3D Relative Position and Orientation Estimation for Rendezvous and Docking Applications Using a 3D Imager

A thesis presented to

the faculty of

the Russ College of Engineering and Technology of Ohio University

In partial fulfillment

of the requirements for the degree

Master of Science

Amy T. Scheithauer

March 2010

This thesis titled

3D Relative Position and Orientation Estimation for Rendezvous and Docking Applications Using a 3D Imager

by

AMY T. SCHEITHAUER

has been approved for

the School of Electrical Engineering and Computer Science

and the Russ College of Engineering and Technology by

Maarten Uijt de Haag

Associate Professor of Electrical Engineering

Dennis Irwin

Dean, Russ College of Engineering and Technology

ABSTRACT

SCHEITHAUER, AMY T., M.S., March 2010, Electrical Engineering

3D Relative Position and Orientation Estimation for Rendezvous and Docking Applications Using a 3D Imager (107 pp.)

Director of Thesis: Maarten Uijt de Haag

This research investigates 4-rotor Uninhabited Aerial Vehicle (UAV) rendezvous and docking precision navigation using 3D imaging sensor data and MATLAB software. The orientation of a UAV is determined relative to a docking station, and in future research the UAV's controller implementation will incorporate the orientation to enable autonomous precision docking. The 3D imaging solution leverages fixed docking station geometry, which is a square based pyramid with a square top surface. This shape has 5 planar surfaces that can be identified by the sensor processing software.

This implementation builds on previous sensor processing research at Ohio University and expands the initial capability in a series of three subsystems. Subsystem one detects planar surfaces from a 3D point cloud docking station data set. Subsystem two consistently identifies the sides of the docking station. Subsystems three implements Horn's algorithm to detect UAV orientation with respect to the dock.

The presented data identifies UAV rotation and translation with respect to the dock using actual data. The research verifies the solution using simulated data.

Approved: _____

Maarten Uijt de Haag

Associate Professor of Electrical Engineering

ACKNOWLEDGEMENTS

I would like to sincerely acknowledge Dr. Maarten Uijt de Haag for his support and guidance in this research effort and thesis development. He supported my education through my undergraduate and graduate programs as a professor, advisor and mentor. He has gone out of his way to continue to support my research effort while I worked virtually on my thesis. I am sincerely grateful for all of his hard work.

The professors on my thesis committee, Dr. Frank van Graas, Dr. Michael Braasch, and Dr. Gene Kaufman, all supported me with their genuine passion for teaching as I studied their coursework in undergraduate and graduate school. Furthermore, I'd like to thank them for their time and efforts supporting my thesis.

The EE Ohio University Graduate Office, especially Tammy Jordan and Julie Venrick, have provided great support and enabled me to complete this thesis.

I would also like to thank everyone in my personal and work life for supporting my research efforts. My parents Rick and Maureen Scheithauer have always been supportive and continue to support me during this effort, especially while I worked on it through many holidays. My sisters, Mindy and Sarah Scheithauer, and their roommates hosted me at Ohio University during the thesis working meetings I had on campus.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

CHAPTER 1: INTRODUCTION

Uninhabited Aerial Vehicles (UAVs) operate without the need for a human onboard. They navigate either autonomously using preprogrammed algorithms or remotely under human control. Three major UAV application areas are military, space, and commercial.

UAVs are a key research development area and initiative for the military. The Department of Defense (DoD) has developed a Fiscal Year (FY) 2009-2034 Unmanned Systems Integrated Roadmap specifically targeted at unmanned system development [1]. In this report, the Joint Unmanned Aircraft Systems (JUAS) Center of Excellence (COE) categorizes military UAVs based on their weight, altitude and speed as shown in **Figure 1-1** [1]. Military UAVs range in performance and cost depending on their mission and capabilities.

Many major defense contractors have initiated UAV development. A few military UAVs are the MQ-1 Predator, the RQ-4 Global Hawk, the XM-156 Class I and the A160 Hummingbird. General Atomics manufactures the MQ-1 Predator, which is a Group 4 UAV with multi-mission capability including surveillance and the ability to deploy "precision guided munitions" [1]. 120+ platforms have been delivered for the Air Force, Army and Navy. Northrop Grumman manufactures the RQ-4 Global Hawk, which is a Group 5 "high-altitude, long-endurance unmanned aircraft designed to provide wide area coverage" [1]. 12 platforms have been delivered to the Air Force. Honeywell manufactures the XM-156 Class I, which is a Group 1 UAV. It "provides the ground Soldier with Reconnaissance, Surveillance, and Target Acquisition (RSTA)" and has 90 systems planned per Future Combat Systems Brigade Combat Team [1]. Boeing

manufactures the A160, which is a Group 4 long-endurance Vertical Take-Off and Landing (VTOL) UAV with 6 platforms delivered to the Army, Navy and DARPA [1].

UAS Category	Maximum Gross Takeoff Weight (lbs)	Normal Operating Altitude (ft)	Speed (KIAS)	Current/Future Representative UAS
Group 1	0-20	< 1,200 AGL	100 kts	WASP III, Future Combat System Class I, TACMAV RQ-14A/B, BUSTER, BATCAM, RQ-11B/C, FPASS, RQ-16A, Pointer, Aqua/Terra Puma
Group 2	21-55	< 3,500 AGL	< 250 kts	Vehicle Craft Unmanned Aircraft System, ScanEagle, Silver Fox, Aerosonde
Group 3	< 1,320	< 18,000 MSL		RQ-7B, RQ-15, STUAS, XPV-1, XPV-2
Group 4	> 1,320		Any Airspeed	MQ-5B, MQ-8B, MQ-1A/B/C, A-160
Group 5		> 18,000 MSL		MQ-9A, RQ-4, RQ-4N, Global Observer, N-UCAS

Note: Lighter than air vehicles will be categorized by the highest level of any of their operating criteria.

(1) Group 1 UA: Typically weighs less than 20 pounds and normally operates below 1200 feet AGL at speeds less than 250 knots.

(2) Group 2 UA: Typically weighs 21-55 pounds and normally operates below 3500 feet AGL at speeds less than 250 knots.

(3) Group 3 UA: Typically weighs more than 55 pounds but less than 1320 pounds and normally operates below 18,000 feet MSL at speeds less than 250 knots.

(4) Group 4 UA: Typically weighs more than 1320 pounds and normally operates below 18,000 feet MSL at any speed.

(5) Group 5 UA: Typically weighs more than 1320 pounds and normally operates higher than 18,000 feet MSL at any speed.

Figure A.4. JUAS CONOPS UAS Categories

***Figure 1-1.* JUAS UAS UAV Categorization.** UAVs are categorized based on their weight, altitude and speed. Figure is from DoD FY 2009-2034 Unmanned Systems Integrated Roadmap [1].

The government's current UAV funding initiatives drive much of this work. The President's Unmanned Aircraft System budget over the next 5 years (2009-2013) is $15.489 billion [1].

Space applications have driven the need for UAV autonomous navigation for years due to the limited human interaction once launched into space. The "first unmanned vehicle in space was Sputnik" in 1957 [2]. Space continues to advance autonomous technology. In 2001, Blais et al. conducted a 3D laser scanner study to advance autonomous navigation aid in space, such as the International Space Station's Space Vision System [3].

In commercial operations, UAVs are used for a broad range of applications, such as human emergency response team assistance, aerial photography, aerial video, law-enforcement, and surveillance. Commercial applications drive the development of more affordable "Commercial-Off-The-Shelf" (COTS) technology, while military and space promote more advanced research.

Commercial companies, such as Draganfly Innovations Inc., sell aircraft that support many applications for government, military, industrial and education. One of Draganfly's UAVs, the Draganflyer X4, is a moderately priced battery powered quad rotor with customizable camera modules as shown in **Figure 1-2** [4]. The Draganflyer X4 advances affordable self stabilizing technology to increase photography and video quality taken from the UAV [4].

Figure 1-2. **Draganflyer X4.** Commercial applications such as the Draganflyer promote affordable UAVtechnology. Figures are from *draganfly.com* [4].

In emergency environments, such as collapsed buildings or bomb threat locations, UAVs can evaluate and map out the disaster zone and so avoid risking human life. Once a UAV evaluates the situation, the human teams can make a safe entry and focus on known victim locations. The RoboCup Rescue competition encourages autonomous navigation advancement in commercial search and rescue environments [5]. Uninhabited Ground Vehicle (UGV) platforms primarily compete in RoboCup Rescue, and UAVs benefit from the competition's affordable autonomous technologies. Ideally, UAVs will eventually replace or be used in addition to the UGV in search and rescue operations.

RoboCup Rescue also advances affordable technology by enabling fellow researchers to compete, observe, and exchange technology [5]. For example, the KURT3D participated in the RoboCup Rescue competition in 2004 and contributed to the advancement of 3D laser scanner technology by using it to map and navigate the course [6].

Ohio University's Avionics Engineering Center (AEC) is currently developing a UAV. Dr. Michael Stepaniak researched the UAV development in his thesis dissertation "A Quadrotor Sensor Platform" [7]. The Ohio University UAV is a quadrotor capable of carrying a 10.6 lb payload. This payload allows the UAV to carry a sensor such as the SICK LD-OEM1000 shown in **Figure 1-3**. The goal for future research incorporates the 3D imaging algorithms from this thesis into the Ohio University UAV's controller implementation to allow it to autonomously dock.

Figure 1-3. **Ohio University's UAV.** Ohio University is currently developing a UAV. The future goal is to incorporate the algorithms from this research into the UAV.

This research effort leverages previous work that has been accomplished and integrates these previous efforts together. In addition to integration, this research effort implemented specific new contributions that enable the advancement of precision rendezvous and docking. This research's contributions are as follows:

- Defined docking station geometry for this research effort
- Modified extract planes algorithm to detect smaller planar surfaces
- Implemented algorithm to detect docking station planes in pre-defined order
    - Derived geometric equations to identify candidate planes
    - Implemented reflectance algorithm to detect a reference plane
    - Implemented convex hull algorithm to detect docking station planes with respect to the reference plane
- Implemented algorithm to determine rotation and translation relative to the dock using data from the SR3000 3D imager
- Developed simulation to verify orientation and translation accuracy
- Determined algorithm's sensitivity to noise in the data
- Provided suggestions for future research efforts

The subsequent chapters of this thesis target UAV navigation, specifically the rendezvous and docking phase of flight. Chapter 2 describes the research problem statement and background. Chapter 3 discusses the proposed methodology and technology background. Chapters 4, 5, and 6 cover the technology implementation in three main subsystems. Specifically, Chapter 6, the focus of this study, discusses the UAV orientation algorithm. Chapter 7 expands future research recommendations, followed by the thesis conclusions.

CHAPTER 2: PROBLEM STATEMENT AND BACKGROUND

Many UAV applications and their associated mission operations are pertinent in current research, as discussed in Chapter 1. This research focuses primarily on an operational scenario in a military application, shown in **Figure 2-1**. Specifically within this operational scenario, the main research focuses on precision UAV navigation during the rendezvous and docking phase, allowing the UAV to navigate within inches of a dock station, as highlighted in Step 7 of **Figure 2-1**.

The operational scenario occurs in an urban military environment. The UAV is deployed on a mission, such as an Intelligence, Surveillance, Reconnaisance (ISR) mission (Step 1 in **Figure 2-1**). The UAV is physically loaded onto a docking station located on a ground vehicle, for example, a HUMVEE (Step 2). The ground vehicle travels to a location where the UAV is required. The UAV launches from the docking station and the ground vehicle departs and continues on its mission (Step 3). The UAV performs its ISR mission, navigating autonomously in the target environment while collecting and relaying intelligence regarding the environment or entities in the environment (Step 4).

Upon completion of its ISR mission, the UAV receives a signal to dock (Step 5). This signal could be initiated from a UAV and sent wirelessly to the ground vehicle, such as a message that the target data has been collected and the mission completed or an alert that the UAV's battery is low. The signal could also be initiated from the ground vehicle and sent wirelessly to the UAV. Once the signal is transmitted and received the ground vehicle finds a location where it can remain stationary for a few minutes and sends the approximate location to the UAV.

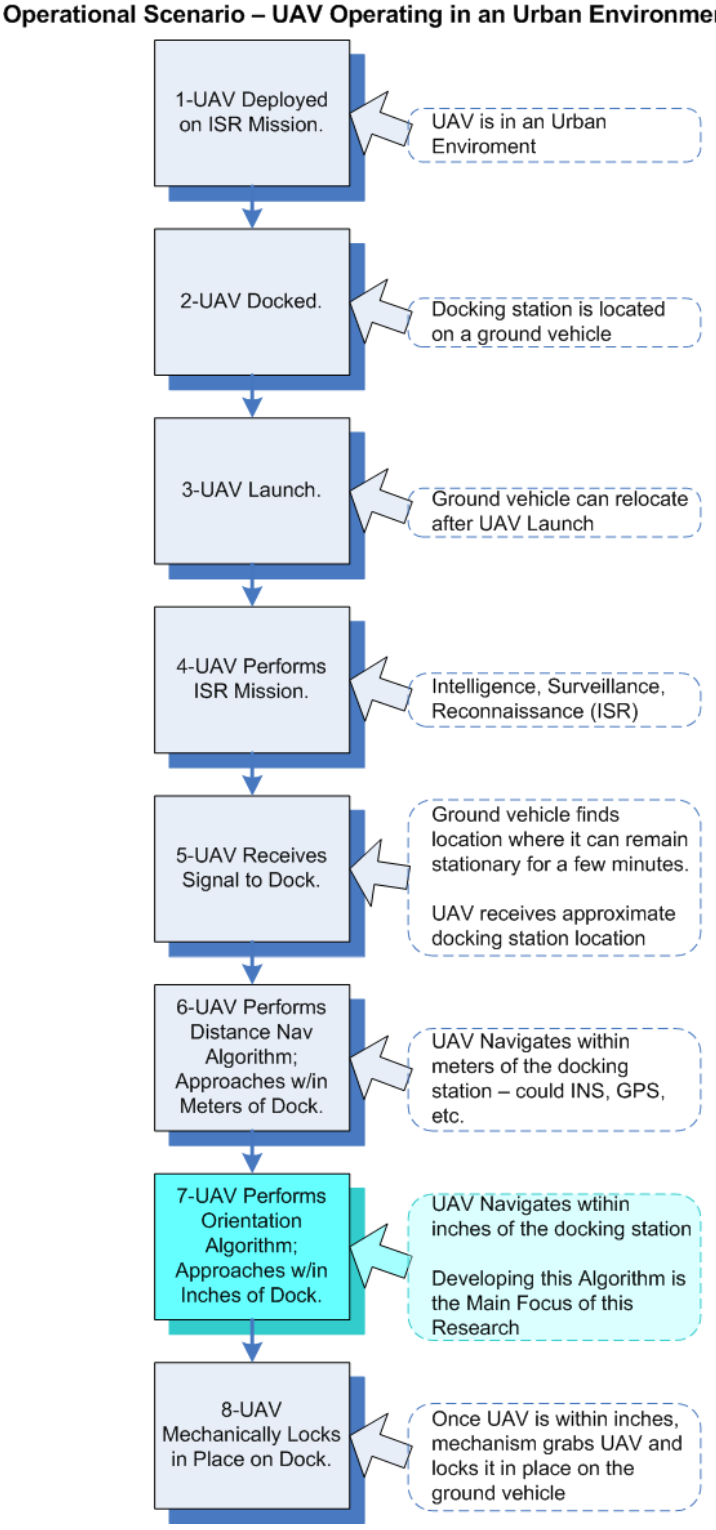


Figure 2-1. **Research Operating Scenario.** The UAV operates in an urban environment. This research focuses on Step 7, the UAV's orientation algorithm.

The UAV initiates a guidance and navigation procedure that positions the UAV to within a few meters of the ground vehicle location such that the docking port is within the field of view of the 3D vision sensor (Step 6). Once it is within a few meters the UAV initiates its precision docking algorithm, which navigates the UAV within a few inches of the docking station (Step 7). Performing navigation and orientation estimation in support of this last step is the main focus of this thesis.

As the UAV approaches the docking station within a few inches, some mechanism is used to lock the UAV into place on the docking station (Step 8). This ensures the UAV is secure before the ground vehicle continues on its mission. Once secured on the dock, the UAV is connected to the ground vehicle and can download its mission data onto a server in the ground vehicle, download its health status, upload its next ISR mission target, or charge its batteries. This mechanism is not part of this thesis research.

Many applications, other than this operational scenario, have a need for autonomous rendezvous and docking. For example, various space applications require autonomous rendezvous and docking based on the limited availability of human interaction when docking with the space station or satellites. Commercial applications may require a UAV to dock at a data station to upload data regarding the locations of dangerous obstacles or human victims.

The precision rendezvous and docking problem requires advanced navigation aids (navaids) on board the UAV. These navaids can include GPS and Inertial Measurement Units (IMUs), however, since our scenario is located in a GPS-challenged environment, an alternative method to GPS is considered here that uses a 3D imaging sensor to perform relative navigation with respect to the docking port upon visual acquisition. Of course

for all other phases of flight navigation aids should be available, but these are not in the context of this thesis. Additional UAV navigation requirements will be developed as the research matures in the future. For this operational scenario, the Ohio University four-rotor Vertical Take-Off and Landing (VTOL) platform discussed in Chapter 1 is being considered. Existing UAVs, such as the Draganflyer X4 detailed in Chapter 1, could also be used as the UAV platform.

Few fielded UAVs are capable of autonomous precision rendezvous and docking. The ScanEagle, developed by Insitu Group and Boeing, has the capability to operate from ships or remote areas. This is a Group 2 UAV that uses a near-vertical recovery system [1]. Although the recovery system provides a capability to rendezvous in remote areas using precision navigation without a runway, it requires significant ground support to dock as shown in **Figure 2-2** [8]. The UAV uses a hook to catch a rope hanging from a 15 meter (50 foot) pole [8]. The ScanEagle uses “a high-quality differential Global Positioning System unit mounted on the top of the pole and UAS.” [8]

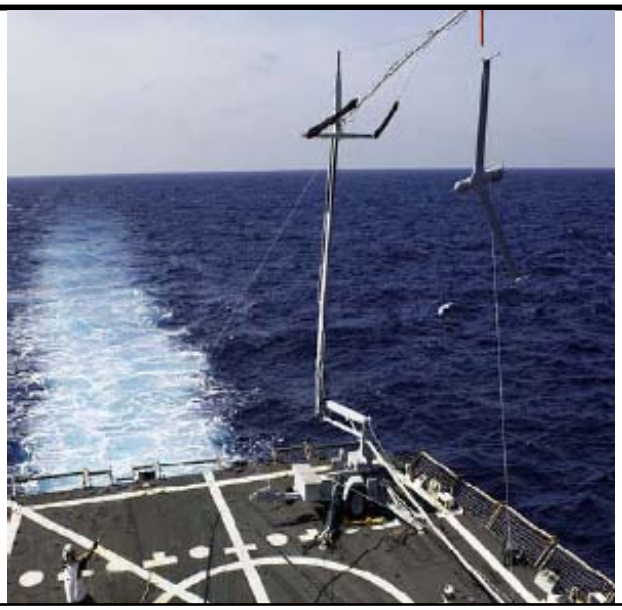


Figure 2-2. **ScanEagle Recovery System.** The ScanEagle requires a 50 foot infrastructure to autonomously rendezvous and dock. Figure is from reference [8].

The advantage of the rendezvous and docking system proposed in this research over current technologies is its portability and limited amount of ground structure. However, it is mainly applicable for rotorcraft UAV's. This solution reduces the large amount of obvious infrastructure and increases UAV landing accuracy.

CHAPTER 3: INTRODUCTION TO PROPOSED METHODOLOGY

Numerous research efforts have focused on precision rendezvous and docking for various docking applications. Space has been the primary application for remote or autonomous control research because it lacks direct access or control by a human. Many research papers discuss solutions for docking in space, such as the papers by Fenton et al. [9] and Subbarao and Dogan [10]. Fenton discusses a method using a 3D imaging sensor comparable to the methodology proposed in this research. Subbarao and Dogan incorporate a similar methodology into UAV control laws, which is a prime example of how this thesis can be expanded into future research by integrating it with UAV control laws.

A specific research paper that describes a similar method to the research discussed in this thesis is "Design of lidar-based sensors and algorithms for determining the relative motion of an impaired spacecraft," authored by Fenton et al [9]. Although Fenton's methodology is different in implementation, operational scenario, and (UAV) platform, their approach does use a LIDAR sensor to collect 3D data and compares the collected data with the known target data to infer the relationship (in position and orientation) between the sensor and the target object. There are also slight differences in nomenclature. In this thesis the rotation is referred to as the orientation whereas Fenton et al. refer to it as the registration [9].

Another research effort (described in "Intelligent Integrated Sensor, Control and Maneuver Mapping Architecture For Autonomous Vehicle Rendezvous and Docking", authored by Subbarao and Dogan) describes lower level rendezvous and docking control details [10]. Subbarao and Dogan's research incorporates control laws to maneuver the UAV based on inputs from a positioning module, which although important is beyond the

scope of the current research. A subset of their efforts uses a similar methodology to the current research efforts, specifically the positioning module that incorporates data from sensors to determine position and orientation. Since this is the same function that the research in this thesis performs, it helps emphasize the need and application for the current research. Furthermore, the overall approach presented by Subbarao and Dogan is similar to the proposed future research that is required to perform the others steps of the operational scenario defined in Chapter 2 [10].

Another common application outside of space is construction. Some researchers use techniques similar to rendezvous and docking to guide a construction crane and pick up objects. "Performance Evaluation of a High-Frame Rate 3D Range Sensor for Construction Applications", authored by Lytle, Katz and Saidi, describes an example of such a method for construction applications [11].

Aerial refueling is also an application for rendezvous precision navigation because a UAV would need to autonomously attach to a tanker to extend the UAVs range. Aerial refueling benefits include being a "force multiplier" (allowing the UAV to serve more missions) and increasing the UAVs "lethal capability" (allowing the UAV to carry less fuel and a larger payload) [12]. The United States Air Force (USAF) has initiated the Automated Aerial Refueling (AAR) program, which will use a GPS solution or a GPS/sensor based solution to autonomously navigate the UAV to a boom receptacle refueling system [12].

These various rendezvous and docking research efforts require precision navigation. Most of these studies employ 3D imaging sensors, but there are many approaches that can provide rendezvous and docking accuracy. A GPS solution is capable of precision navigation when it is available, but the operational scenario

described in Chapter 2 takes place in an urban environment, and GPS performance is not sufficient in urban environments [13]. An Inertial Navigation System (INS) provides localized precise navigation, but it must be integrated with another navigation aid to calibrate the system to prevent drift that occurs over time [13]. A video based solution can provide precise navigation, but additional computer vision algorithms are required to obtain the same level of accuracy that the 3D imaging sensor inherently provides.

Limited research is available for the specific operational scenario defined in this effort. Although many precision navigation approaches of wide application were found in the literature, a 3D imaging sensor has been chosen as one of the more promising technologies to support precision rendezvous and docking for this research.

Employing 3D imaging sensors is still a developmental research area, due to the immaturity and cost of the 3D imaging sensors. Working correctly, 3D sensors offer various advantages over 2D imaging sensors since the target objects in rendezvous and docking are 3D [4]. The proposed methodology for this research uses a 3D imaging sensor to collect data and determine the relative position and orientation of the aerial vehicle relative to a fixed docking station. The three main pieces of technology in the integrated system are the 3D imaging sensor, the docking station geometry, and the algorithms to process the sensor data and determine orientation. These technologies will be described in further detail in the following section.

3.1 3D Imaging Sensor Technology Background

The 3D imaging sensor enables UAV rendezvous and docking in our approach. In this research, a LIght Detection And Ranging (LIDAR) sensor is used. LIDAR and Laser Detection and Ranging (LADAR) are evolving technology. An evolution of

different LIDAR and LADAR technologies was initially captured in [14] and the background is summarized for reference.

In 1990 a non-scanned laser radar was patented [15, 16]. This technology is an earlier version of LADAR, which uses a detector array in conjunction with a laser to measure the phase shift [15, 16]. A charge coupled device (CCD) measures the electron stream and outputs the range to the target [15, 16]. In 1999 a scannerless LADAR with high range resolution was patented [17]. This LADAR uses focal plane detector to detect the signal [17]. The signal feeds into a Fourier transform, which yields a range for a pixel [17]. The single pixel LADAR can be mechanically scanned or placed into an array to compose a 3D image [17]. A modulated laser is used in two additional patents from 1999 and 2000 [18, 19]. The phase shift is measured at a detector, which can cause ambiguous ranges at each multiple of the detected phase [18, 19].

An important differentiator between current available technologies is the method to create a 3D image. Some LIDAR and LADAR applications use a single light or laser and scan it across a 2D plane. Although 2D Scanning LIDAR hardware has matured for Airborne mapping systems and robotic platforms, flash LIDAR and LADAR provide additional benefits. Sandia Laboratories has advanced flash 3D imaging with their patented scannerless imaging solution [20]. This solution allows the scannerless sensor from one of the earlier patents, [15], to be implemented into an array [20]. Flash LIDAR is being studied in current research applications. For example, in 2009 NASA studied flash LIDAR for 3D terrain imaging [21]. Some advantages of flash 3D imaging sensors include time reduction to scan the image and elimination of the processing step to remove motion from the beginning of the scan to the end of the scan [21]. The 3D imaging

sensor used in this research fits into the integrated system as shown in the UAV notional architecture in **Figure 3.1-1**.

