Autonomous 3D Model Generation of Orbital Debris using Point Cloud Sensors

by

Michael Aaron Trowbridge

A.G.S., Central Texas College, 2007

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Science

Department of Aerospace Engineering

2014

This thesis entitled:
Autonomous 3D Model Generation of Orbital Debris using Point Cloud Sensors
written by Michael Aaron Trowbridge
has been approved for the Department of Aerospace Engineering

_____
Dr. Hanspeter Schaub

_____
Dr. Alireza Doostan

_____
Dr. Brandon Jones

Date _____

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Trowbridge, Michael Aaron (M.S., Aerospace Engineering)

Autonomous 3D Model Generation of Orbital Debris using Point Cloud Sensors

Thesis directed by Dr. Hanspeter Schaub

A software prototype for autonomous 3D scanning of uncooperatively rotating orbital debris using a point cloud sensor is designed and tested. The software successfully generated 3D models under conditions that simulate some on-orbit orbit challenges including relative motion between observer and target, inconsistent target visibility and a target with more than one plane of symmetry. The model scanning software performed well against an irregular object with one plane of symmetry but was weak against objects with 2 planes of symmetry.

The suitability of point cloud sensors and algorithms for space is examined. Terrestrial Graph SLAM is adapted for an uncooperatively rotating orbital debris scanning scenario. A joint EKF attitude estimate and shape similiarity loop closure heuristic for orbital debris is derived and experimentally tested. The binary Extended Fast Point Feature Histogram (EFPFH) is defined and analyzed as a binary quantization of the floating point EFPFH. Both the binary and floating point EPFH are experimentally tested and compared as part of the joint loop closure heuristic.

Dedication

To my late grandfather, Howard E. Glasier, the farmboy-turned-elevator-mechanic who let me write my first computer programs on his Apple IIe and later sent me off to basic combat training with a charge: "never let anyone tell you that you're not good enough."

Acknowledgements

I owe a tremendous debt of gratitude to Dr. Hanspeter Schaub for supporting me in the research that I was most interested in, even though it was not directly tied to his lab's main projects. Our frequent technical discussions and research updates were a key part of my success.

This research and analysis would not have been possible without the help of many other people as well. Thanks to Daan Stevenson and Ian Thom for the encoder testing support, my colleagues in the CU Autonomous Vehicle Systems lab for helping me refine the presentation and thesis, Paul Anderson for suggesting the multiple hypothesis loop closure idea, Steve O'Keefe for recommending a closer comparison of the binary and floating point EFPFH, Jack and Arline Trowbridge for the Space Shuttle model used in the experiments, Dr. Rhonda Morgan (JPL) for recommending the use of more distinctive targets during early development, Dr. Brandon Jones for helping me find and fix a bug in the EKF, Dr. Alireza Doostan and Dr. Scott Palo for recommending a more detailed explanation of the closure heuristics and the probability-like traits, Lee Jasper for images in the introduction/motivation and Surrey Satellite Technology US for sponsoring the TRACSat senior design project, which inspired me to pursue this area of research.

Thanks are also in order to Radu B. Rusu, Jochen Spricherhof, Federico Tombari and Martin Sälzle for taking the time to write the detailed papers, tutorials and open source examples that introduced me to working with Point Clouds.

Special thanks to Lee Jasper, Joshua Chabot, and Dr. Hanspeter Schaub for proofreading and pre-screeing this thesis; it was significantly improved by your feedback.

Contents

Chapter

Tables

Table

Figures

Figure

Chapter 1

Introduction

1.1 Motivation

Unmanned spacecraft have historically been designed for little autonomy. Manned craft have handled the less predictable missions, like cosmonaut Vladimir Dzhanibekov's rendezvous and manual docking with the uncooperative Salyut 7 space station [16]. Using only a laser range finder, low-light camera and orbital estimates from mission control, Dzhanibekov piloted his Soyuz spacecraft to the lifeless space station and docked on his first attempt [16]. This involved several tasks that are difficult for unmanned spacecraft:

- Determining whether or not the object in front of him was a Salyut space station
- Estimating the relative position, velocity, orientation and angular rate of the Salyut 7
- Avoiding the Salyut's deployed solar arrays

A key element of this is Dzhanibekov's ability to learn the shapes of new objects, track and predict their relative motion. This spatial awareness is no longer the exclusive domain of humans. With the right software and point cloud sensor data, unmanned spacecraft can also learn the shapes of new objects.

Learning the shapes of new objects and maintaining their 6DOF state estimates are key capabilities for several unmanned spacecraft mission profiles. Tether-based active debris removal systems, such as the Soyuz rocket space tug proposed by Jasper et al. [19] require knowledge of the debris position and orientation to attach the towing tether. On-orbit repair and refueling missions are precise autonomous docking maneuvers that may also involve extended station-keeping and formation flying [57]. The tether-based asteroid towing

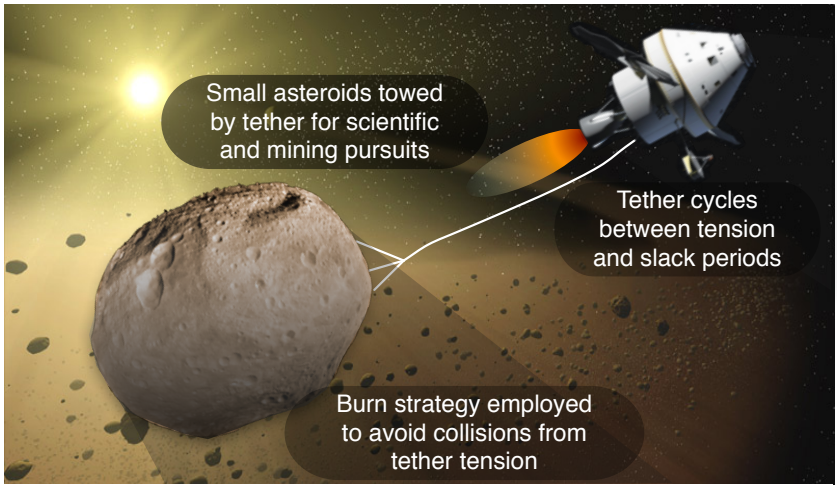

Figure 1.1: Tether-based asteroid capture system proposed by Jasper and Dr. Schaub [20]

system proposed by Jasper and Schaub (figure 1.1) would also need topographical knowledge to choose the tether attachment points [20] .

LIDAR, scanning laser rangefinders, time of flight cameras and structured light sensors (figure 1.2 [1]) could help. These sensors describe their surroundings as sets of distinct points (x, y, z) called point clouds. Point clouds contain a large amount topographical information but require different algorithms than traditional computer vision. This thesis explores how point cloud algorithms can be applied to spacecraft proximity operations, identifies some of their weaknesses and proposes some strategies to mitigate them.

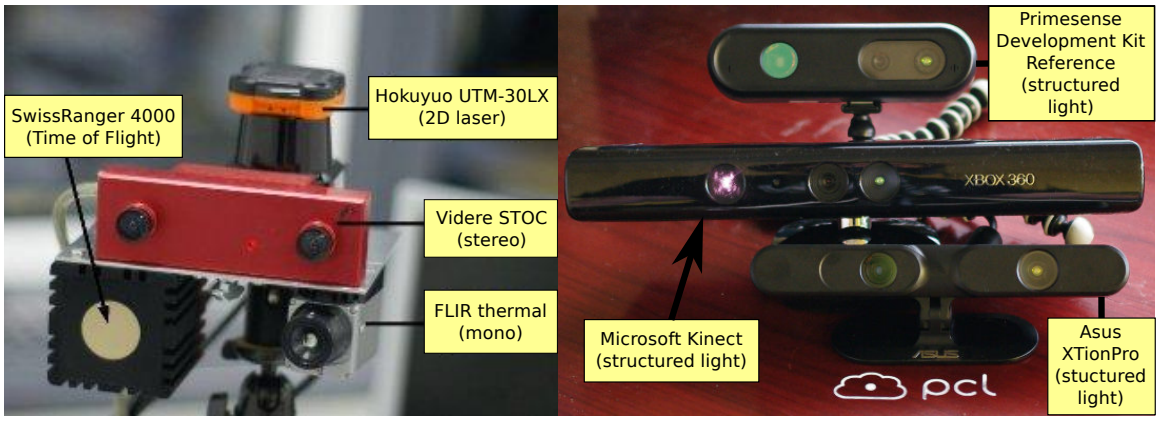

Figure 1.2: Some point cloud sensors that have been used in terrestrial robotics.

[1] These images are from the PointClouds.org website and are licensed under Create Commons Attribution 3.0

1.2 Literature Review

1.2.1 Previous autonomous and semi-autonomous spacecraft docking systems

Autonomous rendezvous, docking and joint flight was demonstrated on-orbit by Cosmos 186 and Cosmos 188 in 1967 [35]. These spacecraft used a collaborative docking system named Igla, in which each spacecraft broadcast a docking beacon and used multiple RF antennas to determine the relative orientation and distance between the spacecraft [58] (figure 1.3 [2]).

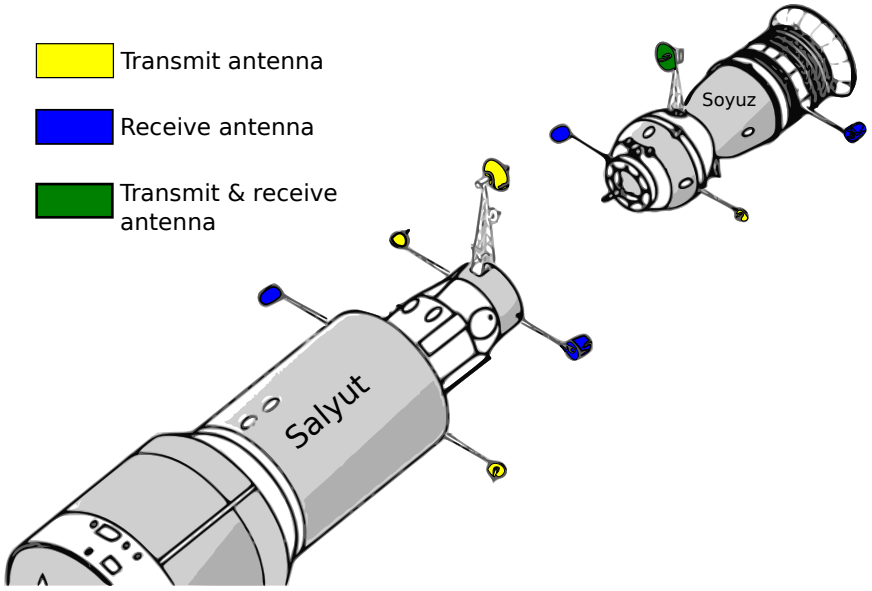

Figure 1.3: Location of Igla RF antennas on the Salyut space station and Soyuz spacecraft.

Igla was succeeded by Kurs and Kurs-NA, which are incremental improvements on the same collaborative, 2-party RF transponder approach [16]. Kurs and Kurs-NA are currently used for human-in-the-loop cargo vehicle docking on the International Space Station [33], [23]. The European Space Agency (ESA) Automated Transfer Vehicle (ATV) spacecraft (figure 1.4) is the next evolution in this concept, replacing the collaborative transponder system with Global Positioning System (GPS) receivers, star trackers, gyros, accelerometers, two computer vision-based video camera sensors and a laser reflectrometer [39]. Kurs, Kurs-NA and the ATV suite are all supplemented by a television camera and visual tracking markers or

[2] This image is a modified version of Marcos Ricardo's diagram [41] and licensed under the Creative Commons Attribution-Share Alike 2.0 Generic license.

retro-reflectors [16, 39], making them unsuitable for autonomous rendezvous with arbitrary, non-cooperative objects.

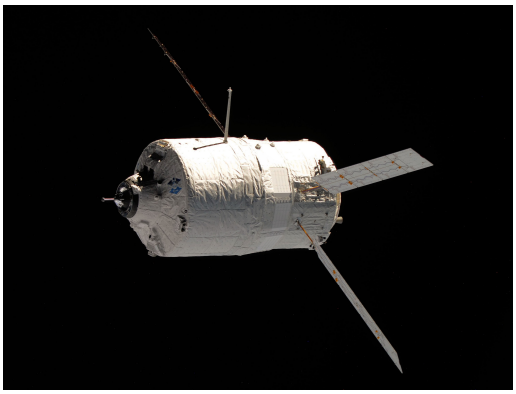

Figure 1.4: Photograph of the ESA ATV-2 on approach to the ISS [34]

Neptec's TriDAR is one of the first systems to demonstrate marker-free 6 degree of freedom (6DOF) pose estimation in space. TriDAR is a model-based, 3-sensor tracking suite was demonstrated by a joint Canadian Space Agency (CSA)/NASA project on Space Shuttle Discovery during mission STS-128 in 2009 (figure1.5) [40]. TriDAR is named for its combination of laser triangulation, LIDAR and a thermal imager to provide overlapping coverage at different ranges [42]. TriDAR is more flexible than its predecessors, but its need for a human generated, 3D CAD model of the target limits its usefulness for arbitrary orbital debris.

Figure 1.5: The TriDAR ISS tracking test flight on STS-128, extracted and modified from [40].

The joint Swedish, German, French and Danish PRISMA project is taking the marker-free approach one step further. The PRISMA project uses greyscale cameras and monocular computer vision pose estimation for a two-satellite autonomous search, rendezvous and formation flight [10]. This mission is particularly focused on technologies required for autonomous, on-orbit spacecraft repair [10], including a scheduled orbital debris inspection flyby based on combined GPS and computer vision relative navigation [24]. Karlsson et al.'s mission overview paper [24] does not list on-board, autonomous 3D model generation as a mission objective. Autonomous shape learning and 3D model generation does not appear to be a PRISMA mission objective.

Two missions are using structured light sensors, a novel new class of 3D sensor that was originally designed for video games [25, 54]. STRaND-2, a joint Surrey Space Centre (SSC) and Surrey Satellite Technology Limited (SSTL) mission will use a Microsoft Kinect^{TM} to demonstrate an on-orbit separation and re-docking of two 3U CubeSats [54]. The National Aeronautics and Space Administration (NASA) Jet Propulsion Laboratory (JPL) has selected an in-house custom-designed structured light sensor for spacecraft inspection on the Micro-Inspector spacecraft [28].

1.2.2 Structured Light Point Cloud Sensors

Structured light sensors work by projecting a predefined pattern of light onto the environment, then correlating the predefined pattern with some deformed version of the pattern that is observed by a camera. If the surface that is illuminated by the pattern is flat, then the observed pattern should be some single homogeneous transformation of the projected pattern. If the object is not flat, some elements of the pattern will appear in an unexpected pixel of the camera. These differences between the expected location of pattern elements and the observed locations of the pattern elements are used to infer the presence of some physical object at a given point.

Which specific patterns should be projected is still an area of active research. The JPL Micro-Inspector's structured light prototype (figure 1.6) produced a 2-d grid created by passing a laser beam through a diffractive grid [28]. This specific sensor triangulates each individual point based on known separation distance d between the laser and the camera [28].

Figure 1.6: JPL Micro-Inspector spacecraft structured light pattern [28]. Courtesy NASA/JPL-Caltech.

1.2.3 Combining 3D point cloud sensors with color cameras

The first generation Microsoft Kinect (figure 1.7) uses a combination of structured light point cloud generation with a conventional red-green-blue digital camera to create 6-dimensional point vectors (x,y,z,r,g,b) [8]. PrimeSense, which owns the intellectual property behind the Kinect™, released a free software suite that allows computers to read the Kinect's raw point cloud data. This free software interface and the Kinect™'s low cost (approximately $30 used) have made it a popular starting point for point cloud research.

Figure 1.7: Microsoft Kinect structured light sensor [1]

Future systems may extend depth+color point cloud sensors to kilometer ranges. HD6D, a suite

developed by Dr. John L. Junkins and Dr. Manoranjan Majji is between one meter and 25 kilometers, with lower precision as the range increases [22]. This range would be ideal for on-board autonomous 3D model generation of uncooperatively rotating space debris, but HD6D is not yet available to the general public. Because the Microsoft Kinect^{TM} produces a lower quality version of the same class of data, a first generation Microsoft Kinect^{TM} was used to capture the point cloud data used in this thesis.

1.2.4 Point cloud sensors for orbital debris removal

Austin Probe and Dr. John L. Junkins of the Texas A&M University Land, Air and Space Robotics (LASR) Laboratory performed a series of experiments demonstrating a guidance, navigation and control (GNC) system for active orbital debris removal in 2013 [2]. These experiments used the terrestrial robot with 2D translation and 1D planar rotation, but with 3DOF attitude control to simulate an autonomous planar rendezvous and docking maneuver with uncooperative rocket body rotating at 7° per second. The robot was programmed to dock by inserting a probe into the rotating rocket nozzle (figure 1.8[3]).

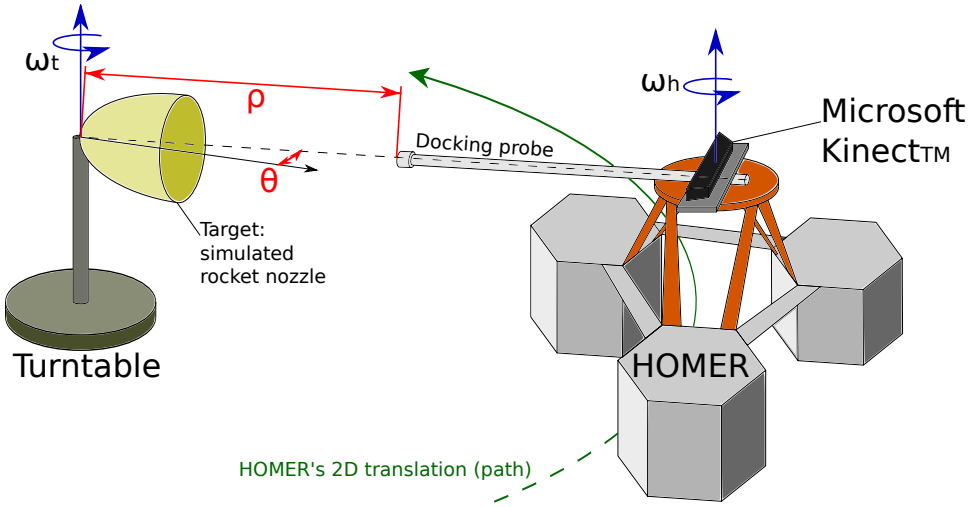

Figure 1.8: Diagram of the Texas A&M LASR Lab active debris removal GNC demonstration

The key spatial sensor in the HOMER robot was a structured light point cloud sensor (originally a Microsoft Kinect^{TM} [27]). This experiment demonstrated that point cloud sensors can provide the 3D

[3] This diagram is an original sketch based on motion observed in the official LASR lab video at http://vimeo.com/61226558

recognition and tracking required for autonomous debris capture. The docking sequence demonstrated by the LASR lab is extremely similar to the capture phase of the tether-based system proposed by Jasper et al. in [19] (figure 1.9).

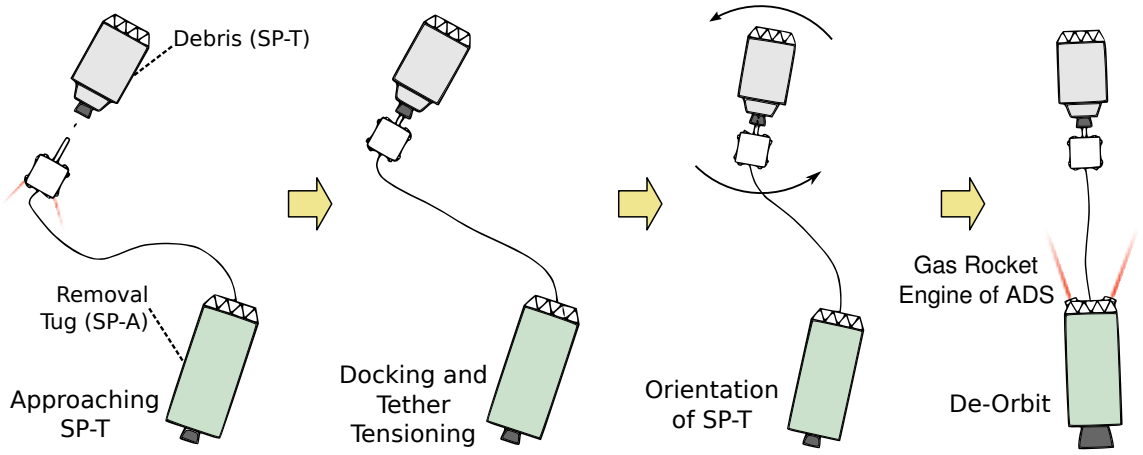

Figure 1.9: The Jasper et al. Soyuz tether-based Active Debris System (ADS). Courtesy of Lee Jasper [19].

This promising capability can be extended further - by upgrading the sensor to let the robot learn new shapes and take on arbitrary debris, not just rocket bodies.

1.2.5 Point cloud sensors and techniques

Point clouds can be generated by recording byproducts from many different physical and electromagnetic processes. Some of these, like sonar, do not function in the near-vacuum of space. Phased array radar can produce good quality data, but currently requires a large amount of power. Optical sensors show particular promise for space applications. LIDAR has considerable range (kilometers [31]) for a competitive power cost. Stereoscopy and time of flight cameras use very small amounts of power, but are limited to close range (tens of meters [28]). Combined, these optical systems could provide coverage from initial approach through docking. All of these systems share one thing in common - they can all be used to produce 3D Cartesian coordinate point clouds.

Stereoscopy and structure from motion systems like Davison's monocular VSLAM algorithm [53] are promising, but their usefulness is limited to periods of sunlight or the range of a headlight. NATO's technical

report on point cloud sensor system presents a "general rule" that stereoscopic point cloud generators are not very useful for ranges $R > 20d$, where d is the baseline distance between the two cameras [36]. The size and lighting limitations make them less flexible than structured light, LIDAR or time of flight cameras. Some time of flight cameras can also detect translucent objects while still detecting and resolving point clouds for the objects behind them [6], which may be useful for inspection of space telescopes or windows of manned spacecraft (the unoccupied Salyut 7 scenario).

1.2.6 Expected challenges for structured light sensors in space

Calderita et al. characterized the performance of the Microsoft Kinect for robotics applications in 2012. Their testing showed that its the Kinect's range estimates were mostly linear over a range of 1-5 meters, with error increasing drastically beyond 5.5 meters. Among other things, the Kinect's performance was characterized against shiny gold and silver paper samples at 2 meters [8].

The Kinect performed poorly against the silver and gold paper samples. Distance measurements against the silver paper, which was selected to resemble aluminum, had a standard deviation that was an order of magnitude larger than the matte surfaces (0.12231 mm at 2m true distance). Both the silver and gold paper were only visible at low incidence angles. At an incidence angle of 20°, 65% of the pixels that should have observed the gold paper reported no sensor data. The silver paper was almost as difficult to see, with 47% of the pixels reporting no data at a 20° incidence angle. Both the gold and silver paper samples were invisible at incidence angles of 40° and 60° [8].

This is troubling because most satellites are wrapped in shiny gold or silver-colored multiple layer insulation (MLI) thermal blankets. This might cause the debris or target spacecraft to blink in and out of existence, from the sensor's perspective, without actually leaving the sensor's field of view. Irregular surfaces could aid detection by creating small patches that have a low enough incidence angle to be visible, but there is still the problem of rapid change in the size and shape of the visible fragment. This rapid change may prevent the Iterative Closest Points algorithm from aligning two successive frames, which would significantly hinder pose estimation or autonomous model generation.

1.3 Scope of Thesis

This thesis focuses on the next step in unmanned spatial awareness - the ability to generate 3D models of uncooperatively rotating objects based on previous observations. The orbital environment, materials spacecraft are constructed from and limitations of point cloud sensors pose some some unique challenges that will be explored in this thesis. The research questions addressed are:

(1) How well do existing robotics algorithms cope with the space environment?

   (a) Inconsistent visibility of the target

   (b) Relative motion between the observer and target

(2) How can existing algorithms be made more robust for space?

(3) Can current point cloud algorithms be combined to autonomously build a model of an uncooperatively rotating object from only an ordered set of point cloud observations and observation times?

(4) Are their specific shapes that these algorithms are strong or weak against?

The first research question will be addressed by examining the SLAM problem and focusing on the Graph SLAM algorithm in particular. These will be introduced in chapter 2, then adapted for the orbital debris scenario. They will then be tested using experiments described in chapter 3 and the results explored in chapter 4.

Section 2.10 derives a tailor-made loop closure heuristic for non-cooperative orbital debris to address the second research question. Specifically, this heuristic will accommodate some relative motion, inconsistent visibility and long period data dropouts. The loop closure heuristic will be tested as part of the experimental software stack described in section 3.2, evaluated in the context of each experiment in chapter 4, then discussed overall in section 5.3.

Research questions 3 and 4 will be addressed by experimental verification using a Microsoft Kinect^{TM} structured light sensor. Three variations of the same experiment will be performed, in which the software attempts to construct a 3D model of a non-cooperatively rotating plastic Space Shuttle model using point clouds from a Microsoft Kinect^{TM}.

Chapter 2

Background and Theory

The autonomous 3D model generation experiments are based on a combination of feature detection, similarity analysis, attitude estimation and frame registration. The frame registration uses the Lu-Milios Graph SLAM algorithm (presented in section 2.2.5). This chapter describes other theory and problem-specific derivations that support the experimental apparatus defined in section 3.1. Junkins-Schaub notation [51] is used for attitude derivations. Tapley, Shutz and Born notation [55] is used for estimation and filtering derivation.

2.1 Graph Theory Primer

This section presents a brief overview of selected graph theory concepts that are used elsewhere in this thesis. For a more broad introduction to the topic, please consult [13]. Reference [9] is recommended for a more detailed, rigorous description that describes the depth first and breadth first search algorithms.

The word "graph" has a special meaning in computer science. A graph, like the one in figure 2.1, is a set of nodes and edges that defines how entities are connected. A node is represented in this thesis as a circle. An edge is a line between nodes that represents a path between nodes. In some texts, nodes are referred to as vertices [13]. Nodes may or may not be connected to other nodes with edges. For the graphs produced and discussed in this thesis, all edges are directed and can only be followed in the direction of the arrow.

Figure 2.1: Nodes and edges

A graph may also contain smaller graphs that are called subgraphs. In figure 2.2, the yellow nodes form the maximal subgraph – the subgraph within the graph that contains the largest number of nodes. Subgraphs that are not part of the maximal subgraph are called "islands." In figure 2.2, the node 6 subgraph, the 7-8-9 subgraph and the 10-11 subgraph are all islands.

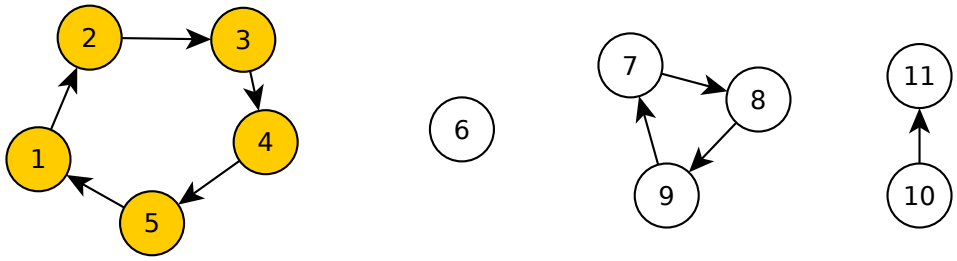

Figure 2.2: Subgraphs, a maximal subgraph (yellow) and islands (white)

A sequence of edges and nodes is called a path. A cycle is a special type of path in which one can start at any node, then follow edges and eventually return to the node that the person started at. A graph may have 0 or more cycles. In figure 2.2, both the 1-2-3-4-5 and 7-8-9 subgraphs are cycles, but the 6 and 10-11 subgraphs are not cycles.

Cycles are the most important graph feature for Graph SLAM because they indicate loop closure. In figure 2.2, node 4 closes a loop with node 1. Real data sets are not always as clear-cut as the 1-2-3-4-5 subgraph. It is possible for a graph to contain a cycle that is smaller than the size of the total graph, as in figure 2.3. The impact this has on Graph SLAM's performance is discussed in more detail in section 5.2.

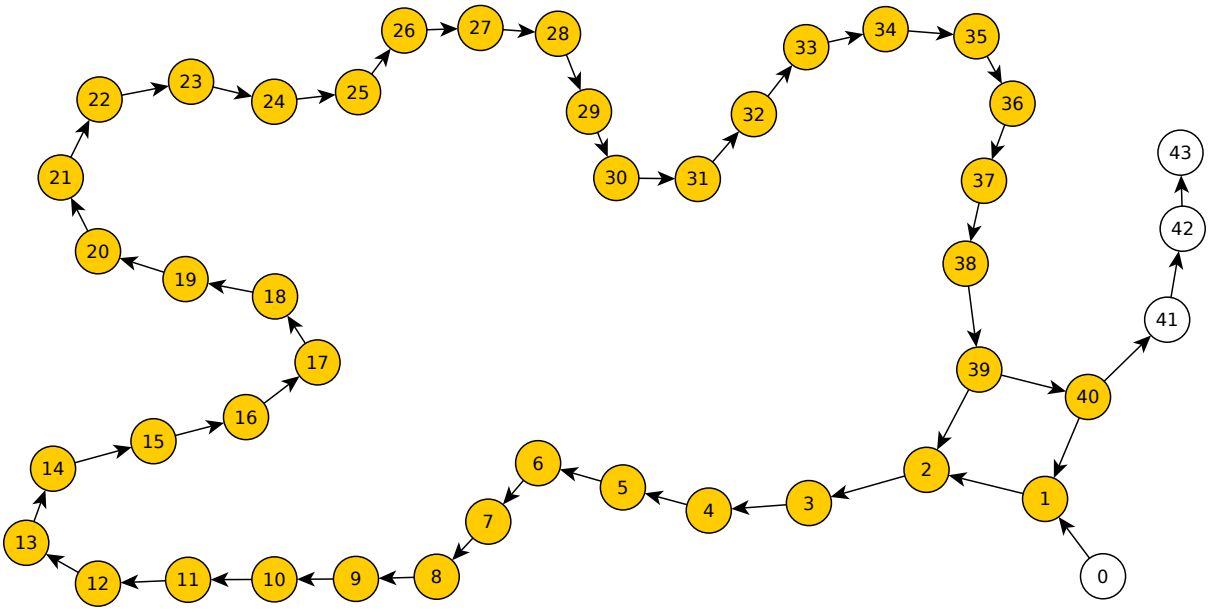

Figure 2.3: Graph that needs to be pruned

Nodes that are not part of a cycle can be identified using an approach derived from set theory. The set of all nodes in the graph in figure 2.3 is

$$N = \{0, 1, 2, \ldots, 43\} \tag{2.1}$$

The set of nodes that are part of the maximal cycle in the maximal subgraph C is

$$C = \{1, 2, 3, \ldots, 40\} \tag{2.2}$$

The set of nodes D that should be deleted because they are not part of a cycle are the difference of these two sets:

$$D = N - C = \{0, 41, 42, 43\} \tag{2.3}$$

A cycle can be identified by performing a depth first or breadth first search of the graph [9]. As described by Cormen et al. in [9], a depth first search starts by selecting some node to be the root node (starting point) for the search. It follows all nodes going out of that node until it either returns to a node that it has previously visited (cycle identified) or reaches a node that has no outgoing nodes (dead end, no cycle identified). If it reaches a node that has no outgoing nodes, the program backtracks to the last prior node

that has an unused outgoing edge, then follows that path until it either reaches a previously visited node or reaches another dead end. When all possible edges have been followed, the depth first search terminates.

