Design and Implementation of a Relative State Estimator for Docking and Formation Control of Modular Autonomous Spacecraft

by

Nicholas R. Hoff iii

Submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of Master of Science at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY June 2007

© Massachusetts Institute of Technology 2007. All rights reserved.
Design and Implementation of a Relative State Estimator for Docking and Formation Control of Modular Autonomous Spacecraft

by

Nicholas R. Hoff III

Submitted to the Department of Aeronautics and Astronautics on May 25, 2007, in partial fulfillment of the requirements for the degree of Master of Science

Abstract

Modularity is a promising design concept for space systems. In a modular satellite, the individual subsystems would be broken down into physically distinct modules, which would then dynamically recombine into an aggregate vehicle. This could improve the flexibility and reusability of satellites, and could even enable some mission objectives which are not possible at all with monolithic vehicles. However, modularity requires that some additional new elements be included in the design that are not needed with a monolithic satellite. Two of these are a docking interface to allow modules to attach, and a position measurement system to allow modules to fly accurately in formation and dock with each other. These two additional elements are explored in this thesis. The central focus is on a relative state estimator based on an extended Kalman filter. The estimator is first presented theoretically, then the results of implementation and hardware testing are discussed. This thesis presents two main hardware applications for the estimator, both of which mirror prime space-based applications of modularity itself: docking and formation maintenance/reconfiguration.

Thesis Supervisor: David W. Miller
Title: Professor of Aeronautics and Astronautics
Acknowledgments

Parts of this work were funded by NASA MSFC under Award # 01-44-49-001. We are thankful for their support.

Several fellow graduate students in the Space Systems Lab at MIT were invaluably helpful during this work. Simon Nolet, on whose work most of my work is based, answered my endless questions about estimation on SPHERES and was extremely helpful throughout. Alvar Saenz-Otero was also invaluable in running tests on SPHERES and SPHERES-related hardware, and spent many hours with us working on the electronics for SWARM. My colleague Swati Mohan was an instrumental partner in the SWARM and SIFFT projects. Lennon Rodgers was a partner in the design and fabrication of the UDP, discussed in Chapter 2. My advisor David W. Miller provided technical suggestions and direction for this work, I am thankful to him for letting me work in his lab. The team at the Flight Robotics Laboratory at NASA MSFC was helpful with the SWARM and SIFFT testing described in Chapter 4, we thank them for letting us use their floor. Steve Sell and Chris Krebs of Payload Systems Inc. have been helpful in creating high-quality mechanical designs and fabricating high-quality hardware for us. The SPHERES project itself began as an undergraduate design course, and credit for the current SPHERES satellites goes to all the students who have worked on it over the years.

Lastly, I am so grateful to my mother for her endless love and support. It is the only thing that I can count on absolutely, and it has made everything possible.
To my grandfather

An engineer
Contents

1 Introduction ............................................................................................................. 15
   1.1 Motivation ......................................................................................................... 15
   1.2 Overview .......................................................................................................... 17

2 Universal Docking Port (UDP) .............................................................................. 19
   2.1 UDP Requirements ......................................................................................... 19
   2.2 UDP Design .................................................................................................. 20
   2.3 Estimation Hardware ...................................................................................... 24
   2.4 Conclusion ...................................................................................................... 25

3 Relative Estimator ................................................................................................. 27
   3.1 Architecture Choices ...................................................................................... 27
   3.2 Available Measurements ................................................................................ 28
   3.3 State Vector and Attitude Representation ...................................................... 30
   3.4 EKF Process - Linear Case ............................................................................. 31
   3.5 Nonlinearities .................................................................................................. 34
      3.5.1 Nonperiodic Measurements .................................................................. 35
      3.5.2 Nonlinear Measurements ....................................................................... 36
   3.6 Prefilter ........................................................................................................... 41
   3.7 Conclusion ........................................................................................................ 42

4 Results and Applications ....................................................................................... 43
   4.1 Testing of Ultrasound Receivers .................................................................... 43
4.2 Testing in Simulation ................................................. 47

4.3 Hardware ............................................................... 50
  4.3.1 SPHERES Satellites .............................................. 53
  4.3.2 Air Carriages .................................................... 55
  4.3.3 Avionics Stack .................................................. 56
  4.3.4 UDPs, Subapertures, and Other Hardware ...................... 57
  4.3.5 Flat Table ...................................................... 57

4.4 SWARM ................................................................. 58
  4.4.1 Setup .............................................................. 59
  4.4.2 Experimental Plan .............................................. 60
  4.4.3 Sample Docking Results ..................................... 61
  4.4.4 Overall Docking Results .................................... 64
  4.4.5 Reconfiguration Results ..................................... 65

4.5 SIFFT ................................................................. 65
  4.5.1 Hardware Setup ................................................ 66
  4.5.2 Estimation Configuration ................................... 66
  4.5.3 Test plan ....................................................... 67
  4.5.4 Results with Module 1 Fixed ................................. 69
  4.5.5 Results with Rotation of Module 1 .......................... 69

4.6 Conclusion ........................................................... 71

5 Future Work and Conclusions ....................................... 73

  5.1 Future Work ........................................................ 73
    5.1.1 UDPs use Beacons and Receivers ................................. 73
    5.1.2 Star Tracker Mode ............................................. 78
    5.1.3 Full Global .................................................... 82
    5.1.4 Use Thruster Commands and Accelerometers ..................... 83

  5.2 Conclusion ........................................................ 85

A Selected Code Segments ........................................... 87

  A.1 Overall Estimator Cycle ........................................... 87
A.2 pads_stateUSBeaconDetPvarEKF() ........................................ 89
A.3 pads_hgenUSBeaconUpdatePvarEKF() ........................... 91
A.4 Qrot() ................................................................. 93

B Quaternion Attitude Representation .................................. 95
  B.1 Quaternion Composition and Properties ...................... 95
  B.2 Quaternion Propagation ........................................ 98
List of Figures

2-1 CAD drawing of a mostly-assembled UDP .......................... 21
2-2 CAD drawing of the UDP locking mechanism .................... 22
2-3 CAD drawing of the UDP ring counterrotation mechanism ...... 24
2-4 CAD drawing of the UDP with metrology rings in place .......... 25
2-5 A picture of the front face of a fully assembled UDP .......... 26

3-1 Timing diagram for ultrasound and infrared sensors ............. 30
3-2 Update/Propagate diagram of an EKF ............................. 32
3-3 Basic cycle of a linear Kalman filter .............................. 33
3-4 Propagate/Update diagram of an EKF .............................. 35
3-5 Diagram showing nonlinearity in beacon-receiver separation measurements ........................................ 36
3-6 Notation of beacon and receiver positions in chaser and target reference frames ............................................. 37

4-1 60 seconds of raw data from one beacon-receiver pair .......... 44
4-2 60 seconds of raw data from one beacon-receiver pair, zoomed in . 45
4-3 Histogram showing all $\hat{T}_{ijr}$ from beacon-receiver pairwise tests .......... 48
4-4 Relative estimator simulation, $\hat{x}_0 = x_0$ .......................... 51
4-5 Relative estimator simulation, $\hat{x}_0 \neq x_0$ .......................... 52
4-6 A single SPHERES satellite ........................................... 53
4-7 The SPHERES expansion port ........................................ 54
4-8 A single puck air carriage .......................................... 55
4-9 The avionics stack .................................................. 56
4-10 The telescope aperture ........................................ 57
4-11 The flat table in the MIT SSL .............................. 58
4-12 The SWARM tug module ..................................... 59
4-13 The SWARM subaperture module ......................... 60
4-14 Data from a docking maneuver at MSFC ............... 62
4-15 Data from a docking maneuver at MSFC ............... 63
4-16 The SIFFT hardware setup .................................. 66
4-17 SIFFT estimation configuration and UDP field-of-view 68
4-18 A SIFFT formation reconfiguration test ................ 70
4-19 A SIFFT formation reconfiguration test with rotation 72
5-1 Notation of beacon and receiver positions in chaser and target reference frames .................................. 74
5-2 A possible measurement reordering ....................... 76
5-3 An external beacon ........................................... 79
5-4 Beacon/receiver arrangement for the star tracker mode 79
5-5 Position/attitude coupling with single beacon star tracker ............... 81
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Summary of results from single beacon-receiver pair test.</td>
<td>46</td>
</tr>
<tr>
<td>4.2</td>
<td>Summary of results from all beacon-receiver pairs.</td>
<td>47</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

The exploration, development, and use of space is limited by the complexity of operating in such a remote and fault-intolerant environment. Large spacecraft are often brought down by a single small failure. Additionally, the increasingly ambitious science objectives and exploration goals are driving spacecraft to larger and more complex designs, further increasing their cost and risk. A modular spacecraft design could ameliorate many of these difficulties. For example, if a spacecraft consisted of separate propulsion, communication, and payload modules, then a damaged subsystem could be replaced without ending the entire mission. Modularity in spacecraft design adds additional complications, though, not found in monolithic designs. One of these additional elements is a subsystem by which different modules can measure each other’s relative positions so that the whole formation can behave in a coherent and useful way. This relative measurement system is the focus of this thesis.

A modular space system would operate differently from most conventional monolithic space systems. Under a modular approach, the system would be broken down into convenient subsystems, each of which would be identified as an individual module. A common and sensible way to make these divisions is by functionality. In a typical satellite system for example, there may be a communication module, a battery module, a thruster module, a science payload module, etc. Manned systems may
have other elements like a food storage module or waste jettison module.

Interfaces between modules, both 'hard' and 'soft,' must be well defined. Perhaps modules do not need to physically connect at all in some systems. In others, standardized connecting hardware is required. The electronic and data connections between modules are equally important. Modules can communicate by physically joining connectors or they can communicate wirelessly. This communication and other common actions must be well coordinated.

Some of the benefits of modularity could be realized simply by dividing the vehicle into modules for purposes of fabrication and construction, and then assembling them into a monolithic vehicle for operation. The Space Shuttle Main Engine (SSME) is constructed in this manner. The H\textsubscript{2} and O\textsubscript{2} turbopumps, some of the controller electronics, parts of the nozzle, the injectors, and some other components can be fabricated, repaired, replaced, or possibly even redesigned in isolation, then reintegrated with the rest of the SSME [20].

The full benefits of modularity, however, come from physically separating the modules, even during operation. Each module would be a physically distinct unit; the modules would then combine and recombine in space in different numbers and arrangements to produce overall functionalities. As an example, consider a hypothetical Earth observing satellite. The modules would be packed in the payload fairing of a launch vehicle, then released upon reaching orbit. Perhaps a sensor module would measure the deployment velocities of all the other modules, then a tug module would collect them all. Then, an assembler module could find the observation payload, connect it to a battery module, a computer module, and an antenna module and it could begin taking observations. Meanwhile, the tug could keep all the other modules nearby in some storage arrangement. This could all be controlled wirelessly by a computer module. Resupply of battery or fuel modules could periodically be sent up and the assembler modules could use them as appropriate. Perhaps even a new payload module could be launched to alter the mission [7].

Building a modular space system, as opposed to a monolithic space system, necessitates some additional hardware and software. First, a docking interface which allows
modules to connect and disconnect is required. Second, the modules must be able to accurately measure their positions relative to each other in order to successfully dock and reconfigure. Both of these additions will be considered in this thesis.

1.2 Overview

The focus of this thesis is on the design, implementation, and testing of a relative state estimator for modular spacecraft.

A docking interface is a critical hardware component of both a modular satellite system in general and the relative estimator presented in this thesis specifically. Chapter 2 quickly presents the requirements and design of the UDP, including the sensing hardware.

The theory and implementation of the relative estimator is presented in Chapter 3. The estimator is an extended Kalman filter (EKF) which takes measurements from various kinds of sensors on the modules and maintains an updated estimate of the relative state of each module. At first, the theoretical basics of the Kalman filter are presented assuming that the measurements are linear functions of the state variables. Then, the analysis is expanded to nonlinear measurements, which is the case with these sensors. The analysis is further expanded to include several other nonlinearities and complications, to finally arrive at the relative estimator coded and used on the hardware in these projects.

The estimator is put to the test in Chapter 4. It begins with a low-level analysis of the reliability of the sensors themselves in Section 4.1, then moves to a test of the full estimator, running in simulation, in Section 4.2. The goal is to test the estimator on hardware and present physical data, but first, a description of the hardware elements to be used in the various tests is presented in Section 4.3. The relative estimator was tested in two main projects – SWARM (Synchronized Wireless Autonomous Reconfigurable Modules) and SIFFT (Synthetic Imaging Formation Flight Testbed). The performance of the estimator in each of these projects is shown in Sections 4.4 and 4.5, respectively.
Finally, there are several clear ways that the estimation system could be improved, and these are presented in the future work section of Chapter 5. Overall conclusions from the thesis are presented at the end of the chapter.

Some of the more critical or important code segments of the estimator are included and explained in Appendix A.
Chapter 2

Universal Docking Port (UDP)

In a modular satellite system that requires modules to physically connect and disconnect, one critical element is a docking interface. This chapter introduces satellite docking interfaces and presents the specific requirements and implementation of the UDP built for this project.

Depending on the specific application, a docking system may need to be rigid or flexible, structural or purely informational, autonomous or manual. It may need to allow transfer of fluids, heat, electricity, data, humans, etc. A docking interface could be as simple as a pure adhesive, it could be a manipulator on the end of an arm, or a port rigidly bolted to the structure of a module [9, 10, 14].

2.1 UDP Requirements

The docking interfaces for the SWARM project were designed and built under the following requirements [2]:

**Autonomously Dock and Undock** Modules must be able to connect and disconnect autonomously. This is the main function of the interface. The module itself must control the acquisition, approach, contact, locking, and any subsequent transfers, as well as the entire undock and departure sequence.