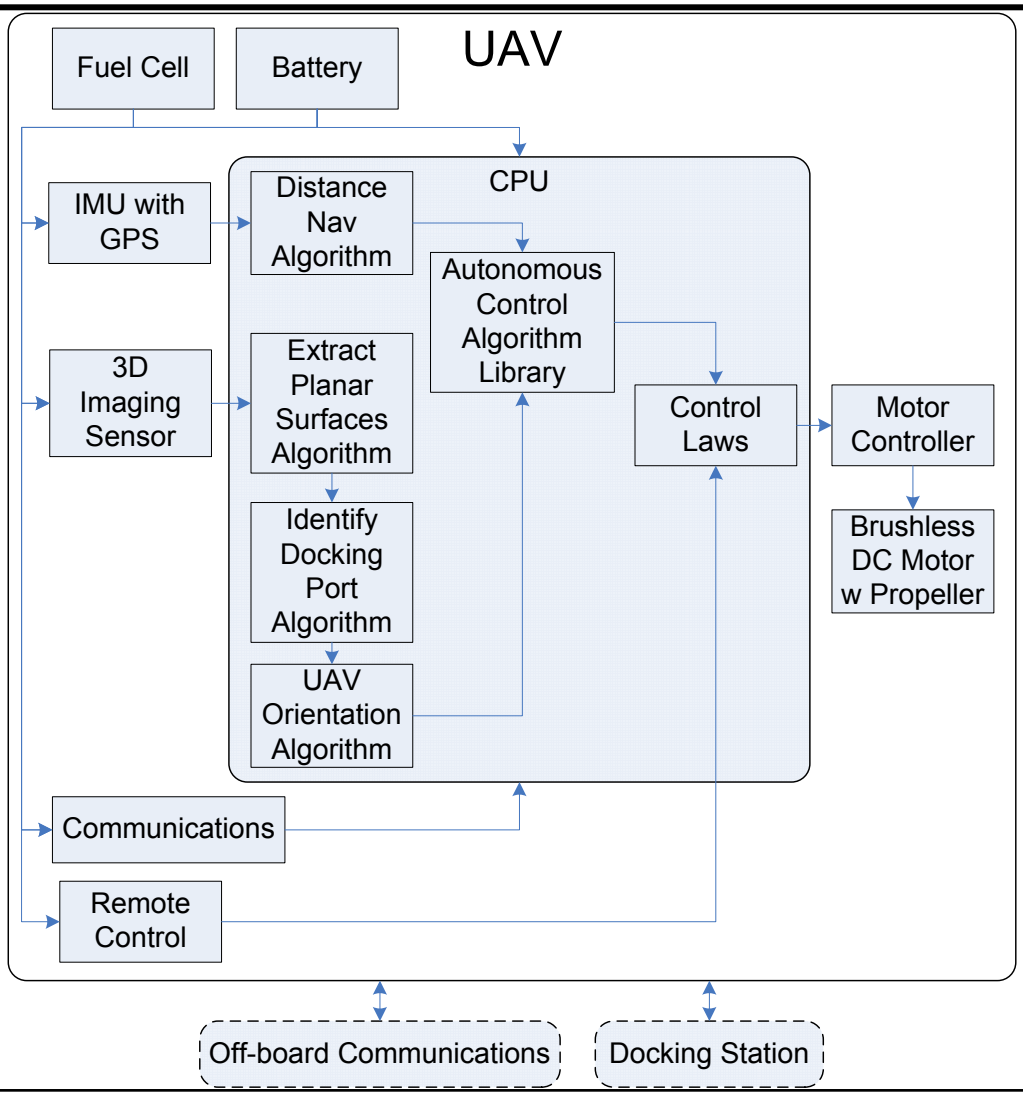


***Figure 3.1-1.* UAV Architecture.** The UAV utilizes 3D imaging sensor to enable navigation. 3D imaging sensor is integrated with other UAV components, such as an IMU with GPS and a CPU.

This research assumes the 3D imager is located on a UAV as shown conceptually in **Figure 3.1-2** and is used as the primary navigation system to guide the UAV to its

docking station. Future 3D imager requirements will include low cost, low altitude capability, and light output that is safe for the eyes.

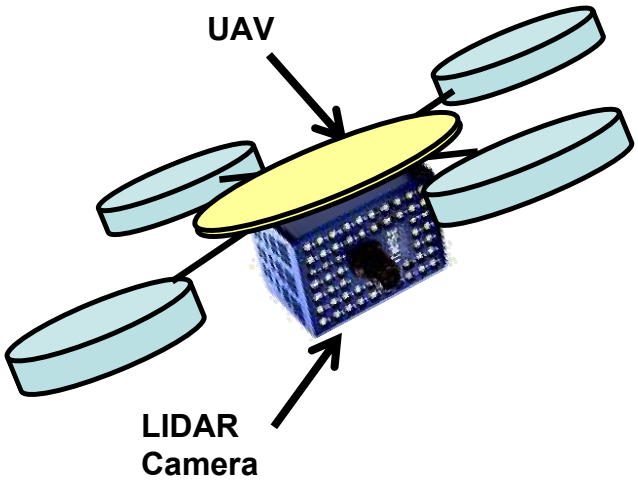


Figure 3.1-2. **UAV with LIDAR.** The LIDAR is on the belly of the UAV. The camera's coordinate frame is coincident with the UAV's body frame..

This research validates some of the proposed concepts using simulation and also data collected from a Swissranger 3000 (SR 3000) 3D imager, on loan from the Air Force Research Laboratories (AFRL) Sensors Directorate (RY). Raw sensor data is collected by the 3D imager and a pre-processing step is performed on-board the imager. The SR 3000 outputs a 3D point cloud and reflectance data for each frame. The 3D point cloud can be used to obtain x, y, z Cartesian coordinates expressed in the camera's body frame, and the reflectance data can be used to create an image plane similar to a 2D camera output. Camera, LIDAR and 3D imaging sensor are used interchangeably to describe the SR 3000 throughout this paper. The SR 3000's specifications are shown in **Figure 3.1-3**.

- 50 mm x 48 mm x 67 mm dimensions
- 176 x 144 pixel resolution
- Maximum 7 meter non-ambiguous range
- 3 internal circuit boards provide on-board raw data processing
    - o Sensorhead
    - o Image Processing Module
    - o Communication Interface Module
- Millimeter resolution at recommended distance

Figure 3.1-3. **SR 3000 Specifications.** This research selects the SR 3000 based on its availability. Specification information is from reference [22].

The SR 3000 is a phase measuring device. Amplitude modulated near infrared (NIR) light is emitted, which reflects off the surface of the illuminated scene. A phase delay is measured at the optical lens into the 3D sensor and provides a "depth map" of the surface or terrain [22]. The manufacturing process is optimized for demodulation and noise performance and results in a sensor that is "specifically optimized for range imaging" [22].

Illumination is provided by 55 LEDs, which are optimal based on their light weight, small size, low system costs, high volumes of illumination, and "eye safe" power and wavelength [22]. The LEDs provide continuous light emission at an 850 nanometer wavelength, with a total mean emitted power of 1 watt [22].

The flight data was taken from an aerial view approaching the docking station. The SR 3000 camera was held approximately 7 feet above the ground and was slowly lowered towards the docking station. This perspective allows the SR 3000 to collect flight or aerial data at a similar height and speed as if it were attached to a UAV platform as shown conceptually in **Figure 3.1-4**. Images from the actual setup where the data was taken are shown in **Figure 3.1-5**.

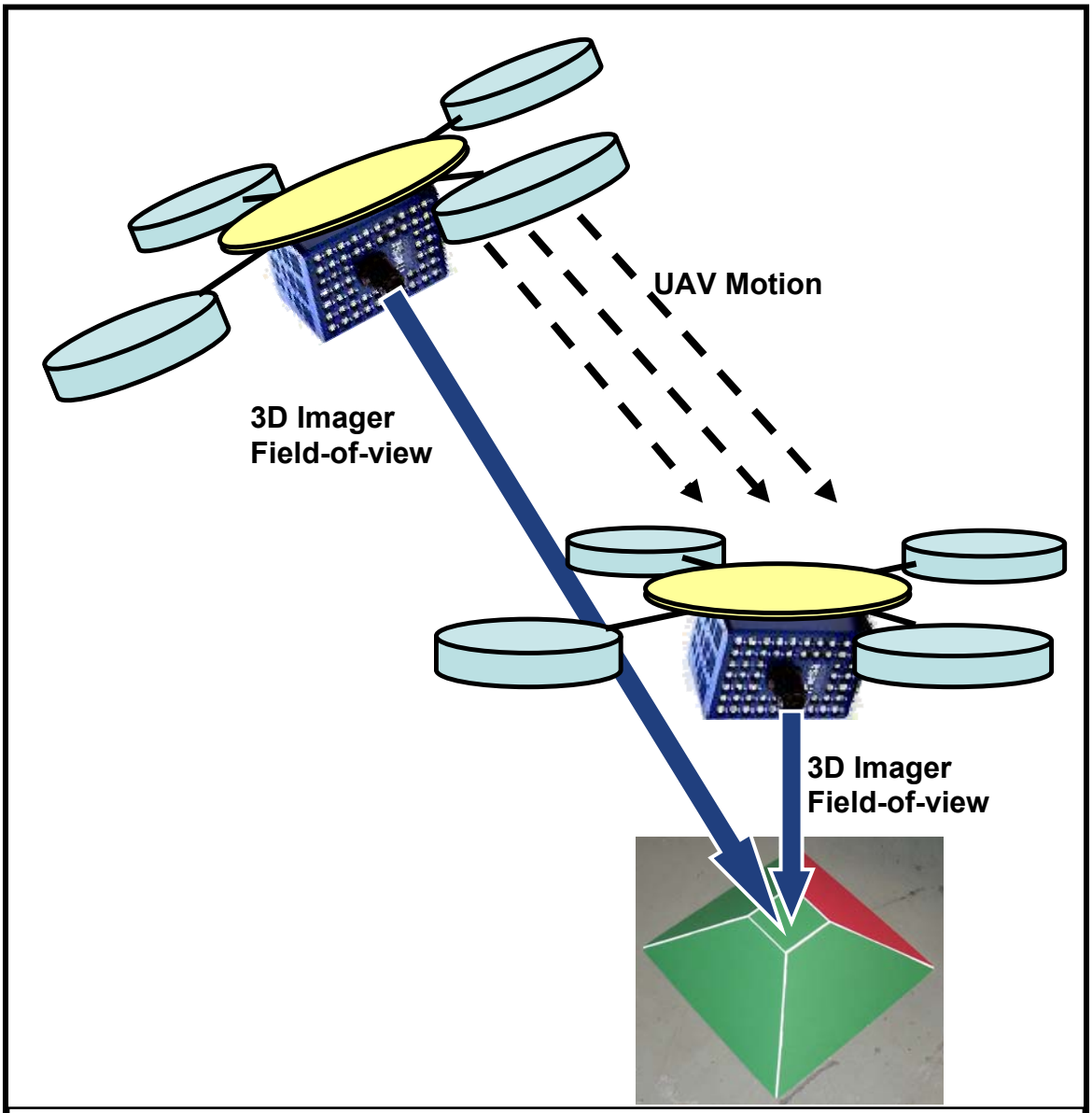


Figure 3.1-4. **SR 3000 and UAV platform.** Conceptually, the SR 3000 is shown attached to a UAV platform. The 3D imaging sensor is attached to the belly of the UAV facing down so the 3D imager's field-of-view is coincident with the UAV's z-axis.

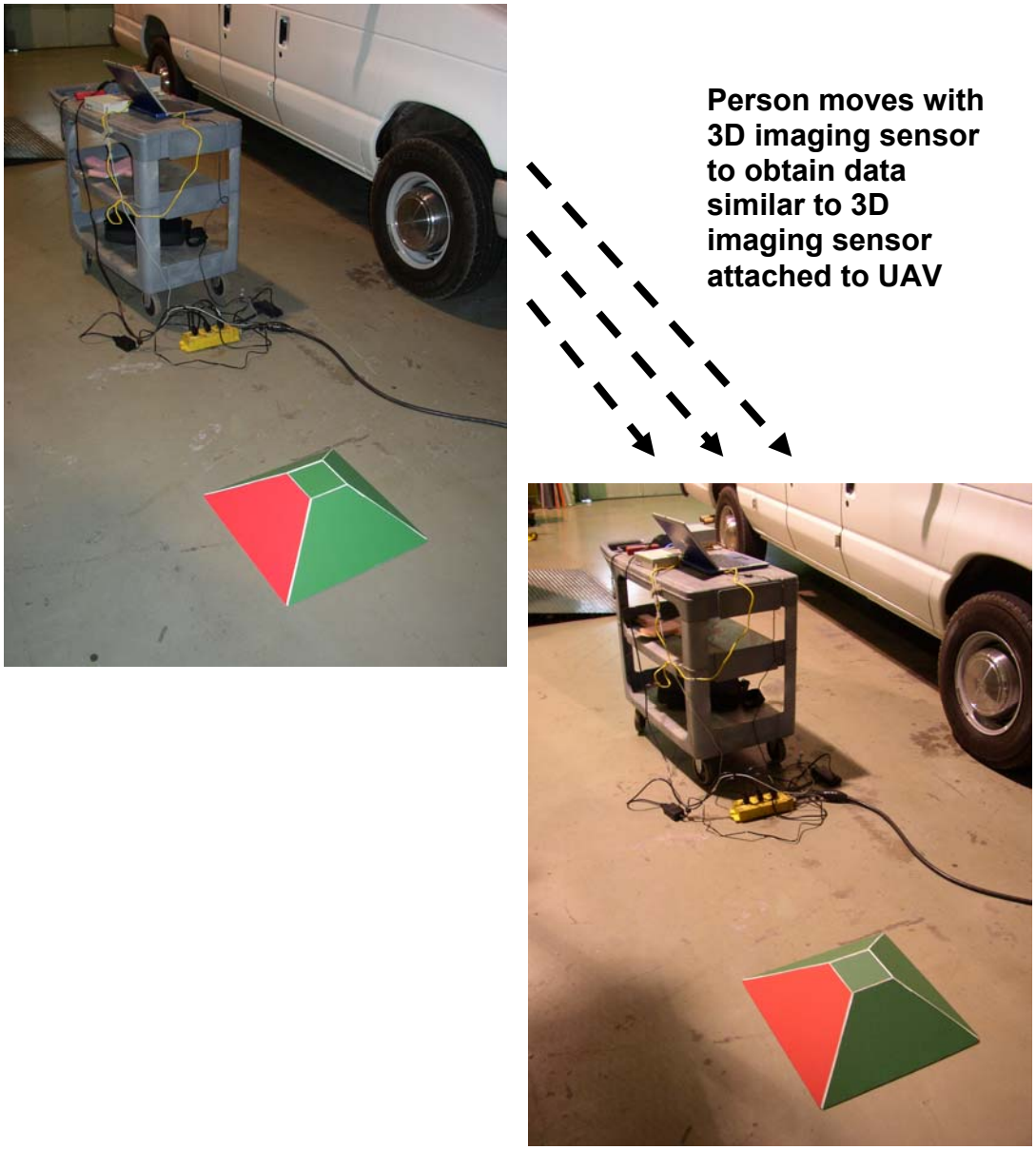


Figure 3.1-5. **Setup for Data Retrieval.** The actual setup when the data was taken using the SR3000 is shown. The 3D imaging sensor was held a few feet away from the dock to simulate the UAV approaching the docking station.

3.2 Docking Station Geometry Background

The docking station geometry is selected based on its simplicity for geometric calculations and observability within the navigation algorithms. As will be discussed later in Chapter 6, relative position estimation requires a total of three non-parallel vectors common between 3D image frames at two time epochs and relative orientation estimation requires at least two non-parallel vectors. This research uses the square based pyramid shown in **Figure 3.2-1**, and the advantages and disadvantages of the shape are discussed below.

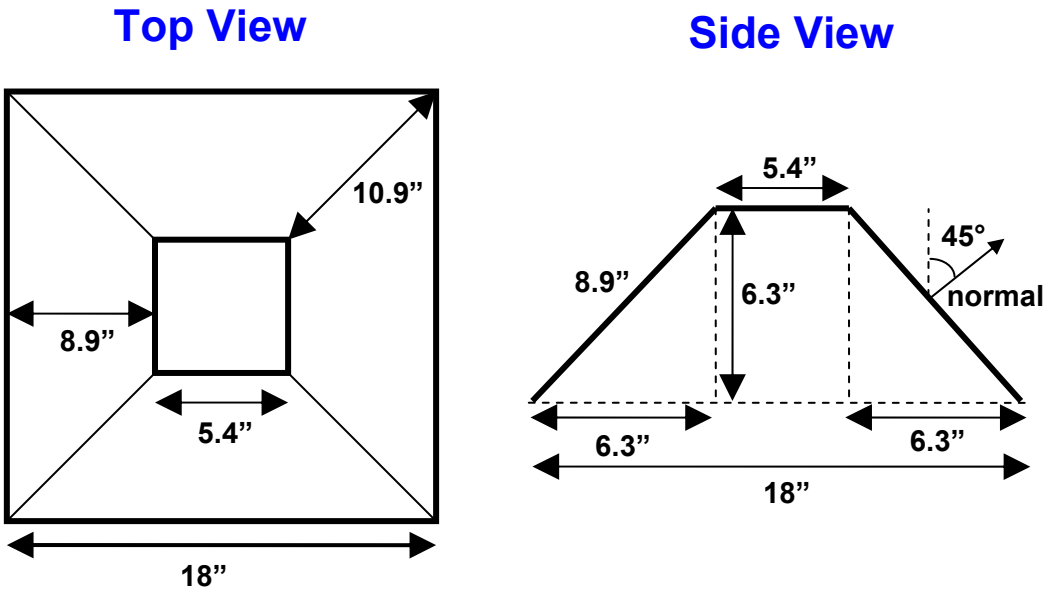


Figure 3.2-1. **Docking Station Dimensions.** A square based pyramid mockup is the fixed docking station geometry. The docking station dimensions are shown.

One advantage is that the docking station has five planar surfaces and only three are required for the algorithm. Even if two planes can not be recognized, enough planar surfaces are available to compute the required information. Another advantage is the

simple geometry. Simple trigonometric equations can be derived to describe characteristics of the docking station, such as the one derived in Chapter 5 that determines the proximal distance between planes.

Another advantage, as shown in **Figure 3.2-2**, is one of the pyramid sides is colored differently from the other sides to provide an absolute orientation of the docking port. The difference in color can be derived from the intensity image and a simple method for this is described in Chapter 6.

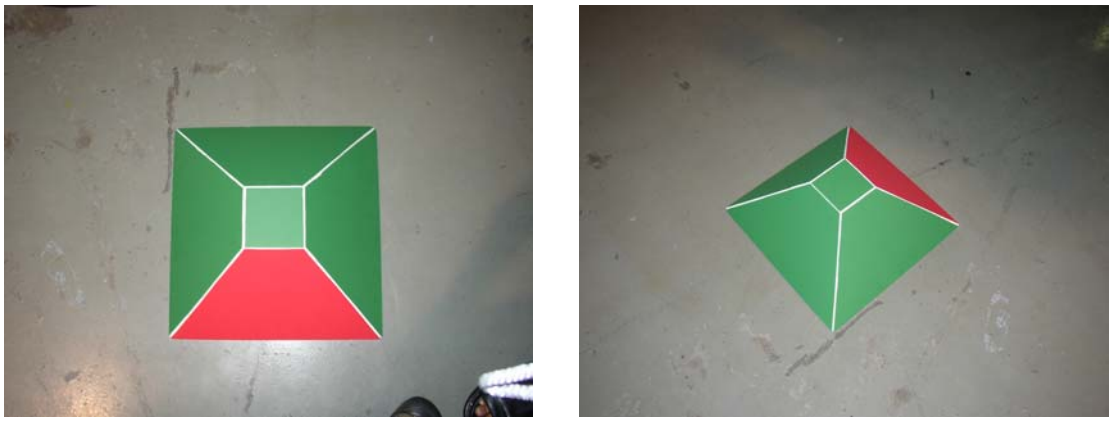


Figure 3.2-2. **Views of Actual Docking Station.** The actual square based pyramid docking station used in data collection for this research is shown in from a top and angle view. One side is a different color, which allows the algorithms in Chapter 6 to obtain an absolute orientation.

Some disadvantages were recognized once the research was initiated. The planar surface farthest from the camera is often out of view for the 3D imaging sensor. Also, the top planar surface is smaller relative to the other planar surfaces, and therefore is harder to recognize. The docking station worked for this research since three out of the five planes were able to be detected, although it is recommended to modify the shape in

future research. Potential shapes are a hexagon based docking station or a square based docking station with a larger top, and both will be further discussed in Chapter 7.

Chapter 4, 5, and 6 describe how the dock is used in the implementation. Chapter 7 suggests potential shapes to experiment with in future research. Dock and docking station are used interchangeably throughout the paper to describe the docking station geometry.

3.3 Data Processing Technology Background

The SR 3000 data is processed using MATLAB. "MATLAB is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation." [23] Image processing is a logical application in the MATLAB programming language due to the matrix suite of operations. This research takes advantage of the capability to store the 3-D point cloud as a matrix. MATLAB code can be converted to C once the code is loaded onto the UAV CPU in future research efforts.

The data processing MATLAB code developed in this thesis builds upon research performed by Dr. Jacob Campbell and Don Venable under the supervision of Dr. Maarten Uijt de Haag. Dr. Jacob Campbell's research developed SR 3000 drivers and converted the SR 3000 camera data into a 3D point cloud. Don Venable furthered Campbell's research and extracted planar features from the 3D point cloud. This research was detailed in the publication "Integration of an Inertial Measurement Unit and 3D Imaging Sensor for Urban and Indoor Navigation of Unmanned Vehicles". In this publication, the primary goal was to estimate the change in position & the change in attitude from one 3D point cloud frame to the next. [14]

The goal of the research at hand is to begin with the 3D point cloud extracted from the SR3000. The code is based on the planar extracts from Venable's code. A modification to the original approach to extract the surfaces of the docking station is proposed in Chapter 4. This modification improves the robustness of the algorithm with respect to the extraction of multiple planar surfaces. Chapter 5 analyzes the data and identifies the sides of the docking station in a consistent order. Finally, Chapter 6 computes position and orientation changes, which has been the focus of this research effort.

This research scope modifies the previous extract planes code to extract the docking station's planar surface features and creates new code to identify planar surfaces and compute relative orientation from these planar surfaces. This scope is broken into three subsystems:

Subsystem 1. Extract Planes from 3D Imager Data (Chapter 4)

Subsystem 2. Detect and Identify Docking Port in 3D Imager Data (Chapter 5)

Subsystem 3. UAV Orientation and Relative Position Algorithm (Chapter 6)

These subsystems are further discussed in the next three chapters.

CHAPTER 4: SUBSYSTEM 1 – PLANAR SURFACE EXTRACTION

The proposed method builds upon previous research in the area of integrated 3D imaging sensors and inertial measurement units for UAV navigation [14]. As discussed in Chapter 3, the existing system (prior to this effort) includes software drivers for the 3D imaging hardware, methods to process 3D point clouds into 3D Cartesian coordinates, and methods to extract planar surfaces from the 3D Cartesian coordinates. The software drivers and pre-processing steps of the 3D Cartesian coordinates is fully functional for this effort. The plane extraction is modified to extract the smaller planar features of the docking station.

4.1 Extract Planes Algorithm

The 3D image can be represented by *M*-by-*N* "pixels" defined by corresponding 6-tuple:

$$T_i = (x_i, y_i, z_i, \eta_i, r_i, c_i) = (\mathbf{p}_i, \eta_i, \mathbf{c}_i)$$

where x_i = x-coordinate of point cloud point corresponding to this pixel,

y_i = y-coordinate of point cloud point corresponding to this pixel,

z_i = z-coordinate of point cloud point corresponding to this pixel,

$\boldsymbol{p}_i$ = 3D point cloud point coordinate corresponding to this pixel,

η_i = intensity value corresponding to this pixel,

r_i = pixel row-coordinate in the image plane,

c_i = pixel column-coordinate in the image plane,

$\boldsymbol{c}_i$ = 2D pixel coordinate in the image plane

The existing planar surface extraction method uses a "split-and-expand" algorithm that splits the *M*-by-*N* image frame into *m*-by-*n* squares called sub-images (Step 1 in **Figure 4.1-1**) resulting in (*M*·*N*)/(*m*·*n*) sub-images. Planar surface properties, like the normal vector and centroid, are estimated for each sub-image using all its pixels. The quality of these planar fits is evaluated in a statistical manner using the mean sum-of-squared-residuals (MSSR) (Step 2 in **Figure 4.1-1**) [14]. Defining the point cloud points associated with the pixels in sub-image 'w' as the set of (*m*·*n*) points S_w, then the centroid and normal are estimated as follows:

$$\hat{\mathbf{p}}_{0,w} = \sum_{p_i \in S_w} \mathbf{p}_i \tag{4.1}$$

$$\hat{\mathbf{n}}_w = arg\,\min_{\mathbf{n}} \{SSE_w\} = arg\,\min_{\mathbf{n}} \left\{ \sum_{\mathbf{p}_i \in S_w} \|d_i\|^2 \right\} \tag{4.2}$$

where d_i is the residual distance from the ith point in S_w to the planar surface defined by $\mathbf{p}_{0,w}$ and $\mathbf{n}_w$ is given by:

$$d_i = \left(\mathbf{p}_i - \hat{\mathbf{p}}_{0,w}\right) \cdot \hat{\mathbf{n}}_w \tag{4.3}$$

Note that S_w includes points that are part of the actual candidate planar surface, but also points that are not. When the number of non-planar surface points during step 1 is much smaller (preferable close to 0) than the number of planar points, the SSE_w will approach the nominal value for planar surfaces observed by the 3D imaging sensor.