A depth first search could be used to identify all of the cycles in a graph by:

(1) Choosing a root node $x_i \in N$

(2) Perform depth first search until a cycle is detected or outgoing (forward) edges are exhausted

(3) Record all of the nodes visited in the cycle detection path (generate cycle set C_i)

(4) For each node that was not part of the set in step 3 ($x \notin C_i$), recurse into this algorithm using that node as the root node.

(5) When all nodes have been visited, merge the C_i sets as $C = \bigcup_{i=1}^{n} C_i$

(6) Remove subset D by discarding the previous set N and redefining $N \equiv C$

2.2 Prior work: Algorithms and Tools for Point Clouds

2.2.1 Iterative Closest Points (ICP) algorithm

The Iterative Closest Points algorithm is a method developed by Paul Besl and Neil McKay in 1992 for aligning two 3-dimensional shapes that was [5]. The algorithm predates the term 'point cloud,' which Besl and McKay [5] refer to as a "point set." They state the problem as finding a transformation vector $\vec{q}$ that aligns a set of measured (observed) points in $P = \left\{\vec{p}_0, \vec{p}_1, \ldots, \vec{p}_{N_p}\right\}$ with a set of reference points $X = \{\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_{N_x}\}$ in an *a priori* object model [5]. They place the constraint that the two point clouds P and X must have the same number of points ($N_p = N_x$) [5].

The transformation vector $\vec{q}$ is defined as as a concatenation of the orientation quaternion $\vec{q}_R = \vec{\beta}$ and translation vector $\vec{q}_T = \mathbf{t}$ [5] [1] :

$$\vec{q} = \begin{bmatrix} \vec{q}_R \\ \vec{q}_T \end{bmatrix} = \begin{bmatrix} \vec{\beta} \\ \mathbf{t} \end{bmatrix} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ x \\ y \\ z \end{bmatrix} \tag{2.4}$$

The core of the algorithm is the variable d_k, which is the result of a mean-squares cost function that describes the Euclidean distance between the points in the observed set P and the reference set Y. In the original paper, the set of points $Y = \left\{\vec{y}_0, \vec{y}_1, \ldots, \vec{y}_{N_p}\right\}$ is defined as a subset of X where each point in Y is the closest point in X to the corresponding position in set P. Y is a set of Cartesian point vectors $\vec{y}_i$ that satisfies the relation

$$\vec{y}_i = \vec{x}_k \in X : |\vec{x}_k - \vec{p}_i| = \min\left\{|\vec{x}_j - \vec{p}_i|\,,\ \vec{x}_j \in X,\ 0 \leq j \leq N_x\right\},\ 0 \leq i \leq N_p \tag{2.5}$$

The cost (objective) function is computed as [5]

$$d_k = \frac{1}{N_p} \sum_{i=1}^{N_p} \left|\vec{y}_{i,k} - \vec{p}_{i,k+1}\right|^2 \tag{2.6}$$

[1] Notation has been altered from Besl and McKay's original paper for consistency with other parts of this thesis.

where the next iteration's observed point cloud P_{k+1} is a linear transformation of the points $\vec{p}_i$ in point cloud P_k[2] :

$$\begin{bmatrix} \vec{p}_{i,k+1} \\ 1 \end{bmatrix} = \begin{bmatrix} R\left(\vec{\beta}_k\right) & \mathbf{t}_k \\ 0_{1x3} & 1 \end{bmatrix} \begin{bmatrix} \vec{p}_{i,k} \\ 1 \end{bmatrix} \tag{2.7}$$

The Iterative Closest Points algorithm has converged when the change in mean squares error drops below some threshold τ [5]:

$$d_k - d_{k+1} < \tau \tag{2.8}$$

Table 2.1 and figure 2.4 summarize the key steps and equations in the classic ICP algorithm.

Table 2.1: Names of key ICP equations/steps

Equation	**Name**
2.5	Compute Correspondences
SVD	Compute registration
2.7	Apply registration
2.8	Check convergence

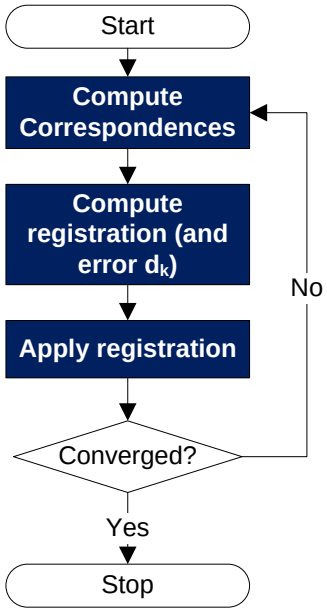

Figure 2.4: Iterative Closest Points Algorithm

[2] Besl and McKay's original transformation [5] has been replaced with an equivalent affine homogeneous transform for consistency with other parts of this thesis.

2.2.2 ICP terminology and extensions in the Point Cloud Library

The Point Cloud Library (PCL) is an open-source software platform for spatial awareness and object recognition using 3D point clouds. It uses slightly different terminology to describe parts of the ICP algorithm. Choosing the set of points Y is referred to as "correspondence estimation." The quantity d_k is referred to as alignment "fitness" and "Euclidean fitness" [49].

The correspondence estimation process in the Point Cloud Library is more generic than Besl and McKay distance-based process. Under Besl and McKay's definition, a correspondence between $\vec{p}_i$ and $\vec{x}_i$ only indicates that point $\vec{x}_i$ is the closest point (Euclidean distance) to point $\vec{p}_i$ [5]. The default ICP implementation uses the Besl and McKay's strict Euclidean distance definition [49], but other implementations in the Point Cloud library include estimated surface normal vectors $\hat{\mathbf{n}}_i$ and $\hat{\mathbf{n}}_j$ at points $\vec{p}_i$ and $\vec{x}_j$, respectively [46] or similarity of the local topography's Intrinsic Shape Signatures near points $\vec{p}_i$ and $\vec{x}_j$ [4].

2.2.3 Point Feature Histograms (PFH)

Point feature histograms, first presented by Rusu et al in [50], are a feature detection tool for regions of a point cloud. They are an information reduction scheme for angles and distances between points within a radius r of some queried point $\mathbf{p}_q$ on the object's surface [50]. Each point in a k-neighborhood of radius r about point $\mathbf{p}_q$ is compared to each other point and the results are binned in a histogram [50] (figure 2.5 [3]). The dark blue points inside the circle contribute to the PFH of point $\mathbf{p}_q$. The points outside of the circle do not contribute.

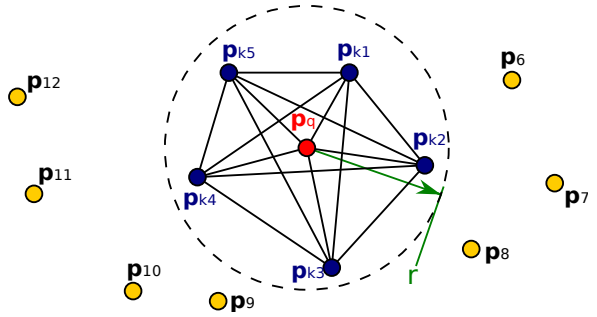

Figure 2.5: Radius-based k neighborhood for point $\mathbf{p}_q$. Diagram courtesy Radu B. Rusu [44].

[3] Figure 2.5 is a re-drawing of a diagram that originally appeared in [50], but is licensed under Creative Commons Attribution 3.0 through the Point Cloud Library website [44].

Rusu's original point feature histogram algorithm [50] defines the difference between two points on the surface of an object as a set of four numbers: f_1, f_2, f_3, and f_4. Three of these four numbers (f_1, f_3, f_4) are derived from orientation angle differences between the surface normal vectors at the two points being compared. The fourth, $f_2 = |\mathbf{p}_s - \mathbf{p}_t|$, is the straight line distance between the points. Figure 2.6[4] shows the frame definition used by to calculate these features.

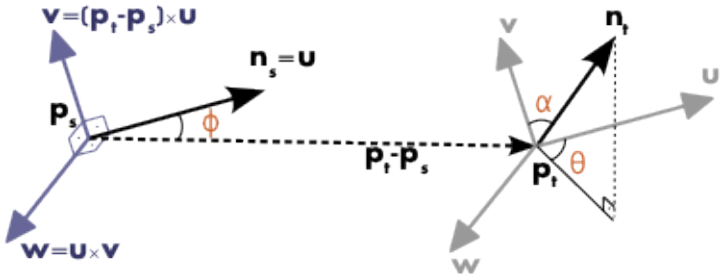

Figure 2.6: Local *uvw* frame established between points $\mathbf{p}_s$ and $\mathbf{p}_t$. Diagram courtesy Radu B. Rusu [44].

The vector valued comparison function $\mathbf{f}(\mathbf{p_s}, \mathbf{p_t})$ produces a feature vector the describes the differences between the two points [50]:

$$\mathbf{f}(\mathbf{p_s}, \mathbf{p_t}) = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{v}} \cdot \hat{\mathbf{n}}_\mathbf{t} \\ |\mathbf{p_t} - \mathbf{p_s}| \\ \hat{\mathbf{u}} \cdot (\mathbf{p_t} - \mathbf{p_s}) / f_2 \\ \arctan(\hat{\mathbf{w}} \cdot \hat{\mathbf{n}}_\mathbf{t}, \hat{\mathbf{u}} \cdot \hat{\mathbf{n}}_\mathbf{t}) \end{bmatrix} \tag{2.9}$$

Rusu's algorithm also includes an ordering step to reduce the amount of redundant information generated. To ensure that two points are only compared once and always compared in the same order, the two points are sorted before being passed as arguments to the feature function defined in equation 2.9. The first point in the ordered pair is the point whose surface normal is most parallel to the line between the two points

[4] Figure 2.6 originally appeared in [50], but is licensed under Creative Commons Attribution 3.0 through the Point Cloud Library website [44].

[50]. Rusu's definition uses angles [50], but this vector notation is equivalent:

$$\{\mathbf{p_s}, \mathbf{p_t}\} = \begin{cases} \{\mathbf{p_i}, \mathbf{p_j}\}, & if\ |\hat{\mathbf{n}}_\mathbf{i} \cdot (\mathbf{p_i} - \mathbf{p_j})| > |\hat{\mathbf{n}}_\mathbf{j} \cdot (\mathbf{p_i} - \mathbf{p_j})| \\ \{\mathbf{p_j}, \mathbf{p_i}\}, & if\ |\hat{\mathbf{n}}_\mathbf{i} \cdot (\mathbf{p_i} - \mathbf{p_j})| < |\hat{\mathbf{n}}_\mathbf{j} \cdot (\mathbf{p_i} - \mathbf{p_j})| \end{cases} \tag{2.10}$$

A Point Feature Histogram is calculated from a very large volume of data. If the radius r of $\mathbf{p}$ contains k neighboring points, then the source data for a point feature histogram is on the order of $O\left(k^2\right)$ individual numeric values (more precisely, a $2k^2 - 2k$ dimensional vector):

$$C(k, 2)\ \text{pairs} \times \frac{4\ \text{values}}{\text{pair}} = \frac{4k!}{2!(k-2)!} = \frac{4k(k-1)(k-2)!}{2!(k-2)!} = 2k^2 - 2k \approx O\left(k^2\right) \tag{2.11}$$

This is too large a data set to be practical data if r is large enough to include enough points to represent significant surface topology features. Rusu's classic Point Feature Histogram algorithm addresses this with a two-stage information reduction process that reduces a $2k^2 + 2k$ dimensional vector into a 16 dimensional feature vector. First, each individual feature vector $\mathbf{f}(\mathbf{p_s}, \mathbf{p_t})$ is converted to an integer idx whose value is between 0 and 15 [50]:

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \Rightarrow \begin{bmatrix} b_1 = \begin{cases} 1, & f_1 \geq 0 \\ 0, & f_1 < 0 \end{cases} \\ b_2 = \begin{cases} 1, & f_2 \geq r \\ 0, & f_2 < r \end{cases} \\ b_3 = \begin{cases} 1, & f_3 \geq 0 \\ 0, & f_3 < 0 \end{cases} \\ b_4 = \begin{cases} 1, & f_4 \geq 0 \\ 0, & f_4 < 0 \end{cases} \end{bmatrix} \Rightarrow b_1 2^0 + b_2 2^1 + b_3 2^2 + b_4 2^3 = idx \tag{2.12}$$

The binary quantization of b_2 makes the Point Feature Histogram pose invariant. The other three features were already pose invariant because they are trigonometric functions that describe angular differences between vectors. Assuming that the point cloud represents a rigid body, the same rotation transformation would be applied to both points and the surface normals, which preserves the angular differences. Increasing

the scale of the point cloud also does not alter the angular differences between these vectors. The only feature that could be affected by a scale is f_2. If the search radius r is chosen as a fraction of the point cloud size, then the quantization of f_2 in terms of r makes the Point Feature Histogram scale invariant.

The value of idx is only useful for comparing two points. If two different sets of points both compute to the same value of idx, then those two points have somewhat similar local topography and point density. For example, if $idx_1 = idx_2 = 14$ and $idx_3 = 2$, then idx_1 is more similar to idx_2 than it is to idx_3. We cannot say precisely how similar or how different; that would require precision that was discarded by equation 2.12.

The second stage of data reduction happens after all of the possible unique feature vectors $\mathbf{f}$ have been calculated and quantized into index values idx. The set of all calculated idx values A is turned into a histogram vector $\mathbf{h}$ by counting the number of times each possible value occurs in the set A, then normalizing by k to make the histogram function less sensitive to changes in the size of the k-neighborhood [50]:

$$\mathbf{h} = \frac{1}{k}\begin{bmatrix} h_0 \\ \vdots \\ h_{15} \end{bmatrix} = \frac{1}{k}\begin{bmatrix} |\{idx \in A : idx = 0\}| \\ \vdots \\ |\{idx \in A : idx = 15\}| \end{bmatrix} \tag{2.13}$$

2.2.3.1 Fast Point Feature Histograms (FPFH)

One year after presenting the Point Feature Histogram, Rusu et al. presented a modified version of the Point Feature Histogram [47]. This new version, the Fast Point Feature Histogram, reduced computational complexity from order nk^2 to order nk, where n is the number of points and k is the number of points in the k neighborhood of queried point $\mathbf{p}_q$ [47]. They did this by changing the search radius r from a fixed quantity to a true k nearest neighbors search, then adding in the histograms of each neighbor in a weighted sum [47].

The second way that Rusu et al. reduced computational complexity was by reducing the size of the feature vector. To make the algorithm less sensitive to scale changes, they removed the distance-based feature f_2 of the original Point Feature Histogram [47] and introduced the term "Simple Point Feature" (abbreviated "SPF") to describe the reduced feature set calculation [47].

A third way that they reduced the complexity of the algorithm was by removing the histogram creation for each individual point, since the weighted sum in equation 2.15 accomplishes the same general purpose. In

lieu of the individual point histograms, the three element SPF tuples are concatenated into a vector. Rusu et al. also removed the binary quantization of equation 2.12, retaining the floating point values[5] [47]:

$$\mathbf{f}(\mathbf{p_s}, \mathbf{p_t}) = \begin{bmatrix} f_1 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} \alpha \\ \phi \\ \theta \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{v}} \cdot \hat{\mathbf{n}}_t \\ \hat{\mathbf{u}} \cdot \frac{(\mathbf{p}_t - \mathbf{p}_s)}{|\mathbf{p}_t - \mathbf{p}_s|} \\ \arctan\left(\hat{\mathbf{w}} \cdot \hat{\mathbf{n}}_t, \hat{\mathbf{u}} \cdot \hat{\mathbf{n}}_t\right) \end{bmatrix} \tag{2.14}$$

Rusu used a $k = 10$ neighborhood in [47], resulting in a $3\,(10 + 1) = 33$ element feature vector of 32-bit floating point numbers [47].

Using a true k nearest neighbors set allows the algorithm to build the neighborhood by traversing a k-dimensional search tree (KD-tree) that is constructed one time, at the beginning of a batch of FPFH calculations. The disadvantage is that the search radius is no longer restricted to a specific distance r. Some of the points that contribute to the FPFH are outside of the target radius r (but no more than $2r$ away) [47] (figure 2.7[6]).

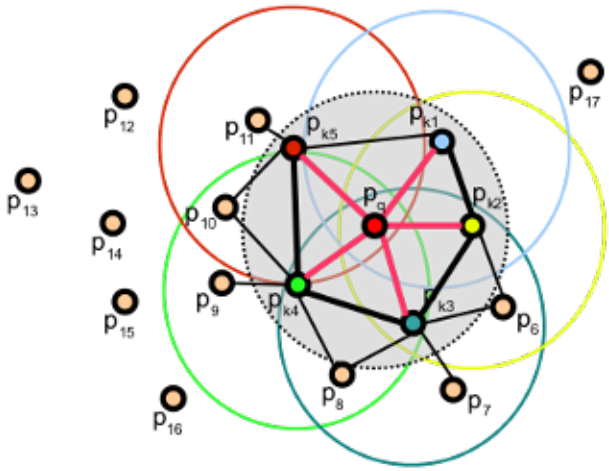

Figure 2.7: Fast Point Feature Histogram k neighborhood. Diagram courtesy Radu B. Rusu [43].

The wider magenta lines in figure are the first order k neighborhood. The SPF of this group forms the

[5] For consistency with figure 2.6, f_1 and f_3 are actually $\cos\alpha$ and $\cos\phi$, respectively. This is derived from the dot product identity:$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}|\,|\mathbf{b}| \cos\beta$, with $\mathbf{a}$ and $\mathbf{b}$ as unit vectors. This conflicts with Rusu's definition in [50] and [47], but the PFH or the FPFH as still valid point features. Cosine of the angle is still a valid feature because α and ϕ are defined between $-180°$ and $+180°$, a domain for which cosine is an onto mapping. Because cosine is an even function, this mapping implies an absolute value operation. It also implies that f_1 and f_3 are nonlinear features.

[6] Figure 2.7 originally appeared in [47], but is licensed under Creative Commons Attribution 3.0 through the Point Cloud Library website [43].

main contribution to $\mathbf{p}_q$'s FPFH, $SPF(\mathbf{p}_q)$. The histograms of first-order k-neighbors are summed based on a weight ω_k that is based on the distance between the points [47]:

$$FPFH(\mathbf{p}_q) = SPF(\mathbf{p}_q) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k}SPF(\mathbf{p}_k) \tag{2.15}$$

Rusu et al. note that this allows some pairs of points to be compared more than once because they appear in multiple k-neighborhoods. These repeated comparisons are thicker black lines in figure 2.7.

2.2.3.2 Viewpoint Feature Histograms (VFH) and extended FPFH (EFPFH)

Rusu et al. noted that both the Point Feature Histogram and Fast Point Feature Histogram are local features because they only describe the neighborhood around a single point [48]. The components of the fast point feature histogram, however, can be used in a way that creates a global signature describing the entire body of points. The Viewpoint Feature Histogram (VFH), published by Rusu et al. in [48], does this by combining two data structures based on Simple Point Features. The first structure is a viewpoint histogram that compares the orientation differences between the surface normal at a point and the vector from the viewer's location to that point (figure 2.8[7]).

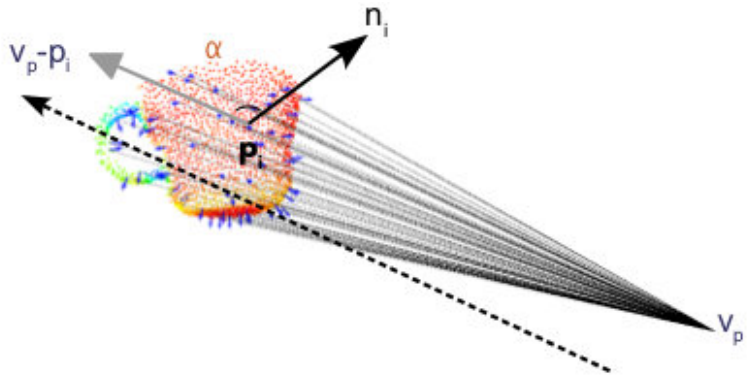

Figure 2.8: Viewpoint portion of the VFH. Diagram courtesy Radu B. Rusu [45]

This produces one feature value per point, α, the cosine of the angle between the two vectors γ:

$$\alpha = \frac{\mathbf{v}_p - \mathbf{p}_i}{|\mathbf{v}_p - \mathbf{p}_i|} \cdot \hat{\mathbf{n}}_i = \cos\gamma \tag{2.16}$$

[7] Figure 2.8 originally appeared in [48], but is licensed under Creative Commons Attribution 3.0 through the Point Cloud Library website [45].

Inspection of the source code file that implements the VFH [38] shows that the Point Cloud Library uses a variation that normalizes the value of α such that $0 \leq \alpha \leq 1$:

$$\alpha = \frac{1}{2}\left(\frac{\mathbf{v}_p - \mathbf{p}_i}{|\mathbf{v}_p - \mathbf{p}_i|} \cdot \hat{\mathbf{n}}_i + 1\right) = \frac{1}{2}\left(\cos\left(\gamma\right) + 1\right) \tag{2.17}$$

In the Point Cloud Library's 308 vector long VFH implementation, the values of α are binned into a 128 element, 32-bit floating point histogram [48].

The second part of the VFH is the Extended Fast Point Feature Histogram (EFPFH), which is a global version of the FPFH augmented with a distance-based feature. The key difference between the EFPFH and its predecessors is that the EFPFH compares each point with the centroid of the point cloud [48], while the FPFH and PFH compare each point cloud with its neighbors [48]. This difference is the key trait that makes the EFPFH a globally valid feature. Figure 2.9[8] shows a diagram of this comparison.

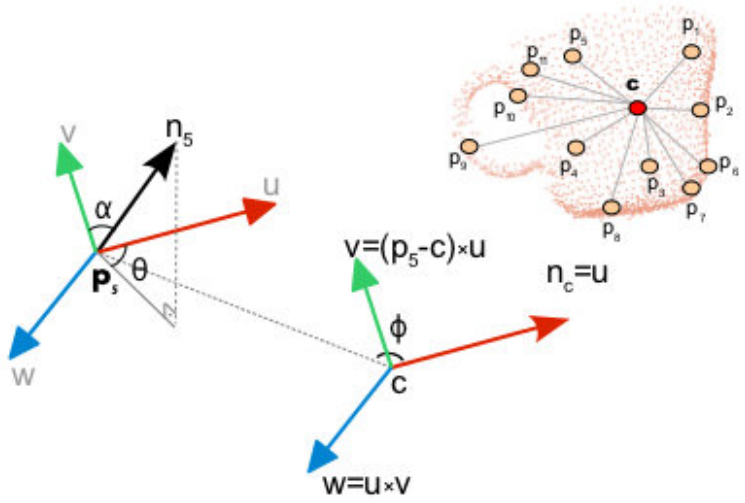

Figure 2.9: Extended FPFH portion of the VFH. Diagram courtesy Radu B. Rusu [45]

Each feature in the EFPFH is computed in a manner similar to the original point feature histogram, but with extra normalization steps [48, 38]. The distance between a point $\mathbf{p}_i$ and the centroid $\mathbf{c}$ (f_4) is normalized by the largest distance d_{max} between a point and the centroid. Angle cosine features f_2 and f_3 are normalized to produce values between 0 and 1, as in equation 2.17. The arctangent feature f_1 is

[8] Figure 2.9 originally appeared in [48], but is licensed under Creative Commons Attribution 3.0 through the Point Cloud Library website [45].

calculated in radians and normalized by π to have a magnitude between 0 and 1. A single comparison between a point $\mathbf{p}_i$ and the centroid $\mathbf{c}$ is the vector valued function $\mathbf{f}$ [9] :

$$\mathbf{f}\left(\mathbf{p}_i, \mathbf{c}\right) = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{\pi} \arctan\left(\frac{\hat{\mathbf{w}} \cdot \hat{\mathbf{n}}_i}{\hat{\mathbf{n}}_c \cdot \hat{\mathbf{n}}_i}\right) \\ \frac{1}{2}\left(\hat{\mathbf{v}} \cdot \hat{\mathbf{n}}_i + 1\right) \\ \frac{1}{2}\left(\frac{\mathbf{p}_i - \mathbf{c}}{|\mathbf{p}_i - \mathbf{c}|} \cdot \hat{\mathbf{n}}_c + 1\right) \\ \frac{1}{d_{max}} |\mathbf{p}_i - \mathbf{c}| \end{bmatrix} \tag{2.18}$$

The four elements of each feature are added to a 45 bin, 32-bit floating point histogram, for a total of 180 EFPFH elements. The viewpoint histogram and EFPFH are combined into a single, 308 element, 32-bit floating point VFH vector:

$$\text{VFH} = \begin{bmatrix} Viewpoint\ histogram \\ \\ EFPFH \end{bmatrix} \tag{2.19}$$

2.2.4 The Simultaneous Localization and Mapping (SLAM) problem

Siegwart et al. describe the SLAM problem in terms of a robot that has been kidnapped and transported to some unknown location ('lost in space') [53]. To do meaningful work, the robot would most likely need to build a map of its new environment, locate itself within this map and locate its objectives on the map [53]. The robot would presumably need to do these things simultaneously and autonomously, hence the problem's name. This is extremely similar to the situation that an autonomous debris-mapping spacecraft would encounter; it does not know the complete shape of some newly encountered orbital debris, nor does it know its location relative to the new object's center of mass.

Full 6-DOF motion SLAM in unfamiliar environments is still an area of active research, but the problem is considered solved for 3-DOF planar motion in closed laboratory environments [26]. Extended Kalman Filter (EKF) estimation was an early, popular strategy for the 3-DOF planar case [53]. In the Kalman filter approach, the state vector is initially declared to be the robot's state vector, relative to some

[9] The hat ˆ is used to denote a unit vector in this equation, not the best estimate of a vector.

arbitrarily chosen origin, like the robot's location at filter initialization time:

$$\mathbf{X}_{\text{world}} = [\mathbf{x}_{\text{robot}}] = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2.20}$$

As the robot moves, it updates its estimate of its current location using odometry [53]:

$$\mathbf{X}_{\text{world},k} = \mathbf{x}_{\text{robot},k-1} + \int_{t_{k-1}}^{t_k} \dot{\mathbf{x}}_{\text{robot},k} dt \tag{2.21}$$

The robot is also simultaneously searching for landmarks within its environment. When the robot identifies a landmark, it redefines the state vector to include the location and orientation vector $\mathbf{m}_i$ of the landmark [53]:

$$\mathbf{X}_{\text{world}} \equiv \begin{bmatrix} \mathbf{x}_{\text{robot}} \\ \mathbf{m}_0 \\ \vdots \\ \mathbf{m}_n \end{bmatrix} \tag{2.22}$$

At this point, the problem is a conventional Bayesian estimation problem, but odometry drift makes it non-trivial. Odometry has error that causes the robot's estimate of how far it has traveled to drift away from the true value [37, 53]. With terrestrial robots, rotary encoders can tell the robot how far it has turned its wheels [53], but what if the wheel slips? If the wheel slips, some error ϵ is added the robot's estimate of angular position of the wheel $\hat{\theta}_w$. The distance traveled d becomes an estimation problem because $d = r\left(\hat{\theta}_w - \epsilon\right)$ and ϵ is a time varying quantity that cannot be directly measured. This problem is called "odometer drift" [53]. Odometer drift is worse when it is based on acceleration or velocity sensor output because the error ϵ is integrated (or double integrated) as part of the pose estimate.

2.2.5 Lu-Milios Graph SLAM

Lu and Milios recognized that pure odometry pose estimation and the model building derived from it have a flaw; that each alignment is only locally consistent with an estimate of the robot's state at the time of measurement, permitting alignments that conflict with other alignments [29]. To build consistent maps,

the scans should be aligned in a way that is globally consistent - each alignment should be consistent with all other alignments and the state of the robot at all other observation times [29].

Lu and Milios achieved this by creating a framework that connects related point clouds like graph nodes in a network [29]. Each edge between node is treated as a spring whose length has been displaced from its rest state by a distance of $d_k = \ell - \ell_0$[10] - the residual error from the Iterative Closest Points alignment of the two frames (equation 2.6) [29]. The graph edges are treated as ideal Hooke's Law springs [29], which are energy storage devices that store potential energy u where [59]

$$u = \frac{1}{2}\kappa\left(\ell - \ell_0\right)^2 = \frac{1}{2}\kappa d_k^2 \tag{2.23}$$

The total energy state of the spring network can be used to create a new global cost function J where

$$J = \frac{1}{2}\sum_{i=1}^{n}\kappa_i d_{k,i}^2 \tag{2.24}$$

Lu and Milios stated the spring energy equation slightly differently. They used a matrix form to allow for simultaneous estimation of a robot state vector D as a scalar Mahalanobis distance W_{ab}:

$$W_{ab} = \left(\overline{D}' - D'\right)^T C'^{-1}\left(\overline{D}' - D'\right) \tag{2.25}$$

where W_{ab} is the energy-like term in the cost function, $\overline{D}$ is an odometry measurement of the robot's current state (pose), D is the robot's true pose, and C is the covariance matrix for the odometry based state estimate.

2.2.6 Explicit Loop Closure Heuristics (ELCH)

The probability function that determines that two frames overlap is called an explicit loop closure heuristic [53, 37]. Nüchter et al. based the autonomous Kurt3D robot's loop closure heuristic on the number of key points that could be identified in both of the separate point clouds (number of correspondences). Loop closure was declared in terms of a set of point clouds that overlap the most recently acquired point cloud. When a frame leaves that set of overlapping clouds, then returns to that set of overlapping clouds, loop closure is declared [37]. Once loop closure has been detected, the misalignment error caused by odometer drift was distributed equally amongst all frames by solving for a set of transformations that globally minimizes the

[10] Nomenclature changed from Lu and Milios' original paper for consistency with other literature reviewed in this section.

residual ICP alignment error d_k between each frame and other frames that overlap with it [37], similar to the spring network method presented by Lu and Milios in [29].

Nüchter et al. also modified the weight of each individual graph edge to reflect the fractional contribution of the odometer distance traveled between the nodes toward the total odometer distance traveled [37] [11] :

$$J = \frac{1}{\int_0^{t_n} \dot{\theta} d\theta} \sum_{i=1}^{n} \left(d_{k,i}^2 \int_{t_{i-1}}^{t_i} \dot{\theta}_i d\theta \right) \tag{2.26}$$

The explicit loop closure heuristic problem is similar to problems encountered in air traffic control tracking systems. These systems must decide if an object detected in one radar scan corresponds to an object detected in previous scans. The ELCH problem could be rephrased in almost identical terms: does the currently observed set of points correspond to a set of points that the spacecraft has previously observed? This similarity of the problems suggests that the probabilistic method presented by Mori, Chang and Chong in [32] may be applicable.

Radar target tracking has also produced frameworks for probabilistic decision making based on multiple sensor input, such as the multiple sensor decision gate presented by Dana in [11]. Dana defines the probabilistic decision gate function $G_{transition}$ [12] as the probability that the flight dynamics of an aircraft have changed and that the sensor should switch to a different model. In [11], Dana recommends choosing one sensor over the others based on the outcome of a Chi-squared test for the validity ξ of that sensor's decision gate probability function $G_{transition}$ [13] :

$$\xi = \left[G\left(\hat{\mathbf{X}}_i\right) - \mathbf{Y}_i \right]^T \left[\hat{P}_i - R_i \right] \left[G\left(\hat{\mathbf{X}}_i\right) - \mathbf{Y}_i \right] < G_{transition} \tag{2.27}$$

Dana's probabilistic approach is relevant because a probabilistic loop closure heuristic returns a probability that the hypothesis "frame v closes a loop that starts with frame u" is valid. Dana's $G_{transition}$ is the probability that the hypothesis "the aircraft has started a maneuver and the current dynamics model $\dot{\mathbf{X}} = F(\mathbf{X})$ should be replaced with the maneuver model $\dot{\mathbf{X}} = F_{maneuver}(\mathbf{X})$" is valid. While the hypotheses tested are quite different, the overall, higher-level task is the same: determine whether one hypothesis is

[11] Notation altered from [37] for consistency with other portions of this thesis
[12] Notation altered from [11] to deconflict with Tapley, Schutz and Born [55] notation.
[13] Notation altered from [11] to deconflict with Tapley, Schutz and Born [55] notation.

more valid than another based on the outputs of multiple observers' probability functions.

