1-1-2010

Principal Component Analysis For ICP Pose Estimation Of Space Structures

Lun H. Mark
Ryerson University

PRINCIPAL COMPONENT ANALYSIS FOR ICP POSE ESTIMATION OF SPACE STRUCTURES

by

Lun Howe Mark

B.A.Sc. Mechanical Engineering, University of Ottawa, 2008
B.Sc. Computing Technology, University of Ottawa, 2008

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the Program of

Aerospace Engineering

Toronto, Ontario, Canada, 2010

Abstract

PRINCIPAL COMPONENT ANALYSIS FOR ICP POSE ESTIMATION
OF SPACE STRUCTURES

Masters of Applied Science, 2008, Lun Howe Mark
Aerospace Engineering, Ryerson University

This thesis investigates how geometry of complex objects is related to LIDAR scanning with the Iterative Closest Point (ICP) pose estimation and provides statistical means to assess the pose accuracy. LIDAR scanners have become essential parts of space vision systems for autonomous docking and rendezvous. Principal Component Analysis based geometric constraint indices have been found to be strongly related to the pose error norm and to the error of each individual degree of freedom. This leads to the development of several strategies for identifying the best views of an object and the optimal combination of localized scanned areas of the object's surface to achieve accurate pose estimation. Also investigated is the possible relation between the ICP pose estimation accuracy and the distribution or allocation of the point cloud. The simulation results were validated using point clouds generated by scanning models of Quicksat and a cuboctahedron using Neptec's TriDAR scanner.

Acknowledgments

I would like to thank my thesis supervisors, Dr. Galina Okouneva and Dr. Guangjun Liu for their help and support over the course of this endeavour. Their knowledge and guidance have proven to be invaluable.

Furthermore, I would also like to thank Dmitri Ignakov for his invaluable assistance.

Finally, I would like to thank Stephanie, for without her love, support and inspiration this would not have been possible.

Dedicated to my mother and the memory of my father

Table of Contents

List of Figures

List of Tables

Nomenclature

ANN	Approximate Nearest Neighbor
CAD	Computer Assisted Design
CSCA	Continuum-Shape Constraint Analysis
DOF	Degree of freedom
EI	Expectivity
ICP	Iterative Closest Point
InvCond	Inverse Condition Number
ISS	International Space Station
LIDAR	Light Detection and Ranging
ME	Minimum Eigenvalue
MSE	Mean Square Error
NAI	Noise Amplification Index
PC	Principal Component
PCA	Principal Component Analysis

λ	Eigenvalue
v	Eigenvector
q	Quaternion
R_{rpy}	Rotation Matrix from Euler Angles
ω	Rotation
t	Translation vector
p_i	Scanner point cloud
m_i	Model point cloud
p_ε	Pose error

1 Introduction

Within the area of computer and robot vision, there is a need for target objects to be processed and registered to find their relative pose. The pose is a 6-element vector comprised of 3 rotational and 3 translational components. The pose defines the transformation between the reference frames associated with the target object and the rangefinder scanner or CCD camera. A commonly used pose estimation tool is the Iterative Closest Point (ICP) algorithm which iteratively processes data from the scanner and corresponding data from an available CAD model of the object. Computer vision algorithms have become integral parts of space vision systems and currently assist in space operations such as autonomous assembly of the International Space Station, space rendezvous, satellite servicing, and the surface inspection of the Shuttle Orbiter (Ruel, English, Anctil, & Church, 2005).

There are many sizeable objects in low-earth orbit (LEO) either man-made, debris from man-made objects or other natural matter. Notable man-made objects in space include the ISS, Hubble space telescope and the Genesis I & II inflatable modules. In addition, there are numerous LEO satellites for the purposes of communications, weather, and observation. There are also satellites in other orbits; polar, geostationary, etc. One problem with orbiting objects is the proliferation of an estimated 200,000 pieces of debris (United Nations, 1999). As a result, collisions between functional equipment and random debris can and does occur. This has the potential of causing damage that must be repaired if full functionality of each satellite is to be maintained. The normal wear and tear of components, poor design, replacing of batteries or refueling represent other mission elements for a satellite that will necessitate some form of service or repair.

Manned flights into space are expensive and are an inherently dangerous undertaking. Not only do they require that astronauts undergo training and conditioning, but astronauts require supplies and specialized habitation and/or crew compartments to remain in orbit. The obvious alternative is to send unmanned flights into space; this has the potential advantages of being cheaper and safer. Building and launching unmanned repair and/or servicing units into space could lead to more robust satellite networks due to more active satellites, less downtime and newer technological components. However, unmanned missions are not without their own problems; communication lag makes remotely controlling anything at that distance more difficult. Therefore, service and repair units should be able to navigate and coordinate their own motion to reach the target space structure.

Ideally, service and repair units should be capable of autonomous movement and rendezvous. Actual repair of a satellite may require human intervention or teleoperation as this task can be much more complex, and may require an operator's discretion as the extent of damage may be unknown until a visual inspection. Thus the area of computer vision is especially useful for docking and rendezvous between spacecraft, space structures and satellites. In 2007, the U.S. Defense Advanced Research Projects Agency (DARPA) created the *Orbital Express,* a feasibility study into the autonomous refueling and reconfiguration of satellites (Friend, 2008). Two prototype satellites were built and then flown into LEO, one acted as a servicing unit and while the other acted as a serviceable unit. The conclusion of the study was that autonomous rendezvous and refueling of a target satellite is possible. The Orbital Express demonstration also showed that it is possible to upgrade and reconfigure a modular satellite on station.

To make the capturing operation simpler, traditional camera devices are eschewed for laser-based devices such as LIDARs (Light Detection and Ranging). This is because LIDAR operations operate on different wavelengths, and by using their own laser beams as an illumination source, they are independent of albedo, the effect of light reflection on earth and other surfaces. Albedo can cause large variations in the surrounding light levels which visual algorithms must take into account, thus LIDARs are preferred for their independence of ambient lightning scenarios. A LIDAR provides a discretized point sampling of the target object's surface, called a point cloud. This point cloud can be registered against a model point cloud to estimate the pose via the Iterative Closest Point algorithm which is accredited to Besl & McKay (1992).

Planning a space mission is a very complicated procedure as the margins of error are very slim. To plan a mission, extensive ground testing and data analysis are required. In an ideal case, full-scale and accurate replications of the spacecraft modules should be manufactured for testing in a space-like environment using certified equipment. All of this can be very expensive and are highly inflexible, mission parameters can change and modules may need alterations based on the tests run. A computer simulation can be a powerful tool and is much cheaper than a full fledged test rig, which can include: full-scale spacecraft modules and flight capable LIDARs or cameras.

The first item required to simulate the LIDAR and ICP operation is a virtual scanner. This scanner requires the input of a CAD model; in this work, triangular meshes are used to represent the CAD model surfaces so that ray tracing may be performed. The ray tracing and virtual scanner must replicate the manner in which a real LIDAR operates. This includes the manner in which the laser beams sweep across the surfaces. In general, LIDARs are accurate tools, however, there is uncertainty stemming from the measurement noise and the accuracy of the LIDAR's internal mechanisms.

The second item required to simulate the LIDAR and ICP operation is a voxelizer which is used to increase the speed of the virtual scanning and of the ICP algorithm. From the original CAD model, a voxelized model is created. The voxelized model is an associated data structure that works alongside the original CAD model. Visually, the voxelized model is a very low resolution representation of the original model using large volume elements. While the speed of the virtual scanner has no bearing in a real application of the ICP algorithm, the speed of the ICP algorithm is very important. If an ICP algorithm can return results at a quicker pace, then it becomes more suitable for real-time tracking.

The third item required to simulate the LIDAR and ICP operation is the ICP algorithm itself. The ICP algorithm requires a scanner point cloud and a model point cloud to align. In addition, an initial guess of the pose is required for iterations to begin. With a good initial guess, the ICP algorithm is able to arrive at very close approximations of the true pose. With an initial guess far from the true pose, the ICP algorithm performance can suffer and a local minimum, as opposed to the global minimum, is found.

The fourth item required to simulate the LIDAR and ICP operation is the ability to evaluate the results. If the results of the ICP algorithm are accurate, it is expected that the error will be small. Conversely, if the ICP algorithm falls into a local minimum, then it is expected that the error will be large. It would be extremely advantageous if the overall pose error and the error of each pose component could be accurately predicted. However, as the ICP algorithm is complex, the next best thing would be to know the constraint or limit on the error levels. This can be done by measuring the geometric constraint in the scanner point cloud using a constraint matrix. Geometric constraint is an interpretation of how the visible surface of a target can reduce ICP pose error. A rich surface geometry has high geometric constraint. Several geometric constraint measures derived from the constraint matrix are evaluated: the Noise Amplification Index (NAI), the Inverse Condition Number (InvCond) the Minimum Eigenvalue (ME) and the Expectivity Index (EI). These measures define upper bounds on the pose error norm, relative pose error norm and the standard deviation of the expected pose error.

There are several factors which can influence the overall registration error. One factor is the manner in which points lie along the edges of a model or along the outline of the model. These points contribute to not only the geometric constraint, but also to the edge constraint. The effect of edge constraint is that the point cloud has less room to manoeuvre, wiggle or oscillate; all of which helps to properly align the point clouds together. The effects of edge constraint are especially evident when there is lower geometric constraint.

To make the ICP algorithm more suitable for real-time tracking, the amount of data can be cut down by selecting small portions or areas of the scanner point cloud. There are several options available for this; however, many of these options are somewhat computationally expensive, requiring the constant recalculation of the constraint matrix. A very simple method to reduce the number of points is to subdivide the point cloud into a grid based on the projected X- and Y- axis coordinates. Each subdivision is called a window, and each window peeks into a different section of the point cloud. While there is occasionally a lone window that provides a good registration result, the use of two is typically much better. This is because there are more types of varying constraint provided by two windows than just one window; consequently, this leads to a much better pose estimation.

While two windows are shown to be better than one, the question remains of how to select the combination of the two windows that will provide the best registration. There are two separate options to consider when designing strategies for window selection. The windows can be combined based upon the geometric constraint found in the individual windows or in the combined window. Ideally, a selection strategy should consistently select good pairs of windows over a range of views.

The organization of this thesis is structured using the requirements listed above. Chapter 2 presents a review of PCA and its application to computer vision. Chapter 3 describes the mathematical formulation of the ICP algorithm as well as all the components needed to simulate an ICP situation. Chapter 4 illustrates the investigation into ICP performance and the application of point cloud windows. Chapter 5 summarizes and concludes the thesis.

2 Principal Component Analysis and Computer Vision

Principal Component Analysis (PCA) is a common technique used to simplify data. PCA is used to discover relations within the data with the goal of identifying the principal, or most important, components in the data. Using the principal components (PCs), a simplification of the data set can be achieved. This means that a large data set can be reduced to a smaller data set by eliminating or ignoring certain portions of the data which are found to be superfluous. In computer vision, PCA was initially used as a tool used for facial recognition. Later, PCA principals were applied to the pose estimation of computer models for applications in space.

2.1 Review of the Foundations of Principal Component Analysis

PCA is a common statistical technique used for reducing the complexity of high volume data sets. By reducing the degrees of freedom (DOFs) or the dimensionality of the data, the dominant variables can be found. PCA can frequently be found in applications calling for statistical analysis; this includes meteorology, computer vision, medicine and biosciences, among others (Jolliffe, 2002).

PCA was initially developed by Karl Pearson in 1901. Pearson outlined the use of centroids and ellipsoidal measures to determine the lines of best and worst fit (Pearson, 1901). At higher dimensions, Pearson used a best-fit plane instead. Later in 1933, Harold Hotelling created his own PCA techniques and is also credited as a founder of PCA.

In practice, the use of PCA is to reduce the complexity of the data to make it easier to interpret. Figure 1 shows a simple data set of 300 X-Y-Z points to which PCA is applied. Figure 1 also shows the three PCs of the data set as vectors which are sized to reflect the value of the PC. While an easy way of reducing complex data would be to ignore certain variables or components, such as the Z variable, PCA will reduce the DOFs without losing vital information. By applying PCA, the PCs are defined by the amount of variation in the original data; the first PC represents the most variation while the last PC represents the least. Depending on the application, only the first few PCs, which represent the most variation, are important. In other applications, such as this work, all PCs are equally important.

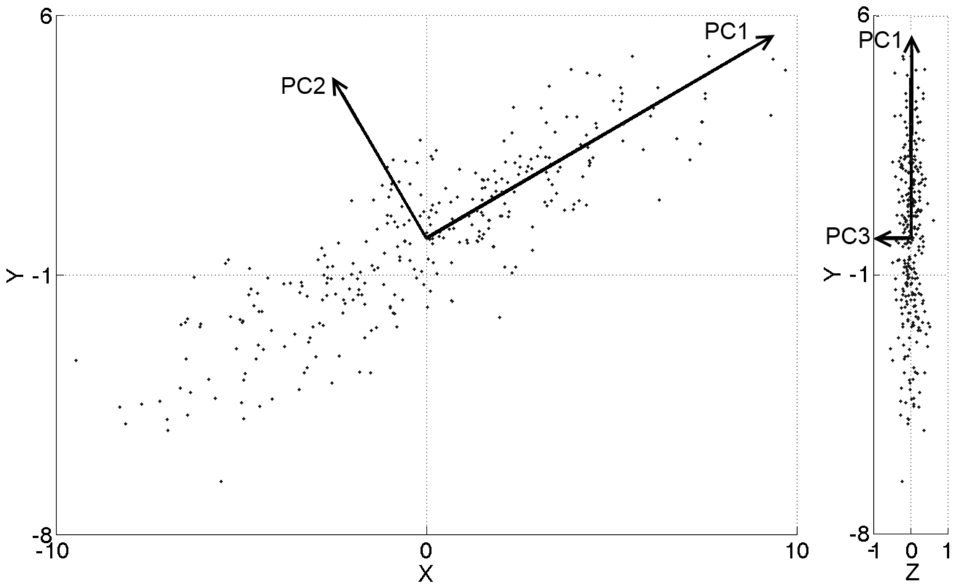


Figure 1 - Sample data set XY and YZ views

A covariance matrix is built by assembling the variances and covariances of a data set with N points. PCA calculates the PCs by decomposing the covariance matrix. The covariances in a data set identify the way that two variables relate to each other. If the covariance between two variables is positive, then it is likely that as one variable increases so does the other. If the sign of the covariance is negative, then as one variable increases, the other should decrease. If two variables are completely independent of each other, then the covariance will be zero. Suppose there are p number of variables, $A, B, \dots, Z$, each with a respective mean $\bar{A}, \bar{B}, \dots, \bar{Z}$, then the covariance between any two variables is defined by

$$\begin{aligned} cov(A,B) &= \frac{1}{N-1}\sum_{\mathrm{i}=1}^{\mathrm{i}=\mathrm{N}}(\mathrm{A_i} - \overline{\mathrm{A}})(\mathrm{B_i} - \overline{\mathrm{B}}) \\ &= \mathrm{E}([\mathrm{A} - \mathrm{E(A)}][\mathrm{B} - \mathrm{E(B)}]) \end{aligned} \quad \mathbf{2.1}$$

where $\mathrm{E}(A)$ represents the expected value of A. The variance of a variable is defined by

$$\begin{aligned} var(A) = cov(A,A) &= \frac{1}{N-1}\sum_{\mathrm{i}=1}^{\mathrm{i}=\mathrm{N}}(\mathrm{A_i} - \overline{\mathrm{A}})(\mathrm{A_i} - \overline{\mathrm{A}}) \\ &= E(A^2) - E(A)^2 \end{aligned} \quad \mathbf{2.2}$$

Together, the variances and covariances can be combined together to form the data set's $p \times p$ covariance matrix, S, is given by

$$S = \begin{pmatrix} var(A,A) & cov(A,B) & \dots & cov(A,Z) \\ cov(B,A) & var(B,B) & \cdots & cov(B,Z) \\ \vdots & \vdots & \ddots & \vdots \\ cov(Z,A) & cov(Z,B) & \dots & var(Z,Z) \end{pmatrix} \qquad \textbf{2.3}$$

A much simpler way of creating and representing the covariance matrix S is used in this work. By subtracting the $N \times p$ data set, $X_{set} = [A, B, \dots, Z]$, by the mean, $\bar{X}_{set} = [\bar{A}, \bar{B}, \dots, \bar{Z}]$, the data becomes centred on the $[0,0,\dots,0]$ position. The covariance matrix calculation then becomes a much simpler matrix math operation

$$S = \frac{1}{N-1}\hat{X}_{set}^{T}\hat{X}_{set} \qquad \textbf{2.4}$$

where $\hat{X}_{set} = (\mathrm{X_{set}} - \mathrm{\bar{X}_{set}}) = [A - \bar{A}, B - \bar{B}, \dots, Z - \bar{Z}], n \times p$ matrix

The covariance matrix S is symmetric as the formula for $cov(A,B)$ is equivalent to $cov(B,A)$. Similarly, the result of $var(A)$ is equivalent to $cov(A,A)$.

After finding the PCs, the next step in PCA is to transform the data into a new coordinate system; the major axes of the new coordinate system will be defined by the direction of the PCs. By definition, PCs are uncorrelated and independent of each other and are orthogonal to each other. A PC, α_i, can be represented as a linear combination of the original variables in the form of $PC_1 = \alpha_1^T X_{set} = \alpha_{11}A + \alpha_{12}B + \dots + \alpha_{1p}Z$. The order the PCs is important; the first PC will correspond to the variable with the greatest variance while the second PC will contain the second greatest variance, and so on. The result of PCA is that the first few PCs contain the majority of the variation and can represent nearly all of the initial data set. In a data set with a large number of variables, it is likely that the number of original variables, p, will be much greater than the final number of principal components, m, such that $p \gg m$. This is because some variables will not contribute to the overall solution of the data set, and are otherwise independent and/or negligible.

To find the direction and magnitude of the PCs, either Eigenvalue Decomposition (ED) or Singular Value Decomposition (SVD) may be performed on the covariance matrix, S. Eigenvalue Decomposition is a technique used to find the eigenvalues and eigenvectors of S (Moler, 2004). The scalar eigenvalues, λ, and vector eigenvectors, v, are defined such that

$$Sv = \lambda v \qquad \textbf{2.5}$$

The eigenvectors give the direction of the PCs in the original data set, therefore $v_i = \alpha_i$. Meanwhile, the eigenvalues give the standard deviations of the original data along the PCs. The PCs are ordered based on the eigenvalues. While Equation 2.5 is a more prevalent form, it does not directly give the eigenvectors. To find the eigenvectors, matrix S is factorized as

$$SX = X\Lambda \tag{2.6}$$

where $\begin{cases} X \text{ is a matrix of eigenvectors; the j}^{\text{th}} \text{ column corresponds to eigenvalue } x_i \\ \Lambda = diag(\lambda_1, \lambda_2, \dots, \lambda_n), \lambda_i \geq 0, \text{ alternatively, } \lambda \cdot I_{n\times n} \end{cases}$

If X is if of full rank, then it should be possible to find the inverse of X and reshape Equation 2.6 as

$$S = X\Lambda X^{-1} \tag{2.7}$$

Singular Value Decomposition (SVD) is another decomposition technique with similar results to that of ED. The SVD technique can also be used to find the PCs of the data set.

$$S = U\Sigma V^T \tag{2.8}$$

where $\begin{cases} U \text{ and } V \text{ are orthogonal matrices such that } U^T U = UU^T = V^T V = V^T V = 1 \\ \Sigma = diag(\lambda_1, \lambda_2, \dots, \lambda_n), \lambda_i \geq 0 \end{cases}$

Matrix U and $V$are the eigenvectors for SS^T and $S^T S$, respectively. When the covariance matrix S is square and symmetric, such as in this work, both ED and SVD will result in the same answer. Therefore, the matrix X from ED is the equivalent to matrices U and V from SVD. While the magnitudes of the individual eigenvector components created by ED and SVD will be similar, the signs of the entire eigenvectors may change as it is possible for them to point in opposite directions. In addition, ED and SVD may be output the eigenvectors and eigenvalues in different orders.

