Computer Vision-Based Localization and Mapping of an Unknown, Uncooperative and Spinning Target for Spacecraft Proximity Operations

Brent Edward Tweddle, David W. Miller

September 2013 SSL # 8-13

Computer Vision-Based Localization and Mapping of an Unknown, Uncooperative and Spinning Target for Spacecraft Proximity Operations

Brent E. Tweddle, David W. Miller

September 2013 SSL # 8-13

Computer Vision-Based Localization and Mapping of an Unknown, Uncooperative and Spinning Target for Spacecraft Proximity Operations

by

Brent Edward Tweddle

Submitted to the Department of Aeronautics and Astronautics
on September 18, 2013, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Prior studies have estimated that there are over 100 potential target objects near the Geostationary Orbit belt that are spinning at rates of over 20 rotations per minute. For a number of reasons, it may be desirable to operate in close proximity to these objects for the purposes of inspection, docking and repair. Many of them have an unknown geometric appearance, are uncooperative and non-communicative. These types of characteristics are also shared by a number of asteroid rendezvous missions.

In order to safely operate in close proximity to an object in space, it is important to know the target object's position and orientation relative to the inspector satellite, as well as to build a three-dimensional geometric map of the object for relative navigation in future stages of the mission. This type of problem can be solved with many of the typical Simultaneous Localization and Mapping (SLAM) algorithms that are found in the literature. However, if the target object is spinning with significant angular velocity, it is also important to know the linear and angular velocity of the target object as well as its center of mass, principal axes of inertia and its inertia matrix. This information is essential to being able to propagate the state of the target object to a future time, which is a key capability for any type of proximity operations mission. Most of the typical SLAM algorithms cannot easily provide these types of estimates for high-speed spinning objects.

This thesis describes a new approach to solving a SLAM problem for unknown and uncooperative objects that are spinning about an arbitrary axis. It is capable of estimating a geometric map of the target object, as well as its position, orientation, linear velocity, angular velocity, center of mass, principal axes and ratios of inertia. This allows the state of the target object to be propagated to a future time step using Newton's Second Law and Euler's Equation of Rotational Motion, and thereby allowing this future state to be used by the planning and control algorithms for the target spacecraft.

In order to properly evaluate this new approach, it is necessary to gather experimental data from a microgravity environment that can accurately reproduce the types of complex spinning motions that may be observed in actual space missions. While the Synchronize Position Hold Engage Reorient Experimental Satellites (SPHERES) can accurately represent these types of complex spinning motions, they did not previously have any onboard cameras (or any other similar sensors). This thesis describes an experimental testbed upgrade to the SPHERES satellites, known as the "Goggles", which adds computer vision-based navigation capability by the addition of stereo cameras and additional onboard computational power. The requirements, design and operation of this testbed is described in this thesis as well as the results of its first operations onboard the International Space Station (ISS).

The SPHERES Goggles testbed was used to capture a dataset of an unknown target object that was spinning at 10 rotations per minute about its unstable intermediate axis. This dataset includes reference measurements of both the inspector spacecraft and the target object with respect to an inertial frame. An implementation of the above algorithm was evaluated using this dataset and the resulting estimates were compared to reference metrology measurements. A statistical analysis of the errors is presented along with a comparison of the geometric and dynamic properties of the target object with respect to its known values. A covariance analysis on the convergence of the smoothing algorithm is also provided.

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

Thesis Supervisor: Alvar Saenz-Otero
Title: Principal Research Scientist, Aeronautics and Astronautics

Thesis Supervisor: John J. Leonard
Title: Professor, Mechanical Engineering

Thesis Supervisor: Larry H. Matthies
Title: Senior Research Scientist, NASA Jet Propulsion Laboratory

Acknowledgments

This work was performed primarily under contract NNH11CC25C, Visual Estimation and Relative Tracking for Inspection of Generic Objects (VERTIGO), with the Defense Advanced Research Projects Agency (DARPA) and the National Aeronautics and Space Administration (NASA) as part of the International Space Station SPHERES Integrated Research Program (INSPIRE).

Additional funding for this work was provided by the National Sciences and Engineering Research Council of Canada (NSERC) Postgraduate Scholarship, the 2009 Gordon M. MacNabb Scholarship (sponsored by MDA Corporation), the AIAA Foundation's 2011 Guidance, Navigation and Control Award and the NASA JPL Graduate Fellowship Program. The author gratefully thanks the sponsors for their generous support that enabled this research.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Autonomous spacecraft proximity operations is a challenging, complicated and multi-faceted field of study in astronautics. A typical problem is shown in Figures 1-1 and 1-2, where a Inspector Spacecraft is navigating around a Target Object. This field can be categorized in a number of different "mission applications." These applications can be grouped by objective (e.g. inspection, rendezvous or repair) as well as by a priori target knowledge (e.g. cooperative or uncooperative, known or unknown appearance or stationary or moving target). This thesis focuses on the problem of how to navigate around an unknown, uncooperative object that may also be moving or spinning with high speed.

As a result of the large number of possible mission applications, there is a broad history of spacecraft proximity operations both in terms of research and development as well as on-orbit missions. This chapter will review the current state of the art in spacecraft navigation, discuss motivating mission applications and provide an overview of the contributions and outline of this thesis.

1.1 Motivation

Since the primary goal of the proposed thesis is to investigate navigation techniques for spinning target objects in space, it is important to ask the following question: How many potential target objects in space can be considered spinning and not stationary

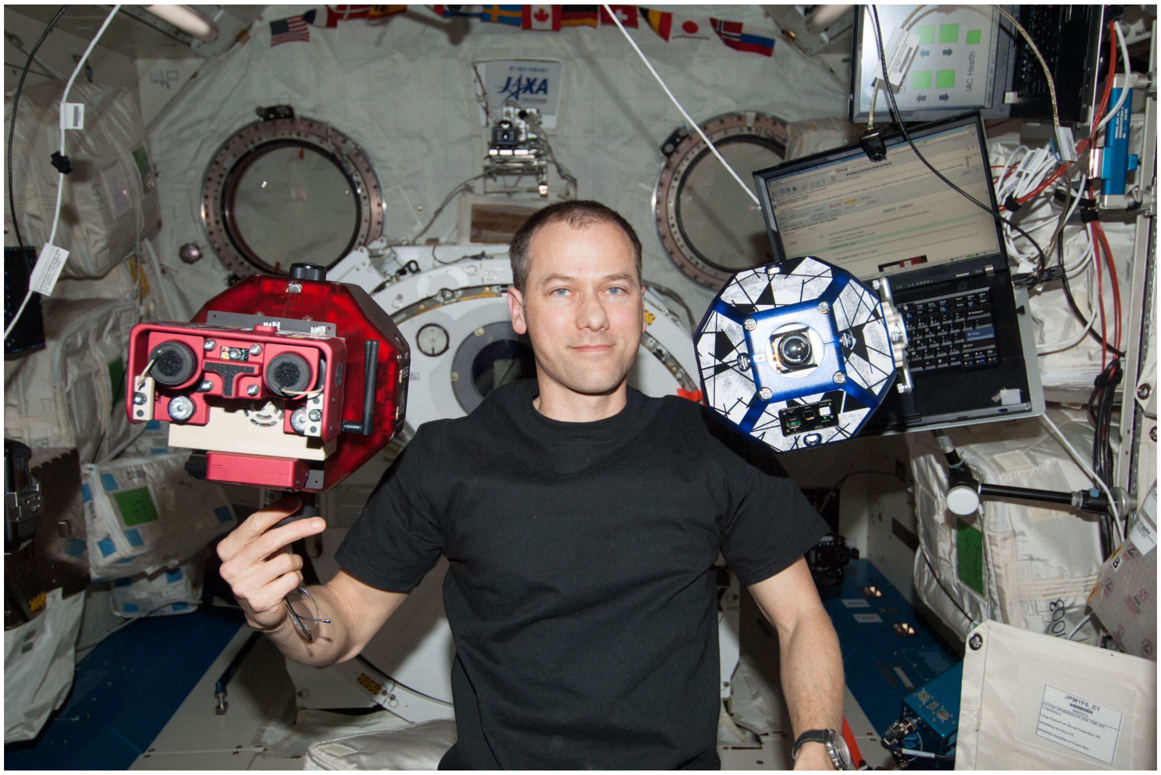


Figure 1-1: SPHERES VERTIGO Goggles onboard International Space Station: Inspector Satellite is Red, Target Satellite is Blue

or tumbling?

This is an important question because many spacecraft that are launched today are 3-axis stabilized (i.e. have no angular velocity by design). However, in the past many spacecraft have been spin-stabilized, such as the Hughes Spacecraft 376 [9, 17] shown in Figure 1-3. This spacecraft is 6.6 meters high by 2.16 meters in diameter and weights 654 kg at launch. It was spin-stabilized at 50 RPM about its minor axis. Additionally, a number of interplanetary missions are spin stabilized during the cruise phase of their mission. For example, DAWN, JUNO and MSL were spin-stabilized at 48, 5 and 2 RPM respectively.

Kaplan et. al. [59] published a study in 2010 to assess the required technologies to perform space debris capture. Included in this study was an analysis of the distribution of space objects and their angular velocities. The authors concluded that in orbits below 500 km, the gravity gradient and atmospheric pressure would provide enough torque to null the spin rates over long enough periods of time. However, in Geostationary Orbit (GEO) and near-GEO (i.e. graveyard orbits), the authors state

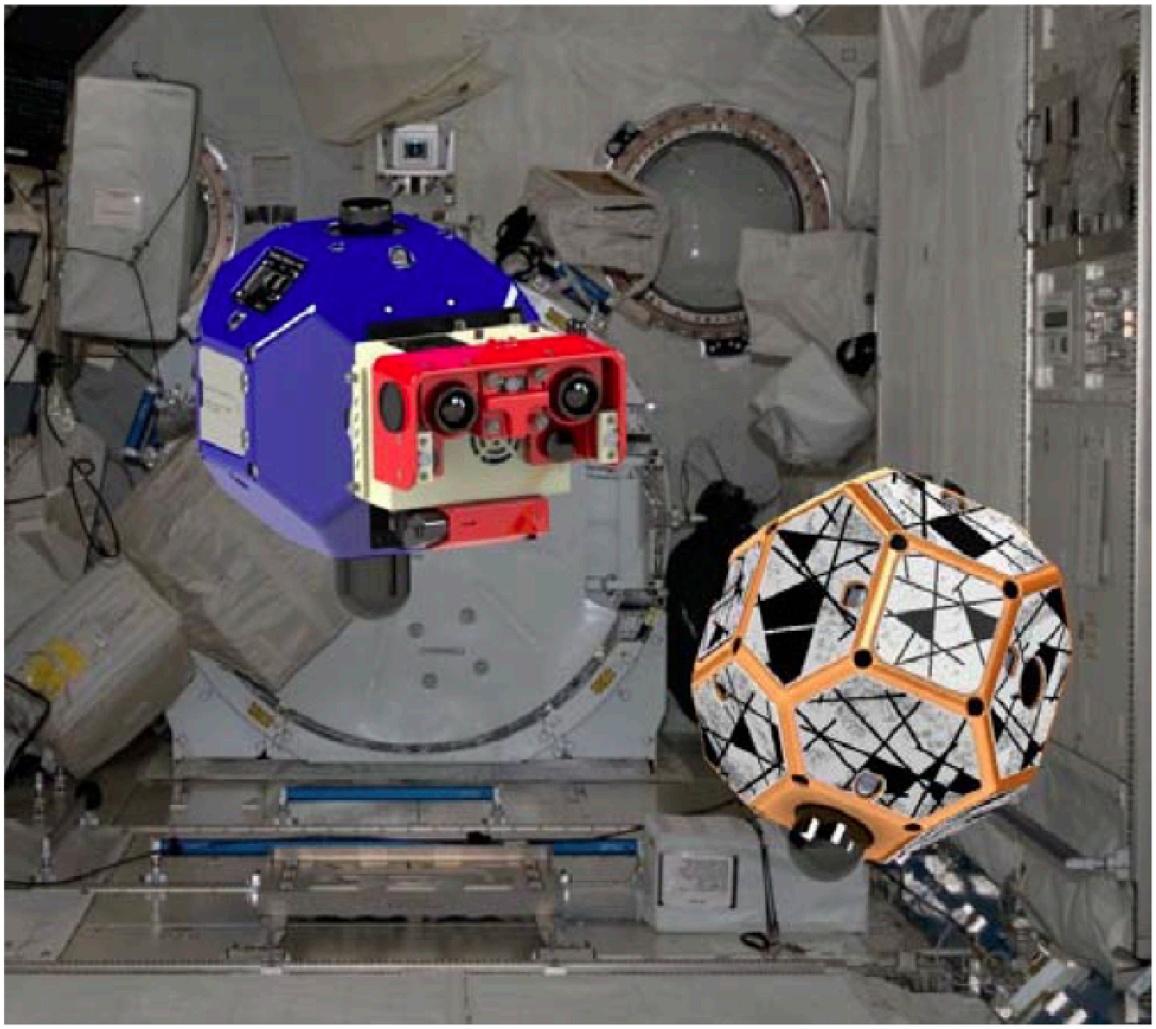


Figure 1-2: CAD Visualization of Proximity Operations Mission using the SPHERES Satellites: Inspector Spacecraft (left) is navigating around the Target Object (right)

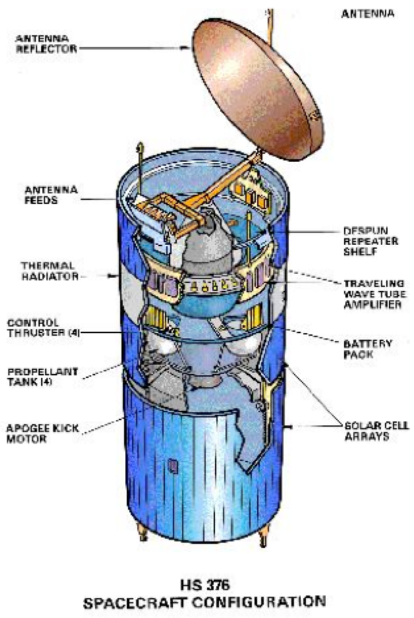


Figure 1-3: Hughes Spacecraft HS 376[9, 17]

that "it is reasonable to estimate that there are over 100 large expired satellites that are still rotating at several 10s of RPM."[59] The ability to perform proximity operations with spacecraft in GEO will become more important as time goes on since this is a very important location for the telecommunications and earth-observation satellite industry.

A number of missions would also be interested in performing proximity operations about an asteroid. Asteroids have many of the same unknown and uncooperative characteristics as disabled, or uncooperative spacecraft. In order to plan a mission to an asteroid, it would be important to understand (and possibly estimate) the asteroid's angular velocity. Cotto-Figeuroa's thesis [25] published a survey of rotation rates of near earth asteroids, the results of which are shown in Figure 1-4. This shows that there are a number of asteroids with diameters between 10 and 100 meters, that are rotating between 0.0167 and 1.67 rotations per minute.

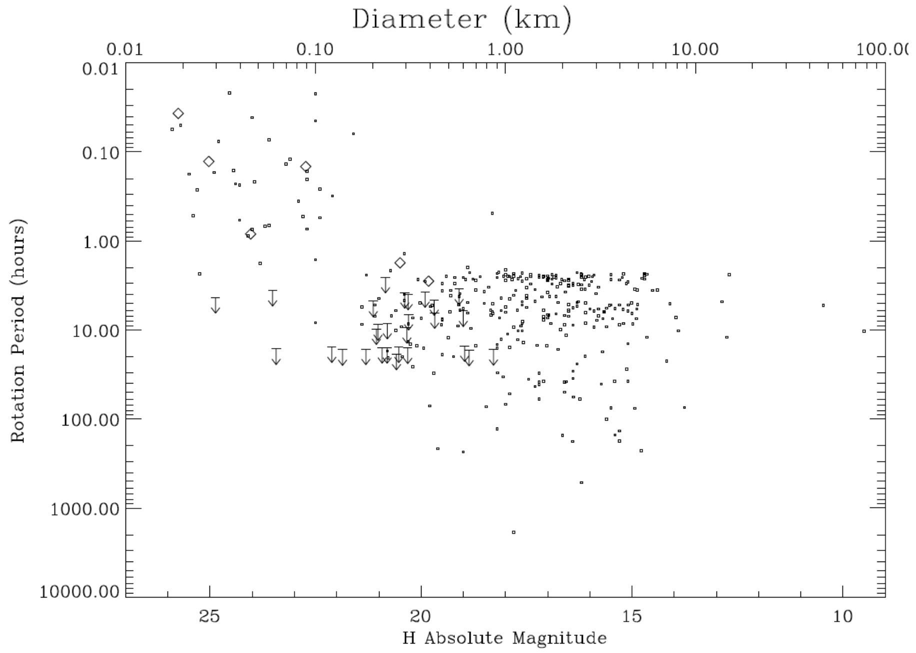


Figure 1-4: Rotation Rates and Diameters of Near Earth Asteroids from Cotto-Figueroa [25]

1.2 Tumbling versus Spinning Space Objects and the Importance of Angular Velocity

The following sections of this chapter will show that typical solutions to the Simultaneous Localization and Mapping (SLAM) problem do not provide angular velocities, which is problematic for the inspection of quickly rotating target objects. This section will discuss why the inspection of slowly rotating (i.e. tumbling) targets does not require angular velocity knowledge, while the inspection of quickly rotating (i.e. spinning) targets do require angular velocity knowledge. Additionally, a relationship for calculating the threshold between tumbling and spinning target objects based on fuel consumption and thruster saturation will be derived and discussed.

During many spacecraft proximity operations missions, it is likely that it would be necessary to perform closed loop control using relative sensors, such as cameras, flash LIDAR's etc., rather than global sensors such as Inertial Measurement Units, space-based Global Positioning Systems, star trackers, etc. This is especially true if the mission objective requires the inspecting spacecraft to make contact with the target object for the purpose of docking, repair, sample gathering etc.

If closed loop control of an inspector spacecraft is performed relative to a target object that is rotating, without any knowledge or control law compensation for this rotation rate, the inspector will follow a circular trajectory that is synchronized with the object's rotation rate, as shown in Figure 1-5. In this situation, the inspector will perceive this rotational motion as a translational and rotational disturbance and will apply a force and torque to correct it, thereby expending "valuable" fuel.

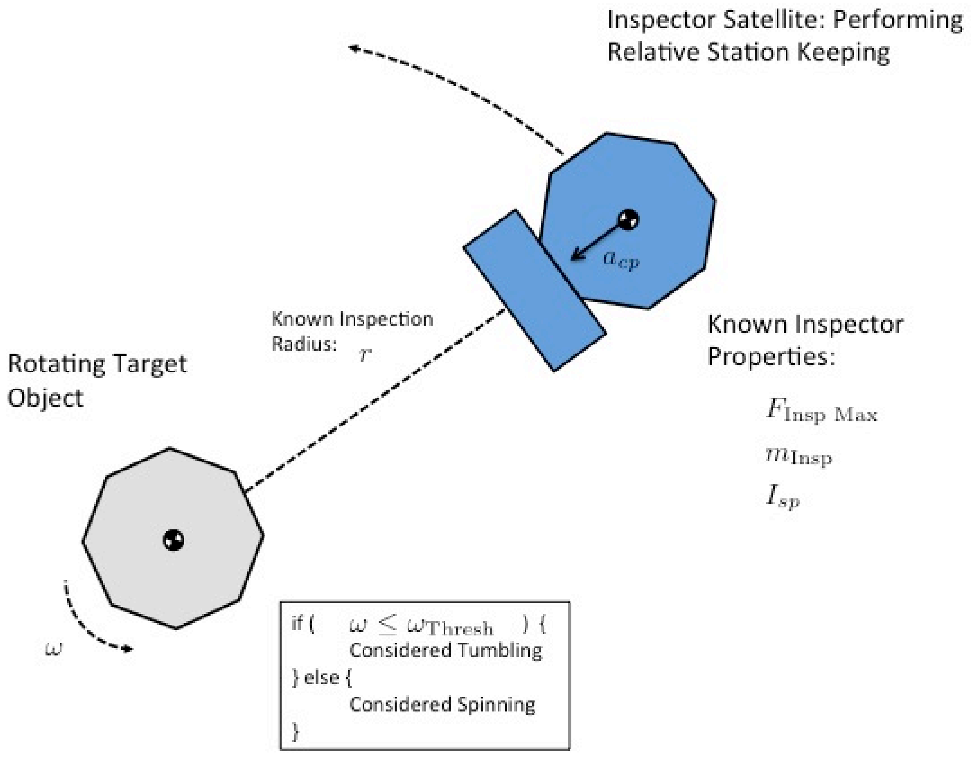

Inspector Satellite: Performing Relative Station Keeping
a_{cp}
Known Inspection Radius: r
Rotating Target Object
Known Inspector Properties:
$F_{\text{Insp Max}}$
m_{Insp}
I_{sp}
ω
if ($\omega \leq \omega_{\text{Thresh}}$) {
Considered Tumbling
} else {
Considered Spinning
}

Figure 1-5: Circular Inspection Trajectory due to Relative Station-Keeping of a Rotating Target Object

It is clear that for very small angular velocities, the expended fuel required to maintain station-keeping about a rotating object is negligible compared to the overall fuel required for the mission. For the purpose of this thesis, the small rotation rate is described as a "slow tumble". However, if the rotation rate is high enough, the fuel required to maintain station-keeping may be significant and in some cases, the thrusters may be saturated and unable to apply the required centripetal acceleration. This thesis describes the rotation rate in this case as a "fast spin".

As a result of these definitions, for a "spinning" target object, it would be necessary to estimate and compensate for the rotational properties of the target object using methods such as those described in this thesis, in order to avoid expending valuable fuel. In contrast, for a "tumbling" target object, it would not be necessary to use methods such as those described in this thesis as the expended fuel is by definition negligible when compared with the fuel required by the overall mission.

This leads to the question of how to quantify whether a target object is spinning or tumbling. A target object will be considered tumbling if and only if both of the following two statements are true.

1. The centripetal force required to maintain station-keeping is less than the maximum possible force the inspector can achieve.

2. The fuel required to maintain station-keeping over the required time (Δt) is less than M_f percent of the inspector's mass. In other words, M_f is the maximum mass fraction of fuel that can be expended on the centripetal force.

For example, spending 10% of the fuel on centripetal force for 10 minutes of station keeping would lead to $M_f = 0.1$ and $\Delta t = 10 \times 60\text{s}$.

The above definitions depend on a number of variables specific to each mission, which are shown in Figure 1-5 and discussed below. Since the primary expenditure of fuel is due to maintaining a centripetal acceleration during station-keeping, the analysis is simplified by considering only this force. As a result, two possible thresholds on angular velocity ($\omega_{\text{Thresh F}}$ and $\omega_{\text{Thresh } M_f}$, corresponding to the above two conditions) can be defined, as well as ω_{Thresh}, which is the minimum of the previous two thresholds.

Now, in order to derive equations for these thresholds, the following variables are assumed to be given: a station-keeping radius r, the mass of the inspector spacecraft m_{Insp} and the maximum force that can be applied by the inspectors thrusters,

$F_{\text{Insp Max}}$. Using these, a relationship can be defined with the maximum angular velocity of the target object $\omega_{\text{Thresh F}}$:

$$F_{\text{Insp Max}} \geq m_{\text{Insp}} r \omega^2_{\text{Thresh F}} \tag{1.1}$$

$$\omega_{\text{Thresh F}} \leq \sqrt{\frac{F_{\text{Insp Max}}}{r m_{\text{Insp}}}} \tag{1.2}$$

The second threshold can be defined by setting a maximum mass ratio M_f of fuel that is consumed in order to maintain the required centripetal acceleration. This is the ratio of the fuel that is expended to apply the centripetal force to the mass of the inspector spacecraft at the beginning of the station-keeping maneuver. To begin, the change in velocity due to the centripetal acceleration, $\Delta V_{\text{centripetal}}$, must be calculated:

$$\Delta V_{\text{centripetal}} = \int r \omega^2_{\text{Thresh } M_f} dt \tag{1.3}$$

Assuming that the radius and angular velocity remain constant over the period of time, Δt, that the station-keeping is performed, this integral can be evaluated:

$$\Delta V_{\text{centripetal}} = \Delta t r \omega^2_{\text{Thresh } M_f} \tag{1.4}$$

Using the specific impulse, I_{sp}, standard gravity, $g_0 = 9.81 m/s^2$, we can impose a limit on the mass fraction M_f, which is the ratio of propellant mass to wet mass, using the above equation for $\Delta V_{\text{centripetal}}$.

$$M_f \geq 1 - e^{-\frac{\Delta V_{\text{centripetal}}}{g_0 I_{sp}}} \tag{1.5}$$

$$M_f \geq 1 - e^{-\frac{\Delta t r \omega^2_{\text{Thresh } M_f}}{g_0 I_{sp}}} \tag{1.6}$$

Using this limit, $\omega_{\text{Thresh } M_f}$ can be solved for:

$$\begin{aligned}
e^{\frac{\Delta t r \omega^2_{\text{Thresh } M_f}}{g_0 I_{sp}}} &\leq (1 - M_f)^{-1} && (1.7)\\
\frac{\Delta t r \omega^2_{\text{Thresh } M_f}}{g_0 I_{sp}} &\leq -\ln(1 - M_f) && (1.8)\\
\omega^2_{\text{Thresh } M_f} &\leq -\frac{g_0 I_{sp}}{r \Delta t} \ln(1 - M_f) && (1.9)\\
\omega_{\text{Thresh } M_f} &\leq \sqrt{-\frac{g_0 I_{sp}}{r \Delta t} \ln(1 - M_f)} && (1.10)
\end{aligned}$$

Now, the overall threshold to define the cutoff between tumbling and spinning can be defined as ω_{Thresh}. If the angular velocity of the target object is less than this, it can be considered tumbling rather than spinning:

$$\begin{aligned}
\omega_{\text{Thresh}} &\leq \min\left(\omega_{\text{Thresh F}}, \omega_{\text{Thresh } M_f}\right) && (1.11)\\
\omega_{\text{Thresh}} &\leq \min\left(\sqrt{\frac{F_{\text{Insp Max}}}{r m_{\text{Insp}}}}, \sqrt{-\frac{g_0 I_{sp}}{r \Delta t} \ln(1 - M_f)}\right) && (1.12)
\end{aligned}$$

Using the above definitions, two examples are considered to determine the angular velocity threshold between tumbling and spinning. The first example is based on the SPHERES satellites, where the centripetal force fuel expenditure is a full tank (0.17 kg) in 5 minutes. The second exampleis based on the Orbital Express Mission's Exercise #1, where the fuel expenditure is 5 kg over 120 minutes [143, 97]. The Matlab code for these examples is presented in Section B.1.

Table 1.1: Angular Velocity Thresholds between Tumbling and Spinning for Two Example Missions

Parameter	**SPHERES**	**Orbital Express**
r	0.7 m	12 m
m_{Insp}	5.91 kg	900 kg
$F_{\text{Insp Max}}$	0.22 N	10.8 N
I_{sp}	37.7 s	235 s
M_f	0.029	0.0056
Δt	5 min	120 min
$\omega_{\text{Thresh F}}$	2.20 RPM	0.3020 RPM
$\omega_{\text{Thresh } M_f}$	2.17 RPM	0.1164 RPM
$\omega_{\textbf{Thresh}}$	**2.17 RPM**	**0.1164 RPM**

1.3 Problem Statement

This thesis develops navigation methods that enable the inspection of an uncooperative, unknown spinning target for the purpose of proximity operations. In order to achieve this, the inspector spacecraft will build a map of the target object and localize itself within that map. If the target was either stationary or tumbling, this problem could be considered solvable within the framework of prior research on Simultaneous Localization and Mapping (SLAM)[122, 71] and Visual Odometery (VO)[80, 88]. Since most approaches to solving the SLAM and VO problems make the assumption that the environment in which the vehicle is moving is static, the fact that the target object is spinning significantly complicates the problem for a number of different reasons.

The first complication is that the navigation system should be able to propagate the state of the target object into the future. Since the spinning target object has, by definition, a high angular velocity, Euler's Equations of Motion and Newton's Second Law must be solved along with the position and attitude kinematics equation in order to predict the location and orientation of the target object at some point in the future. This requires knowledge of not only the position and orientation of the target object, which are typically provided by SLAM algorithms, but also the linear and angular velocity as well as the center of mass, principal axes of inertia and the inertia matrix up to a scale factor.

The second complication that spinning targets introduce is that their motion is very difficult to replicate in any earth-based laboratory environment that undergoes 1-G of gravitation force. Most approaches to using bearings or fluids to simulate a spinning and nutating target will add additional friction and gravity-induced precession forces that would not be present in an orbital environment. A preferable approach for experimenting on spinning spacecraft is to use the microgravity environment that is available inside the International Space Station (ISS), which easily allows objects to spin and nutate at high speeds in six degrees of freedom (6DOF).

This thesis presents the development a navigation system that will allow a spacecraft to inspect an unknown and uncooperative target object and that is undergoing complex, but torque-free, spinning motions. This approach estimates the state of the target object relative to an inertial frame. In addition to the relative position and orientation that a typical SLAM algorithm will estimate, this approach will concurrently estimate the linear and angular velocity, the center of mass and principal axes of the target object, and its principal inertia matrix (up to a scale factor). This thesis also presents the development of an experimental testbed and "open research facility" to evaluate this and other vision-based navigation algorithms in the six degree of freedom microgravity environment of the International Space Station.

1.4 Literature Review

1.4.1 Spacecraft Proximity Operations Missions

A number of space missions have demonstrated autonomous proximity operations. The European Space Agency's Automated Transfer Vehicle (ATV) uses two relative

Global Positioning System (GPS) receivers for navigation until it is 250 m away from the ISS. After this point it uses laser retro-reflectors that are located on the ISS to determine the relative position and orientation[31]. The DARPA Orbital Express mission also used laser based retro-reflectors for docking with a cooperative target spacecraft[49].

The XSS-10 [12] and XSS-11 [98] missions both advanced the state of the art of inspection and proximity operations with a non-cooperative target. XSS-11 performed a visual-inertial circumnavigation and inspection of an un-cooperative target (similar to that performed on SPHERES in 2013[33]), but did not rendezvous with it. It additionally did not estimate its geometric model or its state (i.e. position, orientation, linear velocity, angular velocity and its center of mass, principal axes and ratios of inertia). A similar mission was performed by the MiTEx micro-satellite that was launched in 2009 to inspect the failed DSP-23 satellite[139]. The author of this thesis was unable to find detailed results for any of these three missions and believe they are not publicly available.

The Near Earth Asteroid Rendezvous (NEAR Shoemaker) mission landed a robotic probe on the surface of the near earth astroid Eros in 2001. In order to touch down on the surface, a map of 1624 crater landmarks was assembled manually by a human analyst (using computer-assisted ellipse fitting software). This map was used to perform relative pose estimation with one sigma uncertainties of approximately 10 meters[106].

The Japanese Aerospace Exploration Agency (JAXA) Hayabusa mission performed a sample return on 25143 Itokawa asteroid in 2005. The asteroid landing vehicle, the MUSES-C, was designed to drop a visual fiducial marker onto the surface of the asteroid[141]. It had planned to use this fiducial marker to regulate the spacecraft's velocity in the horizontal plane, while a set of laser altimeters would be used to determine the altitude of the spacecraft during the touchdown maneuver. However, due to laser altimeter sensor failures, this approach was abandoned during operations and a vision-based navigation algorithm, which tracked a number of point features on the asteroid, was selected and developed while the spacecraft was waiting nearby the asteroid. A team at JAXA built a map of global map of Guidance Control Points (similar to typical feature points), that could be used for relative pose estimation. Since this approach was too computationally expensive to be performed onboard the spacecraft, data was transmitted to earth, and flight commands were returned with a 30-minute delay. In November 2005, the Hayabusa spacecraft made multiple touchdowns on the asteroid; however, the navigation system led to a number of errors that resulted in imperfect touchdowns [141, 142, 67].

In 2009 and 2010 the Canadian Space Agency (CSA) and National Aeronautics and Space Administration (NASA) demonstrated the TriDAR[111] system for performing relative navigation to an uncooperative (i.e. no fiducials) but known (i.e. using a geometric model) target. The inspector vehicle was the Space Shuttle, and the TriDAR hardware was mounted to it while it circumnavigated its target, the ISS. The algorithms were optimized to use only 100 sparse points from a LIDAR-like system in an Iterative Closest Point algorithm [15].

In addition to these missions, a theoretical and simulation study was completed by Bayard and Brugarolas [14] that details a state estimation approach for small body relative navigation using camera sensors. They describe an Extended Kalman Filter based approach that incorporates bearing measurements from "Landmark Table (LMT)" of known feature points in a computationally efficient QR-factorization approach with delayed state augmentation. Additionally, a second method is discussed for computing relative orientation based on "Paired Feature Tables (PFT)" and provide an EKF update method that is similarly based on QR-factorization. It is noted that this method is equivalent to solving for the relative motion using the epipolar constraints (i.e. solving for the Fundamental and Essential Matrix using the 8 Point Algorithm [41], and then using this to solve for the relative translation and rotation using Horn's or Hartley's method [47, 41]). Note that this approach only models the relative position and linear velocity of the inspector spacecraft relative to the target object. It does not estimate the attitude or angular velocity of the vehicle in any form. Additionally, the approaches described in Bayard's paper makes the assumption that the target object is stationary and integrates onboard accelerometers with vision measurements based on this assumption. If the target object is rotating with enough angular velocity, the process model will believe the inspector is standing still, while the vision measurements will believe the inspector is following a large circular trajectory, and the filter may eventually diverge.

The above mentioned missions are good examples of the state of the art in autonomous proximity operations. It is important to note that they computed navigation solutions using some form of known feature points or models and a relative pose estimation algorithm (e.g. Horn's absolute orientation[46] or an equivalent method[8, 15]). In the case where the target was non-cooperative, a map of the feature points was generated offline and then used for subsequent navigation. A recently published survey article by Naasz et. al. [98] made a strong point: "...no spacecraft has ever performed autonomous capture of a non-cooperative vehicle, and full 6DOF relative navigation sensing to non-cooperative vehicles has only been shown to a limited extent."[98] While this article's survey did not include asteroid rendezvous missions, based on the above mentioned publications, the fact that the mapping phase was performed by ground operators implies that these missions did not demonstrate a fully autonomous rendezvous.

1.4.2 General Localization and Mapping

The problem of Simultaneous Localization and Mapping has been studied for a number of years beginning with Smith, Self and Cheeseman's Kalman Filter formulation [122]. Early implementations of SLAM systems in Kalman Filter frameworks were performed by Leonard using ultrasonic scanning sensors on a mobile robot [72] and Matthies using stereo cameras [87]. There are a number of alternatives to Kalman Filters that can be used to solve the SLAM problem, most of which focus on exploiting the intrinsic sparsity in the SLAM problem to develop efficient numerical methods. One significant example was developed by Montemerlo et. al. who use an approach based on Rao-Blackwellized particle filters known as FastSLAM[94]. A detailed treatment of many of the Kalman and particle filter methods for solving the SLAM problem is available in Thrun's textbook [131], which describes the state of the art up until 2005. However, it focuses on two-dimensional vehicles and does not cover computer vision techniques in much depth.

One of the main challenges with single-camera (monocular) SLAM algorithms is how to deal with measurements that do not have sufficient information about the depth of a feature point. Davison proposed an inverse-depth model that provides a realistic parameterization of uncertainty within the Gaussian distribution model that is assumed by an Extended Kalman Filter (EKF)[24].

Recently, researchers have focused on improving methods for integrating inertial measurement units with vision sensors. This includes approaches where the map of the environment is known a priori [132, 95], as well as methods to estimate Camera-IMU calibration parameters [74, 60]. Also, a number of researchers have integrated line features into the estimation framework[140, 54, 53, 65]. Typically, vertical line measurements are used to further refine an unmanned aerial vehicle's attitude.

The computer vision community developed an approach to solve the problem of localizing cameras and image feature points using a nonlinear least squares optimization, which is referred to as bundle adjustment [133]. Triggs provides a historical perspective of bundle adjustment in Appendix A of [133], which discusses on of the early implementations of second order bundle adjustment that was solved using least squares by Brown in 1957 to 1959 for the US Air Force[19, 21]. For SLAM applications, there has been a recent resurgence in smoothing or batch methods (as opposed to filtering. This is best highlighted in Davison's 2010 paper [42]. Recently Kaess et. al. developed an approach to solve the SLAM problem in the framework of probabilistic graphical models with sparse linear algebra routines in a system referred to as incremental Smoothing and Mapping (iSAM) [58]. Additionally, Klein and Murray implemented a monocular Parallel Tracking and Mapping [63] approach that introduced a keyframe approach that smooths or batch processes select frames in a bundle adjustment algorithm in parallel with a mapping thread that estimates a three-dimensional model of point features. Newcombe extended this method to a dense reconstruction approach that utilizes graphical processing units to construct a dense three dimensional model of a static scene [99].

Visual Odometry is a term for describing the localization-only problem that uses computer vision methods to determine a pose trajectory for the vehicle [89, 100, 35, 115]. It is important to note that these methods must maintain tracking between frames since there is no loop closure step. Recent work has combined visual odometry with map-building techniques [50].

One other approach to solving the SLAM problem is to build a graphical model whose nodes represent robot poses and edges represent kinematic constraints between the poses [78]. This is similar to the visual odometry approach; however, it allows for loop closures to occur. The SLAM problem is typically solved by expressing the constraints in the graph as a nonlinear cost minimization problem and using optimization methods that can take advantage of the sparsity of the problem. A map of the environment is typically computed after the optimization is solved by re-projecting the sensor measurements into a global frame [40, 105].

1.4.3 Localization and Mapping with Respect to a Moving Target Object

As was previously mentioned, solutions to the SLAM problem are typically formulated with a robot moving within an environment that is assumed to be static. Sibley et. al. pointed out the difficulty of dealing with moving reference frames in real world earth-bound environments such as elevators, trains and passenger aircraft [120]. It was pointed out that in many cases this motion is unobservable.

In terms of the specific problem of navigation with respect to tumbling space objects, numerous papers have presented methods to estimate the angular velocities and inertial parameters (up to a scale factor) using sensors onboard the tumbling object. Sheinfeld uses a least squares batch estimator to estimate the inertial properties and center of mass using gyroscope measurements and tracking of stereo feature points. Also, the velocity of the center of mass must be known at two instants in time for this solution to converge[116].

Three separate researchers (Augenstein at Stanford, Aghili at the Canadian Space

Agency and Lichter at MIT) have recently developed methods to deal with moving objects with six-degrees of freedom. All of these researchers have specifically focused on the problem of tumbling objects in space. Only two researchers (Aghili and Lichter) have tried to solve the same problem, but have used a significantly different approach that is based on an Extended Kalman Filter. Neither of these works included detailed experimental evaluation of six degree of freedom high speed spinning and nutating target objects.

Augenstein's Stanford doctoral thesis [10] and derived paper [11] focus on solving the SLAM problem for a tumbling target with a static monocular camera. One problem Augenstein discusses is that there is additional ambiguity in the orientation estimate, which has a non-convex cost function, so that the nonlinear minimization algorithm can converge to an incorrect, local minimum. Another problem that is discussed is that the center of mass of the target object with respect to the feature points is unknown. This is a critical element for being able to express the motion model of the target object, and Augenstein argues that simply adding additional process noise is not an effective solution to creating a robust estimation scheme. Augenstein presents a hybrid estimation approach that where the rotational dynamics are modeled as a Gaussian driven process and are estimated separately in a Rao-Blackwellized Particle Filter.

Augenstein's methods[10, 11] are likely unable to be applied to the problem in the proposed thesis. The main reason is that Augenstein's motion model does not include the time derivative of the angular momentum vector in the rotating body's frame in Euler's equation of rotational dynamics. Mathematically, it is assumed that $\omega \times J\omega = 0$. This could be due to the fact that the spin axis is aligned with the angular momentum vector or that the angular velocity is so small that it is not a major factor in the dynamics. In other words, this assumption is valid if the target object is tumbling slowly or spinning quickly about a principal axis. However, in the problem discussed in this thesis, these assumptions may not be valid. Additionally, it is not a trivial problem add this term to the motion model, since the inertia matrix is unknown (and must be estimated).

Aghili [7, 6] at the Canadian Space Agency (CSA) developed an EKF and motion planning framework to grapple a tumbling space object. The EKF assumes measurements from a camera system that can determine the relative pose to the target object's grapple point. It estimates the position, orientation and linear and angular velocities of the target's center of mass and principle axis-aligned body frame with respect to the camera's inertial frame (note that it is assumed that the spacecraft and its sensors are stationary). Additionally and most importantly, it estimates the relative position and orientation of the grapple point (from which the measurements are taken) with respect to the target's center of mass and principle axis-aligned body frame. Aghili estimates a rotation and translation between the geometric and rigid body reference frame as a set of constant parameters.

In Aghili's work, the angular velocities in the experimental dataset were less than 0.1 radians per second, which would be categorized as a tumbling target rather than a spinning target. Given these facts, it is reasonable to believe that an EKF based approach would not be able to keep up at higher angular velocities unless the sensor measurements also increased their frequency. In these types of high speed spinning situations, a smoothing based approach appears preferable because old measurements are not thrown out even if the estimator has not yet converged. Additionally, Aghili parameterizes the inertial parameters in the EKF with three variables, even though there are only two degrees of freedom. This parameterization has inequality constraints that are not consistent with a Gaussian distribution. While the presented dataset does not violate these constraints, no guarantee is provided that other datasets will have the same behavior.

Lichter's MIT Ph.D. thesis [75] and related paper [76] solves the problem of estimating the position, orientation, linear and angular velocities, as well as the center of mass and inertia matrix of an unknown, uncooperative and spinning target. Lichter uses a Kalman Filter to estimate the pose and dynamic parameters. This filter is split into two separate filters: a translational Kalman Filter and a rotational Kalman Filter. The attitude Kalman Filter is linear because the full 4 quaternion parameters are included in the state, despite the fact that there is only three degrees of freedom.

One of the main differences between this thesis and Lichter's approach, is that the measurement function does not depend on the currently estimated map, which is based on a voxel grid. While this is useful in reducing the computational requirements, it does not allow further refinement of previous measurements, which can lead to "smearing" of the map [75]. Also, this approach does not allow for loop closure, which would enable significant improvement of the map and reduces the estimation drift. Lichter's method of parameterizing the geometric frame relative to the body frame is similar to Aghili's. Lichter includes a quaternion in the state vector to represent the inertia parameters, even though there is only two degrees of freedom. In order to evaluate the method, a set of simulations is provided, and one dataset for experimental evaluation. The experimental data only provides two dimensional motion, so the inertia parameters are not observable.

Another researcher at Stanford, Kimball, published a doctoral thesis and conference paper [4, 62] on an offline method for solving the SLAM problem with respect to a moving iceberg. His first main contribution was a method for mapping the iceberg and estimating its trajectory using a spline representation of position and heading of a reference frame attached to the iceberg. The least squares estimator optimized for the location of the geometric center of the iceberg by minimizing the squared two-norm of the vector locations in the body fixed iceberg frame from the multi-beam sonar. Kimball found that if a sufficiently detailed spline model was included, this would estimate the geometric center of the iceberg. Kimball's second main contribution was to develop an online particle filter to estimate the underwater vehicle's state as well as the iceberg's state using a prebuilt map of the iceberg. The state includes a six degree of freedom pose estimate in the iceberg frame as well as the two dimensional position and one dimensional rotation of the iceberg with respect to an inertial frame. These two methods were evaluated using two datasets. The first dataset was gathered by an underwater vehicle while navigating an iceberg (moving less than 10 cm/s and 10 deg/hr, while the second dataset was of a stationary sea floor.

The work by Kimball illustrates a method to estimate the motion of both the inspector (the underwater vehicle) and the target object (the iceberg). The use of splines as a motion model for the target object appears well suited for low speed icebergs, however it is not clear how to extend this to the toque free solutions of Euler's rotational equations of motion of high speed spinning and nutating objects. Additionally, it is not clear that the assumption of estimating the geometric center of the object as the center of mass would be generally applicable to objects of varying density.

Kunz's doctoral research [68] is closest to the smoothing approach described in this thesis. Similar to Kimball, Kunz maps a rotating ice flow using an underwater vehicle. Kunz uses the iSAM optimization engine to build a factor graph of poses for the six degree-of-freedom position and orientation of the vehicle. Direct measurements of the vehicle's kinematics are available using gyroscopes and Doppler velocity logs (relative to local terrain). Multibeam sonar and visual cameras were used to generate a map of the ice flow. An interesting part of Kunz's approach (and the key difference from Kimball's) is that he modeled the orientation of the ice flow as a single rotational degree of freedom parameter in a separate Markov process in the factor graph. This orientation was added to the terrain relative yaw estimate in the measurement process. Additionally, inertial space measurements of the orientation were added as factors based on GPS measurements from a ship moored to the ice flow.

Kunz's method solves the SLAM problem with respect to a moving object by modeling the system as a factor graph and using iSAM for optimization. The only states that model the iceberg with respect to an inertial frame is a single orientation parameter, which has occasional and noisy corrections applied to it form the GPS of a ship that is moored (almost a rigid connection) to the iceberg. Kunz's work did not discuss whether how well his method would work if there is no sensor attached to the iceberg. Additionally, how to extend this approach to a full rigid body model is not immediately clear. Lastly, the rotation rates of the iceberg were less than 5 degrees per hour, which again is significantly less than a high speed spinning space object.

Indelman, Williams, Kaess and Dellaert recently published a method to include inertial measurement units (IMUs) in the iSAM factor graph model that has numerous similarities to the approach presented in this thesis[55, 56]. The state vector in their method consists of position, linear velocity and orientation of an object that has an IMU attached to it. Using a kinematic replacement approach, the accelerometer and gyroscope measurements are used to propagate the state between nodes in a factor graph model. There are two main differences between Indelman's work and the modelling approach used in this thesis. The first is that the Indelman does not include the angular velocity as part of the state vector or utilize Euler's Rotational Equation of Motion (likely because gyroscope measurements are directly available). The second and more important difference is that the covariance matrix for the factor does not appear to be updated. The uncertainty in the state at the next timestep should be a function of the accelerometer and gyroscope measurements (or in the case of this thesis, the linear and angular velocity estimates) and will vary with each factor. If the covariance is not updated, it will need to be set to an unnecessarily high value that will hurt the overall estimator performance. The approach in this thesis modified the iSAM system to incorporate varying covariance matrices. Note that a similar modeling approach was used by Luetenegger et. al., but with a different optimization approach[73].

Hillenbrand and Lampariello estimated the position, orientation, angular velocity, center of mass and full inertia tensor using a least squares method[45]. This was based on three dimensional range measurements of an unknown model that was matched using Horn's Absolute Orientation method [46], whose quaternions were differentiated to estimate angular velocity, which became the input to the least squares method. Note that there is no feedback from the estimated velocities and inertial parameters that will smooth out the quaternion estimates to help better match with the no external force or torque assumption.

1.4.4 Testbeds for Spacecraft Proximity Operations

In order to properly test spinning and nutating targets, it is important to have a relevant testbed. This would require a six-degree of freedom testbed that is capable of replicating a micro-gravity environment and performing computer vision-based navigation. One of the most challenging aspects of the micro-gravity environment

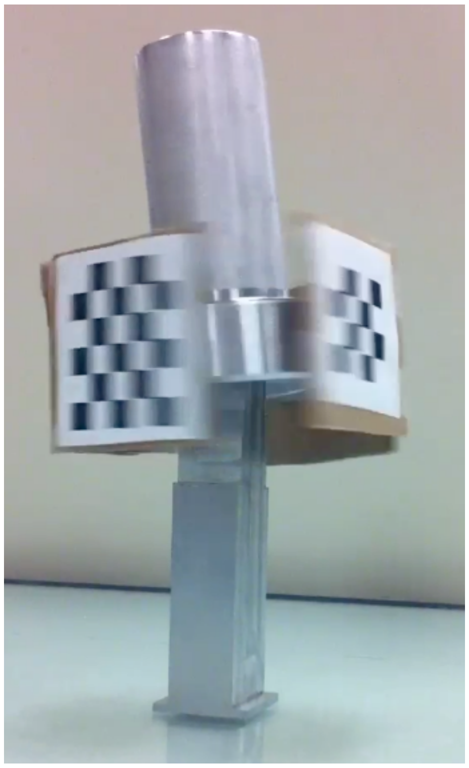


Figure 1-6: High Speed Spin on Sapphire Jewel Bearing

to replicate is high speed spinning, tumbling and nutating motion that follows Euler's Equation of Rotational Motion. The author of this thesis has attempted to build testbeds that spin in three axes on Sapphire jewel bearings (shown in Figure 1-6, however they are very difficult to exactly balance and always include additional forces are most similar to "gravity gradient" attitude dynamics along with rotational friction in the bearing. In addition to this they have a limited range of attitudes and would not be able to replicate a spin about an unstable minor axis. The ideal place to test these types of motion is in a micro-gravity environment such as the International Space Station. On March 23, 2013, during one of the ISS test sessions for the SPHERES VERTIGO program, astronaut Kevin Ford demonstrated a number of spinning motions for a pair of reconfigurable pliers in a micro-gravity environment. This demonstration illustrated how easy it is to see the dynamics of Euler's Rotational Equation of Motion onboard the ISS. An summary and analysis of these demonstrations is presented in Appendix A.

For this and other reasons, prior to this research, there were no "Open Research

Facility" for evaluating computer vision-based navigation algorithms against micro-gravity spinning motions in a six degree-of-freedom environment.

The SPHERES satellites [113, 112, 93] are the best example of a facility that would be able to have the capability of accurately simulating long duration micro-gravity in six degrees of freedom for a spinning target. When this research began in 2010, the SPHERES satellites did not yet have the capability to perform vision-based navigation inside the ISS. However, a prototype of the upgrade that is described in this thesis had been implemented as a ground prototype [136, 135, 134].

Another very similar research project was begun at the same time this research began and was performed by NASA Ames Research Center's Intelligent Robotics Group, with support from the MIT Space Systems Laboratory. Micire modified a Samsung Nexus S smartphone to attach to the SPHERES satellites and perform tele-robotics research [92, 129]. In December 2012, Micire "piloted" the SPHERES satellites on the ISS from a ground station on earth by receiving video and sending up actuation commands up to the satellites. The Nexus S could be used to perform vision-based navigation research, as it has a camera and an onboard CPU, but it has a number of limitations. The main limitation is that there is only a single camera, so there is no method for taking stereo measurements. Additionally, the lens and sensor on the camera are physically small and use a rolling shutter. Therefore taking photos of objects spinning with significant velocity would be extremely challenging and require a very brightly lit scene, but it may be impossible to avoid the distortion effects of the rolling shutter.

The Nexus S runs the Android Gingerbread operating system on an 1 GHz ARM Cortex A8. While this appears to be a powerful processor, there are a number of reasons it is not ideal for vision-based navigation research. The first is that this processor does not have out-of-order execution, which would deliver computational capability per watt of electricity for vision-based navigation applications that typically stall waiting for memory. The Nexus has only a 512 kB L2 cache, which is not quite enough to store two full grayscale 640 by 480 images. Single Instruction Multiple Data (SIMD) instructions are a common method for accelerating image-processing algorithms, and are commonly implemented with the SSE set of instructions on x86 processors. The ARM processors used in Android have an similar "Neon" instruction set, but this requires specific code optimizations and a number of libraries (e.g. OpenCV) have not included extensive support for them. Lastly, while the Android operating system is open source, a number of useful vision-based navigation libraries would need to be ported to operate on this platform. At the beginning of this research in 2010; a number of these libraries did not have Android versions, however in 2013, Android ports of robotics libraries have become much more prevalent.

Other organizations have developed six degree of freedom spacecraft experimental platforms that operate in Earth's 1-G environment. Rensselaer Polytechnic Institute published a good survey of 6DOF testbeds and a description of their design[36]. The Naval Research Laboratory's Spacecraft Robotics Laboratory has an excellent testbed of 6DOF dynamics and on-orbit lighting effects [103] for servicing spacecraft. The Air Force Institute of Technology has a good example of an air-bearing based attitude control testbed [90]. The University of Maryland uses a neutral-buoyancy testbed called SCAMP to perform spacecraft inspection experiments using computer vision [91].

1.5 List of Contributions

The previous section has summarized the current state of the art on vision-based navigation for unknown, uncooperative and spinning spacecraft. The author of this thesis is not aware of any work that has thoroughly and probabilistically integrated rigid body dynamics (i.e. Newton's Second Law and Euler's Equation of Rotational Motion) with a smoothing based solution to the SLAM problem, in order to estimate the position, orientation, linear and angular velocities, center of mass, principal axes of inertia and ratios of inertia. Aghili's work [7, 6] is the closest comparison, but it is a filtering approach, which may have more difficulty converging to the correct solution than a smoothing approach. Additionally, the author of this thesis disagrees with Aghili's inertia parameterization and believes it will lead to numerical conditioning issues (see Chapter 3 for the analytical proof).

The author of this thesis is not aware of any vision-based navigation experimental testbed for spacecraft proximity operations that can replicate high speed spinning motions that are free of external forces and torques in full six degrees of freedom environment.

The research contributions claimed in this thesis are listed below:

1. The development of an algorithm that solves the Simultaneous Localization and Mapping problem for a spacecraft proximity operations mission where the target object may be moving, spinning and nutating.

    (a) The development of a probabilistic factor graph process model based on both rigid body kinematics and rigid body dynamics. This model constrains the position, orientation, linear velocity and angular velocity between two subsequent poses at a defined timestep according to Newton's Second Law and Euler's Equation of Rotational Motion.

    (b) The development of a parameterization approach for estimating the center of mass and principal axes of inertia by incorporating a separate geometric reference frame in which all three dimensional feature points are estimated.

    (c) The development of a two dimensional parameterization approach for estimating the natural logarithm of the ratios of inertia as Gaussian random variables, and a modification of the above process model to incorporate this.

        i. An analysis of the nonlinear observability that confirms the number of observable degrees of freedom as well as the unobservable modes.

    (d) Implementation and evaluation of above algorithm using SPHERES satellites and Goggles with approximately stationary inspector and target spinning at 10 rotations per minute about its unstable minor axis.

        i. Comparison of the above algorithm's performance to the SPHERES Global Metrology System.

ii. Covariance and convergence analysis of the above algorithm.

2. Designed, built, tested and operated the first stereo vision-based navigation open research facility in a micro-gravity environment.

1.6 Outline of Thesis

This thesis begins with Chapter 2, a holistic review of rigid body kinematics and dynamics as well as computer vision-based navigation techniques. While Chapter 2 does not discuss any new contributions, it introduces the concepts, conventions and nomenclature that are required to understand the contributions outlined in the following chapters.

Chapter 3 provides a brief review of nonlinear observability analysis and applies this method to the inertia estimation problem. It is mathematically proved that the inertia matrix is only observable up to a scale factor if no known external forces or torques are applied.

Chapter 4 presents the algorithmic details of how rigid body dynamics are probabilistically incorporated into iSAM's pose graph optimization algorithm for estimating all of the desired quantities. It discusses the choices that were made in formulating the approach as well as outlining alternative approaches that were unsuccessful.

Chapter 5 describes the system requirements, high-level design, testing, operations and ISS results for the vision-based navigation upgrade to the SPHERES satellites, known as the Visual Estimation and Relative Tracking for Inspection of Generic Objects (VERTIGO) Goggles.

Chapter 6 presents the experimental dataset gathered by the VERTIGO Goggles during operations onboard the ISS, and the results when the SLAM algorithm described in Chapter 4 was applied to this dataset. The SPHERES Ultrasonic Global Metrology system as the reference for comparison which is discussed in detail.

Finally Chapter 7 summarizes the contributions, discusses possibilities for future work and describes possible extensions to other applications.

Chapter 2

Review of Rigid Body Kinematics and Dynamics and Computer Vision-Based Navigation

The purpose of this chapter is to provide a holistic review of the background material that is necessary to explain the contributions of this thesis. While this chapter does not describe any specific contributions itself, it defines the terminology and conventions that will be used in the remainder of this thesis.

2.1 Coordinate Frames and Parameterizations of Rotation

The location of a point in three dimensional space must be specified with respect to a reference system. Figure 2-1 defines two three-dimensional Cartesian reference frames that have a different location and orientation. The location of the point with respect to reference frame A is $\mathbf{p}_A$ and the location of the point with respect to reference frame B is $\mathbf{p}_B$. Each of these vectors is described in terms of the right-handed orthonormal basis vectors:

$$\mathbf{p}_A = \mathbf{p}_{A,x}\mathbf{x}_A + \mathbf{p}_{A,y}\mathbf{y}_A + \mathbf{p}_{A,z}\mathbf{z}_A \tag{2.1}$$

$$\mathbf{p}_B = \mathbf{p}_{B,x}\mathbf{x}_B + \mathbf{p}_{B,y}\mathbf{y}_B + \mathbf{p}_{B,z}\mathbf{z}_B \tag{2.2}$$

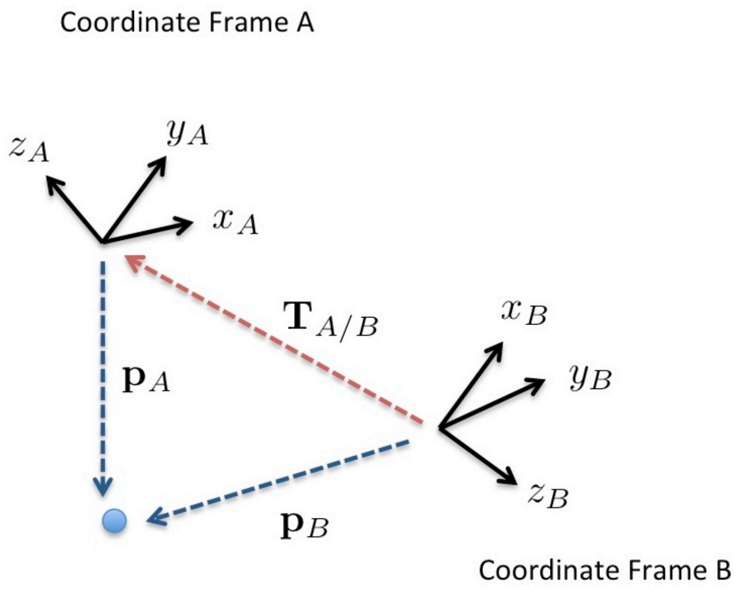


Figure 2-1: Illustration of the Location of a Point in Multiple Coordinate Frames

$\mathbf{p}_B$ can be written in terms of $\mathbf{p}_A$ if the translation $\mathbf{T}_{A/B}$ and a rotation operator $\mathbf{g}_{A/B}()$ is known:

$$\mathbf{p}_B = \mathbf{g}_{A/B}(\mathbf{p}_A) + \mathbf{T}_{A/B} \tag{2.3}$$

Where:

$$\mathbf{T}_{A/B} = \mathbf{T}_{A/B,x}\mathbf{x}_B + \mathbf{T}_{A/B,y}\mathbf{y}_B + \mathbf{T}_{A/B,z}\mathbf{z}_B \tag{2.4}$$

The rotation operator $\mathbf{g}_{A/B}()$ is a known function that rotates a vector from coordinate frame A to coordinate frame B.

All parameterizations of rotation of three dimensional space have three degrees of freedom. However, Stuelpnagel showed that there are no three dimensional parameterizations that are both global and non-singular[126]. As a result there is a large number of approaches for parameterizing rotations. Shuster provides a very detailed survey of a number of these methods [117]. The following section will highlight a few of the approaches that are considered in this thesis.

2.1.1 Rotation Matrices

Either of the orthonormal basis vectors shown in Figure 2-1 can be expressed in terms of the other. For example:

$$\mathbf{x}_B = c_{x1}\mathbf{x}_B + c_{x2}\mathbf{y}_B + c_{x3}\mathbf{z}_B \tag{2.5}$$

$$\mathbf{y}_B = c_{y1}\mathbf{x}_B + c_{y2}\mathbf{y}_B + c_{y3}\mathbf{z}_B \tag{2.6}$$

$$\mathbf{z}_B = c_{z1}\mathbf{x}_B + c_{z2}\mathbf{y}_B + c_{z3}\mathbf{z}_B \tag{2.7}$$

This can be simplified to a matrix notation as follows:

$$\begin{bmatrix}\mathbf{x}_A\\ \mathbf{y}_A\\ \mathbf{z}_A\end{bmatrix} = \begin{bmatrix}c_{x1} & c_{x2} & c_{x3}\\ c_{y1} & c_{y2} & c_{y3}\\ c_{z1} & c_{z2} & c_{z3}\end{bmatrix}\begin{bmatrix}\mathbf{x}_B\\ \mathbf{y}_B\\ \mathbf{z}_B\end{bmatrix} \tag{2.8}$$

$$\begin{bmatrix}\mathbf{x}_A\\ \mathbf{y}_A\\ \mathbf{z}_A\end{bmatrix} = \mathbf{R}_{A/B}\begin{bmatrix}\mathbf{x}_B\\ \mathbf{y}_B\\ \mathbf{z}_B\end{bmatrix} \tag{2.9}$$

Now equation 2.3 can be rewritten:

$$\mathbf{p}_B = \mathbf{R}_{A/B}\mathbf{p}_A + \mathbf{T}_{A/B} \tag{2.10}$$

Note that $\mathbf{R}_{A/B}$ is an orthogonal matrix since its rows and columns are unit vectors. Also, $\mathbf{R}_{A/B}^T = \mathbf{R}_{A/B}^{-1}$ and $\det(\mathbf{R}_{A/B}) = +1$ which means that it is a member of the Special Orthogonal Group $\mathbf{SO(3)}$. As a result, even though it requires 9 numbers to represent this parameterization, the above constraints mean that only three degrees of freedom are free and available. Note that this representation has no singularities or double coverings that are found in other representations.

2.1.2 Euler Angles

Euler angles parameterize rotation defining a set of three subsequent rotations. For rotations about the X, Y and Z axis by angles ϕ, θ and γ respectively are:

$$\mathbf{R}_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{2.11}$$

$$\mathbf{R}_\theta = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.12}$$

$$\mathbf{R}_\gamma = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

There are a number of conventions to choose from that specify how these rotations are applied successively. If the rotations are applied about the rotating axes, they are considered intrinsic rotations. Alternatively if they are applied about the fixed axes, they are considered extrinsic rotations. When the sequence of rotation is applied to each of the three axes, this is referred to as Tait-Bryan Euler angles, while if the first rotation and the last rotation are about the same axes, this is referred to as Classic Euler angles.

Aircraft commonly use an intrinsic Tait-Bryan representation that is intuitively named "yaw-pitch-roll" angles, which is "γ-θ-ϕ". The rotation matrix for this is as follows[121]:

$$\mathbf{R}_{\gamma\theta\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

As with all three parameter representations, Euler angles contain singularities. This occurs in the above example when $\theta = \frac{\pi}{2}$ radians. In this case, if $\mathbf{R}_{\gamma\theta\phi}$ is given, it is not possible to solve for γ and ϕ. This physically corresponds to the case where the first and last rotation axes are aligned; therefore there is a loss in the number of physical degrees of freedom.

2.1.3 Axis and Angle Representation

Euler's rotation theorem states that any rotation can be specified as an angle of rotation θ about an axis of rotation $\mathbf{n}$[138]. Since $\mathbf{n}$ is a three parameter vector with a unit normal constraint ($||\mathbf{n}|| = n_x^2 + n_y^2 + n_z^2 = 1$), there are three degrees of freedom for this four parameter representation.

An axis angle representation can be converted to a rotation matrix as follows:

$$\mathbf{R} = \cos\theta\mathbf{I} + (1 - \cos\theta)\mathbf{n}\mathbf{n}^T - \sin\theta[\mathbf{n}\times] \tag{2.15}$$

Where the cross product matrix $[\mathbf{n}\times]$ of a vector $\mathbf{n}$ is:

$$[\mathbf{n}\times] = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix} \tag{2.16}$$

Given a rotation matrix $\mathbf{R}$, the axis of rotation can be found by solving $\mathbf{Rn} = \mathbf{n}$, which is an eigenvalue problem. $\mathbf{n}$ is the eigenvector corresponding to the eigenvalue 1. The angle can be found using: $1 + 2\cos\theta = \mathrm{Tr}(\mathbf{R})$. Note that there are no singularities or double mappings.

2.1.4 Unit Quaternions

The quaternion $\mathbf{q}$ is a four parameter representation that can be found using the axis angle representation:

$$\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{q}} \\ q_4 \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{n}} \sin \frac{\theta}{2} \\ \cos \frac{\theta}{2} \end{bmatrix} \tag{2.17}$$

Quaternions have unit norm, $||\mathbf{q}|| = 1$, and therefore have only three degrees of freedom. To convert this to a rotation matrix:

$$\mathbf{R} = (q_4^2 - ||\bar{\mathbf{q}}||^2)\mathbf{I} + 2\bar{\mathbf{q}}\bar{\mathbf{q}}^T - 2q_4[\bar{\mathbf{q}}\times] \tag{2.18}$$

Note that the same rotation matrix will be found for $\mathbf{q}$ as for $-\mathbf{q}$. This implies that two different values of the quaternion will represent the same rotation and is why quaternions are referred to as being a double mapping onto the rotation group. To illustrate why this makes sense, assume that $\mathbf{q}_A(\mathbf{n}_A, \theta_A) \equiv -\mathbf{q}_B(\mathbf{n}_B, \theta_B)$:

$$\begin{bmatrix} \bar{\mathbf{q}}_A \\ q_{A4} \end{bmatrix} \equiv \begin{bmatrix} -\bar{\mathbf{n}}_B \sin \frac{\theta_B}{2} \\ -\cos \frac{\theta_B}{2} \end{bmatrix} \tag{2.19}$$

$$\tag{2.20}$$

This implies:

$$\theta_B = 2\arccos(-q_{A4}) = 2(\pi + \arccos(q_{A4})) \tag{2.21}$$

$$\theta_B = 2\pi + \theta_A \tag{2.22}$$

Which leads to

$$\begin{aligned}
\mathbf{n}_B &= -\mathbf{q}_A \frac{1}{\sin \frac{\theta_B}{2}} && (2.23)\\
&= -\mathbf{q}_A \frac{1}{\sin \frac{2\pi + \theta_A}{2}} && (2.24)\\
&= \mathbf{q}_A \frac{1}{\sin \frac{\theta_A}{2}} && (2.25)\\
&= \mathbf{n}_A && (2.26)
\end{aligned}$$

This illustrates that negating a quaternion is the same as adding 2π radians to the angle in its axis angle representation, which is intuitively the equivalent rotation.

Also, note that a rotation about $\mathbf{n}$ by θ is the same as a rotation about $-\mathbf{n}$ by $-\theta$.

$$\mathbf{q}(-\mathbf{n}, -\theta) = \begin{bmatrix} -\bar{\mathbf{n}} \sin \frac{-\theta}{2} \\ \cos \frac{-\theta}{2} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{n}} \sin \frac{\theta}{2} \\ \cos \frac{\theta}{2} \end{bmatrix} = \mathbf{q}(\mathbf{n}, \theta) \tag{2.27}$$

The inverse of a quaternion $\mathbf{q}$ is defined below:

$$\mathbf{q}^{-1} = \begin{bmatrix} -q_1 \\ -q_2 \\ -q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} -\bar{\mathbf{q}} \\ q_4 \end{bmatrix} \tag{2.28}$$

One of the fundamental operations of quaternions is quaternion multiplication. If $\mathbf{q}_A$ is the same rotation as $\mathbf{R}_A$ and $\mathbf{q}_B$ is the same rotation as $\mathbf{R}_B$, then the multiplication operator is defined as follows:

$$\begin{aligned}
\mathbf{q}_A \otimes \mathbf{q}_B &= \mathbf{q}\left(\mathbf{R}(\mathbf{q}_A)\mathbf{R}(\mathbf{q}_B)\right) && (2.29)\\
&= \begin{bmatrix} q_{B4}\bar{\mathbf{q}}_{\mathbf{A}} + q_{A4}\bar{\mathbf{q}}_{\mathbf{B}} + \bar{\mathbf{q}}_{\mathbf{A}} \times \bar{\mathbf{q}}_{\mathbf{B}} \\ q_{A4}q_{B4} - \bar{\mathbf{q}}_{\mathbf{A}} \cdot \bar{\mathbf{q}}_{\mathbf{B}} \end{bmatrix} && (2.30)\\
&= \begin{bmatrix} q_{A4} & q_{A3} & -q_{A2} & q_{A1} \\ -q_{A3} & q_{A4} & q_{A1} & q_{A2} \\ q_{A2} & -q_{A1} & q_{A4} & q_{A3} \\ -q_{A1} & -q_{A2} & -q_{A3} & q_{A4} \end{bmatrix} \begin{bmatrix} q_{B1} \\ q_{B2} \\ q_{B3} \\ q_{B4} \end{bmatrix} && (2.31)
\end{aligned}$$

Now the rotation operator can be defined as follows:

$$\mathbf{g}_{A/B}(\mathbf{p}_A) = \mathbf{q}_{A/B} \otimes \begin{bmatrix} \mathbf{p}_A \\ 0 \end{bmatrix} \otimes \mathbf{q}_{A/B}^{-1} \qquad (2.32)$$

Using this Equation 2.3 can be rewritten as:

$$\mathbf{p}_B = \mathbf{q}_{A/B} \otimes \begin{bmatrix} \mathbf{p}_A \\ 0 \end{bmatrix} \otimes \mathbf{q}_{A/B}^{-1} + \mathbf{T}_{A/B} \qquad (2.33)$$

2.1.5 Modified Rodrigues Parameters

The Modified Rodrigues Parameters (MRP) used in this thesis are defined below in terms of quaternions.

$$\mathbf{a}_p(\mathbf{q}) = \frac{4}{1+q}\begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}^T = \frac{4}{1+q}\bar{\mathbf{q}} \qquad (2.34)$$

Note that the convention here is almost entirely the same as the definition provided by Markley and Shuster [82, 117], except that the parameterization used in this thesis is a factor of four larger.

To determine the equivalent quaternion:

$$\mathbf{q}(\mathbf{a}_p) = \frac{1}{16 + \mathbf{a}_p^T \mathbf{a}_p} \begin{bmatrix} 8\mathbf{a}_p \\ 16 - \mathbf{a}_p^T \mathbf{a}_p \end{bmatrix} \quad (2.35)$$

Note that the singularity in this case occurs when $q = -1$. In the axis angle representation, this is when $\theta = 2\pi$ radians.

2.2 Rigid Body Kinematics

Rigid body kinematics describes how the position and orientation of an object change over time. This is done, in part, by defining the linear and angular velocities of an object and relating them the the time derivative of the rigid body's position and orientation.

2.2.1 Linear and Angular Velocities

Figure 2-2 illustrates a typical method of assigning inertial and body fixed reference frames. Point A is rigidly attached to the body frame and does not move with respect to the rest of the body. The rigid body may translate and rotate relative to the inertial reference frame over time.

A commonly used quantity is the angular velocity vector ω. It specifies the rate of rotation of the body frame with respect to the inertial frame and is a Euclidean vector that is expressed in the body frame. An important property of the angular velocity vector is that it can be added to another angular velocity vector, where the order of operations doesn't matter (this is in stark contrast to orientation parameters where order of operations can make a significant difference).

$$\omega = \omega_x \mathbf{x}_B + \omega_y \mathbf{y}_B + \omega_z \mathbf{z}_B \quad (2.36)$$

One of the most important relationships is the Coriolis Theorem, which calcu-

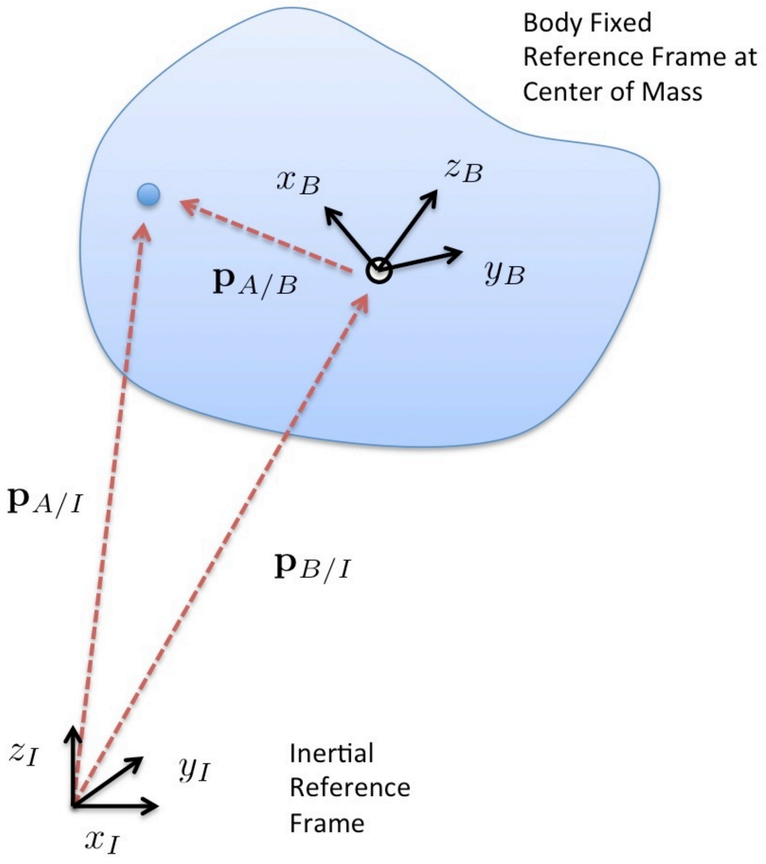


Figure 2-2: Rigid Body Reference Frames

lates the time derivative of a point on a rigid body that is undergoing rotation and translation.

$$\frac{d}{dt}^{I} \mathbf{p}_{A/B} \quad = \quad \frac{d}{dt}^{B} \mathbf{p}_{A/B} + \omega \times \mathbf{p}_{A/B} \tag{2.37}$$

2.2.2 Time Derivatives of Rotation Parameterizations

Now, the time derivative of the parameterizations discussed in Section 2.1 can be found in terms of the angular velocity vector.

The derivative of the rotation matrix $\mathbf{R}_{B/I}$ is as follows (the derivation is given in [138]):

$$\dot{\mathbf{R}}_{B/I} = -[\omega\times]\mathbf{R}_{B/I} \tag{2.38}$$

The derivative of the quaternions can be found as (also derived in [138]):

$$\begin{aligned}
\dot{\bar{\mathbf{q}}} &= \frac{1}{2}\left(q_4\omega - \omega \times \bar{\mathbf{q}}\right) && (2.39)\\
\dot{q}_4 &= -\frac{1}{2}\omega \cdot \bar{\mathbf{q}} && (2.40)\\
\Rightarrow \dot{\mathbf{q}} &= \frac{1}{2}\begin{bmatrix}\omega\\ 0\end{bmatrix} \otimes \mathbf{q} && (2.41)
\end{aligned}$$

Note that an interesting issue occurs with the double mapping of the quaternions. If an object is rotating with a physically smooth and mathematically continuous angular velocity, the quaternion's trajectory (i.e. time history) must also be mathematically continuous. This continuous requirement can be violated if the quaternion jumps between its two double mappings. Differentiating two quaternions with jumps and solving Equation 2.41 for angular velocity would lead to sharp spikes in ω that do not actually occur. Care must be taken to ensure that the quaternions are continuous if they are to be linked to angular velocity.

The derivative of the Modified Rodrigues Parameters is derived here using the vector and scalar parts of $\dot{\mathbf{q}}$ and the quotient rule. These results match with Shuster[117], with the exception of the factor of four.

$$\begin{aligned}
\dot{\mathbf{a}}_p &= \frac{4}{1+q_4}\dot{\bar{\mathbf{q}}} - \frac{4}{(1+q_4)^2}\dot{q}_4\bar{\mathbf{q}} && (2.42)\\
&= \left(-\frac{4}{1+q_4}\frac{1}{2}[\omega\times]\bar{\mathbf{q}} + \frac{4}{1+q_4}\frac{1}{2}q_4\omega\right) - \frac{4}{(1+q_4)^2}(-\frac{1}{2}\omega\cdot\bar{\mathbf{q}})\bar{\mathbf{q}} && (2.43)\\
&= -\frac{1}{2}[\omega\times]\mathbf{a}_p + \frac{2q_4}{1+q_4}\omega + \frac{1}{2}\frac{\omega\cdot\bar{\mathbf{q}}}{1+q_4}\mathbf{a}_p && (2.44)\\
&= -\frac{1}{2}[\omega\times]\mathbf{a}_p + \frac{1}{2}\frac{\omega\cdot\bar{\mathbf{q}}}{1+q_4}\mathbf{a}_p + \frac{2q_4}{1+q_4}\omega && (2.45)\\
&= \frac{1}{2}\left(-[\omega\times] + \frac{1}{4}\omega\cdot\mathbf{a}_p\right)\mathbf{a}_p + \left(\frac{2q_4}{1+q_4}\right)\omega && (2.46)\\
&= \left(-\frac{1}{2}[\omega\times] + \frac{1}{8}\omega\cdot\mathbf{a}_p\right)\mathbf{a}_p + \left(1 - \frac{1}{16}\mathbf{a}_p^T\mathbf{a}_p\right)\omega && (2.47)
\end{aligned}$$

2.3 Mass and Inertia Properties

The mass of a rigid body m is an important property. The center of mass of the object can be found using the density ρ at each point $\mathbf{p}$ over the entire volume V:

$$\mathbf{p}_{\text{com}} = \frac{1}{m}\int_m \rho(\mathbf{p})\mathbf{p}dm \qquad (2.48)$$

The inertia tensor of an object is represented by a three-by-three symmetric matrix as follows:

$$\mathbf{J} = \begin{bmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{xy} & J_{yy} & J_{yz} \\ J_{xz} & J_{yz} & J_{zz} \end{bmatrix} \qquad (2.49)$$

Where the moments of inertia can be found as integrals over mass elements in the body fixed frame:

$$J_{xx} = \int_m (y^2 + z^2) dm \tag{2.50}$$
$$J_{yy} = \int_m (x^2 + z^2) dm \tag{2.51}$$
$$J_{zz} = \int_m (x^2 + y^2) dm \tag{2.52}$$

Similar to the products of inertia:

$$J_{xy} = \int_m xydm \tag{2.53}$$
$$J_{xz} = \int_m xzdm \tag{2.54}$$
$$J_{yz} = \int_m yzdm \tag{2.55}$$

Since $\mathbf{J}$ is a real symmetric matrix, the coordinate frame can always be adjusted so that the products of inertia have all zero values. This is known as the principal axes and can be found by diagonalizing the $\mathbf{J}$ matrix using the eigenvalues, which are $\mathbf{J}_{\text{diag}}$, and the eigenvectors, which are the rotation matrix $\mathbf{R}_{\text{diag}}$ from the old frame to the principal axes frame.

$$\mathbf{J}_{\text{diag}} = \mathbf{R}_{\text{diag}} \mathbf{J} \mathbf{R}_{\text{diag}}^T \tag{2.56}$$
$$= \begin{bmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{bmatrix} \tag{2.57}$$

This is known as the principal axis coordinate frame, and has three planar moments of inertia. The axis that corresponds to the largest of these values is considered the major moment of inertia. These axes can typically be thought of a frisbee-like spinning objects as shown in Figure 2-3.

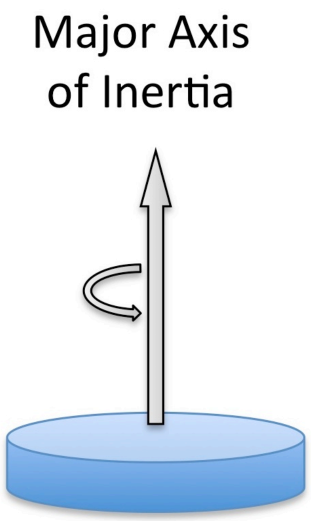


Figure 2-3: Frisbee-like Object

The axis that corresponds to the smallest of these values is considered the minor moment of inertia. These axes can typically be thought of a pencil, rocket or football-like spinning objects as shown in Figure 2-4.

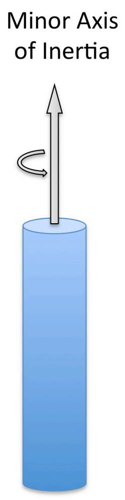


Figure 2-4: Pencil, Rocket or Football-like Object

The axis that corresponds to the middle of these values is considered the intermediate moment of inertia. These axes can typically be thought of spinning a texbook about the horizontal center of the page as shown in Figure 2-5.

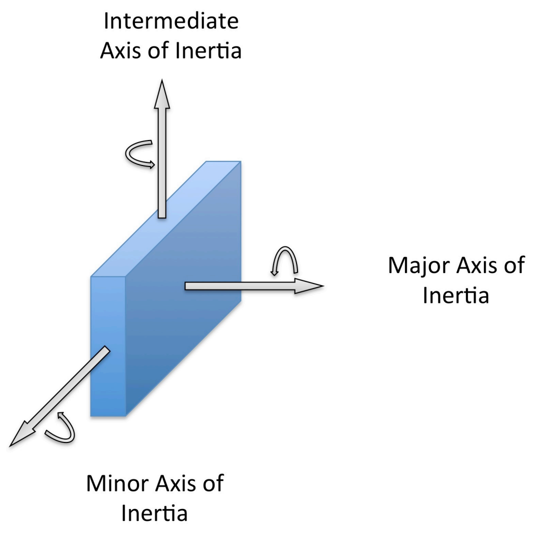


Figure 2-5: Textbook or Brick Like Object

2.4 Rigid Body Dynamics

Where kinematics describes the linear and angular velocities, dynamics describes the linear and angular accelerations. This type of motion is described by Newton's Second Law (for translation) and Euler's Equation of Rotational Motion. Both of these laws are only valid in an inertial reference frame (i.e. a reference frame that is not accelerating or rotating with respect to inertial space).

2.4.1 Newton's Second Law

The force **F** applied on an object is specified in units of $kg\frac{m}{s^2}$ or Newtons (N). The acceleration of a point is defined as $a = \frac{d^2}{dt^2}p$. Now Newton's Second Law states that the total force applied to an object is defined as the change in the momentum mv. Assuming that the mass of the object is constant, the law can be simplified to a common form:

$$\mathbf{F} = \frac{d}{dt}(m\mathbf{v}) \tag{2.58}$$

$$= m\frac{d\mathbf{v}}{dt} \tag{2.59}$$

$$= m\mathbf{a} \tag{2.60}$$

Note that if no force is being applied, there is a constant velocity that may be non-zero.

2.4.2 Euler's Equation of Rotational Motion

The torque applied to an object is defined as a force F applied at a fulcrum point at distance $\mathbf{r}$ away from a rotation point. It is defined as $\mathbf{M} = \mathbf{r} \times \mathbf{F}$. If this rotation point is located at the rigid body's center of mass, the torque is defined as being equal to the change in angular momentum $\mathbf{h} = \mathbf{J}\omega$ [138]. Recall the condition this derivative must be taken in an inertial frame in order for this law to be valid.

$$\mathbf{M} = \frac{d}{dt}^{I}\mathbf{h} = \dot{\mathbf{h}} \tag{2.61}$$

Applying the Coriolis Theorem, Equation 2.37:

$$\mathbf{M} = \frac{d}{dt}^{B}\mathbf{h} + \omega \times \mathbf{h} \tag{2.62}$$

$$= \frac{d}{dt}^{B}(\mathbf{J}\omega) + \omega \times \mathbf{J}\omega \tag{2.63}$$

Since $\mathbf{J}$ is assumed fixed to the body axis and therefore constant, a common form of Euler's Equation of Rotational Motion is as follows:

$$\mathbf{M} = \mathbf{J}\dot{\omega} + \omega \times \mathbf{J}\omega \tag{2.64}$$

This is often used to solve for the angular acceleration:

$$\dot{\omega} = -\mathbf{J}^{-1}\omega \times \mathbf{J}\omega + \mathbf{J}^{-1}\mathbf{M} \tag{2.65}$$

If the body fixed reference frame is aligned with the Principal Axes of Inertia, a substition of $\mathbf{J} = \mathbf{J}_{\text{diag}}$ can lead to a more simplified expression:

$$\begin{aligned}\dot{\omega}_x &= \frac{J_{yy}-J_{zz}}{J_{xx}}\omega_y\omega_z + \frac{1}{J_{xx}}M_x &(2.66)\\ \dot{\omega}_y &= \frac{J_{zz}-J_{xx}}{J_{yy}}\omega_x\omega_z + \frac{1}{J_{yy}}M_y &(2.67)\\ \dot{\omega}_z &= \frac{J_{xx}-J_{yy}}{J_{zz}}\omega_y\omega_x + \frac{1}{J_{zz}}M_z &(2.68)\end{aligned}$$

2.5 Torque Free Motion and Rotational Stability

Note that even if no external torque is being applied, the angular velocity still may vary over time. This depends on the initial conditions and which of the principal axes of inertia the angular velocity is about. This is unlike the translational case and can lead to some visibly peculiar motions as discussed in Appendix A.

$$\begin{aligned}\dot{\omega}_x &= \frac{J_{yy}-J_{zz}}{J_{xx}}\omega_y\omega_z &(2.69)\\ \dot{\omega}_y &= \frac{J_{zz}-J_{xx}}{J_{yy}}\omega_x\omega_z &(2.70)\\ \dot{\omega}_z &= \frac{J_{xx}-J_{yy}}{J_{zz}}\omega_y\omega_x &(2.71)\end{aligned}$$

A linear stability analysis can be performed on the above equations if it is assumed that one of the components of angular velocity is significantly larger than the other two. That is: $\omega_z = \Omega >> \omega_x, \omega_y$. Therefore, $\omega_y\omega_x \approx 0$. Now:

$$\begin{aligned}\dot{\omega}_x &= \frac{J_{yy}-J_{zz}}{J_{xx}}\Omega\omega_y &(2.72)\\ \dot{\omega}_y &= \frac{J_{zz}-J_{xx}}{J_{yy}}\Omega\omega_x &(2.73)\\ \dot{\omega}_z &= 0 &(2.74)\end{aligned}$$

Therefore we can set up a second order linear model:

$$\begin{bmatrix}\dot{\omega}_x\\ \dot{\omega}_y\end{bmatrix} = \begin{bmatrix}0 & \Omega\frac{J_{yy}-J_{zz}}{J_{xx}}\\ \Omega\frac{J_{zz}-J_{xx}}{J_{yy}} & 0\end{bmatrix}\begin{bmatrix}\omega_x\\ \omega_y\end{bmatrix} \tag{2.75}$$

This has the characteristic equation:

$$s^2 - \Omega^2 \left(\frac{J_{yy} - J_{zz}}{J_{xx}}\right)\left(\frac{J_{zz} - J_{xx}}{J_{yy}}\right) = 0 \tag{2.76}$$

With the following roots:

$$s = \pm\Omega\sqrt{\left(\frac{J_{yy} - J_{zz}}{J_{xx}}\right)\left(\frac{J_{zz} - J_{xx}}{J_{yy}}\right)} \tag{2.77}$$

Now, using this, if $J_{zz} > J_{xx}$ and $J_{zz} > J_{yy}$ (i.e. J_{zz} is a major axis of inertia) or if $J_{zz} < J_{xx}$ and $J_{zz} < J_{yy}$ (i.e. J_{zz} is a minor axis of inertia), then the roots are imaginary and the system is a second order harmonic oscillator, which is considered Lyapunov stable (but not Bounded Input, Bounded Output stable). If $J_{xx} > J_{zz} > J_{yy}$ or $J_{xx} < J_{zz} < J_{yy}$ (i.e. J_{zz} is an intermediate axis), then the roots are real with one negative and one positive value. This is considered unstable motion, however it is important to note that the conservation of rotational kinetic energy still applies so that the magnitude of the angular velocity vector will not grow without bounds.

2.5.1 Pointsot's Ellipsoid and Polhode Motion

Also, note that if no external torque is applied, the angular momentum vector is constant in inertial space: $\dot{\mathbf{h}} = \mathbf{M} = 0$. The squared magnitude of the momentum vector, $H^2 = ||\mathbf{h}||_2^2$ is constant and defined as follows:

$$H^2 = ||\mathbf{h}||_2^2 = \mathbf{h} \cdot \mathbf{h} = (J_{xx}\omega_x)^2 + (J_{yy}\omega_y)^2 + (J_{zz}\omega_z)^2 \tag{2.78}$$

This can be rewritten in ellipsoidal form:

$$1 = \frac{\omega_x^2}{(H/J_{xx})^2} + \frac{\omega_y^2}{(H/J_{yy})^2} + \frac{\omega_z^2}{(H/J_{zz})^2} \tag{2.79}$$

The translational kinetic energy is $E_T = \frac{1}{2}m(\mathbf{v} \cdot \mathbf{v})$, while the rotational kinetic energy is $E_R = \frac{1}{2}\omega \mathbf{J} \omega$. For a principal axes inertia frame:

$$E_R = \frac{1}{2}\left(J_{xx}\omega_x^2 + J_{yy}\omega_y^2 + J_{zz}\omega_z^2\right) \tag{2.80}$$

This can also be rewritten in ellipsoidal form:

$$1 = \frac{\omega_x^2}{2E_R/J_{xx}} + \frac{\omega_y^2}{2E_R/J_{yy}} + \frac{\omega_z^2}{2E_R/J_{zz}} \tag{2.81}$$

The solution to to Equations 2.79 and 2.81 is the intersection of these two ellipsoids. An example of this intersection is shown in Figure 2-6. The blue ellipsoid represents Equation 2.81 and the red ellipsoid represents equation 2.79. This intersection describes a path over which the angular velocity can vary over time (note that the angular velocity must follow a smooth trajectory without jumps), and is known as the polhode. Note that these ellipsoids are fixed to the body frame which rotates in the inertial frame. Additionally, it is important to note that the angular momentum vector **h** is constant in inertial space, and therefore set by the initial conditions of the rotation and inertia properties.

In order to picture how the body moves in the inertial frame, the kinetic energy ellipsoid (blue as shown in the figures) will roll without slipping on a plane with the point of contact being the current location of the angular velocity vector, which will be a point on the intersection of the blue and red ellipsoids (i.e. the polhode). This plane will be stationary in inertial space, therefore, it is called the invariant plane[38]. The invariant plane's normal is the angular momentum vector, **h**. The path that the angular velocity traces on the invariant plane is called the herpolhode and is not necessarily closed. However, if two of the inertias have the same values, the herpolhode will be a closed circular path that is commonly called the body cone.

Figure 2-6 shows the path that a spin would take about an intermediate axis (i.e. a spin beginning at the y axis). Note that the angular velocity vector (and therefore the contact to the invariant plane) flips over time.

This is because a spin about an intermediate axis is considered unstable and the angular velocity vector will constantly change direction. More intuitively, the angular velocity vector is trying to move to a location where it is spinning about a major moment of inertia, but must do so without changing the angular momentum. This leads to some very peculiar looking motions.

One classical example of this type of peculiar motion is spinning (or flipping) a textbook about its “horizontal” axis (with the pages taped shut). In gymnastics terminology, the book will add a clear twist to the flipping motion (i.e. a rotation about the book’s minor axis). The reason for this is that this twisting rotation pushes the orientation of the textbook towards a spin about its major axis, which is a stable rotational motion. The problem with this is that there is nothing to slow this twist down once it reaches this stable spin about a major axis (i.e. there is no mechanical damping). As a result, the textbook continues this twisting rotation and overshoots the spin about the major axis. This cycle will repeat itself endlessly as long as no external forces or torques are applied to the textbook. This type of a spin about an intermediate axis is very easy to demonstrate in a microgravity environment such as the International Space Station.

Figure 2-7 shows the polhode intersection for a spin about a major axis of inertia (i.e. the x-axis). Note that the angular velocity vector will remain on a closed circular path. This leads to the conclusion that spins about the major axis of inertia are always Lyapunov stable.

Figure 2-8 illustrates what happens when the spin is about a major axis and the other two axes have equal values of inertia (i.e. axisymmetric). In this case the herpolhode on the invariant plane forms a closed circle that is fixed in inertial space. A “space cone” can be created using this circle and the center of mass. Also, a “body cone” can be created using the polhode circle (shown as the intersection in Figure 2-7) and center of mass. Now the body cone rolls without slip on the outside of the body cone, and this is known as retrograde nutation. [1]

[1] Note that there is some confusion in the use of the term nutation. The motion described above is often called nutation by the “smaller” spacecraft dynamics community (from the nutation angle), and torque free precession by the “larger” classical mechanics community, who use the term nutation to describe a torque-induced oscillation about a rotating object. This thesis will use the term nutation in the sense that the spacecraft dynamics community uses it (i.e. to mean the same as torque free precession).

Figure 2-9 shows the polhode intersection for a spin about a minor axis of inertia (i.e. the z-axis). Note that the angular velocity vector will remain on a closed circular path. This leads to the conclusion that spins about the major axis of inertia are always Lyapunov stable.

Conversely, a spin about a minor moment of inertia is considered "marginally" or just barely stable. If there is a slight loss in kinetic energy or some external torque applied to it, the spin will change over to a spin about a major axis. An everyday example of this is spin of an American football when it is thrown with a spiral, which is a spin about a minor axis. A spiral pass of a football often exhibits a wobble if it is not thrown perfectly or if it is a very long pass (and the spin slows down thereby violating the above assumptions). Figure 2-10 illustrates what happens when the spin is about a minor axis and the other two axes have equal values of inertia (i.e. axisymmetric). Similar to the major spin, the herpolhode on the invariant plane forms a closed circle that is fixed in inertial space. A space and body cone are created in a similar way. However, in this instance the body cone rolls without slipping on the outside of the space cone. This is known as direct or prograde nutation.

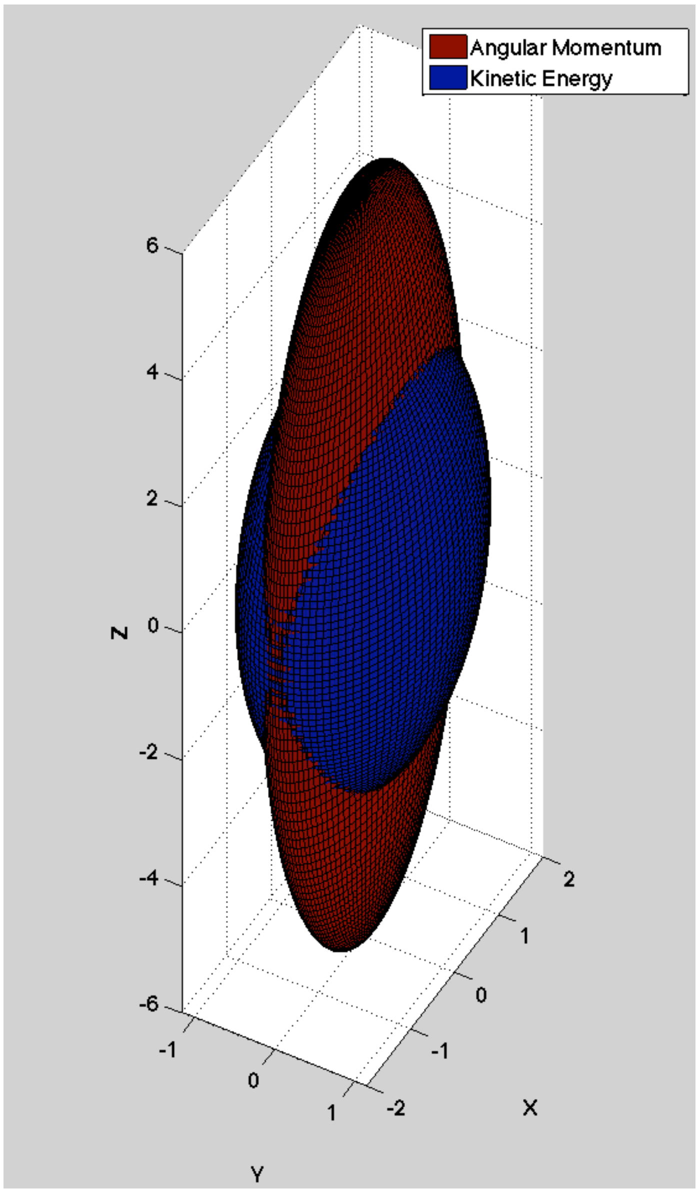


Figure 2-6: Polhode Diagram for Intermediate Axis Spin:
$\omega = [0, 2, 0]$ and $J_{xx} = 3,\ J_{yy} = 1,\ J_{zz} = 1/3$

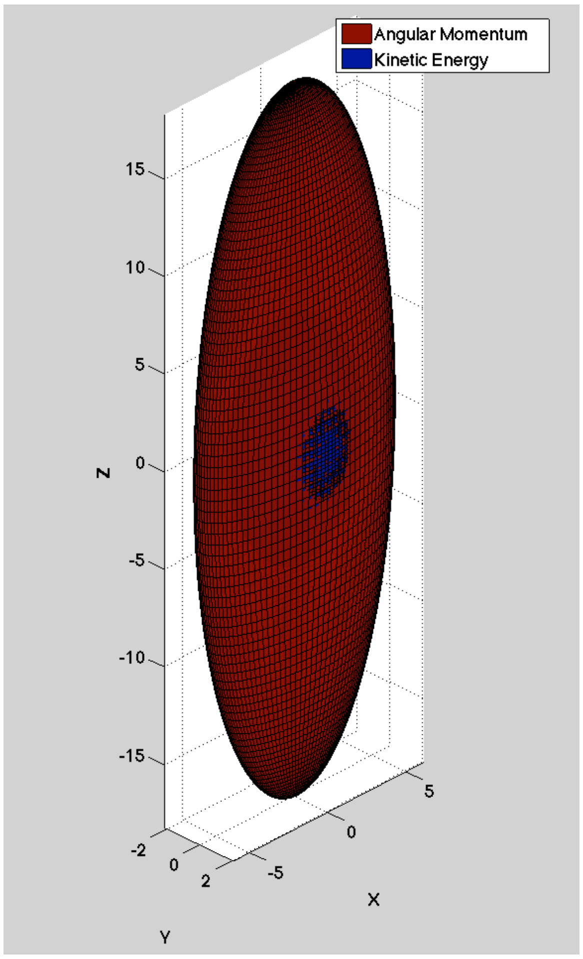


Figure 2-7: Polhode Diagram for Major Axis Spin:
$\omega = [2, 1, 1]$ and $J_{xx} = 3,\ J_{yy} = 1,\ J_{zz} = 1/3$

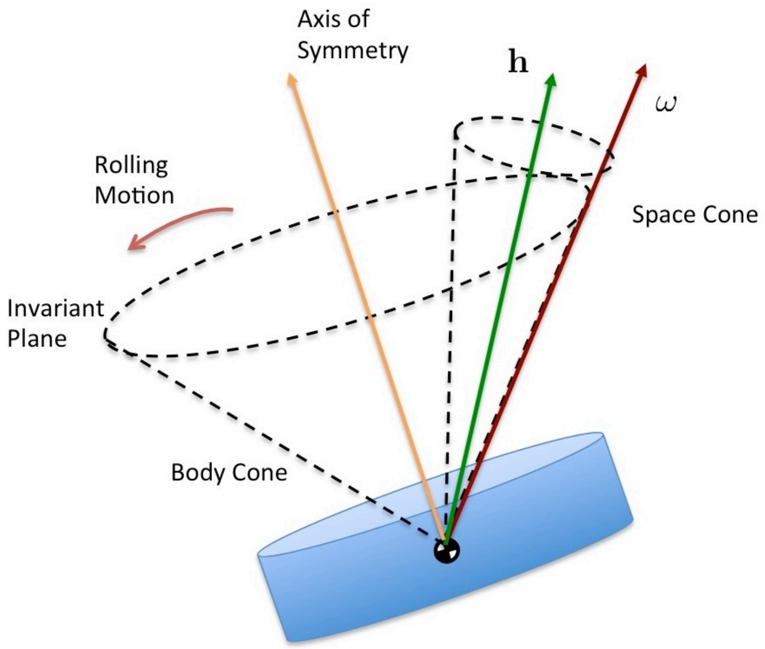


Figure 2-8: Space and Body Cone for Spin about Major Axis: Retrograde Motion

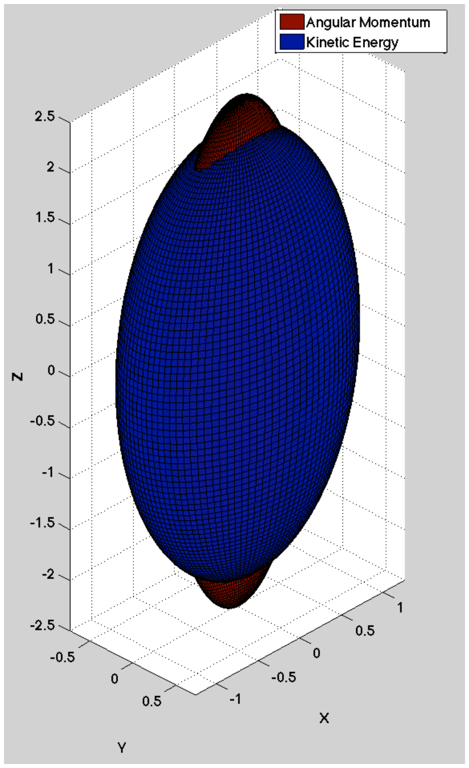


Figure 2-9: Polhode Diagram for Minor Axis Spin:
$\omega = [0, 0.5, 2]$ and $J_{xx} = 3,\ J_{yy} = 1,\ J_{zz} = 1/3$

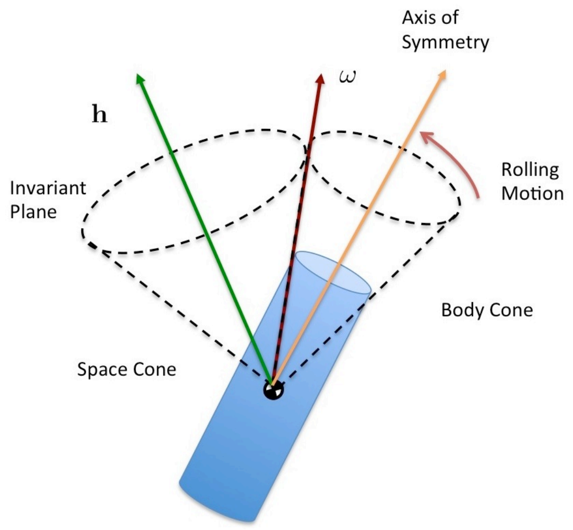


Figure 2-10: Space and Body Cone for Spin about Minor Axis: Prograde Motion

2.6 Pinhole Camera Model

The most common mathematical model of a camera is the pinhole camera model, where it is assumed that all light travels through an infinitely small hole and is projected onto an image frame (i.e. the sensor) whose plane is perfectly parallel to the pinhole. This plane is offset by a fixed distance f away from the pinhole, which is known as the focal length. Figure 2-11 shows the geometry of a pinhole camera model that leads to a perspective projection. A particular point in the world frame is shown by the red star and represented by the homogenous coordinates $\mathbf{p}_I$. This point is projected onto the image plane with the x and y coordinates u and v respectively. The coordinate of this point in terms of the camera frame (centered at the focal point) is $\mathbf{p}_c$. The optical center of the image plane is given by c_x and c_y. Note that there is also a rotation and translation between the inertial or world frame and the camera frame: $[\mathbf{R}_{C/I}, \mathbf{T}_{C/I}]$.