Dana's approach may require modification for loop closure heuristics because it is based on an assumption that the different sensors have a shared uncertainty specified by $\hat{P}_i$ and a unique, sensor-specific uncertainty R_i[14] . For loop closure heuristics, the different heuristics have the same $\hat{P}_i$ and R_i because the heuristics are calculated using the same observation data, but they each have some other unique error ϵ.

2.3 Adapting Graph SLAM for the uncooperative model generation problem

As noted by Dr. Junkins and Dr. Majii, [22] the conventional 6-DOF SLAM problem and the uncooperatively rotating object model generation problem are both situations in which the observer must construct a spatial map of previously unobserved object(s). The biggest difference is that the SLAM problem tracks information about the observer's absolute position over time, while the debris model generation problem does not. Another difference is that SLAM builds a model of the entire surrounding environment, while model generation is only concerned with a small subset of the environment (the debris object). Model generation is a subset of SLAM, which makes SLAM solution techniques particularly appropriate for uncooperatively rotating debris model generation.

These problems are so similar that the relative motion of the observer and orbital debris is governed by the same differential equations as that of a robot exploring its environment if the observer traces a circle while staring at the debris (figure 2.10).

[14] Notation modified: Dana used $\hat{\sum}_P$ and $\sum_Z$ for $\hat{P}_i$ and R_i, respectively [11].

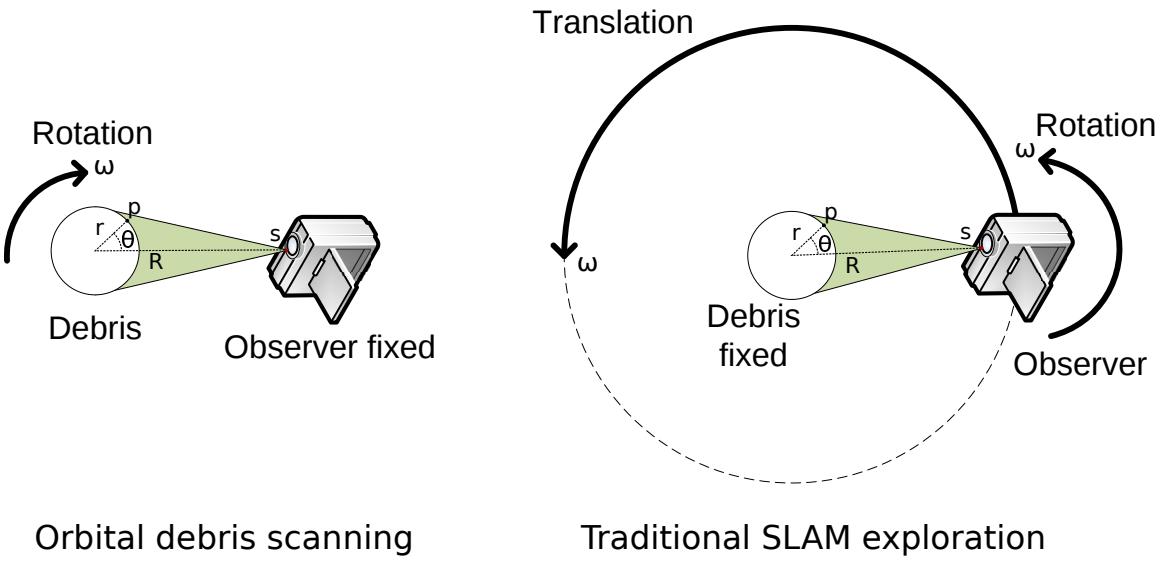

Figure 2.10: Relative motion in SLAM exploration can be diffeomorphic with debris model generation

The Lu-Milios Graph SLAM version of this problem would establish a spring network connecting the observations with the spacecraft's state at each observation (figure 2.11). Because model generation is only concerned with the locations of the landmarks on the observed object over time, the spacecraft state vectors $\mathbf{x}_i$ can be replaced with the object's orientation relative to the observer.

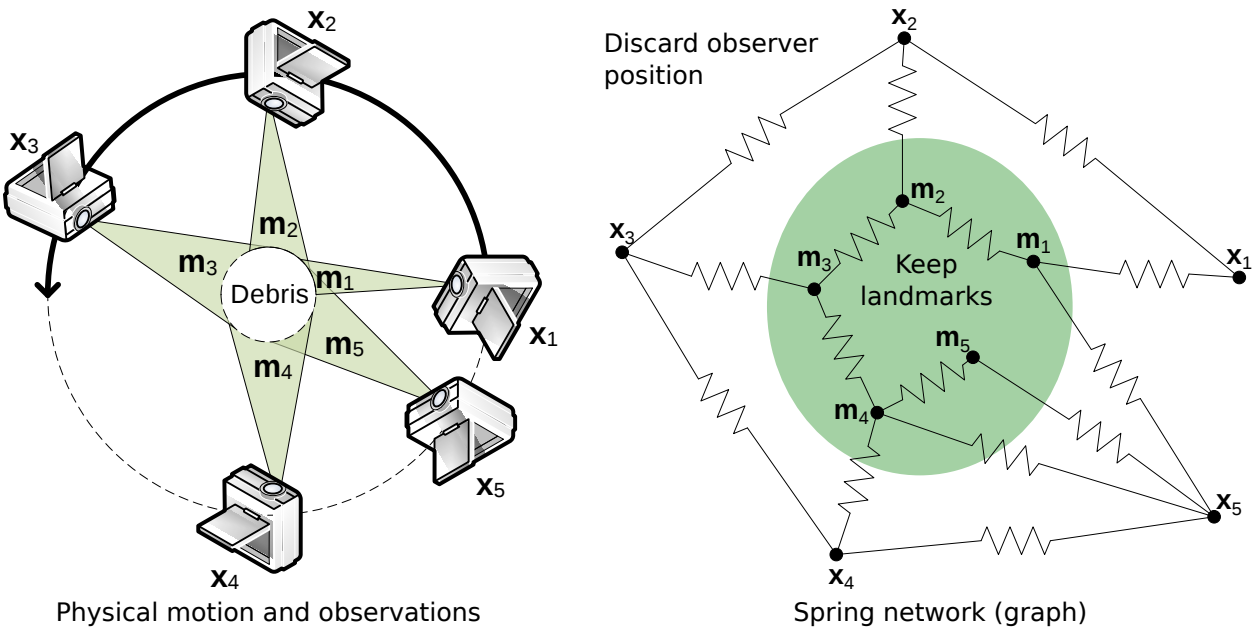

Figure 2.11: Lu-Milios Graph SLAM applied to orbital debris model generation

Closures are the critical feature to detect. Without loop closures, the model is just a string of frames (figure 2.12). If there are no closures, the algorithm may converge on an alignment that does not align the two halves of the same object. Note the separate of the left and right haves of the fuselage, which should line up on nose of the shuttle's external tank and the tail fin.

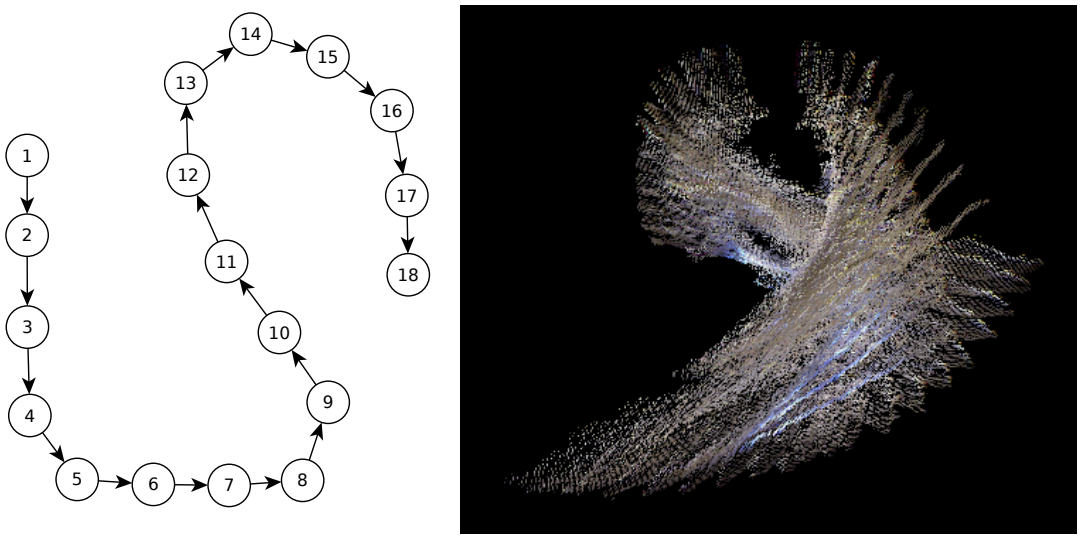

Figure 2.12: An intermediate Graph SLAM alignment without loop closure

Two point cloud data frames a and b observed by the sensor should be connected by an edge in the Graph SLAM model network if:

- They are consecutive frames that are not separated by an extended period without sensor data

$$(t_b - t_a \leq \Delta t_{max}) \wedge (b - a = 1) \wedge (a \geq 0) \tag{2.28}$$

- Some probability function determines that frames a and b overlap:

$$P_{Closure}\,(a, b) \geq P_{Closure,min} \tag{2.29}$$

2.4 Homogeneous transformations

The Iterative Closest Point alignment class in the Point Cloud Library uses a data structure equivalent to a homogeneous transformation to represent the change in translation and rotation between two frames. The alignment produces an affine 4×4 transformation matrix H_n^{n-1} that transforms points from the current frame n into the previous frame $n-1$, where

$$ {}^{n-1}H_n = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^{n-1}R_n & {}^{n-1}\mathbf{t}_n \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.30} $$

Schaub [51] notes that successive homogeneous transformations can be combined by multiplying them:

$$ {}^{n-2}H_n = \left[{}^{n-2}H_{n-1}\right]\left[{}^{n-1}H_n\right] = \begin{bmatrix} \left[{}^{n-2}R_{n-1}\right]\left[{}^{n-1}R_n\right] & \left(\left[{}^{n-2}R_{n-1}\right]{}^{n-1}\mathbf{t}_n + {}^{n-2}\mathbf{t}_{n-1}\right) \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.31} $$

One of the early attempts at the automated model generation in the pclAutoScanner program used equation 2.31 to keep a running homogeneous transform H_n^0 that transforms points from the current frame into the first frame that was processed, where

$$ {}^{0}H_n = \left[{}^{0}H_{n-1}\right]\left[{}^{n-1}H_n\right] \tag{2.32} $$

This is mathematically valid but problematic to implement. The Point Cloud Library uses floating point arithmetic for the homogeneous transform multiplication. Floating point arithmetic usually involves truncating a number's true value to the closest fraction that can be represented within the IEEE-754 floating point standard. This manifests as error that is added each time an arithmetic operation is performed on a number that contributes to the homogeneous transform.

Equation 2.31 is numerically integrating an orientation difference and a rotating translation vector. This means that any error is also integrated, causing the best estimate of the homogeneous transform from the current frame to the original frame to drift from the value that it should have. Figure 2.13 shows the results of accumulating successive homogeneous transforms to build a model. The drift is significant enough to prevent an autonomous 3D model building/scanning system from building a correct model.

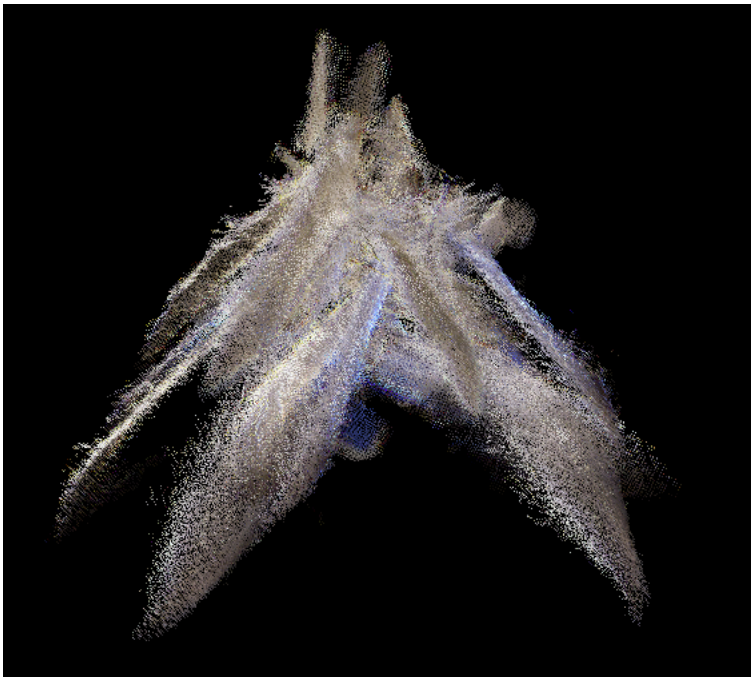

Figure 2.13: Incorrect frame alignment caused by combining successive homogeneous transforms from ICP

This effect is a manifestation of the odometer drift described in section 2.2.4. There are other numerical difficulties hindering the homogeneous transformation accumulation approach. One is that multiplication of two homogeneous transform produces small nonzero terms in the bottom left part of the homogeneous transform, which Bajd et al. refer to as the "perspective transformation" [3]:

$$ {}^{0}H_1 = \left[{}^{0}H_0\right]\left[{}^{0}H_1\right] = \left[I_{4\times4}\right]\left[{}^{0}H_1\right] = \begin{bmatrix} & {}^{0}R_1 & & {}^{0}\mathbf{t}_1 \\ \begin{bmatrix} \mathcal{O}\left(10^{-44}\right) & \mathcal{O}\left(10^{-44}\right) & \mathcal{O}\left(10^{-44}\right) \end{bmatrix} & & & 1 \pm \mathcal{O}\left(10^{-44}\right) \end{bmatrix} \tag{2.33} $$

This manifested experimentally as deformation of the source point clouds. They were visibly stretched along two axes, and compressed along the third. The distortion becomes more exaggerated as more frames are processed. After only four transformations, the 3D point cloud for the most recently processed visible fragment of a rectangular prism appeared to be a flattened projection of the 3D point cloud onto a 2D plane.

2.5 Expected relative motion in Earth orbit

Relative motion in orbit is a significant topic in and of itself. To keep the scope of this thesis manageable, only short range relative motion of two objects in circular orbits will be examined. The Clohessy-Wiltshire/Hill equations are circular orbit simplifications of the two-body problem for short-range motion of two small bodies relative to each other as they orbit a third, much more massive body [56]. Vallado [56] presents the CW/Hill equations as functions of time:

$$x(t) = \frac{\dot{x}_0}{\omega}\sin(\omega t) - \left(3x_0 + \frac{2\dot{y}_0}{\omega}\right)\cos(\omega t) + \left(4x_0 + \frac{2\dot{y}_0}{\omega}\right) \tag{2.34}$$

$$y(t) = \left(6x_0 + \frac{4\dot{y}_0}{\omega}\right)\sin(\omega t) + \frac{2\dot{x}_0}{\omega}\cos(\omega t) - (6\omega x_0 + 3\dot{y}_0)\,t + y_0 - \frac{2\dot{x}_0}{\omega} \tag{2.35}$$

$$z(t) = z_0\cos(\omega t) + \frac{\dot{z}_0}{\omega}\sin(\omega t) \tag{2.36}$$

where x, y and z define the radial, in-track, and orbit normal relative separation distance of the two small objects, respectively, and ω is the mean motion of the orbit around the plane [56].

All three of these equations include trigonometric terms, suggesting that some motion on all three axes will be periodic. Two axes of the relative motion are coupled (x and y). Equation 2.35 can also have a secular term, $(6\omega x_0 + 3\dot{y}_0)\,t$. If both $x_0 = \dot{y}_0 = 0$, then the relative motion will trace out a ellipse. If either x_0 or $\dot{y}_0$ are nonzero, the two bodies drift and the ellipse becomes a helix or corkscrew in three dimensions. Figure 2.14 shows a qualitative plot of the predicted relative motion.

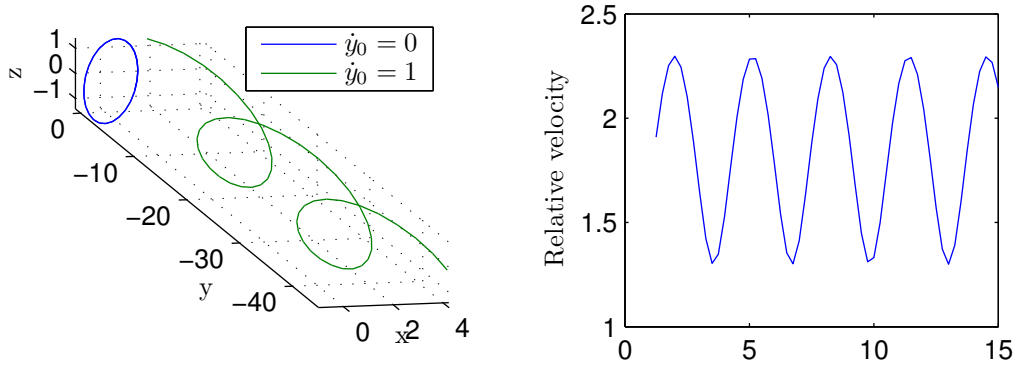

Figure 2.14: Qualitative (unitless) relative motion predicted by Clohessy-Wiltshire/Hill equations

2.6 Dynamics model for 1DOF rotation about $+y$ axis

Both the batch processor used in the purely least-squares method and the EKF used by the loop closure heuristic require a model of the rotating object's dynamics. They will both use the dynamics model derived in this section. This model is only valid for this preliminary laboratory experiments in section 3.1.

2.6.1 Simplifications for a preliminary lab-based experiment

The simplified experiment constrains the angular rotation to one axis using a magnetic platform and a Levitron$^{\text{TM}}$ magnetic levitation platform (figure 2.15). The rotation rate about the axis is controlled by the circuitry of the Levitron$^{\text{TM}}$ platform. This experiment ignores those dynamics and assumes no prior knowledge of the rotation rate. This model assumes that:

- The center of mass of the rotating object does not move ($\dot{\mathbf{r}}_D = \ddot{\mathbf{r}}_D = \vec{0}$)
- The rotating object is a rigid body rotating about its principle axis
- The sensor frame is inertial because the observer does not move or rotate ($\dot{\mathbf{r}}_S = \ddot{\mathbf{r}}_S = \vec{0},\ \dot{\vec{\beta}}_S = \vec{\omega}_S = \vec{0}$)
- Transient effects have dissipated and the Levitron$^{\text{TM}}$-magnet system is in steady state ($\dot{\vec{\omega}}_D = 0$)
- The Levitron$^{\text{TM}}$-magnet system provides torque free motion ($\mathbf{L}_D = \vec{0}$)

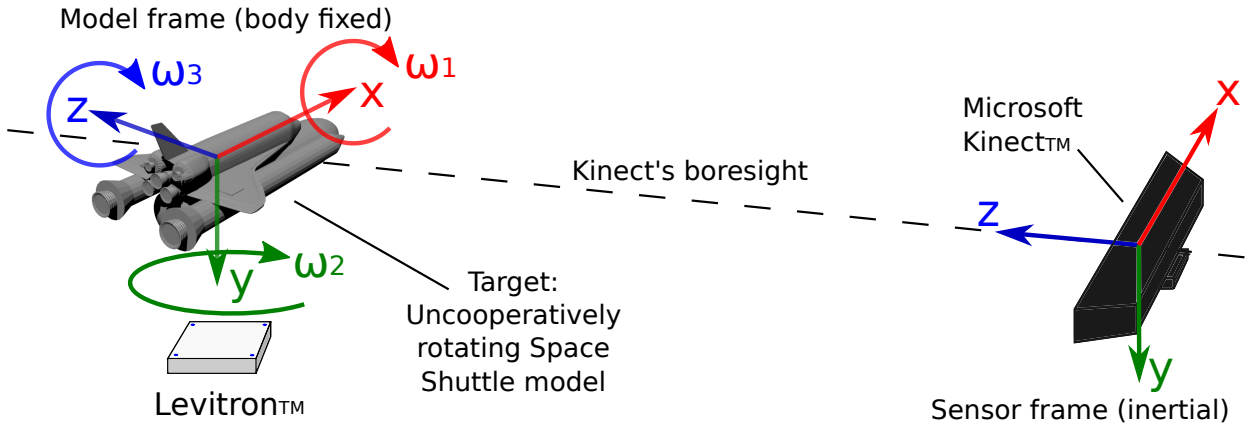

Figure 2.15: Bench-top Levitron$^{\text{TM}}$, fixed sensor experiment diagram. 3D Shuttle model courtesy NASA [7].

2.6.2 State vector and equations of motion

Estimating the dynamics of a single body-fixed point on the surface of a rotating object under torque free motion requires knowledge of the object's center of mass, mass distribution and shape. Because the sensor is encountering the orbital debris for the first time, it is assumed that this knowledge is unavailable. This limits the state vector $\mathbf{X}$ to quantities that can be observed from relative angular changes between observations of the rotating orbital debris, where β_i are quaternion vector elements, ω_i are angular velocity vector components, $\hat{\mathbf{e}} = \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix}^T$ is the principal rotation axis and Φ is the principal rotation angle:

$$\mathbf{X} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} \cos\frac{\Phi}{2} \\ e_1 \sin\frac{\Phi}{2} \\ e_2 \sin\frac{\Phi}{2} \\ e_3 \sin\frac{\Phi}{2} \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{2.37}$$

Reference [51] presents a matrix relationship between the first derivative of the Euler parameter vector $\vec{\beta}$ and the angular rate vector $\vec{\omega}$, with both vectors defined in the body-fixed model frame $\mathcal{B}$:

$$\begin{bmatrix} \dot{\beta}_0 \\ \dot{\beta}_1 \\ \dot{\beta}_2 \\ \dot{\beta}_3 \end{bmatrix}_{\mathcal{B}} = \frac{1}{2} \begin{bmatrix} \beta_0 & -\beta_1 & -\beta_2 & -\beta_3 \\ \beta_1 & \beta_0 & -\beta_3 & \beta_2 \\ \beta_2 & \beta_3 & \beta_0 & -\beta_1 \\ \beta_3 & -\beta_2 & \beta_1 & \beta_0 \end{bmatrix} \begin{bmatrix} 0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}_{\mathcal{B}} \tag{2.38}$$

Removing the first element of the augmented angular rate vector for consistency with the state vector (equation 2.37),

$$\begin{bmatrix} \dot{\beta}_0 \\ \dot{\beta}_1 \\ \dot{\beta}_2 \\ \dot{\beta}_3 \end{bmatrix}_{\mathcal{B}} = \frac{1}{2} \begin{bmatrix} -\beta_1 & -\beta_2 & -\beta_3 \\ \beta_0 & -\beta_3 & \beta_2 \\ \beta_3 & \beta_0 & -\beta_1 \\ -\beta_2 & \beta_1 & \beta_0 \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}_{\mathcal{B}} \tag{2.39}$$

Assuming that no external torques are acting on the rotation object, $\dot{\vec{\omega}} = \mathbf{0}$. Combining these yields a vector-valued function $F(\mathbf{x})$ that returns the first derivative of the state vector:

$$F(\mathbf{X}) = \dot{\mathbf{X}} = \begin{bmatrix} \dot{\beta}_0 \\ \dot{\beta}_1 \\ \dot{\beta}_2 \\ \dot{\beta}_3 \\ \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{\omega}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & 0 & 0 & 0 & -\beta_1 & -\beta_2 & -\beta_3 \\ 0 & 0 & 0 & 0 & \beta_0 & -\beta_3 & \beta_2 \\ 0 & 0 & 0 & 0 & \beta_3 & \beta_0 & -\beta_1 \\ 0 & 0 & 0 & 0 & -\beta_2 & \beta_1 & \beta_0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{2.40}$$

2.7 Offline attitude estimation using the Batch Processor

The relative attitude of the non-cooperatively rotating object can be estimated several ways. This thesis uses two methods: the batch processor and the Extended Kalman Filter, both in an offline, post-processing configuration. The two estimators are both least squares estimators that are derived from the same estimation problem [55]. Because of the shared theoretical underpinning, the batch processor theory will be presented first and later expanded for the Extended Kalman Filter (EKF).

The batch processor is a Bayesian estimator that attempts to find a best estimate $\hat{\mathbf{X}}$ of the system's state vector given a model of the system dynamics, some *a priori* estimate of the state $\overline{\mathbf{X}}$ and imperfect observations of parts of the system $\mathbf{Y}$. It is assumed that the observations $\mathbf{Y}$ contain some error that is modeled as a function of a random variable.

The general strategy of this purely least-squares (odometry-based) model construction approach is to:

(1) Estimate the object's orientation at each frame n using the batch processor

(2) Locate outlier translation vectors ${}^{n-1}\mathbf{t}_n$.

(3) Replace outlier translation vectors ${}^{n-1}\mathbf{t}_n$ with linear interpolation of ${}^{n-1}\mathbf{t}_{n+1}$ to frame n.

(4) Incrementally rebuild the homogeneous transforms using the best estimates of R and $\mathbf{t}$ between frames

(5) Transform each frame's point cloud to the model frame

(6) Merge the component point clouds and downsample to make a 3D model

2.7.1 Definition of the observation Y

An early (unsuccessful) attempt at model generation incrementally built the model, aligning each frame to a previous model, then merging the new frame with the prior model. This exposed a limitation of the Iterative Closest Points algorithm; that it is only as good as the correspondences it uses for its alignments. As Nüchter et al. argued in [37], Besl and McKay's proof of ICP convergence on a global minimum of the cost function in [5] does not guarantee that ICP will converge on the best alignment between frames if an artificial correspondence distance constraint prevents ICP from operating on all points in cloud P. Figure 2.16 shows an example of an Iterative Closest Points misalignment caused by allowing the Point Cloud Library's ICP algorithm to iterate based on too few correspondences between the model and the scene [15] . Drastic misalignments occurred less frequently when the Iterative Closest Points algorithm compared two successive frames, presumably because the clouds covered overlapping regions of the object, which provides the ICP algorithm with a larger body of point-to-point correspondences.

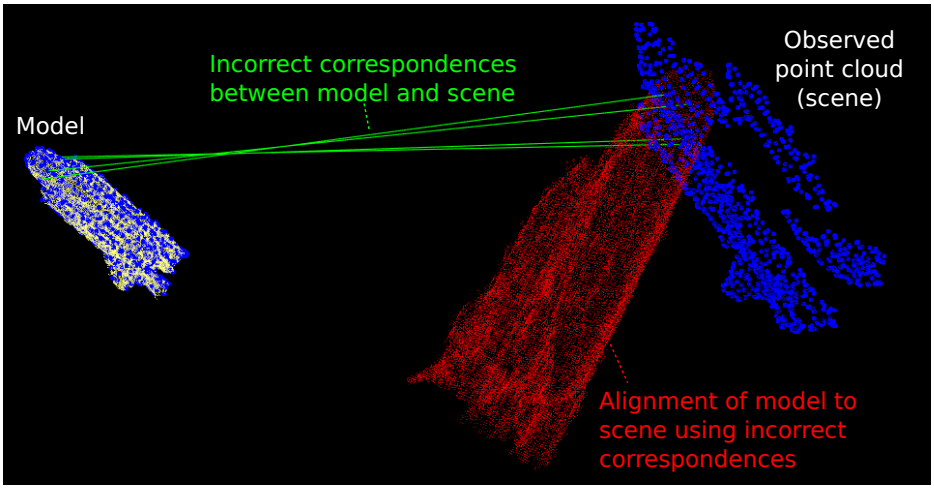

Figure 2.16: Drastic misalignment caused by incorrect correspondence estimation

[15] The correspondences in figure 2.16 were estimated using the geometric consistency estimator and they are indeed geometrically consistent. The problem is that the tank's nose is a patched parabolic cone. Identical conic sections can be obtained by slicing 90° apart at several locations along the nosecone, which makes geometric consistency correspondence estimation too ambiguous for use with nosecones/fairings.

Several outputs of the iterative closest points algorithm could be used as observations. Angular velocity was selected as the observation because it can be approximated using a first difference of only two consecutive frames, which limits the metric to the cases where Iterative Closest Points performs the best. This is expected to have a negative side effect: excessive noise in the estimate caused by the use of a first difference derivative approximation. This will be addressed later with estimation and filtering.