When PCA is applied, the original data set is translated and rotated so that the largest variance in the data will lie along the primary axis dictated by the eigenvector, v_1 , associated with the largest eigenvalue, λ_1. As an eigenvector, v_i, represents the direction of the ith PC, α_i, it is a linear combination of the original variables. The results of ED and SVD should be that $\lambda_1 > \lambda_2 > \dots > \lambda_p$. To transform the data into the new PC coordinate system, the eigenvector matrix is used

$$X_{set}' = X_{set} v \tag{2.9}$$

where $\begin{cases} X_{set}' \text{ represents the data transformed into the frame of the PCs} \\ X_{set} \text{ is the original data set} \end{cases}$

Referring back to Figure 1, by assembling a covariance matrix based on the data shown and performing ED, the eigenvectors and eigenvalues are found. Each column of v represents an eigenvector

$$v = [v_1 \quad v_2 \quad v_3] = \left[\begin{array}{c|c|c} -0.8635 & -0.5044 & -0.0032 \\ -0.5044 & 0.8635 & -0.0020 \\ -0.0018 & 0.0034 & 1.0000 \end{array}\right] \quad \lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} 16.8445 \\ 1.0081 \\ 0.0404 \end{bmatrix}$$

When Equation 2.9 is applied, the rotated data set, X_{set}', becomes plotted as:

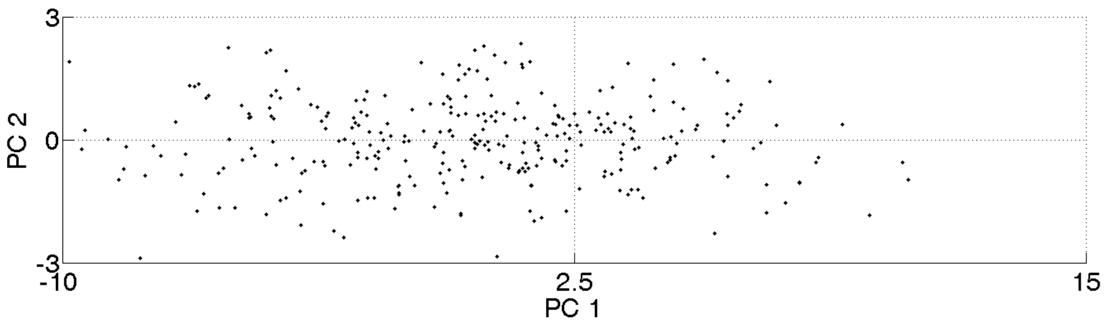


Figure 2 - Principal components

To reduce the complexity of the data set, certain variables may be discarded if their contribution is negligible. This is done by comparing the relative values of the eigenvalues, if an eigenvalue is small compared to the sum of the eigenvalues then it can potentially be ignored. Equivalently, this means that if a variable's variance is much smaller than the total variance, then its contribution will also be negligible. By definition, the first PC will contain the most variation; however, it may require more PCs to account for an accurate representation of the full variation in the data set. Each PC's contribution to the solution is calculated by the cumulative sum of the variances. For example:

$$\frac{1}{17.893}\begin{bmatrix} 16.8445 \\ 1.0081 \\ 0.0404 \end{bmatrix} \times 100\% = \begin{bmatrix} 94.14 \\ 5.63 \\ 0.22 \end{bmatrix}\% \rightarrow \begin{bmatrix} 94.14 \\ 99.77 \\ 100.00 \end{bmatrix}\%$$

As Figure 2 reflects, the first PC contains the most variation and amounts to 94.14% of the data set's variance. If the second PC is included as well, then the variation increases to 99.77%. The variables represented in v_1 and v_2 are mostly based upon a mix of X and Y. With the first two PC's representing most of the data set, the third PC can potentially be dropped. Looking at the eigenvector associated with the third PC, it is easy to see that it is heavily influenced by the third variable, Z. Therefore, it would not be unreasonable to say that Z can be disregarded, effectively dropping the complexity of the system and the degrees of freedom to two.

2.2 Applications of PCA in Computer Vision

The initial application of Principal Component Analysis for facial recognition and detection most likely belongs to Turk & Pentland (1991), for the development of Eigenfaces. The idea behind Eigenfaces is that a face can be described by certain two-dimensional characteristics; however these characteristics are not strictly defined as being ears, nose, mouth, etc. The idea is to begin with an average Eigenface that all other are compared against. By treating a picture, which is an $N \times N$ matrix of pixels, as an $N^2 \times 1$ vector, a covariance matrix can be built using Equation 2.4. This covariance matrix is defined as an Eigenface. Several training faces are used to establish an average Eigenface, from this most other faces can be reconstructed using the PCs.

The work done by Turk & Pentland (1991) has been extended to several other areas. Becker & Ortiz (2009) proposed the use of PCA, as well as other facial recognition methods, in the popular social networking website Facebook where a wide variety of faces are available for processing. One problem with PCA facial recognition is that not all images are taken under the same conditions. There is a wide variety of photograph composition variables such as lighting, framing, orientation and facial expression. There are also image variables including the compression, resolution and clarity of the camera. To this end, Moon and Phillips created the FERET evaluation protocol which has been successfully and extensively tested (Moon & Phillips, 2001). Other facial recognition methods include Linear Discriminant Analysis (LDA) also known as Fisherfaces; Independent Component Analysis (ICA); Gabor jets; and Support Vector Machines (SVM), for a comparison of these methods see Becker & Ortiz and Draper, Baek, Bartlett, & Beveridge (2003). Becker & Ortiz found that for a widespread range of conditions, no single or hybrid registration method performs optimally.

PCA is also used in gesture recognition, the means by which software can understand human motions to recreate them or to understand them. As Braido & Zhang (2004) suggest, there are over 30 degrees of freedom when trying to model the human hand due to the numerous finger segments and joints. Braido & Zhang focused on the grasping gesture and found that 98 percent of the variation was contained within the first two PCs. This led to a much simpler model of the grasping gesture.

Human input devices (HID), such as a keyboard, mouse or pen, are used by humans to interact with a computer. However, these are limited, as they not as intuitive or simple as they could potentially be. To extend human and computer interaction, PCA can be applied for gestures on a touch screen or captured by a camera system. A touch screen for a collaborative information system was designed by Jeong, Ribarsky, & Chang (2009) using a specialized PCA algorithm which allows for many people to work at once on the same touch screen. This touch screen was designed to assist users to share and collaborate by employing an interactive PCA with collaborative elements (iPCA-CE). For capturing hand gestures using cameras, Birk, Moeslund, & Madsen (1997) recognized and translated the American Sign Language using PCA and Hidden Markov Models (HMM) with a high degree of success. Wu & Sutherland (2001) expanded upon this work by using Discrete Hidden Markov Models (DHMM) instead. The use of DHMM helps to speed up the recognition process by allowing fingertips to be recognized when the gesture motion has low variance. Billon, Nédélec, & Tisseau (2008) combined the use of Nintendo Wiimote controllers and full body suits to control a virtual actor mimicking a real world actor. In this case, PCA was not used to extract the gestures, but to act as filter, reducing the data dimensions. This allowed for a recognition success rate better than 80% in the worst case, or 100% in an optimal case.

Closer to the work presented below is the learning and recognition of 3D Objects using images by Murase & Nayar (1995). However, cameras and 2-D images are troublesome due to changing scale and brightness, inclusion of background objects, or changes in position of the camera or pose of the object. By stringing together several training images, the identity and pose of an imaged object can be found. The measure used in this application is a ratio between the summations of the eigenvalues; a value closer to unity indicates that the image is close to one of the training poses. This is later extended by Ohba and Ikeuchi (1997) when the theory is expanded to deal with the case where a portion of the object is hidden or occluded by using the "Eigen windows method". Good eigen windows were selected on the basis of detectability, uniqueness and reliability. With several good training eigen windows pre-stored in the eigenspace, known objects are detected and registered from image(s).

2.3 PCA Indices for Pose Estimation

Simon introduced a new application of PCA, called geometric constraint analysis, to the ICP process of pose estimation in computer vision Simon (1996). Geometric constraint analysis examines the sensitivity of shape registration error to variations in the model's pose. This provides a powerful way of assessing the expected accuracy of iterative registration algorithms. In Simon's work, constraint analysis was used to optimize the selection of target points on human bones for scanning during radiation therapy. The optimization was based on the Noise Amplification Index (NAI) Nahvi & Hollerbach (1996) calculated from the constraint matrix built using the point cloud. Larger NAI values were shown to correspond to smaller values of the norm of the pose error. Originally, NAI was constructed to analyze robot mechanics and the position of an end effector, however, Simon saw the similarities between pose estimation and end effector estimation.

While Simon used a sparse set of key points for bone alignment, Shahid & Okouneva (2007) used the same form of discrete-point constraint analysis and applied it to point clouds generated by windowed areas of spacecraft objects. By localizing geometric features, the optimal local scan areas for pose estimation can be identified. A more recent paper by McTavish, Okouneva, & Choudhuri (2009) generalizes the concept of discrete-point self-registration to a surface integral-based self-registration called Continuum-Shape Constraint Analysis (CSCA). Like discrete-point geometric constraint analysis, CSCA is also used for pose estimation assessment and view selection. To account for the directional nature of a scan, a view factor is incorporated into the CSCA cost matrix calculation. In this case, constraint analysis was used to establish whole-object views for accurate pose estimation. In McTavish, Okouneva, & Okounev (2009), a new constraint analysis index, the Expectivity Index (EI), was introduced. More information on EI can be found in McTavish, Okouneva, & English (2010).

The eigenvectors of the constraint matrix help to identify the direction in which the point clouds are most constrained. A well constrained point cloud, and therefore well constrained view, will be constrained in several degrees of freedom (DOF). Conversely, the eigenvectors also identify the manner in which the point clouds are poorly or weakly constrained.

3 ICP Algorithm and Program

The Iterative Closest Point (ICP) algorithm is a relatively simple method used to align two sets of data. The alignment of the data is computed by finding a six DOF pose vector consisting of three translation and three rotation components. One significant advantage of the ICP algorithm is that it does not need to extract geometrical features or descriptors, such as Gaussian curvature, from a model or data set. The ICP algorithm is mainly accredited to Besl & McKay (1992), however it was also developed independently around the same time by several others (Chen & Medioni, 1992; Menq, Yau, & Lai, 1992; Zhang, 1994). To begin, the ICP algorithm requires the input of an initial guess. Using the initial guess as a starting point, the ICP algorithm then repetitively refines a pose estimate until one of the convergence criteria is met.

The ICP algorithm is a commonly used to align between two sets of a) point sets; b) line segments; c) implicit curves; d) parametric curves; e) triangular sets; f) implicit surfaces; g) parametric surfaces. Although the ICP algorithm is capable of iteratively aligning both 2-D and 3-D data sets based on geometry alone, colour and light intensity can also be included in pose estimation. For this work, 3-D triangular sets, called meshes, and point sets, called point clouds, are used. A mesh consists of two components, a list of vertices in three-space, and a list of faces, see Section 3.4.1. A point cloud is a set of 3-D points generated by the scanner. For each iteration step, the ICP algorithm works to minimize a cost function calculated by pairing corresponding points together. The ICP algorithm in this work is based on quaternions which will be reviewed in the following section.

3.1 Definition of Quaternions

Quaternions are a numbering system that, through the extension of complex numbers, has four dimensions to solve three dimensional problems, and it developed by Sir William Hamilton in 1843. Since then, quaternions have been used as a means of representing rotations (Shoemake, 1985). Using $1, i, j, k$ as system bases, a quaternion, q, is defined as a linear combination of 4 coefficients, η, e_x, e_y and e_z such that

$$q = \begin{bmatrix} \eta \\ e \end{bmatrix} = [\eta \quad e_x \quad e_z \quad e_z]^T$$
$$q = \eta \cdot 1 + e_x \cdot i + e_y \cdot j + e_z \cdot k \qquad \textbf{3.1}$$

Of the coefficients, only η is a real number; e_x, e_y and e_z are purely imaginary. The quaternion coefficients are constrained by

$$i^2 = j^2 = k^2 = ijk = -1$$

$$\|q\| = \sqrt{\eta^2 + {e_x}^2 + {e_y}^2 + {e_z}^2} = 1 \qquad \textbf{3.2}$$

$$\eta \geq 0$$

Other mathematical concepts concerning quaternions include (Shoemake, 1985):

Conjugate	$q^* = [\eta, -e]$
Norm	$\lvert q \rvert = \eta^2 + {e_x}^2 + {e_y}^2 + {e_z}^2$
Addition	$q_1 + q_2 = [\eta_1 + \eta_2, e_1 + e_2]$
Multiplication	$q_1 q_2 = [\,\eta^1\eta^2 - e^1 \cdot e^2, e^1 \times e^2 + \eta^1 e^2 + \eta^2 e^1]$

3.2 ICP Algorithm Mathematical Formulation

The ICP algorithm implemented in this work is the method outlined by Besl & McKay (1992) and uses a quaternion matrix representation given from Horn (1987). The only modification to the basic method is in the cost metric used. Instead of the typical mean square error (MSE), the total Euclidean distance given by the Approximate Nearest Neighbor (ANN) method is used as the minimization metric, and will be further discussed in Section 3.4.4.3.

A pose can be calculated using a 3×1 translation vector, $\vec{t}$, and a 3×3 rotation matrix, R_{rpy}, constructed using Euler angle values of roll, pitch and yaw values as follows:

$$R_3 = \begin{bmatrix} C_3 & -S_3 & 0 \\ S_3 & C_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \; R_2 = \begin{bmatrix} C_2 & 0 & S_2 \\ 0 & 1 & 0 \\ -S_2 & 0 & C_2 \end{bmatrix} \; R_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C_1 & -S_1 \\ 0 & S_1 & C_1 \end{bmatrix} \qquad \textbf{3.3}$$

$$R_{rpy} = R_3(y)R_2(p)R_1(r) = \begin{bmatrix} C_3C_2 & -S_3C_1 + C_3S_2S_1 & S_3S_1 + C_3S_2C_1 \\ S_3C_2 & C_3C_1 + S_3S_2S_1 & -C_3S_1 + S_3S_2C_1 \\ -S_2 & C_2S_1 & C_1C_2 \end{bmatrix} \qquad \textbf{3.4}$$

$$\text{where} \begin{cases} C_3 = \cos(y) \text{ and } S_3 = \sin(y) \\ C_2 = \cos(p) \text{ and } S_2 = \sin(p) \\ C_1 = \cos(r) \text{ and } S_1 = \sin(r) \\ r, p, y = roll, pitch, yaw \end{cases}$$

Combining Equation 3.4 with the translation vector, a 4×4 homogeneous transformation matrix, T, can be defined as

$$T([t_x \quad t_y \quad t_z \quad r \quad p \quad y]) = \begin{bmatrix} R_{rpy} & \vec{t}^T \\ 0 & 1 \end{bmatrix} \tag{3.5}$$

The ICP algorithm begins with two point clouds, the input point cloud $\{\vec{p}_i\}: p_i = (x_i, y_i, z_i), i = 1,2, \dots, N$ and the model point cloud $\{\vec{m}_i\}$: $m_i = (x_i, y_i, z_i), i = 1,2, \dots, N$. The input point cloud is produced from the scanner, while the model point cloud is obtained by transforming $\{\vec{p}_i\}$ using the pose initial guess.

During each iteration of the ICP algorithm, the input cloud $\{\vec{p}_i\}$ is approximated to the model cloud $\{\vec{m}_i\}$ through

$$\vec{m}_i \approx R_{rpy}\, \vec{p}_i + \vec{t} \tag{3.6}$$

Typically, the mean square error (MSE) is found by taking the difference between the model point cloud $\vec{m}_i$ and scanner point cloud $\vec{p}_i$ to calculate the residual errors.

$$Mean\ square\ error = \frac{1}{n}\sum_{i=1}^{n} \left\| \vec{m}_i - \left(R_{rpy}\, \vec{p}_i + \vec{t}\right)\right\|^2 \tag{3.7}$$

As mentioned earlier, this form of the MSE is not used as the ICP cost metric.

To achieve the refined translation and rotation values and therefore an updated $\{\vec{p}_{i+1}\}$, the cross-covariance matrix, $\{\Sigma_{\mathrm{pm}}\}$, between $\{\vec{p}_i\}$ and $\{\vec{m}_i\}$, is used. As with the regular covariance matrix, the centroids of both point clouds are required

$$\vec{\mu}_p = \frac{1}{n}\sum_{i=1}^{n} \vec{p}_i \qquad\qquad \vec{\mu}_m = \frac{1}{n}\sum_{i=1}^{n} \vec{m}_i \tag{3.8}$$

The cross-covariance matrix Σ_{pm} can then be alternatively represented as

$$\Sigma_{\mathrm{pm}} = \frac{1}{\mathrm{n}}\sum_{i=1}^{n} \left[\vec{p}_i \vec{m}_i{}^T\right] - \vec{\mu}_p \vec{\mu}_m{}^T \tag{3.9}$$

Equation 3.9 is an alternative representation of the covariance matrix defined by Equation 2.1 and is found by using the expected values. The cross-covariance matrix Σ_{pm} is then used as part of the 4×4 registration matrix, $\mathrm{Q}(\Sigma_{\mathrm{pm}})$, as given by:

$$\mathrm{Q}(\Sigma_{\mathrm{pm}}) = \begin{bmatrix} \mathrm{tr}(\Sigma_{\mathrm{pm}}) & \Delta^{\mathrm{T}} \\ \Delta & \Sigma_{\mathrm{pm}} + \Sigma_{\mathrm{pm}}^{\mathrm{T}} - \mathrm{tr}(\Sigma_{\mathrm{pm}})\mathrm{I}_3 \end{bmatrix} \qquad \mathbf{3.10}$$

where $\left\{ \begin{array}{l} \Delta \text{ is a vector defined by the cyclic elements of } A,\ \Delta = [A_{23}\ A_{31}\ A_{12}]^T \\ \text{A is the anti-symmetric matrix } A = \Sigma_{\mathrm{pm}} - \Sigma_{\mathrm{pm}}^{\mathrm{T}} \end{array} \right.$

The eigenvector corresponding to the largest eigenvalue of $\mathrm{Q}(\Sigma_{\mathrm{pm}})$ is the refined value of the rotation. This eigenvector is a unitary and is a rotation quaternion, $\vec{q}_R = [q_o\ q_1\ q_2\ q_3]^T$. To find the optimal translation quaternion, $\vec{q}_T = [q_4\ q_5\ q_6]^T$

$$\vec{q}_T = \vec{\mu}_m - R(\vec{q}_R)\vec{\mu}_p \qquad \mathbf{3.11}$$

where $R(\vec{q}_R)$ is the orthogonal rotation matrix as defined by Horn (1987)

$$R = \begin{bmatrix} q_o^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_o^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_o^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \qquad \mathbf{3.12}$$

From the optimal quaternion translation, $\vec{q}_T$, and rotation, $\vec{q}_R$, a quaternion pose vector can be established as $\vec{q} = [\vec{q}_R|\vec{q}_T]^t = [q_o\ q_1\ q_2\ q_3\ q_4\ q_5\ q_6]^t$. The next point cloud $\{\vec{p}_{i+1}\}$ is found by updating $\vec{p}_i$ with the optimal rotations and translations

$$\vec{p}_{i+1} = R(\vec{q}_R)\vec{p}_i + \vec{q}_T \qquad \mathbf{3.13}$$

An alternate representation of the rotation matrix R is given by Horn and allows for a one line matrix calculation

$$R = (n^2 - e^Te)I_3 + 2ee^T + 2ne^x \qquad \mathbf{3.14}$$

where $\left\{ \begin{array}{l} n = q_0 \\ e = [q_1\ q_2\ q_3]^T \\ e^x \text{ is the skew-symmetric matrix} \end{array} \right.$

The skew-symmetric matrix of e^x satisfies the condition that $e^x = -(e^x)^T$

$$e^x = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix}^x = \begin{bmatrix} 0 & -e_3 & e_2 \\ e_3 & 0 & -e_1 \\ -e_2 & e_1 & 0 \end{bmatrix} \qquad \mathbf{3.15}$$

Note that the skew-symmetric matrix can be used to represent the cross product as

$$cross(u, v) = u \times v = u^xv \qquad \mathbf{3\text{-}16}$$

Now that a new point cloud $\{\vec{p}_{i+1}\}$ has been found at each iteration step, the error cost function must be calculated to evaluate the registration. ICP algorithm iterations will continue until one of the cases described by Figure 3 holds true:

- The maximum number of iterations has been reached
- The cost function has dipped below a certain threshold, τ
- The cost function slope has flattened out such that it falls below a threshold, ϵ according to Equation 3.17:

$$\epsilon = \frac{abs(Error_i - Error_{i-1})}{Error_{i-1}} \qquad \textbf{3.17}$$

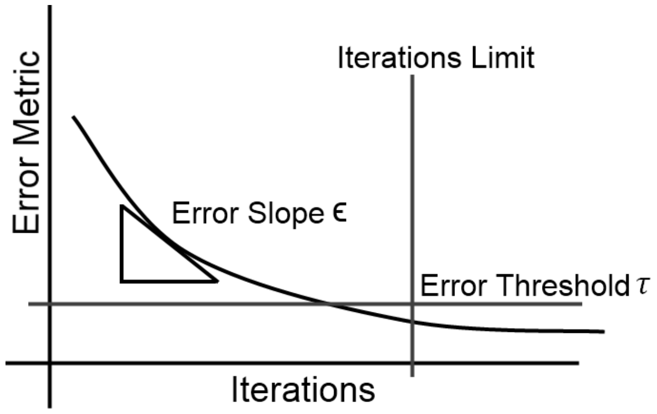


Figure 3 - Error vs. iterations

In summary, the simplified ICP algorithm is

Input	ICP Algorithm	Output
• Sample data • Reference data • Initial guess	1. Calculate the updated pose 2. Calculate error 3. Repeat steps 1 & 2 until convergence	• Pose estimate • Updated point cloud

3.3 Review of ICP Algorithm Stages

Section 3.2 introduces the mathematics for the ICP algorithm, however this is only one stage of an overall ICP program; Rusinkiewicz & Levoy (2001) suggest six other ICP program stages:

1. Point Selection

2. Matching
3. Weighting
4. Rejection
5. Cost Function Determination
6. Cost Minimization

There are multiple ways to speed up the ICP program. Each stage of the ICP program can have several different variations; a review of some implemented or suggested methods follows below.