**Genderless** Any docking port must be able to dock to any other docking port. In
other words, the docking port must be 'universal.' This requirement eliminates the possibility of designing male and female docking ports.

**Transfer Mechanical Loads**  The interface must be structurally strong enough that when two modules are docked, they behave as a rigid body. With this requirement, thruster or ACS modules can translate and rotate the whole assembly.

**Transfer Electrical Power**  Modules must be able to share electrical power, so that battery modules can recharge at a charging station, or modules with low batteries can recharge at the expense of modules with full batteries. This function of the interface is not used in the work presented in this thesis, but it did impact the design.

**Provide Metrology Hardware**  Modules must have data with which to calculate each other's position, and it was decided that the UDP would be the location of the sensors that collect this data.

Another important property of the UDP is its docking tolerance. Both the position measurement system and the thruster system have some precision with which they can measure and control the position and attitude of a module. Therefore, when two UDPs are approaching to dock, they will probably not be perfectly aligned either in angle or position. The UDP must be able to successfully dock even with this initial misalignment in the orientation or position of the modules. This initial misalignment is called docking tolerance. There is not a formal requirement on the docking tolerance of the UDP; the final product has $\pm 2$ cm of translation tolerance and $\pm 40^\circ$ of angle tolerance, which is sufficient for these applications.

### 2.2 UDP Design

A CAD drawing of the UDP is shown in Figure 2-1. There is a pin and a hole on the front of the UDP; when two UDPs come together to dock, the pin from one goes into
Figure 2-1: A CAD drawing of a mostly-assembled UDP. For clarity, the drawing does not show the wrappings of the electromagnet or the metrology boards (shown later).

Design features on the front of the UDP include:

- **Electromagnet** An electromagnet encircles the front plate assembly. When two UDPs are coming together to dock, the electromagnets from each UDP attract to assist in the maneuver. The main purpose of this is to increase the docking tolerance.

- **Steel Face Plate Core** The core of the faceplate is made out of steel. This increases the mass of the UDP, but the steel's mild ferromagnetic properties multiply the effect of the electromagnet.

- **Power Tabs** There are two spring loaded copper tabs extending from the face plate of each UDP. By convention, the bottom one is ground and the top one
is power. Upon docking, the tabs from each UDP touch each other and make an electrical contact.

- **Chamfered Hole and Angled Pin Head** These increase docking tolerance.

The locking mechanism in the back of the interface is shown in Figure 2-2. Two counterrotating rings allow the pin head to enter, then rotate to close around it, as detailed in the figure.

Design features of the back of the UDP include:

- **Central Axle** This extends through the entire interface and helps the rings and motor shaft to be properly mounted and aligned.

- **Locking Rings** There are two of these rings in each interface, placed back-to-back. Irregular shaped holes are cut in each ring such that when the rings are ‘open’ there is a large hole, and when the rings are ‘closed’ there is a small hole. The sizes of these resulting holes are designed so that the pin from another interface can enter when the rings are open, but is locked in place when the rings are closed.

- **Drive Pin** To rotate the rings, a small pin moves through slots cut in both
rings. The slots are cut such that when the pin moves radially toward or away from the central axel, the rings open or close. The pin has a tapped hole and rides along the motor shaft.

- **Motor Shaft** The motor shaft is a strait threaded piece which drives the drive pin. The shaft is connected to the motor on one end. For stability, the other end is secured in place near the axle. The drive pin and motor shaft are shown in Figure 2-3.

- **Motor** A DC motor drives the motor shaft. The motor is secured to the back plate by a simple bracket.

- **Pin Sensor** During a docking, the UDP control board needs to know when the pin from another interface has come far enough in for the rings to begin closing. This is detected by an infrared optical interrupt switch. When the pin breaks the IR beam, the UDP control board will drive the motor which will close the rings.

- **Current Spike Sensor** The interface control board also needs to know when the rings have finished rotating and have grabbed the pin, so that it can turn off the motor. Fortunately, because of the construction of a DC motor, when the motor hits its stop and stalls meaning the rings have finished rotating, it will suddenly draw a large amount of current. A current spike sensor is built into the interface control board to detect these spikes and stop the motors at the correct time.

There is a circuit board (the UDP control board) which controls the functions of the interface and communicates with the module to which the UDP is attached. It controls the motor, the electromagnet, the power tabs, reads the pin sensor and the current spike sensor, receives commands from the module, and reports status to the module. There are also connections for signals from the metrology boards (discussed in Section 2.3). These signals are not processed on the interface control board, they are sent to the computer on board the module where the estimator runs.
Figure 2-3: This CAD diagram shows the mechanism which counterrotates the rings. It consists of the motor, drive shaft, pin, rings, and axle.

2.3 Estimation Hardware

The interface contains the ultrasound and infrared sensors and receivers necessary for state estimation. These components are on circuit boards which are arranged annularly around the front plate assembly, as shown as a simplified CAD drawing in Figure 2-4 and as a photograph in Figure 2-5. There are three each of ultrasound receivers, ultrasound beacons, infrared LEDs, and infrared phototransistors.

When being used for estimation, it is important to know the precise locations of the ultrasound beacons and receivers relative to the CM of the module to which they are attached. These locations, of course, depend on the size and geometry of the module and the location of the UDP on the module. For each module, the estimator must ‘know’ the locations of the UDPs.

24
2.4 Conclusion

The main purpose of this chapter has been to present the design of the genderless UDP, which is used in the projects described later in this thesis. The UDP was designed under a set of requirements which included autonomy, androgyny, and metrology. The final design relies on two counter-rotating rings on each UDP to lock onto a pin from the other UDP during a docking. The UDPs also provides the sensors used in relative state estimation.
Figure 2-5: A picture of the front face of a fully assembled UDP.
Chapter 3

Relative Estimator

In order for two or more modules to maneuver in a collectively useful way, it is almost always necessary for the system to have knowledge of the relative or absolute positions of the individual modules. The relative position is useful in docking, tight formation flight maneuvers, or pointing operations, for example. The absolute position of the formation in the sky would be useful for applications such as ground observation, communication, or solar measurements. For most applications, the relative position must be estimated at higher frequency and accuracy than the absolute position [8, 6, 19].

This chapter is concerned with the development of a relative estimation system for multi-module separated spacecraft, specifically in the SWARM project. These modules will be performing formation control, formation reconfiguration, and docking, and all of these maneuvers require accurate measurements of the modules’ relative separations and attitudes.

3.1 Architecture Choices

Depending on the other requirements of the system, the requirement of having position information could be implemented in several ways. One approach is for each module to take measurements and calculate its position in an absolute global reference frame (probably fixed to the Earth). Relative distances would then be calculated
by subtracting global positions. For actual formation flying satellites, however, this is an unlikely scenario. Proximity operations often require distance measurements accurate to less than a centimeter, and it would be difficult to achieve this accuracy by subtraction of absolute measurements, because few on-orbit satellites can measure their absolute position around the Earth to sub-centimeter accuracy.

A better solution is to implement a relative measurement system. During most formation flight maneuvers, especially docking, it is the relative separation which is most important. For most applications, the separation between the two modules must be known more accurately (and at a higher frequency) than the absolute position in the orbit. This is convenient because high accuracy relative measurements are easier to collect than high accuracy absolute measurements.

Another architecture choice to be made deals with where the measurements and calculations take place. For example, a single master module could take all the measurements, calculate the positions of each module in the formation, and communicate that position to each module. Alternatively, each module could do its own measurement and calculation. The hardware requirements for these two choices are different, and the capabilities of each differ as well.

The relative estimation system implemented here is based on the Extended Kalman Filter (EKF) technique. This chapter will describe the measurements available to the estimator and the process by which those raw measurements are reduced to filtered state estimates [13].

### 3.2 Available Measurements

The state sensing hardware available on each module is

- ultrasound beacons
- ultrasound receivers
- infrared emitters
- infrared receivers
• rate gyros

• accelerometers

The inertial sensors (gyros and accelerometers) will be considered separately. The ultrasound and infrared sensors generate separation information by measuring time of flight differences as discussed here.

During this discussion, it may be useful to keep in mind a docking operation as an example scenario. Additionally, to facilitate the discussion, the terms ‘pinger’ and ‘detector’ will be used to denote two modules involved in relative estimation. This does not have to do with a master/follower or chaser/target distinction, it is simply used to distinguish two modules for this discussion.

When it is time for a position update, one module (the detector) will emit an omnidirectional infrared pulse. The other module will detect this pulse effectively instantaneously, using its infrared receivers. At this point, the ultrasound beacons on the pinger begin emitting chirps at points in time specified by their beacon number, as shown in Figure 3-1. Beacon #1 chirps 10 ms after the reception of the IR pulse, Beacon #2 chirps 30 ms after the IR pulse, and Beacon #3 chirps 50 ms after the IR pulse. The chirps are separated by 20 ms to allow sufficient time for the energy from each chirp to dissipate. If a receiver on the detector detects a chirp 32 ms after the IR pulse, for example, then that chirp must have taken 2 ms to travel from Beacon #2. It is not possible that it took 22 ms to travel from Beacon #1, because after 20 ms the energy from Beacon #1 is fully dissipated. There is margin built into the 20 ms delay, so the estimator can confidently assume that any chirp detected came from the most recent beacon.

When all chirps and receptions are complete, each receiver on the detector has detected a chirp from each beacon on the pinger. The estimator knows when these chirps were emitted and when they were received, so using the speed of sound, the separation distance between each beacon-receiver pair can be calculated. Each of these separation distances is put into an array called the distance matrix. These nine numbers (3 receivers and 3 beacons) provide enough information to calculate the full
Figure 3-1: Timing diagram for ultrasound and infrared sensors on both modules. The dotted line at time $t=0$ indicates the infrared flash by the detector module. The solid black lines at $t=10$, $t=30$, $t=50$ indicate beacon pings (the one at $t=10$ indicates beacon #1, and so on). The grey box on the detector’s timeline from $t=10$ to $t=20$ indicates the window during which ultrasound receptions will be interpreted as coming from beacon #1 on the pinger.

relative state.

### 3.3 State Vector and Attitude Representation

In order to fully maneuver, the satellites need to measure their relative position, velocity, attitude, and rotation rates. The state vector is 13 elements long:

$$\mathbf{x} = \begin{bmatrix} r_x & r_y & r_z & v_x & v_y & v_z & q_1 & q_2 & q_3 & q_4 & \omega_x & \omega_y & \omega_z \end{bmatrix}^T$$

(3.1)

where $\mathbf{r}$ is the position, $\mathbf{v}$ is the velocity, $\mathbf{q}$ is the attitude (represented by a quaternion), and $\mathbf{\omega}$ is the rotation rate.

There are several ways to represent the attitude of the vehicle, such as Euler angles, a direction cosine matrix, or a quaternion. Euler angles are singular and non-intuitive. The direction cosine matrix transforms a vector in one frame to a vector in a rotated frame, but is highly redundant. The matrix has nine elements, and a general rotation only has three independent degrees of freedom, so there are six redundant parameters. The quaternion representation of attitude was chosen because it is intuitive, it is non-singular, and it contains only one redundant element. See Appendix B for a mathematical introduction to quaternions and their use in...
attitude representation.

3.4 EKF Process - Linear Case

A brief summary of the EKF, as it applies to SWARM, is presented here. For more detail on the Kalman filter see [17, 19]. Most of the operation of the estimator can be presented by treating the filter as linear and the measurements as periodic; this is done first. Following in Section 3.5, a few non-linear extensions will be presented, as well as an extension to non-periodic measurements, to finally arrive at the actual full relative estimator [3].

For the purposes of presenting the Kalman filter, the system model can be thought of as a simple linear state space model with additive noise.

\[
\begin{align*}
\dot{x} &= Ax + Bu + w \\
y &= Cx + Du + v
\end{align*}
\]  

(3.2)

where \(w\) and \(v\) are additive noise:

\[
\begin{align*}
w &\sim N(0, Q) \\
v &\sim N(0, R)
\end{align*}
\]  

(3.3)

The \(D\) matrix is assumed to be zero. The noise in the system model, or process noise, is assumed to be normally distributed with zero mean and standard deviation \(Q\). Similarly, the measurement noise is assumed to be unbiased with standard deviation \(R\). The filter also keeps track of the estimated noise \(P\) on the state \(x\) at each time step. The quantities \(Q\), \(R\), and \(P\) are matrices, where each element gives the covariance between two states or measurements. For example, the \((i, i)\) element in \(P\) gives the variance of the \(i^{th}\) element of the state vector, \(x_i\). The \((i, j)\) element, though, gives the covariance between \(x_i\) and \(x_j\). The matrix \(P_k\) contains covariances at time \(k\). The measurements can be considered to be taken periodically.

The EKF maintains a current best estimate of the state of the vehicle. When a new measurement comes in at time \(k\), the filter compares the measurements to
The current estimated state of the vehicle and uses the two to generate a new best estimate. The filter then propagates forward to the next time a measurement is expected. This process is shown in Figure 3-2.

A note about notation – subscripts to the right of $x$ indicate the time step, subscripts to the left indicate whether that state is before or after the update with ‘−’ indicating before and ‘+’ meaning after, and hats indicate an estimated state rather than the true state. So, for example, $\hat{x}_{k+1}$ would indicate the estimated state at time $k+1$, but before the update. Remember that $\hat{x}_k$ and $\hat{x}_k$ are both estimates at the same time; $\hat{x}_k$ is the propagated state from the previous update without taking into account the new measurements, and $\hat{x}_k$ is an estimate of the states at the same time, but taking into account the new measurements.