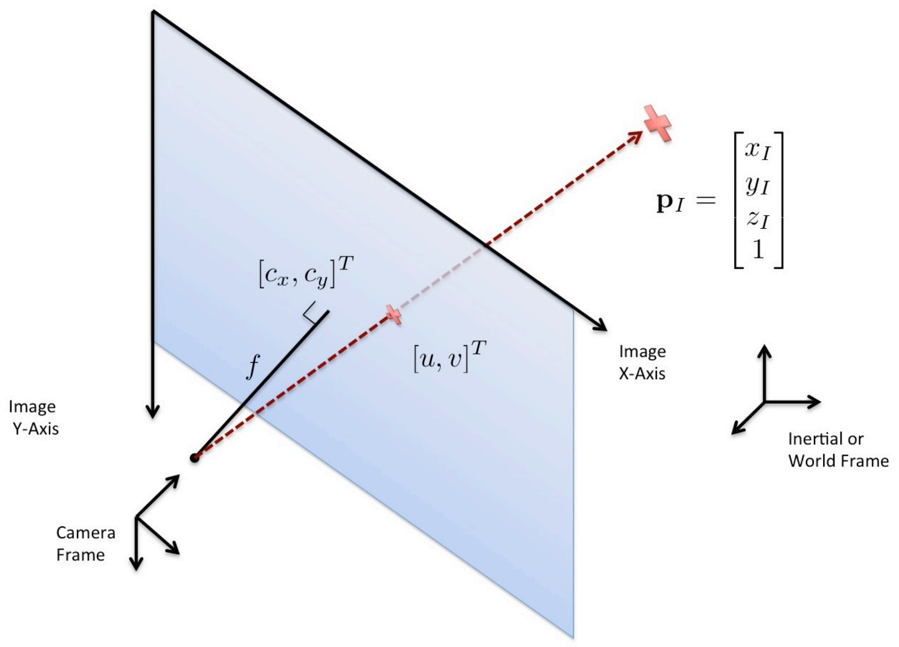


Figure 2-11: Pinhole Camera Perspective Projection Model

The mathematical model for computing the coordinates u and v given $\mathbf{p}_I$ begins by a coordinate frame change from the inertial or world frame to the camera frame:

$$\mathbf{p}_C = \begin{bmatrix}\mathbf{R}_{C/I} & \mathbf{T}_{C/I}\end{bmatrix}\mathbf{p}_I \tag{2.82}$$

$$= \begin{bmatrix}\mathbf{R}_{C/I} & \mathbf{T}_{C/I}\end{bmatrix}\begin{bmatrix}x_I \\ y_I \\ z_I \\ 1\end{bmatrix} \tag{2.83}$$

$$= \begin{bmatrix}x_C \\ y_C \\ z_C\end{bmatrix} \tag{2.84}$$

The next step is to solve for the image coordinates of the projected point. Note that s is the unobservable scale factor.

$$s\begin{bmatrix}u \\ v \\ 1\end{bmatrix} = \begin{bmatrix}f & 0 & -c_x \\ 0 & f & -c_y \\ 0 & 0 & 1\end{bmatrix}\mathbf{p}_C \tag{2.85}$$

$$s\begin{bmatrix}u \\ v \\ 1\end{bmatrix} = \begin{bmatrix}f & 0 & -c_x \\ 0 & f & -c_y \\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\mathbf{R}_{C/I} & \mathbf{T}_{C/I}\end{bmatrix}\mathbf{p}_I \tag{2.86}$$

Considering only the camera frame projection, it is helpful to write this relationship as:

$$u = \frac{f x_C}{z_C} - c_x \tag{2.87}$$

$$v = \frac{f y_C}{z_C} - c_y \tag{2.88}$$

2.7 Stereo Camera Model and Triangulation

Figure 2-12 shows the geometry of two cameras at arbitrary orientations that are imaging the same point $\mathbf{p}_I$. $\mathbf{p}_L$ and $\mathbf{p}_R$ are the location of the points in each of the left and right camera frames respectively. $\mathbf{O}_L$ and $\mathbf{O}_R$ are the focal points of the left and right cameras as well as the origins of those cameras' coordinate frames. The baseline between the two cameras is $\mathbf{T}_{R/L} = \mathbf{O}_R - \mathbf{O}_L$ in the left hand camera's coordinate frame. The rotation matrix $\mathbf{R}_{L/R} = \mathbf{R}_{L/W}\mathbf{R}_{R/W}^T$ is the rotation from the left camera to the right camera.

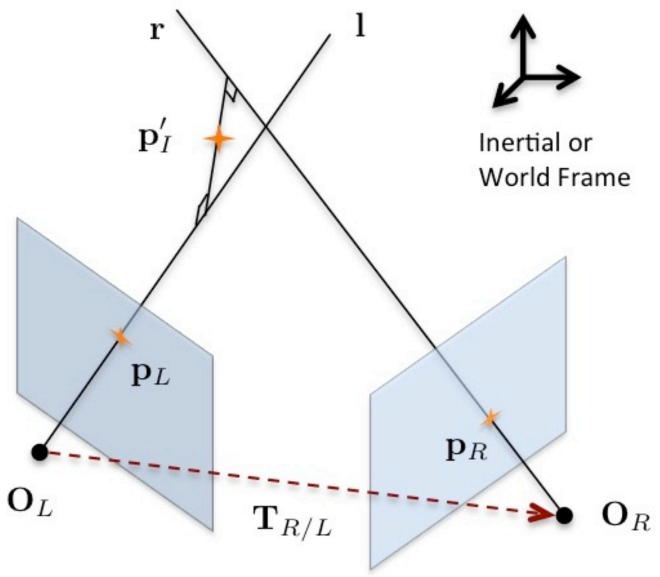


Figure 2-12: Stereo Camera Model

Note that in Figure 2-12 there is a misalignment between the left and right camera and the projected rays do not perfectly intersect. The closest point between the left and right projected rays is $\mathbf{p}'_I$. By setting up an equation for the baseline, a method for solving for $\mathbf{p}'_I$ can be found.

Now, with unknown constants a, b and c:

$$\mathbf{T}_{R/L} = a\mathbf{p}_L - b\mathbf{R}_{L/R}\mathbf{p}_R + c(\mathbf{p}_L \times \mathbf{R}_{L/R}\mathbf{p}_R) \tag{2.89}$$

$$\mathbf{T}_{R/L} = \begin{bmatrix} \mathbf{p}_L & -b\mathbf{R}_{L/R}\mathbf{p}_R & (\mathbf{p}_L \times \mathbf{R}_{L/R}\mathbf{p}_R) \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{2.90}$$

Given a known stereo geometry, the values of $\mathbf{T}_{R/L}$ and $\mathbf{R}_{L/R}$ are both known. For each measured point the projected ray directions are: $\mathbf{p}_L = [u_L - c_{x,L}, v_L - c_{y,L}, f_L]^T$ and $\mathbf{p}_R = [u_R - c_{x,R}, v_R - c_{y,R}, f_R]^T$. As a result Equation 2.90 is a linear system with 3 equations and 3 unknowns. Therefore, a, b and c can be solved for using linear methods and used to compute $\mathbf{p}'_I$ in the left camera frame:

$$\mathbf{p}'_I \quad = \quad a\mathbf{p}_L + \frac{1}{2}c(\mathbf{p}_L \times \mathbf{R}_{L/R}\mathbf{p}_R) \tag{2.91}$$

If the cameras are parallel (i.e. $\mathbf{R}_{R/L} = I$), aligned (i.e. $\mathbf{T}_{R/L} = [b_x, 0, 0]^T$, where b_x is the baseline) and the optical center is at the same location on the x-axis (i.e. $c_{x,L} = c_{x,R}$), then the rays in Figure 2-12 intersect exactly (i.e. $c = 0$ in Equation 2.89) and simpler triangulation model can be used:

$$\mathbf{p}'_I = \mathbf{p}_I \quad = \quad \begin{bmatrix} \frac{(u_L - c_x)b_x}{u_L - u_R} \\ \frac{(v_L - c_x)b_x}{u_L - u_R} \\ \frac{b_x f}{u_L - u_R} \end{bmatrix} \tag{2.92}$$

2.8 Stereo Camera Calibration

The pinhole model is an idealization of what occurs with real world lenses. One of the main differences is that the image is actually distorted through the curvature of the lens's optics. There are two primary modes of distortion. Radial distortion creates a pincushion or barrel effect in the images due to imperfections in the lens optics. Tangential distortion skews the image due to misalignments between the lens and the imaging sensor. Both of these effects were modeled by Brown[20]. The following model is based on Brown's method and used by OpenCV[18] and the remainder of this thesis. Note that two intermediate variables are defined as:

$$x'_c = \frac{x_C}{z_C} \tag{2.93}$$

$$y'_c = \frac{y_C}{z_C} \tag{2.94}$$

Next:

$$r = \sqrt{x'^2_C + y'^2_C} \tag{2.95}$$

$$x''_c = x'_c(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x'_C y'_C + p_2(r^2 + 2x'^2_C) \tag{2.96}$$

$$y''_c = y'_c(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y'^2_C) + 2p_2 x'_C y_C \tag{2.97}$$

The can be substituted back into the equations:

$$u = f x''_C + c_x \tag{2.98}$$

$$v = f x''_C + c_y \tag{2.99}$$

Practically this transformation is implemented as a lookup table that maps the coordinates $(x''_C, y''_C) \rightarrow (x'_C, y'_C)$, so that the appropriate image values can be "remapped" from the original image to an undistorted version. Additionally, corrections are also incorporated in this mapping to remove non-zero values in the y and z components of $\mathbf{T}_{R/L}$, to ensure that $\mathbf{R}_{L/R} = I$.

If all of the following parameters are known:

$$\mathbf{\Theta} = \{f, c_x, c_y, k_1, k_2, k_3, p_1, p_2, \mathbf{R}_{L/R}, \mathbf{T}_{L/R}\} \tag{2.100}$$

Then the stereo images can be undistorted and remapped so that the pinhole model applies, and the images are aligned so that the same points lie along a horizontal line between the two stereo images regardless of depth. An example of the images taken by the VERTIGO Goggles prior to undistortion and rectification is shown in the top half of Figure 2-13, while the same image after undistortion is shown bottom half of the same figure. The red lines connect a straight line in three dimensional space along the bottom of the ISS lights. The red lines top half of Figure 2-13 show a clear curvature in the image that is due to barrel distortion. The red lines in the corrected image (i.e. the bottom half) do not show any curvature. Also the green line connects two of the same points in the left and right image. In the top half, this line has a noticeable slope, indicating a horizontal misalignment, while in the bottom half of the figure the green line is now horizontal.

In order to estimate these parameters, a series of N_m images are taken of a checkerboard pattern of known geometry that is assumed to be flat. An example of the photos taken of this type of checkerboard with the feature correspondences highlighted by colored circles is shown in Figure 2-14. Each of these N_m image pairs shows N_p checkerboard corners, given by the known location $\mathbf{p}_i$ (note all of the points $\mathbf{p}_i$ lie on the same plane). Now a cost function can be written in terms of the known location of the points:

$$\hat{\mathbf{\Theta}} = \arg \min_{\mathbf{\Theta}_j, \forall 0<\mathrm{j}<N_m} \sum_{i}^{N_p} \sum_{j}^{N_m} \mathbf{m}\left(\mathbf{p}_i, \Theta\right) \tag{2.101}$$

$$\mathbf{m}\left(\mathbf{p}_i, \Theta\right) = \left\| \begin{bmatrix} u_L(\mathbf{p}_i, \Theta_j) - \hat{u}_L \\ v_L(\mathbf{p}_i, \Theta) - \hat{v}_L \end{bmatrix} \right\|_2^2 + \left\| \begin{bmatrix} u_R(\mathbf{p}_i, \Theta) - \hat{u}_R \\ v_R(\mathbf{p}_i, \Theta) - \hat{v}_R \end{bmatrix} \right\|_2^2 \tag{2.102}$$

Using Equation 2.101, a non-linear maximum likelihood problem can be solved that finds the best parameters $\hat{\mathbf{\Theta}}$. This is implemented in this thesis by OpenCV's "cv::stereoCalibrate()" function[18].

2.9 Feature Detection and Matching

An important aspect of mapping an unknown object is the ability to detect "feature points" that can be re-located and matched in images that are taken from different locations or at different times. One of the biggest challenges for these algorithms is to match features over large changes in relative position, orientation and scale. A significant amount of research has gone into developing and analyzing different

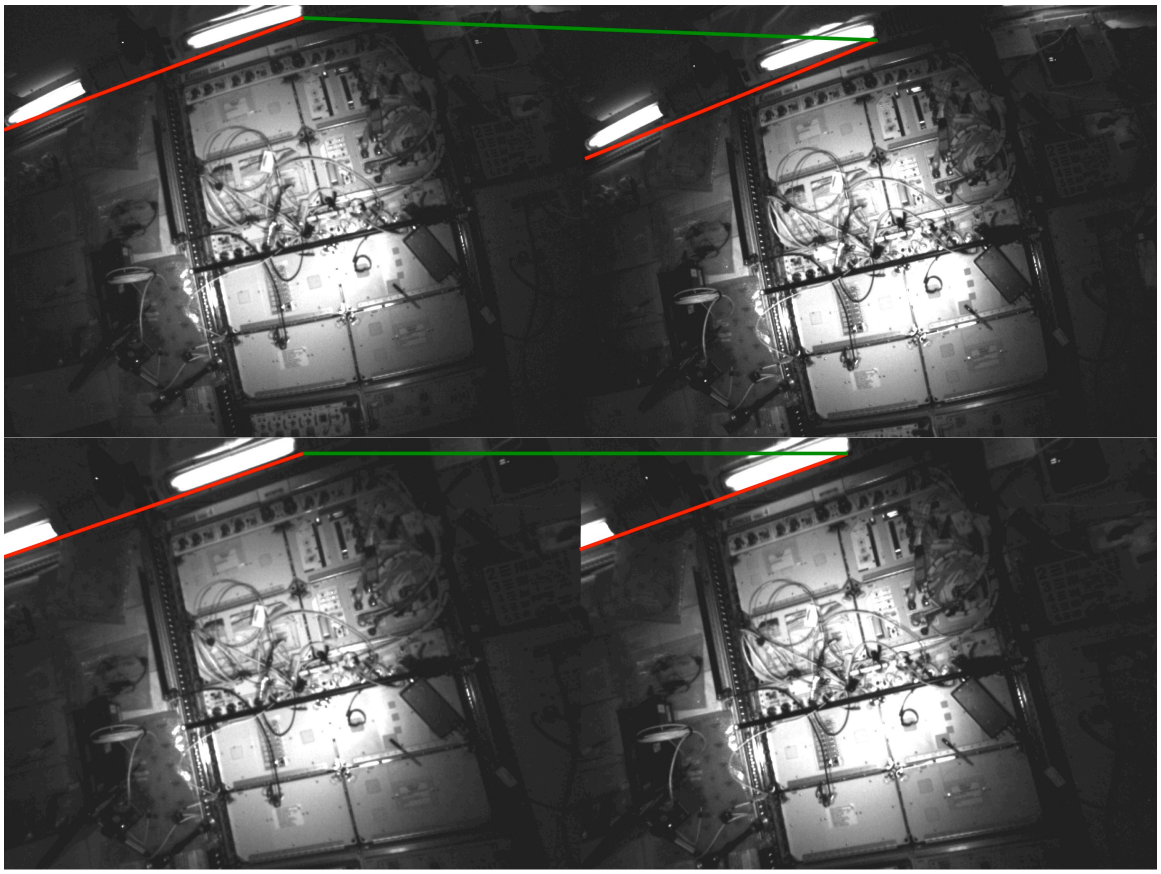


Figure 2-13: Stereo Images Prior to Un-Distortion (top) and After Un-Distortion (Bottom) with Red and Green Guides to Highlight Curvature and Horizontal Misalignment

algorithms for detecting and matching feature points in the past 20 years, [125, 128, 28, 110]. Two of the more popular feature types that are invariant to scale and rotation are the Scale Invariant Feature Transform (SIFT[77]) and Speeded Up Robust Features (SURF[13]).

OpenCV 2.3.1, the version used by the VERTIGO Goggles, contains implementations of both SIFT and SURF ([18]), of which the SURF implementation is considerably faster and therefore is used for the remainder of this thesis. SURF feature descriptors are based on an orientation aligned Haar-wavelet response, that is overlaid with a local grid. Magnitudes and directions of the response at each element in the grid are compiled to create a feature vector that can be efficiently compared using an L2 norm.

Detecting and matching features using only the feature descriptors can lead to a

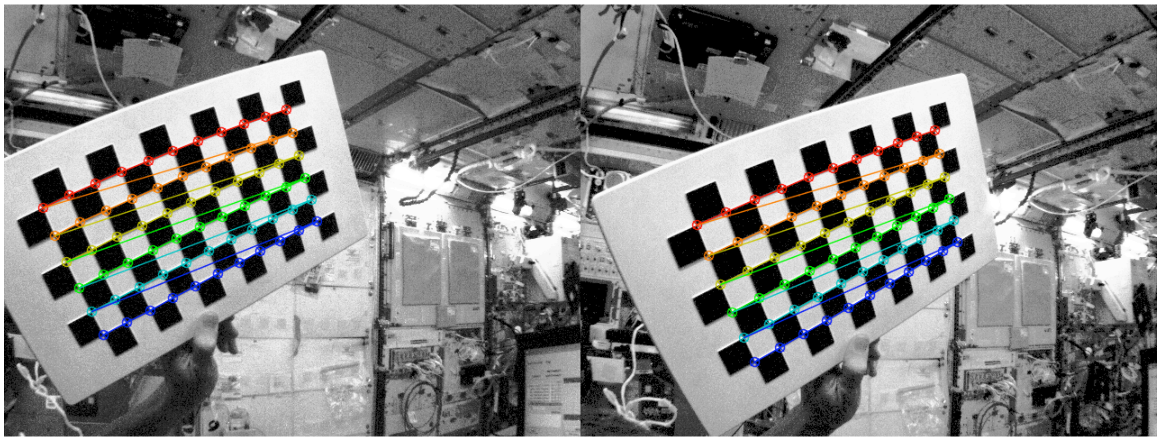


Figure 2-14: Stereo Images taken of Checkerboard Target

significant number of correct matches. However, there will typically be a significant number of incorrect, or "outlier" matches that must be thrown out. To do this, an outlier rejection method is used between stereo frames that have been calibrated and rectified. Features matches are accepted if the pixel location in the y-axis is less than a few pixels (i.e. $v_L - v_R < 1$ pixel), and the difference between the x-axis values (i.e. $d = u_L - u_R$ implies the depth is within a valid range).

However, when matching images taken at multiple times, the pixels will not be perfectly aligned. In order to reject the outliers the Random Sample and Consensus (RANSAC) algorithm[32] is used with Horn's Absolute Orientation[46] method as a geometric model.

2.9.1 Absolute Orientation

Horn's absolute orientation method solves the problem of determining a rotation and translation $(\mathbf{R}_{A/B}, \mathbf{T}_{A/B})$ between two reference frames as shown in Figure 2-15. The three dimensional locations (i.e. using stereo triangulation) of at least four points must be known in both reference frames A and B (i.e. $\mathbf{p}_{i/A}, \mathbf{p}_{i/B}$). The frames A and B may be two sets of stereo camera images taken of the same object at separate times. The mathematical relationship between these sets of points is (s is a scale factor):

$$\mathbf{p}_{i/A} = s\mathbf{R}_{A/B}\mathbf{p}_{i/B} + \mathbf{T}_{A/B} \quad (2.103)$$

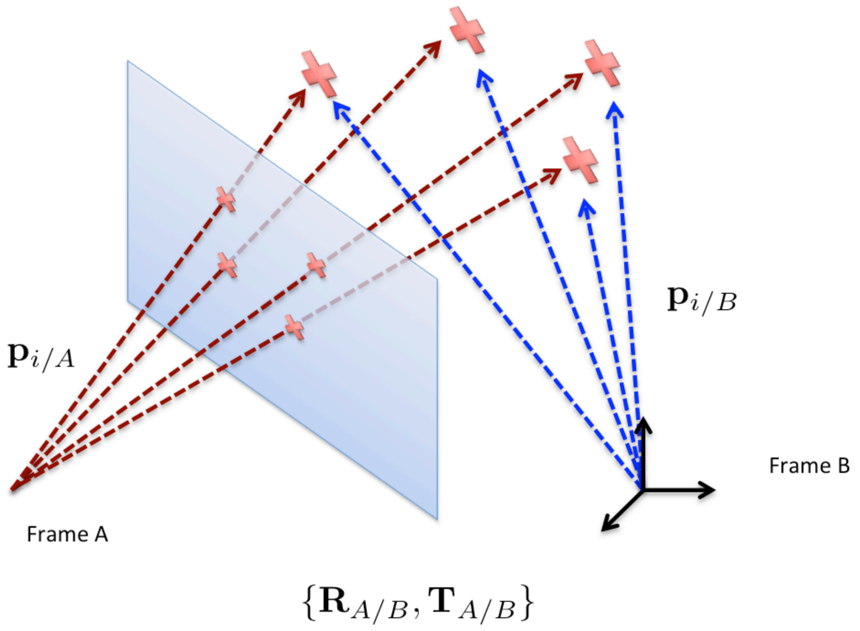


Figure 2-15: Reference Frames for Absolute Orientation

The absolute orientation algorithm begins by subtracting the centroid of the point sets in both frames:

$$\mathbf{p}'_{i/A} = \mathbf{p}_{i/A} - \sum_j \mathbf{p}_{j/A} \quad (2.104)$$

$$\mathbf{p}'_{i/B} = \mathbf{p}_{i/B} - \sum_j \mathbf{p}_{j/B} \quad (2.105)$$

Using this, a few intermediate matrices are computed:

$$\mathbf{M} = \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{bmatrix} \qquad (2.106)$$

$$= \sum_i \mathbf{p}'_{i/A} \mathbf{p}'^{T}_{i/B} \qquad (2.107)$$

And:

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

Now, the eigenvector corresponding to the maximum eigenvalue of $\mathbf{N}$ is the quaternion that represents the rotation between frames A and B. That is: $\mathbf{R}_{A/B} = \mathbf{R}(\mathbf{q}_{A/B})$. This quaternion minimizes the mean squared error cost function below:

$$C = \sum_i ||\mathbf{p}_{i/A} - s\mathbf{R}_{A/B}\mathbf{p}_{i/B} - \mathbf{T}_{A/B}||^2 \qquad (2.108)$$

The scale can be found from the following equation.

$$s = \frac{\sum_i ||\mathbf{p}'_{i/A}||^2}{\sum_i ||\mathbf{p}'_{i/B}} \qquad (2.109)$$

Lastly, the translation can be found by solving for the last remaining variable in equation 2.103.

2.9.2 Random Sample and Consensus (RANSAC) Outlier Rejection

The RANSAC algorithm is an iterative algorithm that utilizes a parametric model to find a set of points that are inliers. Algorithm 1 shows the pseudocode for this algorithm that uses the absolute orientation method as a model and returns a set of indicies M of inlier points. It assumes that the input arguments $\mathbf{p}_{i/A}, \mathbf{p}_{i/B}$ are ordered according to correspondences.

Algorithm 1 RANSAC Algorithm

```
1:  procedure RANSAC($\mathbf{p}_{i/A}, \mathbf{p}_{i/B}, \forall i : 0 < i < N_i$)
2:      $S \leftarrow \{\}$
3:      $M_{\max} \leftarrow 0$
4:      for $k = 1 \rightarrow N_k$ do
5:          $i_1, i_2, i_3, i_4 \leftarrow$ RAND()  ▷ Generates 4 random numbers between 0 and $N_i$
6:          $\{\mathbf{R}_{A/B}, \mathbf{T}_{A/B}, s\} \leftarrow$ AbsOrientation($\mathbf{p}_{\{i_1,i_2,i_3,i_4\}/A}, \mathbf{p}_{\{i_1,i_2,i_3,i_4\}/B}$)
7:          for $j = 1 \rightarrow N_i$ do
8:              $M \leftarrow \{\}$
9:              if $||\mathbf{p}_{j/A} - s\mathbf{R}_{A/B}\mathbf{p}_{j/B} - \mathbf{T}_{A/B}||_2 < \epsilon_{\text{Threshold}}$ then
10:                 $M \leftarrow \{M, j\}$
11:             end if
12:             if Size($M$) $> S$ then
13:                 $M_{\max} \leftarrow M$
14:                 $S \leftarrow$ Size($M$)
15:             end if
16:         end for
17:     end for
18:     return $M$
19: end procedure
```

Figure 2-16 shows the results of using SURF features matched across rectified stereo cameras and triangulated and filtered using RANSAC with the Absolute Orientation algorithm. The top two images in Figure 2-16 were taken at one timestep while the bottom two images were taken 0.5 seconds later while the target object was spinning. The bright green lines show the matches between the left and right camera, while the cyan (light blue) lines show where that feature was in the previous frame. Figure 2-16 shows that all of the green lines are horizontal and correspond to the same location on the target object. Also, the cyan lines have a slight angle that is in the direction of the spinning motion. This indicates that the frame-to-frame tracks are good.

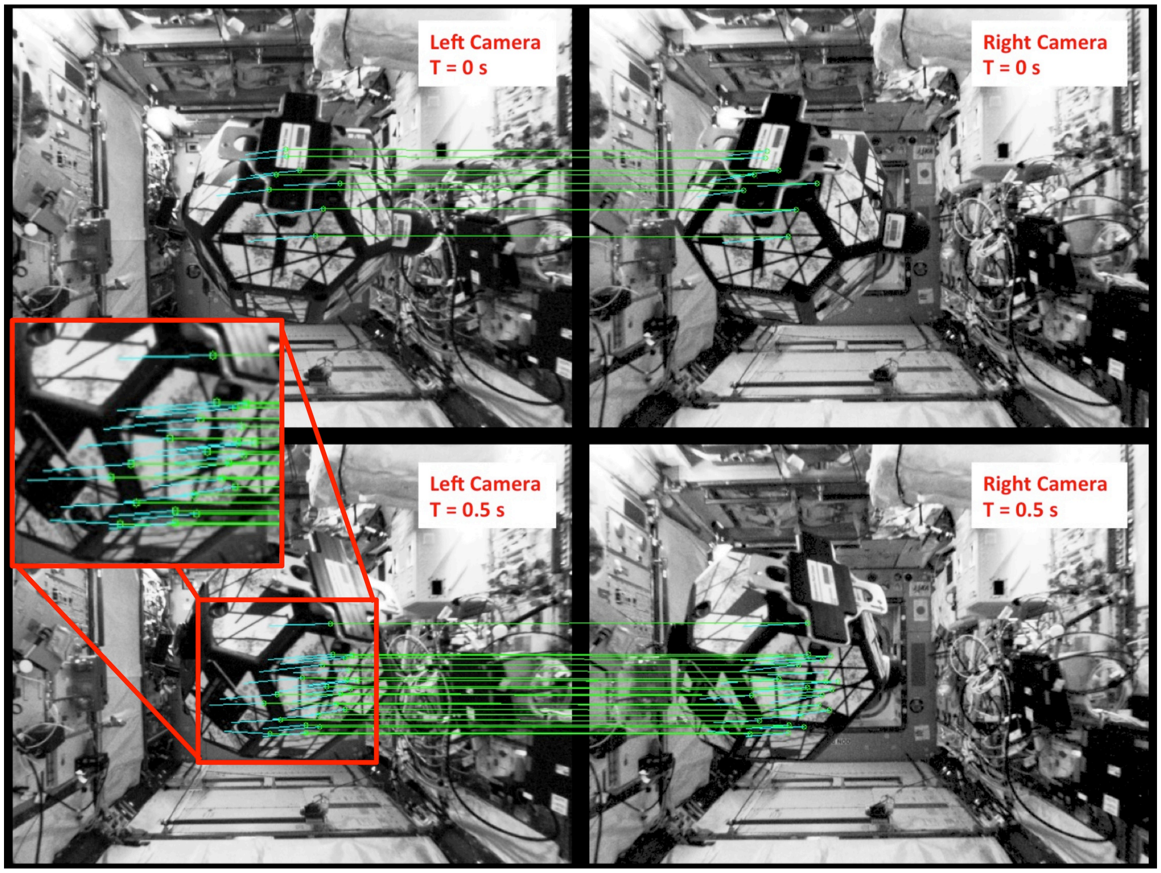


Figure 2-16: Tracked Features: Stereo (Green) and Frame to Frame (cyan)

2.10 Depth Map Computation

Feature provide correspondences between points in an image, however these points are sparse and leave considerable voids in the image. In contrast, a depth map is a single image computed from two aligned and rectified stereo images. An example depth map is shown in Figure 2-17. The correspondences of depth maps are much more dense than feature points.

In order to compute a depth map, the value of each pixel specifies the disparity value $d = u_L - u_R$, between that pixel and its corresponding location in the opposite image. As a result, pixels with higher values indicate that the object viewed by that pixel is closer while lower values indicate the object is further away. Notice that the depth map seems to work well when there is a large amount of texture in a small area and poorly when there is not enough visual texture to properly register a disparity.

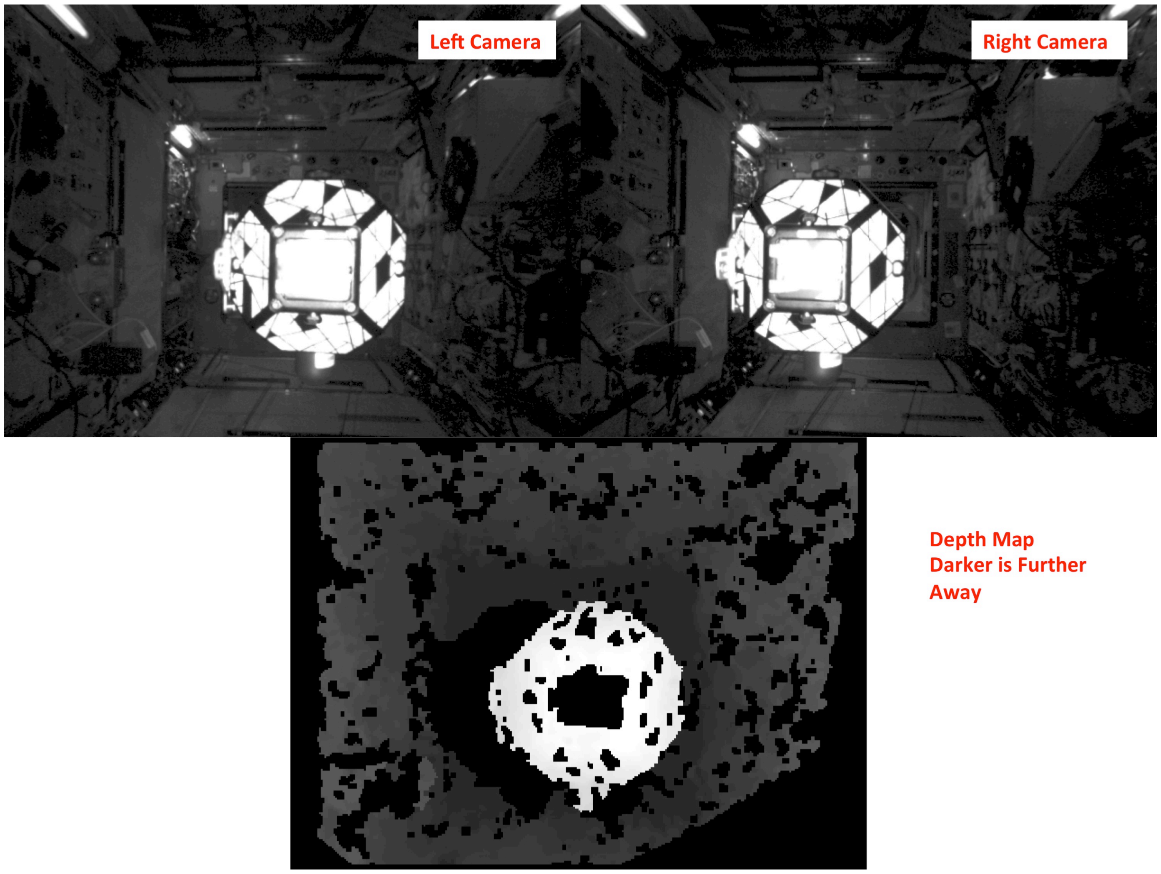


Figure 2-17: Stereo Camera View and Corresponding Depth Map

Typical methods for computing the texture are based on what is known as the sum of absolute differences. Given an image where at each pixel u, v there is a grayscale intensity value $I(u, v)$, an error function can be computed for a local window size of m:

$$\hat{d}(u,v) = \arg\min_{d} \sum_{x=-m}^{m} \sum_{y=-m}^{m} |I_L(u+x+d, u+y) - I_R(u+x, u+y)| \quad (2.110)$$

Now this is the typical type of approach that is found in libraries such as OpenCV. However, this is considered a local approach since each value only takes into consideration the m nearest pixel values. The depth map for local algorithms often appear to have a large number of unconnected patches. Global registration methods propagate their results throughout the image, thereby reducing the patch-like appearance, but are often much more computationally complex. However, new methods have been developed for computationally efficient global registration methods. An algorithm published by Geiger [3] illustrates a new method to perform efficient global registration. The algorithm is based on a set of support Sobel feature points that are filled in using Delaunay triangulation and belief propagation to estimate all of the pixel level disparity values. This algorithm is what was used for Figure 2-17 and is used in the remainder of this thesis.

2.11 Probabilistic Graphical Models and Factor Graphs

Probabilistic graphical models are sets of data structures and algorithms for operating on complex probabilistic models. The structure of the graph typically describes factorizations that allow the marginalization operation to be performed in a more computationally efficient manner. There are three main types of graphical models: Bayesian Networks, Markov Fields and Factor Graphs[16, 29]. While a comparison of these three representations is outside the scope of this thesis, factor graphs often lead to a slightly simpler formulation and more understandable representation for a number of pragmatic problems. The remainder of this thesis will only discuss factor graph representations.

Consider the following joint probability distribution $p(a, b, c)$ with its corresponding joint likelihood function $f(a, b, c)$:

$$p(a, b, c) \quad \propto \quad f(a, b, c) \tag{2.111}$$

This likelihood function may be able to be split up and factorized into smaller components that make the marginalization process computationally simpler. A simple example of this is shown in Figure 2-18.

The graph consists of nodes (circles) that represent variables, and rectangles that

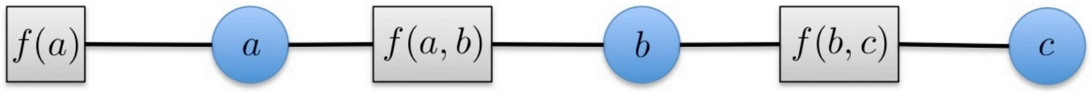


Figure 2-18: Simple Factor Graph Example

represent factors. Now the factor can be simplified as follows:

$$f(a,b,c) \quad = \quad f(a)f(a,b)f(b,c) \tag{2.112}$$

The computational reduction can be seen if the variable c is marginalized out. If a, b and c can each take on k values, the left hand side of this equation has a summation that requires $O(k^3)$ operations while the right hand side requires $O(k^2)$ operations.

$$f(a,b) = \sum_c f(a,b,c) \quad = \quad f(a)f(a,b)\sum_c f(b,c) \tag{2.113}$$

This type of model generalizes to the following factorization approach and a generic structure of graphs with variable and factor nodes.

$$p(\mathbf{s}) \quad \propto \quad f(\mathbf{s}) = \prod_{\mathbf{s}_i} f(\mathbf{s}_i) \tag{2.114}$$

A set of distributed "message passing" algorithms can be formulated to compute the marginal distributions of the node variables (a detailed discussion of these is outside of the scope of this thesis, but can be found in [16, 29]). When the node variables can be represented as Gaussian Random Variables, it has been shown that the computation of the marginal distributions can be mapped to a linear algebra problem[137]. This approach is the fundamental basis of the incremental Smoothing and Mapping (iSAM [58]) algorithm that was developed to estimate marginal Gaussian random variables for SLAM problems, when they are modeled as Factor Graphs. The iSAM algorithm was developed to take advantage of the inherent conditional independence structure in many SLAM problems and map it into a sparse linear algebra problem that can be solved efficiently[57].

Chapter 3

Observability Analysis

This chapter presents a non-linear observability analysis of the inertia matrix for torque free motion about an arbitrary axis. The method for analysis is based on Hermann and Krener's method, which is reviewed and applied to a specific problem. The unobservable mode is found to be a scale factor of the inertia matrix, which is consistent with intuitive expectations.

3.1 Review of Nonlinear Observability Analysis

The theory of observability for nonlinear systems was first developed by Griffith[39], Kou [66] and Kostyukovskii [64]. Observability is defined as the ability to recover the initial state of a system from a sequence of measurements.

Hermann and Krener developed the definitions of local and weak observability and developed an algebraic test for local weak observability in 1977[44], which is the approach that will be used here.

Hermann and Krener point out that observability is a global concept. In other words, a trajectory may need to be arbitrarily long in order for the system to be fully observable. They introduced a stronger concept of observability called local observability, that requires to system to be distinguishable for every open neighborhood of the initial point on the trajectory (i.e. instantly distinguishable). Additionally, they weakened the concept of observability to only require distinguishability from its neighbors, which they called weak observability. This is particularly useful for rotations where 0^o and 360^o are the same orientation, but are not neighbors. The concept of local weak observability can be intuitively thought of as the ability to instantly distinguish a trajectory from its neighbors. Figure 3-1 shows the relationship between these definitions. Note that for linear systems, all of these concepts and definitions are equivalent.

$$\begin{tikzcd}
\Sigma \text{ locally observable} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \Sigma \text{ observable} \arrow[d, Rightarrow] \\
\Sigma \text{ locally weakly observable} \arrow[r, Rightarrow] & \Sigma \text{ weakly observable.}
\end{tikzcd}$$

Figure 3-1: Observability Diagram [44]

Control Affine Form and Lie Derivatives

The control affine form is a representation of a system:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}) + \sum_i \mathbf{f}_i(\mathbf{x})\mathbf{u}_i \tag{3.1}$$

$$\mathbf{y}(t) = \mathbf{h}(\mathbf{x}(t)) \tag{3.2}$$

The Lie Derivative is the change of one vector field along the flow of another vector field, and can be defined recursively n times.

$$L_f^0 h = \mathbf{h}(\mathbf{x}(t)) \tag{3.3}$$

$$L_f^1 h = \frac{\partial L_f^0 h}{\partial \mathbf{x}} \cdot \mathbf{f}(\mathbf{x}) \tag{3.4}$$

$$L_f^n h = \frac{\partial L_f^{n-1} h}{\partial \mathbf{x}} \cdot \mathbf{f}(\mathbf{x}) \tag{3.5}$$

They can also be defined with respect to other fields, but do not commute in general.

$$L_f^0 h = \mathbf{h}(\mathbf{x}(t)) \quad (3.6)$$

$$L_{f_1}^1 L_f^1 h = \frac{\partial L_f^1 h}{\partial \mathbf{x}} \cdot \mathbf{f}_1(\mathbf{x}) \quad (3.7)$$

3.2 Algebraic Test of Nonlinear Local Weak Observability

The observability matrix $\mathbf{O}$ is created whose rows are defined by the elements of ∇L. In other words, the space $\mathbf{\Omega}$ is defined as the space that is closed with respect to Lie Differentiation on $\mathbf{f}$ and $\mathbf{f}_i$ and the observability matrix $\mathbf{O}$ is defined as the gradient of this space with respect to the state X, (i.e. $O = \nabla\mathbf{\Omega}$). This has lead to a rank test of local weak observability. In the case where the observability matrix is not fully observable, the null space of the observability matrix can identify unobservable modes (a.k.a. continuous symmetries)[51, 85].

Prior work on nonlinear observability of the SLAM and automatic calibration problems have been considered by a number of authors [84, 86, 52, 107, 69, 60]. For example, it has been shown that the kidnapped robot problem is not globally observable[108]. Also, Soatto discussed state representation for the structure from motion problem [123].

Note that all tests of observability are necessary, but not sufficient, conditions for estimator convergence. A positive result from an observability test does not specify what state trajectories are necessary for convergence of all of the observable estimation variables. In fact there may be a number of trajectories where the estimation system will never converge on the observable variables. The author of this thesis is not aware of any methods for testing specific trajectories for their convergence properties aside from running a variance analysis using the estimator itself.

3.3 Observability of Inertia Properties

One question of interest is as follows: Given a angular velocity trajectory $\mathbf{w}(t)$, is the inertia matrix observable? To analyze this, Hermann and Krener's test of local and weak observability will be applied to Euler's Equation of Rotational Motion for torque free input conditions (i.e. Equation 4.86).

$$\dot{\omega}_x = \frac{J_{yy} - J_{zz}}{J_{xx}}\omega_y\omega_z \tag{3.8}$$

$$\dot{\omega}_y = \frac{J_{zz} - J_{xx}}{J_{yy}}\omega_x\omega_z \tag{3.9}$$

$$\dot{\omega}_z = \frac{J_{xx} - J_{yy}}{J_{zz}}\omega_y\omega_x \tag{3.10}$$

The state variables are modeled as follows:

$$\mathbf{x} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ J_{xx} \\ J_{yy} \\ J_{zz} \end{bmatrix} \tag{3.11}$$

The control affine form of the dynamics is shown below. Note the last three elements are zero since the inertia matrix is constant.

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}) = \begin{bmatrix} \frac{J_{yy}-J_{zz}}{J_{xx}}\omega_y\omega_z \\ \frac{J_{zz}-J_{xx}}{J_{yy}}\omega_x\omega_z \\ \frac{J_{xx}-J_{yy}}{J_{zz}}\omega_y\omega_x \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.12}$$

The measurement model is:

$$\mathbf{y}(t) \;=\; \mathbf{h}(\mathbf{x}(t)) = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} \tag{3.13}$$

The Lie derivatives were computed symbolically using matlab. The source code is included in Appendix B.2, and the details of the derivatives will not be included here for brevity. The observability matrix was found to have its maximum rank when defined as follows:

$$\mathbf{O} \;=\; \begin{bmatrix} \nabla L_f^0 h \\ \nabla L_f^0 h \end{bmatrix} \tag{3.14}$$

The number of variables in $\mathbf{X}$ is six, while the rank of $\mathbf{O}$ is five. This means that there is one unobservable degree of freedom. In order to compute this the nullspace vector of $\mathbf{O}$ was found as:

$$\text{Null}(\mathbf{O}) \;=\; \begin{bmatrix} 0 \\ 0 \\ 0 \\ -J_{xx}/J_{zz} \\ -J_{yy}/J_{zz} \\ 1 \end{bmatrix} \tag{3.15}$$

This nullspace confirms that any multiplicative changes that are applied to J_{xx}, J_{yy} and J_{zz} cannot be estimated. In other words, the inertia matrix is observable only up to a scale factor for the torque free case. This can be intuitively confirmed by using an inertia matrix $\mathbf{J} = s_J\mathbf{J}'$, and seeing that the scale factor s_J cancels out.

$$\begin{aligned}
\dot{\omega} &= \frac{1}{s_J}\mathbf{J}'^{-1}\omega \times (s_J\mathbf{J}')\omega && (3.16)\\
&= \frac{1}{s_J}s_J\mathbf{J}'^{-1}\omega \times \mathbf{J}'\omega && (3.17)\\
&= \mathbf{J}'^{-1}\omega \times \mathbf{J}'\omega && (3.18)
\end{aligned}$$

It was previously mentioned that Hermann and Krener's test is necessary but not sufficient. While the analysis in this section has passed the observability test, there are a number of situations where there would not be enough information to fully estimate all parameters. One example is if the three principal moments of inertia are equal. In other words: $J_{xx} = J_{yy} = J_{zz}$. In this case Equation 3.13 becomes: $\dot{\mathbf{x}}(t) = \mathbf{0}$, and the Hermann and Krener test will indicate that the value of the moment of inertia is an unobservable mode.

This is of particular interest to the experimental analysis in Chapter 6 because it used a SPHERES satellite as a spinning target object, which has moments of inertia that are very close in value, because it is a roughly spherically shaped object.

A number of other similarly unobservable cases will occur for observing the inertia parameters including the center of mass, principal axes and ratios of inertia. One example is if two of the angular velocity components are close to zero, then it will be very difficult to estimate the center of mass, principal axes and inertia ratios. If all three of the angular velocity components are close to zero, inertial properties estimation would be equally difficult.

In addition to the types of motion, the number and distribution of data samples will have an effect on the observability, especially when low period rotations and nutations occur (a good example is the Y-axis angular velocity in Figure 6-14) . This is theoretically related to the Nyquist Sampling Theorem, which requires the number of data points to be twice the highest frequency component (a practiced "rule-of-thumb" is that a factor of 10 is used for good performance).

One question that arises is what to do if parameters are unobservable. This depends on whether the inertia parameters are only needed for further propagating a torque free solution to Euler's Rotational Equation of Motion, or if they are needed to predict the motion when forces and torques are applied. If these parameters are only needed for propagating the motion, and all of the Nyquist Sampling Theorem requirement on sufficient data has been met, then it is possible to propagate these equations with the incorrect values, since they by definition of unobservability have no effect on the outcome.

Alternatively, if these parameters are needed to predict the motion to applied forces or torques at a later time, it may be necessary to use some additional a priori information and heuristics to determine these properties. For example, by using the geometric model of the target object and an assumed density, a number of these properties could be estimated.

Chapter 4

Incorporation of Dynamics with Simultaneous Localization and Mapping

One of the main contributions of this thesis is a solution of a Simultaneous Localization and Mapping (SLAM) problem for an object that is spinning about any of its axes of inertia, while estimating the linear and angular velocities of the object as well as its center of mass, principal axes and diagonal inertia matrix (up to a scale factor). This chapter will provide the details of the approach used. Subsequent chapters in this thesis will evaluate this approach with experimental results.

One of the first decisions that must be made in developing an estimation approach is whether to use a filtering or smoothing algorithm. This has recently become an active topic of discussion, where the prevailing view presented by Davison [42] is that smoothing should always be used unless computational limitations require filtering in order to achieve real-time performance. Spacecraft proximity operations are definitely an application with limited computational resources, and others have selected a filtering approach for similar work[6, 76]. Despite this, a smoothing approach was chosen for this work out of concern for estimator convergence. It is entirely possible that the estimation system may converge to a local minimum for the estimates of the mass and inertia properties (i.e. center of mass, principal axes and ratios of inertia) when the system is first turned on. If the prior states, especially angular velocities, are marginalized out (as is the case with filtering), the data that the algorithm needs to get out of that local minimum is no longer available. In contrast, with a smoothing approach, the position, orientation, linear and angular velocities can be estimated and converged upon first. Once these sufficiently rich trajectory values are available, the inertia properties can be fully estimated, which may require hopping in and out of a number of local minimums. This is shown in Section 6.7 where the inertia properties didn't finally converge until after the full angular velocity trajectory that showed a low period oscillation of Euler's Equation of Rotational Motion was available.

To date, most approaches for solving a SLAM problem with smoothing techniques (e.g. iSAM [57, 58] or other pose graph optimization methods[105, 78]) only model the rigid body kinematic transformations between subsequent time-steps[40]. There are a small number of exceptions to this generalization, but they typically only model the linear velocity and use a strap-down gyroscope for kinematic replacement of the angular velocity[55, 56] and do not update the covariance matrix based on the velocities. If the rigid body dynamics are not incorporated into the process model, a priori assumptions about the linear and angular velocity must be made and covariance of the process noise applied to the position and orientation must be large enough to accommodate any variations in the velocities. The problem with this is that size of the process noise sets an upper bound on the estimation accuracy, so it is desirable to keep the process noise as low as possible.

At a high level, the approach used in this thesis is to redo the probabilistic process model so that the state variables and likelihoods in the factor graph incorporate rigid body dynamics (i.e. Newton's Second Law and Euler's Equation of Rotational Motion). The majority of this chapter is devoted to the description and discussion of how this probabilistic modeling was developed, along with some of the design decisions that produced the models. While this approach should be generic to any methods for estimating the marginal distributions of a Gaussian factor graph formulation, a few steps were specifically tailored to the iSAM system.

This chapter begins with a brief review of how factor graphs are applied to the

SLAM problem as per iSAM's assumptions. Next is a simple one-dimensional example to illustrate the dynamics modeling method, in contrast with with the typical kinematics only approach. This is followed by the details of a generic non-linear model and the approach for calculating the process noise covariance. Using this generic non-linear model, the specifics of the six degree of freedom dynamic model are described. A discussion is provided of the approach for modeling the overall center of mass, principal axes of inertia and ratios of inertia. The overall graphical model representation of the entire solution is presented. Lastly some of the complication and conditioning issues that arise with time-step and velocity unit selection will be discussed.

4.1 Review of Factor Graph Formulation and Incremental Smoothing and Mapping (iSAM) Algorithm

The overall modeling approach merges discrete-time state space representations of rigid body kinematics and dynamics with pose graph representations. A generic pose graph model is shown as a factor graph in Figure 4-1. In this generic Simultaneous Localization and Mapping example, the states at two instants in time are represented by $\mathbf{x}[1]$ and $\mathbf{x}[2]$, and two landmarks are represented by $\mathbf{l}_1$ and $\mathbf{l}_2$. The nodes of the graph are shown as circles, which contain state variables that must be estimated (such as $\mathbf{x}[k]$ and $\mathbf{l}_i$), while the rectangles represent factors, which model the joint probability distribution between some number of nodes. These factors are denoted by $f(a, b)$ which describes a Gaussian probability distribution between the random variables a and b. This probability distribution often represents some type of error between the variables in two nodes that must be minimized.

In order to solve this pose-graph optimization problem, the methods in this thesis utilize the Incremental Smoothing and Mapping (iSAM) system [58], which is available online as open source software. The iSAM algorithm assumes that the means of the factors, $f()$, are zero (note that non-zero means can be handled by augmenting

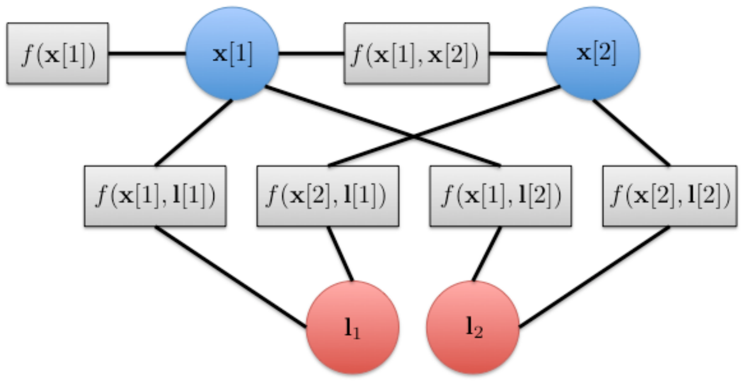

$f(\mathbf{x}[1])$
$\mathbf{x}[1]$
$f(\mathbf{x}[1],\mathbf{x}[2])$
$\mathbf{x}[2]$
$f(\mathbf{x}[1],\mathbf{l}[1])$
$f(\mathbf{x}[2],\mathbf{l}[1])$
$f(\mathbf{x}[1],\mathbf{l}[2])$
$f(\mathbf{x}[2],\mathbf{l}[2])$
$\mathbf{l}_1$
$\mathbf{l}_2$

Figure 4-1: Generic Pose Graph Model

the state variables), and optimizes over all of the state variables in all of the nodes to minimize a cost function of the errors, which are specified by each of the factors. iSAM performs this optimization by converting the factor-graph problem to an equivalent linear algebra problem. This is beneficial due to the inherent sparsity of the SLAM problem in the Information form [130], which directly causes the linear algebra problem to be inherently sparse. This allows the optimization problem to be solved in a computationally efficient manner using sparse linear algebra routines.

4.2 A One Dimensional Example

It is helpful to begin with a simple, but typical, system model. Consider a one-dimensional linear position model. The state variable r_k represents the position of the vehicle along a line at timestep k, while landmark i is located at position l_i also along the same one-dimensional line. With reference to Figure 4-1, the discrete time state and landmark variables are defined simply as follows:

$$\mathbf{x}[k] = r_k \tag{4.1}$$

$$\mathbf{l}_i = l_i \tag{4.2}$$

This system is described by the following continuous time process and discrete time measurement model:

$$\begin{aligned} \dot{r}(t) &= w_r(t) && (4.3) \\ z_i[k] &= l_i - r_k + w_z[k] && (4.4) \end{aligned}$$

Where w_r is white noise and and $w_z[k]$ is the discrete time measurement error.

$$\begin{aligned} E[w_r(t)] &= 0 && (4.5) \\ E[w_r(t_1)w_r(t_2)] &= a_{w_r}^2 \delta(t_1 - t_2) && (4.6) \\ E[w_z[k]] &= 0 && (4.7) \\ E[w_z[k]w_z[l]] &= \begin{cases} \sigma_{w_z}^2, & k = l \\ 0, & k \neq l \end{cases} && (4.8) \end{aligned}$$

After converting this model to a discrete time form, the factors shown in Figure 4-1 are specified as shown below. Each of the factors is formulated so that its error has a mean of zero and a covariance that can be specified.

$$\begin{aligned} f(\mathbf{x}[1]) &= r_1 - r_{\text{origin}} \sim N(0, \sigma_{\text{origin}}^2) && (4.9) \\ f(\mathbf{x}[k-1], \mathbf{x}[k]) &= r_k - r_{k-1} \sim N(0, a_{w_r}^2 (t_k - t_{k-1})) && (4.10) \\ f(\mathbf{x}[k], \mathbf{l}[i]) &= r_k - l_i + z_i[k] \sim N(0, \sigma_{w_z}^2) && (4.11) \end{aligned}$$

Note that r_{origin} is the location of the origin and $z_i[k]$ is a range measurement from the vehicle at time k to landmark i. Also, both r_{origin} and $z_i[k]$ are constants (i.e. a prior and a measurement respectively) and not states to be estimated.

Now, using the simple model described above, a few key facts can be observed. It is important to note that the process model described in Equation 4.3 models only the system's kinematics and not the dynamics. This is evident since there is no estimate of the vehicle's velocity. As a result, the velocity is assumed to be a zero mean white Gaussian noise with strength $a_{w_r}^2$. Therefore, this implies that Equation 4.10 describes a Wiener process[22] for the one-dimensional position of the vehicle.

In practice, this requires the process noise to be large enough to account for the vehicle's range of possible velocities. The problem with this approach is that the value of a_{w_r}, the strength of the process noise, sets a lower bound on the covariance of the estimated state variables. If it needs to be very large to account for a large range of velocities, this will lead to state estimates that have poor accuracy. Since the system is estimated in discrete time, the discrete time variance of $f(\mathbf{x}[k-1], \mathbf{x}[k])$ will increase proportionally to the time-step between measurements $t_k - t_{k-1}$ as shown in Equation 4.10.

Alternatively, if it is known that the system will obey certain dynamics, for example Newton's Second Law ($F = ma$), these dynamics can be used to further constrain the estimate as shown below.

Note that v_k is the vehicle's velocity at time k. The new state space model is specified below and includes the velocity as a state to be estimated. The landmark and measurement equations remain the same as described above.

$$\mathbf{x}[k] = \begin{bmatrix} r_k \\ v_k \end{bmatrix} \tag{4.12}$$

The continuous time model is shown below (m is the vehicle mass):

$$\begin{bmatrix} \dot{r}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} r(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} w_F(t) \tag{4.13}$$

We can discretize the system and replace the factor for the process model with the two dimensional factor model below

$$f(\mathbf{x}[k-1], \mathbf{x}[k]) = \begin{bmatrix} r_k \\ v_k \end{bmatrix} - \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} r_{k-1} \\ v_{k-1} \end{bmatrix} \sim N(\mathbf{0}_{2\times 1}, \mathbf{\Lambda}_{\text{process}}) \tag{4.14}$$

The first effect of this model is that the velocity will become part of the state space that is estimated at each time-step. Additionally, the relationship between velocity, position and the timestep (i.e. $v_k = \frac{r_k - r_{k-1}}{\Delta t}$) is a constraint that must be satisfied (or at least the error is part of the cost function that is minimized). An estimate of the velocity is often very useful for control and state propagation purposes.

The second effect is that the noise in the process model is now the force that is applied to the system. For typical space robotics applications, the applied forces are usually known to much higher accuracy than the velocities, because robotic space vehicles often have accelerometers, well calibrated actuators and few external disturbance forces; whereas the velocities are the result of integration of a long history of forces that are generally unknown.

The discussion of how to compute Λ_{process} based on $w_F(t)$ and Δt will be postponed until Section 4.4.

For the kinematics and dynamics model, the measurement factor is exactly the same as Equation 4.11. However, the origin factor is slightly different due to the fact that a prior must be placed on the velocity. This is shown below:

$$f(\mathbf{x}[1]) \;=\; \begin{bmatrix} r_1 - r_{\text{origin}} \\ v_1 - v_{\text{origin}} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma^2_{\text{r}_{\text{origin}}} & 0 \\ 0 & \sigma^2_{\text{v}_{\text{origin}}} \end{bmatrix} \right) \tag{4.15}$$

4.3 General Nonlinear Model

The next step is to generalize from the specific one-dimensional model described in the previous section. This general pose graph model is needed so that the specific non-linear rigid body dynamics can be included (described in Section 4.5). Note that a number of "less exact" methods were attempted during this research and were unsuccessful. These are described in the next two sections as a number of alternatives, which includes using zero-order hold process noise, using the matrix exponential for the state transition, using a constant process noise covariance matrix and using the matrix exponential for the process covariance matrix. Sections 4.3 to 4.5 present one of the main contributions, which is how to incorporate nonlinear rigid body dynamics into probabilistic factor graphs.

The concepts discussed in the previous section on one-dimensional models can be generalized to a nonlinear process model with a state vector of arbitrary size, as well as a discrete time measurement model as follows:

$$\dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t)\right) + \mathbf{B}_{\mathbf{w}}\mathbf{w}(t) \tag{4.16}$$

$$\mathbf{z}_i[k] = \mathbf{h}\left(\mathbf{x}[k], \mathbf{l}_i\right) + \mathbf{v}[k] \tag{4.17}$$

Where the process variables are defined with the following properties:

$$\mathbf{x}(t) \in \mathbb{R}_{n\times 1} \tag{4.18}$$

$$\mathbf{w}(t) \in \mathbb{R}_{m\times 1} \tag{4.19}$$

$$E[\mathbf{w}(t)] = \mathbf{0}_{m\times 1} \tag{4.20}$$

$$E[\mathbf{w}(t_1)\mathbf{w}(t_2)^T] = \mathbf{Q}\delta(t_1 - t_2) \tag{4.21}$$

$$\mathbf{Q} > 0 \tag{4.22}$$

This shows that $\mathbf{w}(t)$ is a Gaussian white noise process. Additionally, the measurement variables are defined with the properties below. Note that in this case the measurement noise has the same size as the measurement vector.

$$\mathbf{z}_i[k] \in \mathbb{R}_{u\times 1} \tag{4.23}$$

$$\mathbf{v}[k] \in \mathbb{R}_{u\times 1} \tag{4.24}$$

$$E[\mathbf{v}[k]] = \mathbf{0}_{u\times 1} \tag{4.25}$$

$$E[\mathbf{v}[k]\mathbf{v}[l]^T] = \begin{cases} \mathbf{R}, & k = l \\ \mathbf{0}, & k \neq l \end{cases} \tag{4.26}$$

$$\mathbf{R} \in \mathbb{R}_{u\times u} \tag{4.27}$$

$$\mathbf{R} > 0 \tag{4.28}$$

$$\mathbf{l}_i \in \mathbb{R}_{l\times 1} \tag{4.29}$$

From above, the measurement error process $\mathbf{v}[k]$ is also assured to be Gaussian. As before, the process and measurement noise are set as the error function in the iSAM factors that are the inputs to the cost function that is minimized. The measurement model is straightforward to transform to discrete time and incorporate into a factor:

$$\mathbf{f}\left(\mathbf{x}[k], \mathbf{l}[i]\right) = \mathbf{v}[k] = \mathbf{z}_i[k] - \mathbf{h}\left(\mathbf{x}[k], \mathbf{l}_i\right) \sim N(\mathbf{0}_{u\times 1}, \mathbf{R}) \tag{4.30}$$

However, the process dynamics equation can be discretized in one of two ways:

The first approach is to assume the noise is constant during the integration (i.e. a zero-order-hold [34]), where $t_{k+1} = t_k + \Delta t$.

$$\mathbf{x}[k+1] \;=\; \mathbf{x}[k] + \int_{t_{k-1}}^{t_k} \mathbf{f}\left(\mathbf{x}(\tau)\right) d\tau + \mathbf{\Gamma}_1 \mathbf{w}_1[k] \tag{4.31}$$

Note that the discussion of the specific method for evaluating $\int_{t_{k-1}}^{t_k} \mathbf{f}\left(\mathbf{x}(\tau)\right) d\tau$ will be deferred until sub-section 4.4. This leads to the factor below. The implication of this is that $\mathbf{w}_1[k] \in \mathbb{R}_{m\times 1}$ and therefore the dimension of the factor is m.

$$f_1(\mathbf{x}[k], \mathbf{x}[k+1]) \;=\; \mathbf{w}_1[k] \tag{4.32}$$

Where:

$$\mathbf{w}_1[k] \;=\; (\mathbf{\Gamma}_1^T \mathbf{\Gamma}_1)^{-1} \mathbf{\Gamma}_1^T \left(\mathbf{x}[k+1] - \mathbf{x}[k] - \int_{t_{k-1}}^{t_k} \mathbf{f}\left(\mathbf{x}(\tau)\right) d\tau \right) \tag{4.33}$$

$$E[\mathbf{w}_1[k]] \;=\; \mathbf{0} \tag{4.34}$$

$$E[\mathbf{w}_1[k]\mathbf{w}_1[l]^T] \;=\; \begin{cases} \mathbf{\Lambda}_1, & k = l \\ \mathbf{0}, & k \neq l \end{cases} \tag{4.35}$$

$$\;=\; \begin{cases} \Delta t \mathbf{Q}, & k = l \\ \mathbf{0}, & k \neq l \end{cases} \tag{4.36}$$

As will be discussed below, this first approach causes certain problems (i.e. insufficient constraints) when used with optimization systems such as iSAM. The second, and preferred, approach would be to integrate the noise throughout the timestep (this is described in Section 3.6 of[37] and in Section 5.3 of [22]).

$$\mathbf{x}[k+1] = \mathbf{x}[k] + \int_{t_{k-1}}^{t_k} \mathbf{f}\left(\mathbf{x}(\tau)\right) d\tau + \mathbf{w}_2[k] \tag{4.37}$$

$$f_2(\mathbf{x}[k+1], \mathbf{x}[k]) = \mathbf{w}_2[k] \tag{4.38}$$

$$\mathbf{w}_2[k] = \mathbf{x}[k+1] - \mathbf{x}[k] - \int_{t_{k-1}}^{t_k} \mathbf{f}\left(\mathbf{x}(\tau)\right) d\tau \tag{4.39}$$

$$E[\mathbf{w}_2[k]] = \mathbf{0} \tag{4.40}$$

$$E[\mathbf{w}_2[k]\mathbf{w}_2[l]^T] = \begin{cases} \mathbf{\Lambda}_2, & k = l \\ \mathbf{0}, & k \neq l \end{cases} \tag{4.41}$$

Note that the discussion of the method for evaluating $\mathbf{\Lambda}_2$ will also be deferred until section 4.4.

In this second case: $\mathbf{w}_2[k] \in \mathbb{R}_{n\times 1}$ and the dimension of the factor is n. Therefore, the number of rows of $\mathbf{w}_1$, (m) will be smaller than the number of rows of $\mathbf{w}_2$, (n), since $m \leq n$ for most systems. This lower dimensionality (and hence lower computational cost) is why the first approach is typically considered for Extended Kalman Filters or graph-based message passing algorithms (see [134] for an EKF example).

However, pose graph optimizations algorithms, such as iSAM, solve a system of equations using weighted least squares minimization algorithm. Specifically, iSAM minimizes the Mahalanobis distance of the factors' error functions as shown below.

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} ||\mathbf{w}(\mathbf{x})||^2_{\mathbf{\Lambda}(\mathbf{x})} \tag{4.42}$$

$$= \arg\min_{\mathbf{x}} \mathbf{w}(\mathbf{x})^T \mathbf{\Lambda}^{-1}(\mathbf{x})\mathbf{w}(\mathbf{x}) \tag{4.43}$$

Note that the covariance Λ will also be a function of the state. This is not typical for standard kinematics-only models of pose-graph problems. This required a few minor software modifications to be made to the iSAM software to support this, which are shown in Listings B.18 and B.19.

As an aside, note that this requires the computation of the Jacobian of the error: $\mathbf{J}(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}}\mathbf{w}(\mathbf{x})$. The iSAM software system will compute the Jacobian (J) numerically, which will allow a numerical integration method to be used for evaluating the error $\mathbf{w}(\mathbf{x})$ in Section 4.4. Note that for the weighted least squares problem, the Jacobian is only the partial derivative of the error, and does not require a partial on the Information matrix $\mathbf{\Lambda}^{-1}(\mathbf{x})$.

To decide whether $\mathbf{w}_1[k]$ or $\mathbf{w}_2[k]$ should be used, the number of variables and constraints in the optimization problem must be considered. Note that the variables are generated by the nodes while the constraints are generated by the factors. The total number of variables is equal to the total number of rows in all of the nodes. Similarly, the total number of constraints in a pose graph is equal to the total number of rows in all of the factors' error functions. The weighted least squares minimization for each of the nodes is as follows:

When the first approach is used, $\mathbf{w}_1$, the total number of constraints will be less than the total number of variables. This is because when time derivatives of states are also included as state vectors, this leads to more states in the state vector that are driven by the same noise. As a result, the number of constraints $m < n$. and the pose graph optimization algorithm will fail to find a solution if the first method ($\mathbf{w}_1$) is used, since the problem is under-constrained. The second approach, $\mathbf{w}_2$, will not have this problem, since there will be an equal number of constraints and variables.

Admittedly, this conclusion seems counter-intuitive, and asks the question: Does the second approach just add combinations of other constraints (so that the problem remains ill-posed), or does it actually add new independent constraints that allow for the problem to be solved?

The answer to this question depends on the covariance matrix $\mathbf{\Lambda}_2$. If $\mathbf{\Lambda}_2$ is rank deficient, Equation 4.42 definitely can not be solved, and the pose-graph optimization will fail. If the matrix Λ_2 is full rank, but poorly conditioned, a solution to the system of equations exists, but not all solution methods will find the correct solution[48]. Practically speaking, this conclusion aligns with the implementation of iSAM: the first step in constructing a factor is to find a Cholesky factorization of $\mathbf{\Lambda}_2^{-1}$. If $\mathbf{\Lambda}_2^{-1}$ is poorly conditioned, the factorization will be inaccurate, and hence iSAM will find an inaccurate solution.

The following section discusses the evaluation of $\mathbf{\Lambda}_2$, while Section 4.9 looks at the conditioning of $\mathbf{\Lambda}_2^{-1}$, and under what conditions it will allow the least squares model to be solved.

4.4 Evaluation of Process Noise

As discussed in the previous section, for a general nonlinear process model as described above, there is an equivalent factor that can be created:

$$
\begin{aligned}
\dot{\mathbf{x}}(t) &= \mathbf{f}\left(\mathbf{x}(t)\right) + \mathbf{B}_\mathbf{w}\mathbf{w}(t) && (4.44)\\
f(\mathbf{x}[k+1], \mathbf{x}[k]) &= \mathbf{x}[k+1] - \mathbf{x}[k] - \int_{t_{k-1}}^{t_k} \mathbf{f}\left(\mathbf{x}(\tau)\right) d\tau && (4.45)\\
&\sim N(\mathbf{0}, \mathbf{\Lambda}(\Delta t, \mathbf{x}[k])) && (4.46)
\end{aligned}
$$

One option is to use the fact that nonlinear systems can be linearized about the point $\mathbf{x}(t_k)$ as follows. Also, $A(\mathbf{x}(t)) = \frac{\partial \mathbf{f}(\mathbf{x}(t))}{\partial \mathbf{x}(t)}|_{\mathbf{x}(t)=\mathbf{x}(t_k)}$ and may be a function of the state vector:

$$
\begin{aligned}
\dot{\mathbf{x}}(t) &= \mathbf{f}\left(\mathbf{x}(t)\right) + \mathbf{B}_\mathbf{w}\mathbf{w}(t) && (4.47)\\
&\approx \mathbf{A}\left(\mathbf{x}(t_k)\right)\mathbf{x}(t) + \mathbf{B}_\mathbf{w}\mathbf{w}(t) && (4.48)
\end{aligned}
$$

Applying a discretization to transform the system from continuous to discrete time with a time step of Δt leads to:

$$
\begin{aligned}
\mathbf{x}[k+1] &= \mathbf{\Phi}(\Delta t, \mathbf{x}[k])\mathbf{x}[k] + \int_{t_k}^{t_{k+1}} e^{\mathbf{A}(\mathbf{x}[k])(t_{k+1}-\tau)}\mathbf{B}_\mathbf{w}\mathbf{w}(\tau)d\tau && (4.49)\\
&= \mathbf{\Phi}(\Delta t, \mathbf{x}[k])\mathbf{x}[k] + \mathbf{w}[k] && (4.50)\\
\mathbf{\Phi}(\Delta t, \mathbf{x}[k]) &= e^{\mathbf{A}(\mathbf{x}[k])\Delta t} = \mathcal{L}^{-1}(s\mathbf{I} - \mathbf{A}(\mathbf{x}[k])\Delta t)^{-1} && (4.51)
\end{aligned}
$$

The properties of $\mathbf{w}[k]$, the factors' error function, are found using the same approach as in Equation 4.39 (i.e. Section 3.6 of[37] and in Section 5.3 of [22]):

$$\begin{aligned}
\mathbf{w}[k] &= x[k+1] - \Phi(\Delta t, \mathbf{x}[k])x[k] && (4.52)\\
E[\mathbf{w}[k]] &= \mathbf{0} && (4.53)\\
\mathbf{\Lambda}(\Delta t, \mathbf{x}[k]) &= E[\mathbf{w}[k]\mathbf{w}[k]^T] && (4.54)\\
&= \int_{t_k}^{t_{k+1}} \mathbf{\Phi}(\tau, \mathbf{x}[k])\mathbf{B_w}\mathbf{Q}\mathbf{B_w}^T\Phi(\tau, \mathbf{x}[k])^T d\tau && (4.55)
\end{aligned}$$

Equation 4.55 has a very important property: Whenever the transition matrix $\mathbf{\Phi}$ explicitly depends on $\mathbf{x}[k]$, then the covariance of the factors' error Λ will also depend on $\mathbf{x}[k]$. This will occur in the section 4.5, for the attitude kinematics and dynamics. This dependency has the practical implementation that the transition matrix, $\mathbf{\Phi}$, and factor covariance Λ must be recalculated whenever the value of the estimate $\mathbf{x}[k]$ (i.e. the vehicle's state) changes, which can add significant computation to the overall system. If the dynamic models are not included and only the kinematic models are used, this will typically not be required and $\mathbf{\Phi}$ and Λ can be pre-programmed ahead of operations.

While the linearization seems like a perfectly valid approach, there are two inherent problems with using: $A(\mathbf{x}(t)) = \frac{\partial \mathbf{f}(\mathbf{x}(t))}{\partial \mathbf{x}(t)}|_{\mathbf{x}(t)=\mathbf{x}(t_k)}$. The first is a theoretical problem: This linearization, and its use in computing $\mathbf{\Phi}(\Delta t, \mathbf{x}[k])$ and $\mathbf{\Lambda}(\Delta t, \mathbf{x}[k])$, assumes that $\mathbf{x}[k]$ remains constant over the time step Δt. However, if Δt is sufficiently large, such that the linearization point from the previous time step is no longer "close enough" to the linearization point at the next time step, additional errors will be introduced to the estimation system.

The second problem is due to the software implementation of Equation 4.51 and 4.55. If the system is sufficiently complex, and this includes a 6 Degree of Freedom rigid body as described in section 4.5, then the symbolic evaluations of the inverse Laplace Transform and definite integral can be prohibitively complicated, even when using automatic code generation from symbolic representations such as Matlab's Symbolic and Coder Toolkit.

An alternative and preferable method, which was used in this thesis, is to compute the process model and noise covariance numerically, despite the fact that this adds additional computational steps to the estimation scheme. This method uses a custom fourth-order Runge-Kutta (RK4) integration, with adaptive step sizes. The custom method to adapt the step size initially evaluates the RK4 integration with Δt as the first step size and $\Delta t/2$ as the second step size. It compares the two results, and if the root mean squared (RMS) error is greater than a predetermined threshold, it reduces the step size by a factor of two until the RMS error is below the threshold (or if the step size is smaller than an absolute minimum). The source code for the RK4 integration is shown in Listing B.14 and B.15.

In order to compute the process model, Equation 4.16 is used, along with the fact that $E\left[\mathbf{w}(t)\right] = \mathbf{0}$, and the RK4 can be applied in the usual manner.

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}\left(\mathbf{x}(t)\right) && (4.56) \\ \mathbf{x}(t) &= \mathbf{x}_0 && (4.57) \end{aligned}$$

Now to compute the process covariance, we do not use Equation 4.55, due to the above mentioned problems, but rather use the fact that $\boldsymbol{\Lambda}$, from Equation 4.21, follows a Lyapunov equation[22, 37]:

$$\begin{aligned} \dot{\boldsymbol{\Lambda}}(\Delta t, \mathbf{x}(t)) &= \mathbf{A}(\mathbf{x}(t))\boldsymbol{\Lambda}(\Delta t, \mathbf{x}(t)) + \boldsymbol{\Lambda}(\Delta t, \mathbf{x}(t))\mathbf{A}^T(\mathbf{x}(t)) + \mathbf{B}_{\mathbf{w}}\mathbf{Q}\mathbf{B}_{\mathbf{w}}^T && (4.58) \\ \boldsymbol{\Lambda}(0, \mathbf{x}(t)) &= \mathbf{0} && (4.59) \end{aligned}$$

Note that if $\mathbf{A}$ depends on $\mathbf{x}(t)$, then $\boldsymbol{\Lambda}$ also depends on $\mathbf{x}(t)$, and both must be solved simultaneously. As a result Equations 4.56 through 4.59 are solved simultaneously using the RK4 method described above. Note that the positive definite matrix $\boldsymbol{\Lambda}$ can be vectorized to integrate with the software functions. The model for the Lyapunov equation must be linearized, using $\mathbf{A}(\mathbf{x}(t))$, otherwise the Gaussian process noise would not remain Gaussian. As a result the RK4 step size must be able to be adjusted to be small enough that the linearization is valid between integration steps. This is taken care of by the adaptive step size method described above.

4.5 Six Degree of Freedom Rigid Body Model

The next step is to apply the previously generic mode to the specific problem at hand.The stochastic, continuous-time, nonlinear dynamics are specified by the following equations. This is a constant linear and angular acceleration model with zero mean white Gaussian noise as the only disturbance forces and torques. $\mathbf{W_v}, \mathbf{W}_\omega$ are process noise models that incorporate disturbance forces that are applied to the vehicle. Note that no assumptions are made that the angles or angular velocity is small.

$$\begin{aligned}
\dot{\mathbf{r}} &= \mathbf{v} && (4.60)\\
\dot{\mathbf{v}} &= \frac{1}{m}\mathbf{W_v} && (4.61)\\
\dot{\mathbf{q}} &= \frac{1}{2}\mathbf{\Omega}(\omega)\mathbf{q} && (4.62)\\
&= \frac{1}{2}\begin{bmatrix}\omega\\0\end{bmatrix} \otimes \mathbf{q} && (4.63)\\
&= \frac{1}{2}\begin{bmatrix}\omega\\0\end{bmatrix} \otimes \begin{bmatrix}\bar{\mathbf{q}}\\q_4\end{bmatrix} && (4.64)\\
\dot{\omega} &= \mathbf{J}^{-1}(-\omega \times \mathbf{J}\omega + \mathbf{W}_\omega) && (4.65)\\
&= -\mathbf{J}^{-1}\omega \times \mathbf{J}\omega + \mathbf{J}^{-1}\mathbf{W}_\omega && (4.66)
\end{aligned}$$

Where the $\mathbf{J}$ is the inertia matrix, and:

$$\mathbf{\Omega}(\omega) = \begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 \\ -\omega_3 & 0 & \omega_1 & \omega_2 \\ \omega_2 & -\omega_1 & 0 & \omega_3 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 \end{bmatrix} \qquad [\omega\times] = \begin{bmatrix} 0 & \omega_3 & -\omega_2 \\ -\omega_3 & 0 & \omega_1 \\ \omega_2 & -\omega_1 & 0 \end{bmatrix} \qquad (4.67)$$

It is clear that any representations of attitude include nonlinear transformations and kinematics. This causes a problem for modelling and propagating probability distribution functions with Gaussian random variables, such as those typically used in Extended Kalman Filters or the iSAM system. It is well understood that the covariance matrix of a quaternion is rank deficient due to its normalization constraint. While there is active research in a number of estimation systems that do not use Gaussian random variables [83, 109, 127, 118, 119], a typical approach for dealing with this is to use three vector error parameterization and reset the quaternion (see [26, 70, 82, 134]), which is what will be used here since it fits well with the iSAM system for Gaussian random variables and has a history of good performance.

This error vector and reference quaternion approach can be applied to pose graph optimization methods such as iSAM. For each of the nodes that specify the vehicle's 6DOF trajectory, the reference quaternion approach is mirrored. This means that at the vehicle's state nodes for each timestep, both a four parameter reference quaternion and a three parameter attitude error is stored. Each time the optimization problem is re-linearized, a reset step is performed. This reset step transfers all of the attitude error into the reference quaterion.

A three parameter representation was chosen so that the actual state vector is 12 by 1. Modified Rodriguez Parameters (MRP), $\mathbf{a}_p$, were chosen so that the singularity is at a 360^o rotation.

$$\mathbf{x} \;=\; \begin{bmatrix} \mathbf{r} & \mathbf{v} & \mathbf{a}_p & \omega \end{bmatrix}^T \tag{4.68}$$

Where:

$$\mathbf{a}_p(\mathbf{q}) \;=\; \frac{4}{1+q_4}\begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}^T = \frac{4}{1+q_4}\bar{\mathbf{q}} \tag{4.69}$$

And, inversely:

$$\mathbf{q}(\mathbf{a}_p) \;=\; \frac{1}{16+\mathbf{a}_p^T\mathbf{a}_p}\begin{bmatrix} 8\mathbf{a}_p \\ 16-\mathbf{a}_p^T\mathbf{a}_p \end{bmatrix} \tag{4.70}$$

In the Multiplicative Extended Kalman Filtering (MEKF) approaches referenced above, a reference quaternion $\mathbf{q}_{\text{ref}}$ is stored separately from the state vector and its covariance matrix. Periodically, a MEKF will apply the reset step below based on the error represented by the Modified Rodrigues Parameters. This idea is mirrored in the iSAM implementation described in this paper. Each of the nodes that contains the vehicle's state vector includes a three-parameter MRP that is part of the state vector and covariance matrix for that node. Additionally, each node includes a reference quaternion that is not part of the state vector. With a few simple modifications, the publicly available iSAM implementation can be modified to apply the reset step as part of its re-linearization routine.

$$\mathbf{q}[k] = \delta\mathbf{q}(\mathbf{a}_p[k]) \otimes \mathbf{q}_{ref}[k] \tag{4.71}$$

$$\mathbf{a}_p[k] = \mathbf{0}_{3\times1} \tag{4.72}$$

It is important to note that any error functions that make use of the full attitude, must include the full attitude function and do so in a way that avoids the singularities associated with the MRP. The method is used as part of the error function (Equation 4.39) is listed below. Note that $\mathbf{q}^{-1}$ is the inverse of a quaternion, where the three direction components are multiplied by -1. The details of $\dot{\mathbf{a}}_p(t)$ are derived below.

$$\mathbf{a}[k+1] - \mathbf{a}[k] = \int_{t_k}^{t_{k+1}} \dot{\mathbf{a}}_p(\tau)d\tau + \mathbf{a}_p\left(\mathbf{q}[k+1] \otimes \mathbf{q}[k]^{-1}\right) \tag{4.73}$$

In order to model the state transitions using $\mathbf{a}_p$, the time derivative $\dot{\mathbf{a}}_p = \frac{d\mathbf{a}_p}{dt}$ must be found. The two definitions below are used in the derivation:

$$\dot{\bar{\mathbf{q}}} = \frac{1}{2}\left(q_4\omega - \omega \times \bar{\mathbf{q}}\right) \tag{4.74}$$

$$\dot{q}_4 = -\frac{1}{2}\omega \cdot \bar{\mathbf{q}} \tag{4.75}$$

Now, $\dot{\mathbf{a}}_p$ was derived in Equation 2.47.

$$\frac{d\mathbf{a}_p(t)}{dt} = \dot{\mathbf{a}}_p(\mathbf{a}_p(t), \omega(t)) = \left(-\frac{1}{2}[\omega\times] + \frac{1}{8}\omega \cdot \mathbf{a}_p\right)\mathbf{a}_p + \left(1 - \frac{1}{16}\mathbf{a}_p^T\mathbf{a}_p\right)\omega \tag{4.76}$$

Also, $\dot{\mathbf{a}}_p$ is only a function of $\mathbf{a}_p$ and $\mathbf{w}$. To summarize, the full nonlinear dynamics are as follows:

$$\begin{bmatrix} \dot{\mathbf{r}} \\ \dot{\mathbf{v}} \\ \dot{\mathbf{a}} \\ \dot{\mathbf{w}} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{0} \\ \left(-\frac{1}{2}[\omega\times] + \frac{1}{8}\omega \cdot \mathbf{a}_p\right)\mathbf{a}_p + \left(1 - \frac{1}{16}\mathbf{a}_p^T\mathbf{a}_p\right)\omega \\ -\mathbf{J}^{-1}\omega \times \mathbf{J}\omega \end{bmatrix} + \tag{4.77}$$

$$\begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \frac{1}{m}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{J}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{W}_{\mathbf{v}} \\ \mathbf{W}_{\omega} \end{bmatrix} \tag{4.78}$$

Equation 4.58 requires that a linearized model be computed to propagate through the Lyapunov equation to ensure the process noise remains Gaussian.

$$\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{B}_W\mathbf{W} \tag{4.79}$$

$$\begin{bmatrix} \dot{\mathbf{r}} \\ \dot{\mathbf{v}} \\ \dot{\mathbf{a}}_p \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\frac{1}{2}[\omega\times] + \frac{1}{8}\omega \cdot \mathbf{a}_p\mathbf{I}_{3\times3} & \left(1 - \frac{1}{16}\mathbf{a}_p^T\mathbf{a}_p\right)\mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathbf{J}^{-1}[\omega\times]\mathbf{J} \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{v} \\ \mathbf{a}_p \\ \omega \end{bmatrix} \tag{4.80}$$

$$+ \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \frac{1}{m}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{J}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{W}_{\mathbf{v}} \\ \mathbf{W}_{\omega} \end{bmatrix} \tag{4.81}$$

The only assumption in the linear model is that the state remains constant during the integration step, which is accommodated by the adaptive step size RK4 algorithm. Neither the linear or nonlinear model make any assumptions about small angles or positions or small linear or angular velocities. The source code for the dynamics model is shown in Listing B.16 and B.17. Note that it assumes the inertia ratios will be estimated (as discussed in Section 4.7) and therefore uses Equation 4.101 and

4.103.

The singularity in the three parameter rotation representation is handled by the reset step in Equation 4.71. This was added to the standard open source iSAM implementation.

4.6 Modeling of Center of Mass and Principal Axes of Inertia

This section outlines one of the main contributions of this thesis: how to parameterize and estimate the center of mass and principal axes of inertia. All of the equations representing the process model have been written with the assumption that the body fixed frame is located at the center of mass and (optionally) aligned with the principal axes. The approach taken in this thesis is to enforce this assumption, by minimizing the external forces and torques as part of the factors' errors, while introducing another reference frame that all of the feature points are attached to. The translation and rotation between the first, body-fixed frame and the second, geometric frame is a constant parameter that must be estimated.

The reason for taking this approach is due to the following issue with the more conventional approach: As the estimates for the center of mass and principal axes are refined, there is a "common mode" adjustment that must be made to all of the feature points. Adding these translation and rotation parameters between the rigid body frame and the geometric frame moves these common mode adjustments from the feature points to these new parameters, and thereby minimizing the number of variables that must be updated.

This is different than the traditional SLAM approach where the feature points would be estimated within the body fixed reference frame. This approach is conceptually similar to a single, three-dimensional anchor node as proposed by Kim [61], where the measurements are triangulated SURF features rather than kinematic transformations. The biggest difference is where Kim's anchor nodes are used to "align" two robot trajectories with each other, the geometric frame transformation described here is used to “align” the body frame trajectory into the center of mass and principal axes. Aghili [7, 6] and Lichter [75, 76] both specify a very similar model, but neither of the two authors estimate feature positions with respect to the geometric frame as part of the state vector. This would allow for loop closure and future refinement (which are both possible with the methods described by this thesis), but it introduces additional observability issues. With the appropriate prior models for the feature points (as described later in this section), this thesis shows how to handle this observability issue.

This approach is illustrated in Figure 4-2. The state variables include a large number three dimensional vectors $\mathbf{p}_{i/G}$, which are the location of each matched and tracked SURF feature (green and purple stars in the figure) with respect to the geometric frame. The fact that the features are estimated with respect to the geometric frame and not the body frame is the first key to this approach. The second key to this approach is that the parameters for the coordinate frame transformation between the body and geometric frame are state variables estimated by a single node iSAM system. The fact that the parameters of this transformation are constant and do not vary over the time-steps k is the main reason for choosing this approach. This means that while the relative positions of the feature points, with respect to other feature points, may have converged to very accurate estimates, the transformation between the geometric and the body frame may not converge until a much later time, when the motion is rich enough to enforce the zero force and torque constraints.

Note that the parameters are shown in the diagram as $\mathbf{R}_{G/B}, \mathbf{T}_{G/B}$ for simplicity but the rotation is actually represented as the reference quaternion, $\mathbf{q}_{G/B,\text{ref}}$, and MRP error vector, $\mathbf{a}_{p,G/B,\text{ref}}$, as in Equation 4.71.

Figure 4-3 illustrates the factor graph relationship between the geometric frame, the sensor factors and the body frame state variables. Again, note that there is only one instance of the $\mathbf{R}_{G/B}, \mathbf{T}_{G/B}$ node for all of the time k. This parameter “collects” all of the common-mode adjustments to the geometric feature points. These rotation and translation parameters are able to coverge at a later time when the trajectory is

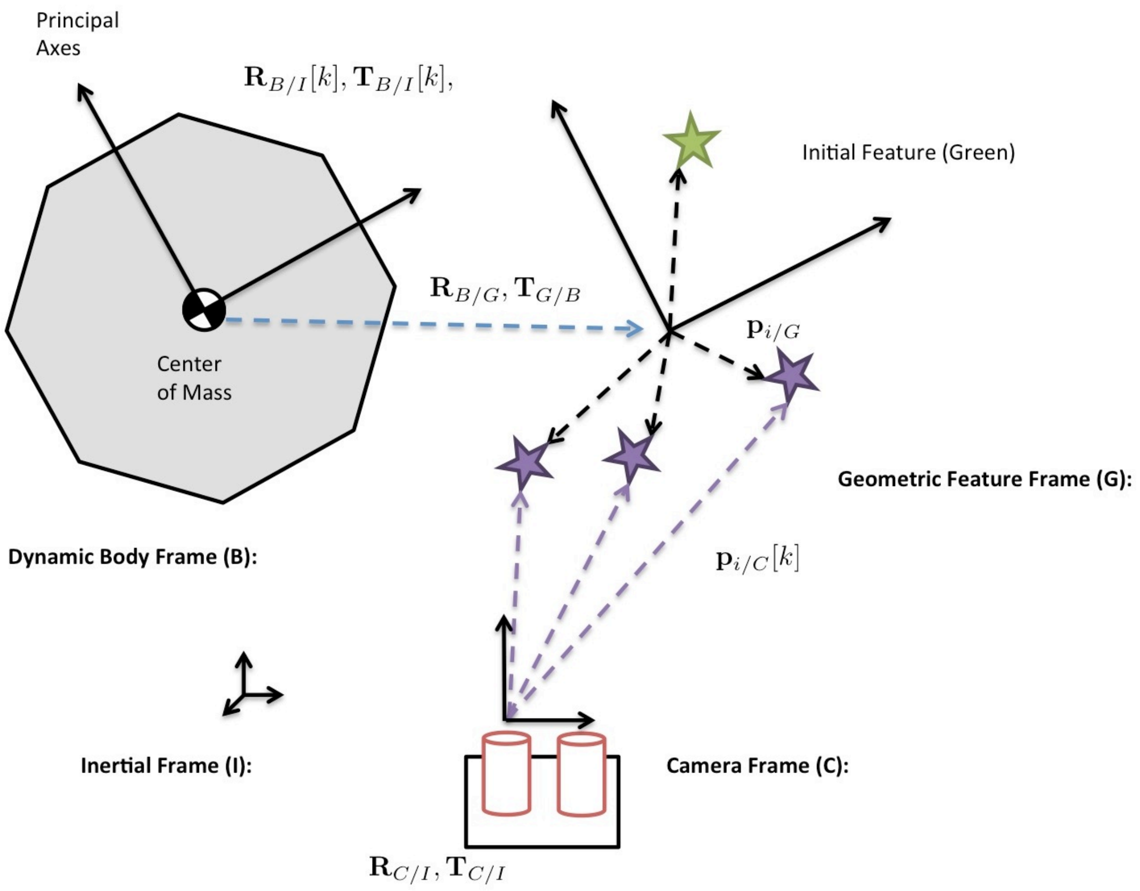


Figure 4-2: Geometric, Body, Camera and Inertial Reference Frames

rich enough that the center of mass and principal axes of inertia become observable.

Using these reference frames, the feature points can be transformed into the inertial frame to find $\mathbf{p}_{i/I}$, by adding the second transformation matrix in the equation below.

$$\begin{bmatrix} \mathbf{p}_{i/I} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}'_{B/I}[k] & \mathbf{T}_{G/B}[k] \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}'_{B/G} & \mathbf{T}_{G/B} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i/G} \\ 1 \end{bmatrix} \tag{4.82}$$

Once these features are known in the inertial frame, they can be projected into the camera frame, which for this thesis is assumed to be stationary (removing this assumption is planned as future work and is described in Section 7.2).

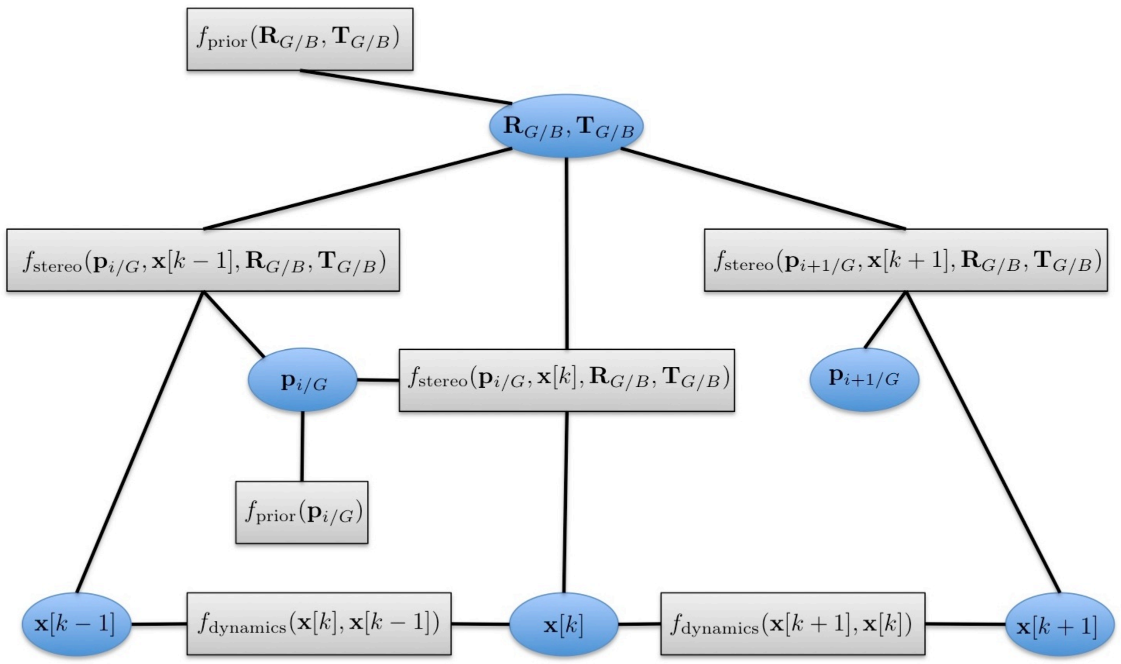

$f_{\text{prior}}(\mathbf{R}_{G/B}, \mathbf{T}_{G/B})$
$\mathbf{R}_{G/B}, \mathbf{T}_{G/B}$
$f_{\text{stereo}}(\mathbf{p}_{i/G}, \mathbf{x}[k-1], \mathbf{R}_{G/B}, \mathbf{T}_{G/B})$
$f_{\text{stereo}}(\mathbf{p}_{i+1/G}, \mathbf{x}[k+1], \mathbf{R}_{G/B}, \mathbf{T}_{G/B})$
$\mathbf{p}_{i/G}$
$f_{\text{stereo}}(\mathbf{p}_{i/G}, \mathbf{x}[k], \mathbf{R}_{G/B}, \mathbf{T}_{G/B})$
$\mathbf{p}_{i+1/G}$
$f_{\text{prior}}(\mathbf{p}_{i/G})$
$\mathbf{x}[k-1]$
$f_{\text{dynamics}}(\mathbf{x}[k], \mathbf{x}[k-1])$
$\mathbf{x}[k]$
$f_{\text{dynamics}}(\mathbf{x}[k+1], \mathbf{x}[k])$
$\mathbf{x}[k+1]$

Figure 4-3: Factor Graph Model of Sensor and Process Model using Geometric Frame

$$s_L \begin{bmatrix} u_{i/L} \\ v_{i/L} \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & -c_{x,L} \\ 0 & f & -c_{y,L} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{C_L/I} & -\mathbf{R}_{C_L/I}\mathbf{T}_{C_L/I} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i/I} \\ 1 \end{bmatrix} \tag{4.83}$$

$$s_R \begin{bmatrix} u_{i/R} \\ v_{i/R} \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & -c_{x,R} \\ 0 & f & -c_{y,R} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{C_R/I} & -\mathbf{R}_{C_R/I}\mathbf{T}_{C_R/I} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i/I} \\ 1 \end{bmatrix} \tag{4.84}$$

Using these image frame coordinates, the error vector of the factor can be computed using the measured image coordinates of the SURF feature $(u_{L,\text{measured}}, v_{L,\text{measured}}, u_{R,\text{measured}})$.

$$f_{\text{stereo}}(\mathbf{p}_{i/G}, \mathbf{x}[k+1], \mathbf{R}_{G/B}, \mathbf{T}_{G/B}) = \begin{bmatrix} u_{i/L,\text{measured}} \\ v_{i/L,\text{measured}} \\ u_{i/R,\text{measured}} \end{bmatrix} - \begin{bmatrix} u_{i/L} \\ v_{i/L} \\ u_{i/R} \end{bmatrix} \quad (4.85)$$

Another important point is that the model shown in Figure 4-3 includes two prior factors. The first prior is $f_{\text{prior}}(\mathbf{R}_{G/B}, \mathbf{T}_{G/B})$, which is required to allow the number of constraints to equal the number of degrees of freedom. However, this prior does not introduce a significant amount of information due to the fact that the covariance of this factor is very high.