1. Point Selection

From a larger superset point cloud, smaller point clouds may be generated by various point selection strategies. The simplest strategy at this stage would be to simply select all the points; this should give the ICP algorithm more data to work with and increase the registration accuracy. However, using all the points may increase the computer processing time and power. Turk & Pentland (1991) suggest to uniformly subsampling the available point cloud, whereas, Masuda (2002) suggests to randomly choose points. Others such as Weik (1997) advocate using colour or intensity gradients. Rusinkiewicz & Levoy (2001) suggest choosing points such that the normal-space will be as large as possible. Simon (1996) describes several hill climbing algorithms used to create the point set. Gelfand & Rusinkiewicz (2003) suggest selecting points for the PCA constraint matrix such that the weakest eigenvalues and eigenvectors are constrained.

Rusinkiewicz & Levoy (2001) show that the uniform, random and normal-space sampling result in similar performance and conclude that point selection is not a critical component. However, Gelfand & Rusinkiewicz (2003) show that selection of points to fill out the constraint matrix can lead to better results than the uniform and normal-space sampling. The downside of using such a method is the associated cost to the algorithm speed; this method requires the repetitive building of the constraint matrix.

Although random and uniform sampling may on average perform similarly, a uniform point selection strategy should intuitively be better as all features on an object will be covered and contribute to the constraint. A random selection strategy may leave gaps or holes in the surface coverage, thus missing some geometric features. Covariance and normal-space selection should also intuitively do better as more emphasis is placed on feature-rich areas as opposed to planar surfaces. Point cloud generation will be discussed in Section 3.4.3.

2. Point Matching

For this work, the inputs to the ICP algorithm are a point cloud from a scanner and a CAD model which are closely, but not perfectly, aligned. To relate the point cloud to the model, each element in the point cloud is examined and the closest point on the surface of the model is generated. The closet points on the surface form the model point cloud, $\vec{m}_i$. The ith element of $\vec{m}_i$ corresponds directly to the ith element in $\vec{p}_i$. To evaluate the registration, the scanner and model point clouds are compared and examined.

Generating the model point cloud is one of the most time intensive portions of the ICP program. Besl & McKay (1992) recognized that the computational costs were high but continued to use the brute-force method to find the closest point. Chen & Medioni (1992) made use of the normal vectors which resulted from the scanner points on the surface of the model mesh to find the correspondence. The use of varying resolution models was suggested by Jost & Hügli (2003), for the initial iterations a coarser resolution model is used. For more refined matching, higher and higher levels of resolution are subsequently used. This would help to reduce the number of normal vectors needed, and possibly the computational power and/or space. Both Simon (1996) and Greenspan & Godin (2001) use different caching methods to store a number of closest points from each query, either in the current iteration step or preprocessed beforehand. Subsequent iteration steps can then use the cached points to avoid re-searching the entire data set.

The method used in this work uses k-dimensional (k-d) trees to improve the search time by using binning. A k-d tree is a binary search tree where, starting from the root, each node is split into two subsequent nodes or leaves. Each split node represents a division of the point cloud so that eventually several bins each contain a few points. To traverse a binary tree, the query is compared against the node and is directed to the appropriate branch. However, under specific circumstances it is possible for a query to travel into the incorrect bin. Once the incorrect bin selection recognized, a costly backtrack through the binary tree is required Greenspan & Yurick (2003). The response to this is to use the Approximate Nearest Neighbour (ANN) algorithms to create the approximate k-d tree, see Section 3.4.4.3.

3. Point Weighting

Assigning a weight to corresponding pairs can give certain points or certain areas more emphasis. Greater emphasis on one area should increase the alignment for that area, but possibly at the cost of misaligning another area. The neutral and basic state is to weigh all points equally. Godin, Rioux, & Baribeau (1994) take a look at the angle between normals and at the use of colours as ways of assigning weight. Godin et al. also look at assigning weights based on the distance between corresponding points; points that are separated by a greater degree receive less weight. This is one possible method to reduce or remove the influence of outliers and other process errors.

4. Rejection

Rejection is similar to weighting; a point with a zero weighting can be considered to be rejected. Most rejection algorithms will rely on some metric to establish the quality of a pairing Masuda (2002) suggests rejecting points that are too far apart based on a factor of the standard deviation. Like weighting, rejection is not necessarily required and can also be used to remove outliers.

5. Cost and Cost Minimization

The original ICP algorithm by Besl & McKay (1992) uses the Euclidean distance between corresponding points as the error metric for minimizing. Of all the variants, this metric may be the simplest. The work by Chen & Medioni(1992) pairs a scan point with a point on a virtual plane tangent. The tangent is located at the closest model point on the model curve. Zhang (1994) created a limit upon the error metric to help filter out detrimental pairings; this was done by adding a maximum to the individual distances as well as a method to verify the orientation of pairings.

There are other options which use the Euclidean distance by changing the transformation matrix representing the rigid-body transform. Arun, Huang, & Blostein (1987) developed a strategy using the SVD of a covariance matrix. Two different representations by Horn (1987) and Horn, Hilden, & Negahdaripour (1988) represent the transform using a unit quaternion and as orthonormal matrices.

To search for the next step of the alignment, a new transformation that minimizes the error metric is applied to generate a new point cloud. Besl & McKay (1992) use an extrapolation to find the next transformation while Chen & Medioni (1992) do not.

3.4 Implementation of ICP and PCA Indices in a Simulated Environment

In a real-life application, the input to the ICP algorithm is a point cloud generated by a Light Distance and Ranging (LIDAR) device scanning a space structure. This point cloud, $\vec{p} = \{x, y, z\}$, will be in the LIDAR's frame of reference. In a simulated environment, which is useful for path planning and for scenario testing, the LIDAR scanner must be virtualized and implemented in code. Using a virtual scanner, simulated point clouds can be generated for use with the ICP program. This allows for the testing of ICP program components as well as predictions on the results. To extend the ICP algorithm as an ICP program and simulation framework, three major modules are used:

1. A module for virtualized scanning and point cloud generation
2. A module to calculate ICP and estimate the pose
3. A module to compute ICP accuracy and PCA indices

The virtual scanning module requires a CAD model to scan. The surface representation of a CAD model can be discretized from continuous surfaces and is approximated by a set of triangles known as a mesh. The mesh is comprised of a set of vertices and a set of faces. To make the processing of CAD models easier, the mesh is voxelized. Voxelizing a model is the process of representing the model using large volume elements called voxels. Each voxel acts like a bin by containing the list of all triangular elements that are contained within or intersect it. Adapting algorithms to use the voxels will increase the run speed of the ICP program. However, voxelization is a preprocessing which can take a long time to complete; in addition a significant amount of memory may be required to store it.

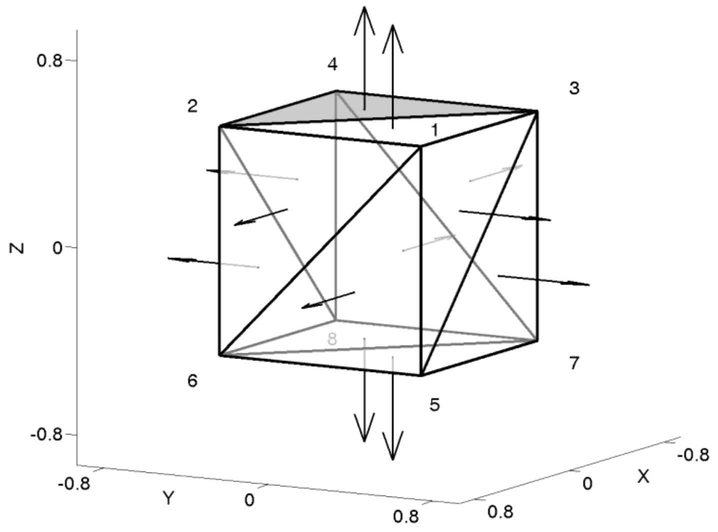
The simulated LIDAR generates an ideal point cloud that is free of any process noise or measurement noise. An ideal point cloud will most likely converge with very accurate pose estimation. To model a real process or scenario, the point clouds must be corrupted in some fashion. Measurement noise can be easily added by individually translating each element of the point cloud according to a zero-mean normal distribution without bias. Process noise is much more difficult, and will require equations to model a LIDAR.

After registration by the ICP module, the accuracy of the simulation is evaluated by using the pose error norm and the error of the individual pose components. This module is also used to calculate the PCA indices by using the point cloud and normals. Although the PCA indices can be computed before ICP registration, they are calculated alongside the pose errors. One of the major outputs of this module is a collection of graphs which relate values of the PCA indices and pose errors. In order to statistically represent the relation, all graphs contain data for 1000 randomized views of the model. The descriptions of the ICP program above will be expanded upon further below. Portions of this work are based on the framework outlined by Shahid (2007).

If real scans are being processed, the scanning module may be bypassed, but voxelization is still performed. In addition, being real scans, the true pose of the point cloud is also an estimate. This makes pose error calculation a relative process rather than absolute.

3.4.1 Meshed Model

The models used in this thesis are three-dimensional triangular sets called meshes. Typically, a mesh begins as a Computer Assisted Design (CAD) model with continuous smooth surfaces and is discretized and converted into a triangular mesh. If the resolution of the mesh is sufficiently high, finely detailed models can be fully represented. Each mesh consists of two sets, a set of vertices and a set of faces. The elements in the vertex set are 3-D points and represent the vertices for each of the triangular surfaces. The entries in the face set are indices which point to entries in the vertex set. Each row in the face set selects three vertices to form a triangular mesh element. Together the vertex and face sets form the entire triangular mesh. Each entry in the vertex set is unique, but can be reused to represent several triangular faces, thus there will always be more faces than vertices. Figure 4 shows an example of a meshed cube alongside a portion of the vertex and face sets, there are 8 vertices and 12 faces in the meshed cube. The second triangular element defined by the face set selects vertices $[2,3,4]$ and has been highlighted in Figure 4.



Vertex Set				Face Set			
#	X	Y	Z	#	v_1	v_2	v_3
2	0.5	-0.5	0.5	1	1	3	2
3	-0.5	0.5	0.5	2	2	3	4
4	-0.5	-0.5	0.5	3	2	4	8
⋮	⋮			⋮	⋮		

Figure 4 - Meshed model

The only constraint on the face set is that the triangles must conform to the right-hand rule; in Figure 4 the normal vectors always point outwards. If the triangle does not conform, then it is possible to have errors in the algorithms for ray tracing or closet point matching. For this work, several models are examined, including: a) Space Shuttle Atlantis; b) International Space Station (ISS) airlock; c) Hubble

Telescope; d) Radarsat satellite; e) Quicksat satellite; f) a skewed pyramid; and g) a reduced-ambiguity cuboctahedron.

The space models are freely available from (NASA) and (The Celestia Motherlode), and can be imported into *Autodesk 3DS* or *Biturn* for conversion or manipulation. While simple CAD models with planar surfaces can easily be represented by a triangular surface; highly detailed ones often include smooth curves which require many more triangular elements to represent them. In addition, the CAD models often include small details that are unnecessary or do not contribute to the overall ICP solution. One possible action to clean and simplify a model is to decimate the number of meshes using *MeshLab* or *MATLAB*. Decimation involves the simplification of a surface to reduce the number of faces and vertices while preserving the overall shape and features, effectively decreasing the model's resolution in certain areas. Decimation typically works very well if it is done slowly in many steps; however, if a large number of triangles are removed in a step, it can potentially leave gaps in the mesh resulting in unintended holes in a model's surface. A gap in the mesh will cause errors when scanning the surface. Another way of simplifying a model is to remove smaller features that will not contribute to the pose estimation or are much smaller than the scanner's scan density and result in zero to few to zero ray intersections. Finally, as the models are highly detailed, there may be inner surfaces and features that are hidden from the outside and cannot be seen by the LIDAR; these should be removed. Removing extraneous surfaces will create savings in computer memory and program runtime as there are fewer elements to process.

3.4.2 Voxelizing

Voxelizing a mesh model is a way of sorting and binning a 3-D object to make it easier for computation. Originally, the use of voxelization was to represent continuous objects as discrete voxel elements. For example, instead of a smooth line, a lower resolution one would take its place; it is akin to modelling a curve using large Lego blocks. Early ray tracing methods used a basic brute force method to process the mesh faces to find where an element and the ray intersected. This means that each and every element was tested to see if it was intersected by every ray generated. This resulted in many intersection tests and a lot of wasted time and effort. To increase the efficacy and efficiency in simulated scanning and ray tracing applications, each model can be broken up into much smaller voxels; a voxel is typically a cube, however, spheres or some other polygon could potentially be used. A cube may be preferred because it allows all the voxels to be mutually exclusive or non-overlapping. Figure 5 shows the meshed model for the Space Shuttle; each cube represents a voxel element. If a sectioned view were taken, there would be a gap in centre of the model as there are no triangular faces in that area and therefore no voxels are required to hold them.

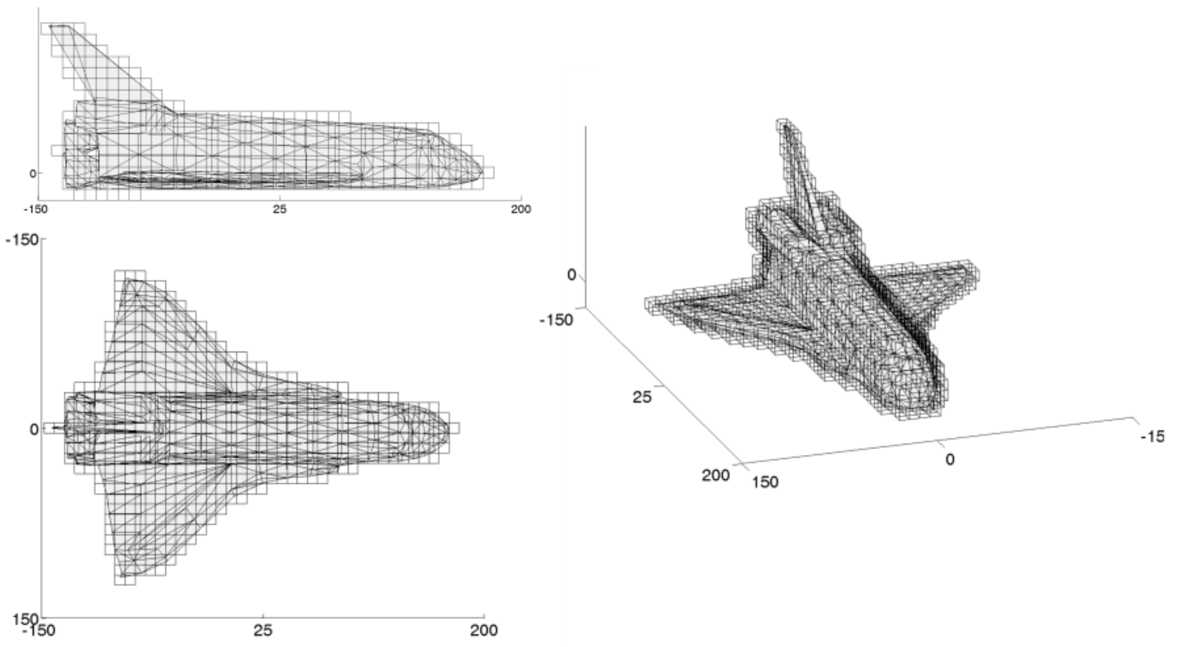


Figure 5 - Meshed and voxelized Space Shuttle

Instead of using the voxels to approximate the model, voxels are used to hold pointers to all the triangular elements that are within or intersect the voxel boundaries. By having a list of triangular elements associated with each voxel a much quicker lookup can be performed and the number of intersection tests is reduced.

Voxelization is a one-time preprocessing step but it can potentially be time consuming. Starting either at the centroid of the model or at one corner of a box bounding, small cubes are established and tested to see if they contain or intersect any triangle elements. A bounding box is a volume that is described by the maximum and minimum values of the entire model along the X, Y and Z axes. Voxels with no associated triangular elements are discarded. A voxel is defined by its centroid and the triangular elements it contains.

The sizing of voxels is important as it dictates the voxel resolution, the number of associated triangles and the size of the voxel in computer memory. Generally, the voxel sizing is constant throughout; however, a variable size voxel could theoretically be used as well. An adaptive voxel could be more useful to contain large triangular elements rather than many small voxels. As voxel sizes decrease, each voxel will have fewer associated triangles; this is advantageous once a voxel selected for testing as fewer ray intersection tests are required. However, a smaller voxel size will increase the number of voxels; if the side length of a cube is divided by two, then the volume of the voxel decreases by eight. Consequently the number of voxels should increase eight-fold; however it is typically less than this amount. Larger voxels tend to include large amounts of empty space, especially if it is located on the fringe of a model. Smaller voxel sizing is generally preferred as it can capture finer details of the overall model; however, Greenspan & Yurick (2003) report that voxel size is relatively independent of accuracy.

A MATLAB implementation of the voxelizer program written in C/C++ was attempted. To make the program speed comparable, incorporation of C code in MATLAB was done through the *MEX* software interface layer. However, one C function did not operate properly and made this implementation unusable until the next iteration of the C/C++ compiler. The *fmod* function frequently returned an indeterminate result which it did not do when run through the standard C/C++ compiler[1]. Currently Microsoft Visual Studio 2008 is being used as the C/C++ compiler.

3.4.3 Scanning and Ray Tracing

Simulating the scanning of a space structure is done by ray tracing a model. With multiple light rays emanating from a virtual LIDAR source, a point cloud can be generated by finding where the ray intersects one or more triangular mesh elements. A valid intersection requires that the mesh element is not occluded from the ray source and is the closest intersection to the ray source. In addition, the normal vector from the triangular mesh element should be in the opposite direction to the ray. The angle between the ray and normal should be greater than 90° and the dot product less than 0. This indicates that the ray is hitting the front side of the triangle, however if the angle is less than 90°, then the ray has hit the back face of a triangle. This most likely means that the model contains a gap or hole in the mesh which allowed the ray to enter inside the model. The other possibility is that the model is not right-hand conformant. Hitting a back-facing triangle can cause that triangle to be ignored or deleted from the mesh or it can be tested for intersection as if it conformed to the right-hand rule; however this is not advised. Figure 6 shows the potential scenario when a ray passes through a triangular face with an incorrect normal. A ray tracing algorithm may choose to bypass face A and select face B as it has a correct normal.