While Iterative Closest Points does not directly compute angular rates, these rates can be derived from the time interval between two frames and their relative orientation quaternion (Euler parameter vector). The Point Cloud Library is used in this implementation to align each frame with the previous frame under the assumption that the object has rotated only a small amount between the two frames. It returns a homogeneous transform from frame n to frame $n-1$ of the form

$$ {}^{n-1}H_n = \begin{bmatrix} {}^{n-1}R_n & {}^{n-1}\mathbf{t}_n \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{2.41} $$

The Eigen3 library can be used to convert the rotation matrix ${}^{n-1}R_n$ into an Euler parameter vector (quaternion) ${}^{n-1}\vec{\beta}_n$ representing the alignment of frame $n-1$ relative to frame n. Negating the first member of the quaternion ${}^{n-1}\vec{\beta}_n$ produces the inverse quaternion ${}^{n}\vec{\beta}_{n-1}$:

$$ {}^{n}\vec{\beta}_{n-1} = {}^{n-1}\begin{bmatrix} -\beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}_n \tag{2.42} $$

A first difference scheme can be used to calculate the average angular rate ${}^{n}\vec{\omega}_{n-1}$ between frames $n-1$ and n, starting with the definition of the Euler parameter vector:

$$ \vec{\beta} \equiv \begin{bmatrix} \cos\frac{\Phi}{2} \\ e_1 \sin\frac{\Phi}{2} \\ e_2 \sin\frac{\Phi}{2} \\ e_3 \sin\frac{\Phi}{2} \end{bmatrix} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \tag{2.43} $$

This can be decomposed into the principle rotation axis vector $\hat{\mathbf{e}}$ and the principle rotation angle Φ. While this decomposition is ambiguous (a rotation of $-\Phi$ about axis $-\hat{\mathbf{e}}$ is the same as a rotation of Φ about

axis $\hat{\mathbf{e}}$), this is not a problem, because we are solving for

$$ {}^{n}\vec{\omega}_{n-1} = \dot{\Phi}\hat{\mathbf{e}} \approx \frac{\Phi}{t_n - t_{n-1}}\hat{\mathbf{e}} \tag{2.44} $$

The multiplication of Φ and $\hat{e}$ causes the sign ambiguity to cancel out. Solving for $\sin\frac{\Phi}{2}$,

$$ \cos\frac{\Phi}{2} = \beta_0 $$
$$ \frac{\Phi}{2} = \cos^{-1}\beta_0 $$

$$ \sin\frac{\Phi}{2} = \sin\left(\cos^{-1}\beta_0\right) \tag{2.45} $$

Substituting into the bottom 3 members of the Euler parameter vector,

$$ \begin{bmatrix} e_1 \sin\frac{\Phi}{2} \\ e_2 \sin\frac{\Phi}{2} \\ e_3 \sin\frac{\Phi}{2} \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} $$
$$ \sin\left(\frac{\Phi}{2}\right)\hat{\mathbf{e}} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} $$
$$ \hat{\mathbf{e}} = \frac{1}{\sin\left(\frac{\Phi}{2}\right)} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} $$

$$ \hat{\mathbf{e}} = \frac{1}{\sin\left(\cos^{-1}\beta_0\right)} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \tag{2.46} $$

The principle rotation angle Φ can be obtained from the definition of the scalar element β_0 of the quaternion $\vec{\beta}$:

$$ \cos\frac{\Phi}{2} = \beta_0 \Rightarrow \Phi = 2\cos^{-1}\beta_0 \tag{2.47} $$

Reassembling these into an average angular rate vector between frames $n-1$ and n,

$$ {}^{n-1}\vec{\omega}_n = \dot{\Phi}\hat{e} \approx {}^{n-1}\left(\frac{2\cos^{-1}\beta_0}{(t_n - t_{n-1})\sin\left(\cos^{-1}\beta_0\right)} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \right)_n $$

Other operations are more convenient if we compare backwards. Negating the scalar component of the quaternion to reverse the rotation,

$$ {}^{n}\vec{\omega}_{n-1} \approx {}^{n}\left(\frac{2\cos^{-1}\left(-\beta_0\right)}{(t_n - t_{n-1})\sin\left(\cos^{-1}\left(-\beta_0\right)\right)} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \right)_{n-1} \approx \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}_n \tag{2.48} $$

The average angular rotation vector ${}^{n}\vec{\omega}_{n-1}$ will be the observation vector **Y:**

$$ \mathbf{Y} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}_n \tag{2.49} $$

2.7.2 Predicted observation function G

The observation function G that relates the state vector to the observation is

$$ G(\mathbf{X}) = \begin{bmatrix} 0_{3\times4} & I_{3\times3} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} 0_{3\times4} & I_{3\times3} \end{bmatrix} \mathbf{X} \tag{2.50} $$

which defines the $\tilde{H}$ matrix as

$$ \tilde{H} = \frac{\partial G}{\partial \mathbf{X}} = \begin{bmatrix} 0_{3\times4} & I_{3\times3} \end{bmatrix} \tag{2.51} $$

2.7.3 Observation weighting

Weighting of the individual measurements will be based on the "fitness" score d_k (equation 2.6), which is the residual mean squares distance between the corresponding points in each frame's point cloud after the ICP algorithm converges. A small fitness means a good alignment, so the observation weight matrix W_i for observation i will use the inverse of the fitness score to give more weight to good alignments:

$$W_i = d_{k,i}^{-1} \left[I_{3x3} \right] \tag{2.52}$$

2.7.4 State transition matrix $\mathbf{\Phi}(t, t_o)$

Under the assumption of torque free rotation about the body-frame $-y$-axis,

$$\hat{\mathbf{e}}_{\text{expected}} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}_{\text{body}} \tag{2.53}$$

This means that the angular rates should all be negative and all concentrated on the 2-axis (body y-axis), or that

$$\vec{\omega}_{\text{expected}} = \begin{bmatrix} 0 \\ \omega_2 \\ 0 \end{bmatrix} \Longrightarrow \dot{\Phi} = -\omega_2 \tag{2.54}$$

This yields an expected value for the state vector as a function of time:

$$\mathbf{X}(t) = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{1}{2}\int_0^t \dot{\Phi}(t)\,dt\right) \\ 0 \\ -\sin\left(\frac{1}{2}\int_0^t \dot{\Phi}(t)\,dt\right) \\ 0 \\ 0 \\ \omega_2 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\left(-\frac{1}{2}\int_0^t \omega_2 dt\right) \\ 0 \\ -\sin\left(-\frac{1}{2}\int_0^t \omega_2 dt\right) \\ 0 \\ 0 \\ \omega_2 \\ 0 \end{bmatrix} \tag{2.55}$$

The assumption of torque free rotation means that ω_2 is a constant that is determined by the initial state of the system. This allows the simplification that

$$\int_0^t \omega_2 dt = \omega_2 t \tag{2.56}$$

Because sine is an odd function, $\sin(-x) = -\sin(x)$. Combining these two simplifications,

$$\mathbf{X}(t) = \begin{bmatrix} \cos\left(-\frac{1}{2}\omega_2 t\right) \\ 0 \\ \sin\left(\frac{1}{2}\omega_2 t\right) \\ 0 \\ 0 \\ \omega_2 \\ 0 \end{bmatrix} \tag{2.57}$$

Following Nüchter et al.'s convention of assigning the first frame to be a "master frame" that establishes the model frame's coordinate system[37] and applying the torque-free, constant angular velocity assumption yields an initial system state of

$$\mathbf{X}(t_0) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \omega_2 \\ 0 \end{bmatrix} \tag{2.58}$$

The state transition matrix for this system, as defined by Tapley, Schutz and Born [55] is

$$\mathbf{\Phi}\left(t,t_0\right)=\frac{\partial\mathbf{X}(t)}{\partial\mathbf{X}(t_0)}=\begin{bmatrix}1&0&0&0&0&-\frac{1}{2}t\sin\left(\frac{1}{2}\omega_2 t\right)&0\\0&1&0&0&0&0&0\\0&0&1&0&0&\frac{1}{2}t\cos\left(\frac{1}{2}\omega_2 t\right)&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\end{bmatrix} \tag{2.59}$$

The diagonal elements of equation 2.59 have been set to 1 so that the state transition matrix correctly maps time t_0 back to itself [55], satisfying the requirement that

$$\mathbf{X}\left(t_0\right)=\mathbf{\Phi}\left(t_0,t_0\right)\mathbf{X}\left(t_0\right) \tag{2.60}$$

2.8 Relative attitude estimation with the Extended Kalman Filter (EKF)

The EKF implementation used in this thesis expands on the dynamics model derived in section 2.6 and the batch estimation approach defined in section 2.7. This section contains only the extensions to the previously presented model and estimation problem definition needed for the EKF.

2.8.1 Predicted state vector update

The incremental orientation change required to align two successive frames 1 and 2 using the iterative closest points algorithm can be used to produce a first difference approximation of the angular velocity between two successive frames:

$$\vec{\omega}_{\text{inertial}}\approx\frac{\Phi_{1\to2}}{t_2-t_1}\hat{\mathbf{e}}_{1\to2}\approx-\vec{\omega}_{\text{body}} \tag{2.61}$$

where $\hat{\mathbf{e}}_{1\to2}$ is the principle rotation axis about which frame 2 is rotated to align with frame 1, and $\Phi_{1\to2}$ is the principle rotation angle about the principle rotation axis to align frame 2 with frame 1. The difference t_2-t_1 is the time that has elapsed between frames 1 and 2. From [51], this can be used to create differential

equation describing the rate of change of the orientation of the inertial frame relative to the body:

$$\begin{bmatrix} \dot{\beta}_0 \\ \dot{\beta}_1 \\ \dot{\beta}_2 \\ \dot{\beta}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} \beta_0 & -\beta_1 & -\beta_2 & -\beta_3 \\ \beta_1 & \beta_0 & -\beta_3 & \beta_2 \\ \beta_2 & \beta_3 & \beta_0 & -\beta_1 \\ \beta_3 & -\beta_2 & \beta_1 & \beta_0 \end{bmatrix} \begin{bmatrix} 0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}_{\text{body}} \tag{2.62}$$

This can be integrated using the forward Euler method as long as the quaternion vector $\vec{\beta}$ is re normalized at each step to preserve its unit norm magnitude constraint [51]:

$$\vec{\beta}_{k+1} = \frac{1}{\left|\vec{\beta}_k + (t_{k+1} - t_k)\,\dot{\vec{\beta}}\right|} \vec{\beta}_k + (t_{k+1} - t_k)\,\dot{\vec{\beta}} \tag{2.63}$$

2.8.2 Observation uncertainty

The fitness score d_k (equation 2.6) from the Iterative Closest Points alignment is retained and used to create the observation uncertainty matrix R. It is applied equally to all three axes of the angular velocity vector $\vec{\omega}$:

$$R = d_k \left[I_{3\times 3}\right] \tag{2.64}$$

2.8.3 Process noise

If a Kalman filter is provided with a constant stream of measurements but no process noise matrix, the uncertainty of the filter's estimate will collapse, causing it to almost ignore any new observations that disagree with the filter's estimate of the system's state [55]. This leads to a condition called filter divergence, where the system's true state changes, but the filter does not track the change (it "diverges" from the ground truth) [55]. The process noise matrix Q is added to reduce Kalman filter divergence by artificially inflating the covariance matrix, which widens the filter's tracking envelope.

The formulation of the process noise matrix Q from [21] is used with a constant state noise magnitude

$\sigma_u^2 = 1^{rad}/_s$. The ICP alignment fitness is re-used as the observation variance $\sigma_v^2 = d_k$:

$$Q = \begin{bmatrix} \left(\sigma_v^2 \Delta t + \frac{1}{3}\sigma_u^2 \Delta t^3\right)[I_{4\times4}] & \frac{1}{2}\sigma_u^2 \Delta t^2 \begin{bmatrix} 0_{1\times3} \\ I_{3\times3} \end{bmatrix} \\ \frac{1}{2}\sigma_u^2 \Delta t^2 \begin{bmatrix} 0_{3\times1} & I_{3\times3} \end{bmatrix} & \sigma_u^2 \Delta t\,[I_{3\times3}] \end{bmatrix} \tag{2.65}$$

This differs slightly from the form presented in [21]. In Junkins & Crassidis' estimator, the state vector does not include the scalar component of the quaternion [21]. The process noise matrix in equation (2.65) was zero-padded in the extra rows and columns that affect the scalar component of the quaternion, which is also the first element of the state vector $\mathbf{X}$.

The choice of variables in the process noise matrix Q qualitatively describes the performance of the filter. The term Δt is the time difference between two frames, appearing as a cubic term in the attitude estimate (top left quadrant) and quadratic in the angular rate estimate. If there is a large gap in the data set, the Δt term will dominate, expanding the covariance matrix (increasing the uncertainty of the estimate).

The $\sigma_v^2 = d_k$ term affects only the orientation estimate, but it actually describes the accuracy of the most recent frame to frame alignment (the angular rate first difference approximation in the dynamics model). This penalizes the orientation estimate twice for the inaccuracy of the angular rate measurement, with the first penalty being measurement uncertainty in the matrix R. This is appropriate here for the same reason that it is appropriate when integrating a rate gyro [21]. If an angular rate measurement Y has an error ϵ, that is,

$$Y = \omega + \epsilon \tag{2.66}$$

and the orientation is calculated by integrating the observation Y, then

$$\hat{\theta} = \int_0^t Y\, d\tau = \int_0^t (\omega + \epsilon)\, d\tau = \omega t + \epsilon t \tag{2.67}$$

While this is a planar rotation simplification, the principle still applies to higher dimensional cases. The error ϵ is a bias for angular rates, but produces unbounded linear growth for the integral of angular rates. Penalizing the uncertainty of the orientation estimate twice is appropriate because the rate integration process exaggerates the effects of angular rate error on the orientation estimate.

2.9 Binary Extended Fast Point Feature Histogram (EFPFH)

The binary Extended Fast Point Feature Histogram (EFPFH) is introduced in this thesis as a quantization of the EFPFH to reduce memory consumption and processing time while estimating the similarity of two point clouds. It is used as part of the explicit loop closure heuristic (section 2.10.4). The binary EFPFH was motivated by the low computational cost of the Google/Stanford similarity hash [30]. At its core, the binary EFPFH is an information reduction algorithm, like the Point Feature Histogram (PFH) [50] and Google/Stanford similarity hash [30].

The binary EFPFH is a 180 bit feature vector quantized from the Extended Fast Point Feature Histogram (EFPFH) section of the viewpoint feature histogram (VPFH). This makes it a digest of the relative topography of the set of visible points Y_i in the current view of the object being scanned. Represented as a vector, the binary EFPFH $\mathbf{b}_i$ is a function of the object's shape, the object's current orientation $\vec{\beta}_i$, and random variables $\eta_{j,i}$ that represent errors in the measured positions of the visible points:

$$\mathbf{b}_i = \begin{bmatrix} b_{i,1} \\ b_{i,2} \\ \vdots \\ b_{i,180} \end{bmatrix} = f\left(Y_i\right) = f\left(shape, \vec{\beta}_i, \begin{bmatrix} \eta_{0,i} \\ \vdots \\ \eta_{m,i} \end{bmatrix} \right) \tag{2.68}$$

2.9.1 Assessing the similarity of two binary EFPFHs

The difference binary EFPFH $\mathbf{b}_{v-u}$ describes the bits that are different between binary EFPFHs of frames u and v. It is calculated by taking the bit-wise exclusive or of each corresponding bit in binary EFPFH bit vectors $\mathbf{b}_u$and $\mathbf{b}_v$:

$$\mathbf{b}_{v-u} = \begin{bmatrix} b_{v-u,1} \\ b_{v-u,2} \\ \vdots \\ b_{v-u,180} \end{bmatrix} = \begin{bmatrix} b_{u,1} \wedge \overline{b_{v,1}} \\ b_{u,2} \wedge \overline{b_{v,2}} \\ \vdots \\ b_{u,180} \wedge \overline{b_{v,180}} \end{bmatrix} \tag{2.69}$$

The cardinality (size) of the intersection of the features contained in frames u and v is the number of

bit positions in $\mathbf{b}_{v-u}$ that are 0:

$$|U \cap V| = |\{k : b_{v-u,k} = 0,\ 1 \leq k \leq 180\}| \tag{2.70}$$

The total number of distinct features contained in $U \cup V$ is the size of the bit vector $\mathbf{b}_{v-u}$ plus the number of bit positions in which $\mathbf{b}_u$ and $\mathbf{b}_v$ have different values:

$$|U \cup V| = |\mathbf{b}_{v-u}| + |\{k : b_{v-u,k} = 1,\ 1 \leq k \leq 180\}| \tag{2.71}$$

The Jaccard coefficient of similarity[16] J is used to describe the similarity of feature sets U and V [18, 15]. It is a fraction between 0 and 1 that describes fractional similarity in terms of a set union and intersection [15]. For the binary EFPFH, J is equation 2.70 divided by equation 2.71:

$$J = \frac{|U \cap V|}{|U \cup V|} = \frac{|\{k : b_{v-u,k} = 0,\ 1 \leq k \leq 180\}|}{|\mathbf{b}_{v-u}| + |\{m : b_{v-u,m} = 1,\ 1 \leq m \leq 180\}|},\ 0 \leq J \leq 1 \tag{2.72}$$

2.9.2 Complexity comparison of the EFPFH and binary EFPFH

The primary difference between the binary EFPFH and the EFPFH is that the EFPFH has higher precision, but requires 60 times more memory to store. The number of operations required to compare two binary EFPFHs is on the same order of magnitude as comparing two standard EFPFH, but a binary EFPFH comparison uses only bit-wise operations and unsigned integer addition. Standard EFPFH comparison uses 32-bit floating point addition and comparisons.

Table 2.2: Computational complexity of binary and conventional EFPFH over n comparisons

Binary EFPFH difference		**EFPFH histogram union**	
Operation	Θ	**Operation**	Θ
Bit-wise XOR	$3n$	Compare 32-bit floating point numbers	$180n$
Right shift XOR result	$180n$		
Add bit value to difference counter	$180n$	32-bit floating point addition: $\sum(U \cup V)$	$180n$
Calculate J (similarity coefficient)	$3n$		
Total	$366n$	**Total**	$360n$

[16] Jaccard used the term "community" [18]

2.9.3 Expected computation and memory cost

While the binary EFPFH and conventional (floating point) EFPFH are both $\mathcal{O}(n)$ complexity, the CPUs available at the time of writing compute them differently. The differences between CPUs are significant enough that this section cannot address all cases. Instead, this section analyzes the performance on an AMD Steamroller series microprocessor as a representative example. For an AMD Steamroller series microprocessor, the bit-wise XOR, right shift and integer addition each require one instruction and one clock cycle to compute [14]. The floating point division in equation 2.72 requires 1 operation and between 9 and 37 clock cycles to compute [14]. This gives a per-comparison execution time estimate of

$$t_{binary\ EFPFH} \approx 3 + 180 + 1 + 1 \times 37 = 221 \text{ cycles} \tag{2.73}$$

For the same microprocessor series, floating point addition requires 5 clock cycles [14]. A floating point (conventional) EFPFH comparison is expected to require almost five times as many clock cycles as a binary EFPFH:

$$t_{EFPFH} \approx 180 + 180 \times 5 = 1080 \text{ cycles} \tag{2.74}$$

While both of these comparison techniques could be parallelized, the conventional EFPFH may fare better on a graphics processing unit (GPU) than a binary EFPFH. GPUs are generally designed for high latency but massively parallel floating point arithmetic (favoring the EFPFH), while CPUs are generally designed for low latency serial operations (favoring the binary EFPFH).

Storing one binary EFPFH in memory requires

$$\frac{3 \text{ long}}{\text{binary EFPFH}} \times \frac{8 \text{ bytes}}{\text{long}} = 24 \ \frac{\text{bytes}}{\text{binary EFPFH}} \tag{2.75}$$

while storing one conventional EFPFH requires

$$\frac{180 \text{ float}}{\text{EFPFH}} \times \frac{4 \text{ bytes}}{\text{float}} = 720 \ \frac{\text{bytes}}{\text{EFPFH}} \tag{2.76}$$

Storing one binary EFPFH is expected to require $1/30$ the memory of a conventional EFPFH.

2.9.4 Loss of precision with the binary EFPFH comparisons

Calculating the union of two extended fast point feature histograms requires 180 additions at 32-bit floating point precision. This limits the precision of an EFPFH union to $\epsilon_{machine} \approx 10^{-7}$, or approximately 10^8 possible values. A binary EFPFH similarity coefficient, on the other hand, can only hold 181 possible values. This gives the EFPFH union approximately 6 orders of magnitude more precision than a binary EFPFH similarity comparison in the worst possible case.

2.10 Explicit Loop Closure Heuristics for uncooperatively rotating debris

2.10.1 Problem statement for loop closure detection

Given a set of 3D point clouds Y representing an ordered sequence of observations of the visible portion of an uncooperatively rotating piece of space debris,

$$Y = \{Y_1, Y_2, \ldots Y_n\} \tag{2.77}$$

how can an autonomous model scanning program determine that the same face has been observed two or more times? If the same face has been observed two or more times, which prior observation frame does this new observation correspond to? Stated mathematically, what is the probability $P_{closure}$ that frame v closes a loop with frame u:

$$P_{closure}\,(u, v)\,,\ 0 \leq u < v \leq n \tag{2.78}$$

Each observation is itself an unordered set of Cartesian coordinates (point vectors) that were observed by the sensor at that instant in time. The observed point cloud Y_i can be considered a function of the object's true topography, true orientation $\vec{\beta}$, relative location and orientation of the observer, and sensor noise on each point $\mathbf{r}_j$ observed in the cloud Y_i:

$$Y_i = \{\mathbf{r}_0 + \vec{\eta}_0, \mathbf{r}_1 + \vec{\eta}_1, \ldots, \mathbf{r}_m + \vec{\eta}_m\} \tag{2.79}$$

This set Y_i is a function of the object's true shape, the orientation quaternion $\vec{\beta}$ of the object relative

to the sensor, and sensor noise on each point :

$$Y_i = f\left(shape, \vec{\beta}_i, \begin{bmatrix} \eta_{0,i} \\ \vdots \\ \eta_{m,i} \end{bmatrix}\right) \tag{2.80}$$

Sensor noise and quantization error can cause the sensor to report points that are not the same, even if the view of the object is exactly the same in both frames. Calderita's performance characterization of the Kinect [8] can also be interpreted as meaning that even the detection of a point is uncertain. It is reasonably safe to assume that even if the exact same surface is viewed in frames u and v,

$$Y_u \neq Y_v \tag{2.81}$$

$$|Y_u| \neq |Y_v| \tag{2.82}$$

Because the presence of uncertainties that are modeled as stochastic processes in this derivation, a probabilistic, threshold-based approach is used. If the probability of closure $P_{closure}(u, v)$ meets or exceeds 0.8, then the heuristic will score a loop closure ($Y_u \Leftrightarrow Y_v$). Loop closure means that the same face has been observed twice and a graph edge will be added to the SLAM observation/model graph connecting frame u to frame v.

2.10.2 What does an ideal loop closure heuristic look like?

One of the ways that heuristic functions can differ is in how much they favor or reject extreme values near 1. This thesis defines a greedy heuristic as a heuristic that will return a fairly wide range of values near one. Figure 2.17 shows the behavior and distribution of an ideal greedy loop closure heuristic (the cosine function). This heuristic provides the largest concentration of values near 1 and 0 while quickly moving through the intermediate region. The behavior is good for a greedy loop closure scheme that ties one frame to a cluster of frames that it overlaps with (as implemented by Nüchter et al in [37]).

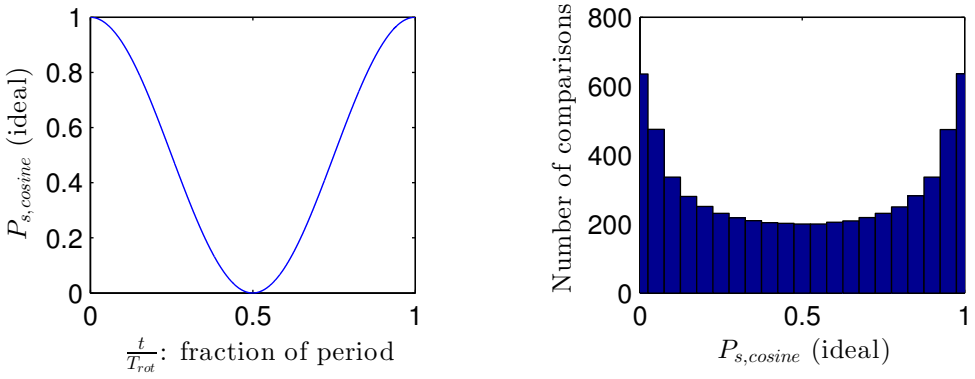

Figure 2.17: An ideal greedy loop closure heuristic for 1D rotation

As a set, greedy loop closures are a set L of previous frame indices i such that

$$L = \{i : (P_{closure,i} \geq P_{closure,min})\} \tag{2.83}$$

Contrastingly, frugal behavior provides a small set L of values i corresponding to the best probability of a single loop's closure:

$$L = \{i : \ P_{closure,i} = \max\left(\{P_{closure,j} : \ (P_{closure,j} \geq P_{closure,min}) \wedge ({}^{\Delta t_j}/_{T_{rot}} \geq 0.5)\}\right)\} \tag{2.84}$$

A triangle wave (figure 2.18) produces a uniform distribution that rejects the neighbors of extreme values 1 and 0. Raising the exponent of the triangle wave's value concentrates more values near 0 and fewer near 1, making it a good archetype for rejecting false positive loop closures:

$$P_{s,cubic} = P_{s,triangle}^3 \tag{2.85}$$

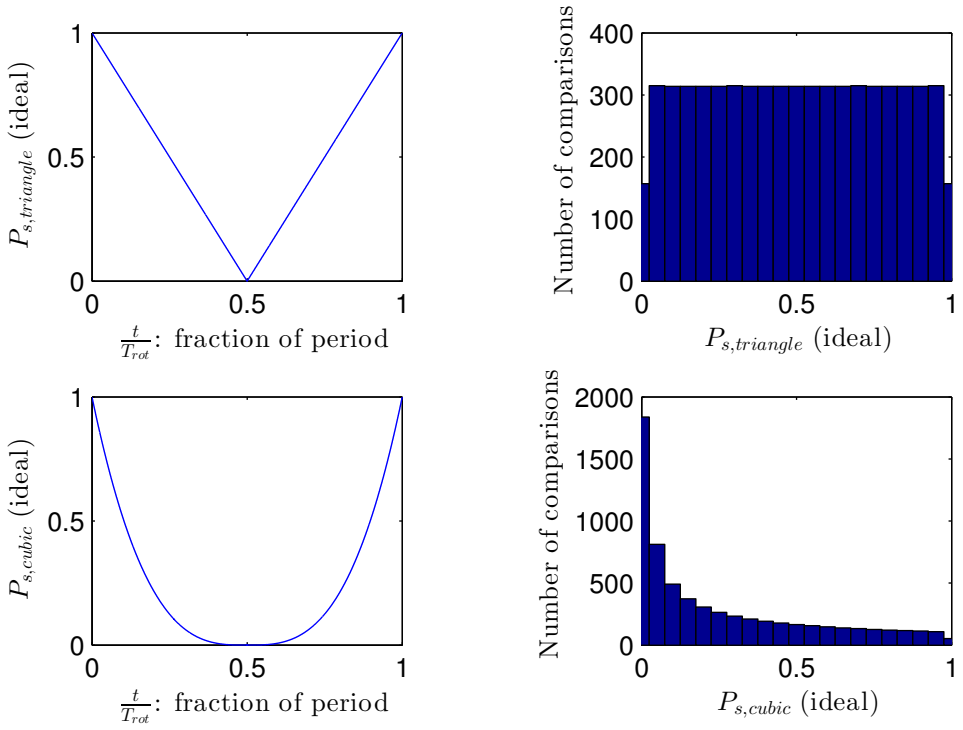

Figure 2.18: Ideal frugal loop closure heuristics for 1D rotation

2.10.3 Floating point (conventional) EFPFH similarity heuristic

If the point clouds observed at two different times are from the same face of the rotating object, then they should have very similar Extended Fast Point Feature Histograms (EFPFH). The viewpoint may be different, so that portion of the VFH is discarded and only the EFPFH, which is pose and scale invariant, is retained. This forms a 180 element feature histogram that can be renormalized to sum to 1.

Treating the two histograms U and V as sets, the Jaccard coefficient of similarity J can be used to quantify their similarity [18, 15]. From [18, 15], the Jaccard coefficient of similarity J between two sets U and V is defined as

$$J = \frac{|U \cap V|}{|U \cup V|} \tag{2.86}$$

Hetzel [17] defines a computationally frugal method of comparing the similarity of two histograms U and V [17] :

$$|U \cap V| = \sum_{i=1}^{n} \min(U_i, V_i) \tag{2.87}$$

[17] Notation modified from [17] for consistency with the other detection heuristics in this section.

The cardinality of each histogram is defined as the sum of its individual elements. Assuming the EFPFH is normalized to sum to 1,

$$|U| = |V| = \sum_{i=1}^{180} U_i = 1 \tag{2.88}$$

From [13], the cardinality union of two sets U and V can be calculated using the cardinality of the intersection and the individual sets:

$$|U \cup V| = |U| + |V| - |U \cap V| \tag{2.89}$$

Treating the EFPFH (equation 2.72) as a set, the Jaccard coefficient of similarity is a fraction that represents the topographical similarity between two point clouds. It is re-used as the probability $P_{s,f}$ that frame v closes a loop with frame u based on the topographical similarity of the two frames:

$$P_{s,f} = J = \frac{|U \cap V|}{|U \cup V|},\ 0 \leq P_{s,f} \leq 1 \tag{2.90}$$

The quantity $P_{s,f}$ has several probability-like characteristics. If all of the histogram bins are equal, the intersection and union of U and V are the same, so $P_{s,f} = 1$. If no histogram positions are equal, the intersection of U and V is $\emptyset$, so $P_{s,f} = 0$.

$P_{s,f}$ also has several characteristics that are not probability-like. Because the EFPFH is an information reduction, $P_{s,f} = 1$ does not necessarily indicate that two frames are the same fragment. Because the EFPFH does not include information about which points are on which side of the fragment, the $P_{s,f}$ cannot discriminate between a fragment A and its mirror image U:

$$P_{s,f}(A, V) = 1 = P_{s,f}(U, V),\ A \neq U \tag{2.91}$$

Because the EFPFH similarity heuristic is derived from the EFPFH, $P_{s,f}$ is pose invariant[18] .

[18] See section 2.2.3.1 for a more detailed discussion of the FPFH's pose invariance.

2.10.4 Binary EFPFH similarity heuristic

Using the same strategy as the floating point EFPFH, the binary EFPFH (equation 2.72) will be treated as a set. The Jaccard coefficient of similarity will also be the probability $P_{s,b}$ that frame v closes a loop with frame u based on frame similarity:

$$P_{s,b} = J = \frac{|U \cap V|}{|U \cup V|},\ 0 \leq P_{s,b} \leq 1 \tag{2.92}$$

The size of the intersection is calculated using the method presented in [30]. If $U_i = V_i$, then $U_i = V_i$ is contained in the set $U \cap V$. If the bit values at position i differ ($U_i \neq V_i$), then neither U_i nor V_i are in the set $U \cap V$. The cardinality of the union can be calculated using the cardinality of $U \cap V$ and the length of the feature sets $|U| = |V| = 180$ as

$$|U \cup V| = |U| + |V| - |U \cap V| = 360 - |U \cap V| \tag{2.93}$$

Equivalently,

$$P_{s,b} = \frac{360 - 2\,|U - V|}{360 + 2\,|U - V|} = \frac{180 - |U - V|}{180 + |U - V|} \tag{2.94}$$

The binary EFPFH shares many its statistical characteristics with the floating point EFPFH. Both are bounded to the interval $[0, 1]$, $P_{s,b}\,(A, B) = 1$ does not necessarily imply that $A = B$, and both are unable to distinguish between a point cloud and its mirror image. Like the floating point FPFH, the binary EFPFH is pose invariant[19] .

[19] See section 2.2.3.1 for a more detailed discussion of the FPFH's pose invariance.

2.10.5 Orientation based heuristic

The quaternion vector $\vec{\beta}_i$ represents the best estimate of the object's body-frame attitude (defined as the first visible frame of the object) relative to the inertial frame. From [51], two quaternions can be combined using the quaternion composition matrix $G\left(\vec{\beta}\right)$, where

$$G\left(\vec{\beta}\right) = \begin{bmatrix} \beta_0 & -\beta_1 & -\beta_2 & -\beta_3 \\ \beta_1 & \beta_0 & -\beta_3 & \beta_2 \\ \beta_2 & \beta_3 & \beta_0 & -\beta_1 \\ \beta_3 & -\beta_2 & \beta_1 & \beta_0 \end{bmatrix} \tag{2.95}$$

A rotation through quaternion $\vec{\beta}_u$, then $\vec{\beta}_v$ is

$$\vec{\beta}_{u+v} = \left[G\left(\vec{\beta_u}\right)\right]\vec{\beta_v} \tag{2.96}$$

By negating the rotation (scalar component) of one of the quaternions, equation 2.96 can be turned into an orientation difference between two quaternions. For the orientation difference from quaternion $\vec{\beta}_u$ to $\vec{\beta}_v$, the $\vec{\beta}_u$ rotation is negated by reversing the sign of the scalar parameter $\beta_{0,u}$ [51]. Following the computer graphics convention of describing a negated rotation quaternion as a quaternion inverse[20] [12],

$$\vec{\beta}_u^{-1} \equiv \begin{bmatrix} -\beta_{0,u} \\ \beta_{1,u} \\ \beta_{2,u} \\ \beta_{3,u} \end{bmatrix} \tag{2.97}$$

This allows the orientation difference between two quaternion vectors $\vec{\beta}_v$ and $\vec{\beta}_u$ to be defined as

$$\vec{\beta}_{v-u} = \left[G\left(\vec{\beta_u}^{-1}\right)\right]\vec{\beta_v} \tag{2.98}$$

This can be algebraically manipulated to yield a principle axis rotation angle between frames u and v:

$$\Phi_{v-u} = 2\cos^{-1}\beta_{v-u,0} \tag{2.99}$$

[20] Special thanks to Joshua Chabot for correctly pointing out that a quaternion, as a vector, technically does not have an inverse. The notation in equation 2.97 is a slightly confusing computer graphics convention that is used in some software libraries used by pclAutoScanner. Similarly, quaternion composition, which Schaub [51] writes as a matrix-vector product $\vec{\beta} = \left[G\left(\vec{\beta}'\right)\right]\vec{\beta}''$, appears in [12] as $\vec{\beta} = \vec{\beta}'\vec{\beta}''$.