The second prior is slightly more complicated as it is only applied to the first point feature in the entire geometric map (i.e. the green star in figure 4-2). Note that there is an unobservable mode between the feature point locations in the geometric frame and $\mathbf{R}_{G/B}, \mathbf{T}_{G/B}$. For example if the geometric frame is translated by 10 centimeters with respect to the body frame, and all of the features are translated by 10 centimeters in the opposite direction, the same stereo vision measurements will result (note that a similar situation applies for rotation). In order to deal with this and help the system converge faster, a single prior is placed on one of the feature points, $f_{\text{prior}}(\mathbf{p}_{i/G})$ that has very low covariance. An offset is computed based on the coordinate frames' locations when this feature is first measured in order to maintain a zero mean factor. This in effect "locks down" the position and orientation of the features within the geometric frame. Note that because this procedure does not introduce any information in the body or inertial frame, it does not add any a priori information to the problem.

The initialized value of this first low covariance feature is computed by triangulating the feature measurement to obtain $\mathbf{p}_{i/I}$ and then solving for $\mathbf{p}_{i/G}$ with Equation 4.82 using the current estimated location of the body frame and the current (likely initial) value of $\mathbf{R}_{B/G}$ and $\mathbf{T}_{G/B}$. The initial values for $\mathbf{R}_{B/G}$ and $\mathbf{T}_{G/B}$ are the identity rotation matrix and the zero vector for translation.

One important potential issue is that location of this low covariance point must be far enough away from the origin of the geometric frame so that it can properly constrain the rotation. In other words, solving for $\mathbf{p}_{i/G}$ must not lead to a value that is very close to zero and if this occurs a different feature should be selected. A good "rule-of-thumb" would be that the value cannot be within 6 standard deviations of origin. Since the prior standard deviation is $1.0E-6$ meters, this feature has to be at least 6 nano-meters away from the origin. This is sufficiently small that it is unlikely to occur. As a result, the implementation of this algorithm did not include this check, but it could easily be modified to include this for robustness.

4.7 Parameterizing the Ratios of Inertia

This section describes another of the main contributions of this thesis: How to parameterize and estimate the observable modes of the principal inertia matrix. Given an angular velocity trajectory, $\omega(t)$, it is desirable to find the inertia properties that solve Euler's Equation of Rotation Motion for torque free input. This is Equation 4.86, which is repeated below:

$$\dot{\omega}_x = \frac{J_{yy} - J_{zz}}{J_{xx}}\omega_y\omega_z \tag{4.86}$$

$$\dot{\omega}_y = \frac{J_{zz} - J_{xx}}{J_{yy}}\omega_x\omega_z \tag{4.87}$$

$$\dot{\omega}_z = \frac{J_{xx} - J_{yy}}{J_{zz}}\omega_y\omega_x \tag{4.88}$$

The observability of the ratios of inertia is discussed in Section 3.3, where it was shown that only two degrees of freedom are observable. Therefore it is desirable to parameterize the inertia matrix as two "ratios of inertia".

As a comparison, Aghili [7] described an Extended Kalman Filter based approach where the inertias were parameterized as three variables as follows:

$$p_x = \frac{I_y - I_z}{Ix} > -1 \tag{4.89}$$

$$p_y = \frac{I_z - I_x}{Iy} > -1 \tag{4.90}$$

$$p_z = \frac{I_x - I_y}{Iz} > -1 \tag{4.91}$$

It is clear that these three parameters only have two degrees of freedom, since any of the parameters can be found as a function of the other two. An example of this is:

$$p_x = -\frac{p_y + p_z}{1 + p_y p_z} \tag{4.92}$$

Although a single set of experimental results were presented in [7] showing good convergence of an EKF, the author of this thesis does not believe that it is good practice to parameterize a problem with more variables than degrees of freedom. Additionally, p_x, p_y and p_z must be greater than -1 in order for Euler's equations to generate physically realistic results. This does not seem compatible with Gaussian random variables that are defined for all real numbers between infinity and negative infinity. These two reasons are why the author of this thesis believes that Aghili's parametrization is not ideal.

Lichter proposed a parameterization of the inertia properties in his doctoral thesis [76] that are implemented as an Unscented Kalman Filter[1]. His parameterization is as defined as:

$$I_{xx} = |z_2| + |z_3| \tag{4.93}$$

$$I_{yy} = |z_1| + |z_3| \tag{4.94}$$

$$I_{zz} = |z_1| + |z_2| \tag{4.95}$$

Where the four values of the quaternion $\begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$ are estimated by the UKF:

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \mathbf{R}(\mathbf{q}_I) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 2(q_1 q_3 + q_2 q_0) \\ 2(q_2 q_3 + q_1 q_0) \\ q_0^2 - q_1^2 - q_2^2 - q_3^2 \end{bmatrix} \tag{4.96}$$

This method is problematic since it uses four variables to represent two degrees of freedom. Additionally, there are a number quaternions that do not correspond to physical situations. For example, the quaternion $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$ would imply that $I_{zz} = 0$, which does not make physical sense.

In contrast to Aghili's and Lichter's methods, this thesis proposes a parameterization that is the natural logarithm of the ratios of inertia. This approach has only two random variables for two degrees of freedom: k_1 and k_2,

$$k_1 = \ln\left(\frac{J_{xx}}{J_{yy}}\right) \tag{4.97}$$

$$k_2 = \ln\left(\frac{J_{yy}}{J_{zz}}\right) \tag{4.98}$$

These parameters are assumed to be Gaussian random variables. Using this approach, the diagonal inertia matrix can be computed up to a scale factor:

$$\mathbf{J}_{\text{diag}} = \begin{bmatrix} e^{k_1} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & e^{-k_2} \end{bmatrix} = \begin{bmatrix} \frac{J_{xx}}{J_{yy}} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{J_{zz}}{J_{yy}} \end{bmatrix} \tag{4.99}$$

Additionally, this parameterization is an appropriate selection for a Gaussian random variable. This is due to the fact that $0 < \frac{J_{xx}}{J_{yy}}, \frac{J_{yy}}{J_{zz}} < \infty$, which is the same as the input domain for the natural logarithm, ln.

If $k_1, k_2 = 0$, which maximizes the probability distribution function for a zero mean normal random variable, then$J_{xx} = J_{yy} = J_{zz}$, which means that the object has a spherical inertia ellipsoid. Also, as $k_1, k_2 \to \pm\infty$ the Gaussian distribution approaches zero, implying that one of the inertials is infinitesimally small. Such an occurrance is highly unlikely from a physical perspective, so the probabilistic model is consistent with the physical model.

This approach was used in this thesis by adding a new node for k_1 and k_2 and augmenting the dynamics factor f_{dynamics} as shown in Figure 4-4.

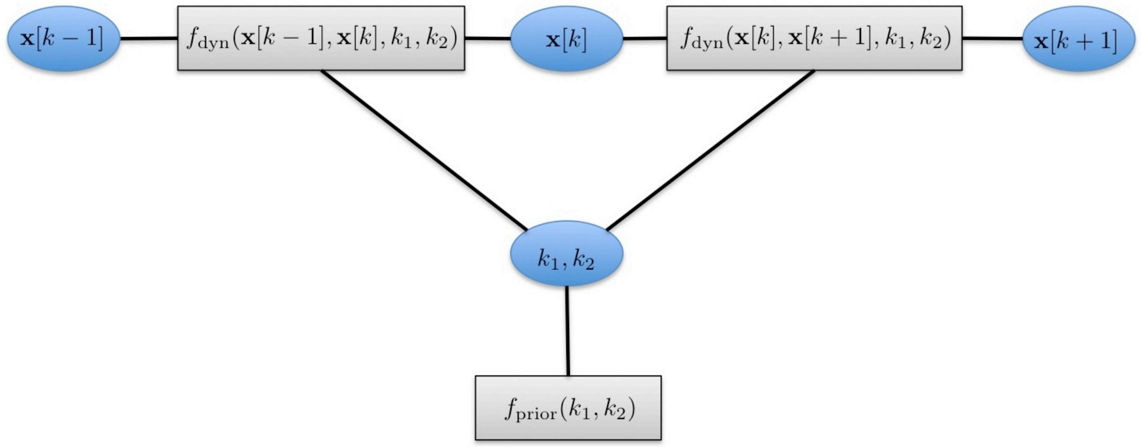

$\mathbf{x}[k-1]$
$f_{\text{dyn}}(\mathbf{x}[k-1], \mathbf{x}[k], k_1, k_2)$
$\mathbf{x}[k]$
$f_{\text{dyn}}(\mathbf{x}[k], \mathbf{x}[k+1], k_1, k_2)$
$\mathbf{x}[k+1]$
k_1, k_2
$f_{\text{prior}}(k_1, k_2)$

Figure 4-4: Factor Graph Process Model with Inertia Ratios

The dynamics model from Equation 4.61 and 4.66 is adjusted using $\mathbf{W}'_v$ and $\mathbf{W}'_\omega$ so that the process noise incorporates the mass and inertia matrix, since these are not observable.

$$\dot{\mathbf{v}} = \frac{1}{m}\mathbf{W}_\mathbf{v} \tag{4.100}$$

$$= \mathbf{W}_\mathbf{v}{}' \tag{4.101}$$

$$\dot{\omega} = -\mathbf{J}^{-1}\omega \times \mathbf{J}\omega + \mathbf{J}^{-1}\mathbf{W}_\omega \tag{4.102}$$

$$= -\mathbf{J}^{-1}\omega \times \mathbf{J}\omega + \mathbf{W}_\omega{}' \tag{4.103}$$

This required an adjustment to the full six degree of freedom dynamic model

(Equation 4.79 and 4.78) as follows:

$$\begin{bmatrix} \dot{\mathbf{r}} \\ \dot{\mathbf{v}} \\ \dot{\mathbf{a}} \\ \dot{\mathbf{w}} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{0} \\ \left(-\frac{1}{2}[\omega\times] + \frac{1}{8}\omega \cdot \mathbf{a}_p\right)\mathbf{a}_p + \left(1 - \frac{1}{16}\mathbf{a}_p^T\mathbf{a}_p\right)\omega \\ -\mathbf{J}^{-1}\omega \times \mathbf{J}\omega \end{bmatrix} + \tag{4.104}$$

$$\begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix} \begin{bmatrix} \mathbf{W_v}' \\ \mathbf{W}_\omega' \end{bmatrix} \tag{4.105}$$

And:

$$\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{B}_W\mathbf{W} \tag{4.106}$$

$$\begin{bmatrix} \dot{\mathbf{r}} \\ \dot{\mathbf{v}} \\ \dot{\mathbf{a}}_p \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\frac{1}{2}[\omega\times] + \frac{1}{8}\omega \cdot \mathbf{a}_p\mathbf{I}_{3\times3} & \left(1 - \frac{1}{16}\mathbf{a}_p^T\mathbf{a}_p\right)\mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathbf{J}^{-1}[\omega\times]\mathbf{J} \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{v} \\ \mathbf{a}_p \\ \omega \end{bmatrix} \tag{4.107}$$

$$+ \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix} \begin{bmatrix} \mathbf{W_v}' \\ \mathbf{W}_\omega' \end{bmatrix} \tag{4.108}$$

4.8 Full Factor Graph Model

Figure 4-5 summarizes the full factor graph model for this approach, and brings together all of the elements of one of the major contributions of this thesis: the development of a stereo visual SLAM algorithm that can handle moving and spinning objects. It illustrates how the full factor graph diagram is constructed to estimate the geometric frame, body frame and ratios of inertia for three timesteps and two features. Note that there is a prior placed on state $\mathbf{x}[0]$ with a very high covariance; therefore, very little a priori information is introduced. This allows for the number of variables to equal the number of constraints and is related to the un-observability of the "kidnapped robot" problem.

The source code for this model is shown in Listings B.20, B.21, B.22, B.23, B.24 and B.25.

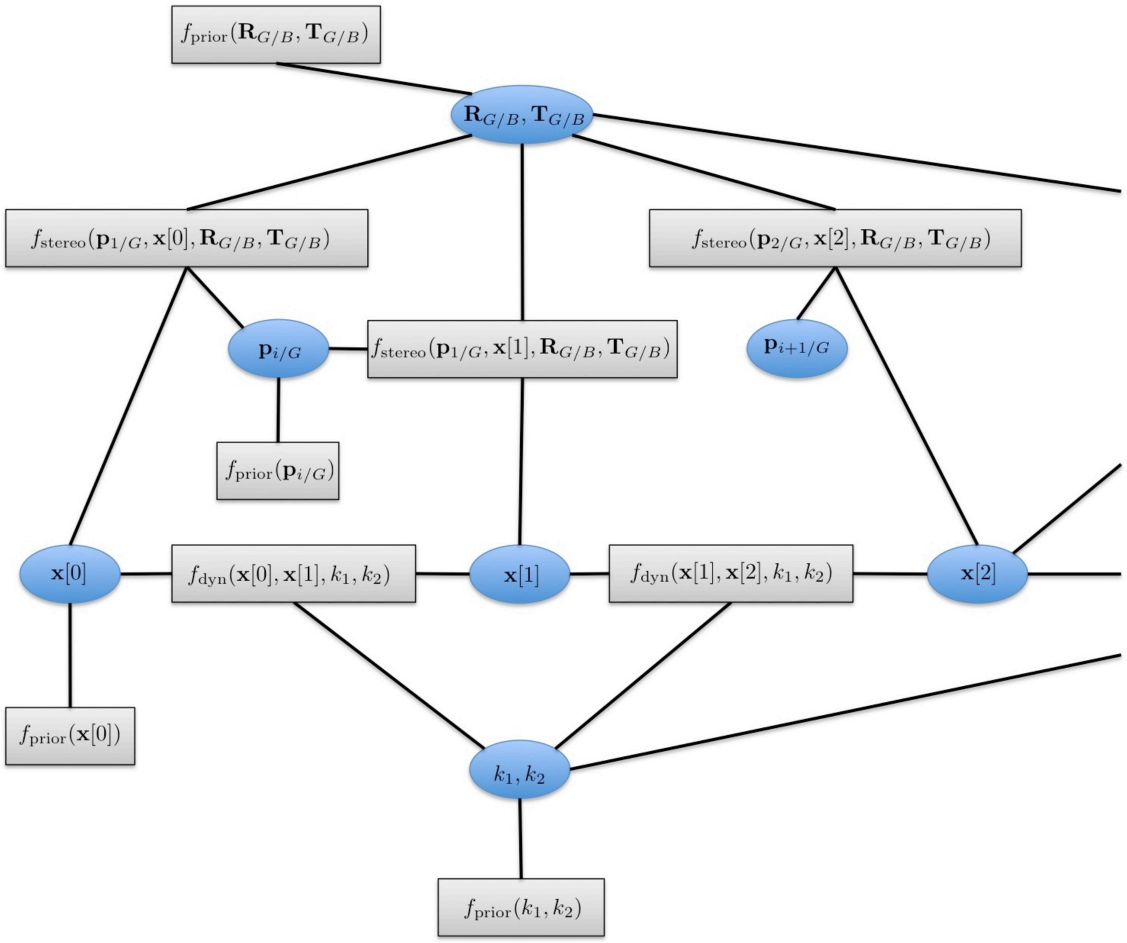

$f_{\text{prior}}(\mathbf{R}_{G/B}, \mathbf{T}_{G/B})$
$\mathbf{R}_{G/B}, \mathbf{T}_{G/B}$
$f_{\text{stereo}}(\mathbf{p}_{1/G}, \mathbf{x}[0], \mathbf{R}_{G/B}, \mathbf{T}_{G/B})$
$f_{\text{stereo}}(\mathbf{p}_{2/G}, \mathbf{x}[2], \mathbf{R}_{G/B}, \mathbf{T}_{G/B})$
$\mathbf{p}_{i/G}$
$f_{\text{stereo}}(\mathbf{p}_{1/G}, \mathbf{x}[1], \mathbf{R}_{G/B}, \mathbf{T}_{G/B})$
$\mathbf{p}_{i+1/G}$
$f_{\text{prior}}(\mathbf{p}_{i/G})$
$\mathbf{x}[0]$
$f_{\text{dyn}}(\mathbf{x}[0], \mathbf{x}[1], k_1, k_2)$
$\mathbf{x}[1]$
$f_{\text{dyn}}(\mathbf{x}[1], \mathbf{x}[2], k_1, k_2)$
$\mathbf{x}[2]$
$f_{\text{prior}}(\mathbf{x}[0])$
k_1, k_2
$f_{\text{prior}}(k_1, k_2)$

Figure 4-5: Full Factor Graph Model for Three Time-Steps

4.9 Implications of Time-Step Selection on Conditioning

The approach described in the previous sections allows for variable timesteps to be used between measurements. This has the potential to improve computational performance by allowing fewer measurements to be made per unit time while propagating their motion for future data association more accurately. However there is one potential issue that limits the extent of the variability of the timestep between measurements.

This issue arises from the fact that iSAM minimizes the Mahalanobis Distance of the factor error (Equation 4.42). As was previously discussed, the optimization is sensitive to the covariance matrix being ill-conditioned. This is due to both the inverse and the Cholesky factorization that is taken when computing the square root Information Matrix, which is required to compute the Mahalanobis distance cost function. If $\mathbf{\Lambda}$ is rank deficient, Equation 4.42 definitely can not be solved, and the pose-graph optimization will fail. If the matrix $\mathbf{\Lambda}$ is full rank, but poorly conditioned, a solution to the system of equations exists, but not all solution methods will find the correct solution[48].

It will be shown here that when double integrators are included in the factors' error function, as previously discussed (i.e. Newton's Second Law or Euler's Equation of Motion), this can lead to ill-conditioning if care is not taken in parameterizing the state space to appropriately match the time step, Δt.

To illustrate this problem, consider the second order system in Equation 4.13. It can be discretized using Equation 4.51 as follows:

$$\begin{bmatrix} r[k+1] \\ v[k+1] \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} r[k] \\ v[k] \end{bmatrix} + w_F[k] \tag{4.109}$$

With the following assumptions:

$$E[w_F(t)] = 0 \tag{4.110}$$

$$E[w_F(t_1)w_F(t_2)] = \begin{cases} \sigma^2, & t_1 = t_2 \\ 0, & t_1 \neq t_2 \end{cases} \tag{4.111}$$

The covariance of $w_F[k]$ can be computed using Equation 4.55:

$$\Lambda = \frac{\sigma^2}{m^2} \begin{bmatrix} \frac{1}{3}\Delta t^3 & \frac{1}{2}\Delta t^2 \\ \frac{1}{2}\Delta t^2 & \Delta t \end{bmatrix} \tag{4.112}$$

As was previously discussed, if Λ is rank deficient or poorly conditioned, any pose graph optimization algorithm will either fail or converge to an incorrect solution respectively. Obviously if $\Delta t = 0$ then Λ looses rank (but this is an impractical case). However, Figure 4-6 shows the condition number of Λ as a function of the timestep Δt. Figure 4-6 clearly illustrates that the covariance becomes more ill-conditiononed as the timestep tends towards very small or very large values.

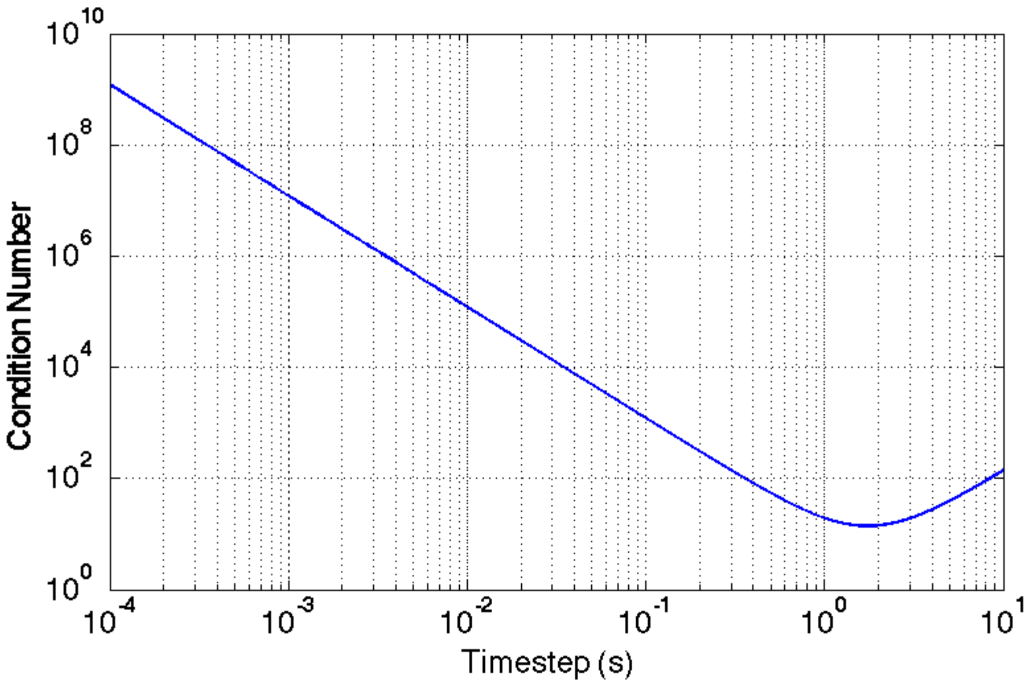


Figure 4-6: Condition Number vs. Timestep Δt in seconds

Figure 4-7 visually illustrates the uncertainty distribution as the time step changes. This shows that at small time steps, there is more uncertainty in the velocity than the position, while at large time steps there is more uncertainty in the position than the velocity.

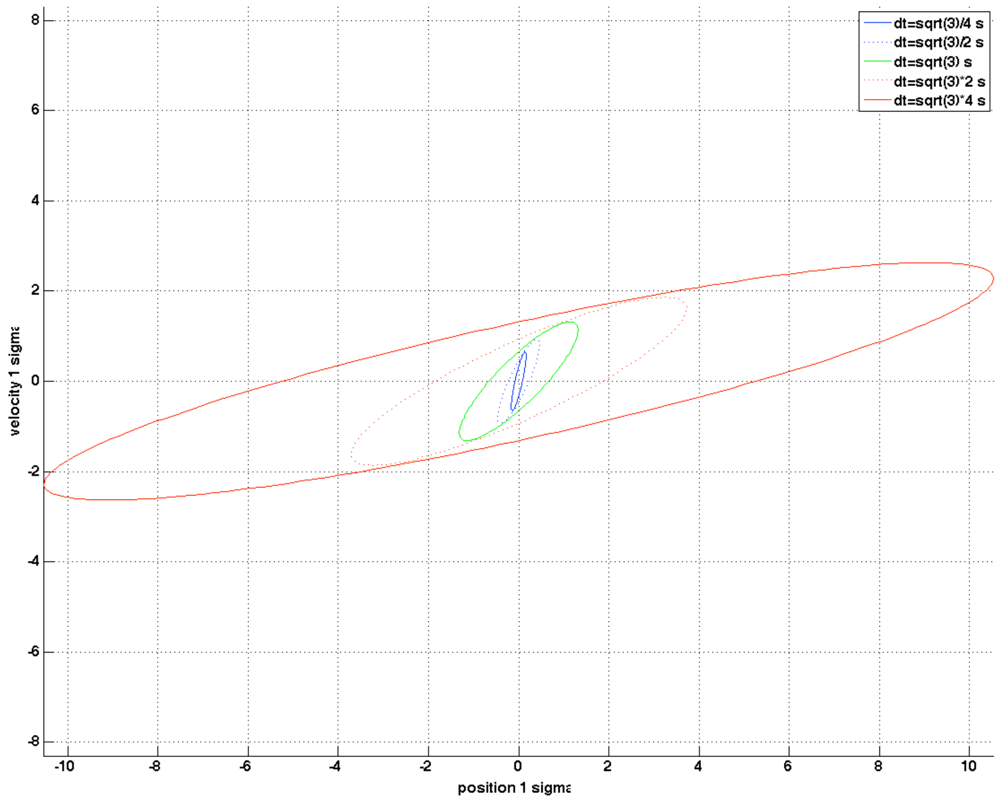


Figure 4-7: Two-Dimensional One-Sigma Uncertainty for Covariance with Differing Time Steps

The physical intuition behind this issue is as follows: Assume that zero mean white Gaussian noise is applied as a force to a mass that is moving in a straight line (i.e. Equation 4.13). Also assume that the initial position and velocity is non-zero, but known perfectly. Over a relatively short time step, the random forces may have an effect on the velocity, but since not much time has elapsed, the position will still be relatively accurately known. This corresponds to the blue circles in Figure 4-7.

Now, when the timestep is relatively long, the random forces will change velocity, which will have relatively more time to propagate into position. In other words, the random forces are passed through a single integrator for velocity, and a double integrator for position. Over a long enough time, the position will be affected relatively more than the velocity due to the fact that a double integrator will "drift" relatively more than a single integrator. This corresponds to the red circles in Figure 4-7.

The minimum point on the curve shown in Figure 4-6 indicates that there is a timestep that is slightly larger than one second, which provides the best conditioning possible. This timestep can be computed by by calculating the condition number as the ratio of eigenvalues of Equation 4.112, and finding the minimum point by differentiating with respect to Δt and solving for the point with the derivative equal to zero. When this is done, the best conditioned timestep is $\Delta t = \sqrt{3}$ seconds. This can be considered the best ratio between the position and velocity uncertainty is obtained when the covariance matrix has its best conditioning. This corresponds to the green circle in Figure 4-7, which has its major axis at exactly a 45^o angle.

This result may initially seem very surprising. It appears to imply that there is something "special" about a time step of $\sqrt{3}$ seconds. In actuality, there is nothing particularly special about this time step. This is because the time step that provides the best conditioning is a function of the way the state variables are defined. For example, changing the units of velocity (e.g. to meters per millisecond) will change the best time step. More generally, the state vector can be redefined using a scaling factor α, to adjust what velocity represents in terms of the position element of the state vector:

$$x_1(t) = r(t) \tag{4.113}$$

$$x_2(t) = \frac{1}{\alpha} v(t) \tag{4.114}$$

This leads to the following continuous time model:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & \alpha \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{\alpha m} \end{bmatrix} w(t) \tag{4.115}$$

Performing a discretization with time step Δt:

$$\begin{bmatrix} x_1[k] \\ x_2[k] \end{bmatrix} = \begin{bmatrix} 1 & \alpha\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1[k-1] \\ x_2[k-1] \end{bmatrix} + w[k] \tag{4.116}$$

$$w[k] \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Lambda\right) \tag{4.117}$$

$$\Lambda = \frac{\sigma^2}{m^2}\begin{bmatrix} \frac{1}{3}\Delta t^3 & \frac{1}{2}\frac{1}{\alpha}\Delta t^2 \\ \frac{1}{2}\frac{1}{\alpha}\Delta t^2 & \frac{1}{\alpha^2}\Delta t \end{bmatrix} \tag{4.118}$$

Now the condition number of this new covariance matrix is shown in Figure 4-8 for three different values of α.

An intuitive explanation for this behaviour can be seen using the following example: Consider a state estimation system for a commerical automobile. If the units of a position and velocity state estimate are meters and millimeters per day respectively (i.e. $\alpha = 10^{-9}$), the conditioning at a time step of 1 second would be very poor, because the numerical value of the uncertainty in velocity would be much larger than the position uncertainty. However, if a timestep of 11 nano-seconds (the best conditioning point) were used the numerical values of uncertainty in position and velocity would be similar.

From Figure 4-8, it is evident that the parameterization can be adjusted to better condition a particularly desirable time-step. In this example, the equation $\Delta t = \alpha\sqrt{3}$, can be used to solve for α for a given Δt. Using the car example, if a time-step of 1 hour and position units of meters was required ($\alpha = 2078$), the velocity should be in units of approximately kilometers per second since over the period of an hour, the uncertainty on kilometers per second should be numerically similar to the uncertainty in meters of position.

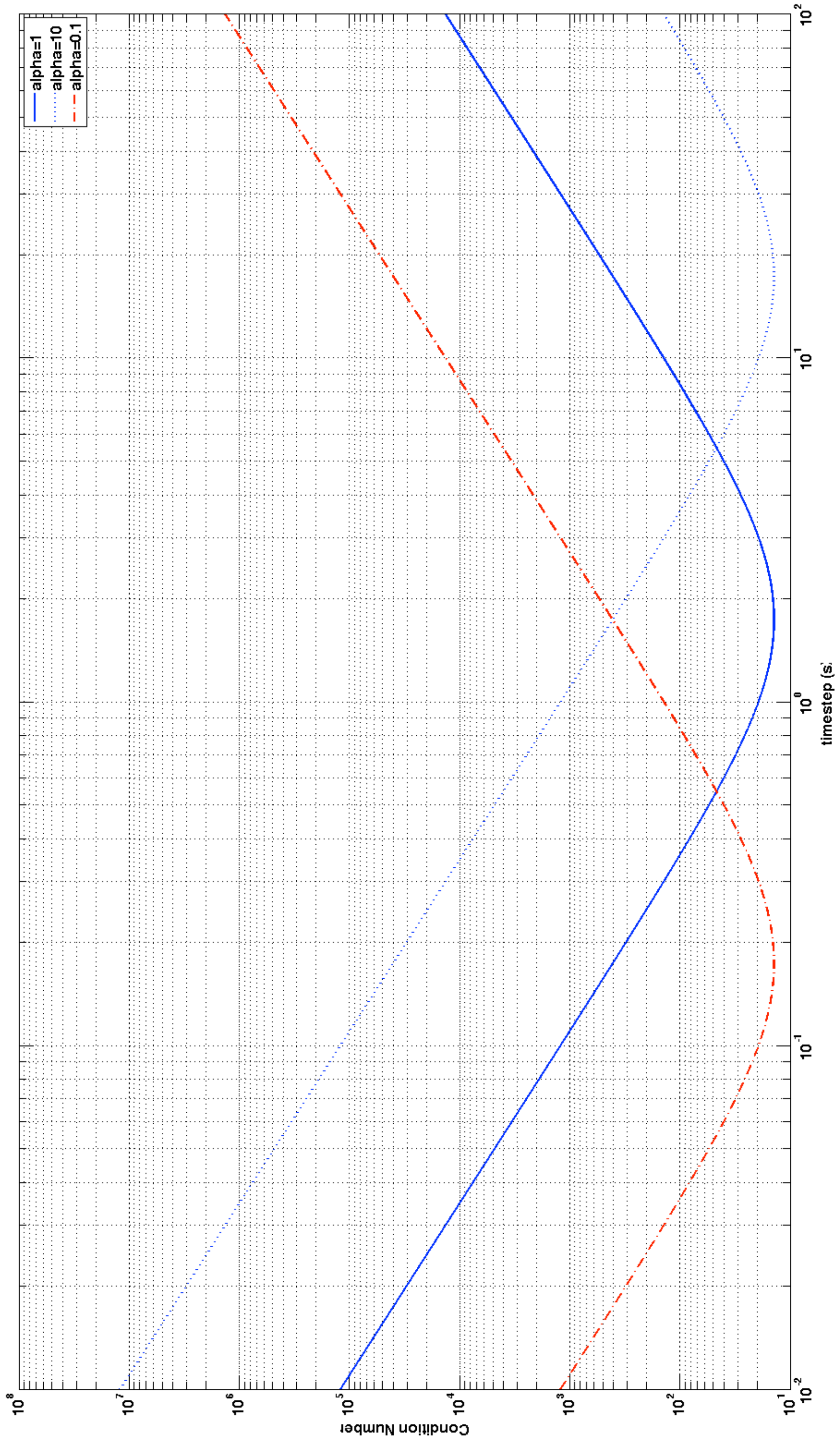


Figure 4-8: Condition Number vs Timestep for Different Values of α

Chapter 5

Systems Design of the SPHERES Goggles

The main issue for any Earth-based spacecraft proximity operations testbed is that it would be very hard to perform high speed spinning and nutating maneuvers. The first reason is that most of these systems are limited in their range of motions; therefore, an unstable spin about an intermediate axis of inertia would be very challenging to simulate. Also, it is very difficult to avoid the mechanical friction caused by Earth's gravity in any of these types of testbeds, whether it be from motors and gears or fluids. Additionally, if a testbed is based on a spherical air bearing that is not precisely balanced, additional precession modes caused by external torques will be visible at high spin rates. The best method to test spinning and nutating spacecraft is onboard the International Space Station [23]. The only testbed that can accurately reproduce and measure 6DOF spinning and nutating motion between multiple spacecraft is the SPHERES satellites. Prior to February 2013, the SPHERES satellites did not have the capability to perform vision-based navigation in a microgravity environment. One of the main contributions of this thesis is the design, build, test and operations of a stereo-vision based navigation testbed for a micro-gravity environment and is described in this chapter.

In order to test the previously mentioned algorithmic contributions, a vision-based navigation upgrade to the SPHERES satellite is needed for testing in the microgravity environment of the ISS. A "flight-ready" set of Goggles was developed in the author's Master's thesis [136, 135] (referred to as the LIIVe Goggles). When compared against Saenz-Otero's seven Microgravity Laboratory Design Principles[113], the LIIVE Goggles meet the Principle of Iterative Research, the Principle of Enabling a Field of Study, the Principle of Optimized Utilization, the Principle of Focused Modularity, the Principle of Incremental Technology Maturation and the Principle of Requirements Balance. The one principle they do not meet is the Principle of Remote Operation and Usability.

An essential element of this contribution is that this testbed is an "open research facility." The openness of this testbed allows other researchers to easily develop new software and hardware based experiments to perform on this testbed. This is an important element of the Goggles primary objective and drives a number of the requirements of the design.

5.1 SPHERES Satellite Overview

The Synchronized Position Hold Engage Reorient Experimental Satellites (SPHERES [93, 102, 113], shown in Figure 5-1) are a set of volleyball sized micro-satellites that have been operating inside the International Space Station (ISS) since 2006. The SPHERES satellites are considered a research and development testbed for guidance, navigation and control algorithms for formation flying satellites. In the past 6 years, MIT has held 36 test sessions onboard the ISS covering research topics such as rendezvous and docking, formation flight, decentralized control, satellite reconfiguration, inertial navigation and other research areas. Additionally, the SPHERES satellites have been used for national and international STEM programming competitions where middle and high school students have programmed the SPHERES satellites that are on-orbit[114].

The SPHERES satellites are designed as a small satellite bus. Some of the detailed features are shown in Figure 5-2. They weigh 4.16 kg and are 21.3 cm in diameter. They utilize a carbon dioxide (CO2) cold gas propulsion system to produce both forces

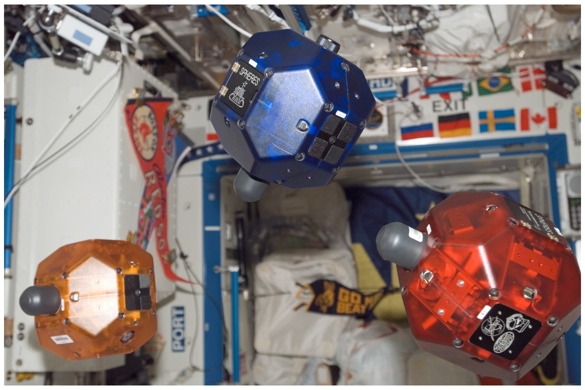


Figure 5-1: The SPHERES Satellites inside the ISS

and torques. A “pseudo-GPS” ultrasonic time-of-flight sensing system is used with onboard gyroscopes to estimate the position, orientation, linear and angular velocity with respect to the interior of the ISS. A Texas Instruments C6701 Digital Signal Processor is used to perform onboard computations and a 900 MHz low bandwidth modem is used for communication with the laptop that is used by the ISS astronaut. The entire SPHERES satellite is powered by 16 AA non-rechargeable batteries.

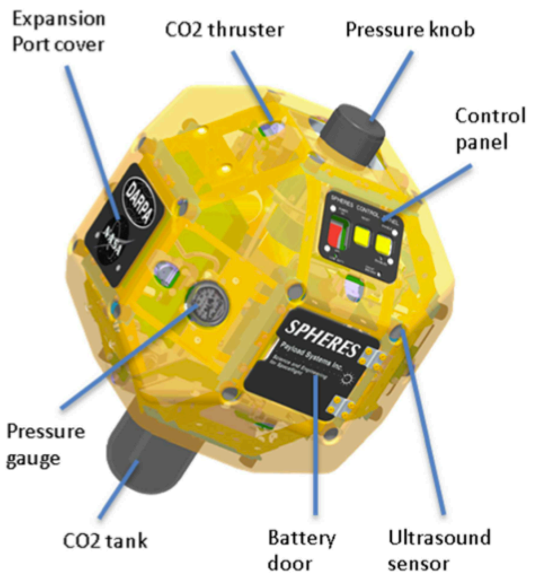


Figure 5-2: Details of the SPHERES Satellite

5.2 Goggles Mission Objective, Systems Design Constraints, Budget and Schedule

The principal mission objective of the Goggles system is defined below:

Mission Objective: *The Goggles must enable the representative, experimental testing and evaluation of new algorithms and approaches for computer vision-based navigation for spacecraft proximity operations using the SPHERES satellites and the microgravity environment of the International Space Station. The SPHERES Goggles system must abide by Saenz-Otero's seven Microgravity Laboratory Design Principles[113]. Additionally, this system must become an "open research facility" and therefore provide an open and expandable interface for incorporation with follow-on software and hardware that is developed in the future for research in other fields of study.*

Note that while the objective places significant priority on developing a testbed for spacecraft proximity operations, there is an explicit constraint on utilizing the SPHERES satellites that were already operating on-orbit. Using the SPHERES satellite as a platform to be upgraded provides a number of advantages and disadvantages. The main advantage is that it provides a actuation platform and reference sensor system that is available for use immediately and with no required development time. This was the principal motivating factor for selecting the SPHERES satellites as a starting point for the Goggles system design.

The biggest disadvantage for using the SPHERES satellites inside the ISS is that the appearance of the interior of the ISS cannot exactly replicate the types of lighting conditions, materials and distances that will likely be encountered on-orbit in a "real" spacecraft proximity operations mission. Despite this, the Goggles system design must be able to make a reasonable and representative approximation for the types of sensor input that would be gathered by a "real" proximity operations mission.

Another disadvantage is that the design of the SPHERES satellites cannot be changed and may not be the ideal match for any possible new design. Additionally, since the SPHERES satellites are already operated by astronauts onboard the ISS, the design of the Goggles must match as closely as possible to the existing operational model.

The development of the SPHERES Goggles was initially funded by the Naval Research Laboratory as a ground prototype, the LIIVe Goggles [135, 136], between 2008 and 2009. In 2010, the Defense Advanced Research Projects Agency (DARPA) funded the development of a flight version of the Goggles under the International Space Station SPHERES Integrated Research Experiments (InSPIRE). The MIT Space Systems Laboratory with their industry partner Aurora Flight Sciences received $1 Million to develop the Goggles into a flight version. This was known as the Visual Estimation and Relative Tracking for Inspection of Generic Objects (VERTIGO) Program, which had 8 months from contract start (January 2011) to the Critical Design/Testbed Review (September 2011). This was followed by 8 months from Critical Design/Testbed Review to software delivery (May 2012) and 11 months from Critical Design/Testbed Review to flight hardware delivery (August 2012). The hardware was launched to the ISS in October 2012 and was first operated in February 2013.

In addition to the design and development of the hardware and software, a significant amount of effort was devoted to ensuring compliance with the environmental and safety requirements of the International Space Station. This placed constraints on the types of materials and manufacturing processes that could be used. Additionally, there was a safety testing process and series of reviews to ensure that the hardware does not pose a safety risk to the crew. The Goggles were also required to undergo electro-magnetic interference testing (EMI), off-gass and vibration testing. Additionally, human factors and usability requirements were placed on the design of the hardware and software.

Given the limitations of this schedule and budget, the overall design approach was to use as much commercial off the shelf (COTS) hardware and software as possible. The in-house design and build of a number of the electrical, mechanical hardware and software components was required to properly interface many of the COTS subsystem components. One common question is why not use the manufacturing techniques that are now common-place in the smartphone industry to reduce the mass and volume of the Goggles, which are relatively large for a similar set of capabilities. Most modern smartphones use very expensive electro-mechanical manufacturing techniques, such as Flexible Printed Circuits (FPC) and 10+ layer Printed Circuit Boards (PCB). These are prohibitively cost expensive in low volumes and very difficult to debug if incorrectly manufactured.

The development of the requirements of these components and high level design for the “open research testbed” is detailed in this chapter as a contribution of this thesis. The detailed design of the LIIVe Goggles prototype was presented in the author’s master’s thesis [135]. This prototype design was used as a starting point for Aurora Flight Sciences who lead the fabrication and testing of the VERTIGO Goggles.

5.3 Optical Requirements and Design

The main objective of the Goggles is to add a new type of sensor to the SPHERES satellites that is representative and appropriate for evaluating computer vision-based navigation techniques. In the design of a spacecraft proximity operations mission there are a number of types of “computer vision” sensors that may be used. Visible wavelength cameras are the most obvious option, however in on-orbit lighting conditions there will be sensitivity to a number of specular reflections that commonly occur with reflective materials such as multi-layer insulation (MLI). Additionally, visible wavelength cameras will capture images that are high in contrast due to solar illumination.

An alternative type of sensor is a long wavelength infrared camera that detects the black body radiation of an object. Since this radiation is relatively stable with changing lighting conditions, it can reduce the specular reflections that typically cause problems for feature detection and matching algorithms. Figure 5-3 shows the visible and infrared images taken by SpaceX’s DragonEye. Figure 5-4 shows a number of images of the ISS taken by Neptec’s infrared camera. The bottom left image was taken when the ISS was in the night-time shadow of the earth, all of the other images were taken during the day-time.

Both of the two types of cameras that have been discussed so far can only measure bearing angles (i.e. two dimensional pixel locations). The third dimension is depth, which is not detectable with a monocular camera. The first option for measuring range or depth is to use two cameras in a stereo configuration. If relative position and orientation between the cameras (i.e. the baseline in a calibrated camera system) is known, the depth can be triangulated using the methods in Section 2.7. It is important to note that the accuracy of the triangulation improves as the baseline increases, but causes objects that are close in to be observed only in one of the two cameras. As a result, baseline selection in stereo cameras is an important design parameter.

An alternative method for detecting range is to use a flash Light Detection and

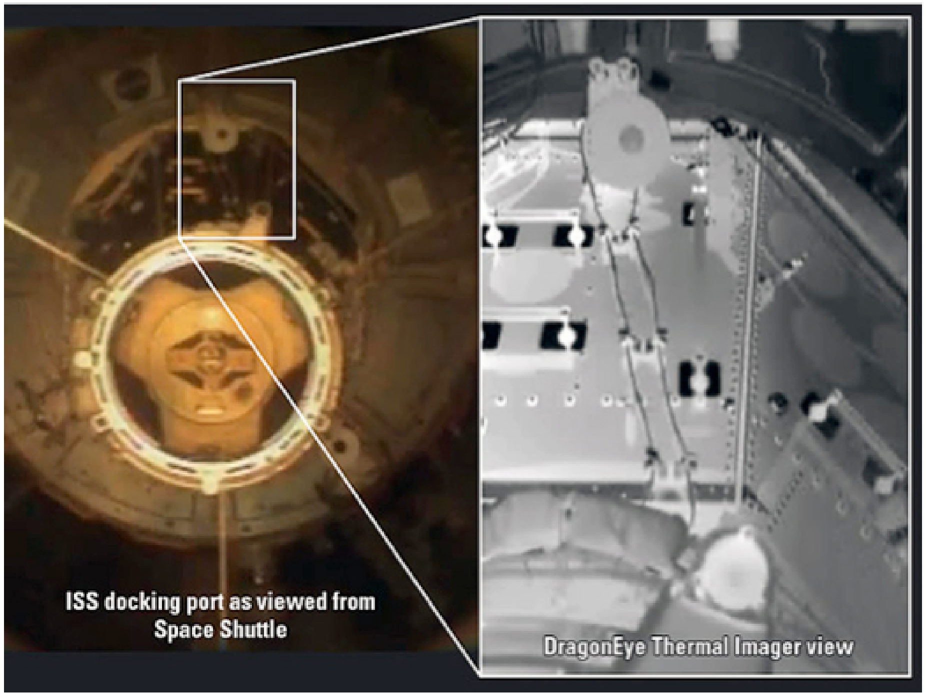

ISS docking port as viewed from Space Shuttle
DragonEye Thermal Imager view

Figure 5-3: Images from DragonEye Visible Wavelength Sensor (left) and Long Wavelength Infrared [124]

Ranging (Lidar), which illuminates a target with laser and measures the time of flight to detect range and the reflectance intensity to create a greyscale map. This method is robust to varying lighting conditions, but the return intensity can be affected by specular reflections. Figure 5-5 shows results from Neptec's TriDAR imaging the ISS (red colors are closer).

Lastly, systems that are based on projecting known patterns onto the scene (i.e. structured light similar to the Microsoft Kinect) are often considered for ground based applications, but are not applicable to on-orbit lighting conditions.

Since the SPHERES Goggles should be able to perform representative experiments for any of the above types of sensors, it should be able to collect range and intensity images. This means that either a lidar or stereo camera system should be used. A stereo camera system was selected for the Goggles since it will have lower mass and power consumption, while costing significantly less.

The disadvantage of choosing a stereo camera system is that it may be sensitive to lighting conditions and specular reflections and will require sufficient texture to

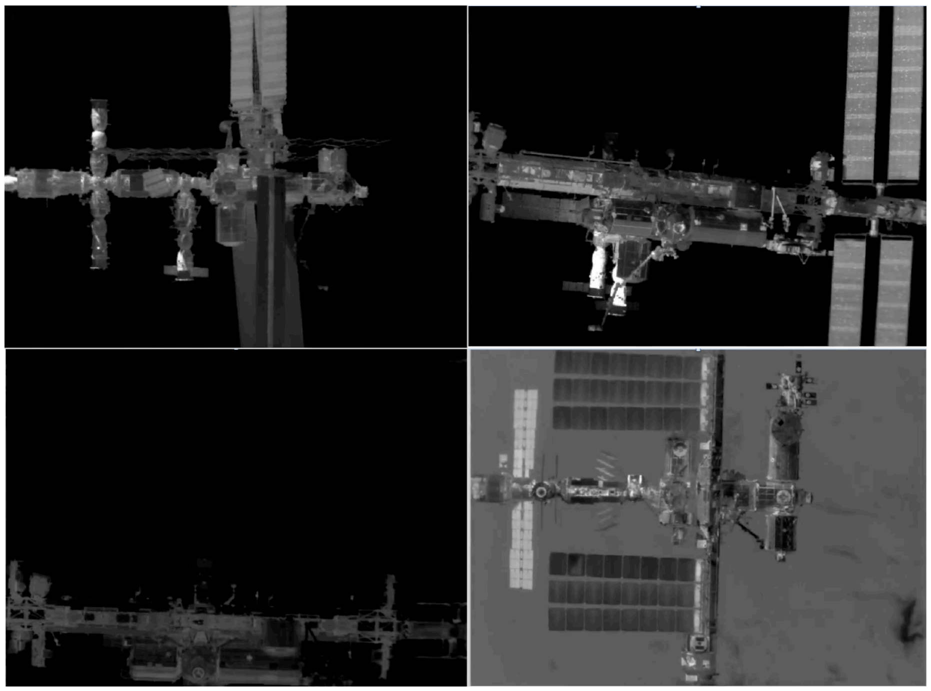


Figure 5-4: Images from Neptec Infrared Camera [79]

perform proper feature tracking and matching. For these reasons it should be realized the SPHERES VERTIGO Goggles will not be a photo-realistic representation of on-orbit imaging systems. If this type of photo-realism is required, alternative testbeds should be investigated [27, 43, 103].

However, the VERTIGO Goggles is a unique testbed to gather experimental data that is exactly representative of the 6DOF dynamics that occur in a microgravity environment, and can easily gather imaging data of objects that have complicated spinning motions.

When building a stereo imaging system for photographing an object that is moving or spinning with high speed, care must be taken to ensure that there is no motion blur and that all of the pixels are captured at the exact same time in both cameras. This leads to the requirement that the cameras must have a global electronic shutter (i.e. all the pixels are simultaneously exposed) and have a method for synchronization between the two cameras.

Additionally, it is important to ensure that the exposure time of the camera can be controlled, in software, and that there will be sufficient light gathered by the imager.

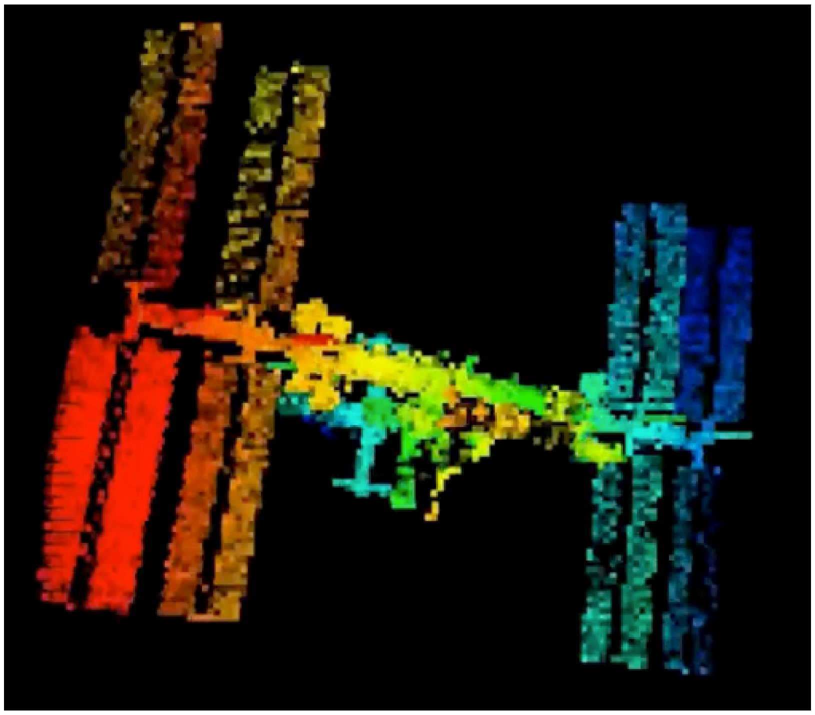


Figure 5-5: Images from Neptec TriDAR [79]

This means that the camera sensor and lens size must be sufficiently large for the lighting conditions. Given that the lighting conditions were somewhat unknown and variable onboard the ISS, an additional requirement was added to include onboard "flash" lights that will illuminate the environment.

One approach to designing the optics system could have involved a detailed mathematical analysis. However, given that the optics would be assembled entirely from COTS parts, a more emperical approach was taken. The experience of developing the LIIVe goggles provided a good starting point for a trial and error evaluation of different lenses, cameras, stereo baselines and illuminating lights.

The results of this evaluation led to the selection of the components listed in Table 5.1.

With the above components, a number of theoretical properties can be calculated. The sensor on the imager is 4.51 mm wide by 2.88 mm high, with a diagonal of 5.4 mm. Note that when only 640 pixels are used for width the sensor imager is 3.84 mm. The field of view of the system is computed using the formula below for the angle of view α, where d_{sensor} is the sensor size and f is the focal length. Using this

Table 5.1: Optics Component Specifications

Cameras	IDS-Imaging uEye LE 1225-M-HQ
Camera Sensor	1/3” Monochrome (10 bits per pixel) CMOS with Global Shutter, HQ Filter
Camera Configuration	9.0cm Stereo baseline, HW exposure timer and sync
Camera Resolution	752 x 480 pixels at 6 μm per square pixel
Camera Power Consumption	5V, 100-130 mA
Lens Mount	CS-Mount
Lens Type	Fujinon 2.8mm, f/1.3 (CCTV Lens for 1/3“ and 1/4“ Imager)
Frame Rate	87 FPS (Camera Max), 10 FPS (Typical)
Exposure	80 μ s - 5.5 s
Lights	2× Phillips Rebel Star LED Red-Orange
Lights Dominant Wavelength	617 nm
Lights Intensity	134 lm @ 700mA (per LED)

equation the horizontal angle of view is $\alpha_{\text{horz}} = 68.9^o$, while the vertical angle of view is $\alpha_{\text{vert}} = 54.4^o$. This is a fairly wide field of view that allows a large range of motion of an object to be visible in frame.

$$\alpha = 2\tan^{-1}\left(\frac{d}{2f}\right) \tag{5.1}$$

The distance to a stereo object based on the angle can be found with Equation 5.2. If α is set to the maximum angle of view for the lens, α_{horz}, the minimum distance to an object that can be seen in both cameras is found to be $6.56cm$. If α is set to the angle of one pixel (at the optical center), then $\alpha_{1 \text{ pix}} = \tan^{-1}(\frac{6\times E-6}{f}) = 0.123^o$ and the maximum distance is 42.0 meters.

$$d_{\text{stereo}} = \frac{b/2}{\tan(\alpha/2)} \tag{5.2}$$

It is useful to determine the expected accuracy of triangulation. An analytical calculation was performed using Equation 2.7 for the baseline and focal length listed in Table 5.1, with an object located 0.5 meters away (the nominal distance for the experimental results in Chapter 6) that is directly centered between the two cameras.

An error of 1 pixel in the horizontal location of either image leads to a change in the triangulated location of $[0.5, 0.0, -5.9]$ millimeters. An error of 1 pixel in the vertical location of the feature leads to a change in the triangulated location of $[0.0, 1.1, 0.0]$ millimeters.

The last major requirement is based on the fact that the intrinsic and extrinsic parameters described in Sections 2.6 and 2.7, must be known with very high accuracy in order to properly triangulate points. These parameters are estimated through the camera calibration procedure described in Section 2.8. If the lenses or imagers ever change their position relative to the rest of the optics setup the extrinsics may need to be recalculated. This may occur if the system is subjected to vibration, impact shock, temperature changes or other reasons. Given that re-calibration is a procedure that involves a significant amount of crew time, the optics system should be designed to minimize the frequency of required re-calibrations. Since the optics assembly is made of many connected COTS parts, a large aluminum shell with mechanical supports for the lenses was designed to ensure that recalibration was needed as infrequently as possible.

Figure 5-6 illustrates the key components of the Goggles Optics Mount, and Figure 5-7 illustrates its assembly.

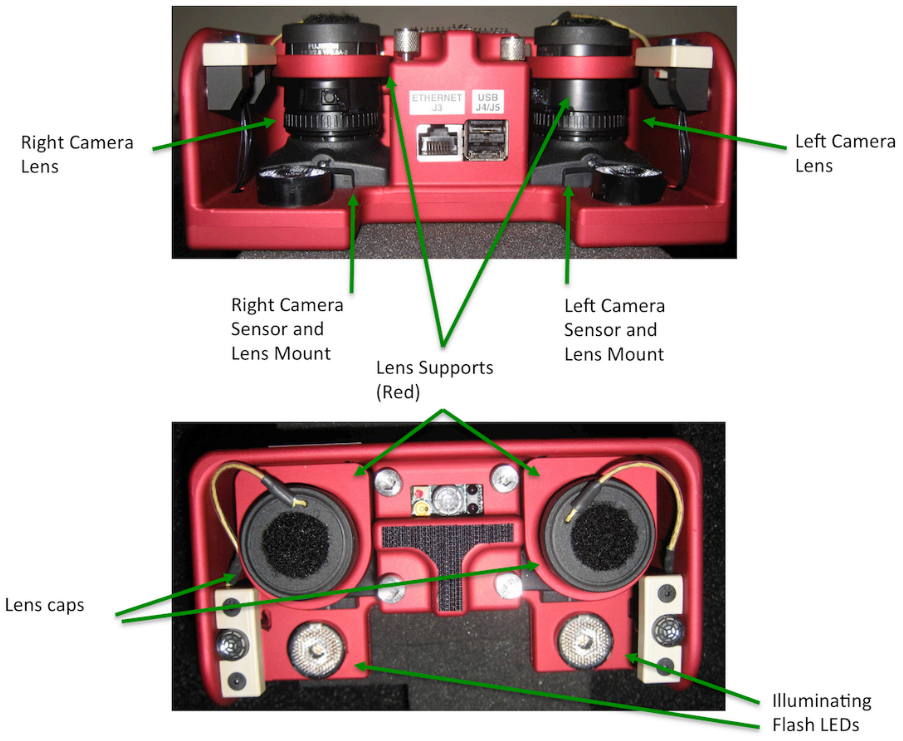


Figure 5-6: Goggles Optics Mount

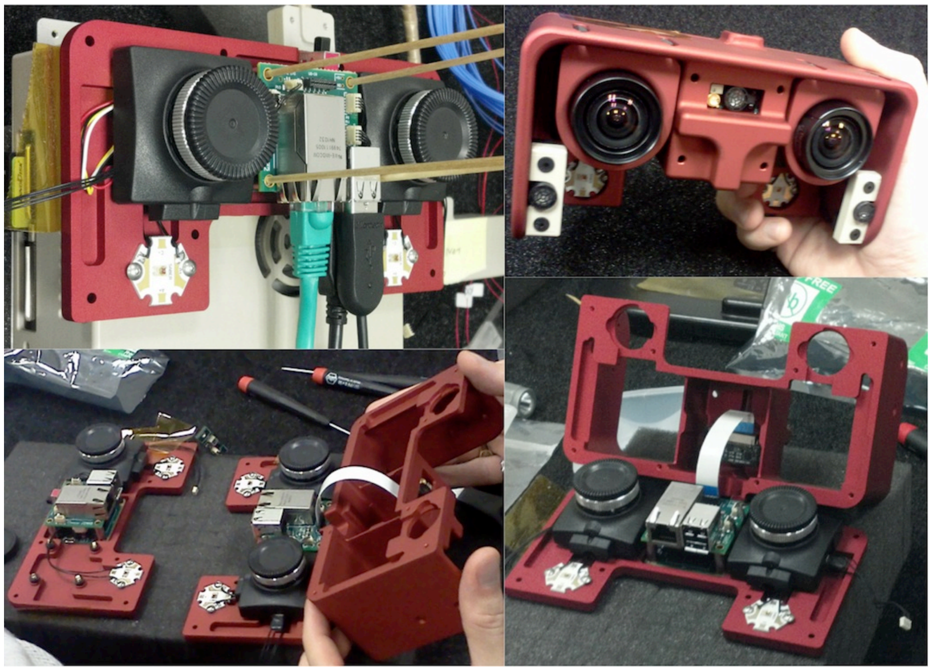


Figure 5-7: Assembly of Goggles Optics Mount

5.4 Computer and Software Requirements and Design

Typical spacecraft avionics systems often include radiation hardening or tolerance and real time operating systems. Since the SPHERES Goggles are operated inside the crew volume of the ISS, radiation hardening is not required. Additionally, since the Goggles are not a mission critical system, and can be reset by a crew member if an error occurs, real time requirements are not intrinsically required.

Instead, the primary objective of the Goggles is to be an experimental research testbed for new vision-based navigation algorithms. This implies the following requirement: Implementing new algorithms for guidance, navigation and control or any other function, should be as quick and easy as possible.

This is an important and fundamentally defining requirement, because software programming and testing for embedded systems can often be a challenging and time-consuming endeavor. Typically the process involves the mixing and matching of programming interfaces, device drivers, library dependencies and compiler optimizations to get the best possible functionality and performance. Although this type of software engineering effort is usually time consuming and requires very specific expertise, it is usually secondary and independent of the fundamental research questions that are to be evaluated by the micro-gravity experiment. The final software implementations are often hardware dependent and do not carry over between multiple flight hardware or programs.

Therefore, it is important to minimize the time involved in the embedded system implementation details by maximizing software reuse. In other words, the computer and software architecture of the Goggles should support as many existing libraries as possible and require as little porting effort as possible.

This requirement leads to the design decision that the computer hardware and software system should be as similar as possible to the most common computer architectures that are used by robotics researchers. Computer vision and robotics libraries such as OpenCV, Eigen, iSAM, PCL and others, typically offer the best and most up to date support for x86 Linux systems. OpenCV is an industry standard image processing library, which in 2010 only offered hardware acceleration for processors that supported the SSE set of SIMD instructions, which are only available on x86 based processors. Since the speedup of these functions is so significant, and these functions are so critical to vision-based navigation for spacecraft, it was decided to make x86 and SSE acceleration a requirement.

At the time the Goggles design was finalized, the most popular linux distribution for robotics research was 32bit Ubuntu 10.04, Lucid Lynx. For the reasons mentioned above, it became a requirement that the Goggles use a version of the Lucid Lynx Ubuntu operating system.

The next major requirement was based on the power requirements described in Section 4 and required battery lifetime. The Goggles embedded computer should consume 15 watts of electricity or less, averaged over typical ISS operational scenarios. Table 5.2 shows the two primary options for the Goggles computer, an Intel Atom and a Via Nano. While the Atom has a higher clock speed, it has significantly lower real-world performance on computer vision tasks due to its In-Order execution and smaller L2 cache. For these performance reasons the Via Nano was selected for the Goggles.

Table 5.2: Processor Comparison

Feature	Intel Atom (N455)	Via Nano (U3300)
Clock Frequency	1.66 GHz	1.2 GHz
Execution	In-Order	Out-of-Order
L2 Cache	512 kB	1 MB
Electrical Power (Watts TDP)	6.5 W	6.8 W
SIMD Instructions	SSE1 to SSE3	SSE1 to SSE3

At the time the Goggles was developed, the smallest available form factor for an embedded single board computer that hosted the Via Nano U3300 was the Via Pico-ITX P830. This board is 10 cm by 7.2 cm and provides up to 4GB of DDR3 RAM, 2 SATA connectors, 1 Gbps Ethernet, USB 2.0, two TTL UARTs, an SPI bus, and a number of other standard features (i.e. PS2 keyboard, mouse and VGA). At the time the Goggles were designed, the largest capacity SATA FLASH Disk-on-Modules were 64GB. Two of these were selected and used in each of the Goggles. Note that the hard drives are partitioned (using ext4) and mounted to three different locations: The first 16GB partition is mapped to the "/" directory and is the primary boot partition with all of the operating system functions. The second partition is the 48 GB (minus swap space) that immediately follows the first partition and is mapped to the "/home2" directory. The third partition is the entire 64GB of the second flash drive and is mapped to the "/home" directory. This approach was selected to separate the operating system software from the goggles test program software and data.

5.5 Operational Requirements and Design

The VERTIGO Goggles are required to be operated by astronauts onboard the International Space Station. When designing an experiment to be operated by crew members, it is important to recognize that they are not necessarily experts in the area research being performed. This is simply due to the diversity of their backgrounds (e.g. pilots, medical doctors, engineers, scientists and teachers), the number and variety of tasks they are expected to perform and the scheduling constraints prior to operations, which limits the time they can spend learning about any one experiment. This is discussed in detail by Saenz-Otero's "Principal of Remote Operation & Usability"[113].

Additionally, the mission objective of the VERTIGO Goggles includes the broad requirement that the system must be able to evaluate any type of vision-based navigation algorithm. In order to achieve this, a significant amount of flexibility in software architecture is required. The previous method of operating the SPHERES satellites is that new software is developed prior to each test session and uploaded to the ISS a few weeks prior to operations. Since this type of operation provides the required flexibility and was already approved, it was mirrored in the Goggles approach.

The astronauts use a laptop Graphical User Interface (GUI) that allows them to upload new code to SPHERES, start and stop tests and save data that will be downloaded to the Earth. It is again desirable to continue this type of operation due to familiarity and the fact that it is already approved by NASA. Therefore, all of the required software interactions that are required by the VERTIGO Goggles must be performed through a modification to the SPHERES GUI. This is possible through the "software plug-in" interface that the SPHERES GUI provides.

In order to keep the Goggles operational system as simple as possible, there should be a method to run research software on the Goggles whenever a SPHERES test is run. In order to simplify operations, this should not require any additional interaction by the crew member. In other words, the software should be set up to run a test on both the SPHERES and the Goggles hardware when it is required. However, there are situations where it is required to run code on the Goggles without running code on the SPHERES satellites. This type of capability enables the required level of flexibility, so that a variety of "software maintenance" can be performed.

These requirements lead to the design of two types of software that must be run on the Goggles: "Tests" and "Maintenance Scripts". In order to maintain the most software flexibility, both of these are implemented as a single Linux Bash command that does not require user input. Tests are run whenever a SPHERES test is started, while Maintenance Scripts are preprogrammed on the ground and run or executed by the astronaut from within the upgraded GUI.

Astronauts may also interact directly with the hardware, however this should be limited as much as possible, and should only be required during set-up, clean-up, consumables (i.e. battery changing) and checking status indicators. A simple set of indicators and switches should be easily visible and intrinsically understandable for crew members.

To summarize, the following operations are basic nominal requirements that must be easily performed by an astronaut:

- Attachment and Removal of Goggles from SPHERES Satellites
- Powering On and Shutting Down the Goggles and SPHERES Satellites

- Uploading New Test and Maintenance Software to the Goggles
- Running a Test with code on both SPHERES and Goggles
- Running a Maintenance Script on the Goggles
- Viewing Live Video feeds from the Goggles
- Checking Goggles Battery Charge and Battery Changing
- Download data from Goggles and SPHERES satellites to be stored on the ELC (for later download to Earth)

It is good design practice to include several off-nominal situations that can be handled during operations. This is especially true given the level of software complexity of the Goggles.

- Determine Goggles Status
- Reset Goggles CPU
- Attach a Virtual Keyboard, Mouse and Monitor to the Goggles Onboard Computer for debugging.

These requirements lead to an operational architecture as shown in Figure 5-8. This design includes test program research code that runs on both the SPHERES and the VERTIGO Goggles as discussed, as well as a SPHERES-VERTIGO GUI that runs on the ELC.

Additionally, the Goggles Daemon is a software program that is always running and managing all operational aspects of the Goggles. The Daemon primarily communicates with the GUI on the ELC and executes the instructions that it has been sent. The Daemon monitors and logs the regulated and unregulated battery votages, the battery current, the temperature at a number of physical locations, if the Goggles is attached to a SPHERES satellite and the flash disk status. The Goggles Daemon handles the receiving and unpacking of new code, known as a Goggles Program File or (GPF). It will execute any “run test” or “run script” commands that are received.

The Daemon will perform a soft/safe shutdown of the Goggles if it receives a message requesting a shutdown, if the temperature rises above a threshold, or if the voltage drops below a threshold.

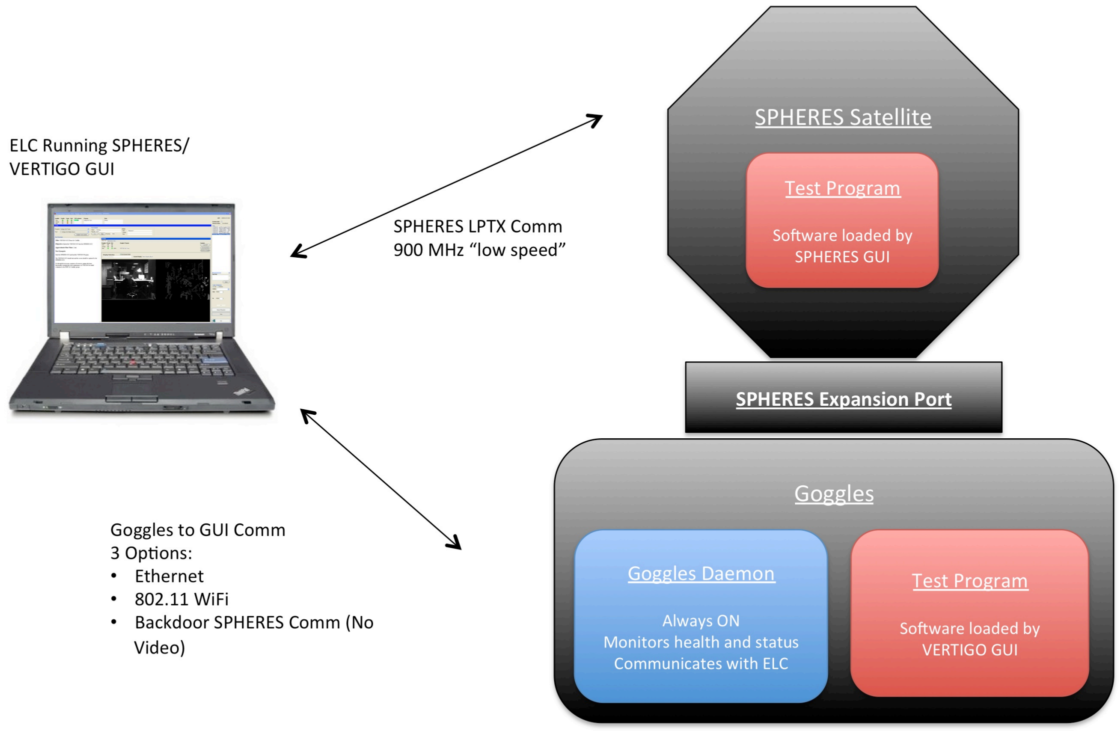
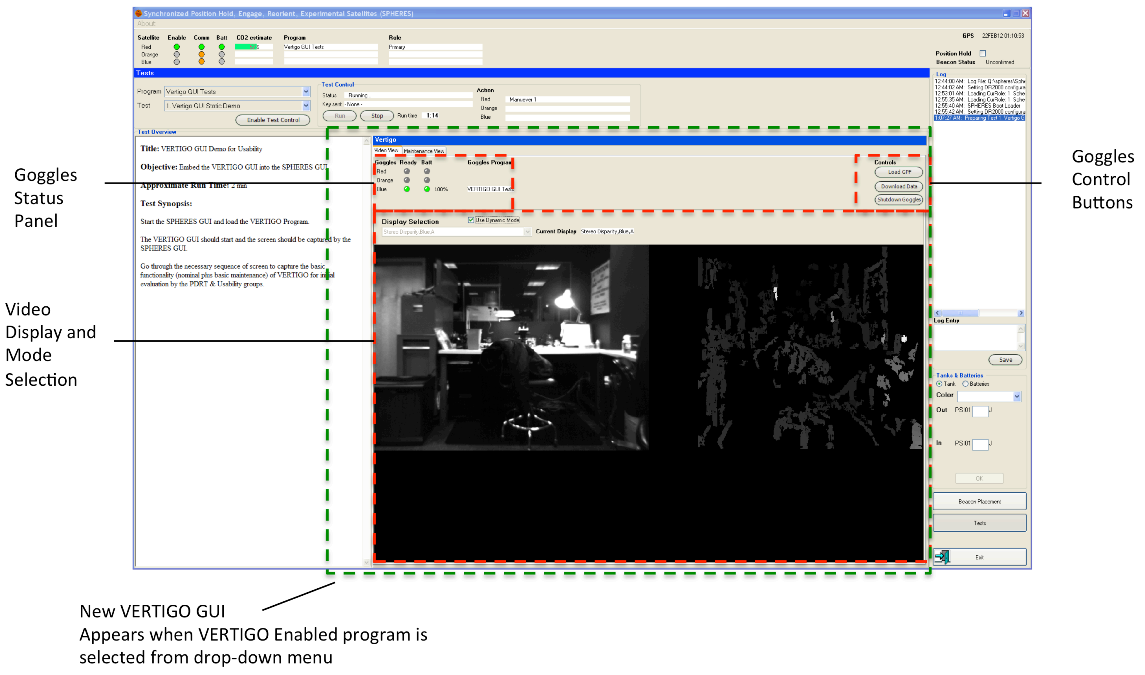

ELC Running SPHERES/ VERTIGO GUI
SPHERES LPTX Comm 900 MHz "low speed"
SPHERES Satellite
Test Program
Software loaded by SPHERES GUI
SPHERES Expansion Port
Goggles
Goggles Daemon
Always ON
Monitors health and status
Communicates with ELC
Test Program
Software loaded by VERTIGO GUI
Goggles to GUI Comm
3 Options:
• Ethernet
• 802.11 WiFi
• Backdoor SPHERES Comm (No Video)

Figure 5-8: SPHERES Goggles Operational System Architecture

Note that Figure 5-8 illustrates three possible modes for the Goggles to communicate with the ELC. The first is through an ethernet connection between the ELC and Goggles. The second is an 802.11n wireless connection, using the wireless card onboard the Goggles. Both of these options are high-speed data connections. However, it is not desirable to have the Goggles tethered to an ELC, and in 2013, MIT had not yet been given approval to use the WiFi system on the ISS. In order to deal with this constraint, the Goggles Daemon was developed to have the capability to route all messages through the Expansion Port and the SPHERES satellite, so that these messages can be sent to the VERTIGO GUI using the 900 MHz low-speed wireless system, which is known as back-door communication. This is a very low speed system that is only suitable for status messages and basic commands. Uploading code or downloading large amounts of data is not possible using back-door communications.

Figure 5-9 shows the SPHERES Flight GUI with the VERTIGO GUI plug-in activated. At the top of the VERTIGO section of the GUI (highlighted in green), there are two tabs that can be selected with the "Video View" being selected.



Title: VERTIGO GUI Demo for Usability

Objective: Embed the VERTIGO GUI into the SPHERES GUI

Approximate Run Time: 2 min

Test Synopsis:

Start the SPHERES GUI and load the VERTIGO Program.

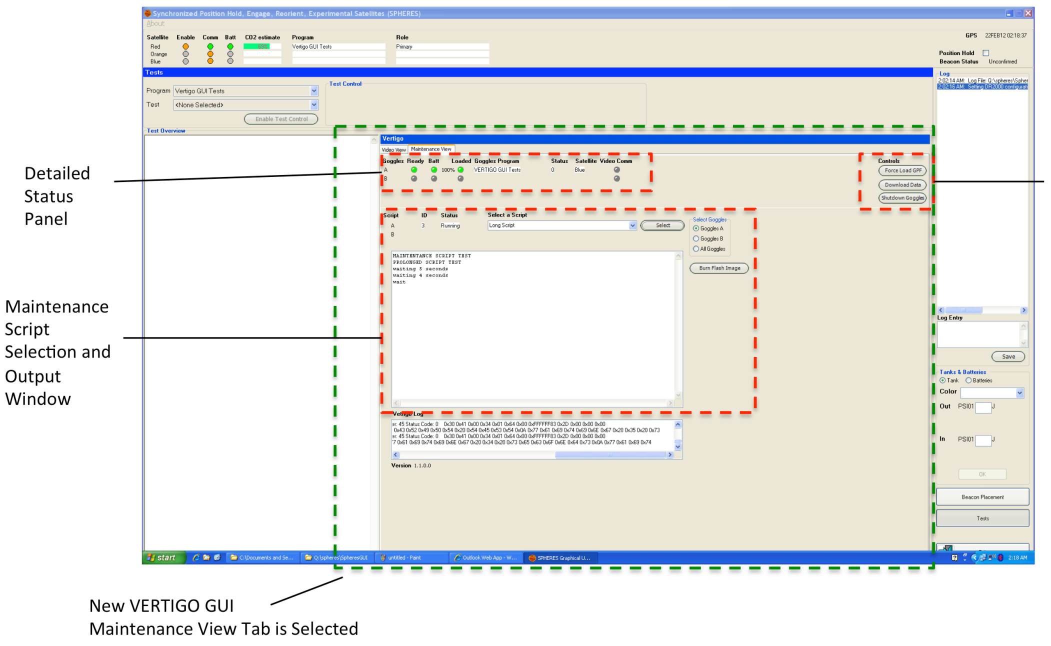
The VERTIGO GUI should start and the screen should be captured by the SPHERES GUI.

Go through the necessary sequence of screen to capture the basic functionality (nominal plus basic maintenance) of VERTIGO for initial evaluation by the PDRT & Usability groups.

Figure 5-9: SPHERES VERTIGO Flight GUI in Video View

This view contains a simplified status panel for the Goggles according to which Satellite the Goggles are attached to (Red, Orange or Blue). A ready indicator that determines if a test is ready to begin, a battery indicator with a percentage of charge remaining and the Program Name for the currently loaded Goggles Program File (GPF). A display selection picks which video mode is being received from the Goggles and displays it in the GUI. This can include raw video from the Goggles cameras or post-processed images that are streamed to the ELC Laptops GUI. Any key-presses and mouse-clicks (along with coordinates) can be sent back to the Goggles for event handling by the research code. This allows for the development of an interactive astronaut interface, which was used for the re-calibration procedures.

Figure 5-10 shows the Maintenance View of the VERTIGO GUI. It has similar control buttons on the right hand side and a similar, but more detailed, status panel for each of the Goggles. The main difference is a Maintenance Script selection drop down box. Here the astronaut can select the script and the Goggles that the script should run on. When the “Select” button is pushed, the script runs and sends its standard output to the Output Window that the crew member can see and is logged for later download.



Goggles	Ready	Batt	Loaded	Goggles Program	Status	Satellite	Video Comm
A	●	● 100%	●	VERTIGO GUI Tests	0	Blue	●
B	●	●	●				●

Script	ID	Status	Select a Script
A	3	Running	Long Script
B			

```
MAINTENTANCE SCRIPT TEST
PROLONGED SCRIPT TEST
waiting 5 seconds
waiting 4 seconds
wait
```

Vertigo Log

```
er: 45 Status Code: 0   0x30 0x41 0x00 0x34 0x01 0x64 0x00 0xFFFFFF83 0x2D 0x00 0x00 0x00
 0x43 0x52 0x49 0x50 0x54 0x20 0x54 0x45 0x53 0x54 0x0A 0x77 0x61 0x69 0x74 0x69 0x6E 0x67 0x20 0x35 0x20 0x73
er: 45 Status Code: 0   0x30 0x41 0x00 0x34 0x01 0x64 0x00 0xFFFFFF83 0x2D 0x00 0x00 0x00
'7 0x61 0x69 0x74 0x69 0x6E 0x67 0x20 0x34 0x20 0x73 0x65 0x63 0x6F 0x6E 0x64 0x73 0x0A 0x77 0x61 0x69 0x74
```

Version 1.1.0.0

Figure 5-10: SPHERES VERTIGO Flight GUI in Maintenance View

Figure 5-11 shows a top-down view of the Goggles highlighting the hardware control panel. The Power Switch is the main switch that connects power from the batteries. The Goggles are set up to automatically begin the booting process when power is connected via this switch. It has an embedded LED that illuminates when there is power. The CPU Reset button causes a reset interrupt to be issued to the onboard processor and triggers a reboot of the operating system. The low battery LED is an “early warning” that the battery is getting close to low. The CPU PWR LED indicates that the CPU is getting the required power and should be running. The CPU RDY LED is activated by the Goggles Daemon once it has been started within the Linux operating system and indicates that the Goggles has completed the boot-up process.

Also in Figure 5-11, there is also a mechanical switch for the illuminating LED lights. A thumb-screw panel hides the CMOS battery that retains the bios settings and clock, as well as a connector for a dongle that includes a PS2 keyboard and mouse, VGA output and two USB 2.0 ports. Also, the hard power button is contained under this panel.

Since the Goggles may create up to 24 GB of data that must be downloaded to the ground, a process is needed to download and manage this data. Note that compressing the data as a whole is very memory and computationally intensive. It is actually more time consuming to compress this data on the Goggles then to just download it uncompressed. Also, NASA requires files to be no larger than 50 MB in size, so these files must be split.

In order to do this correctly, a number of conventions must be set up. The first is that the "/home/GPF_DIR" directory is a symbolic link to the directory of the current program. In this directory, there must be a "Results" folder that contains all of the data that must be downloaded. The first step that happens when the crew member pushes the download data button, the current Goggles Daemon logs are copied to the results directory. Next the Goggles uses the Linux tar program to create an archive file in the "/home2/TempResults" directory. The Linux program then splits this tar file into 50MB chunks in "/home/TempResults". Following this, an MD5 sum is computed for all of these files. The next step is that an ftp client on the ELC laptop logs into the ftp server on the Goggles (over a high speed ethernet connection) and gets each of the split files. This whole process takes approximately two minutes per gigabyte.

Once the files are on the ELC, NASA uses the ISS network to downlink these files to Earth. The above mentioned steps are implemented in reconfigurable ".ini" files that are able to be easily upgraded at a later date. The implementation of these scripts is shown in Listing B.4 and B.5. Once these are on the ground Listing B.6 can be used to verify that no data corruption occurred and B.7 can be used to rebuild the original directory. Once this data has been successfully downloaded, maintenance scripts can be used to delete the data.

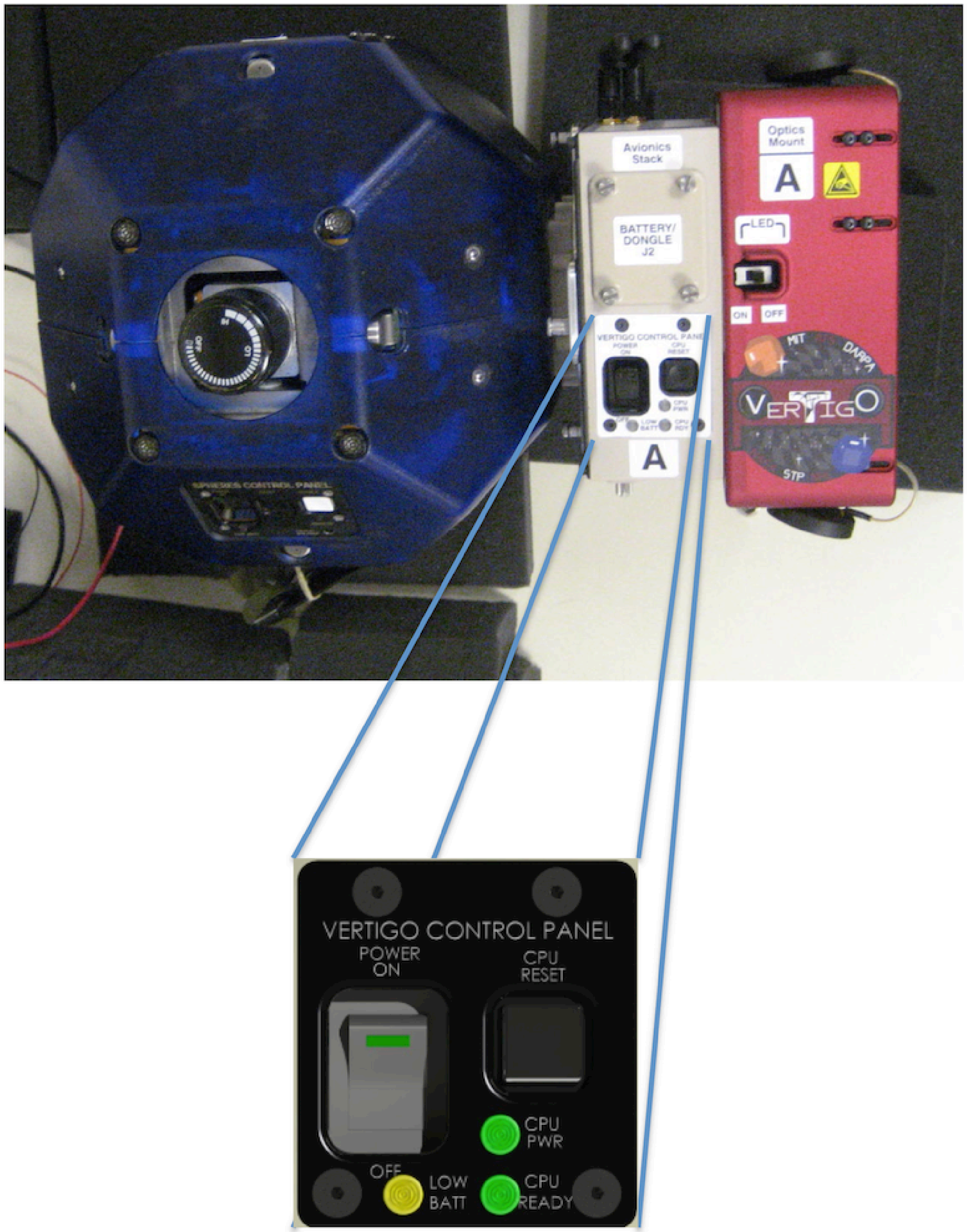


Figure 5-11: VERTIGO Goggles Astronaut Control Panel

5.6 Expansion Requirements and Design

One of the key requirements of an experimental testbed onboard the ISS is flexibility in research applications. An important element of achieving this is to ensure that the payload is expandable. This philosophy led to the requirement that the Goggles should be divided into a computational component and a sensing component. This led to the creation of the Goggles Avionics Stack and the Goggles Optics Mount. The Avionics Stack includes the onboard computer, battery, power electronics, astronaut interface. The Optics Mount includes the cameras, illuminating LED lights, global metrology replacement sensors and other components.

This allows the Avionics Stack to be reused as a computational platform by a future experiment that may choose to build and launch a different sensing system. Alternative, an experiment may choose to build something completely different to attach in place of the optics mount. The interconnection between this system is required to be a simple electro-mechanical interface that can be attached by an astronaut. It must also provide access to data and battery power so that future hardware designs are not un-necessarily restricted.

Figure 5-12 shows the connecting faces of the Goggles Avionics Stack and Optics Mount. The “POWER/DATA P1” connector contains regulated 5 and 12 volt power, USB 2.0 data connections, RS232 serial data, a single 1Gbps Ethernet connection and signal lines for the global metrology replacement sensors. Four thumbscrews surround the connector and are used to mechanically attach the two components by the crew member.

The second main element of expandability is to allow the Goggles to communicate with other hardware wirelessly. While the SPHERES has a 900MHz wireless connection, this does not have sufficient bandwidth to transmit video data in real time. For example, sending two 640 by 480 images at 5 Hz requires 3 megabits per second (Mbps), without compression (note that real time compression is often too computationally expensive to run on an embedded platform). As a result, the addition of an 802.11 wireless system was required to allow the Goggles to communicate with each and to stream live video to the SPHERES VERTIGO GUI.

In 2010, when the Goggles hardware was designed, 802.11n was the fastest commercially available standard wireless connection. A USB 2.0 wireless card was selected to be included in the Goggles that could operate in both of 802.11n's frequency ranges (2.4 GHz and 5GHz), while having Linux device drivers that gave good performance. The DLink DWA-160, rev A2 was selected and is based on an Atheros chipset.

The third main element of expandability is to allow the Flash hard disks to be entirely replaced by an astronaut onboard the ISS. This allows entirely new operating systems and software to be updated with very little up-mass requirements. Figure 5-13 shows the astronaut removable "Flash Hard Drive Cover", underneath which there is a spring-loaded tray containing the SATA drives that can be replaced.

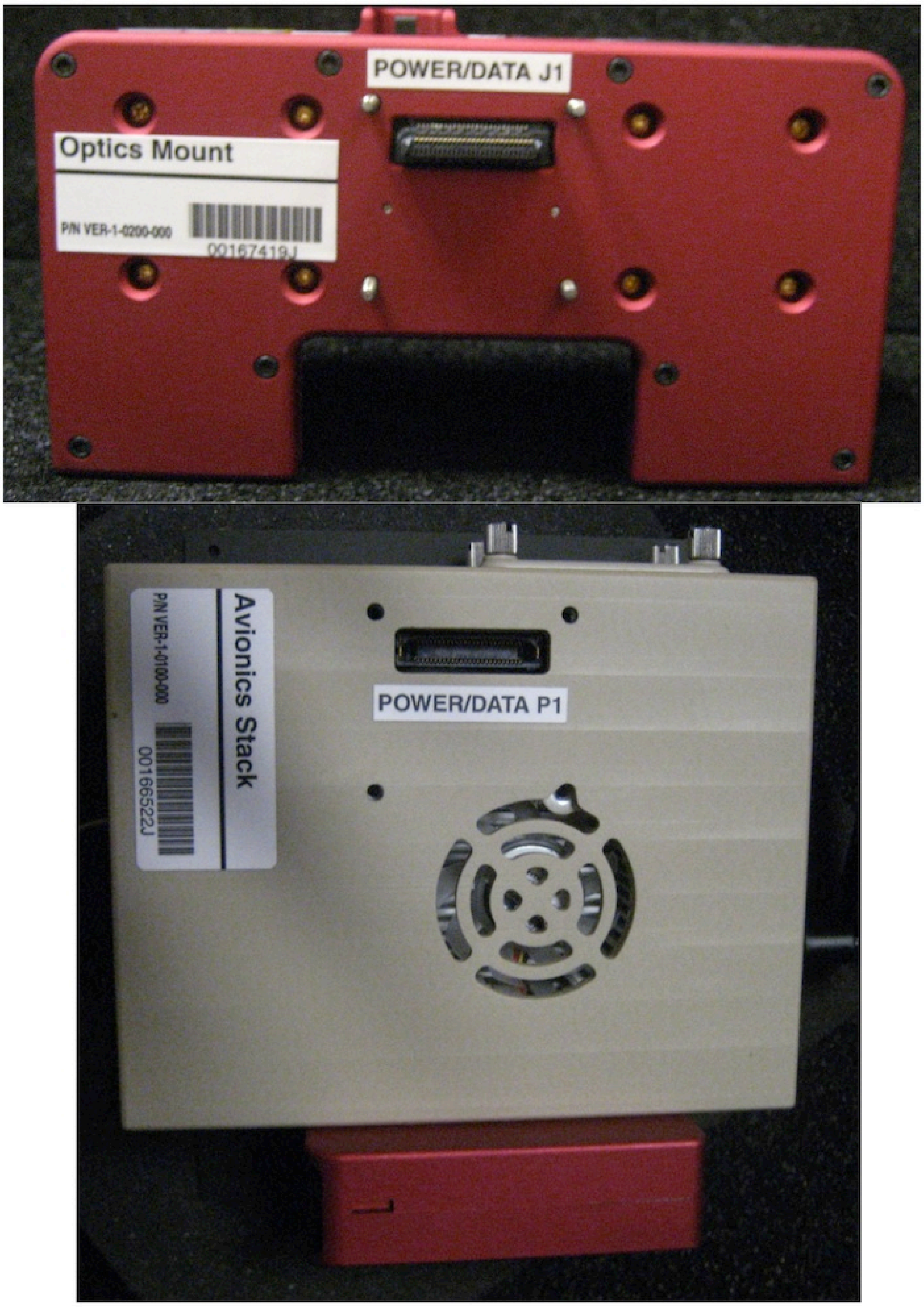


Figure 5-12: Goggles Avionics Stack and Optics Mount Connection

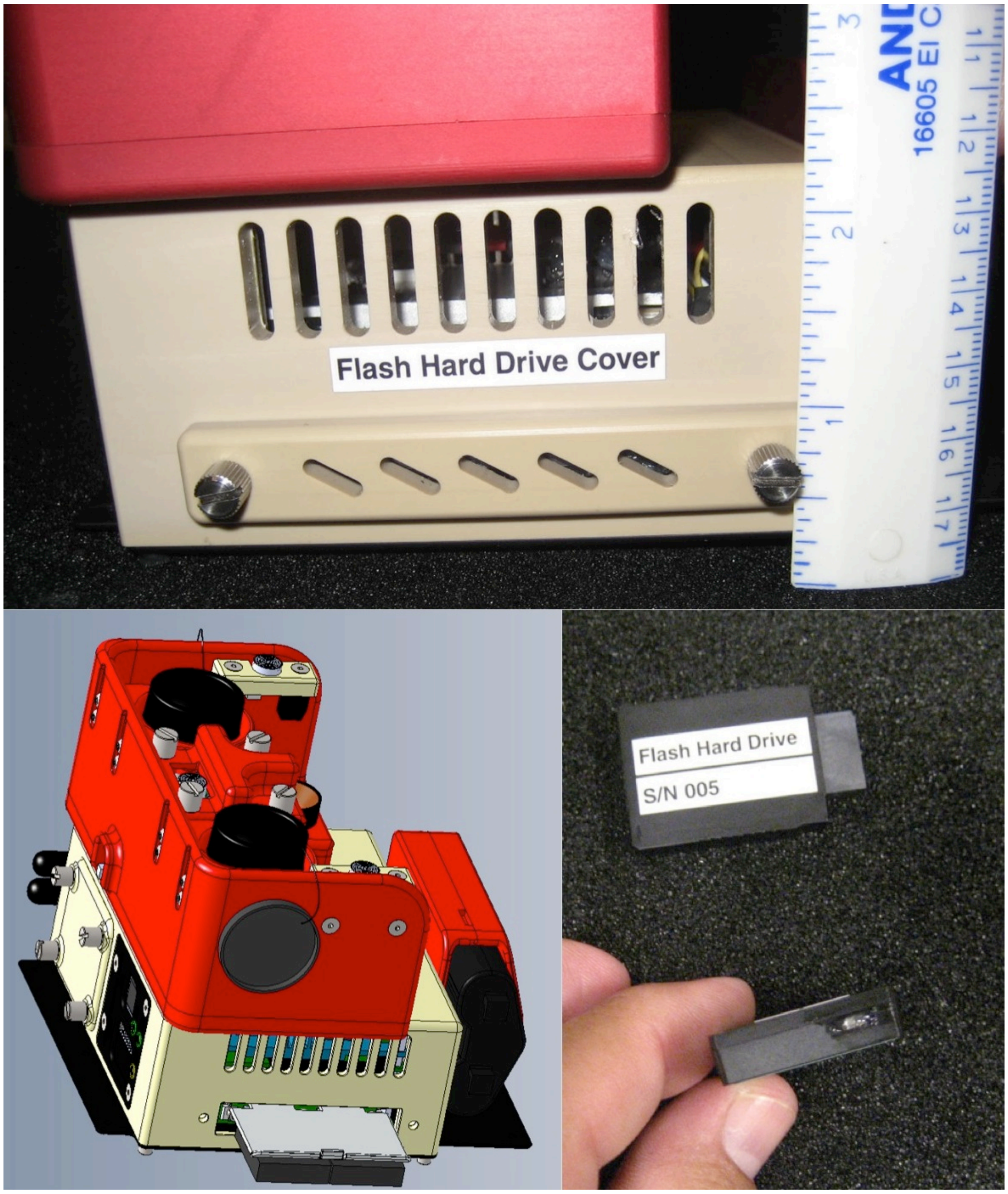


Figure 5-13: Removable Flash Drives

5.7 Power System Requirements and Design

The power consumption of the main components are shown in Table 5.3

Table 5.3: Components Power Consumption

	Minimum (W)	Typical (W)	Maximum (W)
Pico-ITX P830	10.5	13.0	17.7
2 Cameras	1.0	1.0	1.3
WiFi	1.18	2.50	2.50
Illuminating LEDs	0.0	0.322	3.22
Flash Drives	1.25	1.25	3.4
Miscellaneous Items and Margin	0.5	0.5	0.75
Components Total	**14.4**	**18.1**	**28.9**
10% Regulator Inefficiency	1.44	1.81	2.89
Grand Total	**15.87**	**19.88**	**31.76**

Given the power budget, it is important to select a battery that provides enough operational duration so that the crew member does not spend an unnecessary amount of time changing batteries, while keeping the overall system mass down. The highest energy density for commercially available batteries comes from Lithium batteries. However, the requirements to certify these batteries for use on the ISS are extremely time consuming and costly. Therefore, it was desirable to select a battery that is already certified and onboard the ISS. Since the power consumption is similar to the LIIVe Goggles[135, 136], a similar size battery is desirable.

The Nikon EN-EL4A battery, see Figure 5-14 has similar specifications and is already onboard the ISS since it is used for the Nikon D2X cameras the crew members use. It is a three cell lithium-ion battery that outputs 11.1 Volts nominally and has 2.5 ampere hours of capacity. This battery has a mass of 162 grams. This battery should last 104 minutes, 84 minutes and 52 minutes respectively for the minimum, typical and maximum power consumption in Table 5.3.

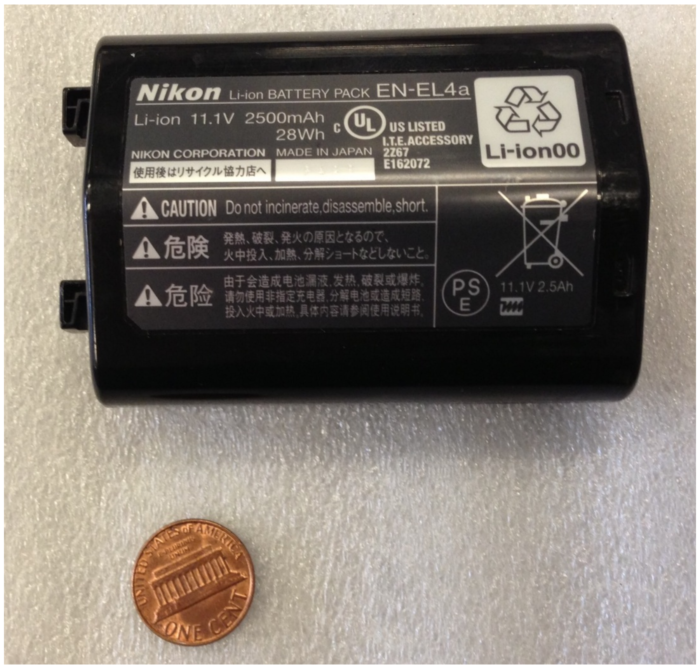

Nikon Li-ion BATTERY PACK EN-EL4a
Li-ion 11.1V 2500mAh 28Wh
NIKON CORPORATION MADE IN JAPAN
US LISTED I.T.E. ACCESSORY 2Z67 E162072
Li-ion00
CAUTION Do not incinerate, disassemble, short.
11.1V 2.5Ah
ONE CENT

Figure 5-14: Nikon EN-EL4A Battery with US Penny for Reference

5.8 Electro-Mechanical Requirements and Design

The functionality that is required for the VERTIGO Goggles to meet their mission objective along with the selected high level design has been described in the previous sections. In addition to this, there were significant requirements placed on the electro-mechanical design by NASA in order to operate safely within the crew volume of the International Space Station (it is outside of the scope of this thesis to review these requirements).

The objective of the electrical and mechanical design is to integrate all of the required functionality into a single package that minimizes the system mass. Additionally, the development was under a tight schedule and budget, which must be managed against the risk of failure. A small five-person team of developers from Aurora Flight Sciences led the detailed design and fabrication, with day-to-day assistance and management by two people from the MIT Space Systems Laboratory.

While delivering four copies of VERTIGO Goggles within the time and budget was a significant challenge (see O'Connor for details [104]), it did not involve any new or novel approaches in design or manufacturing. For all of these reasons, the electro-mechanical design of the Goggles is not claimed as a contribution in this thesis. However, its final design will be reviewed in this section.

The high level electrical connection diagram is shown in Figure 5-15 and the mechanical layout of the printed circuit boards is shown in Figure 5-16. Figure 5-17 shows the mechanical layout for the Goggles system. The final version of the Goggles is shown in Figure 5-18 and 5-19 at the MIT Space Systems Laboratory. Figure 5-16 and 5-21 are photos of the flight hardware taken on the day it was delivered to NASA for launch.

