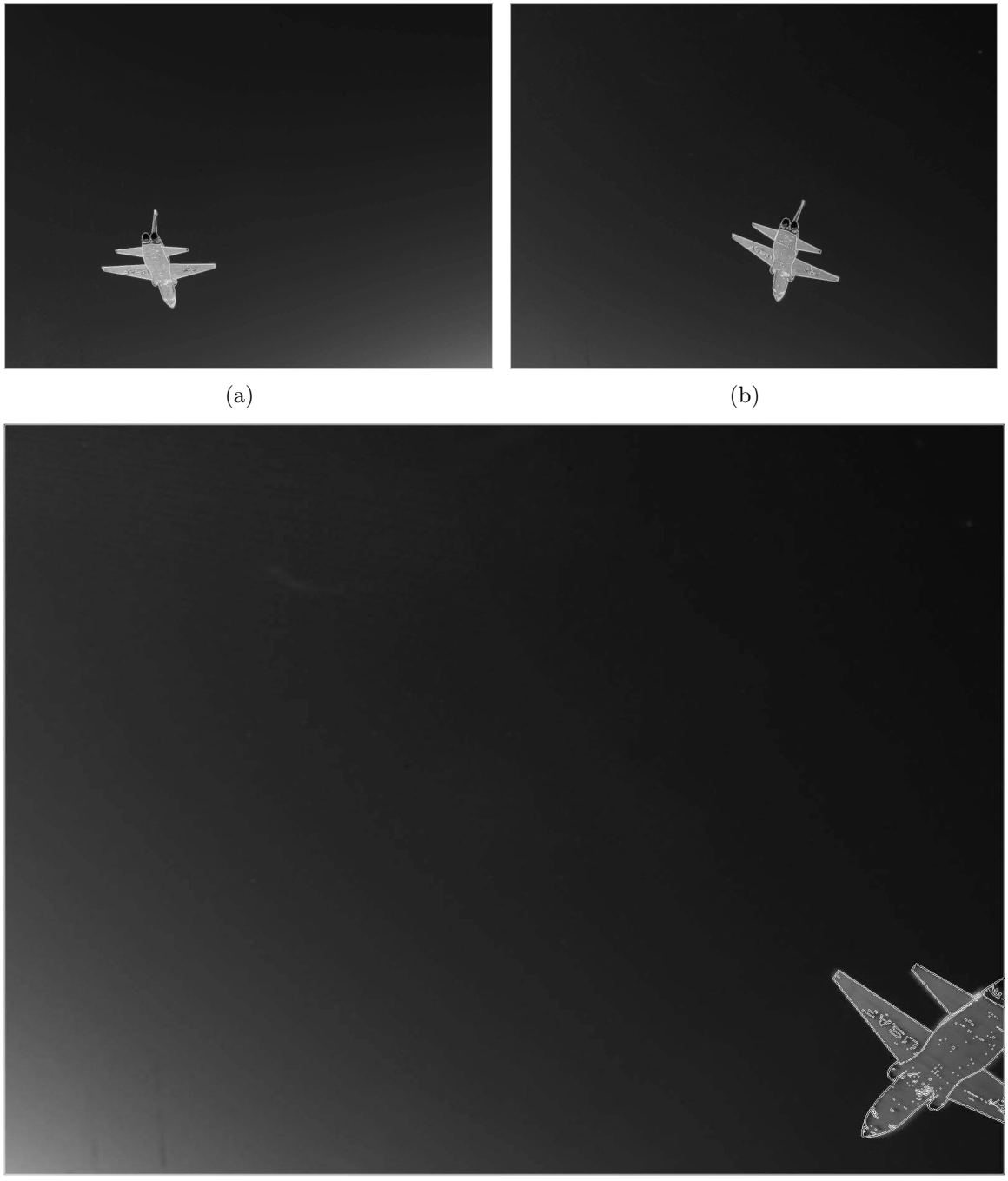
As discussed previously, the interaction between the $\mathbf{Y}_{b_L}$ axis and the $\mathbf{X}_{b_L}$ is more apparent in Figure 5.21 than previous error charts.

Through all the runs presented in this chapter, the average processing time was between 1.5 to 4 *seconds*, depending on the size of the aircraft in the image. When the aircraft was farther away, a smaller region of interest for the template matching allowed faster processing.

The following are potential solutions to the errors presented in this section:

- **Template Matching Theory:** The theory requires independence of $\mathbf{X}_{image}$ and $\mathbf{Y}_{image}$ from changes in $\mathbf{Z}_{cam}$ during the *size plus image location* group. The RIPE digital zoom alternative approach, or the use of spherical coordinates in the *cam*-frame (adjusting only the distance between the aircraft) might correct these errors.
- **Resolvable Distances:** Correctable with a higher resolution camera and lens. This will increase the number of pixels representing the aircraft. At farther distances, this will allow small perturbations in the $\mathbf{Z}_{cam}$ axis to make visual changes to the appearance of the aircraft, minimizing measurement errors to the filter.
- **Kalman Filter:** Better initial estimates plus potentially tracking in the n-frame where the motions of the aircraft were modeled [29]. Account for translational changes of the wing aircraft by incorporating INS velocity or acceleration values into the filter.

Improvements are necessary to this approach; however, the errors are manageable for implementation into an autopilot solution. This concludes the section on experimental data and error analysis. The next section presents process improvements discovered through the research and experimentation.



(a) (b)

(c)

Figure 5.18: Field work, dynamic motion. The collected images are shown with the most-likely, rendered, perturbation images shown as overlays.
(a) Initial position, the wing aircraft maintained pre-contact position with approximately 30° roll angle.
(b) 0.5 *seconds* after the wing aircraft initiated a rapid roll angle reversal.
(c) 1.3 *seconds* later the lead aircraft exits the cameras FOV.