However, each distinct orientation can be represented by two different quaternions: either a rotation of $+\Phi$ about $+\hat{\mathbf{e}}$ or a rotation of $-\Phi$ about $-\hat{\mathbf{e}}$. To make the comparison less sensitive to which quaternion the algorithms return and or non-deterministic ordering of frames u and v, the principle rotation angle will be normalized to $-180^\circ \leq \Phi \leq 180^\circ$:

$$\Phi_{u\to v} = \begin{cases} \Phi_{v-u} & if\ \Phi_{v-u} \leq 180^\circ \\ \Phi_{v-u} - 360^\circ & if\ \Phi_{v-u} > 180^\circ \end{cases} \tag{2.100}$$

Assuming rotation about some axis that does not pass through the observer's location when drawn from the rotating object's center of mass, if u and v are the same face, then $|\Phi_{u\to v}| \approx 0^\circ$. This can be used to create a probability of loop closure based on orientation estimate:

$$P_\beta = 1 - \frac{|\Phi_{u\to v}|}{180^\circ} \tag{2.101}$$

There are some limitations on equation 2.101. It only holds if the principle rotation axis $\hat{\mathbf{e}}$, when drawn at the center of mass of the rotating object, does not pass through the observer's location. Another interpretation is that if the principle rotation axis $\hat{\mathbf{e}}$ points at the observer, then u and v are the same face and P_β should be 1.

The attitude heuristic P_β has some probability-like characteristics:

- As $|\Phi_{u\to v}| \to 0^\circ,\ P_\beta \to 1$
- As $|\Phi_{u\to v}| \to 180^\circ,\ P_\beta \to 0$

The attitude heuristic has a flaw that prevents it from being a stand-alone probabilistic heuristic. Equations 2.63 and 2.21 show that the angle $\Phi_{u\to v}$ is the equivalent of an odometry-based state estimate. Just like odometry-based mapping in terrestrial robotics [37, 29, 53], P_β is expected to drift from the true value.

The odometer drift in P_β will be exaggerated because it is a first difference of a particularly volatile quantity. First differences, like the angular rate vector $\vec{\omega}$ whose forward Euler integral drives the value of angle $\Phi_{u\to v}$, are generally noisy. Compounding this, the integral may have included a drastic ICP frame alignment error (section 2.7.1).

2.10.6 Joint heuristic probability of loop closure detection

The probability of detecting loop closure is a joint probability of two related (non-independent) probability of closure heuristic functions P_s and P_β. Both of these are influenced by random variables. Most of the random variables affecting the similarity heuristic are expected to be independent of those affecting the attitude heuristic.

The two heuristics have complementary strengths and weaknesses. The similarity based heuristic is prone to false positives because it cannot distinguish between a point cloud and its reflection. The attitude-based heuristic has a phase delay inherited from the attitude estimation EKF. To mitigate these weaknesses, the square root of the product of these heuristics is used as a joint heuristic (equation 2.102).

$$P_{closure} = \sqrt{P_s P_\beta} = F_{s\beta}\left(s, \beta\right) \tag{2.102}$$

The multiplication causes one signal to shape the other, similar to cross-correlation. The false positives of the similarity heuristic are likely to occur when the orientation heuristic has a value near zero, nulling the false positives out of the joint heuristic. The similarity heuristic's true positive is expected to peak abruptly, then drop abruply just before the orientation heuristic peaks. The decrease of the similarity heuristic should drive the joint heuristic below the threshhold when the phase-delayed orientation heuristic peaks.

The square root moves the expected value of the joint heuristic to 0.5, under the assumption that the random variables affecting each component heuristic are independent and each heuristic's mean value is 0.5:

$$\langle P_s \rangle = \langle P_\beta \rangle = 0.5$$

$$\langle P_s \rangle \langle P_\beta \rangle = 0.25 \approx \langle P_s P_\beta \rangle$$

$$\left\langle \sqrt{P_s P_\beta} \right\rangle \approx 0.5$$

The joint probability of closure $P_{closure} = \sqrt{P_s P_\beta}$ shares the range restriction of its two components:

$$\begin{aligned} 0 \leq \quad & P_s & \leq 1 \\ 0 \leq \quad & P_s P_\beta & \leq 1 \\ 0 \leq \quad & \sqrt{P_s P_\beta} & \leq 1 \end{aligned}$$

Chapter 3

Methods and Materials

3.1 Experimental Apparatus

Three experiments are used to test the performance of Graph SLAM and the loop closure heuristics for semi-autonomous model generation of an uncooperatively rotating plastic model of the NASA space shuttle. All three of the experiments capture data using a Microsoft Kinect$^{\text{TM}}$ structured light sensor and execute the model-building algorithms offline, using the previously captured point cloud data sets.

The different experiments vary the types of target and the types of relative motion. The first experiment uses an inertially fixed observer, with the Space Shuttle model rotating on top of a magnetic platform. The second experiment keeps the observer inertially fixed, but adds a mechanical constraint on the target's motion. The third experiment allows both the observer and target to translate and rotate to test the performance of the algorithms with relative motion.

3.1.1 Inertially fixed observer, 1-plane symmetric target

In this experiment, the plastic Space Shuttle model will rest on top of the Levitron$^{\text{TM}}$ platform and rotate freely, but primarily about its $+y$-axis. The Levitron$^{\text{TM}}$ magnetic levitation platform is a small executive toy for displaying models. The rectangular base includes four electromagnets that repulse a short cylindrical magnet, holding it (and the Shuttle model) in place above the platform. While the electromagnetic fields emitted by the Levitron$^{\text{TM}}$ keep the magnetic platform's poles mostly aligned with the $+y$-axis, there is no hard mechanical constraint on motion, so the model is expected to wobble slightly.

The magnetic repulsion acts as a low friction bearing. It should be noted that the electromagnets are

actively controlled, which means that there are internal dynamics that may exert small unintended torques on the magnet and model. These are not a problem, however, because the goal of this experiment is to test performance against an uncooperative target. Figure 3.1 shows a photograph of the experimental apparatus. A sketch of the apparatus also appears in figure 2.15.

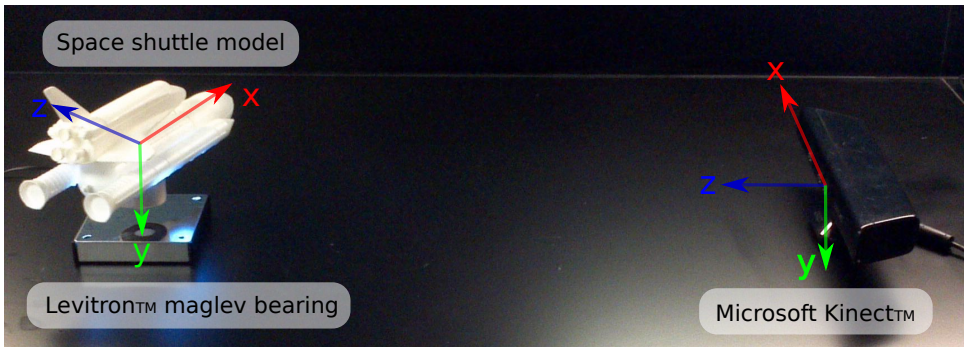

Figure 3.1: Photograph of the inertially fixed observer, 1-plane symmetric target experiment

The experiment started by powering on all of the equipment, giving the shuttle a small spin about the $+y$-axis by hand, then collecting raw frames from the Kinect^{TM} using the pcl_openni_pcd_recorder utility on a laptop computer running Fedora 20. Three data sets were collected in this configuration: shuttle-end-on, shuttle-end-on-redone and shuttle-end-on-redone2. Data was analyzed offline, post-experiment.

3.1.2 Inertially fixed observer, 2-plane symmetric target

The Space Shuttle model used in the other experiments is easier for topographic feature-based heuristics because it only has one plane of symmetry. The front of the shuttle is also shaped very differently from the back. Spent upper stages of space launch vehicles are not so uniquely defined. Some of them are axisymmetric and rotate about their axis of symmetry.

This is the absolute worse case for angular rate-based model generation method described in this thesis because the topography of the rotating debris does not change much as the rocket rotates. In this case, the algorithms in this paper cannot generate a model of the debris because angular rotation rate cannot be observed (P_β ineffective). The similarity loop closure heuristic should also saturate with an expected value of 1 (P_s ineffective).

While these model generation methods are expected to fail for rotation about an axis of symmetry, there is a fringe case in which they may or may not function. Principal axis rotation of an object with two planes of symmetry, such as the Shuttle's tank and solid rocket boosters (SRBs) may be observable as a rotation rate from one frame to the next. The topography of the visible fragment may change enough that the similarity loop closure heuristic does not saturate (has an expected value less than one).

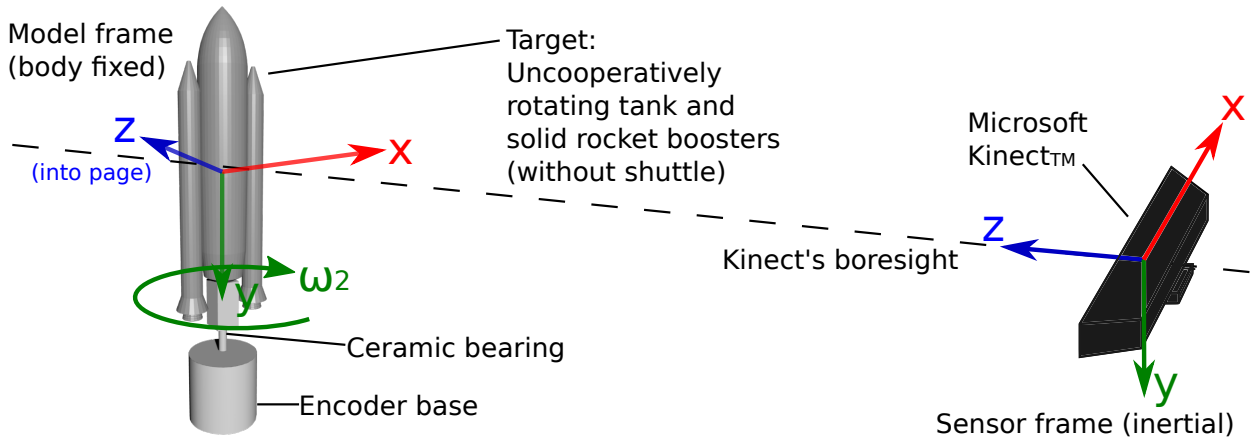

Figure 3.2: External tank and SRBs (minus Shuttle) on encoder. 3D Shuttle model courtesy NASA [7]

To test the performance of the algorithms and the heuristic in this fringe case, the Shuttle was removed from the plastic model and affixed to a digital rotation encoder with a ceramic bearing (figure 3.2). This mechanically constrains the rotation to the object's principle axis. There is no mechanical drive on the bearing, so it was spun by hand then released. The friction in the bearing slowly decelerated the rotation of the tank and SRB plastic model as the Microsoft Kinect^{TM} collected point cloud observations. The point cloud data is then post-processed through the pclAutoScanner application to align the point clouds and generate a model of the tank and SRBs in the same manner as the previous experiment. Two data sets were collected: shuttle-encoder and shuttle-encoder2.

3.1.3 Moving observer with rotating target

Most of the use cases described in the chapter 1 take place in the Earth orbit environment. Rendezvous and inspection of a target will most likely involve formation flying that can be simplified using the Clohessy-Wiltshire equations for small relative distances. To simulate these conditions, the 1-plane symmetric experiment is repeated, but with the modification that the Kinect™ sensor is suspended vertically from four cables, deflected from its equilibrium condition and allowed to swing like a pendulum (figure 3.3). This should produce relative motion similar to that predicted by the Clohessy-Wiltshire/Hill equations, with the exception that the helix should wrap around on itself because the y-axis, which is unconstrained in the Clohessy-Wiltshire/Hill equations, is now constrained to pendulum-like motion.

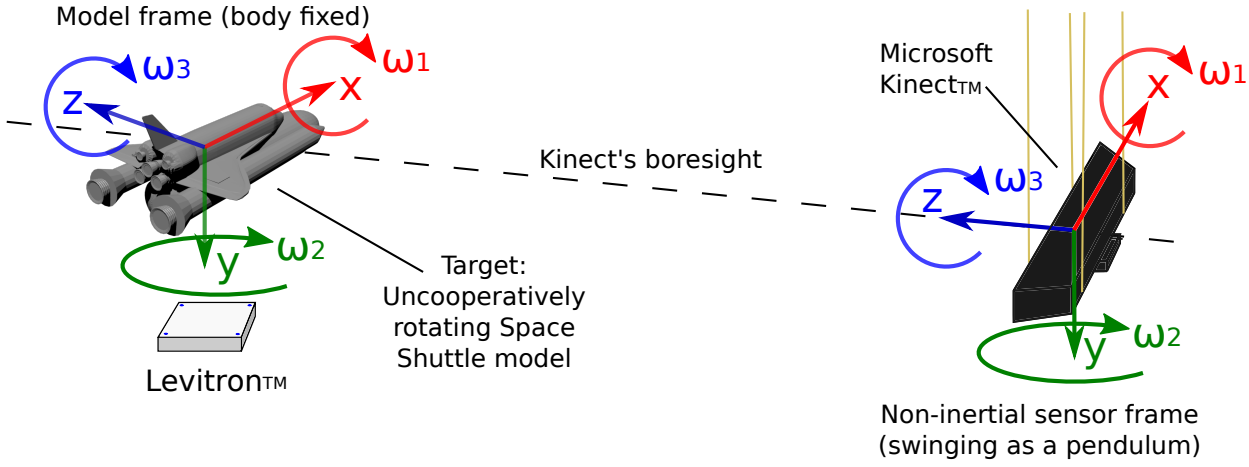

Figure 3.3: Diagram of the moving observer experiment. 3D Shuttle model courtesy NASA [7]

3.2 Data Processing Pipeline

The preliminary dynamics model will be validated using point clouds that have been captured by a Microsoft Kinect™ and saved to a hard drive. These will be aligned to the previous frame and adaptively downsampled using principal axis odometry to approximately 10° of principal axis rotation between frames. These downsampled attitude estimates will be exported to a comma-separated values file, then processed in MATLAB using an implementation of the batch processor described in [55]. Figure 3.4 shows a diagram of the preliminary data processing pipeline.

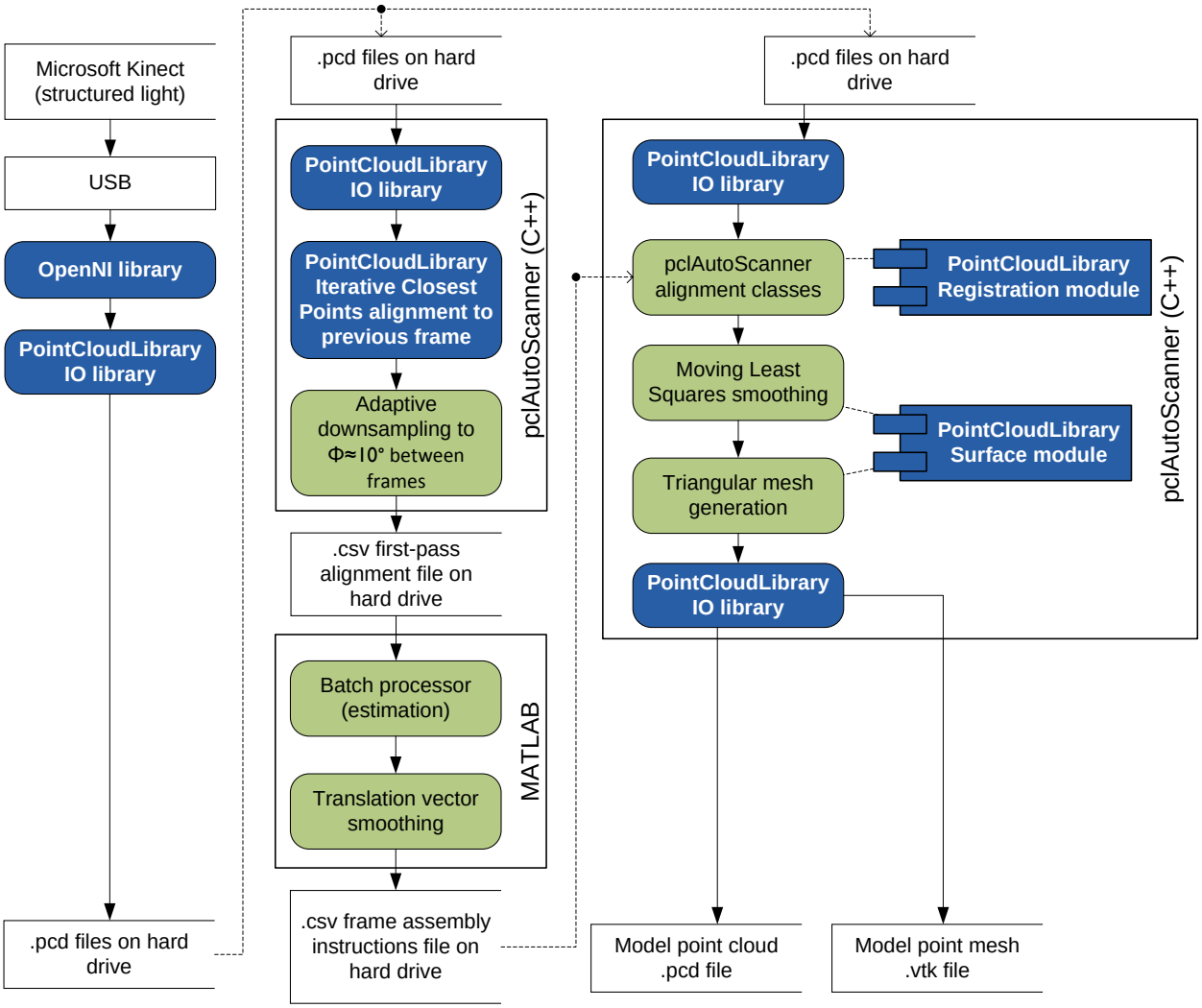

Figure 3.4: *A priori* least squares data processing pipeline

Next, an estimate of the orientation and translation transformation to align each downsampled frame to the model frame is exported to a different comma-separated values file, which is then ingested by the pclAutoScanner application. The pclAutoScanner is custom code written for this research that uses the Point Cloud Library to align frames and autonomously generate models from a set of point cloud files.

The batch processor in this demonstration is not intended to be part of the final configuration. It was only used for this validation because it was previously existing, tested software from a previously taken class on estimation.

After validating the basic dynamics model using the batch processor, a more realistic data processing

suite (figure 3.5) was implemented using an Extended Kalman Filter in place of the batch processor for the least squares attitude estimate. The EKF data processing pipeline also added more pre-processing to the input point cloud files. Adaptive principle axis downsampling was removed because it required an ICP frame alignment for each file loaded (considerable computational cost). It was also filtered to a bounding box to remove points outside of the volume near the Shuttle model (walls, bench, cabinets, etc). Space is expected to be a comparatively low-clutter environment, where the sensor sees a vast, empty space containing only the target object, so this does not decrease the realism of the experiment.

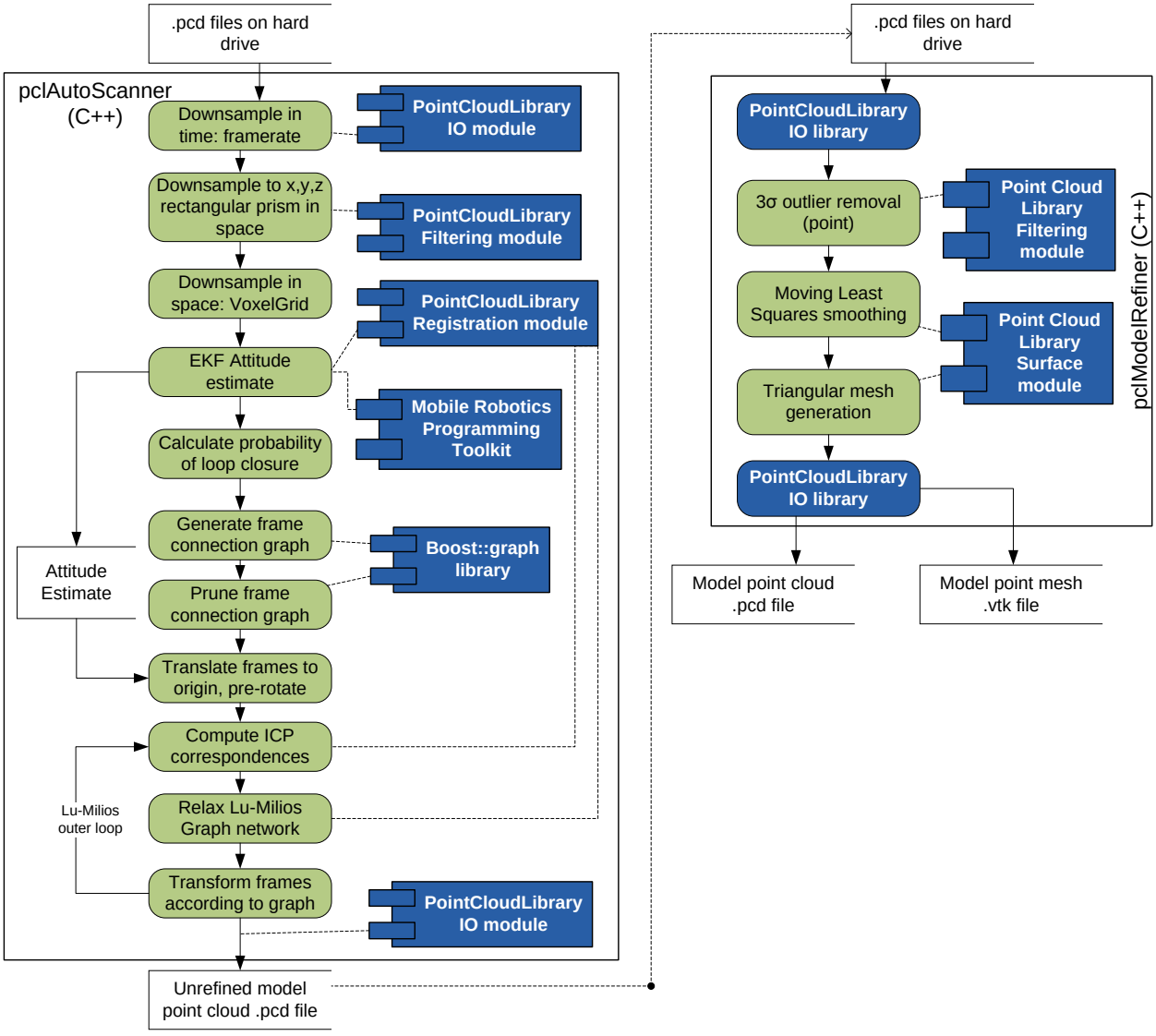

Figure 3.5: EKF-based data processing pipeline

The incoming point clouds are also downsampled using a voxel grid. Voxel gridding as a quantization technique for 3D data in which several points that happen to fall within the same cubic region of space are replaced with a single point located at the center of the bounding voxel. This experiment uses cube-shaped voxels.

The next step in the process is to create an estimate of the probability that each frame closes a loop with some prior frame. The attitude based estimate is derived from the EKF's estimate of the unknown object's attitude based on differential ICP alignment of point cloud frames. The similarity-based estimates are created by computing an EFPFH and binary EFPFH on each data frame in the downsampled set of point cloud frames. The EFPFH/binary EFPFH from each frame is compared with the EFPFH/binary EFPFH of each frame that preceded it. The two probabilities of loop closure are then combined into joint probabilities that each frame closes a loop with each of its preceding frames.

The probabilities of loop closure are then used to create the loop closure graph. There are two variations on this stage. In the greedy variation, all loop closure whose probability of closure exceeds a user-specified threshold are records as loop closures, including the nodes that directly precede the node in question. In the frugal (default) version, loop closures are only recorded if the preceding frame is at least $1/2$ of one rotational period prior to the frame in question. Connections between consecutive frames are only assumed if the consecutive frames are within 1.5 time quanta (inverse of sampling rate) of each other.

This step can produce chains of frames that do not include any loops, implausibly short loops cause by false positive closure detections, or island nodes chains or loops that are separated from the main body due to a data gap. These are all conditions that will prevent Graph SLAM from converging on an accurate alignment of the fragments. This process, called graph pruning, is a two-stage process. The first stage identifies islands by performing a breadth-first search starting at frame 0 (the root node). When the node visitor has visited each node in the root node's graph, it chooses another unvisited node and randomly conducts another breadth first search. As each graph is exhausted, the size of the graph is recorded and compared against the largest subgraph size previously observed in breadth first search.

When all subgraphs have been visited, the largest subgraph is retained. All nodes that are not part of that maximal subgraph are deleted. This removes all of the islands (disconnected sub-maximal graphs).

Next, each node is tested to see if it is a member of a cycle. A directed graph depth-first search is started from each node. If the node is revisited through an incoming edge, it is recorded as being part of a cycle. When each of the nodes has been tested in this manner, all nodes that were not recorded to be part of a cycle are deleted. This may have broken apart a cycle, so the pruning process is repeated in a while loop until either a single graph containing only cycles remains or all points have been removed (i.e. there are no full loop closures).

Next, the centroid of each frame is calculated. The centroid is used to translate the entire point cloud back to the origin. Each frame is then rotated to the body frame orientation that was estimated by the EKF as part of the closure probability estimation. This makes the algorithm less vulnerable to relative motion between the observer and the target, because all point cloud images of the target are translated back to the origin before the model is generated.

After that, Lu-Milios graph relaxation is iterated in a nested loop. The other loop runs a small number of graph relaxation iterations, rotating and translating each frame to lower the overall sum of misalignment errors for frames that are connected (according to the previously generated loop closure graph). After the relaxation converges, the resulting transformations are applied to each frame. Then, the loop is repeated.

This configuration is inspired by the Jochen Spricherhof's demo implementation in the Point Cloud Library. It was reused for this research because the maximum correspondence distance limits the quality of each ICP frame alignment within the Lu-Milios graph relaxation. After the alignment of the frames has been corrected by the graph relaxation, the correspondence estimator may be able to find more correspondences between the two frames because the frames themselves are closer together in space. This allows the next iteration of graph relaxation to reduce the overall error state further. Both the Lu-Milios relaxation convergence threshold and maximum number of iterations are user-configurable run-time parameters.

After completing all Lu-Milios graph relaxation iterations, the source point clouds are concatenated to a single cloud and written to a file. The pclModelRefiner is then invoked to the number of points in the model and smooth out noise and misalignments. The first step is another voxel grid reduction. After that, 3σ outliers are removed based on distance from the centroid of the point's $k = 50$ nearest neighbor sample set. The remaining points are then smoothed and downsampled again using a 5th-order Moving

Least Squares curve fitting algorithm. Lastly, a triangular mesh is applied to the smoothed points using the greedy projection algorithm. This results in a final output XYZRGBA point cloud and Visualization Toolkit (VTK) mesh file of the object that was observed.

Chapter 4

Results

4.1 Performance of purely least-squares model generation

Figure 4.1 shows a plot of the post-fit residuals for the angular rate estimate produced by the batch processor for the static observer, rotating target experiment. The 1- and 3-axis (x and z, respectively) residuals are small, suggesting that the dynamics model is acceptable for those axes. The large sinusoidal 2-axis residual suggests that the assumption of torque free planar rotation about the y axis is not valid.

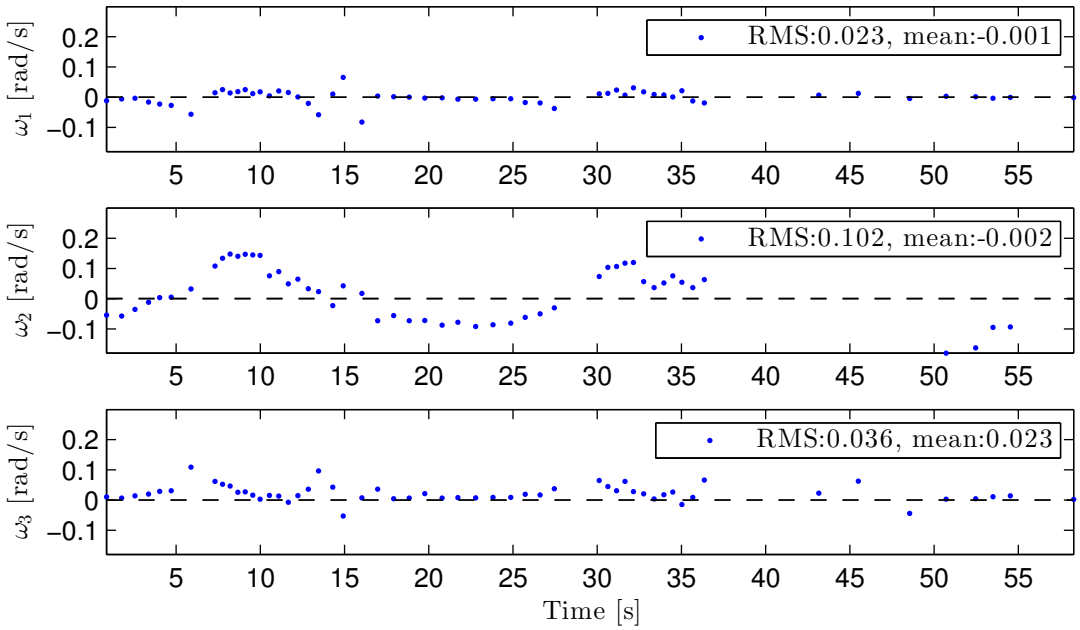

Figure 4.1: Post-fit residuals of the batch processor

The estimated 2-axis (y) angular velocity is 0.2795 radians per second (figure 4.2). That value is on the same order of magnitude as the sinusoidal RMS of the post-fit residual, suggesting that the orientation estimate is also likely to have significant error. Figure 4.2 also shows that the batch processor's orientation estimate is for constant rate planar rotation about the y-axis, which is consistent with the dynamics model. This, combined with the post-fit residuals in figure 4.1 suggest that the batch processor is functioning correctly for the dynamics model and data that it processed. Any errors in the state estimate are probably due to an incorrect dynamics model or incorrect alignments between frames.