The filter takes $\hat{x}_k$ along with the measurements at time $k$ to generate the new estimate $\hat{x}_k$, and the Kalman gains determine whether the propagated estimate ($\hat{x}_k$) or the actual measurement has more impact on deciding $\hat{x}_k$. If the measurements are more trustworthy (less noisy) than the model, then the measurements would have a larger impact on the new estimate. If the measurements are known to be very noisy, then the model propagation from the previous step would take priority. This balance is captured in the Kalman gains as described below.
data comes in at time $k$

previous estimates $\hat{x}_k$ and $P_k$ are known

compute Kalman gains

$$L_k = -P_k C^T \left( C P_k C^T + R \right)^{-1}$$

propagate

$$\hat{x}_{k+1} = A_{d} \hat{x}_k$$
$$P_{k+1} = A_{d} P_k A_{d}^T + Q_k$$

update state estimate

$$\hat{x}_k = \hat{x}_k + L_k (y_k - C \hat{x}_k)$$

update covariance

$$\bar{P}_k = (I - L_k C) P_k$$

Figure 3-3: This diagram shows the basic cycle of the linear Kalman filter. It begins at the top with new data coming in.

Once a new piece of data becomes available, the EKF follows the following general cycle, described in this section and illustrated in Figure 3-3 [18].

1. compute Kalman gains
2. update state estimates
3. update covariance
4. propagate

When a new measurement comes in at time $k$, the propagated estimates from the previous update, $\hat{x}_k$ and $P_k$ are known. Now, Kalman gains $L_k$ must be computed.

$$L_k = -P_k C^T \left( C P_k C^T + R \right)^{-1} \quad (3.4)$$
This takes into account the covariance of the previous estimates \( -P_k \) and the noise in the sensors \( R \) to create a set of gains which will control how much the new measurements are able to alter the new state estimate. The new state estimate \( +\hat{x}_k \) is then calculated using the Kalman gains according to

\[
+\hat{x}_k = -\hat{x}_k + L_k(y_k - C_-\hat{x}_k)
\]  

(3.5)

We have the new updated state estimate \( +\hat{x}_k \), and we must also update the covariance matrix \( P \). The covariance matrix is updated using

\[
+P_k = (I - L_k C) -P_k
\]  

(3.6)

Now, the update phase (the 'hop' at time \( k \) in Figure 3-2) is complete, and the only remaining task is to propagate in preparation for the next measurement at \( k + 1 \). With this system model, the state propagation is simple:

\[
-\hat{x}_{k+1} = A_d + \hat{x}_k
\]  

(3.7)

where the subscript \( d \) emphasizes that \( A_d \) is part of a discretized, not continuous, system model. At this stage, the control inputs are not accounted for in the propagation, see Section 5.1.4 for a discussion of this as future work. Propagating the covariance matrix is simple as well, keeping in mind that the model noise \( Q_k \) must be included.

\[
-P_{k+1} = A_d + P_k A_d^T + Q_k
\]  

(3.8)

We now have \( -\hat{x}_{k+1} \) and \( -P_{k+1} \), which when the filter runs the next time will become \( -\hat{x}_k \) and \( -P_k \), so the filter is complete and ready to be run again.

### 3.5 Nonlinearities

Two extensions to the linear periodic Kalman filter have been made for the relative estimator. The first allows for non-periodic measurements, and the second allows for
Figure 3-4: When a new piece of data comes in, the estimator shown here would first propagate the state from the last known time to the current time, then update the estimate with the new data (bold blue arrows).

The real system has both ultrasound measurements and gyro measurements. These two forms of data do not come at the same frequency, and occasionally, an ultrasound receiver will ‘drop out’ for one cycle. This means that it is not possible to predict when the next measurement will occur. The filter must be able to handle this.

The consequence for the filter presented above is that it is no longer possible to propagate at the end of the cycle. The last step is to take \( +\hat{x}_k \) and propagate it forward in time to the point of the next measurement, creating \( -\hat{x}_{k+1} \), but this is no longer possible because the filter would not know when the next measurement will be available so it would not know how far forward to propagate.

The problem can be solved by simply changing the order of steps in the filter. When the filter begins, it will be given \( +\hat{x}_{k-1} \) and its first step will be to propagate to the present time. It will then compute gains and update the measurements. In effect, the diagram has been changed to the one shown in Figure 3-4.
Figure 3-5: Crosses represent beacons and circles represent receivers. If a receiver is at position 1 and moves slightly to position 2, then the measurement (distance between beacon and receiver) will change by a small amount. However, if the receiver is at position 3 and moves slightly to position 4, then the measurement will change by a larger amount. This is evidence of nonlinearity in the relationship between change in measurements and change in states.

### 3.5.2 Nonlinear Measurements

The ultrasound measurements from the UDPs are definitely nonlinear. Consider the case of two UDPs facing each other. If one UDP moves slightly, the ultrasound distance measurements will change by a certain amount, as shown in Figure 3-5. However, if the UDPs are separated transversely and move by the same amount, then the measurements will change by a different amount.

The use of the $C$ matrix above implicitly ignored this problem, now we must correct it. The correction can be made by creating a full nonlinear model of the sensor behavior, and linearizing a $C$ matrix at each time step. The linearized $C$ is the Jacobian, called $H$.

The plan is to find the function $h_k(x(t_k))$ which outputs the expected measurements given the state of the system, then differentiate that function. The function $h_k(x(t_k))$ is a vector valued function of a vector, so its derivative will be a matrix, the Jacobian $H$. The linearized $C$ matrix at each time step is the Jacobian at that time step,

$$H = C_k(-\dot{x}_k) = \frac{\partial h_k(x(t_k))}{\partial x(t_k)} \bigg|_{x(t_k)=-\dot{x}_k}$$  \hspace{1cm} (3.9)

First, a comment about notation. As stated earlier, this filter is built assuming
Figure 3-6: This figure shows the notation of the position of beacons and receivers in the reference frames of the two modules. Crosses represent beacons and circles represent receivers. The beacons and receivers would actually be on a UDP, of course, but for clarity, just the SPHERES satellite and the beacons and receivers are drawn.

that one module has beacons pinging, and the other module uses its receivers to measure the pings. The module with beacons pinging will be called the target and the module with receivers active will be called the chaser. As before, this has nothing to do, in principle, with a target/chaser distinction in a docking maneuver, they are just names to distinguish the two. The filter must 'know' the locations of the beacons and receivers relative to the CM of each module, but reference frames are important to specify. As shown in Figure 3-6, the position of beacon number $i$ in the body coordinates of the target module will be denoted $b_i^{TT}$. The position of receiver number $j$ in the body coordinates of the target module will be denoted $s_j^{CT}$. The position of receiver number $j$ in the body coordinates of the chaser module will be denoted $s_j^{CC}$. So, $b$ represents beacons (on the target), $s$ represents receivers (on the chaser), the first superscript represents the location of the beacon or receiver, and the second superscript represents the reference frame in which the vector is expressed. Keep in mind that $s_j^{CT}$ represents the position of a receiver on the chaser, but in the reference frame of the target.

The first step is to calculate the expected measurements $h$ given the state of the vehicle $x$. This calculation, like most in this section, will take place in the reference frame of the target (the module with active beacons). The measurements are times
of flight of ultrasound signals, which are converted into distances using the speed of sound. So, for each beacon ping, \( h \) is a vector containing separations between a beacon and each receiver. In other words, when beacon \( i \) pings, the \( j \)th element of \( h \) is the separation distance between beacon \( i \) on the target and receiver \( j \) on the chaser. Calculating their separation distance is simply a matter of finding all their positions in one reference frame and subtracting.

The mechanical construction of the modules and the UDP dictates \( b_i^{TT} \) and \( s_j^{CC} \). These are hard coded in the software. The quantity that changes is \( s_j^{CT} \). Finding \( s_j^{CT} \) knowing the relative state of the two vehicles is simple:

\[
s_j^{CT} = Qs_j^{CC} + r
\]  

(3.10)

where \( Q \) is the rotation matrix which rotates vectors expressed in the chaser's frame to the target's frame, and \( r \) is the first three elements of the relative state vector – the Cartesian separation between the two modules. Now, the vector from receiver \( j \) to beacon \( i \) is

\[
d_{ij} = b_i^{TT} - s_j^{CT}
\]  

(3.11)

but the ultrasound metrology system only measures the magnitude of the separation, not the full vector, so the distance from the beacon to receiver \( j \) is

\[
h_j = |d_{ij}|
\]  

(3.12)

\[
= \sqrt{(b_{ix}^{TT} - s_{jx}^{CT})^2 + (b_{iy}^{TT} - s_{jy}^{CT})^2 + (b_{iz}^{TT} - s_{jz}^{CT})^2}
\]  

(3.13)

This is the function \( h_k(x_k) \). The elements of the state vector \( x_k \) do not explicitly appear in Equation 3.13, but they are involved in calculating \( s_j^{CT} \) in Equation 3.11. The position appears as \( r \), and the attitude quaternion is required in calculating the rotation matrix \( Q \).

Now, we must find the Jacobian, \( H \), which is \( \frac{\partial h_k}{\partial x_k} \). The number of rows of \( H \) will be the number of measurements (number of receivers), and the number of columns will be the length of the state vector.
It can be seen at this point that $H$ will only have 7 non-zero columns, the other 6 columns will be full of zeros. Only 7 of the state elements appear in $h_k(x_k)$ – the position and attitude states. The velocity and rotation rate state elements do not appear at all, so their derivatives will be zero.

Block #1 describes how changing each position state element affects each measurement, and Block #2 shows how changing each attitude state element affects each measurement.

To calculate Block #1, we must calculate $\frac{\partial h_i}{\partial r_i}$. The notation is simplified by writing

$$\left( \mathbf{b}_{ix}^T - \mathbf{s}_{jx}^{CT} \right)^2 + \left( \mathbf{b}_{iy}^T - \mathbf{s}_{jy}^{CT} \right)^2 + \left( \mathbf{b}_{iz}^T - \mathbf{s}_{jz}^{CT} \right)^2 \text{ as } \alpha.$$  

$$\frac{\partial h_i}{\partial r} = \frac{\partial}{\partial r} \sqrt{\left( \mathbf{b}_{ix}^T - \mathbf{s}_{jx}^{CT} \right)^2 + \left( \mathbf{b}_{iy}^T - \mathbf{s}_{jy}^{CT} \right)^2 + \left( \mathbf{b}_{iz}^T - \mathbf{s}_{jz}^{CT} \right)^2}$$  

$$= \frac{1}{2} \frac{1}{\sqrt{\alpha}} \frac{\partial}{\partial r} \alpha$$  

$$= \frac{1}{2} \frac{\partial}{\partial r} \left( \left( \mathbf{b}_{ix}^T - \mathbf{s}_{jx}^{CT} \right)^2 + \left( \mathbf{b}_{iy}^T - \mathbf{s}_{jy}^{CT} \right)^2 + \left( \mathbf{b}_{iz}^T - \mathbf{s}_{jz}^{CT} \right)^2 \right)$$  

$$= \frac{1}{2} \frac{\partial}{\partial r} \left( \left( \mathbf{b}_{ix}^T - \mathbf{s}_{jx}^{CT} \right)^2 + \left( \mathbf{b}_{iy}^T - \mathbf{s}_{jy}^{CT} \right)^2 + \left( \mathbf{b}_{iz}^T - \mathbf{s}_{jz}^{CT} \right)^2 \right)$$

To analyze the vector derivative, consider it one element at a time. For the $r_x$ component,

$$\frac{\partial h_i}{\partial r_x} = \frac{\mathbf{b}_{ix}^T - \mathbf{s}_{jx}^{CT}}{\sqrt{\left( \mathbf{b}_{ix}^T - \mathbf{s}_{jx}^{CT} \right)^2 + \left( \mathbf{b}_{iy}^T - \mathbf{s}_{jy}^{CT} \right)^2 + \left( \mathbf{b}_{iz}^T - \mathbf{s}_{jz}^{CT} \right)^2}}$$  

$$39$$
for the $r_y$ component,

$$\frac{\partial h_i}{\partial r_y} = \frac{b_{iy}^{TT} - s_{iy}^{CT}}{\sqrt{(b_{ix}^{TT} - s_{ix}^{CT})^2 + (b_{iy}^{TT} - s_{iy}^{CT})^2 + (b_{iz}^{TT} - s_{iz}^{CT})^2}} \quad (3.19)$$

and for the $r_z$ component,

$$\frac{\partial h_i}{\partial r_z} = \frac{b_{iz}^{TT} - s_{iz}^{CT}}{\sqrt{(b_{ix}^{TT} - s_{ix}^{CT})^2 + (b_{iy}^{TT} - s_{iy}^{CT})^2 + (b_{iz}^{TT} - s_{iz}^{CT})^2}} \quad (3.20)$$

So,

$$\frac{\partial h_i}{\partial r} = \frac{b_i^{TT} - s_i^{CT}}{\sqrt{(b_{ix}^{TT} - s_{ix}^{CT})^2 + (b_{iy}^{TT} - s_{iy}^{CT})^2 + (b_{iz}^{TT} - s_{iz}^{CT})^2}} \quad (3.21)$$

$$\frac{\partial h_i}{\partial r} = \frac{b_i^{TT} - s_i^{CT}}{h_i} \quad (3.22)$$

This completes the calculation of Block #1.