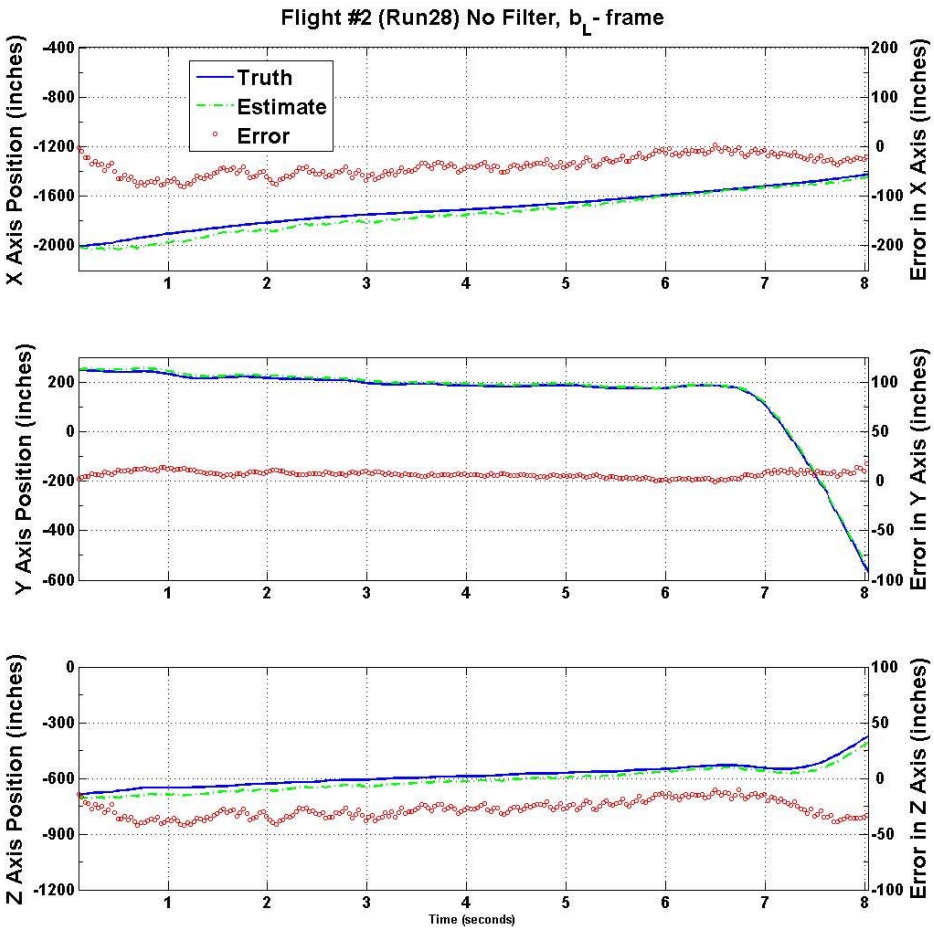


Figure 5.19: Flight two, run 28 RIPE error. The error associated with the field work, dynamic motion shown in Figure 5.18.

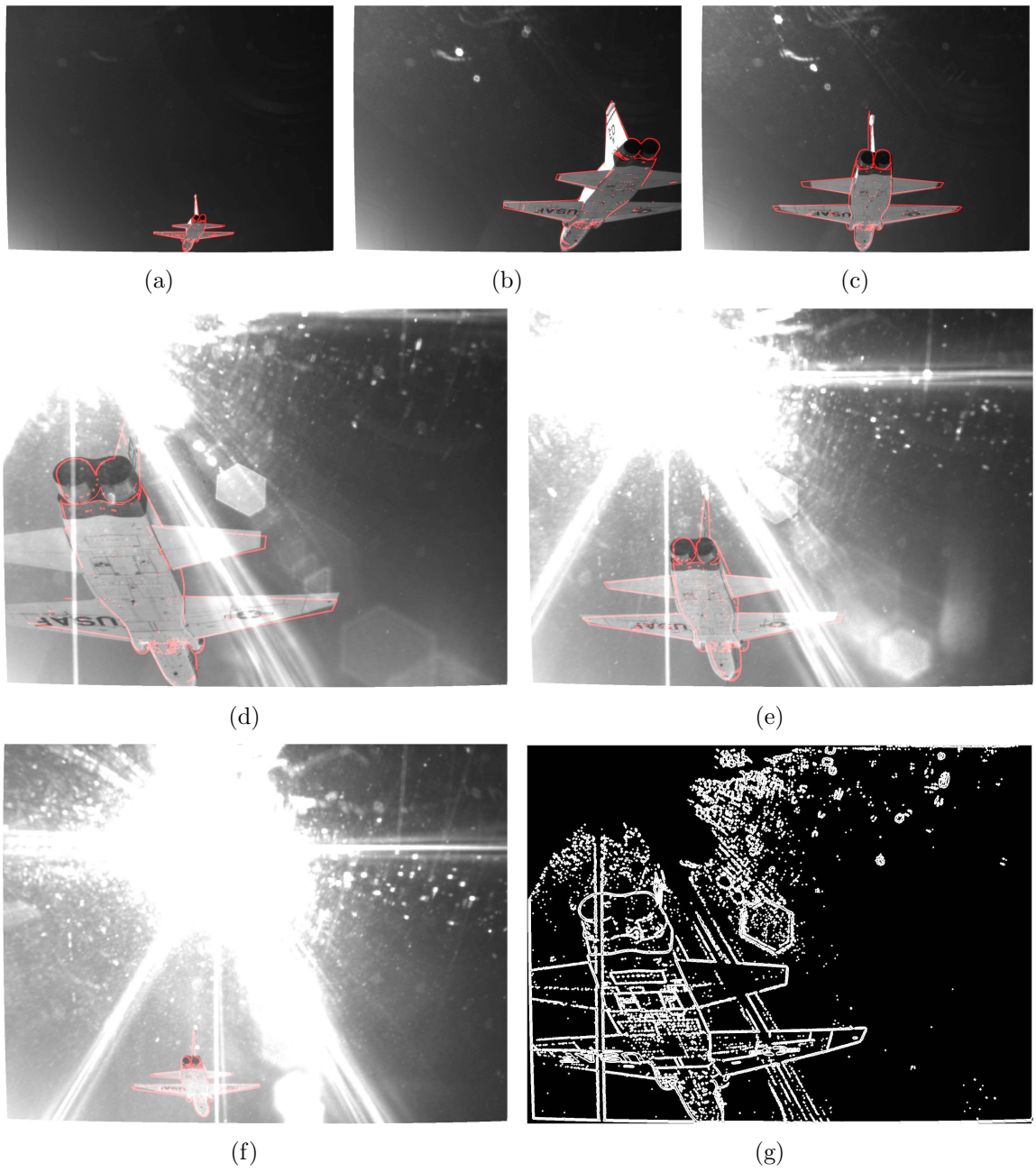


Figure 5.20: Field work, visual challenge. Entire run conducted with approximately 30° roll angle. The collected images are shown with the most-likely, rendered, perturbation images shown as overlays (except (g)).
(a) Initial position.
(b) Occlusion, lead on the right.
(c) Cross to the other side.
(d) Occlusion, lead on the left.
(e) Sun covers a good portion of the wing and elevator.
(f) Backing away.
(g) Actual view of image (d) used by the algorithm.