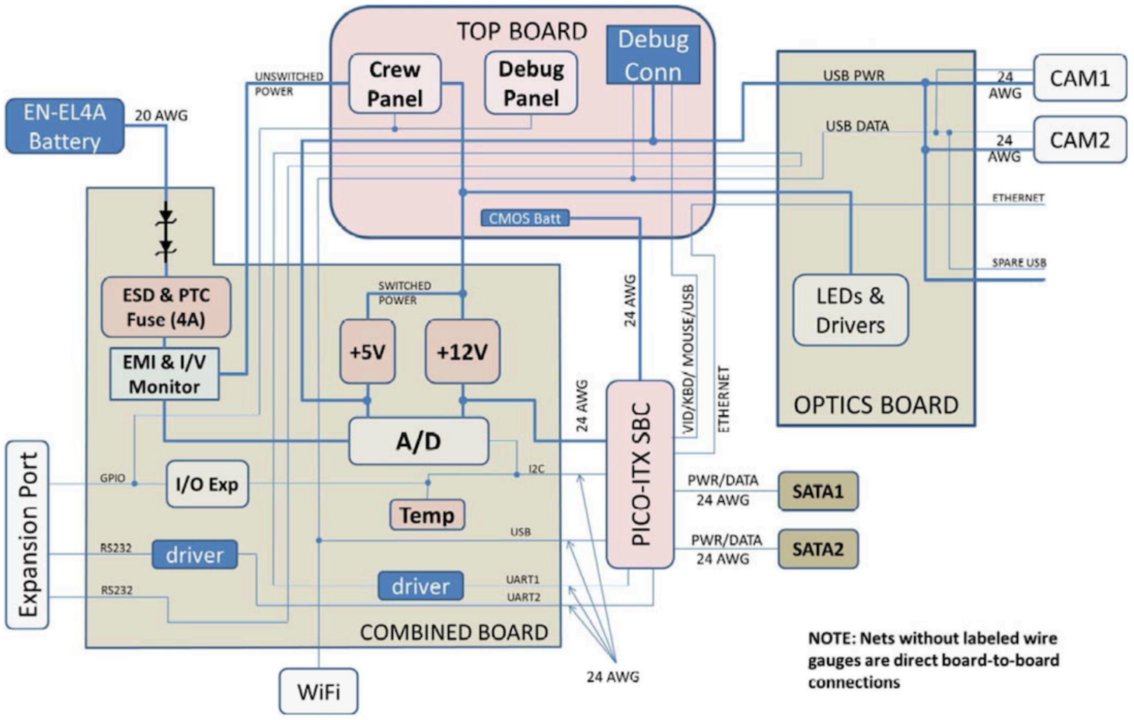


Figure 5-15: High Level Electrical Diagram for VERTIGO Goggles (Image Courtesy of Aurora Flight Sciences)

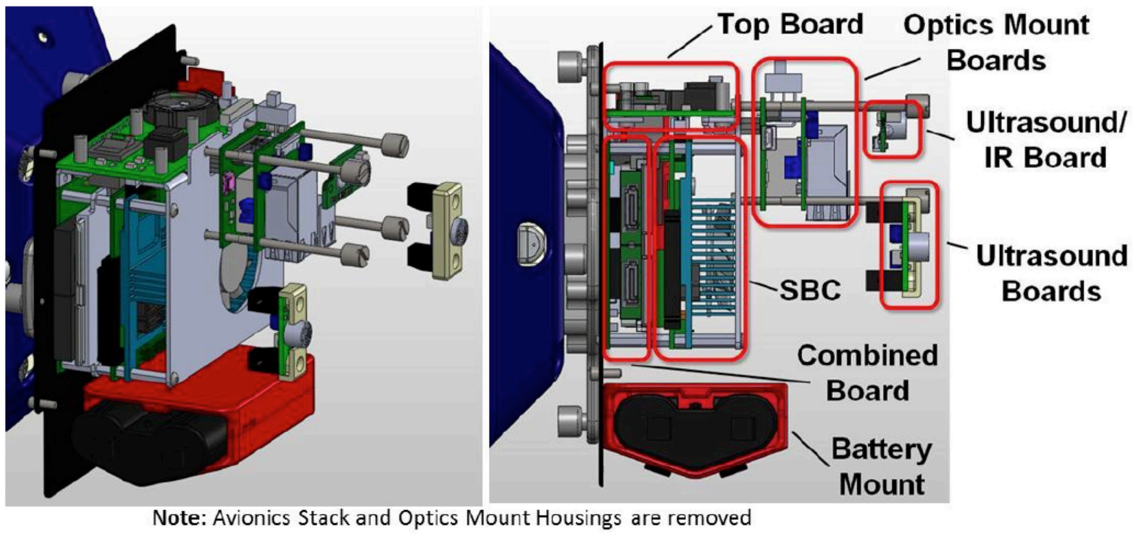


Figure 5-16: Printed Circuit Board Mechanical Layout for VERTIGO Goggles (Image Courtesy of Aurora Flight Sciences)

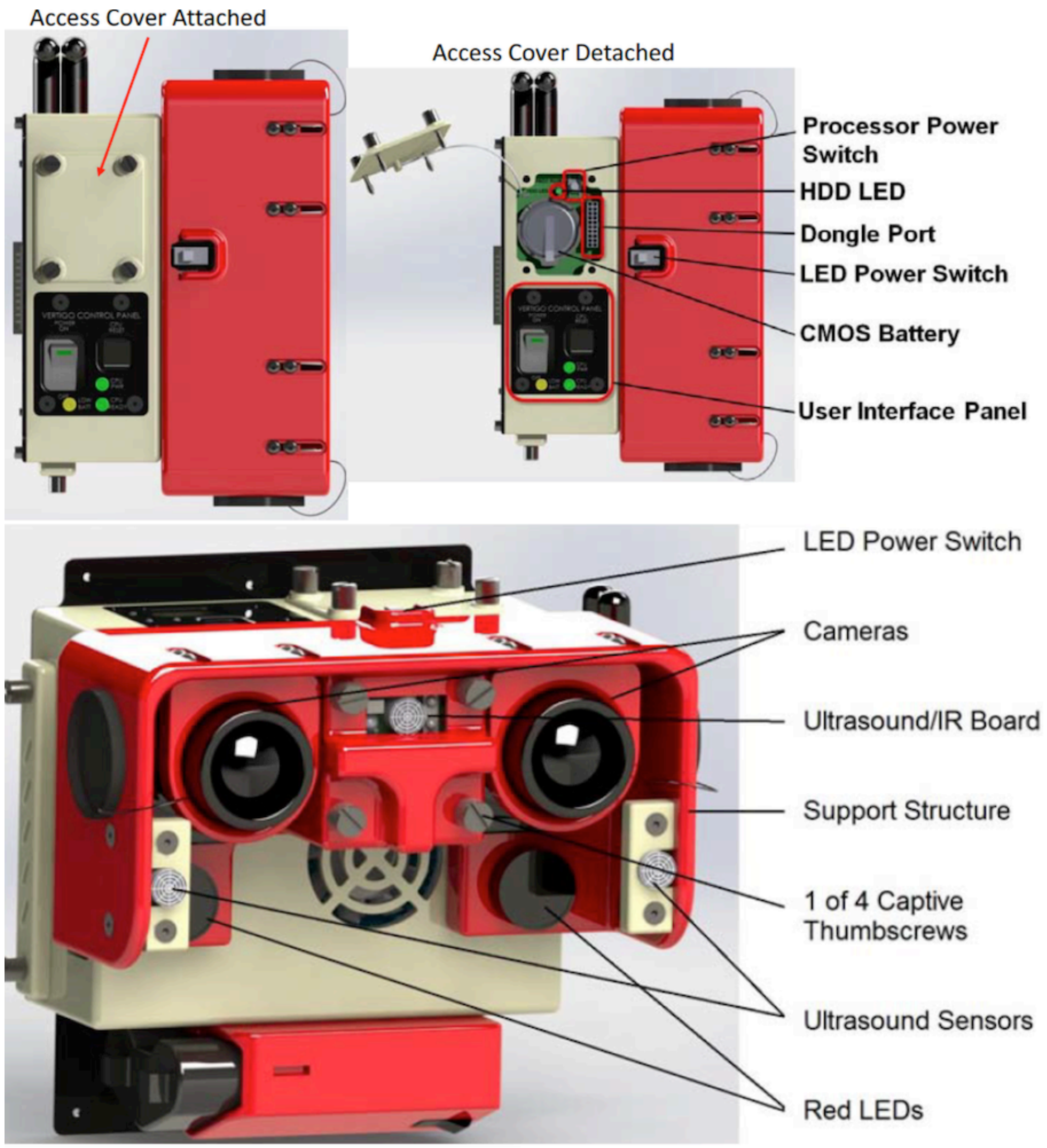


Figure 5-17: VERTIGO Goggles Mechanical Layout (Image Courtesy of Aurora Flight Sciences)

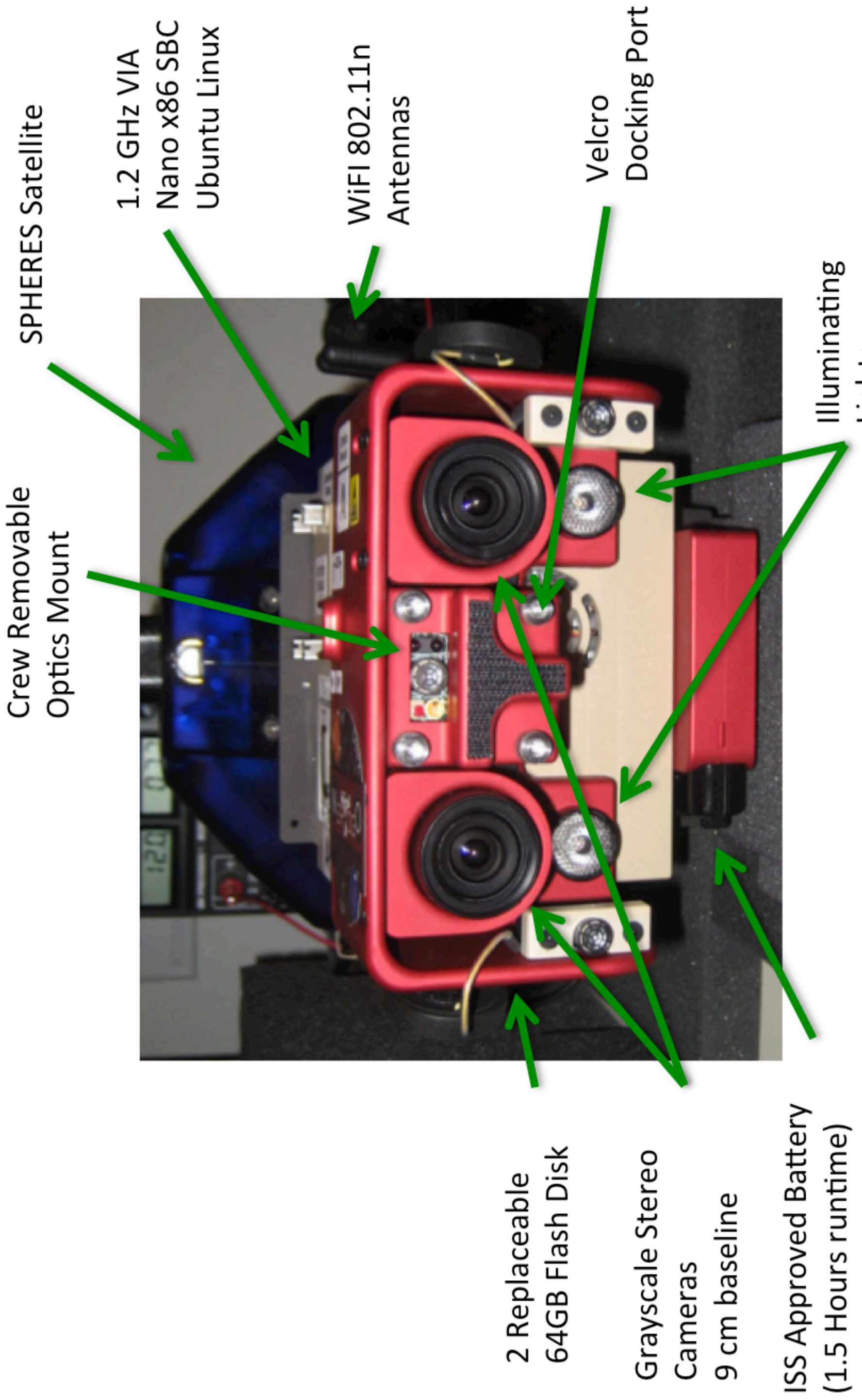

SPHERES Satellite
1.2 GHz VIA Nano x86 SBC Ubuntu Linux
WiFI 802.11n Antennas
Velcro Docking Port
Crew Removable Optics Mount
Illuminating Lights
2 Replaceable 64GB Flash Disk
Grayscale Stereo Cameras 9 cm baseline
ISS Approved Battery (1.5 Hours runtime)

Figure 5-18: VERTIGO Goggles Major System Components

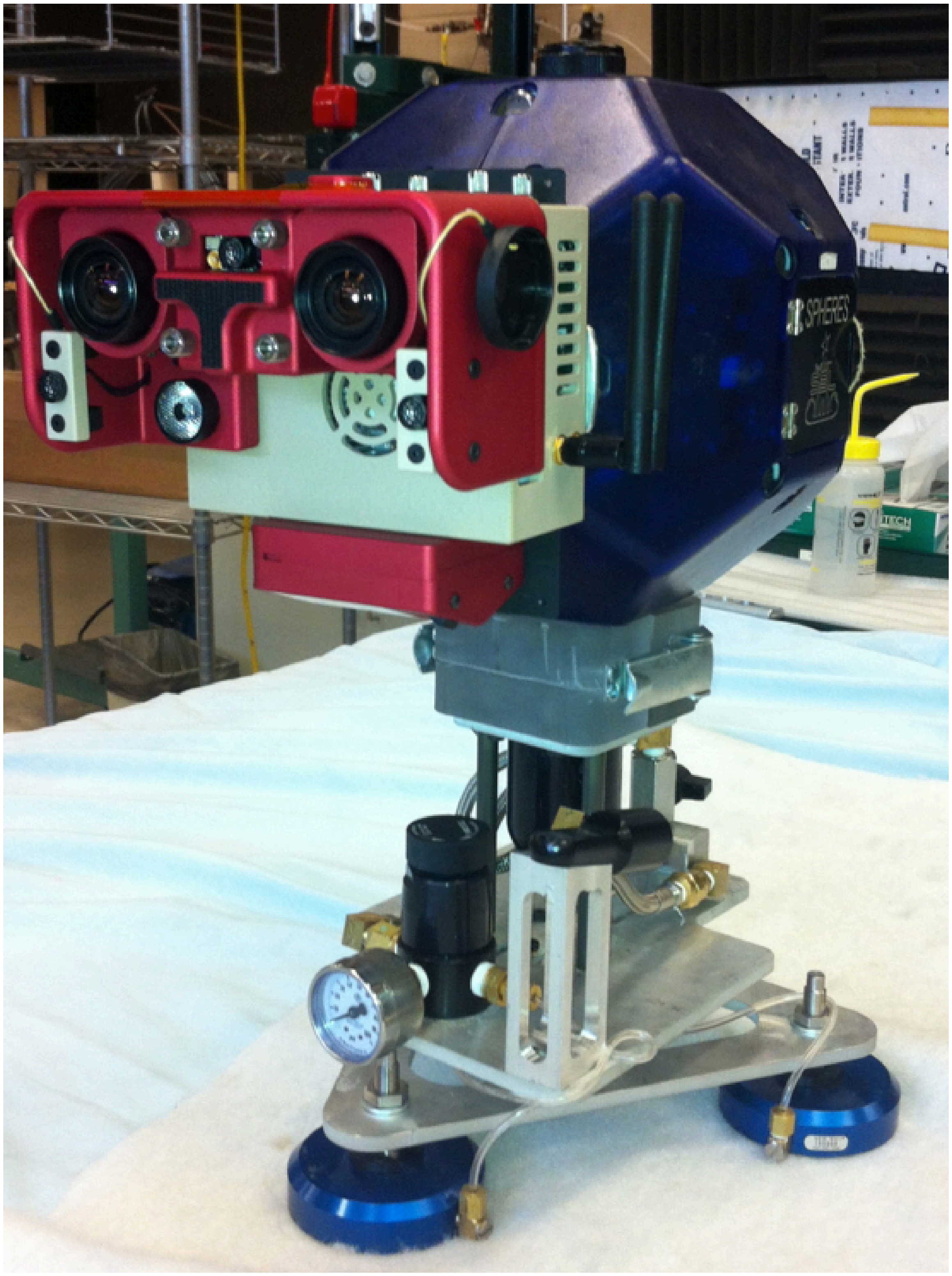


Figure 5-19: VERTIGO Goggles attached to SPHERES Satellite in the MIT Space Systems Laboratory

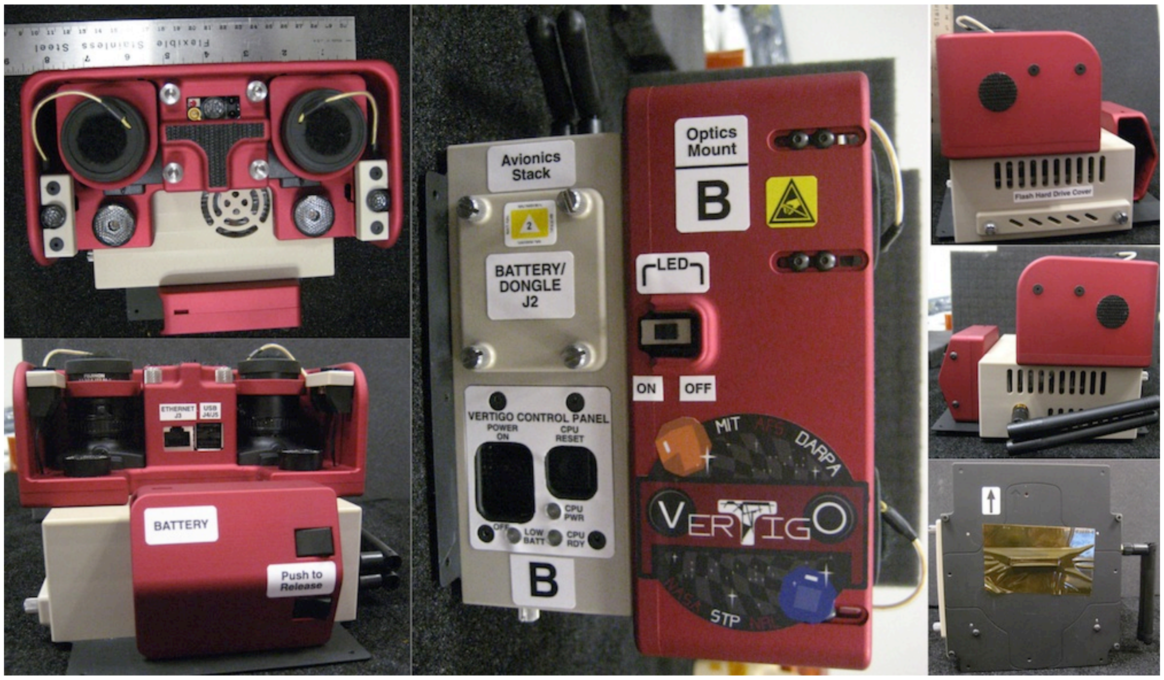


Figure 5-20: VERTIGO Goggles Flight Hardware Prior to Delivery

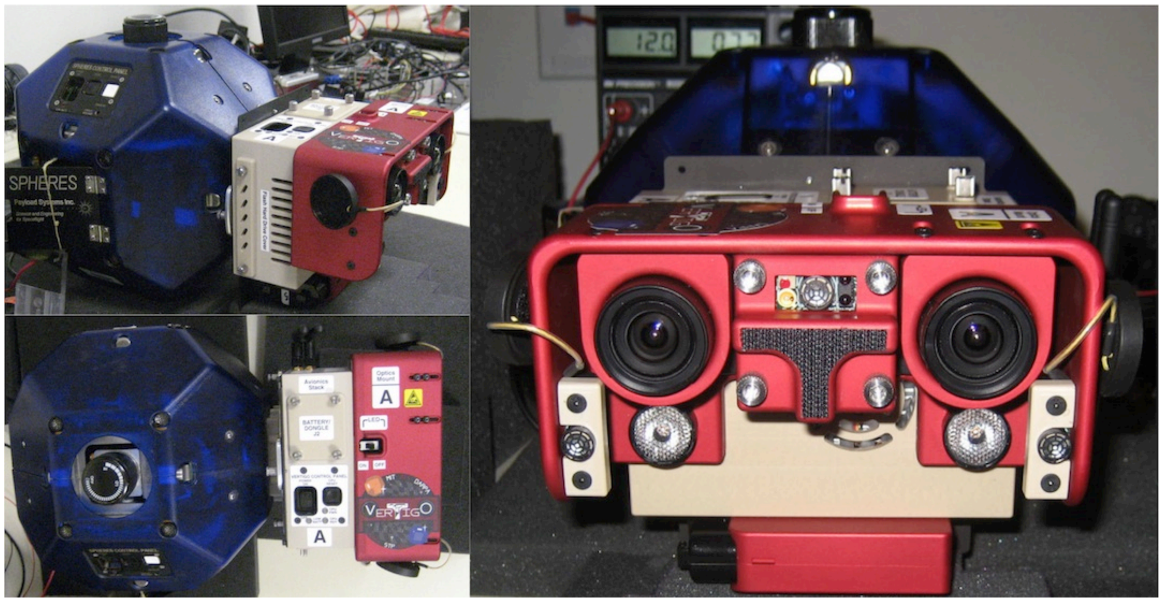


Figure 5-21: VERTIGO Goggles Flight Hardware attached to SPHERES Satellite

5.9 Specifications Summary of SPHERES VERTIGO Goggles

Table 5.4 summarizes the specifications and capabilities of the VERTIGO Goggles.

Table 5.4: VERTIGO Goggles Complete Specifications

Processor	1.2 GHz Via Nano U3300 (Single Core, OOE, 1MB L2 Cache)
Chipset	VIA VX900 (Via Pico-ITX P830 SBC)
RAM	4GB DDR3 1066 MHz
Flash Disk	Two 64 GB SATA (128 GB Total)
Operating System	Ubuntu Linux 10.04 Server
Cameras	IDS-Imaging uEye LE 1225-M-HQ
Camera Sensor	1/3" Monochrome CMOS with Global Shutter, HQ Filter
Camera Configuration	9.0cm Stereo baseline, HW exposure timer and sync
Camera Resolution	640 x 480 pixels at 6 μm per square pixel
Lens Mount	CS-Mount
Lens Type	Fujinon 2.8mm, f/1.3 (CCTV Lens for 1/3“ and 1/4“ Imager)
Frame Rate	87 FPS (Camera Max), 10 FPS (Typical)
Exposure	80 μ s - 5.5 s
Lights	2× Phillips Rebel Star LED Red-Orange
Lights Dominant Wavelength	617 nm
Lights Intensity	134 lm @ 700mA (per LED)
Wireless Communications	802.11n, 2.4 and 5 GHZ (DLink DWA-160, rev A2, Atheros Chipset)
Battery	Nikon EN-EL4a Rechargeable Li-ion: 11.1V, 2500mAh, 162 g
Power Consumption	16 W (Idle), 20 W (Typical), 32 W (Max)
External Ports	2 × USB 2.0, Gigabit Ethernet
Optics Mount Connector	Unreg PWR (2.0A), RS232 SPH, RS232 Pico, US/IR Met Beacons, Gigabit ETH, 2 × USB
Dongle Connector	Keyboard, Mouse and VGA, 2× USB 2.0
Total Mass	1.326 kg (without battery)

5.10 Camera Calibration Approach

Despite the fact that there are mechanical supports designed for the lenses to help lock them into position, it was still considered a possibility that the cameras may be vibrated or knocked out of calibration during transportation to the launch site in Khazakstan. The Goggles were shipped via FedEx to Houston, repackaged and sent via United Airlines Cargo to Moscow, were loaded onto a train to Baikonur Cosmodrome, where they were packaged into a Soyuz crew compartment and launched to the International Space Station. Additionally, during operations, there could be some kind of collision that knocks the lenses out of calibration.

As a result, there was a need for the crew members to be able to evaluate whether or not the calibration values (i.e. the intrinsic and extrinsic parameters) correctly matched the current physical set up of the cameras, and if not, to be able to perform a re-calibration using the methods described in Section 2.8.

One of the important aspects of using typical camera calibration methods (including the ones provided by OpenCV), is that the calibration target must provide a checkerboard with a number of corners for correspondences which lie on a "perfectly" flat plane. Variations of more than a few millimeters will be detectable by the cameras and will likely affect the final results. Therefore, a rigid camera calibration target was built out of aluminum that is 0.25 inches thick. The target is shown in Figure 5-22 and is 9 inches by 15.5 inches and weighs a hefty 1.6 kg. Each square on the checkerboard is exactly one inch by one inch.

A second important issue is to have good visual texture on any object so that the stereo depth maps and feature points have strong signatures to match against. A set of textured stickers was designed by Makowka [81] to have a strong response to both stereo disparity algorithms and feature points. The resulting stickers were designed to fit on the SPHERES satellites and are shown in Figure 5-23 with astronaut Thomas Marshburn.

Prior to each session, the astronaut is required to perform a camera checkout. This evaluates whether the parameters correctly match the current camera configuration.

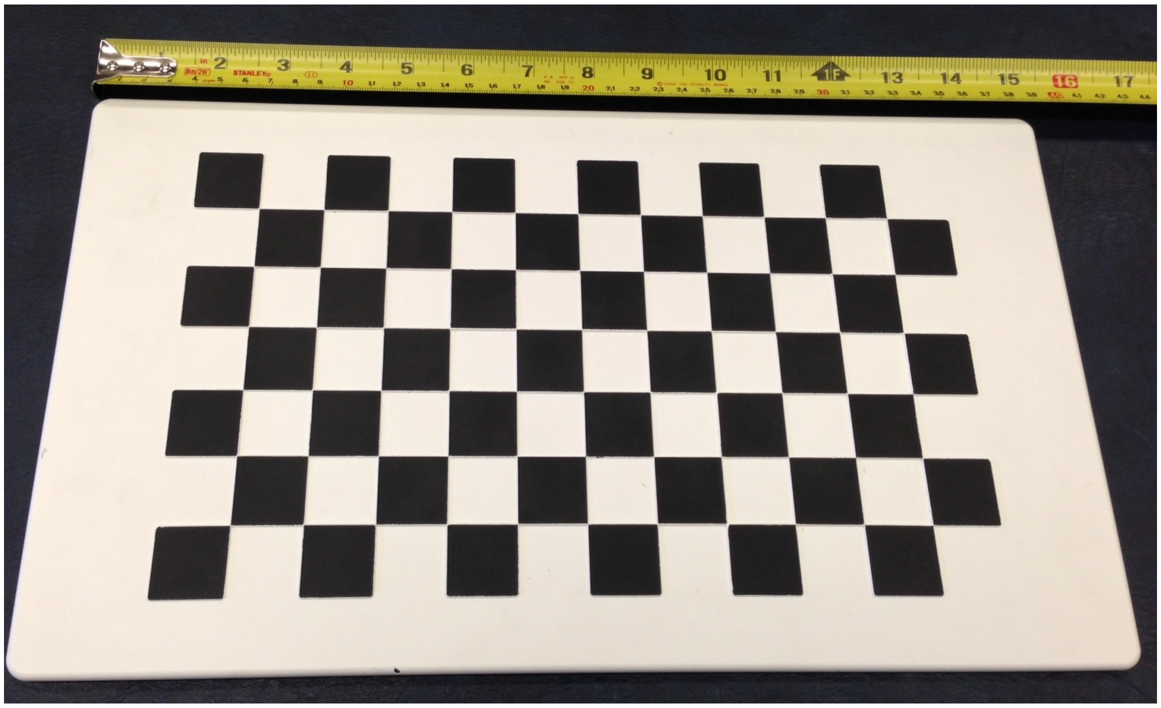


Figure 5-22: Camera Calibration Target

This is implemented as a maintenance script that calls a program which streams video the Flight GUI over a TCP connection (i.e. ethernet or possibly WiFi). This maintenance script runs a program (source code is shown in Section B.4) that guides the crew member through a checkout and a recalibration if necessary.

The first screen that is shown to the astronaut during the Camera Checkout is Figure 5-24. This checks a number of parameters against preprogrammed ranges to ensure that the parameters are close to what is expected. If the value is in range, a green light is shown next to it, if the value is out of range an orange circle is shown next to it, and the crew member should perform a re-calibration by pushing the "r" button. 5-24 shows a good example on the top with all green circles, while a bad example is shown on the bottom with a few orange circles.

The second screen that is shown to the astronaut is Figure 5-25. This figure shows a view from the left camera with a set of green horizontal lines overlaid. Additionally, when the calibration target is in view of both of the cameras, it draws a blue line between the corresponding checkerboard corners. This line should be horizontal and aligned with the green lines. Additionally, the checkerboard corners are triangulated

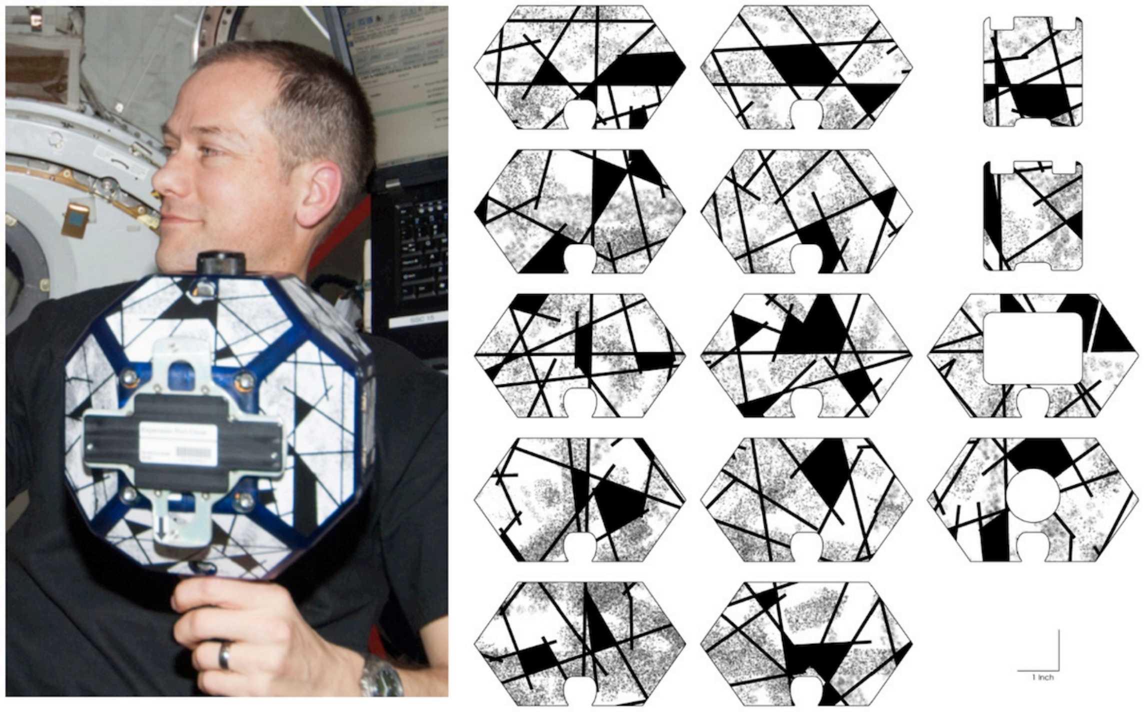
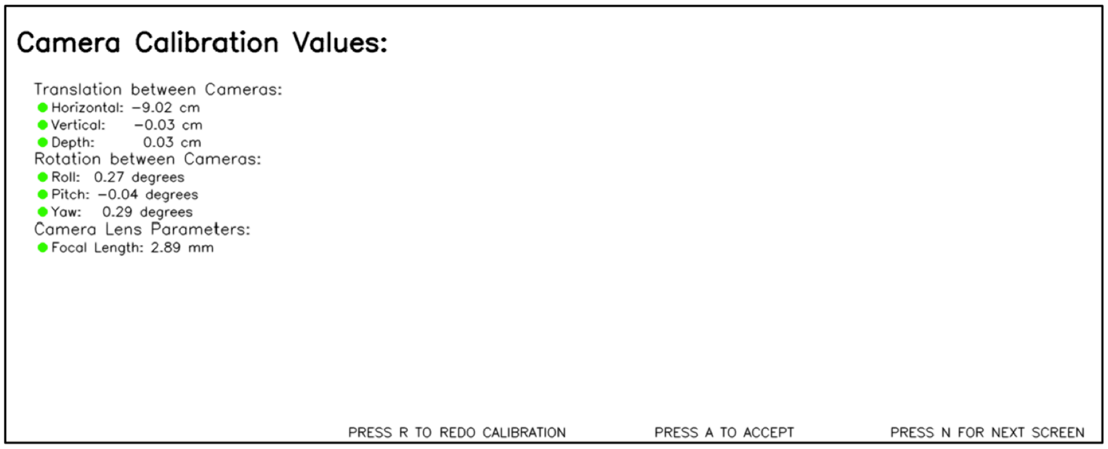


Figure 5-23: Textured Stickers Design and Use on the International Space Station by Astronaut Thomas Marshburn

and it is checked whether or not they are close enough to one inch, and draws an orange circle if it is not. The astronaut must make a judgement call on how close the blue lines are to horizontal, if they deem they are not horizontal, they are instructed to perform a recalibration.

The third screen that is shown to the astronaut is Figure 2-15. This shows a live image on the left hand side and the computed depth map using OpenCV's sum of absolute differences method on the right hand side. The crew members are told to hold a target SPHERE (with textured stickers) approximately 0.5 to 1.0 meters away from the camera so that it is visible on the screen. The crew member then checks that the satellite is mostly filled in gray in the right hand depth map. Next they move the target away from the camera to verify that the target turns darker and remains filled in. Finally, they move the target closer to the camera to verify that the target turns lighter and remains filled in (up to approximately 20 cm away from the camera where it will be too close to be detectable). If any of these steps fail, the crew member is told to perform a recalibration.

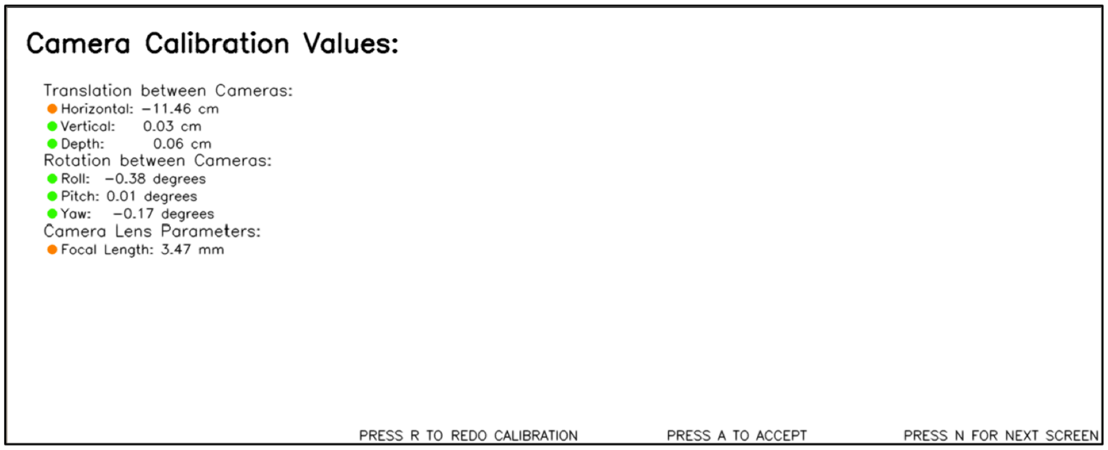
GOOD EXAMPLE:


Camera Calibration Values:

Translation between Cameras:
● Horizontal: −9.02 cm
● Vertical: −0.03 cm
● Depth: 0.03 cm
Rotation between Cameras:
● Roll: 0.27 degrees
● Pitch: −0.04 degrees
● Yaw: 0.29 degrees
Camera Lens Parameters:
● Focal Length: 2.89 mm

PRESS R TO REDO CALIBRATION PRESS A TO ACCEPT PRESS N FOR NEXT SCREEN

BAD EXAMPLE:


Camera Calibration Values:

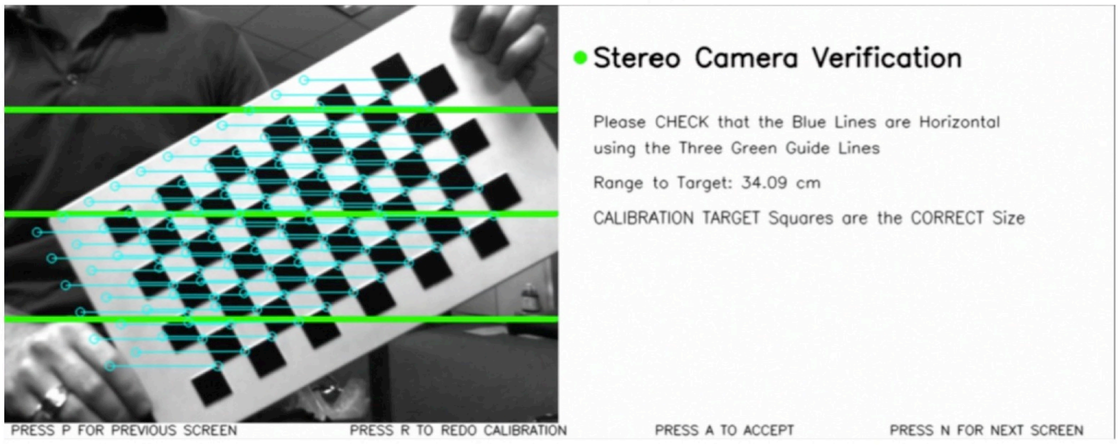
Translation between Cameras:
● Horizontal: −11.46 cm
● Vertical: 0.03 cm
● Depth: 0.06 cm
Rotation between Cameras:
● Roll: −0.38 degrees
● Pitch: 0.01 degrees
● Yaw: −0.17 degrees
Camera Lens Parameters:
● Focal Length: 3.47 mm

PRESS R TO REDO CALIBRATION PRESS A TO ACCEPT PRESS N FOR NEXT SCREEN

Figure 5-24: Camera Checkout Screen #1: Calibration Values

In order to redo the calibration, the astronaut must capture a set of 30 photos that illustrate good correspondences between the left and right camera to be used by the methods in Section 2.8. The crew members are instructed to hold the calibration target approximately 0.5 meters away so that there is multi-colored circles that appear in both the left and right images. An example of this is shown in Figure 5-27. The astronaut must push the space bar to capture an image, which will increment the counter in the lower left of the screen and store that image only if the correspondences are good in both images. They are then instructed to place the target in each of the four corners of the images, taking six captures in each location for a total of 30 images. Once this is done, pushing the "f" key runs the optimization algorithm to solve for

GOOD EXAMPLE:

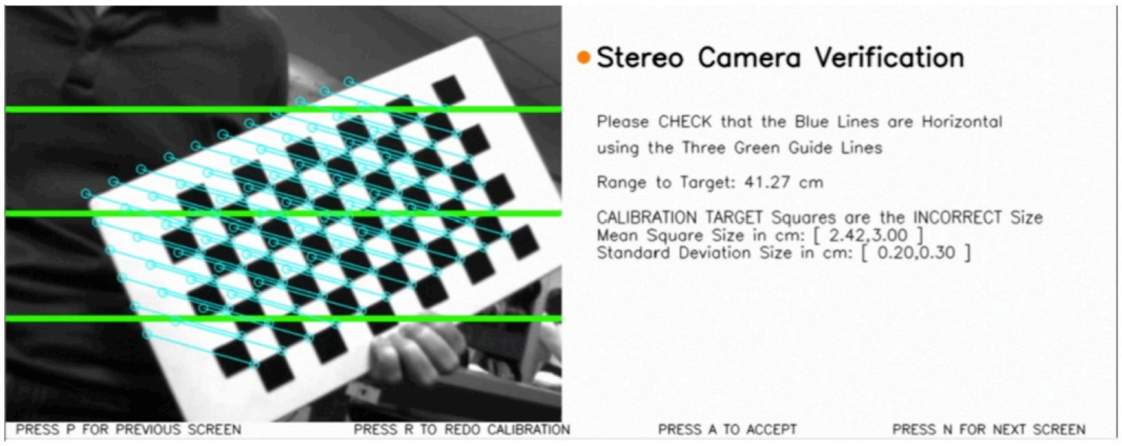

● Stereo Camera Verification

Please CHECK that the Blue Lines are Horizontal
using the Three Green Guide Lines

Range to Target: 34.09 cm

CALIBRATION TARGET Squares are the CORRECT Size

BAD EXAMPLE:


● Stereo Camera Verification

Please CHECK that the Blue Lines are Horizontal
using the Three Green Guide Lines

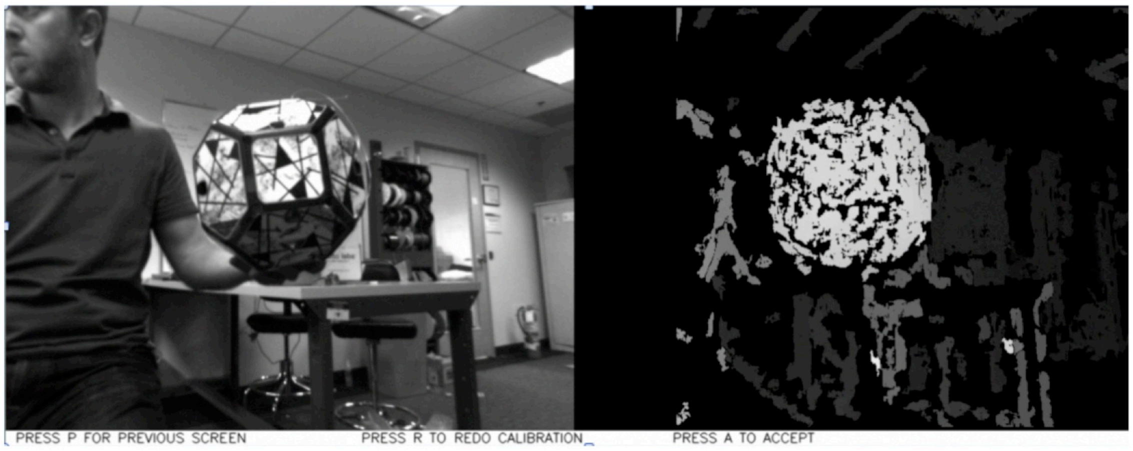
Range to Target: 41.27 cm

CALIBRATION TARGET Squares are the INCORRECT Size
Mean Square Size in cm: [2.42,3.00]
Standard Deviation Size in cm: [0.20,0.30]

Figure 5-25: Camera Checkout Screen #2: Stereo Camera Verification

the intrinsic and extrinsic parameters. This optimization takes approximately five minutes to run on the Goggles, after which they are required to perform a full checkout of the calibration to ensure the new solution is correct. If it is correct, they will press the “a” key to accept the values and store them permanently to the flash drives. If they are not correct, they will press the “r” key to perform another recalibration.

GOOD EXAMPLE:



BAD EXAMPLE:

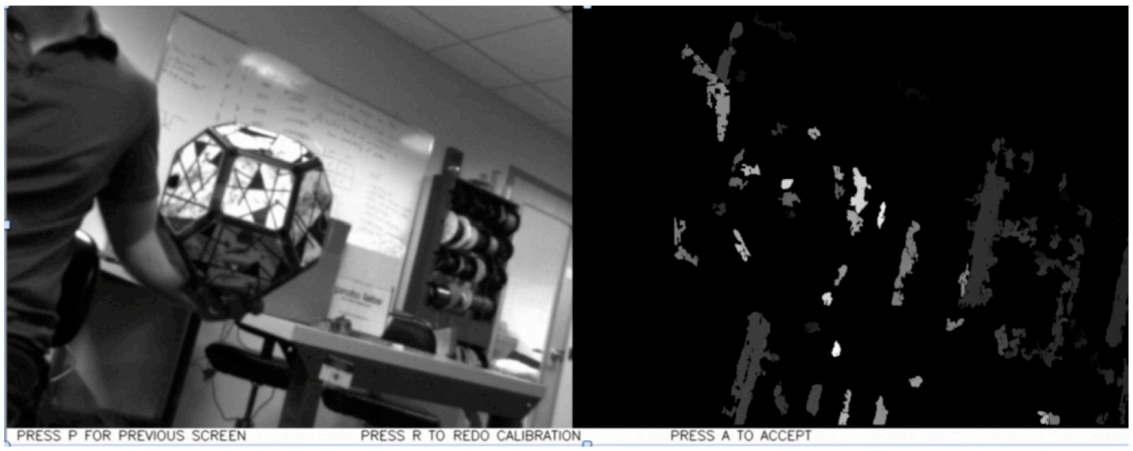


Figure 5-26: Camera Checkout Screen #3: Stereo Depth Map

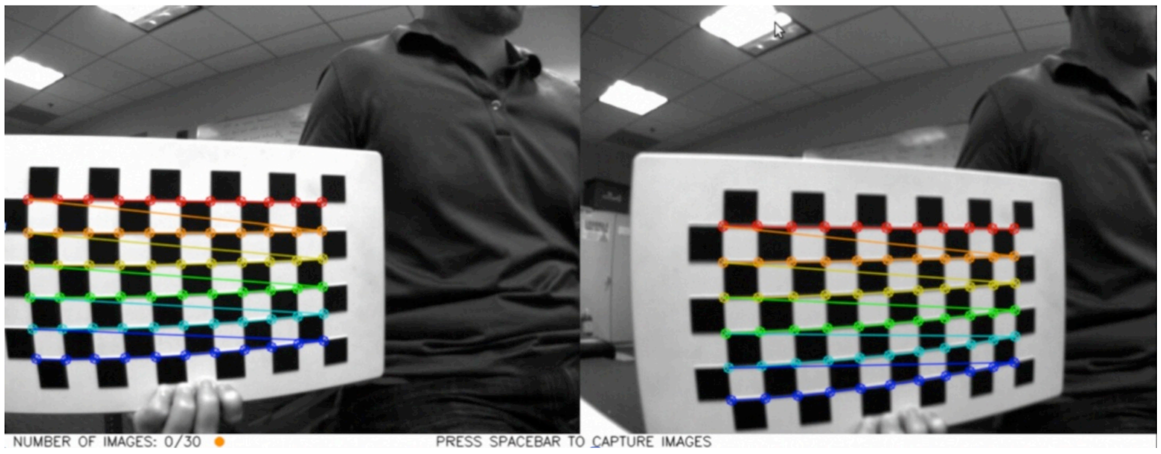


Figure 5-27: Camera Recalibration Screen

5.11 Results of Checkout on the International Space Station

Four VERTIGO Goggles were built and tested at Aurora Flight Sciences and MIT Space Systems Laboratory. The best two were selected to go to space and given the designation Goggles A and Goggles B. On August 1, 2012, both Goggles, and the associated hardware, were delivered to NASA and shipped to Houston, Texas. There, they were repackaged and shipped by air and train to Baikonur Cosmodrome, where they were packaged into Soyuz TMA-06M and were launched along with Astronauts Kevin Ford, Oleg Novitskiy and Evgeny Tarelkin on October 23, 2012. On February 26, 2013, Astronaut Thomas Marshburn performed the first checkout of Goggles B. Photos from this checkout are shown in Figure 5-28.

The checkout on February 26, 2013 involved running a set of scripts and tests that were developed as part of a SPHERES Program File (SPF) and Goggles Program File (GPF). In order to do this, the new SPHERES Flight GUI with the VERTIGO plugin was installed. During most of the operations, the author of this thesis, Brent Tweddle, was enabled to speak directly with the crew member Thomas Marshburn on the space-to-ground audio communications link.

The VERTIGO Goggles and SPHERES hardware functioned as it was supposed to. The hardware was able to be installed correctly, booted when powered on, and the Goggles Daemon was able to successfully communicate with the Flight GUI. A few issues due to IP address configuration arose and were handled in real time. A new SPF and GPF was loaded to the SPHERES satellite and Goggles respectively.

The checkout began with the running of two maintenance scripts. The first set-up a new directory structure to be compatible with a few download changes that were made after the hardware was shipped. The second script recorded the "software status" by checking a number of linux system configurations (i.e. running processes, hard disk mountings, network configurations, USB devices, etc.). Both of these scripts ran successfully and recorded the required data.

The third script was the camera checkout and calibration. Using the procedures

Thomas Marshburn correctly concluded that a new calibration needed to be performed, since the Goggles were slightly out of calibration. The images in Figure 5-29 show the screens that lead to a failing evaluation. After recalibration, the images in Figure 5-30 show the results. The differences in parameters are summarized in Table 5.5 (note that the rotation is a change in rotation between Delivery and ISS Recalibration specified using axis angle parameters). Note that the main difference appears to be the translation parameters (mainly X and Z axis) which moved less than half a millimeter. This indicates that the camera lens had moved slightly during shipping, but had not rotated, which is likely due to the fact that the optics mount is manufactured with screws and not press fits.

Table 5.5: Parameter Changes Between Delivery and ISS Recalibration on Test Session 37

Parameter	**Delivery**	**Recalibration**
Left Focal Length	2.8979 mm	2.8007 mm
Left Optical Center	[280.8 282.3] pix	[280.7 281.0] pix
Right Focal Length	2.8778 mm	2.7923 mm
Right Optical Center	[302.3 230.8] pix	[301.3 233.5] pix
Translation	[-9.0443 -0.1159 -0.0358] cm	[-9.0321 -0.1121 -0.0074] cm
Rotation Change	N/A	0.3791^o about [0.3449 0.9372 0.0521]

The remainder of the checkout session involved running tests that confirmed that the Goggles could communicate with the SPHERES satellites. This is required for operational purposes as well as scientific purposes. These tests proved that the Goggles could indeed provide information to the SPHERES satellites for use in their control algorithm. A VERTIGO “Quick Checkout” similar to the typical SPHERES “Quick Checkout”, whose main difference was that the cameras captured videos during the test. Two tests were preformed to estimate the gyro biases and determine the new inertia properties with the Goggles attached[30]. Lastly, a visual-inertial test proved that vision from the Goggles could be integrated with inertial measurements from the SPHERES to perform a low computational visual inspection algorithm [33]. Once the test was complete, 13 GB of data was downloaded from the Goggles to the ELC where it was transferred down to the ground over the next few days.

Three issues occurred during this test session that. The first and most significant issue is that it was discovered that the Goggles were occasionally dropping frames during the image-saving process. This is due to Linux's non-real-time nature and the method of interactions with the virtual memory file system. This problem was fixed in later test sessions by adding a buffer for the images that would not loose data if a there was a delay in processing the write to disk operation.

The second issue was a file corruption during the data download. NASA required the multiple giga-bytes of files to be split into 50 MB files in order to be compatible with the space to ground link. During the first test session, it was found that three of the files were corrupted (this was confirmed by the MD5 checksum and the Linux tar program). It was determined that the error occurred between the ELC and the ground system, and the three files were re-downloaded successfully a few days later.

The third issue was that the gains on the inertial navigation algorithm were incorrect. While the system worked well in simulation, the dynamics onboard the ISS were different enough to cause the algorithm to be underdamped. By lowering the gains, it was possible to get stable performance.

Other minor operational and procedural questions and issues arose during the operations that were easily corrected by talking directly with the astronaut. Notes were taken to further improve the procedures and operations for future sessions.

A second test session was performed by Kevin Ford on March 12, 2013, that mainly gathered science data, including the data set used in Chapter 6. A third test session occurred on April 16, 2013, where Thomas Marshburn performed a checkout on Goggles A. This session had similar results and required a recalibration for a minor failure of the camera checkout.

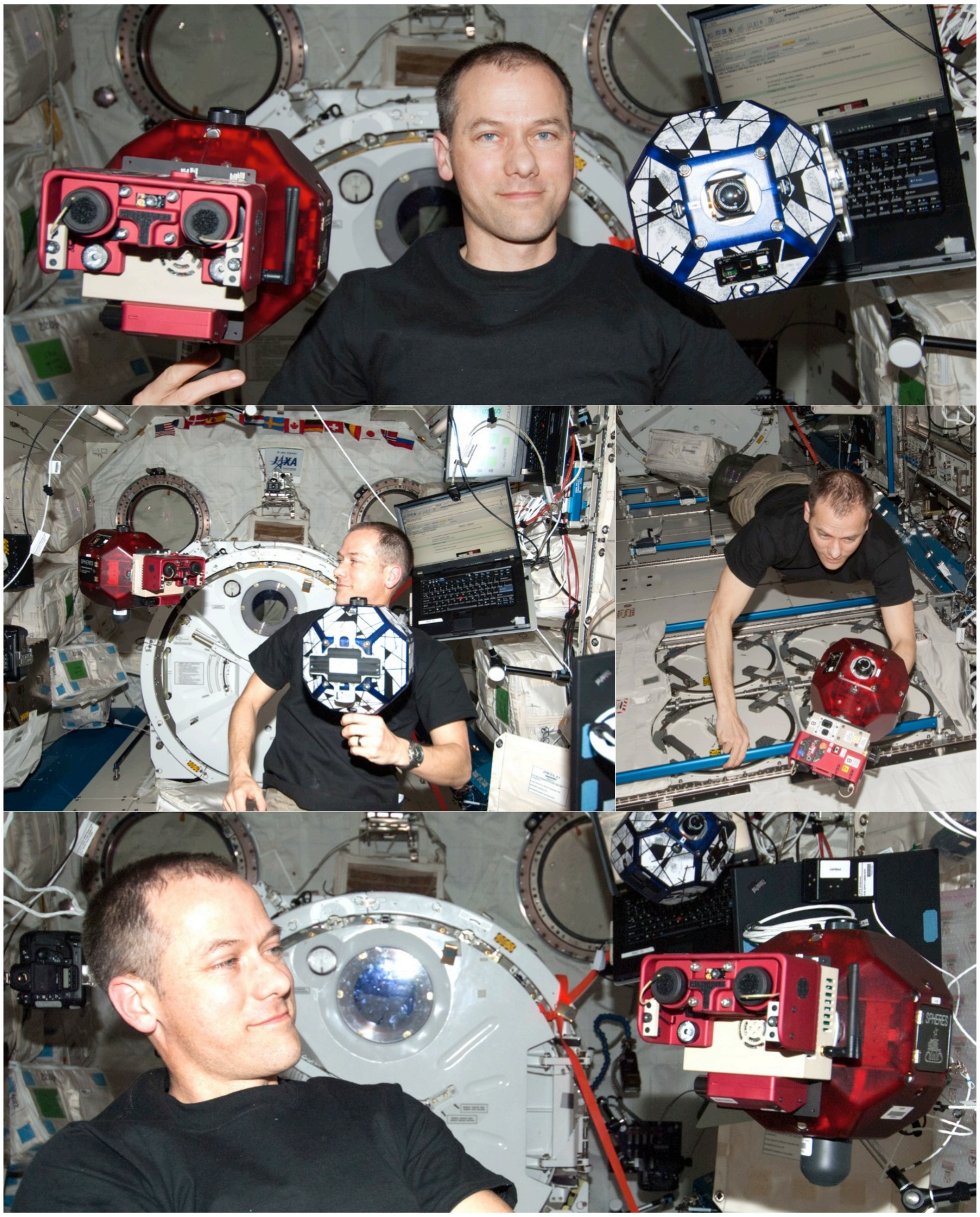
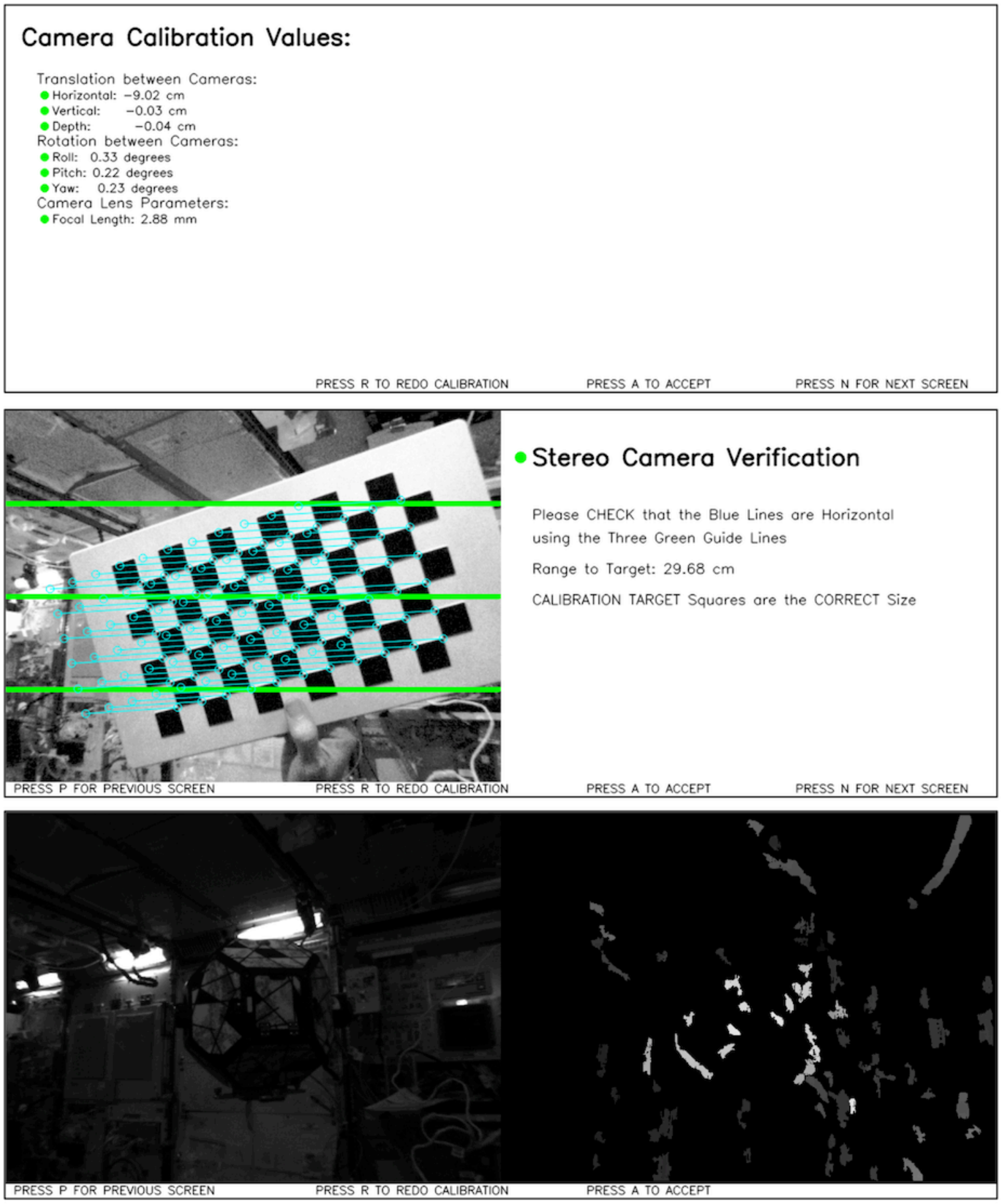


Figure 5-28: First Goggles B Checkout by Astronaut Thomas Marshburn on February 26, 2013



Camera Calibration Values:

Translation between Cameras:
- Horizontal: −9.02 cm
- Vertical: −0.03 cm
- Depth: −0.04 cm

Rotation between Cameras:
- Roll: 0.33 degrees
- Pitch: 0.22 degrees
- Yaw: 0.23 degrees

Camera Lens Parameters:
- Focal Length: 2.88 mm

PRESS R TO REDO CALIBRATION PRESS A TO ACCEPT PRESS N FOR NEXT SCREEN

● Stereo Camera Verification

Please CHECK that the Blue Lines are Horizontal using the Three Green Guide Lines

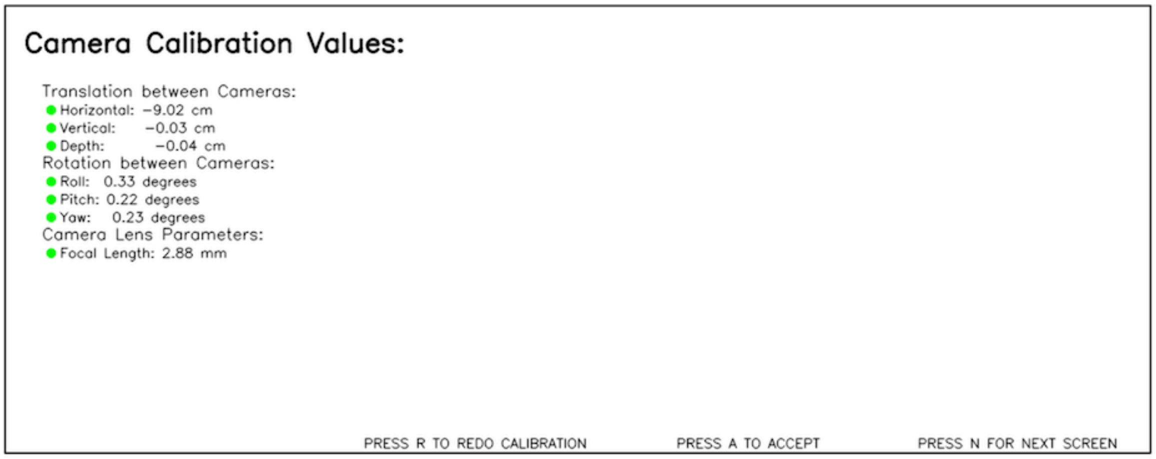
Range to Target: 29.68 cm

CALIBRATION TARGET Squares are the CORRECT Size

PRESS P FOR PREVIOUS SCREEN PRESS R TO REDO CALIBRATION PRESS A TO ACCEPT PRESS N FOR NEXT SCREEN

PRESS P FOR PREVIOUS SCREEN PRESS R TO REDO CALIBRATION PRESS A TO ACCEPT

Figure 5-29: Failed Camera Checkout Results for Test Session 37

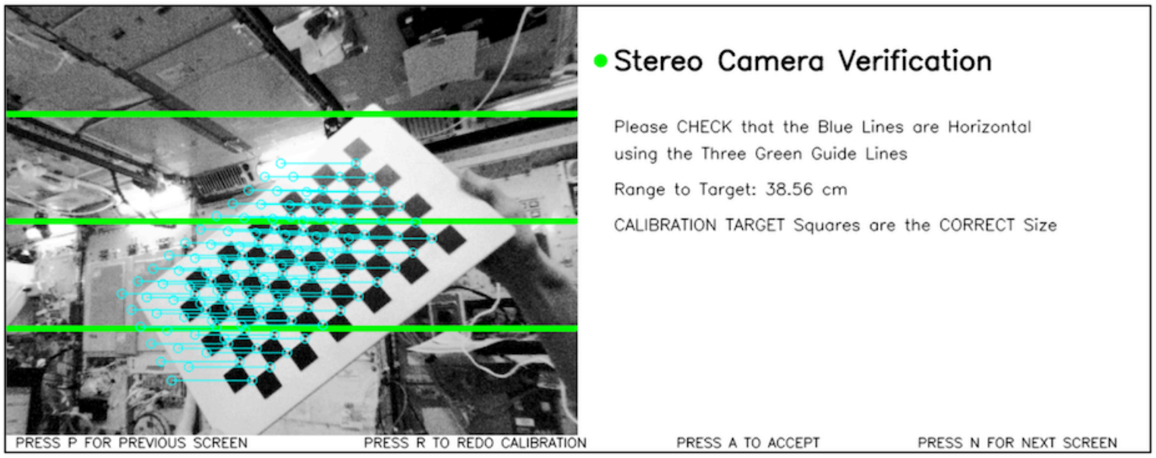


Camera Calibration Values:

Translation between Cameras:
- Horizontal: −9.02 cm
- Vertical: −0.03 cm
- Depth: −0.04 cm

Rotation between Cameras:
- Roll: 0.33 degrees
- Pitch: 0.22 degrees
- Yaw: 0.23 degrees

Camera Lens Parameters:
- Focal Length: 2.88 mm



● Stereo Camera Verification

Please CHECK that the Blue Lines are Horizontal using the Three Green Guide Lines

Range to Target: 38.56 cm

CALIBRATION TARGET Squares are the CORRECT Size

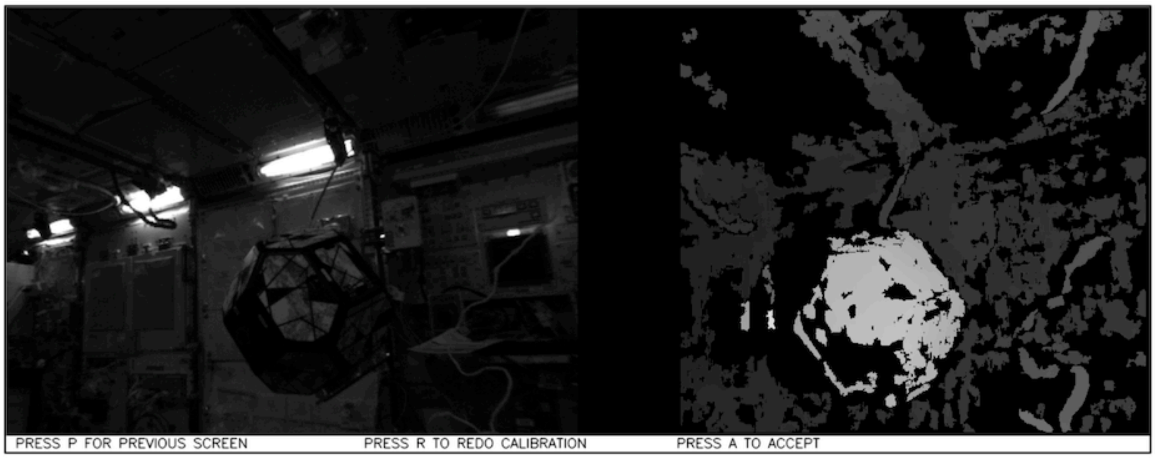


Figure 5-30: Recalibrated and Passed Camera Checkout Results for Test Session 37

5.11.1 Astronaut Feedback on VERTIGO Goggles Operations

After operations were completed and the crew members returned to Earth, Kevin Ford and Thomas Marshburn responded to feedback questions that were prepared by the author of this thesis. The questions and responses for Kevin Ford (summarized by NASA Ames Research Center) are shown in Table 5.6. Thomas Marshburn's responses are shown in Table 5.7. These comments prove that the Goggles were able to be operated by non-experts, however there were some minor areas that could be improved in terms of operations and procedures.

Table 5.6: Crew Feedback: Kevin Ford

Question	**Response**
Was having the PI enabled helpful during the VERTIGO session? Do you think this direct line of communication is advantageous? Would you recommend this for future SPHERES test sessions (not necessarily just VERTIGO sessions)?	Fantastic. Yes, should be enabled during the run, when we are in the nighty-gritty.
Do you think having the PI enabled for the Work Area Setup procedure would have been helpful (the PI was not enabled during that time)? Would you have preferred to have him enabled for the setup procedures?	Stick with PAYCOM for setup. Battery insertion is technique sensitive. Strap order and technique matters. I can explain it during my visit (MIT?). Maybe a short video on board would be appropriate. I showed Tom and he agreed. If we do it wrong, it can cause the doors to pop open.
Any suggestions to make overall operations go smoother?	Procedure, GUI, Test Plan are a lot to manage. We talked about this while I was on board. I made my own cheat sheet on my iPad. Try to consolidate each run into one page with all the parameters. I recommend a 1-page overview for every run with all the parameters in 1 place. I never read the Test Overview except for deployment position and orientation info.
During this last test session you used a new SPHERES GUI. Did you like the additional information available under test control, such as the status, run time and maneuver? Is there other information youd like to see in the GUI, as an operator? How long do you feel it took you to get comfortable with the SPHERES and VERTIGO GUIs? Any suggestions to improve the usability of the GUIs?	It didnt make much difference. I never read the text (Test Overview). I never really used the GUI except to push the required buttons.
Do you feel you understood well the camera checkout and calibration procedure? Any suggestions to make this procedure clearer?	It was trained well on the ground and good refresher on board. I think it worked well for us on orbit.
Do you feel the training, review material, and crew conference were adequate to prepare you for the operations? Any suggestions to improve these elements?	Everything was good. No changes.

Table 5.7: Crew Feedback: Thomas Marshburn

Question	**Response**
You did an excellent job with the camera re-calibration in both VERTIGO sessions. There was some concern amongst the VERTIGO team as to whether or not this procedure would be too operationally challenging for a crew member to perform. Can you comment on what helped you figure this out the best (i.e. the procedures, PI enablement, videos, briefing slides, etc.).	The calibration was not the most challenging. Its fine. Procedures were well written, photos, software, etc. was fine. Good products to support it.
Did you feel more familiar with VERTIGO on your second operation, and if so in what ways?	Yes. Mostly because I knew the pitfallscamera caps. Major reconfigurations should be on test plan. Test overview in GUI not used much by me or Kevin.
Was having the PI enabled helpful during the VERTIGO session? Do you think this direct line of communication is advantageous? Would you recommend this for future SPHERES test sessions (not necessarily just VERTIGO sessions)?	It is essential. It was just great. I enjoyed that interaction. Recommend for all SPHERES sessions.
Any suggestions to make overall VERTIGO operations go smoother?	Give the crew a heads up on Test Session expectations.How far to get through the test plan. We feel bad when cant get through all the tests
During the test sessions you used a new SPHERES GUI. Did you like the additional information available under test control, such as the status, run time and maneuver? Is there other information youd like to see in the GUI, as an operator? How long do you feel it took you to get comfortable with the SPHERES and VERTIGO GUIs? Any suggestions to improve the usability of the GUIs?	I didnt notice the GUI was new. Making the CO2 and battery info real/accurate, would be helpful. We would want to change them out before a run, if needed, so we wouldnt have to repeat the test. Touch screens are great. Cursors are tough in micro-g.
Do you feel the training, review material, and crew conference were adequate to prepare you for the operations? Any suggestions to improve these elements?	The prep was great. Level of prep was greatuse of videos during training was nice. Interested in more info on what software we are testing. Please send me a link. Improvements: need battery doors to stay closed. Velcro on satellites could be improved for temp stowing. It is difficult on the soft rack fronts in the JEM.

5.12 Evaluation of Goggles by Microgravity Design Principles

A set of design principals for microgravity laboratories have been developed by Saenz-Otero and described in his doctoral thesis[113]. The objective of these principals is to: "*guide towards the development of a laboratory environment, supported by facilities, to allow multiple scientists the conduct of research under microgravity conditions, correctly utilizing the resources provided by the ISS, such that they cover a field of study to accomplish technology maturation.*" These can be considered requirements for a scientific payload that is operated onboard the International Space Station. In the following subsections, the methods for which the Goggles meet the principles or requirements is discussed. As will be shown, the VERTIGO Goggles meet all of Saenz-Otero's principles.

5.12.1 Principal of Iterative Research

The Goggles copies the SPHERES approach for iterative research. New science algorithms can be uploaded to both the SPHERES satellites and VERTIGO Goggles through the SPF file, which has a GPF file embedded within it. This is typically uploaded to the ISS two weeks prior to operations, however for the first VERTIGO test session, this was uploaded the day before it was run.

5.12.2 Principle of Enabling a Field of Study

The Goggles science algorithms enables the field of computer vision for robotic spacecraft navigation. It does this by providing a significant amount of software flexibility. Any non-interactive test program that runs as a bash shell command can be executed as a test, which provides for an incredibly large amount of software that can be executed. Since the Goggles runs x86 Ubuntu 10.04 this allows for a significant amount of code reuse (i.e. through third party libraries), which is a key enabler for robotic and computer vision software. The additional capability of installing new software through maintenance scripts (or simply swapping out the Flash Drives on orbit) allows the Goggles software to be upgraded to stay up-to-date with the current state of the art.

5.12.3 Principle of Optimized Utilization

The Goggles use the resources onboard the ISS in an efficient and effective manner. The crew time is one of the most valuable resources onboard the ISS. While it would be more efficient to have the SPHERES and Goggles operate without crew interaction, this interaction is what allows it to be such a flexible payload that enables iterative research. While the SPHERES and Goggles operations and procedures have been refined to minimize the impact on crew time, there are still improvements that can be made. One example is to improve the clarity and intuitiveness of the SPHERES Goggles procedures to further reduce the time for a new astronaut to become proficient in operating the hardware. Another example is to restructure the procedures so that the crew does not need to supervise data downloads. Both of these items are considered for future work.

Power sources are another scarce resource onboard the ISS that must be carefully optimized. The Goggles use batteries that were already onboard the ISS in order to minimize the development and certification time of a new set of batteries. Since the Goggles do not use more than 120 Watt-hours per test session, it is not a significant burden on the ISS resources.

Another scarce resource is data downlink bandwidth. While the Goggles significantly increase the quantity of data that is created during a SPHERES test session, this data is very valuable to the researchers to understand the results of each test. While significant effort has gone into minimizing the data transfer requirements, the Goggles still produce more than 10 GB of data per typical test session. Despite this large volume of data that must be transferred to the Earth, the timing requirements are quite low. It may take weeks for this data to get to the researcher and it will not impact the science results.

The wireless network on the ISS is a resource provided that must be optimized.

The Goggles were initially intended to use wireless 802.11n networking to stream real-time views from the Goggles to the ELC, however this was cancelled due to lack of availability of this network onboard the ISS for payloads to use. During the first run of the visual-inertial navigation algorithm, the lens caps were not removed and the algorithm did not perform as expected. This would have been easily visible in the GUI if wireless networking was enabled and video was streaming to the Flight GUI, however since this was not available it took a number of runs before it was determined that this was the cause of the problems. Since the wireless networking capability onboard the ISS has recently become available, the certification of the VERTIGO Goggles will begin in the near future. It is hoped that the first 802.11n operations will occur in early 2014.

5.12.4 Principle of Focused Modularity

The VERTIGO Goggles allow both the software and hardware to be upgraded through a number of methods. The hardware can be modified by removing the optics mount and optionally replacing it with a different system that interfaces through a electro-mechanical connector providing power and data in common interfaces.

The software can be updated with new GPFs that may include maintenance scripts that install new software. Alternatively, new flash disks can be installed with a completely new operating system. The Goggles can also be networked over the 802.11n network with other science payloads to provide additional functionality.

5.12.5 Principle of Remote Operation and Usability

This was the only principal that was not achieved with the LIIVe Goggles. With the VERTIGO Goggles, a Flight GUI and a set of procedures have been developed so that it can be operated at a remote location by a non-expert. This was achieved on the first three of the VERTIGO test sessions in 2013. The evidence supporting this is that the Goggles were successfully operated three times by two different astronauts. This includes two camera calibrations that were successfully performed by Thomas

Marshburn, who is not an engineer by training, but rather a medical doctor. The fact that a medical doctor was able to perform these tasks, and provided mostly positive feedback in Table 5.7 indicates that the VERTIGO Goggles achieves this principle.

5.12.6 Principle of Incremental Technology Maturation

The VERTIGO Goggles provides a relevant environment to test computer vision-based spacecraft navigation technologies within a micro-gravity environment. While it is acknowledged that the interior of the ISS is not a relevant for in-space lighting and reflectance properties, the full process of incremental technology maturation is now possible through the use of both ground testbeds for realistic lighting conditions along with the ISS laboratory for microgravity and dynamics research.

5.12.7 Principle of Requirements Balance

While there is no hard metric to determine whether or not the requirements are perfectly balanced, the previous discussions in this chapter highlight the rationale for each of the Goggles design. Due to the generality and flexibility of the software implementation, it provides the ability to investigate a field of research with significant depth as well as the breadth to span multiple research fields. This methodical approach leads to a balanced approach to the Goggles design. In addition, there was no single requirement that overly impacted the design more than any others.

Chapter 6

Experimental Results from the International Space Station's Microgravity Environment

This chapter presents the results of the algorithm described in Chapter 4 being applied to a dataset gathered by the SPHERES VERTIGO Goggles. The SPHERES Ultrasonic and Gyroscope based Global Metrology System is used as a reference reference for comparison with the new algorithm's estimated values. This chapter presents one of the primary contributions of this thesis: the implementation and evaluation of the algorithm previously described in this thesis with a dataset that was obtained in a micro-gravity environment using the SPHERES and the Goggles. This chapter begins with a description of the test and a review of the reference measurements. It discusses the specific approach for data association, and describes the selected gain values. The estimated values are compared with the reference measurements and statistics of the differences are presented. The dynamics and inertia properties as well as the three dimensional geometric model are compared against the known values of the SPHERES target satellite. Lastly a discussion of the convergence and computational properties of the dynamic iSAM algorithm is presented.

6.1 Data Collection of Open Loop Intermediate Axis Spin from SPHERES ISS Test Session 38

On March 12, 2013 the SPHERES satellites with the VERTIGO Goggles were operated by NASA Astronaut Dr. Kevin A. Ford, and a dataset was collected that is used for validating the algorithms described in this thesis.

The setup of the this test is shown in Figure 6-1. A view from the port camera looking in the starboard direction is shown in Figure 6-2, with an inlaid view of the starboard camera looking towards port. The Primary SPHERE has the Goggles (B) attached and is the inspector satellite. The secondary SPHERE is acting as the unknown, uncooperative and spinning target. Both SPHERES have the textured stickers applied to their surfaces.

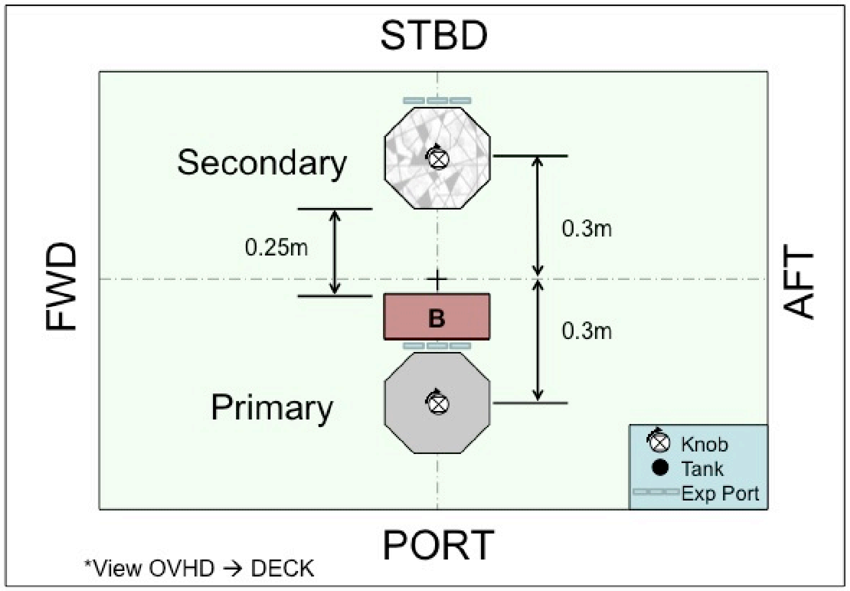


Figure 6-1: SPHERES and Goggles Initial Setup and Configuration

The purpose of this test (SPHERES Test #2 in Test Session #38) was to record video data of an unstable spin about an intermediate axis. The inspector SPHERE used the global metrology system to maintain a fixed position and orientation with the cameras pointing towards the target object. The target object performed closed loop

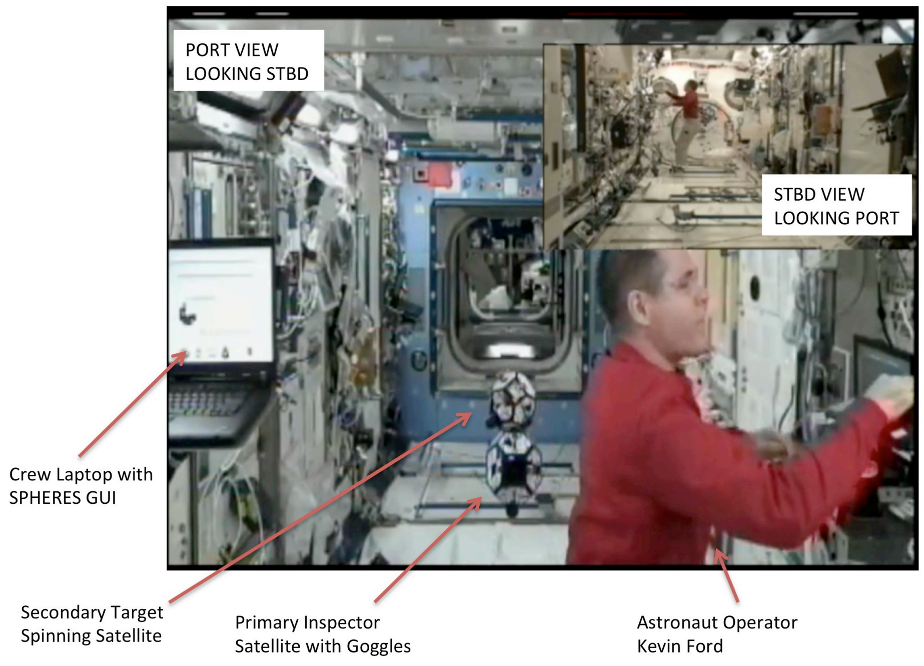


Figure 6-2: SPHERES and Goggles Initial Setup and Configuration

control to spin itself up to 10 rotations per minute (RPM) about the axis that was believed to be the intermediate axis. Based on prior data [30] it was concluded that the SPHERES y-axis (the axis through the battery doors of the satellite) would be the intermediate axis. After actively maintaining this angular velocity for 30 seconds, the satellite stops all thrusting and actuation and enters a free spin for the remainder of the test.

During this time, there should be an instability in the angular velocity vector with respect to the body fixed frame and the satellite should start flipping in order to maintain the conservation of angular momentum as described in Section 2.5 and shown in Figure 2-6.

Throughout the entire test the Goggles on the primary SPHERES satellite was capturing and storing video at 10 frames per second, which was later sent via a downlink to Earth, along with other telemetry and data logs that occurred during this test.

As was discussed in Section 3.3, there are a number of cases where it is difficult to observe the inertial properties such as the center of mass, principal axes and ratios of inertia. The SPHERES satellites are a challenging case due to the fact that their principal moments of inertia are quite close. However, the spin about the unstable, minor moment of inertia was chosen to make the inertia properties as observable as possible, by allowing the exact time constants of the unstable “flip” to be visible.

6.2 SPHERES Metrology Motion Estimates

The SPHERES Global Metrology system was used as the reference measurement system[102, 101]. It uses an ultrasonic time of flight system to measure position, linear velocity and orientation. A process model in incorporated into the estimation process using an Extended Kalman Filter. The onboard gyroscopes are used to measure angular velocity, but are not fused with the ultrasonic measurement system. The SPHERES Global Metrology system is known to produce estimates that are repeatable within two millimeters for position and one degree for orientation. A detailed study on the accuracy has not been performed, but experience indicates that the accuracy is likely within 10 millimeters for position and three degrees for orientation[101, 102]. Note that global metrology system estimates states in a right-handed reference frame fixed to the interior of the International Space Station where the Forward direction is positive X, the Starboard direction is positive Y and the Deck direction is positive Z (see Figure 6-1).

6.2.1 Target Object Reference Metrology: Secondary SPHERES

The Global Metrology data for the secondary SPHERES satellite (i.e. the spinning target object) is shown in the following figures. Figures 6-3, 6-4, 6-5 and 6-6 show the position, linear velocity, quaternion and angular velocity of the target SPHERES satellite with respect to the Global Metrology reference frame.

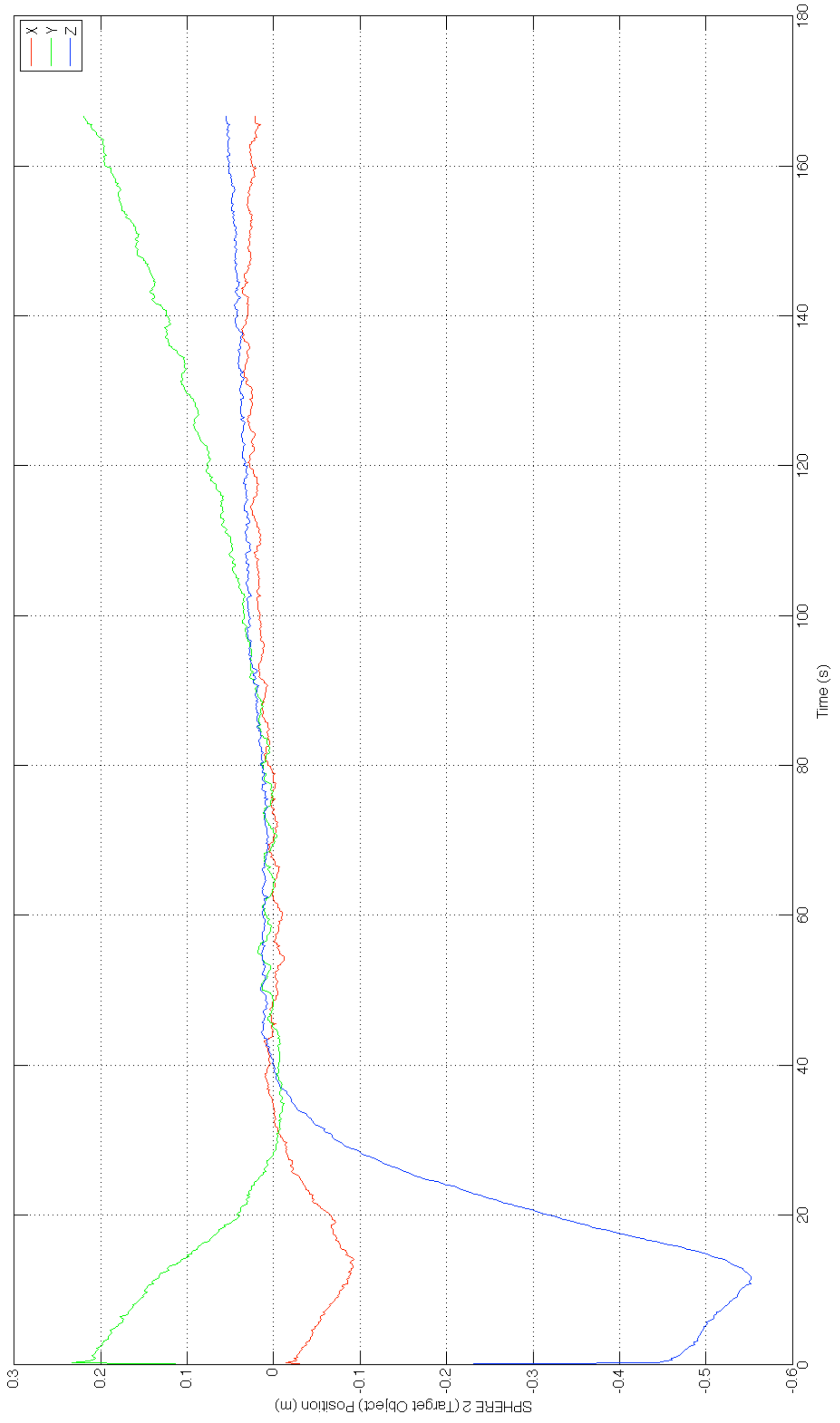


Figure 6-3: Global Metrology (Ultrasonic) Measurement of Target SPHERES' Position

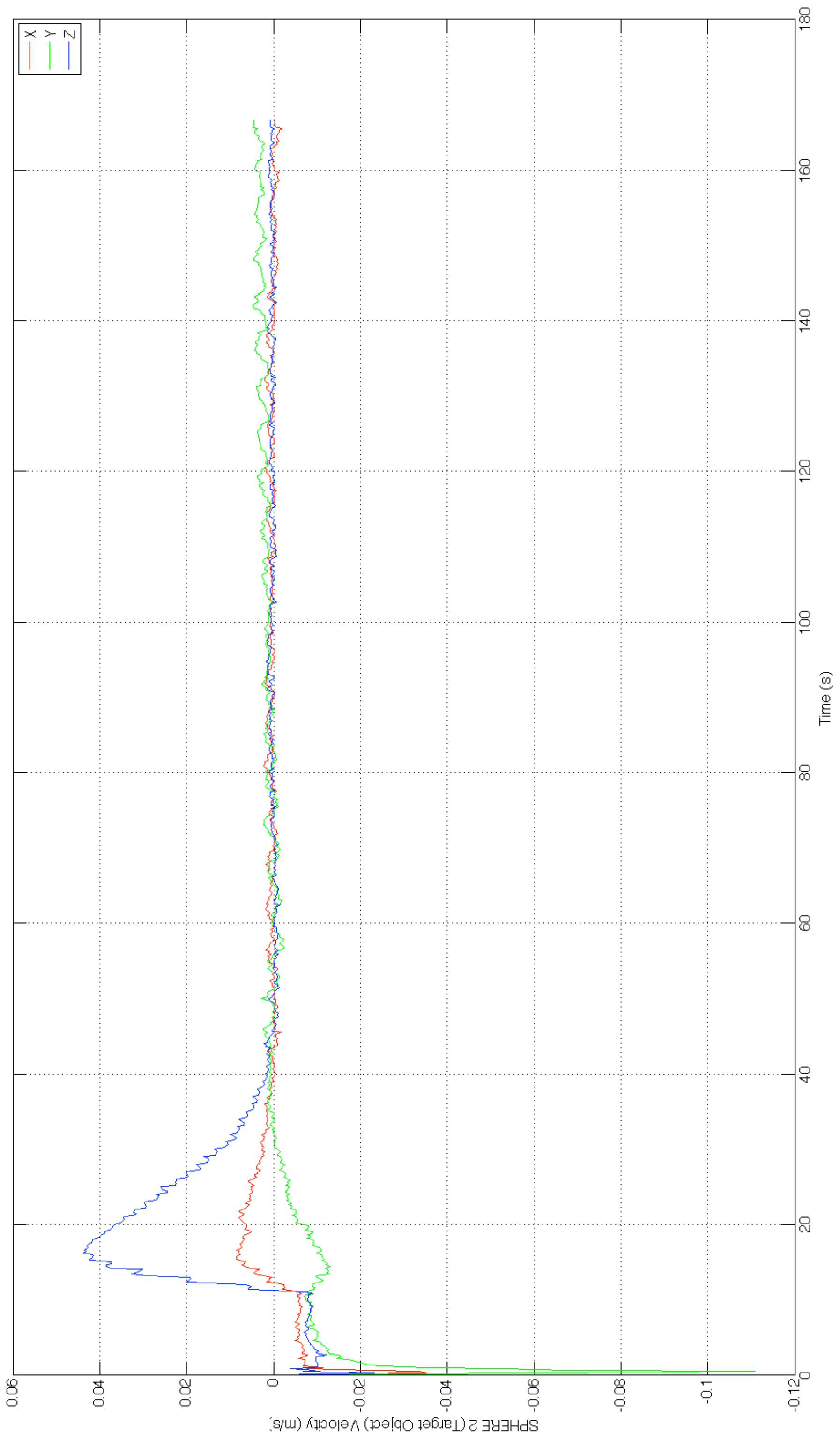


Figure 6-4: Global Metrology (Ultrasonic) Measurement of Target SPHERES' Linear Velocity

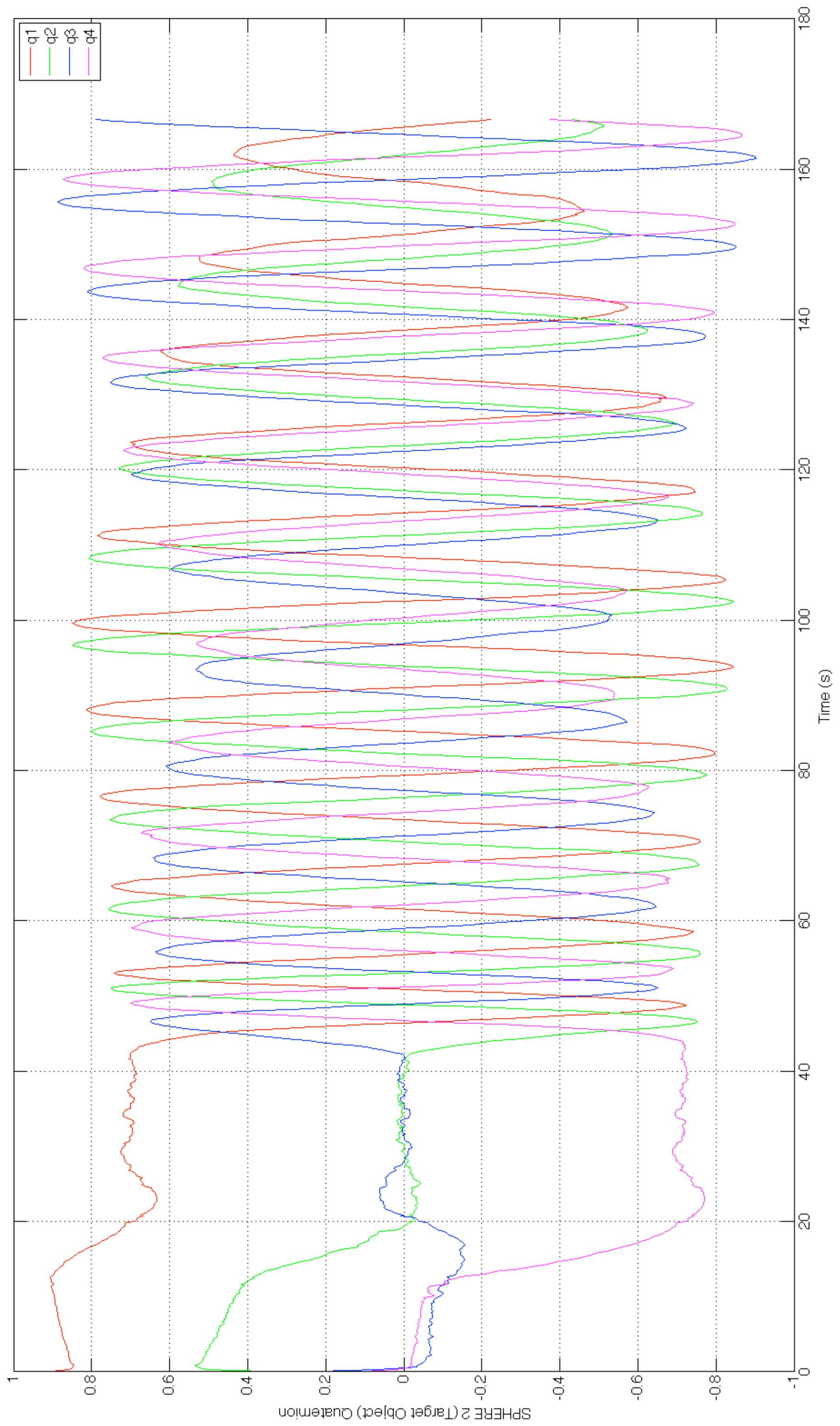


Figure 6-5: Global Metrology (Ultrasonic) Measurement of Target SPHERES' Orientation

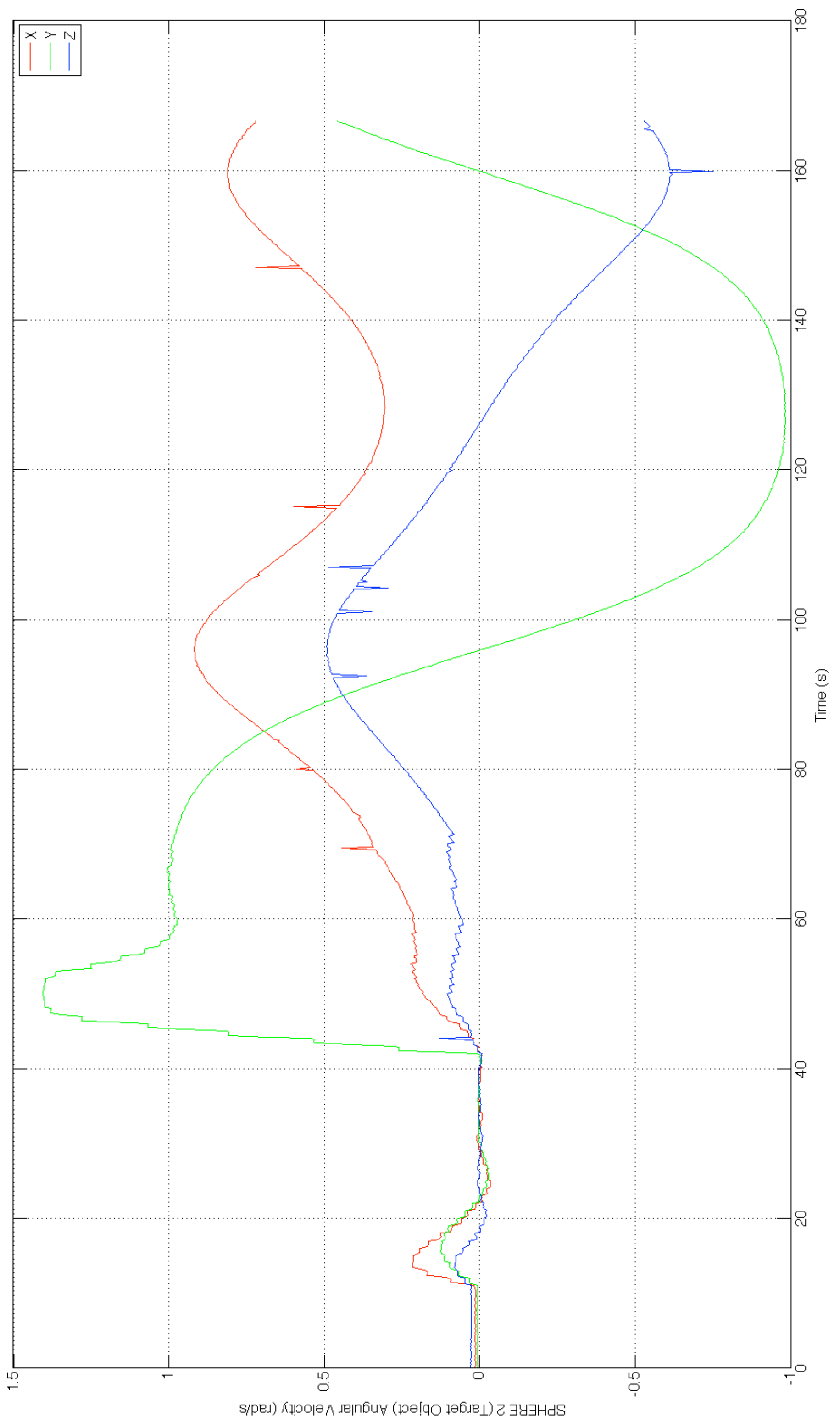


Figure 6-6: Global Metrology (Gyroscope) Measurement of Target SPHERES' Angular Velocity

The different phases or "maneuvers" in the test can be seen best by looking at the angular velocity in Figure 6-6. For the first 10 seconds of the test, no actuation occurs while the global metrology estimator is allowed to converge. Between 10 seconds and 40 seconds the satellites move into their initial positions and orientations. Starting at 40 seconds the target object begins to spin up to a desired angular velocity vector of $\omega = [0.5, 10, 0.5]$ RPM. It finishes its initial spin up at 75 seconds and begins freely spinning with no applied forces or torques. Note that the control to achieve the initial angular velocity has significant overshoot on the y-axis and does not actually achieve the desired value on the x and y axis. Despite this, the unstable, periodic spinning motion about an intermediate axis is clearly apparent between 75 seconds and 175 seconds. Figure 6-5 shows the quaternions that were estimated by the global metrology ultrasonic system. It shows that despite being measured by an independent sensor, the quaternion estimate did not diverge at such high angular velocities. One adjustment was made to the raw quaternion data: Since the SPHERES satellites always store quaternions with the four (scalar) element positive, there are frequently discontinuous jumps in the quaternions. In Figure 6-5 this was corrected in post-processing. Figure 6-3 and 6-4 show the position in the global metrology frame. Note that beginning at approximately 100 seconds into the test the target SPHERES satellite begins drifting and moves approximately 20 centimeters over a one minute period. This is approximately 3 millimeters per second, which is confirmed in Figure 6-4.

6.2.2 Mass, Inertia and Kinetic Energy

Previous studies on the SPHERES satellites have undertaken considerable effort to estimate the mass and inertia properties of the SPHERES satellites. These properties changed recently in 2012 when new expansion ports were launched and attached to the SPHERES satellites. Eslinger's Masters' thesis provides an analysis and characterization of the new mass properties based on previous data and new results that were taken on February 26, 2013[30].