For Block #2, we must differentiate the measurements $h_i$ with respect to the attitude of the module, contained in the rotation matrix $Q$.

$$\frac{\partial h_i}{\partial q} = \frac{1}{\sqrt{\alpha}} (b_i^{TT} - s_i^{CT}) \frac{\partial}{\partial q} (b_i^{TT} - s_i^{CT}) \quad (3.23)$$

The vector derivative $\frac{\partial}{\partial q}$ can be broken down by splitting it into three general components $(u,v,w)$. Recall that $b_i^{TT} - s_i^{CT}$ can be written as $d_{ij}$, as is done for Equation 3.24.

$$\frac{\partial h_i}{\partial q} = \frac{1}{\sqrt{\alpha}} d_{ij} \left( \frac{\partial}{\partial u} (d_{ij}) \frac{\partial u}{\partial q} + \frac{\partial}{\partial v} (d_{ij}) \frac{\partial v}{\partial q} + \frac{\partial}{\partial w} (d_{ij}) \frac{\partial w}{\partial q} \right) \quad (3.24)$$

If the vector $(u,v,w)$ is set equal to $Qs_j^{CC}$, then Equation 3.24 reduces to

$$\frac{\partial h_i}{\partial q} = \frac{b_i^{TT} - s_i^{CT}}{\sqrt{\alpha}} \left( \frac{\partial u}{\partial q} + \frac{\partial v}{\partial q} + \frac{\partial w}{\partial q} \right) \quad (3.25)$$

$$\frac{\partial h_i}{\partial q} = \frac{b_i^{TT} - s_i^{CT}}{\sqrt{\alpha}} \left( \frac{\partial}{\partial q} (Qs_j^{CC})_x + \frac{\partial}{\partial q} (Qs_j^{CC})_y + \frac{\partial}{\partial q} (Qs_j^{CC})_z \right) \quad (3.26)$$
However, $s_j^{CT}$ is a constant, so it can be removed from the derivatives.

$$\frac{\partial h_i}{\partial q} = \frac{b_i^{TT} - s_j^{CT}}{\sqrt{\alpha}} \left( \frac{\partial Q}{\partial q} s_j^{CO} \right)$$  (3.27)

Now, we must calculate $\frac{\partial Q}{\partial q}$. To generate a rotation matrix from the elements of a quaternion, the relationship is [13]:

$$Q(q) = \begin{bmatrix}
q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\
2(q_1q_2 - q_3q_4) & q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 + q_1q_4) \\
2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & q_1^2 - q_2^2 + q_3^2 + q_4^2
\end{bmatrix}$$  (3.28)

Differentiating all nine of these elements by each of the elements of $q$ creates a quantity with three dimensions and 36 elements. Each differentiation is simple, so they will not be explicitly shown here. They are contained in a piece of code attached in Appendix A.4.

We can now construct $H$, using Equation 3.22 for Block #1, and Equation 3.27 for Block #2. This completes the development of nonlinear measurement capability in the relative estimator.

### 3.6 Prefilter

As will be discussed and analyzed in Section 4.1, the ultrasound sensors sometimes give bad readings. They occasionally give a reading which is obviously incorrect (a meter off, for example), and they sometimes miss readings all together and return zero. In either of these cases, this erroneous data could throw off the estimator. It is helpful to have a prefilter before the estimator to screen the data. The prefilter currently has three stages.

- **Remove Zeros** When a receiver returns exactly zero for its measurement, it means that the receiver did not receive anything. It doesn't actually mean that the beacon and the receiver are very close, it just indicates a missed reading.
These cases are removed by the prefilter, so the data does not get to the estimator where it would be interpreted as the beacon and receiver being separated by 0 cm.

- **Remove Geometrically Impossible Readings** The receivers on the UDP are separated by fixed distances. If two receivers give readings that are off by more than the largest of these distances, than one (or both) of the readings is wrong. Instead of figuring out which one is correct (which takes time and is not always possible), the prefilter throws out the whole update, and the estimator just propagates through that time step without a measurement update.

- **Remove Large Readings** The ultrasound energy from a beacon dissipates after 3 meters, so any readings larger than that must be incorrect. They are removed.

### 3.7 Conclusion

This chapter presented the core theoretical basis for the EKF-based relative state estimator. Data for the estimator comes from the ultrasound range measurements between beacons and receivers. To generate a new state estimate each time new data becomes available, the estimator follows a four step process - propagate, compute Kalman gains, update state estimate, and update covariance. For actual implementation, there are several nonlinear extensions to the basic linear Kalman filter, including the nonlinear measurement model and the ability to handle nonperiodic measurements. To further filter the data, a prefilter runs before the estimator to screen out obviously erroneous measurements. The next chapter will cover actual testing of the estimator.
Chapter 4

Results and Applications

This chapter discusses the use of the relative estimator in several relevant hardware projects. The estimator is most useful and enabling in projects involving modularity, formation flight, docking, and reconfiguration. The projects discussed here are SWARM (Synchronized Wireless Autonomous Reconfigurable Modules) and SIFFT (Synthetic Imaging Formation Flight Testbed).

The relative estimator was tested in three phases. First, a low-level verification and analysis of the ultrasound receivers was carried out. This test was necessary to show that the receivers gave steady and accurate time-of-flight measurements, and that they did not respond to extraneous ultrasound energy. Second, the full estimator was coded in Matlab® and then run on actual data. This showed that the estimator was coded correctly and that the estimates produced were reasonable. Third, the estimator was translated into C code and run in actual docking experiments on real hardware. This is the final test of usefulness for the estimator. Each of these three phases of testing is described in this chapter.

4.1 Testing of Ultrasound Receivers

To test the ultrasound receivers (and beacons), a beacon-receiver pair was placed a fixed distance apart, and a large number of time-of-flight measurements was taken. The measurement here is just the pure time delay of the ultrasound ping, there is
Figure 4-1: This plot shows 60 seconds of data for one beacon-receiver pair. Overall, the data is very consistent. Notice one error at approximately 10 seconds, and several zero readings, indicating times when the sensor temporarily dropped out.

no filtering or estimation. Because neither the beacon nor the receiver is moving, the expected result of the test is that all the measurements will be tightly clustered around the actual separation. The most useful result of this test, though, will be to determine if the receivers commonly pick up ultrasound noise produced in the environment, if they commonly detect pings from the previous beacon (echoes), or if they commonly miss a ping altogether.

Because there are three receivers and three beacons on each UDP, many combinations of beacon-receiver pairs were tested. If just one beacon and one receiver were chosen for this test, then the results would not be generalizable to all pairs.

Figure 4-1 shows the time series data from a single beacon-receiver pair. The beacon is pinging at 5 Hz for 60 seconds. Clearly, the measurements are very steady.
with no large scale deviations. There are occasional large errors and occasional zeros. In this data set, there are six zero measurements, when for unknown reasons, the sensor did not read anything (so returns zero). There is one case (just after 10 seconds) where the sensor did record a different, non-zero range, but the reading was wrong. Overall, in this data set, 2% of the measurements were missed (zero), and less than 1% were non-zero but erroneous.

Figure 4-1 does not show the small amount of noise around the mean. In Figure 4-2, the vertical axis is tightened to half a centimeter on either side of the mean to show this behavior. Even at this small scale, the sensor is still quite consistent – the standard deviation of the measurements is only 0.267 mm.

A missed measurement is not really a reading of zero, it is just a time when no
Table 4.1: Summary of results from single beacon-receiver pair test.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>number of measurements</td>
<td>300</td>
</tr>
<tr>
<td>standard deviation (mm)</td>
<td>0.267</td>
</tr>
<tr>
<td>number of lost measurements</td>
<td>6</td>
</tr>
<tr>
<td>percentage lost</td>
<td>2%</td>
</tr>
</tbody>
</table>

data came in. The prefilter will remove these data points before they even get to the estimator. Table 4.1 summarizes the data from this test.

The data also shows that the sensor does not often receive signals from extraneous ultrasound sources and record them as data. This is significant because it is possible that there could be such sources in the environment in the lab (or in the space station), but they do not seem to cause frequent errors in the data.

This data is just from one beacon-receiver pair in a time series. Although it is useful to analyze the data like this to make sure that there are no large scale or low frequency errors, it is more useful to analyze all data from all beacon-receiver pairs and analyze their deviations from their expected values. To this end, a larger set of data was collected (in several runs). In each run, data was collected from all nine possible beacon-receiver pairs. That's approximately 60 seconds of data from each pair. A total of seven data runs were carried out for a total of 16947 measurements. So, for a given beacon-receiver pair, approximately 60 seconds of data was taken and this was repeated seven times.

Let $T_{ijr}$ represent the set of measurements taken from receiver $j$ of the distance to beacon $i$ during run $r$. (The data plotted in Figure 4-1 and Figure 4-2 happens to be $T_{118}$.) With data being taken at 5 Hz for approximately 60 seconds, each $T_{ijr}$ will contain approximately 300 measurements. To compare data from one beacon-receiver pair to another, the means are subtracted out and just the errors are analyzed.

$$\tilde{T}_{ijr} = T_{ijr} - \mu(T_{ijr})$$

(4.1)

where $\mu()$ indicates the mean of the data set and the tilde $\tilde{}$ indicates a zero-mean
Table 4.2: Summary of results from all beacon-receiver pairs.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>number of measurements</td>
<td>16947</td>
</tr>
<tr>
<td>standard deviation (mm)</td>
<td>0.459</td>
</tr>
<tr>
<td>number of lost measurements</td>
<td>850</td>
</tr>
<tr>
<td>percentage lost</td>
<td>5%</td>
</tr>
</tbody>
</table>

data set. So, for each beacon-receiver pair, the means are subtracted out, then the data is combined into one large set for analysis.

\[
\hat{T}_r = \bigcup_{i=1}^{3} \bigcup_{j=1}^{3} \hat{T}_{ijr}
\]

(4.2)

Now, \(\hat{T}_r\) contains all zero-mean measurements from all beacon-receiver pairs for a specific data run. Finally, because the UDPs were not moved and nothing was changed between data collection runs, the runs should be comparable to each other. We can create one very large data set \(\hat{T}\) will all the measurements from all the runs.

\[
\hat{T} = \bigcup_{r=1}^{7} \hat{T}_r
\]

(4.3)

\(\hat{T}\) contains the error in each measurement, so the deviation of the elements of \(\hat{T}\) from zero is the measure of interest. A histogram of \(\hat{T}\) is shown in Figure 4-3. The data is summarized in Table 4.2. Overall, it shows that the receivers are reliable and accurate. The data is tightly clustered around zero error (standard deviation less than half a millimeter), and large outliers are infrequent and can be easily identified and removed.

4.2 Testing in Simulation

The relative estimator was first coded in Matlab® because the code is more concise and easier to debug than C code. Raw data was taken on the hardware then processed in the Matlab® simulation to prove the algorithm.
Figure 4-3: Histogram showing all $\bar{\hat{T}}_{ijr}$. Obviously, nearly all the data is clustered close to zero error, with small amounts slightly off. Nearly all the data seems to be centered within 1 mm of error.
Two UDPs were placed at fixed positions and attitudes, pointed at each other. The metrology data collection system was run as usual, with infrared pulses at 5 Hz. Distance matrices were recorded (with nine numbers each) at each time step and saved to a laptop. The data collection typically ran for one minute, generating 300 distance matrices.

These distance matrices are fed to the Matlab® simulation in the same way that they would be fed to the C estimator running on the hardware. The estimator will produce a time history of estimated states, which can be examined after the run. The Matlab® simulation, like the actual C estimator, requires an initial position estimate \( \dot{x}_0 \) from which to begin estimating, and an initial state covariance matrix \( P_0 \). In a real situation, the initial state of the vehicles would not be known; a reasonable starting value would be given based on the test setup, and the estimator would converge to the actual state. In this case, though, the true state is known. So, the first test is to give the estimator the true initial state of the vehicles. This way, there is no convergence and the estimator just needs to continue reporting the same position. The next test is to give the estimator an incorrect initial state, which is much more likely to occur in actual use. By varying the error in initial state, the capability of the estimator to handle initial errors can be examined.

The actual state of the vehicles on the table for this test setup was:

\[
x_0 = \begin{bmatrix} 0.6 & 0 & 0 & 0 & 0 & 0 & 0.259 & 0 & 0.966 & 0 & 0 & 0 \end{bmatrix}^T \quad (4.4)
\]

which represents 60 cm separation (CM to CM, not UDP to UDP) of two SPHERES satellites and a 30° rotation around the y axis. All velocities are zero. Using the fact that the state vector is split into position, velocity, attitude, and rotation rates by

\[
x = \begin{bmatrix} r & v & q & \omega \end{bmatrix}^T \quad (4.5)
\]

it is easier to specify the configuration by just stating
\[
\begin{align*}
\mathbf{r}_0 &= \begin{bmatrix} 0.6 & 0 & 0 \end{bmatrix}^T \tag{4.6} \\
\mathbf{q}_0 &= \begin{bmatrix} 0 & 0.259 & 0 & 0.966 \end{bmatrix}^T \tag{4.7}
\end{align*}
\]

For the first test, the estimator was given the true \( \mathbf{x}_0 \) as an initial condition, so \( \dot{\mathbf{x}}_0 = \mathbf{x}_0 \). Figure 4-4 shows the time history of estimated states when the estimator begins with the correct initial condition. The figure shows all 13 state elements, broken down into position, velocity, attitude, and rotation rates. As expected, because the modules are fixed and not moving relative to each other, and because the initial state is correct, all state estimates are constant. This is a simple test, but it shows that the estimator has been mostly debugged. It shows that measurements are being interpreted correctly to yield position and attitude, which is a significant step. It does not, however, say anything about the estimator’s ability to handle data which is not consistent with the current estimated state, which is the case during convergence from an incorrect initial condition.