[1] fmod: "http://support.microsoft.com/kb/972497"

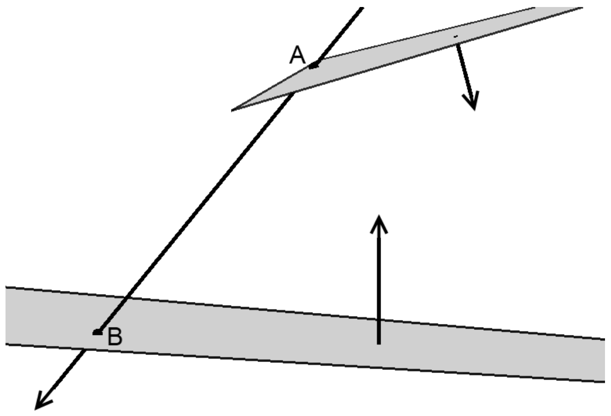


Figure 6 - Occluded triangle

3.4.3.1 LIDARs and Ray Generation

To properly replicate the scanning of an object in the virtual scanner, a Light Detection and Ranging (LIDAR) device must be first introduced. A LIDAR is a laser based device used for tracking and scanning by sending out laser pulses. The range between the LIDAR and a target object can be calculated from the time-of-flight, Equation 3.18. The distance from an object is measured based on the half the time it takes a pulse to return, so the outgoing and returning times are averaged. Because a LIDAR uses laser pulses, the speed of the beam can be approximated as that of light, c, roughly $3 \times 10^8 m/s$. Note that

$$Range = c \times \frac{Time\ delay}{2} \tag{3.18}$$

where range is the distance from LIDAR source to the target, represented by R in Figure 7.

A LIDAR device is also capable of scanning and triangulating objects. By varying the direction of the laser pulses, a point cloud covering the visible faces of the target object is formed. Figure 7, adapted from Beraldin, Blais, & El-Hakim (1995), shows a LIDAR device scanning a surface. The advantage of a LIDAR is that it is highly accurate and is capable of constructing good point clouds for pose estimation. As previously mentioned a LIDAR is independent of the lighting effects and can operate under various light conditions Blais, Beraldin, & El-Hakim (2000).

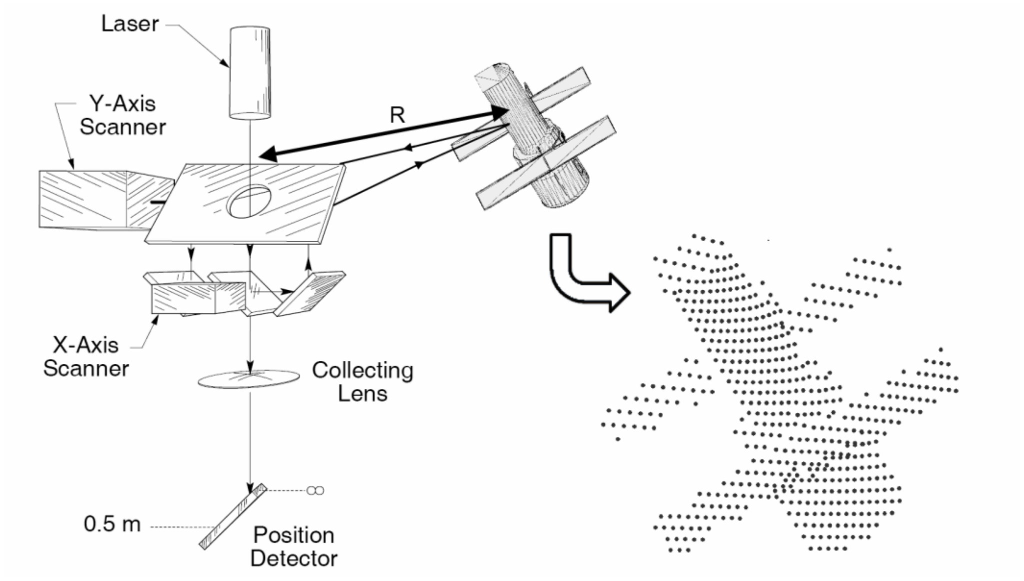


Figure 7 - LIDAR scanning

Figure 7 depicts a LIDAR performing a raster scan to create a point cloud. A raster scan is akin to human eyes reading western text or an old dot matrix printer. Assume that the LIDAR scanner works by rotating a mirror on its vertical and another on its horizontal axis to be able to sweep the forward arc. Therefore, a raster can simply be implemented by simply constructing a direction based on the rotation of along the X and Y axes. If a model is expected to be between θ_{min} and θ_{max}, then a raster scan will define the X or Y components through the beam projections, defined by:

$$\vec{d} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \sin(\theta_x) \\ \sin(\theta_y) \\ 1 \end{bmatrix} \quad \textbf{3.19}$$

The direction vector, $\vec{d}$, is then normalized. Figure 7 shows the X and Y axes scanners or galvanometers used to rotate the mirrors for 3-D scanning.

The problem with a raster scanning is the non-continuous motion required of the mirrors; a line is scanned and then the direction is switched and the next line is scanned. A smooth motion is more advantageous as the mirror can sweep in continuous arcs. Continuous scan patterns are possible by making use of the lissajous, and rose/rosette waveforms Ruel, English, Anctil, & Church (2005) described by Equation 3.20 and shown in Figure 8.

$$\underbrace{\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \mathrm{A}\sin(a\,t+\delta) \\ \mathrm{B}\sin(bt) \\ 1 \end{bmatrix}}_{Lissajous} \qquad \underbrace{\begin{bmatrix} r \\ z \end{bmatrix} = \begin{bmatrix} \cos(k\theta) \\ 1 \end{bmatrix}}_{Rose} \tag{3.20}$$

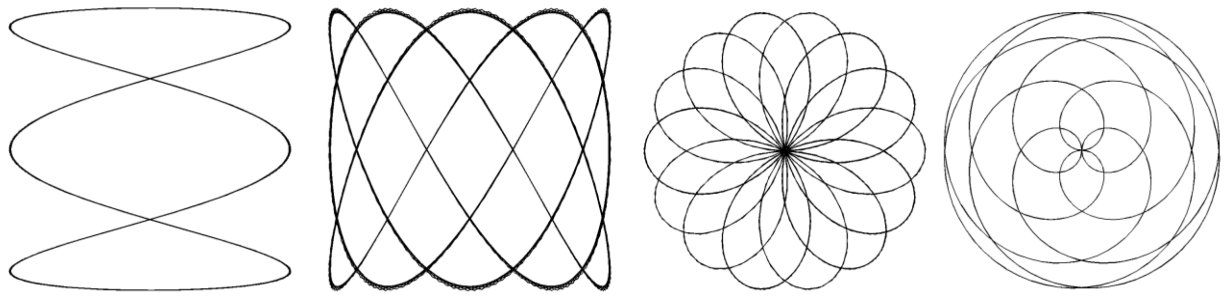


Figure 8 - Two Lissajous and two rosette curves

McTavish, Okouneva, & English (2010) suggest to increase the randomness of a scanning process by rotating the scanner around the boresight or barrel before every scan. This will randomize the scan points if the same view is scanned multiple times or if the model is scanned anew for each trial. Figure 9 demonstrates the difference when boresight spin is employed. The lines indicate the direction of motion of the raster scan over the model; by varying the motion, a different point cloud is generated.

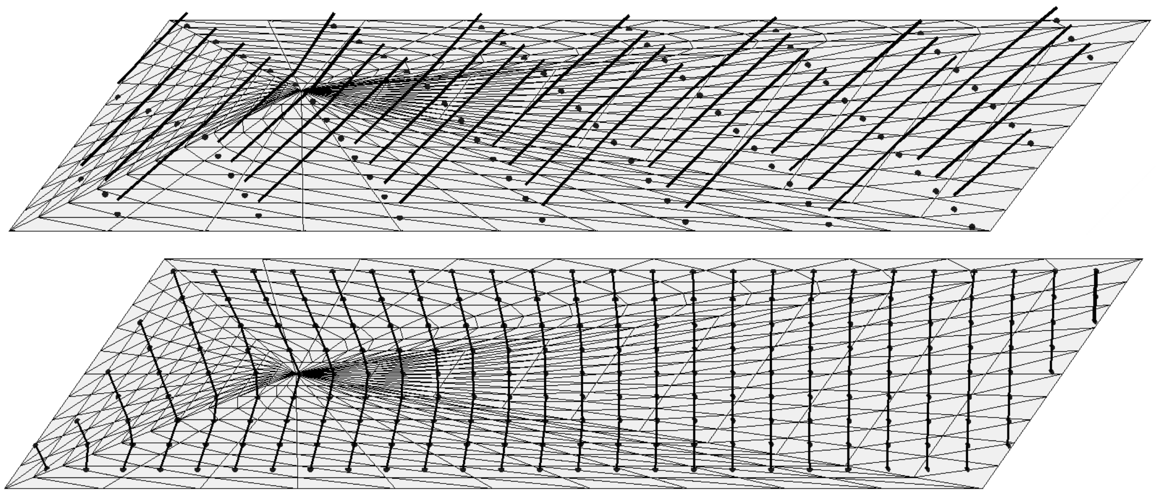


Figure 9 - Boresight spin

3.4.3.2 Ray Tracing

The rays generated from the scanner must be tested to see which, if any, mesh elements are intersected. Ray tracing is a technique used to trace a path from a common source onto the surface of a model. Ray tracing has been used in computer imaging and graphics for rendering special effects such as reflection and area lighting. There are several different methods that have been developed to test if a ray has intersected a triangular surface. The rays have a common source, $\vec{P}_{origin}$, that can be used to represent either a LIDAR, camera, light source or eye. Figure 10 shows the typical ray and mesh intersection; to find the point of intersection the as a function of distance along the ray

$$\vec{P} = \vec{P}_{origin} + t\vec{d} \qquad \textbf{3.21}$$

where $\left\{ \begin{array}{l} \text{t is the Euclidian distance between the origin and the intersection} \\ \vec{d} \text{ is the unitary direction vector of the ray given by Equations 3.19 or 3.20} \end{array} \right.$

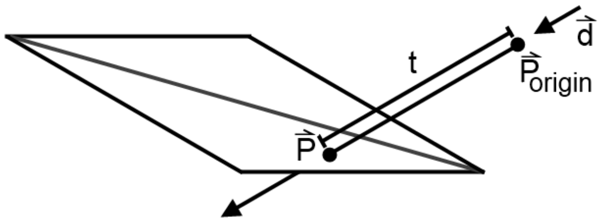


Figure 10 - Ray intersection

There are many algorithms available for ray and triangle intersection testing, including some notable ones by Segura & Feito, O'Rourque, Snyder, Sunday, Möller-Trumbore, Held and Badouel. The common thread with the last four algorithms is that they are all based on barycentric coordinates. Barycentric coordinates can be defined as the ratios of vertices with respect to a reference vertex and can be used a simplex of any dimension. A simplex is the abstraction of a triangle into other dimensions; a 1-D simplex is a line, a 2-D simplex is a triangle and a 3-D simplex is a tetrahedron. Figure 11 shows the typical formulation of barycentric coordinates.

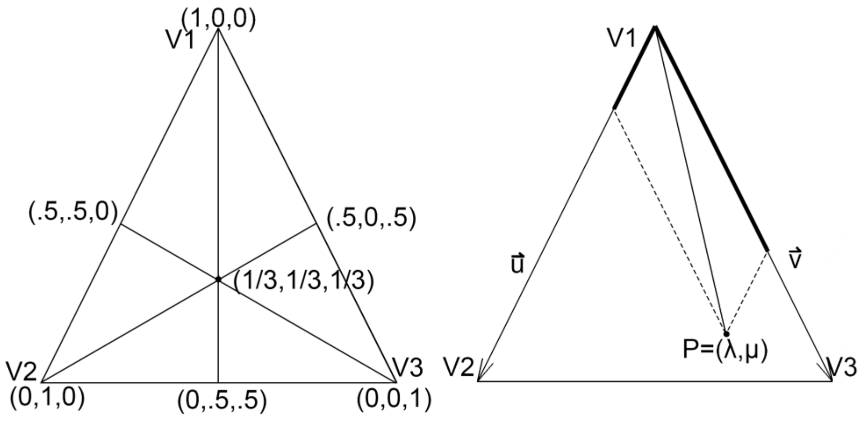


Figure 11 - Barycentric coordinates

If V_1 is chosen to be the reference vertex, then all other points within and around the triangle can be described by

$$\begin{gathered}\overrightarrow{V_1P} = \lambda\overrightarrow{V_1V_2} + \mu\overrightarrow{V_1V_3} \\ P = V_1 + \lambda(V_2 - V_1) + \mu(V_3 - V_1) \\ P = V_1 + \lambda\vec{u} + \mu\vec{v} \\ \vec{P}_{origin} + t\vec{d} = V_1 + \lambda\vec{u} + \mu\vec{v}\end{gathered} \tag{3.22}$$

where $\begin{cases} \vec{u} \text{ and } \vec{v} \text{ are vectors pointing away from the reference vertex} \\ \lambda \text{ and } \mu \text{ are ratios of the overall position on the vector} \end{cases}$

From Figure 11, the query point, $P = [0.2, -0.2]$, is built from by combining vectors parallel to $\vec{u}$ and $\vec{v}$, as seen in Equation 3.22. In barycentric coordinates, $P = (0.233, 0.633)$, this means that the query point P can be found by combining two vectors, moving 23.3% in the direction of V_2 and 63.3% in the direction of V_3.

The main concern with barycentric coordinates is if the query point is within the bounds of the triangle. There are four conditions that will dictate if a point is contained within a triangle

$$\begin{gathered}\lambda \geq 0 \\ \mu \geq 0 \\ \lambda + \mu \leq 1 \\ t \geq 0\end{gathered} \tag{3.23}$$

if the first three constraints hold true, then the point P lies inside a triangle. If $t \leq 0$, then according to Equation 3.22, it is likely that the meshed model has been improperly placed as this indicates the model is behind the LIDAR camera. Several ray-triangle intersection test algorithms use the barycentric principal; within this work, the Badouel and Möller-Trumbore algorithms have been implemented. See Shahid (2007) for a comparison of the various algorithms.

3.4.3.3 *Scanning Implementation*

One way of scanning a model is through an imaging plane, by using the Open Graphics Library (GL) to establish a viewport. The image plane can be used to define the direction vector $\hat{d}$ as seen in Shahid (2007). As an alternative to iterating through all the triangular mesh elements in a model, the voxel set that was created for the k-d tree in the ICP algorithm may be used. Shahid indicated that voxel traversal is possible for ray tracing. Voxel traversal allows a ray to move from one voxel to another without having to search for the next. When a model is voxelized, a matrix of voxels is formed, because of this, it is easy to calculate the next voxel a ray will enter. Using this property, ray tracing can be performed very quickly by avoiding the need to process the entire mesh in search of intersections. Figure 12 (a) demonstrates how a ray can pass through a voxel matrix using the neighbours to move quickly though the model.

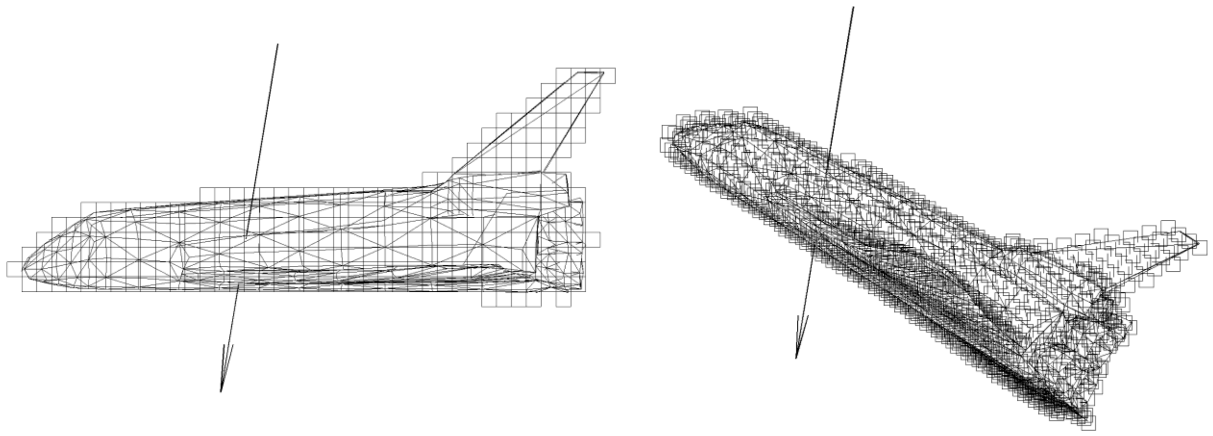


Figure 12 - Voxel traversal

The problem with voxel traversal is that it only works under certain circumstances. There is no problem if the model is only translated as the voxels can be translated as well; see Figure 12 (a). However, if the model is arbitrarily rotated by a value other than $\pm n \cdot 90, n = 0{,}1{,}2$ …, then the voxels will no longer be aligned in a convenient matrix. Moreover, the voxels were previously aligned to the principal X, Y and Z axes; once the model is rotated, and the voxel matrix is rotated as well, but the voxels will not be. Figure 12 (b) shows how the voxels are no longer aligned once the model is rotated. As the voxel matrix is no longer aligned, voxel traversal is not possible. This is most noticeable in the area around the tail.

To make use of the voxels in ray tracing, a program was written to select all the voxels which pass close to the ray. By selecting the closest voxels, there should be a reduction in the number of triangle intersection tests. This method used MATLAB's function *inpolygon*, which finds all points within and upon a prescribed polygon. In this case, the points were the centroids defining the voxel elements and the polygon was a wide ray. The intention is to grab all the voxels that are within a certain projected distance of the ray projection. As C++ programs generally operate quickly in procedural programs, the C++ implementation is much quicker when there are fewer faces and mesh elements. However, when there are more faces, above 10,000, then the MATLAB implementation using voxels operates quicker.

The general steps for scanning a model for a given view are:

1. Determine camera or origin position, and the scanning method used (raster, lissajous, rosette)
2. Based on the scanning method, determine the ray directions
3. Find the intersecting triangle(s)
4. Determine the intersection point with parametric value t
5. Repeat steps 2-4 until the scanning method has been completed

3.4.3.4 Pose Randomization

In the OpenGL/C++ version of the scanner, there is a limit as to where a model can be placed. The scanned model must be contained within an X and Y frontal arc which defines a viewing volume, see Figure 13. To randomize the views, the half chord of the model is found as the vertex the furthest point from the model's centroid. Using the half chord, a spherical envelope is defined to represent all possible combinations of roll, pitch and yaw. The spherical envelope may be shifted by its centroid within the frontal arc without worrying if some part of the model will exceed the viewing volume. If the viewing volume is exceeded, then that part of the model will not be scanned.

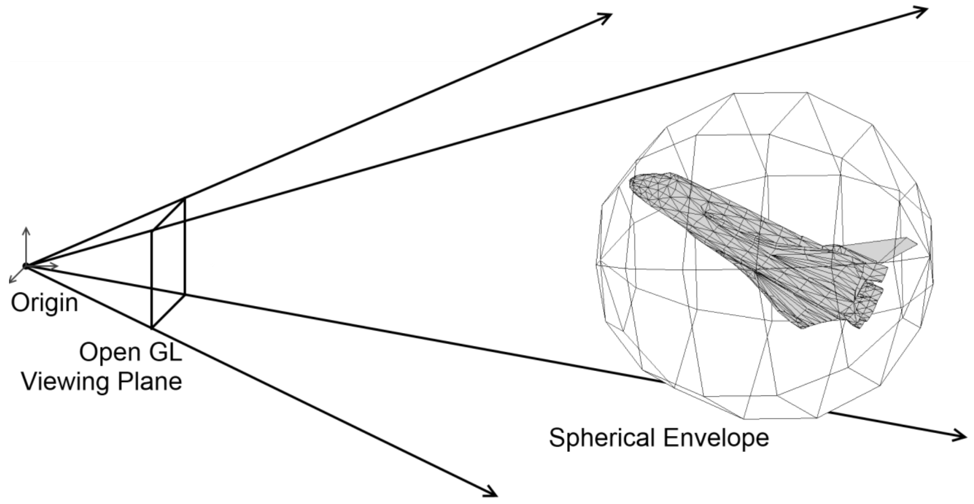


Figure 13 - Open GL viewing volume

For each model, 1000 poses are typically generated using this random process. The only stipulation is that gimbal lock must be avoided. Due to the way that the roll, pitch and yaw angles are represented, it is possible to be stuck in a position where there is a loss of one degree of freedom, typically this happens when pitch angle approaches $\pm 90°$. In this position, the yaw axis and the roll axis align which allow yaw and roll lose their uniqueness. Therefore, the random pose rotation values are limited to $[\text{roll}, \text{pitch}, \text{yaw}] = [\pm 180°, \pm 89°, \pm 180°]$. Additionally, some models should be limited to avoid certain views; i.e., the airlock is currently mounted onto the ISS, so the back face and arc where it is mounted should not be accessible.