Eslinger estimates the mass of the target sphere to be 4.487 kg with a standard deviation of 0.0567 kg. Additionally, Eslinger estimates the following inertia matrix for the target satellite:

$$\mathbf{J} = \begin{bmatrix} 2.41\times10^{-2} & -1.30\times10^{-4} & -1.42\times10^{-4} \\ -1.30\times10^{-4} & 2.34\times10^{-2} & 5.74\times10^{-5} \\ -1.42\times10^{-4} & 5.74\times10^{-5} & 2.01\times10^{-2} \end{bmatrix} \text{kg m}^2 \tag{6.1}$$

The diagonalization of this inertia matrix is (dropping the units for convenience):

$$\mathbf{J} = \mathbf{R}\mathbf{J}_{\text{diag}}\mathbf{R}^T \tag{6.2}$$

$$= \mathbf{R}\begin{bmatrix} 0.0241 & 0 & 0 \\ 0 & 0.0234 & 0 \\ 0 & 0 & 0.0201 \end{bmatrix}\mathbf{R}^T \tag{6.3}$$

$$\mathbf{R} = \begin{bmatrix} 0.9833 & 0.1787 & 0.0349 \\ -0.1783 & 0.9838 & -0.0160 \\ -0.0372 & 0.0095 & 0.9993 \end{bmatrix} \tag{6.4}$$

The axis angle representation of this inertia matrix is given in Equation 6.5 and 6.6. Note that this is roughly negative ten degrees about the z-axis (the CO2 tank axis).

$$\mathbf{n} = [-0.0697, -0.1975, 0.9778]^T \tag{6.5}$$

$$\theta = 10.52 \text{ degrees} \tag{6.6}$$

Additionally, the center of mass, $\mathbf{T}_{\text{PA/SPH}}$, was estimated by Eslinger, and the entire transformation from the principal axes (PA) to the SPHERES Geometric frame (SPH) can be summarized as follows:

$$\begin{aligned}\mathbf{p}_{\text{SPH}} &= \mathbf{R}_{\text{SPH/PA}}\mathbf{p}_{\text{PA}} + \mathbf{T}_{\text{PA/SPH}} && (6.7)\\ \mathbf{R}_{\text{SPH/PA}} &= \begin{bmatrix} 0.9833 & 0.1787 & 0.0349 \\ -0.1783 & 0.9838 & -0.0160 \\ -0.0372 & 0.0095 & 0.9993 \end{bmatrix} && (6.8)\\ \mathbf{T}_{\text{PA/SPH}} &= \begin{bmatrix} 3.883.00 \times 10^{-3} \\ -1.49 \\ 1.923.00 \times 10^{-3} \end{bmatrix} \text{meters} && (6.9)\end{aligned}$$

Now, using these properties, the translational and rotational kinetic energy can be computed and graphed over time. Figure 6-7 shows the rotational energy of the target object over time. The dashed blue line shows that the value of the rotational kinetic energy that was computed at 75 seconds and is plotted as a reference for going forward in time. This can be compared to the red line that shows the actual measured rotational kinetic energy. This figure shows that after 75 seconds, the rotational kinetic energy remains almost constant over time. However, there is a slight decrease that begins just before the 150 second mark. By the end of the test, the kinetic energy is approximately 7.8% less than at the 75 seconds into the test. This is likely due to effects such as wind and air resistance inside the crew volume of the ISS as well as fluid slosh within the SPHERES fuel tanks.

Figure 6-8 shows an equivalent plot of the translational kinetic energy. Again, note that after 75 seconds, the kinetic energy remains virtually constant but has an ever-so-slight increase starting at approximately 120 seconds. This again is likely due to aerodynamic effects such as air currents within the ISS.

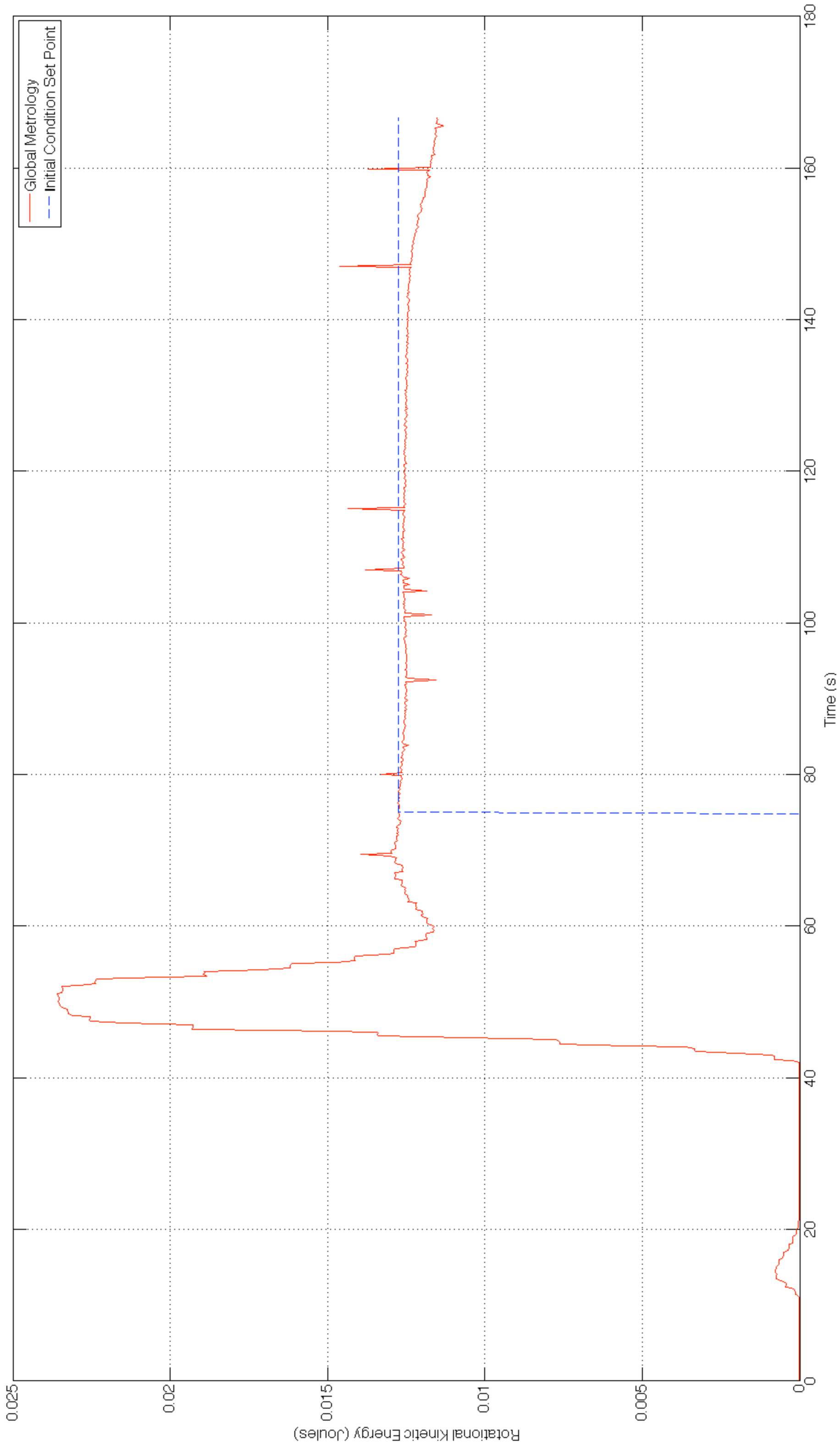


Figure 6-7: Rotational Kinetic Energy of Target SPHERES Satellite

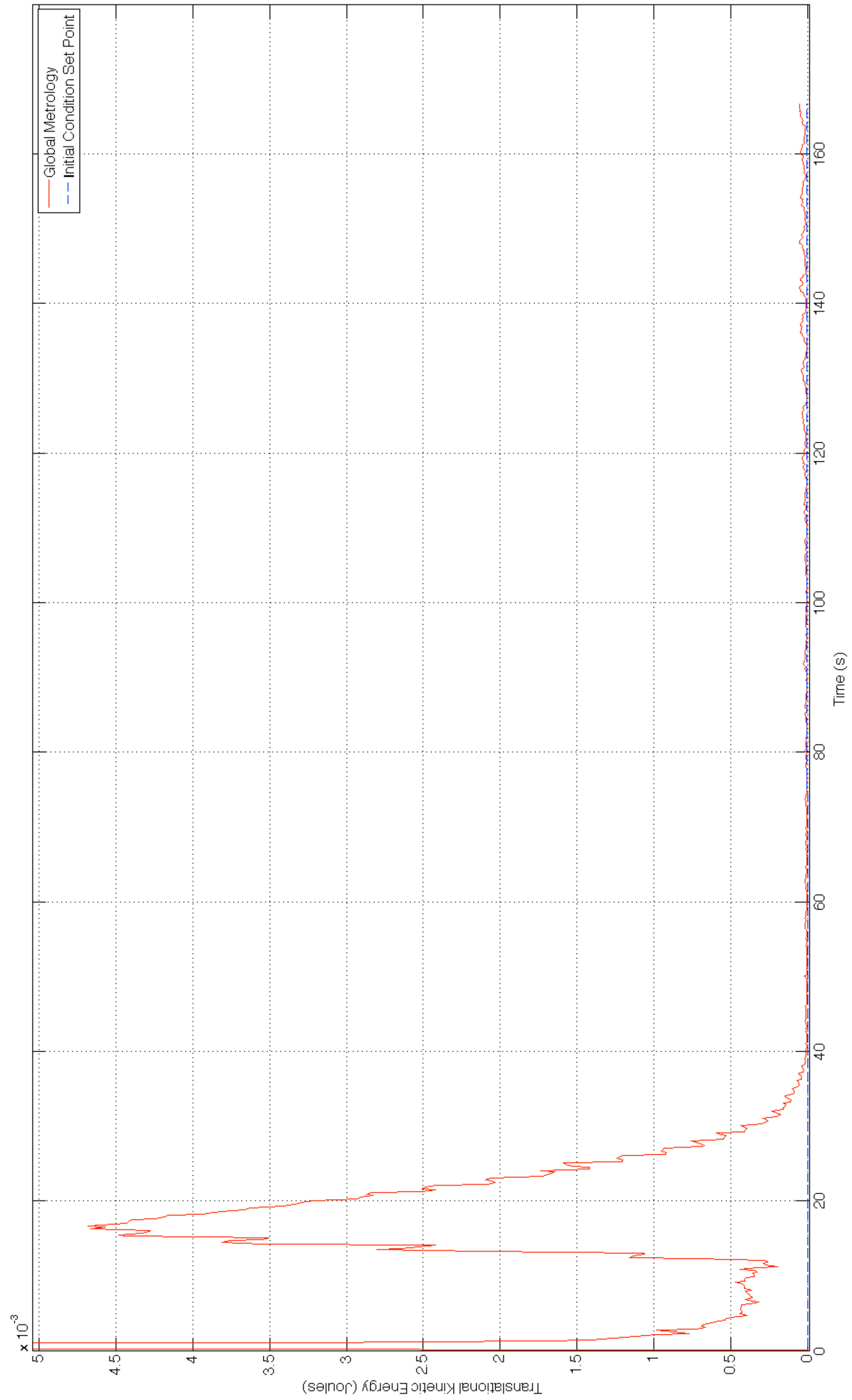


Figure 6-8: Translational Kinetic Energy of Target SPHERES Satellite

6.2.3 Inspector Reference Metrology: Primary SPHERES

Figure 6-9, 6-10, 6-11 and 6-12 respectively show the reference measurements for position, linear velocity, orientation and angular velocity with respect to the global metrology frame. Note that the estimator has been adjusted to provide estimates at the center of mass with the VERTIGO Goggles attached to the satellite. Note that from Figure 6-11 and 6-12, it is clear that the attitude of the satellite was dead-banding (i.e. performing slight oscillations within the minimum impulse bit of the estimation and control system). This is evident in the video data that is shown in Section 6.2.5. This is an important fact to note, because this motion is not accounted for in the algorithm evaluated in this thesis. This algorithm assumes the inspector is perfectly stationary.

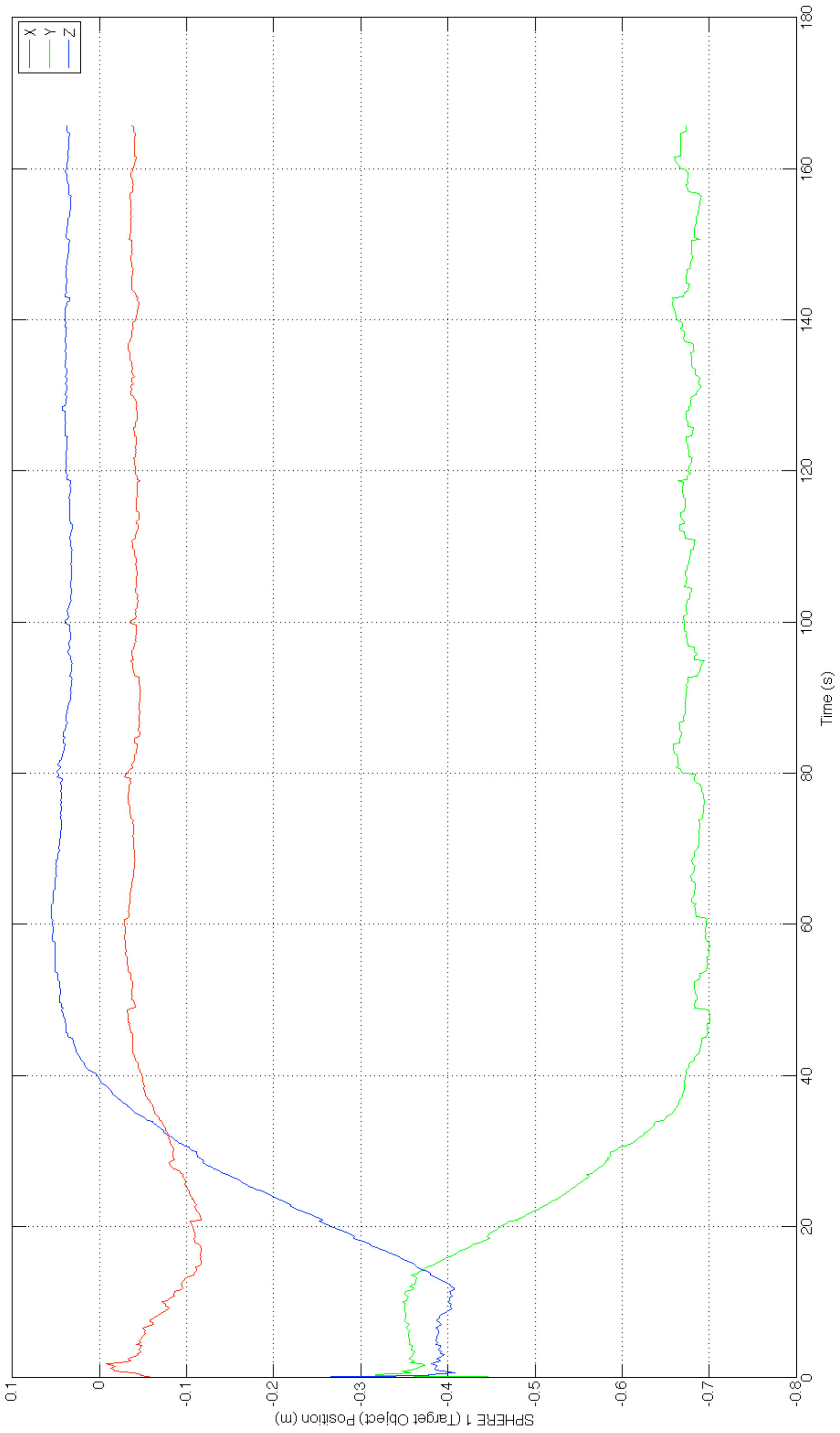


Figure 6-9: Global Metrology (Ultrasonic) Measurement of Inspector SPHERES' Position

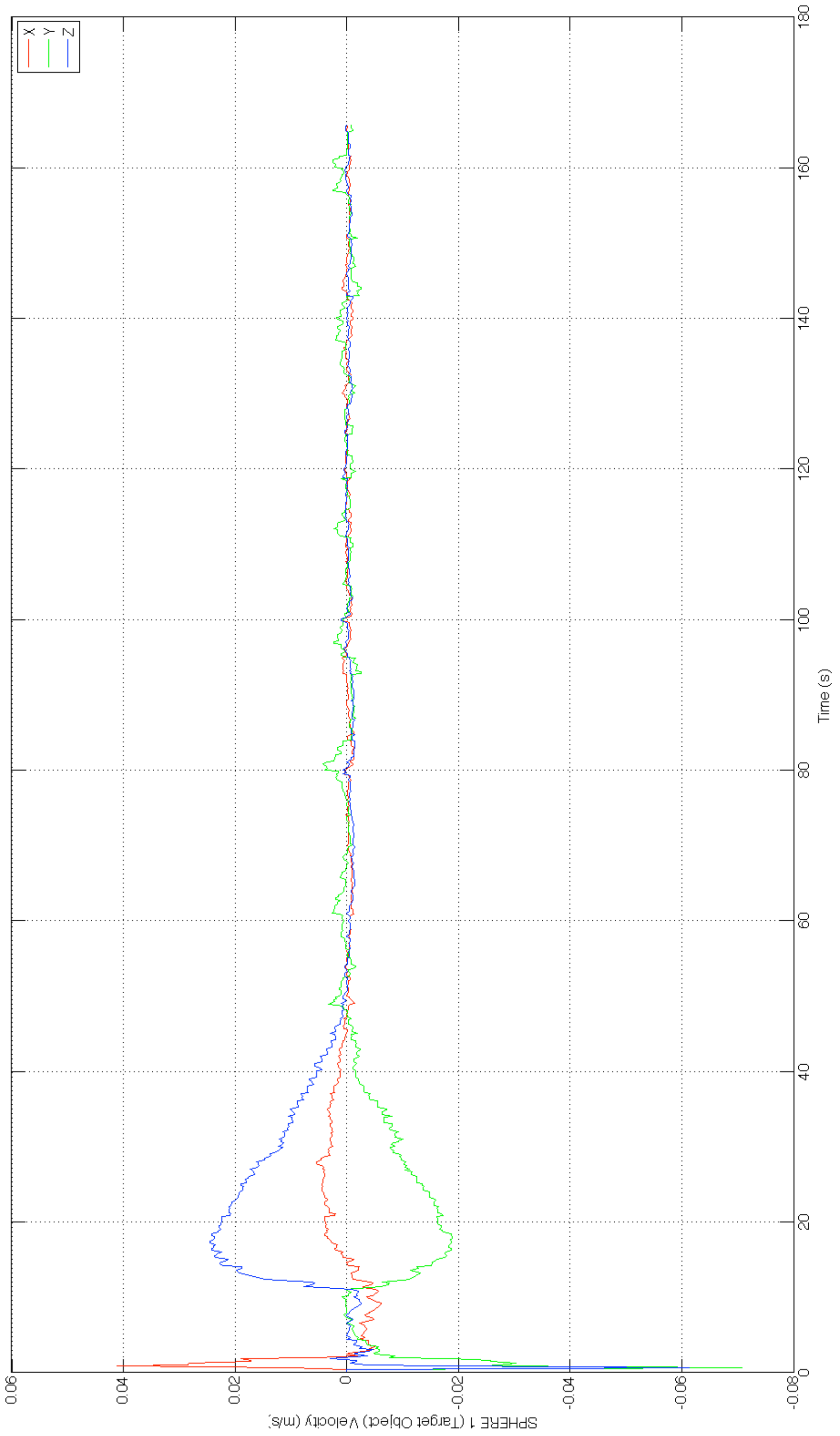


Figure 6-10: Global Metrology (Ultrasonic) Measurement of Inspector SPHERES' Linear Velocity

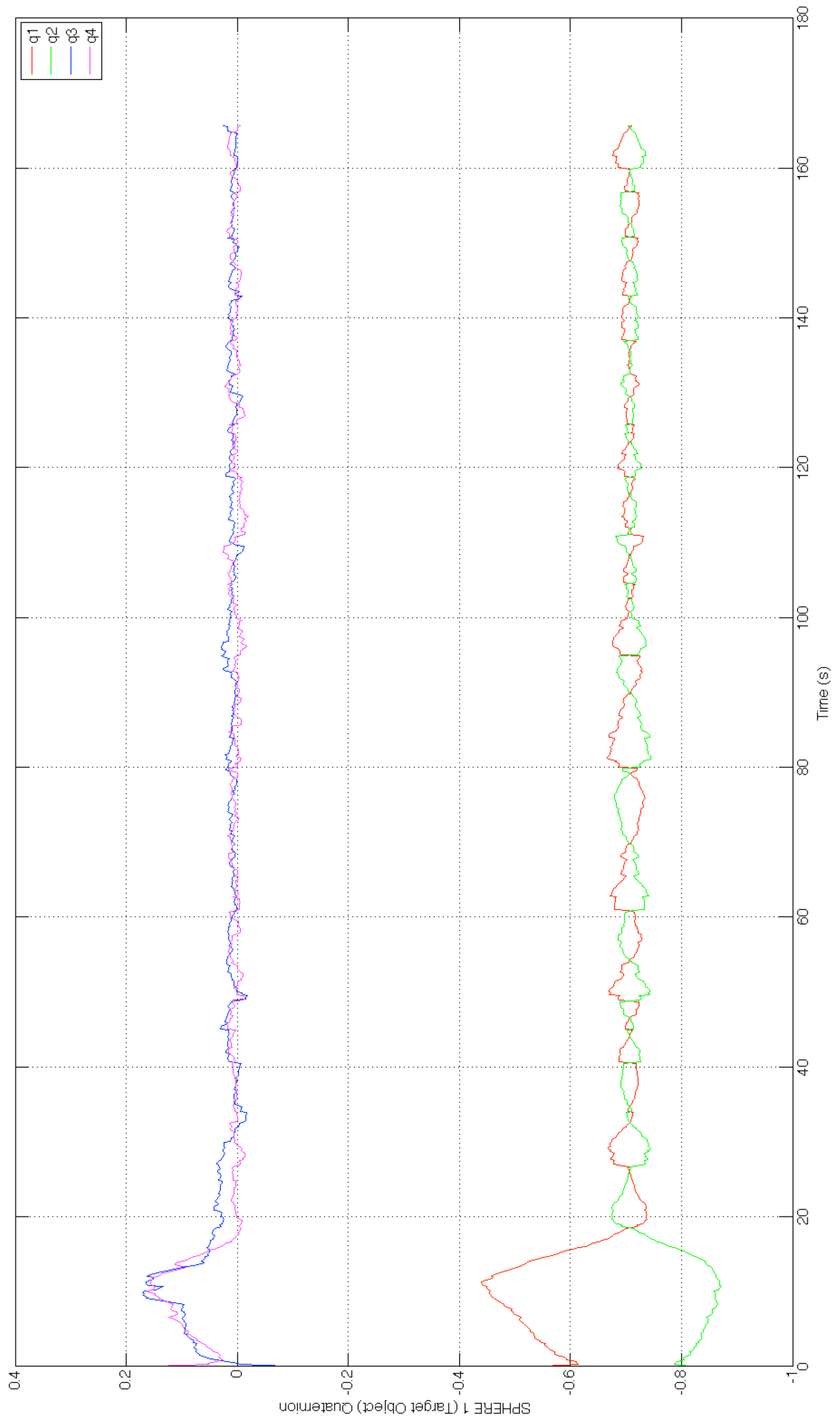


Figure 6-11: Global Metrology (Ultrasonic) Measurement of Inspector SPHERES' Orientation

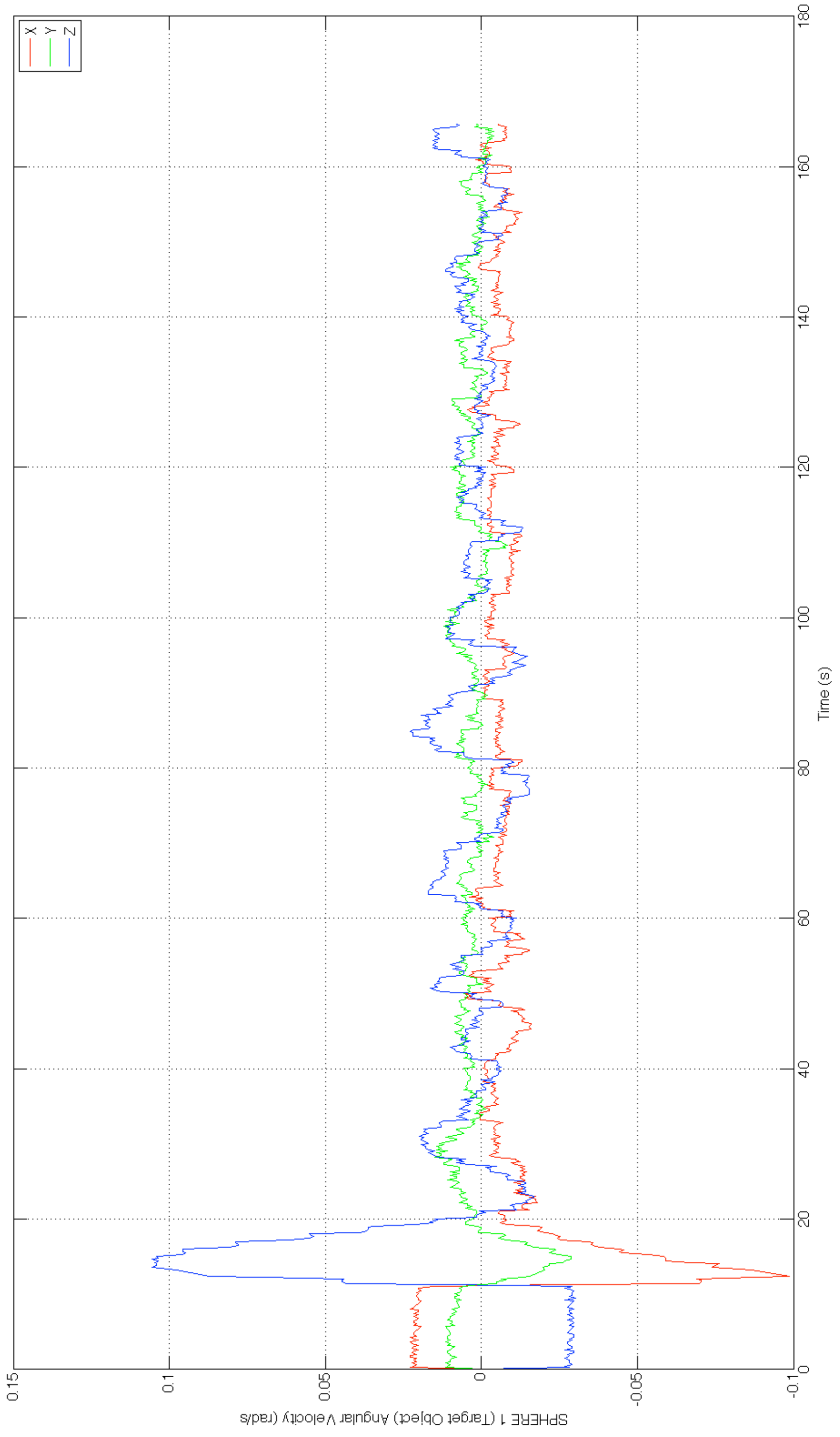


Figure 6-12: Global Metrology (Gyroscope) Measurement of Inspector SPHERES' Angular Velocity

6.2.4 Relative Reference Metrology

The reference motion estimates of the primary and secondary SPHERES satellites with respect to the global or inertial reference frame were described in Section 6.2.3 and 6.2.1 respectively. Since the algorithm described in this thesis assumes the camera frame is stationary, any actual motions of the camera frame as measured by the reference system must be appropriately accounted for to determine the correct relative position and orientation. The relative position and velocity of the target object with respect to the camera frame is $\mathbf{r}_{T/C}$ and is computed as follows:

$$\mathbf{r}_{T/C} = \mathbf{R}_{C/I}(\mathbf{r}_{T/I} - \mathbf{r}_{C/I}) \tag{6.10}$$

The SPHERES reference angular velocity is ω_C and is measured by the inspector SPHERES satellite's onboard gyroscopes. It is used to compute the relative velocity $\mathbf{v}_{T/C}$ as follows. Both $\mathbf{r}_{T/C}$ and $\mathbf{v}_{T/C}$ are plotted in Figure 6-13.

$$\mathbf{v}_{T/C} = \mathbf{R}_{C/I}(v_{T/I} - v_{C/I}) - \omega_C \times \mathbf{r}_{T/C} \tag{6.11}$$

It is interesting to see that there is distinct oscillations in the Y and Z axis position and velocity in Figure 6-13, despite the fact that the position and orientation in each satellites Global Metrology estimates do not have these oscillations. This is due to the dead-banding in the control of the inspector satellite's position and attitude as discussed in Section 6.2.3, which is evident in both the Global Metrology estimates and the onboard video taken by the Goggles.

Since the dynamic iSAM algorithm presented in this thesis makes the assumption that the inspector is stationary, there were four options for how to deal with this. The first option is to modify the images using "image stabilization" techniques so that they appear to be taken from a perfectly still location. This was considered outside the scope of this thesis. The second option is to assume the inspector's body frame is static, and any motion will appear as disturbance forces applied to the target object. This is what was done in this thesis, since the disturbances are small and it helps determine how robust the algorithm presented in this thesis is to disturbances. The third option is to modify the camera frame measurements so that they were taken in the inertial (or Global Metrology) reference frame. Although this may be a preferable engineering approach, it was not chosen so that the reference metrology estimate was not coupled to the estimator presented in this algorithm, and could thereby allow a completely independent comparison. A fourth option is to add a Markov chain of poses for the Inspector trajectory to the factor graph and estimate these with either the Global Metrology system or the IMUs.

The SPHERES satellite provides estimates with respect to its geometric coordinate frame while the dynamic localization and mapping method provides estimates with respect to the principal axis frame. In order to provide an equivalent basis for comparison, all of the quaternions from the global metrology estimate of the target satellite had the initial quaternion subtracted.

$$\mathbf{q}_T(t) \quad = \quad \mathbf{q}_{T/I}(t) \otimes \mathbf{q}_{T/I}(0)^{-1} \tag{6.12}$$

Now the reference angular velocity of the target is ω_T which is taken directly from the global metrology gyro measurements. For an additional point of verification, a first order difference of the quaternion was computed to verify that the quaternion and angular velocity were in fact related by Equation 2.41.

$$\begin{bmatrix} \mathbf{w}_{\text{Diff}} \\ 0 \end{bmatrix} = 2 \begin{bmatrix} q_4 & q_3 & -q_2 & -q_1 \\ -q_3 & q_4 & q_1 & -q_2 \\ q_2 & -q_1 & q_4 & -q_3 \\ q_1 & q_2 & q_3 & q_4 \end{bmatrix} \frac{\mathbf{q}[k] - \mathbf{q}[k-1]}{\Delta t} \tag{6.13}$$

The orientation, $\mathbf{q}_T(t)$, its numerical derivative, and the angular velocity ω_T as measured by the gyroscopes are shown in Figure 6-14.

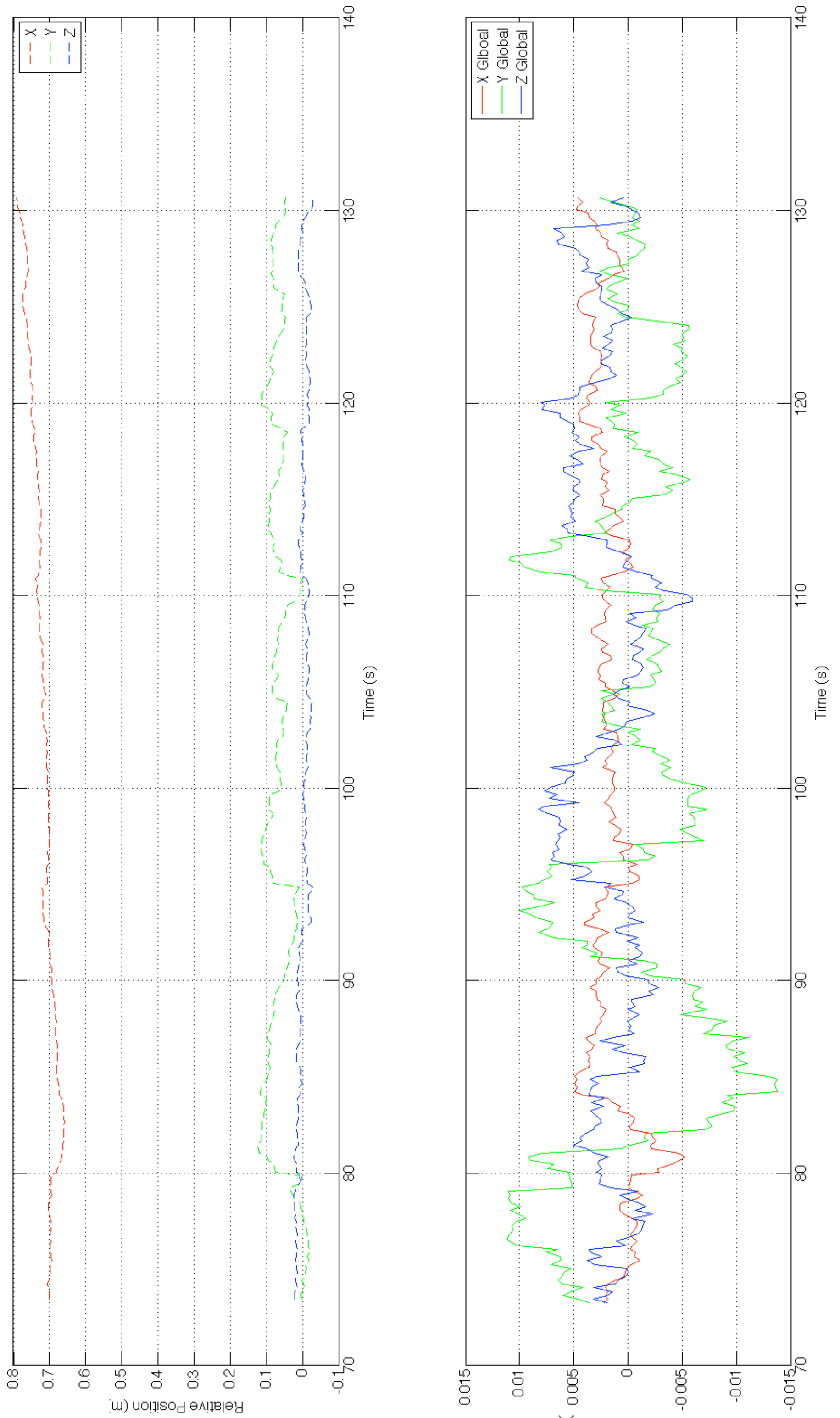


Figure 6-13: Relative Global Metrology Reference Position and Linear Velocity with repsect to the Camera Frame

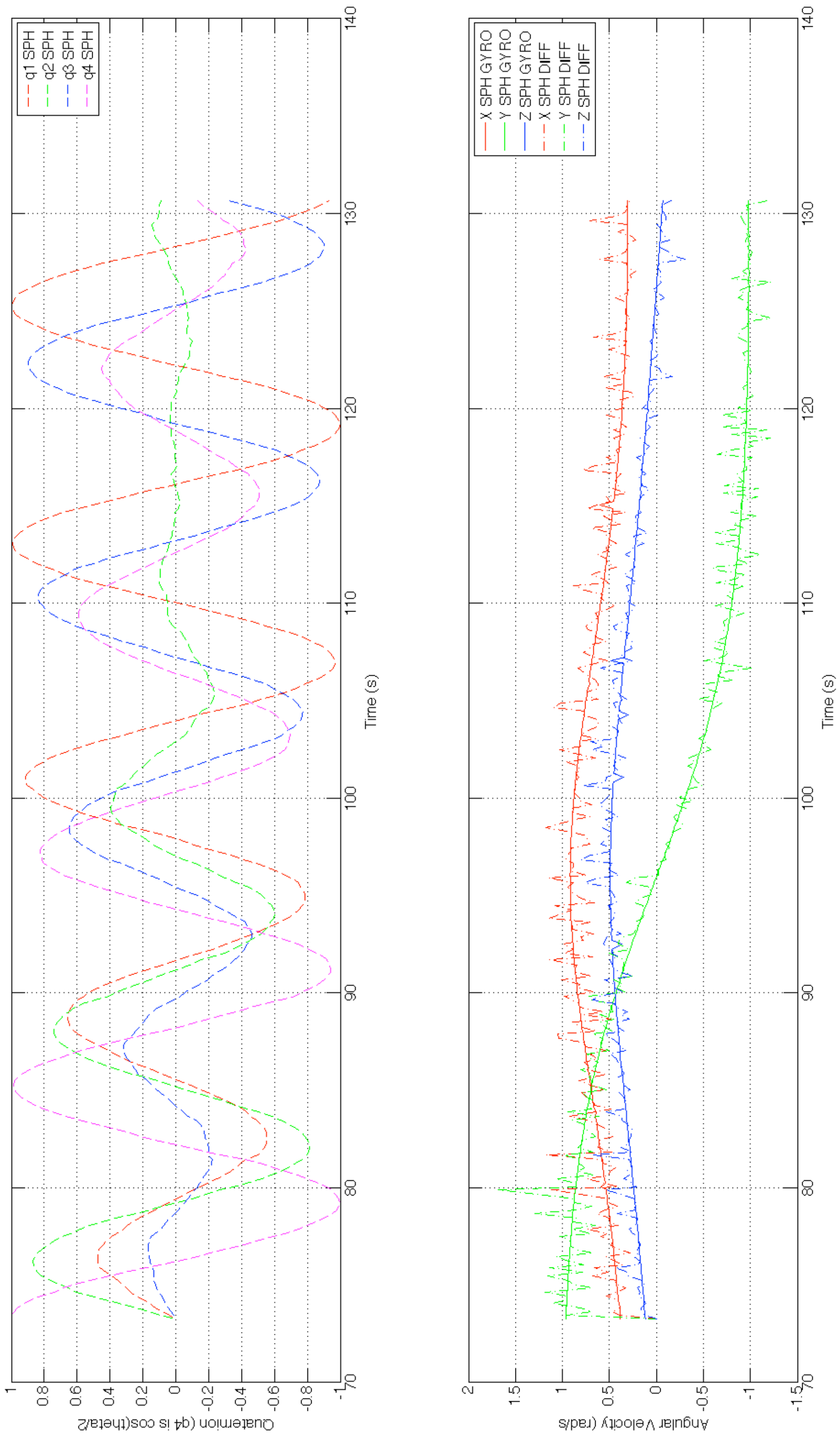


Figure 6-14: Relative Global Metrology Reference Orientation and Angular Velocity with respect to the Camera Frame

6.2.5 Stereo Image Data of Tumbling Satellite from SPHERES VERTIGO Goggles

Figure 6-15 shows four stereo pairs of images from the data set taken during Test #2 of SPHERES ISS Test Session #38. It shows the stereo images taken by the Goggles mounted to the Primary SPHERES and illustrates the evolution of the unstable spin over time. In the left hand images, the SPHERES body frame axes are labelled with red vectors. These axes are the coordinate frame for the angular velocity vector of the target object shown in Figure 6-6.

Figure 6-15 clearly shows a flipping motion due to the spin about an intermediate axis. Note that in the top figure at $T = 39s$ the y axis is pointing up in the image. At time $T = 75s$ and $T = 91s$ it is clear that the y axis is starting to flip down in the image. By time $T = 111s$ the y axis has completed nearly a 180 degree flip. Again, this is confirmed by the flip in the angular velocity vector in Figure 6-6.

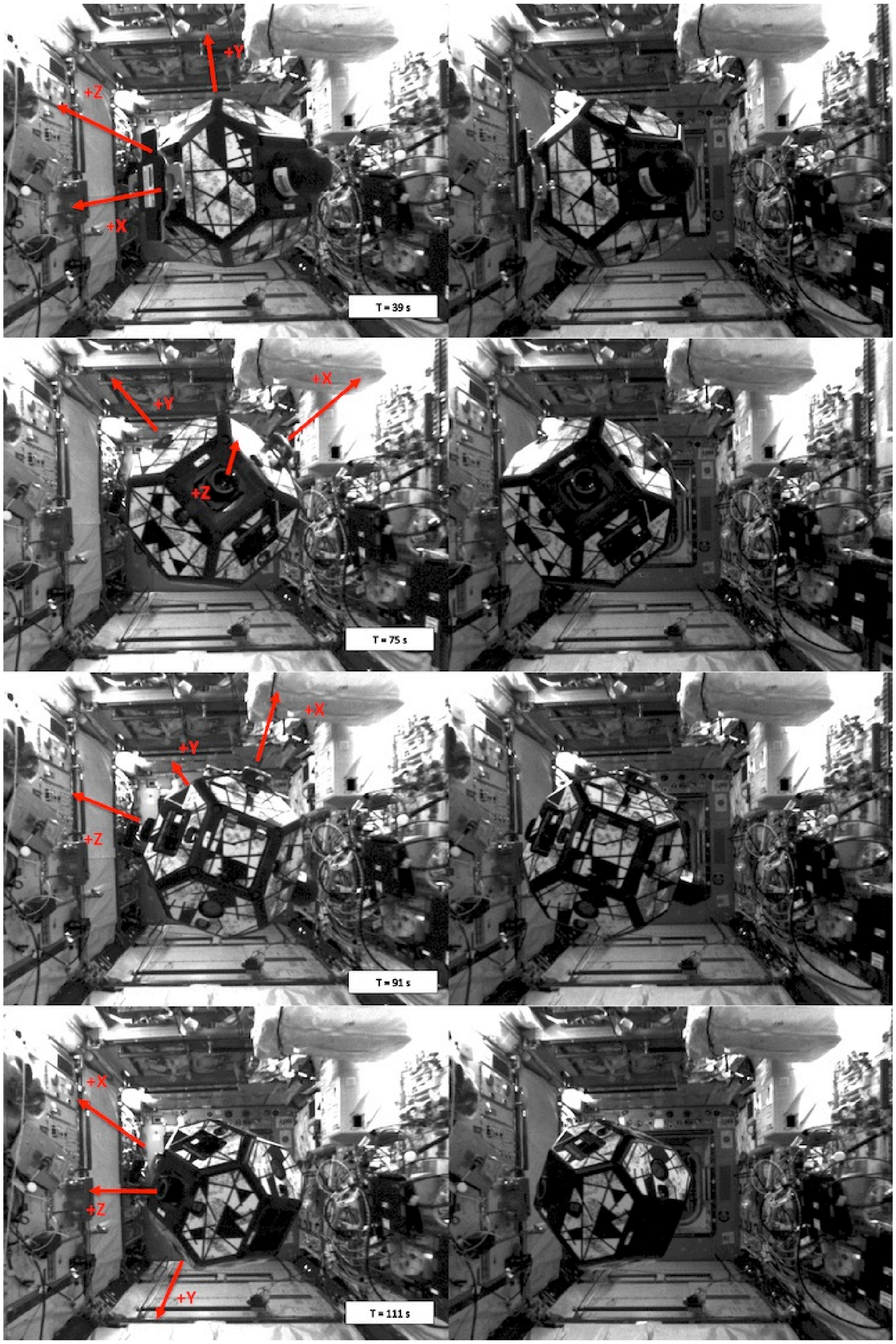


Figure 6-15: Time Lapsed Images Illustrating Unstable Spin about Intermediate Axis

6.3 Data Association and Feature Matching

Data association in the context of perception is a very important aspect of the localization and mapping problem. This involves ensuring that the two dimensional image location in multiple images corresponds to the same three dimensional feature point. If this is done incorrectly it can have a significant adverse affect on the output of the estimation system.

While it makes sense that utilizing the linear and angular velocities of a spinning target would help match features between stereo images of a spinning object taken at different time-steps, this was not implemented for this thesis since it would inherently couple the data association and estimation system. In other words, if there was a bad state estimate, it could lead to a bad data association, which in turn could lead to even more bad state estimates.

Since one of the main purposes of this thesis is to evaluate a new approach for incorporating dynamics into the iSAM optimization system, it was decided to use a known and understood data association system that was previously tested and developed in Muggler's thesis[96].

The first step of the data association process is match features between the left and right stereo images. This is done using OpenCV SURF features as described in Section 2.9. The following step matches features between two frames (pairs of stereo images) taken at two instances in time. The same OpenCV SURF features are triangulated and used with RANSAC and Absolute Orientation as described in Section 2.7, Section 2.9 and illustrated in Figure 2-16. Both of these steps were originally implemented by Muggler and modified slightly for this thesis.

The last step is to compute a table of "global" features. For every feature that was detected by the system, there is a list of frames in which it was visible along with its image coordinates in the left and right stereo image frames (i.e. u_L, v_L, u_R, v_R). This final step was implemented by Muggler as follows: The relative pose computed by the Absolute Orientation algorithm is added to the trajectory to create a visual odometry trajectory over time. This trajectory is optimized in a pose-graph framework with the iSAM engine (and no feature points). Next an iSAM based bundle adjustment approach is run using the visual odometry estimate as its initial conditions. Finally all of the frames are matched to all of the frames, again using the SURF and RANSAC approach previously described, and a global database of features and their corresponding frames and stereo image coordinates is built up. This approach along with the subsequent steps of implementing the estimation system are shown in Algorithm 2.

Figure 2-16 shows a few images that are representative of the entire dataset. The dataset shows excellent matches between the left and right images. Additionally, the frame to frame matches appear to be visually consistent and indicate the correct motion.

Algorithm 2 Overall Data Association and Estimation Process

```
 1: for Image Pair $k$ do
 2:     Raw_Left_Features ← DetectSURF(Left_Image)
 3:     Raw_Right_Features ← DetectSURF(Right_Image)
 4:     Tri_Features(k) ← TriangulateFeatures(Raw_Left_Features, Raw_Right_Features)
 5:     {$\mathbf{R}_k$, $\mathbf{T}_k$, RANSAC_Features} ← RANSAC_AbsOrient(Tri_Features(k),Tri_Features(k-1))
 6:     $\{\mathbf{R}, \mathbf{T}\} \leftarrow \{\mathbf{R}, \mathbf{T}, \mathbf{R}_k, \mathbf{T}_k\}$
 7: end for
 8: $\{\mathbf{R}, \mathbf{T}\}$ ← iSAM_Smooth($\mathbf{R}, \mathbf{T}$)
 9: for Image Pair $i$ do
10:     for Image Pair $j$ do
11:         {$\mathbf{R}_k$, $\mathbf{T}_k$, Global_Features} ← RANSAC_AbsOrient(Tri_Features(i),Tri_Features(j))
12:     end for
13: end for
14: Factor_Graph ← AddInitializationNodesFactors()
15: for Image Pair $c$ do
16:     Curr_Pose ← NewPoseNodeAndFactor()
17:     Factor_Graph ← Curr_Pose
18:     for Curr_Feature ← NextFeature(Tri_Features(c)) do
19:         if IsNew(Curr_Feature) then
20:             Factor_Graph ← NewMeasurementFactorAndNode(Curr_Pose, Curr_Feature)
21:         else
22:             Factor_Graph ← NewMeasurementFactor(Curr_Pose, Curr_Feature)
23:         end if
24:     end for
25:     Factor_Graph ← iSAM_Update(Factor_Graph)
26: end for
```

6.4 Gain and Weight Selection and Tuning Parameters

The algorithm described in this thesis has a number of parameters, gains and weights that must be selected for proper estimation. The values described in this section were selected based on hand tuning to achieve good estimation performance while minimizing computational time as much as possible

Note that the time-step is $\Delta t = 0.5$ seconds. This is because every fifth frame captured was used in order to keep the size of the factor graph small. The iSAM system used a pseudo-huber cost function with Powell's DOG-LEG optimization engine. After each new frame was added at a time-step, an iSAM step was run seven times with full relinearization (i.e. the iSAM "mod-batch" setting was 1).

The covariance matrices used in Equation 4.100 have the values:

$$\mathbf{W}'_v = (0.001\text{m}/s^2)\mathbf{I}_{3\times3} \tag{6.14}$$

$$\mathbf{W}'_\omega = (0.001\text{rad}/s^2)\mathbf{I}_{3\times3} \tag{6.15}$$

The standard deviation for the pixel error in Equation 4.85 is 1 pixel in the x and y directions.

A prior was placed on the first (origin) pose of the principal inertia axis frames. This prior is zero mean with a number of very large standard deviations. The position standard deviation is 1.34 m (almost the entire field of view of the cameras at the operating distance). The velocity standard deviation is 0.1 meters per second. The angular velocity standard deviation is 3.16 radians per second (or 30 RPM). The error function on the prior for the orientation was found by computing the total quaternion according to Equation 4.71, and converting it to a MRP representation using Equation 2.34. Therefore, this is expected to have zero mean error and the standard deviation of all elements is set as 4.0.

Next, the translation and rotation between the geometric frame have a prior associated with them. The position has a standard deviation of 0.5 meters, while the orientation (computed the same as above) has a MRP standard deviation of 10.0. The first of the feature points has a very low uncertainty prior associated with it. This feature has a prior applied to its point estimate with the value $1.0E-6$ meters.

Lastly, the inertia ratios k_1 and k_2 are assumed to have zero mean and a standard deviation of 3.0 in order to imply very high uncertainty in the knowledge of the ratios of inertia. In other words, one standard deviation of the value of one moment of inertia with respect to another is $e^{3.0} \approx 20$.

6.5 Experimental Localization Results and Comparison to SPHERES Metrology

Chapter 4 described the new algorithmic approach that is being evaluated, while Sections 6.1 through 6.4 described the experimental approach for evaluating this algorithm. The output estimation results of the new algorithm are presented in this section and compared to reference metrology dataset that was described previously in this chapter. This compairison is one of the primary contributions of this thesis.

One important point to note is that the SPHERES satellites global metrology is measured in the reference frame of the geometric body, which does not necessarily coincide with the principal axes or center of mass, but should be close. Additionally, it has arbitrarily assigned reference axes, which the dynamic SLAM algorithm has no knowledge of. A flip of the reference frames to align with the SPHERES convention was performed on the estimated results in post-processing.

6.5.1 Position, Orientation, Linear and Angular Velocity

Figure 6-16 shows the position and velocity of the center of mass in the camera reference frame for both the estimated values and the reference measurements. Figure 6-17 shows the position and velocity difference between the Dynamic iSAM estimated values and the SPHERES Global Metrology estimates.

Figure 6-18 shows the orientation and angular velocity estimates and reference measurements. Figure 6-19 shows the angle difference (note that the closest or smallest angle between the two reference frames is the angle of the axis-angle representation of the difference) that is found by performing subtraction between the Dynamic iSAM estimated quaternion and the SPHERES Global Metrology quaternion. The mean and standard deviation of the differences are shown in Table 6.1.

Table 6.1: Statistics for Difference between the Estimated and SPHERES Position, Velocity, Orientation and Angular Velocity

	Mean	**Standard Deviation**
X Position	1.59 cm	0.920 cm
Y Position	-6.27 cm	3.13 cm
Z Position	4.91 cm	1.36 cm
X Velocity	-0.00369 cm/s	0.134 cm/s
Y Velocity	0.0828 cm/s	0.229 cm/s
Z Velocity	-0.236 cm/s	0.158 cm/s
Closest Angle	17.69 deg	3.59 deg
X Angular Velocity	-3.16 deg/s	1.80 deg/s
Y Angular Velocity	-1.29 deg/s	0.930 deg/s
Z Angular Velocity	-5.44 deg/s	2.20 deg/s

The position difference along the X-axis (i.e. the range measurement) is approximately 1.5 cm, however the Y and Z-axis differences are significantly larger. Note that the reference metrology is a relative measurement between the two SPHERES global metrology estimates. If the SPHERES reference attitude for the inspector orientation is off by a small amount, the relative position may be off a significant distance (i.e. it is magnified by the distance between the inspector and the target). The fact that there may be an error in the Global Metrology estimate of the orientation about the X-axis provides an acceptable explanation for the apparent bias in in the relative Y and Z positions. This hypothesis is further supported by the fact that the velocities between the dynamic SLAM approach match very closely with the global metrology reference. Table 6.1 shows that these are typically less than 1 millimeter per second, which is quite good. The smoothness of the velocities is due to the fact that the iSAM algorithm enforces the no external forces or torques constraint. An alternative differencing approach that does not enforce these constraints would lead to a much noisier velocity signal.

Table 6.1 and Figure 6-19 shows that the orientation estimates are close, but there are differences that vary between 10 and 15 degrees. It is not immediately obvious from this data which of the Global Metrology or Dynamic iSAM estimates are more representative of the physical motion (this will be further analyzed and discussed in Section 6.6). Additionally, it appears that the orientation difference is growing in Figure 6-18. Lastly, while the Y-axis angular velocity is very close to the global metrology measurements, the X and Z axis have a significant bias. This may be due to the fact that the Global Metrology gyroscope measurements are made with respect to the geometric frame and not the principal axis frame.

Now, if a correction is applied according to the results in Section 6.5.2, which is a 9.094^o rotation about the Y-axis so that the angular velocities are all measured in terms of the same SPHERES axis frame, the angular velocities comparison is now apples-to-apples. The attitude estimates, reference measurements and differences are shown in Figure 6-20 and 6-21. This shows that the attitude difference no longer grows with time and the X and Z axis of angular velocity now matches as well as the Y axis. The statistics of these results are shown in Table 6.2. These results now have lower mean and standard deviations which further validates the attitude and angular estimates, but not the inertia properties estimates.

Table 6.2: Statistics for Difference between the Corrected Estimated and SPHERES Orientation and Angular Velocity

	Mean	**Standard Deviation**
Closest Angle	13.48 deg	2.91 deg
X Angular Velocity	-0.30 deg/s	1.50 deg/s
Y Angular Velocity	-1.29 deg/s	0.930 deg/s
Z Angular Velocity	-0.25 deg/s	0.868 deg/s

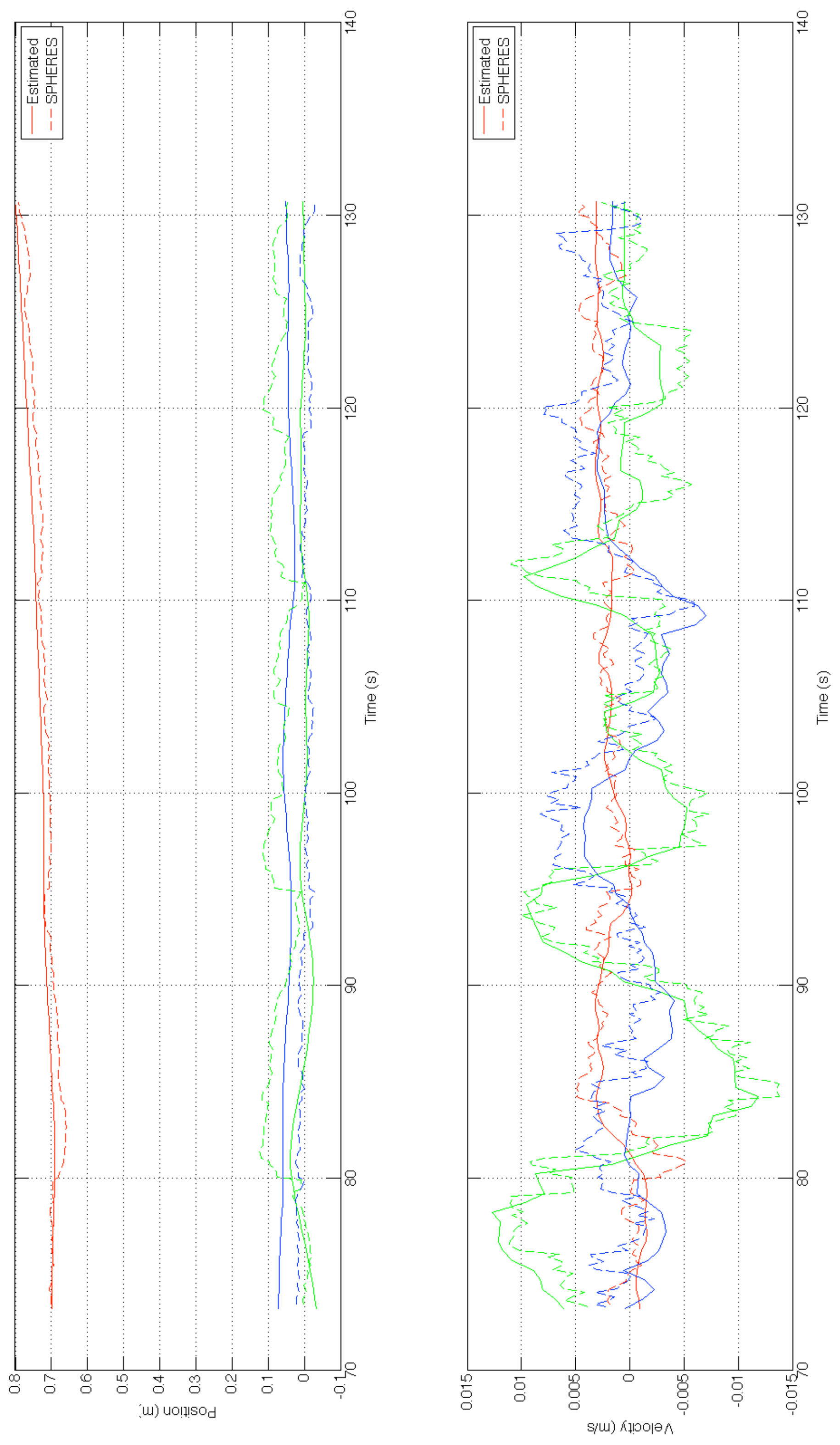


Figure 6-16: Position and Linear Velocity: Dynamic iSAM Estimate vs SPHERES Global Metrology

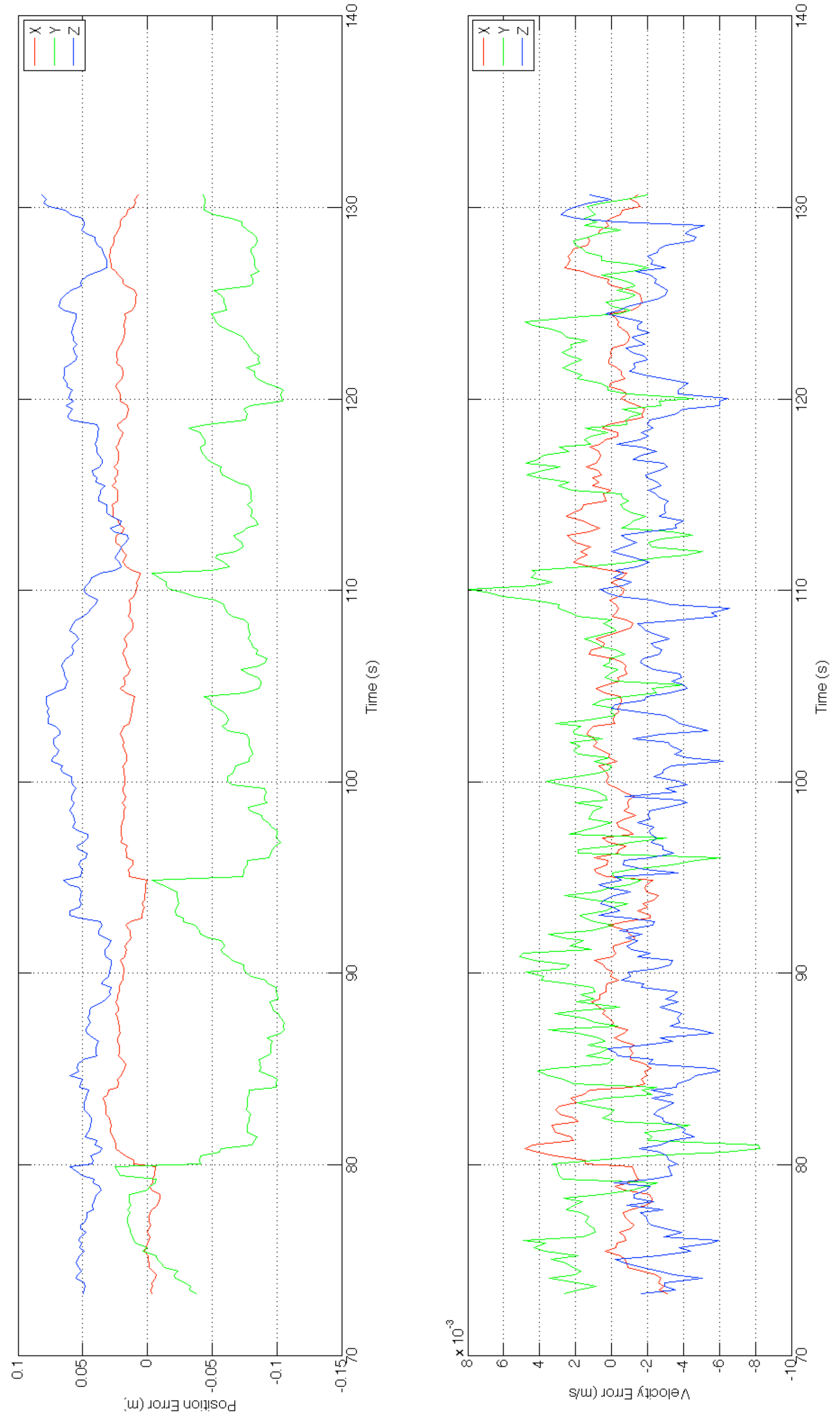


Figure 6-17: Position and Linear Velocity Difference: Dynamic iSAM Estimate minus SPHERES Global Metrology

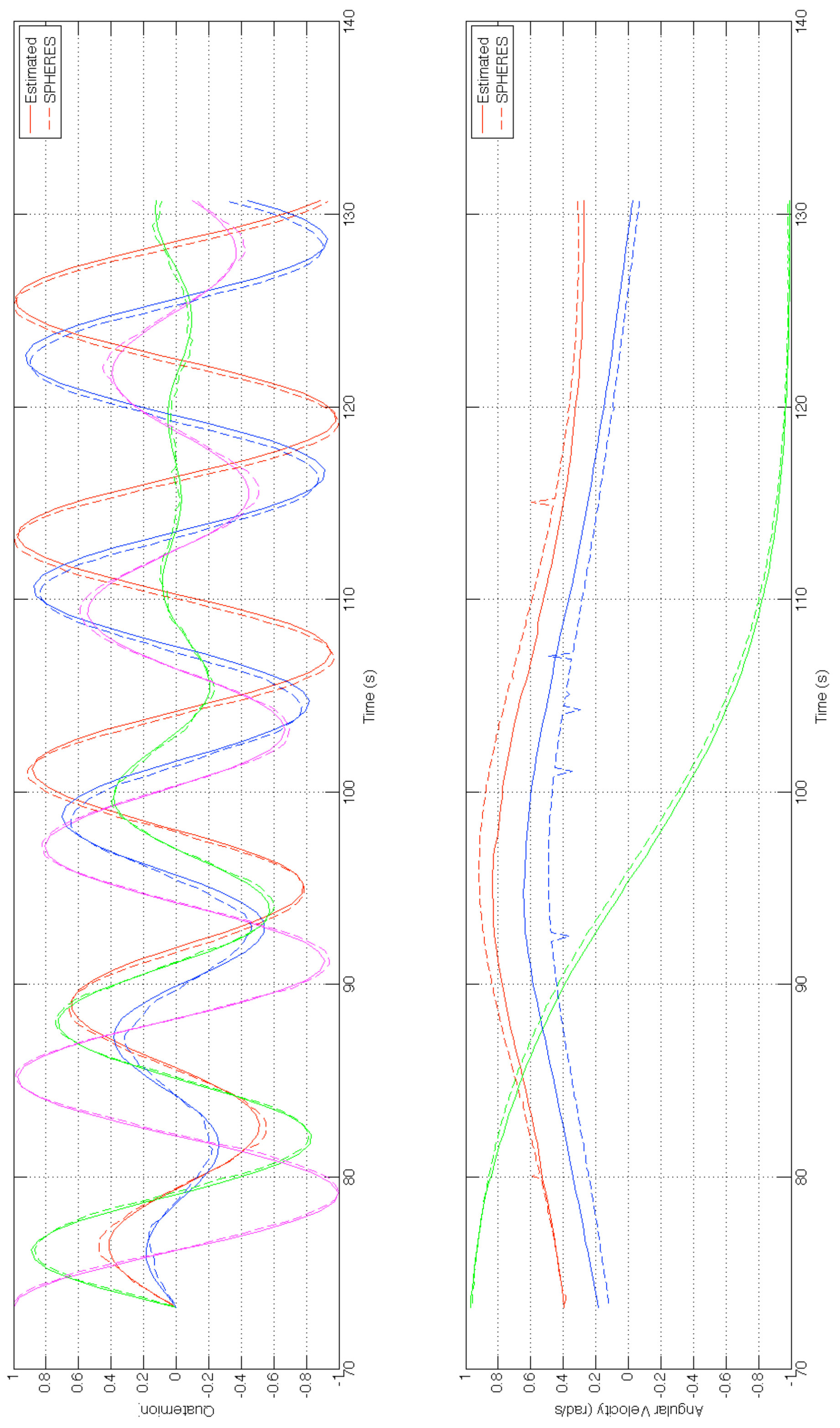


Figure 6-18: Quaternion and Angular Velocity: Dynamic iSAM Estimate vs SPHERES Global Metrology

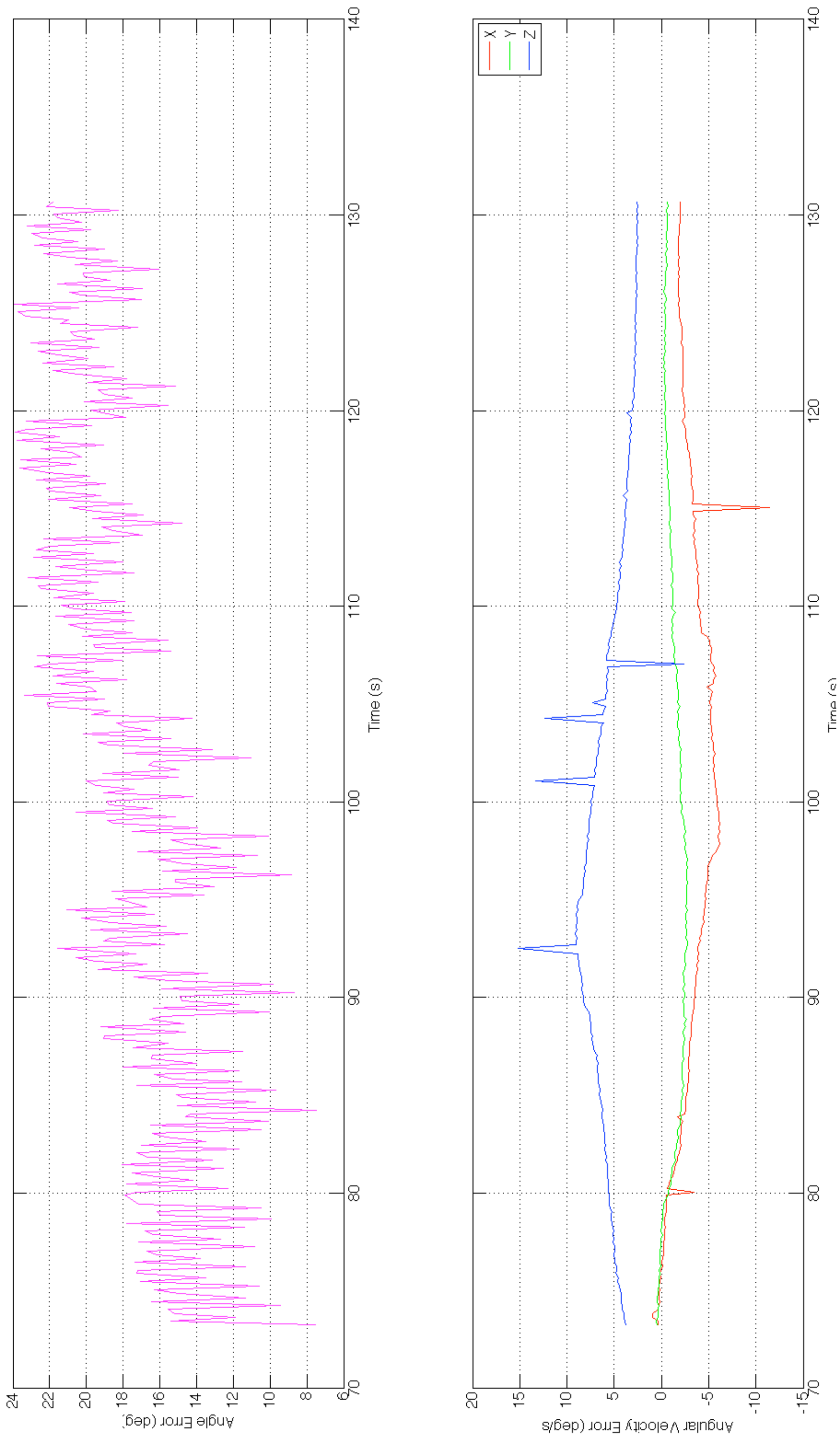


Figure 6-19: Angle and Angular Velocity Difference: Dynamic iSAM Estimate minus SPHERES Global Metrology

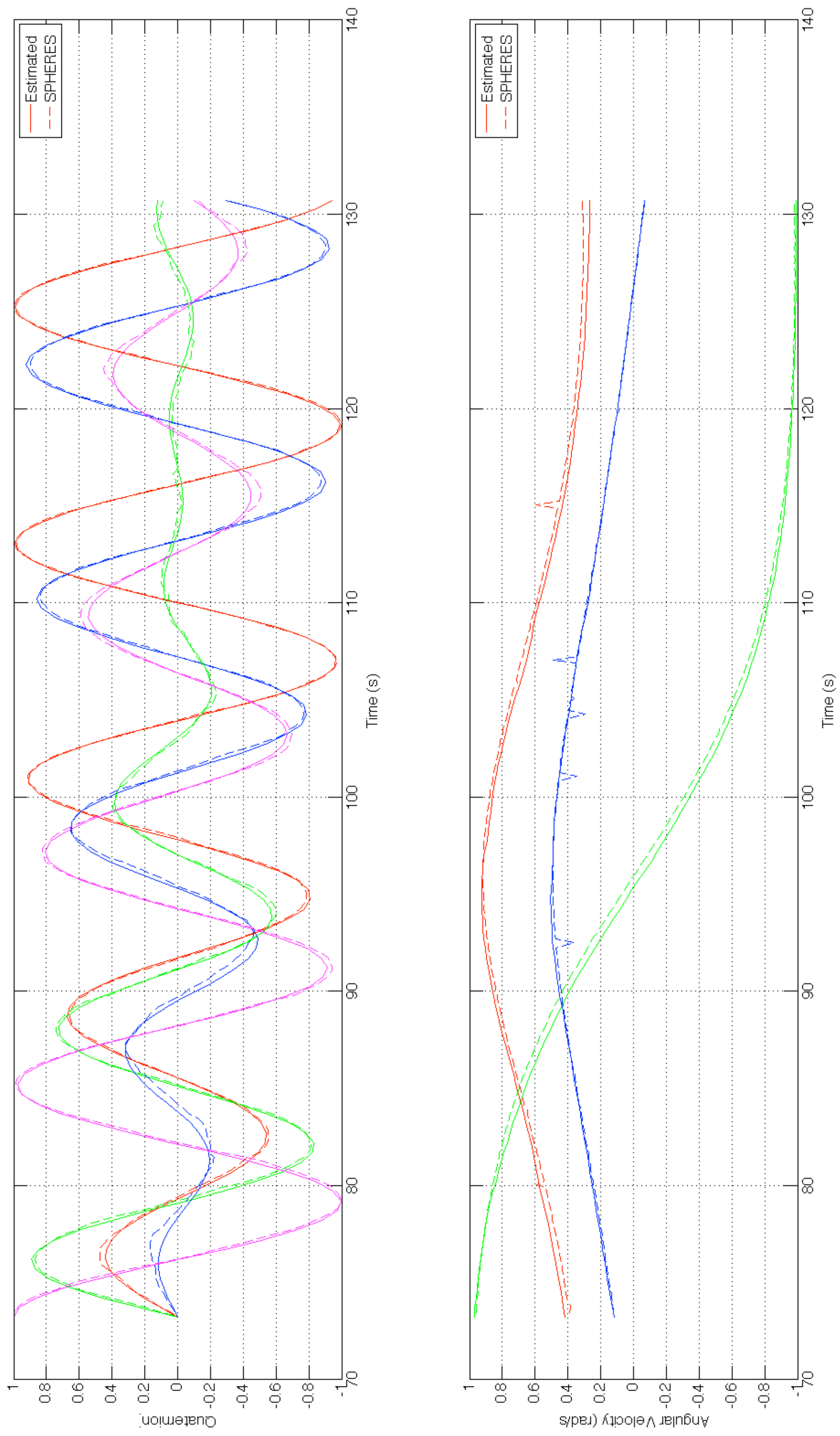


Figure 6-20: Quaternion and Angular Velocity: Dynamic iSAM Estimate vs SPHERES Global Metrology

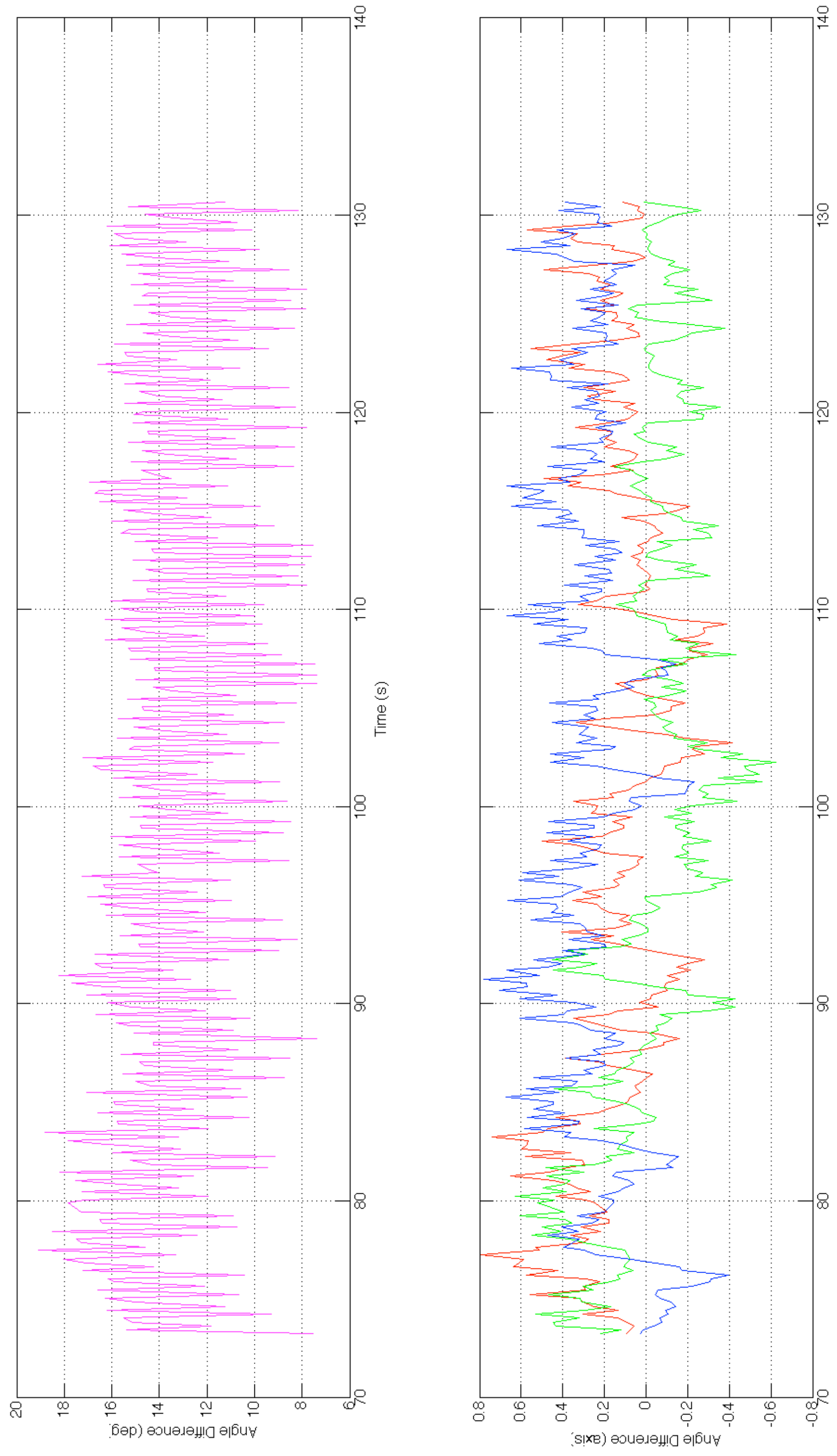


Figure 6-21: Angle and Angular Velocity Difference: Dynamic iSAM Estimate minus SPHERES Global Metrology

6.5.2 Inertial Properties

This section describes the estimates of the inertial properties including the center of mass, principal axes and inertia ratios and compares them to the reference values. This is a challenging comparison due to the fact that there is no high accuracy "ground truth" values for these parameters. This is partly due to the fact that in 2012, the SPHERES were upgraded with a new expansion port that was attached on its side, and there is limited amounts of data collected to establish the inertia properties with this new configuration. As a result, the results in this section do not present a clear consensus or exact match to prior values, but rather seem equally plausible. The values estimated by the methods in this thesis could in fact be the most accurate estimates to date, however there is not a sufficient quantity of data to make that determination.

This section presents the inertia values that were estimated and compares them primarily to Eslinger's values [30], which is the only other dataset available with the new expansion port.

The estimated values of the natural logarithm of the inertia ratios is k_1 and k_2:

$$k_1 = 0.0517 \tag{6.16}$$

$$k_2 = 0.0971 \tag{6.17}$$

This implies that the estimated inertia matrix (up to a scale factor) is:

$$\mathbf{J} = \begin{bmatrix} e^{k_1} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & e^{-k2} \end{bmatrix} \tag{6.18}$$

$$= \begin{bmatrix} 1.0530 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.9074 \end{bmatrix} \tag{6.19}$$

Compare this to the $\mathbf{J}_{\text{diag}}$ matrix from Equation 6.2, which is normalized so that the intermediate axis is 1.0 and shown below:

$$\mathbf{J}_{\text{diag, ref}} = \begin{bmatrix} 1.0322 & 0 & 0 \\ 0 & 1.0 & 0 \\ 0 & 0 & 0.8596 \end{bmatrix} \tag{6.20}$$

Note that the global metrology reference measurements are made relative to the SPHERES satellites geometric reference frame. The locations of the ultrasonic sensors and gyroscopes are what determine this reference frame. The satellites were designed so that these would be conveniently aligned with the symmetrical axes of the satellites. From the perspective of the algorithm in this thesis, this axis is arbitrary and, most importantly, unobservable. However, in order to make an accurate comparison, the transformation between the principal axis frame and the conventional SPHERES geometric frame must be determined. In Section 6.6, a dense three dimensional model that was estimated by the dynamic localization and mapping algorithm is presented. Using this model and a three dimensional computer aided design model with the MeshLab program [5], a manual alignment was calculated as shown in Figure 6-22.

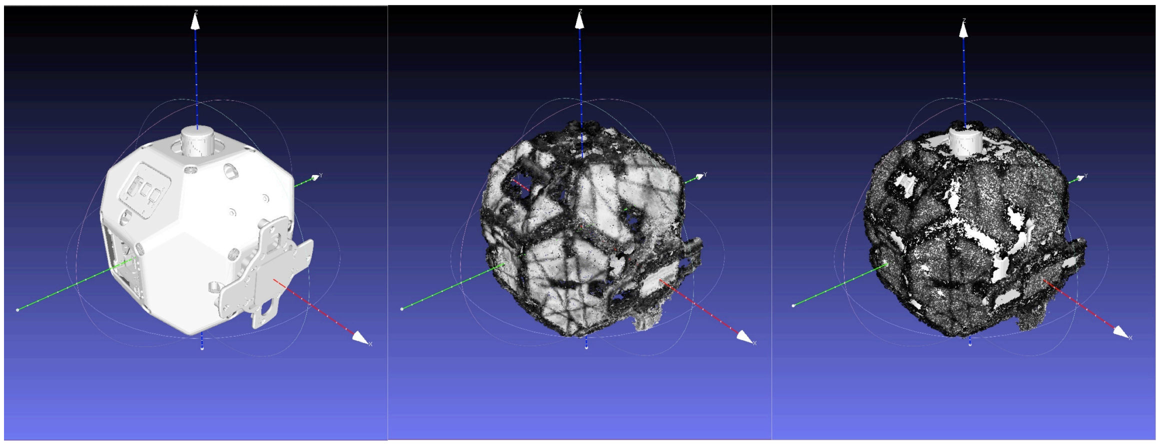


Figure 6-22: Final Alignment of SPHERES Engineering Model and Estimated Dense Map

The transformation between the SPHERES geometric axes and the estimated principal axes has the rotation and transformation shown below.

$$\begin{aligned}
\mathbf{p}_{\text{SPH}} &= \mathbf{R}_{\text{SPH/PA}}\mathbf{p}_{\text{PA}} + \mathbf{T}_{\text{PA/SPH}} && (6.21)\\
\mathbf{R}_{\text{SPH/PA}} &= \begin{bmatrix} 0.987 & 0.00 & 0.158 \\ 0 & 1.00 & 0.00 \\ -0.158 & 0.00 & 0.987; \end{bmatrix} && (6.22)\\
\mathbf{T}_{\text{PA/SPH}} &= \begin{bmatrix} 3.00 \times 10^{-3} \\ 0.00 \\ 0.9663.00 \times 10^{-3} \end{bmatrix} \text{meters} && (6.23)
\end{aligned}$$

Note that in comparing this to the value in Equation 6.9, the location of the center of mass estimates differ from Eslinger's estimates by -0.88 mm, 1.49 mm and 0.95 mm for the X, Y and Z axes respectively.

The axis angle representation of this rotation matrix is as follows.

$$\begin{aligned}
\mathbf{n} &= [0.00, 1.00, 0.00]^T && (6.24)\\
\theta &= 9.094 \text{ degrees} && (6.25)
\end{aligned}$$

Note that this is the same magnitude of rotation, but it is about the positive Y axis rather than the negative Z axis. While this difference in terms of angles seems significant, it is practically difficult to determine which is correct. The following sections will investigate how well each of these inertia matrices can be used for propagating attitude states, and how closely these results match experimental measurements.

Applying the rotation transformation to the estimated inertia matrix, the inertia matrix in the conventional SPHERES geometric frame can be computed:

$$\mathbf{J} = \begin{bmatrix} 1.0494 & 0 & 0.0227 \\ 0 & 1.0000 & 0 \\ 0.0227 & 0 & 0.9110 \end{bmatrix} \qquad (6.26)$$

Scaling this matrix to match the reference matrix using $s = 2.34 \times 10^{-2}$, we can directly compare the estimated inertia matrix in the SPHERES geometric frame to previous estimates and the reference in Equation 6.1.

$$s\mathbf{J} = \begin{bmatrix} 2.46 \times 10^{-2} & 0.00 & 5.31 \times 10^{-4} \\ 0.00 & 2.34 \times 10^{-2} & 0.00 \\ 5.31 \times 10^{-4} & 0.00 & 2.13 \times 10^{-2} \end{bmatrix} \text{kg m}^2 \tag{6.27}$$

An important method for validating the inertia matrix is to begin with a set of known initial conditions, and to propagate the orientation and angular velocity using Euler's Rotational Dynamics equation and the quaternion kinematics equation. The more accurate the estimate of the inertia matrix, the less this propagation will drift over time.

The figures below shows the results of propagating the kinematics and dynamics with the estimated inertia matrices (Equation 6.27 as "DISAM") versus Eslinger's reference inertia matrix (Equation 6.1 for "truth") and compares this to the global metrology measurements.

Figure 6-23 shows the propagation for the same dataset that has been used in previous sections, where the initial conditions at time $t = 75.0$ seconds and propagated until $t = 155$. Figure 6-24 shows the difference between both the two propagations when subtracted from the global metrology measurements.

These two figures show that the propagation between both the reference inertia and estimated inertia are very close. During the first 50 seconds, the propagation using the reference inertia has lower error compared to the global metrology measurements, while during the last 40 seconds the propagation using the DISAM estimated inertia has the lower error.

A second set of similar data was gathered during Test Session # 37. It performed a high speed spin about the intermediate axis of SPHERES and let this spin uncontrolled for 12 seconds. The same comparison for propagating inertia is shown in Figure 6-25 and 6-26. In this short timespan the reference inertia has lower error in both the orientation and angular velocity, however the difference is only 16 degrees and 6 degrees per second.

Some of this error may be due to errors in the reference (i.e. the gyros not being aligned with the principal axes) and with the loss in kinetic energy due to slosh and aerodynamic drag (note that the drift in the propagation appears to be correlated with knees in the kinetic energy curve in Figure 6-7.

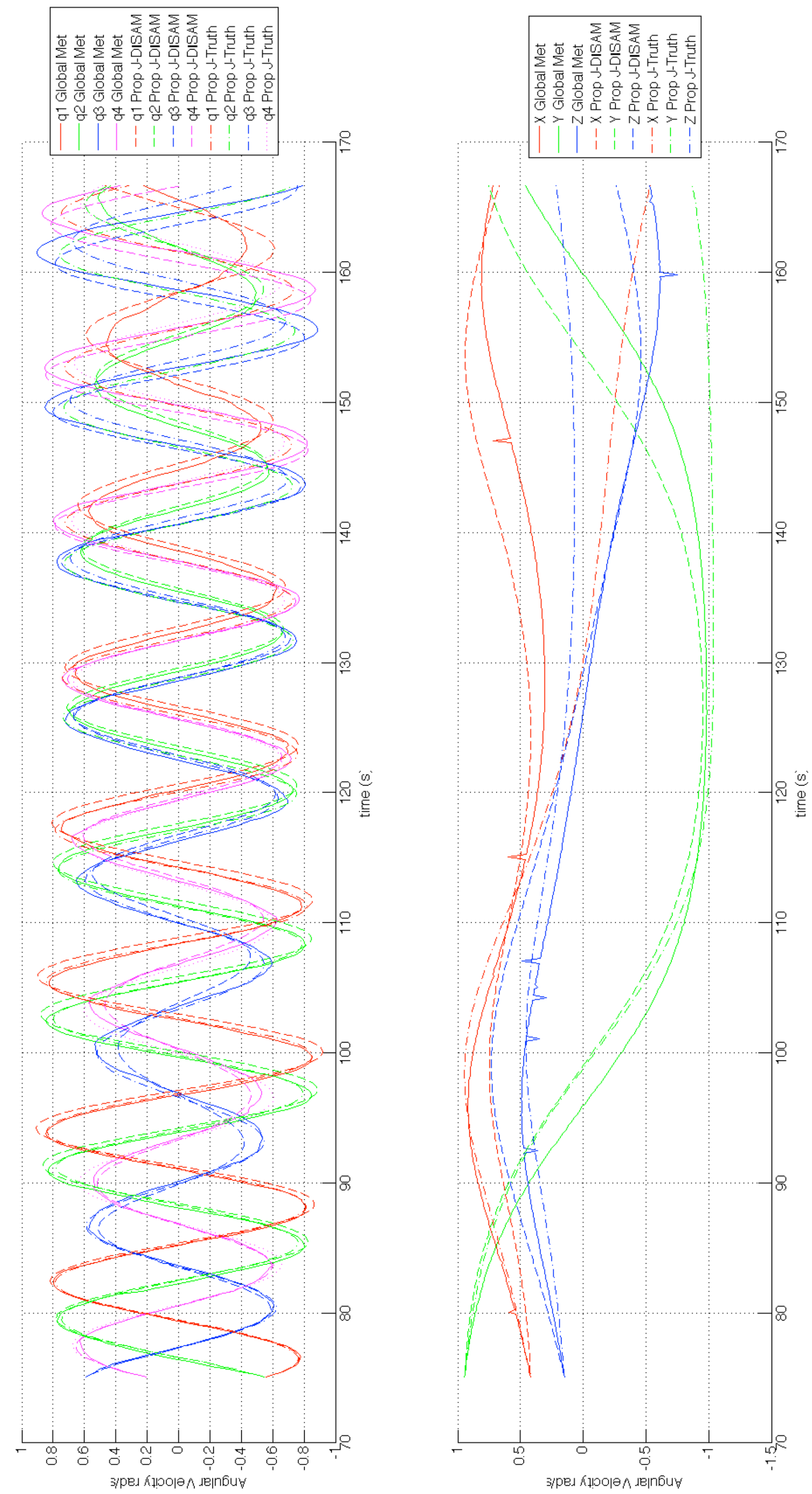


Figure 6-23: Attitude Propagation compared to Reference Measurements using DISAM Estimated Inertia vs Reference Inertia during Test Session # 38

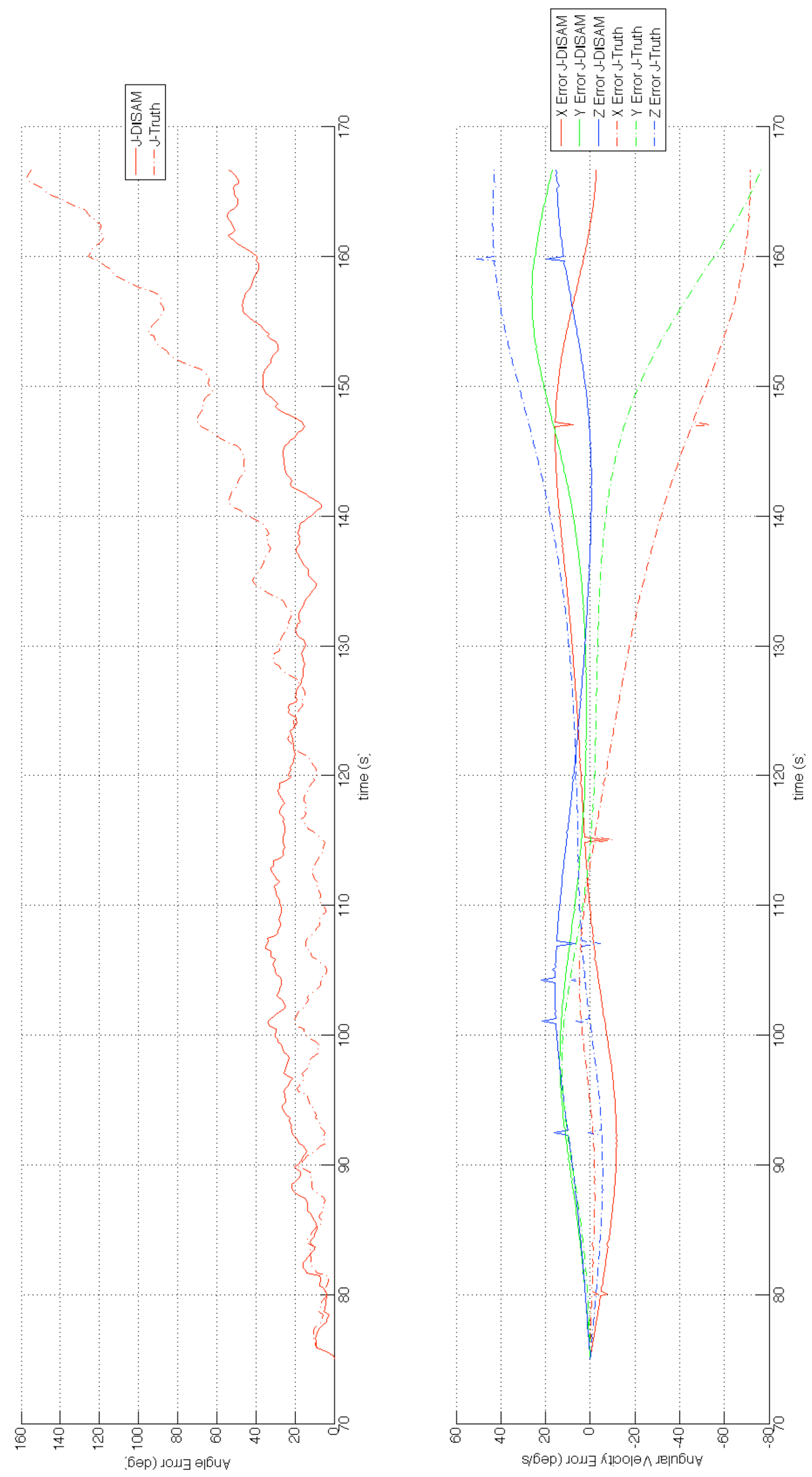


Figure 6-24: Attitude Propagation Error using DISAM Estimated Inertia vs Reference Inertia during Test Session # 38

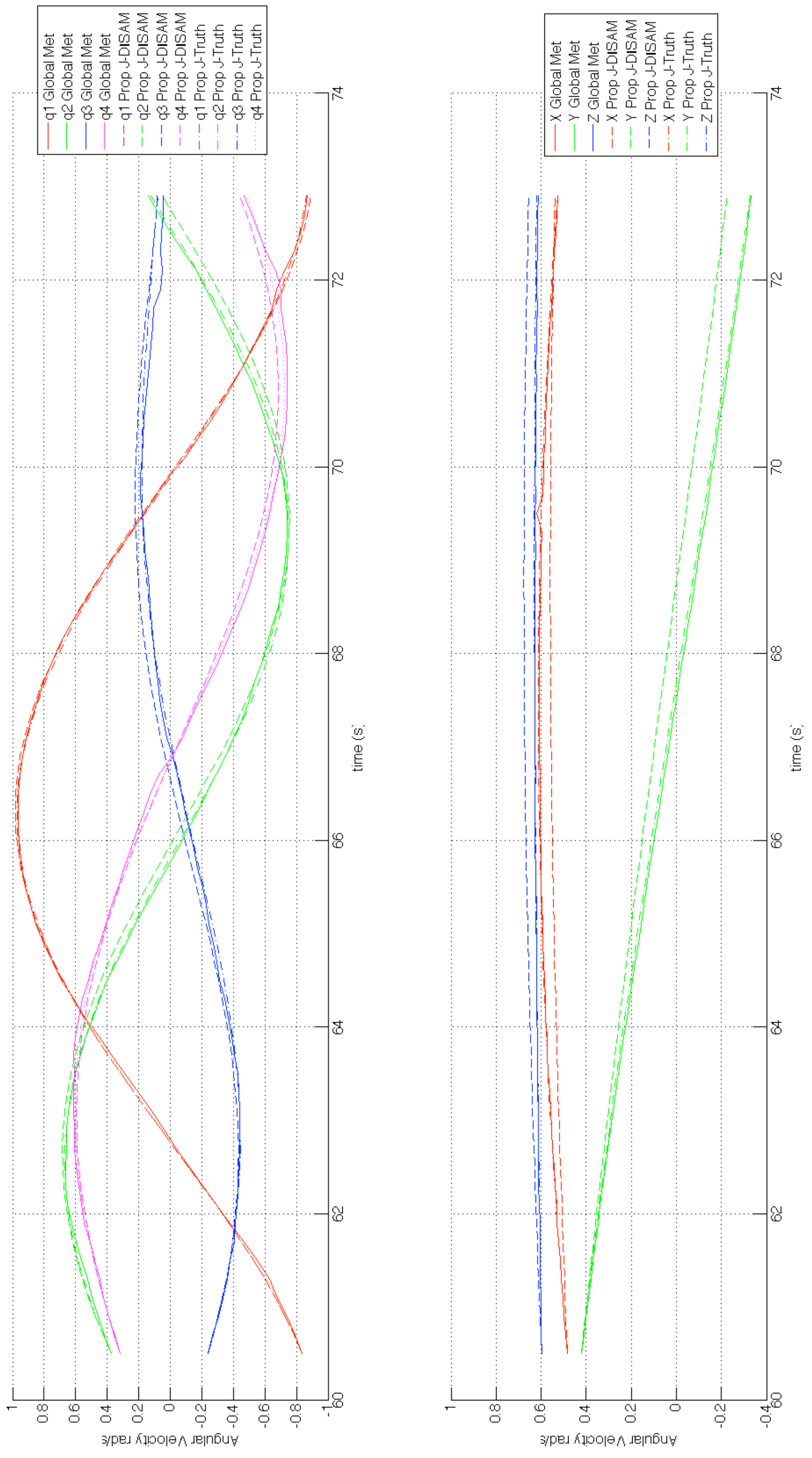


Figure 6-25: Attitude Propagation compared to Reference Measurements using DISAM Estimated Inertia vs Reference Inertia during Test Session # 37

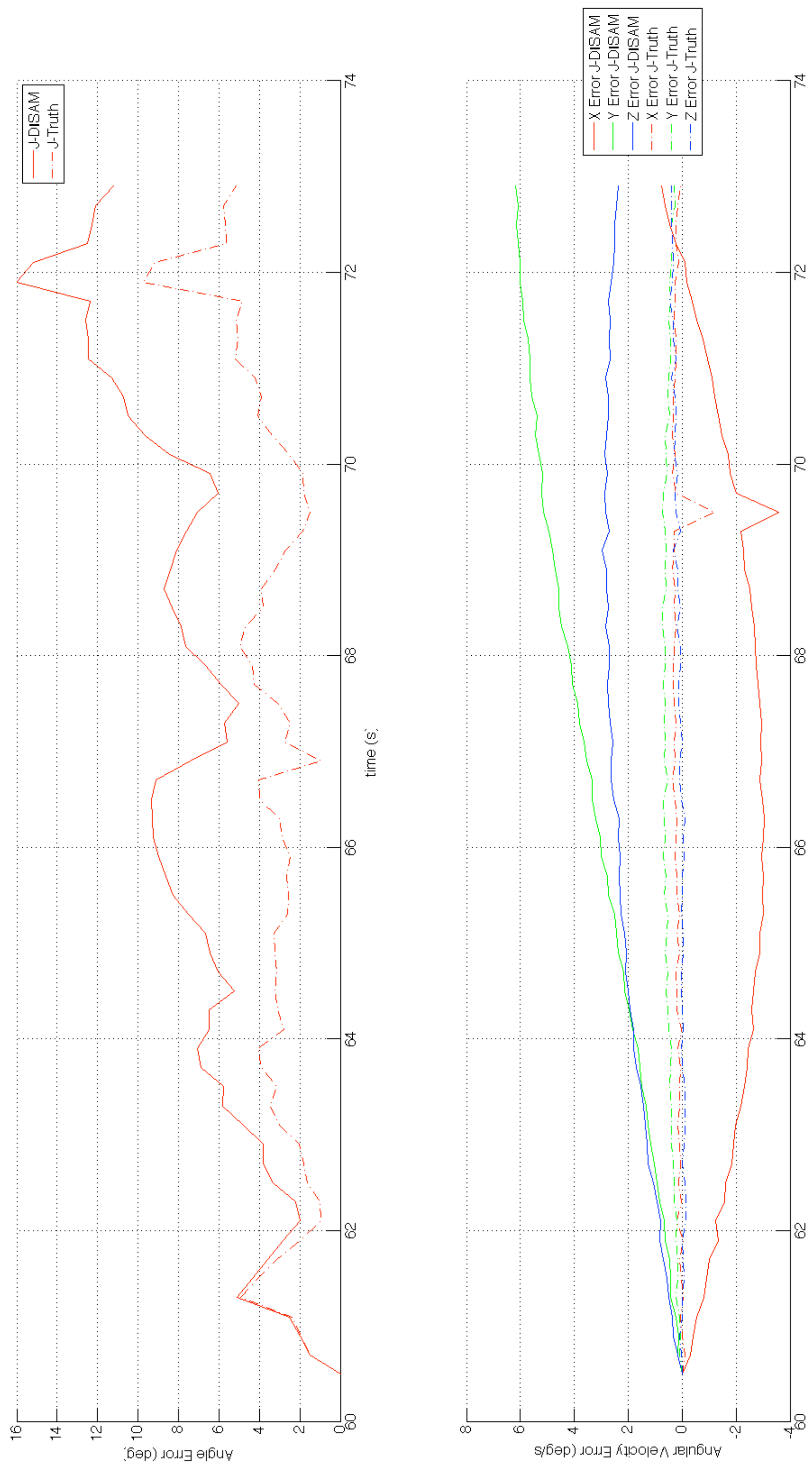


Figure 6-26: Attitude Propagation Error using DISAM Estimated Inertia vs Reference Inertia during Test Session # 37

6.6 Experimental Three Dimensional Mapping Results

Figure 6-27 shows three screen captures from a three dimensional visualization of the estimation process. The purple dots represent the location of the SURF features. The images on the left and right show the stereo images at that time. The closely spaced red, green and blue (RGB) arrows correspond to the X, Y and Z axes of the body frame trajectory. The RGB arrows far from the feature points is the camera frame location. The single RGB arrow close to the SURF features is the geometric frame location.