Convergence can be tested by beginning at a slightly incorrect initial state. For the second test run, the initial state given to the estimator was

\[
\begin{align*}
\mathbf{r}_0 &= \begin{bmatrix} 0.6 & 0.3 & -0.3 \end{bmatrix}^T \tag{4.8} \\
\mathbf{q}_0 &= \begin{bmatrix} 0 & 0 & 0.174 & 0.985 \end{bmatrix}^T \tag{4.9}
\end{align*}
\]

which is 30 cm off in position in two axes, 30° off in angle, and rotated around the wrong axis. Figure 4-5 shows the convergence in this case. The filter successfully converges to the correct state.

### 4.3 Hardware

The various applications described in this chapter share some of the same supporting hardware. This section presents the main hardware elements, describing their capabilities and functions individually but not their use in a particular project [4, 5].
Figure 4-4: These graphs show the time series of estimated states from the relative estimator, starting from the correct initial state. The $x$ position, $x$-pos, reads about 0.6, and the quaternion elements $\text{quat}2$ and $\text{quat}4$ read about 0.259 and 0.996, as expected. The estimated states shown here match the true state given in Equation 4.4. As expected, all states are constant throughout the simulation.
Figure 4-5: The filter begins from an incorrect initial position, but converges to the correct values quickly. Velocity takes longer because there is no direct velocity sensor; velocity must be calculated from changes in position.
4.3.1 SPHERES Satellites

The SPHERES satellites are complete vehicles. They have all the major systems of a regular satellite—propulsion, communication, sensors, structure, propellant storage, power, avionics, etc. They are quite small, only 21 cm in diameter and 4 kg wet. A SPHERES satellite is shown in Figure 4-6.

SPHERES satellites have 12 cold gas CO\textsubscript{2} thrusters which provide 0.12 N of thrust each. These thrusters are arranged on the vehicle to provide full control over all three translational degrees-of-freedom and all three angular degrees-of-freedom. There is a 200 g CO\textsubscript{2} tank inside the satellite which feeds 800 psi CO\textsubscript{2} to a regulator, which then reduces it to about 30 psi for the thrusters. SPHERES satellites do not have on-board reaction wheels.
Communication is wireless, and there are two frequencies available – one for satellite-to-satellite communication, and one for satellite-to-ground communication. Usually, a laptop with an appropriate wireless transceiver functions as the ground station. The range of both communication channels is approximately 5 m.

There are 24 ultrasound receivers arranged around the vehicle, with 4 on each face. Additionally, there are infrared LEDs and phototransistors around the vehicle to receive and emit IR flashes.

Finally, each SPHERES satellite has an expansion port, shown in Figure 4-7, consisting of a 100-pin connector and a structural connection to the body of the satellite. The 100-pin connector provides access to power, the internal data bus, and serial communication. It allows expansion items to attach to the port, communicate with the SPHERES satellite, and control or be controlled by the satellite. When not in use, or when on orbit inside the International Space Station (ISS), the expansion port is covered by a face plate.
4.3.2 Air Carriages

To enable floating on a flat surface for 2D testing, several types of air carriages were built. In all cases, their function is to slowly but steadily expel a cushion of CO₂ from the bottom, and float on this cushion across the surface.

Three types of air carriages were built: single-puck, three-puck, and ACS. Single puck carriages, like the one shown in Figure 4-8, have a single air bearing puck on top of which is a plate containing two CO₂ tanks, tubes, valves, a manifold, and a regulator. The tanks can be easily refilled when they become empty. Above the plate is a mount (the green piece in the figure) which is sized to hold the bottom of a SPHERES tank.

Three puck carriages are very similar to single puck carriages, but they have three smaller pucks. To more easily slide over small ‘hills’ in the flat surface, the pucks are on bearings allowing them to pivot. The main operational difference between single
puck and three puck carriages is that three puck carriages float better, while single puck carriages float longer before running out of gas.

Finally, there are ACS carriages, which are air carriages with reaction wheels built in. They are single puck designs. They have a lot of torque authority, but the disadvantage is that they are significantly heavier than the other two types. ACS carriages are not used in the work presented in this thesis.

### 4.3.3 Avionics Stack

There is a piece of hardware called the ‘stack’ which has most of the avionics and communication of a full SPHERES satellite, but none of the thrusters or attendant pressure hardware. It is a useful test platform to test code before running it on a real satellite. Additionally, the stack can be used to control or read data from expansion port items. The stack is shown in Figure 4-9.
4.3.4 UDPs, Subapertures, and Other Hardware

To attach modules to the air carriages, some structural hardware is required. A base plate sits on the air carriage, and allows other hardware such as SPHERES satellites or payloads to be attached to it. Each plate has four positions where posts to hold UDPs can be attached. The post holds the UDP at a specific height off the plate ensuring that it is at the same height as other UDPs on other modules. A plate with two posts, two UDPs, and a SPHERES satellite is shown a few pages later in Figure 4-12.

Telescope subapertures were used as example payloads. In this case, they were Takahashi Mewlon 210 type telescopes, shown in Figure 4-10. For these experiments, the subapertures were not actually used to collect light or combine light, they were simply stand-ins. They were useful, though, because they have a representative mass and mass distribution.

4.3.5 Flat Table

In the Space Systems Laboratory at MIT, there is a 4 ft × 6 ft flat glass table, suitable for floating air carriages. The glass sheet is on a large optics table which allows leveling of the table. The table is shown in Figure 4-11.
4.4 SWARM

The SWARM project is experimenting with modularity in spacecraft design. All common functions of a satellite are split into separate modules, then recombined dynamically to form a useful vehicle. These modules communicate wirelessly and maneuver autonomously. They connect to each other using a universal docking port (UDP).

The actual design and construction of each module is not the main research objective of SWARM. Little will be learned about modular spacecraft by spending time optimizing the design of the battery or storage modules, for example. Instead, the critical technologies to test are those which are unique to modular spacecraft – relative estimation, docking, and reconfiguration.

Reconfiguration is the updating of mass properties, control gains, and thruster effectiveness for the new collection of modules after a docking. Consider, for example, a tug module docking to a propulsionless payload module. Before the docking, the tug’s thrusters have a known affect on the acceleration and angular rates of the tug based on the thruster positions, CM location, and thruster strengths. After the
Figure 4-12: A SPHERES satellite on a SWARM plate with two UDPs. There is a three puck air carriage supporting the module. This is the SWARM tug module.

docking, the CM has moved significantly because of the large additional mass, but the thrusters have not. Each thruster will now have a significantly different effect on the acceleration and rotation rates of the vehicle. These updates must be properly accounted for [23].

On-orbit assembly is the main target mission scenario for SWARM. The goal is for a collection of modules to autonomously assemble themselves into a useful system starting from scattered and unknown initial positions. Under this scenario, there would be a tug module, and several payload modules. All modules would have UDPs, but only the tug would have thrusters.

4.4.1 Setup

SPHERES satellites were used as the tug module for the SWARM project. This was convenient because SPHERES satellites already have thrusters, propellant storage, and wireless communication. So, to create the tug module, a SPHERES satellite is placed on a base plate which is placed on an air carriage. To allow docking, two UDPs on two posts are secured to the base plate.

The payloads used for SWARM are telescope subapertures. This choice was made
because the subapertures have a reasonable mass and mass distribution, and also because interferometry is an attractive application for modular formation flying space systems, so subapertures seemed to be a reasonable first choice for a payload. In this project so far, the subapertures have not been used to collect light and actually create interference fringes, that is a future step.

The subaperture modules were created by placing a telescope on a base plate, placing that on an air carriage, and connecting UDPs. The stack was also mounted on this module to control the UDPs. The subaperture module is shown in Figure 4-13.

A laptop with a wireless transmitter was used to transfer code to the tug, to issue commands during operation, and to download data for later analysis.

4.4.2 Experimental Plan

The flat table in the SSL at MIT is 4 ft $\times$ 6 ft. This is sufficient for tests involving only SPHERES, but it is too small to fit all the SWARM hardware with room to ma-
neuver. Therefore, most of the development took place at MIT, but full testing was conducted at the flat floor facility of the Flight Robotics Laboratory at NASA Marshall Spaceflight Center (MSFC). This facility is a very large epoxy floor measuring 44 ft × 68 ft, the largest of its kind in the world.

There were two main test objectives: docking and reconfiguration. First, it had to be shown that two modules could accurately measure their position, control their position along a trajectory using a docking algorithm, and successfully dock. Second, the assembly must reconfigure its mass properties in order to move in an intentional and coordinated manner after docking.

This sequence of objectives mirrors the full sequence of execution steps. First, the satellite begins estimating. When the estimator converges, the chaser satellite points toward the target, then aligns docking ports. The glideslope approach begins and the chaser begins to approach the target. When the two satellites are within range, the docking mechanism activates and connects the two modules. Next, the system reconfigures its mass properties and thruster configuration. Finally, a simple rotation maneuver is conducted to demonstrate successful reconfiguration [11, 12].

4.4.3 Sample Docking Results

The chaser satellite measures its position in a reference frame attached to the target satellite. The target satellite was fixed in place because only relative measurements between the vehicles are meaningful, and because errors such as inaccuracies in the flatness of the floor and wind currents would cause drift of the target (and thus the reference frame). Fixing the target was achieved by simply not floating its air carriage. During the docking maneuver, the chaser satellite recorded its estimated position relative to the target, and this data is plotted in Figure 4-14 and Figure 4-15. These two figures show data from two separate docking test runs. The x-axis is in the plane of the face of the docking ports and perpendicular to gravity, and the y-axis is normal to that plane, pointing 'forward' from the docking ports.

The portion at the top of the graph (of approximately constant y) shows the first part of the test, where the satellites were on and measuring position, but the docking
Figure 4-14: Position data for the chaser satellite in a reference frame fixed to the target, for an entire docking maneuver. The position are CM-to-CM, so the UDPs are near docking when the CMs are separated by approximately 60 cm.
Figure 4-15: In this test, the chaser may have encountered a slope in the floor causing it to drift to the right during the beginning of its approach. It measured and corrected this motion, managing to get back to $x \approx 0$. 
maneuver had not yet begun. During this phase, the satellite was simply trying to cancel out any slopes in the floor and get back to the proper position to start the straight-in docking motion. The satellite drifts approximately 25cm before arriving at the proper starting position.

Then, the satellite begins its approach, shown in the black colored points of Figure 4-14 (moving in the \(-y\) direction). It is fairly straight, and the velocity visibly slows near the end which can be seen in the increased density of points at the end of the trajectory. There is some final maneuvering at the end (approximately 5cm) to properly align with the target UDP.

Figure 4-15 shows another docking test at MSFC. The chaser was clearly started at a different position from the first test, this time a bit closer in. During the approach, the chaser encountered a disturbance, probably from a slope in the floor, which caused it to drift to the right. The estimator measured this position error, and the module was able to correct back to \(x \approx 0\). Despite some state noise at the end, the modules were still able to dock \([21, 23]\).

### 4.4.4 Overall Docking Results

A total of 61 docking tests were attempted. Of these, 48 passed the estimator convergence maneuver and began the docking approach. A total of 21 got close enough to begin berthing, and 12 achieved capture. This is a success rate of approximately 19%, which is a good preliminary result. There were several common reasons that docking was not achieved in a particular test. In some of the failures (approximately 20%), the estimator did not converge initially. Most of the rest of the failures were caused by unmodeled imperfections in the flat floor which cause the chaser to drift too far or too fast, or by insufficient thruster authority for the given mass. Limitations and extensions are discussed in Section 5.1.
4.4.5 Reconfiguration Results

A second goal of the SWARM project was to study and demonstrate reconfiguration of mass properties and thruster positions after a successful docking operation. Unfortunately, the SPHERES satellites did not have enough thruster authority to move well after a docking because the mass of the additional module was too great. There was some motion, though, and some conclusions were possible. These results are presented in [25].

4.5 SIFFT

The SIFFT project provided another opportunity to test the relative estimator. The goals of SIFFT center more on formation flight than on docking. Using modular spacecraft to build an interferometer requires multiple modules to enter a formation, hold that formation under disturbances, and change the formation.

This is exactly what was tested in the SIFFT project. Three modules were placed on the floor, but not in the correct configuration. They estimated their relative positions and bearing angles, then moved to the correct location. They held this position, using thrusters to counteract external disturbances such as wind or slopes in the floor. After a fixed amount of time, the formation changed and the modules moved to the new locations and held there.

Only relative estimation was used, no absolute measurements are available. Therefore, the formation geometry can be expected to be accurate, but it can not be positioned at a specific place on the floor. A pure translational motion of the CM of the entire array can not be sensed or controlled. If each module is deployed with minimal velocity, though, neither of these issues present problems for flat floor testing.

Each UDP has an effective Field of View (FOV) – the UDP can only ‘see’ beacons which are inside this FOV cone. The cone half-angle is approximately 45°. In a true formation deployment, the modules would start in truly random positions and orientations, with no guarantee that each satellite will find beacons inside its FOV. So, each module would need to execute a “lost-in-space” maneuver first, to locate the
other modules; only then could they enter and maintain a useful formation. The first
phase of the SIFFT project did not study the lost-in-space aspect of formation flight,
although progress on these algorithms has been made since the first phase of testing.
To eliminate the need for lost-in-space maneuvers and to directly attack the central
problems of formation maintenance and reconfiguration, the modules were placed in
initial positions such that estimation was immediately possible.

4.5.1 Hardware Setup

Because each of the three modules needs to have estimation and thruster capability,
the clear choice was to make them all SPHERES satellites with docking ports and
air carriages. The full experimental hardware setup is shown in Figure 4-16.