3.4.4 ICP Program

Once the point cloud $\{p_i\}$ has been generated the virtually scanner, the ICP algorithm can utilize it to simulate pose estimation using the equations from Section 3.2. At this point, $\{p_i\}$ is made up of perfect/ideal data without noise of any sort, and if registered to the CAD model, the ICP algorithm would mostly result in little to no error. To simulate real LIDAR scanning, noise must be added to the point cloud $\{p_i\}$. A new noisy point cloud $\{\widetilde{p_i}\}$ is generated every time ICP is run. An initial guess is also generated every time the ICP program is run.

To align $\{\widetilde{p_i}\}$ with the CAD meshed model, a model point cloud, $\{m_i\}$, is generated during each iteration step. To generate $\{m_i\}$, each point in $\{\widetilde{p_i}\}$ is projected onto the surface of the meshed model. The closest projected point on the surface of the meshed model is used to create $\{m_i\}$. Since there are many triangular elements in the mesh and many points in $\{\widetilde{p_i}\}$, an efficient algorithm to search for the closest projected point is required.

For each view being registered, the ICP program is run 100 times to generate statistically valid results. Each run includes the generation of $\{\widetilde{p_i}\}$ and the ICP pose registration.

3.4.4.1 LIDAR Simulation and Noise

Process and measurement noise can be added to $\{p_i\}$ by modelling the LIDAR parameters and limitations. In this work, the performance of Optech's Rendezvous Laser Vision System (RELAVIS) apparatus is replicated. The relevant LIDAR parameters are presented below in Table 1.

Table 1 - RELAVIS specifications

Minimum Range	0.5 m
Maximum Range	5000 m
Range Accuracy	0.01 m
Bearing Accuracy	0.00035 rad

To model a LIDAR, the internal arrangement of mirrors must be considered. Typically, the mirrors in a LIDAR are not completely aligned, as seen in Figure 14, adapted from Blais, Beraldin, & El-Hakim (2000).

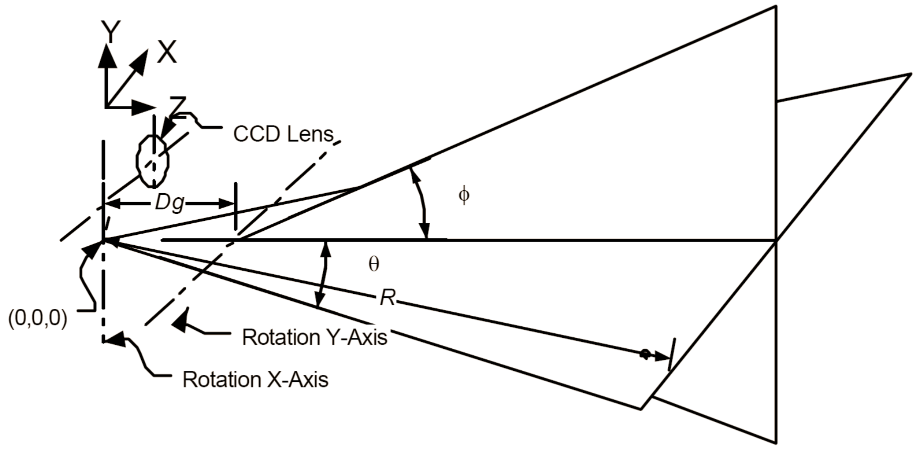


Figure 14 - LIDAR astigmatism

For a LIDAR to generate the beams, the laser source itself is fixed and does not rotate or translate. Instead, several sets of mirrors are rotated to direct the beam. For accurate measurements, precision galvanometers are used to reflect the laser beam to the correct location. The space between the X-axis mirror, Y-axis mirror, the charge-coupled device (CCD) and the lens creates the astigmatism seen in Figure 14. The offset between the X and Y rotation axes is the called the astigmatism which occur when two scanning axes are not aligned or symmetric. Although astigmatisms are frequently intentionally built into some telescopes, they are detrimental if developed in the human eye. The X, Y and Z coordinates of a point given by the LIDAR are modelled by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \times \begin{bmatrix} \sin(\theta) \\ (\cos(\theta) - \psi) \times \sin(\phi) \\ (1 - \cos(\theta)) \times \psi) + \cos(\theta) \times \cos(\phi) \end{bmatrix} \qquad \mathbf{3.24}$$

where $\left\{ \begin{array}{l} D_g \text{ is the distance between the rotating mirror axes} \\ R \text{ is the distance between the LIDAR and the scanner target, see Equation 3.18} \\ \theta \text{ and } \phi \text{ are the deflection angles along their respective axes} \\ \psi \text{ is the ratio between } D_g \text{ and } R \end{array} \right.$

Typically, the separation between the rotation axes is small relative to the distance to the target, $D_g \ll R$; therefore both D_g and ψ are negligible. The overall system uncertainty can then be approximated by taking the partial derivatives of Equation 3.24

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \frac{d}{dR}\begin{bmatrix} x \\ y \\ z \end{bmatrix} + \frac{d}{d\theta}\begin{bmatrix} x \\ y \\ z \end{bmatrix} + \frac{d}{d\phi}\begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \mathbf{3.25}$$

and applying $\psi \approx 0$,

$$\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta z\end{bmatrix} = \begin{bmatrix}\sin(\theta)\\ \cos(\theta)\times\sin(\phi)\\ \cos(\theta)\times\cos(\phi)\end{bmatrix}\times\Delta R + \begin{bmatrix}\cos(\theta)\\ -\sin(\theta)\times\sin(\phi)\\ -\sin(\theta)\times\cos(\phi)\end{bmatrix}\times R\times\Delta\theta + \begin{bmatrix}0\\ \cos(\theta)\times\cos(\phi)\\ -\cos(\theta)\times\sin(\phi)\end{bmatrix}\times R\times\Delta\phi \tag{3.26}$$

Now that Equation 3.26 has been defined, noise can be added to the point cloud $\{p_i\}$. The maximum range error, σ_{range}, and the maximum angle error, σ_{angle} are found from Table 1.

$$\begin{bmatrix}\Delta \mathrm{R}\\ \Delta\theta\\ \Delta\phi\end{bmatrix} = \begin{bmatrix}\sigma_{range}\times randn(-1,1)\\ \sigma_{angle}\times randn(-1,1)\\ \sigma_{angle}\times randn(-1,1)\end{bmatrix} \tag{3.27}$$

Equation 3.27 uses MATLAB's *randn* function to create the noise in the system with a mean of zero and without basis. The *randn* function distributes values from [0,1] based on the normal distribution. Each time a trial is run, a new set of noise variables are generated.

Using Equation 3.27 is somewhat computationally intensive as the noise values θ, ϕ and R all need to be computed for each and every point. Once $\Delta x, \Delta y$, and Δz have been computed, the scanner point cloud, $\{\widetilde{p_i}\}$, is updated through

$$\{\widetilde{p_i}\} = \begin{bmatrix}\Delta_x + p_{i,x}\\ \Delta_y + p_{i,y}\\ \Delta_z + p_{i,z}\end{bmatrix} \tag{3.28}$$

The noise introduced must be at an appropriate level. If too little noise is added, the scanner point cloud will not resemble a real point cloud and will be closer to an ideal point cloud; thus making it difficult to simulate a true LIDAR and ICP program. However, if too much noise is added, the ICP closest point matching algorithms will have difficulty establishing the points to their true CAD model surfaces. Unless the scanning is very dense, the noise level should be much smaller than that of the scanning density. Figure 15 represents a Shuttle raster scan with two different noise levels; the circles represent the possible location of a point with noise added. If the noise is low, Figure 15 (a), then the points will appear within circles with a small radius r_1 and the points will not move too far out of position. Figure 15 (b) depicts a raster scan with high noise; the possible point locations overlap with a larger radius r_2. This means that the points move more, criss-cross and can cause holes in the raster scan, all of which is undesirable. When the noise is low, the ICP results are inherently more accurate. When the noise on a point is capable of bringing it further than the adjacent point(s), the ICP algorithm may struggle.

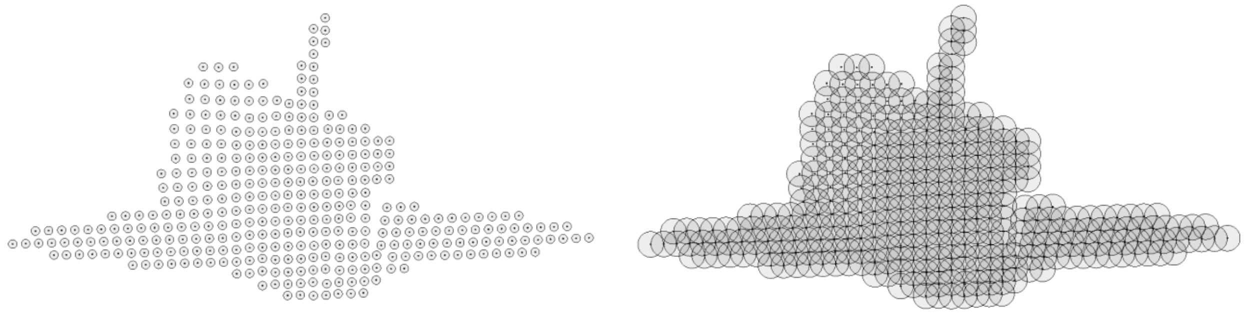


Figure 15 - Raster scan low and high noise

3.4.4.2 Initial Guess

To begin, an ICP algorithm requires an initial guess to iteratively refine. In simulation, the initial guess is used to move the point cloud $\{\widetilde{p_i}\}$ away to form the initial pos;, see Figure 16. If the initial guess translation and/or rotation values are too far away from the true values, it is possible for the ICP algorithm to fall into a local minimum. Once in a minimum, the ICP algorithm is unable to climb out and will stop iterating due to the increasing cost associated with moving out of the minimum. A new initial guess and initial pose is calculated as each time an ICP trial is run. If a view is well-constrained, then it will typically converge to the correct alignment. If a view is poorly-constrained, then the initial guess may cause the ICP algorithm to fall into a local minimum.

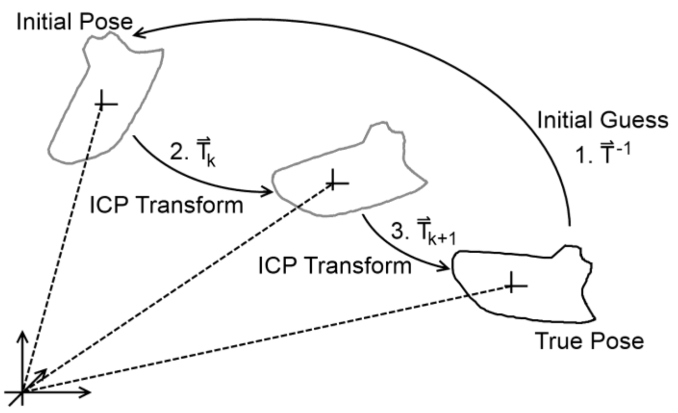


Figure 16 - Initial guess and pose

There have been attempts to work around the problem of local minimums by applying basin filling methods Ling, Singh, & Brown (2007). Figure 17, adapted from Ling et al., shows an example of basin filling; this technique involves slowly increasing the weight on points that have poor registration cost.

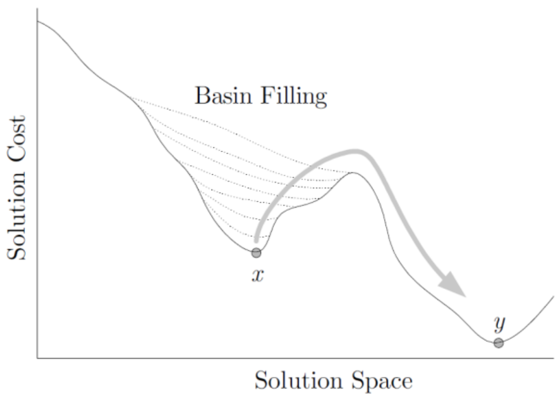


Figure 17 - Basin filling

By focusing the algorithm on the poorly registered point pairs, the point cloud should move to correct them and bring them into alignment. The problem with basin filling is the time and power it takes to move the point out of the minimum with trial and error. This can lead to the ICP algorithm being infeasible for real-time operations due to the computational time required.

If the initial guess is too far from the true pose, then it is possible that other algorithms are used to achieve a closer initial pose. However, in a space rendezvous application, the most likely target will be a known satellite in need of servicing or repair. In this case, a good initial pose will likely be known from other means.

A poor initial rotation value is more likely to impact the convergence than a poor initial translation. Although the initial guess requires both a translational and rotational component, rotational component seems to be much more important. The translational component of an initial guess can be partially solved by bringing the centroids of the two point clouds together.

After noise has been added to create point cloud $\{\tilde{p}_i\}$, it is transformed so that it is intentionally misaligned. If a point cloud is brought into alignment using the transformation T, then the initial pose and initial point cloud can be found roughly via T^{-1}; see Figure 16. Note, this is not the same transformation used to bring the model from the base state to the current pose, but is the transformation between the ideal and guess poses.

To calculate T^{-1}, an initial rotational disturbance, θ, and an initial translational disturbance, τ, are required. The maximum range of translational and rotational disturbances are user set values, τ_{max} and θ_{max}, respectively. By making use of MATLAB's random uniform number generator function, *rand*, a random initial guess can be computed via

$$\begin{aligned} t_x &= \tau_{max} \times rand(-1,1) \\ t_y &= \tau_{max} \times rand(-1,1) \\ t_z &= \tau_{max} \times rand(-1,1) \\ \theta &= \theta_{max} \times rand(-1,1) \end{aligned} \qquad 3.29$$

Using the rotational disturbance θ, T^{-1} can be constructed using the axis angle theorem. Euler's rotation theorem dictates that any rotation can be represented by a rotation about an arbitrary axis as seen in Figure 18.

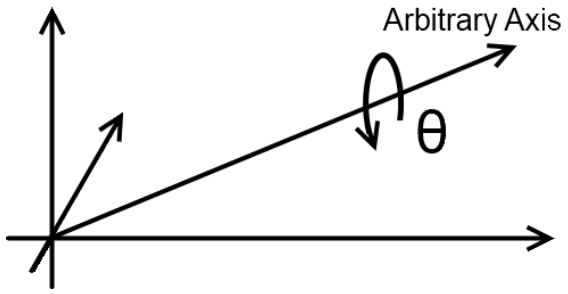


Figure 18 - Axis angle

Using the Euler rotation matrices, the sequence

$$\tilde{R}_0 = \tilde{R}_x \tilde{R}_y \tilde{R}_z \tilde{R}_{-y} \tilde{R}_{-x} \qquad 3.30$$

outlines the rotation matrix for a rotation about an arbitrary axis (Ellzey, Kreinovich, & Peña, 1993). $\tilde{R}_x$, $\tilde{R}_y$ use projected line segments for rotation instead of the primary X and Y axis. Equation 3.30 can be simplified to

$$\tilde{R}_0 = I_{3\times3} \cdot cos\theta + (1 - cos\theta) \cdot u \otimes u - u^x \sin\theta \qquad 3.31$$

where $\begin{cases} u \text{ is the normalized unit vector defining the rotation axis} \\ I_{3\times3} \text{ is the } 3 \times 3 \text{ identity matrix} \\ \otimes \text{ is the outer product operator} \\ \theta \text{ is the amount of rotation about the arbitrary axis} \end{cases}$

In axis angle rotation, a random axis, $\hat{u}$, is required. The normalized axis, $\hat{u}$, is given by

$$\begin{aligned} u_x &= rand(-1,1) \\ u_y &= rand(-1,1) \\ u_z &= rand(-1,1) \\ \hat{u} &= u/|u| \end{aligned} \qquad 3.32$$

After the rotational disturbance has been applied, the translation disturbance is applied. Performing the translation and rotation in opposite order would cause a very large rotation.

3.4.4.3 Voxel and k-d Search

A k-dimensional (k-d) tree is a binary search tree; this means that, starting from the root, each node only has two possibilities, left or right, which allows for a quick search requiring only a few comparisons. A k-d tree can be an efficient means of finding the nearest point to a query point. K-d trees were developed by Friedman, Bentley, & Finkel (1977) and work by repetitively subdividing the k-space until the desired number of values is in each subdivision. For this work, the k-d tree was applied to the 3-D voxels, with $k = 3$. Search trees, such as k-d trees, are made up of several components analogous to a real tree; the root, the branches and the leaves. The root and the branches are nodes where decisions are made as to how the program will traverse through the tree; a node contains only the information needed to make the comparison. Each node might contain the dimension to compare and the over/under value. The root is the starting point of the tree from which several branches may be visited before finding the final leaf. The leaves are where the data is located; once a leaf is reached by the search algorithm, it stops the tree traversal and begins to process the data found within that leaf. A leaf can represent a bin containing one or more points.

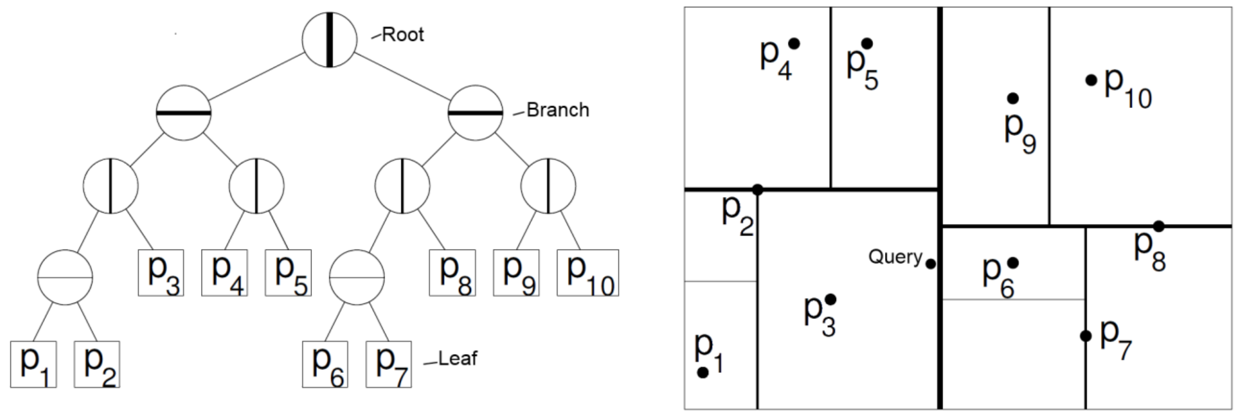


Figure 19 - k-d tree

Figure 19, adapted from Mount & Arya (2010), represents a typical 2-D k-d tree. Each point, p_i, has X-Y coordinate values and has been split into partitions with a bin size of one. To search for the query point, p_q, positioned at (x_q, y_q), the root is the first comparison. The query point's X-dimension value is compared to the value held by the root node, as it is less than the root's, the program traverses into the left branch. At the next branch, the y_q is compared and is again found to be smaller, the program traverses into the left branch and so on. Eventually the traversal selects the leaf containing p_q and selects p_3 as the closest point. The advantage of a k-d tree is that they grant the able to search quickly for a leaf without having to resort to the brute force method of calculate the Euclidean distance for each p_i. By comparing the values of a query point in the k-space, the nodes are traversed to efficiently find the leaf. The problem with k-d tree is when $p_q$is too close to the partition boundaries, as seen in Figure 19, the true closest point of $p_q$is p_6 and will be discussed further below.