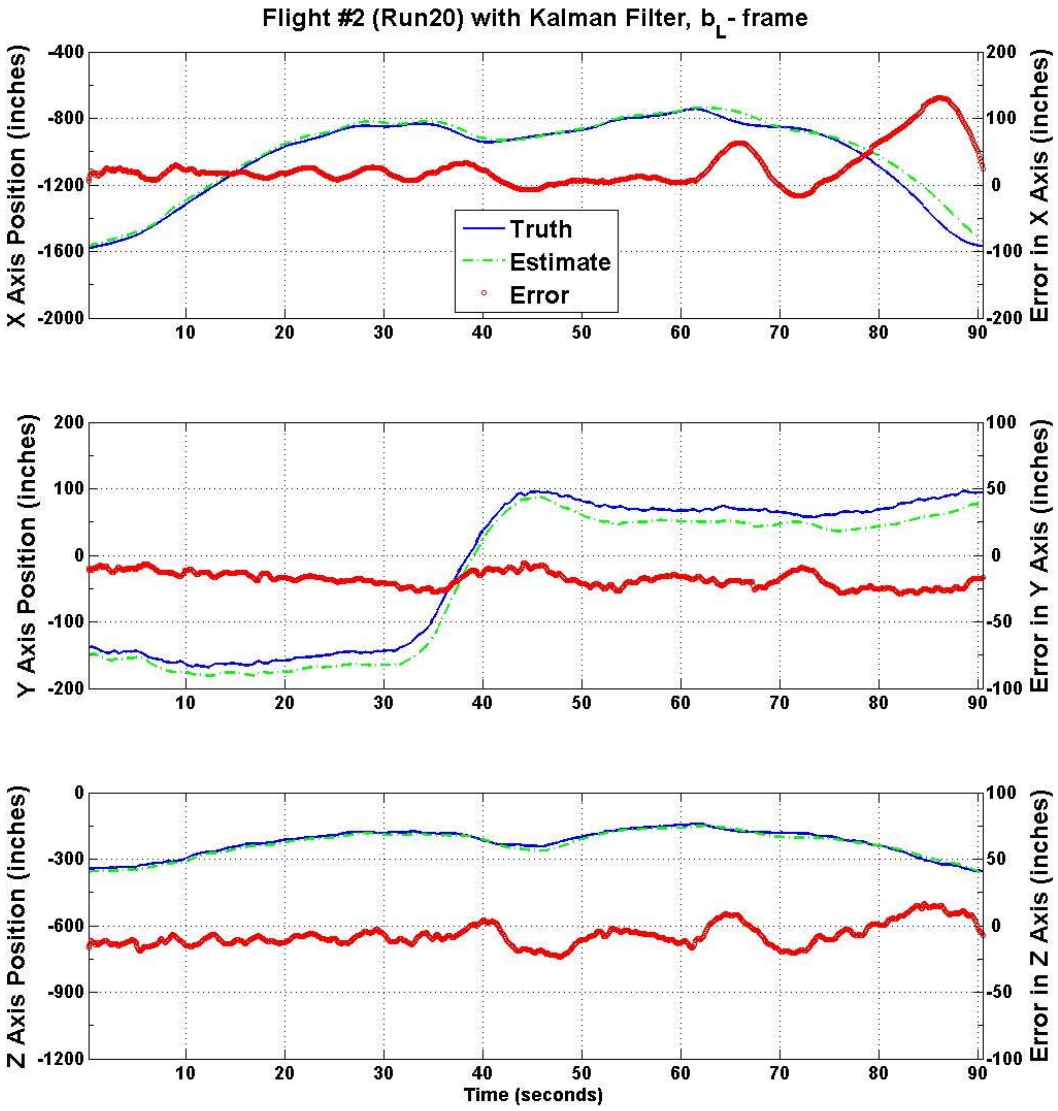


Figure 5.21: Flight two, run 20 RIPE error. The error associated with the field work, visual challenge shown in Figure 5.20

5.5 Areas of Improvement

Throughout the laboratory and field work, many improvements and refinements to the process were discovered and warrant presentation for future experimentation. They include a better representation of a camera in OpenGL, a validation process that should be considered when using OpenGL, and a slightly more efficient method of converting images from OpenGL to OpenCV.

5.5.1 A More Accurate OpenGL Camera Representation. As mentioned in Section 2.2.1.2, the transformation matrix created by glFrustum( ) is actually the combination of two transformation matrices, $\mathbf{T}_1$ and $\mathbf{T}_2$ [22]. The transformations are labeled in their order of occurrence and shown in the equation as pre-multipliers.

$$\text{glFrustum} = \mathbf{T}_2\mathbf{T}_1 \tag{5.2}$$

$$\mathbf{T}_2 = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{zNear-zFar} & \frac{zFar+zNear}{zNear-zFar} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

$$\mathbf{T}_1 = \begin{bmatrix} zNear & 0 & 0 & 0 \\ 0 & zNear & 0 & 0 \\ 0 & 0 & zNear + zFar & zNear * zFar \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{5.4}$$

The first matrix, $\mathbf{T}_1$, transforms the pyramid frustum into a rectangle shape. This process is similar to the mapping of the *cam*-frame to *image*-frame by the $\mathbf{K}$ matrix shown in Section 2.1.3.5, with a few exceptions: the $\mathbf{T}_1$ transformation maintains the $\mathbf{Z}$ axis translation information and there is no skew or principal points. The principal points are accounted for in the translation from CVV to *GLimage*-frame in Equations (2.38 and 2.39). If we label the transformations with the same notation as the DCMs, they become: $\mathbf{K} \rightarrow \mathbf{K}_{cam}^{image}$ and $\mathbf{T}_1 \rightarrow \mathbf{T}_{GLcam}^{NDC}$.

Because the location of frames in OpenGL are decided by the user, the *GLcam*-frame is collocated with the *cam*-frame. As a result, instead of using $\mathbf{T}_1$ to map the

points to the rectangle shape the parameters of $\mathbf{K}_{cam}^{image}$ can be used, accounting for the difference in orientations of the frames. However, because OpenGL maintains the $\mathbf{Z}_{GLcam}$ translation information until placing the contents of the CVV onto an image, an additional row is needed in the matrix $\mathbf{K}_{cam}^{image}$. It can be shown that the third row would be similar to the third row in $\mathbf{T}_1$, with a negative sign for the difference in z translation directions.

To facilitate the transformation, the cam'-frame is introduced. This frame is the projection of the cam-frame onto the near clipping plane, and is shown in Figure 5.22.