4.5.2 Estimation Configuration

Estimation configuration is the software-based arrangement of beacons, receivers, esti-
mator code, and communication on the various modules in a formation. For example,
consider two modules with UDPs pointed at each other, and requiring relative range
data. In this example, of course, there is only one meaningful relative state, because
the position of Module 1 with respect to Module 2 is the opposite of the position of Module 2 with respect to Module 1. One possible estimation configuration is that each module will ping its beacons for the other to hear. Module 1, then, will see Module 2 in its FOV, receive pings from Module 2’s beacons, and run estimation on these ping receptions to measure its position with respect to Module 2. Module 2 will do the mirror-image operation – estimate its position with respect to Module 1 by listening to Module 1’s pings. This configuration requires estimators to run on both modules, but requires minimal communication.

Another configuration is for only one module to ping its beacons. Module 1 would ping, Module 2 would receive and estimate, and then Module 2 would transmit the estimated state to Module 1. This way, both modules have relative state estimates, but only one set of beacons pings and only one estimator needs to run.

SIFFT used this latter estimation configuration, but with three modules. One module, Module 1, used its beacons to emit pings, and the other two used their receivers, measured the pings, and estimated their position with respect to Module 1. Modules 2 and 3 did not directly estimate their position with respect to each other. This estimation configuration adds the constraint that Modules 2 and 3 must remain inside the beacon cone of Module 1, as shown in Figure 4-17.

4.5.3 Test plan

The test procedure is largely as described above. Module 1 pings its beacons. Modules 2 and 3 begin their estimators, and upon convergence of their estimators, they move to their state targets. These state targets are chosen to form a triangular formation and keep Modules 2 and 3 inside the beacon cone of Module 1. After a prescribed amount of time, the state targets change and Modules 2 and 3 move to their new targets. These new targets are chosen such that the formation expands to a larger triangle. The state targets for the small and large formations are shown in the following figures as green and red diamonds, respectively.

There are two degrees of difficulty in these tests. The first is for Module 1 to not move. Because the beacons are on Module 1, Module 1 defines the reference
Figure 4-17: In the estimation configuration used here, Modules 2 and 3 must remain inside the beacon cone of Module 1 in order to reliably hear the ultrasound pings. As before, circles represent receivers and crosses represent beacons.
frame. The easiest case would be for Module 1 to not even be floating. That way, the reference frame would remain fixed with respect to the floor, so state deviations in Modules 2 and 3 would be caused only by movement of Modules 2 and 3, not Module 1. If Module 1 were floating and drifting, the test should still work, but when the three modules came into formation, the formation CM would have a non-zero velocity. So, the first level of difficulty is for Module 1 to be fixed.

A second level of difficulty is for Module 1 to rotate during the formation reconfiguration. This is more difficult because the reference frame is attached to Module 1, so Modules 2 and 3 must estimate in a rotating frame.

4.5.4 Results with Module 1 Fixed

Figure 4-18 shows a sample result from one of the tests with Module 1 fixed. First, the modules initialize and the estimators converge, as shown in the purple parts of the trajectories in the figure. Then, the modules begin tracking to the first state targets, indicated by the green diamonds. The modules regulate their positions at those points for a fixed time, then the state targets switch to the red diamonds. The modules move to the new positions and regulate there.

The module on the 'top' in Figure 4-18 (the one with positive y coordinate) started nearly on its desired target. The one with negative y components, though, started slightly off in position and also started with a significant velocity. It was able to measure this and correct.

4.5.5 Results with Rotation of Module 1

When Module 1 is rotating, the reference frame in which Modules 2 and 3 estimate their positions is rotating as well, because the reference frame is attached to Module 1. If Modules 2 and 3 stayed in the same place on the floor, this movement of the reference frame would be perceived by Modules 2 and 3 as an increasing position error. In order to stay in the same location in the reference frame of Module 1, they would actually traverse a circle on the floor.
Figure 4-18: A SIFFT test where three modules enter into and maintain one formation, then move to a different formation. The black diamond at (0, 0) indicates the position of Module 1. The green diamonds indicate the target positions of the two modules in the first formation, and the red diamonds indicate the target positions in the second formation.
For this test, Module 1 held its position and attitude during formation 1, while Modules 2 and 3 estimated their states and regulated their positions to maintain formation 1. Then, after a prescribed amount of time, Module 1 rotated 10° about the vertical and Modules 2 and 3 updated their state targets as before.

Figure 4-19 shows a sample test with Module 1 performing a rotation at the time that the configuration changes. As before, the modules begin estimating and move to their state targets, the green diamonds. This time, the module with positive $y$ component began with a significant position error, and was able to correct. When the state targets switched from the green diamonds to the red diamonds, Module 1 rotated 10° clockwise (which took approximately 20 seconds). As this rotation starts, Modules 2 and 3 read the rotation as increasing position error, which is visible in the skew departure of the trajectories from the green diamonds. As the rotation continues, Modules 2 and 3 counter the rotation and end up at their correct targets.

In summary, the modules successfully estimated their position, moved to their initial state targets and maintained those positions, then correctly reconfigured to their new state targets and maintained those positions.

4.6 Conclusion

This chapter discussed the validation and testing of the relative estimator. First, a low level test of the ultrasound sensors themselves was carried out. This test showed that the sensors are accurate and reliable. Next, the estimator was coded in Matlab® and tested in simulation. This showed that the estimator was correctly written and was capable of producing accurate state estimates, even with incorrect initial conditions. Finally, the estimator was translated to C and implemented on SPHERES-related hardware for full integrated testing. This was also successful. It demonstrated that the estimator could be used for autonomous docking, as well as for formation maintenance and reconfiguration. There are still some ways it could be improved, though, and these will be discussed in the next chapter.
Figure 4-19: A SIFFT test where three modules enter into and maintain one formation, then Module 1 rotates while Modules 2 and 3 move to a different formation. The black diamond at (0, 0) indicates the position of Module 1. The green diamonds indicate the target positions of the two modules in the first formation, and the red diamonds indicate the target positions in the second formation.
Chapter 5

Future Work and Conclusions

5.1 Future Work

Development and refinement of the relative estimator is an ongoing process. Now that the estimator works at a basic level, several improvements and additions can be considered.

5.1.1 UDPs use Beacons and Receivers

Currently during relative estimation, one UDP has active beacons and the other UDP has active receivers. One way to improve the performance of the relative estimator is to have both UDPs activate their beacons and their receivers. This would increase the number of measurements taken each cycle from 9 to 18.

Because the estimator is run on one module, but measurements would now be taken on multiple modules, communication would be required. For example, if the chaser is doing the estimating, the target will have to transmit the readings from its receivers to the chaser. There are nine numbers to transmit per cycle. The chaser, then, will have the nine numbers that it received from the target and the nine numbers that it measured from its own receivers.

The notation to be used in this section is similar to that used in Chapter 3, but is extended slightly. It is summarized in Figure 5-1. The UDP on each module has
three beacons and three receivers, but for simplicity, the figure just shows one of each on each module. Also, there is no difference in principle between the modules shown on the top of the figure and the modules shown on the bottom of the figure; the reason for drawing two sets is to avoid clutter.

Expanding from 9 to 18 measurements mostly just involves expanding the size of some of the vectors and matrices involved. The basic equations of the Kalman filter remain the same. The part of the filter that needs to be changed is the calculation of the linearized $C$ matrix, $H$.

As before, we must find the function $h_k(x(t_k))$ which outputs the expected measurements given the state of the system, then differentiate that function producing the Jacobian $H$. The linearized $C$ matrix at each time step is the Jacobian at that time step,

$$H = C_k(\ddot{x}_k) = \left. \frac{\partial h_k(x(t_k))}{\partial x(t_k)} \right|_{x(t_k)=\ddot{x}_k} \tag{5.1}$$

The 18 measurements will be taken one beacon at a time, as in the 9 measurement...
case, so for each global update, the filter will be run six times because there are six beacons. Each time the filter is run, the vector $h$ will contain separations between beacons and receivers, with the exact arrangement to be specified below. The task now is to calculate the distances between each beacon and each receiver given a state of the system.

The quantities $b_i^{TT}$, $s_j^{TT}$, $b_i^{CC}$, and $s_j^{CC}$ are known by the mechanical construction of the modules. Quantities expressed in the chaser's frame can be transformed to the target's frame by

\begin{align}
    s_j^{CT} &= Qs_j^{CC} + r \
    b_i^{CT} &= Qb_i^{CC} + r
\end{align}

These can then be subtracted to create the desired beacon-receiver separations

\begin{align}
    d_{Tij} &= b_i^{TT} - s_j^{CT} \
    d_{Cij} &= b_i^{CT} - s_j^{TT}
\end{align}

where $d_{Tij}$ is the separation vector between beacon $i$ on the target (subscript $T$) and receiver $j$ on the chaser, and $d_{Cij}$ is the separation vector between beacon $i$ on the chaser (subscript $C$) and receiver $j$ on the target.

Just as before, we can now pass $d_{Tij}$ to the estimator and run an update, then pass $d_{T2j}$, then $d_{T3j}$. This is where the current estimator would stop. With the current upgrade, though, all $d_{Cij}$ are now also available. The difficulty in making use of these is in recalculating the jacobian matrix $H$ for the new beacon locations. This can be avoided by rearranging the elements of the measurement vector $h$. In the first three updates, the ones for which the beacons are on the target, there is one update per beacon. For the next three updates, however, the measurements can be rearranged so that there is one update per receiver on the target. This is slightly non-intuitive, but assists in the calculation of the $H$ matrix for the last three updates. The reordering is shown graphically in Figure 5-2.

75
Figure 5-2: This figure shows how the measurements are ordered into $h$ vectors for each update.
Now, calculating $H$ for the second three updates can proceed similarly to the original case. For the first three updates, it can remain exactly the same.

For the first three updates, Block #1 (see page 39) can be calculated using

$$\frac{\partial h_i}{\partial r} = \frac{b_i^{CT} - s_i^{CT}}{h_i}$$

and for the second three updates, it is just

$$\frac{\partial h_i}{\partial r} = \frac{b_i^{CT} - s_i^{CT}}{h_i}$$

A parallel alteration is used to update the equations for Block #2. In the first three updates, Block #2 is calculated as before:

$$\frac{\partial h_i}{\partial q} = \frac{b_i^{CT} - s_i^{CT}}{\sqrt{\alpha}} \left( \frac{\partial Q}{\partial q} s_j^{CC} \right)$$

and for the second three updates, it is

$$\frac{\partial h_i}{\partial q} = \frac{b_i^{CT} - s_i^{CT}}{\sqrt{\alpha}} \left( \frac{\partial Q}{\partial q} s_j^{CC} \right)$$

where, as before, $\alpha$ is \((b_i^{TT} - s_i^{CT})^2 + (b_i^{TT} - s_i^{CT})^2 + (b_i^{TT} - s_i^{CT})^2\).

No code has yet been written to implement the use of beacons on both UDPs during relative estimation. In fact, the vector reordering method proposed here is only a suggestion of how to achieve the desired result, there are other possible ways to do it.

The first step to an implementation is expanding the Matlab® simulation. Additional loops will need to be added to handle the second three updates, and the code will need to be modified to accommodate the use of the vector reordering algorithm. The Matlab® simulation is where most of the ‘meaningful’ changes will be made and theory related bugs will be worked out. The next step is to collect real data on the hardware and run it through the simulation. It should output the correct estimated states. Additionally, an analysis of the relative precisions of the 9-measurement and
18-measurement estimators could be carried out, both by analyzing the empirical standard deviations of the history of estimated states and also by analyzing the behavior of the $P$ matrix. Finally, the simulation would need to be translated to C-code for use on the SPHERES satellites. This would probably be accomplished by modifying the current C estimator. The C implementation would require including the communication required for the target to transmit its measurements to the chaser.

5.1.2 Star Tracker Mode

Another possible improvement to the relative estimator system would be to make use of external beacons in a 'star tracker' mode, to generate absolute attitude measurements. Several mission types require absolute attitude information, such as taking solar, stellar, or Earth images, or studying the Earth’s magnetic or atmospheric properties, for example. Additionally, while the UDP-based sensors can estimate the full relative attitude, they are less sensitive in roll than in the other axes. An absolute attitude measurement ability could help improve the relative roll measurement.

To measure absolute attitude, one or more external beacons would be placed around the test area at known locations. These beacons are electronically the same as the beacons in the UDP, except they are battery powered and have built in structural mounts. They can be assigned beacon numbers using a small selector switch on the beacon face. A picture of an external beacon mounted in the lab at MIT is shown in Figure 5-3. SPHERES satellites have receivers on all sides, not just on the UDP face, so regardless of the orientation of the satellite, the external beacons would be heard by some receivers.

The setup here is for the chaser satellite (the one that does the estimation) to run two estimators in parallel – one that estimates the relative state to the other module, as before, and a second one that estimates the absolute attitude of the chaser module itself. The beacon setup is shown in Figure 5-4.

A particularly useful application of the star tracker estimator is at the flat floor at MSFC. When operating at MSFC, there is no global beacon setup as there is in the lab at MIT, and creating such a setup would be difficult because each beacon must
Figure 5-3: An external beacon, mounted in the lab at MIT. The white box at the bottom distributes electrical power to this beacon and several others in the lab setup.

Figure 5-4: This diagram shows the beacon/receiver arrangement to be used in the star tracker estimator. The cross on the right represents the external beacon. Dotted lines represent beacon-receiver distance measurements. Receivers shown on the other faces of the chaser represent the receivers that are built into the SPHERES satellites.
be precisely positioned. However, if instead of setting up five beacons, only one were set up and a star tracker mode were used, the satellites could have absolute attitude information based on that beacon. This seems to be a good balance between the complexity of setting up beacons and obtaining measurable state information. So, this section will outline a single beacon star tracker estimator for use at MSFC.