There are a few different ways to formulate a k-d tree from data; one commonly used method is to split the points across the dimension with the greatest range. By selecting the dimension with the maximum range, the points are quickly sorted in two partitions based on the average value of the maximum range. In Figure 19, initially the X-dimension has the greatest range and the points are split based on the average X value. Points less than or equal to the average are shunted into the left node of the k-d tree while the rest are placed into the right node of k-d tree. Each node will store the average value and the dimension to compare for future use. The k-d tree creation continues by searching for the greatest range within the new partitions, the splitting dimension does not have to alternate like in Figure 19. Splitting will occur until the number of points is below a desired threshold in each resulting partition.

The concept of a nearest neighbour search is to find the closest existing data point to a query point. The k-d tree enables the search for the nearest neighbour of a query point much quicker than a brute force method. Rather than have to check every single point to find the closest, the k-d tree quickly narrows the number of points to test to the number of data points in the bin. While the k-d tree traversal may correctly discover the bin that would contain p_q, the resulting closest point may be incorrect. It is possible to backtrack through the k-d tree traversal in an attempt to find the proper bin containing the point closest to p_q. However, backtracking requires extra computation power and time. The response to this is to use the approximate k-d tree method (Greenspan & Yurick, 2003).

The approximate k-d tree uses the Approximate Nearest Neighbour method by Mount & Arya (2010). *ANN: A Library for Approximate Nearest Neighbour Searching* will quickly return what it believes to be the approximate closest neighbour using the k-d tree (Greenspan & Yurick, 2003). Using the approximate k-d tree will generally return the correct nearest point. However, when the approximate k-d tree does not return the closest point, the returned point is still serviceable due to the small point cloud density. The advantage of using the *ANN* library and the approximate k-d tree is that the results are returned quickly and fairly accurately in comparison to the basic k-d tree. Additionally, the *ANN* library will return the Euclidian distance between the query point and the resultant point. It is possible to request n closest neighbours from the *ANN* library, which would result in n results and distances. Using this library, each entry in the scanner point cloud $\{\tilde{p_i}\}$ can be tested and the total Euclidian distance can be computed. The Euclidian distance can later be used as part of the MSE seen in Section 3.2.

For the implemented ICP program, the ANN library will return the voxel containing the triangular mesh elements that are the closest. By testing these triangular mesh elements, the closest point can be calculated using a projection algorithm.

3.4.4.4 Closest Point Matching

The ICP algorithm listed in Section 3.2 evaluates the match between two point clouds $\{\tilde{p_i}\}$ and $\{m_i\}$. However, at this point in the ICP program, the only input available is the noisy point cloud $\{\tilde{p_i}\}$, a meshed model, and the voxel set. To generate the second point cloud, a corresponding point on the surface of the mesh is projected for each entry in $\{p_i\}$. This is done by finding the triangular element with the closest projection of $\{p_i\}$. Evaluating each element of $\{p_i\}$ as a query point, the approximate k-d tree algorithm will return the voxel with its centroid that is either the closest, or one of the closest points to the query point. Using the voxel's list of triangular mesh elements, the query point is projected several times in an effort to find the projection with the smallest distance. The projected point that has the smallest normal distance is selected as the closest point $\{m_i\}$. In this manner, the scanner point cloud is used to generate a similar sized model point cloud $\{m_i\}$, where the ith element of $\{m_i\}$ corresponds directly to the ith element of $\{p_i\}$.

Figure 20 depicts a point p_i that searching a corresponding point m_i. The after searching for the nearest voxel, the highlighted voxel elements are processed to find the triangular element that contains the closest point to p_i. By using the voxel list, there are much fewer elements to project upon than if the whole mesh were used. The number of projections is reduced when the voxel size is also reduced, thus limiting the average number of triangles that intersect or are contained by the voxel. The voxel size in Figure 20 is large for demonstration purposes.

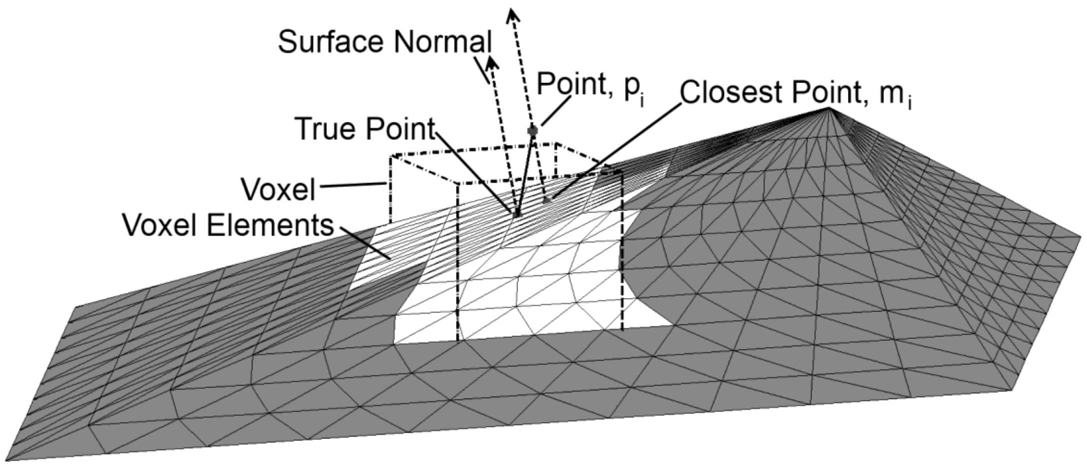


Figure 20 - Closest point generation

Figure 21 demonstrates that the smallest distance between a query point and a plane is through the plane's normal, $\hat{n}$. Each triangle in the mesh can define its own plane and normal. At this point, the distance between the query point and a plane is unknown, as is where the location of the query point when it is projected onto the triangle plane. It is possible that the projection of the query point will fall inside the triangle, outside the triangle or on one of the triangle edges.

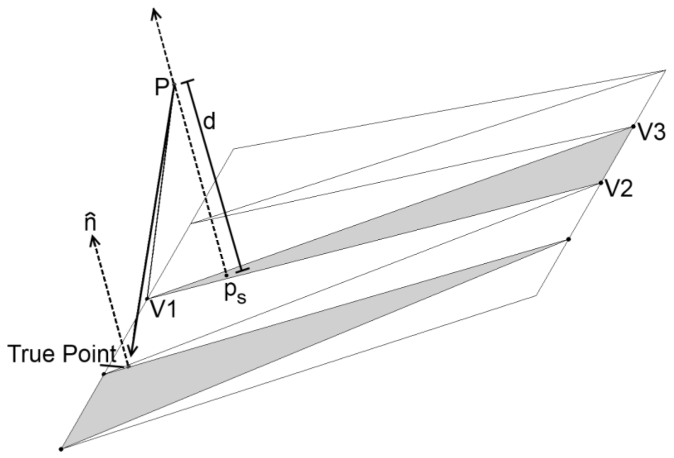


Figure 21 - Point to plane distance

The absolute normal distance between a query point, $\vec{P}$, and a triangle defined by vertices $[\overrightarrow{V_1}, \overrightarrow{V_2}, \overrightarrow{V_3}]$ can be related through the dot product

$$d = \hat{n} \cdot (\vec{P} - \vec{V}_1) \tag{3.33}$$

Using the normal distance, d, the projection point on the surface, p_s, can be found through

$$p_s = \vec{P} - d\hat{n} \tag{3.34}$$

As previously mentioned, the projected point can potentially lie outside boundaries defined of the triangle. A point lying outside the triangle can be projected onto the closest line segment of the triangle or onto the closest vertex. Ericson (2004) describes a way to use triangle barycentric coordinates to define the zones in Figure 22.

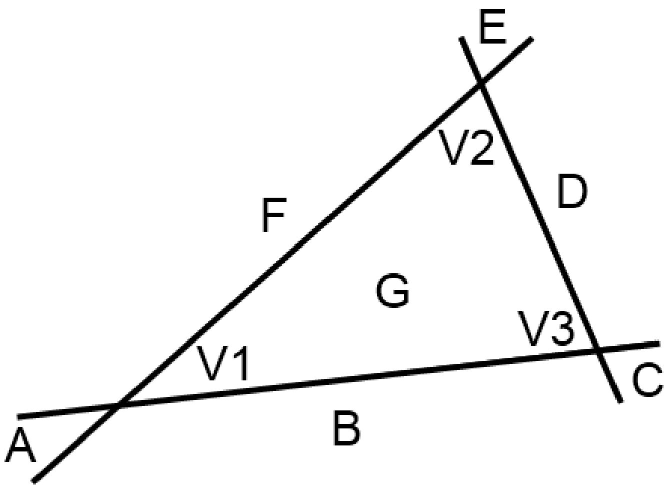


Figure 22 - Triangle projection areas

The method used to find the barycentric coordinates was listed in Section 3.4.3.2. From Ericson, Table 2 summarizes the options for point projection.

Table 2 - Triangle projection

Region	$\lambda < 0$	$\mu < 0$	$\lambda + \mu > 1$	Closest Point
A	✓	✓	×	$\vec{V}_1$
B	×	✓	×	Project on line segment $\vec{V}_1\vec{V}_2$
C	×	✓	✓	$\vec{V}_2$
D	×	×	✓	Project on line segment $\vec{V}_2\vec{V}_3$
E	✓	×	✓	$\vec{V}_3$
F	✓	×	×	Project on line segment $\vec{V}_1\vec{V}_3$
G	×	×	×	p_s

To project a point onto a line segment, the decomposition of the dot product may be used. The line formed by the query point and the projected point is perpendicular to the line segment and the dot product between the two is zero (cosine of 90°.) The projected point can also be defined by its relative distance from one end of the line segment, i.e.,

$$p_s = V_1 + \alpha \times (V_2 - V_1) \qquad \textbf{3.35}$$

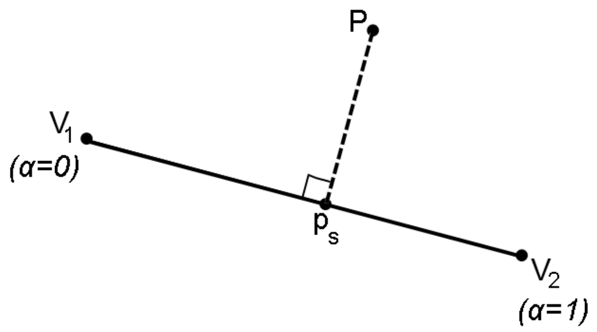


Figure 23 - Point to line

From Figure 23

$$(P - p_s) \cdot (V_2 - V_1) = 0$$

$$\left(\underbrace{(P - V_1)}_{v} - \alpha \times (V_2 - V_1)\right) \cdot \underbrace{(V_2 - V_1)}_{u} = 0 \quad \textbf{3.36}$$

Where α is the ratio of the overall position on the vector, similar to the barycentric parameters λ and μ

Solving Equation 3.36 for α, one arrives at:

$$\alpha = \frac{u \cdot v}{u \cdot u} \quad \textbf{3.37}$$

The parameter α differs from the barycentric parameters λ and μ in that it is bounded by $[0,1]$. The projected point must fall somewhere on the line segment.

$$\alpha = \begin{cases} 1 & \alpha > 1 \\ \alpha & 0 \leq \alpha \leq 1 \\ 0 & \alpha < 0 \end{cases} \quad \textbf{3.38}$$

Using the voxel search and the closest point projection, the following steps are used to generate the model point cloud $\{m_i\}$ from the noisy point cloud $\{\widetilde{p_i}\}$:

1. Select a point from $\{\widetilde{p_i}\}$ to be the query point P
2. Use the approximate k-d tree to find the voxel closest to P
3. Test the triangular mesh elements in the voxel to find the smallest normal distance, $\hat{n}$
4. Project P onto the surface the plane of the closest triangle to form p_s
5. Convert p_s into barycentric coordinates λ and μ
6. Use the barycentric coordinates to project p_s onto the closest triangle
7. Store p_s in $\{m_i\}$ as the corresponding point to P
8. Repeat steps 1 through 7 for all points in $\{\tilde{p}_i\}$ to complete the model point cloud $\{m_i\}$

3.4.5 Evaluation of ICP Results

After the ICP program has completed the registration of the point cloud to the CAD model, the accuracy of the result must be evaluated. There are six individual degrees of freedom in the system, this corresponds to a translational and rotational value for each of the X, Y and Z axes. One way to predict or indicate the registration accuracy is through the use of PCA-based indices computed from the constraint matrix. As previously mentioned, the evaluation of errors is absolute in a simulation environment; an exact true pose is known. When real data is known, the true pose is an estimate which makes the evaluation of errors a relative comparison.

3.4.5.1 Registration Error

There are a few ways to represent the pose error. The most common way can be seen in Simon (1996); the error is represented by the transformation required to translate and rotate the output of the ICP program to the true position. The transformation from the camera origin to the target model is represented as the true transformation, T_{True}. The result of the ICP algorithm is the ICP transformation T_{ICP}. The error transformation, T_E, is due to the noise added to the data and relates T_{True} to T_{ICP}; see Figure 24.

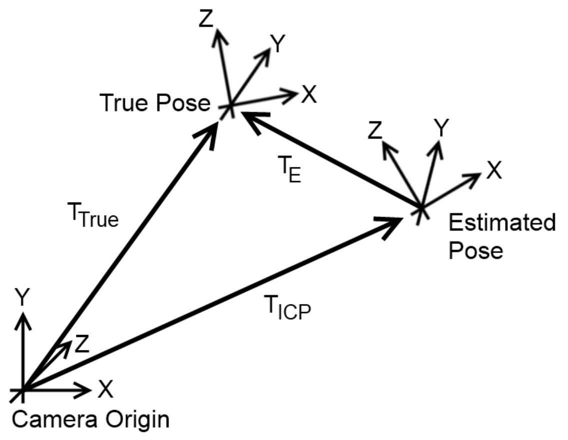


Figure 24 - Error transformation

From vector mechanics, the error transformation can be found through

$$\begin{aligned} T_{ICP} \cdot T_E &= T_{True} \\ T_E &= (T_{ICP})^{-1} \cdot T_{True} \end{aligned} \qquad 3.39$$

T_{ICP}, T_E and T_{True} are all represented by a homogeneous transform matrix in the form of Equation 3.5. The rotation values in T_E are the rotational errors, $\omega = [\omega_x, \omega_y, \omega_z]$, and T_E is represented as

$$T_E = \begin{bmatrix} c\omega_x c\omega_y & -s\omega_x c\omega_z + c\omega_x s\omega_y s\omega_z & s\omega_x s\omega_z + c\omega_x s\omega_y s\omega_z & t_x \\ s\omega_x c\omega_y & c\omega_x c\omega_z + s\omega_x s\omega_y s\omega_z & -c\omega_x c\omega_z + s\omega_x s\omega_y c\omega_z & t_y \\ -s\omega_y & c\omega_y c\omega_z & c\omega_y c\omega_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{3.40}$$

$$\text{where} \begin{cases} c\omega_x \text{ is } \cos(\omega_x) \\ s\omega_z \text{ is } \sin(\omega_z) \\ t \text{ is the error in the translation of the point cloud} \end{cases}$$

It is expected that the translation and rotation errors will be small in general and comparatively to T_{ICP}. Because of this, the small angle assumption can be used:

$$\theta_{error} \ll \theta$$

$$\cos(\theta_{error}) \cong 1$$

$$\sin(\theta_{error}) \cong \theta_{error}$$

The small angle assumption simplifies the transformation matrix, Equation 3.40, to

$$T = \begin{bmatrix} 1 & -\omega_x & \omega_y & t_x \\ \omega_x & 1 & \omega_z & t_y \\ -\omega_y & \omega_z & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{3.41}$$

which can be simplified to

$$T = \begin{bmatrix} (1 + \omega^{\times}) & t \\ 0 & 1 \end{bmatrix} \qquad \mathbf{3.42}$$

From the above two formulations, the rotational errors ω_x, ω_y and ω_z can easily be found by selecting corresponding elements from the transform matrices.

The translation error can be calculated directly from the ICP registration results as

$$t_{error} = t_{true} - t_{ICP} \qquad \mathbf{3.43}$$

The total error for translation and rotation can be found by normalizing the values. The translation error is given as

$$|E_t| = \sqrt{t_x^2 + t_y^2 + t_z^2} \qquad \mathbf{3.44}$$

Similarly, the rotation error is found through

$$|E_\omega| = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2} \quad \textbf{3.45}$$

The translation and rotational component errors can be combined as a six-element vector called the pose error:

$$p_\varepsilon = [t_x \quad t_y \quad t_z \quad \omega_x \quad \omega_y \quad \omega_z] \quad \textbf{3.46}$$

However, depending on the size and scaling of the model, the rotational components may outweigh or be outweighed by the translational components, and will be discussed in Section 3.4.5.2.

The error representing the rotation is not of the same form as that of the translation; it is not represented as $\omega_{error} = \omega_{true} - \omega_{ICP}$. This is the value that, in part, represents the amount of rotation needed to bring the point cloud from the frame defined by T_{ICP} to T_{True}. Therefore, it is the incremental rotation that will bring the point clouds into alignment. Another way of representing the rotational error is by selecting components from both the rotation matrix, built using the roll, pitch and yaw values, and the quaternion matrix.

The problem with this representation is that gimbal lock may occur when the value of pitch approaches $\pm\pi/2$. At this value, errors will be difficult to recover due to the loss of one degree of freedom and the non-exclusivity of the pose representation. If the values of roll, pitch and yaw values are allowed to exceed 2π, then another set of values maybe used to represent the identical pose. Values for roll, pitch and yaw can be found by taking the inverse sine, inverse cosine and inverse tangent of select elements of R_{rpy}.

Another problem with this representation is due to the cyclic nature of the angles. For example, when the true angle is approximately 0°, the ICP program will return the roll, pitch or yaw some runs as $-1°, 0°, 2°$ or $359°$. Each of the previous answers is valid, as 359° and -1° are identical. However, if the standard deviation and mean are calculated using the unmodified results, then the mean is found to be 89.75°, which is incorrect as the true mean should be $0°$. It may be possible to either add or subtract 360° to see which error is smaller on a per run basis, but the resulting value must be consistent and the true value known beforehand. For example, the program should always result in an answer that is between $[350°, 365°]$ or between $[-10°, 5°]$.

A third option for error representation is implementing $\Delta\omega = \omega_{error} = \omega_{true} - \omega_{ICP}$. The rotation matrix R_{rpy} is calculated using

$$\begin{aligned} roll &= r + \delta r \\ pitch &= p + \delta p \\ yaw &= y + \delta y \end{aligned} \qquad 3.47$$

and using the trigonometric identity

$$\tan(\alpha \pm \beta) = \frac{\tan(\alpha) \pm \tan(\beta)}{1 \mp \tan(\alpha) \cdot \tan(\beta)} \qquad 3.48$$

such that the values of δr, δp and δy may be solved for.

This representation was found to be slightly misleading as in practice; as the error was reported as being high when it should not have been. As a result, the representation that is used is the same as the one used by Simon (1996).

3.4.5.2 Rotational Balancing

Depending on the size of the model, the rotational components may outweigh or be outweighed by the translational components. This is because the rotational components are akin to the moment of force defined by $M = r \times F$, where M is the moment $(N \cdot m)$, r is the radius (m) and F is the force vector (N).

If two identical shapes of different scale are both rotated by $d\alpha$, then a point on the surface of model 1 will move a different amount than a similar point on model 2. Figure 25 shows the effect of scale on rotation on the point displacement, d_1 and d_2. Thus, a smaller shape will undergo smaller point displacements for a similar rotation. This means that the displacement due to rotation is dependent on scale, as changes in r increases or decreases the moment/torque.

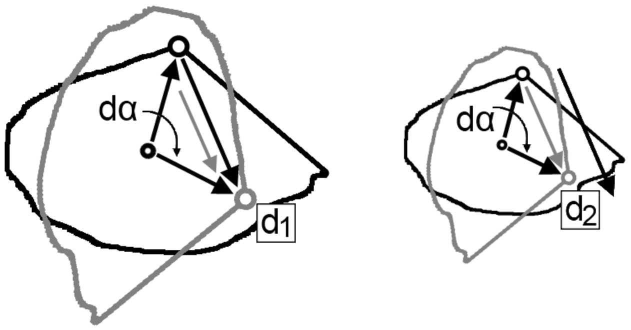


Figure 25 - Rotational imbalance

A similar study is performed for the translational component. If two identical shapes of different scale are both translated by d_T, then a point on the surface of model 1 will move the same amount as a similar point on model 2. Figure 26 shows the effect of scale on translation on the point displacement, d_1 and d_2. Therefore, a smaller shape will undergo the same point displacements for a similar translation as a larger shape. This means that the displacement due to translation is independent of scale.