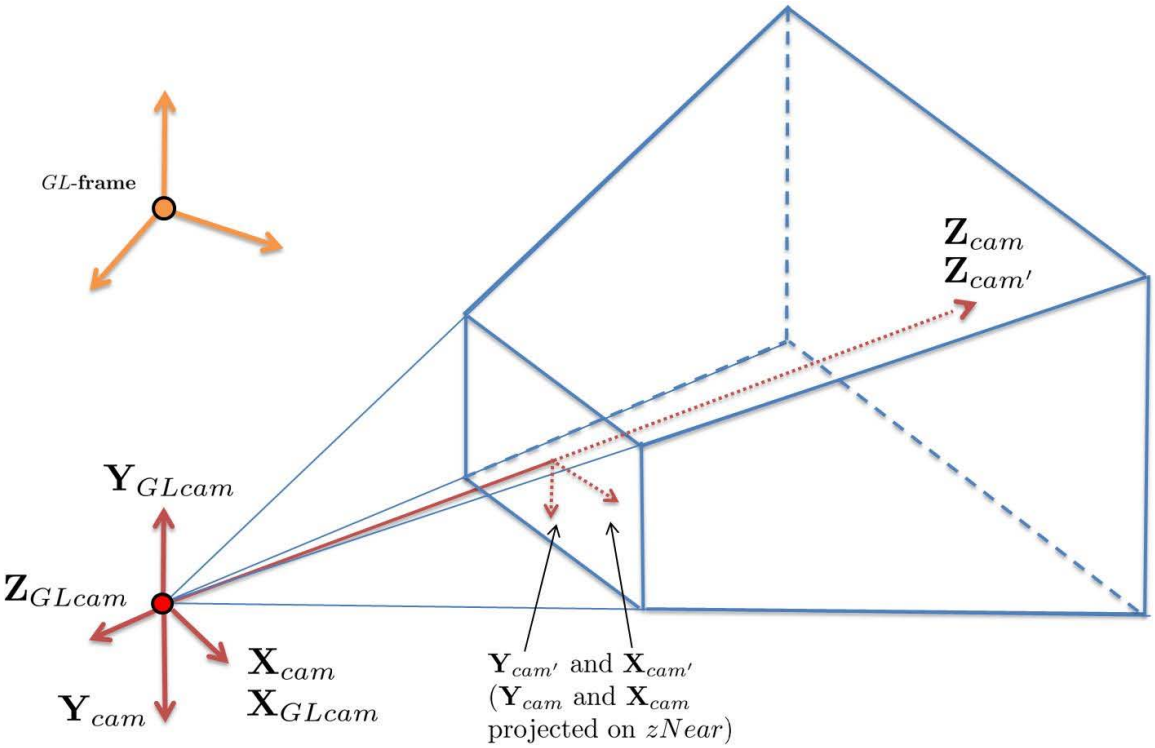


Figure 5.22: The $GLcam$-frame and cam-frame. They are collocated to permit the use of the camera parameters to transform the pyramid frustum to the rectangular shape. The projection of the cam-frame onto the near clipping plane denotes the cam'-frame.

By maintaining a symmetric viewing window (not accounting for off-center principal points), the transformation can be broken down into components:

$$\underline{\mathbf{P}}^{NDC} = \mathbf{T}_2 \, \mathbf{T}^{NDC}_{cam'} \mathbf{K}^{cam'}_{cam} \mathbf{T}^{cam}_{GLcam} \underline{\mathbf{P}}^{GLcam} \tag{5.5}$$

$$\mathbf{T}_2 = \begin{bmatrix} \frac{2}{W} & 0 & 0 & 0 \\ 0 & \frac{2}{H} & 0 & 0 \\ 0 & 0 & \frac{2}{zNear - zFar} & \frac{zFar + zNear}{zNear - zFar} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.6}$$

$$\mathbf{T}^{NDC}_{cam'} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}^{cam}_{GLcam} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.7}$$

$$\mathbf{K}^{cam'}_{cam} = \begin{bmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & -zFar - zNear & zNear \cdot zFar \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.8}$$

where, $\mathbf{T}^{NDC}_{cam'}$ and $\mathbf{T}^{cam}_{GLcam}$ account for the difference in orientation of the frames; $\mathbf{T}_2$ does not convert frames and only scales the points in the NDC-frame, and $\mathbf{K}^{image}_{cam}$, as described in Section 2.1.3.5, was 3x4 and the 4x4 $\mathbf{K}^{cam'}_{cam}$ is required in this equation.

To account for off-set principal points, the creation of $\mathbf{T}_2$ (or glOrtho( )) is offset by the width and height of the desired image, (right=W, left=0, top=0, and bottom=H) and $\mathbf{K}^{cam'}_{cam}$ is offset in the opposite direction by the amount of the principal points:

$$\mathbf{T}_2 = \begin{bmatrix} \frac{2}{W} & 0 & 0 & -1 \\ 0 & \frac{2}{H} & 0 & -1 \\ 0 & 0 & \frac{2}{zNear - zFar} & \frac{zFar + zNear}{zNear - zFar} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.9}$$

$$\mathbf{K}^{cam'}_{cam} = \begin{bmatrix} \alpha & 0 & x_o & 0 \\ 0 & \beta & y_o & 0 \\ 0 & 0 & -zFar - zNear & zNear \cdot zFar \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.10}$$

The two translations ($\mathbf{T}_{cam'}^{NDC}$ and $\mathbf{T}_{GLcam}^{cam}$) can now be accounted for in a single $\mathbf{K}$ matrix, relabeled $\mathbf{K}_{GL}$, including the addition of the skewing value which is solved in the same manner as the skew in Section 2.1.3.5:

$$\underline{\mathbf{P}}_1^{NDC} = \mathbf{T}_2 \, \mathbf{K}_{GL} \underline{\mathbf{P}}_1^{GLcam} \tag{5.11}$$

$$\mathbf{T}_2 = \begin{bmatrix} \frac{2}{W} & 0 & 0 & -1 \\ 0 & \frac{2}{H} & 0 & -1 \\ 0 & 0 & \frac{2}{zNear - zFar} & \frac{zFar + zNear}{zNear - zFar} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.12}$$

$$\mathbf{K}_{GL} = \begin{bmatrix} \alpha & -s & -x_o & 0 \\ 0 & \beta & -y_o & 0 \\ 0 & 0 & -zFar - zNear & zNear \cdot zFar \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.13}$$

In place of the function call to glFrustrum( ), glLoadMatrixf( ) loads the combination of these two matrices. As an important note, contrary to the matrix multiplication conventions presented here, OpenGL post-multiplies matrices, so the combination of these two matrices has to be transposed before using the glLoadMatrixf( ) command. The next section introduces a validation concept for future work using OpenGL.

5.5.2 Validation. Because of the many transformations, units, data sources, and people involved in this research effort, many items were not overseen by the author. Many items were assumed accurate until proven otherwise, which proved to be an incorrect assumption. Because of the many sources of information, an ad-hoc validation process was created to address some these problems. The validation process amounted to rendering the truth data and visually determining the accuracy of the entire process: the truth data, the model, and the OpenGL rendering process. It was quickly determined that matching images in an un-validated process introduced significant errors in the ultimate solution.