This estimator will be running in parallel with the relative estimator, so it can be built and analyzed separately. In this case, the state vector is only four elements long. The MSFC flat floor is a 2D environment, and representing attitude in 2D only requires one number. The estimator also needs to measure the distance to the beacon, so the state vector contains the range and bearing to the global beacon, as well as their derivatives

\[
\mathbf{x} = \begin{bmatrix} r & \dot{r} & \theta & \omega \end{bmatrix}^T
\]  

(5.10)

where \( r \) will be the distance from the center of the satellite to the beacon, \( \theta \) will be the angle that the UDP makes with the beacon and \( \omega \) will be the rotation rate.

The Kalman filter can be built in a manner similar to the relative estimator of Chapter 3. The basic equations will stay the same but again the function \( h_k(x_k) \) will need to be changed. Proceeding as before, the goal is to calculate what the measurements would be, given an estimated state of the satellite.

The receiver positions on the chaser are \( \mathbf{s}_j^{GC} \). The task now is to find the position of the global beacon in the frame of the chaser using the information in the state vector. This position is just

\[
\mathbf{b}^{GC} = \begin{bmatrix} r \sin(\theta) \\ r \cos(\theta) \end{bmatrix}
\]  

(5.11)

where the superscript \( GC \) on \( \mathbf{b}^{GC} \) indicates the position of the global beacon (G) in the frame of the chaser (C). The measurements that will be collected, then, are just the distance between beacon and receiver,
Figure 5-5: With a single beacon star tracker, the SPHERES satellite could be in any of these three places, and the measurements would be the same in each case. Attitude is measured relative to the global beacon.

\[
d_j = b^{GC} - s_j^{CC} \quad (5.12)
\]

\[
h_j = |d_j| \quad (5.13)
\]

Of course, in this single beacon star tracker estimator, the beacon functioning as the star is not 'infinitely' far away as it would be for an on-orbit application. This creates a situation shown in Figure 5-5, where the attitude that the satellite measures depends on its position. The satellite measures attitude by calculating the angle to the beacon, which does not change in any of the three cases shown in the figure. There is still an absolute attitude measurement available, though, and this will likely be useful for many applications.

If, however, this coupling is undesirable, there are several ways to mitigate it. The best way is to place a second absolute beacon. This way, the chaser would be able to see two global beacons, which would provide enough information to calculate absolute position, velocity, attitude, and rotation rates. Again, the chaser would run two parallel estimators – one measuring this global state and the other measuring the relative state to the target. This can be used to more accurately simulate an on-orbit
star tracker. With two global beacons, attitude information can be collected which does not have the degeneracies of Figure 5-5, and the position information that comes with it (which would not be available from a real star tracker) can be disregarded.

5.1.3 Full Global

An expansion of the use of external beacons from just one in the case of the star tracker to a full five or more is another possible extension of the estimation system. The concept is to set up a full set of global beacons so that each SPHERES satellite can robustly measure its global position and attitude, and also make use of the relative beacons and receivers on the UDPs. If both modules could only measure their absolute positions, then the relative state could be calculated by subtracting the state vectors of the two modules\(^1\), but a more accurate measurement could be created by using both the global system and the relative system.

There are at least two ways to do this. The first way is for each module to run several separate estimators in parallel. Both modules would run an estimator to measure their absolute positions in the global frame, and also a separate estimator to measure the relative position to the other satellite. This would produce four different state estimates – the absolute position of the target, the absolute position of the chaser, the position of the chaser relative to the target, and the position of the target relative to the chaser. These four estimates could then be combined, possibly treating them as measurements and giving them as inputs to a fifth estimator, to create an overall state estimate.

A second way to use global and relative measurements is to take all the raw ultrasound time-of-flight data from both modules and feed it to a single estimator. This is a more centralized approach to estimation because it requires all modules to transmit all their raw data to one ‘estimation master’ which does all the calculation then transmits state estimates back to the various modules. There is also more

\(^1\)Actually, it would be a little more complicated than a straight subtraction, because quaternions can not be simply added and subtracted, but the procedure is still simple. See Appendix B for more detail on quaternion operations.
data to transmit in this case; full distance matrices from both global and relative beacons are large (combined, approximately 736 bytes at 5 Hz) compared to just transmitting state vectors (approximately 52 bytes at 5 Hz) as in the first approach. This centralized approach would probably yield a better state estimate, but because of the increased communication, it would probably also introduce more delay.

There are several challenges facing a combined global and relative estimation system. One of these challenges is the number of available beacons. From experience in the Space Systems Lab at MIT, at least five global beacons are required for robust, precise, and error-tolerant absolute estimation. To measure the full 3D relative state between two modules, a minimum of three beacons is required, and experience shows that when the modules are favorably aligned, three is sufficient. However, the hardware and low-level software of the ultrasound receivers limits the total number of beacons to nine. Increasing this number would require a significant change to the receivers and their electronics, and would also have the adverse effect of further increasing the time for an estimation cycle (recall that each beacon requires a 20 ms window). Both of these issues can be overcome if more than nine beacons are definitely required, but it is desirable to find a solution using nine or fewer. If there are five global beacons and only one module has relative beacons, then there are eight total. However, if both modules have relative beacons, then there are 11. If a combined global and absolute system is desired, and having relative beacons on both modules is also desired, then either the total number of beacons will need to be increased, or a way will need to be found to conserve beacons numbers and stay under nine.

The first step would be to carry out an analysis to determine how much more accurate a combined estimator would be compared with straight global subtraction of state vectors. If it is determined that such an estimator is desired, a decision must be made between the first and second implementation, or a possible third idea.

5.1.4 Use Thruster Commands and Accelerometers

Currently, the propagator in the relative estimator does not use information on thruster firings or measurements from accelerometers to improve its propagation.
The propagator just takes current estimates regarding velocity and rotation rates and propagates the position and attitude forward accordingly for a specific length of time. Therefore, this propagation is only correct if there are no thruster firings or other accelerations.

Thruster firings could be used to improve the accuracy of the propagation by including additional terms in the calculation. The propagator as it is now basically propagates state $\mathbf{r}_1$ forward a time $\Delta t$ to state $\mathbf{r}_2$ using

$$
\mathbf{r}_2 = \mathbf{r}_1 + \mathbf{v} \Delta t
$$

where $\mathbf{v}$ is the fourth, fifth, and sixth elements of the state vector $\mathbf{v} = [v_x \ v_y \ v_z]^T$. If the known thruster commands were converted into accelerations using the mass of the module, then the propagator could be updated to

$$
\mathbf{r}_2 = \mathbf{r}_1 + \mathbf{v} \Delta t + \frac{1}{2} \mathbf{a} \Delta t^2
$$

where $\mathbf{a} = [a_x \ a_y \ a_z]^T$.

This, of course, is only a notional description of how the thruster commands would be used. One way it is incomplete is that it assumes that the thrusters are open for the entire propagation time, which they usually are not. Usually, the thrusters are open for small bursts, so this must be accounted for. It is also possible that for long thruster firings, the module will change attitude significantly during the firing, not just before and after it. If the module is attempting to alter its translational speed while also spinning, the acceleration vector will not point in the same direction throughout the maneuver. This also may need to be incorporated into the propagator.

Finally, because the thruster commands and accelerometers give similar, but not identical information, they could be used to do error checking on each other. In an ideal situation, with perfect thrusters, perfect accelerometers, and no external disturbances, the accelerometers would give the same information as the thruster firing times, because the thrusters are the only things that can make the module accelerate. In reality, though, thrusters can fail open or closed, accelerometers can
fail, astronauts can bump the SPHERES causing accelerations, and friction on the flat floor can cause accelerations. Perhaps the commanded thruster firing times should only be used in the propagator if they are consistent with the measured acceleration from the accelerometers.

5.2 Conclusion

The central focus of this thesis is on an EKF-based relative state estimator for modular autonomous spacecraft. This estimator measures the relative position, velocity, attitude, and rotation rate of modules in a formation. Overall, the estimator works well. Low level tests of the sensors proved their usefulness and precision, static tests running the estimator in simulation proved the full implementation, and full dynamic tests on the actual hardware were successful. These results are encouraging, and there are several ways the estimator could be improved, including using the thruster firings, implementing a star tracker mode, and using beacons on both UDPs.
Appendix A

Selected Code Segments

Some of the Matlab\textsuperscript{®} code for the relative estimator is included in this appendix.\textsuperscript{1} The Matlab\textsuperscript{®} code is shown here instead of the C code because the Matlab\textsuperscript{®} code is cleaner and easier to understand. The translation to the actual C that runs on the hardware is a mostly routine process and does not provide additional insight regarding the functioning of the estimator.

A.1 Overall Estimator Cycle

This code runs through the overall propagate/update cycle of the Kalman filter, and includes a few simple prefilters. First, the function \texttt{pads\_statePropPvar()} propagates the previous state (passed in as \texttt{xestHistory(:,end)}) and the previous covariance matrix (passed in as \texttt{PHistory{end}}) forward to the current time, and stores the results in \texttt{xest} and \texttt{P}. The next few lines store various quantities for post-analysis.

Next, two prefilters remove distance matrices consisting of all zeros, and distance matrices in which some measurements are too far from each other.

The second main step in the filter process is the update segment. The function \texttt{pads\_stateUSBeaconDetPvarEKF()} takes the distance matrix \texttt{dist\_matrix}, the beacon number that it came from, and the current state and covariance matrix

\textsuperscript{1}The code for the relative estimator is adapted from code originally written for SPHERES by Simon Nolet or Edmund Kong.
(xestHistory(:,end) and PHistory(end)) and outputs the updated state xest and covariance matrix P. Again, these are stored for post-analysis.

A condensed version of the code for this process is shown here:

```matlab
[xest, P] = pads_statePropPvar(time, xestHistory(:,end), PHistory{end}, ...
    fEstimatorMode);
xestHistory = [xestHistory xest];
PHistory{length(PHistory)+1} = P;
timeHistory = [timeHistory time-timeInit];
biasHistory = [biasHistory padsCountsBiasGet()];
beaconHistory = [beaconHistory beacon];
dist_matrix = dist_matrix([1 2 4]);

% prefilter - if the measurements are all zero, reject distance matrix
if max(dist_matrix) == 0
    dist_matrix = [0 0 0];
end

% prefilter - if the measurements are too far from eachother, reject ...
    distance matrix
if max(dist_matrix - mean(dist_matrix)) > 0.05  % tolernace 5 cm
    dist_matrix = [0 0 0];
end

[xest,P] = pads_stateUSBeaconDetPvarEKF(beacon,dist_matrix,PHistory{end},...
    xestHistory(:,end), fEstimatorMode);
xestHistory = [xestHistory xest];
PHistory{length(PHistory)+1} = P;
timeHistory = [timeHistory time-timeInit];
biasHistory = [biasHistory padsCountsBiasGet()];
beaconHistory = [beaconHistory beacon];
```

A lot of this thesis concerns calculations contained in `pads_stateUSBeaconDetPvarEKF()`. The next few sections will give more detail on this function and its subfunctions.
A.2 pads_stateUSBeaconDetPvarEKF()

This function runs the update portion of the Kalman filter. It contains the linearization of the $C$ matrix into the Hamiltonian $H$ as described in the text.

Coming into this function, beacon is the ID number of the beacon that just pinged, dist_mat is a 3-element vector consisting of the distances from beacon beacon to each of the three receivers, Pkm is $-P_k$, and xkm is $-\hat{x}_k$.

If all three measurements in dist_mat are non-zero, the filter proceeds on, otherwise, the update is ignored and the previous values are kept. In this case, $+P_k$ would be set to $-P_k$, and $+\hat{x}_k$ would be set to $-\hat{x}_k$.

```matlab
function [xkp,Pkp] = pads_stateUSBeaconDetPvarEKF(beacon,dist_mat,...
Pkm,xkm,fEstimatorMode)

% Global variables
global Rvar  % variance on distance measurements
xkmsub = xkm;

% Measurement Update
if length(find(dist_mat))==3   %only accept full distance matrices

% Form y vector
[rxNum y nGoodMeasurements] = find_distmat(dist_mat);

R = eye(length(y))*Rvar;
[h,H,S] = pads_hgenUSBeaconUpdatePvarEKF(beacon, xkm, rxNum);
Ltild = Pkm*H'*inv(H*Pkm*H'+R);
L = S*Ltild;
xkp = xkmsub + L*(y-h)';
q = xkp(7:10);
if q(4)<0
    q(1) = -q(1);
    q(2) = -q(2);
    q(3) = -q(3);
    q(4) = -q(4);
end
q = q/sqrt(sum(q.^2));
xkp(7:10) = q;
```
The function `find_distmat()` takes the elements in `dist_mat` and puts them into the measurement vector `y`. Then, the matrix `R` is created. The noise properties of all the sensors is assumed to be the same, so the matrix is a purely diagonal matrix with `Rvar` on each diagonal entry.

The next step is to calculate the Hamiltonian `H`. This is done by the function `pads_hgenUSBeaconUpdatePvarEKF()`. This function takes in the beacon and receiver identifications and the estimated state \( \hat{x}_k \), and returns the vector of beacon-receiver separations `h` and its derivative \( \frac{\partial h}{\partial x} \). This is a complicated function which is detailed further in the next section.

Next, the Kalman gains are computed using

\[
    L_k = \hat{P}_k H^T \left( H \hat{P}_k H^T + R \right)^{-1}
\]

and the state is updated using

\[
    \hat{x}_{k+1} = \hat{x}_k + L_k (y_k - H \hat{x}_k)
\]

These are the standard Kalman equations discussed in Chapter 3, with the the linearized `C` matrix `H` substituted in.