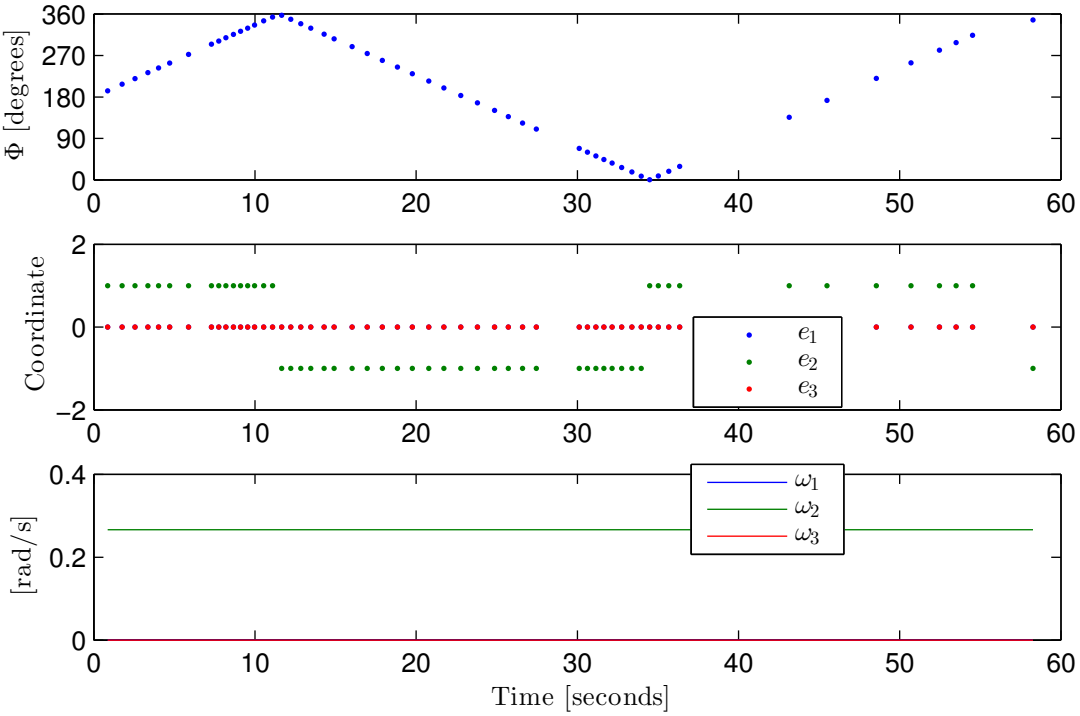

Figure 4.2: State estimate used for model generation

The sign change in the y-axis component of the principle rotation axis vector $\hat{\mathbf{e}}$ is purely an artifact of the principle rotation angle wrapping behavior convention in the implementation MATLAB script. The object is not actually changing its direction of rotation; it is just reaching 360° and wrapping around to 0°.

Figure 4.3 shows the model generated by rotating and translating the individual point cloud observations according to the state estimates produced by the batch processor. This plot shows that there is drift in both the orientation and the translation between observed frames. As predicted, accurate autonomous 3D model generation does require a drift correction mechanism like Graph SLAM.

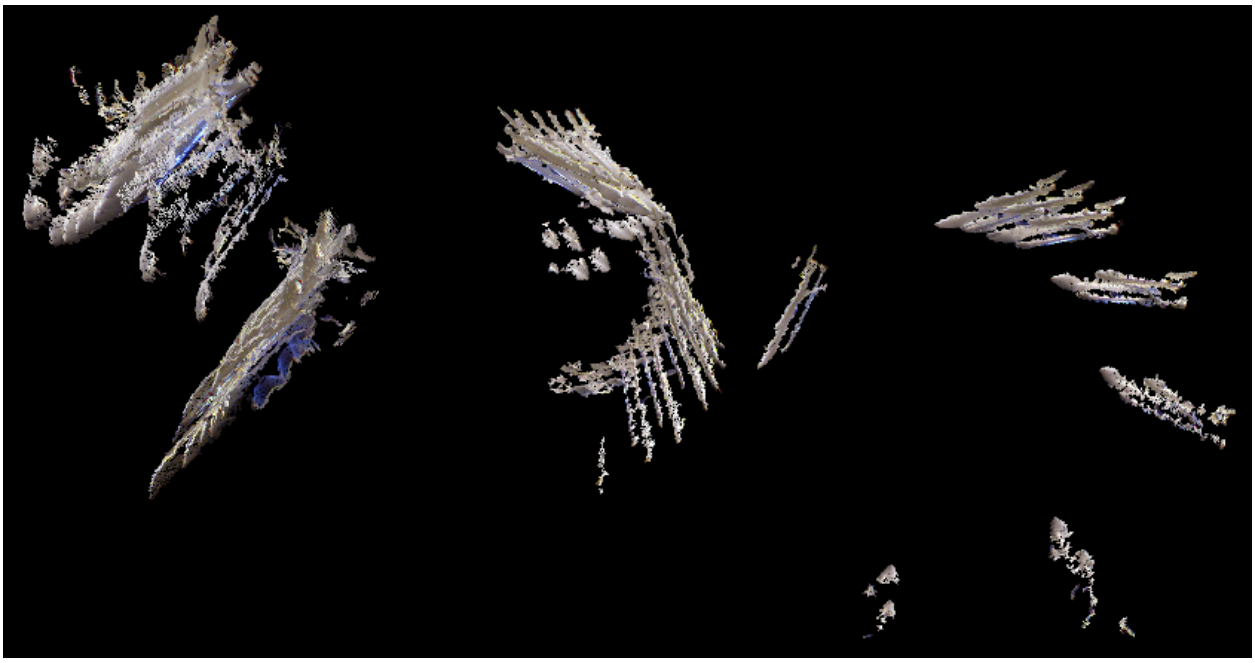

Figure 4.3: Model generated from batch processor orientation estimate

The translation vector cannot be estimated by the batch processor because the shape of the object that is rotating is unknown to the sensor when it generates a model of the object. The work-around adds odometer drift because it is a numerical integration of the translation vector ${}^{n-1}\mathbf{t}_n$ from the original ICP alignment between two successive frames $n-1$ and n:

$$ {}^{n-1}R_n = \left[{}^{0}R_{n-1}\right]^T \left[{}^{0}R_n\right] \tag{4.1} $$

$$ {}^{n-1}H_n = \begin{bmatrix} {}^{n-1}R_n & {}^{n-1}\mathbf{t}_n \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{4.2} $$

$$ {}^{0}H_n = \left[{}^{0}H_{n-1}\right] \left[{}^{n-1}H_n\right], \; {}^{0}H_0 \equiv I_{4\times 4} \tag{4.3} $$

4.2 Inertially fixed observer, 1-plane symmetric target

Figure 4.4 shows that Graph SLAM and the joint loop closure probability heuristic were able to assemble a model of the shuttle from Kinect data at a range of 0.85-1.2 meters. This is a longer range, lower resolution observation than the other data sets used in this thesis, but still a larger target size/distance ratio (closer fly-by) than would be expected on-orbit based on the PRISMA spacecraft's orbital debris rendezvous profile [24].

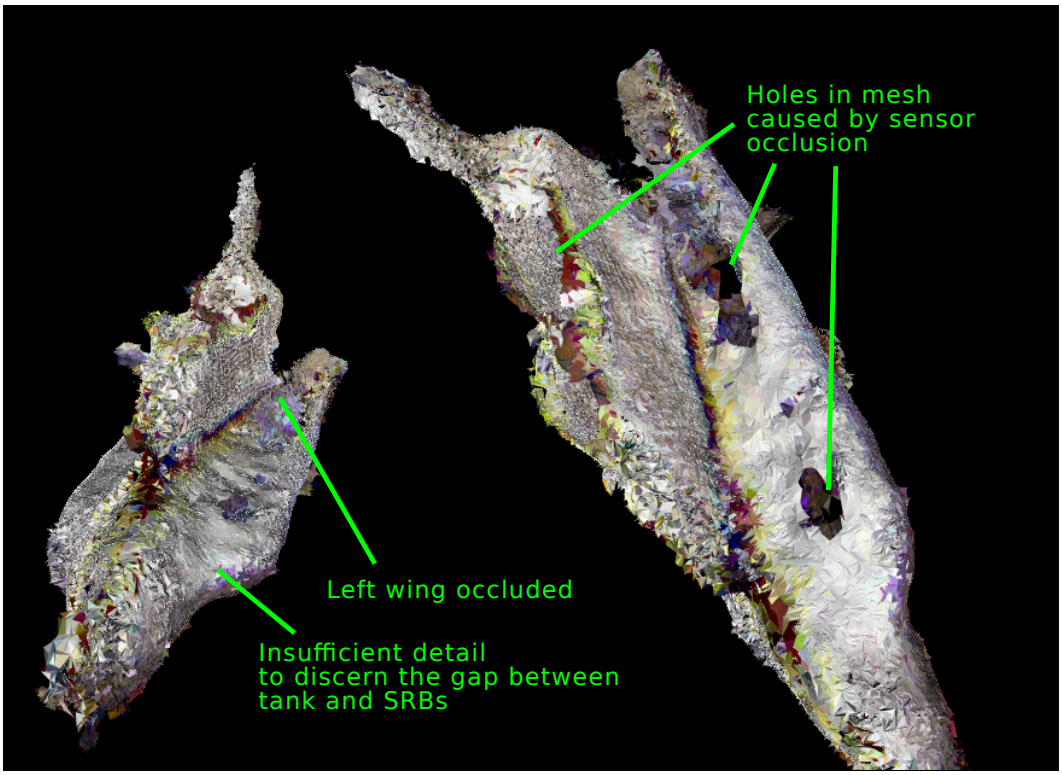

Figure 4.4: Triangular mesh of the Shuttle model further away from the sensor

At this range and orientation, there are significant sensor occlusions (regions that the sensor cannot see). These occlusions create gaps in the model (figure 4.4). The gaps can only be filled in by observing the rotating object from a different angle, which may require the observing vehicle to perform a maneuver. The source point clouds have too little detail to discern the gap between the external tank and the solid rocket boosters (SRBs). This could be addressed by using a higher quality sensor or moving the sensor closer to

the rotating target.

The most sensitive model generation parameters for this experiment were the number Lu-Milios graph net relaxations and the minimum probability of closure $P_{Closure,min}$. Too few Lu-Milios relaxation iterations produced two overlapping models of the Space Shuttle. Too low a convergence criteria results in a failure to detect any viable observation loops after graph pruning.

Table 4.1: Successful 1-plane of symmetry model generation settings

Parameter	**Value**
Data set	shuttle-end-on-redone
Sample rate	5 Hz
Lu-Milios convergence tolerance	10^{-9}
Maximum outer loop alignment iterations	**60**
Maximum inner Lu-Milios relaxation iterations	**60**
Process noise magnitude σ_u^2	0.1
Observation noise magnitude σ_v^2	d_k (ICP fitness)
$P_{closure,min}$	**0.82**
A priori initial state $\overline{\mathbf{X}}$	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0.2795 & 0 \end{bmatrix}^T$
A priori covariance $\overline{P}$	$I_{7\times7}$
Voxel grid downsample size	2mm
Pass-through point filter dimensions (m)	$-0.25 \leq x \leq 0.15$ $-0.1 \leq y \leq 0.11$ $0.85 \leq z \leq 1.20$

This experiment also showed that the initial state estimate is not a very sensitive parameter. The initial conditions were accidentally re-used from a previous data set, causing a 143% error in the ω_2 angular rate initial estimate. The EKF still converged within approximately 2 seconds, with covariance low enough to reject frame misalignment outliers at 5.3 seconds, 9.3 seconds and 13.5 seconds (figure 4.11).

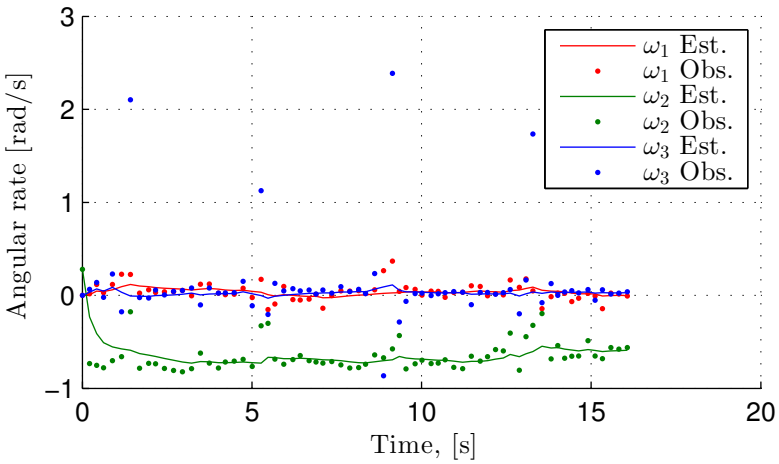

Figure 4.5: EKF converging in spite of a sign error on the *a priori* estimate

Figure 4.6 shows that the covariance is still high for the EKF attitude estimator and the RMS of the residual is on the same order of magnitude as the quantities being measured. This shows that the EKF is not over-fitting to the data, but also that the first difference angular rate measurement from the ICP frame-to-frame alignment is noisy.

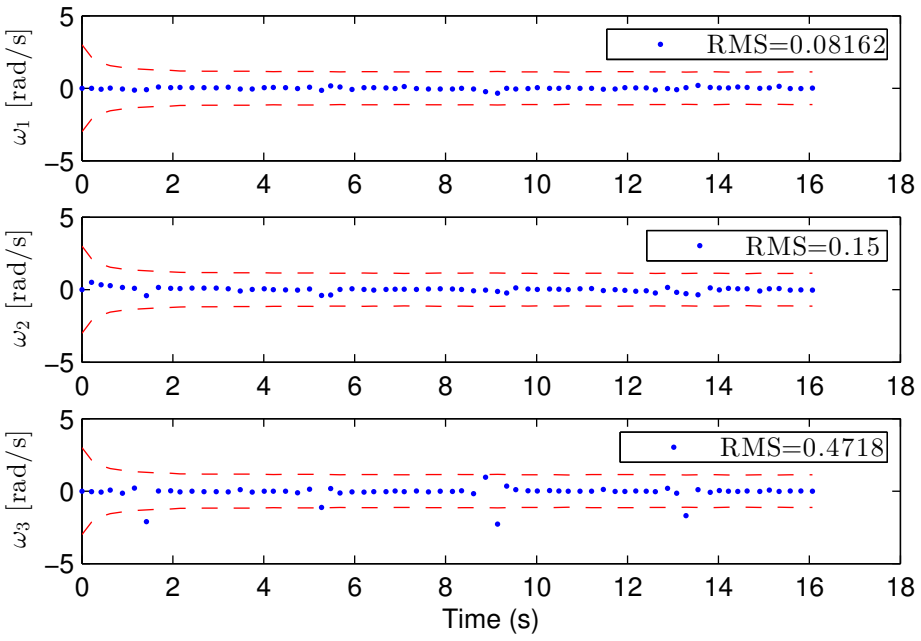

Figure 4.6: EKF post-fit residuals, shuttle-end-on-redone data set. Red line is the 3σ interval.

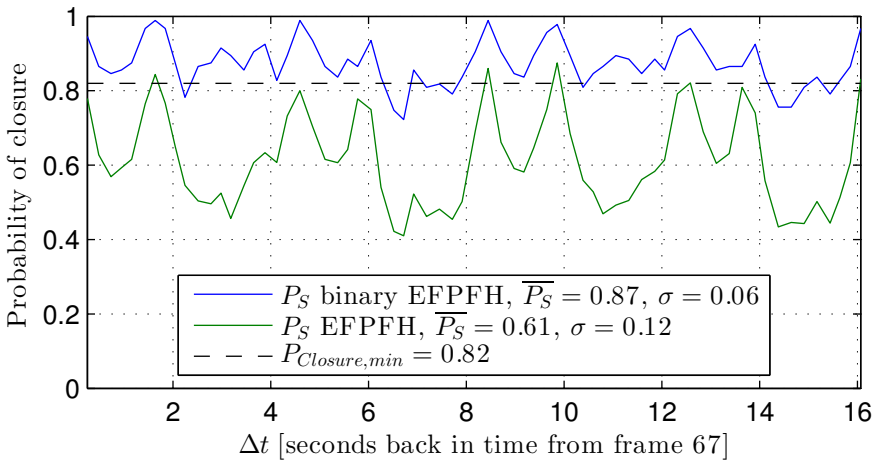

Figure 4.7: Comparison of binary and floating point P_S loop closure, shuttle-encoder frame 67

Figure 4.7 shows that the mean value of the binary EFPFH loop closure heuristic is very high (0.87) with a low standard deviation. The periodicity of the signal (which is expected for an object in planar rotation) is much more clear in the floating point EFPFH heuristic, suggesting a rotational period of approximately 8.2 seconds. This is within the 2σ confidence interval of the EKF's rotational period estimate after converging ($\langle T_{rot} \rangle = 9.40 \pm 1.6$s).

The floating point EFPFH heuristic has more dynamic range in this data set. Figure 4.8 show that this gives the floating point loop closure heuristic more impact on the shape of the joint loop closure heuristic.

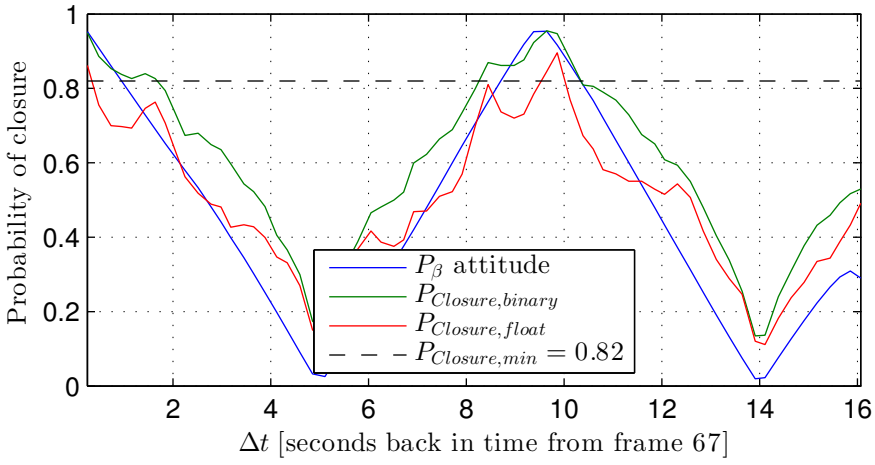

Figure 4.8: Joint $P_{Closure}$ of loop closure, shuttle-end-on-redone frame 67

The best model produced in this research (figure 4.9) was produced from an earlier version of this experiment, using a pure similarity loop closure heuristic (binary EFPFH bits difference) that sought out chains of loop closures. This heuristic performed very well against a the shuttle-end-on data set, but was overwhelmed with false positive loop closure detections when it was tested against the shuttle-encoder data set. The similarity heuristic's poor performance against an object with multiple planes of symmetry was the primary reason for developing the joint EKF-similarity probability of loop closure heuristic. The pure similarity implementation also required a user-selected time window, while the joint heuristic increases autonomy by selecting the best time interval without user input.

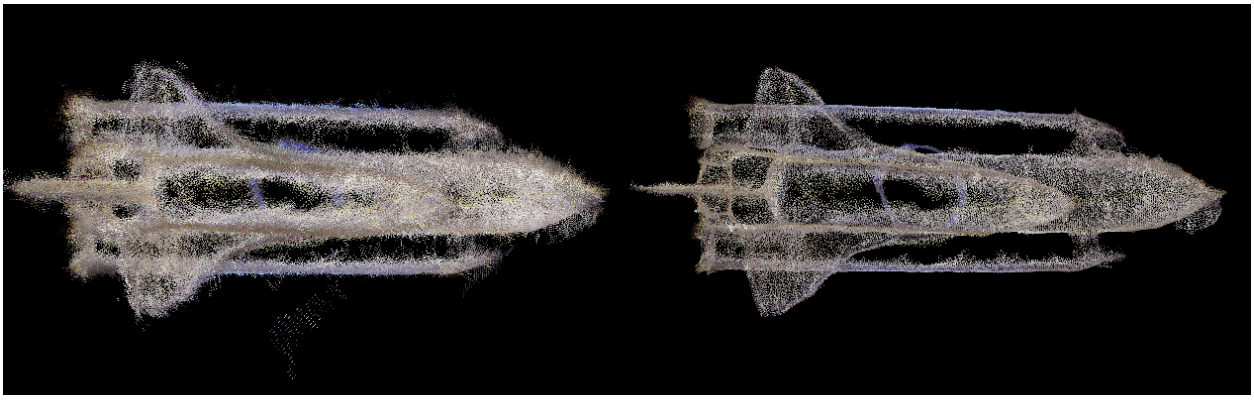

Figure 4.9: Shuttle alignment using purely binary EFPFH loop closure heuristic

Note the clear, almost circular alignment of the ring at the base of the shuttle model in figure 4.10

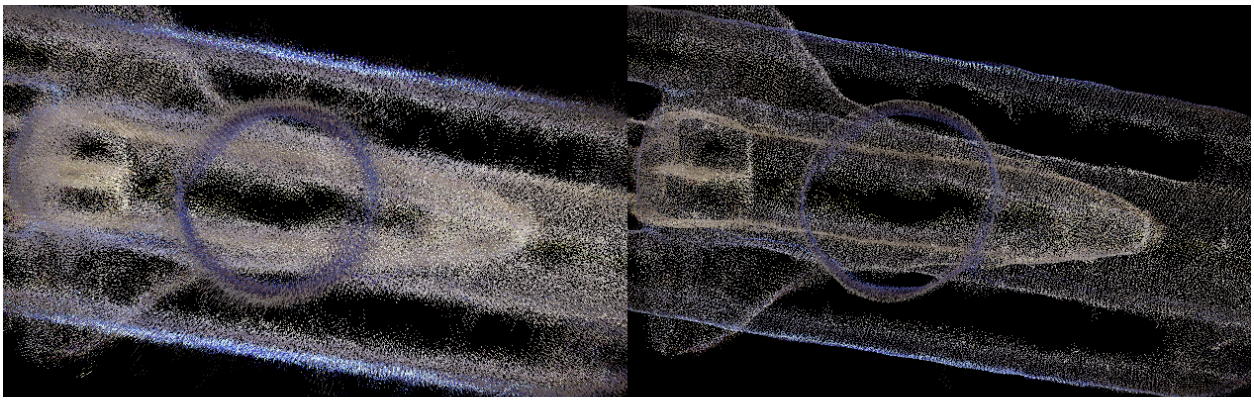

Figure 4.10: Shuttle alignment using purely binary EFPFH loop closure heuristic

4.3 Inertially fixed observer, 2-plane symmetric target

The similarity-based heuristics produced a large amount of false positive loop closure detections in the 2-plane symmetric data set. Figure 4.11 shows one example in which the binary EFPFH loop closure heuristic is unable to detect that only one side of the tank and SRB assembly has a Mylar-covered mounting bracket. This is most likely because the size of the bracket is very small relative to the rest of the tank and SRB assembly. The tank and SRB assembly is also a set of three cylinders for most of its vertical cross-section, producing very little variation in topography.

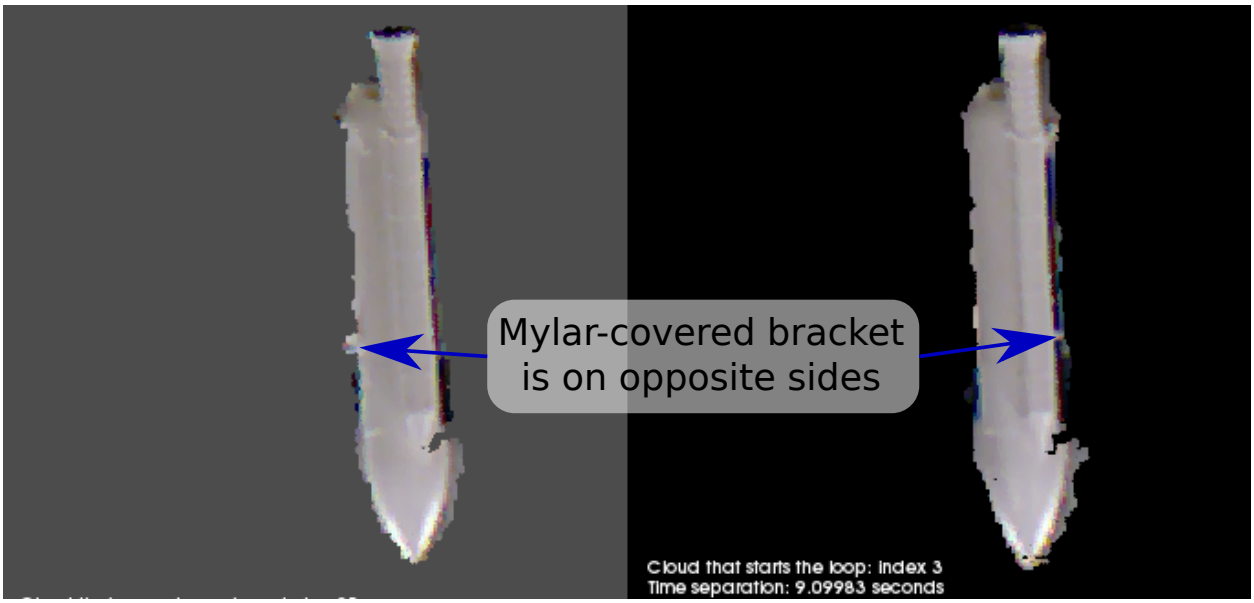

Figure 4.11: False positive closure detection from binary extended point feature histogram.

This weakness of the EFPFH heuristics against smooth, symmetric objects agrees with the theory behind the EFPFH. Irregularly shaped objects with rapidly changing topography have a wide variety of surface angles, producing a distinctive EFPFH signature. The similarity heuristics should be most effective when each frame has a distinctive EFPFH signature. Regularly shaped, smooth and symmetric objects produce less unique EFPFH values, which is expected to cripple the EFPFH-based heuristics for this type of object.

Figure 4.12 shows that both the binary and floating point EFPFH loop closure heuristics are saturated and ineffective for a two-plane symmetric target. The binary EFPFH does not dip below $P_{\text{Closure,min}} = 0.9$ and the floating point EFPFH heuristic is within one standard deviation of its mean value for the entire data set.

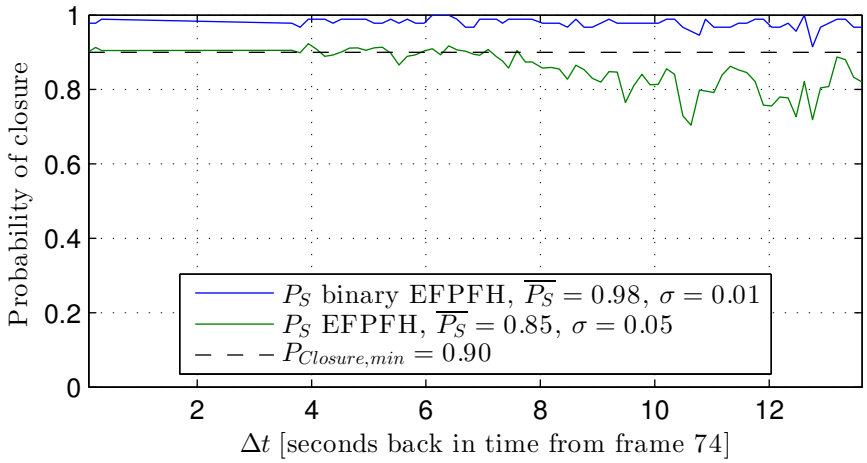

Figure 4.12: Comparison of binary and floating point P_S loop closure, shuttle-encoder frame 74

Figure 4.13 shows that the behavior of the loop closure heuristic against the two-plane symmetric target is dominated by P_β from the EKF, including its phase error and odometer drift. This makes the automated 3D model generator's performance very sensitive to filter tuning for the 2-plane symmetric target.

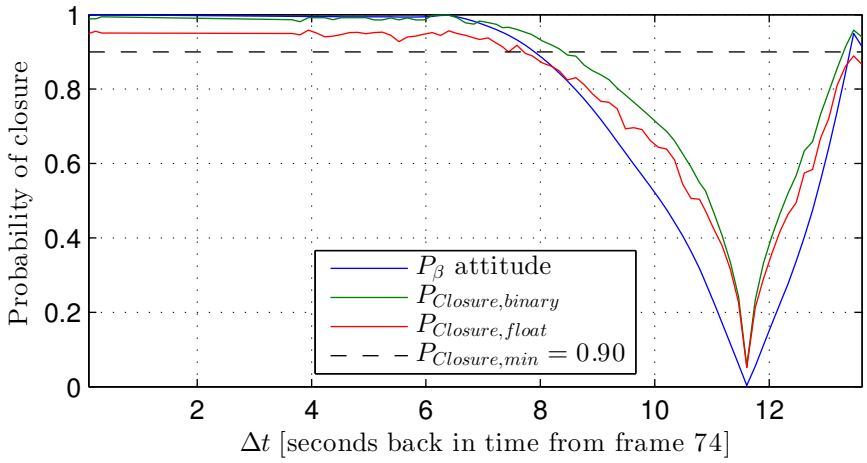

Figure 4.13: Joint P_{Closure} of loop closure, shuttle-encoder frame 74

After extensive EKF tuning, varying the sample rate and voxel grid size and adding an *a priori* rotational rate estimate, Graph SLAM and the loop closure heuristics produced a representative 3D model of the tank and SRBs (figure 4.14). While this shows that Graph SLAM and the joint loop closure heuristic can work in this fringe case, it was only with extensive human-in-the-loop parameter tuning and 2σ outlier observation rejection in the EKF.

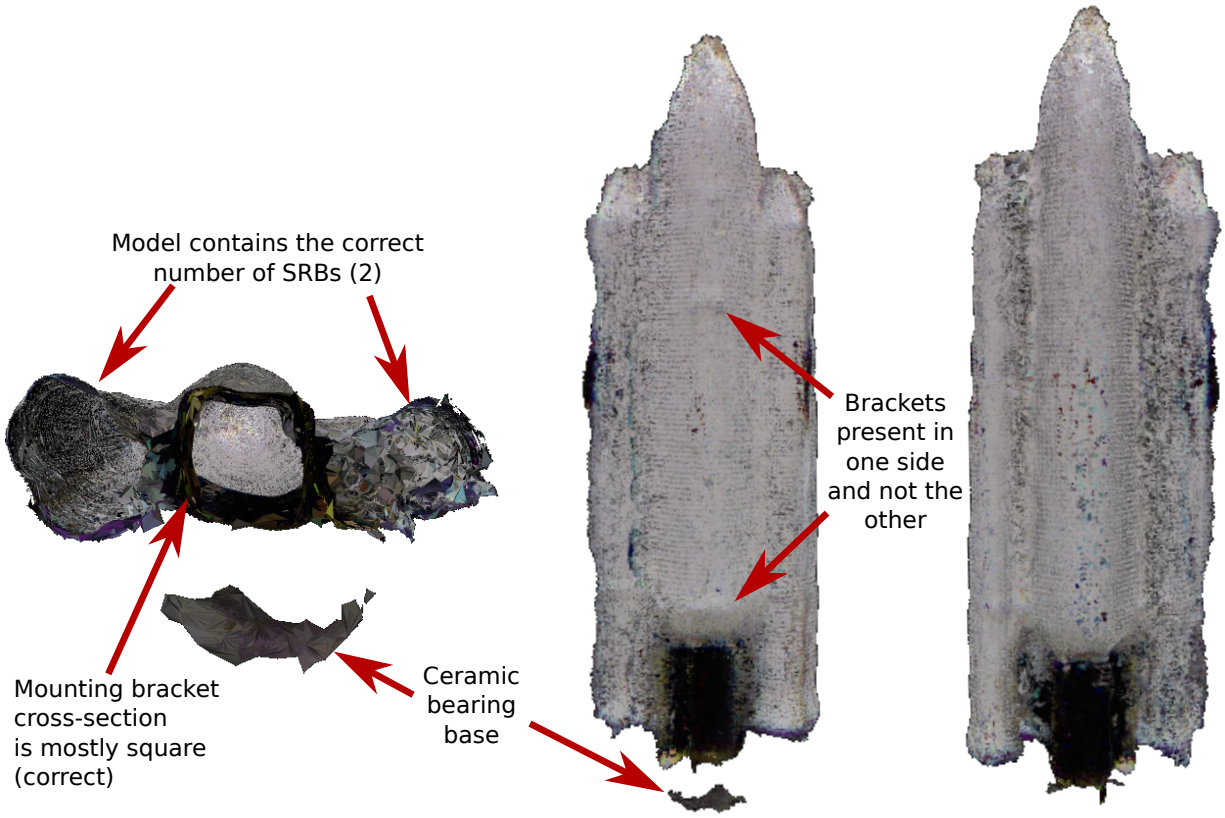

Figure 4.14: 3D model assembled from shuttle-encoder Microsoft Kinect^{TM} data set

It is important that the alignment produced a model that only contains two SRBs. Several of the failed attempts at model generation based on the EKF and autonomous Graph SLAM loop closure detection converged on a model that had three SRBs. The reason for this goes all the way back to the foundation of Graph SLAM.