The truth data collected were also used in this validation process. For future work this data should be from a dissimilar truth-data set. Because of this, errors

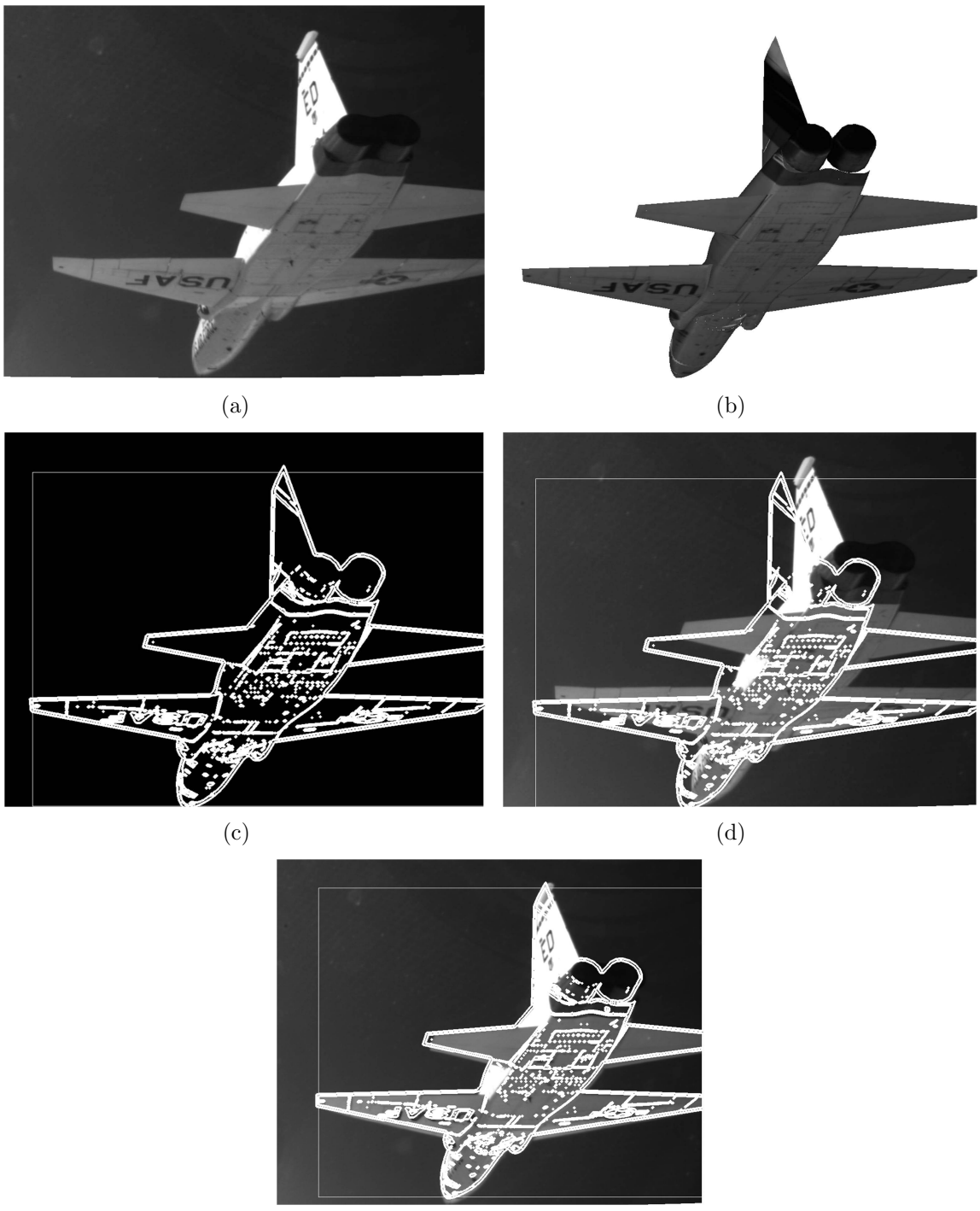
associated with the validation of the process can not be separated from the navigation-solution algorithm.

The validation process involved rendering images using the truth data and comparing them with collected images from the same time instant. Because of the following issues: errors in boresighting, data collection equipment, truth data collection, model creation, and OpenGL implementation, there will be some visual differences between rendered and collected images. The goal of validation is to minimize the visual difference between the two types of images by determining these process errors, truth-data biases, and incorrect process operations. The validation process defined here was based on human estimation of image-match comparison, because this was an unforseen complication in the research. Better comparison methods are available and should be addressed more in-depthly in future projects involving OpenGL. Comparisons methods such as those described in Section 2.2.2.2 (non-template matching versions) can be used for this validation process.

A quick description of the validation process is presented, with an example shown in Figure 5.23, followed by the determined errors.

Taking truth data from various times throughout the data collection process, $\mathbf{I}_r$ images were created with the same parameters as the collected $\mathbf{I}_c$ image. A contour function (Section 2.2.2.1) was used to determine the outline of the rendered aircraft in the $\mathbf{I}_r$. This outline was added to the $\mathbf{I}_c$ for a visual analysis of similarity. A created graphical user interface (GUI) perturbed the rendered aircraft in all six degrees of freedom to allow a more accurate overlay of the $\mathbf{I}_r$ on the $\mathbf{I}_c$. The perturbations in the GUI could be accomplished in the *cam*-frame or *b*-frame of either aircraft, as changing translation and attitude in these each of these frames results in different visual changes.

By perturbing the truth values of multiple images at different times during the flight (at different attitudes and different distances between the aircraft) a validation of the OpenGL rendering process was determined. The validation process discovered



(a) (b)

(c) (d)

(e)

Figure 5.23: OpenGL validation process.
(a) $\mathbf{I}_c$ from the test flights.
(b) Truth data collected with each $\mathbf{I}_c$ is used to create an $\mathbf{I}_r$.
(c) An edge detector was used to determine the contour of the $\mathbf{I}_r$.
(d) The contour outline was added to the $\mathbf{I}_r$ for visual similarity comparison.
(e) Required perturbations about the truth state determine the validity of the process.

that there were, at least, five biases consistent throughout the flight. The first two were obviously caused by errors in the boresighting of the C2B and/or the inclusion of those values into the truth data. Neither of these two processes involved the author and no determination about them could be made. These biases were confirmed at various other portions of the flight, during cruise, landing, and takeoff. Aircraft in these flight phases are generally at zero degrees of bank, yet both aircraft were consistently showing approximately five degrees of bank during these times.

After removing the roll biases, the other three biases were between the truth sources. Throughout the validation process, the required translational perturbations at each position were fairly consistent in each axis. These translational differences were averaged across the flight for an overall estimated bias. The cause of these biases was never determined; however, they were fairly constant throughout the flight (the standard deviation between the determined biases was between five and ten *inches*). They were most likely from errors in the addition of the boresighting values to the truth data.

The perturbation required for correcting the $\mathbf{I}_r$ in Figure 5.23 to overlay on the $\mathbf{I}_c$ was approximately ten *inches* in both the camera $\mathbf{Y}$ axis and $\mathbf{Z}$ axis. Because of the difficulties in determining translations in the $\mathbf{X}$ axis for both humans and computers, the visual-estimate validation process is not as accurate as it could be.

As part of the entire process, an improvement to the conversion between OpenGL to OpenCV images is presented next.

5.5.3 OpenGL to OpenCV improvements. It was evident early in the research that the process detailed in Section 2.2.3 is not as efficient as it needs to be. Rendering multiple images by OpenGL is quick and so is the comparison of those images (once converted) in OpenCV. However, to utilize the benefits of each library, a quicker conversion between them is needed.