Numerical errors in the calculations up to this point can drive the quaternion out of normalization, so the next few lines renormalize the quaternion and ensure that the fourth element \( q_4 \) is positive. The normalization sets the magnitude to 1:

\[
    q = \frac{q}{|q|}
\]
where

\[ |q| = \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2} \]

Finally, the covariance matrix is updated using

\[ +P_k = (I - L_kH) - P_k \]

which, again, uses the proper linearized \( C \) matrix \( H \). The calculations up to this point can introduce numerical errors which may cause \( +P_k \) to be non-symmetric. The code removes these errors and forces the matrix to be symmetric by averaging it with its transpose:

\[ +P_k = \frac{+P_k + +P_k^T}{2} \]

This concludes a global measurement cycle of the relative estimator.

A.3 pads_hgenUSBeaconUpdatePvarEKF()

This function calculates the vector of beacon-receiver separations \( h \) and its derivative with respect to the state vector \( H = \frac{\partial h}{\partial x} \).

function [h, Htild, S] = pads_hgenUSBeaconUpdatePvarEKF(beacon, xkm, rxNum)

global STATE_LENGTH TX_POS

[Qr,dQdx] = Qrot(xkm); %gives rotation matrix, chaser to target
htemp = zeros(length(rxNum),STATE_LENGTH); %going to become Jacobian of \( h \)
[rxPos rxPosx] = pads_rxPos(xkm, Qr); %gives rcvr positions in both frames

for i=1:length(rxNum) %number of good measurements

    % rcvPos becomes vector of position of receiver \# i in target frame
    % rcvPosx becomes vector of position of receiver \# i in chaser frame
    for j=1:3
        rcvPos(j,1) = rxPos(j,rxNum(i));
        rcvPosx(j,1) = rxPosx(j,rxNum(i));
    end

end
h(i) = 0.0;
for j=1:3
    txrxSepVec(j) = rcvPos(j) - TX_POS(beacon,j);
    h(i) = h(i) + txrxSepVec(j)*txrxSepVec(j);
end
h(i) = sqrt(h(i));
% h(i) is now the estimated distance from beacon # beacon to receiver # i

% now, find Jacobian of h
% block 1
for j=1:3
    htemp(i,j) = txrxSepVec(j)/h(i);
end

% block 2
for j=1:4
    ctemp = dQdx(:,j)*rcvPosx;
    htemp(i,j+6) = 0.0;
    for k=1:3
        htemp(i,j+6) = htemp(i,j+6) + txrxSepVec(k)*ctemp(k);
    end
    htemp(i,j+6) = htemp(i,j+6) / h(i);
end
end
H = htemp;

First, the function calculates the rotation matrix $Q$ (called $Qr$ in the code) based on the propagated state $\dot{x}_k$. This matrix rotates vectors expressed in the chaser's frame to expression in the target's frame. Then, the function calculates the receiver positions in both frames. (These would be called $s_{ij}^{CC}$ and $s_{ij}^{CT}$ in the notation of Chapter 3). $rxPos$ is the receiver positions in the target's frame ($s_{ij}^{CT}$), and $rxPosx$ is the receiver position in the chaser's frame ($s_{ij}^{CC}$). The functions that perform these calculations are $Qrot()$ and $pads_rxPos()$. $pads_rxPos()$ simply implements

$$s_{ij}^{CT} = Qs_{ij}^{CC} + r$$

and $Qrot()$ will be discussed in the next section of this appendix.
Next, the function loops through each receiver. Each run through the loop generates an entry in $h$ and a row in $H$. The first half of the for loop calculates the vector from beacon to receiver using

$$d_{ij} = b_i^{TT} - s_j^{CT}$$

and simply calculates the magnitude of the distance, implementing

$$h_j = |d_{ij}|$$

The second half of the for loop calculates $H$. There is a simple loop to calculate block # 1, using

$$\frac{\partial h_i}{\partial r} = \frac{b_i^{TT} - s_j^{CT}}{h_i}$$

and a loop to calculate block # 2, using

$$\frac{\partial h_i}{\partial q} = \frac{b_i^{TT} - s_j^{CT}}{\sqrt{\alpha}} \left( \frac{\partial Q}{\partial q} s_j^{CC} \right)$$

A.4 Qrot()

This function takes a state vector and returns a rotation matrix rotating vectors from the chaser’s frame to the target’s frame.

A rotation matrix calculated from quaternion elements is:

$$Q(q) = \begin{bmatrix}
q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_3 - q_2q_4) & 2(q_1q_4 + q_2q_3) \\
2(q_1q_2 + q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 - q_1q_4) \\
2(q_1q_3 + q_2q_4) & 2(q_2q_3 + q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2
\end{bmatrix}$$

As discussed in Chapter 3, the derivative of this matrix with respect to each element of the quaternion, $\frac{\partial Q}{\partial q}$, is also required. Each of these differentiations is simple, and is contained in the function Qrot.
function [Q,dQdx] = Qrot(x)

q1 = x(7);
q2 = x(8);
q3 = x(9);
q4 = x(10);

Q(1,:) = [q1^2-q2^2-q3^2+q4^2, 2*(q1*q2-q3*q4), 2*(q1*q3+q2*q4)];
Q(2,:) = [2*(q1*q2+q3*q4), -q1^2+q2^2-q3^2+q4^2, 2*(q2*q3-q1*q4)];
Q(3,:) = [2*(q1*q3-q2*q4), 2*(q2*q3+q1*q4), -q1^2-q2^2+q3^2+q4^2];
Q = diag([-1 -1 1])*Q;

dQdx(1,:,1) = -[ q1, q2, q3];
dQdx(2,:,1) = -[ q2, -q1, -q4];
dQdx(3,:,1) = [ q3, q4, -q1];

dQdx(1,:,2) = -[-q2, q1, q4];
dQdx(2,:,2) = -[ q1, q2, q3];
dQdx(3,:,2) = [-q4, q3, -q2];

dQdx(1,:,3) = -[-q3, -q4, q1];
dQdx(2,:,3) = -[ q4, -q3, q2];
dQdx(3,:,3) = [ q1, q2, q3];

dQdx(1,:,4) = -[ q4, -q3, q2];
dQdx(2,:,4) = -[ q3, q4, -q1];
dQdx(3,:,4) = [-q2, q1, q4];

dQdx = dQdx*2;
Appendix B

Quaternion Attitude Representation

Any 3D rotation of a rigid body can be expressed by specifying a single axis of rotation $\tau$ and a magnitude of rotation $\theta$ around that axis. [19] This “axis and angle” representation only represents rotations of an object from one position to another position. It can be used to represent the attitude of a module by calculating the rotation that would be required to move the module from a fixed ‘reference’ attitude to its current attitude. This appendix will introduce the basics of quaternions and show how they can be used to implement an “axis and angle” representation of attitude.

B.1 Quaternion Composition and Properties

A quaternion is a 4-dimensional hyperimaginary vector quantity. It contains three different imaginary components and one real component; the basis elements are $i$, $j$, $k$, and 1. For comparison, the familiar ‘complex number’ has one imaginary and one real part – a basis of $i$ and 1. The quaternion just extends basic complex numbers to include two more complex elements.
The numbers $i$, $j$, and $k$ are all distinct square roots of $-1$:

$$i^2 = j^2 = k^2 = -1 \tag{B.1}$$

$$ij = k \tag{B.2}$$
$$jk = i \tag{B.3}$$
$$ki = j \tag{B.4}$$

The multiplication of these imaginary numbers is anticommutative:

$$ij = -ji \tag{B.5}$$
$$jk = -kj \tag{B.6}$$
$$ki = -ik \tag{B.7}$$

A quaternion $q$ contains elements of $i$, $j$, $k$, and 1, so

$$q = q_1 i + q_2 j + q_3 k + q_4 \tag{B.8}$$

It is useful to place the real coefficients into a vector:

$$q = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}^T \tag{B.9}$$

Now, we interpret the elements of a quaternion to represent the axis-and-angle attitude of a module using

$$q = \begin{bmatrix} \tau \sin \left( \frac{\theta}{2} \right) \\ \cos \left( \frac{\theta}{2} \right) \end{bmatrix} \tag{B.10}$$

where $\tau$ is a unit vector in the direction of the rotation axis and $\theta$ is the angle of rotation about that axis. For example, a $20^\circ$ rotation ($\theta = 20^\circ$) around the $y$ axis
would be represented by

\[
q = \begin{bmatrix}
\tau \sin \left( \frac{\theta}{2} \right) \\
\cos \left( \frac{\theta}{2} \right)
\end{bmatrix}
= \begin{bmatrix}
0 \sin (10^\circ) \\
1 \sin (10^\circ) \\
0 \sin (10^\circ) \\
\cos (10^\circ)
\end{bmatrix}
= \begin{bmatrix}
0 \\
0.1736 \\
0 \\
0.9848
\end{bmatrix} \quad (B.11)
\]

It may be tempting to simply use a four element vector containing the raw axis and angle \([\tau \; \theta]^T\) instead of using trigonometric functions and hyperimaginary quantities as are needed for the quaternion representation in Equation B.10. However, the quaternion representation has some useful properties that make it worthwhile. One of these properties is that it always normalizes to unity:

\[
|q| = \sqrt{q^T q}
= \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2}
= \sqrt{\tau^T \tau \sin^2 \left( \frac{\theta}{2} \right) + \cos^2 \left( \frac{\theta}{2} \right)}
= \sqrt{\sin^2 \left( \frac{\theta}{2} \right) + \cos^2 \left( \frac{\theta}{2} \right)}
= 1 \quad (B.12)
\]

Combining two sequential rotations is also straightforward using the quaternion representation. Let one rotation be represented by the quaternion \(q_a\) and another rotation be represented by \(q_b\). The attitude of a module after experiencing rotation \(q_a\) followed by rotation \(q_b\) would be \(q_{ab} = q_a q_b\). It can be shown using Equations B.1 through B.7 that:
\[
\mathbf{q}_{ab} = \mathbf{q}_a \mathbf{q}_b \quad (B.13)
\]

\[
= i(q_{a1}q_{b4} + q_{a2}q_{b3} - q_{a3}q_{b2} + q_{a4}q_{b1})
+ j(-q_{a1}q_{b3} + q_{a2}q_{b4} + q_{a3}q_{b1} + q_{a4}q_{b2})
+ k(q_{a1}q_{b2} - q_{a2}q_{b1} + q_{a3}q_{b4} + q_{a4}q_{b3})
- q_{a1}q_{b1} - q_{a2}q_{b2} - q_{a3}q_{b3} + q_{a4}q_{b4} \quad (B.14)
\]

It is also sometimes useful to have a $3 \times 3$ rotation matrix to rotate vectors from one reference frame to another. The matrix $\mathbf{Q}(\mathbf{q})$ would rotate vectors expressed in the unrotated standard frame to vectors expressed in the rotated frame. This matrix can be calculated from the elements of the quaternion using Equation 3.28 (on page 41).

Note that sequential rotations have a different order of operation when represented by quaternions then when represented by rotation matrices. Rotation $a$ followed by rotation $b$, in both notations, would be:

\[
\mathbf{q}_{ab} = \mathbf{q}_a \mathbf{q}_b \quad (B.15)
\]

\[
\mathbf{Q}(\mathbf{q}_{ab}) = \mathbf{Q}(\mathbf{q}_b) \mathbf{Q}(\mathbf{q}_a) \quad (B.16)
\]

## B.2 Quaternion Propagation

The quaternion $\mathbf{q}$ in the state vector $\mathbf{x}$ represents attitude, but during the propagation phase of the estimator cycle, this attitude must be propagated forward using the estimated rotation rates $\mathbf{\omega}$. In other words, given a current attitude $\mathbf{q}$, the propagator must find what the attitude would be if the module rotated with angular rates $\mathbf{\omega}$ for a given length of time. This can be done by calculating $\dot{\mathbf{q}}$ (as in [13]):

\[
\dot{\mathbf{q}} = \frac{1}{2} \mathbf{\Omega}(\mathbf{\omega}) \mathbf{q} \quad (B.17)
\]
where \( \Omega(\omega) \) uses the body rotation rates

\[
\omega = \begin{bmatrix}
\omega_x \\
\omega_y \\
\omega_z 
\end{bmatrix}
\]  \hspace{1cm} \text{(B.18)}

to transform the elements of \( q \) into \( \dot{q} \), and

\[
\Omega(\omega) = \begin{bmatrix}
0 & \omega_z & -\omega_y & \omega_x \\
-\omega_z & 0 & \omega_x & \omega_y \\
\omega_y & -\omega_x & 0 & \omega_z \\
-\omega_x & -\omega_y & -\omega_z & 0
\end{bmatrix}
\]  \hspace{1cm} \text{(B.19)}

Of course, \( \Omega(\omega) \) will need to be recalculated at each time step because the rotation rates \( \omega \) change with time.
Bibliography

[1] Massachusetts Institute of Technology, “SWARM design document, revision 3.2”
   http://spacestation.mit.edu/spheres/swarm/documents.html

   Assembly and Reconfiguration of Modular Space Systems” Masters Thesis, Mass-
   chusetts Institute of Technology, Department of Mechanical Engineering, De-
   cember 2005

   velopment and Verification of Separated Spacecraft Control Algorithms” Masters
   Thesis, Massachusetts Institute of Technology, Department of Aeronautics and
   Astronautics, June 2002

   and Initial Control” Masters Thesis, Massachusetts Institute of Technology, De-
   partment of Electrical Engineering and Computer Science, August, 2000

   ogy Maturation Laboratories Aboard the International Space Station” Doctoral
   Thesis, Massachusetts Institute of Technology, Department of Aeronautics and
   Astronautics