The expand part of the algorithm starts with the best sub-image (Step 3 in **Figure 4-1**), the *m*-by-*n* sub-image with the lowest residual value (SSE_w). This sub-image is marked as the candidate planar surface (denoted in the code as *best_win).*

$$best\_win = arg\,\min_{w} \left(SSE_w\right) \tag{4.4}$$

Next, the algorithm uses an iterative histogramming method to expand the planar surface found from the "best" available sub-image. Detailed results for that method are given in [24]. Given that the set S_I is defined as the set of all the points in the image that have either not yet been assigned to any planar surface, or have been discarded for other reasons, the method identifies which points lie within a certain distance from the planar surface, or:

$$S_V(k) = \{\mathbf{p}_i \in S_I \mid \mu_k - 2\sigma_k < d_i < \mu_k + 2\sigma_k\} \tag{4.5}$$

where k is the k^{th} iteration. After each iteration, the mean (μ_k) and standard deviation (σ_k) are re-calculated using all available points and standard histogramming methods such as Otsu's and Kittler's methods [25, 26]. Since these methods assume the presence of only two dominant modes (object and background) in the image, an alternative windowing method could be used. Essentially, one can look at this method as computing a histogram of the residual distances of all eligible points, and extracting the mode around '0'-distance. The "stopping" criterion is based on monitoring the change in the standard deviation of the mode.

After identification of all image points S_V that would make good candidates for the planar surface, all pixels in each neighboring sub-images are inspected. If any pixel in the neighboring sub-image is part of the candidate planar surface, the sub-image is included as part of the candidate planar surface.

This algorithm continues to iterate and expand the candidate planar surface until all neighbors of the expanded plane are evaluated to determine if any of their pixels are within a threshold value of d_i (Step 4 in **Figure 4.1-1**) [14]. Each sub-image is marked as used when it is evaluated, whether or not it has been included as part of the candidate

planar surface. If the sub-image is part of the candidate planar surface it is included as part of the plane.

Once all neighbors of the expanded plane are evaluated the code iterates and finds the next best window and restarts the previous steps. The subsequent candidate planar surface is the next best planar fit sub-image that is unused (Step 3 in **Figure 4.1-1**). Candidate planes continue to be identified and iteratively expanded until a percentage of the current frame's sub-images are marked as used or no candidate planar surfaces can be found that satisfy a minimum MSSR (Step 3 and 4 in **Figure 4.1-1**) [14]. The required percent of the frame's sub-images marked as used can be modified and is currently 90%.

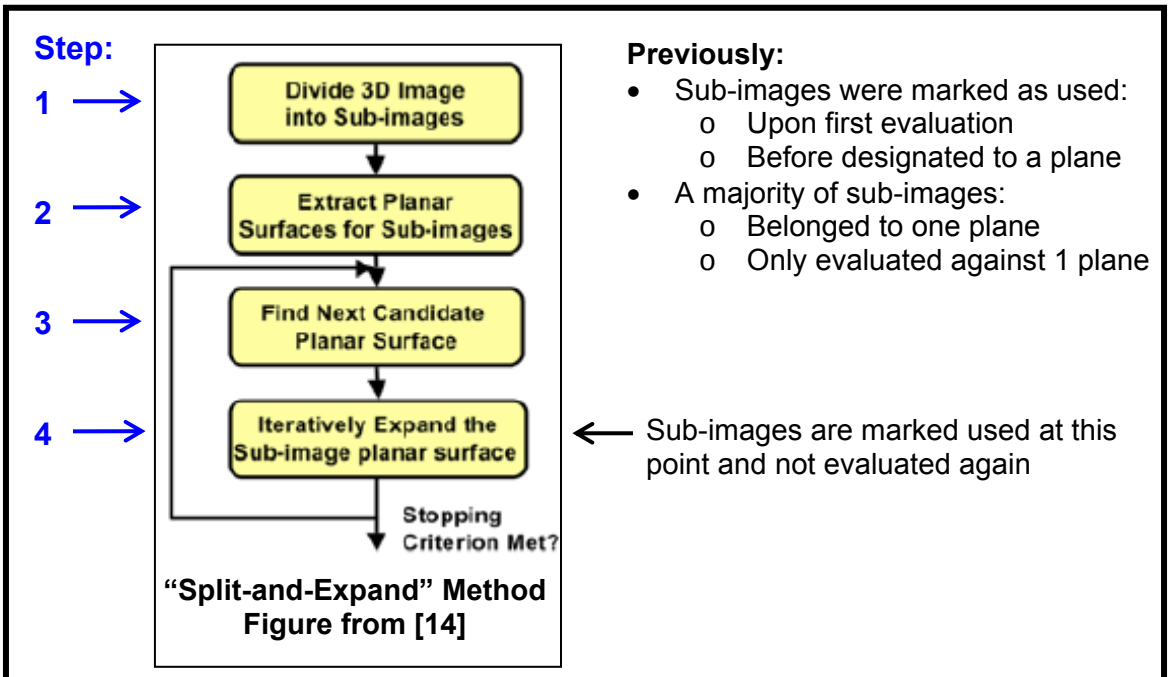


***Figure 4.1-1.* "Split-and-Expand" Method.** Revised method allows a sub-image to be evaluated against multiple planes until it finds a fit. This allows smaller planes to be detected.

4.2 Extract Planes Modification Summary

The extract_planes algorithm was efficient when detecting larger planar surfaces that are composed of multiple sub-images, but not as effective in detecting smaller planar surfaces in the scene. The original algorithm allowed a few sub-images along the perimeter of the planes to be lost, which did not impact the overall result of planar detection when detecting larger planes in previous research. When each planar surface consists of only a few sub-images losing a few sub-images could result in misidentifying a whole plane. Three approaches were considered to improve the performance of the algorithm for detecting smaller planar surfaces a) adjustment of the sub-image window size parameters *m* and *n*, b) a change in the way the sub-images are marked when pixels in the sub-image have been used, c) or a change of the pixel selection criteria.

Conceptually, making the *m*-by-*n* sub-images smaller would help detect smaller planes. When the sub-images were reduced in approach (a), the statistical properties used to define the plane were skewed because there were too few points within the sub-image to make them statistically significant. **Figure 4.2-1** shows these results. The left side shows the misidentified planes from reducing sub-image size in approach (a) and the right side shows the correctly identified planes using approaches (b) and (c).

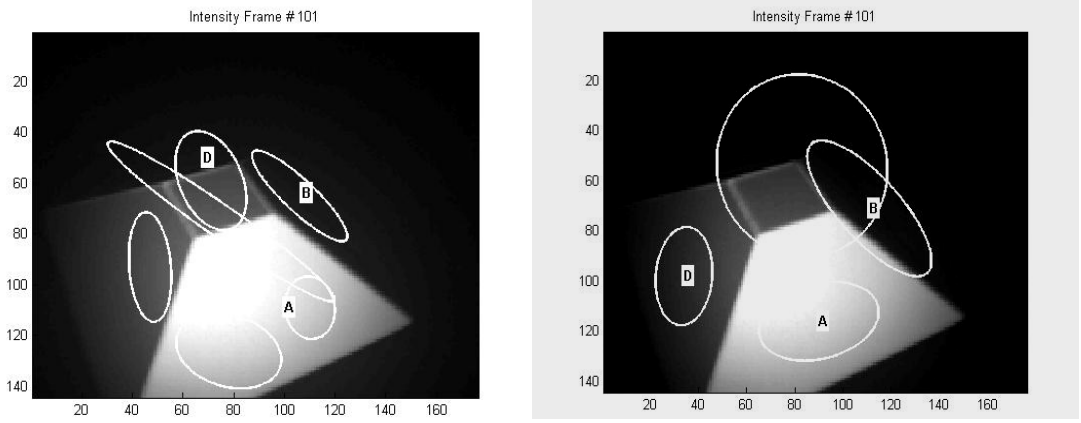


***Figure 4.2-1*. Reducing Sub-Image Size.** The left side shows the results from approach (a), the attempt to reduce the sub-images in the code. This caused false plane identification. The right side shows the results from approach (b) and (c), plane extraction and identification after code revision.

Since altering the dimensions of the sub-image did not yield the desired results, the extract planes code has been modified. The first modification, approach (b), allows sub-images to be evaluated against multiple planes until the sub-image is associated with a candidate planar surface or the algorithm for the frame is complete. The second modification, approach (c), requires a percent of pixels of the sub-image to be part of the candidate planar surface before adding the sub-image to the candidate planar surface.

Approach (b) was initiated because the code previously iteratively expanded the candidate planar surface and marked all neighboring planes as used, whether they had been associated with the candidate plane or not. Some sub-images were marked as used but not associated with the candidate plane. These sub-images were not evaluated against any additional planes, even though a majority of sub-images generally belong to at least one plane. Marking all neighbors as used caused the perimeter of the planes that were not associated with the candidate plane to be lost and never included with any plane. This impacts the results for smaller planar surfaces because a smaller plane consists of only a few sub-images that could be lost along the perimeter of other planes.

The code is now modified to only mark a sub-image as used if it is included as part of the candidate planar surface. If a certain percent of the pixels "belongs" to S_V, that sub-image is marked as "used" and no longer being considered as a starting point for the next planar surface. If it is not part of the candidate planar surface it can be evaluated against future candidate planar surfaces. The sub-image remains unused until it is included as part of a candidate planar surface, or until the algorithm iterates to the next frame because the required percent of the frame's sub-images have been included in a plane. This modification is shown in line (4.6d) and (4.6f) below.

Approach (c) was initiated because the code previously categorized the sub-image as part of the candidate plane if any portion of the sub-image held the statistical characteristics of the candidate plane. This was not effective for smaller planar surfaces that are easily absorbed into other planes since their plane does not span more than a few sub-images. The top portion of **Figure 4.2-2** shows a conceptual example of this problem.

The code has been updated to require a certain percent of pixels in the neighboring sub-image to be within a threshold value of d_i before the sub-image is added to the candidate planar surface. The percent of pixels can be modified in the code, and the percent is currently set to 12.5%.

This modification allows the plane extraction algorithm to identify three sides more consistently because it allows smaller planes that were previously absorbed into other planar surfaces to have the chance to be identified as their own planar surface. The modification is shown in line (4.6-b), (4.6-c), and (4.6-e) of the algorithm below and is illustrated in an example in the bottom portion of **Figure 4.2-2**.

loop iterates through each sub-image in temp_best_win (4.6-a)

*if (length(curr_in_plane$_i$) > (sub_image total length*12.5%))* (4.6-b)

include sub-image in best_win (4.6-c)

mark plane as used (4.6-d)

else

do not include sub-image in best_win (4.6-e)

do not mark plane as used, it can be evaluated again (4.6-f)

end

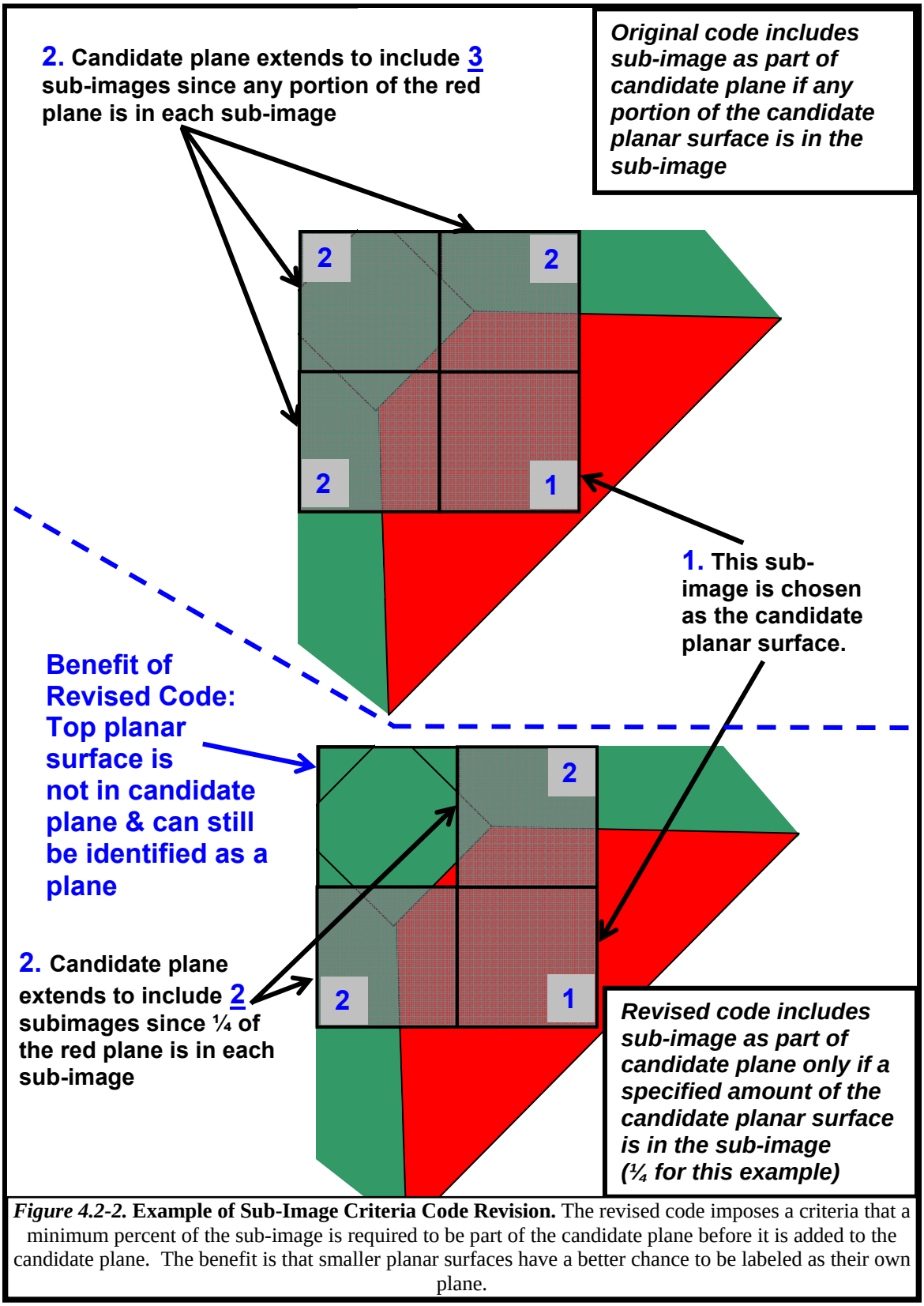


Figure 4.2-2. **Example of Sub-Image Criteria Code Revision.** The revised code imposes a criteria that a minimum percent of the sub-image is required to be part of the candidate plane before it is added to the candidate plane. The benefit is that smaller planar surfaces have a better chance to be labeled as their own plane.

The exact improvement of these algorithm updates is not quantified, although **Figure 4.2-3** shows an example of the change in planes identified. Before modifications, the algorithm identified the planes as shown in the left hand picture. After modifications, the algorithm detected three sides of the docking station more consistently as shown in the right hand picture.

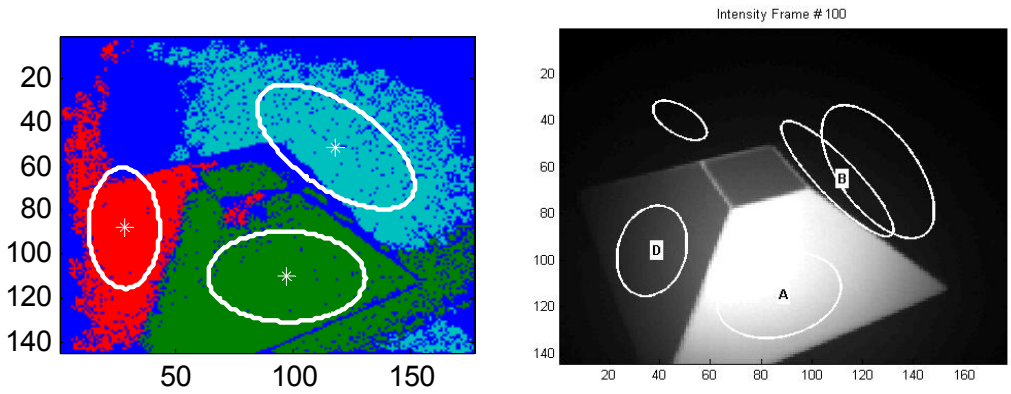


Figure 4.2-3. **Improvement from Code Revisions.** The left side shows the plane extraction for the docking station data set before code revision. The right side shows plane extraction and identification after code revision.

CHAPTER 5: SUBSYSTEM 2 – IDENTIFICATION OF THE DOCKING PORT

Subsystem two is initiated upon completion of subsystem one. Subsystem one extracts planes from 3D images of the docking port. Subsystem two identifies the extracted planar surfaces in a consistent order in every frame.

The planes must be consistently identified because this input is required for subsystem three where the change in observed orientation of the docking station is used to estimate the user orientation. Each planar surface in the first coordinate system must be compared to the corresponding planar surface in the second coordinate system. In order to know which planes are associated, the order of the planar surfaces must be determined consistently every time epoch. **Figure 5-1** shows the pre-defined order of the docking station planar surfaces and their corresponding letter assignment (A, B. C, D, E) as defined for this research.

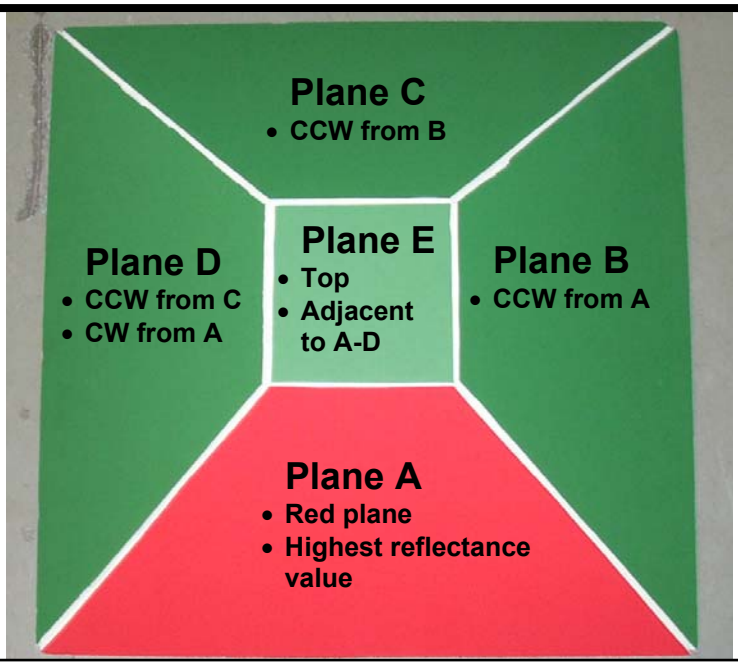


Figure 5-1. **Pre-defined Order of Docking Station Planes.** Plane A is easy to identify by its high reflectance value. Planes B through D are CCW from A. Plane E is the top planar surface.

5.1 Implementation Overview

The subsystem described in this chapter identifies which input planar surfaces are associates with the pre-defined docking station reference planes. The output of this subsystem provides the necessary data for subsystem three to determine the 3D imaging sensor's position with respect to the docking station. Implementation of this algorithm is coded in MATLAB Version 7.3.0 (R2006b). The identification and association tasks are implemented in *identify_planes*, which currently identifies three specific planes since only three planes are identified consistently in subsystem one with the current data set. This implementation can easily be modified to identify all planes once more than three planes are available in the data set. It is called from the main code "processframes.m" and the function call is as follows:

```
plane_id = identify_planes(plane_id, jj, plane_store_post, I);
```

The *identify_planes* function is called for every 3D imaging frame. The variable *jj* defines the current camera frame. The *plane_id* output variable is a structure that stores the three associated reference planes for each frame and all of their relevant information (planar surface centroid and normal vector, the image plane centroid, etc.). In each frame the planes are stored in the structure in the position associated with their letter (A in 1, B in 2, etc). This output data is stored across all frames in the data set and is subsequently passed into the third subsystem as an input variable.

The input variable *plane_store_post* contains the output data from subsystem one, which includes the planes extracted in subsystem one and their relevant information. The docking station planes that are identified and associated in *identify_planes* are a subset of the planes received from the input variable *plane_store_post*.

5.2 Reflectance Algorithm to Identify Plane A

The input variable *I* is a regular 2D monochrome image containing the reflectance data (η_i) for each frame. This data is used to determine which planar surface corresponds to side A. Side A is a different color than the other sides and the background behind the docking station.

Identifying the planes in the specified order requires an algorithm that can consistently determine plane A since it is the only plane with a differentiating feature. Plane A's red color has a higher reflectance value (η_i) than the green adjacent planes and the grey floor. To correctly distinguish planar surface A from the other planar surfaces and the background, the mean reflectance value is computed for all input planes:

$$\overline{\eta}_k = \sum_{i \in S_k} \eta_i$$

Next, the planar surface with the highest average reflectance value is chosen as the candidate planar surface A. The pseudocode to detect the surface with the highest reflectance value is as follows:

$store\ \eta_1\ as\ temp\eta_{\max}$
$for\ i = 1:K$
    $if\ (\overline{\eta}_k \geq temp\eta_{\max})$
        $temp\eta_{\max} = \overline{\eta}_k$
    end
end
$V_A = temp\eta_{\max}$

where *K* is the number of extracted planar surfaces and $\overline{\eta}_k$ is the average reflectance value for the k^{th} surface. Note that it may be possible to have other more "reflecting" planar objects within the field-of-view of the camera; however, these candidates may be discarded since they may not necessarily meet the geometric constraints of the docking

station that will be used in the remainder of this chapter to identify the other planar surfaces (B, C, etc.).

Plane A can also be used by the UAV to determine the way it is approaching the docking station. The data used in this research approaches towards plane A, so plane A is always in the 3D imaging sensor's field-of-view. Plane A can still be identified when it is not directly in the 3D imaging sensor's field-of-view, although keeping it in the field-of-view mitigates the probability of false detection.

Once this code is integrated with the UAV's controller implementation, it is recommended that the UAV approach the docking station facing plane A. When the UAV is approaching planar surface A the 3D imaging sensor will view the docking station from the vantage point shown in **Figure 5-1** above. This is easily implemented in code by comparing the image plane coordinates of the planar surfaces.

When approaching the docking station facing planar surface A, plane B will always be located to the right of plane A, and plane D will always be located to the left of plane A in the 2D image plane. This ad hoc method can be implemented in the code by a simple evaluation comparison of the image plane centroid coordinates of planes A (r_A,c_A), B (r_B,c_B) and D (r_D,c_D) as follows:

$$V_B = \{V_k \mid r_k > r_A \cap c_k > c_A\}$$
$$V_D = \{V_k \mid r_k < r_A \cap c_k > c_A\}$$

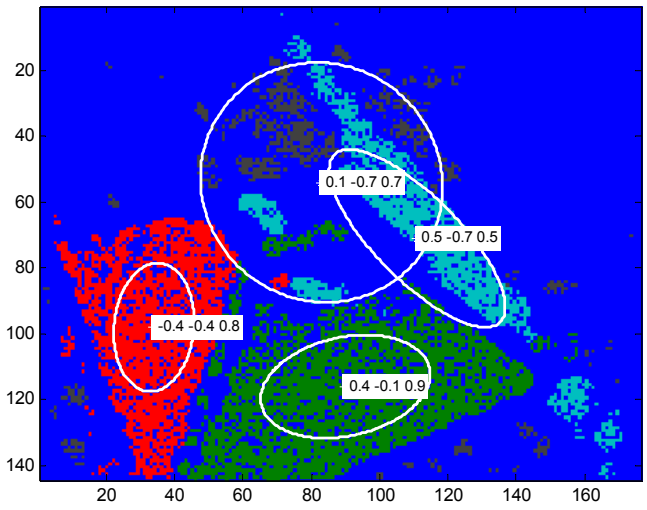
The UAV controller implementation can command the UAV to circle the dock until the equality equations are true. Once the UAV is approaching towards plane A the controller implementation can use the Euler angles that are output in subsystem three to fine tune the approach.