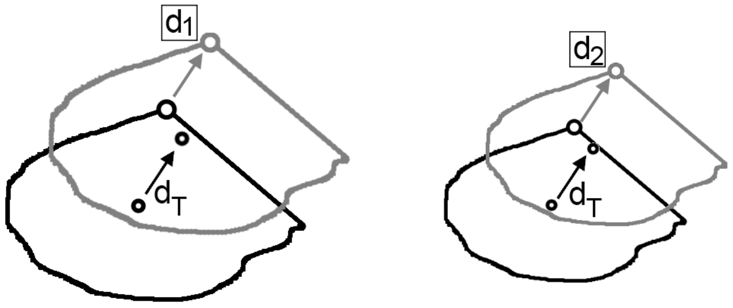


Figure 26 - Translation balance

To combat the rotational imbalance, the CAD model is normalized and scaled so that the average vertex, $\{vertex_i\}$, distance from the centroid is 1. First, the centroid of the CAD model is shifted to the origin $[0,0,0]$, scaling will not work properly if the centroid is not translated. Second, $\{vertex_i\}$ is normalized so that the average distance to the centroid is 1

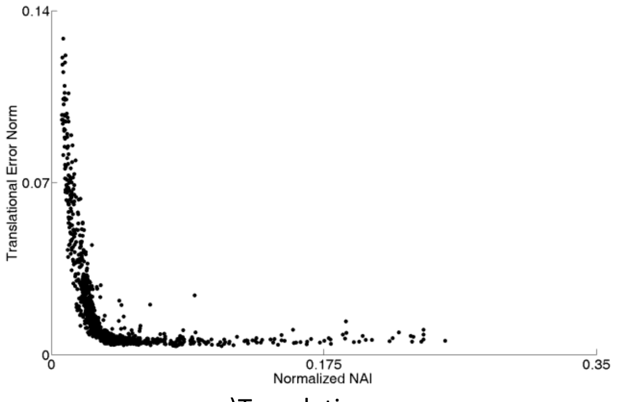
$$scale_{CAD} = \frac{1}{N}\sum_{i=1}^{N} \|vertex_i\| \tag{3.49}$$

The scale factor is then applied to the vertices so that

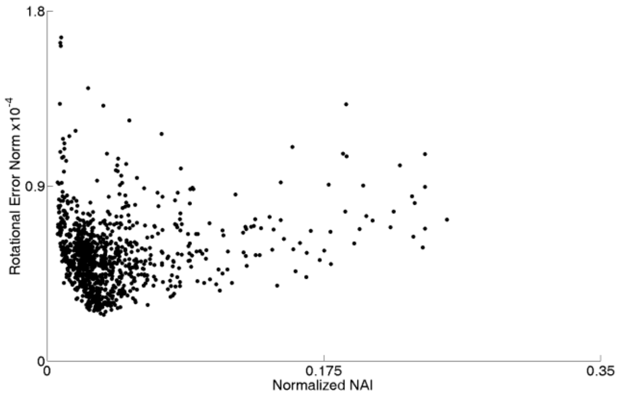
$$\frac{1}{N}\sum_{i=1}^{N} \left\|\frac{vertex_i}{scale_{CAD}}\right\| = 1 \tag{3.50}$$

In this work, the mean distance to the centroid was set so that $\|vertex_i\|$ is between 1.5 and 2. Anything smaller than this can result in a point cloud that is $\|p_i\| < 1$. By applying the scale factor to the vertices, the effects of using different units for the CAD model is negated. CAD models can be modelled in inches or in feet; meters or centimetres; a model with smaller units will most likely be dominated by the rotation. By performing the scaling, the model can be brought into the scanner's reference frame and unit of measurement.

The effect of not scaling a model can be seen in Figure 27. Figure 27 (a) shows the translation error norm values for an unscaled Shuttle model, while Figure 27 (b) shows the rotation error norm values. The rotational components, $\times 10^{-4}$, are several magnitudes smaller than the translational, $\times 10^{-2}$.



a)Translation



b)Rotation

Figure 27 - Translation vs. rotation error levels

A similar scaling factor is built for balancing the pose error. Based on Equation 3.49, the points in the scanner point cloud are normalized so that $\|p_i\| = 1$.

$$scale = \frac{1}{N}\sum_{i=1}^{N}\|p_i\| \quad \textbf{3.51}$$

Equation 3.46 becomes

$$p_{\varepsilon\, scaled} = [t_x \quad t_y \quad t_z \quad scale \times (\omega_x \quad \omega_y \quad \omega_z)] \quad \textbf{3.52}$$

and should balance the magnitude of the translation and rotation errors. This balance allows for a combined value indicating the overall registration error for a specific view by taking the norm of the pose error components:

$$|p_\varepsilon| = \sqrt{t_x^2 + t_y^2 + t_z^2 + scale^2 \times \left(\omega_x^2 + \omega_y^2 + \omega_z^2\right)} \quad \textbf{3.53}$$

3.4.5.3 Constraint Matrix and PCA Measures of ICP performance

While the ICP program gives the translational and rotational error, this is only known after the completion of several trials. To generate statistically valid data, 100 trials are run for each view. Each trial run has a different initial guess and measurement noise. Therefore, a prediction on the translation and rotational errors would be advantageous. One way to indicate the error is through the geometric constraint contained within the point cloud. The derivation of predictive measures and indices methods is a reformulation of the work done by Simon (1996) and by Gelfand & Rusinkiewicz (2003).

There are several different measures that can be used to assess a view for pose estimation and to predict the pose error. The Noise Amplification Index (NAI), which was originally developed for robot dexterity, will be used. Other measures that will also be considered include the Minimum Eigenvalue (ME); Inverse Condition Number (InvCond) and the Expectivity Index (EI), which was developed in McTavish, Okouneva, & Okounev (2009) and studied further in McTavish, Okouneva, & English (2010). These measures are derived from constraint matrices decomposed using PCA. By assembling a constraint matrix to numerically define the geometric constraint, Eigenvalue Decomposition can be performed. ED will yield the strength and direction of the constraint through the eigenvalues, λ_i, and eigenvectors, v_i, recalling that $\lambda_1 > \lambda_i > \lambda_6$. Therefore, the constraint matrix is very similar to the covariance matrix found using the typical PCA methods.

The MSE in Equation 3.7 calculates the Euclidean distances between the scanner point cloud $\{\tilde{p}_i\}$ and the model point cloud $\{m_i\}$, each of which has N points. When the MSE is minimized, the ICP algorithm results in a more accurate pose estimation. This means that it will find accurate translational and rotational values which align the two point cloud sets. Similar to minimizing the MSE, is the notion of minimizing the point-to-plane error metric proposed by Chen & Medioni (1992). The point-to-plane error metric calculates the distances between a point $\{\tilde{p}_i\}$ and the tangent plane defined by $\{m_i\}$ and $\{n_i\}$. For this error metric, $E_\perp$, each model point in $\{m_i\}$ requires its own normal $\{n_i\}$.

$$E_\perp = \sum_{i=1}^{N} ([R_\perp p_i + t_\perp - m_i] \cdot n_i)^2 \qquad \textbf{3.54}$$

where $\begin{cases} R_\perp \text{ is the point-to-plane Rotation matrix minimizing } E_\perp \\ t_\perp \text{ is the point-to-plane translation vector minimizing } E_\perp \\ n_i \text{ is are the normal vectors associated with } m_i \\ \perp \text{ denotes point-to-plane} \end{cases}$

This is represented in Figure 28, which is a simplification of Chen & Medioni (1992)

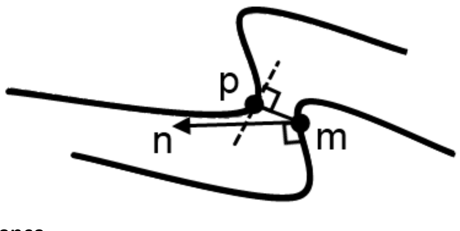


Figure 28 - Point to plane correspondence

In the final iterations of the ICP algorithm, the rotational increments are very small. At this point the rotation matrix, defined by Euler angle rotations about the primary axes $\omega = [\omega_x \quad \omega_y \quad \omega_z]$, can be reduced to

$$R_\perp \cong \begin{bmatrix} 1 & -\omega_x & \omega_y \\ \omega_x & 1 & -\omega_z \\ -\omega_y & \omega_z & 1 \end{bmatrix} = [1 + \omega^x] \qquad 3.55$$

Expanding the terms of Equation 3.54 and substituting Equation 3.55,

$$\begin{aligned} E_\perp &= \sum_{i=1}^{N} (R_\perp p_i \cdot n_i + t_\perp \cdot n_i - m_i \cdot n_i)^2 \\ &= \sum_{i=1}^{N} ([1 + \omega^x] p_i n_i + t_\perp n_i - m_i n_i)^2 \\ &= \sum_{i=1}^{N} (p_i n_i + \omega^x p_i n_i + t_\perp n_i - m_i n_i)^2 \end{aligned} \qquad 3.56$$

Equation 3.56 can be simplified further by combining the terms together, and applying cross product and skew-symmetric matrix properties:

$$\begin{aligned} a^x b \cdot c &= a \times b \cdot c \\ a \times b \cdot c &= a \cdot b \times c \end{aligned} \qquad 3.57$$

which results in

$$E_\perp = \sum_{i=1}^{N} \left((p_i - m_i) \cdot n_i + t_\perp \cdot n_i + \omega \cdot p_i \times n_i \right)^2 \qquad 3.58$$

To align the point sets, the translation and rotation required to minimize Equation 3.58, parameterized by the vector ζ as

$$\zeta = [t_x \quad t_y \quad t_z \quad \omega_x \quad \omega_y \quad \omega_z] \qquad 3.59$$

are found by equating the partial derivatives of Equation 3.58 with respect to ζ as zero

$$\frac{\partial}{\partial \zeta} E_\perp = \begin{bmatrix} \frac{\partial D}{\partial \mathrm{t_x}} & \frac{\partial D}{\partial \mathrm{t_y}} & \frac{\partial D}{\partial \mathrm{t_z}} & \frac{\partial D}{\partial \omega_\mathrm{x}} & \frac{\partial D}{\partial \omega_\mathrm{y}} & \frac{\partial D}{\partial \omega_\mathrm{z}} \end{bmatrix} = 0 \qquad 3.60$$

The result of Equation 3.60 is a 6×6 linear system of the form

$$J \cdot J^T x = J \cdot f \tag{3.61}$$

where J is a $6 \times N$ jacobian,

$$\begin{aligned} J &= \begin{bmatrix} n_1 & \dots & n_N \\ p_N \times n_1 & \dots & p_N \times n_N \end{bmatrix} \\ &= \begin{bmatrix} n_N & \dots & n_N \\ p_N{}^x n_N & \dots & p_N{}^x n_N \end{bmatrix} \end{aligned} \tag{3.62}$$

and Jf is the 6-element vector representing the residuals. The constraint matrix JJ^T can be considered a sensitivity matrix for small displacements. The Equation 3.61 represents the system of normal equations $J^T x = f$. The solution of Equation 3.61 will minimize the norm $||J^T x = f||$, or the norm $||J^T x||$, as f is small near the end of the alignment. The vector $J^T x$ represents the total amount by which the point-to-plane distances will change if a small differential transformation ζ is applied.

Strang (1976) showed that the relation between f and ζ is bounded by

$$\sigma_6 \leq \frac{||f\,||}{||\zeta||} \leq \sigma_1 \tag{3.63}$$

where $\sigma_1 \geq \ \dots \geq \sigma_6$ are the ordered singular values of J^T. Therefore when undergoing a small transformation ζ, the residuals f are bounded by

$$||f\,|| \geq \sigma_6 ||\zeta|| \tag{3.64}$$

In addition, Strang also relation between the bound of a variable's relative error to the relative change. The small transformation ζ is expected to have both noise and error associated to it through $\delta\zeta$. Additionally, the residual error is also expected to have uncertainty associated to it through δf. δf represents the noise and error from sensors and other sources. Thus Strang gives the inequality

$$\frac{||\delta\zeta||}{||\zeta||} \leq \frac{\sigma_1}{\sigma_6} \frac{||\delta f||}{||f||} \tag{3.65}$$

The implication of combining inequalities 3.64 and 3.65 is

$$\frac{||\delta\zeta||}{||\zeta||} \leq \frac{\sigma_1}{\sigma_6} \frac{1}{\sigma_6} \frac{||\delta f||}{||\zeta||} \tag{3.66}$$

which is simplified to

$$||\delta\zeta|| \leq \frac{\sigma_1}{\sigma_6^2}||\delta f|| \qquad \textbf{3.67}$$

The inequality of Equation 3.67 states that in order to minimize the pose error norm $||\delta\zeta||$, the ratio σ_1/σ_6^2 must be minimized. Alternatively, σ_6^2/σ_1 can be maximized instead. Nahvi & Hollerbach (1996) called this ratio the *Noise Amplification Index* (NAI) and defined it as

$$||\delta\zeta|| \leq \frac{1}{NAI}||\delta \text{f}|| \text{ and } NAI = K_{NAI} = \frac{\sigma_6^2}{\sigma_1} \qquad \textbf{3.68}$$

Similar to 3.68, the ratio σ_1/σ_6 from Equation 3.65 is called the matrix conditioning number. In order to minimize the relative pose error, the condition number must also be minimize. Conversely, to create the same form as 3.68, the *inverse conditioning number* (InvCond), which must be maximized to minimize relative pose error, is defined as

$$\frac{||\delta\zeta||}{||\zeta||} \leq \frac{1}{InvCond}\frac{||\delta\text{f}||}{||\text{f}||} \quad and\ K_{InvCond} = \frac{\sigma_6}{\sigma_1} \qquad \textbf{3.69}$$

Finally, looking back at inequality 3.64, the value of the residuals f will be greatest felt when the magnification of ζ by σ_6 is maximized. This means that with a larger value of σ_6 the system constraint will increase, and the error function will be affected to a greater degree. A larger error function value will cause the ICP algorithm to take more notice to ζ. Thus having higher levels of σ_6 will result in a better pose estimation. Remembering that $\sigma_6 = \sqrt{\lambda_6}$, the *Minimum Eigenvector* (ME) index is defined as $K_{\lambda min} = \sqrt{\lambda_6}$.

Equation 3.62 can also be reformulated on a per point basis. From McTavish, Okouneva, & English (2010), each point is represented as

$$\begin{aligned} J_i J_i^T &= \begin{bmatrix} n_i \\ p_i{}^x n_i \end{bmatrix} [n_i{}^T \quad (p_i{}^x n_i)^T] \\ &= \begin{bmatrix} n_i n_i{}^T & -n_i n_i{}^T p_i{}^x \\ p_i{}^x n_i n_i{}^T & p_i^x n_i n_i{}^T \cdot p_i{}^x \end{bmatrix} \end{aligned} \qquad \textbf{3.70}$$

Note that the skew symmetric matrix gives $(p_i{}^x n_i)^T = -n_i{}^T p_i{}^x$.

To calculate the entire constraint matrix, Equation 3.70 is computed as a running total which reduces the need to store Equation 3.62 as a very large $6 \times N$ matrix.

$$JJ^T = \sum_{i=1}^{N} J_i J_i^T \tag{3.71}$$

If the squared norm $\|f\|^2$ is the misalignment error $E_\perp$, then it can be represented as

$$E_\perp = \|f\|^2 = \zeta^T J J^T \zeta \tag{3.72}$$

The geometric constraint analysis of the constraint matrix JJ^T is used to reflect the constraint or unconstraint directions in the current view of CAD model. Using PCA to perform ED on JJ^T results in

$$\mathrm{JJ^T} = V\Lambda V^T = [v_1\ v_2\ v_3\ v_4\ v_5\ v_6] \begin{bmatrix} \lambda_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda_6 \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \\ v_4^T \\ v_5^T \\ v_6^T \end{bmatrix} \tag{3.73}$$

where $\begin{cases} V \text{ is a matrix of eigenvectors, each column is an eigenvector, } v_i \\ \Lambda \text{ is a matrix of eigenvalues, } diag(\lambda_i) \end{cases}$

This means that the misalignment error is also represented as

$$E_\perp = \|f\|^2 = \zeta^T V\Lambda V^T \zeta = \sum_{i=1}^{N} \lambda_i (\zeta^T v_i)^2 \tag{3.74}$$

The singular values of σ_i^J and $\sigma_i^{J^T}$ from J and J^T, respectively, correspond to the eigenvalues of JJ^T as $\sigma_i^J = \sigma_i^{J^T} = \sqrt{\lambda_i} = \sqrt{\sigma_i^{JJ^T}}$.

The eigenvectors represent the amount of rotational and translational constraint of the point cloud. The first three components of the eigenvector, v_i^{1-3}, represent the translation components, while the last three eigenvector components v_i^{4-6} represent the rotational components.

The surface geometry and complexity determines the amount of geometric constraint in the point cloud. The geometric constraint plays an important role in the convergence of the ICP to the global solution. For example, two planes can slide with respect to each other without ever changing the error in Equation 3.58. Similarly, two spheres can rotate with respect to each other for the same reason.

Motion where rotation and/or translation does not change the error metric is considered to be unconstrained. Alternatively, motion where rotation and/or translation vastly changes the error metric is considered to be constrained.

To find the constrained or unconstrained motion, the eigenvectors v_i and eigenvalues λ_i are used. Eigenvector v_1, corresponding to the largest eigenvalue λ_1, represents the direction in which the point cloud $\{p_i\}$ is most constrained. This is called the "transformation of maximum constraint". If $\{p_i\}$ is transformed by a ζ that is parallel to v_1, $E_\perp$ will experience a large change, thus indicating that this disturbance is incorrect and costly.

Conversely, eigenvector v_6, corresponding to the smallest eigenvalue λ_6, represents the direction in which the point cloud $\{p_i\}$ is the least constrained. If $\{p_i\}$ is transformed by a ζ parallel to q_6, $E_\perp$ will experience a small or negligible change, thus allowing this transformation to continue and incorrectly influence the rotation and translation values. It is possible to have values of eigenvalue λ_6 either zero or negligible, in this case rotating or translating around v_6 will have no discernable impact on $E_\perp$. Ideally, this scenario should be avoided as it leads to large amounts of uncertainty in the system. Referring back to Section 2.1 and the definition of PCA, this means that there is a loss of a degree of freedom.

For the examples below, the constraint matrices were assembled by taking points from all sides of the model in sphere, cube and diamond examples. That eigenvalues $\lambda_{4\to6}$ for the sphere in Figure 29 are very small, theoretically they should be zero. Small $\lambda_{4\to6}$ demonstrates that rotation is unconstrained. Because the diamond is nothing more than a rotated cube, the overall constraint between the two objects is similar.

Only one side of the model was scanned in the flat plate examples to show the ability of plates to slide. Scanning both sides would not change the overall constraint. By examining the eigenvalues of the flat plate in Figure 32, the zero values apparent in the X and Y translation and the Z rotation are a direct result of the lack of constraint. However, by rotating the plate, the translational and rotational components are convoluted and coupled together. Figure 33 and Figure 34 have more geometric constraint as more DOFs are affected by v_2^{4-6} and v_3^{4-6}.