Graph SLAM creates a fictitious spring network between several observed point clouds. Like real spring networks or control systems, the Lu-Milios graph network has equilibria, poles and zeros. By design,

we are searching for a configuration in which one node eventually forms a feedback loop with a previously observed node. Adding more frames to the Graph SLAM network is equivalent to adding more masses and springs to a mass-spring system. This produces more poles and zeros (stable and unstable equilibria). By iteratively relaxing this spring network, Lu-Milios Graph SLAM is pushing the alignment system through a dynamical alignment state-space toward an equilibrium point. The relaxation algorithm converges when it reaches a stable equilibrium point. Convergence of Lu-Milios Graph SLAM does not imply that the stable equilibrium point it converged on is the best alignment of frames.

Table 4.2: Successful 2-plane of symmetry model generation settings

Parameter	**Value**
Data set	shuttle-encoder
Sample rate	10 Hz
Lu-Milios convergence tolerance	10^{-7}
Maximum outer loop alignment iterations	50
Maximum inner Lu-Milios relaxation iterations	10
Process noise magnitude σ_u^2	0.5
Observation noise magnitude σ_v^2	d_k (ICP fitness)
$P_{\text{closure,min}}$	0.80
A priori initial state $\overline{\mathbf{X}}$	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 3 & 0 \end{bmatrix}^T$
A priori covariance $\overline{P}$	$I_{7\times 7}$
Voxel grid downsample size	2mm
Closure search method	greedy
Maximum correspondence distance	1cm
Pass-through point filter dimensions (m)	$-0.15 \leq x \leq 0.05$ $-0.25 \leq y \leq 0.05$ $0.58 \leq z \leq 0.80$

The most sensitive parameters for this data set were the sample rate, process noise magnitude, minimum closure probability and *a priori* state/covariance for the EKF. The data set is extremely short and the object comes to rest quickly (8 seconds after first observation, figure 4.15), so the EKF has very little time to converge on an estimate.

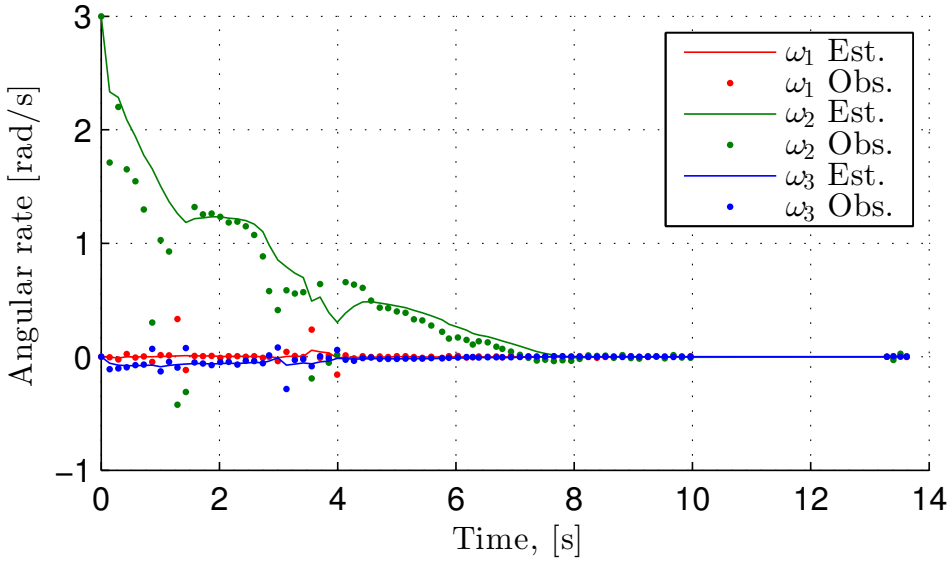

Figure 4.15: EKF angular rate estimate and frame-to-frame observations, shuttle-encoder data set

This is a shortcoming of the experimental apparatus that is not representative of Earth orbit environment. In Earth orbit, debris is expected to be in primarily major axis rotation at low, almost constant rates for the period that the debris mitigation spacecraft observes the debris. This suggests that if a space-based system were permitted to observe its debris target for a longer time period, it may not need the extensive EKF hand-tuning and *a priori* state knowledge that this experiment required (autonomy may be possible).

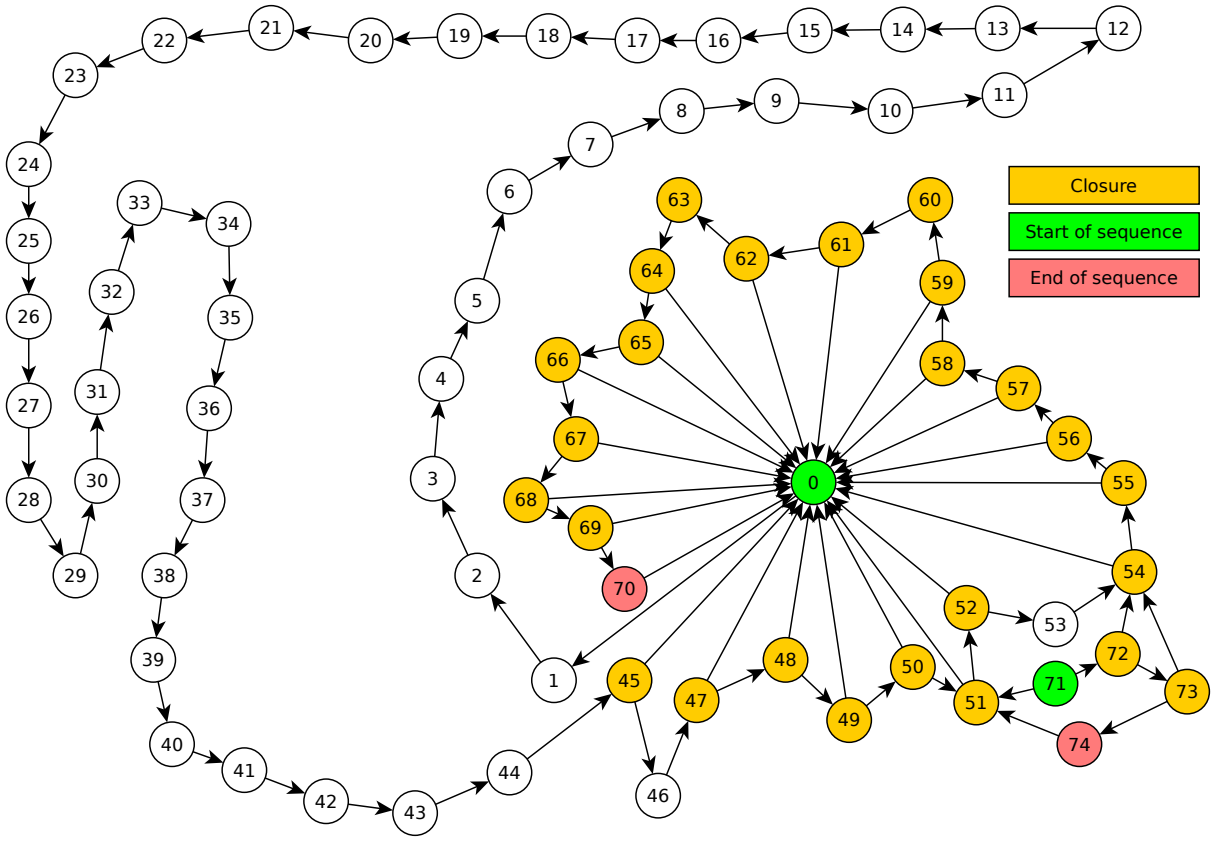

Figure 4.16: Successful 2-plane symmetry closure graph

The closure graph in figure 4.16 provides some insight into why the automated model generator was able to reconstruct the shuttle-encoder data set. The tank and SRB model rotated exactly one time before coming to rest. The most important loop closure was from 0 to 45 to 0, covering one revolution of the target. After the object came to rest, most frames are connected to their predecessor and successor, but also to frame 0. The subgraph from 71 to 74 is not part of the primary 0 to 70 cycle because there was a large time gap between frames 70 and 71. Graph SLAM was still able to use these frames to refine the model because they formed loop closures with frames 51 and 54 (like patching a second pass on top of the first pass).

None of the loop closures move backwards while the tank and SRBs are rotating. The first closure is detected at frame 45, and no frame after frame 45 closes a loop with a frame less than frame zero. The closures that move backwards after the target came to rest are not a problem because they are all second order closures with frame 0.

4.4 Model generation with relative motion between observer and target

The Levitron experiment with the full Space Shuttle model, external tank and two SRBs was repeated, but with the Kinect suspended from above. The Kinect was pulled back by hand, then released and allowed to swing freely, producing an oscillatory motion relative to the target object. Figure 4.17 shows that the swinging motion of the suspended Kinect^{TM} produced sinusoidally oscillating relative velocity and traced a closed helical path similar to the effects predicted by the Clohessy-Wiltshire equations (figure 2.14).

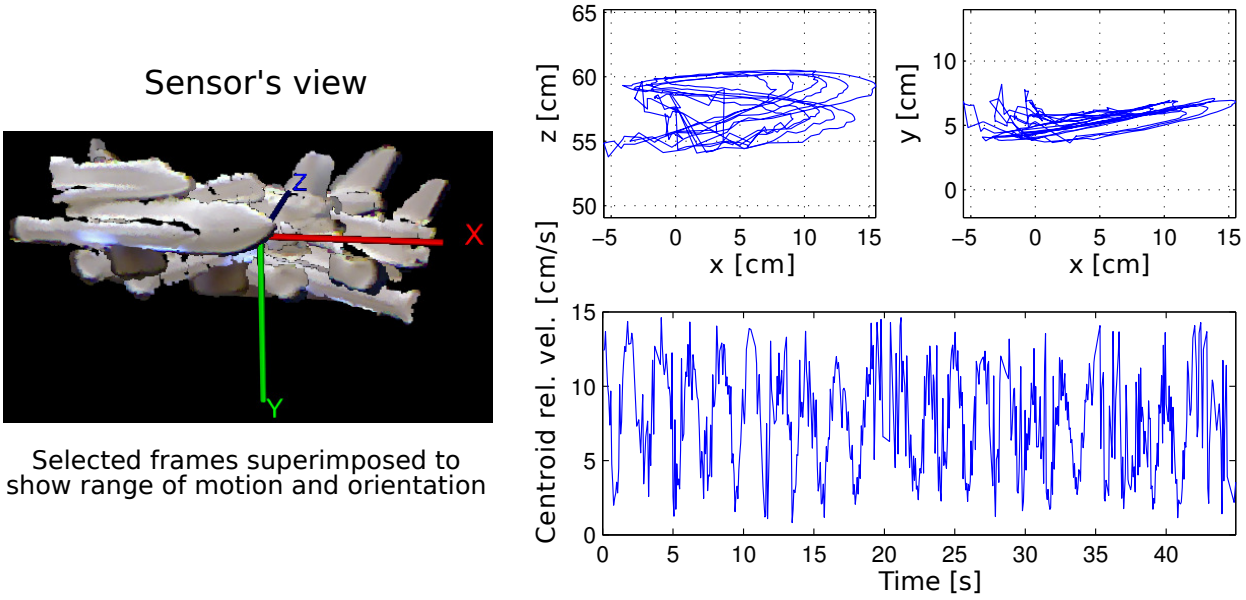

Figure 4.17: Relative motion observed by the Kinect sensor

This experiment also compared two different variations of the closure detection phase. The greedy version allows up to three preceding to be connected to the frame being analyzed if the joint probability of closure exceeds 0.7. Under this algorithm, a frame may also be connected to any other frame that occurred at least $1/2$ of a rotational period prior with which it has a joint probability $P_{Closure} \geq 0.7$. This produces a larger number of frame-to-frame connections (hence the name "greedy"). The frugal method, on the other hand, searches for all loop closures, then down-selects to the single highest probability loop closure in the maximal (largest) subgraph.

Graph Slam and the joint loop closure heuristic was able to partially construct a 3D model of the Space Shuttle. While aspects of the form are correct, the alignment of the magnetic platform is incorrect. The tail fin is also incorrectly aligned.

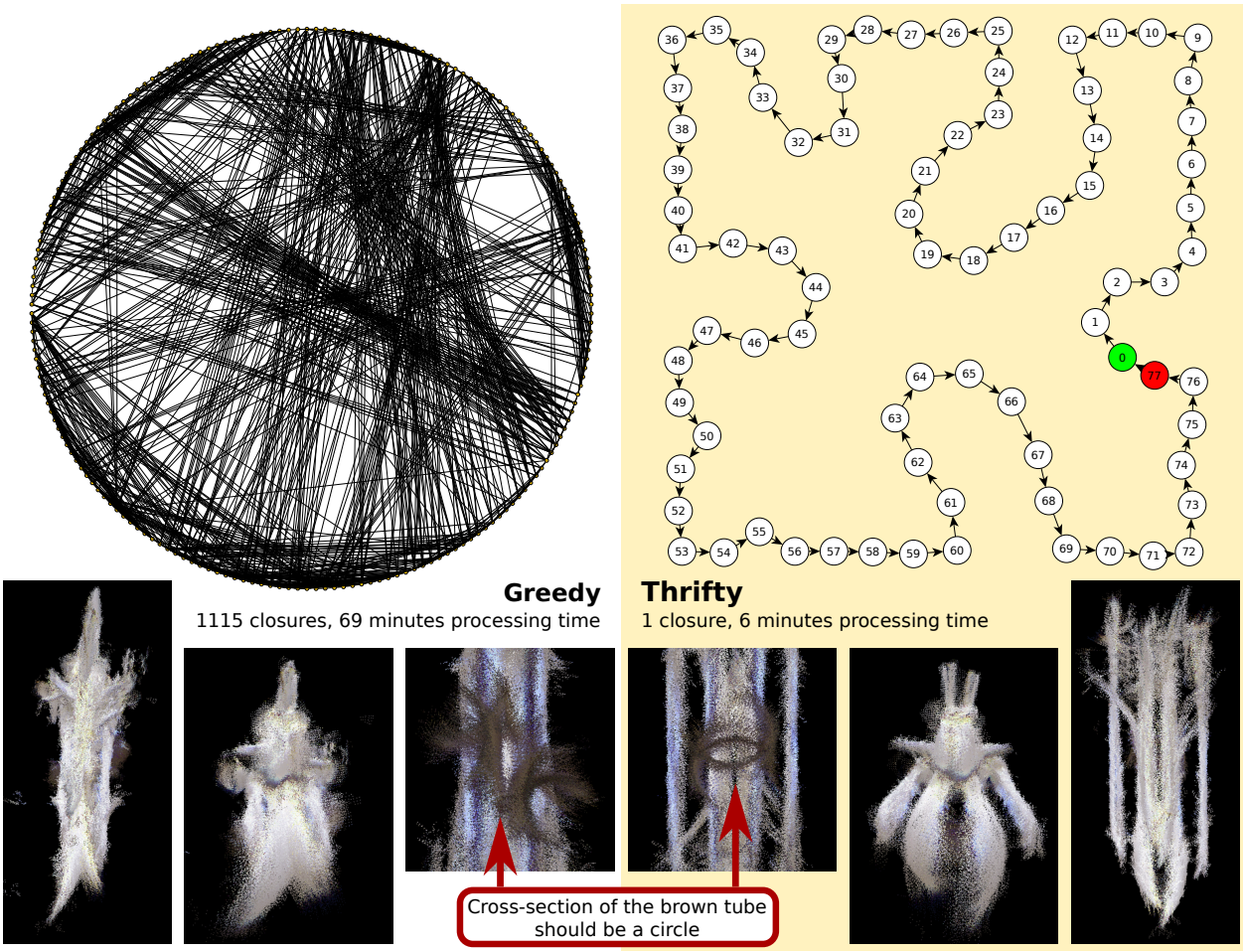

Figure 4.18: Graph of the greedy and thrifty loop closure algorithms used with the moving target

The loop closure diagram (figure 4.18) for the greedy variant looks like a partially shaded in circle. The center is actually crisscrossed with 1115 frame-to-frame closure edges (arrows), each of which is a fictitious spring that must be adjusted using the Iterative Closest points algorithm on two point clouds with approximately 10000 points each. The greedy closures algorithm solves a 1115-dimensional vector optimization problem, while the thrifty algorithm optimizes a 77-dimensional problem. Each each dimension of optimization is itself a nonlinear multiple-dimensional optimization problem for an alignment quaternion

$\vec{\beta}$ and translation vector $\mathbf{t}$ between the two connected clouds.

The large number of loop closures has a negative impact on the computational resources required to align the frames. To align the frames in the configuration listed in figure 4.18 required 1 hour, 9 minutes on an AMD A8-4500M APU. Most of the processing was single-threaded. The frugal method, which used only one observation loop, completed in required 6 minutes and produced more geometrically consistent results.

Table 4.3: pclAutoScanner settings for model generation with relative motion

Parameter	**Value**
Data set	shuttle-end-on-swing
Sample rate	5 Hz
Lu-Milios convergence tolerance	10^{-9}
Maximum outer loop alignment iterations	40
Maximum inner Lu-Milios relaxation iterations	40
Process noise magnitude σ_u^2	0.5
Observation noise magnitude σ_v^2	d_k (ICP fitness)
$P_{\text{closure,min}}$	0.7
A priori initial state $\overline{\mathbf{X}}$	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0.2795 & 0 \end{bmatrix}^T$
A priori covariance $\overline{P}$	$I_{7\times7}$
Voxel grid downsample size	2mm
Maximum correspondence distance	3cm
Pass-through point filter dimensions (m)	$-0.25 \leq x \leq 0.25$ $-0.35 \leq y \leq 0.11$ $0.47 \leq z \leq 0.65$

The most sensitive parameters for this data set were the number of iterations and the maximum correspondence distance. The relative motion decreases the accuracy of the EKF's attitude estimate, which means that the pre-rotation at the start of the alignment has more error (two frames that should be aligned

start out with larger orientation differences). The ICP alignment algorithm will produce incorrect alignments if it has insufficient correspondences between the two frames. To improve the algorithm's performance, the correspondence estimator was allowed to search up to 3cm away from a current point (three times larger search radius than normal). This is an engineering trade-off; larger correspondence distances make misalignments less likely, but significantly increase run-time because more points must be compared against each other during the correspondence estimation phase.

Chapter 5

Discussion

5.1 Validity of the dynamics model

The results of the batch processor estimation of the angular rates (figures 4.1 and 4.2) showed that the dynamics model is partially correct. The model is right about motion being primarily about the $+y$ axis, but torque free motion was a bad assumption. There may also be some small off-axis torque due to imbalance, but the angular velocity observations are too noisy to draw any conclusions.

The incorrectness of the dynamics model may not be a problem. The best performance came when the process noise matrix was tuned to be as large as possible while still rejecting frame-to-frame misalignment outliers. This may be appropriate for an autonomous model generation system because the true rotational dynamics of an unconstrained object are governed by its mass distribution (inertia) and torques imparted on it. The problem is that the autonomous inspecting spacecraft doesn't know anything about the target's shape, inertia, attitude stabilization, or any other torques acting on it.

While these could possibly be estimated by adding them to the EKF's state vector, the relationships between them are complicated. Some may not even be observable because the observer only has one sensor and a relatively short time period of angular rate measurements. Davison's VSLAM structure from motion (SfM) algorithm addresses this same challenge by assuming some linear motion between the camera frames and using a large amount of process noise to prevent the EKF's covariance matrix from collapsing [53]. The results of the experiments (chapter 4) suggest that applying Davison's approach to this problem is accommodating the error in the dynamics model well enough for the joint loop closure heuristic.

5.2 Need for Graph Pruning

Figure 5.1 shows a graph that was produced by an early revision of the loop closure detection function. While it can in some cases be valid to say that one frame 72 could be a loop closure with both nodes 1 and to, the overall trend of the closures in figure 5.1 is not consistent with the physically observed motion. The sequence of the loop closures implies that the object changed its direction of rotation shortly after frame 68 closed a loop with frame 5, then reversed its rotation direction again, approximately 3 seconds later.

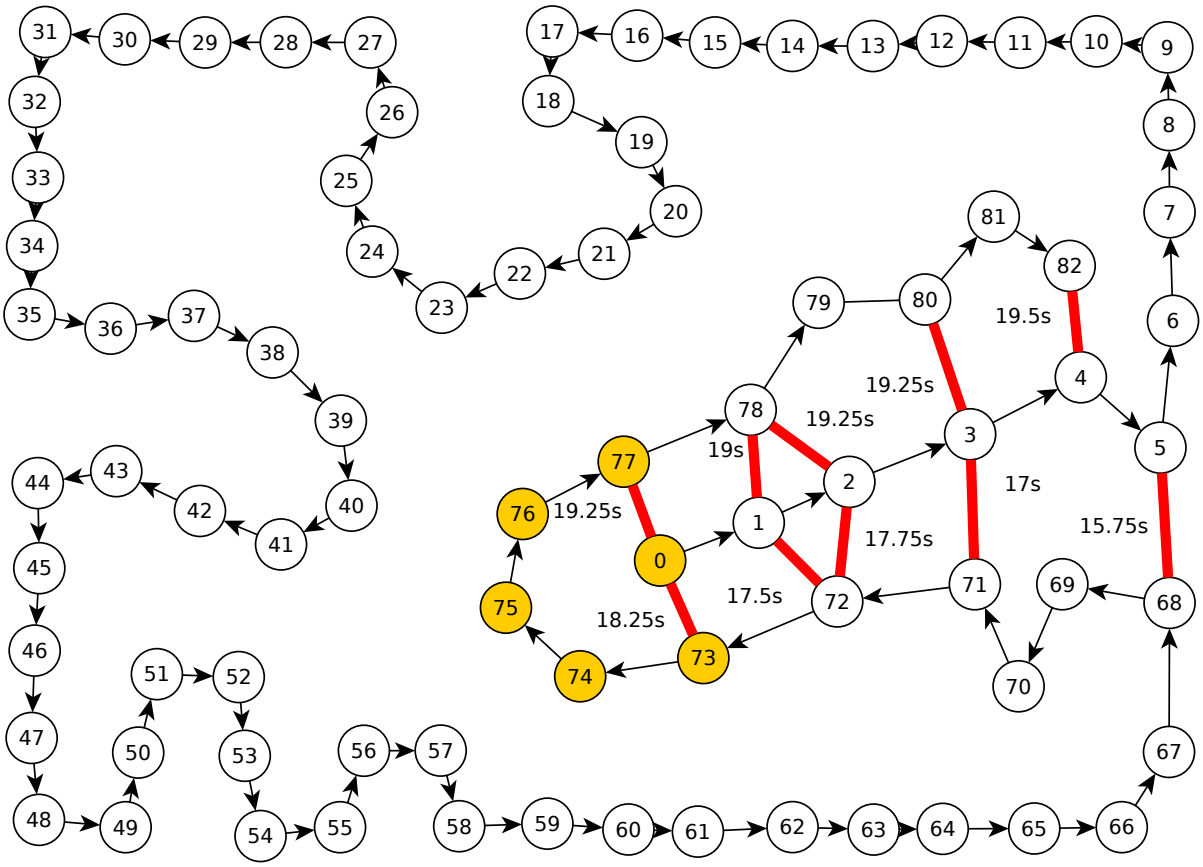

Figure 5.1: Unpruned graph that includes false positive loop closures. Red lines indicate loop closures.

This produced an incorrect frame alignment that caused parts of the fuselage to pass through each other (inconsistent surface).

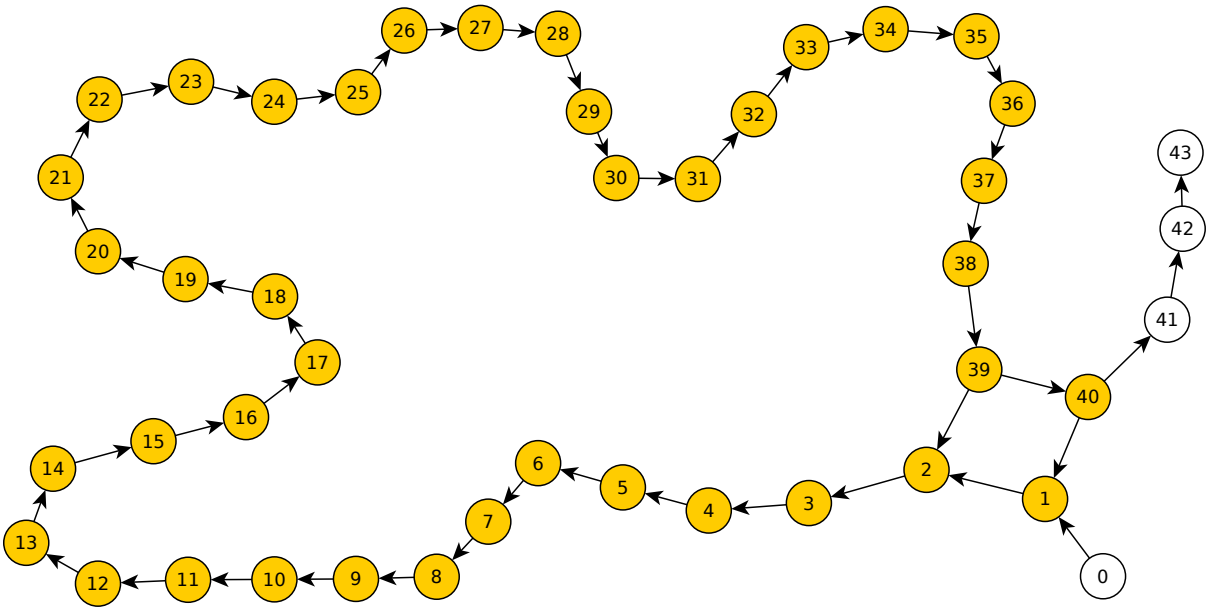

Figure 5.2: Frame connection graph for the shuttle-end-on-redone data set

In Graph SLAM, an edge between nodes 7 and 8 means that Graph SLAM will use the ICP algorithm to align the points in node 7's point cloud to the points in node 8's point cloud. In figure 5.2, the yellow nodes are part of a cycle, so Graph SLAM will adjust the relative alignment of the frames until the alignment error d_k between frames is minimized.

If the closure is just a single loop, this also corresponds to equally shared error (as noted by Nüchter et al. [37]). If the closures form a more complicated structure, error may not be distributed equally. The link between node 38 and 39 may have more error than the link between node 39 and 40. The serial connection path between nodes 2 and 39 may also cause the error between nodes 38 and 39 to be distributed amongst nodes 2-38, which pulls apart frames that would otherwise align very closely.

Nodes 0, 41, 42 and 43 are a problem because they are not part of a cycle. Graph SLAM will align each frame to its neighbor, but frame 43 has no outgoing edge, so it is dangling off of the model. Graph SLAM may converge with a chain of fragments (41-43, 0, or both) that is not aligned with the rest of the model body. Figure 2.12 shows an extreme example of this, where the model is aligned from a chain of frames with no closures.

One of the earlier strategies for loop closure detection was to detect as many repeated views of the same face as possible, then interpolate loop closures between the repeated observations. The misalignment at the external tank's nose in figure 5.3 was corrected by modifying the closure detection function to prune back the repeated closures to just a single loop with a single closure (a ring).

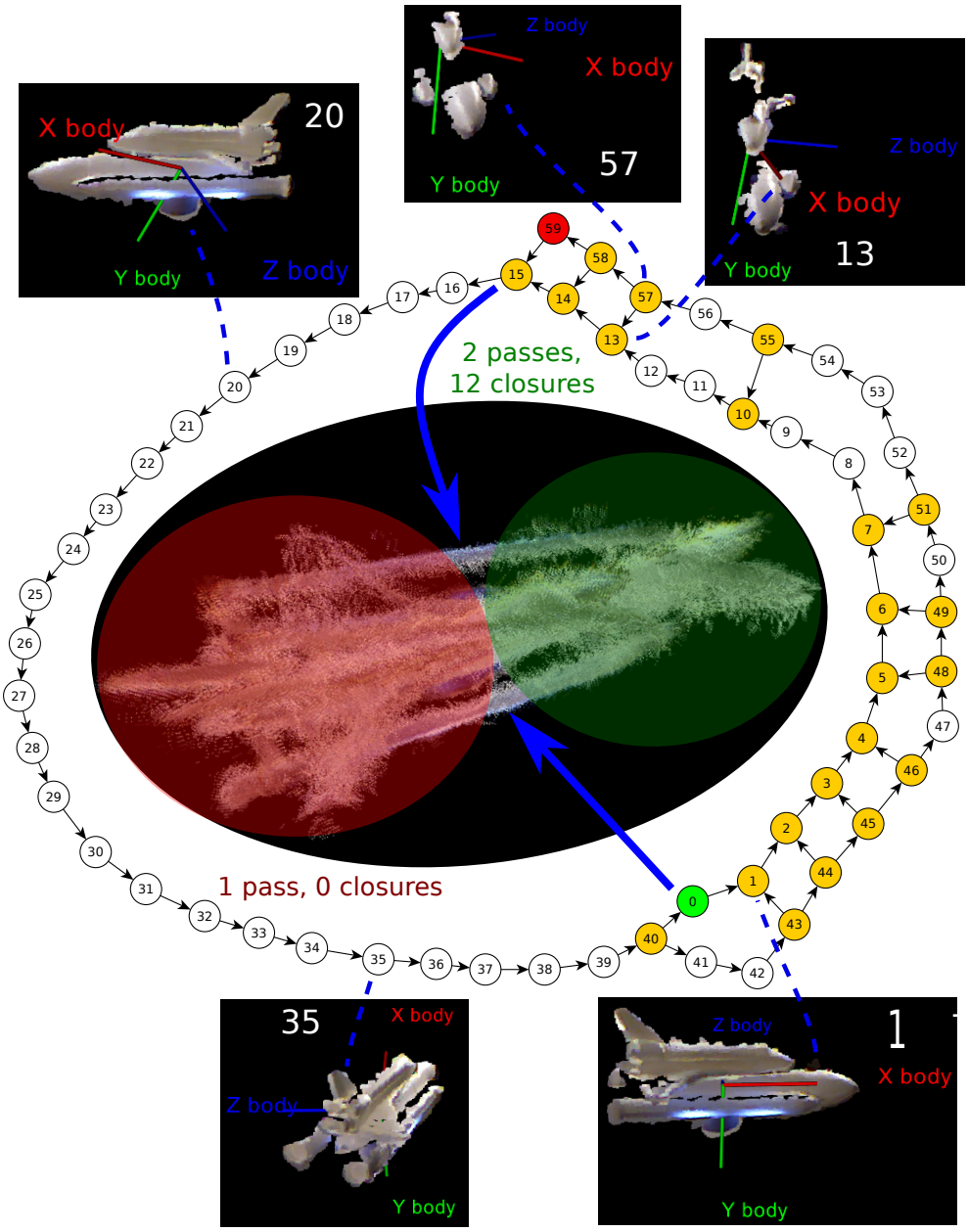

Figure 5.3: Lopsided loop closures compress one end of the model and stretch the other

5.3 Effectiveness the loop closure heuristics

The EFPFH-based similarity heuristics $P_{s,b}$ and $P_{s,f}$ are heuristics that should discriminate between shapes. This shape discrimination is then used to correct phase lag and odometer drift in the attitude-based heuristic P_β. This implies that $P_{s,b}$ and $P_{s,f}$ should disagree with the attitude estimate somewhat. If their distributions favor 1, then they lose their ability to shape the attitude estimate and move the $P_{closure}$ peaks back to the correct location in time. Similarity heuristics whose distribution favor 1 are poor heuristics because it is effectively a statement that the software should blindly trust the attitude estimate because everything looks the same. Similarity heuristics that favor 0 are more effective in the joint loop closure probability heuristic because they have more authority when multiplied with the attitude estimate.