Approaching towards a specified side of the docking station has additional advantages in the operational scenario. Landing on a dock in a specific orientation can allow the UAV to self-initiate tasks that require the UAV contacts to be aligned with the docking station contacts, such as recharging batteries or uploading data.

5.3 Overview of Methodology to Detect Planes B, C, D, E

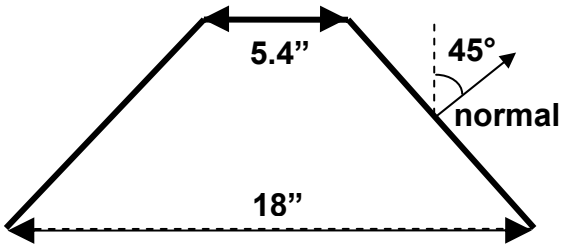
The initial concept to detect planes B, C, D, E primarily used the plane's normal vectors so these vectors were analyzed. Based on this analysis, the plane's normal vectors were examined to conceptually detect a pattern that could be translated into an algorithm. It was determined that the angle between two planar surface normals was approximately 0.8 radians. This value fluctuates slightly so it cannot be used as the primary input parameter, but section 5.4 discusses how it is useful to detect planes when paired with other parameters. The normal vectors are also useful for the algorithm in subsystem three, where the change in normal vectors based on the vantage point helps estimate orientation. **Figure 5-2** shows the plane's normal vectors that were analyzed.

This analysis also discovered that the angles between the vectors are slightly imprecise compared to the docking station geometry that was originally constructed. The camera's vantage point and the quality of subsystem one outputs accounts for a portion of this inaccuracy, but part of it is because the docking station slightly "settled". The docking station "settling" is not quantified because the settling error does not directly impact this research and is further described in Chapter 6.



Subset of Normal Vectors Derived in Subsystem 1

	X	Y	Z
A	-0.4	-0.4	0.8
B	0.4	-0.1	0.9
D	0.5	-0.7	0.5



Original Estimated Normal Vectors

	X	Y	Z
A	-0.7	0	0.7
B	0	-0.7	0.7
D	0	0.7	0.7

Figure 5-2. **Docking Station Normal Vectors.** The top illustrations show a sample of normal vectors from the code output when the camera is not directly above the dock. When the camera is directly above the docking station the vectors are at approximately a 45° angle as shown in the bottom illustration.

The second concept, which is implemented in this research, narrows down the candidate planes using the approximate 3D distance between the plane centroids. Next, the candidate planes are further narrowed down using the normal vectors to determine the angle between planes. Finally, the candidate planes are evaluated using the 2D image plane and basic computer vision concepts. Specifically, convex hulls compute the counter-clockwise order of a set of points and identify the docking station planes. The convex hull equations will be discussed in more detail in Section 5.5, and a brief overview is provided below.

The convex hull algorithm identifies the additional sides after plane A is identified. The planes in this algorithm are ordered relative to plane A because plane A is easy to differentiate based on its reflectance properties. The planes are ordered in counter-clockwise order using Andrew's variant of the Graham Scan convex hull algorithm [27]. Once they are in counter-clockwise order planes B and D are identified based on their relative position to A. B is identified as the plane adjacent to and counter-clockwise from A, and D is identified as the plane adjacent to and clockwise from A. This implementation currently is set up to identify two sides, planes B and D, and can be easily modified in the future to also identify planes C and E.

5.4 Algorithm to Detect Candidate Planes for B, C, D, E

The candidate planes are identified in a series of two steps. The first step after identifying plane A is to calculate the other planes' approximate 3D distance from A, in order to remove planar surfaces that are too far away from and thus not suitable candidates. The fixed docking station geometric properties allow a standard equation to define the criteria for the set of candidate planes proximal to A. The second step evaluates the angles between the normal vectors to further narrow down the set of candidate planes.

The first step currently uses the centroids as the reference point to compare the distances between planes. Since the centroids can shift when part of the plane is out of the camera's field of view, the centroids are not the most optimal reference point. In future research, when the top plane is able to be detected in subsystem one, the intersection of three planes could be used in place of the centroids. Since the 3D centroid of A, B and D should have equivalent z coordinates, the x and y distance is calculated.

Figure 5-3 illustrates the geometry that is used to calculate the approximate distance between 3D plane centroids.

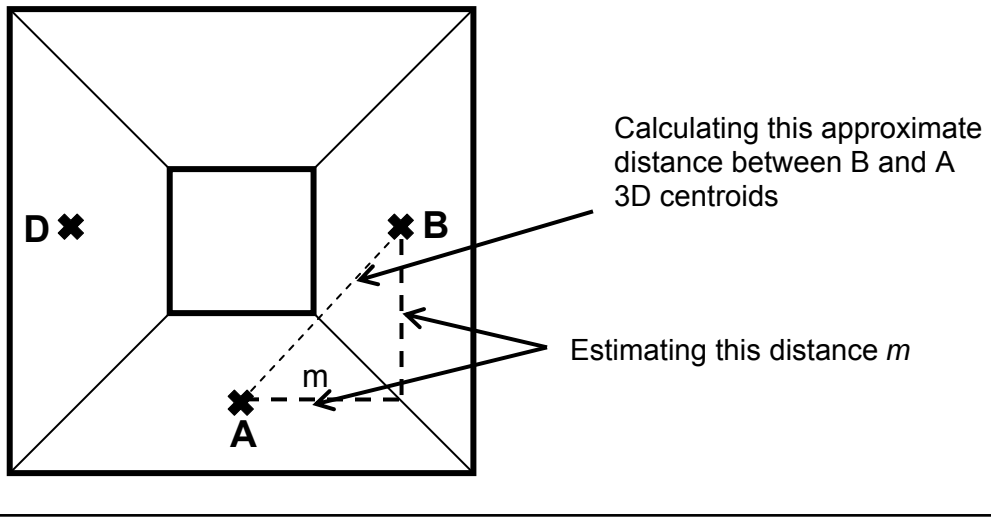


Figure 5-3. **Approximate Distance between Plane Centroids.** To identify candidate docking station planes neighboring A, all planes distances relative to A are compared to the approximate distance between plane centroids.

In order to calculate the approximate value for m, the trapezoid geometry is evaluated as shown in **Figure 5-4**. The middle of the trapezoid is calculated using the equation M = ½ (a + b).

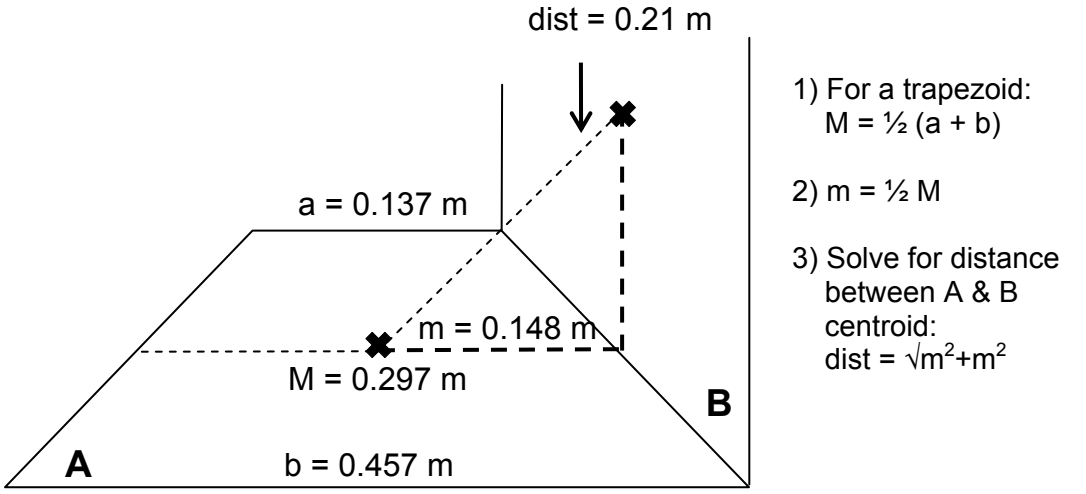


Figure 5-4. **Criteria for Candidate Planes Proximal to A.** The trapezoid geometry is evaluated to determine the criteria for candidate planes. The output from the geometric calculation is compared to the actual data outputs to determine the approximate location of the centroid on the trapezoid.

This value of 0.21 was compared to the actual values between 3D centroids. The average actual value of the centroid was approximately 0.18 m. This value is smaller than estimated because part of the plane is out of the camera's field of view in many of the frames. In order to maximize the number of eligible data points to feed into subsystem three, the criteria was modified to approximate the centroid value at 0.18 m. **Figure 5-5** illustrates potential modified criteria. The current data set seems to be closest to the $M_{5/8}$ centroid and distance. In the future, once a more stable reference point is available the initial geometric value can be used and the extra analysis and modification will not be required.

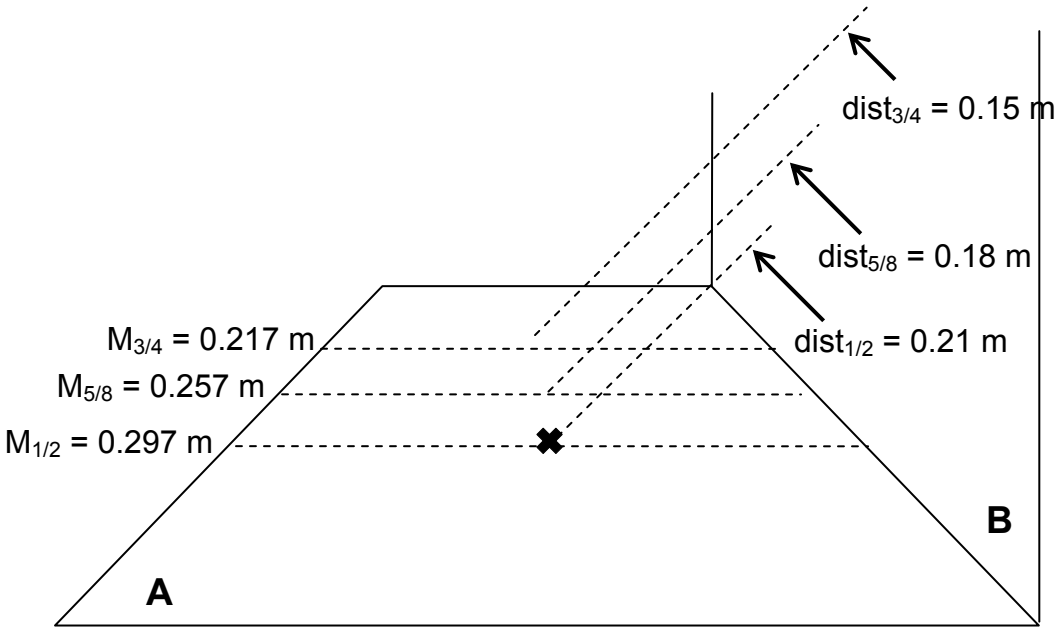


Figure 5-5. **Modified Criteria for Candidate Planes Proximal to A.** The trapezoid geometry is evaluated to determine the criteria for candidate planes. The output from the geometric calculation is compared to the actual data outputs to determine the approximate location of the centroid on the trapezoid.

Next, the second step further narrows down the set of candidate planes. The angle between plane A and each candidate normal from the first step is evaluated using the following equation:

$$\alpha = \cos^{-1}\left(\frac{\left(\mathbf{n}_a \cdot \mathbf{n}_i\right)}{\|\mathbf{n}_a\|\|\mathbf{n}_i\|}\right)$$

Since the docking station settled as previously discussed and the exact angle between planes is not known, the data was analyzed to determine the average angle between the planes. The angle between planes was approximately 0.8 radians, so the α that are closest to 0.8 are selected as the narrowed set of candidate planar surfaces.

5.5 Convex Hull Algorithm to Identify Planes B, C, D, E

After the set of candidate planes is reduced, the convex hull algorithm is initiated. There are a variety of convex hull algorithms available. Some convex hull algorithms are capable of processing 3D images, such as the Divide-and-Conquer convex hull algorithm [27]. Since this portion of the research only requires processing of a 2D image plane, a simpler algorithm is chosen and implemented, namely Andrew's variant of the Graham Scan algorithm [27].

Andrew's variant of the Graham Scan convex hull algorithm first divides the number of points, *S*, into a candidate upper (S_{CU}) and lower (S_{CL}) hull. The points are ordered clockwise, proceeding to the right in the top half and to the left in the bottom half. A line across the image from the far most left point to the far most right point splits the image into the top and bottom hulls. This is easily identified in the code by assigning the point with the largest 'c' value as the far most right point and the point with the smallest 'c' value as the far most left point [27, 28].

Next, each candidate point (P_C) is evaluated and distributed into a candidate upper or lower hull based on its relative position to the line between the far most left (P_L) and right (P_R) points. This is determined by the following equation:

$$P_c sign = [(P_L x - P_R x) * (P_C y - P_R y)] - [(P_C x - P_R x) * (P_L y - P_R y)]$$

$$S_{CU} = \{S \mid P_C sign <= 0\}$$
$$S_{CL} = \{S \mid P_C sign > 0\}$$

If the candidate point is above the line ($P_C sign <= 0$) it is added to the candidate upper convex hull. If the candidate point is below the line ($P_C sign > 0$) it is added to the candidate lower convex hull.

Once the candidate upper and lower hulls are defined, each point is evaluated to determine if they are actually part of the convex hull. If the points are part of the convex hull they are added to a queue in the order they are identified, which is clockwise for the upper hull and counterclockwise for the lower hull. A function orders the points by iterating through each point in the hull, creating a line between a previous point (P_{k-2}) and the current point (P_k), and evaluating if the previous point (P_{k-1}) is above or below the line formed by P_{k-2} and P_k. This is determined by the following equation

$$P_{k-1}sign = [(P_{k-2}x - P_kx) * (P_{k-1}y - P_ky)] - [(P_{k-1}x - P_kx) * (P_{k-2}y - P_ky)]$$

$$S_U = \{S_{CU} \mid P_{k-1}sign <= 0\}$$
$$S_L = \{S_{CL} \mid P_{k-1}sign > 0\}$$

If the candidate upper hull point P_{k-1} is to the left of the line formed by P_{k-1} and P_k it is added to the upper hull, if not it is removed. If candidate lower hull point P_{k-1} is to the right of the line formed by P_{k-1} and P_k it is added to the lower hull, if not it is removed. **Figure 5-6** shows an example of the ordering portion of the algorithm [27, 28].

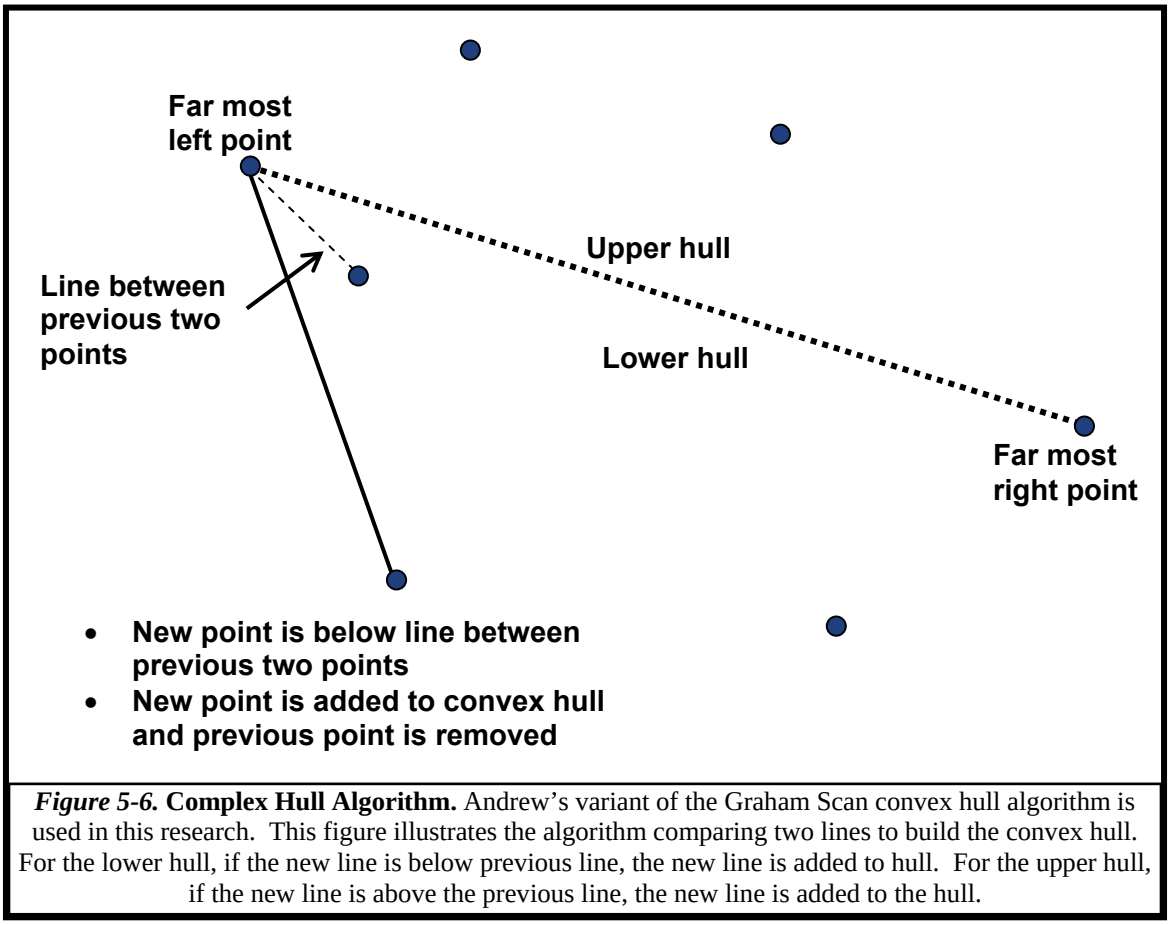


Figure 5-6. **Complex Hull Algorithm.** Andrew's variant of the Graham Scan convex hull algorithm is used in this research. This figure illustrates the algorithm comparing two lines to build the convex hull. For the lower hull, if the new line is below previous line, the new line is added to hull. For the upper hull, if the new line is above the previous line, the new line is added to the hull.

The results of this implementation are mainly limited based on the quality of the data input from subsystem one, which can be improved based on recommendations in Chapter 7. The quality of subsystem two outputs are hard to quantify because the docking station sides are not consistently extracted from subsystem one. Planes are extracted correctly throughout a larger number of data frames, although they are only extracted consecutively in a limited number of data frames.

There is a set of 12 consecutive data frames where the docking station sides are extracted correctly from subsystem one, and the subsystem two algorithm correctly identifies the sides in 11 out of those 12 frames. Although this data subset is minimal, it

provides enough of a sample to test subsystem three with real data. This is sufficient to enable the research goal of providing a data subset to subsystem three using only a 3D imaging sensor. Inertial sensors could enable this task over a larger set of consecutive frames, although the goal of the current research is to accomplish the orientation only using a 3D imaging sensor.

CHAPTER 6: SUBSYSTEM 3 – UAV ORIENTATION ALGORITHM

Subsystem three determines the UAV orientation with respect to the docking station by leveraging the data from the previous two subsystems. The UAV precision docking basic operational scenario is illustrated in **Figure 6-1**. Subsystem three focuses on the 3D imaging sensor orientation and navigation algorithm, which is highlighted below.

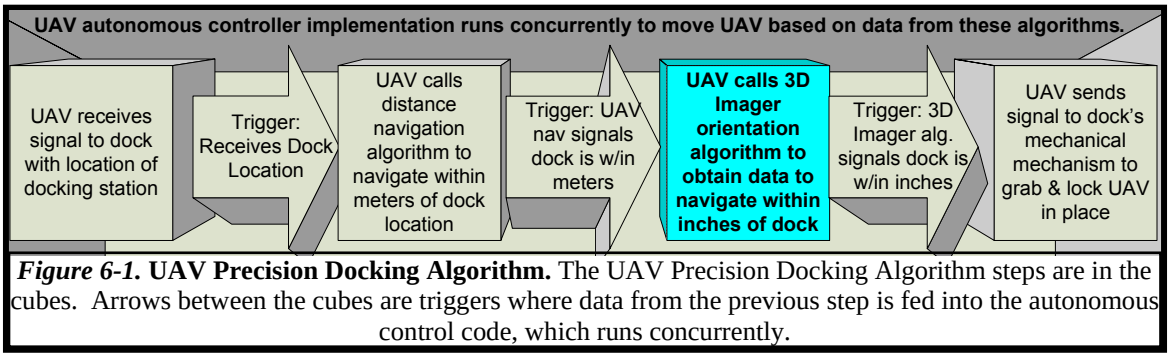


Figure 6-1. **UAV Precision Docking Algorithm.** The UAV Precision Docking Algorithm steps are in the cubes. Arrows between the cubes are triggers where data from the previous step is fed into the autonomous control code, which runs concurrently.

In the operational scenario, the UAV gains proximity to the docking station using navigation aids, such as GPS and INS or alternatives such as vision or laser-based navigation systems. The UAV cues a 3D imaging sensor orientation algorithm when the UAV is within meters of the dock, to complement the limited precision docking capability of GPS and INS. The 3D imaging sensor orientation algorithm provides precision data to the UAV, which allows the UAV to navigate to within inches of the dock.

The UAV autonomous controller implementation runs concurrently with the UAV precision docking algorithm. The precision docking algorithms feed data into the controller implementation at the triggers (denoted as arrows in **Figure 6-1**). Subsystem

three currently implements only the 3D imaging sensor navigation and orientation algorithms. Once Ohio University's UAV and its autonomous controller implementation reach a sufficient maturity level, the autonomous control code should be integrated with the UAV precision docking algorithm. This implementation should be developed and tested in future research.

6.1 Orientation Theory

6.1.1 Conceptual Theory.

The UAV navigation and orientation algorithm provides critical precision docking data to enable the UAV to navigate within a few inches of the dock. The UAV and the docking station each have their own rigidly attached coordinate frames. The navigation and orientation algorithm then finds the rotation and translation from the UAV (or camera) coordinate frame to the docking station coordinate frame.

The algorithm can determine the relative position vector (in meters) that the UAV needs to transverse to reach the docking station. The rotation is output in Euler angles representing roll (φ), pitch (θ) and yaw (ψ). Given the relative rotation and position, the UAV controller implementation can input this data to a control algorithm that will guide the UAV to the docking station in future research.

To estimate the rotation and translation, the subsystem three algorithm requires data from subsystems one and two as inputs. Specifically, the three planes identified and associated by subsystem two can be seen as three vectors expressed in the camera (or UAV) coordinate frame. **Figure 6.1-1** pictorially shows the body frame attached to the UAV and **Figure 6.1-2** shows the fixed frame, which is set with the camera facing down

a few inches from the center of the dock. The fixed frame coordinates stay constant and represent the fixed docking station location.

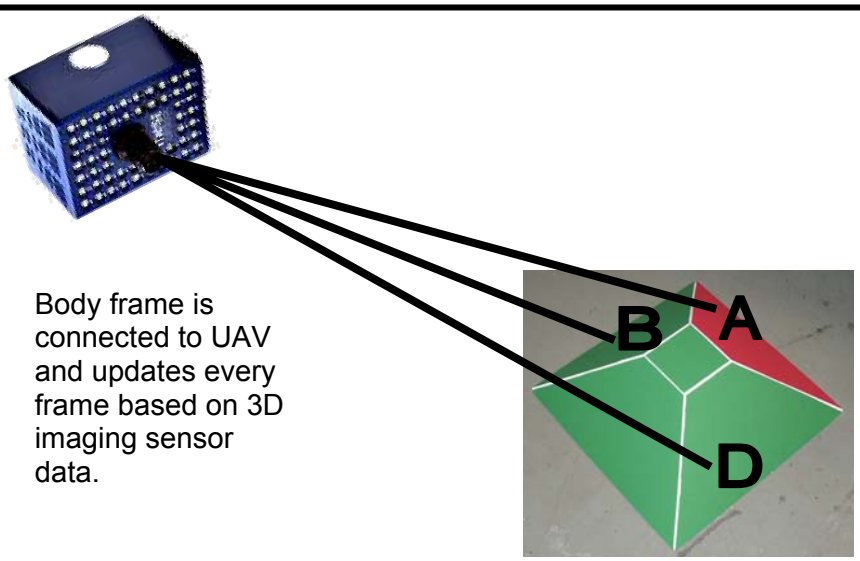


Figure 6.1-1. **Body Coordinate Frame.** 3D imaging sensor and UAV are connected to the body coordinate frame. The inputs from subsystem twp are in the body frame.