This research used a slightly more efficient method as a partial solution, shown in Listing 5.1. This method uses an OpenGL function, glReadPixels( ) (also used

in the example in Section 2.2.3) to copy pixels of the image to another memory location. However, instead of the pixel-parsing process of Section 2.2.3, the entire image transfers to OpenCV collectively, in one step, with the image still requiring a flip.

Listing 5.1: Simpler OpenGL to OpenCV image conversion pseudocode

```
glReadPixels(width, height, CVimage->imageData)
cvFlip(CVimage)
```

This warrants some explanation. Even though the OpenCV references the pixels of an image in a matrix as shown in Figure 2.24, the memory storage used is similar to the OpenGL image storage as shown in Figure 2.23. OpenCV simply accesses those memory locations for the user, through the IplImage structure. This does not completely eliminate the delay needed to transfer the image, mainly because of the inefficiency of the glReadPixels( ) function. This function was created to copy a portion of the screen. It was not created to read an entire image. The image must first be rendered, placed into the video buffer, and all other OpenGL commands completed before glReadPixels( ) is allowed to run and find the pixels in the user's region of interest. A more efficient solution would possibly render the OpenGL image directly to the memory location of a blank OpenCV image, skipping the video buffer completely. Finding an efficient conversion between the two libraries was not further researched, but will be critical to the success of this approach to navigation.

5.6 Summary

The RIPE algorithm as presented was able to meet three of the four goals outlined in Chapter 1. The presented algorithm, despite multiple discovered sources of error met the following goals: it increased the accuracy of *whole-aircraft* tracking, required no modification to the lead aircraft, and used only open source programming libraries. The algorithm did not satisfactory complete the fourth goal: complete navigation updates in a timely manner.

The errors relating to the position estimation of RIPE ranged from 0-9% based on distance between the aircraft, including the most dynamic of maneuvers and challenging environmental noise. Average error in the pre-contact position and closer was 2-3%, or less than two *feet* error at a distance of 62.5 *feet*. Based on the author's experience with AR, this is as accurate as most pilots in the contact position and more accurate than pilots in the pre-contact position. Because the errors were often close to or less than the accuracy of the truth data, an accurate characterization is difficult; however, this process consistently showed less than five *feet* of error in the contact position, meeting the objective. The assumption that a boom operator can fly the boom as needed in the envelope to effect the rendezvous, this accuracy is sufficient to permit AAR.

From the pictures taken of the aircraft, no texture or paint modifications were done to the lead aircraft. A model of the aircraft using current paint schemes was determined to be adequate.

The open source C programming libraries worked well for this process and within each library, operations were quick enough to accomplish the necessary processing. However, the conversion between libraries was not efficient enough to permit real-time processing. Until an efficient method is determined, or other processing time efficiencies are discovered, the RIPE process using OpenGL and OpenCV will be difficult to implement real-time.

VI. Conclusions

This thesis presented a method of image-aided navigation for AAR. This chapter concludes the thesis with a final assessment of the methods presented and recommendations for future research.

6.1 Conclusions

Many aspects of the RIPE algorithm worked well: the rendering aspects of OpenGL in conjunction with the image manipulation and comparison functions of OpenCV provided accurate representations and analysis of the real-world. Other aspects of the RIPE approach were not optimally implemented: the incorrect use of *size* in template matching, perturbations which were too small to cause visual differences in appearance, and most obviously processing speed.

6.1.1 OpenGL. With a new method to represent a camera in OpenGL, the renderings produced were remarkably similar to the camera images. With a photo textured bottom on the aircraft, high matching values between the two images were possible. But, as noticeable in many of the photos throughout the thesis, the OpenGL lighting never accurately represented the real-world. This is because of time limitations in this thesis, not because of OpenGL. In fact, this limitation drove the algorithm to miss-match images (find the direct opposite image), requiring a very light background in the renderings instead of the dark background of the collected images. This ultimately added to the errors reported in Chapter 5. A better lighting environment, including ambient and direct sunlight will create a better representation of the real-world, thus reducing a portion of the estimation error. With the location of the aircraft known in addition to the time of day flown, the sun's trajectory with respect to the aircraft can be modeled and included in the lighting scheme. By modeling a better lighting environment, it might be possible to use the simple matching algorithms presented in Chapter 2 rather than developing more complex ones.

Other limitations to the OpenGL process was the model itself. It was never fully completed. Many aspects of the aircraft were not accounted for in the model used - most notably the remaining paint schemes including the vertical tail and sides of the fuselage in addition to the oblong, curved disk at the top of the vertical tail. The backup model accounts for a few of these aspects, but is still not complete and was not validated in time for completion of this thesis.

After incorporating the new camera representation, the partially completed model, and after the validation process, the location, attitude, and size of the rendered aircraft closely matched the true aircraft. There was some residual error as a result of the arbitrarily located zFar clipping plane. Because of the scaling involved in the OpenGL process, an infinite zFar clipping plane is not possible and recreating a real-world scene at far distances was challenging (beyond the distances of contact and pre-contact). The errors at this far distance were exacerbated by the relatively small sized perturbations at the farther distance. This either limits the distance away an object can be for the RIPE approach to be valid, or dynamic zNear and zFar clipping planes are needed to compensate.

6.1.2 OpenCV. The performance of OpenCV was remarkable. The contouring and template matching function were essential to the RIPE approach. Improvements to the template matching process are possible: a more thorough trigonometric equation might possibly reduce some of the estimation errors. Overall it worked very well for the position and orientation estimation. The template matching did suffer at farther distances, mainly because of resolvable distances of the images. Another method, such as the center and tips approach in [15], might improve performance at farther distances, with a transition to RIPE at closer distances. This would also allow the zFar plane to remain a fixed distance, closer to the camera origin.

6.1.3 RIPE. The research presented, including the resulting errors, demonstrated RIPE as a valid approach to AAR and potentially pose in general. Admittedly, aircraft motion in AR is relatively benign and the RIPE approach leverages the actual

dynamics of the aircraft to balance speed with accuracy. It was intended that the process would estimate the position in the $\mathbf{Y}_{cam}$ axis as well as it did in the $\mathbf{X}_{cam}$ axis. They are essentially very similar; however, the size group ($\mathbf{Z}_{cam}$ axis) influence on the $\mathbf{Y}_{cam}$ axis was apparent. This was a result of implementing the template matching process with Cartesian coordinates. Another method, such as the digital zoom alternate method presented in Chapter 4 or the use of spherical coordinates in the *cam*-frame, could eliminate errors associated with this interaction. Ultimately, the approach adequately performs as needed. Further projects involving accurate implementation of the RIPE method are required to understand fully its capabilities and limitations.

6.1.4 Processing Speed. The slow processing speed of transferring images between libraries was unexpected. The process was exacerbated by the number of required images to render and the size of each image. No efficiencies, other than the OpenGL to OpenCV process presented in Chapter 5 were pursued. Recommendations for decreasing the time to make estimations is presented in the next section.