To examine the effectiveness of the similarity heuristics, the probabilities of loop closure for the last half of all the data sets is analyzed in this section. The source data came from four data sets: shuttle-end-on-short, shuttle-end-on-redone, shuttle-end-on-swing and shuttle-end-on-off-axis. This forms a sample pool of $N = 71277$ probabilities of loop closure in the statistical analysis.

The Shannon Entropy of a discrete data set is a measure of the amount of variation in the data set if it is modeled as a stochastic process [52]. As defined by Shannon in [52], the information entropy H of a data set of length N is a function of the probability mass function $p(x_i)$[1] :

$$H = -\sum_{i=1}^{N} p(x_i) \log_2 p(x_i) \ \text{[bits]} \tag{5.1}$$

A smaller entropy value represents less variation in the data set (as in thermodynamics). Figure 5.4 shows that the binary EFPFH heuristic favors 1, while the floating point EFPFH similarity heuristic favors 0.5. The binary EFPFH heuristic also has approximately 1/3 the entropy of the the floating point EFPFH giving it less ability to distinguish shapes. Because the binary EFPFH is clustered near 1 with less variation than the floating point heuristic, it should be less effective at correcting the phase lag in the attitude-based probability of closure P_β.

[1] This thesis takes the same convention of setting $K = 1$ and using base 2 to represent binary data structures that Shannon established in [52].

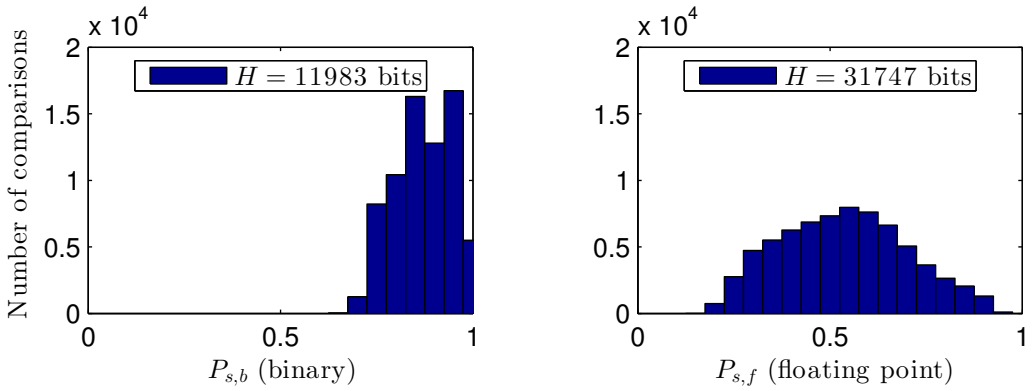

Figure 5.4: Comparison of binary $P_{s,b}$ and floating point $P_{s,f}$ probabilities of loop closure

The binary EFPFH favors 1 because it is a lower resolution quantization of the floating point EFPFH that reduces the data resolution from 10^7 discrete values to 180 discrete values, most of which are below the quantization threshold. Figure 5.5 shows a representative binary EFPFH quantization. The variation in the first 20 histogram bins is discarded, as is much of the variation between bins 150-165.

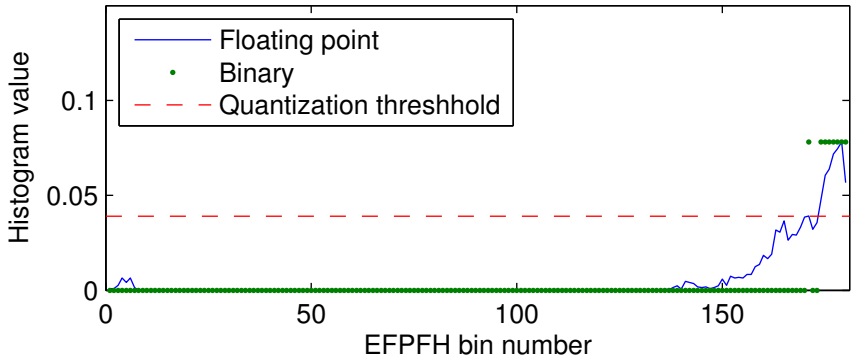

Figure 5.5: Representative binary EFPFH quantization (frame 143 of shuttle-end-on-short)

The quantization threshold is set to the histogram median, which varies from point cloud to point cloud. Over the shuttle-end-on-short data set, the mean value of this median threshold was 0.04 with a standard deviation of $\sigma = 0.02$. Switching the quantization point to the sample mean increases the bias toward 1 and decreases entropy H, making the mean a less effective quantization threshold than the median for the shuttle-end-on-short data set (figure 5.6).

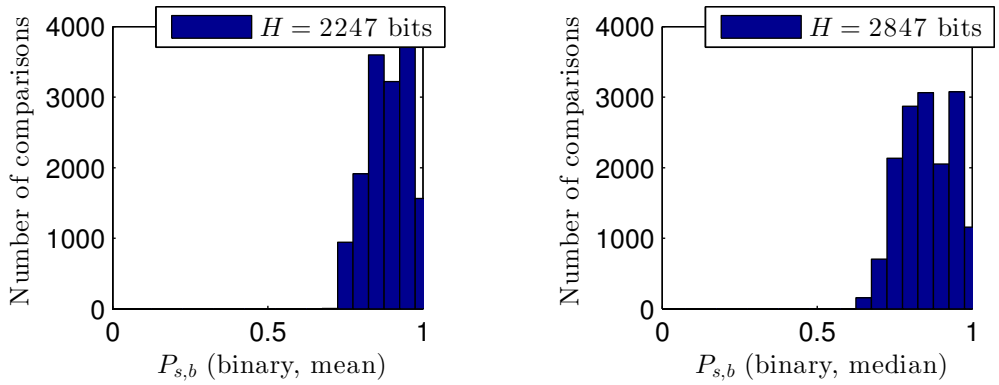

Figure 5.6: Impact of quantization point for binary $P_{s,b}$ probability of loop closure (shuttle-end-on-short)

The binary EFPFH's lack of entropy and bias towards 1 carries through to the joint probability of loop closure (figure 5.7), biasing it towards loop closure (bad for correcting the EKF).

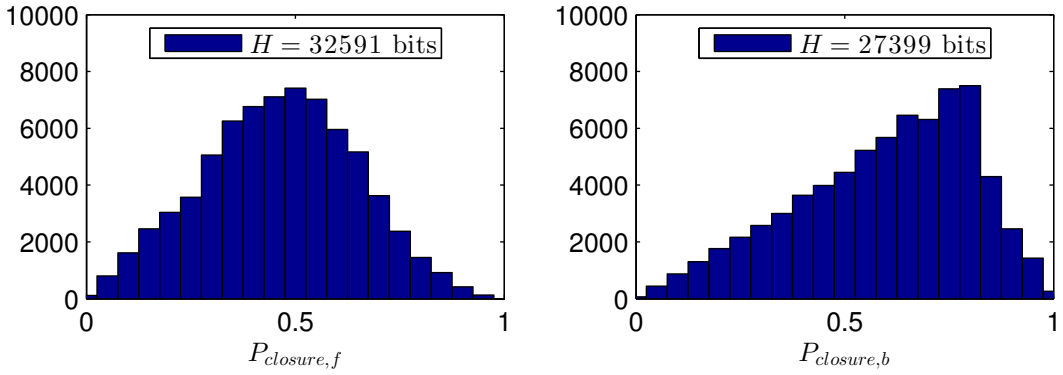

Figure 5.7: Histogram of joint probability of loop closure for binary, floating point EFPFH

The joint heuristic entropies are on the same order of magnitude, however. This suggests that the binary EFPFH could be as effective as the floating point EFPFH in the joint statistic if the distribution were shifted toward the middle and spread out. One way might be to re-normalizing it based on empirically observed values. This is a risky strategy because it is difficult to know that enough samples have been observed to estimate the variance of the probability function. To provide a margin in case insufficient samples have been observed, it may be re normalized to a 6σ window about the mean:

$$P_{s,b,\text{normalized}} = \frac{1}{6\sigma}\left(P_{s,b} - \overline{P_{s,b}}\right) + 0.5 \tag{5.2}$$

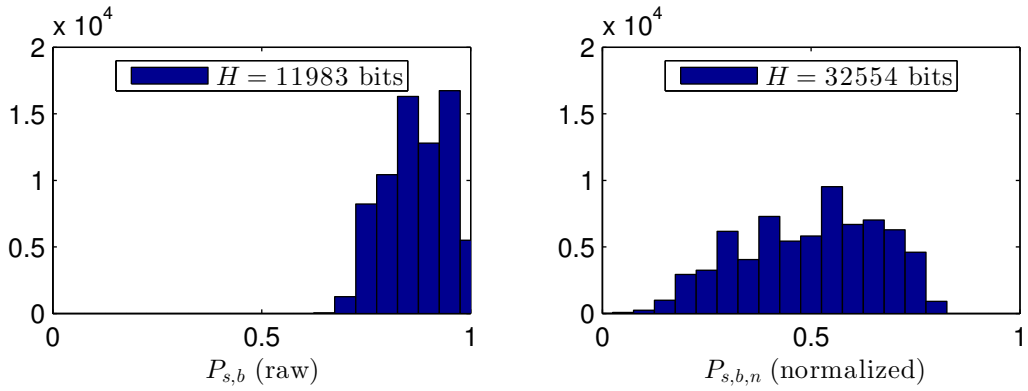

Figure 5.8: Impact of changing quantization point for binary $P_{s,b}$ probability of loop closure

Figure 5.9 suggests that this normalized binary loop closure heuristic may provide loop closure detection performance comparable to the floating point EFPFH. Time constraints on this thesis prevent further exploration of this idea, so it is left as a recommendation for future work.

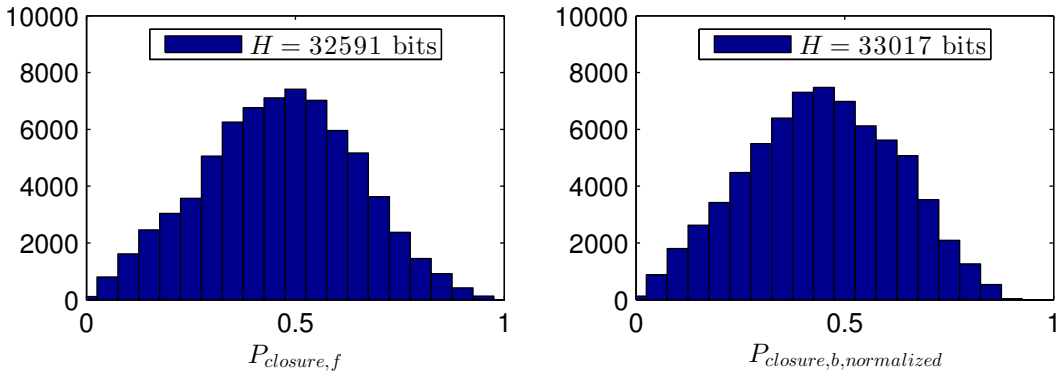

Figure 5.9: Floating point and normalized binary EFPFH loop closure histograms

5.4 Performance Against Different Shapes

All three systems are very sensitive to EKF and alignment tuning parameters. It is unclear if this is due to the types of object being scanned, the quality of the Kinect^{TM} data, or shortcomings in the algorithms and their software implementations.

In general, objects with more planes of symmetry were more difficult to reconstruct. Early testing attempted to scan a rectangular prism with little success (figure 5.10). Experiment one produced the highest

quality models in this research, but also had one of the easier data sets (a shape that varies and little relative motion). Experiment three required the least tuning of all all of the experiments, but it also did not produce the highest quality models. Experiment 2, with two planes of symmetry and longitudinal rotation, required the most tuning and most time to produce a successful autonomously generated model.

Based on these experiments, upper stages of rocket bodies are expected to be the most difficult objects for an autonomous 3D debris scanner, closely followed by spin-stabilized, axisymmetric spacecraft. Spin-stabilized satellites may be easier to scan then rockets, because some of them have have instruments or antennas that only face one direction. Satellites with only one plane of symmetry should be next easiest, followed by collision debris and asteroids. Collision debris and asteroids are expected to be the easiest shapes because of their irregularity and asymmetry.

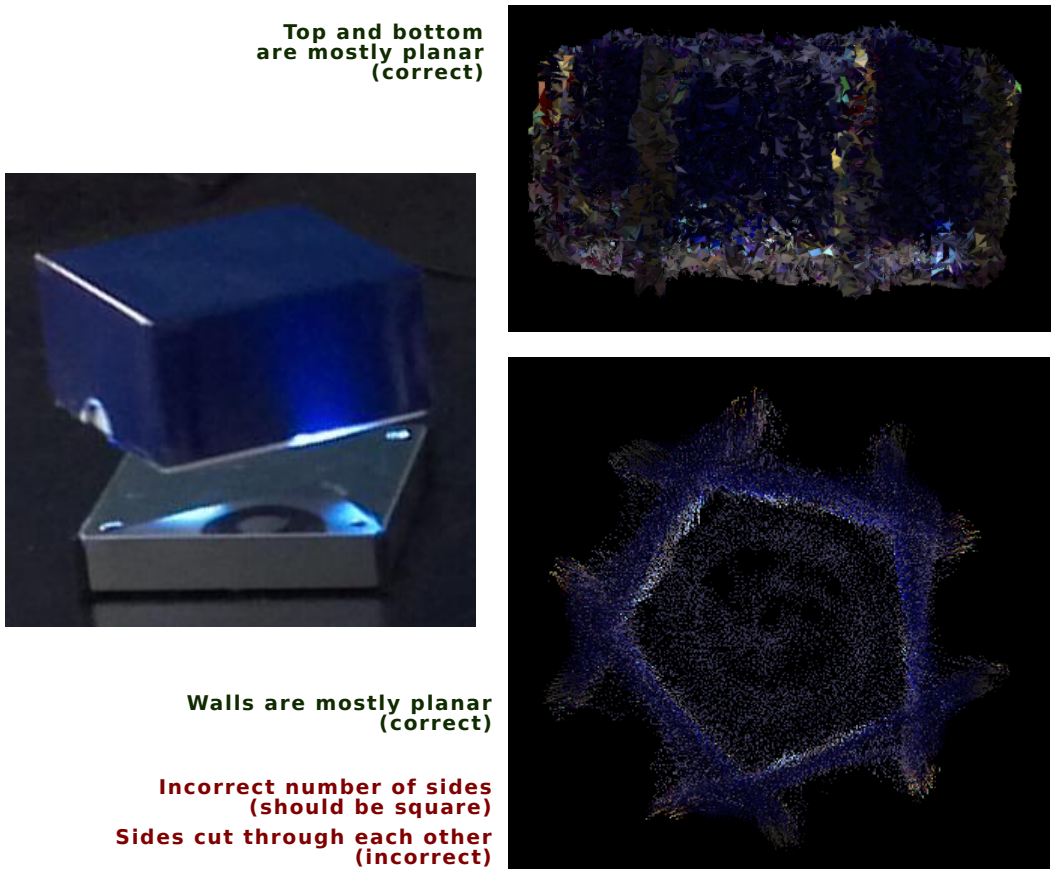

Figure 5.10: Multiple planes of symmetry and flat sides are specific weaknesses (rectangular prism)

Chapter 6

Conclusion

6.1 Summary

The combination of point cloud sensors, the Iterative Closest Points (ICP) algorithm, Graph SLAM and two tailor-made autonomous loop closure heuristics were examined, derived and experimentally tested using a plastic model of a Space Shuttle and 3D point cloud data from a Microsoft Kinect^{TM}. Purely least squares alignment methods were found to be insufficient for model generation due to odometer drift.

Methods that relied entirely on similarity were found to be capable, producing the highest quality model 3D model in these three experiments, but were too reliant on human selection of start and stop times for the model generation window. They also were too inflexible, completely failing to create a model of the two-plane of symmetry external tank and solid rocket booster model.

Shaping the attitude based heuristic with a similarity based loop closure heuristic allowed the heuristics to complement each other, with the similarity heuristic compensating for the EKF's phase lag and the EKF nulling out the false positive when the similarity heuristic sees a mirror image. This combination was able to autonomously produce a model from at least one data set per experiment, including one involving relative motion. The EKF propagation also helped the loop closure heuristic tolerate some small data gaps.

Graph SLAM's ability to distribute alignment error and rectify some of the frame-to-frame misalignments was also a key piece of this success. The Graph SLAM algorithm is expected to perform well in an orbital debris scanning role based on the results of these experiments.

The binary EFPFH-based loop closure heuristic was compared against the floating point EFPFH loop

closure heuristics and found to have insufficient dynamic range to correct the EKF's phase lag. While there may be a way to improve the binary EFPFH's performance by renormalizing it, the floating point EFPFH loop closure heuristic is recommended over the binary EFPFH until the renormalized binary EFPFH has been tested and analyzed.

The key, overall finding is that a combination of point cloud sensors, a joint loop closure heuristic based on floating point EFPFH comparison and an EKF attitude estimate, the Iterative Closest Points and Graph SLAM algorithms is capable of autonomously building 3D models of unfamiliar, uncooperative objects in 1D planar rotation with minor off-axis perturbations and relative motion. Moreover, this combination can autonomously build 3D models in spite of intermittent data dropouts and inconsistent visibility of the target and relative motion between the target, suggesting it may be robust to some of the orbital debris challenges.

6.2 Recommendations for Future Work

Several interesting aspects and possibilities were uncovered during this research. There are also some shortcomings to the present proof-of-concept that should be extended in further work. This section summarizes some key ways to carry this work forward.

6.2.1 Investigate the Dynamics of the Spring-Mass Graph SLAM Network

The spring-mass analogy presented in [53] is particularly apt. Prior to implementation of the graph pruning function, some of the closures detected by the similarity heuristic would connect a spring mass network that was unstable, causing the algorithm to throw several fragments away from the center of the model. One unstable graph network stretched all of the nodes out in a straight line and continued stretching them more and more with each relaxation iteration.

Thinking of the Graph SLAM network as a dynamical system may allow for quick checks that can identify an unstable graph. If the error functions are linearized around their present value, the rate of change of the error function can be approximated as a linear system using the direct state space method ($\dot{\vec{\epsilon}} = A\vec{\epsilon}$). Calculating the eigenvalues and eigenvectors of A would yield a set of stable and unstable equilibria and their locations in the ICP error-space. The software system could then use this to check the current error

state of the frame alignment for proximity to an unstable equilibrium point and reject graph networks that start near unstable equilibria as bad hypotheses.

6.2.2 Create a Custom Depth First Search for Graph SLAM Pruning

The boost library's default depth first search visitor template can be used to identify cycles, but it has some undesirable behaviors for this particular loop closure detection scheme. One of the quirks is that it uses the node ID (which is the frame sequence number in this research) to determine whether the edge linking it to a node is a forward edge (outgoing) or a back edge (incoming). In certain cases, this prevents the graph pruning function from identifying nodes that are not part of a cycle. This could also allow the depth first search to segment the nodes by which cycle they belonged to, which would reduce the number of depth first searches that are performed.

6.2.3 Repeat Calderita's Kinect^{TM} Performance Characterization for MLI

Because Multiple Layer Insulation (MLI) blankets are likely to be involved in many debris scanning missions and shiny metallic surfaces have already been identified as a weak area for structured light, more detailed information on detection of objects wrapped in MLI is needed.

6.2.4 Repeat the experiments with a wider variety of shapes

The experiments discussed in this thesis all involve either a delta-winged spacecraft or a two booster and tank rocket assembly. Performance should be characterized against models of upper stages from launch vehicles, small satellites and spacecraft with solar panels. As suggested by Dr. Rhonda Morgan and Dr. Hanspeter Schaub, the asymmetry and topographical variation of a sweet potato may provide a good stand-in for an asteroid-like object.

6.2.5 Repeat the experiments with less constrained target motion

To robustly scan debris on orbit, the sensor/algorithm suite would have to handle a target that is in an arbitrary tumble. The experiments in this thesis only tested 1D planar rotation with small off-axis

perturbations and pendulum-like relative motion. Before this type of system is used to scan orbital debris, it should be tested against a target that is in an arbitrary tumble (unconstrained 3DOF rotation).

6.2.6 Increase the Robustness of the pclAutoScanner

All of the data processed for the experiments in this thesis was saved to a solid state drive and reloaded at each test run. A more realistic use would be for the software to continuously obtain point clouds from the Kinect, pause acquisition when it detects loop closure, generate a model, then switch either a tracking mode upon success or return to scanning mode if it fails to generate a model.

The pclAutoScanner's current frugal behavior is to choose only one loop closure, then attempt to prune it to a single loop. If the pruning process causes the loop to be broken, the pclAutoScanner fails due to failure to detect loop closure. Many times, there are several loop closures that satisfy the minimum criteria for loop closure. The pclAutoScanner should be modified to try some of the other loop closures that exceeded the minimum probability of loop closure but were not the highest probability detected.

6.2.7 Test the Performance of the Normalized Binary EFPFH Heuristic

Figure 5.9 suggests that the renormalized binary EFPFH described in section 5.3 may be capable of loop closure detection when combined with the attitude estimation EKF. There was insufficient time to test the normalized binary EFPFH during this research, but it could improve performance and reduce memory requirements for an autonomous 3D model scanner if it the normalization restores the binary EFPFH's ability to correct EKF phase lag.

Bibliography

[1] Evan Amos. Xbox 30 Kinect Standalone. Web (photo), August 2011.

[2] Austin Probe and John L. Junkins. Robotic simulation experiments demonstrating docking proximity operations and contact dynamics. In AIAA Modeling and Simulation Technologies Conference, AIAA SciTech. American Institute of Aeronautics and Astronautics, January 2014.

[3] Tadej Bajd, Matjaz Mihelj, and Marko Munih. Introduction to Robotics. Springer, 2013.

[4] Gioia Ballin. pcl::ISSKeypoint3D< PointInT, PointOutT, NormalT > Class Template Reference. Web (software API documentation), 2013 December.

[5] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 14(2):239–256, Feb 1992.

[6] Ayush Bhandari, Achuta Kadambi, Refael Whyte, Christopher Barsi, Micha Feigin, Adrian Dorrington, and Ramesh Raskar. Resolving multi-path interference in time-of-flight imaging via modulation frequency diversity and sparse regularization. Optics Letters, February 2014 (accepted, pending publication).

[7] Michael Cabajal. Nasa - shuttle. Web (3D model), October 2008.

[8] L Calderita, L Manso, and P Núnez. Assessment of primesense rgb-d camera for robotic applications. In Proceedings of the XIII Workshop of Physical Agents, 2012.

[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. Introduction To Algorithms. MIT Press, 2001.

[10] S D'Amico, Mathias Benn, and John Leif Jørgensen. Pose estimation of an uncooperative spacecraft from actual space imagery. In 5th International Conference on Spacecraft Formation Flying Missions and Technologies, 2013.

[11] Martin P. Dana. Multitarget-Multisensor Tracking: Applications and Advances, volume I, chapter 5: Registration: A Prerequisite for Multiple Sensor Tracking, pages 155–185. Artech House, 685 Canton Street Norwood, MA 02062 USA, 1990.

[12] F. Dunn and I. Parberry. 3D Math Primer for Graphics and Game Development, 2nd Edition. An A.K. Peters book. Taylor & Francis, 2011.

[13] D.E. Ensley and J.W. Crawley. Discrete Mathematics, Student Solutions Manual: Mathematical Reasoning and Proof with Puzzles, Patterns, and Games. Wiley, 2009.

[14] Agner Fog. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs. Technical University of Denmark, February 2014.

[15] Jiawei Han, Micheline Kamber, and Jian Pei. Data Mining: Concepts and Techniques: Concepts and Techniques. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011.

[16] D.M. Harland. The Story of Space Station Mir. Springer Praxis Books. Springer, 2007.

[17] Günter Hetzel, Bastian Leibe, Paul Levi, and Bernt Schiele. 3d object recognition from range images using local feature histograms. In Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, volume 2, pages II–394. IEEE, 2001.

[18] Paul Jaccard. The d0istribution of the flora in the alpine zone. New Phytologist, 11(2):37–50, 1912.

[19] Lee Jasper, Hanspeter Schaub, Carl Seubert, Valery Trushkyakov, and Evgeny Yutkin. Tethered tug for large low earth orbit debris removal. In 22nd AAS/AIAA Space Flight Mechanics Meeting, 2012.

[20] Lee E. Z. Jasper and Hanspeter Schaub. Discretized input shaping for a large thrust tethered debris object. In AAS/AIAA Spaceflight Mechanics Meeting, Santa Fe, New Mexico, January 2014.

[21] John L. Junkins and John L. Crassidis. Estimation of Dynamic Systems. Chapman & Hall/CRC Applied Mathematics & Nonlinear Science. Chapman and Hall/CRC, October 2011.

[22] John L. Junkins and Manoranjan Majji. Advances in real time computational vision. In UAS Video Tracking Workshop and Challenge. Texas A&M University, October 2011 2011.

[23] Göktuğ Karacalıoğlu. Malfunction of Kurs-NA resulted in a manual dock with ISS. Web: Space Safety Magazine, December 2013.

[24] Thomas Karlsson, Robin Larsson, Björn Jakobsson, and Per Bodin. The PRISMA story: Achievements and final escapades. In 5th International Conference on Spacecraft Formation Flying Missions and Technologies, 2013.

[25] Rob Knies. Collaboration, expertise produce enhanced sensing in xbox one. Web, October 2013.

[26] Rolf Lakaemper, Andreas Nüchter, Nagesh Adluru, and Longin Jan Latecki. Performance of 6d lum and ffs slam: An example for comparison using grid and pose based evaluation methods. In Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems, PerMIS '07, pages 250–256, New York, NY, USA, 2007. ACM.

[27] LASR Lab, Texas A&M University. LASR GNC System for Active Debris Removal. Web (video), March 2013.

[28] C.C. Liebe, C. Padgett, J. Chapsky, D. Wilson, K. Brown, S. Jerebets, H. Goldberg, and J. Schroeder. Spacecraft hazard avoidance utilizing structured light. In Aerospace Conference, 2006 IEEE, pages 10 pp.–, 2006.

[29] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. Autonomous Robots, 4(4):333–349, 1997.

[30] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In Proceedings of the 16th International Conference on World Wide Web, WWW '07, pages 141–150, New York, NY, USA, 2007. ACM.

[31] Aongus McCarthy, Nils J. Krichel, Nathan R. Gemmell, Ximing Ren, Michael G. Tanner, Sander N. Dorenbos, Val Zwiller, Robert H. Hadfield, and Gerald S. Buller. Kilometer-range, high resolution depth imaging via 1560 nm wavelength single-photon detection. Opt. Express, 21(7):8904–8915, Apr 2013.

[32] Shozo Mori, Kuo-Chu Chang, and Chee-Yee Chong. Multitarget-Multisensor Tracking: Applications and Advances, volume II, chapter 7: Performance Analysis of Optimal Data Association with Applications to Multiple Target Tracking, pages 183–235. Artech House, 685 Canton Street Norwood, MA 02062 USA, 1992.

[33] NASA. Iss on-orbit status 07/04/10. Web, July 2010.

[34] NASA. International Space Station Imagery: ISS026-E-037172. Web, February 2011.

[35] NASA. National space science data center: Cosmos 186. Web, August 2013.

[36] NATO. Sensor systems for generating 3d point clouds. Technical Report RTO-TR-SET-118, North Atlantic Treaty Organization (NATO) Research & Technology Organization, September 2011.

[37] Andreas Nüchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. Heuristic-based laser scan matching for outdoor 6d slam. In Ulrich Furbach, editor, KI 2005: Advances in Artificial Intelligence, volume 3698 of Lecture Notes in Computer Science, pages 304–319. Springer Berlin Heidelberg, 2005.

[38] Inc. Open Perception. vfh.hpp source code (point cloud library software api documentation). Web, December 2013.

[39] Didier Pinard, Stéphane Reynaud, Patrick Delpy, and Stein E. Strandmoe. Accurate and autonomous navigation for the ATV. Aerospace Science and Technology, 11(6):490 – 498, 2007.

[40] Daniel Rey. CSA activiites in on-orbit robotic servicing (ORS). In NASA GSFC International Workshop on On-Orbit Satellite Servicing. Canadian Space Agency, March 2010.

[41] Marcos Ricardo. Système de rendez-vous spatial automatique soviétique igla : les différents types d'antenne. Web, December 2012.

[42] Stephane Ruel. Tridar model based tracking vision system for on-orbit servicing. In NASA GSFC International Workshop on On-Orbit Satellite Servicing, March 2010.

[43] Radu Bogdan Rusu. Fast point feature histograms (fpfh) descriptors. Web, March 2014.

[44] Radu Bogdan Rusu. Point feature histograms (pfh) descriptors. Web, March 2014.

[45] Radu Bogdan Rusu. Estimating vfh signatures for a set of points. Web, 2013 December.

[46] Radu Bogdan Rusu. pcl::IterativeClosestPointWithNormals< PointSource, PointTarget, Scalar > Class Template Reference. Web (software API documentation), 2013 December.

[47] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, pages 3212–3217. IEEE, 2009.

[48] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, pages 2155–2162. IEEE, 2010.

[49] Radu Bogdan Rusu and Michael Dixon. pcl::iterativeclosestpoint< pointsource, pointtarget, scalar > class template reference. Web (software API documentation), 2013 December.

[50] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent point feature histograms for 3d point clouds. Intelligent Autonomous Systems 10: Ias-10, page 119, 2008.

[51] Hanspeter Schaub and John L. Junkins. Analytical Mechanics of Space Systems. AIAA Education Series, Reston, VA, 2nd edition, October 2009.

[52] Claude E. Shannon. A mathematical theory of communication. Bell System Technical Journal, 27(3):379–423, July 1948.

[53] R. Siegwart, I.R. Nourbakhsh, and D. Scaramuzza. Introduction to Autonomous Mobile Robots. Intelligent robotics and autonomous agents. MIT Press, 2011.

[54] Surrey Satellite Technology Limited. Surrey engineers use games console technology to make "space building blocks". Web (press release), May 2012.

[55] B.D. Tapley, B.E. Schutz, and G.H. Born. Statistical Orbit Determination. Elsevier Academic Press, 2004.

[56] D.A. Vallado and W.D. McClain. Fundamentals of Astrodynamics and Applications. Space Technology Library. Springer, 2007.

[57] T Weismuller and M Leinz. GN&C technology demonstrated by the orbital express autonomous rendezvous and capture sensor system. In 29th Annual AAS Guidance and Control Conference, pages 4–8, 2006.

[58] D.C. Woffinden and Utah State University. Engineering. Angles-only Navigation for Autonomous Orbital Rendezvous. Utah State University, 2008.

[59] R. Wolfson. Essential University Physics, Volume 1 [With Access Code]. Number v. 1 in Mastering-Physics Series. Pearson Education, Limited, 2011.

Chapter 7

Appendix: CSA copyright/re-use supplement

In compliance with the Canadian Space Agency's copyright and reuse policies, the image in figure 1.5 was extracted from a publicly released Canadian Space Agency presentation and cited with the original author's name as [40] in the bibliography of this document.

The image was modified to enlarge a portion of the screenshot that showed the algorithms used by TriDAR for tracking and point cloud alignment. The full, unmodified original presentation can be downloaded from the URL

http://ssco.gsfc.nasa.gov/workshop_2010/day2/Daniel_Rey/GSFC_RFI_CSA_Presentation_RevA_May2010.pdf