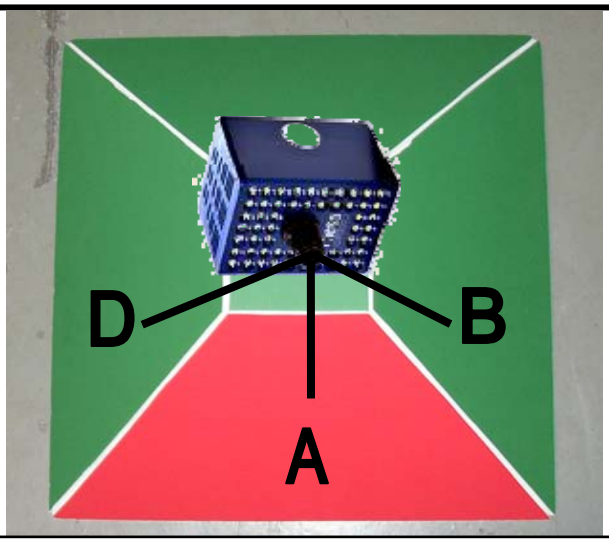


Figure 6.1-2. **Fixed Frame.** The fixed orientation algorithm compares the sensor location to the docking station location. The goal is to align the sensor with the fixed sensor location a few inches from the dock as shown.

The first algorithm step calculates the relative rotation between the 3D imaging sensor coordinate frame and the fixed docking station coordinate frame. The second step computes the relative position vector between the 3D imaging sensor coordinate frame and the fixed docking station coordinate frame. Currently, the subsystem three outputs computationally line up the coordinate frames, and these outputs will be used in the controller implementation in future research. There are many methods to physically align the UAV to the dock and these will be evaluated in future research.

6.1.2 Preferred Orientation Methodology.

The approach to estimate the relative orientation is based on Horn's closed form solution of absolute orientation using quaternions, hereafter referred to as Horn's algorithm. All orientation algorithms adhere to similar linear algebra theorems, but numerous forms can represent rotation. In this research, quaternions are used to represent and calculate rotations in the algorithm. After the algorithm completes calculation of the quaternion, the algorithm converts the quaternion to Rotation Matrices and Euler angles for output to the outer-loop controller. The translation is a straightforward algorithm that has been used in previous research so these equations will be defined in the implementation in Section 6.2.

Popular methods for expressing rotation include "Euler angles, Gibbs vector, Cayley-Klein parameters, Pauli spin matrices, axis and angle, orthonormal matrices, and Hamilton's quaternions." [29] Our rotation calculation requires the most effective method based on the available data from the first two subsystems. One method to perform the rotation between two coordinate frames uses orthonormal matrices. However, the rotation matrix must be orthonormal (orthogonal and normalized), and a complex algorithm is required to check for its orthonormality [32].

Quaternions are a preferred method for this application and are "widely used in simulation, robotics, guidance and navigation calculations, attitude control, and graphics animation." [33] Because orthonormal rotation matrices and Euler angles are easier to conceptualize than quaternions, the quaternions are converted into rotation matrices and Euler angles for data output rotation estimate is completed. Furthermore, most controller implementations utilize Euler angles, so the conversion supports future incorporation of this algorithm into the UAV's autonomous controller implementation.

6.1.3 Quaternions.

Quaternions offer the best rotation solution with minimal computation time and complexity. Symbolically, a quaternion $\dot{q}$ is represented as follows:

$$\dot{q} = q_0 + iq_x + jq_y + kq_z$$

Quaternions are conceptualized in 4 dimensions (4D) with a 1D scalar component and a 3D vector component or in 4D with a real number component and three imaginary components. Rotations require a unit quaternion, and verifying unit quaternions is less complex than verifying orthonormality for a rotation matrix [29].

The quaternion's additional dimension is advantageous, although it imposes special mathematical properties that must be considered. Each imaginary component has its own multiplication properties as follows:

$$i^2 = j^2 = k^2 = -1 \qquad ji = -k \qquad kj = -i \qquad ik = -j$$

These unique relationships drive multiplication of quaternions to be noncommutative. [29]

This section discusses the quaternion properties to familiarize the reader only with operations that are used in Section 6.2, the implementation of the orientation algorithm.

A more comprehensive review of quaternion properties and operations can be commonly found on the internet and is also further discussed in Horn, *Closed form solution of absolute orientation using unit quaternions* [29].

This research requires quaternions to be represented in MATLAB. Since MATLAB does not have built-in functions to handle typical quaternion representations, it is necessary to use another format. In addition to quaternions being conceptualized as a 1D scalar and 3D vector, quaternions can also be mathematically represented by the sum of a scalar and a vector. This representation is useful because the 1D scalar can be stored as a scalar in MATLAB and the 3D vector can be stored as a 1x3 matrix. Symbolically quaternions are represented by the sum of a scalar and a vector as follows: [29]

$$\dot{q} = q + \mathbf{q}$$

$$\text{Where } q = q_0 \text{ and } \mathbf{q} = \begin{bmatrix} q_x & q_y & q_z \end{bmatrix}^T$$

6.1.4 Direction Cosine Matrices and Euler Angles.

After rotation calculations are computed using quaternions, they are converted into Direction Cosine Matrices (DCM) and Euler angles. Quaternions are converted into rotation matrices as follows [29]:

$$R = \begin{bmatrix} \left(q_0^2 + q_x^2 - q_y^2 - q_z^2\right) & 2\left(q_x q_y - q_0 q_z\right) & 2\left(q_x q_z + q_0 q_y\right) \\ 2\left(q_y q_x + q_0 q_z\right) & \left(q_0^2 - q_x^2 + q_y^2 - q_z^2\right) & 2\left(q_y q_z - q_0 q_x\right) \\ 2\left(q_z q_x - q_0 q_y\right) & 2\left(q_z q_y + q_0 q_z\right) & \left(q_0^2 - q_x^2 - q_y^2 + q_z^2\right) \end{bmatrix}$$

The 3D rotation matrix is composed of three successive rotation matrices about the coordinate frame axes. The rotation matrices are formed based on the rotation about the X, Y and Z axis. For the navigation to body coordinate transformation, Euler angles are assigned to rotation about each axis; roll (φ) is about the X axis, pitch (θ) is about the

Y axis, and yaw (ψ) is about the Z axis. "In the aerospace field the rotations are performed, in a specified order, about each of the three Cartesian axes in succession." [33]

The three 2D rotation matrices are multiplied to obtain the 3D rotation matrix. Since matrix multiplication is not commutative, the order of the 2D matrices must be defined and kept constant. In order to keep the equations consistent within the algorithm, this research will base its rotation matrices on the following recommendation from Aircraft Control and Simulation: "Standard aircraft practice is to describe the aircraft orientation by the z, y, x (also called 3, 2, 1) right-handed rotation sequence that is required to get from a reference system on the surface of the Earth into alignment with an aircraft body-fixed coordinate system." [33] The standard z, y, x navigation to body frame Direction Cosine Matrix (C_n^b) is not necessary for the implementation using actual data since the data inputs are already expressed in the body frame (the camera frame in our application). The C_n^b is required for the simulation and is discussed in Section 6.4. The C_n^b is as follows:

$$C_n^b = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix}$$

The orientation algorithm requires estimation of the rotation from the UAV body-fixed coordinate frame (body frame) to the coordinate frame that has the same orientation as the coordinate frame rigidly attached to the docking station (navigation frame). The body to navigation frame DCM (C_b^n) is required because the data inputs are in the body frame and the calculated orientation is expressed in the navigation frame. Therefore, the transpose of C_n^b is used [31]:

$$C_b^{\ n} = \left[C_n^{\ b}\right]^T = \left[C_3 C_2 C_1\right]^T = C_1^{\ T} C_2^{\ T} C_3^{\ T}$$

The algorithm represents the C_b^n as R and the C_b^n is as follows [31]:

$$C_b^{\ n} = R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

$$R = \begin{bmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}$$

The orientation algorithm computes Euler angles from the rotation matrix. As previously mentioned, the algorithms in **Figure 6-1** use Euler angles as the common input/output so our orientation algorithm outputs are in the correct format to integrate into the controller implementation algorithms. We derive equations for Euler angles (φ, θ and ψ) by taking the inverse trigonometric functions of selected components of the rotation matrix. The following set of equations for Euler angles are obtained:

$$\psi = a\tan 2\left(R_{21}, R_{11}\right)$$
$$\phi = a\tan 2\left(R_{32}, R_{33}\right)$$
$$\theta = -a\sin\left(R_{31}\right)$$

This completes the background information on rotation matrices, Euler angles and quaternions that is necessary to implement the orientation algorithm. Translation is not addressed because it does not require any complex calculations. Section 6.2 orders these equations into an algorithm to obtain the UAVs orientation, Section 6.3 describes the results of the MATLAB implementation of this algorithm, and Section 6.4 covers the simulation that confirms this algorithm yields correct results.

6.2 Implementation

6.2.1 Horn's Algorithm Overview.

Horn's algorithm solves for orientation and outputs the rotation, translation and scale between two coordinate systems. The coordinate systems are each defined by a minimum of three planes. Horn computes the orientation by using 9 available input parameters (from the required three planar surfaces) to solve for the 6 transformation parameters. Scale is Horn's 7th transformation parameter, but scale is not required in this research since we use 3D inputs rather than stereo-photogrammetry. [29]

The x, y and z components from each of the three planes' normal vectors compose the nine input parameters. The transformation parameters are composed of six Degrees Of Freedom (DOF); three from rotation (roll φ, pitch θ, and yaw ψ) and three from translation (x, y, and z). Horn uses the available data to solve for the transformation parameters and minimizes the least-squares of the errors to obtain the output. [29]

Horn's algorithm was selected because it provides "a closed-form solution to the least-squares problem of absolute orientation, one that does not require iteration. [29]" This algorithm uses all information from all available points, which is advantageous for future research where more than three vectors are available (like using all visible planar surfaces on the docking station). Horn also provides a simplified algorithm that can be used when only three vectors are available. Subsystem three implements the algorithm for more than three vectors so the algorithm can be applied to data sets with more than three vectors in the future. [29]

The algorithm is broken into two steps for rotation and translation. Rotation is the main focus since it is the most complicated transformation parameters to compute. The

translation is straightforward once the rotation is computed and will be discussed after treatment of the rotation estimation procedure.

6.2.2 Horn's Algorithm Inputs.

This research implements Horn's algorithm to detect the orientation between two coordinate systems: a fixed set of planes on the docking station and the set of planes identified by the 3D imaging sensor (the output from subsystem two). The algorithm outputs the necessary data for the UAV to determine its position and orientation with respect to the docking station. Horn's algorithm is implemented in MATLAB Version 7.3.0 (R2006b). Subsystem three's main function is *find_orientation*, which implements Horn's algorithm and outputs orientation and relative position. It is called in the main code "processframes.m" and the function call is as follows:

```
[orientation] = find_orientation(fixed, orientation, plane_id, jj)
```

The processframes.m function iterates through every frame and calls the find_orientation function once for each 3D imaging sensor data frame. The *jj* variable defines the current 3D imaging sensor frame. The *orientation* structure contains the rotation and translation for each frame. It is passed into the function with the previous frame's orientation data and passed out of the function with the previous and current frame's orientation data. This allows the user to analyze the orientation across a number of frames after the code is processed.

This function detects the UAV's orientation change between two camera frames (relative orientation) or the distance from the UAV to the docking station (fixed orientation). The *fixed* input variable allows the user to select the fixed or relative orientation.

When the *fixed* input variable equals 1, the fixed orientation is found, which detects the orientation from the UAV in the current frame to the fixed docking station. This orientation is referred to as fixed since one set of planes (the docking station) is used as a reference and remains constant (fixed) throughout the code.

When the *fixed* input variable equals 0, the relative orientation is found, which detects the orientation from the UAV in the previous frame to the UAV in the current frame. This orientation is referred to as relative because it is relatively based on the position of the UAV in the previous frame, and it is independent of a fixed reference point, such as the dock.

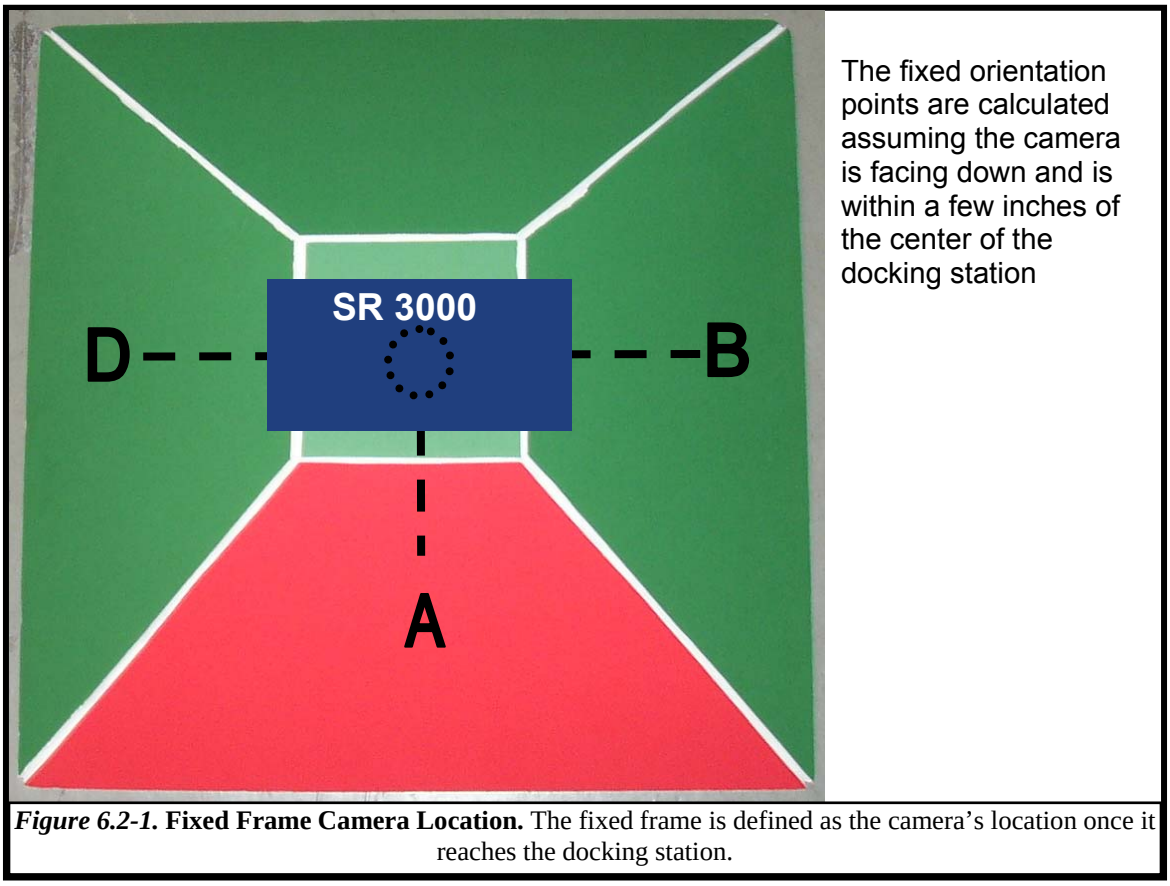


Figure 6.2-1. **Fixed Frame Camera Location.** The fixed frame is defined as the camera's location once it reaches the docking station.

The input variable *plane_id* contains the data from subsystem two, which includes three planes identified consistently in a specific order and the planes' properties like its normal vector and centroid. Since the planar surfaces are identified in a specific order that is consistently known, the fixed coordinates required for the fixed orientation are set at the point where the UAV would be within inches of the docking station as shown in **Figure 6.2-1**. The consistent order also allows the relative orientation to consistently compare the three planes from the previous frame to the same three planes in the current frame.

This implementation focuses on the fixed orientation option because the fixed orientation outputs the data required for the controller implementation that helps guiding the UAV to the dock. The relative orientation is an additional feature that provides supplemental data. For example, the relative data could be used in the future as a more precise navigation aid at lower altitudes to complement INS/GPS during the mission.

For the fixed orientation, the fixed coordinates are identified geometrically and hard coded into the *find_orientation* algorithm. The fixed vectors are hard-coded because this research is dependent on the shape of the docking station. Opportunities for additional docking station geometries are further discussed in Chapter 7.

The actual docking station mockup size defines the fixed coordinates for the dock. The mockup used for data collection was constructed with five pieces of foam cardboard taped together. When the mockup was originally constructed the four sides had normals that were approximately 45° from a 90° line. Between the docking station mockup construction and the data collection, the docking station "settled" a few inches. The approximate dimensions of the docking station before and after "settling" is shown in **Figure 6.2-2** below.

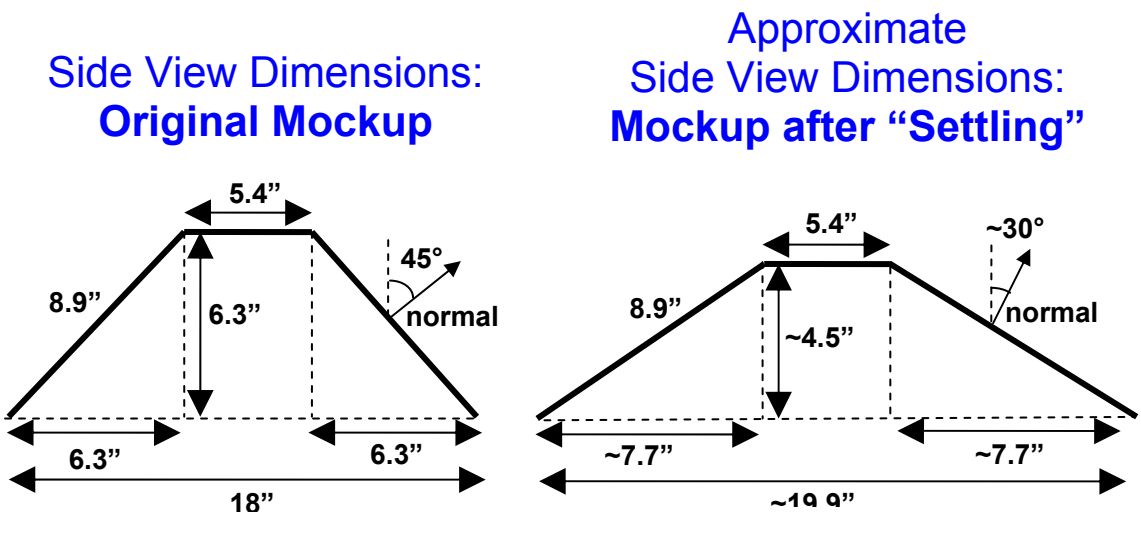


Figure 6.2-2. **Docking Station Dimensions.** The docking station “settled” between construction and data collection. The left side shows the original dock dimensions, and the right side shows approximate “settled” dock dimensions.

There was also a time delay between the data collection and processing. Although the dock “settled” before data collection, it was not recognized until after the data was processed. The order of these events is shown chronologically in **Figure 6.2-3**. It is necessary to approximate the docking station dimensions at the time of data collection since the dock “settling” was recognized well after the data collection.

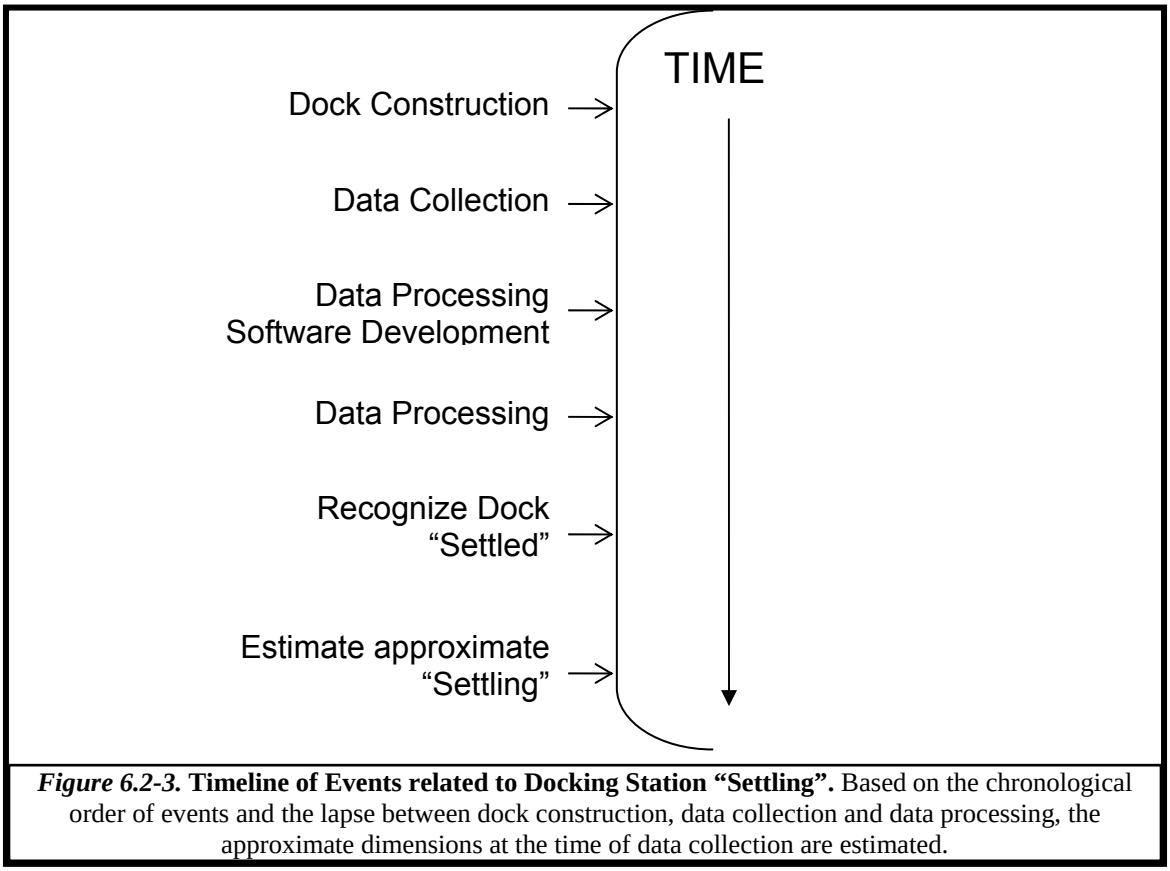


Figure 6.2-3. **Timeline of Events related to Docking Station "Settling".** Based on the chronological order of events and the lapse between dock construction, data collection and data processing, the approximate dimensions at the time of data collection are estimated.

The physical sides of the docking station remain the same dimensions so it is only necessary to estimate the change to the normal vectors of the dock sides after "settling". Based on the photographic images taken when the data was collected and the reflectance data from the 3D imaging sensor it appears the docking station "settled" only a few inches. This reflects approximately a 10-15° change in the normal vectors. For documentation purposes, it is assumed the dock "settled" 15° and the respective approximate dimensions are shown in **Figure 6.2-2** above.

For this research, it is not necessary to utilize the dimensions of the "settled" docking station. The goal for subsystem three is to output data that will allow the UAV to navigate within inches of the docking station. Using dimensions that are 1-2 inches

above the actual dock ensures that we are within inches of the dock, while leaving margin to allow a mechanic latch to bring the UAV in the last few inches. This mitigates the risk of the UAV accidentally bumping into the dock, which could cause damage to the dock or the UAV.

A mechanic latch also allows the UAV to consistently connect power or data sources so it can immediately begin to charge, upload surveillance data, or download new mission commands. Once a more structurally robust docking station mockup or the production docking station is constructed it is recommended to alter the fixed location of the docking station in the code to 1-2 inches above the actual top of the dock to allow for this margin.