6.2 Future Research

Currently, the main limitation to the RIPE process applied to AAR is the processing speed to make estimations based on the images. Other limitations have been presented and possible improvements are discussed in this section. In addition to improvements of the RIPE process, broadening applications and adaptations of its potential are presented.

6.2.1 Processing Speed. To improve the processing speed of the RIPE estimation, a few aspects can be considered: reduce the number of transfers from OpenGL to OpenCV per update, reduce the size of the required images, improve the transfer process, and change the coordinate frame for tracking to improve the tracking process such that less measurements are required.

The first option is to reduce the number of image transfers needed per update. This could be realized in two different methods: either reduce the number of rendered images used by the RIPE process or pre-render a batch of images. Since the presented process only required 10-15 images per update, reducing this number any further requires one of the presented alternate methods to RIPE (Section 4.3.1.4.) Additionally, this would only solve the problem for AAR, not for RIPE in general. Another method, pre-rendering images, could be accomplished completely prior to flight, or real-time with a parallel processor making estimates of where the aircraft might be and the images required. Either option has potential to help reduce a portion of the processing time.

A second option is to reduce the size of the required images. This method has potential to decrease the processing time dramatically. By halving both dimensions of the images used in the field work (1200 x 1600 to 600 x 800), the processing time could potentially be reduced by 3/4. A thorough analysis on the impact to precision would determine the viability in this approach. Most likely the largest impact would be to the farther distances, the pre-contact and contact position would not increase in error dramatically. Incorporating a secondary technique for further distances was also suggested for improving errors from the zFar clipping plane and resolvable distances and could also compensate for the smaller image size impact at further distances.

The next possible improvement, with the least impact on the theory presented, is to create a better transfer process. With a better understanding of the inner workings of both libraries and the video processing hardware, a more efficient method should be possible. The OpenGL process was built for speed, and getting an image to screen is the priority. If the renderings could be re-directed to a more appropriate memory location, accessible by OpenCV, the true limit to the number of rendered images would mainly be from hardware capacity.

Finally, by changing the coordinate frame to the n-frame, the benign nature of AR presented in Chapter 3 can be exploited. This method was used in other research

efforts [29], and is probably a more effective coordinate frame for this navigation. In this frame, the tanker is less dynamic, the required measurement updates can be reduced, permitting a longer processing time to make measurement updates.

Other improvements to the process have been discussed in the thesis: access the full potential of OpenGL lighting, determine a better image processing technique when the environmental model does not represent the real-world well enough (full sun in FOV), create a standardized validation process similar to a camera calibration process, and investigate the use of spherical coordinates in the *cam*-frame to implement the template-matching theory. With the spherical coordinate system, the distance from the camera origin can be adjusted without movement to the two angles that make up the position.

With these improvements, the theory presented here can make improvements to the current pose estimation with a point tracking process. With improving processing speed and a maturing method, the RIPE process can be applied to a wide variety of pose estimation problems.

6.2.2 Alternate Applications and Adaptations. With the full capability of the theoretical RIPE process, many applications including AAR could benefit.

Within OpenGL, multiple cameras can be defined with different viewing angles and positions from a rendered object. Incorporating a two (or more) camera cluster with the RIPE process could decrease many of the errors presented. With the cameras separated by a large enough distance, determining the image location could potentially bypass the sizing group completely. This could be accomplished through epi-polar line estimation, or triangulating the objects position, without the need to determine its distance from either camera. Two or more cameras with known position and orientation with each other can improve the RIPE process by reducing the number of rendered images required, while improving accuracy.

Also within the OpenGL capabilities is rendering multiple objects, including representing multiple objects that are connected (articulated). Multiple-object OpenGL

scenes can represent dynamic environments accounting for each object's additional degrees of freedom, including objects that are attached to a single aircraft. Objects such as flight controls or the tanker boom or drogue can be represented in OpenGL and their state information determined through a parallel RIPE process. With the knowledge of a boom's location and extension, the visual representation of the tanker is improved and the pose on the aircraft is more accurate. The work in [6] demonstrated articulated objects in both single and multiple camera arrangements with a wire frame model of an object. While this did not utilize OpenGL, many of the same principles apply. Such a process would increase the number of rendered images required and many process improvements would be required to operate it real-time.

Other adaptations of the RIPE process include different types of cameras (infrared, 3D, fisheye lens), applications (manufacturing, space vehicles, missile defense), or model types (simple or symmetric). A different kind of camera or lens, such as an infrared combination, would permit operations in night and in poor visibility. Manufacturing applications could further tailor the quick-RIPE method, reducing the needed rendered images for faster operations. Objects on an assembly line, for example, might only have two DOFs to make perturbations about. Highly-symmetric object applications (such as tracking missiles) could make use of *a priori* collected photos, instead of renderings, of the object with perturbations in only one or two axis. This would create a database of images of the object in these DOFs. This database, in conjunction with a digital zoom, could bypass the rendering and transferring of images real-time completely.

With the current processing capabilities of contemporary computers, areas that could benefit immediately would have fewer DOFs of the object(s) in the scene or an object with limited dynamic motion potential. For example, land or naval-based navigation applications would operate faster with a decreased number of rendered images because of the limited capability of objects in these environments. These environments would have fewer DOFs, fewer required perturbation images, and therefore quicker

processing. In conclusion, the current limitations notwithstanding many applications requiring pose could be improved with a RIPE-type process.

6.3 Summary

The science of image-aided navigation is still in its infancy. Many methods and alternatives provide specific solutions to specific problems as a balance of each method's benefits and weaknesses. As a viable method, the theory of this thesis has been proven adequate in some areas and weak in others. As an approach to AAR, the theory has identified those aspects that limit its full capability. With the proposed improvements, an efficient RIPE process can serve as augmentation and backup to DGPS in solving not only the AAR problem, but pose in general.