Figure 6-28 visualizes the mapped SURF feature points. This figure shows the feature points orthographically projected into the x-y, y-z and x-z planes. Note that the object is split in half with the points with the positive coordinates being projected in the left hand column while the negative coordinates are projected in the right hand column.

These points were estimated in the geometric frame and then rotated to the body frame using $\mathbf{R}_{G/B}, \mathbf{T}_{G/B}$ from Figure 4-3. Therefore the $(0, 0)$ coordinates in all of the images in Figure 6-28 correspond to the estimated center of mass location and the axes of these figures corresponds to the estimated principal axes. In other words, if a cube with perfectly even mass distribution was projected in this manner, its geometric center should be at all of the centers of the figure and it should be perfectly aligned with the figure axes.

Note that Figure 6-28 clearly looks to be the shape of a SPHERE facing the correct directions. Also, it appears that the center of the SPHERE is very close to the center of the figure, indicating a good center of mass estimate. Lastly, the geometry appears to be somewhat well aligned with the axes, indicating that the estimated principal axes are very close to the geometric axes (as expected).

In order to build a detailed geometric model, the depth map was computed and triangulated for each stereo image pair using the methods discussed in Section 2.10. Next using the estimated position and orientation of the body fixed (principal axes) reference frame, the pixels were projected into the body frame. Figure 6-29 illustrates a three dimensional rendering of this dense reconstruction. This figure shows a visually acceptable model that could be used for planning and control purposes. Note that there are holes in the model where there was not enough texture for the stereo depth map algorithm to determine a disparity. The source code for this method is shown in Listings B.26 and B.27.

The dense reconstructed model was animated using the position and orientation state that was estimated. Three frames from this animation are shown in Figure 6-30. The animation is done from the same camera location as the left hand image captured by the Goggles (shown on the left of the figure). Note that the field of view and focal length is not an exact match. Figure 6-30 shows that the animated motion is "visually similar", and provides a high level verification that the position and orientation estimates appear correct to the human eye.

This figure shows that there are no detectable alignment seams, which would occur if the Dynamic iSAM position and orientation estimates had time varying errors. This is an significant observation. It provides strong validation of the Dynamic iSAM estimates of the position and orientation. In Section 6.5 there was some discrepancy and error between the estimated values and the reference measurements. The fact that there are no alignment seams strongly supports the hypothesis that the Dynamic iSAM estimated values are in fact the best estimate of the physical motion.

The dimensions of the estimated satellite were measured using the open source program Meshlab (see Figure 6-31). Additionally, measurements of the SPHERES satellites were made by hand and compared with the Meshlab results. For comparison purposes, the size of the SPHERE based on the CAD model is 22.5 cm, 21.3 cm and 21.4 cm for the maximum X, Y and Z axes respectively (the difference is 0.4, 0.3 and 0.2 cm). These results are summarized in Table 6.3. The absolute errors between the estimated and hand measured size has a mean of 0.183 cm and a standard deviation of 0.113 cm. This shows that the estimated model is extremely close to the actual target object and therefore could be used for very precise planning and control purposes.

Table 6.3: SPHERES Geometric Size Comparison

	Hand Measured	**Estimated Size**	**Difference**
Maximum X-Axis (Figure 6-31)	22.9 cm	23.0 cm	0.1 cm
Maximum Y-Axis	21.6 cm	21.7 cm	0.1 cm
Maximum Z-Axis	21.2 cm	21.6 cm	0.4 cm
Sticker Width	13.0 cm	13.3 cm	0.3 cm
Sticker Height	7.50 cm	7.70 cm	0.2 cm
Sticker Line (Figure 6-32)	5.88 cm	6.02 cm	0.14 cm
Battery Door Width	7.07 cm	7.21 cm	0.14 cm
Battery Door Height	6.23 cm	6.15 cm	-0.08 cm

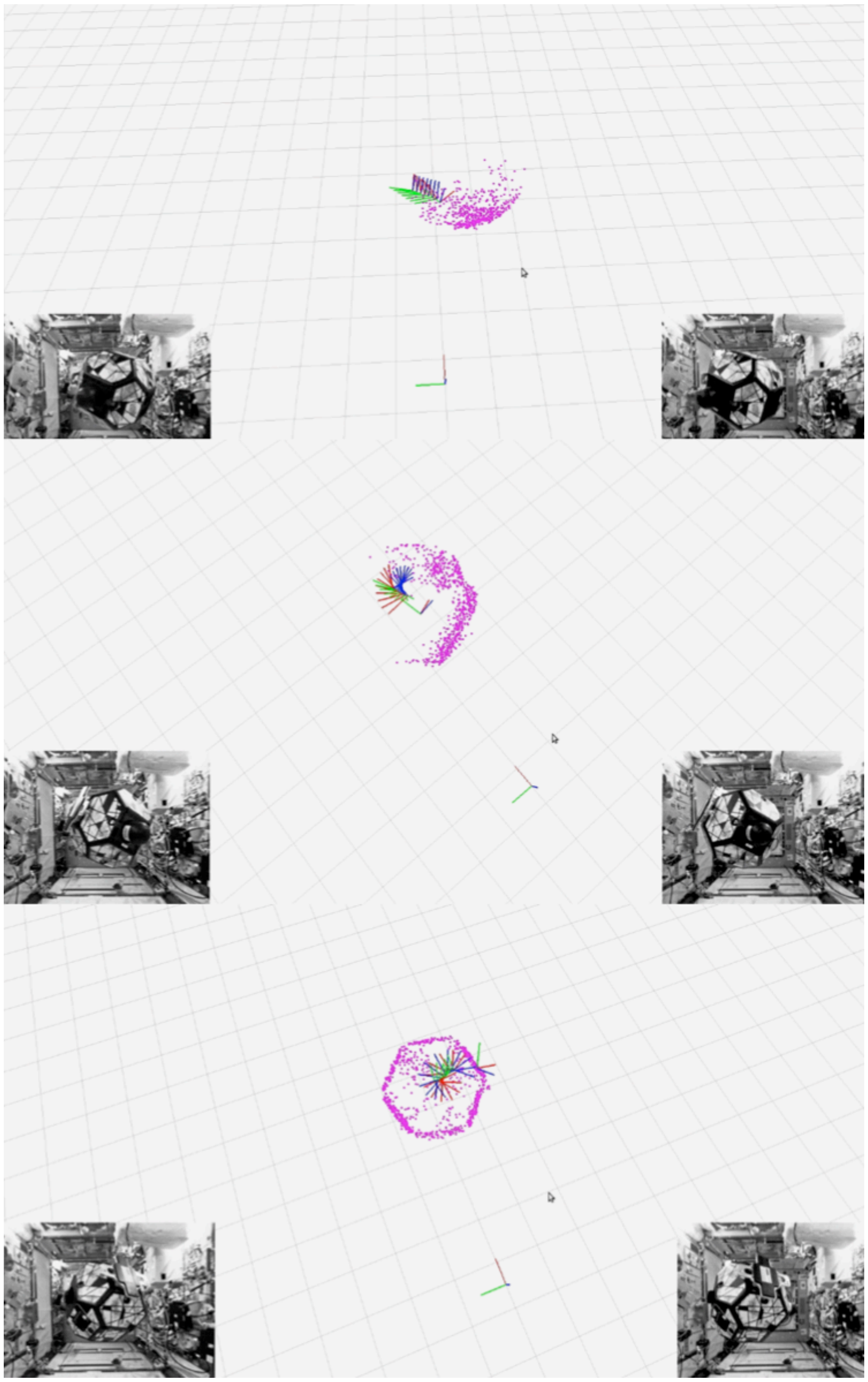


Figure 6-27: Three Dimensional Map of SURF Point Features, Body Frame Trajectory, Geometric Reference Frame and Camera Frame

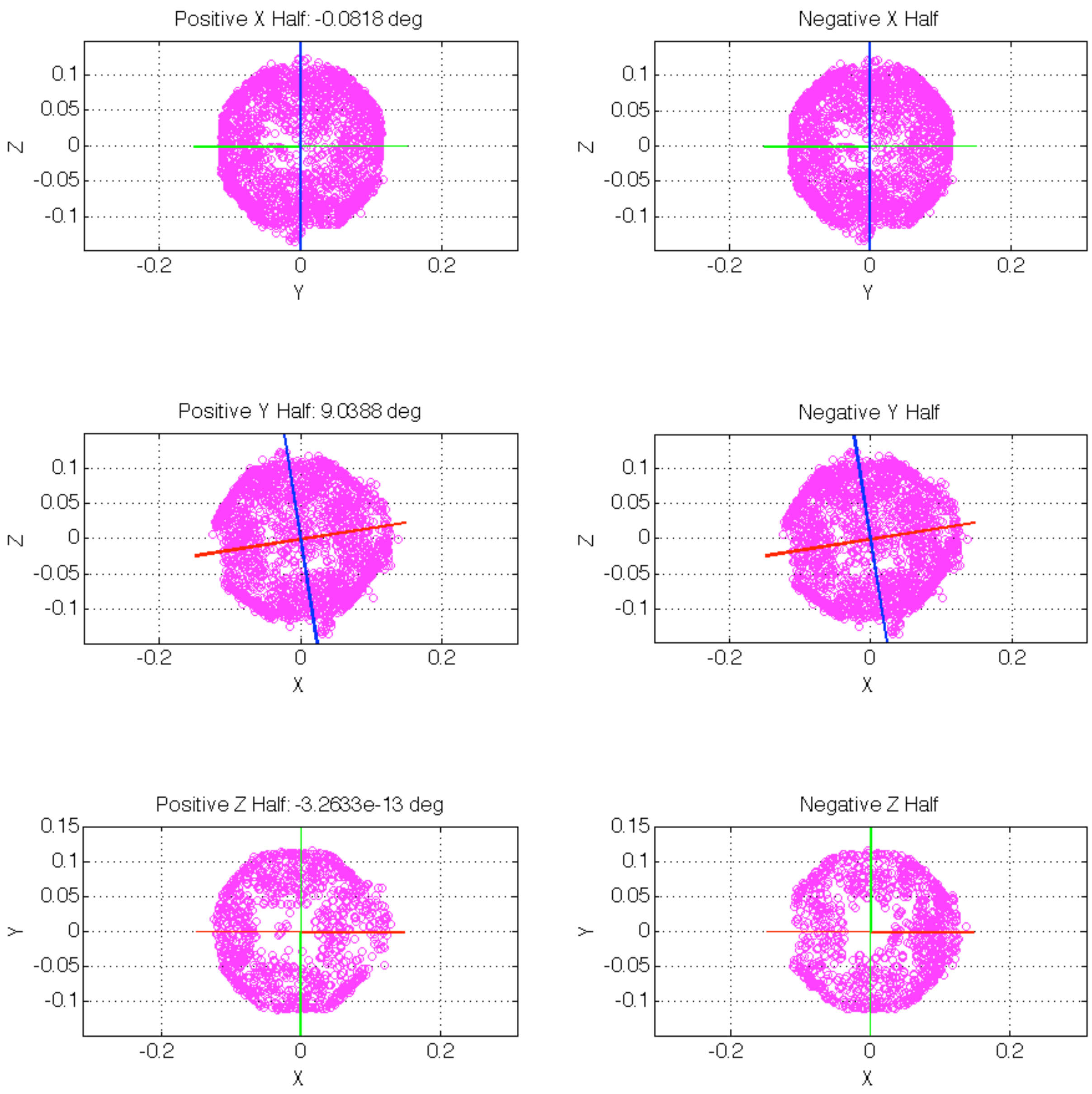


Figure 6-28: SURF Point Features in Principal Axes Body Frame with Reference Geometric Axes

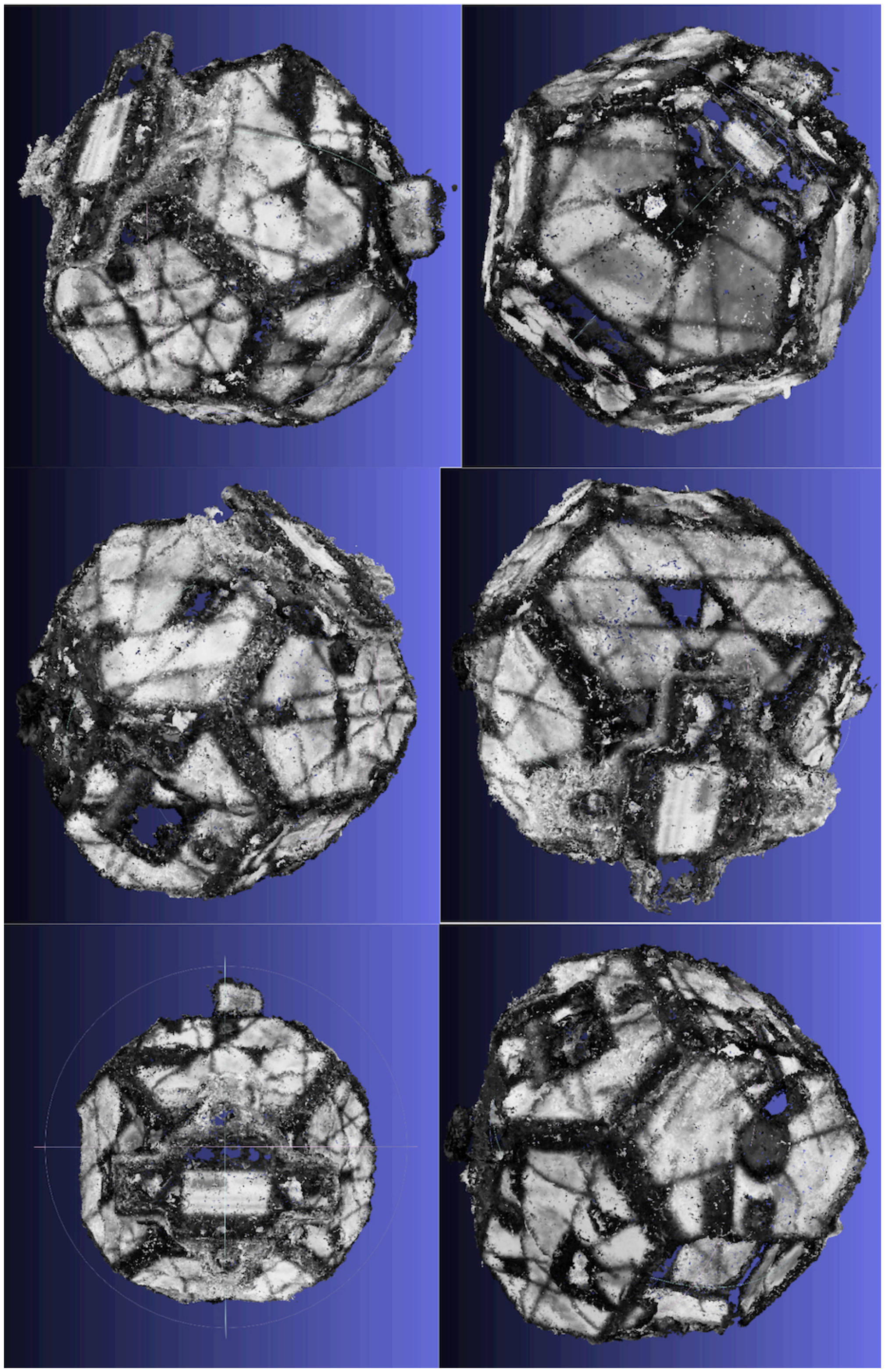


Figure 6-29: Dense Three Dimensional Reconstruction in Principal Axes Body Frame

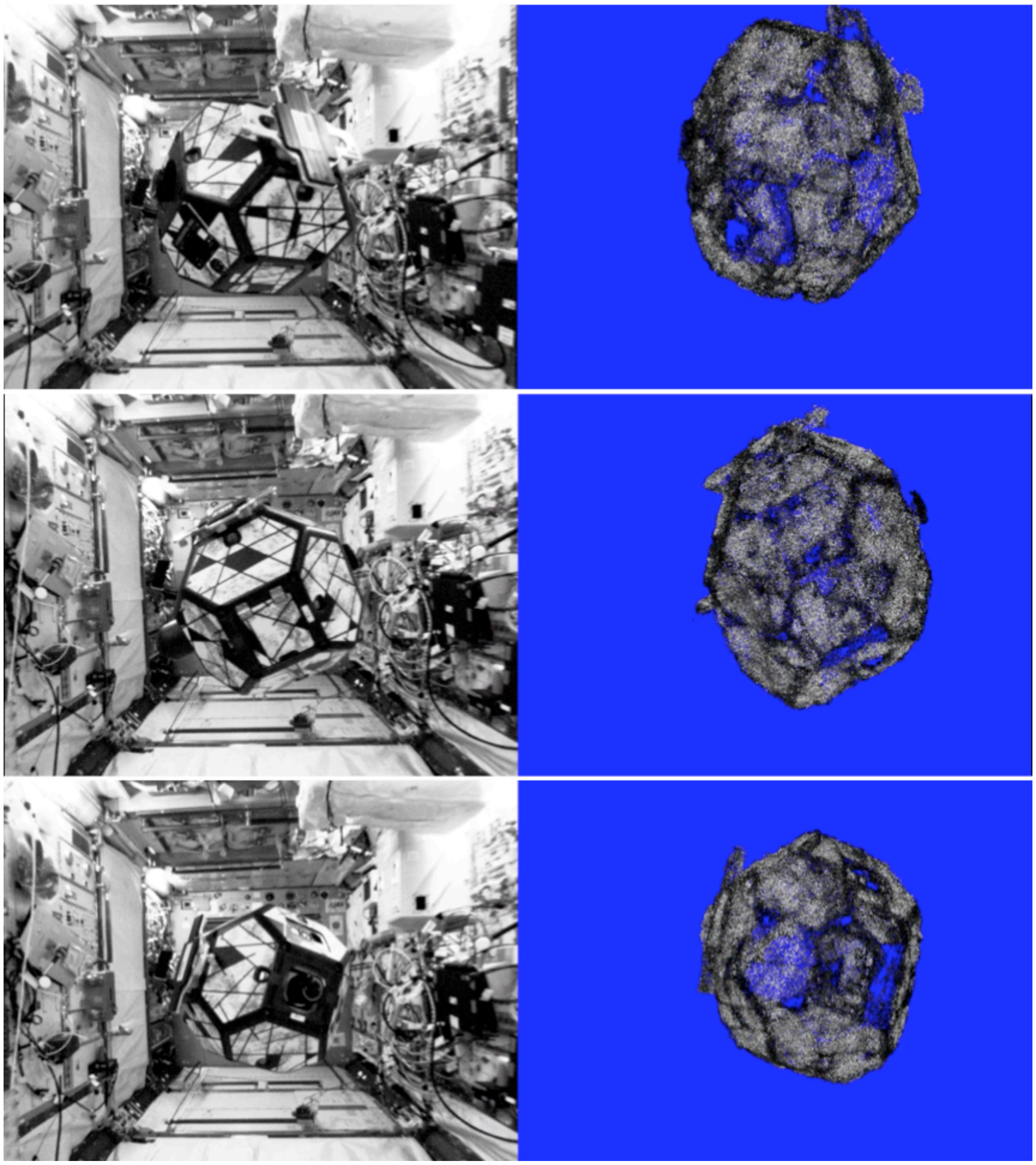


Figure 6-30: Animation of Dense Model Reconstruction

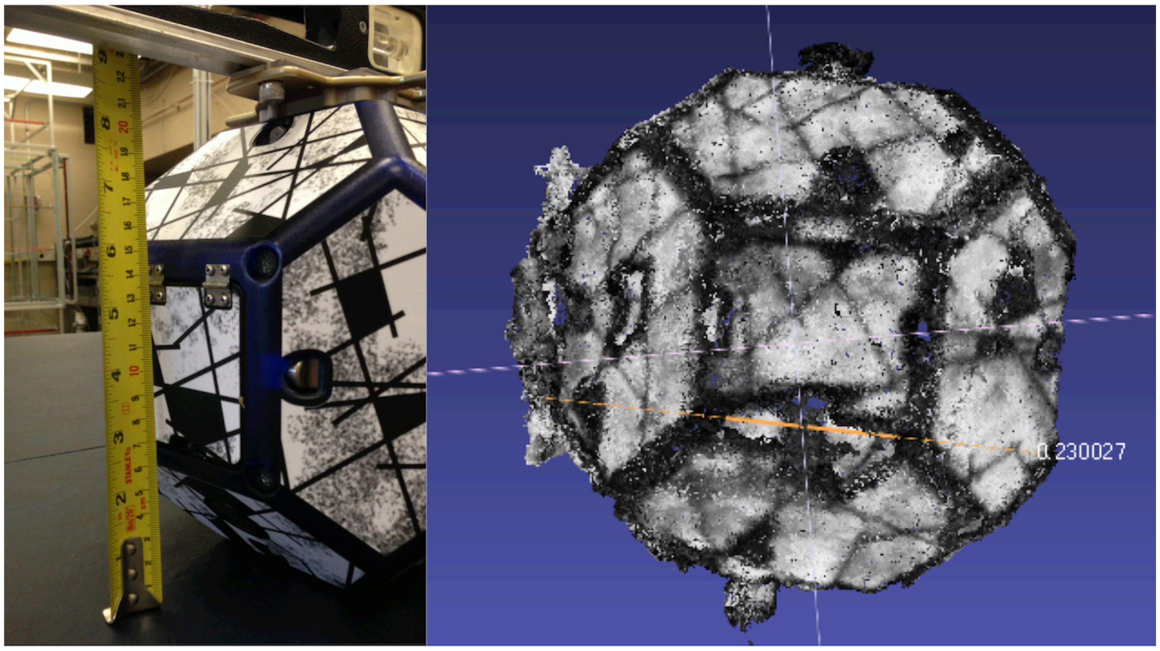


Figure 6-31: X-Axis Size: Hand Measurements and Meshlab Measurements

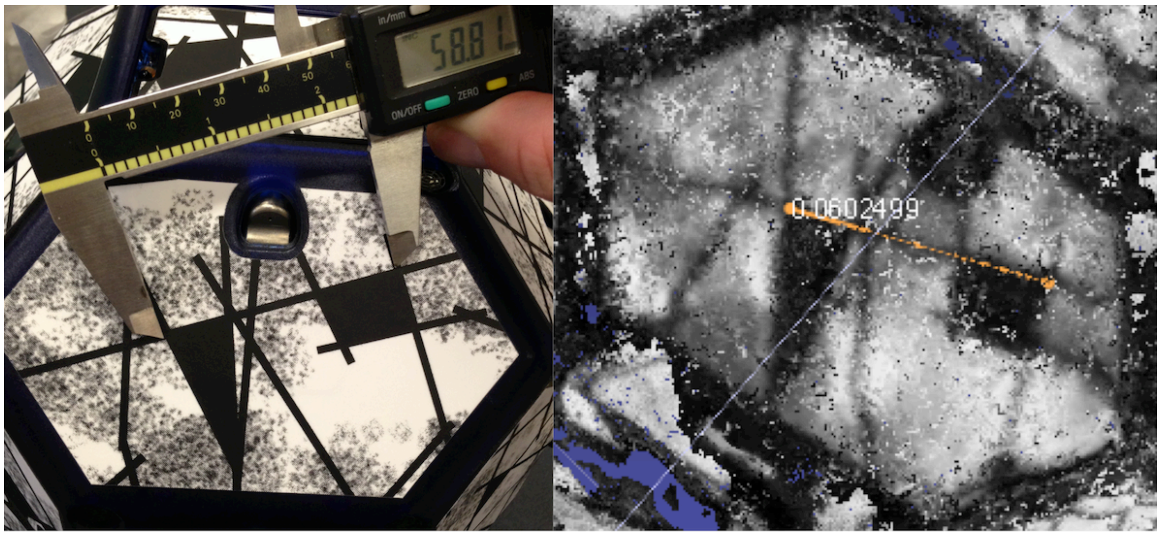


Figure 6-32: Size of Line on Sticker: Hand Measurements and Meshlab Measurements

6.7 Covariance and Convergence Analysis

The iSAM system is a smoothing estimator that models its parameters as Gaussian random variables. Each time a new image is added (or iteration as it is referred to in this section), the entire joint distribution is updated to minimize the cost function. The previous sections have only examined the mean of the individual variables at the final iteration. It is important to investigate how the uncertainty evolves over the iterations of the algorithm (i.e. as new images are added).

Each of the three-dimensional plots in this section look at the convergence of the estimates, while trying to follow a standard pattern so that they are simpler to read. Figure 6-33 is an example of this for the position of the target object. The top row of plots shows the uncertainty as a surface plot. The Z value shows the standard deviation on a $\log_{10}$ scale. The X-axis shows the iterations, where 115 images were used and the Y-axis shows the location over time (i.e. the trajectory in the smoothing framework). Note that this leads to a triangular structure for all of the plots, since there is no estimate for the position at a point in time until the iteration where that image was added. Since iSAM is a smoothing system, that estimate will never be marginalized out and will be updated at all subsequent iterations, which is what leads to the triangular structure.

The bottom row of plots shows the mean values for these estimates. Note that the previous sections slices of this figure at the final iteration (e.g. Figure 6-16 shows a slice containing only the final iteration of the bottom row of Figure 6-33). Note that this data was taken from a separate data association run that resulted in a different set permutation of the principal axes.

An alternative way of visualizing the convergence data is to animate the mean and standard deviation of the estimated trajectory as it evolves over the iteration axis in the previously discussed figures. Images from this animation are shown in Figure 6-40 and 6-41. Each image shows the SPHERES metrology estimate in red, the dynamic iSAM mean estimate in solid blue and the plus or minus one standard deviation in dashed blue.

Figure 6-40 and 6-41 show the initial estimate is very far from the SPHERES Metrology value, but after 5 to 10 seconds, the velocities closely track the SPHERES metrology values, but with less high frequency content. Note that the standard deviation shrinks over time and that previous values of the trajectory are updated as new measurements become available, thereby confirming the full trajectory smoothing aspect of this system.

The first time that the target object completes a revolution and thereby closes a loop is at time 79.25 in Figure 6-14. This is exactly 6 seconds and 12 images after the first frame. Note that in all of the plots in this section, the uncertainty significantly reduces and the means become much closer to the reference values at this point in time, which is exactly what is expected. Prior to this loop closure, the means and covariances of the estimate are not consistent due to the fact that the estimate is biased and the range of values within the standard deviation does not include the values that were converged to after the loop closure.

Note that for the position and velocity shown in Figure 6-33 and 6-34, it takes approximately 30 iterations for the algorithm to converge to its final value. However, it takes closer to 100 iterations for the algorithm to converge on the X-axis orientation and angular velocity shown in Figure 6-35 and 6-36, but only 40 iterations for the Y and Z axis. The center of mass in Figure 6-37 required approximately 50 iterations, the principal axes in Figure 6-38 required approximately 40 iterations and the inertia ratios in Figure 6-39 required approximately 100 iterations.

Table 6.4 documents the estimated marginal covariance once the algorithm had completely incorporated and estimated all 115 images. The averages for the pose values were taken as the mean of all 115 values. Given that the accuracy of the triangulated feature points are approximately a few millimeters, and the number of measurements that were made, the values of the estimated marginal standard deviations seem plausible. The values almost appear too low (or good) to be true, which could indicate the possibility of inconsistent covariance estimation, however the reference metrology is insufficient to absolutely determine whether this estimator is consistent.

Table 6.4: Final Estimated Marginal Standard Deviations

Quantity	**Standard Deviation** [1]
Average X Position	0.343 mm
Average Y Position	0.287 mm
Average Z Position	0.329 mm
Average X Velocity	0.509 mm/s
Average Y Velocity	0.566 mm/s
Average Z Velocity	0.457 mm/s
Average X MRP	0.0074
Average Y MRP	0.0063
Average Z MRP	0.0030
Average X Angular Velocity	0.228 deg/s
Average Y Angular Velocity	0.254 deg/s
Average Z Angular Velocity	0.296 deg/s
X Center of Mass	3.72 mm
Y Center of Mass	4.34 mm
Z Center of Mass	3.57 mm
Principal Axis X MRP	0.0074
Principal Axis Y MRP	0.0063
Principal Axis Z MRP	0.0030
Inertia Ratio 1	0.0025
Inertia Ratio 1	0.0019

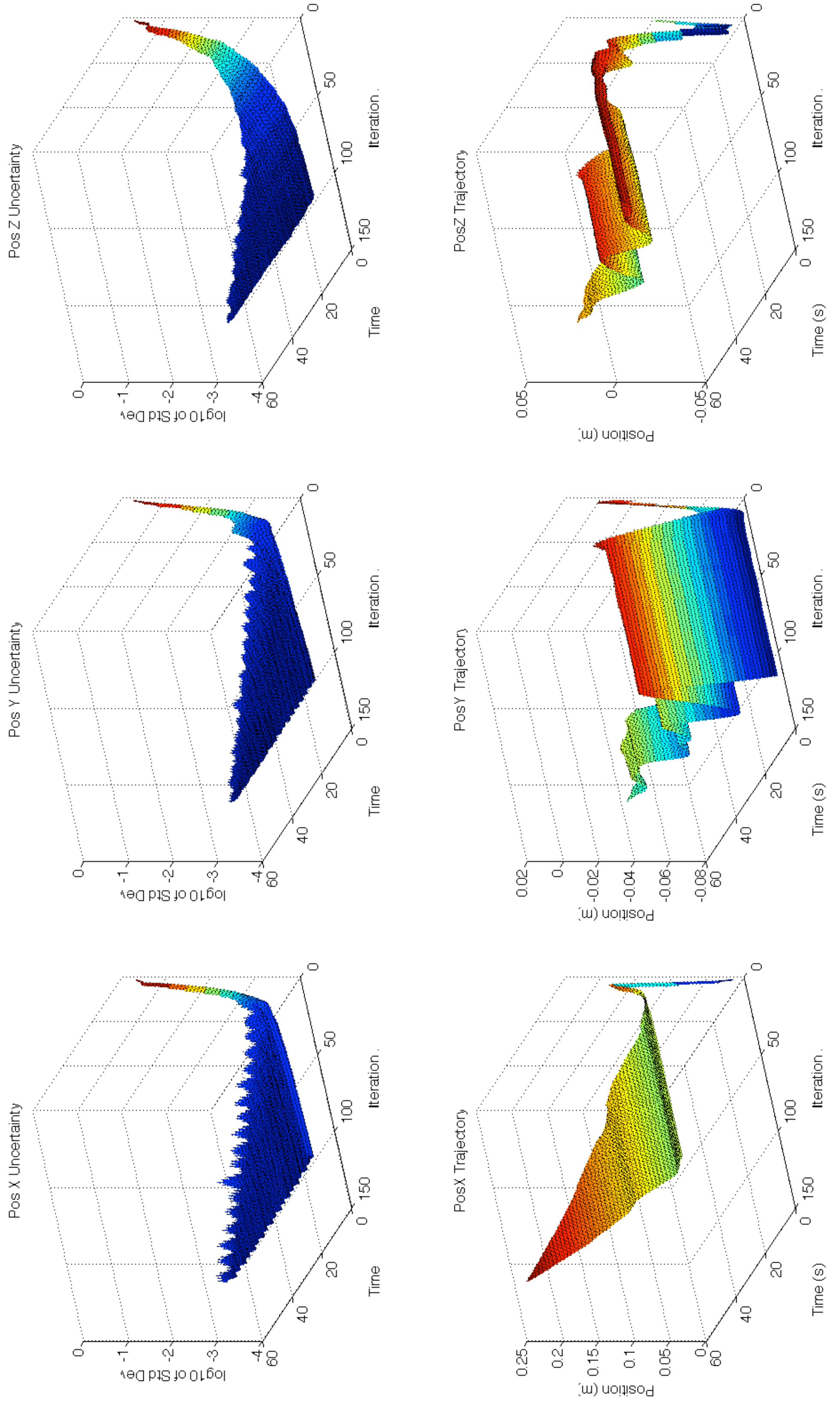


Figure 6-33: Uncertainty and Value of Position over Iterations

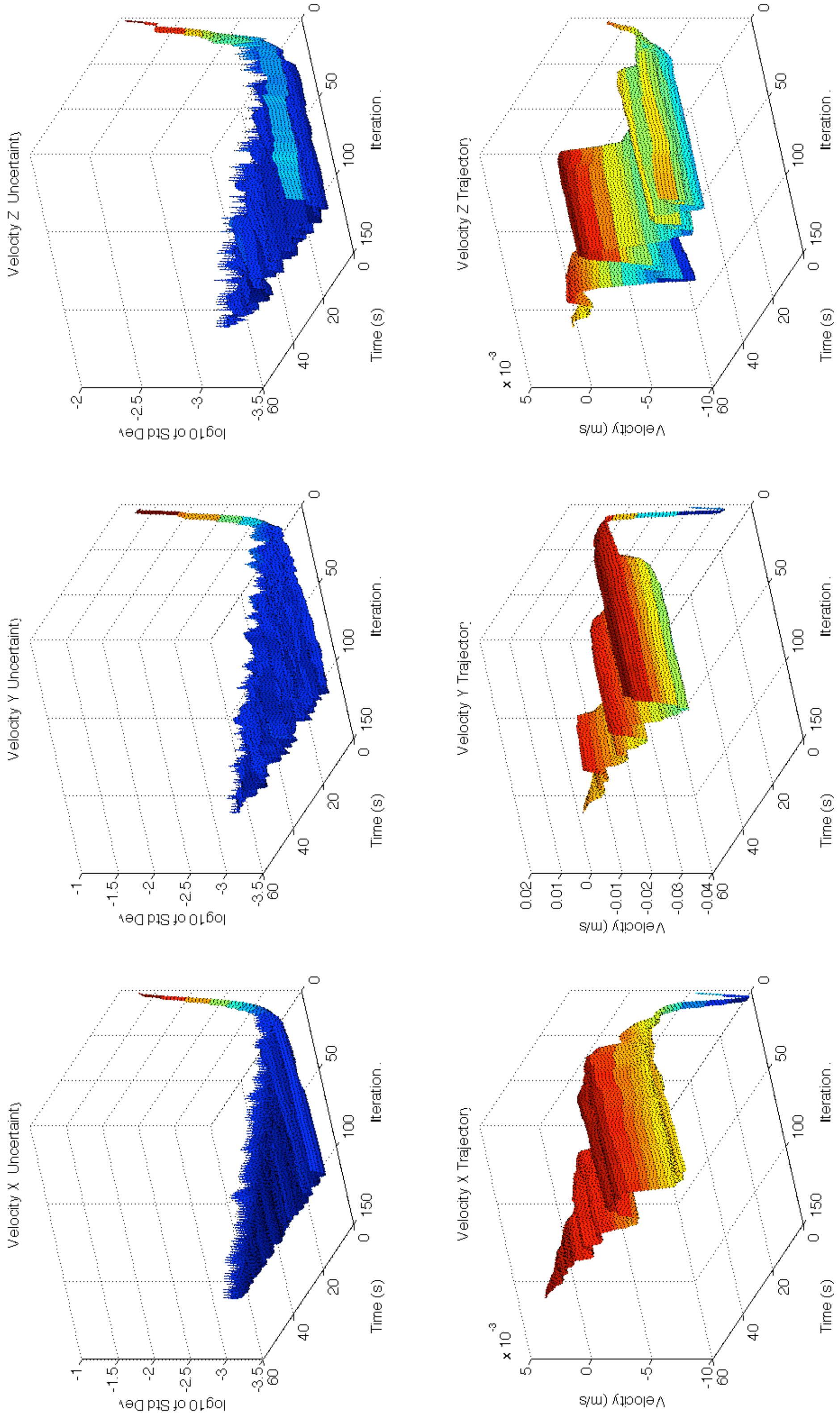


Figure 6-34: Uncertainty and Value of Velocity over Iterations

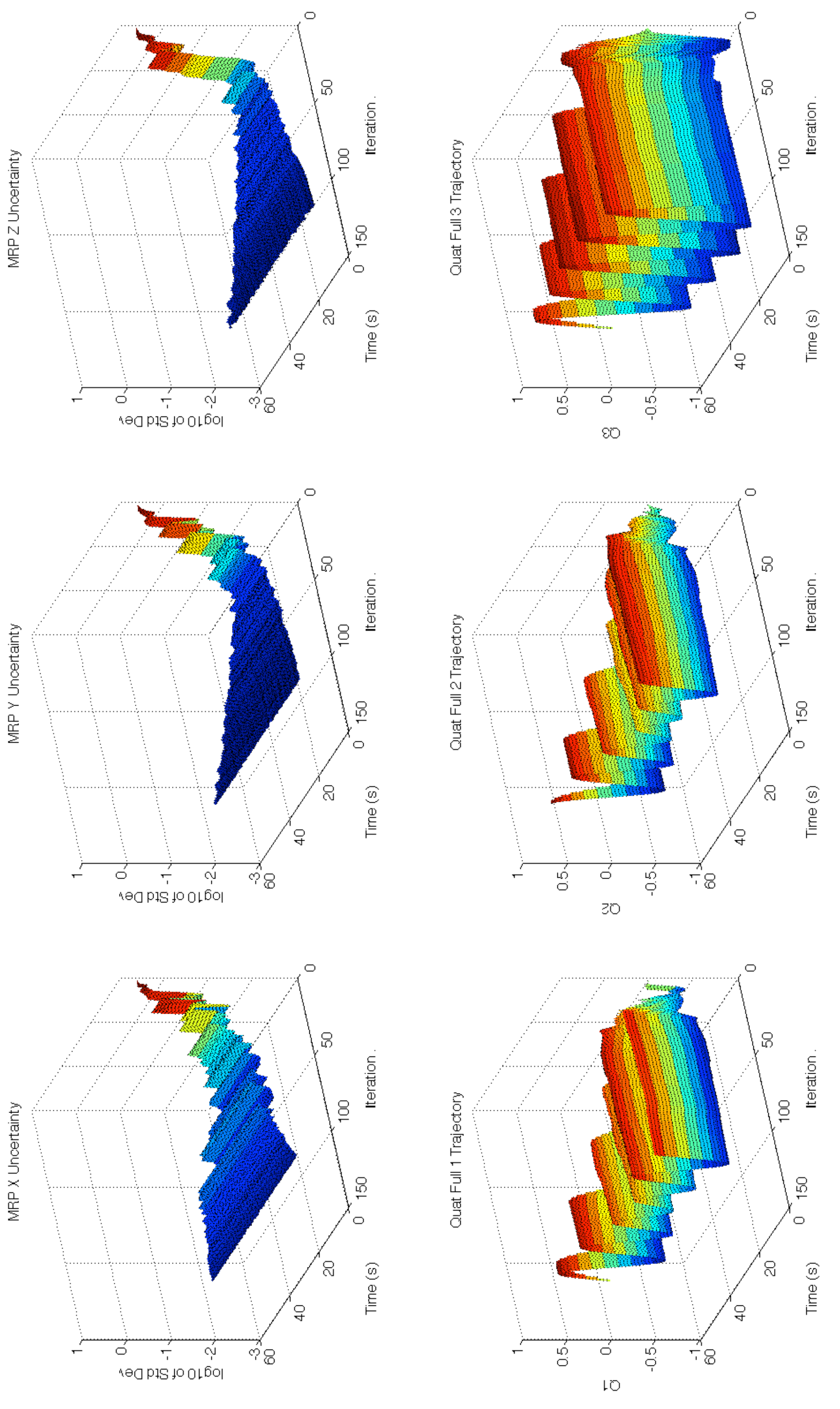


Figure 6-35: Uncertainty and Value of Attitude over Iterations

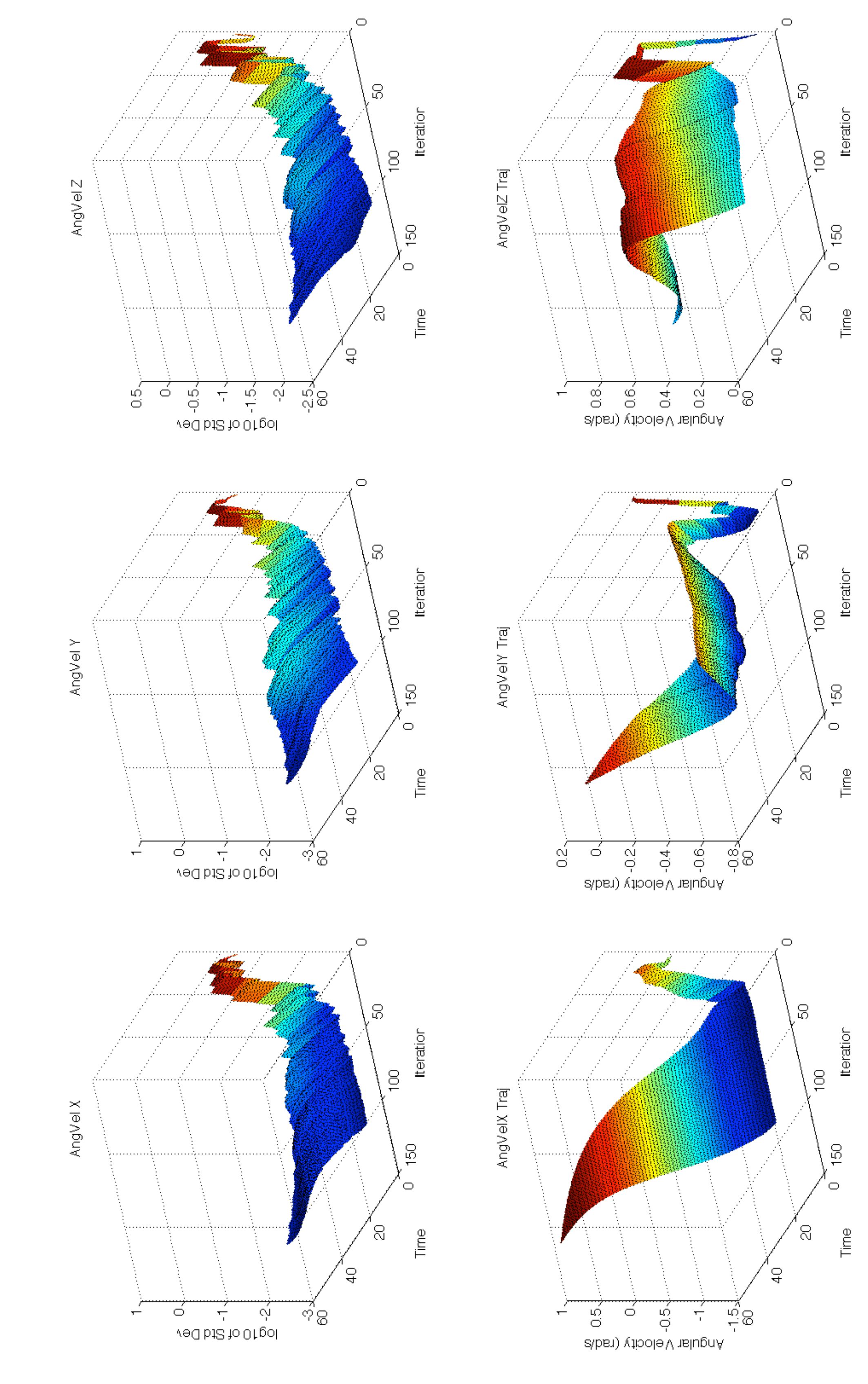
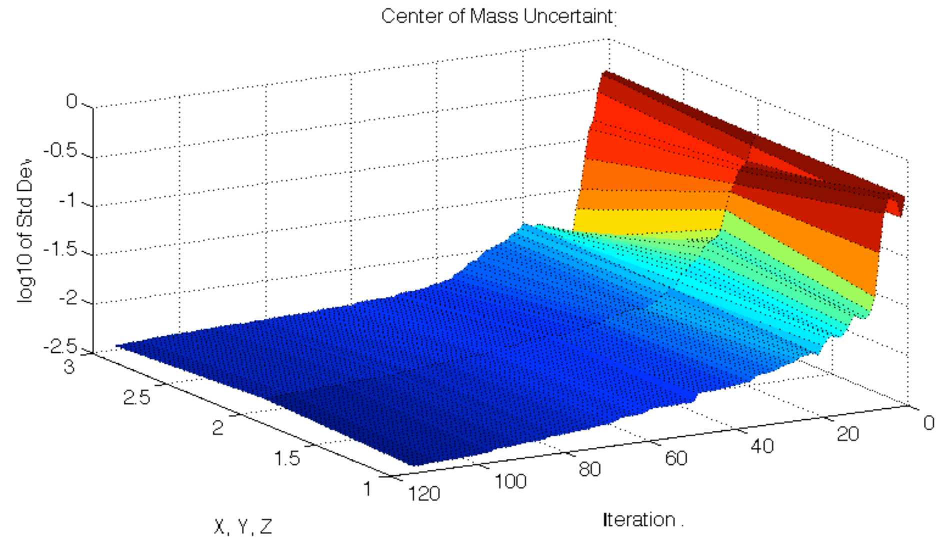


Figure 6-36: Uncertainty and Value of Angular Velocity over Iterations



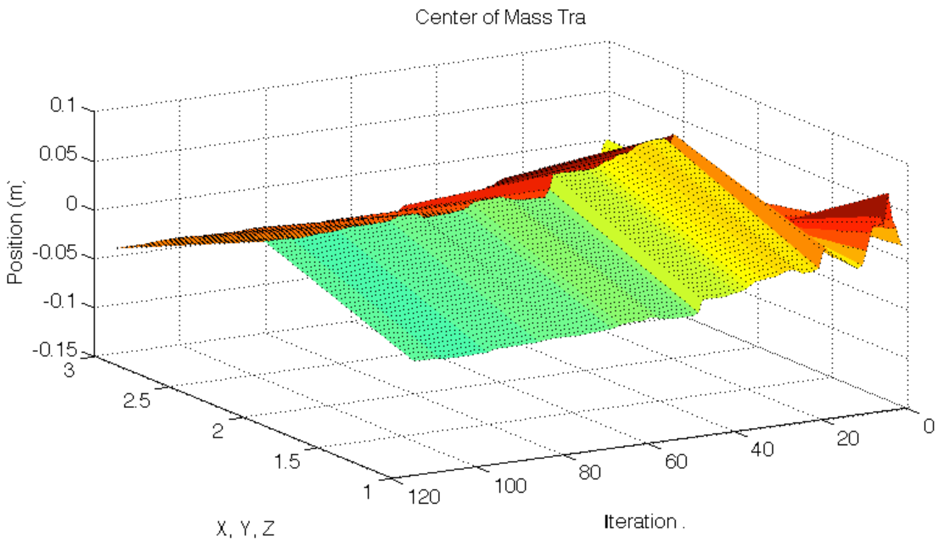
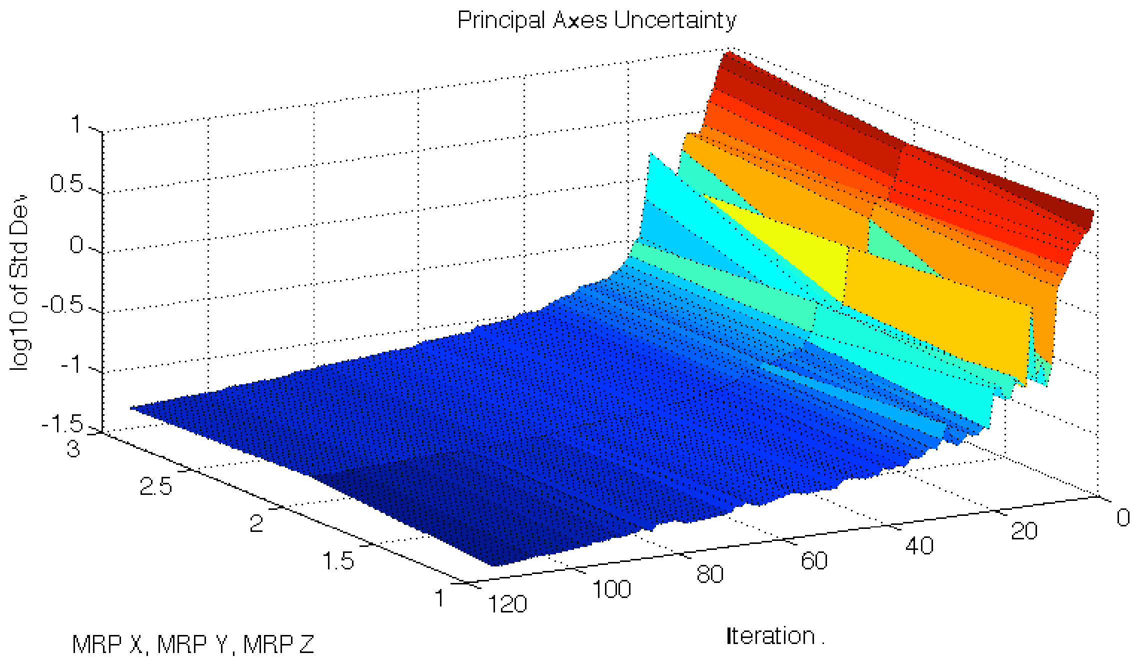


Figure 6-37: Uncertainty and Value of Center of Mass Location over Iterations



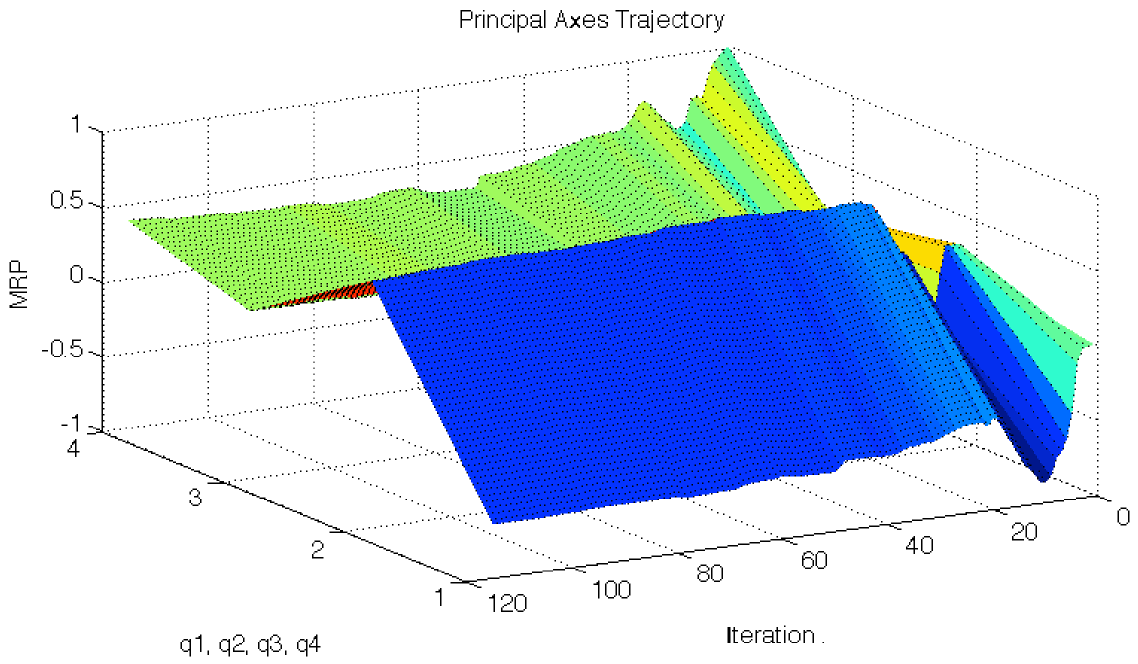
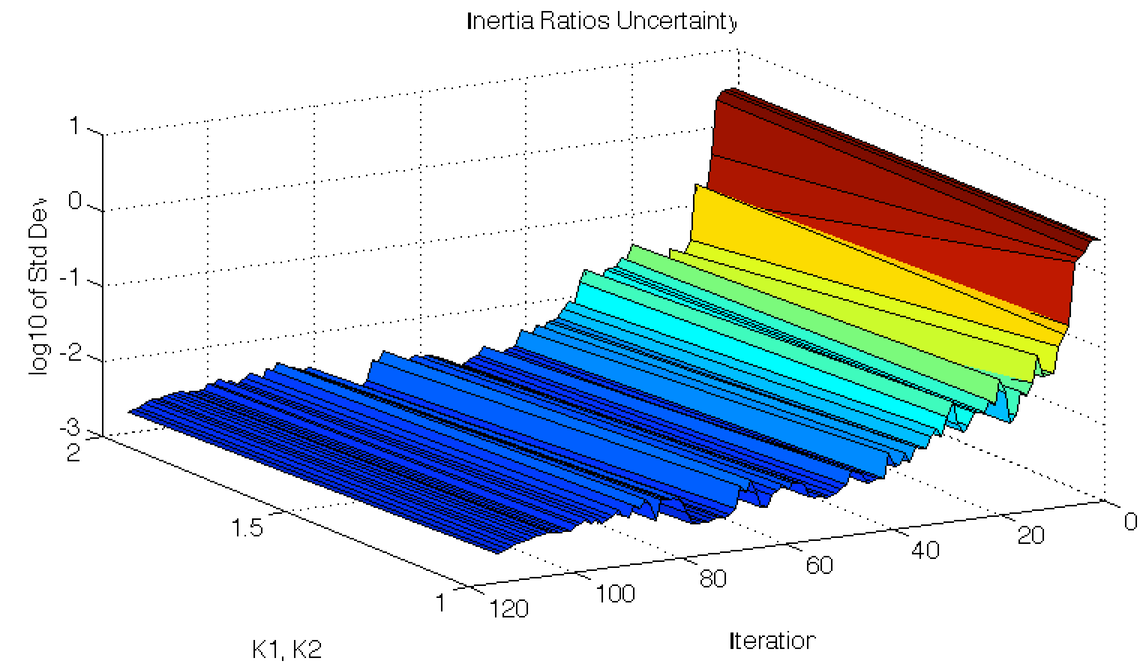


Figure 6-38: Uncertainty and Value of Principal Axes Orientation over Iterations



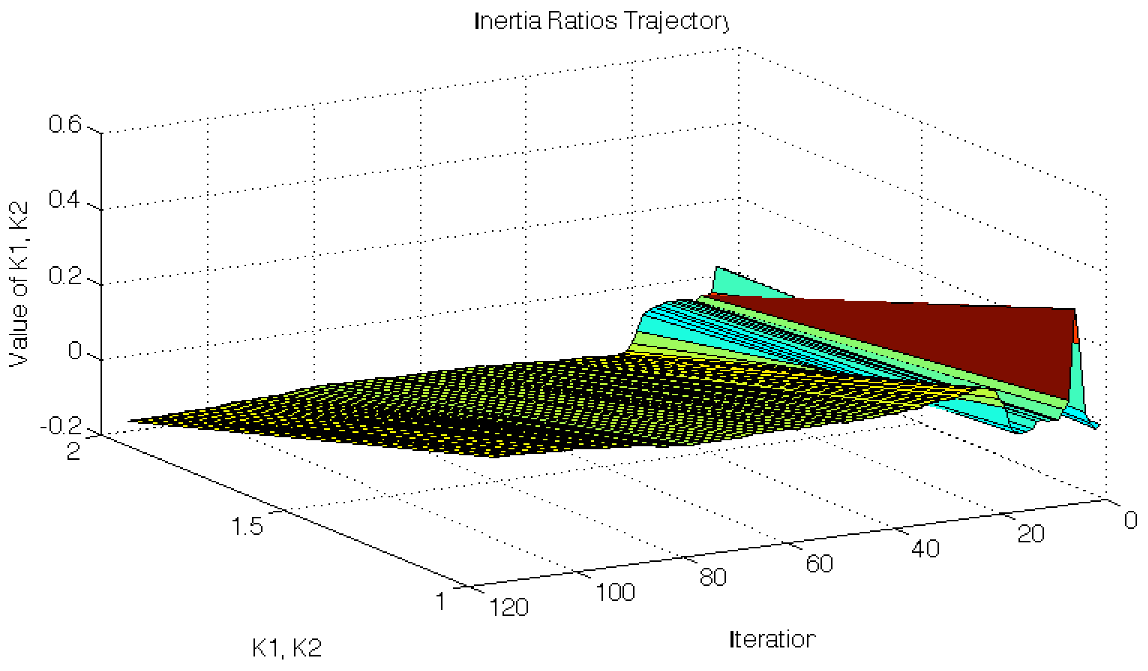


Figure 6-39: Uncertainty and Value of Inertia Ratios over Iterations

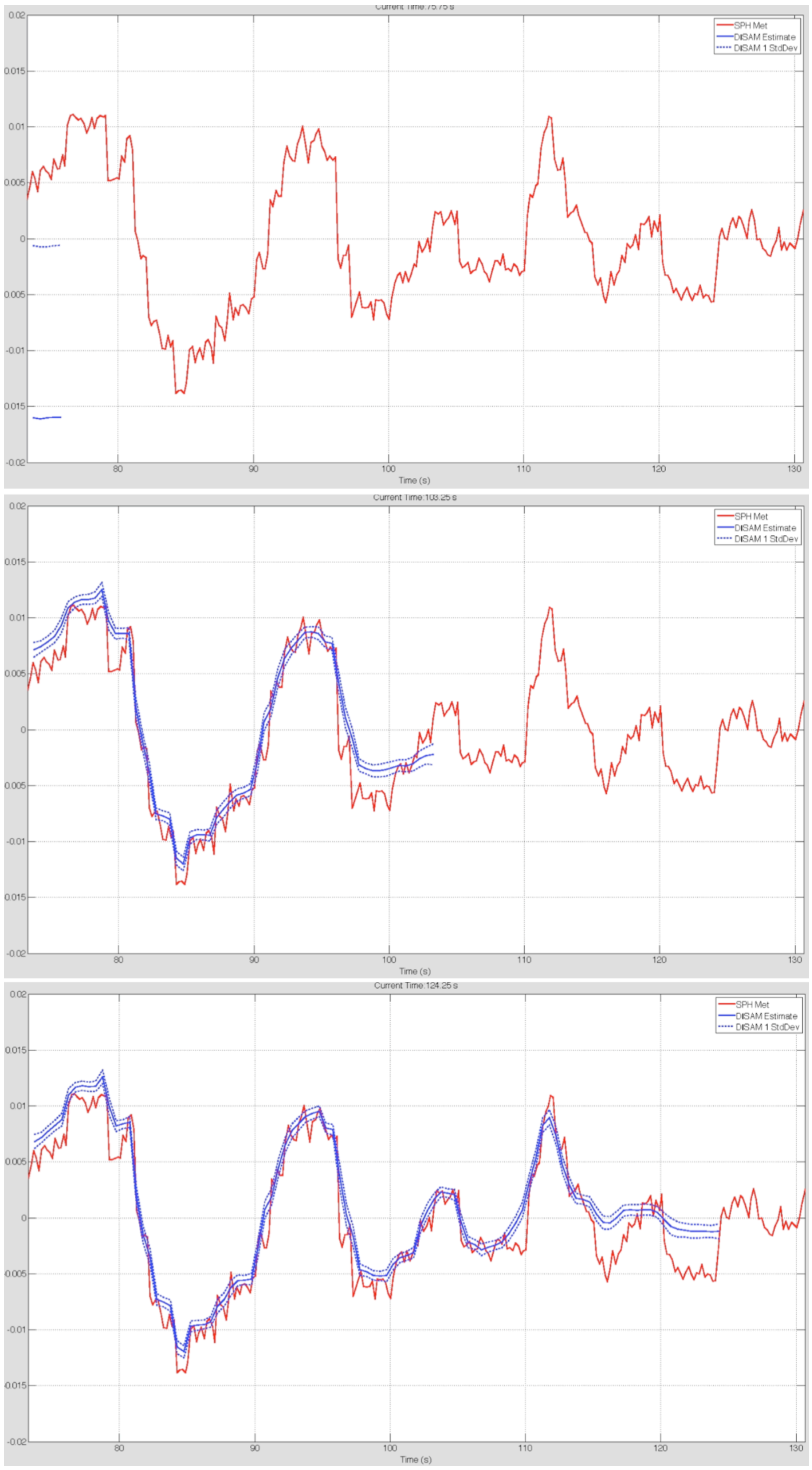


Figure 6-40: Progression of Y Velocity Trajectory Mean and Covariance over Time

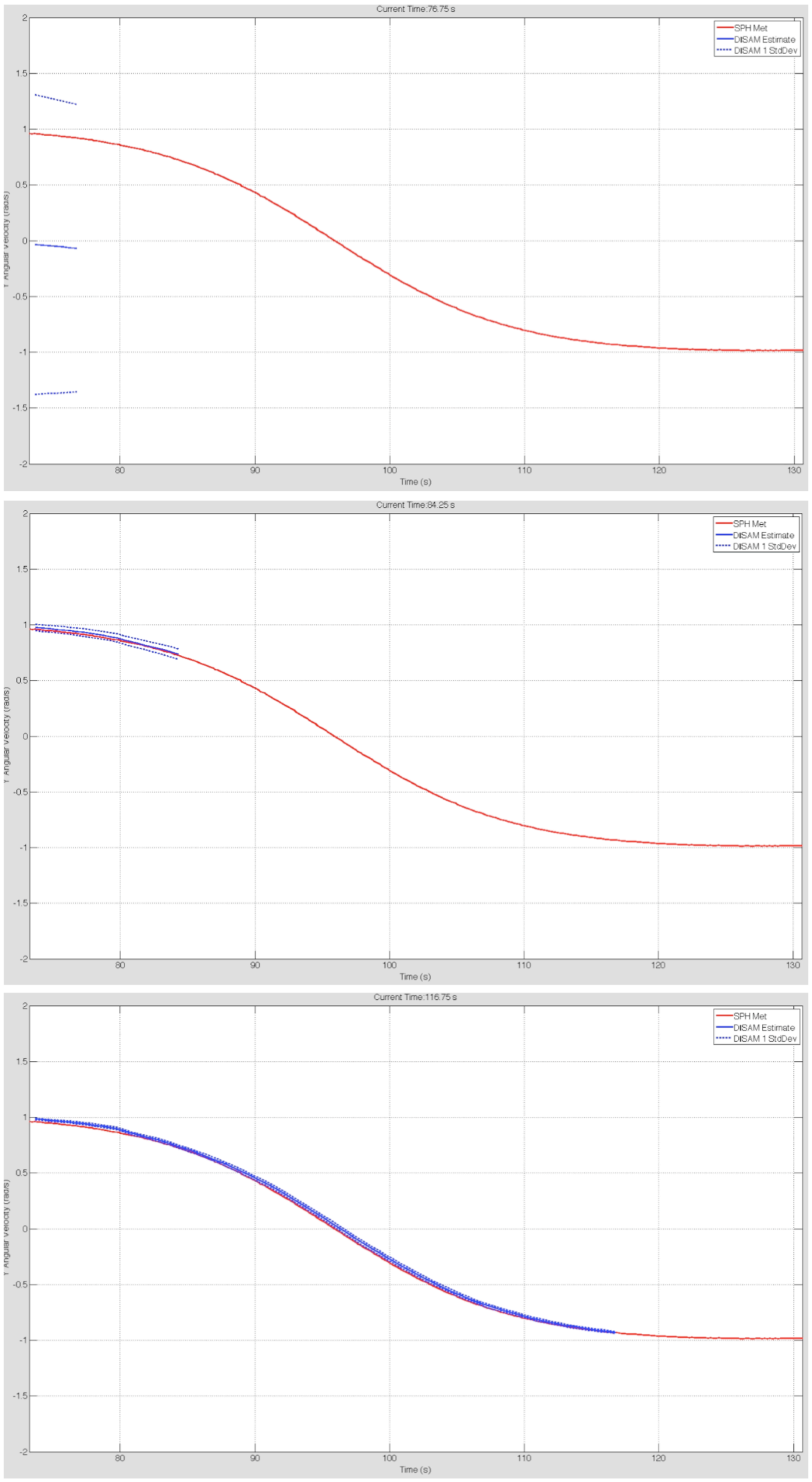


Figure 6-41: Progression of Y Angular Velocity Trajectory Mean and Covariance over Time

6.8 Computational Performance

This section provides a brief review of the computational performance of the algorithm presented in this thesis. The algorithm was implemented in the same software environment as the Goggles (i.e. Ubuntu Linux 10.04 with OpenCV and IPP), but was run on a MacBook Pro (2.3 GHz i7) under VMWare Fusion 5.3. It was written primarily in C/C++ and compiled with GCC version 4.4.3 using the -O3 flags. While the algorithm implementation was not heavily optimized and still contained debugging code, it is useful to provide the running time of the algorithm for reference purposes.

The initial stage data association step is broken into three stages, and is applied to all 115 of the stereo image pairs that were taken 0.5 seconds apart. The first stage detects features, extracts descriptors, matches features between left and right stereo frames and triangulates their location. The second step matches the features between two subsequent frames and computes the relative kinematic transformation using RANSAC and Absolute Orientation. Both of these first two steps together required 3 minutes and 16 seconds to run. The third step computes the global features as described in Muggler's thesis and in Section 6.3. This third step required 1 minute and 14 seconds to complete.

Once all of the global features were computed, the algorithm in this thesis was run by building up a factor graph model incrementally in the iSAM system. For each new pair of stereo images a new pose node was added, along with factors between the nodes and to the features. If any new features were seen for the first time they would be added to the graph as well.

Figure 6-42 shows the computational performance characteristics of the algorithm. Each of the four figures shows the number of poses along the X-axis. The upper left image shows the computational time in minutes as the number of poses grows. Note that the overall time to complete all 115 stereo pairs is just over 35 minutes, however almost all of the estimates (including the inertia parameters) had converged by the 80th image, which occurred after 15 minutes.

The upper right diagram of Figure 6-42 shows the computational time per pose that is required, which is steadily growing with each new node. This is because the batch optimization method with full relinearization is run after a set of new nodes (pose and features) is added at each timestep. The number of features and the number of overall factors is shown at each timestep in the lower two plots of Figure 6-42. There is a total of 2447 landmarks and 24723 factors at the end of the algorithm.

Since the batch optimization has to recompute all of the jacobians and covariances at each time step, it is expected that the incremental run time (i.e. the delta time) should be proportional to the number of factors in the graph. The incremental time is plotted against the number of factors in the graph in Figure 6-43. This clearly shows a that there is a linear relationship between the number of factors or poses and the computational time, which is consistent with the performance of iSAM running in batch mode[57]. As a result, any way to systematically reduce the number of factors would help the computational performance.

The results in this section do not illustrate a real time system, but rather a system whose computational time grows with the size of the problem that is better suited to offline applications. While the iSAM system can perform constant time updates, further research is needed to make sure that the incremental updates do not hurt the convergence properties. Note that if another method to solve the factor graph problem more efficiently is available (e.g. loopy belief propagation for non-linear systems), the methods in this thesis should be able to take advantage of those computational performance improvements.

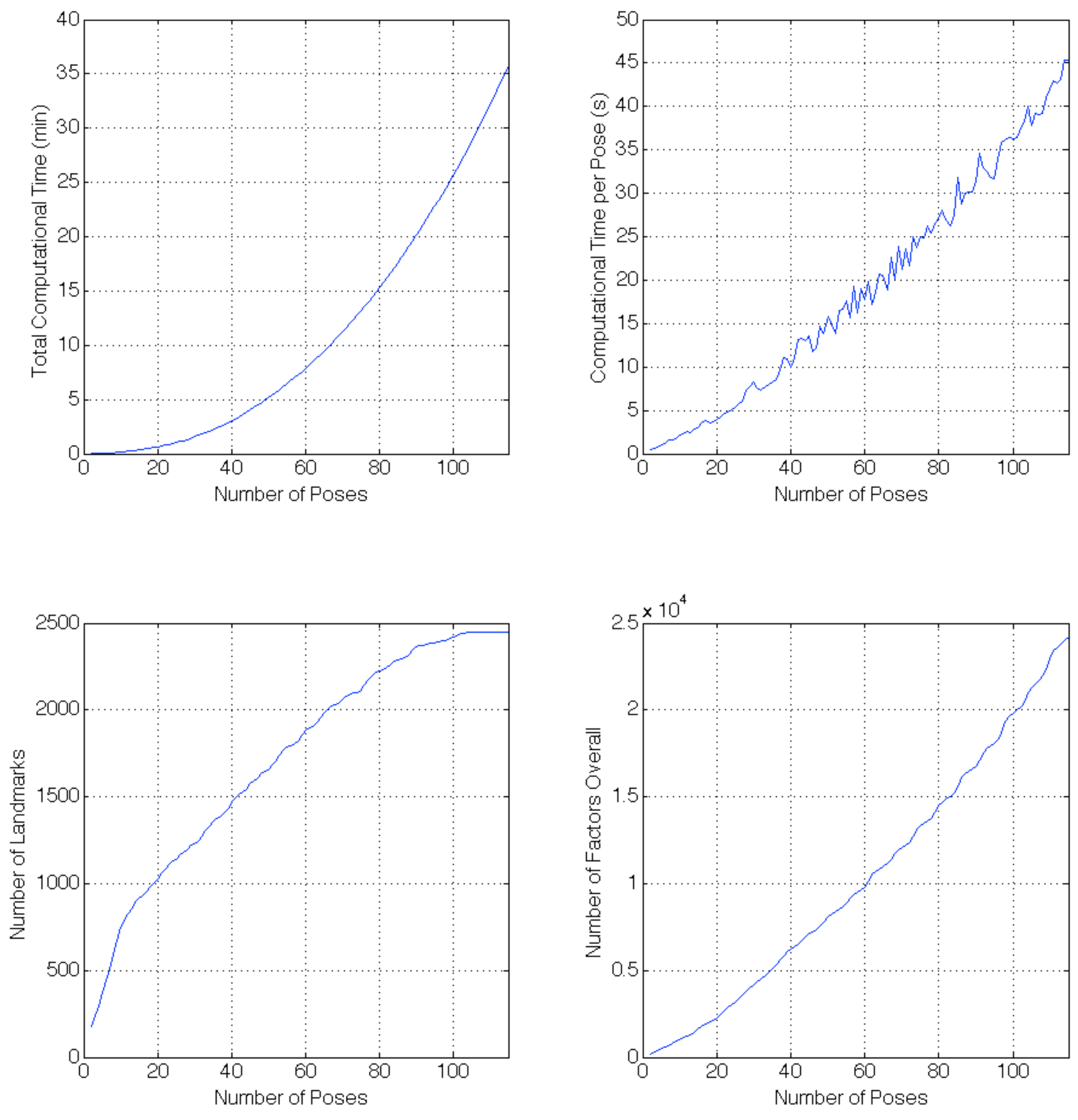
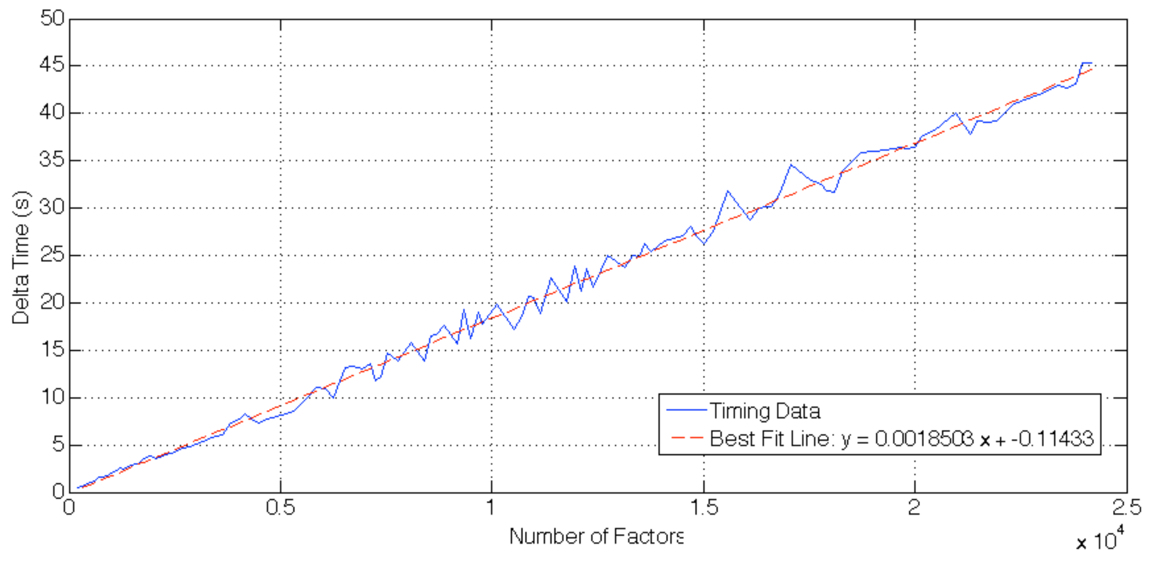


Figure 6-42: Computational Run Time of Algorithm



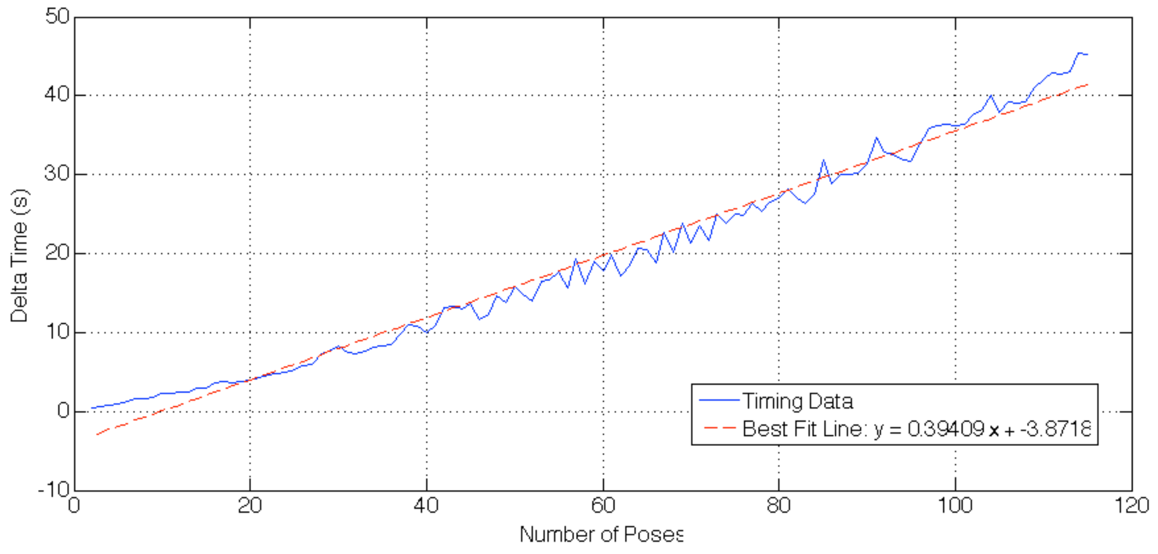


Figure 6-43: Incremental Computational Time versus Number of Factors and Poses

Chapter 7

Conclusions

In conclusion, this thesis has presented a new algorithm and approach for performing localization and mapping of an unknown, uncooperative and spinning target object that is relevant and applicable to a number of proximity operations missions. The main difference between the approach described in this thesis and other approaches available in the literature is that this thesis mathematically integrates rigid body dynamics into the probabilistic model so that dynamic quantities such as the linear and angular velocities, as well as the center of mass, principal axes of inertia and ratios of inertia can be estimated simultaneously and in a smoothing fashion with the geometric map and kinematic pose.

Additionally, this thesis presented the VERTIGO Goggles, which is an upgrade to the SPHERES satellite testbed that was designed, built, tested and operated onboard the International Space Station (ISS). It provides the capability to perform experimental evaluation of a wide range of computer vision-based navigation algorithms within the micro-gravity environment of the ISS. This testbed was used to gather a dataset for the evaluation of the new algorithm presented in this thesis. A detailed statistical comparison of this algorithm was made with respect to the SPHERES reference measurements and properties, along with a covariance analysis of the algorithm's convergence properties.

7.1 Review of Contributions

The research contributions claimed in this thesis are listed below:

1. The development of an algorithm that solves the Simultaneous Localization and Mapping problem for a spacecraft proximity operations mission where the target object may be moving, spinning and nutating.
   (a) The development of a probabilistic factor graph process model based on both rigid body kinematics and rigid body dynamics. This model constrains the position, orientation, linear velocity and angular velocity between two subsequent poses at a defined timestep according to Newton's Second Law and Euler's Equation of Rotational Motion.
   (b) The development of a parameterization approach for estimating the center of mass and principal axes of inertia by incorporating a separate geometric reference frame in which all three dimensional feature points are estimated.
   (c) The development of a two dimensional parameterization approach for estimating the natural logarithm of the ratios of inertia as Gaussian random variables, and a modification of the above process model to incorporate this.
      i. An analysis of the nonlinear observability that confirms the number of observable degrees of freedom as well as the unobservable modes.
   (d) Implementation and evaluation of above algorithm using SPHERES satellites and Goggles with approximately stationary inspector and target spinning at 10 rotations per minute about its unstable minor axis.
      i. Comparison of the above algorithm's performance to the SPHERES Global Metrology System.
      ii. Covariance and convergence analysis of the above algorithm.
2. Designed, built, tested and operated the first stereo vision-based navigation open research facility in a micro-gravity environment.

7.2 Future Work

There are a number of areas of future follow-on work that can be considered for this thesis. The first step would be to modify the algorithm to handle a moving inspector spacecraft. The implementation of the algorithm in this thesis assumed the inspector spacecraft was stationary, but this is not a representative assumption. This is not expected to be a challenging endeavor provided enough sensor measurements are available, but it should be performed in order to more fully verify this approach.

The second area of future work is to perform a detailed, comparative analysis of computational performance onboard the VERTIGO Goggles. While a brief analysis of the current research implementation was presented in this thesis, a more comprehensive analysis requires a significant amount of optimization of the source code presented in this thesis as well as further research to investigate the difference in performance using iSAM's incremental, and constant time updates. It is also important to make considerations to ensure that the evaluation is representative for real-work mission hardware applications.

The third area of future work is to implement online data association methods. The author of this thesis hypothesizes that the incorporation of linear and angular velocities will help make the data association methods more accurate, but this must be evaluated using a variety of different datasets. Since the processing time of the iSAM algorithm is known to grow with the square of the number of factors, care must be taken so that an analysis is performed that is both representative and widely applicable.

In order to perform a full comparison of the algorithms in this thesis, two comparative analyses could be performed. The first comparative analysis would be to evaluate whether or not the same parameters could be estimated by a more straightforward method. For instance if a typical SLAM algorithm was used to estimate the map and the location of inspector spacecraft, it would determine that the inspector was orbiting around the target object (when in fact the target object was spinning while the inspector was static). If the kinematics were inverted, it may be possible to estimate the equivalent properties; however, the author of this thesis hypothesizes that these estimates will not be as robust or as accurate since they do not inherently constrain the rigid body dynamics.

The second comparative analysis would be to run the above algorithm on different types of spinning motions, including those for which all of the parameters may not be observable, and compare the estimated means and covariances. It is anticipated that in situations where the full state is not observable, the incorrect value of those parameters will not have a significant impact on state propagation.

A fourth area of future work is to gather more micro-gravity datasets of spinning, tumbling and nutating motion of the all of the SPHERES satellites so that high accuracy estimates of the inertia properties (i.e. center of mass, principal axis and inertia tensor) can be obtained. This should include multiple measurements of the same satellite on different days with different tank fill levels to determine what variation this may cause.

The last area of future work for the algorithm, would be to replace the iSAM estimation engine with a message passing-based algorithm[29], in a way that is similar to Ranganathan's work [2]. This type of algorithm would likely help with computational performance as the state vector grows as well as being more suitable to optimizations using parallel-processing computational hardware. The author is not aware of any off-the-shelf implementations of these algorithms, especially those that handle non-linearities in the probabilistic model.

An area of future work for the VERTIGO Goggles is to further update and refine the operational procedures to make even more efficient use of crew time and ISS resources. This will involve reformulating the procedures for increased clarity as well as partitioning the operations to ensure crew time is only used for critical tasks.

Once the above items have been addressed, the algorithm should be implemented to run online on the VERTIGO Goggles for performing an inspection as well as localization and mapping in space. Once this is completed, it is believed that this will be the first time a Simultaneous Localization and Mapping algorithm has been run on a computer that is in space (i.e. over 100 km above sea level).

As this thesis mentioned, the VERTIGO Goggles are not a representative environment for on-orbit lighting conditions and spacecraft surface materials. This means that the image processing and feature detection and matching algorithms will likely not carry over to real world space missions. This is definitely an area of research that needs to be investigated more fully, but it does not need to be done in a micro-gravity environment. The author of this thesis believes that there is significant potential in performing matching on range images or depth maps that are gathered from either stereo, long-wavelength infrared cameras, or from Flash LIDAR sensors.

7.3 Extensions to Other Applications

The VERTIGO Goggles system was designed so that it can be extended and expanded to perform research on a number of other areas that include fluid slosh, robotic manipulation, supervised autonomy for inspection. Some of these research areas can be achieved with new software, while others may require a hardware upgrade.

One of the principal assumptions of the algorithms in this thesis is that no external forces and torques are applied to the spacecraft. Over moderately short periods of time, this is highly representative for a number of spacecraft proximity operations missions. However, most Earth-bound applications of interest and any in-space target objects that are actuating would violate this assumption. The method presented in this thesis should be able to easily extend to handling external forces and torques by adding a force and torque node and prior factor on each of the process factors shown at the bottom of Figure 4-3. If the forces and torques are unknown, a high uncertainty prior would be needed, however if measurements of the applied forces and torques are known, the prior factor could have a low uncertainty and a bias added. If force and torque measurements are available, the full inertia matrix would be observable and should have a three degree of freedom parameterization. This may be possible by added the natural logarithm of the scale factor as a Gaussian random variable.

If the above approach allows the estimation of applied forces and torques, as well the incorporation of known measurements of some of the forces and torques, this would allow for a widely applicable dynamic localization and mapping algorithm. This could handle a number of automotive, ground, underwater and aerial robotic perception applications. If some of these force and torque values were known, even with low accuracy, this could provide the basis for estimating the scale factor of the inertia matrix as well as the mass of the target object. This could provide tremendous assistance in robotic planning and manipulation of unknown objects.

Appendix A

Spinning Pliers on the International Space Station

On March 12, 2013, astronaut Kevin Ford performed a demonstration of rotational dynamics by spinning a set of diagonal pliers in the microgravity environment of the International Space Station. Since there is no gravity, the pliers “hang” in space for a long period of time where the full three-dimensional rotational motion can be accurately observed. It is very difficult to build an apparatus on earth to simulate the equivelant motion. An image from this demonstration is shown in Figure A-1.

Figure A-1: Kevin Ford Spinning a Set of Diagonal Pliers on the ISS

The reason for using the diagonal pliers to demonstrate this motion is that by changing the angle of the pliers, the mass distribution, and therefore the inertia properties can be easily changed. It is actually possible to switch the minor and intermediate spin axes as shown in Figure A-2.

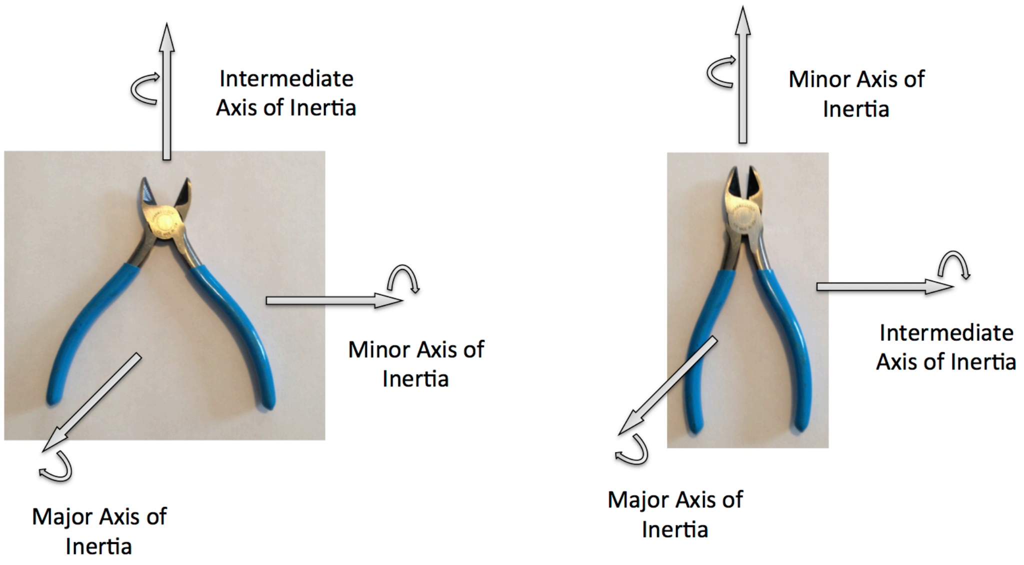


Figure A-2: Diaganol Pliers and their Adjustable Inertia Properties

The first demonstration that Ford performed was a spin about the minor axis with the pliers almost mostly closed (i.e. the right hand side of Figure A-2). It is clear from the video that the pliers maintained a spin about the minor axis, but the spin axis slowly moved in a circle, or “wobbled”. This is known as torque free precession. Because the spin is about a minor axis, this precession will occur in the same direction as the direction of spin (for a major axis spin the precession direction will be reversed). This type of motion should appear very similar to the wobble of a football that is not thrown with a perfect spiral.

The second demonstration that Ford performed was a spin about the major axis. Due to the higher inertia, this spin was visibly slower than the minor axis spin previously shown. From the video it is not clearly visible whether there was any “wobble”, but if there was, it would be in the opposite direction of the spin, which is known as retrograde torque free precession. This spin should appear to be very similar to a frisbee type of flat spin.

The third demonstration that Ford performed was a spin about the intermediate axis with the pliers mostly closed (i.e. the right hand side of Figure A-2). This motion is known to be unstable. It was initiated with a flipping motion, however, unless the flip occurs exactly about the intermediate axis ,with no motion about any other axis, the unstable dynamics will always introduce a periodic twisting motion.

The reason for this twisting motion is that the angular velocity vector will always remain stationary in an inertial frame. This is due to the conservation of angular momentum. In the case of Ford’s demonstration, the angular velocity vector is pointing towards the aft direction of the ISS and never changes while the pliers are spinning freely. However, because the intermediate axis is unstable, while the major axis is stable, the pliers try to twist over so that the major axis is aligned with the angular velocity vector. Since the system has nothing to slow it down (i.e. no mechanical damping) this twisting motion will always overshoot its target and will try over and over again in a periodic fashion to align the major axis with the angular velocity vector in the inertial frame.

After some discussion on the radio and demonstrating another minor axis spin, Ford adjusts the angle of the diagonal pliers so that they are spread out as in the left hand side of Figure A-2, thereby changing their inertial properties. Ford performs his fourth demonstration: a spin about the new intermediate axis (i.e. the old minor axis). As in the third demonstration, this spin is unstable. Note that the angular velocity vector is now pointing towards the overhead direction. If the video were rotated 90 degrees clockwise, a similar pattern to the motion would be evident. While the angular velocity vector always stays in the same direction with respect to the inertial frame of the International Space Station, the pliers try to flip in an attempt to get the stable major axis aligned with the angular velocity vector. However, they repeatedly overshoot their target and have try again and again at regular periodic intervals governed by Equation 4.86.

Ford continues to demonstrate intermediate axis spins while fine tuning the angle of the diagonal pliers, which shows that the period of the flipping can be adjusted by changing the inertial properties. Additionally, he finds the exact angle where the spin axis switches from an intermediate to a minor axis of inertia (i.e. the cutoff angle between the right and left sides of Figure A-2).