The hard coded docking station centroid points are geometrically derived from the original mockup construction dimensions as shown below in **Figure 6.2-4**. The center of the camera is assumed to be aligned with the center of the docking station, and the roll, pitch and yaw of the UAV and 3D imaging sensor are assumed to be zero. The normal vectors are also hard coded in assuming the original 45 degree normal vectors. The points hard coded into the *find_orientation* function are as follows:

```
centroidA = [0; -0.18; 0.07];     normalA = [-0.7; 0; 0.7];
centroidB = [-0.18; 0; 0.07];     normalB = [0; -0.7; 0.7];
centroidD = [0.18; 0; 0.07];      normalD = [0; 0.7; 0.7];
```

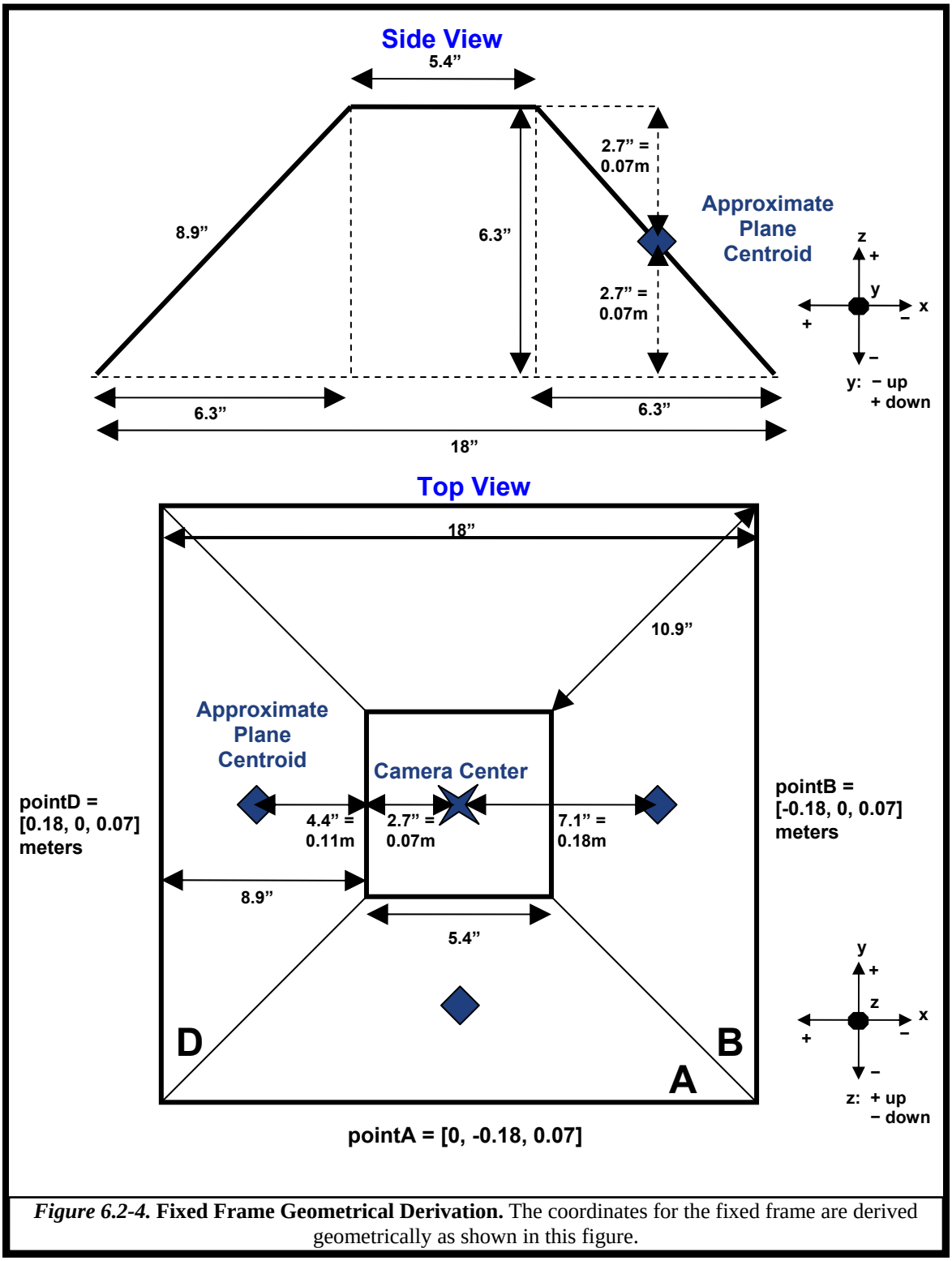


Figure 6.2-4. **Fixed Frame Geometrical Derivation.** The coordinates for the fixed frame are derived geometrically as shown in this figure.

6.2.3 Orientation and Position Algorithm Implementation.

Horn's algorithm [29] is implemented to calculate orientation and the position change estimation from [14] is used to calculate translation. Horn's algorithm refers to one set of planes as the left coordinate system and the other set of planes as the right coordinate system so this algorithm uses that nomenclature. The left coordinate system data is from the *plane_id* variable. This variable contains the output for each of the planes identified by subsystem two. [29]

The only difference between the fixed and relative orientation in the code are the values for the right coordinates. The fixed docking station plane properties that were previously defined are used as the right coordinate system when calculating fixed orientation. The previous frame's plane properties are used as the right coordinate system when calculating relative orientation. Orientation is not calculated for either fixed or relative orientation in the first frame. The remainder of the discussion will assume the fixed orientation is being calculated, although after the coordinates are defined the algorithm is exactly the same for either fixed or relative orientation. [29]

This algorithm requires planar surface properties including the centroid and the normals. The normals of each plane are denoted by the variable name *normal* and the points in the center of each plane are denoted with the variable name *r*. The normals are used to calculate rotation and the Cartesian coordinates of the center of each plane are used to calculate translation.

This section follows the same order as the algorithm: the variables are defined, then rotation is computed, then translation is computed. In the following equations the first variable name is the annotation based on Horn's algorithm and the second variable name is the MATLAB code annotation. [29]

First, the rotation variables are defined. The rotation uses the planar surfaces' normal vectors. The left coordinate is the *normal* vector variable from the *plane_id* input structure and the right coordinate is the fixed set of normals. For fixed orientation the normal vectors are stored as follows:

Left normals (`plane_id(jj).name(x).normal`) Right normals (Fixed dock normals)

$$\mathbf{n}_{l,1} = L1normal = \begin{bmatrix} Nx_{l,1} & Ny_{l,1} & Nz_{l,1} \end{bmatrix}^T \qquad \mathbf{n}_{r,1} = R1normal = \begin{bmatrix} -0.7 & 0 & 0.7 \end{bmatrix}^T$$
$$\mathbf{n}_{l,2} = L2normal = \begin{bmatrix} Nx_{l,2} & Ny_{l,2} & Nz_{l,2} \end{bmatrix}^T \qquad \mathbf{n}_{r,2} = R2normal = \begin{bmatrix} 0 & -0.7 & 0.7 \end{bmatrix}^T$$
$$\mathbf{n}_{l,3} = L3normal = \begin{bmatrix} Nx_{l,3} & Ny_{l,3} & Nz_{l,3} \end{bmatrix}^T \qquad \mathbf{n}_{r,3} = R3normal = \begin{bmatrix} 0 & 0.7 & 0.7 \end{bmatrix}^T$$

Second the translation variables are defined. The translation uses the planar surfaces' normal vectors (shown above) and the centroids. The left coordinate centroid (*x0)* variable is from the *plane_id* input structure and the right coordinate centroid is the fixed set of points. For fixed orientation the centroid coordinates are stored as follows:

Left points (`plane_id(jj).name(x).x0`) Right points (Fixed docking station points)

$$\mathbf{r}_{l,1} = rL1 = \begin{bmatrix} A_x & A_y & A_z \end{bmatrix} \qquad \mathbf{r}_{r,1} = rR1 = \begin{bmatrix} 0; & -0.18; & 0.07 \end{bmatrix}$$
$$\mathbf{r}_{l,2} = rL2 = \begin{bmatrix} B_x & B_y & B_z \end{bmatrix} \qquad \mathbf{r}_{r,2} = rR2 = \begin{bmatrix} -0.18; & 0; & 0.07 \end{bmatrix}$$
$$\mathbf{r}_{l,3} = rL3 = \begin{bmatrix} C_x & C_y & C_z \end{bmatrix} \qquad \mathbf{r}_{r,3} = rR3 = \begin{bmatrix} 0.18; & 0; & 0.07 \end{bmatrix}$$

Third, the rotation is calculated. The rotation calculation is based on [29]:

$$\hat{\mathbf{C}} = \arg\max_{\mathbf{C}} \underbrace{\left\{ \sum_{i=1}^{\mathrm{N}} \mathbf{C}\mathbf{n}_{l,i} \cdot \mathbf{n}_{r,i} \right\}}_{F}$$

In other words we are trying to find that value of the DCM 'C' that maximizes the sum of the dot products *F*. Equation X.X can now be rewritten using quaternions, given that a rotation using a quaternion is given by [29]:

$$\mathbf{n}_{r,i} = \mathbf{C}\mathbf{n}_{l,i}$$
$$\dot{\mathbf{n}}_{r,i} = \dot{\mathbf{q}}\dot{\mathbf{n}}_{l,i}\dot{\mathbf{q}}^{*}$$

where the quaternion of the normal vectors is formed by adding a zero real part and using the normal vectors $\mathbf{n}_{l,i} = \begin{pmatrix} nx_{l,i} & ny_{l,i} & nz_{l,i} \end{pmatrix}$ and $\mathbf{n}_{r,i} = \begin{pmatrix} nx_{r,i} & ny_{r,i} & nz_{r,i} \end{pmatrix}$ as the imaginary part of the quaternions. Parameter *F* in Equation X.X then can be re-written as [29]:

$$\begin{aligned} F &= \sum_{i=1}^{\mathrm{N}} \dot{\mathbf{q}}\dot{\mathbf{n}}_{l,i}\dot{\mathbf{q}}^{*} \cdot \dot{\mathbf{n}}_{r,i} \\ &= \sum_{i=1}^{\mathrm{N}} \dot{\mathbf{q}}\dot{\mathbf{n}}_{l,i} \cdot \dot{\mathbf{n}}_{r,i}\dot{\mathbf{q}} \end{aligned}$$

Since multiplication of a quaternion with another quaternion can also be written as a matrix multiplication, the elements of the previous equation can be expanded as follows [29]:

$$\dot{\mathbf{q}}\dot{\mathbf{n}}_{l,i} = \begin{bmatrix} 0 & -nx_{l,i} & -ny_{l,i} & -nz_{l,i} \\ nx_{l,i} & 0 & nz_{l,i} & -ny_{l,i} \\ ny_{l,i} & -nz_{l,i} & 0 & nx_{l,i} \\ nz_{l,i} & ny_{l,i} & -nx_{l,i} & 0 \end{bmatrix} \dot{q} = N_{L,i}\dot{q}$$

$$\dot{\mathbf{n}}_{r,i}\dot{\mathbf{q}} = \begin{bmatrix} 0 & -nx_{r,i} & -ny_{r,i} & -nz_{r,i} \\ nx_{r,i} & 0 & -nz_{r,i} & ny_{r,i} \\ ny_{r,i} & nz_{r,i} & 0 & -nx_{r,i} \\ nz_{r,i} & -ny_{r,i} & nx_{r,i} & 0 \end{bmatrix} \dot{q} = N_{R,i}\dot{q}$$

Next, the matrix multiplication quaternion property and subsequently the properties of quaternion dot products are used to rewrite the parameter *F*. Then, the summation can be pulled inside since only the middle elements are impacted by it. Finally, the normal vector elements of the matrices $N_{L,i}$ and $N_{R,i}$ are multiplied together to create the matrix *N*. These modifications result in the parameter *F* being rewritten as follows [29]:

$$
\begin{aligned}
F &= \sum_{i=1}^{\mathrm{N}} (N_{L,i}\dot{\mathbf{q}}) \cdot (N_{R,i}\dot{\mathbf{q}}) \\
&= \sum_{i=1}^{\mathrm{N}} \dot{\mathbf{q}}^T {N_{L,i}}^T N_{R,i}\dot{\mathbf{q}} \\
&= \dot{\mathbf{q}}^T \left(\sum_{i=1}^{\mathrm{N}} {N_{L,i}}^T N_{R,i} \right) \dot{\mathbf{q}} \\
&= \dot{\mathbf{q}}^T \left(\sum_{i=1}^{\mathrm{N}} N_i \right) \dot{\mathbf{q}} = \dot{\mathbf{q}}^T N \dot{\mathbf{q}}
\end{aligned}
$$

The matrix *M* is introduced because it contains all of the components needed to compose the matrix *N*. The matrix *M* is a sum of products of the normal vectors. These elements are calculated as follows [29]:

$$
M = \sum_{i=1}^{n} \mathbf{n}_{l,i}\mathbf{n}_{r,i} = \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{bmatrix}
$$

$$
S_{xx} = Sxx = \sum_{i=1}^{n} nx_{l,i}nx_{r,i} = nx_{l,1}nx_{r,1} + nx_{l,2}nx_{r,2} + \ldots nx_{l,i}nx_{r,i}
$$

$$
\begin{aligned}
S_{xy} &= Sxy = \sum_{i=1}^{n} nx_{l,i}ny_{r,i} & \qquad S_{yz} &= Syz = \sum_{i=1}^{n} ny_{l,i}nz_{r,i} \\
S_{xz} &= Sxz = \sum_{i=1}^{n} nx_{l,i}nz_{r,i} & \qquad S_{zx} &= Szx = \sum_{i=1}^{n} nz_{l,i}nx_{r,i} \\
S_{yx} &= Syx = \sum_{i=1}^{n} ny_{l,i}nx_{r,i} & \qquad S_{zy} &= Szy = \sum_{i=1}^{n} nz_{l,i}ny_{r,i} \\
S_{yy} &= Syy = \sum_{i=1}^{n} ny_{l,i}ny_{r,i} & \qquad S_{zz} &= Szz = \sum_{i=1}^{n} nz_{l,i}nz_{r,i}
\end{aligned}
$$

The sums and differences of the components in *M* compose the 4 x 4 matrix N. [29]:

$$N = \begin{bmatrix} \left(S_{xx} + S_{yy} + S_{zz}\right) & S_{yz} - S_{zy} & S_{zx} - S_{xz} & S_{xy} - S_{yx} \\ S_{yz} - S_{zy} & \left(S_{xx} - S_{yy} - S_{zz}\right) & S_{xy} + S_{yx} & S_{zx} + S_{xz} \\ S_{zx} - S_{xz} & S_{xy} + S_{yx} & \left(-S_{xx} + S_{yy} - S_{zz}\right) & S_{yz} + S_{zy} \\ S_{xy} - S_{yx} & S_{zx} + S_{xz} & S_{yz} + S_{zy} & \left(-S_{xx} - S_{yy} + S_{zz}\right) \end{bmatrix}$$

It is now necessary to find the unit quaternion that maximizes the equation *F*. This unit quaternion represents the rotation and maximizes the following equation [29]:

$$F = \dot{q}^T N \dot{q}$$

To solve for the quaternion rotation, it is necessary to find the most positive eigenvalue (λ_m) of *N*. The eigenvalues are found by expanding the following equation and solving for the fourth order polynomial [29]:

$$\det\left(N - \lambda I\right) = 0\,,$$

where *I* is the identity matrix and λ is the set of eigenvalues.

The maximum eigenvalue is selected from the set of eigenvalues and the associated eigenvector $\dot{e}_m$ is identified by solving the following equation:

$$\left[N - \lambda_m I\right]\dot{e}_m = 0$$

This eigenvector $\dot{e}_m$ is the unit quaternion $\dot{q}$ that maximizes the equation *F*. [29]

$$\dot{q} = \dot{e}_m$$

$\dot{q}$ is the rotation quaternion between the two frames. Since quaternions are conceptually difficult to understand the output is converted into Euler angles. First the quaternion is converted to a rotation matrix. Next, equations derived in Section 6.1 are implemented to convert the rotation matrix to Euler angles. It is important to remember from Section 6.1 that it is necessary to maintain a consistent rotation sequence for accurate results. The conversion equations are as follows: [29, 30, 31, 33]

Extract Components from Overall Rotation Quaternion

$$q_0 = q0 = q$$
$$q_1 = q1 = \mathbf{q}(1)$$
$$q_2 = q2 = \mathbf{q}(2)$$
$$q_3 = q3 = \mathbf{q}(3)$$

Convert from Quaternions to Rotation Matrix

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} = \begin{bmatrix} \left(q_0^2 + q_x^2 - q_y^2 - q_z^2\right) & 2\left(q_x q_y - q_0 q_z\right) & 2\left(q_x q_z + q_0 q_y\right) \\ 2\left(q_y q_x + q_0 q_z\right) & \left(q_0^2 - q_x^2 + q_y^2 - q_z^2\right) & 2\left(q_y q_z - q_0 q_x\right) \\ 2\left(q_z q_x - q_0 q_y\right) & 2\left(q_z q_y + q_0 q_x\right) & \left(q_0^2 - q_x^2 - q_y^2 + q_z^2\right) \end{bmatrix}$$

Convert from Rotation Matrix to Euler Angles

(assumes x, y, z rotation sequence)

$$\psi = a\tan 2\left(R_{21}, R_{11}\right)$$
$$\phi = a\tan 2\left(R_{32}, R_{33}\right)$$
$$\theta = -a\sin\left(R_{31}\right)$$

Fourth, and finally, the translation is computed. The translation is an important orientation output for this research and is straightforward to compute. When using relative orientation translation shows how far the UAV has traveled from one 3D imaging sensor frame to the next, and when using fixed orientation translation shows how far the UAV is from the docking station. Originally, the translation was going to be calculated using the method proposed in Horn's algorithm. This equation relies on a consistent fixed point on each plane in every frame, and takes the centroid of those fixed points ($\bar{\mathbf{r}}$) in each frame. The translation equation is calculated as follows: [29]

Calculate Translation

$$\mathbf{r}_0 = translation = \bar{\mathbf{r}}_r - (\mathbf{R}\bar{\mathbf{r}}_l)$$

Where $\bar{\mathbf{r}}_r$ is the right coordinate frame centroid, $\bar{\mathbf{r}}_l$ is the left coordinate plane centroid, and **R** is the rotation matrix between the right and left coordinate frames.

This equation is not effective with the current data set because the best available fixed point is the centroid of each plane. As previously discussed, the centroid shifts if part of the plane is out of the camera's field-of-view. Although it is available in every frame, it is not consistent. Horn's equation for translation can be used in the future when the top planar surface is able to be detected. With the top planar surface and planes A, B and D available, three planar intersections can be defined and used as the consistent point.

For this implementation the equation from [14] is used. This equation also needs a point on the plane, but it does not require it to be consistent from frame to frame so the centroid is sufficient. It also uses the normal vectors. The distance d_i from the camera to the docking station is calculated for the right and left coordinate frames. This distance is calculated as follows [14]:

$$dL_i = \left| r_{l,i} \cdot n_{l,i} \right| \qquad dR_i = \left| r_{r,i} \cdot n_{r,i} \right|$$

$$\Delta d_i = dL_i - dR_i$$

The distance and the normals are used to determine the change in user position, $\Delta\mathbf{x}$ as follows [14]:

$$\begin{bmatrix} \mathbf{n}_1^T \\ \vdots \\ \mathbf{n}_N^T \end{bmatrix} \Delta\mathbf{x} = \begin{bmatrix} \Delta d_1 \\ \vdots \\ \Delta d_N \end{bmatrix} \Rightarrow \mathbf{A}\Delta\mathbf{x} = \Delta\mathbf{d}$$

Any least squares implementation can be used to solve for $\Delta\mathbf{x}$. This implementation uses the QR decomposition to solve for the position change [14].

This concludes the implementation of the orientation algorithm to compute rotation and translation for this research. Section 6.3 discusses the results from this implementation and Section 6.4 discusses the simulation and proves this implementation yields correct results.

6.3 Results Using 3D Imaging Sensor Data

The implementation discussed in section 6.2 has been implemented in MATLAB and the results are shown in this section. The data file has been obtained using the data collection setup with the Swissranger SR3000 described in Section 3.1. Since the 3D imaging sensor has not yet been installed on the Ohio University UAV, all motion (translational and rotational) is introduced by a person holding and moving the camera and performing all approaches to the docking station for data collection. This imitates the movement of the UAV approaching the docking station. The camera is held approximately 7-8 feet above the ground level at a slight angle. The person is a couple of feet away from the docking station. Note that no accurate truth reference was available for this data collection approach. It will be recommended to repeat the experiment with a good indoor truth reference, when possible.

Figures 6.3-1 – 6.3-6 show the results from a segment of the data. In the top left and right corners of each figure are the rotation and translation measurements, respectively. These estimates are relative to the fixed docking station (fixed orientation).