Bibliography

1. AIT Computer Vision. "Saving image from OpenGL to OpenCV (IplImage format)". Instructional article. n. pag. `http://se.cs.ait.ac.th/cvwiki/gltocv:main`, 11 February 2011.
2. Bouguet, Jean-Yves. "Camera Calibration Toolbox for Matlab®". Instructional article. n. pag. `http://www.vision.caltech.edu/bouguetj/calib_doc/`, 11 February 2011.
3. Bradski, Gary and Adrian Kaehler. *Learning OpenCV*. Sebastopol, CA: O'Reilly Media, Inc., 2008.
4. Brown, Robert Grover and Patrick Y.C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering* (3rd Edition). New York: John Wiley & Sons, Inc., 1997.
5. Calise, Anthony J., James C. Neidhoefer, Yoko Watanabe Jincheol Ha, and Eric N. Johnson. "Real-Time Vision-Based Relative Aircraft Navigation". *AIAA Journal of Aerospace Computing Information*, 4(4):707–738, 2007.
6. Drummond, Tom and Roberto Cipolla. "Real-time visual tracking of complex structures". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):932–946, 2002.
7. Edwards AFB, Ridley, Range Division, TSPI Section. "Sensor Accuracy Chart", Handout for TSPI customers. 21 May 2010.
8. Hartley, Richard and Andrew Zisserman. *Multiple View Geometry in computer vision* (2nd Edition). Cambridge: Cambridge University Press, 2003.
9. Heikkilä, Janne and Olli Silvén. "A Four-step Camera Calibration Procedure with Implicit Image Correction". *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 22(10):1106–1112, 1997.
10. Howard, James M. et al. *Limited Evaluation of an Image and Data Collection System for Air Refueling*. USAF TPS Technical Information Memorandum (unpublished), Edwards AFB, CA: USAF Test Pilot School, December 2010.
11. Integrated Systems, Western Region Northrop Grumman Corporation. *Automated Aerial Refueling Hybrid System Phase II Program, Station Keeping Flight Test Hybrid System Report*. Technical Report FA8650-04-2-3450, n.d.
12. Ma, Yi, Stefano Soatto, Jana Kosecka, and S. Shankar Sastry. *An Invitation to 3-D Vision*. New York: Springer-Verlag, Inc., 2004.
13. National Imagery and Mapping Agency. *Department of Defense World Geodetic System 1984: its definition and relationships with local geodetic systems*. Technical Report TR8350.2, National Imagery and Mapping Agency, St. Louis, MO, USA,

July 1997. URL http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html.

14. NATO Standardization Agency (NSA). *ATP 3.3.4.2 Air To Air Refueling ATP-56(B) Change 2, Part 5 - National Annex, Annex ZA - USA, KC-135 Stratotanker*. January 2010.

15. Oh, Seung-Min and Eric N. Johnson. "Relative Motion Estimation for Vision-based Formation Flight using Unscented Kalman Filter". *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2007.

16. Ross, Steven M. *Formation Flight Control For Aerial Refueling*. MS thesis, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, March 2006. AFIT/GAE/ENY/06-M35.

17. de Ruiter, Hans. *3D-Tracking of A Piriori Unknown Objects in Cluttered Dynamic Environments*. Ph.D. thesis, Graduate Department of Mechanical and Industrial Engineering, University of Toronto, 2008.

18. de Ruiter, Hans and Beno Benhabib. "Visual-model-based, real-time 3D pose tracking for autonomous navigation: methodology and experiments". *Auton Robot*, 25:267–286, 2008.

19. Segal, Mark and Kurt Akeley. "The OpenGL Graphics System: A Specification". Silicon Graphics, Inc. technical specification. http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf, 11 February 2011.

20. Sempere, Juan. "File:Plane.svg". Original artwork. n. pag. http://en.wikipedia.org/wiki/File:Plane.svg, 11 February 2011.

21. Shreiner, Dave. *OpenGL Programming Guide* (7th Edition). Upper Saddle River, NJ: Pearson Education, Inc., 2009.

22. Simek, Kyle. "Simulating Calibrated Cameras in OpenGL". Instructional article. n. pag. http://sightations.wordpress.com/2010/08/03/, 11 February 2011.

23. Spencer, James H. *Optical Tracking for Relative Positioning in Automated Aerial Refueling*. MS thesis, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, March 2007. AFIT/GE/ENG/07-22.

24. Spinelli, Christopher J. *Development and Testing of a High-Speed Real-Time Kinematic Precise DGPS Positioning System Between Two Aircraft*. MS thesis, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, September 2006. AFIT/GCD/ENG/06-12.

25. Titterton, David and John Weston. *Strapdown Inertial Navigation Technology* (2nd Edition). Peter Peregrinus Ltd, 2004.

26. United States Government Accountability Office. *UNMANNED AIRCRAFT SYSTEMS Comprehensive Planning and a Results-Oriented Training Strategy Are Needed to Support Growing Inventories*. Report to the Subcommittee on

Air and Land Forces, Committee on Armed Services, House of Representatives GAO-10-331. Washington:GAO, March 2010. URL http://www.fas.org/irp/gao/uas2010.pdf.

27. Veth, Michael J. *Fusion of Imaging and Inertial Sensors for Navigation*. Ph.D. dissertation, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, September 2006. AFIT/DS/ENG/06-09.

28. Warwick, Graham. "Navy Wants NG X-47B To Demo Aerial Refueling". On-line periodical article. n. pag. http://www.aviationweek.com/aw/generic/story_generic.jsp?channel=defense&id=news/UCAS120808.xml&headline=Navy%20Wants%20NG%20X-47B%20To%20Demo%20Aerial%20Refueling, 11 February 2011.

29. Weaver, Adam D. *Characterization of a Robust Vision-Aided Technique for Autonomous Aerial Refueling*. MS thesis, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, March 2009. AFIT/GE/ENG/09-45.

30. Wu, Allen D., Eric N. Johnson, and Alison A. Proctor. "Vision-Aided Inertial Navigation for Flight Control". *Journal of Aerospace Computing, Information, and Communication*, 2:348–360, 2005.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24–03–2011	Master's Thesis	Sept 2008 — Mar 2011

4. TITLE AND SUBTITLE

Image Dependent Relative Formation Navigation for Autonomous Aerial Refueling

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

James Howard, Maj, USAF

5d. PROJECT NUMBER

ENG 09-319

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/GE/ENG/11-16

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

Air Force Research Laboratory (Jacob Campbell)
2241 Avionics Circle
Wright-Patterson Air Force Base, OH 45433-7333
((937) 255-6127 x4154 jacob.campbell@wpafb.af.mil)

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RYRN

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

12. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research tests the feasibility, accuracy, and reliability of a predictive rendering and holistic comparison algorithm with use of an optical sensor to provide relative distance and position behind a lead or tanker aircraft. Using an accurate model of a tanker, an algorithm renders image(s) for comparison with actual collected images by a camera installed on the receiver aircraft. Based on this comparison, information used to create the rendered image(s) is used to provide the relative navigation solution required for autonomous air refueling. Given enough predicted images and processing time, this approach should reliably find an accurate solution. Building on previous work, this research aims to minimize the number of required rendered images to provide a real-time navigational solution with sufficient accuracy for an auto-pilot controller installed on future Unmanned Aircraft Systems.

15. SUBJECT TERMS

Automated Aerial Air Refueling, Unmanned Aircraft System, OpenGL, OpenCV, Predictive RIPE Rendered Image Position Orientation Estimation, Template Matching, Whole Holistic Group Tracking, Tanker, Receiver, Kalman Filter, Image-Aided Navigation, Vision, 3D Model

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Michael J. Veth, Lt Col, USAF (ENG)
U	U	U	UU	188	19b. TELEPHONE NUMBER *(include area code)* (937) 255–3636 ext 4541; mike.veth@us.af.mil