Appendix B

Source Code

B.1 Matlab Code for Tumbling versus Spinning Threshold

Listing B.1: Matlab Spinning Script

```
close all; clear all; clc;
sel = 2; %select variable
if (sel == 1)
    %SPHERES Goggles Inspection
    r = 0.7;
    m = 4.16 + 1.75;
    Fmax = 0.22;
    Isp = 37.7; %CO2
    Mf = .17/m;
    t = 5*60;
elseif (sel == 2)
    %Orbital Express
    r = 12;
    m = 900;
    Fmax = 3*3.6;
    Isp = 235;
    Mf = 5.0/m;
    t = 120*60;
end

[ w_F, w_Mf, w_min] = ...
    spinning_tumbling_threshold( r, m, Fmax, Isp, t, Mf)

radpersec2rpm = 60/(2*pi);

W_F = radpersec2rpm * w_F
W_Mf = radpersec2rpm * w_Mf
W_min = radpersec2rpm * w_min
```

Listing B.2: Matlab Tumbling Threshold

```
function [ w_F, w_Mf, w_min ] = ...
    spinning_tumbling_threshold( r, m, Fmax, Isp, t, Mf)
% INPUT PARAMETERS:
% r: station keeping radius (m)
% m: inspector satellite mass (kg)
% Fmax: maximum force inspector can apply (N)
% Isp: specific impulse of inspector propulsion system (s)
% t: duration of station keeping (s)
% Mf: maximum mass fraction ratio that can be spent on centripetal
%   force (ratio: 0 ≤ Mf ≤ 1)
%
% OUTPUT PARAMETERS:
% w_F: the angular velocity threshold that corresponds to saturating
%   the inspector propulsion system (rad/s)
% w_Mf: the angular velocity threshold that corresponds to requiring
%   a mass fraction of exactly Mf (rad/s)
% w_min: angular velocity threshold between tumbling and spinning for
%   this mission configuration (rad/s)

%%
g0 = 9.81;

w_F = sqrt(Fmax / (r*m));

w_Mf = sqrt(-(g0*Isp)/(r*t) *log(1 - Mf));

w_min = min(w_F, w_Mf);
end
```

B.2 Matlab Code for Nonlinear Observability Test of Inertia Matrix

Listing B.3: Matlab Inertia Observability

```
close all; clear all; clc;
reset(symengine);
syms wx wy wz Jx Jy Jz real
%set up variables and functions in control affine form

dwx = (Jy - Jz)/Jx * wy * wz;
dwy = (Jz - Jx)/Jy * wz * wx;
dwz = (Jx - Jy)/Jz * wx * wy;

dJx = 0;
dJy = 0;
dJz = 0;

X = [wx; wy; wz; Jx; Jy; Jz]
f = [dwx; dwy; dwz; dJx; dJy; dJz]
h = [wx; wy; wz]

%%
%compute Lie derivatives
L0_h = h
d_L0_h = jacobian(L0_h,X)

L1_h = jacobian(L0_h,X) * f
d_L1_h = jacobian(L1_h,X)

L2_h = jacobian(L1_h,X) * f
d_L2_h = jacobian(L1_h,X)

L3_h = jacobian(L2_h,X) * f
d_L3_h = jacobian(L2_h,X)

%%
% set up observability matrix
O = [d_L0_h; d_L1_h ]

%%
%compute observability properties
osize = size(O)
obs_rank = rank(O)
dim_X = size(X)

%%
%compute null space and row space
nu = null(O)
r = rref(O)
```

B.3 Bash Code for VERTIGO GUI Data Download and Reconstruction

Listing B.4: Tar and Split of Results Data Files

```bash
echo "Version 2.0 "

GID=$(cat /opt/GogglesDaemon/SIGNATURE)
TIME=$(date -u '+%Y-%m-%d_%H-%M-%S')

cd /home/
echo "Iterating through GPF_ directories... "
for DIR in `ls ./ | grep GPF | grep -v GPF_DIR`
do
    if [ -d "$DIR" ]; then
        echo $DIR
        SYSCALL="sudo mkdir "/home/Results/$DIR-$TIME""
        echo "$SYSCALL"
        $SYSCALL

        sudo mv /home/$DIR/Results/* /home/Results/$DIR-$TIME/
    fi
done

cd /home/Results/

echo "Copying Log Files... "
sudo cp -RL /opt/GogglesDaemon/GogglesLogFiles /home/Results

echo "Creating Tar File... "
sudo tar cf /home2/TempResults/Vertigo_Data.tar *

echo "Splitting Tar File... "
sudo split -d --suffix-length=5 --bytes=50M /home2/TempResults/Vertigo_Data.tar /home/
    TempResults/Vertigo_Data_$TIME"_GID_"$GID"_"
sudo rm /home2/TempResults/Vertigo_Data.tar

for f in /home/TempResults/Vertigo_Data_*
do
    sudo mv $f $f".sdf"
done

echo "Computing CheckSum... "
sudo md5sum /home/TempResults/* > /home/TempResults/Vertigo_Data_$TIME"_GID_"$GID"_MD5
    .sdf"
```

Listing B.5: FTP Download Command

```
goggles
vertigo1
cd /home/TempResults
binary
prompt n
lcd Q:\SPHERES\Data
mget Vertigo_Data_*
quit
```

Listing B.6: FTP Download Command

```
#!/bin/bash
echo VERTIGO MAINTENTANCE
echo "*** Check Data Files"
DIR=$1
OLD_DIR=$(pwd)
cd $DIR
pwd
md5sum -c Vertigo_Data_*_MD5.sdf
cd $OLD_DIR
echo VERTIGO MAINTENTANCE COMPLETE
```

Listing B.7: FTP Download Command

```
#!/bin/bash
echo VERTIGO MAINTENTANCE
echo "*** Rebuild Data Files"
echo "Use only 1 set of split files in this directory"
DIR=$1
OLD_DIR=$(pwd)
cd $DIR
pwd
cat Vertigo_Data_*_GID_*_0*.sdf > Vertigo_Data.tar
tar --ignore-failed-read -xvif Vertigo_Data.tar &> untar_output.txt
rm Vertigo_Data.tar
cd $OLD_DIR
echo VERTIGO MAINTENTANCE COMPLETE
```

B.4 Goggles Guest Scientist Code for Camera Calibration

Listing B.8: Camera Calibration: testproject.cpp

breaklines

```
#include "testproject.h"

#include <iostream>

using namespace std;

testproject *test = NULL;

void terminateHandlerHelper(int sig)
{
    test->terminateHandler(sig);
}

void preInit()
{
    test = new testproject;

    signal(SIGTERM, terminateHandlerHelper);
    signal(SIGABRT, terminateHandlerHelper);
    signal(SIGINT, terminateHandlerHelper);

}



int main (int argc, char *argv[])
{
    preInit();

    test->runMain(argc, argv);

    delete test;
    return 0;

}
```

Listing B.9: Camera Calibration: testproject.h

```
#ifndef _TESTPROJECT_H_
#define _TESTPROJECT_H_

#include "gogglesGSP.h"
#include "GSsupportingFiles/stereoCalib.h"
#include "GSsupportingFiles/additionalGuestscientistCode.h"
//#include "GSsupportingFiles/reproject.h"
#include <stereoSAD.h>
#include <iostream>
#include <iomanip>
#include <sstream>
#include <string>
#include <cstring>
#include <math.h>
#include <fstream>
#include <string>
#include <dirent.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <sys/stat.h>

#include <opencv2/opencv.hpp>

#define CALIBRATION_WINDOW_NAME "Camera Calibration"
//#define CHESS_WIDTH   8
//#define CHESS_HEIGHT  6
#define CHESS_WIDTH     11
#define CHESS_HEIGHT    6

#define GOOD_COLOR          CV_RGB(0x00,0xFF,0x00)
#define BAD_COLOR           CV_RGB(0xFF,0x7F,0x00)
#define NORMAL_COLOR        CV_RGB(0xFF,0xFF,0xFF)
#define GUIDE_COLOR         CV_RGB(0x00,0xFF,0x00)
#define S_LINE_COLOR        CV_RGB(0x00,0xFF,0xFF)
#define BKGD_COLOR          CV_RGB(0xFF,0xFF,0xFF)
#define TEXT_COLOR          CV_RGB(0x00,0x00,0x00)

#define MINVAL 0
#define MAXVAL 1


typedef struct {
    float horz_translation[2];
    float vert_translation[2];
    float depth_translation[2];
    float rot_roll[2];
    float rot_pitch[2];
    float rot_yaw[2];
    float focal_len[2];
    float square_size[2];
} limit_struct;

#define inRange(limits,value)   limits[MINVAL] ≤ value && value ≤ limits[MAXVAL]

using namespace cv;

class testproject : public gogglesGSP {

    // own declaration go here
    char key;
    int numGoodImages;
    int numImageSet;
    int maneuverNumber;
    int maxCalibrationIterations;
    double maxCalibrationChange;
    FileStorage fs;
    int minimumNumberImages;
    stringstream ImgListFilename;

    pthread_mutex_t keymutex;
    unsigned char charKey;

    bool performedCalibration;
    bool useImagesFromFile;

    string currentCalibImageDir;
    GuestScientistClass guestscientistclass;

    Rectifier newRectifier;

//    bool useReprojection;
//    ReprojectClass reprojection;
    bool runningReproject;

    limit_struct limits;

    Mat smallImg;

    //video buffers
    MatVideoBuffer checkoutCalibrationVideoBuffer;
    MatVideoBuffer goodExampleBuffer;
    MatVideoBuffer badExampleBuffer;

    pthread_mutex_t backgroundMutex;
    bool computeCalibrationBackground;
    bool runningCalibration;

    //video saving
    bool saveflag;
    int savedircount, saveimagecount;
    string saveImageDir;

    //stored videos
    vector<Mat> imgVec_goodEx_capCalib;
    vector<Mat> imgVec_badEx_capCalib;
    vector<Mat> imgVec_goodEx_metrics;
    vector<Mat> imgVec_badEx_metrics;
    vector<Mat> imgVec_goodEx_poseEst;
    vector<Mat> imgVec_badEx_poseEst;
    vector<Mat> imgVec_goodEx_depth;
    vector<Mat> imgVec_badEx_depth;

    int storedFrameCountGood, storedFrameCountBad;

    string dir_storedImages_goodEx_capCalib;
    string dir_storedImages_badEx_capCalib;
    string dir_storedImages_goodEx_metrics;
    string dir_storedImages_badEx_metrics;
    string dir_storedImages_goodEx_poseEst;
    string dir_storedImages_badEx_poseEst;
    string dir_storedImages_goodEx_depth;
    string dir_storedImages_badEx_depth;

public:

    testproject () {
        numGoodImages = 0;
        numImageSet = 0;
        maneuverNumber = 4;
        minimumNumberImages = 60;
        newRectifier = Rectifier();
        maxCalibrationIterations = 30;
        maxCalibrationChange = 1.0e-6;

        //default limits
        limits.horz_translation[MAXVAL] = -8.60;
        limits.horz_translation[MINVAL] = -9.40;
        limits.vert_translation[MAXVAL] = 0.50;
        limits.vert_translation[MINVAL] = -0.50;
        limits.depth_translation[MAXVAL] = 0.50;
        limits.depth_translation[MINVAL] = -0.50;

        limits.rot_roll[MAXVAL] = 1.0;
        limits.rot_roll[MINVAL] = -1.0;
        limits.rot_pitch[MAXVAL] = 1.0;
        limits.rot_pitch[MINVAL] = -1.0;
        limits.rot_yaw[MAXVAL] = 1.0;
        limits.rot_yaw[MINVAL] = -1.0;

        limits.focal_len[MAXVAL] = 3.0;
        limits.focal_len[MINVAL] = 2.7;
        limits.square_size[MAXVAL] = 2.8;
        limits.square_size[MINVAL] = 2.3;

        pthread_mutex_init(&keymutex, NULL);
        pthread_mutex_init(&backgroundMutex,NULL);

        performedCalibration = false;


        computeCalibrationBackground = false;
        runningCalibration = false;

        useImagesFromFile = false;

        saveflag = false;
        savedircount = 0;
        saveimagecount = 0;

        storedFrameCountGood = 0;
        storedFrameCountBad = 0;

        saveimagecount = 0;


    }

    void loadStoredImages(string directory, vector<Mat>& img_vec) {
        stringstream tempfilename;
        Mat tempmat;
        tempfilename.str("");
        for (int i = 0; i < 2048; i++) {
            tempfilename << directory << "/img" << i << ".jpg";
            tempmat = imread(tempfilename.str());
            if (tempmat.empty()) {
                //cout << "Number: " << i << " is empty." << endl;
                break;
            } else {
                img_vec.push_back(tempmat);
            }
            tempfilename.str("");
        }
        cout << directory << " - Number of files: " << img_vec.size() << endl;
    }

    void GSsetup(){

////////// Guest Scientist initialization calls go here
//      namedWindow(CALIBRATION_WINDOW_NAME, CV_WINDOW_AUTOSIZE);

//      cout << "This Program is an example TestProject" << endl;

        string dataPath = this->datastorage.getGSdatastoragePath();


        cout << "Data Location: " << dataPath << endl;

        rectifier.rectifierOn = false;

        cout << cameras.getExposureTime() << "   " << cameras.getFrameRate() << endl;

        rectifier.calcRectificationMaps(this->cameras.getImageWidth(), this->cameras.
            getImageHeight(), this->calibParamDir);
        newRectifier.calcRectificationMaps(this->cameras.getImageWidth(), this->
            cameras.getImageHeight(), this->calibParamDir);
        guestscientistclass.updatePhotogrammetry(newRectifier);

        videostreaming.createNew_MatVideoBuffer(checkoutCalibrationVideoBuffer, "
            Checkout & Calibration");
        videostreaming.setAsDefaultVideoMode(checkoutCalibrationVideoBuffer);
        videostreaming.createNew_MatVideoBuffer(goodExampleBuffer, "Good Example (KEYS
             DISABLED)");
        videostreaming.createNew_MatVideoBuffer(badExampleBuffer, "Bad Example (KEYS
            DISABLED)");


        useBackgroundTask = true;

        loadStoredImages(this->dir_storedImages_goodEx_capCalib, this->
            imgVec_goodEx_capCalib);
        loadStoredImages(this->dir_storedImages_badEx_capCalib, this->
            imgVec_badEx_capCalib);
        loadStoredImages(this->dir_storedImages_goodEx_metrics, this->
            imgVec_goodEx_metrics);
        loadStoredImages(this->dir_storedImages_badEx_metrics, this->
            imgVec_badEx_metrics);
        loadStoredImages(this->dir_storedImages_goodEx_poseEst, this->
            imgVec_goodEx_poseEst);
        loadStoredImages(this->dir_storedImages_badEx_poseEst, this->
            imgVec_badEx_poseEst);
        loadStoredImages(this->dir_storedImages_goodEx_depth, this->
            imgVec_goodEx_depth);
        loadStoredImages(this->dir_storedImages_badEx_depth, this->imgVec_badEx_depth)
            ;

        saveImageDir = this->datastorage.getGSdatastoragePath();

    }

    void setupCalibrationCapture() {
        char newdirname[200];
        sprintf(newdirname, "ImageSet%d", numImageSet);

        if (this->useImagesFromFile == false) {
            currentCalibImageDir = this->datastorage.newGSDataDirectory(newdirname);
        } else {
            cout << "Current Calib Name: " << currentCalibImageDir << endl;
            printf("Directories for image only mode not yet implemented\n");
        }

        ImgListFilename.str("");
        ImgListFilename << currentCalibImageDir << "/imglist.yaml";
        fs = FileStorage(ImgListFilename.str(), FileStorage::WRITE);
        fs << "images" << "[";

    }

    void captureCalibrationImages(Mat& leftImage, Mat& rightImage) {
        Mat outImg;
        vector<Point2f> pointbuf1, pointbuf2;
        stringstream filename1, filename2;
        stringstream filename3;

        equalizeHist(leftImage, leftImage);
        equalizeHist(rightImage, rightImage);

        Size size( leftImage.cols + rightImage.cols, MAX(leftImage.rows, rightImage.
            rows) + 18 );

        //place two images side by side
        outImg.create( size, CV_MAKETYPE(leftImage.depth(), 3) );
        outImg = BKGD_COLOR;
        Mat outImgLeft = outImg( Rect(0, 0, leftImage.cols, leftImage.rows) );
        Mat outImgRight = outImg( Rect(leftImage.cols, 0, rightImage.cols, rightImage.
            rows) );

        cvtColor( leftImage, outImgLeft, CV_GRAY2BGR );
        cvtColor( rightImage, outImgRight, CV_GRAY2BGR );

        //find chessboard
        bool patternfoundLeft = findChessboardCorners(leftImage, Size(CHESS_WIDTH,
            CHESS_HEIGHT), pointbuf1, CV_CALIB_CB_ADAPTIVE_THRESH +
            CV_CALIB_CB_FAST_CHECK /*+ CV_CALIB_CB_NORMALIZE_IMAGE*/ +
            CV_CALIB_CB_FILTER_QUADS);
        bool patternfoundRight = findChessboardCorners(rightImage, Size(CHESS_WIDTH,
            CHESS_HEIGHT), pointbuf2, CV_CALIB_CB_ADAPTIVE_THRESH +
            CV_CALIB_CB_FAST_CHECK /*+ CV_CALIB_CB_NORMALIZE_IMAGE*/ +
            CV_CALIB_CB_FILTER_QUADS);

        if(patternfoundLeft)
          cornerSubPix(leftImage, pointbuf1, Size(5, 5), Size(-1, -1), TermCriteria(
              CV_TERMCRIT_EPS + CV_TERMCRIT_ITER, 30, 0.1));

        if(patternfoundRight)
          cornerSubPix(rightImage, pointbuf2, Size(5, 5), Size(-1, -1), TermCriteria(
              CV_TERMCRIT_EPS + CV_TERMCRIT_ITER, 30, 0.1));

        drawChessboardCorners(outImgLeft, Size(CHESS_WIDTH,CHESS_HEIGHT), Mat(
            pointbuf1), patternfoundLeft);
        drawChessboardCorners(outImgRight, Size(CHESS_WIDTH,CHESS_HEIGHT), Mat(
            pointbuf2), patternfoundRight);

        //flip images about vertical axis
        flip(outImgLeft, outImgLeft,1);
        flip(outImgRight, outImgRight,1);

        //display text
        Point org;
//      CvScalar textcolor;
        if (numGoodImages ≥ minimumNumberImages) {
            //Finish
            org.x = 1020;
            org.y = leftImage.rows + 13;
            stringstream finishText;
//          textcolor = CV_RGB(0,255,0);
            finishText << "PRESS F TO FINISH CAPTURING";
            putText(outImg, finishText.str(), org, FONT_HERSHEY_SIMPLEX,
                    0.5,        //fontScale (double)
                    TEXT_COLOR,     //color
                    1,          //thickness (
                    CV_AA,          //linetype
                    false);     //bottom left origin

            circle(outImg, Point(240,leftImage.rows+8), 5,GOOD_COLOR,-1, CV_AA);

        } else {
            circle(outImg, Point(240, leftImage.rows+8), 5,BAD_COLOR,-1, CV_AA);
//          textcolor = CV_RGB(0,255,255);
        }

        //number of images
        org.x = 10;
        org.y = leftImage.rows + 13;
        stringstream imgCntText;
        imgCntText << "NUMBER OF IMAGES: " << numGoodImages << "/" <<
            minimumNumberImages;
        putText(outImg, imgCntText.str(), org, FONT_HERSHEY_SIMPLEX,
                0.5,        //fontScale (double)
                TEXT_COLOR,     //color
                1,          //thickness (
                CV_AA,          //linetype
                false);     //bottom left origin

        //spacebar
        org.x = 480;
        org.y = leftImage.rows + 13;
        stringstream spacebarText;
        spacebarText << "PRESS SPACEBAR TO CAPTURE IMAGES";
        putText(outImg, spacebarText.str(), org, FONT_HERSHEY_SIMPLEX,
                0.5,        //fontScale (double)
                TEXT_COLOR,     //color
                1,          //thickness (
                CV_AA,          //linetype
                false);     //bottom left origin

        //// Display Rectified Images
        videostreaming.update_MatVideoBuffer(checkoutCalibrationVideoBuffer, outImg);

        videostreaming.update_MatVideoBuffer(goodExampleBuffer, this->imgVec_goodEx_capCalib[storedFrameCountGood]);
        videostreaming.update_MatVideoBuffer(badExampleBuffer, this->imgVec_badEx_capCalib[storedFrameCountBad]);
        if (goodExampleBuffer.active)
            storedFrameCountGood = (storedFrameCountGood + 1) % this->imgVec_goodEx_capCalib.size();
        if (badExampleBuffer.active)
            storedFrameCountBad = (storedFrameCountBad + 1) % this->imgVec_badEx_capCalib.size();

        if (saveflag) {
            filename3 << saveImageDir << "/displayImg" << saveimagecount++ << ".png";
            imwrite(filename3.str(), outImg);
            //printf("Saved File: %s\n", filename3.str().c_str());

        }

        pthread_mutex_lock(&keymutex);
        if (charKey == 0x1B) { // ESC
            printf("Quitting...\n");
            shutdownCriterion = true;
        } else if (charKey == 0x20 && checkoutCalibrationVideoBuffer.active) { // Spacebar

            if (patternfoundLeft && patternfoundRight) {
                numGoodImages++;
                filename1 << currentCalibImageDir << "/Left" << numGoodImages << ".bmp";
                filename2 << currentCalibImageDir << "/Right" << numGoodImages << ".bmp";
                imwrite(filename1.str(), leftImage);
                imwrite(filename2.str(), rightImage);
                printf("Saved: %s, %s\n", filename1.str().c_str(), filename2.str().c_str());

                stringstream leftName, rightName;
                leftName <<  "Left" << numGoodImages << ".bmp";
                rightName << "Right" << numGoodImages << ".bmp";
                fs << leftName.str();
                fs << rightName.str();
            }

        } else if (numGoodImages ≥ minimumNumberImages && (charKey == 0x46 || charKey == 0x66)&& checkoutCalibrationVideoBuffer.active) { //f or F
            fs << "]";
            fs.release();
            maneuverNumber = 3;
            storedFrameCountGood = 0;
            storedFrameCountBad = 0;
        }
        charKey = 0x00;
        pthread_mutex_unlock(&keymutex);
    }

    void displayComputeCalibration(Mat& leftImage, Mat& rightImage) {
        Mat outImg;
        Size size( leftImage.cols + rightImage.cols, MAX(leftImage.rows, rightImage.rows) + 18 );
        stringstream filename3;

        //place two images side by side
        outImg.create( size, CV_MAKETYPE(leftImage.depth(), 3) );
        outImg = BKGD_COLOR;
        //Finish
        stringstream waitText;
        waitText << "Computing Camera Calibration";
        putText(outImg, waitText.str(), Point(30,100), FONT_HERSHEY_SIMPLEX,1.0, TEXT_COLOR,2,CV_AA,false);
        waitText.str("");
        waitText << "Please Wait...";
        putText(outImg, waitText.str(), Point(30,200), FONT_HERSHEY_SIMPLEX,1.0, TEXT_COLOR,2,CV_AA,false);

        Mat outImgRight = outImg( Rect(leftImage.cols, 0, rightImage.cols, rightImage.rows) );
        cvtColor( rightImage, outImgRight, CV_GRAY2BGR );

        videostreaming.update_MatVideoBuffer(checkoutCalibrationVideoBuffer, outImg);

        if (saveflag) {
            //too much data
            //filename3 << saveImageDir << "/displayImg" << saveimagecount++ << ".bmp
                ";
            //imwrite(filename3.str(), outImg);
            //printf("Saved File: %s\n", filename3.str().c_str());

        }

        pthread_mutex_lock(&keymutex);
        if (charKey == 0x1B) { // ESC
            printf("Quitting...\n");
            shutdownCriterion = true;
        }
        charKey = 0x00;
        pthread_mutex_unlock(&keymutex);
    }

    void computeCalibration(Mat& leftImage, Mat& rightImage) {

        displayComputeCalibration(leftImage,rightImage);

        pthread_mutex_lock(&backgroundMutex);
        if (computeCalibrationBackground == false && runningCalibration == false) {
            //not yet started
            std::time_t result = std::time(NULL);
            std::cout << "Starting Calibration Computation: " << std::asctime(std::
                localtime(&result));
            computeCalibrationBackground = true;
            runningCalibration = true;
        } else if (runningCalibration = true && computeCalibrationBackground == false)
             {
            //run complete - move on to next maneuver
            runningCalibration = false;
            performedCalibration = true;
            maneuverNumber = 4;
            std::time_t result = std::time(NULL);
            std::cout << "Completed Calibration Computation: " << std::asctime(std::
                localtime(&result));

        } else if (runningCalibration = true && computeCalibrationBackground == true)
            {
            //middle of run
            //do nothing
        }
        pthread_mutex_unlock(&backgroundMutex);
```

```
}

    void GSbackgroundTask() {
//      printf("Background task\n");
        pthread_mutex_lock(&backgroundMutex);
        if (computeCalibrationBackground) {
            pthread_mutex_unlock(&backgroundMutex);
            Size boardSize;
//          printf("Background task is calibrating\n");
            Size imageSize;
            string imagelistfn;
            vector<string> imagelist;
            vector<vector<Point2f> > imagePoints[2];
            vector<vector<Point3f> > objectPoints;
            vector<string> goodImageList;

            float rms_error, reprojection_error;
            int nimages;

            boardSize = Size(CHESS_WIDTH, CHESS_HEIGHT);

            printf("Calibrating(%s)...\n", ImgListFilename.str().c_str());
            bool ok = initStereoCalib(ImgListFilename.str(), imagelist,
                currentCalibImageDir,
                    boardSize, imagePoints, objectPoints, imageSize, goodImageList, &
                        nimages);


            boardSize = Size(CHESS_WIDTH,CHESS_HEIGHT);
            StereoCalib(imagelist, boardSize, /*true, false,*/ rectifier,
                currentCalibImageDir,
                    imagePoints, objectPoints, imageSize, goodImageList,
                    maxCalibrationIterations, maxCalibrationChange,
                    &rms_error, &reprojection_error);

            rectifier.calcRectificationMaps(this->cameras.getImageWidth(), this->
                cameras.getImageHeight(), this->calibParamDir);
            newRectifier.calcRectificationMaps(this->cameras.getImageWidth(), this->
                cameras.getImageHeight(), currentCalibImageDir.c_str());
            guestscientistclass.updatePhotogrammetry(newRectifier);

//          printf("Completed Calibration in Thread\n");

            pthread_mutex_lock(&backgroundMutex);
            computeCalibrationBackground = false;
        }
        pthread_mutex_unlock(&backgroundMutex);
    }

    void displayCalibrationMetrics(Mat& leftImage, Mat& rightImage) {
        Mat outImg;
        Mat Rnew, Tnew, M1new, D1new, M2new, D2new;
        Mat Rold, Told, M1old, D1old, M2old, D2old;
        double roll, pitch, yaw;
        Size size( leftImage.cols + rightImage.cols, MAX(leftImage.rows, rightImage.
            rows) + 18 );
        int leftboundary = 20;
        int yoffset = 70;
        stringstream filename3;

        //place two images side by side
        outImg.create( size, CV_MAKETYPE(leftImage.depth(), 3) );
        outImg = BKGD_COLOR;
        //Finish
        stringstream waitText;
        waitText << "Camera Calibration Values: ";
        putText(outImg, waitText.str(), Point(leftboundary,50), FONT_HERSHEY_SIMPLEX
            ,1.0, TEXT_COLOR,2,CV_AA,false);

        rectifier.getCameraParameters(Rold, Told, M1old, D1old, M2old, D2old);
        newRectifier.getCameraParameters(Rnew, Tnew, M1new, D1new, M2new, D2new);


        waitText.str("");
        waitText << std::fixed << std::setprecision(2) << "Translation between Cameras
            :";
        putText(outImg, waitText.str(), Point(leftboundary+20,yoffset+30),
            FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);

        waitText.str("");
        waitText << std::fixed << std::setprecision(2) << "Horizontal: "
                << 2.54*Tnew.at<double>(0,0) << " cm";
        putText(outImg, waitText.str(), Point(leftboundary+40,yoffset+50),
            FONT_HERSHEY_SIMPLEX,0.5, TEXT_COLOR,1,CV_AA,false);
        circle(outImg, Point(leftboundary+30,yoffset+45), 5,
                inRange(limits.horz_translation, 2.54*Tnew.at<double>(0,0)) ?
                    GOOD_COLOR : BAD_COLOR,
                -1, CV_AA);

        waitText.str("");
        waitText << std::fixed << std::setprecision(2) << "Vertical:    "
                << 2.54*Tnew.at<double>(1,0) << " cm";
        putText(outImg, waitText.str(), Point(leftboundary+40,yoffset+70),
            FONT_HERSHEY_SIMPLEX,0.5, TEXT_COLOR,1,CV_AA,false);
        circle(outImg, Point(leftboundary+30,yoffset+65), 5,
                inRange(limits.vert_translation, 2.54*Tnew.at<double>(1,0)) ?
                    GOOD_COLOR : BAD_COLOR,
                -1, CV_AA);

        waitText.str("");
        waitText << std::fixed << std::setprecision(2) << "Depth:       "
                << 2.54*Tnew.at<double>(2,0) << " cm";
        putText(outImg, waitText.str(), Point(leftboundary+40,yoffset+90),
            FONT_HERSHEY_SIMPLEX,0.5,TEXT_COLOR ,1,CV_AA,false);
        circle(outImg, Point(leftboundary+30,yoffset+85), 5,
                inRange(limits.depth_translation, 2.54*Tnew.at<double>(2,0)) ?
                    GOOD_COLOR : BAD_COLOR,
                -1, CV_AA);


        yaw = atan(Rnew.at<double>(0,1)/Rnew.at<double>(1,1))*180 / M_PI;
        pitch = -asin(Rnew.at<double>(2,1))*180 / M_PI;
        roll = atan(Rnew.at<double>(2,0)/Rnew.at<double>(2,2))*180 / M_PI;
        waitText.str("");
        waitText << std::fixed << std::setprecision(2) << "Rotation between Cameras:";
        putText(outImg, waitText.str(), Point(leftboundary+20,yoffset+110),
            FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);

        waitText.str("");
        waitText << std::fixed << std::setprecision(2) << "Roll:  "
                << roll << " degrees";
        putText(outImg, waitText.str(), Point(leftboundary+40,yoffset+130),
            FONT_HERSHEY_SIMPLEX,0.5,TEXT_COLOR,1,CV_AA,false);
        circle(outImg, Point(leftboundary+30,yoffset+125), 5,
                inRange(limits.rot_roll, roll) ? GOOD_COLOR : BAD_COLOR,
                -1, CV_AA);

        waitText.str("");
        waitText << std::fixed << std::setprecision(2) << "Pitch: "
                << pitch << " degrees";
        putText(outImg, waitText.str(), Point(leftboundary+40,yoffset+150),
            FONT_HERSHEY_SIMPLEX,0.5,TEXT_COLOR,1,CV_AA,false);
        circle(outImg, Point(leftboundary+30,yoffset+145), 5,
                inRange(limits.rot_pitch, pitch) ? GOOD_COLOR : BAD_COLOR,
                -1, CV_AA);

        waitText.str("");
        waitText << std::fixed << std::setprecision(2) << "Yaw:   "
                << yaw << " degrees";
        putText(outImg, waitText.str(), Point(leftboundary+40,yoffset+170),
            FONT_HERSHEY_SIMPLEX,0.5,TEXT_COLOR,1,CV_AA,false);
        circle(outImg, Point(leftboundary+30,yoffset+165), 5,
                inRange(limits.rot_yaw, yaw) ? GOOD_COLOR : BAD_COLOR,
                -1, CV_AA);

        waitText.str("");
        waitText << std::fixed << std::setprecision(2) << "Camera Lens Parameters:";
        putText(outImg, waitText.str(), Point(leftboundary+20,yoffset+190),
            FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);

        waitText.str("");
        waitText << std::fixed << std::setprecision(2) << "Focal Length: "
                << M1new.at<double>(0,0)*0.006 << " mm";
        putText(outImg, waitText.str(), Point(leftboundary+40,yoffset+210),
            FONT_HERSHEY_SIMPLEX,0.5,TEXT_COLOR,1,CV_AA,false);
        circle(outImg, Point(leftboundary+30,yoffset+205), 5,
                inRange(limits.focal_len, M1new.at<double>(0,0)*0.006) ? GOOD_COLOR :
                    BAD_COLOR,
                -1, CV_AA);


        waitText.str("");
        waitText << "PRESS N FOR NEXT SCREEN";
        putText(outImg, waitText.str(), Point(1020,leftImage.rows + 13),
            FONT_HERSHEY_SIMPLEX,0.5, TEXT_COLOR,1,CV_AA,false);

        waitText.str("");
        waitText << "PRESS R TO REDO CALIBRATION";
        putText(outImg, waitText.str(), Point(400,leftImage.rows + 13),
            FONT_HERSHEY_SIMPLEX,0.5, TEXT_COLOR,1,CV_AA,false);

        waitText.str("");
        waitText << "PRESS A TO ACCEPT";
        putText(outImg, waitText.str(), Point(750,leftImage.rows + 13),
            FONT_HERSHEY_SIMPLEX,0.5, TEXT_COLOR,1,CV_AA,false);


        //send to video buffer
//      Mat bigOutImg;
//      pyrUp(outImg, bigOutImg, Size(outImg.cols*2, outImg.rows*2));
```

```
        videostreaming.update_MatVideoBuffer(checkoutCalibrationVideoBuffer, outImg);

        videostreaming.update_MatVideoBuffer(goodExampleBuffer, this->
            imgVec_goodEx_metrics[storedFrameCountGood]);
        videostreaming.update_MatVideoBuffer(badExampleBuffer, this->
            imgVec_badEx_metrics[storedFrameCountBad]);
        if (goodExampleBuffer.active)
            storedFrameCountGood = (storedFrameCountGood + 1) % this->
                imgVec_goodEx_metrics.size();
        if (badExampleBuffer.active)
            storedFrameCountBad = (storedFrameCountBad + 1) % this->
                imgVec_badEx_metrics.size();

//      cout << "Stored count good/bad: " << storedFrameCountGood << ", " <<
    storedFrameCountBad << endl;
//      cout << "Good Vector size: " << this->imgVec_goodEx_metrics.size() << endl;
//      cout << "Bad Vector size: " << this->imgVec_badEx_metrics.size() << endl;

        if (saveflag) {
            filename3 << saveImageDir << "/displayImg" << saveimagecount++ << ".png";
            imwrite(filename3.str(), outImg);
//          printf("Saved File: %s\n", filename3.str().c_str());

        }

        pthread_mutex_lock(&keymutex);
        if ((charKey == 0x4E || charKey == 0x6E) && checkoutCalibrationVideoBuffer.
            active) { // N
            //next screen
            maneuverNumber = 5;
            storedFrameCountGood = 0;
            storedFrameCountBad = 0;
        } else if ((charKey == 0x50 || charKey == 0x70)&&
            checkoutCalibrationVideoBuffer.active) { // P
            //previous screen
            maneuverNumber = 4;     //can't go further back
        } else if ((charKey == 0x41 || charKey == 0x61)&&
            checkoutCalibrationVideoBuffer.active) { // A
            //previous screen
            maneuverNumber = 8;
        } else if ((charKey == 0x52 || charKey == 0x72)&&
            checkoutCalibrationVideoBuffer.active) { // R
            numImageSet++;
            numGoodImages = 0;
            newRectifier = Rectifier();
            maneuverNumber = 1;
            storedFrameCountGood = 0;
            storedFrameCountBad = 0;
        } else if (charKey == 0x1B) { // ESC
            printf("Quitting...\n");
            shutdownCriterion = true;
        }
        charKey = 0x00;
        pthread_mutex_unlock(&keymutex);
    }

    void displaySADmap(Mat& leftImage, Mat& rightImage) {
        Mat disparityImage, outImg;
        StereoSAD stereosad;
        stringstream waitText;
        stringstream filename3;

        Mat Qin, Rin, Tin, R1in, P1in, P2in, R2in, M1in, D1in, M2in, D2in;
        double Txin, Tyin, Tzin, fin, cxin, cyin;

        newRectifier.getCameraParameters(Qin, Rin, Tin, R1in, P1in, R2in, P2in, M1in,
            D1in, M2in, D2in, Txin, Tyin, Tzin,
                fin, cxin, cyin);

        newRectifier.rectifyImages(leftImage, rightImage);
        stereosad.computeDisparity(leftImage, rightImage, disparityImage);

        Size size( leftImage.cols + rightImage.cols, MAX(leftImage.rows, rightImage.
            rows) + 18 );

        //place two images side by side
        outImg.create( size, CV_MAKETYPE(leftImage.depth(), 3) );
        outImg = BKGD_COLOR;
        Mat outImgLeft = outImg( Rect(0, 0, leftImage.cols, leftImage.rows) );
        Mat outImgRight = outImg( Rect(leftImage.cols, 0, rightImage.cols, rightImage.
            rows) );

        cvtColor( leftImage, outImgLeft, CV_GRAY2BGR );
        cvtColor( disparityImage, outImgRight, CV_GRAY2BGR );

        waitText.str("");
        waitText << "PRESS P FOR PREVIOUS SCREEN";
        putText(outImg, waitText.str(), Point(10,leftImage.rows + 13),
            FONT_HERSHEY_SIMPLEX,0.5, TEXT_COLOR,1,CV_AA,false);

        waitText.str("");
        waitText << "PRESS R TO REDO CALIBRATION";
        putText(outImg, waitText.str(), Point(400,leftImage.rows + 13),
            FONT_HERSHEY_SIMPLEX,0.5, TEXT_COLOR,1,CV_AA,false);

        waitText.str("");
        waitText << "PRESS A TO ACCEPT";
        putText(outImg, waitText.str(), Point(750,leftImage.rows + 13),
            FONT_HERSHEY_SIMPLEX,0.5, TEXT_COLOR,1,CV_AA,false);

        //send to video buffer
        videostreaming.update_MatVideoBuffer(checkoutCalibrationVideoBuffer, outImg);

        videostreaming.update_MatVideoBuffer(goodExampleBuffer, this->
            imgVec_goodEx_depth[storedFrameCountGood]);
        videostreaming.update_MatVideoBuffer(badExampleBuffer, this->
            imgVec_badEx_depth[storedFrameCountBad]);
        if (goodExampleBuffer.active)
            storedFrameCountGood = (storedFrameCountGood + 1) % this->
                imgVec_goodEx_depth.size();
        if (badExampleBuffer.active)
            storedFrameCountBad = (storedFrameCountBad + 1) % this->imgVec_badEx_depth
                .size();


        if (saveflag) {
            filename3 << saveImageDir << "/displayImg" << saveimagecount++ << ".png";
            imwrite(filename3.str(), outImg);
            //printf("Saved File: %s\n", filename3.str().c_str());

        }

        //printf("Key: %d\n", charKey);
        pthread_mutex_lock(&keymutex);
        if ((charKey == 0x4E || charKey == 0x6E)&& checkoutCalibrationVideoBuffer.
            active) { // N
            //can't go further
            storedFrameCountGood = 0;
            storedFrameCountBad = 0;
        } else if ((charKey == 0x50 || charKey == 0x70)&&
            checkoutCalibrationVideoBuffer.active) { // P
            //previous screen
            maneuverNumber--;
            storedFrameCountGood = 0;
            storedFrameCountBad = 0;
        } else if ((charKey == 0x41 || charKey == 0x61)&&
            checkoutCalibrationVideoBuffer.active) { // A
            //previous screen
            maneuverNumber = 8;
        } else if ((charKey == 0x52 || charKey == 0x72)&&
            checkoutCalibrationVideoBuffer.active) { // R
            numImageSet++;
            numGoodImages = 0;
            newRectifier = Rectifier();
            maneuverNumber = 1;
            storedFrameCountGood = 0;
            storedFrameCountBad = 0;
        }  else if (charKey == 0x1B) { // ESC
            printf("Quitting...\n");
            shutdownCriterion = true;
        }
        charKey = 0x00;
        pthread_mutex_unlock(&keymutex);
    }

    void drawChessboardConnectingLines(Mat& img, vector<Point2f> & leftPoints, vector<
        Point2f> & rightPoints) {
        vector<Point2f>::iterator left_iter, right_iter;
        Scalar linecolor = S_LINE_COLOR;

        if (leftPoints.size() != rightPoints.size()) {
            cout << "Error: Point sizes don't match.";
            return;
        }

        left_iter = leftPoints.begin();
        right_iter = rightPoints.begin();
        do {
            line(img, *left_iter, *right_iter, linecolor,1, CV_AA);
            circle(img, *left_iter, 5,linecolor,1, CV_AA);
            circle(img, *right_iter, 5, linecolor,1, CV_AA);
            left_iter++;
            right_iter++;
        } while (left_iter < leftPoints.end() && right_iter < rightPoints.end());
    }

    void chessboardPoseEstimation(Mat& leftImage, Mat& rightImage) {
        Mat matchesImage, outImg;
        vector<Point2f> pointbuf1, pointbuf2;
        stringstream waitText;
        double roll, pitch, yaw;
        Mat T;
        double mean_squares[2];
        double stddev_squares[2];
        stringstream filename3;

        int leftboundary = 40;
        int yoffset = 40;

        equalizeHist(leftImage, leftImage);
        equalizeHist(rightImage, rightImage);

        Size size( leftImage.cols + rightImage.cols, MAX(leftImage.rows, rightImage.
            rows) + 18 );

        //place two images side by side
        outImg.create( size, CV_MAKETYPE(leftImage.depth(), 3) );
        outImg = BKGD_COLOR;
        Mat outImgLeft = outImg( Rect(0, 0, leftImage.cols, leftImage.rows) );
        Mat outImgRight = outImg( Rect(leftImage.cols, 0, rightImage.cols, rightImage.
            rows) );

        newRectifier.rectifyImages(leftImage, rightImage);

        cvtColor( leftImage, outImgLeft, CV_GRAY2BGR );
//      cvtColor( rightImage, outImgRight, CV_GRAY2BGR );

        //find chessboard
        bool patternfoundLeft = findChessboardCorners(leftImage, Size(CHESS_WIDTH,
            CHESS_HEIGHT), pointbuf1, CV_CALIB_CB_ADAPTIVE_THRESH +
            CV_CALIB_CB_FAST_CHECK /*+ CV_CALIB_CB_NORMALIZE_IMAGE*/ +
            CV_CALIB_CB_FILTER_QUADS);
        bool patternfoundRight = findChessboardCorners(rightImage, Size(CHESS_WIDTH,
            CHESS_HEIGHT), pointbuf2, CV_CALIB_CB_ADAPTIVE_THRESH +
            CV_CALIB_CB_FAST_CHECK /*+ CV_CALIB_CB_NORMALIZE_IMAGE*/ +
            CV_CALIB_CB_FILTER_QUADS);

        line(outImgLeft, Point2d(0,240), Point2d(640,240), GUIDE_COLOR,6);
        line(outImgLeft, Point2d(0,120), Point2d(640,120), GUIDE_COLOR,6);
        line(outImgLeft, Point2d(0,360), Point2d(640,360), GUIDE_COLOR,6);

        waitText.str("");
        waitText << "Stereo Camera Verification";
        putText(outImgRight, waitText.str(), Point(leftboundary,yoffset+30),
            FONT_HERSHEY_SIMPLEX,1.0, TEXT_COLOR,2,CV_AA,false);


        if(patternfoundLeft && patternfoundRight) {
          cornerSubPix(leftImage, pointbuf1, Size(5, 5), Size(-1, -1), TermCriteria(
              CV_TERMCRIT_EPS + CV_TERMCRIT_ITER, 20, 0.1));
          cornerSubPix(rightImage, pointbuf2, Size(5, 5), Size(-1, -1), TermCriteria(
              CV_TERMCRIT_EPS + CV_TERMCRIT_ITER, 20, 0.1));

          drawChessboardConnectingLines(outImgLeft, pointbuf1, pointbuf2);
//        drawChessboardCorners(outImgLeft, Size(CHESS_WIDTH,CHESS_HEIGHT), Mat(
    pointbuf1), patternfoundLeft);
//        drawChessboardCorners(outImgLeft, Size(CHESS_WIDTH,CHESS_HEIGHT), Mat(
    pointbuf2), patternfoundRight);

          guestscientistclass.triangulateChessboard(pointbuf1, pointbuf2, CHESS_WIDTH,
               CHESS_HEIGHT, roll, pitch, yaw, T,
                  mean_squares, stddev_squares);

/*
              std::cout << "Mean: [" << mean_squares[0] << "," << mean_squares[1]
                    << "]\nStandard Deviation: [" << stddev_squares[0]<< "," <<
                        stddev_squares[1] << "]\n";
*/

            waitText.str("");
            waitText << std::fixed << std::setprecision(2) << "Please CHECK that the
                Blue Lines are Horizontal";
            putText(outImgRight, waitText.str(), Point(leftboundary,yoffset+100),
                FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);
            waitText.str("");
            waitText << std::fixed << std::setprecision(2) << "using the Three Green
                Guide Lines";
            putText(outImgRight, waitText.str(), Point(leftboundary,yoffset+130),
                FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);


            waitText.str("");
            waitText << std::fixed << std::setprecision(2) << "Range to Target: "
                    << 100*T.at<double>(2,0) << " cm";
            putText(outImgRight, waitText.str(), Point(leftboundary,yoffset+170),
                FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);

/*          waitText.str("");
            waitText << std::fixed << std::setprecision(2) << "Rotation (Roll, Pitch,
                Yaw in deg): [ "
```

```
                    << roll << ","<< pitch <<","<< yaw << " ]";
            putText(outImgRight, waitText.str(), Point(leftboundary,yoffset+120),
                FONT_HERSHEY_SIMPLEX,0.5, CV_RGB(0,255,0),1,8,false);
*/

            if (inRange(limits.square_size,100*mean_squares[0]) && inRange(limits.
                square_size,100*mean_squares[1])) {
                //square size good
                waitText.str("");
                waitText << std::fixed << std::setprecision(2) << "CALIBRATION TARGET
                    Squares are the CORRECT Size";
                putText(outImgRight, waitText.str(), Point(leftboundary,yoffset+210),
                    FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);
                circle(outImgRight, Point(leftboundary-15,yoffset+20), 7,GOOD_COLOR
                    ,-1, CV_AA);

            } else {
                //square size bad
                circle(outImgRight, Point(leftboundary-15,yoffset+20), 7,BAD_COLOR,-1,
                     CV_AA);

                waitText.str("");
                waitText << std::fixed << std::setprecision(2) << "CALIBRATION TARGET
                    Squares are the INCORRECT Size";
                putText(outImgRight, waitText.str(), Point(leftboundary,yoffset+210),
                    FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);

                waitText.str("");
                waitText << std::fixed << std::setprecision(2) << "Mean Square Size in
                     cm: [ "
                        << 100*mean_squares[0] << ","<< 100*mean_squares[1] << " ]";
                putText(outImgRight, waitText.str(), Point(leftboundary,yoffset+230),
                    FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);

                waitText.str("");
                waitText << std::fixed << std::setprecision(2) << "Standard Deviation
                    Size in cm: [ "
                        << 100*stddev_squares[0] << ","<< 100*stddev_squares[1] << " ]
                            ";
                putText(outImgRight, waitText.str(), Point(leftboundary,yoffset+250),
                    FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);
            }


        } else {
            circle(outImgRight, Point(leftboundary-15,yoffset+20), 7,CV_RGB(0x7F,0x7F
                ,0x7F),-1, CV_AA);
            waitText.str("");
            waitText << "CALIBRATION TARGET is Not Visible in both cameras";
            putText(outImgRight, waitText.str(), Point(leftboundary,yoffset+100),
                FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);
            waitText.str("");
            waitText << "Please hold CALIBRATION TARGET approximately";
            putText(outImgRight, waitText.str(), Point(leftboundary,yoffset+160),
                FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);
            waitText.str("");
            waitText << "30 cm in front of both cameras";
            putText(outImgRight, waitText.str(), Point(leftboundary,yoffset+190),
                FONT_HERSHEY_SIMPLEX,0.6, TEXT_COLOR,1,CV_AA,false);

        }

        waitText.str("");
        waitText << "PRESS N FOR NEXT SCREEN";
        putText(outImg, waitText.str(), Point(1020,leftImage.rows + 13),
            FONT_HERSHEY_SIMPLEX,0.5, TEXT_COLOR,1,CV_AA,false);

        waitText.str("");
        waitText << "PRESS P FOR PREVIOUS SCREEN";
        putText(outImg, waitText.str(), Point(10,leftImage.rows + 13),
            FONT_HERSHEY_SIMPLEX,0.5, TEXT_COLOR,1,CV_AA,false);

        waitText.str("");
        waitText << "PRESS R TO REDO CALIBRATION";
        putText(outImg, waitText.str(), Point(400,leftImage.rows + 13),
            FONT_HERSHEY_SIMPLEX,0.5, TEXT_COLOR,1,CV_AA,false);

        waitText.str("");
        waitText << "PRESS A TO ACCEPT";
        putText(outImg, waitText.str(), Point(750,leftImage.rows + 13),
            FONT_HERSHEY_SIMPLEX,0.5, TEXT_COLOR,1,CV_AA,false);


        //send to video buffer
        videostreaming.update_MatVideoBuffer(checkoutCalibrationVideoBuffer, outImg);

        videostreaming.update_MatVideoBuffer(goodExampleBuffer, this->
            imgVec_goodEx_poseEst[storedFrameCountGood]);
        videostreaming.update_MatVideoBuffer(badExampleBuffer, this->
            imgVec_badEx_poseEst[storedFrameCountBad]);
        if (goodExampleBuffer.active)
            storedFrameCountGood = (storedFrameCountGood + 1) % this->
                imgVec_goodEx_poseEst.size();
        if (badExampleBuffer.active)
            storedFrameCountBad = (storedFrameCountBad + 1) % this->
                imgVec_badEx_poseEst.size();


        if (saveflag) {
            filename3 << saveImageDir << "/displayImg" << saveimagecount++ << ".png";
            imwrite(filename3.str(), outImg);
            //printf("Saved File: %s\n", filename3.str().c_str());

        }

        //printf("Key: %d\n", charKey);
        pthread_mutex_lock(&keymutex);
        if ((charKey == 0x4E || charKey == 0x6E)&& checkoutCalibrationVideoBuffer.
            active) { // N
            //next screen
            maneuverNumber++;
            storedFrameCountGood = 0;
            storedFrameCountBad = 0;
        } else if ((charKey == 0x50 || charKey == 0x70)&&
            checkoutCalibrationVideoBuffer.active) { // P
            //previous screen
            maneuverNumber--;
            storedFrameCountGood = 0;
            storedFrameCountBad = 0;
        } else if ((charKey == 0x52 || charKey == 0x72)&&
            checkoutCalibrationVideoBuffer.active) { // R
            //redo
            numImageSet++;
            numGoodImages = 0;
            newRectifier = Rectifier();
            maneuverNumber = 1;
            storedFrameCountGood = 0;
            storedFrameCountBad = 0;
        } else if ((charKey == 0x41 || charKey == 0x61)&&
            checkoutCalibrationVideoBuffer.active) { // A
            //previous screen
            maneuverNumber = 8;
        }  else if (charKey == 0x1B) { // ESC
            printf("Quitting...\n");
            shutdownCriterion = true;
        }
        charKey = 0x00;
        pthread_mutex_unlock(&keymutex);


    }

    void displayAcceptQuestion(Mat& leftImage, Mat& rightImage) {
        Mat matchesImage, outImg;
        vector<Point2f> pointbuf1, pointbuf2;
        stringstream waitText;
        double roll, pitch, yaw;
        Mat T;
        double mean_squares[2];
        double stddev_squares[2];
        stringstream filename3;

        int leftboundary = 20;
        int yoffset = 150;

        Size size( leftImage.cols + rightImage.cols, MAX(leftImage.rows, rightImage.
            rows) + 18 );

        //place two images side by side
        outImg.create( size, CV_MAKETYPE(leftImage.depth(), 3) );
        outImg = BKGD_COLOR;
        Mat outImgLeft = outImg( Rect(0, 0, leftImage.cols, leftImage.rows) );
        Mat outImgRight = outImg( Rect(leftImage.cols, 0, rightImage.cols, rightImage.
            rows) );

        waitText.str("");
        waitText << "Would you like to accept the new calibration and exit?";
        putText(outImg, waitText.str(), Point(70,150), FONT_HERSHEY_SIMPLEX,1.0,
            TEXT_COLOR,2,CV_AA,false);

        waitText.str("");
        waitText << "Press Y for Yes";
        putText(outImg, waitText.str(), Point(70,190), FONT_HERSHEY_SIMPLEX,0.7,
            TEXT_COLOR,1,CV_AA,false);

        waitText.str("");
        waitText << "Press N for No";
        putText(outImg, waitText.str(), Point(70,220), FONT_HERSHEY_SIMPLEX,0.7,
            TEXT_COLOR,1,CV_AA,false);


        //send to video buffer
        videostreaming.update_MatVideoBuffer(checkoutCalibrationVideoBuffer, outImg);

        if (saveflag) {
            filename3 << saveImageDir << "/displayImg" << saveimagecount++ << ".png";
            imwrite(filename3.str(), outImg);
            //printf("Saved File: %s\n", filename3.str().c_str());

        }

        pthread_mutex_lock(&keymutex);
        if (charKey == 0x4E || charKey == 0x6E) { // N
            //previous screen
            maneuverNumber = 4;      //can't go further back
        } else if (charKey == 0x59 || charKey == 0x79) { // Y
            char newSysCommand[200];
            char newCalibDir[200];

            sprintf(newCalibDir,"/opt/GogglesOptics/Calib_Params/calib_%s", this->
                datastorage.getTimeOfTestStart().c_str());
            sprintf(newSysCommand, "sudo mkdir %s", newCalibDir);
            system(newSysCommand);

            sprintf(newSysCommand, "sudo chmod 777 %s", newCalibDir);
            system(newSysCommand);

            sprintf(newSysCommand, "sudo ln -sfn %s /opt/GogglesOptics/Calib_Params/
                CURRENT_CALIB_DIR", newCalibDir, this->datastorage.getGPFpath().c_str
                ());
            printf("System Command: %s\n", newSysCommand);
            system(newSysCommand);

            sprintf(newSysCommand, "sudo cp %s/*.yml %s/", currentCalibImageDir.c_str
                (),newCalibDir);
            printf("System Command: %s\n", newSysCommand);
            system(newSysCommand);

            sprintf(newSysCommand, "sudo chmod 777 %s/*", newCalibDir);
            printf("System Command: %s\n", newSysCommand);
            system(newSysCommand);

            printf("Accepting Calibration: %s\n", newCalibDir);
            shutdownCriterion = true;
        } else if (charKey == 0x1B) { // ESC
```

```
            printf("Quitting...\n");
            shutdownCriterion = true;
        }
        charKey = 0x00;
        pthread_mutex_unlock(&keymutex);

    }

    void GSprocessImages(Mat& leftImage, Mat& rightImage) {

        //// Check if currently captured frames are in synch
        //// If they are out of synch, the previous image frames are provided, which
            are still in synch
        if (this->synchCheckFlag == -1)
        {
            printf("Current frames are out of synch! Previous frames are used.\n");
        }

        switch (maneuverNumber) {
        case 1:     //setup calibration capture
            setupCalibrationCapture();
            maneuverNumber = 2;
            break;
        case 2:
            captureCalibrationImages(leftImage, rightImage);
            break;
        case 3:
            computeCalibration(leftImage, rightImage);
            break;
        case 4:
            displayCalibrationMetrics(leftImage, rightImage);
            break;
        case 5:
            chessboardPoseEstimation(leftImage,rightImage);
            break;
        case 6:
            displaySADmap(leftImage, rightImage);
            break;
        case 7:
            this->shutdownCriterion = true;
            break;
        case 8:
            if (performedCalibration) {
                displayAcceptQuestion(leftImage, rightImage);
            } else {
                this->shutdownCriterion = true;
            }
            break;
        }

    }

    void GSprocessGuiKeyPress(unsigned char networkkey) {
        pthread_mutex_lock(&keymutex);
        charKey = networkkey;
/*
        if (charKey == 0x53 || charKey == 0x73) //s or S
        {
            saveflag = !saveflag;

            if (saveflag) {
                char newdirname[200];
                sprintf(newdirname, "SavedImages%d", savedircount++);
                saveimagecount = 0;
                saveImageDir = this->datastorage.newGSDataDirectory(newdirname);
                cout << "Save Flag Turned ON: " << saveImageDir << endl;
            } else {
                cout << "Save Flag Turned OFF: " << endl;
            }

        }
*/
        pthread_mutex_unlock(&keymutex);
    }


    void GSparseParameterFile(string line) {
          string searchString;
          string foundString;
          size_t found;

          searchString = "MINIMUM_NUMER_IMAGES";
          found = line.find(searchString);
          if (found != string::npos)
          {
              foundString = line.substr( found+searchString.size()+1, string::npos );
              minimumNumberImages =  atoi(foundString.c_str());
              cout << "MINIMUM_NUMER_IMAGES " << foundString << endl;
          }
          searchString.clear();
```

```
            found = string::npos;

            searchString = "MAX_CALIBRATION_ITERATIONS";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                maxCalibrationIterations =  atoi(foundString.c_str());
                cout << "MAX_CALIBRATION_ITERATIONS " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "MAX_CALIBRATION_CHANGE";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                maxCalibrationChange =  atof(foundString.c_str());
                cout << "MAX_CALIBRATION_CHANGE " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "HORZ_TRANSLATION_MAX";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                limits.horz_translation[MAXVAL] =  atof(foundString.c_str());
                cout << "HORZ_TRANSLATION_MAX " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "HORZ_TRANSLATION_MIN";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                limits.horz_translation[MINVAL] =  atof(foundString.c_str());
                cout << "HORZ_TRANSLATION_MIN " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "VERT_TRANSLATION_MAX";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                limits.vert_translation[MAXVAL] =  atof(foundString.c_str());
                cout << "VERT_TRANSLATION_MAX " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "VERT_TRANSLATION_MIN";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                limits.vert_translation[MINVAL] =  atof(foundString.c_str());
                cout << "HORZ_TRANSLATION_MIN " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "DEPTH_TRANSLATION_MAX";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                limits.depth_translation[MAXVAL] =  atof(foundString.c_str());
                cout << "DEPTH_TRANSLATION_MAX " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "DEPTH_TRANSLATION_MIN";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                limits.depth_translation[MINVAL] =  atof(foundString.c_str());
                cout << "HORZ_TRANSLATION_MIN " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "ROT_ROLL_MAX";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                limits.rot_roll[MAXVAL] =  atof(foundString.c_str());
                cout << "ROT_ROLL_MAX " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "ROT_ROLL_MIN";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                limits.rot_roll[MINVAL] =  atof(foundString.c_str());
                cout << "ROT_ROLL_MIN " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "ROT_PITCH_MAX";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                limits.rot_pitch[MAXVAL] =  atof(foundString.c_str());
                cout << "ROT_PITCH_MAX " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "ROT_PITCH_MIN";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                limits.rot_pitch[MINVAL] =  atof(foundString.c_str());
                cout << "ROT_PITCH_MIN " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "ROT_YAW_MAX";
```

```
        found = line.find(searchString);
        if (found != string::npos)
        {
            foundString = line.substr( found+searchString.size()+1, string::npos );
            limits.rot_yaw[MAXVAL] =  atof(foundString.c_str());
            cout << "ROT_YAW_MAX " << foundString << endl;
        }
        searchString.clear();
        found = string::npos;

        searchString = "ROT_YAW_MIN";
        found = line.find(searchString);
        if (found != string::npos)
        {
            foundString = line.substr( found+searchString.size()+1, string::npos );
            limits.rot_yaw[MINVAL] =  atof(foundString.c_str());
            cout << "ROT_YAW_MIN " << foundString << endl;
        }
        searchString.clear();
        found = string::npos;

        searchString = "FOCAL_LEN_MAX";
        found = line.find(searchString);
        if (found != string::npos)
        {
            foundString = line.substr( found+searchString.size()+1, string::npos );
            limits.focal_len[MAXVAL] =  atof(foundString.c_str());
            cout << "FOCAL_LEN_MAX " << foundString << endl;
        }
        searchString.clear();
        found = string::npos;

        searchString = "FOCAL_LEN_MIN";
        found = line.find(searchString);
        if (found != string::npos)
        {
            foundString = line.substr( found+searchString.size()+1, string::npos );
            limits.focal_len[MINVAL] =  atof(foundString.c_str());
            cout << "FOCAL_LEN_MIN " << foundString << endl;
        }
        searchString.clear();
        found = string::npos;

        searchString = "SQUARE_SIZE_MAX";
        found = line.find(searchString);
```

```
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                limits.square_size[MAXVAL] =  atof(foundString.c_str());
                cout << "SQUARE_SIZE_MAX " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            searchString = "SQUARE_SIZE_MIN";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                limits.square_size[MINVAL] =  atof(foundString.c_str());
                cout << "SQUARE_SIZE_MIN " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

            // use images from file
            searchString = "USE_IMAGES_FROM_FILE";
            found = line.find(searchString);
            if (found != string::npos)
            {
              foundString = line.substr( found+searchString.size()+1, string::npos );
              if (foundString != "false") {
                  this->currentCalibImageDir = foundString;
                  this->ImgListFilename << currentCalibImageDir << "/imglist.yaml";
                  this->useImagesFromFile = true;
                  this->maneuverNumber = 3;
                  cout << "USE_IMAGES_FROM_FILE " << foundString << endl;
              }
            }
            searchString.clear();
            found = string::npos;

            // use images from file
            searchString = "STORED_IMAGES_GOOD_BAD_EX";
            found = line.find(searchString);
            if (found != string::npos)
            {
              foundString = line.substr( found+searchString.size()+1, string::npos );
              if (foundString != "false") {
                  cout << "STORED_IMAGES_GOOD_BAD_EX " << foundString << endl;
                  this->dir_storedImages_goodEx_capCalib = foundString + "/goodEx_CapCalib";
                  this->dir_storedImages_badEx_capCalib = foundString + "/badEx_CapCalib";
                  this->dir_storedImages_goodEx_metrics = foundString + "/goodEx_Metrics";
                  this->dir_storedImages_badEx_metrics = foundString + "/badEx_Metrics";
                  this->dir_storedImages_goodEx_poseEst = foundString + "/goodEx_PoseEst";
                  this->dir_storedImages_badEx_poseEst = foundString + "/badEx_PoseEst";
                  this->dir_storedImages_goodEx_depth = foundString + "/goodEx_Depth";
                  this->dir_storedImages_badEx_depth = foundString + "/badEx_Depth";

              }
            }
            searchString.clear();
            found = string::npos;

            // autoImageStorage
            searchString = "SAVE_DISPLAY_IMAGES";
            found = line.find(searchString);
            if (found != string::npos)
            {
                foundString = line.substr( found+searchString.size()+1, string::npos );
                if (foundString == "true")
                    saveflag = true;
                if (foundString == "false")
                    saveflag = false;
                cout << "SAVE_DISPLAY_IMAGES " << foundString << endl;
            }
            searchString.clear();
            found = string::npos;

      }

  };

  #endif
```

Listing B.10: Camera Calibration: additionalGuestScientistCode.h

```
#ifndef MYHEADERFILE_H_
#define MYHEADERFILE_H_

```

```
//#include "cv.h"
//#include "highgui.h"
#include <opencv2/opencv.hpp>

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string>
#include <iostream>
#include <vector>
#include <fstream>
#include <sys/time.h>
#include <sys/resource.h>
#include <photogrammetry.h>

using namespace std;
using namespace cv;

#include <stereoSAD.h>
#include "stereoFeatureMatcher.h"


class GuestScientistClass{

    //// feature vectors and feature extractors
    vector<KeyPoint> leftFeatures, rightFeatures;
    int FASTthresh;
    int FASTthreshDummy;

    //// feature descriptors and descriptor extractor
    SurfDescriptorExtractor surfDescriptorExtractor;
    SiftDescriptorExtractor siftDescriptorExtractor;
    ORB orbDetector;
    SURF surfDetector;
    SIFT siftDetector;
    Mat leftDescriptors, rightDescriptors;

    //// feature stereo matching
    vector<DMatch> stereoFeatureMatches, stereoFeatureMatchesRANSAC;
    vector<KeyPoint> leftFeaturesPreMatch, rightFeaturesPreMatch;
    vector<KeyPoint> leftFeaturesRANSAC, rightFeaturesRANSAC;
    vector<KeyPoint> leftFeaturesRANSAC_filtered;
    vector<KeyPoint> rightFeaturesRANSAC_filtered;
    Mat leftDescriptorsRANSAC, rightDescriptorsRANSAC;
    int maxiterRANSAC;
    int numberOfSufficientMatches;
    vector<Point3d> stereoMatchedFeaturesRANSACcoord3D;
    vector<Point3d> stereoMatchedFeaturesRANSACcoord3D_transformed;

    //photogrammetry
    Photogrammetry* photo;

    Mat Q;
    Mat R, T, R1, P1, R2, P2;
    Mat M1, D1, M2, D2;
    double Tx;
    double f;
    double cx;
    double cy;

    int iterationNumber;

    vector<double> dval;
    bool dval_init;

public:

    StereoMatcher stereomatcher;
    StereoSAD stereoSAD;

    int imagewidth, imageheight;

    //// for visualization
    Mat featureMatchesImage;
    Mat featureMatchesImagePreRansac;

    // GSdata storage
    string GSstoragePath;
    ofstream positionVOEstimateFile;

    GuestScientistClass() {

        /// stereo RANSAC settings
        stereomatcher.SDthreshold = 0.20; // "abstract" value, since error = x_right'
            * F * x_left
        stereomatcher.maxiterRANSAC = 600;

        /// FAST settings
        FASTthresh = 40;

```

```
            /// other settings
        iterationNumber = 1;

        dval_init = false;

    }

    void extractFeatures(Mat& leftImage, Mat& rightImage);

    void extractFeatureDescriptors(Mat& leftImage, Mat& rightImage);

    void stereoMatchFeatures(Mat& leftImage, Mat& rightImage, int& numberMatches);

    void showStereoMatches(Mat& leftImage, Mat& rightImage, Mat& matchesImage);

    void updatePhotogrammetry(Rectifier & rectifier) {
        photo = new Photogrammetry(rectifier);
    }

    int triangulateChessboard(vector<Point2f>& leftImgPoints, vector<Point2f>&
        rightImgPoints,
            int chessboardWidth, int chessboardHeight, double & roll, double & pitch,
                double & yaw, Mat & Tout,
            double mean_squares[], double stddev_squares[]);

    int monocularChessboard(vector<Point2f>& imgPoints,
            int chessboardWidth, int chessboardHeight, bool usingLeft, double & roll,
                double & pitch, double & yaw, Mat & Tout, double & mse);

};

#endif
```

Listing B.11: Camera Calibration: additionalGuestScientistCode.cpp

```
#include "additionalGuestscientistCode.h"

using namespace std;
using namespace cv;


void GuestScientistClass::extractFeatures(Mat& leftImage, Mat& rightImage) {

//  FAST(leftImage, this->leftFeatures, this->FASTthresh, true);
//  FAST(rightImage, this->rightFeatures, this->FASTthresh, true);


    SURF surfDetector(1500, 4, 2, false, true);
    Mat zeroMask;
    surfDetector(leftImage, zeroMask, this->leftFeatures);
    surfDetector(rightImage, zeroMask, this->rightFeatures);

/*
    Mat zeroMask;
    siftDetector(leftImage, zeroMask, this->leftFeatures);
    siftDetector(rightImage, zeroMask, this->rightFeatures);
*/

}

void GuestScientistClass::extractFeatureDescriptors(Mat& leftImage,
        Mat& rightImage) {

    SurfDescriptorExtractor surfDescriptorExtractor(4, 2, false);
    surfDescriptorExtractor.compute(leftImage, this->leftFeatures, this->
        leftDescriptors);
    surfDescriptorExtractor.compute(rightImage, this->rightFeatures, this->
        rightDescriptors);

/*
    siftDescriptorExtractor.compute(leftImage, this->leftFeatures, this->
        leftDescriptors);
    siftDescriptorExtractor.compute(rightImage, this->rightFeatures, this->
        rightDescriptors);
*/

}

void GuestScientistClass::stereoMatchFeatures(Mat& leftImage, Mat& rightImage, int&
    numberMatches) {

    this->stereomatcher.initialStereoMatch(this->leftFeatures, this->rightFeatures,
        this->leftDescriptors, this->rightDescriptors, this->stereoFeatureMatches,
        this->leftFeaturesPreMatch, this->rightFeaturesPreMatch);

    // this is the Fundamental Matrix RANSAC version
    if (this->stereoFeatureMatches.size() > 8) // RANSAC needs at least 8 matches, and
         we need more than 8 for RANSAC to make sense
    {
        this->stereomatcher.getStereoInliersRANSAC_Fundamental(this->leftFeatures,
            this->rightFeatures, this->leftFeaturesRANSAC,
                this->rightFeaturesRANSAC, this->leftDescriptors, this->
                    rightDescriptors, this->stereoFeatureMatches,
                this->stereoFeatureMatchesRANSAC);
    }

    numberMatches = this->stereoFeatureMatchesRANSAC.size();

//  printf("STEREO: numOf initial: %d \t numOf RANSAC: %d\n", this->
    leftFeaturesPreMatch.size(), this->stereoFeatureMatchesRANSAC.size());

/*
    // this is the Homography Matrix RANSAC version
    if (this->stereoFeatureMatches.size() > 5) // RANSAC needs at least 8 matches, and
         we need more than 8 for RANSAC to make sense
    {
        this->stereomatcher.getStereoInliersRANSAC_Homography(this->leftFeatures, this
            ->rightFeatures, this->leftFeaturesRANSAC,
                this->rightFeaturesRANSAC, this->leftDescriptors, this->
                    rightDescriptors, this->stereoFeatureMatches,
                this->stereoFeatureMatchesRANSAC);
    }
*/

/*
    // this is the Mutliple Homography Matrix RANSAC version
    if (this->stereoFeatureMatches.size() > 5) // RANSAC needs at least 8 matches, and
         we need more than 8 for RANSAC to make sense
    {
        this->stereomatcher.getStereoInliersRANSAC_MultiHomography(this->leftFeatures,
             this->rightFeatures, this->leftFeaturesRANSAC,
                this->rightFeaturesRANSAC, this->leftDescriptors, this->
                    rightDescriptors, this->stereoFeatureMatches,
                this->stereoFeatureMatchesRANSAC);
    }
*/

}

void GuestScientistClass::showStereoMatches(Mat& leftImage, Mat& rightImage, Mat&
    matchesImage) {

    drawMatches(leftImage, this->leftFeatures, rightImage, this->rightFeatures, this->
        stereoFeatureMatchesRANSAC, matchesImage, 255, Scalar::all(-1));
```

```
//  drawMatches(leftImage, this->leftFeatures, rightImage, this->rightFeatures, this->
    stereoFeatureMatches, matchesImage, 255, Scalar::all(-1)); // this is just to see,
     what initialStereoMatching does
}

int GuestScientistClass::triangulateChessboard(vector<Point2f>& leftImgPoints, vector<
    Point2f>& rightImgPoints,
        int chessboardWidth, int chessboardHeight, double & roll, double & pitch,
            double & yaw, Mat & Tout,
        double mean_squares[], double stddev_squares[]) {
    int index;
    vector<Point3d> measuredPoints;
    vector<Point3d> objectPoints;
    Point3d xyzPoint;
    Point3d p1, p2;
    double squareSize = 0.0254;
    Mat R, T;
    double scale;

    R.create(3, 3, CV_64FC1);
    T.create(3, 1, CV_64FC1);

    if (leftImgPoints.size() == chessboardWidth*chessboardHeight && rightImgPoints.
        size() == chessboardWidth*chessboardHeight) {
        for (int y = 0; y < chessboardHeight; y++) {
            for (int x = 0; x < chessboardWidth; x++) {
                index = x+y*chessboardWidth;

                //std::cout << "left: " << leftImgPoints[index] << " right: " <<
                    rightImgPoints[index];
                this->photo->triangulate(leftImgPoints[index], rightImgPoints[index],
                    xyzPoint);
                //std::cout << "(" << x << "," << y << "): " << xyzPoint << endl;

                //cout << "XYZ Point: " << xyzPoint << endl;

                measuredPoints.push_back(xyzPoint);
                objectPoints.push_back(Point3d((x - (int) floor(chessboardWidth/2))*
                    squareSize, (y - (int) floor(chessboardHeight/2))*squareSize, 0));
            }
        }

//      cout << endl;

        this->photo->absoluteOrientation(objectPoints, measuredPoints, R, T, scale);
```

```
//      std::cout << "R: " << endl << R << endl << "T: " << endl << T << endl << "
    Scale: " << scale << endl;

        yaw = atan(R.at<double>(0,1)/R.at<double>(1,1))*180 / M_PI;
        pitch = -asin(R.at<double>(2,1))*180 / M_PI;
        roll = atan(R.at<double>(2,0)/R.at<double>(2,2))*180 / M_PI;

        Tout = T;

        //compute statistics
        //X DIRECTION
        Mat p1obj, p2obj, d3;
        vector<double> dvect;
        double d;
        double mean_d = 0;
        double stddev_d = 0;
        int n = 0;
        for (int y = 0; y < chessboardHeight-1; y++) {
            for (int x = 0; x < chessboardWidth-1; x++) {
                p1 = measuredPoints[x+y*chessboardWidth];
                p2 = measuredPoints[(x+1)+y*chessboardWidth];

                //gemm(R, Mat(p1), 1/scale, NULL, 0, p1obj, GEMM_1_T);
                p1obj = (1/scale) * R * Mat(p1);
                //gemm(R, Mat(p2), 1/scale, NULL, 0, p2obj, GEMM_1_T);
                p2obj = (1/scale) * R * Mat(p2);

                subtract(p1obj, p2obj, d3);
                d = norm(d3);
                dvect.push_back(d);
                mean_d += d;

                n++;
            }
        }
        mean_d /= n;

        vector<double>::iterator it;
        for (it = dvect.begin(); it < dvect.end(); it++) {
            d = *it;
            stddev_d += (d - mean_d)*(d - mean_d);
        }
        stddev_d = sqrt(stddev_d / (n-1));

        stddev_squares[0] = stddev_d;
        mean_squares[0] = mean_d;

        //Y DIRECTION
        dvect.clear();
        mean_d = 0;
        stddev_d = 0;
        n = 0;
        for (int y = 0; y < chessboardHeight-1; y++) {
            for (int x = 0; x < chessboardWidth-1; x++) {
                p1 = measuredPoints[x+y*chessboardWidth];
                p2 = measuredPoints[x+(y+1)*chessboardWidth];

                //gemm(R, Mat(p1), 1/scale, NULL, 0, p1obj, GEMM_1_T);
                p1obj = (1/scale) * R * Mat(p1);
                //gemm(R, Mat(p2), 1/scale, NULL, 0, p2obj, GEMM_1_T);
                p2obj = (1/scale) * R * Mat(p2);

                subtract(p1obj, p2obj, d3);
                d = norm(d3);
                dvect.push_back(d);
                mean_d += d;

                n++;
            }
        }
        mean_d /= n;

        for (it = dvect.begin(); it < dvect.end(); it++) {
            d = *it;
            stddev_d += (d - mean_d)*(d - mean_d);
        }
        stddev_d = sqrt(stddev_d / (n-1));

        stddev_squares[1] = stddev_d;
        mean_squares[1] = mean_d;


    } else {
        roll = 0;
        pitch = 0;
        yaw = 0;

        Tout = T;

        return -1;
    }

    return 0;
}

int GuestScientistClass::monocularChessboard(vector<Point2f>& imgPoints,
        int chessboardWidth, int chessboardHeight, bool usingLeft, double & roll,
            double & pitch, double & yaw, Mat & Tout, double & mse) {
    int index;
    vector<Point3d> objectPoints;
    double squareSize = 0.0254;
    Mat R, T;

    R.create(3, 3, CV_64FC1);
    T.create(3, 1, CV_64FC1);

    if (imgPoints.size() == chessboardWidth*chessboardHeight) {
        for (int y = 0; y < chessboardHeight; y++) {
            for (int x = 0; x < chessboardWidth; x++) {
                index = x+y*chessboardWidth;

                objectPoints.push_back(Point3d((x - (int) floor(chessboardWidth/2))*
                    squareSize, (y - (int) floor(chessboardHeight/2))*squareSize, 0));

                if (dval_init == false) {
                    dval.push_back(0.5);
                }
            }
        }

        dval_init = true;

        this->photo->exteriorOrientation(imgPoints, objectPoints, R, T, dval, mse,
            usingLeft, 1000, 1.0e-10);

        yaw = atan(R.at<double>(0,1)/R.at<double>(1,1))*180 / M_PI;
        pitch = -asin(R.at<double>(2,1))*180 / M_PI;
        roll = atan(R.at<double>(2,0)/R.at<double>(2,2))*180 / M_PI;

        Tout = T;
    } else {
        roll = 0;
        pitch = 0;
        yaw = 0;

        Tout = T;

        return -1;
    }

    return 0;
}
```

Listing B.12: Camera Calibration: stereoCalib.h

```
#ifndef STEREOCALIB_H
#define STEREOCALIB_H

#include <optics.h>

#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <vector>
#include <string>
#include <algorithm>
#include <iostream>
#include <iterator>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

using namespace cv;
using namespace std;

void StereoCalib(const vector<string>& imagelist, Size boardSize,
        Rectifier & rectifier, string currentCalibImageDir,
        vector<vector<Point2f> > imagePoints[2], vector<vector<Point3f> > &
            objectPoints, Size & imageSize, vector<string>& goodImageList,
        int maxIterations, double maxChange,
        float * rms_error, float * mean_reprojection_error);

bool initStereoCalib(const string& filename, vector<string>& l, string
    currentCalibImageDir, Size boardSize,
        vector<vector<Point2f> > imagePoints[2], vector<vector<Point3f> > &
            objectPoints, Size & imageSize, vector<string>& goodImageList,
        int* nimages);

```

```
#endif
```

Listing B.13: Camera Calibration: stereoCalib.cpp

```
#include "stereoCalib.h"

//RATION MODEL PARAMETER IS ON LINE 173


void StereoCalib(const vector<string>& imagelist, Size boardSize,
        Rectifier & rectifier, string currentCalibImageDir,
        vector<vector<Point2f> > imagePoints[2], vector<vector<Point3f> > &
            objectPoints, Size & imageSize, vector<string>& goodImageList,
        int maxIterations, double maxChange,
        float * rms_error, float * mean_reprojection_error) {

    int i, j, k, nimages = (int) goodImageList.size() / 2;
    bool displayCorners = false;//true;
    const int maxScale = 2;
    const float squareSize = 1.0;

    cout << "Imagelist size: " << imagelist.size() << endl;
    cout << "nimages: " << nimages << endl;
    cout << "goodImagelist size: " << goodImageList.size() << endl << endl;

    cout << "objpoints size: " << objectPoints.size() << endl;
    cout << "imagePoints0 size: " << imagePoints[0].size() << endl;
    cout << "imagePoints1 size: " << imagePoints[1].size() << endl;

    for (i = 0; i < nimages; i++) {
        for (j = 0; j < boardSize.height; j++)
            for (k = 0; k < boardSize.width; k++)
                //cout << "ijk: " << i << "," << j << "," << k << endl;
                objectPoints[i].push_back(Point3f(j * squareSize, k
                        * squareSize, 0));
    }

    cout << "Running stereo calibration ...\n";

    Mat cameraMatrix[2], distCoeffs[2];
    Mat R, T, E, F;

    rectifier.getCameraParameters(R,T,cameraMatrix[0], distCoeffs[0], cameraMatrix[1],
         distCoeffs[1]);

    double rms = stereoCalibrate(objectPoints, imagePoints[0], imagePoints[1],
            cameraMatrix[0], distCoeffs[0], cameraMatrix[1], distCoeffs[1],
            imageSize, R, T, E, F, TermCriteria(CV_TERMCRIT_ITER
                    + CV_TERMCRIT_EPS, 300, 1e-10),
            //                     0
                    CV_CALIB_USE_INTRINSIC_GUESS +
            //      CV_CALIB_FIX_INTRINSIC +
            //CV_CALIB_FIX_ASPECT_RATIO +
            //                     CV_CALIB_ZERO_TANGENT_DIST +
            //      CV_CALIB_SAME_FOCAL_LENGTH
            + CV_CALIB_RATIONAL_MODEL
            //                     CV_CALIB_FIX_K3 + CV_CALIB_FIX_K4 + CV_CALIB_FIX_K5
            );

    *rms_error = rms;
    cout << "done with RMS error=" << rms << endl;

    // CALIBRATION QUALITY CHECK
    // because the output fundamental matrix implicitly
    // includes all the output information,
    // we can check the quality of calibration using the
    // epipolar geometry constraint: m2^t*F*m1=0
    double err = 0;
    int npoints = 0;
    vector<Vec3f> lines[2];
    for (i = 0; i < nimages; i++) {
        int npt = (int) imagePoints[0][i].size();
        Mat imgpt[2];
        for (k = 0; k < 2; k++) {
            imgpt[k] = Mat(imagePoints[k][i]);
            undistortPoints(imgpt[k], imgpt[k], cameraMatrix[k], distCoeffs[k],
                    Mat(), cameraMatrix[k]);
            computeCorrespondEpilines(imgpt[k], k + 1, F, lines[k]);
        }
        for (j = 0; j < npt; j++) {
            double errij = fabs(imagePoints[0][i][j].x * lines[1][j][0]
                    + imagePoints[0][i][j].y * lines[1][j][1] + lines[1][j][2])
                    + fabs(imagePoints[1][i][j].x * lines[0][j][0]
                            + imagePoints[1][i][j].y * lines[0][j][1]
                            + lines[0][j][2]);
            err += errij;
        }
        npoints += npt;
    }

    *mean_reprojection_error = err / npoints;

    cout << "average reprojection err = " << err / npoints << endl;

    // save intrinsic parameters
    FileStorage fs(currentCalibImageDir + "/intrinsics.yml", CV_STORAGE_WRITE);
    if (fs.isOpened()) {
        fs << "M1" << cameraMatrix[0] << "D1" << distCoeffs[0] << "M2"
                << cameraMatrix[1] << "D2" << distCoeffs[1];
        fs.release();
    } else
        cout << "Error: can not save the intrinsic parameters\n";

    Mat R1, R2, P1, P2, Q;
    Rect validRoi[2];

    stereoRectify(cameraMatrix[0], distCoeffs[0],
                  cameraMatrix[1], distCoeffs[1],
                  imageSize, R, T, R1, R2, P1, P2, Q,
                  CALIB_ZERO_DISPARITY, 0, imageSize, &validRoi[0], &validRoi[1]);

    fs.open(currentCalibImageDir+"/extrinsics.yml", CV_STORAGE_WRITE);
    if (fs.isOpened()) {
        fs << "R" << R << "T" << T << "R1" << R1 << "R2" << R2 << "P1" << P1
                << "P2" << P2 << "Q" << Q << "F" << F;
        fs.release();
    } else
        cout << "Error: can not save the intrinsic parameters\n";
/*
    // OpenCV can handle left-right
    // or up-down camera arrangements
    bool isVerticalStereo = fabs(P2.at<double> (1, 3)) > fabs(P2.at<double> (0,
            3));

    // COMPUTE AND DISPLAY RECTIFICATION
    if (!showRectified)
*/      return;

}

bool initStereoCalib(const string& imagelistfn, vector<string>& imagelist, string
    currentCalibImageDir, Size boardSize,
        vector<vector<Point2f> > imagePoints[2], vector<vector<Point3f> > &
            objectPoints, Size & imageSize, vector<string>& goodImageList,
        int* nimages) {
    imagelist.resize(0);
    FileStorage fs(imagelistfn, FileStorage::READ);
    if (!fs.isOpened()) {
        cout << "can not open " << imagelistfn << endl
            << " or the string list is empty" << endl;
        return false;
    }

    FileNode n = fs.getFirstTopLevelNode();
    if (n.type() != FileNode::SEQ) {
        cout << "Can not open Top Level Node" << endl;
        return false;
    }
    FileNodeIterator it = n.begin(), it_end = n.end();
    for (; it != it_end; ++it) {
        imagelist.push_back((string) *it);
    }

    if (imagelist.size() % 2 != 0) {
        cout << "Error: the image list contains odd (non-even) number of elements\n";
        return false;
    }

    bool displayCorners = false;//true;
    const int maxScale = 1;
    const float squareSize = 1.f; // Set this to your actual square size
    // ARRAY AND VECTOR STORAGE:

    int i, j, k;
    *nimages = (int) imagelist.size() / 2;

    imagePoints[0].resize(*nimages);
    imagePoints[1].resize(*nimages);

    for (i = j = 0; i < *nimages; i++) {
        for (k = 0; k < 2; k++) {
            const string& filename = imagelist[i * 2 + k];
            Mat img = imread(currentCalibImageDir + "/" + filename, 0);
            if (img.empty())
                break;
            if (imageSize == Size())
                imageSize = img.size();
            else if (img.size() != imageSize) {
                cout << "The image " << filename
```

```
                        << " has the size different from the first image size. Skipping the pair\n";
                    break;
                }
            bool found = false;
            vector<Point2f>& corners = imagePoints[k][j];
//          for (int scale = 1; scale ≤ maxScale; scale++) {
                Mat timg;
//              if (scale == 1)
                    timg = img;
//              else
//                  resize(img, timg, Size(), scale, scale);
                found = findChessboardCorners(timg, boardSize, corners,
                        CV_CALIB_CB_ADAPTIVE_THRESH + CV_CALIB_CB_FAST_CHECK /*+ CV_CALIB_CB_NORMALIZE_IMAGE*/ + CV_CALIB_CB_FILTER_QUADS);
/*              if (found) {
                    if (scale > 1) {
                        Mat cornersMat(corners);
                        cornersMat *= 1. / scale;
                    }
                    break;
                }
            }
*/
/*          if (displayCorners) {
                Mat cimg, cimg1;
                cvtColor(img, cimg, CV_GRAY2BGR);
                drawChessboardCorners(cimg, boardSize, corners, found);
                double sf = 640. / MAX(img.rows, img.cols);
                resize(cimg, cimg1, Size(), sf, sf);
                imshow("corners", cimg1);
                char c = (char) waitKey(500);
                if (c == 27 || c == 'q' || c == 'Q') //Allow ESC to quit
                    exit(-1);
            } else
*/
                putchar('.');
            if (!found) {
                printf("The following image did not find the target. i: %d,j: %d,k: %d
                    ", i, j, k);
                break;
            }
            cornerSubPix(img, corners, Size(11,11), Size(-1, -1),
                    TermCriteria(CV_TERMCRIT_ITER + CV_TERMCRIT_EPS, 30, 0.1));
        }
        if (k == 2) {
            goodImageList.push_back(imagelist[i * 2]);
            goodImageList.push_back(imagelist[i * 2 + 1]);
            j++;
        }
    }
    cout << j << " pairs have been successfully detected.\n";
    *nimages = j;
    if (*nimages < 2) {
        cout << "Error: too little pairs to run the calibration\n";
        return false;
    }

    imagePoints[0].resize(*nimages);
    imagePoints[1].resize(*nimages);
    objectPoints.resize(*nimages);

}
```

B.5 Source Code for Dynamic iSAM Algorithm

Listing B.14: Dynamic iSAM: nonlinearSystem.h

```
#ifndef NONLINEARSYSTEM_H_
#define NONLINEARSYSTEM_H_

#include <Eigen/Dense>
#include <iostream>
#include <math.h>

#define DEFAULT_H               0.05
#define LOWER_H_LIMIT_FACTOR    20
#define RMS_ERR_CUTOFF          1.0e-5
#define INITIAL_H_FACTOR        1


using namespace Eigen;

class nonlinearSystem {
    double h;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    nonlinearSystem();
    VectorXd propagateRK4(double tf, VectorXd x0);
    VectorXd propagateRK4_adaptive(double tf, VectorXd x0);
    void setStepSize(double _h) { h = _h;};
    virtual VectorXd f(VectorXd x) = 0;
};

#endif
```

Listing B.15: Dynamic iSAM: nonlinearSystem.cpp

```
#include "nonlinearSystem.h"

nonlinearSystem::nonlinearSystem() {
    h = DEFAULT_H;
}

VectorXd nonlinearSystem::propagateRK4(double tf, VectorXd x0){
    VectorXd k1;
    VectorXd k2;
    VectorXd k3;
    VectorXd k4;
    double t = 0;
    double dt;
    bool done = false;
    VectorXd x = x0;

//  std::cout << "x(" << t << "): " << x.transpose() << std::endl;

    while (!done) {
        if (tf - h - t > 0) {
            dt = h;
        } else {
            dt = tf - t;
            done = true;
        }

        k1 = dt * this->f(x);
        k2 = dt * this->f(x + 0.5 * k1);
        k3 = dt * this->f(x + 0.5 * k2);
        k4 = dt * this->f(x + k3);
        x = x + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
        t += dt;

//      std::cout << "x(" << t << "): " << x.segment<12>(0).transpose() << std::endl;
    }

    return x;
}

VectorXd nonlinearSystem::propagateRK4_adaptive(double tf, VectorXd x0){
    bool done = false;
    double h_starting = this->h;

    this->h = tf / INITIAL_H_FACTOR;

    VectorXd newX, errX;
    VectorXd currX  = this->propagateRK4(tf, x0);

    while (!done) {

        //new h step size
        this->h = this->h/2;

        //try the new step size
        newX = this->propagateRK4(tf, x0);

        //compute rms error
        errX = newX - currX;
        double rms_err = sqrt(errX.squaredNorm() / errX.size());

//      std::cout << "propagateRK4 Adaptive, h=" << h << ", rms_err=" << rms_err <<
    std::endl;

        //check rms_error or if h is too small that it will take too long
        if (rms_err < RMS_ERR_CUTOFF || this->h ≤ (tf / LOWER_H_LIMIT_FACTOR)) {
            done = true;
            if (this->h ≤ (tf / LOWER_H_LIMIT_FACTOR)) {
            //  std::cout << "adaptive RK4 timestep was cutoff" << std::endl;
            }
        } else {
            currX = newX;
        }
    }

    this->h = h_starting;
    return newX;
}
```

Listing B.16: Dynamic iSAM: rigidBodyDynamics.h

```cpp
#pragma once

#include "nonlinearSystem.h"
#include <Eigen/Dense>
#include <iostream>
#include "inertiaRatios.h"

using namespace Eigen;

class rigidBodyDynamics: public nonlinearSystem {
    isam::inertiaRatios _ir;
    Vector4d _qref;
    Vector3d _r, _v, _a, _w;
    Matrix<double,6,6> _Q;
    double _sigma_v, _sigma_w;

    Matrix3d crossProductMat(Vector3d vec);

public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    rigidBodyDynamics(isam::inertiaRatios ir, double sigma_v, double sigma_w);
    void setMassProperties(isam::inertiaRatios ir);
    void setCovProperties(double sigma_v, double sigma_w);
    VectorXd f(VectorXd x);
    void setState(VectorXd x, Vector4d q);
    void setState(VectorXd x);
    void reset_qref();
    Vector4d qref() const {return _qref;};
    Vector4d qTotal() const;
    VectorXd symmMat2Vec(Matrix<double, 12, 12> M);
    Matrix<double, 12, 12> vec2symmMat(VectorXd v);
    Vector4d quaternionFromRot(Matrix3d& R)const;
    Vector4d mrp2quaternion(Vector3d mrp)const;
    Vector3d quaternion2mrp(Vector4d q) const;
    Vector4d addQuaternionError(Vector3d& mrp, Vector4d& qref) const;
    Vector4d quaternionMultiplication(Vector4d& q1, Vector4d& q2) const;
    Vector4d quaternionDivision(Vector4d& q1, Vector4d& q2) const;
    Vector3d diffQuaternion(Vector4d& q, Vector4d& qprev, double dt) const;
    Matrix3d rotationMatrix(Vector4d& q) const;
    Matrix3d getJ() const;
    isam::inertiaRatios getIR() const;
    void setIR(isam::inertiaRatios ir);
    MatrixXd getBw() const;
    double getSigmaV() const;
    double getSigmaW() const;

    VectorXd x() const;

};
```

Listing B.17: Dynamic iSAM: rigidBodyDynamics.cpp

```

#include "rigidBodyDynamics.h"

rigidBodyDynamics::rigidBodyDynamics(isam::inertiaRatios ir, double sigma_v, double
    sigma_w) {
    _ir = ir;
    _qref << 0, 0, 0, 1;

    _r = Vector3d::Zero();
    _v = Vector3d::Zero();
    _a = Vector3d::Zero();
    _w = Vector3d::Zero();

    setMassProperties(ir);
    setCovProperties(sigma_v, sigma_w);
}

void rigidBodyDynamics::setMassProperties(isam::inertiaRatios ir) {
    _ir = ir;
}

void rigidBodyDynamics::setCovProperties(double sigma_v, double sigma_w) {
    _sigma_v = sigma_v;
    _sigma_w = sigma_w;
    _Q = Matrix<double,6,6>::Zero();
    _Q.block<3,3>(0,0) = _sigma_v * _sigma_v * Matrix<double,3,3>::Identity();
    _Q.block<3,3>(3,3) = _sigma_w * _sigma_w * Matrix<double,3,3>::Identity();
}

void rigidBodyDynamics::reset_qref() {
    Vector3d a_ = _a;
    Vector4d qref_ = _qref;
    _qref = addQuaternionError(a_, qref_);
    _a = Vector3d::Zero();
}

Vector4d rigidBodyDynamics::qTotal() const {
    Vector3d a_ = _a;
    Vector4d qref_ = _qref;
    return addQuaternionError(a_, qref_);
};

VectorXd rigidBodyDynamics::f(VectorXd x) {
    Vector3d dr, dv, da, dw;
    Matrix<double,12,12> lambda, dLambda;
    VectorXd vec_dLambda;
    VectorXd dx(90);

    Vector3d r = x.segment<3>(0);
    Vector3d v = x.segment<3>(3);
    Vector3d a = x.segment<3>(6);
    Vector3d w = x.segment<3>(9);

    MatrixXd Bw = getBw();
    Matrix3d J = _ir.getJ();


    //Nonlinear State Model \dot x = f(x)

    /*
     *  \mathbf{\dot r} = \mathbf{v}
     */
    dr = v;

    /*
     *  \mathbf{\dot v} = 0
     */
    dv = Vector3d::Zero();

    /*
     * \frac{d \mathbf{a}_p}{dt} =
     *          \frac{1}{2}\left(\mathbf{[\omega \times]} +
     *          \mathbf{\omega} \cdot \mathbf{\bar q} \right) \mathbf{a}_p +
     *          \frac{2 q_4}{1+q_4} \mathbf{\omega}
     */
    double c1, c2, c3;
    c1 = 0.5;
    c2 = 0.125 * w.dot(a);
    c3 = 1 - a.dot(a)/16;
    da = -c1 * w.cross(a) + c2* a + c3 * w;



    /*
     * \dot \mathbf{w} = -\mathbf{J}^{-1} \mathbf{\omega} \times \mathbf{J} \mathbf{\
        omega}
     */
    dw = - J.inverse() * w.cross(J * w);


    //Covariance Propagation according to Lyapunov function
    //see Brown & Hwang pg 204

    //Compute Linear transition matrix
    Matrix<double,12,12> A = Matrix<double,12,12>::Zero();

    //position derivative
    A.block<3,3>(0,3) = Matrix<double,3,3>::Identity();

    //mrp kinematics
    A.block<3,3>(6,6) = -0.5*crossProductMat(w) + w.dot(a)/8 * Matrix3d::Identity();
    A.block<3,3>(6,9) = (1-a.dot(a/16))*Matrix3d::Identity();

    //angular velocity dynamics
    A.block<3,3>(9,9) = - J.inverse() * crossProductMat(w) * J;

    lambda = vec2symmMat(x.segment<78>(12));
    dLambda = A * lambda + lambda *A.transpose() + Bw * _Q * Bw.transpose();
    vec_dLambda = symmMat2Vec(dLambda);
    //write to dx
    dx.segment<3>(0) = dr;
    dx.segment<3>(3) = dv;
    dx.segment<3>(6) = da;
    dx.segment<3>(9) = dw;
    dx.segment<78>(12) = vec_dLambda;

    return dx;
}

Matrix3d rigidBodyDynamics::crossProductMat(Vector3d vec) {
    Matrix3d M = Matrix3d::Zero();
    M(0,1) = -vec(2);
    M(0,2) = vec(1);
    M(1,0) = vec(2);
    M(1,2) = -vec(0);
    M(2,0) = -vec(1);
    M(2,1) = vec(0);

    return M;
}

VectorXd rigidBodyDynamics::symmMat2Vec(Matrix<double, 12, 12> M) {
    VectorXd v(78);
    int count = 0;
    for (int row = 0; row < 12; row++) {
        for (int col = row; col < 12; col++) {
            v(count) = M(row,col);
            count++;
        }
    }
    return v;

}

Matrix<double, 12, 12> rigidBodyDynamics::vec2symmMat(VectorXd v) {
    Matrix<double, 12, 12> M = Matrix<double, 12, 12>::Zero();
    int count = 0;
    for (int row = 0; row < 12; row++) {
        for (int col = row; col < 12; col++) {
            M(row,col) = v(count);
            M(col,row) = v(count);
            count++;
        }
    }
    return M;

}

VectorXd rigidBodyDynamics::x() const{
    VectorXd x(12);
    x.segment<3>(0) = _r;
    x.segment<3>(3) = _v;
    x.segment<3>(6) = _a;
    x.segment<3>(9) = _w;
    return x;
}

void rigidBodyDynamics::setState(VectorXd x, Vector4d q) {
    _r = x.segment<3>(0);
    _v = x.segment<3>(3);
    _a = x.segment<3>(6);
    _w = x.segment<3>(9);
    _qref = q / q.norm();
}

void rigidBodyDynamics::setState(VectorXd x) {
    _r = x.segment<3>(0);
    _v = x.segment<3>(3);
    _a = x.segment<3>(6);
    _w = x.segment<3>(9);
```

```
}

Vector4d rigidBodyDynamics::mrp2quaternion(Vector3d mrp) const{
    Vector4d dq;
    dq << 8*mrp / (16 + mrp.transpose() * mrp), (16 - mrp.transpose() * mrp) / (16+mrp
        .transpose() * mrp);
    dq /=dq.norm();

    return dq;
}

Vector3d rigidBodyDynamics::quaternion2mrp(Vector4d q) const{
    Vector3d mrp;
    if (q(3) < 0) {
        q = -q;
    }

    mrp << 4*q(0)/(1+q(3)), 4*q(1)/(1+q(3)), 4*q(2)/(1+q(3));
    return mrp;
}

Vector4d rigidBodyDynamics::addQuaternionError(Vector3d& mrp, Vector4d& qref) const{
    Vector4d qnew, dq;
    dq = mrp2quaternion(mrp);

    Vector4d qnew1 = quaternionMultiplication(dq, qref);

    if (qnew1.dot(qref) ≥ 0) {
        return qnew1;
    } else {
        Vector4d qnew2 = -1 * qnew1;
        return qnew2;
    }
}

Vector4d rigidBodyDynamics::quaternionMultiplication(Vector4d& q1, Vector4d& q2) const
     {
    //q1 \mult q2
    Matrix4d qm;
    Vector4d result;
    qm <<   q1(3),  q1(2),  -q1(1), q1(0),
            -q1(2), q1(3),  q1(0),  q1(1),
            q1(1),  -q1(0), q1(3),  q1(2),
            -q1(0), -q1(1), -q1(2), q1(3);

    result = qm*q2;
    result /= result.norm();

    return result;
}


Vector4d rigidBodyDynamics::quaternionDivision(Vector4d& q1, Vector4d& q2) const {
    Vector4d q2inv;

    q2inv << -q2(0) , -q2(1) , -q2(2) , q2(3);

    Vector4d result = quaternionMultiplication(q1,q2inv);
    return result;
}

Vector3d rigidBodyDynamics::diffQuaternion(Vector4d& q, Vector4d& qprev, double dt)
    const {
    Vector4d dq = (q - qprev) / dt;
    Matrix4d M;

    M <<    qprev(3) , qprev(2),  -qprev(1),   -qprev(0),
            -qprev(2),   qprev(3),   qprev(0),  -qprev(1),
            qprev(1),  -qprev(0),   qprev(3),   -qprev(2),
            qprev(0),  qprev(1),  qprev(2),  qprev(3);

    Vector4d wp = 2*M*dq;
    Vector3d w = wp.head(3);

    return w;
}


Matrix3d rigidBodyDynamics::rotationMatrix(Vector4d& q) const {
    Matrix3d rot;

    rot(0,0) = q(0)*q(0)-q(1)*q(1)-q(2)*q(2)+q(3)*q(3);
    rot(0,1) = 2*(q(0)*q(1)+q(2)*q(3));
    rot(0,2) = 2*(q(0)*q(2)-q(1)*q(3));

    rot(1,0) = 2*(q(0)*q(1)-q(2)*q(3));
    rot(1,1) = -q(0)*q(0)+q(1)*q(1)-q(2)*q(2)+q(3)*q(3);
    rot(1,2) = 2*(q(2)*q(1)+q(0)*q(3));

    rot(2,0) = 2*(q(0)*q(2)+q(1)*q(3));
    rot(2,1) = 2*(q(2)*q(1)-q(0)*q(3));
    rot(2,2) = -q(0)*q(0)-q(1)*q(1)+q(2)*q(2)+q(3)*q(3);

    return rot;
}

Vector4d rigidBodyDynamics::quaternionFromRot(Matrix3d& R) const{
    Vector4d q;
    double div1, div2, div3, div4;

    double numerical_limit = 1.0e-4;

    if (abs(R.determinant()-1) > numerical_limit ) {
        std::cerr << "R does not have a determinant of +1" << std::endl;
    } else {
        div1 = 0.5*sqrt(1+R(0,0)+R(1,1)+R(2,2));
        div2 = 0.5*sqrt(1+R(0,0)-R(1,1)-R(2,2));
        div3 = 0.5*sqrt(1-R(0,0)-R(1,1)+R(2,2));
        div4 = 0.5*sqrt(1-R(0,0)+R(1,1)-R(2,2));

        //if (div1 > div2 && div1 > div3 && div1 > div4) {
        if (fabs(div1) > numerical_limit) {
            q(3) = div1;
            q(0) = 0.25*(R(1,2)-R(2,1))/q(3);
            q(1) = 0.25*(R(2,0)-R(0,2))/q(3);
            q(2) = 0.25*(R(0,1)-R(1,0))/q(3);
        } else if (fabs(div2) > numerical_limit) {
        //} else if (div2 > div1 && div2 > div3 && div2 > div4) {
            q(0) = div2;
            q(1) = 0.25*(R(0,1)+R(1,0))/q(0);
            q(2) = 0.25*(R(0,2)+R(2,0))/q(0);
            q(3) = 0.25*(R(1,2)+R(2,1))/q(0);
        } else if (fabs(div3) > numerical_limit) {
        //} else if (div3 > div1 && div3 > div2 && div3 > div4) {
            q(2) = div3;
            q(0) = 0.25*(R(0,2)+R(2,0))/q(2);
            q(1) = 0.25*(R(1,2)+R(2,1))/q(2);
            q(3) = 0.25*(R(0,1)-R(1,0))/q(2);
        //} else {
        } else if (fabs(div4) > numerical_limit) {
            q(1) = div4;
            q(0) = 0.25*(R(0,1)+R(1,0))/q(1);
            q(2) = 0.25*(R(1,2)+R(2,1))/q(1);
            q(3) = 0.25*(R(2,0)-R(0,2))/q(1);
        } else {
            std::cerr << "quaternionFromRot didn't convert: [" << div1 << ", " << div2
                 << ", " << div3 << ", " << div4 << std::endl;
            std::cerr << "Rotation Matrix: " << R << std::endl;
        }
    }
    q /=q.norm();

    return q;
}

MatrixXd rigidBodyDynamics::getBw() const {
    Matrix<double, 12,6> Bw;
    Bw = Matrix<double,12,6>::Zero();
    Bw.block<3,3>(3,0) = Matrix3d::Identity();
    Bw.block<3,3>(9,3) = Matrix3d::Identity();

    return Bw;
}


Matrix3d rigidBodyDynamics::getJ() const{
    return _ir.getJ();
}

isam::inertiaRatios rigidBodyDynamics::getIR() const{
    return _ir;
}

void rigidBodyDynamics::setIR(isam::inertiaRatios ir) {
    _ir = ir;
}

double rigidBodyDynamics::getSigmaV() const{
    return _sigma_v;
}

double rigidBodyDynamics::getSigmaW() const {
    return _sigma_w;
}
```

Listing B.18: Dynamic iSAM: NodeExmap.h

```
//modified from Node.h - tweddle

#pragma once

#include <list>
#include <Eigen/Dense>

#include <isam/Element.h>
#include <isam/Noise.h>
#include <isam/Node.h>

namespace isam {

template <class T>
class NodeExmapT : public Node {

 protected:
  T* _value;  // current estimate
  T* _value0; // linearization point

public:

  NodeExmapT() : Node(T::name(), T::dim) {
    _value = NULL;
    _value0 = NULL;
  }

  NodeExmapT(const char* name) : Node(name, T::dim) {
    _value = NULL;
    _value0 = NULL;
  }

  virtual ¬NodeExmapT() {
    delete _value;
    delete _value0;
  }

  void init(const T& t) {
    delete _value; delete _value0;
    _value = new T(t); _value0 = new T(t);
  }

  bool initialized() const {return _value != NULL;}

  T value(Selector s = ESTIMATE) const {return (s==ESTIMATE)?*_value:*_value0;}
  T value0() const {return *_value0;}

  Eigen::VectorXd vector(Selector s = ESTIMATE) const {return (s==ESTIMATE)?_value->
      vector():_value0->vector();}
  Eigen::VectorXd vector0() const {return _value0->vector();}

  void update(const Eigen::VectorXd& v) {_value->set(v);}
  void update0(const Eigen::VectorXd& v) {_value0->set(v);}

  void linpoint_to_estimate() {*_value = *_value0;}
  void estimate_to_linpoint() {*_value0 = *_value;}
  void swap_estimates() {T tmp = *_value; *_value = *_value0; *_value0 = tmp;}

/
//  void apply_exmap(const Eigen::VectorXd& v) {*_value = _value0->exmap(v);}
// void self_exmap(const Eigen::VectorXd& v) {*_value0 = _value0->exmap(v);}

  void apply_exmap(const Eigen::VectorXd& v);
  void self_exmap(const Eigen::VectorXd& v) {*_value0 = _value0->exmap(v);}

  void rezero() {
      _value->rezero();
      _value0->rezero();
  }

  void write(std::ostream &out) const {
    out << name() << "_Node " << _id;
    if (_value != NULL) {
      out << " " << value();
    }
  }
};

}

//snippet code goes "elsewhere" for compilation
template <class T> void NodeExmapT<T>::apply_exmap(const Eigen::VectorXd& v) {
    *_value = _value0->exmap_reset(v);

    //update factor noise
    std::list<Factor*> factor_list= this->factors();
    for (std::list<Factor*>::iterator it = factor_list.begin(); it != factor_list.end
        (); it++) {
        Factor* factor = *it;
        dynamicPose3d_NL_dynamicPose3d_NL_Factor * dynamic_factor;
        dynamic_factor = dynamic_cast<dynamicPose3d_NL_dynamicPose3d_NL_Factor *>(
            factor);
        if (dynamic_factor !=0) {
            if (dynamic_factor->checkPose1(this)) {
                //std::cout << "Found Dynamic Factor in apply_exmap(), adjusting noise
                    " << std::endl;
                Eigen::MatrixXd sqrtinf = dynamic_factor->get_sqrtinf();
                Noise newnoise = isam::SqrtInformation(sqrtinf);
                dynamic_factor->setNoise(newnoise);
            }
        }

    }
}
```

Listing B.19: Dynamic iSAM: FactorVariableNoise.h

```
#pragma once

#include <vector>
#include <string>

#include <math.h> // for sqrt
#include <Eigen/Dense>

#include<isam/util.h>
#include<isam/Jacobian.h>
#include<isam/Element.h>
#include<isam/Node.h>
#include<isam/Noise.h>
#include<isam/numericalDiff.h>

namespace isam {


// Generic template for easy instantiation of new factors
template <class T>
class FactorVarNoiseT : public Factor {

    /* Not a const variable
     * This is important because it allows the factor's uncertainty to be updated in
         real-time
     */
    Noise _noise_variable;
    cost_func_t *ptr_cost_func_local;
protected:

  const T _measure;

public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  FactorVarNoiseT(const char* name, int dim, const Noise& noise, const T& measure) :
      Factor(name, dim, noise), _measure(measure) {
      _noise_variable = noise;
      ptr_cost_func_local = NULL;
  }

  virtual void setNoise(Noise& newNoise) {
      _noise_variable = newNoise;
  }

  virtual void set_cost_function(cost_func_t* ptr) {ptr_cost_func_local = ptr;}

  virtual Eigen::VectorXd error(Selector s = ESTIMATE) const {
    Eigen::VectorXd err = _noise_variable.sqrtinf() * basic_error(s);
    // optional modified cost function
    if (*ptr_cost_func_local) {
      for (int i=0; i<err.size(); i++) {
        double val = err(i);
        err(i) = ((val≥0)?1.:(-1.)) * sqrt((*ptr_cost_func_local)(val));
      }
    }
    return err;
  }

  virtual const Eigen::MatrixXd& sqrtinf() const {return _noise_variable.sqrtinf();}

  const T& measurement() const {
      return _measure;
  }

  void write(std::ostream &out) const {
    Factor::write(out);
    out << " " << _measure << " " << noise_to_string(_noise_variable);
  }

};


}
```