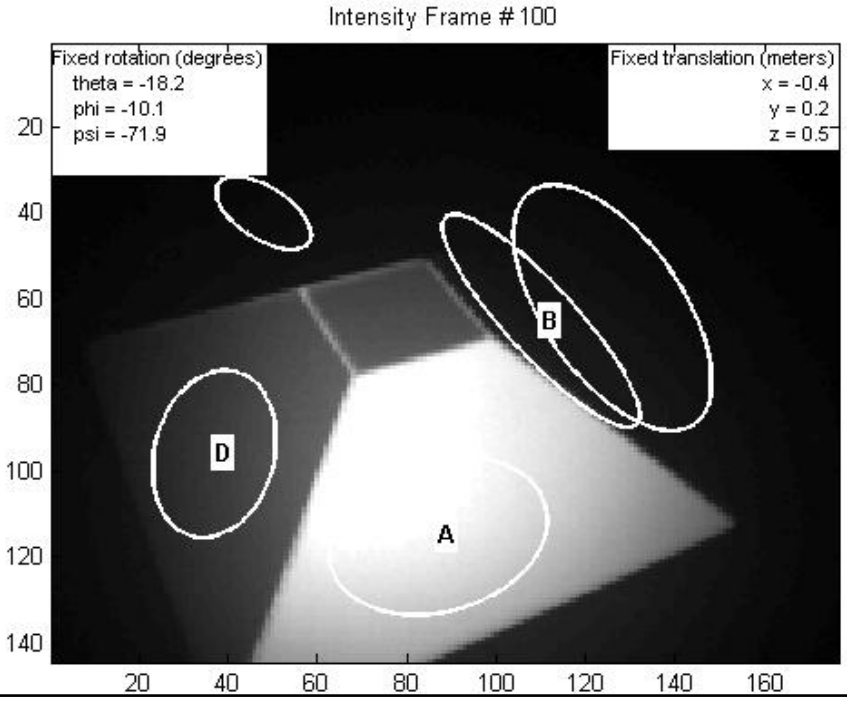


Figure 6.3-1. **Orientation Algorithm Results – Frame 100.**

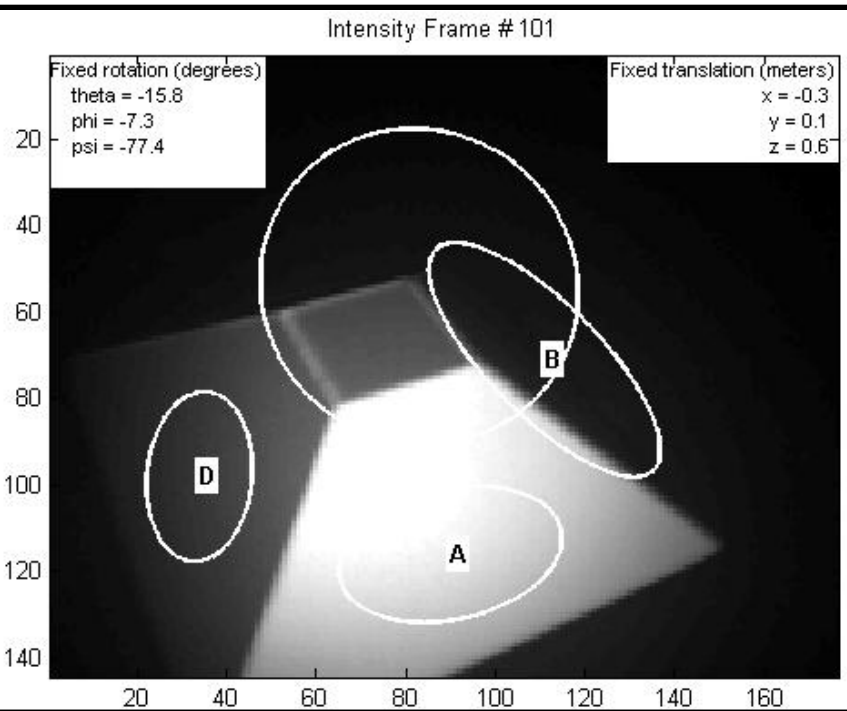


Figure 6.3-2. **Orientation Algorithm Results – Frame 101.**

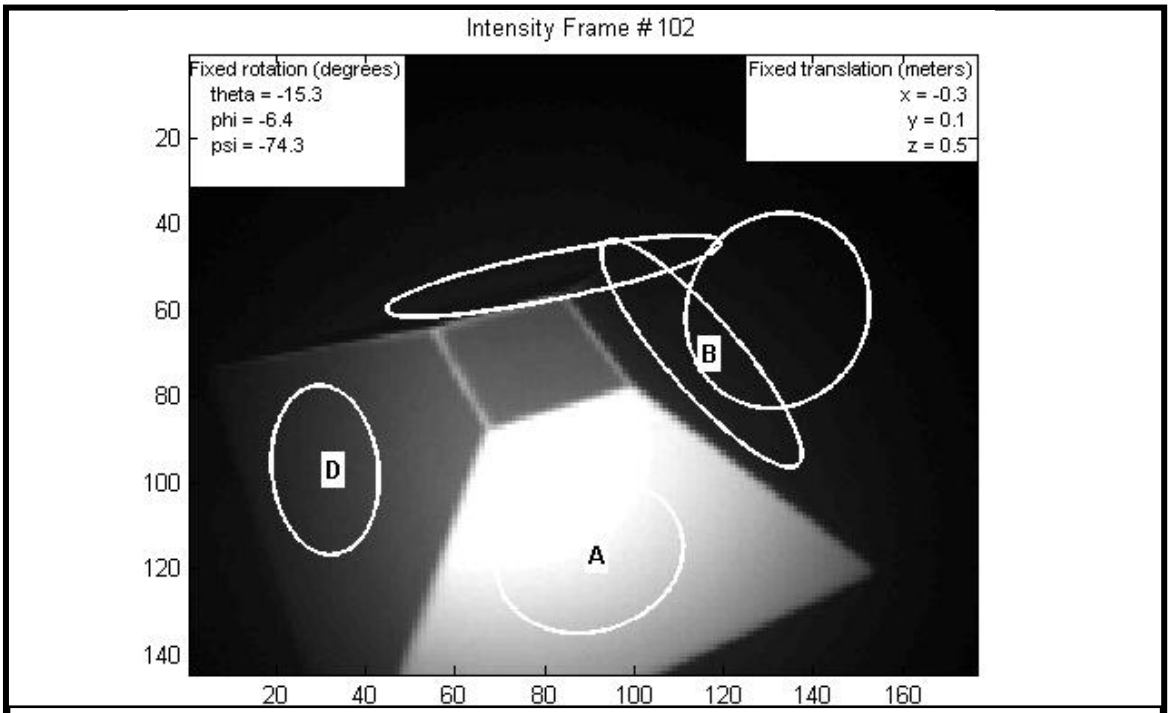


Figure 6.3-3. **Orientation Algorithm Results – Frame 102.**

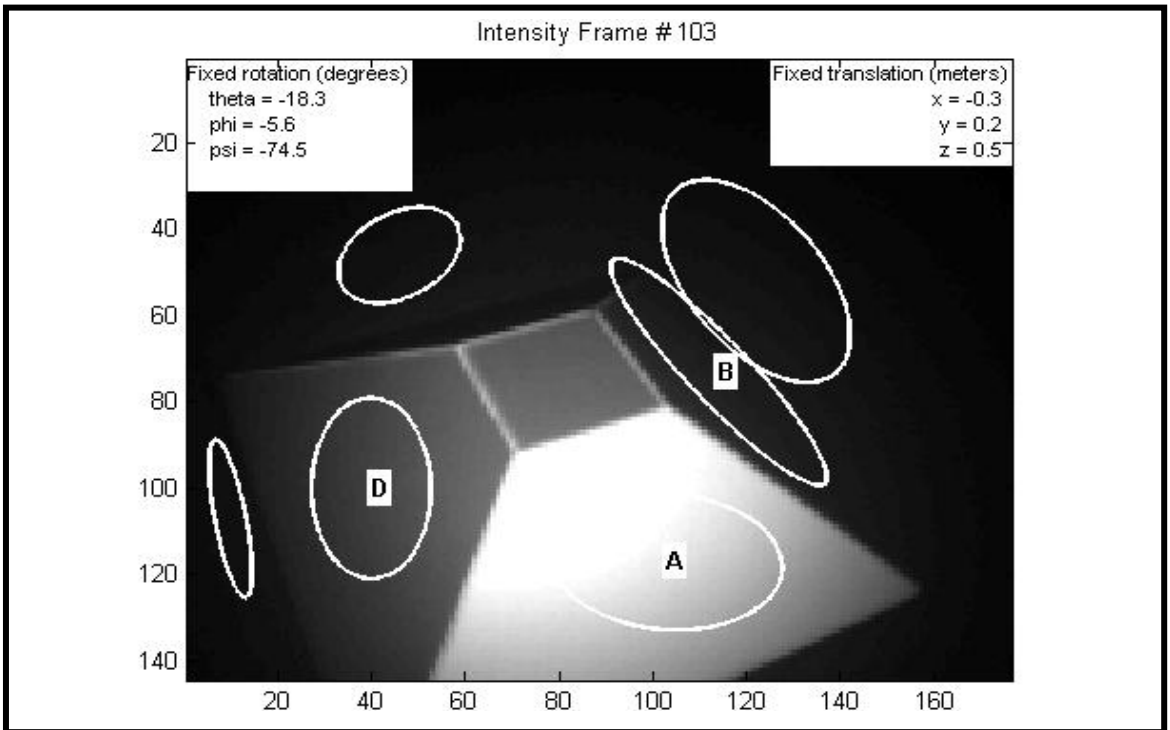


Figure 6.3-4. **Orientation Algorithm Results – Frame 103.**

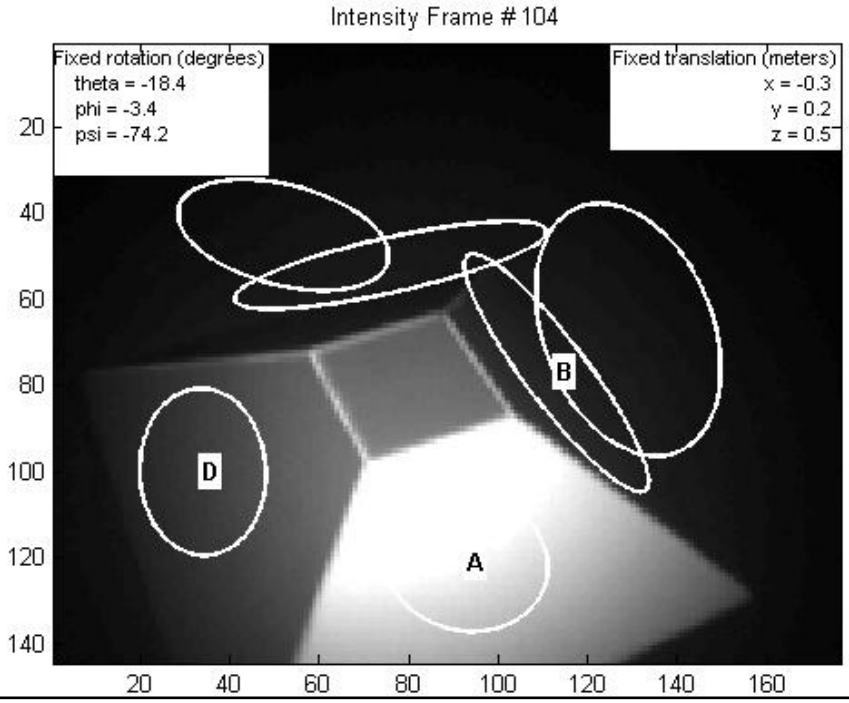


Figure 6.3-5. **Orientation Algorithm Results – Frame 104.**

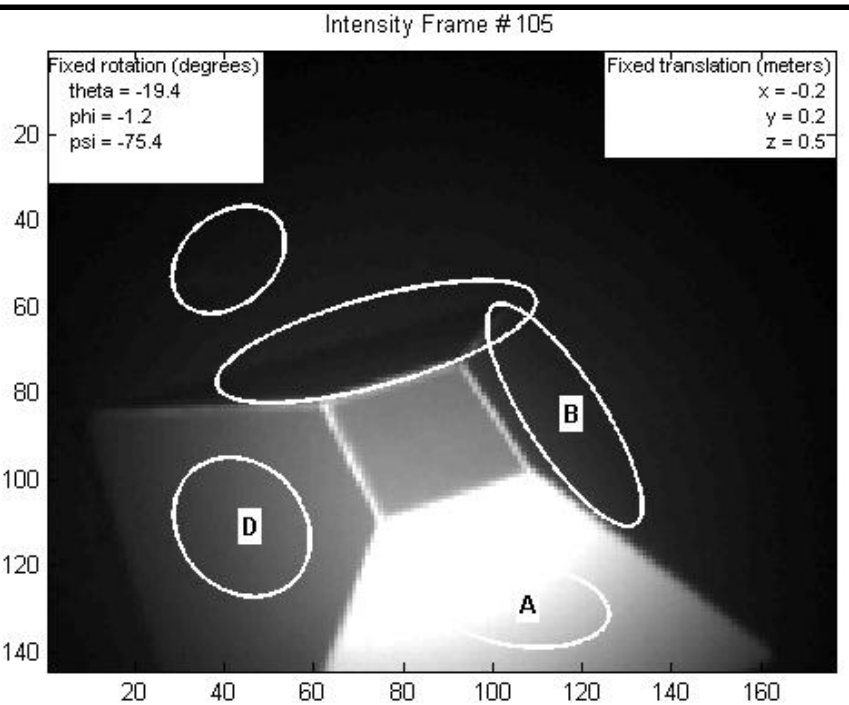


Figure 6.3-6. **Orientation Algorithm Results – Frame 105**

Although not validated, the results look reasonable from observations. The scope of these results and the data quality is limited at this time based on the SR 3000 hardware capabilities. The hardware capability with the software developments completed in this research is sufficient to yield results within the expected range. In the next section, the algorithms will be validated with simulated data to better assess the performance of the algorithms regardless of available hardware or truth reference system.

Based on limited camera resolution (176x144 pixels), the camera is not able to sufficiently observe multiple sides of the docking station at greater distances. The distance can be improved in the future while maintaining the current hardware (i.e. current resolution) by modifying the docking station geometry. This topic is further discussed in Chapter 7. Also, since other capabilities already exist to navigate a UAV from far distances such as INS/GPS solutions this capability is not necessarily required for the 3D imaging sensor solution.

If the 3D imaging sensor is too close to the docking station the LED based illuminator causes washout of the image resulting in the closest plane having the highest intensity rather than plane A. This is evident by comparing **Figure 6.3-1** (Frame 100) with **Figure 6.3-6** (Frame 105). It is apparent that side A has the highest intensity value by looking at **Figure 6.3-1** (100). The intensity is a slightly grey consistent color. In **Figure 6.3-6**(105) there is a gradient in the intensity of side A: the top is completely white due to the LED washout, while the bottom is the grey tone that reflects the actual intensity of side A. This effect continues to deteriorate the ability to process the data as the 3D imaging sensor gets closer to the docking station.

The hardware is commercial off the shelf and a more robust version of the 3D imager is currently available with similar range and resolution. This hardware has

worked well in previous research efforts, such as identifying walls or other larger planar surfaces. It is effective for this research because the 3D point cloud data and the 2D intensity data can be exploited to "observe" the docking station. However, improved resolution and range as well as better calibration techniques would significantly increase the utility of such a sensor for the rendezvous and docking problem using the methods proposed in this thesis.

6.4 Simulated Results

Since no truth reference data was available for our data collection effort, a simulation was used to validate the algorithms in subsystem three. This simulation based evaluation uses known input values to confirm the implementation in Section 6.2 outputs correct results. Section 6.3 presents results using 3D imaging sensor data inputs that are within the range of expected results, although there is no way to determine the results are exact. The simulation confirms input data yields precise outputs and experiments with varying magnitudes of Gaussian noise to determine the sensitivity of the algorithm to erroneous inputs. The simulation uses simulated data instead of the outputs from subsystems one and two. The simulation implements a main simulation utility that defines simulated data and then calls the *find_orientation* code described in Section 6.2. The simulation input requires world frame data (relative to dock), while the algorithm expects body frame data (relative to camera) as shown in **Figure 6.4-1**. The simulation converts the data from the world frame to the body frame and initiates the *find_orientation* algorithm.

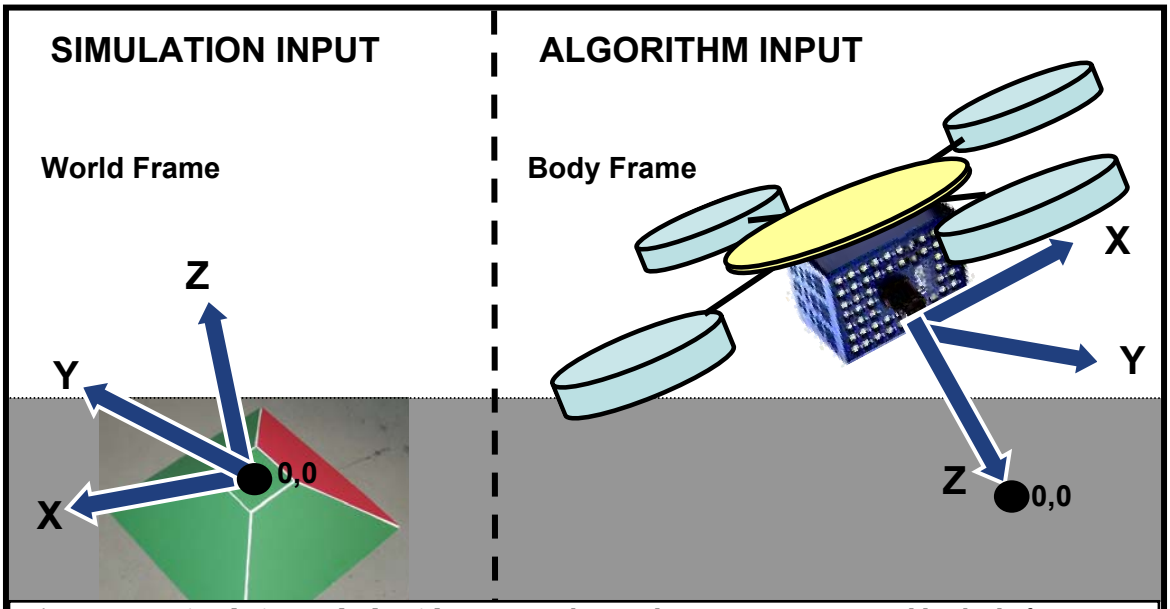


Figure 6.4-1. **Simulation and Algorithm Input.** The simulation must convert world to body frame since simulation inputs are in the world frame while the find_orientation algorithm requires body frame inputs.

First, world frame data is selected including the X, Y, and Z centroid coordinates, Euler angles and normal vectors. These inputs define the camera's position relative to the dock (expressed in the world frame attached to the docking station). The sketches in **Figure 6.4-2** show two docking station views in the world frame. This illustration also shows the centroid inputs ($\mathbf{r}_{w,i}$) and Euler angles (φ_i, θ_i, ψ_i) that are selected as inputs to the simulation. These inputs simulate translational motion by defining a path and orientation from the UAV to the docking station in the world frame. The normal vectors ($\mathbf{n}_{w,i}$) are defined using the same world frame views in **Figure 6.4-2**.

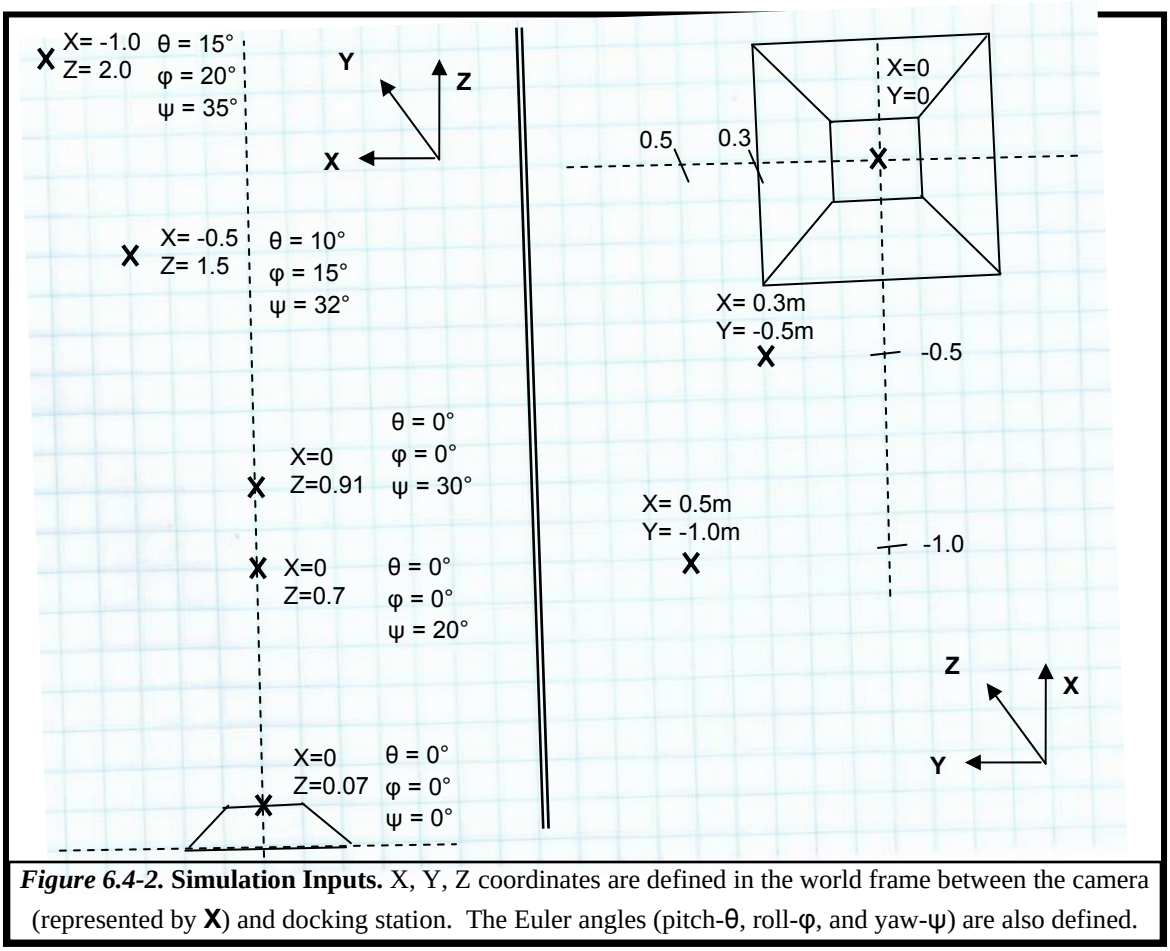


Figure 6.4-2. **Simulation Inputs.** X, Y, Z coordinates are defined in the world frame between the camera (represented by **X**) and docking station. The Euler angles (pitch-θ, roll-φ, and yaw-ψ) are also defined.

The world frame and navigation frame have opposing z-axes. The world frame positive z-axis points up, while the navigation frame z-axis points down. Following the right hand rule, the x and y axis are also switched between the two frames. The world to navigation conversion is straightforward and is shown in the middle of **Figure 6.4-3**. The conversion matrix changes the sign on the Z-axis and switches the X and Y axis. The need for this conversion is illustrated on the right and left side of **Figure 6.4-3**. The centroids and normal vectors in the world frame are multiplied by the world-to-navigation DCM to obtain centroid and normal vectors in the navigation frame as follows:

$$\mathbf{n}_{n,i} = C_w{}^n \mathbf{n}_{\mathbf{w},i} \qquad \mathbf{r}_{n,i} = C_w{}^n \mathbf{r}_{\mathbf{w},i}$$

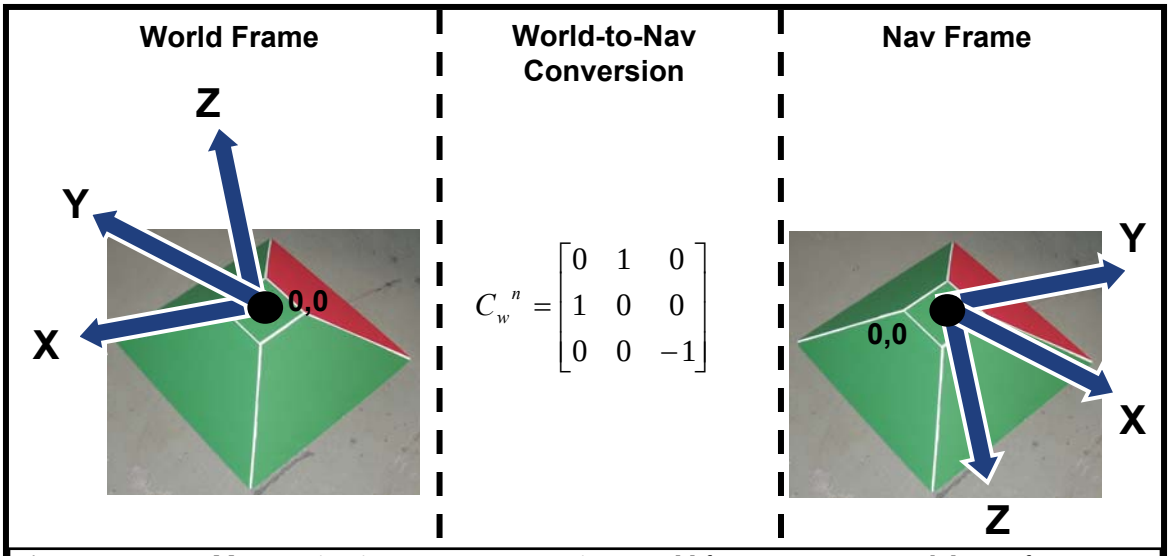


Figure 6.4-3. **World to Navigation Frame Conversion.** World frame Z-axis is up, while nav frame Z-axis is down. The conversion matrix captures this transformation including the Z-axis conversion and the associated X, Y axis switch due to right hand rule.

Next, the converted navigation frame is multiplied by a standard navigation to body frame Direction Cosine Matrix (DCM). The camera's roll, pitch and yaw Euler angles from **Figure 6.4-2** are input into the standard aerospace XYZ nav to body Direction Cosine Matrix ($C_n{}^b$) shown in the following equations.

$$C_n{}^b = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix}$$

The centroids and normal vectors in the navigation frame are multiplied by the $C_n{}^b$ to yield body frame centroids and normals that are input into the subsytem three algorithm.

$$\mathbf{n}_{b,i} = C_n^{\ b}\mathbf{n}_{n,i} \qquad \mathbf{r}_{b,i} = C_n^{\ b}\mathbf{r}_{n,i}$$

Recall from Section 6.2 that the orientation algorithm translates these body frame inputs into a translation (X,Y,Z world frame inputs) and Euler angles. These orientation algorithm outputs should match the inputs from **Figure 6.4-2** above. The simulation was successful and the inputs were identical to the outputs.

Once the inputs were identical to the outputs, Gaussian noise was applied to the inputs to determine the function's sensitivity to erroneous inputs. The Gaussian noise is added to the inputs after they are converted to the body frame and before they are sent to the find_orientation function. Other noise models may be considered based on further investigation of the error distributions of the outputs of subsystem one and two.

In subystem three, the plane centroids were originally used in Horn's algorithm to compute rotation. However, the estimates of the centroid locations from subsystem two are very prone to systematic errors due to occlusions or incomplete extraction of the complete planar surface. When these inputs were ran with the noise's standard deviation varying from 0.000001m to 0.1m the error was very high. The results are shown in **Figure 6.4-4** below. The exact threshold is not yet defined, but conceptually this error is too great. For example, if one of the planes is partially out of view from the camera, this could shift the centroid about 0.03m. According to the Gaussian noise samples, this would cause over a 93% error, which is unacceptable.

Gaussian Noise	Error Run 1	Error Run 2	Error Run 3	Error Run 4	Error Run 5	Error Average
0.000001	1%	1%	0%	0%	0%	0%
0.00001	1%	1%	2%	1%	1%	1%
0.0001	4%	8%	9%	6%	6%	7%
0.001	21%	32%	17%	22%	16%	22%
0.01	132%	112%	38%	70%	112%	93%
0.1	110%	213%	194%	135%	191%	169%

Figure 6.4-4. **Simulation Rotation Error with Gaussian Noise using Centroids.** The simulation rotation output percent error is calculated for Gaussian noise varying with a standard deviation of 0.1 to 0.000001 over 5 runs.

Since the plane centroid error was too high, the subystem three algorithm was modified. The normal vectors are now used to compute rotation instead of the centroids, which is the method documented earlier in this chapter. The errors were reduced as shown in **Figure 6.4-5**. Even if the input errors are an order of magnitude higher than those expected for the centroids, the normals only yield a 34% error. Note that the Gaussion noise was uncorrelated between the x, y and z components, which is more than likely not a valid assumption. Analysis of the covariance matrix of the normal vectors is outside the context of this thesis.

Gaussian Noise	Error Run 1	Error Run 2	Error Run 3	Error Run 4	Error Run 5	Error Average
0.000001	0%	0%	0%	0%	0%	0%
0.00001	0%	0%	0%	0%	0%	0%
0.0001	0%	1%	1%	0%	1%	1%
0.001	3%	3%	1%	4%	1%	2%
0.01	13%	12%	9%	16%	9%	12%
0.1	31%	30%	50%	45%	47%	41%

Figure 6.4-5. **Simulation Rotation Error with Gaussian Noise using Normals.** The simulation rotation output percent error is calculated for Gaussian noise varying with a standard deviation of 0.1 to 0.000001 over 5 runs.

Applying varying levels of noise to the simulation introduces the function's sensitivity to erroneous inputs, and the exact error threshold will be determined once the control laws are developed. The simulation is advantageous because it compares different approaches and arrives at the best solution, in this case using normals. The simulation proves that the subsystem three algorithm yields correct results, which allows future research to focus on recommendations in Chapter 7 to yield better data from subsystems one and two.

CHAPTER 7: OPPORTUNITIES FOR TECHNOLOGICAL ADVANCEMENT

This effort enhances and advances previous research, while identifying new areas for future research. Various lessons were learned from the research effort described in this thesis.

Three individual components were successfully integrated together, and important ideas were collected that can be applied to future research. The rest of this section focuses on each component and the integrated system to capture lessons learned and suggestions for future research in this area.

7.1 Technological Advancement of Components

7.1.1 Docking Station.

The docking station shape was originally selected because it takes advantage of previously developed software's capability to detect planes from a 3D imaging sensor 3D point cloud. As discussed in Chapter 3 and depicted in **Figure 7-1** the shape is a square based pyramid with the point removed so the top surface is a square. This design was chosen because it has five planar surfaces that can be identified, but the design could be optimized based on information learned in this research.