Listing B.20: Dynamic iSAM: slam_dynamic3d_NL.h

```

#ifndef SLAMDYNAMICS_H_
#define SLAMDYNAMICS_H_


#include <Eigen/Dense>
#include "dynamicPose3d_NL.h"
#include "camera3d.h"
#include "FactorVariableNoise.h"
#include <isam/Node.h>
#include <isam/Factor.h>
#include <isam/Pose3d.h>
#include <isam/Point3d.h>
#include <isam/slam_stereo.h>
#include "NodeExmap.h"
#include "inertiaRatios.h"
#include "kinematicPose3d.h"


namespace isam{

typedef NodeExmapT<dynamicPose3d_NL> dynamicPose3d_NL_Node;
typedef NodeT<Point3d> Point3d_Node;

/**
 * Prior on dynamicPose3d.
 */
class dynamicPose3d_NL_Factor : public FactorT<dynamicPose3d_NL> {
public:
  dynamicPose3d_NL_Node* _pose;

  dynamicPose3d_NL_Factor(dynamicPose3d_NL_Node* pose, const dynamicPose3d_NL& prior,
      const Noise& noise)
    : FactorT<dynamicPose3d_NL>("dynamicPose3d_NL_Factor", 12, noise, prior), _pose(
        pose) {
    _nodes.resize(1);
    _nodes[0] = pose;
  }

  void initialize() {
    if (!_pose->initialized()) {
      dynamicPose3d_NL predict = _measure;
      _pose->init(predict);
    }
  }

  Eigen::VectorXd basic_error(Selector s = ESTIMATE) const {

      dynamicPose3d_NL p1 = _pose->value(s);
      Eigen::VectorXd err = p1.vectorFull() - _measure.vector();

      Eigen::Vector4d p1qTot = p1.qTotal();
      Eigen::Vector4d mqTot = _measure.qTotal();
      Vector3d da = p1.getMRPdifference(p1qTot, mqTot);
      err.segment<3>(6) = da;

      return err;
  }
};


//Process Model Factor - one of the main contributions of this thesis
class dynamicPose3d_NL_dynamicPose3d_NL_Factor : public FactorVarNoiseT<
    dynamicPose3d_NL > {
  dynamicPose3d_NL_Node* _pose1;
  dynamicPose3d_NL_Node* _pose2;
  inertiaRatios_Node* _ir_node;
  double dt;

public:

  /**
   * Constructor.
   * @param pose1 The pose from which the measurement starts.
   * @param pose2 The pose to which the measurement extends.
   * @param measure DOES NOTHING - DON'T USE IT!!!! (could be extended in future
       release to add forces/torques
   * @param noise The 12x12 square root information matrix (upper triangular).
   */
  dynamicPose3d_NL_dynamicPose3d_NL_Factor(dynamicPose3d_NL_Node* pose1,
      dynamicPose3d_NL_Node* pose2, inertiaRatios_Node* ir_node,
    const dynamicPose3d_NL& measure, const Noise& noise, double timestep)
    : FactorVarNoiseT<dynamicPose3d_NL>("dp3dNL_dp3dNL_IR_Factor", 12, noise, measure)
      ,
    _pose1(pose1), _pose2(pose2), _ir_node(ir_node), dt(timestep) {
    _nodes.resize(3);
    _nodes[0] = pose1;
    _nodes[1] = pose2;
    _nodes[2] = ir_node;
  }

  void initialize() {
    dynamicPose3d_NL_Node* pose1 = _pose1;
    dynamicPose3d_NL_Node* pose2 = _pose2;
    inertiaRatios_Node* ir_node = _ir_node;
    require(pose1->initialized() || pose2->initialized(),
        "dynamicSLAM: dynamicPose3d_NL_dynamicPose3d_NL_Factor requires pose1 or pose2
             to be initialized");

    if(!_ir_node->initialized()) {
        inertiaRatios init_ir;
        _ir_node->init(init_ir);
    }

    if (!pose1->initialized() && pose2->initialized()) {
        std::cout << "No BACKWARDS PROPAGATE" << std::endl;
    } else if (pose1->initialized() && !pose2->initialized()) {
        inertiaRatios ir = _ir_node->value();
      dynamicPose3d_NL a = pose1->value();
      dynamicPose3d_NL predict = a.propagate(dt, ir);
      pose2->init(predict);
    }
  }

  Eigen::VectorXd basic_error(Selector s = ESTIMATE) const {

    dynamicPose3d_NL p1 = _pose1->value(s);
    dynamicPose3d_NL p2 = _pose2->value(s);
    inertiaRatios ir = _ir_node->value(s);

    Eigen::VectorXd err = p2.computeStateChange(p1, dt, ir);

    return err;
  }

  Eigen::MatrixXd get_sqrtinf() const {
      inertiaRatios ir = _ir_node->value();
      Eigen::MatrixXd new_sqrtinf = _pose1->value().getProcessNoise(dt,ir)._sqrtinf;
      return new_sqrtinf;
  }

  bool checkPose1(dynamicPose3d_NL_Node* poseRef) {
      if(_pose1 == poseRef) {
          return true;
      } else {
          return false;
      }
  }

  bool checkPose1(inertiaRatios_Node* ir_node) {
      if(_ir_node == ir_node) {
          return true;
      } else {
          return false;
      }
  }

  bool checkPose1(kinematicPose3d_Node* poseRef) {
      return false;
  }

  double get_dt() { return dt;}

  void write(std::ostream &out) const {
      FactorVarNoiseT<dynamicPose3d_NL >::write(out);
  }
};


typedef NodeT<Point3dh> Point3dh_Node;

//stereo camera class
class StereoCameraDebug { // for now, camera and robot are identical
  double _f;
  Eigen::Vector2d _pp;
  double _b;

public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  StereoCameraDebug() : _f(1), _pp(Eigen::Vector2d(0.5,0.5)), _b(0.1) {}
  StereoCameraDebug(double f, const Eigen::Vector2d& pp, double b) : _f(f), _pp(pp),
      _b(b) {}

  inline double focalLength() const {return _f;}

  inline Eigen::Vector2d principalPoint() const {return _pp;}

  inline double baseline() const {return _b;}

  StereoMeasurement project(const Pose3d& pose, const Point3dh& Xw) const {
    Point3dh X = pose.transform_to(Xw);
    // camera system has z pointing forward, instead of x
    double x = -X.y();
    double y = -X.z();
    double z = X.x();

    // left camera
    double fz = _f / z;
    double u = x * fz + _pp(0);
    double v = y * fz + _pp(1);
    // right camera
    double u2 = u -_b*fz;
    bool valid = ( z > 0.0); // infront of camera?

    if (valid == false) {
        std::cout << "invalid." << std::endl;
    }

    return StereoMeasurement(u, v, u2, valid);
  }

  StereoMeasurement project(const cameraPose3d& pose, const Point3dh& Xw) const {
    Point3dh X = pose.transform_to(Xw);
    // camera system has z pointing forward, instead of x
    double x = -X.y();
    double y = -X.z();
    double z = X.x();

    // left camera
    double fz = _f / z;
    double u = x * fz + _pp(0);
    double v = y * fz + _pp(1);
    // right camera
    double u2 = u -_b*fz;
    bool valid = ( z > 0.0); // infront of camera?
    if (valid == false) {
        std::cout << "invalid." << std::endl;
    }

    return StereoMeasurement(u, v, u2, valid);
  }


  Point3dh backproject(const Pose3d& pose, const StereoMeasurement& measure) const {
    double disparity = measure.u - measure.u2;
    double lz = _f*_b / disparity;
    double lx = (measure.u-_pp(0))*lz / _f;
    double ly = (measure.v-_pp(1))*lz / _f;
    if (disparity<0.) {
      std::cout << "Warning: StereoCameraDebug.backproject called with negative
          disparity\n";
    }

    Point3dh X(lz, -lx, -ly, 1.0);

    return pose.transform_from(X);
  }

  Point3dh backproject(const cameraPose3d& pose, const StereoMeasurement& measure)
      const {

    double disparity = measure.u - measure.u2;
    double lz = _f*_b / disparity;
    double lx = (measure.u-_pp(0))*lz / _f;
    double ly = (measure.v-_pp(1))*lz / _f;
    if (disparity<0.) {
      std::cout << "Warning: StereoCameraDebug.backproject called with negative
          disparity\n";
    }
    Point3dh X(lz, -lx, -ly, 1.0);
    return pose.transform_from(X);
  }

};


//stereo measurement factor with geometric frame reference
class dStereo_MovingMap_CoM_Factor : public FactorT<StereoMeasurement> {
  dynamicPose3d_NL_Node* _pose;
  Point3d_Node* _point;
  Point3dh_Node* _point_h;
  StereoCameraDebug* _camera;
  cameraPose3d_Node* _camera_pose3d;
  kinematicPose3d_Node* _centerOfMass_princAxes;

  Point3dh predict_inertial_stored;

public:
```

```

// constructor for projective geometry
dStereo_MovingMap_CoM_Factor(dynamicPose3d_NL_Node* pose, Point3dh_Node* point,
    StereoCameraDebug* camera, cameraPose3d_Node* camera_pose3d,
    kinematicPose3d_Node* centerOfMass_princAxes,
                       const StereoMeasurement& measure, const Noise& noise)
  : FactorT<StereoMeasurement>("Stereo_Factor COM", 3, noise, measure),
    _pose(pose), _point(NULL), _point_h(point), _camera(camera), _camera_pose3d(
        camera_pose3d), _centerOfMass_princAxes(centerOfMass_princAxes) {
  // StereoCameraDebug could also be a node later (either with 0 variables,
  // or with calibration as variables)
  _nodes.resize(3);
  _nodes[0] = pose;
  _nodes[1] = _centerOfMass_princAxes;
  _nodes[2] = point;

}

// constructor for Euclidean geometry
// WARNING: only use for points at short range
dStereo_MovingMap_CoM_Factor(dynamicPose3d_NL_Node* pose, Point3d_Node* point,
    StereoCameraDebug* camera, cameraPose3d_Node* camera_pose3d,
    kinematicPose3d_Node* centerOfMass_princAxes,
                       const StereoMeasurement& measure, const Noise& noise)
  : FactorT<StereoMeasurement>("Stereo_Factor COM", 3, noise, measure),
    _pose(pose), _point(point), _point_h(NULL), _camera(camera), _camera_pose3d(
        camera_pose3d), _centerOfMass_princAxes(centerOfMass_princAxes) {
  _nodes.resize(3);
  _nodes[0] = pose;
  _nodes[1] = _centerOfMass_princAxes;
  _nodes[2] = point;
  }

void initialize() {
  require(_pose->initialized(), "dynamic Stereo_Factor requires pose to be
      initialized");
  if(!_centerOfMass_princAxes->initialized()) {
      kinematicPose3d com_pa_init;
      _centerOfMass_princAxes->init(com_pa_init);
  }
  bool initialized = (_point_h!=NULL) ? _point_h->initialized() : _point->
      initialized();
  if (!initialized) {
    Point3dh predict_inertial = _camera->backproject(_camera_pose3d->value(),
        _measure);
    predict_inertial_stored = predict_inertial;

    Point3dh predict_body = _pose->value().transform_to_body(predict_inertial);

    Point3dh predict_feature(_centerOfMass_princAxes->value().oTw() * predict_body.
        vector());

    //subtract Center of mass offset
    Vector3d vec_point_feat_frame = predict_feature.vector().head(3);// -
        _com_offset->value().vector();
    Point3dh point_com = Point3dh(Point3d(vec_point_feat_frame));

    if (_point_h!=NULL) {
      _point_h->init(point_com);
    } else {
      _point->init(point_com.to_point3d());
    }
  }
}

Eigen::VectorXd basic_error(Selector s = ESTIMATE) const {
  //point in body feature frame
  Point3dh point = (_point_h!=NULL) ? _point_h->value(s) : _point->value(s);

  //add center of mass offset
  Vector4d vec_point_feat_frame;
  vec_point_feat_frame << point.vector().head(3), 1.0;
  Vector3d vec_point_com_frame =  (_centerOfMass_princAxes->value(s).wTo()*
      vec_point_feat_frame).head(3);
  Point3dh point_com = Point3dh(Point3d(vec_point_com_frame));

  //transform from body frame to inertial frame
  Point3dh inertialPoint = _pose->value(s).transform_to_inertial(point_com);

  //project into camera
  StereoMeasurement predicted = _camera->project(_camera_pose3d->value(s),
      inertialPoint);

  //create error measurement
  if (_point_h!=NULL || predicted.valid == true) {
    return (predicted.vector() - _measure.vector());
  } else {
    std::cout << "Warning - dynamicStereo_MovingMap_Factor.basic_error: point behind
         camera, dropping term.\n";
    std::cout << "_camera_pose3d->value(s): " << _camera_pose3d->value(s) << std::
            endl;
    std::cout << "_pose->value(s): " << _pose->value(s) << std::endl;
    std::cout << "inertialPoint: " << inertialPoint << std::endl << std::endl;
    return Eigen::Vector3d::Zero();
  }
}

};

//point3d prior factor
class Point3d_Factor : public FactorT<Point3d> {
Point3d_Node* _point;

public:

Point3d_Factor(Point3d_Node* point, const Point3d& prior, const Noise& noise)
  : FactorT<Point3d>("Point3d_Factor", 3, noise, prior), _point(point) {
  _nodes.resize(1);
  _nodes[0] = point;
}

void initialize() {
  if (!_point->initialized()) {
    Point3d predict = _measure;
    _point->init(predict);
  }
}

Eigen::VectorXd basic_error(Selector s = ESTIMATE) const {
  return (_point->vector(s) - _measure.vector());
}

};


}

#endif
```

Listing B.21: Dynamic iSAM: kinematicPose3d.h

```
#pragma once

#include <cmath>
#include <Eigen/Dense>
#include <Eigen/Geometry>
#include "NodeExmap.h"


namespace isam {

typedef Eigen::Matrix<double, 6, 1> Vector6d;

class kinematicPose3d {
    frend std::ostream& operator<<(std::ostream& out, const kinematicPose3d& p) {
        p.write(out);
        return out;
    }
    Eigen::Vector4d _qref;
    Eigen::Vector3d _r;
    Eigen::Vector3d _a;

public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    static const int dim = 6;
    static const char* name() {
        return "kinematicPose3d";
    }

    kinematicPose3d()  {
        _qref << 0.0, 0.0, 0.0, 1.0;
        _a << 0.0, 0.0, 0.0;
        _r << 0.0, 0.0, 0.0;
    }



    kinematicPose3d(const Eigen::MatrixXd& hm) {
        //Convert matrix to R,T
        Eigen::Matrix4d HM = hm / hm(3,3); // enforce T(3,3)=1
        Eigen::Matrix3d R = HM.topLeftCorner(3,3);
        Eigen::Vector3d _r = HM.col(3).head(3);

        //compute quaternion
        _qref = quaternionFromRot(R);
        _a = Eigen::Vector3d::Zero();
    }

    Eigen::VectorXd x() const{
        Vector6d x;
        Eigen::Vector3d r = _r;
        Eigen::Vector3d a = _a;
        x.segment<3>(0) = r;
        x.segment<3>(3) = a;
        return x;
    }

    void setState(Eigen::VectorXd x, Eigen::Vector4d q) {
        _r = x.segment<3>(0);
        _a = x.segment<3>(3);
        _qref = q / q.norm();
    }

    void setState(Eigen::VectorXd x) {
        _r = x.segment<3>(0);
        _a = x.segment<3>(3);
    }

    Eigen::Vector4d mrp2quaternion(Eigen::Vector3d mrp) const{
        Eigen::Vector4d dq;
        dq << 8*mrp / (16 + mrp.transpose() * mrp), (16 - mrp.transpose() * mrp) /
            (16+mrp.transpose() * mrp);
        dq /=dq.norm();
        return dq;
    }

    Eigen::Vector3d quaternion2mrp(Eigen::Vector4d q) const{
        Eigen::Vector3d mrp;
        if (q(3) < 0) {
            q = -q;
        }

        mrp << 4*q(0)/(1+q(3)), 4*q(1)/(1+q(3)), 4*q(2)/(1+q(3));
        return mrp;
    }


    Eigen::Vector4d addQuaternionError(Eigen::Vector3d& mrp, Eigen::Vector4d& qref)
        const{
        Eigen::Vector4d qnew, dq;
        dq = mrp2quaternion(mrp);

        qnew = quaternionMultiplication(dq, qref);

        return qnew;
    }

    Eigen::Vector4d quaternionMultiplication(Eigen::Vector4d& q1, Eigen::Vector4d& q2)
         const {
        //q1 \mult q2
        Eigen::Matrix4d qm;
        Eigen::Vector4d result;
        qm <<   q1(3),  q1(2),  -q1(1), q1(0),
                -q1(2), q1(3),  q1(0),  q1(1),
                q1(1),  -q1(0), q1(3),  q1(2),
                -q1(0), -q1(1), -q1(2), q1(3);

        result = qm*q2;
        result /= result.norm();

        return result;
    }

    Eigen::Vector4d quaternionDivision(Eigen::Vector4d& q1, Eigen::Vector4d& q2) const
         {
        Eigen::Vector4d q2inv;

        q2inv << -q2(0) , -q2(1) , -q2(2) , q2(3);

        Eigen::Vector4d result = quaternionMultiplication(q1,q2inv);
        return result;
    }


    Eigen::Matrix3d rotationMatrix(Eigen::Vector4d& q) const {
        Eigen::Matrix3d rot;

        rot(0,0)  = q(0)*q(0)-q(1)*q(1)-q(2)*q(2)+q(3)*q(3);
        rot(0,1)  = 2*(q(0)*q(1)+q(2)*q(3));
        rot(0,2)  = 2*(q(0)*q(2)-q(1)*q(3));

        rot(1,0)  = 2*(q(0)*q(1)-q(2)*q(3));
        rot(1,1)  = -q(0)*q(0)+q(1)*q(1)-q(2)*q(2)+q(3)*q(3);
        rot(1,2)  = 2*(q(2)*q(1)+q(0)*q(3));

        rot(2,0) = 2*(q(0)*q(2)+q(1)*q(3));
        rot(2,1) = 2*(q(2)*q(1)-q(0)*q(3));
        rot(2,2) = -q(0)*q(0)-q(1)*q(1)+q(2)*q(2)+q(3)*q(3);

        return rot;
    }

    Eigen::Vector4d quaternionFromRot(Eigen::Matrix3d& R) const{
        Eigen::Vector4d q;
        double div1, div2, div3, div4;

        double numerical_limit = 1.0e-4;

        if (abs(R.determinant()-1) > numerical_limit ) {
            std::cerr << "R does not have a determinant of +1" << std::endl;
        } else {
            div1 = 0.5*sqrt(1+R(0,0)+R(1,1)+R(2,2));
            div2 = 0.5*sqrt(1+R(0,0)-R(1,1)-R(2,2));
            div3 = 0.5*sqrt(1-R(0,0)-R(1,1)+R(2,2));
            div4 = 0.5*sqrt(1-R(0,0)+R(1,1)-R(2,2));

            //if (div1 > div2 && div1 > div3 && div1 > div4) {
            if (fabs(div1) > numerical_limit) {
                q(3) = div1;
                q(0) = 0.25*(R(1,2)-R(2,1))/q(3);
                q(1) = 0.25*(R(2,0)-R(0,2))/q(3);
                q(2) = 0.25*(R(0,1)-R(1,0))/q(3);
            } else if (fabs(div2) > numerical_limit) {
            //} else if (div2 > div1 && div2 > div3 && div2 > div4) {
                q(0) = div2;
                q(1) = 0.25*(R(0,1)+R(1,0))/q(0);
                q(2) = 0.25*(R(0,2)+R(2,0))/q(0);
                q(3) = 0.25*(R(1,2)+R(2,1))/q(0);
            } else if (fabs(div3) > numerical_limit) {
            //} else if (div3 > div1 && div3 > div2 && div3 > div4) {
                q(2) = div3;
                q(0) = 0.25*(R(0,2)+R(2,0))/q(2);
                q(1) = 0.25*(R(1,2)+R(2,1))/q(2);
                q(3) = 0.25*(R(0,1)-R(1,0))/q(2);
            //} else {
            } else if (fabs(div4) > numerical_limit) {
                q(1) = div4;
                q(0) = 0.25*(R(0,1)+R(1,0))/q(1);
                q(2) = 0.25*(R(1,2)+R(2,1))/q(1);
                q(3) = 0.25*(R(2,0)-R(0,2))/q(1);
            } else {
                std::cerr << "quaternionFromRot didn't convert: [" << div1 << ", " <<
                    div2 << ", " << div3 << ", " << div4 << std::endl;
                std::cerr << "Rotation Matrix: " << R << std::endl;
            }
        }
        q /=q.norm();

        return q;
    }


    Eigen::Vector3d r() const {return _r;}
    Eigen::Vector3d a()   const {return _a;}
    Eigen::Vector4d qref() const {return _qref;}


    void reset_qref() {
        Eigen::Vector3d a_ = _a;
        Eigen::Vector4d qref_ = _qref;
        _qref = addQuaternionError(a_, qref_);
        _a = Eigen::Vector3d::Zero();
    }

    Eigen::Vector4d qTotal() const {
        Eigen::Vector3d a_ = _a;
        Eigen::Vector4d qref_ = _qref;
        return addQuaternionError(a_, qref_);
    };



    kinematicPose3d exmap(const Vector6d& Δ) {
        kinematicPose3d res = *this;
        res._r += Δ.head(3);
        res._a += Δ.tail(3);
        return res;
    }

    kinematicPose3d exmap_reset(const Vector6d& Δ) {
        kinematicPose3d res = *this;
        res._r += Δ.head(3);
        res._a += Δ.tail(3);
        res.reset_qref();
        return res;
    }
    Vector6d vector() const {
        Vector6d tmp;
        tmp << _r, _a;
        return tmp;
    }

    void set(const Vector6d& v) {
        _r = v.head(3);
        _a = v.tail(3);
    }

    void write(std::ostream &out) const {
        out << std::endl << "kinPose3d x: " << x().transpose() << std::endl;
        out <<  "kinPose3d qref: " <<  qref().transpose() << std::endl;
        out << std::endl;
    }


    /**
     * Convert Pose3 to homogeneous 4x4 transformation matrix.
     * The returned matrix is the object coordinate frame in the world
     * coordinate frame. In other words it transforms a point in the object
     * frame to the world frame.
     *
     * @return wTo
     */
    Eigen::Matrix4d wTo() const {
        /*
        Eigen::Matrix4d T;
        Eigen::Vector4d qtot = qTotal();
        T.topLeftCorner(3,3) = rotationMatrix(qtot).transpose();
        T.col(3).head(3) = _r;
        T.row(3) << 0., 0., 0., 1.;
        return T;
        */
        Eigen::Vector4d qtot = qTotal();
        Eigen::Matrix3d R = rotationMatrix(qtot);
        Eigen::Matrix3d oRw = R;
        Eigen::Vector3d C = - oRw * _r;
        Eigen::Matrix4d T;
        T.topLeftCorner(3,3) = oRw;
        T.col(3).head(3) = C;
        T.row(3) << 0., 0., 0., 1.;
        return T;

    }

    /**
     * Convert Pose3 to homogeneous 4x4 transformation matrix. Avoids inverting wTo.
     * The returned matrix is the world coordinate frame in the object
     * coordinate frame. In other words it transforms a point in the world
     * frame to the object frame.
     *
     * @return oTw
     */
    Eigen::Matrix4d oTw() const {
        Eigen::Matrix4d T;
        Eigen::Vector4d qtot = qTotal();
        T.topLeftCorner(3,3) = rotationMatrix(qtot).transpose();
        T.col(3).head(3) = _r;
        T.row(3) << 0., 0., 0., 1.;
        return T;

    }


};

typedef NodeExmapT<kinematicPose3d> kinematicPose3d_Node;

    class kinematicPose3d_Factor : public FactorT<kinematicPose3d> {
    public:
        kinematicPose3d_Node* _pose;

        kinematicPose3d_Factor(kinematicPose3d_Node* pose, const kinematicPose3d&
            prior, const Noise& noise)
        : FactorT<kinematicPose3d>("kinematicPose3d_Factor", 6, noise, prior), _pose(
            pose) {
        _nodes.resize(1);
        _nodes[0] = pose;
      }

      void initialize() {
        if (!_pose->initialized()) {
          kinematicPose3d predict = _measure;
          _pose->init(predict);
        }
      }

      Eigen::VectorXd basic_error(Selector s = ESTIMATE) const {

          kinematicPose3d p1 = _pose->value(s);
          Eigen::VectorXd err = p1.vector() - _measure.vector();
          Eigen::Vector4d q1_tot = p1.qTotal();
          Eigen::Vector4d qm_tot = _measure.qTotal();
          Eigen::Vector4d dq = p1.quaternionDivision(q1_tot,qm_tot);
          Eigen::Vector3d da = p1.quaternion2mrp(dq);

          err.segment<3>(3) = da;

        return err;
      }
    };
}
```

Listing B.22: Dynamic iSAM: dynamicsPose3d_NL.h

```

#pragma once

#include <ostream>
#include <Eigen/Dense>
#include "isam/isam.h"
#include "rigidBodyDynamics.h"
#include "FactorVariableNoise.h"


using namespace Eigen;
namespace isam{

class dynamicPose3d_NL {
    frend std::ostream& operator<<(std::ostream& out, const dynamicPose3d_NL& p)
    {
        p.write(out);
        return out;
    }

    rigidBodyDynamics rbd;
public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
  // assignment operator and copy constructor implicitly created, which is ok
  static const int dim = 12;
  static const char* name() {
    return "dynamicPose3d_NL";
  }

  Noise* factor_noise;      //check if this is ever used

  dynamicPose3d_NL(inertiaRatios ir, double sigma_v, double sigma_w) : rbd(ir, sigma_v
      , sigma_w) {
  }

  //copy constructor
  dynamicPose3d_NL(const dynamicPose3d_NL& cSource) :
      rbd(cSource.rbd.getIR(), cSource.rbd.getSigmaV(), cSource.rbd.getSigmaW() )
  {
      rbd.setState(cSource.rbd.x(), cSource.rbd.qref());
  }

  dynamicPose3d_NL& operator= (const dynamicPose3d_NL& cSource) {
      rbd = rigidBodyDynamics(cSource.rbd.getIR(), cSource.rbd.getSigmaV(), cSource.
              rbd.getSigmaW() );
      rbd.setState(cSource.rbd.x(), cSource.rbd.qref());
      return *this;
  }

  dynamicPose3d_NL(VectorXd x, inertiaRatios ir, double sigma_v, double sigma_w)
  : rbd(ir, sigma_v, sigma_w)
  {
      Vector4d qref;
      qref << 0, 0, 0, 1;
      if (x.size() == 12) {
          rbd.setState(x,qref);
      }
  }

  dynamicPose3d_NL(VectorXd x, Vector4d qref, inertiaRatios ir, double sigma_v, double
       sigma_w)
  : rbd(ir, sigma_v, sigma_w)
  {
      if (x.size() == 12) {
          rbd.setState(x,qref);
      }
  }

  dynamicPose3d_NL(const Matrix4d& hm, bool initVelocities, double dt, inertiaRatios
      ir, double sigma_v, double sigma_w)
  : rbd(ir, sigma_v, sigma_w)
  {
      Matrix<double,12,1> x;
      Vector3d r;
      Vector3d v;
      Vector3d a;
      Vector4d q;
      Vector3d w;

      //Convert matrix to R,T
      Matrix4d HM = hm / hm(3,3); // enforce T(3,3)=1
      Matrix3d R = HM.topLeftCorner(3,3);
      Vector3d T = HM.col(3).head(3);

      //compute quaternion
      q = rbd.quaternionFromRot(R);
      a = Vector3d::Zero();

      if (initVelocities) {
          //differentiate linear velocity
          v = T / dt;

          /* Differentiate quaternion:
           * dq/dt = (q[k] - q[k-1])/dt = 0.5 O(w[k-1]) q[k-1]
           * where O(w[k-1]) is an orthonormal quaternion mult matrix for [w1; w2; w3;
               0] (i.e. quaternionMultiplication())
           * set q[k-1] = [0;0;0;1] by definition (from a refererence frame) and solve
               for w[k-1] gives
           * w[k-1] = 2 [q1[k]; q2[k]; q3[k]] / dt
           */
          w = 2*q.head(3) / dt;
      } else {
          v = Vector3d::Zero();
          w = Vector3d::Zero();
      }

      x.block<3,1>(0,0) = T;
      x.block<3,1>(3,0) = v;
      x.block<3,1>(6,0) = a;
      x.block<3,1>(9,0) = w;
      rbd.setState(x,q);
  }

  VectorXd x() { return rbd.x();};
  Vector4d q() { return rbd.qref();};
  Vector4d qTotal() const { return rbd.qTotal(); };

  dynamicPose3d_NL exmap(const Matrix<double,12,1>& Δ) const {
      dynamicPose3d_NL res = *this;
      res.rbd.setState(res.rbd.x() + Δ);
      return res;
    }

  dynamicPose3d_NL exmap_reset(const Matrix<double,12,1>& Δ)  {
      dynamicPose3d_NL res = *this;
      res.rbd.setState(res.rbd.x() + Δ);
      res.rbd.reset_qref();

    /* We should REALLY, REALLY update Factor::_sqrtinf at this location
     * in the code. However it is a const variable, and there is no way
     * to do callbacks to the Factor class. So I will leave this for future
     * work. Now, the value that is created on initialization is the final
     * version, even after many relinearizations.
     */

    //NOTE - THIS IS NOW DONE in NodeExmapT->apply_reset();

    return res;
  }

  void set(const VectorXd& v) {
      rbd.setState(v);
  }

  void set_qref(const Vector4d& qset) {
      rbd.setState(rbd.x(), qset);
  }

  void rezero() {
      VectorXd x = VectorXd::Zero(12);
      Vector4d q;
      q << 0 , 0, 0, 1;
      rbd.setState(x,q);
  }


  dynamicPose3d_NL propagate(double dt, inertiaRatios& ir) {
      VectorXd x0 = VectorXd::Zero(90);
      x0.head(12) = rbd.x();
      rbd.setIR(ir);
//    std::cout << "x0: " << x0.head(12).transpose() << std::endl;
      VectorXd newX = rbd.propagateRK4_adaptive(dt, x0).head(12);

//    std::cout << "dt: " << dt << std::endl;
//    std::cout << "newX: " << newX.transpose() << std::endl;

      dynamicPose3d_NL xNew(newX, this->rbd.qref(), this->rbd.getIR(), this->rbd.
          getSigmaV(), this->rbd.getSigmaW());
      xNew.exmap(Matrix<double,12,1>::Zero());
      return xNew;
  }

  Vector3d getMRPdifference(Vector4d qtot1, Vector4d qtot2)  {
      Vector4d dq = rbd.quaternionDivision(qtot1,qtot2);
      Vector3d da = rbd.quaternion2mrp(dq);
      return da;
  }

  //compute the control input using w_t = x_{t+1} - \int_t^{t+1}f(x_t)
```

```
    VectorXd computeStateChange(dynamicPose3d_NL& prev, double dt, inertiaRatios& ir) {
        VectorXd w;

        dynamicPose3d_NL predicted = prev.propagate(dt, ir);

        Vector4d qTot = this->qTotal();
        Vector4d qTotpred = predicted.qTotal();
        Vector3d da = getMRPdifference(qTot,qTotpred);

        w = this->x() - predicted.x();
        w.segment<3>(6) = da;

        return w;
    }

    Vector6d getOdometry() {
        Vector6d odo;
        VectorXd x = rbd.x();
        Vector4d qref = rbd.qref();
        Vector3d a = x.segment<3>(6);
        odo.head(3) = x.segment<3>(0);
        Vector4d qnew = rbd.addQuaternionError(a,qref);
        odo.tail(3) = rbd.quaternion2mrp(qnew);
        return odo;
    }

    Vector6d getOdometry(dynamicPose3d_NL& prev) {
        Vector6d odo;
        VectorXd x, xprev;
        Vector4d q, qprev;
        Vector3d a, aprev;

        x = rbd.x();
        xprev = prev.x();
        a = x.segment<3>(6);

        q = rbd.qref();

        qprev = prev.q();

        aprev = xprev.segment<3>(6);

        Vector3d dr = x.segment<3>(0) - xprev.segment<3>(0);
        Vector4d qtot_this = rbd.addQuaternionError(a, q);
        Vector4d qtot_prev = rbd.addQuaternionError(aprev, qprev);

        Vector4d qprev_inv;
        qprev_inv << -qtot_prev(0), -qtot_prev(1), -qtot_prev(2), qtot_prev(3);
        Vector4d qDiff = rbd.quaternionMultiplication(qtot_this, qprev_inv);
        Vector3d mrp = rbd.quaternion2mrp(qDiff);
        odo.tail(3) = mrp;

        Matrix3d Rprev = rbd.rotationMatrix(qtot_prev);
        odo.head(3) = Rprev.transpose() * dr;

        return odo;
    }

    dynamicPose3d_NL getOdometryPose(dynamicPose3d_NL& prev, bool initVelocities, double
         dt) {
        dynamicPose3d_NL newPose(prev.rbd.getIR(), prev.rbd.getSigmaV(), prev.rbd.
            getSigmaW());
        VectorXd new_x(12);
        Vector3d new_r;
        Vector3d new_v;
        Vector3d new_a;
        Vector4d new_q;
        Vector3d new_w;

        VectorXd x, xprev;
        Vector4d q, qprev;
        Vector3d a, aprev;

        //get x's
        x = rbd.x();
        xprev = prev.x();

        //attitude gets
        a = x.segment<3>(6);
        aprev = xprev.segment<3>(6);
        q = rbd.qref();
        qprev = prev.q();
        //total attitude
        Vector4d qtot_this = rbd.addQuaternionError(a, q);
        Vector4d qtot_prev = rbd.addQuaternionError(aprev, qprev);
        Vector4d qprev_inv;
        qprev_inv << -qtot_prev(0), -qtot_prev(1), -qtot_prev(2), qtot_prev(3);
        Vector4d qDiff = rbd.quaternionMultiplication(qtot_this, qprev_inv);
        //previous rotation mat
        Matrix3d Rprev = rbd.rotationMatrix(qtot_prev);

        new_r = Rprev.transpose()*(x.segment<3>(0) - xprev.segment<3>(0));
        Matrix3d Rdiff = rbd.rotationMatrix(qDiff);
        new_q = rbd.quaternionFromRot(Rdiff);

        if (initVelocities) {
            //differentiate linear velocity
            new_v = new_r / dt;

            /* Differentiate quaternion:
             * dq/dt = (q[k] - q[k-1])/dt = 0.5 O(w[k-1]) q[k-1]
             * where O(w[k-1]) is an orthonormal quaternion mult matrix for [w1; w2; w3;
                  0] (i.e. quaternionMultiplication())
             * set q[k-1] = [0;0;0;1] by definition (from a refererence frame) and solve
                  for w[k-1] gives
             * w[k-1] = 2 [q1[k]; q2[k]; q3[k]] / dt
             */
            new_w = 2*new_q.head(3) / dt;
        } else {
            new_v = Vector3d::Zero();
            new_w = Vector3d::Zero();
        }
        new_a = Vector3d::Zero();

        new_x.block<3,1>(0,0) = new_r;
        new_x.block<3,1>(3,0) = new_v;
        new_x.block<3,1>(6,0) = new_a;
        new_x.block<3,1>(9,0) = new_w;
        newPose.rbd.setState(new_x, new_q);
        return newPose;

      }

    dynamicPose3d_NL adjustAttitude(dynamicPose3d_NL& prev) {
        Vector4d q, qprev;
        dynamicPose3d_NL newPose(prev.rbd.getIR(), prev.rbd.getSigmaV(), prev.rbd.
            getSigmaW());

        VectorXd x = rbd.x();
        q = rbd.qTotal();
        qprev = prev.qTotal();

        std::cout << "q: " << q.transpose() << std::endl;
        std::cout << "qprev: " << qprev.transpose() << std::endl;

        Matrix3d R = rbd.rotationMatrix(q);
        Matrix3d Rprev = rbd.rotationMatrix(qprev);
        Matrix3d Rdiff = R * Rprev.transpose();
        Vector4d new_qdiff = rbd.quaternionFromRot(Rdiff);

        std::cout << "R: " << R << std::endl;
        std::cout << "Rprev: " << Rprev << std::endl;
        std::cout << "Rdiff: " << Rdiff << std::endl;
        std::cout << "new_qdiff: " << new_qdiff.transpose() << std::endl;

        Vector4d qnew = rbd.quaternionMultiplication(new_qdiff, qprev);

        std::cout << "qnew aa: " << qnew.transpose() << std::endl << std::endl;
        if (isnan(qnew(1))) {
            std::cout << "qnew aa nan\n";
            new_qdiff = rbd.quaternionFromRot(Rdiff);
        }

        x.segment<3>(6) = Vector3d::Zero();
        rbd.setState(x, qnew);
        newPose.rbd.setState(x, qnew);
        return newPose;

    }

    void shortenQuaternion(dynamicPose3d_NL& prev) {
        Vector4d q, qprev, qnew;

        VectorXd x = rbd.x();
        q = rbd.qTotal();
        qprev = prev.qTotal();
        if(q.dot(qprev) < 0) {
            qnew = -q;
            x.segment<3>(6) = Vector3d::Zero();
            rbd.setState(x, qnew);
        }
    }


    dynamicPose3d_NL applyOdometry(dynamicPose3d_NL& prev) {
        dynamicPose3d_NL newPose(prev.rbd.getIR(), prev.rbd.getSigmaV(), prev.rbd.
            getSigmaW());
        VectorXd new_x(12);
        Vector3d new_r;
        Vector3d new_v;
```

```
      Vector3d new_a;
      Vector4d new_q;
      Vector3d new_w;

      VectorXd x, xprev;
      Vector4d q, qprev;
      Vector3d a, aprev;

      //get x's
      x = rbd.x();
      xprev = prev.x();

      //attitude gets
      q = rbd.qTotal();
      qprev = prev.qTotal();

      new_q = rbd.quaternionMultiplication(q,qprev);

      Matrix3d Rprev = rbd.rotationMatrix(qprev);
      new_r = Rprev * x.head(3) + xprev.head(3);

      new_v = Vector3d::Zero();
      new_a = Vector3d::Zero();
      new_w = Vector3d::Zero();

      new_x.block<3,1>(0,0) = new_r;
      new_x.block<3,1>(3,0) = new_v;
      new_x.block<3,1>(6,0) = new_a;
      new_x.block<3,1>(9,0) = new_w;

      newPose.rbd.setState(new_x, new_q);
      return newPose;
  }


  Matrix4d wTo() const {
    Matrix4d T;

    //error quaternion is applied
    Vector4d qtot = rbd.qTotal();
    VectorXd x = rbd.x();
    T.topLeftCorner(3,3) = rbd.rotationMatrix(qtot).transpose();
    T.col(3).head(3) << x.segment<3>(0);
    T.row(3) << 0., 0., 0., 1.;
    return T;
  }

  Matrix4d oTw() const {
      Matrix4d T;
      Matrix3d R;

      //error quaternion is applied
      Vector4d qtot = rbd.qTotal();
      VectorXd x = rbd.x();
      R = rbd.rotationMatrix(qtot);

      T.topLeftCorner(3,3) = R;
      T.col(3).head(3) << - R * x.segment<3>(0);
      T.row(3) << 0., 0., 0., 1.;
      return T;
    }


  Pose3d getPose3d() {
      return Pose3d(this->wTo());       //may be wrong: Mar 25, 2013, B.E.T.
      //return Pose3d(this->oTw());
  }

  Point3dh transform_to_inertial(const Point3dh& pBody) const{
      Vector3d p;
      p << pBody.x(), pBody.y(), pBody.z();
      Vector4d qtot = rbd.qTotal();
      VectorXd x = rbd.x();
      Vector3d T = x.head(3);
      Matrix3d Rt = rbd.rotationMatrix(qtot).transpose();

      Vector3d pInertial = Rt*p + T;

      return Point3dh(pInertial(0), pInertial(1), pInertial(2), 1.0);
  }

  Point3dh transform_to_body(const Point3dh& pInertial) const{
      Vector3d p;
      p << pInertial.x(), pInertial.y(), pInertial.z();
      Vector4d qtot = rbd.qTotal();
      VectorXd x = rbd.x();
      Vector3d T = x.head(3);
      Matrix3d R = rbd.rotationMatrix(qtot);

      Vector3d pBody = R*(p - T);

      return Point3dh(pBody(0), pBody(1), pBody(2), 1.0);
    }


  Noise getProcessNoise (double dt, inertiaRatios ir) {
      VectorXd x0 = VectorXd::Zero(90);
      x0.head(12) = rbd.x();
      rbd.setIR(ir);
      VectorXd newLambda = rbd.propagateRK4_adaptive(dt, x0).tail(78);

      Matrix<double,12,12> lambda = rbd.vec2symmMat(newLambda);
      Noise n = isam::Covariance(lambda);
      return n;
  }

  VectorXd vectorFull() const {
      VectorXd x = rbd.x();
      Vector4d q = rbd.qref();
      Vector3d mrp = rbd.quaternion2mrp(q);
      x(6) += mrp(0);
      x(7) += mrp(1);
      x(8) += mrp(2);
      return x;
  }

  VectorXd vector() const{
      return rbd.x();
    }

  void write(std::ostream &out) const {

      out << std::endl << "dP3d_NL x: " << rbd.x().transpose() << std::endl;
      out <<  "dP3d_NL qref: " <<  rbd.qref().transpose() << std::endl;
      out << std::endl;
  }


};
}
```

Listing B.23: Dynamic iSAM: camera3d.h

```
#pragma once

#include <Eigen/Dense>

#include <isam/Node.h>
#include <isam/Factor.h>
#include <isam/Pose3d.h>
#include <isam/Point3d.h>
#include "dynamicPose3d_NL.h"
#include <isam/slam_stereo.h>

namespace isam {

class cameraPose3d {
  frend std::ostream& operator<<(std::ostream& out, const cameraPose3d& p) {
    p.write(out);
    return out;
  }

  Point3d _t;
  Rot3d _rot;
public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  static const int dim = 3;
  static const char* name() {
    return "cameraPose3d";
  }

  cameraPose3d() : _t(0,0,0), _rot(0,0,0) {}

  cameraPose3d(double x, double y, double z, double yaw, double pitch, double roll) :
      _t(x, y, z), _rot(yaw, pitch, roll) {}

  cameraPose3d(const Eigen::MatrixXd& m) {
    if (m.rows()==6 && m.cols()==1) {
      _t = Point3d(m(0), m(1), m(2));
      _rot = Rot3d(m(3), m(4), m(5));
    } else if (m.rows()==4 && m.cols()==4) {
      // Convert a homogeneous 4x4 transformation matrix to a Pose3.
      Eigen::Matrix4d wTo = m / m(3,3); // enforce T(3,3)=1
      Eigen::Vector3d t = wTo.col(3).head(3);
      Eigen::Matrix3d wRo = wTo.topLeftCorner(3,3);
      _t = Point3d(t(0), t(1), t(2));
      _rot = Rot3d(wRo);
    } else {
      require(false, "Pose3d constructor called with matrix of wrong dimension");
    }
  }

  explicit cameraPose3d(const Eigen::Isometry3d & T) {
    Eigen::Vector3d t(T.translation());
    _t = Point3d(t(0), t(1), t(2));
    _rot = Rot3d(T.rotation());
  }

  cameraPose3d(const Point3d& t, const Rot3d& rot) : _t(t), _rot(rot) {}

  double x() const {return _t.x();}
  double y() const {return _t.y();}
  double z() const {return _t.z();}
  double yaw()   const {return _rot.yaw();}
  double pitch() const {return _rot.pitch();}
  double roll()  const {return _rot.roll();}

  Point3d trans() const {return _t;}
  Rot3d rot() const {return _rot;}

  void set_x(double x) {_t.set_x(x);}
  void set_y(double y) {_t.set_y(y);}
  void set_z(double z) {_t.set_z(z);}
  void set_yaw  (double yaw)   {_rot.set_yaw(yaw);}
  void set_pitch(double pitch) {_rot.set_pitch(pitch);}
  void set_roll (double roll)  {_rot.set_roll(roll);}

  cameraPose3d exmap(const Eigen::Vector3d& Δ) {
    cameraPose3d res = *this;
    res._t   = res._t.exmap(Δ.head(3));
//    res._rot = res._rot.exmap(Δ.tail(3));
    return res;
  }

  Eigen::Vector3d vector() const {
//    double Y, P, R;
    // cheaper to recover ypr at once
    //_rot.ypr(Y, P, R);
    Eigen::Vector3d tmp;
    tmp << x(), y(), z();//, Y, P, R;
    return tmp;
  }

  void set(double x, double y, double z, double yaw, double pitch, double roll) {
    _t = Point3d(x, y, z);
    _rot = Rot3d(yaw, pitch, roll);
  }

  void set(const Eigen::Vector3d& v) {
    _t = Point3d(v(0), v(1), v(2));
    //_rot = Rot3d(standardRad(v(3)), standardRad(v(4)), standardRad(v(5)));
  }

  void of_pose2d(const Pose2d& p) {
    set(p.x(), p.y(), 0., p.t(), 0., 0.);
  }

  void of_point2d(const Point2d& p) {
    set(p.x(), p.y(), 0., 0., 0., 0.);
  }

  void of_point3d(const Point3d& p) {
    set(p.x(), p.y(), p.z(), 0., 0., 0.);
  }

  void write(std::ostream &out) const {
    out << x() << ", " << y() << ", " << z() << "; "
        << yaw() << ", " << pitch() << ", " << roll();
  }

  /**
   * Convert Pose3 to homogeneous 4x4 transformation matrix.
   * The returned matrix is the object coordinate frame in the world
   * coordinate frame. In other words it transforms a point in the object
   * frame to the world frame.
   *
   * @return wTo
   */
  Eigen::Matrix4d wTo() const {
    Eigen::Matrix4d T;
    T.topLeftCorner(3,3) = _rot.wRo();
    T.col(3).head(3) << x(), y(), z();
    T.row(3) << 0., 0., 0., 1.;
    return T;
  }

  /**
   * Convert Pose3 to homogeneous 4x4 transformation matrix. Avoids inverting wTo.
   * The returned matrix is the world coordinate frame in the object
   * coordinate frame. In other words it transforms a point in the world
   * frame to the object frame.
   *
   * @return oTw
   */
  Eigen::Matrix4d oTw() const {
    Eigen::Matrix3d oRw = _rot.wRo().transpose();
    Eigen::Vector3d t(x(), y(), z());
    Eigen::Vector3d C = - oRw * t;
    Eigen::Matrix4d T;
    T.topLeftCorner(3,3) = oRw;
    T.col(3).head(3) = C;
    T.row(3) << 0., 0., 0., 1.;
    return T;
  }

  /**
   * Calculate new pose b composed from this pose (a) and the odometry d.
   * Follows notation of Lu&Milios 1997.
   * \f$ b = a \oplus d \f$
   * @param d Pose difference to add.
   * @return d transformed from being local in this frame (a) to the global frame.
   */
  Pose3d oplus(const Pose3d& d) const {
    return Pose3d(wTo() * d.wTo());
  }

  /**
   * Odometry d from b to this pose (a). Follows notation of
   * Lu&Milios 1997.
   * \f$ d = a \ominus b \f$
   * @param b Base frame.
   * @return Global this (a) expressed in base frame b.
   */
  Pose3d ominus(const Pose3d& b) const {
    return Pose3d(b.oTw() * wTo());
  }

  /**
   * Project point into this coordinate frame.
   * @param p Point to project
   * @return Point p locally expressed in this frame.
   */
  Point3dh transform_to(const Point3dh& p) const {
    return Point3dh(oTw() * p.vector());
  }


  /**
   * Project point into this coordinate frame.
   * @param p Point to project
   * @return Point p locally expressed in this frame.
   */
  Point3d transform_to(const Point3d& p) const {
    return transform_to(Point3dh(p)).to_point3d();
  }

  /**
   * Project point from this coordinate frame.
   * @param p Point to project
   * @return Point p is expressed in the global frame.
   */
  Point3dh transform_from(const Point3dh& p) const {
    return Point3dh(wTo() * p.vector());
  }

  /**
   * Project point from this coordinate frame.
   * @param p Point to project
   * @return Point p is expressed in the global frame.
   */
  Point3d transform_from(const Point3d& p) const {
    return transform_from(Point3dh(p)).to_point3d();
  }

  Pose3d getPose3d() {
      Pose3d val(this->x(), this->y(), this->z(), this->yaw(), this->pitch(), this->
          roll());
      return val;
  }

};

typedef NodeT<cameraPose3d> cameraPose3d_Node;

class cameraPose_Factor : public FactorT<cameraPose3d> {
  cameraPose3d_Node* _pose;
```

```

public:

  cameraPose_Factor(cameraPose3d_Node* pose, const cameraPose3d& prior, const Noise&
      noise)
    : FactorT<cameraPose3d>("CameraPose3d_Factor", 3, noise, prior), _pose(pose) {
    _nodes.resize(1);
    _nodes[0] = pose;
  }

  void initialize() {
    if (!_pose->initialized()) {
      cameraPose3d predict = _measure;
      _pose->init(predict);
    }
  }

  Eigen::VectorXd basic_error(Selector s = ESTIMATE) const {
    Eigen::VectorXd err = _nodes[0]->vector(s).head(3) - _measure.vector().head(3);
    return err;
  }

};

}
```

Listing B.24: Dynamic iSAM: inertiaRatios.h

```
#pragma once

#include <cmath>
#include <Eigen/Dense>
#include <Eigen/Geometry>
#include <isam/Node.h>
#include <isam/Factor.h>
#include "NodeExmap.h"

namespace isam {

class inertiaRatios {
    frend std::ostream& operator<<(std::ostream& out, const inertiaRatios& p) {
        p.write(out);
        return out;
    }

    /*
     * k1 = ln(J11 / J22)
     * k2 = ln(J22 / J33)
     */
    double _k1;
    double _k2;

public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    static const int dim = 2;
    static const char* name() {
        return "inertiaRatios";
    }

    inertiaRatios()  {
        _k1 = 0;
        _k2 = 0;
    }


    inertiaRatios(const double& k1, const double& k2) {
        _k1 = k1;
        _k2 = k2;
    }

    Eigen::Matrix3d getJ() const {
        Eigen::Matrix3d J = Eigen::Matrix3d::Zero();
        double Jscale = 1.0; //0.0116;
        J(0,0) = exp(_k1);
        J(1,1) = 1.0;
        J(2,2) = exp(-_k2);

        J *= Jscale;

        return J;
    }

    Eigen::VectorXd x() const{
        Eigen::Vector2d x;
        x(0) = _k1;
        x(1) = _k2;
        return x;
    }

    void setState(Eigen::VectorXd x) {
        _k1 = x(0);
        _k2 = x(1);
    }

    inertiaRatios exmap(const Eigen::Vector2d& Δ) {
        inertiaRatios res = *this;
        res._k1 += Δ(0);
        res._k2 += Δ(1);
        return res;
    }

    inertiaRatios exmap_reset(const Eigen::Vector2d& Δ) {
        inertiaRatios res = *this;
        res._k1 += Δ(0);
        res._k2 += Δ(1);
        return res;
    }

    Eigen::VectorXd vector() const {
        Eigen::Vector2d tmp;
        tmp << _k1, _k2;
        return tmp;
    }

    void set(const Eigen::Vector2d& v) {
        _k1 = v(0);
        _k2 = v(1);
    }

    void write(std::ostream &out) const {
        Eigen::Matrix3d Jcurr = getJ();
        out << std::endl << "inertaRatios x: " << x().transpose() << std::endl <<
            Jcurr(0,0) << " , " << Jcurr(1,1) << " , " << Jcurr(2,2) << std::endl;
    }

};

typedef NodeExmapT<inertiaRatios> inertiaRatios_Node;

/**
 * Prior on inertiaRatios.
 */
class inertiaRatios_Factor : public FactorT<inertiaRatios> {
public:
    inertiaRatios_Node* _ir_node;

    inertiaRatios_Factor(inertiaRatios_Node* ir_node, const inertiaRatios& prior,
        const Noise& noise)
    : FactorT<inertiaRatios>("inertiaRatios_Factor", 2, noise, prior), _ir_node(
        ir_node) {
        _nodes.resize(1);
        _nodes[0] = ir_node;
    }

    void initialize() {
        if (!_ir_node->initialized()) {
            inertiaRatios predict = _measure;
            _ir_node->init(predict);
        }
    }

    Eigen::VectorXd basic_error(Selector s = ESTIMATE) const {
        inertiaRatios ir = _ir_node->value(s);
        Eigen::VectorXd err = ir.vector() - _measure.vector();
        return err;
    }
};
}
```

Listing B.25: Dynamic iSAM: inertialParams.h

```
#pragma once

#include <cmath>
#include <ostream>
#include <iostream>

#include <isam/util.h>
#include "math.h"
#include <Eigen/Dense>

using namespace isam;
using namespace Eigen;

class principalAxesFrame {
  frend std::ostream& operator<<(std::ostream& out, const principalAxesFrame& p) {
    p.write(out);
    return out;
 }

  Vector3d _r;      //position - from the target frame to the principal frame
  Vector4d _q;      //quaternion - from the target frame to the principal frame

  //3 parameter attitude error
  Vector3d _a;      //Modified Rodrigues Parameter

public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
  static const int dim = 6;
  static const char* name() {
    return "principalAxesFrame";
  }
  Matrix<double, 6, 6> _sqrtinf;

  principalAxesFrame() {
    _r << 0.0, 0.0, 0.0;
    _a << 0.0, 0.0, 0.0;
    _q << 0.0, 0.0, 0.0, 1.0;
  }

  principalAxesFrame(Matrix<double,3,1> r) {
     _r = r;
  }

  principalAxesFrame(Matrix<double,3,1> r, Vector4d q) {
      _r = r;
      _q = q;
  }

  VectorXd vector() const{
      Matrix<double, 6, 1> x;
      x << _r, _a;
      return x;
    }

  void set(const VectorXd& v) {
      _r = v.block<3,1>(0,0);
      _a = v.block<3,1>(3,0);
  }


  Matrix<double,6,1> x() {
      Matrix<double,6,1> x;
      x << _r, _a;
      return x;
  }

  Vector4d q() {
      return _q;
  }

  Vector4d mrp2quaternion(Vector3d mrp) const {
      Vector4d dq;
      dq << 8*mrp / (16 + mrp.squaredNorm()), (16 - mrp.squaredNorm()) / (16+mrp.
          squaredNorm());
      return dq;
  }

  Vector4d addQuaternionError(Vector3d mrp, Vector4d qref) const {
      Vector4d qnew, dq;
      dq = mrp2quaternion(mrp);

      qnew = quaternionMultiplication(dq, qref);
      return qnew;
  }


  principalAxesFrame exmap(const Matrix<double,6,1>& Δ) const {
      principalAxesFrame res = *this;
      res._r += Δ.block<3,1>(0,0);
      res._a += Δ.block<3,1>(3,0);
      return res;
    }

  principalAxesFrame exmap_reset(const Matrix<double,6,1>& Δ)  {
      principalAxesFrame res = *this;

    res._r += Δ.block<3,1>(0,0);
    res._a += Δ.block<3,1>(3,0);

    res.write();

    //reset step
    res._q = addQuaternionError(res._a, res._q);
    res._a = Vector3d::Zero();

    printf("inertial reset\n");

    return res;
  }


  void write(std::ostream &out = std::cout) const {
        out << " " << _r.transpose();
        out << " " << _q(0) << " " << _q(1) << " " << _q(2) << " " << _q(3);
        out << " " << _a.transpose();
        out << std::endl;
  }

  Vector4d quaternionMultiplication(Vector4d q1, Vector4d q2) const {
      //q1 \mult q2
      Matrix4d qm;
      Vector4d result;
      qm <<     q1(3),  q1(2),  -q1(1), q1(0),
                -q1(2), q1(3),  q1(0),  q1(1),
                q1(1),  -q1(0), q1(3),  q1(2),
                -q1(0), -q1(1), -q1(2), q1(3);

      result = qm*q2;
      result /= result.norm();

      return result;
  }

  Matrix3d rotationMatrix(Vector4d q) const {
      Matrix3d rot;

      rot(0,0) = q(0)*q(0)-q(1)*q(1)-q(2)*q(2)+q(3)*q(3);
      rot(0,1) = 2*(q(0)*q(1)+q(2)*q(3));
      rot(0,2) = 2*(q(0)*q(2)-q(1)*q(3));

      rot(1,0) = 2*(q(0)*q(1)-q(2)*q(3));
      rot(1,1) = -q(0)*q(0)+q(1)*q(1)-q(2)*q(2)+q(3)*q(3);
      rot(1,2) = 2*(q(2)*q(1)+q(0)*q(3));

      rot(2,0) = 2*(q(0)*q(2)+q(1)*q(3));
      rot(2,1) = 2*(q(2)*q(1)-q(0)*q(3));
      rot(2,2) = -q(0)*q(0)-q(1)*q(1)+q(2)*q(2)+q(3)*q(3);

//      std::cout << "q2rot: " << q << rot << std::endl;
      return rot;
  }

  Point3d toPrincipalFrame(const Point3d& p_m) const {
      Matrix3d R = rotationMatrix(addQuaternionError(_a,_q));
      Vector3d vecBody =  R * (p_m.vector() - _r);
      Point3d p_c(vecBody);

      return p_c;
  }

  Point3d fromPrincipalFrame(const Point3d& p_m) const {
      Matrix3d R = rotationMatrix(addQuaternionError(_a,_q));
      Vector3d vecBody = R.transpose() * p_m.vector() + _r;
      Point3d p_c(vecBody);

      return p_c;
  }

};
```

Listing B.26: Dynamic iSAM: DenseVis.h

```
/*
 * DenseVis.h
 *
 *  Created on: May 5, 2013
 *      Author: tweddle
 */

#ifndef DENSEVIS_H_
#define DENSEVIS_H_

#include <fstream>
#include <vector>
#include <string>
#include <iostream>
#include <sstream>
#include <exception>

// Eigen
#include <Eigen/Core>
#include <Eigen/Geometry>
#include <Eigen/StdVector>

// OpenCV
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/highgui/highgui.hpp"

// libelas
#include "elas.h"

#include "Triangulator.h"
#include "DenseStereo.h"
//#include "Frame.h"

#include "LCMPublisher.h"


#include <pcl/point_types.h>
#include <pcl/io/ply_io.h>
#include <pcl/io/pcd_io.h>
#include <pcl/kdtree/kdtree_flann.h>
#include <pcl/features/normal_3d.h>
#include <pcl/surface/gp3.h>
```

```
#include <pcl/io/vtk_io.h>
#include <pcl/io/vtk_lib_io.h>
#include <pcl/filters/voxel_grid.h>
#include <pcl/filters/statistical_outlier_removal.h>
#include <pcl/visualization/pcl_visualizer.h>
#include <pcl/common/transforms.h>
#include <pcl/visualization/image_viewer.h>

class DenseVis {
    cv::Mat elasDisp, nonthresholded_img;
    DenseStereo* denseStereo;
    LCMPublisher* lcmpub;

     std::vector<Eigen::Vector3d, Eigen::aligned_allocator<Eigen::Vector3d> >
         princAxisPoints;
     std::vector<int> princAxisColors;

     pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud, totalCloud;
     pcl::PLYWriter plyWriter;

public:

        pcl::PolygonMeshPtr smallTriangles;
        pcl::PolygonMeshPtr totalTriangles;

    DenseVis(Triangulator* triangulator, LCMPublisher* _lcmpub);

    void computeDensePoints(isam::cameraPose3d_Node* cam, isam::dynamicPose3d_NL_Node*
         pose, cv::Mat& leftImage, cv::Mat& rightImage);

    void buildDenseMap(isam::cameraPose3d_Node* cam, isam::dynamicPose3d_NL_Node*
        princAxis, std::vector<isam::dynamicPose3d_NL_Node*>& poselist, std::vector<cv
        ::Mat>& leftImageList, std::vector<cv::Mat>& rightImageList);
    void buildDenseCloud(isam::cameraPose3d_Node* cam, /*isam::dynamicPose3d_NL_Node*
        princAxis,*/std::vector<isam::dynamicPose3d_NL_Node*>& poselist, std::vector<
        cv::Mat>& leftImageList, std::vector<cv::Mat>& rightImageList);
    void updatePrincipalAxis(isam::dynamicPose3d_NL_Node* princAxis, int listsize);
    void generateMesh(pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud, pcl::
        PolygonMeshPtr triangles, std::string filename, int maxNN = 400);
    pcl::PointCloud<pcl::PointXYZRGB>::Ptr downsampleCloud(pcl::PointCloud<pcl::
        PointXYZRGB>::Ptr cloud, float dimension);
    void visualizeMesh(pcl::PolygonMeshPtr mesh, std::vector<isam::
        dynamicPose3d_NL_Node*> pose_list, std::vector<cv::Mat>& leftImageList);

    void clear();

};

#endif /* DENSEVIS_H_ */
```

Listing B.27: Dynamic iSAM: DenseVis.cpp

```
#include "DenseVis.h"

DenseVis::DenseVis(Triangulator* triangulator, LCMPublisher* _lcmpub) {
    denseStereo = new DenseStereo(*triangulator, false);
    lcmpub = _lcmpub;

    cloud = pcl::PointCloud<pcl::PointXYZRGB>::Ptr(new pcl::PointCloud<pcl::
        PointXYZRGB>);
    totalCloud = pcl::PointCloud<pcl::PointXYZRGB>::Ptr(new pcl::PointCloud<pcl::
        PointXYZRGB>);

    smallTriangles = pcl::PolygonMeshPtr(new pcl::PolygonMesh);
    totalTriangles = pcl::PolygonMeshPtr(new pcl::PolygonMesh);


}

void DenseVis::clear() {
     princAxisPoints.clear();
     princAxisColors.clear();
}

void DenseVis::buildDenseMap(isam::cameraPose3d_Node* cam, isam::dynamicPose3d_NL_Node
    * princAxis,std::vector<isam::dynamicPose3d_NL_Node*>& poselist, std::vector<cv::
    Mat>& leftImageList, std::vector<cv::Mat>& rightImageList) {

    int size = poselist.size();
    lcmpub->clearBody3DPoints();
    lcmpub->addPrincipalAxis(princAxis, poselist.size());
    for (int i = 0; i < size ; i++) {
        std::cout << "Dense Map Iteration: " << i << std::endl;
        this->clear();
        computeDensePoints(cam, poselist[i], leftImageList[i], rightImageList[i]);

        lcmpub->addBody3DPoints(i, princAxisPoints, princAxisColors);
        usleep(100000);
    }
}

void DenseVis::updatePrincipalAxis(isam::dynamicPose3d_NL_Node* princAxis, int
    listsize) {
    lcmpub->addPrincipalAxis(princAxis, listsize);
}

```

```
void DenseVis::computeDensePoints(isam::cameraPose3d_Node* cam, isam::
    dynamicPose3d_NL_Node* pose, cv::Mat& leftImage, cv::Mat& rightImage) {
     double lx, ly, lz;
     isam::Point3dh X;
     isam::Point3dh inertX;
     isam::Point3dh bodyX;
     Eigen::Vector3d bodyVec;
     cv::equalizeHist(leftImage, leftImage);
     cv::equalizeHist(rightImage, rightImage);

     denseStereo->clear();

     elasDisp = denseStereo->calculate(leftImage, rightImage);
     nonthresholded_img = denseStereo->getPreThreshDisp();

     for (unsigned int i = 0; i < denseStereo->points.size(); i++) {
         lx = denseStereo->points[i](0);
         ly = denseStereo->points[i](1);
         lz = denseStereo->points[i](2);

         X.set(lz, -lx, -ly, 1.0);
         inertX = cam->value().transform_from(X);
         bodyX = pose->value().transform_to_body(inertX);
         princAxisPoints.push_back(Eigen::Vector3d(bodyX.x(), bodyX.y(), bodyX.z()));
         princAxisColors.push_back(denseStereo->colors[i]);
     }

}

void DenseVis::buildDenseCloud(isam::cameraPose3d_Node* cam,std::vector<isam::
    dynamicPose3d_NL_Node*>& poselist, std::vector<cv::Mat>& leftImageList, std::
    vector<cv::Mat>& rightImageList) {
    std::stringstream filename, filename2, filename3, filename4, filename5, filename6;
    denseStereo->clear();
    std::cout << "Poselist, left, right: " << poselist.size() << "," << leftImageList.
        size() << "," << rightImageList.size() << std::endl;

    for (int j = 0; j < poselist.size() ; j++) {
        filename.str(std::string());
        filename2.str(std::string());
        filename3.str(std::string());
        filename4.str(std::string());
        filename5.str(std::string());
        filename6.str(std::string());
        this->clear();
```

```cpp
        computeDensePoints(cam, poselist[j], leftImageList[j+1], rightImageList[j+1]);

        filename4 << "/home/tweddle/Desktop/disparity/mergeImg" << j << ".bmp";
        filename5 << "/home/tweddle/Desktop/disparity/elasDisp" << j << ".bmp";

        cv::Mat mergeImg;
        std::vector<cv::Mat> channels;
        channels.push_back(leftImageList[j+1]);
        channels.push_back(leftImageList[j+1]);
        channels.push_back(elasDisp);
        cv::merge(channels, mergeImg);


        cv::imwrite(filename4.str(), mergeImg);
        cv::imwrite(filename5.str(), elasDisp);

        std::cout << "PrincAxisPoints: " << princAxisPoints.size() << std::endl;

        cloud->points.clear();
        cloud->points.resize(princAxisPoints.size());
        cloud->width = cloud->size();
        cloud->height = 1;

        //totalCloud->points.resize(totalCloud->points.size() + princAxisPoints.size());

        for (unsigned int i = 0; i < princAxisPoints.size(); i++) {
            cloud->points[i].x = princAxisPoints[i](0);
            cloud->points[i].y = princAxisPoints[i](1);
            cloud->points[i].z = princAxisPoints[i](2);

            uint8_t r = princAxisColors[i];
            uint8_t g = princAxisColors[i];
            uint8_t b = princAxisColors[i];    // Example: Red color
            uint32_t rgb = ((uint32_t)r << 16 | (uint32_t)g << 8 | (uint32_t)b);
            cloud->points[i].rgb = *reinterpret_cast<float*>(&rgb);

            pcl::PointXYZRGB newPoint;
            newPoint.x = princAxisPoints[i](0);
            newPoint.y = princAxisPoints[i](1);
            newPoint.z = princAxisPoints[i](2);
            newPoint.rgb = *reinterpret_cast<float*>(&rgb);


            totalCloud->points.push_back(newPoint);
            totalCloud->width = totalCloud->points.size();
            totalCloud->height = 1;

        }
        filename << "/home/tweddle/Desktop/plyfolder/plyfile" << j << ".ply";
        filename2 << "/home/tweddle/Desktop/pcdfolder/pcdfile" << j << ".pcd";
        plyWriter.write(filename.str(),*(cloud.get()));

        std::cout << "Width/Height: " << cloud->width << "," << cloud->height << ","
            << cloud->size() << "," <<  princAxisPoints.size() << std::endl;
        std::cout << "TotalCloud Width/Height: " << totalCloud->width << "," <<
            totalCloud->height << "," << totalCloud->size() << std::endl;

        pcl::io::savePCDFile(filename2.str(), *(cloud.get()));

        filename3 << "/home/tweddle/Desktop/meshfolder/mesh" << j << ".vtk";
        std::cout << "filename3: " << filename3.str() << std::endl;



    }

    std::cout << "totalCloud->points.size(): " << totalCloud->points.size() << std::
        endl;

    pcl::PointCloud<pcl::PointXYZRGB>::Ptr totalSampleSmall = downsampleCloud(
        totalCloud, 0.001);
    filename6 << "/home/tweddle/Desktop/meshfolder/totalmesh.vtk";
    generateMesh(totalSampleSmall, totalTriangles, filename6.str(), 500);

    pcl::io::savePolygonFileSTL("/home/tweddle/Desktop/meshfolder/totalMesh.stl", *
        totalTriangles);
    pcl::io::savePLYFile("/home/tweddle/Desktop/meshfolder/totalMesh.ply", *
        totalTriangles,6);



}

pcl::PointCloud<pcl::PointXYZRGB>::Ptr DenseVis::downsampleCloud(pcl::PointCloud<pcl::
    PointXYZRGB>::Ptr inputCloud, float dimension) {
    std::cout << "Downsampling cloud - initial size: " << inputCloud->points.size() <<
         std::endl;

    pcl::PointCloud<pcl::PointXYZRGB>::Ptr outputCloud(new pcl::PointCloud<pcl::
        PointXYZRGB>);
    pcl::PointCloud<pcl::PointXYZRGB>::Ptr returnCloud(new pcl::PointCloud<pcl::
        PointXYZRGB>);

    pcl::StatisticalOutlierRemoval<pcl::PointXYZRGB> sor;
    sor.setInputCloud(inputCloud);
    sor.setMeanK(50);
    sor.setStddevMulThresh(1.0);
    sor.filter(*outputCloud);

      std::cout << "SOR completed - final size: " << outputCloud->points.size() << std
          ::endl;

    // Create the filtering object
      pcl::VoxelGrid<pcl::PointXYZRGB> vgrid;
      vgrid.setInputCloud(outputCloud);
      vgrid.setLeafSize(dimension, dimension, dimension);
      vgrid.filter(*returnCloud);

      std::cout << "Downsample completed - final size: " << returnCloud->points.size()
           << std::endl;

      return returnCloud;
}

void DenseVis::generateMesh(pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud, pcl::
    PolygonMesh::Ptr triangles, std::string filename, int maxNN) {

    // Normal estimation*
      pcl::NormalEstimation<pcl::PointXYZRGB, pcl::Normal> n;
      pcl::PointCloud<pcl::Normal>::Ptr normals (new pcl::PointCloud<pcl::Normal>);
      pcl::search::KdTree<pcl::PointXYZRGB>::Ptr tree (new pcl::search::KdTree<pcl::
          PointXYZRGB>);
      tree->setInputCloud (cloud);
      n.setInputCloud (cloud);
      n.setSearchMethod (tree);
      n.setKSearch (20);
      n.compute (*normals);
      //* normals should not contain the point normals + surface curvatures

      // Concatenate the XYZ and normal fields*
      pcl::PointCloud<pcl::PointXYZRGBNormal>::Ptr cloud_with_normals (new pcl::
          PointCloud<pcl::PointXYZRGBNormal>);
      pcl::concatenateFields (*cloud, *normals, *cloud_with_normals);

      std::cout << "Point A\n";

```

```
        // Create search tree*
        pcl::search::KdTree<pcl::PointXYZRGBNormal>::Ptr tree2 (new pcl::search::KdTree<
            pcl::PointXYZRGBNormal>);
        tree2->setInputCloud (cloud_with_normals);

        std::cout << "Point B\n";

        // Initialize objects
        pcl::GreedyProjectionTriangulation<pcl::PointXYZRGBNormal> gp3;

        std::cout << "Point C\n";

        // Set typical values for the parameters
        gp3.setMu (2.5);
        gp3.setSearchRadius(0.05);
        gp3.setMaximumNearestNeighbors (maxNN);
        gp3.setMaximumSurfaceAngle(M_PI/4); // 45 degrees
        gp3.setMinimumAngle(M_PI/18); // 10 degrees
        gp3.setMaximumAngle(2*M_PI/3); // 120 degrees
        gp3.setNormalConsistency(true);


        // Get result
        gp3.setInputCloud (cloud_with_normals);
        gp3.setSearchMethod (tree2);
        std::cout << "Point D: cloudnormals: " << cloud_with_normals->size() << std::
            endl;
        std::cout << "num of triangles1: " << triangles->polygons.size() << std::endl;
        std::cout << "search radius: " << gp3.getSearchRadius() << std::endl;;

        gp3.reconstruct(*triangles);

        std::cout << "Point E\n";

        // Additional vertex information
        std::vector<int> parts = gp3.getPartIDs();
        std::vector<int> states = gp3.getPointStates();

        std::cout << "num of triangles: " << triangles->polygons.size() << std::endl;
        std::cout << "write filename: " << filename << std::endl;
        pcl::io::saveVTKFile(filename, *triangles);

}

void DenseVis::visualizeMesh(pcl::PolygonMeshPtr mesh, std::vector<isam::
      dynamicPose3d_NL_Node*> pose_list, std::vector<cv::Mat>& leftImageList) {
      Eigen::Vector3f offset;
      Eigen::Vector4f offset4;
      Eigen::Quaternionf quat;
      Eigen::Vector4d currq;
      Eigen::Vector3d currp;
      std::stringstream visFilename;
      std::stringstream leftFilename;

      pcl::visualization::PCLVisualizer vis("Mesh Viewer");
      //pcl::visualization::ImageViewer iv("Left Camera Viewer");

      cv::Mat combined_img;
      //place two images side by side
      combined_img.create( cv::Size(2*640,480), CV_MAKETYPE(leftImageList[0].depth(), 3)
           );
      cv::Mat imgLeft = combined_img( cv::Rect(0, 0, leftImageList[0].cols,
          leftImageList[0].rows));
      cv::Mat imgRight = combined_img( cv::Rect(leftImageList[0].cols, 0, leftImageList
          [0].cols, leftImageList[0].rows) );


      std::cout << "Visualizing Mesh" << std::endl;

      offset << -0.35, 0, 0;
      quat.x() = 0.0;
      quat.y() = 0.0;
      quat.z() = 0.0;
      quat.w() = 1.0;

      Eigen::Affine3f affineTransform = Eigen::Translation3f(offset) * Eigen::AngleAxisf
          (quat);

      vis.setBackgroundColor(0.1, 0.01,1.0);
      vis.addPolygonMesh(*mesh);
      vis.initCameraParameters();
      vis.setCameraPosition(-0.5,0,0, 0,0,1);
      vis.spinOnce(100);


      pcl::PointCloud<pcl::PointXYZRGBNormal>::Ptr meshcloud(new pcl::PointCloud<pcl::
          PointXYZRGBNormal>);
      pcl::PointCloud<pcl::PointXYZRGBNormal>::Ptr temp2(new pcl::PointCloud<pcl::
          PointXYZRGBNormal>);
      pcl::fromROSMsg(mesh->cloud, *meshcloud);


      std::cout << "About to spin" << std::endl;

      for (unsigned int i = 0; i < pose_list.size(); i++) {
          if (vis.wasStopped()) {
              break;
          }

          currp = pose_list[i]->value().x().head(3);
          currq = pose_list[i]->value().qTotal();
          offset(0) = currp(0);
          offset(1) = currp(1);
          offset(2) = currp(2);
          offset4.head(3) = offset;
          offset4(3) = 0.0;
          quat.x() = currq(0);
          quat.y() = currq(1);
          quat.z() = currq(2);
          quat.w() = currq(3);

          affineTransform = Eigen::Translation3f(offset) * Eigen::AngleAxisf(quat);

          std::cout << "Counter: " << i << std::endl;

          pcl::transformPointCloudWithNormals(*meshcloud, *temp2, offset, quat);
          vis.updatePolygonMesh<pcl::PointXYZRGBNormal>(temp2,mesh->polygons);
          visFilename.str(std::string());
          visFilename << "/home/tweddle/Desktop/visfolder/visfile" << i << ".png";
          vis.saveScreenshot(visFilename.str());

          cv::Mat tempVis = cv::imread(visFilename.str());
          cv::resize(tempVis, imgRight, cv::Size(640,480));
          cvtColor( leftImageList[i+1], imgLeft, CV_GRAY2BGR );

          leftFilename.str(std::string());
          leftFilename << "/home/tweddle/Desktop/visfolder/combinedImg" << i << ".png";
          cv::imwrite(leftFilename.str(),combined_img);


          //500 ms with 1ms draw every 50 ms
          for (int j = 0; j < 10; j++) {
              vis.spinOnce(100);
              usleep(50000);
          }

      }

      //5000 ms with 1ms draw every 50 ms
      while(!vis.wasStopped()) {
          vis.spinOnce(100);
          usleep(100000);
      }

      std::cout << "Done with spin" << std::endl;
  }
```

Bibliography

[1] *A New Extension of the Kalman Filter to Nonlinear Systems*, 1997.

[2] Loopy SAM. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2191–2196, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[3] Efficient Large-Scale Stereo Matching. In *Asian Conference on Computer Vision (ACCV)*, Queenstown, New Zealand, November 2010.

[4] *Iceberg-Relative Navigation for Autonomous Underwater Vehicles.* PhD thesis, Stanford University, Stanford, CA, 08 2011.

[5] 3D-CoForm. Meshlab, 2013. [Online; accessed 7-June-2013].

[6] F. Aghili, M. Kuryllo, G. Okouneva, and C. English. Robust vision-based pose estimation of moving objects for Automated Rendezvous & Docking. In *Mechatronics and Automation (ICMA), 2010 International Conference on*, pages 305–311, 2010.

[7] Farhad Aghili. A Prediction and Motion-Planning Scheme for Visually Guided Robotic Capturing of Free-Floating Tumbling Objects With Uncertain Dynamics. *IEEE Transactions on Robotics*, 28(3):634–649.

[8] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-D point sets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 9(5):698–700, 1987.

[9] Encyclopedia Astronautica. Hs 376, 2013. [Online; accessed 22-May-2013].

[10] S. Augenstein. Monocular Pose and Shape Estimations of Moving Targets, for Autonomous Rendezvous and Docking.

[11] S. Augenstein and S M Rock. Improved frame-to-frame pose tracking during vision-only SLAM/SFM with a tumbling target. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3131–3138, 2011.

[12] David Barnhart, Roger Hunter, Alan Weston, Vincent Chioma, Mark Steiner, and William Larsen. XSS-10 Micro-Satellite Demonstration. In *AIAA Defense and Civil Space Programs Conference and Exhibit*, August 1998.

[13] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, 2008.

[14] David Bayard and Paul Brugarolas. On-Board Vision-Based Spacecraft Estimation Algorithm for Small Body Exploration. *IEEE Transactions on Aerospace and Electronic Systems*, 44:243–260, January 2008.

[15] P J Besl and N D McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern and Machine Intelligence*, 14(2), 1992.

[16] Christopher M Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[17] Boeing. Boeing 376 Fleet, 2013. [Online; accessed 9-June-2013].

[18] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, Cambridge, MA, 2008.

[19] D.C. Brown. A solution to the general problem of multiple station analytical stereo triangulation. Technical report, Patrick Airforce Base, Florida, 1958.

[20] D.C. Brown. Decentering Distortion of Lenses. 32(3):444–462, 1966.

[21] D.C. Brown. The bundle adjustment—progress and prospects. *Int. Archives Photogrammetry*, 21(3), 1976.

[22] Robert G Brown and Patrick Y C Hwang. *Introduction to Random Signals and Applied Kalman Filtering*. Wiley, 3rd edition, 1996.

[23] Caley Burke. *Nutation in the Spinning SPHERES Spacecraft and Fluid Slosh*. PhD thesis, Massachusetts Institute of Technology, MIT, May 2010.

[24] Javier Civera, Andrew J Davison, and J M M Montiel. Inverse Depth Parametrization for Monocular SLAM. *IEEE Transactions on Robotics*, 24(5):932–945, 2008.

[25] Desireé Cotto-Figueroa. The Rotation Rate Distribution of Small Near-Earth Asteroids. Master's thesis, Ohio University, January 2008.

[26] John L Crassidis and John L Junkins. *Optimal Estimation of Dynamic Systems*. Optimal Estimation of Dynamic Systems. Chapman and Hall, 2004.

[27] N Glenn Creamer, Ralph Hartley, C Glen Henshaw, Stephen Roderick, Jamison Hope, and Jerome Obermark. Autonomous Release of a Snagged Solar Array: Technologies and Laboratory Demonstration. In *AIAA Guidance, Navigation and Control Conference*, Reston, Virigina, September 2012. American Institute of Aeronautics and Astronautics.

[28] B Cyganek and J P Siebert. *An introduction to 3D computer vision techniques and algorithms*. Wiley, 2009.

[29] D. Koller and N Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Probabilistic Graphical Models: Principles and Techniques. MIT Press, 2009.

[30] Gregory John Eslinger. Dynamic Programming Applied to Electromagnetic Satellite Actuation. Master's thesis, Massachusetts Institute of Technology.

[31] Wigbert Fehse. *Automated Rendezvous and Docking of Spacecraft*. Cambridge University Press, 2003.

[32] M A Fischler and R C Bolles. Random Sample Consensus: a Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the Association of Computing Machinery*, 24(6):381–395, 1981.

[33] Dehann Fourie, Brent E Tweddle, and Steve Ulrich. Relative Vision-Based Navigation and Control for Spacecraft Inspection. In *AIAA Guidance, Navigation and Control Conference 2013*, August 2013.

[34] Gene F Franklin, J David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. Pearson Prentice Hall, Reading, MA, 2006.

[35] Friedrich Fraundorfer and Davide Scaramuzza. Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications. *IEEE Robotics & Automation Magazine*, 19(2):78–90.

[36] Daniele Gallardo, Riccardo Bevilacqua, and Richard E. Rasmussen. Advances on a 6 Degrees of Freedom Testbed for Autonomous Satellites Operations. In *AIAA Guidance, Navigation and Control Conference*, pages 1–17, August 2011.

[37] Arthur Gelb, editor. *Applied Optimal Estimation*. Applied Optimal Estimation. The MIT Press, 1974.

[38] Jerry Ginsberg. *Advanced Engineering Dynamics*. Advanced Engineering Dynamics. Cambridge University Press, 1998.

[39] E W Griffith and K S P Kumar. On the Observability of Nonlinear Systems: I. *Journal of Mathematical Analysis and Applications*, 35(1):135–147, 1971.

[40] G Grisetti, R Kümmerle, C. Stachniss, and W. Burgard. A Tutorial on Graph-Based SLAM. *Intelligent Transportation Systems Magazine, IEEE*, 2(4):31–43, 2010.

[41] R I Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[42] Javier Montiel Hauke Strasdat and Andrew Davison. Real-Time Monocular SLAM: Why Filter? In *International Conference on Robotics and Automation*, 2010.

[43] Carl Henshaw, Liam Healy, and Stephen Roderick. LIIVe: A Small, Low-Cost Autonomous Inspection Vehicle. In *AIAA SPACE Conference and Exposition*, 2009.

[44] R. Hermann and A. Krener. Nonlinear Controllability and Observability. *Automatic Control, IEEE Transactions on*, 22(5):728–740, October 1977.

[45] Ulrich Hillenbrand and Roberto Lampariello. Motion and Parameter Estimation of a Free-Floating Space Object from Range Data for Motion Prediction. In *International Symposium on Artificial Intelligence, Robotics and Automationin Space (i-SAIRAS)*, September 2005.

[46] B K P Horn. Closed-Form Solution of Absolute Orientation Using Unit Quaternions. *Journal of the Optical Society of America*, 4(4):629–642, 1987.

[47] Berthold K. P. Horn. Relative orientation revisited. *Journal of the Optical Society of America A*, 8:1630–1638, 1991.

[48] P D Hough and S A Vavasis. Complete Orthogonal Decomposition for Weighted Least Squares. *SIAM J. Matrix Analysis Appl*, 1997.

[49] R.T. Howard, A.F. Heaton, R.M. Pinson, and C.K. Carrington. Orbital Express Advanced Video Guidance Sensor. In *Institue for Electrical and Electronics Engineers Aerospace Conference*, pages 1–10, March 2008.

[50] Albert S. Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera. In *Int. Symposium on Robotics Research (ISRR)*, pages 1–16, Flagstaff, AZ, September 2011.

[51] G.P. Huang, A.I. Mourikis, and S.I. Roumeliotis. Analysis and Improvement of the Consistency of Extended Kalman Filter Based SLAM. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 473–479, May 2008.

[52] Guoquan Huang, Nikolas Trawny, Anastasios Mourikis, and Stergios Roumeliotis. Observability-Based Consistent EKF Estimators for Multi-Robot Cooperative Localization. *Autonomous Robots*, 30:99–122, 2011.

[53] Myung Hwangbo. *Vision-Based Navigation for a Small Fixed-Wing Airplane in Urban Environment*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, June 2012.

[54] Myung Hwangbo and T Kanade. Visual-inertial UAV attitude estimation using urban scene regularities. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2451–2458, 2011.

[55] V Indelman, S Williams, M Kaess, and F Dellaert. Factor Graph Based Incremental Smoothing in Inertial Navigation Systems. In *Intl. Conf. on Information Fusion, FUSION*, pages 2154–2161, Singapore, July 2012.

[56] V Indelman, S Williams, M Kaess, and F Dellaert. Information Fusion in Navigation Systems via Factor Graph Based Incremental Smoothing. *Journal of Robotics and Autonomous Systems, RAS*, 61(8):721–738, 2013.

[57] M Kaess. *Incremental Smoothing and Mapping.* PhD thesis, Georgia Institute of Technology, December 2008.

[58] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM: Incremental Smoothing and Mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, December 2008.

[59] Marshall H. Kaplan, Bradley Boone, Robert Brown, Thomas B. Tunstel Edward W Criss, and Edward W. Tunstel. Engineering Issues for All Major Modes of In Situ Space Debris Capture. *AIAA SPACE*, August 2010.

[60] Jonathan Kelly and Gaurav S Sukhatme. Visual-Inertial Sensor Fusion: Localization, Mapping and Sensor-to-Sensor Self-Calibration. *International Journal of Robotics Research*, 30(1):56–79, January 2011.

[61] Been Kim, Michael Kaess, Luke Fletcher, John Leonard, Abraham Bachrach, Nicholas Roy, and Seth Teller. Multiple Relative Pose Graphs for Robust Cooperative Mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, AK, May 2010.

[62] P. Kimball and S. Rock. Estimation of Iceberg Motion for Mapping by AUVs. In *Autonomous Underwater Vehicles (AUV), 2010 IEEE/OES*, pages 1–9, September 2010.

[63] Georg Klein and David Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2007.

[64] Y M L Kostyukovskii. Observability of Nonlinear Controlled Systems. *Automation and Remote Control*, (9):1384–1396, 1968.

[65] Dimitrios Kottas and Stergios Roumeliotis. Exploiting Urban Scenes for Vision-aided Inertial Navigation. In *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

[66] Shauying R Kou, David L Elliott, and Tzyh Jong Tarn. Observability of Nonlinear Systems. *Information and Control*, 22(1):89–99, 1973.

[67] T Kubota, T Hashimoto, J Kawaguchi, M. Uo, and K. Shirakawa. Guidance and Navigation of Hayabusa Spacecraft for Asteroid Exploration and Sample Return Mission. In *SICE-ICASE, 2006. International Joint Conference*, pages 2793–2796, October 2006.

[68] Clayton Kunz. *Autonomous Underwater Vehicle Navigation and Mapping in Dynamic, Unstructured Environments*. PhD thesis, Massachusetts Institute of Technology, 2012.

[69] Kwang Wee Lee, W.S. Wijesoma, and I.G. Javier. On the Observability and Observability Analysis of SLAM. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3569–3574, October 2006.

[70] E J Lefferts, F L Markley, and M D Shuster. Kalman Filtering for Spacecraft Attitude Estimation. *AIAA 20th Aerospace Sciences Meeting*, 1982.

[71] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous Map Building and Localization for an Autonomous Mobile Robot. In *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*, pages 1442–1447 vol.3, November 1991.

[72] John Leonard and Hugh Durrant-Whyte. *Directed Sonar Sensing for Mobile Robot Navigation*. Directed Sonar Sensing for Mobile Robot Navigation. Kluwer Academic Publishers, January 1992.

[73] Stefan Leutenegger, Paul Furgale, Vincent Rabaud, Margarita Chli, Kurt Konolige, and Roland Siegwart. Keyframe-Based Visual-Inertial SLAM Using Nonlinear Optimization. In *Proceedings of Robotics: Science and Systems (RSS)*, 2013.

[74] M Li and A.I. Mourikis. 3-D Motion Estimation and Online Temporal Calibration for Camera-IMU Systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 5689–5696, Karlsruhe, Germany, May 2013.

[75] M D Lichter and S Dubowsky. State, shape, and parameter estimation of space objects from range images. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2974–2979 Vol.3, 2004.

[76] Matthew D Lichter. *Shape, Motion, and Inertial Parameter Estimation of Space Objects using Teams of Cooperative Vision Sensors*. PhD thesis, Massachusetts Institute of Technology, February 2005.

[77] David G Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, pages 1150–1150, Washington, DC, USA, 1999. IEEE Computer Society.

[78] F Lu and E Milios. Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, 4:333–349, 1997.

[79] Tim Luu, Stéphane Ruel, and Martin Labrie. TriDAR Test Results Onbaord Final Shuttle Mission, Applications for Future of Non-Cooperative Autonomous Rendezvous & Docking. In *International Symposium on Artificial Intelligence, Robotics and Automationin Space (i-SAIRAS)*, September 2012.

[80] Mark Maimone, Yang Cheng, and Larry Matthies. Two years of Visual Odometry on the Mars Exploration Rovers: Field Reports. *J. Field Robot.*, 24(3):169–186, 2007.

[81] Konrad Makowka. *Vision Based Navigation with an Experimental Satellite Testbed*. PhD thesis, Technical University of Munich.

[82] F Landis Markley. Attitude Error Representations for Kalman Filtering. *Journal of Guidance, Control, and Dynamics*, 26(2):311–317, 2003.

[83] F Landis Markley. Attitude Filtering on SO(3). *Journal of the Optical Society of America*, 54(3 & 4):319–413, July 2006.

[84] A. Martinelli. Local Decomposition and Observability Properties for Automatic Calibration in Mobile Robotics. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 4182–4188, May 2009.

[85] A. Martinelli. State Estimation Based on the Concept of Continuous Symmetry and Observability Analysis: The Case of Calibration. *Robotics, IEEE Transactions on*, 27(2):239–255, April 2011.

[86] A. Martinelli and R. Siegwart. Observability Analysis for Mobile Robot Localization. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 1471–1476, August 2005.

[87] L Matthies and S Shafer. Error modeling in stereo navigation. *Robotics and Automation, IEEE Journal of*, 3(3):239–248, 1987.

[88] Larry Matthies. *Dynamic Stereo Vision*. PhD thesis, Carnegie Mellon University, 1989.

[89] Larry Matthies, Mark Maimone, Andrew Johnson, Yang Cheng, Reg Willson, Carlos Villalpando, Steve Goldberg, Andres Huertas, Andrew Stein, and Anelia Angelova. Computer Vision on Mars. *International Journal of Computer Vision*, 2007.

[90] Christopher McChesney. Attitude Control Actuators for a Simulated Spacecraft. In *AIAA Guidance, Navigation and Control Conference*, pages 1–25, August 2011.

[91] Catharine L R McGhan, Rebecca L Besser, Robert M Sanner, and Ella M Atkins. Semi-Autonomous Inspection with a Neutral Buoyancy Free-Flyer. In *AIAA Guidance, Navigation and Control Conference and Exhibition*, pages 1–11, 2006.

[92] Mark Micire. SMART SPHERES: Testing Free-Flyer Robot Concepts for Future Human Missions. In *International Space Station Research and Development Conference*, June 2012.

[93] Swati Mohan, Alvar Saenz-Otero, Simon Nolet, David W Miller, and Steven Sell. SPHERES Flight Operations Testing and Execution. *Acta Astronautica*, 2009.

[94] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fast-SLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002.

[95] Anastasios I. Mourikis, Nikolas Trawny, Stergios I. Roumeliotis, Andrew E. Johnson, Adnan Ansar, and Larry Matthies. Vision-aided inertial navigation for spacecraft entry, descent, and landing. *Robotics, IEEE Transactions on*, 25(2):264–280, 2009.

[96] Elias Müggler. Visual Mapping of Unknown Space Targets for Relative Navigation and Inspection. Master's thesis, ETH Zurich, July 2012.

[97] Tom Mulder. Orbital Express Autonomous Rendezvous and Capture Flight Operations, Part 2 of 2: AR&C Exercise 4,5, and End-Of-Life. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, August 2008.

[98] Bo Naasz, Doug Zimpfer, Ray Barrington, and Tom Mulder. Flight Dynamics and GN&C for Spacecraft Servicing Missions. In *AIAA Guidance, Navigation and Control Conference*, pages 1–16, Toronto, November 2010.

[99] Richard Newcombe and Andrew Davison. Live Dense Reconstruction with a Single Moving Camera. In *Comference on Computer Vision and Pattern Recognition.* IEEE, 2010.

[100] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, 1:I–652–I–659 Vol. 1, 2004.

[101] Simon Nolet. *Development of a Guidance, Navigation and Control Architecture and Validation Process Enabling Autonomous Docking to a Tumbling Satellite.* PhD thesis, Massachusetts Institute of Technology, 2007.

[102] Simon Nolet. The SPHERES Navigation System: from Early Development to On-Orbit Testing. In *AIAA Guidance, Navigation and Control Conference and Exhibition*, 2007.

[103] Jerome Obermark, Glenn Creamer, Bernard E Kelm, William Wagner, and C Glen Henshaw. SUMO/FREND: Vision System for Autonomous Satellite Grapple. In Richard T Howard and Robert D Richards, editors, *Sensors and Systems for Space Applications*, page 65550Y. SPIE, 2007.

[104] Michael C. O'Connor. Design and Implementation of Small Satellite Inspection Missions. Master's thesis, Massachusetts Institute of Technology.

[105] E Olson and J Leonard. Fast iterative alignment of pose graphs with poor initial estimates. *Robotics and Automation*, 2006.

[106] WM Owen Jr, TC Wang, A. Harch, M. Bell, and C. Peterson. NEAR optical navigation at Eros. *Advances in the Astronautical Sciences*, 2002.

[107] L.D.L. Perera, A. Melkumyan, and E. Nettleton. On the Linear and Nonlinear Observability Analysis of the SLAM Problem. In *Mechatronics, 2009. ICM 2009. IEEE International Conference on*, pages 1–6, April 2009.

[108] L.D.L. Perera and E. Nettleton. On the Nonlinear Observability and the Information Form of the SLAM Problem. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 2061–2068, October 2009.

[109] Mark L Psiaki. Estimation Using Quaternion Probability Densitites on the Unit Hypersphere. *Journal of the Optical Society of America*, 54(3 & 4):415–431, July 2006.

[110] E Rublee, V Rabaud, K Konolige, and G Bradski. ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571, 2011.

[111] Stéphane Ruel, Tim Luu, and Andrew Berube. Space shuttle testing of the TriDAR 3D rendezvous and docking sensor. *Journal of Field Robotics*, 29(4):535–553, January 2012.

[112] Alvar Saenz-Otero. *The SPHERES Satellite Formation Flight Testbed: Design and Initial Control.* PhD thesis, Massachusetts Institute of Technology, MIT.

[113] Alvar Saenz-Otero. *Design Principles for the Development of Space Technology Maturation Laboratories Aboard the International Space Station.* PhD thesis, Massachusetts Institute of Technology, MIT, June 2005.

[114] Alvar Saenz-Otero, Jacob Katz, and Alvin T MwiJuka. The Zero Robotics SPHERES Challenge 2010. In *IEEE Aerospace Conference*, pages 1–13, Big Sky, Montana, March 2010.

[115] Davide Scaramuzza and Friedrich Fraundorfer. Visual Odometry [Tutorial]. *IEEE Robotics & Automation Magazine*, 18(4):80–92.

[116] Daniel Sheinfeld and Stephen M. Rock. Rigid Body Inertia Estimation with Applications to the Capture of a Tumbling Satellite. *Proceedings of 19th AAS/AIAA Spaceflight Mechanics Meeting*, 2009.

[117] Malcolm D Shuster. A survey of attitude representations. *Navigation*, 8:9, 1993.

[118] Malcolm D Shuster. Constraint in Attitude Estimation Part I: Constrained Estimation. *Journal of the Optical Society of America*, 51(1):51–74, January 2003.

[119] Malcolm D Shuster. Constraint in Attitude Estimation Part II: Unconstrained Estimation. *Journal of the Optical Society of America*, 51(1):75–101, January 2003.

[120] G. Sibley, C. Mei, I. Reid, and P. Newman. Planes, Trains and Automobiles - Autonomy for the Modern Robot. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 285–292, May 2010.

[121] Marcel J Sidi. *Spacecraft Dynamics and Control: A Practical Engineering Approach*. Spacecraft Dynamics and Control: A Practical Engineering Approach. Cambridge University Press, 1997.

[122] R. Smith, M. Self, and P. Cheeseman. Estimating Uncertain Spatial Relationships in Robotics. *Autonomous Robot Vehicles*, pages 167–193, 1990.

[123] S. Soatto. Observability/Identifiability of Rigid Motion under Perspective Projection. In *Decision and Control, 1994., Proceedings of the 33rd IEEE Conference on*, pages 3235–3240 vol.4, December 1994.

[124] SpaceRef and NASA. SpaceX's Dragoneye Navigation Sensor Successfully Demonstrated on STS-127, 2013. [Online; accessed 9-June-2013].

[125] George Stockman and Linda G Shapiro. *Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

[126] J Stuelpnagel. On the parameterization of the three-dimensional rotation group. 1948.

[127] J P Swensen and N J Cowan. An almost global estimator on SO(3) with measurement on S2. In *American Control Conference (ACC), 2012*, pages 1780–1786, 2012.

[128] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[129] Ars Technica. How NASA got an Android handset ready to go into space, 2013. [Online; accessed 9-June-2013].

[130] S. Thrun. Simultaneous Localization and Mapping with Sparse Extended Information Filters. *The International Journal of Robotics Research*, 23(7-8):693–716, August 2004.

[131] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics.* The MIT Press, 2005.

[132] Nikolas Trawny, Anastasios I. Mourikis, Stergios I. Roumeliotis, Andrew E. Johnson, and James F. Montgomery. Vision-Aided Inertial Navigation for Pin-Point Landing Using Observations of Mapped Landmarks: Research Articles. *Journal of Field Robotics*, 24(5):357–378, 2007.

[133] Bill Triggs, Philip McLauchlan, Richard Hartley, and Andrew Fitzgibbon. Bundle Adjustment - A Modern Synthesis. In *International Conference on Computer Vision*, pages 298–372. International Workshop on Vision Algorithms, Springer-Verlag, 1999.

[134] B E Tweddle. Relative Computer Vision Based Navigation for Small Inspection Spacecraft. In *AIAA Guidance, Navigation and Control Conference*, 2011.

[135] Brent Edward Tweddle. Computer Vision Based Proximity Operations for Spacecraft Relative Navigation. Master's thesis, Massachusetts Institute of Technology, 2010.

[136] Brent Edward Tweddle, Alvar Saenz-Otero, and David W Miller. Design and Development of a Visual Navigation Testbed for Spacecraft Proximity Operations. In *AIAA SPACE Conference and Exposition*, 2009.

[137] Yair Weiss and William T Freeman. Correctness of belief propagation in gaussian graphical models of arbitrary topology. *Neural Computation*, 13:2173–2200, 2001.

[138] Bong Wie. *Space Vehicle Dynamics and Control.* Space Vehicle Dynamics and Control. AIAA Education Series, 1998.

[139] Wikipedia. MiTEx, 2013. [Online; accessed 4-August-2013].

[140] Brian Williams, Nicolas Hudson, Brent Tweddle, Roland Brockers, and Larry Matthies. Feature and Pose Constrained Visual Aided Inertial Navigation for Computationally Constrained Aerial Vehicles. In *International Conference on Robotics and Automation.* IEEE, 2011.

[141] Hajime Yano, T Kubota, H Miyamoto, T Okada, D Scheeres, Y Takagi, K Yoshida, M Abe, S Abe, O Barnouin-Jha, A Fujiwara, S Hasegawa, T Hashimoto, M Ishiguro, M Kato, J Kawaguchi, T Mukai, J Saito, S Sasaki, and M Yoshikawa. Touchdown of the Hayabusa Spacecraft at the Muses Sea on Itokawa. *Science*, 312(5778):1350–1353, 2006.

[142] Tetsuo Yoshimitsu, Jun'ichiro Kawaguchi, Tatsuaki Hashimoto, Takashi Kubota, Masashi Uo, Hideo Morita, and Kenichi Shirakawa. Hayabusa-final autonomous descent and landing based on target marker tracking. *Acta Astronautica*, 65(5-6):657–665, 2009.

[143] Douglas Zimpfer, Peter Kachmar, and Seamus Tuohy. Autonomous Rendezvous, Capture and In-Space Assembly: Past, Present and Future. In *1st Space Exploration Conference: Continuing the Voyage of Discovery*, 2005.