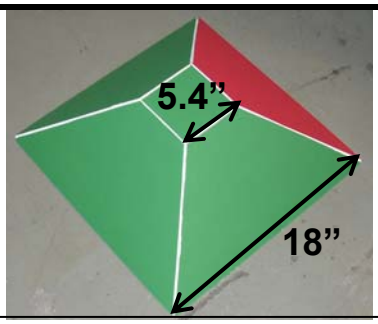


Figure 7-1. **Docking Station Shape.** The original shape was selected because it has 5 planar surfaces

The current shape's top planar surface is significantly smaller than the side planar surfaces, which is undesirable. It poses a problem when modifying the code's parameters to identify small or large planes. When the parameters are set to identify smaller planes the top of the docking station is accurately identified, but the sides are not. Furthermore, many false planar surfaces are detected in that case. However, when the parameters are set to identify larger planes the sides are accurately identified, but the top is not. Fortunately, this does not hinder the end results since Horn's algorithm requires a minimum of three planes, and the three sides are consistently identified when the parameters are set to identify larger planes. But it could improve the observability of the translation estimation algorithm shortly alluded to in Chapter 6.

Increasing the size of the top planar surface as shown in **Figure 7-3** would be advantageous for many reasons. As the UAV gets closer to the dock, the top is the easiest planar surface to detect. Also, since three planes are required and only three planes are currently available, if one is inaccurately identified, the output is slightly erroneous. Having an extra plane as a backup would help mitigate this risk. The extra plane could also take advantage of Horn's algorithm's ability to use input from more than three vectors, further reducing the error in the estimates.

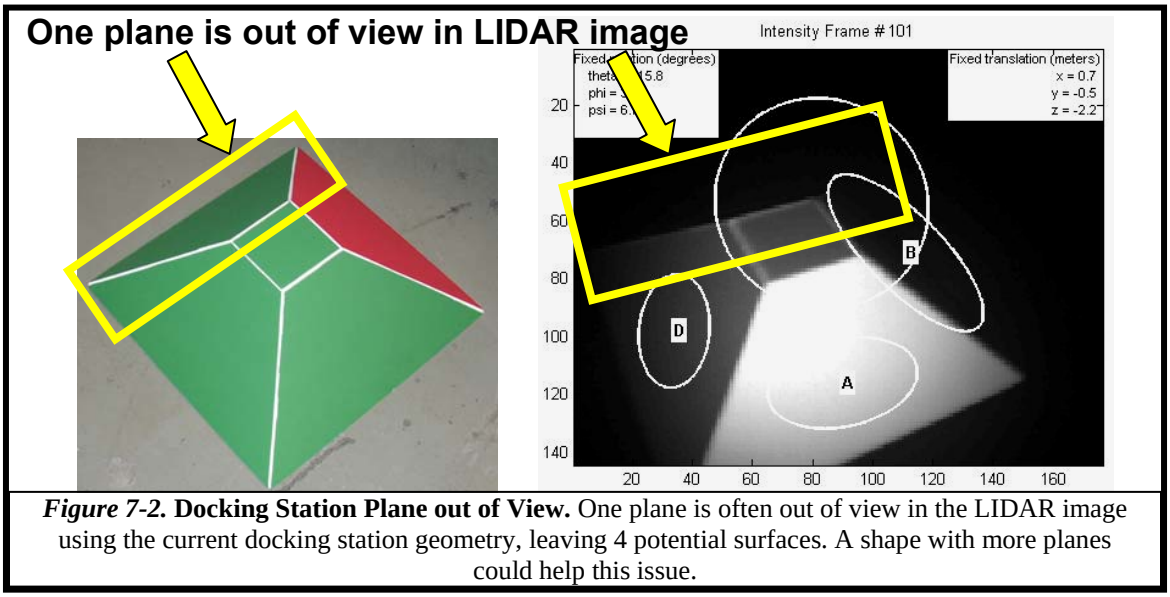


Figure 7-2. **Docking Station Plane out of View.** One plane is often out of view in the LIDAR image using the current docking station geometry, leaving 4 potential surfaces. A shape with more planes could help this issue.

A second drawback of the current docking station geometry is that one side is often completely out of view of the camera as shown in the 3D imaging sensor picture in **Figure 7-2**. A honeycomb based pyramid shape would allow more planar surfaces in the UAV's field-of-view without the need to improve the 3D imaging sensor hardware. The honeycomb based pyramid could either be a hexagon as pictured in **Figure 7-3** or an octagon, both could be tested to determine the better shape. The additional planar surfaces in this geometry carry the same advantages as previously discussed; it would help mitigate the risk of erroneous results from inaccurate plane identification and it would use the extra data to further reduce estimation errors.

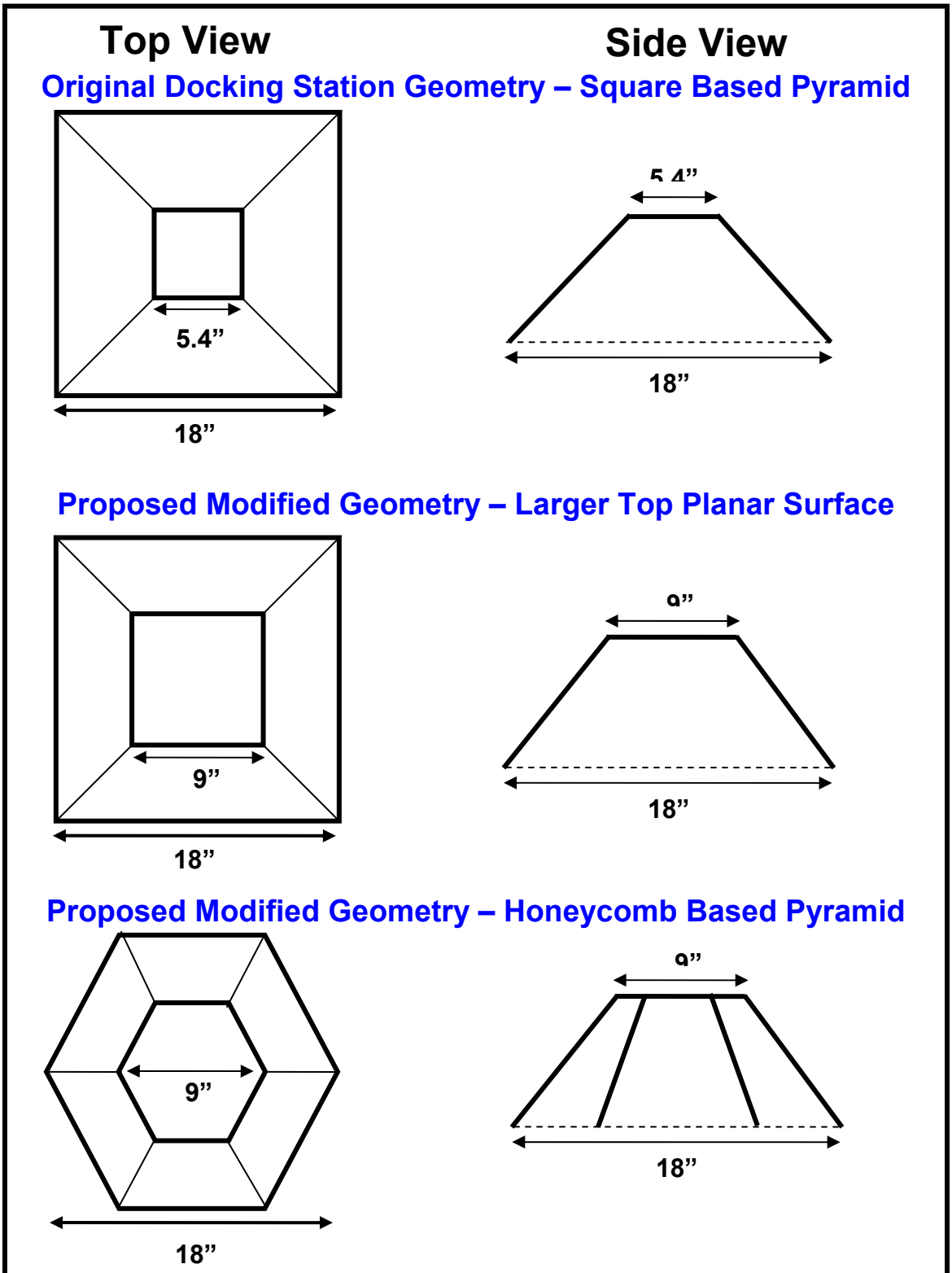


Figure 7-3. **Potential Docking Station Geometries.** The first geometry is the current one. The second geometry increases the top planar surface recognition, the third increases the number of planar surfaces.

A third overlook is the mockup size relative to the 3D imaging sensor detection capability. The docking station scale was originally selected to allow portability and flexibility in the UAV docking location. Since the previously developed software application detected larger planar surfaces it did not work as well with the docking station's reduced planar surfaces. Subsystem one improved the software's detection of smaller planar surfaces, as covered in Chapter 4. This improvement enables a subset of data to be detected, although it could be more effective across the entire data set with larger planar surfaces.

A larger scale docking station mockup can be constructed to temporarily relieve the issue of mockup size without the need to improve the 3D imaging sensor hardware. This would ultimately decrease portability, but since it will more than likely be mounted on a larger vehicle that would not be a problem. Improved resolution in the camera hardware, would allow for the docking station to again be scaled back to a portable size.

Adding retro-reflectors to the docking station could also improve docking station detection and identification. Corner reflectors at points on the docking station where planes intersect could provide good reference points to use in the algorithms. This concept is commonly used in space applications. For example, Blais used retro-reflectors in his research to track targets using a laser [3].

The purpose of improving the shapes is to allow three vectors to be more consistently available, which requires testing the code from subsystems one and two rather than the orientation algorithm from subsystem three. It is difficult to accurately simulate the subsystems one and two because the real aspect that requires testing is the 3D imaging sensor quality and data output sufficiency to identify the features of each proposed docking station.

The recommended path forward for future research of the docking station shape is to create mockups of the new shapes and record new data sets with the new shapes and retro-reflectors. Running it through the code developed for the first two subsystems will determine whether the data is improved and if the sides can be consistently identified. Improvements to the other components may need to be explored if optimizing the docking station shape is not successful.

7.1.2 Data Processing and Orientation Software.

The software development from subsystem one has been modified and the software for subsystem two has been developed as discussed in Chapters 4 and 5. This software detects smaller planar surfaces on better quality data subsets. The algorithms are more robust, although this code should be analyzed in the future to identify additional opportunities to enable a more effective solution across the entire data set. This is a critical portion of the system because the orientation algorithm defined in subsystem three is highly dependent on the outputs from subsystems one and two.

Additionally, this software requires a specific docking station. It assumes the docking station has a specific shape, one side of the docking station is a different color, and the area surrounding the docking station is of lower intensity than the docking station. The algorithm can be customized in the future to fit other docking station colors and shapes if desirable.

The complexity of the code is another area for improvement. It was not accounted for during programming in this research, but the code could be optimized in the future for shorter run times and more efficient processing of larger data sets.

7.1.3 3D imaging sensor Hardware.

The final component that could be improved in the future is the 3D imaging sensor hardware, as referenced throughout this section. Increased operational range (only 7m now), increased resolution (only 176x144 now), and better calibration, could lessen the burden of the software processing components, although a cost/benefit analysis should be calculated to determine the most beneficial future research efforts. Improving the other components may span over multiple research efforts, but the hardware cost would be negligible. On the contrary, replacing the 3D imaging sensor would require minimal research effort, but the hardware cost would be significant. If the budget isn't available for a more precise camera the other components can be improved including the docking station shape and software algorithms.

7.1.4 UAV

The UAV is a component that is not yet integrated but will be important in the future. The Ohio University UAV research is currently a separate effort, but must mature in parallel with this research in order to eventually accomplish the goal of a UAV autonomously docking. Communication between the research projects should continue in the future to ensure both efforts include the required capabilities to interact in the future.

7.2 Technological Advancement of Integrated System

7.2.1 Incorporating Additional Components and Reducing Current Constraints

The integrated system can technologically advance by incorporating additional components and relieving current constraints to expand the basic solution, operational

scenario and application. For example, the system is currently constrained because it assumes a fixed docking station. In the future, the relative and fixed orientation could be fused to detect a docking station geometry that is in motion. This could expand the operational scenario to include the UAV docking to a docking station on a moving ground vehicle.

Future systems could also incorporate additional components. Enemy UAV interception is a risk in military applications. For this solution, the enemy could recreate the docking station geometry to trick the UAV to land on a false dock. An additional component could be installed in the dock that the UAV can detect, such as RFID tags or other unique identifiers. The UAV can use these unique identifiers to detect if the docking station is legit. This mitigates the risk of the UAV landing on a false dock and the enemy apprehending the UAV.

Future integrated system advancements could use additional sensors for the UAV rendezvous and docking. Fusing the 3D imaging sensor with other sensors would further minimize errors. NASA's Marshall Space Flight Center (MSFC) Flight Robotics Laboratory (FRL) is conducting research in this area as they detail in the article "Multi-Sensor Testing for Automated Rendezvous and Docking." [34]

7.2.2 Expanding Current Application.

The current research application can be expanded to apply 3D imaging sensor and its capabilities beyond the scope of rendezvous and docking on the earth. 3D imaging sensors have advantages over other navigation aids because it is not susceptible to jamming like GPS and does not experience drift like INS. It could eventually be used to complement those navigation aids and provide more robust navigation through sensor integration for UAV phases of flight other than precision docking.

Furthermore, the basic solution is currently aimed at UAV rendezvous and docking on the earth's surface. This is applicable to military or commercial purposes, but there is also a need for autonomous docking in space, as discussed in Chapter 1. The overall concept of operations could be expanded to include space applications and the integrated system could eventually be customized for space operation. A 3D imaging sensor has recently been used in NASA's Phoenix MARS Lander "to detect dust, clouds and fog" as described on NASA's website [35]. Other related applications that expand upon autonomous navigation, such as geomapping, could also be explored.

7.2.3 Exploring Future Applications.

Adjacent markets that build off of the core competencies from this research can be explored. The integrated system that detects distance and angles from the 3D imaging sensor to a defined geometric object can be applied to other applications outside of UAVs and navigation.

One future market involves household applications. Autonomous navigation has already been introduced into the market by iRobot, which offers autonomous vacuums (Roomba and Dirt Dog), pool cleaners (Verro), floor washers (Scooba), and gutter cleaners (Looj) [36]. If applied in a cost effective manner, 3D imaging sensors could eventually improve accuracy of existing systems or be used in new systems for household applications.

Other markets could take advantage of the 3D imaging sensor's ability to create a 3D image and detect precise distances. 3D cameras are used in video game or movie making to reduce time to create 3D graphics. 3D imaging sensors are one of the technologies that were recently used to make music artist Radiohead's recent music video [37]. In the future this technology could also enable sports officials and broadcasters to

determine exact position of a ball rather than guessing based on playback of varying angles of 2D images.

These are just a few of the many possibilities for 3D imaging sensors. As future research matures the future applications should be revisited to define the most feasible efforts.

CHAPTER 8: CONCLUSIONS

In summary, a relative position and orientation estimation method in support of autonomous rendezvous and docking has been developed in this research. This effort addresses an integrated docking system including a 3D imaging sensor, a fixed docking station geometry and data processing algorithms. This integration achieves the research goal and provides the UAV location and orientation with respect to a fixed docking station.

The 3D imaging sensor collects data that is processed using the data processing algorithms. The fixed docking station geometry is captured in the 3D imaging sensor data and its geometry is used as an input to the UAV orientation algorithm.

Previously developed algorithms for planar surface extraction were analyzed and various weak points were identified and removed. Furthermore, a plane identification and association algorithm has been developed to ensure the docking station planes are identified in a specific order consistently throughout the data set. Both of these algorithms feed input to the UAV position and orientation estimator, which applies Horn's algorithm to determine the rotation and translation from the UAV to the fixed docking station geometry. These three algorithms are applied to the docking station data set and the orientation of the 3D imaging sensor is accurately identified with respect to the docking station, which is proven using simulated data.

This effort proves the rendezvous and docking concept and uncovers many topics to expand upon in future research. In conclusion, this research provides an integrated solution to the rendezvous and docking problem by defining the orientation between the UAV and the dock.

REFERENCES

[1] *FY2009-2034 Unmanned Systems Integrated Roadmap*. Rep. Department of Defense, 2009. Web. 16 Oct. 2009. <http://www.acq.osd.mil/uas/docs/UMSIntegratedRoadmap2009.pdf>.

[2] United States. Department of Energy. Office of Nuclear Energy, Science, and Technology. *DOE/NE-0071 Nuclear Power in Space*. Department of Energy. Web. 24 Oct. 2009. <http://ocw.mit.edu/NR/rdonlyres/4C7E6DFE-ACE0-437A-B397-7F7A8AFAF549/0/nuc_pow_space.pdf>.

[3] Blais, F., J.A. Beraldin, L. Cournoyer, S. El-Hakim, J. Domey, M. Rioux, I. Christie, R. Serafini, G. Pepper, S.G. MacLean, and D. Laurin. *Target Tracking, Object Pose Estimation, and Effects of the Sun on the NRC 3-D Laser Tracker*. Proc. of Sixth International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2001), St-Hubert, Quebec, Canada. National Research Council of Canada, 2001. Print.

[4] "Draganflyer X4 Four Rotor UAV Helicopter Aerial Video Platform." *Draganfly.com*. Draganfly Innovations Inc., 2009. Web. 24 Oct. 2009. <http://www.draganfly.com/uav-helicopter/draganflyer-x4/index.php>.

[5] "RoboCup 2009 | robocup rescue." *RoboCup 2009 | home*. Web. 24 Oct. 2009. <http://www.robocup2009.org/21-0-robocup-rescue.html>.

[6] Surmann, Hartmut, Rainer Worst, Matthias Hennig, Kai Lingemann, Andreas Nuechter, Kai Pervoelz, Kiran Raj Tiruchinapalli, Thomas Christaller, and Joachim Hertzberg. *RoboCup2004, Rescue Robot League Competition*. Proc. of RoboCupRescue - Robot League Team KURT3D, Germany, Lisbon, Portugal. Sankt Augustin, Germany, 2004. Print.

[7] Stepaniak, Michael. "A Quadrotor Sensor Platform." Diss. Ohio University, 2008. Print.

[8] Passner, Jeffrey E., Robert E. Dumais, Jr., Robert Flanigan, and Stephen Kirby. *Using the Advanced Research Version of the Weather Research and Forecast Model in Support of ScanEagle Unmanned Aircraft System Test Flights*. Rep. no. ARL-TR-4746. Army Research Laboratory, Mar. 2009. Web. 25 Oct. 2009. <http://www.arl.army.mil/arlreports/2008/ARL-TR-4575.pdf>.

[9] Fenton, Ronald C., R. Rees Fullmer, and Robert T. Pack. "Design of lidar-based sensors and algorithms for determining the relative motion of an impaired spacecraft." *Proceedings of SPIE*. Vol. 5778. Bellingham, WA: SPIE, 2005. 809-18. Print.

[10]Subbarao, Kamesh, and Atilla Dogan. "Intelligent Integrated Sensor, Control and Maneuver Mapping Architecture For Autonomous Vehicle Rendezvous and Docking, AIAA 2004-6573." *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit*. Chicago, Illinois. American Institue of Aeronautics and Astronautics Inc., 2004. Print.

[11] Lytle, Alan M., Itai Katz, and Kamel S. Saidi. "Performance Evaluation of a High-Frame Rate 3D Range Sensor for Construction Applications." *22nd International Symposium on Automation and Robotics in Construction, ISARC 2005 - September 11-14, 2005, Ferrara (Italy)*. Print.

[12] Nalepka, Joseph P., and Jacob L. Hinchman. "AIAA 2005-6005 Automated Aerial Refueling: Extending the Effectiveness of Unmanned Air Vehicles." *Proceedings of the AIAA Modeling and Simulation Technologies Conference and Exhibit*. San Francisco, California. 2005. Print.

[13] Soloviev, Andrey, Dustin Bates, and Frank Van Graas. *Tight Coupling of Laser Scanner and Inertial Measurements for a Fully Autonomous Relative Navigation Solution*. Tech. Ohio University. Print.

[14] Uijt de Haag, Maarten, Don Venable, and Mark Smearcheck. *Integration of an Inertial Measurement Unit and 3D Imaging Sensor for Urban and Indoor Navigation of Unmanned Vehicles*. Tech. Print.

[15] Scott, Marion W. Range Imaging Laser Radar. The United States of America as represented by the Department of Energy, Washington, D.C., assignee. Patent 4,935,616. 19 June 1990. Print.

[16] Ulich, Bobby L., and Kent Pflibsen. Range Finding Array Camera. Kaman Aerospace Corporation, Bloomfield, Conn., assignee. Patent 5,200,793. 6 Apr. 1993. Print.

[17] Stann, Barry, William C. Ruff, and Zoltan G. Sztankay. Scannerless Ladar Architecture Employing Focal Plane Detector Arrays and FM-CW Ranging Theory. The United States of America as represented by the Secretary of the Army, Washington, D.C., assignee. Patent 5,877,851. 2 Mar. 1999. Print.

[18] Rappaport, Saul, Geert Wyntjes, and Orr Shepherd. Three-Dimensional Imaging System. Visidyne, Inc, Burlington, Mass., assignee. Patent 6,002,423. 14 Dec. 1999. Print.

[19] Yahav, Giora, and Gavriel Iddan. Three Dimensional Camera. 3D Systems Ltd., Yokneam, Israel, assignee. Patent 6,100,517. 8 Aug. 2000. Print.

[20] Muguira, Maritza R., John T. Sackos, Bart D. Bradley, and Robert Nellums. Range Determination for Scannerless Imaging. Sandia Corporation, Albuquerque, N.Mex., assignee. Patent 6,088,086. 11 July 2000. Print.

[21] Bulyshev, Alexander, Diego Pierrottet, Farzin Amzajerdian, George Busch, Michael Vanek, and Robert Reisse. *Processing of 3-Dimensional Flash Lidar Terrain Images Generated From an Airborne Platform*. Tech. no. 20090016357. NASA Langley Research Center, 13 Apr. 2009. Web. 29 Nov. 2009. <http://ntrs.nasa.gov/search.jsp?R=606348&id=1&as=false&or=false&qs=Ntt%3Dflash%2Blidar%26Ntk%3Dall%26Ntx%3Dmode%2Bmatchall%26Ns%3DHarvestDate%257c1%26N%3D0>.

[22] Oggier, Thierry, Bernhard Buutgen, Felix Lustenberger, Guido Becker, Bjorn Ruegg, and Agathe Hodac. *SwissRanger SR3000 and First Experiences based on Miniaturized 3D-TOF Cameras*. Rep. Web. 25 Oct. 2009. <http://www.mesa-imaging.ch/pdf/Application_SR3000_v1_1.pdf>.

[23] "Introduction & Key Features." MATLAB. 16 Aug. 2008 <http://www.mathworks.com/products/matlab/description1.html>.

[24] Venable, Donald T. "Implementation of a 3D Imaging Sensor Aided Inertial Measurement Unit Navgi." Thesis. Ohio University, 2008. Print.

[25] Otsu, N., "A threshold selection method from gery-leevl histogram," *IEEE Transactions of Systems, Man and Cybernetics*, Vol. SMC-8, pp. 62-66, 1978.

[26] Kittler, J. and J. Illingworth, "Minimum Error Thresholding," *Pattern Recognition*, Vol. 19, No. 1, pp. 41-47, 1987.

[27] "Convex Hull of a 2D Point Set or Polygon." Geometry Algorithm Home. Web. 15 Oct. 2009. <http://softsurfer.com/Archive/algorithm_0109/algorithm_0109.htm>.

[28] "Building a Convex Hull." Web. 15 Oct. 2009. <http://marknelson.us/2007/08/22/convex/>.

[29] Horn, Berthold K. P. "Closed-form solution of absolute orientation using unit quaternions." *Journal of the Optical Society of America* 4 (1987): 629-42. Print.

[30] Berthold, Horn K.P. *Some Notes on Unit Quaternions and Rotation*. Tech. 2001. Print.

[31] Braasch, Michael S. *Fundamentals of Inertial Navigation Systems*. Lecture Notes. 2005. Print.

[32] Mukundan, R. "Quaternions: From Classical Mechanics to Computer Graphics, and Beyond." *Proceedings of the 7th Asian Technology Conference in Mathematics 2002.* 97-106. Print.

[33] Hung, Faith, and Brian L. Stevens. *Aircraft control and simulation*. Hoboken, N.J: J. Wiley, 2003. Print.

[34] Howard, Richard T., and Connie K. Carrington. *Multi-Sensor Testing for Automated Rendezvous and Docking*. Rep. NASA Marshall Space Flight Center. Web.

[35] "NASA - Phoenix Lidar Operation Animation." *Nasa.gov*. 29 May 2008. Web. 13 Oct. 2009. <http://www.nasa.gov/mission_pages/phoenix/multimedia/10311.html>.

[36] "IRobot: Cleaning Robots." *IRobot.com*. 2009. Web. 13 Oct. 2009. <http://store.irobot.com/shop/index.jsp?categoryId=2804605>.

[37] "RADIOHEAD / HOUSE OF CARDS - Google Code." *Code.google.com*. 2009. Web. 13 Oct. 2009. <http://code.google.com/creative/radiohead/#the-making-of>.