λ_i	t_x	t_y	t_z	ω_x	ω_y	ω_z
27.78	0.00	-0.11	-0.99	0.11	0.00	0.00
24.74	0.98	0.01	0.00	0.00	-0.13	0.12
21.00	0.01	-0.98	0.10	-0.14	0.00	0.00
0.13	-0.14	0.00	0.01	0.03	-0.99	0.07
0.12	-0.01	0.13	-0.12	-0.98	-0.02	0.08
0.11	0.11	0.01	-0.01	-0.08	-0.09	-0.99

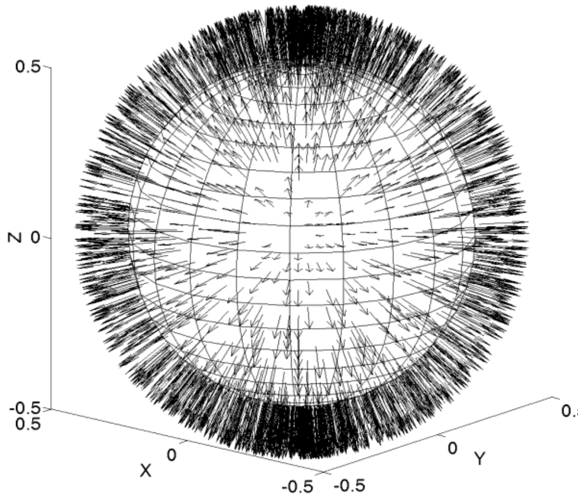


Figure 29 - Sphere

λ_i	t_x	t_y	t_z	ω_x	ω_y	ω_z
8.98	0.80	0.55	-0.23	0.00	0.00	0.00
8.98	-0.27	0.00	-0.96	0.00	0.00	0.00
8.98	-0.53	0.83	0.15	0.00	0.00	0.00
4.07	0.00	0.00	0.00	0.00	0.00	1.00
4.07	0.00	0.00	0.00	0.33	-0.94	0.00
4.07	0.00	0.00	0.00	-0.94	-0.33	0.00

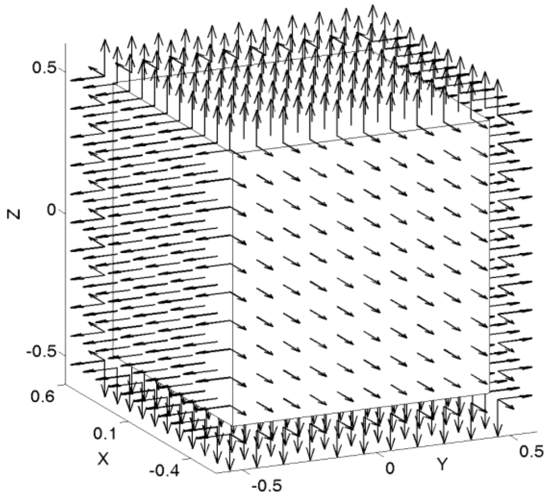


Figure 30 - Cube

λ_i	t_x	t_y	t_z	ω_x	ω_y	ω_z
8.98	-0.43	0.88	-0.20	0.00	0.00	0.00
8.98	0.70	0.19	-0.69	0.00	0.00	0.00
8.98	0.57	0.44	0.69	0.00	0.00	0.00
4.07	0.00	0.00	0.00	0.98	0.17	-0.04
4.07	0.00	0.00	0.00	0.07	-0.14	0.99
4.07	0.00	0.00	0.00	-0.16	0.98	0.15

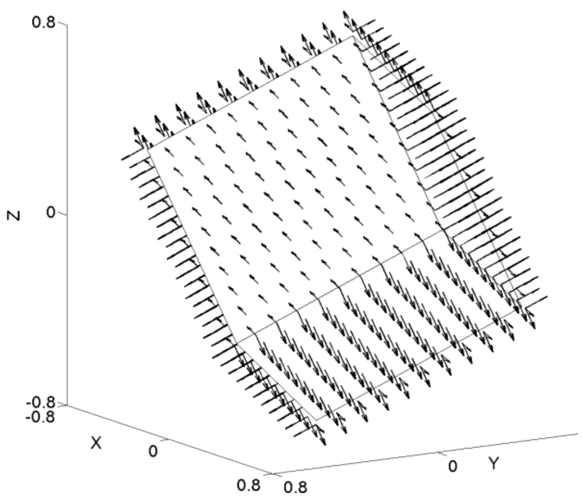


Figure 31 - Diamond

λ_i	t_x	t_y	t_z	ω_x	ω_y	ω_z
11.00	0.00	0.00	1.00	0.00	0.00	0.00
6.26	0.00	0.00	0.00	0.00	1.00	0.00
6.26	0.00	0.00	0.00	-1.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	1.00
0.00	0.00	1.00	0.00	0.00	0.00	0.00
0.00	1.00	0.00	0.00	0.00	0.00	0.00

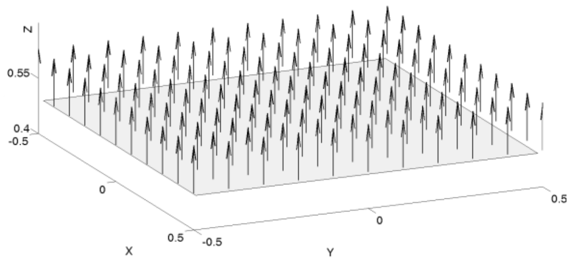


Figure 32 - Flat Plate

λ_i	t_x	t_y	t_z	ω_x	ω_y	ω_z
11.00	0.00	-0.71	-0.71	0.00	0.00	0.00
6.26	0.00	0.00	0.00	0.00	0.71	-0.71
6.26	0.00	0.00	0.00	-1.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	-0.71	-0.71
0.00	1.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	-0.71	0.71	0.00	0.00	0.00

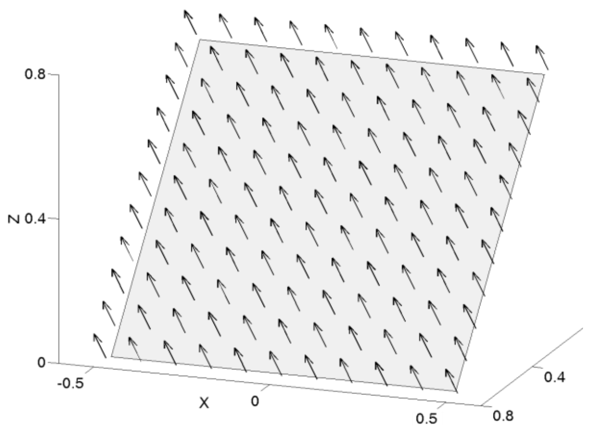


Figure 33 - Plate rotated Yaw = 45°

λ_i	t_x	t_y	t_z	ω_x	ω_y	ω_z
11.00	-0.34	0.66	0.66	0.00	0.00	0.00
6.26	0.00	0.00	0.00	-0.87	-0.49	0.04
6.26	0.00	0.00	0.00	0.36	-0.56	0.75
0.00	0.00	0.10	-0.11	0.34	-0.66	-0.66
0.00	-0.03	0.69	-0.71	-0.05	0.10	0.10
0.00	0.94	0.26	0.22	0.00	0.00	0.00

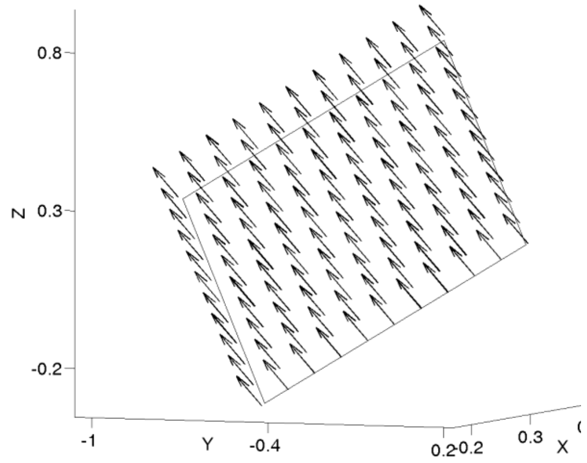


Figure 34 - Plate rotated Yaw = 45° Pitch = 20°

Recently in McTavish, Okouneva, & English (2010), a new geometric constraint analysis index was introduced as the *Expectivity Index* (EI)

$$Expectivity = K_{Exp} = \left(\sqrt{\sum \frac{1}{\lambda_i}}\right)^{-1} \qquad \textbf{3.75}$$

It was shown that the standard deviation of the pose error is related to the EI as

$$\sqrt{\mathcal{E}(\| p_\varepsilon \|^2)} = \sigma_\varepsilon \cdot \sqrt{\sum \frac{1}{\lambda_k}} \qquad \textbf{3.76}$$

where $\begin{cases} \sigma_\varepsilon \text{ is the standard deviation of the measurement noise} \\ p_\varepsilon \text{ is the misalignment error} \end{cases}$

Of the four constraint measures, only the inverse conditioning number is dimensionless, making it more suitable for comparisons as it is independent of the number of points N in $\{p_i\}$. $E_\perp$ and all other indicies will grow as the number of points in $\{p_i\}$ rises; this can potentially alter the results between a high or low density scan on the same shape and view. Therefore, each dimensional measure above should be point normalized, as the Expectivity, NAI and Minimum Eigenvalue indicies are of dimension $\sqrt{\lambda}$, they are divided by $\sqrt{N}$. The error metric $E_\perp$ is normalized by $1/N$. The point normalization can help in the assessment of differing views of the same object or differing objects altogether:

$$\begin{gathered} \sqrt{\mathcal{E}(\|p\|^2)} = \frac{1}{K_{Exp}/\sqrt{N}} \sigma_\varepsilon \\ \|p\| \leq \frac{1}{K_{\lambda\, min}/\sqrt{N}} \|\varepsilon\| \\ \|p\| \leq \frac{1}{K_{NAI}/\sqrt{N}} \|\varepsilon\| \end{gathered} \qquad \textbf{3.77}$$

The eigenvectors and eigenvectors of JJ^T can also be used to represent an error hyperellipsoid. To establish whether or not a view and the point cloud will perform well under the ICP algorithm, the JJ^T constraint matrix is used to generate constraint measures. Consider that the general equation of an ellipsoid is

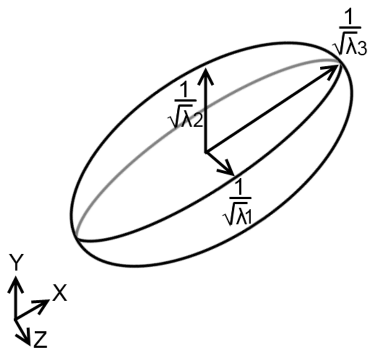
$$x^T A x = 1 \qquad \textbf{3.78}$$

where A is a symmetric, positive definite matrix.

Then Equation 3.78 can represent a hyperellipsoid of any dimension and is used in conjunction with $\mathrm{JJ^T}$ to define

$$x^T \mathrm{JJ^T x} = 1 \qquad \textbf{3.79}$$

Equation 3.79 describes an error ellipsoid with principal axes defined by the eigenvectors, v_i. The length of the primary axes is defined by the square root of the inverse of the eigenvectors, $1/\sqrt{\lambda_i}$. This geometry can be seen in Figure 35, which represents the 3D scenario. The ellipsoid may be referred to as the uncertainty or error hyperellipsoid, a greater primary axis will result in larger registration errors. Smaller primary axis lengths will occur when the eigenvalues λ_i are larger, this is typical a well constrained shape.



$\frac{1}{\sqrt{\lambda 3}}$
$\frac{1}{\sqrt{\lambda 2}}$
$\frac{1}{\sqrt{\lambda 1}}$
Y
X
Z

Figure 35 - 3D - Ellipsoid

The main properties of the hyperellipsoid that require addressing to decrease the uncertainty are the

1. hyperellipsoid volume
2. hyperellipsoid eccentricity
3. length of primary axes

However, property 1 and 2 are tied directly to the length of the largest primary axis, $1/\sqrt{\lambda_6}$. A smaller volume should appear when the hyperellipsoid approaches a spheroid shape. The eccentricity of the hyperellipsoid will diminish as the six primary axes become smaller. A well conditioned hyperellipsoid will be spheroidal as it should constrain all degrees of freedom equally and effectively.

The ME, NAI and InvCond indices are composed of only the largest and smallest eigenvalues; as the result, they only give an upper boundary of the pose error norm. The Expectivity Index, is composed of all eigenvalues, and by its definition should give an exact boundary on the standard deviation of the pose norm.

3.4.5.4 Rotational Balancing for the Constraint Matrix

Similar to how the CAD model is scaled as the means to balance the rotational effects in Section 3.4.5.2. The point cloud $\{p_i\}$ which is used to generate JJ^{T} from Equation 3.70 must also be scaled and adjusted. The rotational components again have the potential to dominate the constraint matrix. Looking at the components of $\mathrm{J_i J_i^T}$ in Equation 3.70, when the points are located further out, the rotational constraint is reported as being large. Therefore, the further $\{p_i\}$ is translated, then the larger moment/torque will develop. As seen in Figure 36, if the point cloud is not at the origin, then $r_2 \times \hat{n} \gg r_1 \times \hat{n}$. Therefore $\{p_i\}$ is adjusted so that its centroid lies at $(0,0,0)$. For the same reasons outlined in Section 3.4.5.2, $\{p_i\}$ is scaled so that $|p_i| = 1$ using Equation 3.51.

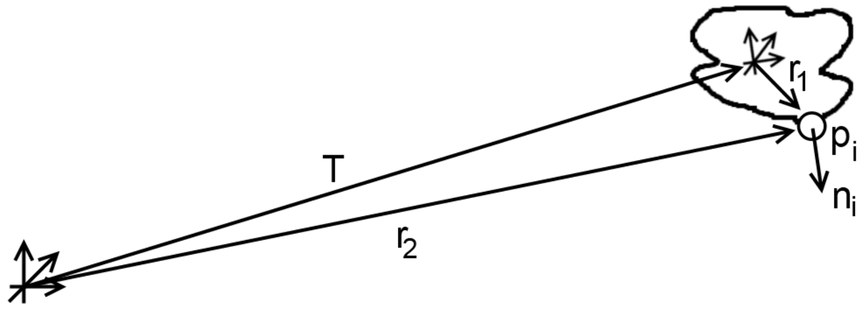


Figure 36 - Translation balance

3.4.5.5 Point Cloud Normals

Each point in the point cloud $\{p_i\}$ has an associate normal vector, n_i, that has been calculated by the scanner while ray tracing the CAD model. Because the normals are calculated from the model, they are ideal and without any added noise. However, there are no associated normals for any point of a real scan, which poses a problem as normals are required to calculate the constraint matrix, see Equation 3.70. Normals are not involved in the registration process as the ICP algorithm does not require them. To generate normals there are several different options will be briefly discussed:

1. The true normal generated by scanner program
2. Adding noise to the meshed model and recalculating each point and normal
3. Corrupting the true normals
4. Reconstructing from the point cloud

True surface normals are generated by the scanner module. This can be done either as a preprocessing step to calculate all surface normals at once to be made available for use; or, on demand once a ray has calculated an intersection with the model surface. The normal is computed by using the right-hand rule in conjunction with the face set for the triangular mesh elements.

When Gaussian, zero-mean noise is added the LIDAR point cloud during ICP registration, it may be possible to also morph the surface of the CAD model by altering the vertex positions. If the triangular mesh element is warped based on the Gaussian noise added to the intersection points, then the resulting normals change as well. This method is likely the most difficult way of generating normals.

The corruption of the true normals can be achieved by applying a process similar to that of the initial pose generation. Each normal can be rotated slightly about an arbitrary axis so that it still points within a degree or two of the true normal or so that the dot product between the two normals is ≈ 1. Over a large planar surface, the normals should all point in the same direction, however with this random process, it is likely that the normals will not be parallel.

The final method for normal generation is to reconstructed the normals from the point cloud itself. This requires no other data available, such as the CAD model, and is the method used to process real scan data. Hoppe, DeRose, Duchamp, McDonald, & Stuetzle (1992) describe a way to use fundamentals of PCA for normal calculation. Small portions a point cloud are fitted to temporary planes to generate the normal for a point. By using k number of neighbouring points from the point cloud, a covariance matrix is built using

$$C = \sum_{\mathrm{i}=1}^{\mathrm{i}=\mathrm{n}} (\mathrm{p_i} - \bar{\mathrm{p}})(\mathrm{p_i} - \bar{\mathrm{p}}) \qquad \textbf{3.80}$$

which is similar to Equation 2.2, but is not normalized. After using ED, the eigenvector associated with the smallest eigenvalue will be the normal vector for that point. In the example shown in Section 2.1, the smallest eigenvector is $\approx [0,0,1]$, which is out of the dominant X-Y plane. To make sure that a normal calculated using the PCA method is oriented in the correct direction, the dot product of the reconstructed normal and a vector from the intersection point to the camera origin is used. As previously mentioned, the sign of eigenvectors from ED may be flipped. Using the Approximate Nearest Neighbour library, k neighbours were selected to form the temporary plane, and experimentally $k = 5$ gave reasonable results. Figure 37 and Figure 38 show the true and reconstructed normal vectors from simulated data.

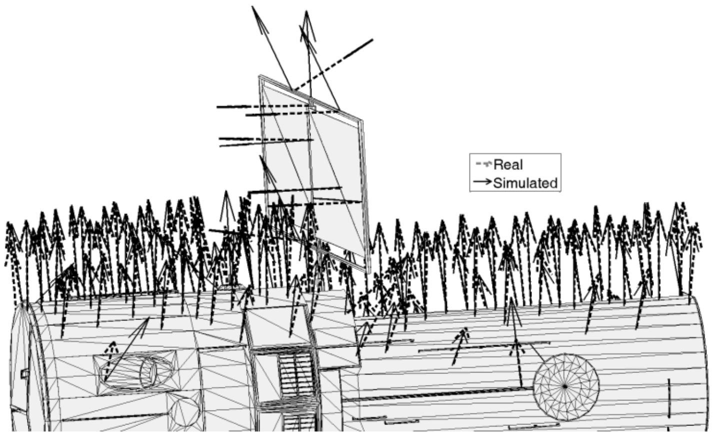


Figure 37 - Normal reconstruction: Hubble Telescope (zoomed)

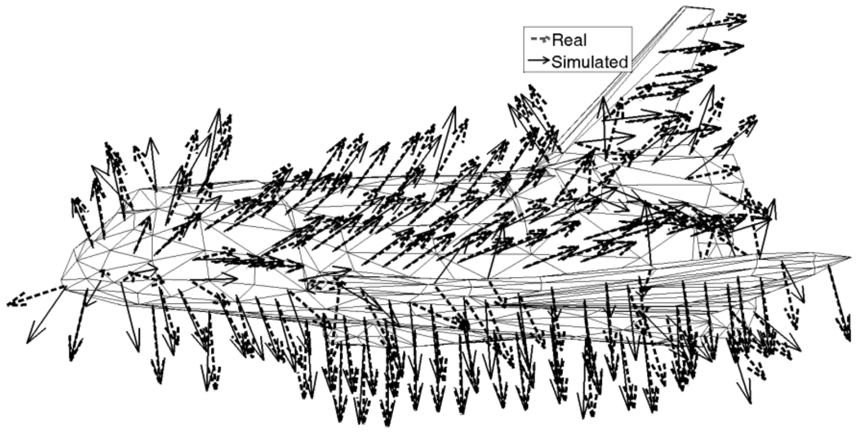


Figure 38 - Normal reconstruction: Space Shuttle

To establish some sort of qualitative measure, the dot product was used. The dot product for each normal vector was taken and resulted in an average of [0.9454, 0.9340], for the Space Shuttle and the Hubble Telescope, respectively. For this test, views that predominately featured rounded surfaces were scanned; planar and flat surfaces will give much better results as the neighbouring normal vectors will generally be parallel. A comparison of the geometric constraint measures calculated using the true and reconstructed normals are shown in Table 3.

Table 3 - Normal reconstruction results

	Hubble	Space Shuttle
View Number	33	1
Num Points	365	497
Average Dot product	0.9340	0.9454
Reconstructed Normalized NAI	0.0096	0.0435
True Normalized NAI	0.0324	0.0529
Reconstructed Normalized Expectivity	0.0838	0.1429
True Normalized Expectivity	0.1202	0.1513

While the reconstructed constraint measures for the Shuttle model are fairly good, this is not typically the case. The disparity between the true and reconstructed measures seen in the Hubble results is more common. This can be attributed to the rounded surfaces of the Hubble telescope, especially on the outer edges of the model where the query point is no longer in the center of a plane fitted by the PCA algorithm. This view of the Shuttle also featured the bottom hull where there is low curvature. Typically, because the Expectivity Index uses all six eigenvalues, the results between the true and reconstructed data are much closer than NAI.

4 Results and Application of PCA Measures for Prediction

4.1 Individual Degrees of Freedom

While the pose error norm from Equation 3.53 is an efficient way of presenting the pose error, it is possible that one DOF will be either very well or very poorly constrained compared to the others; this is lost in Equation 3.53. To show that geometric constraint is a valid tool for selecting the view of an object, the results of the individual DOF results for the Space Shuttle are shown in Figure 39. The units of translation and rotation errors are metres and radians, respectively.

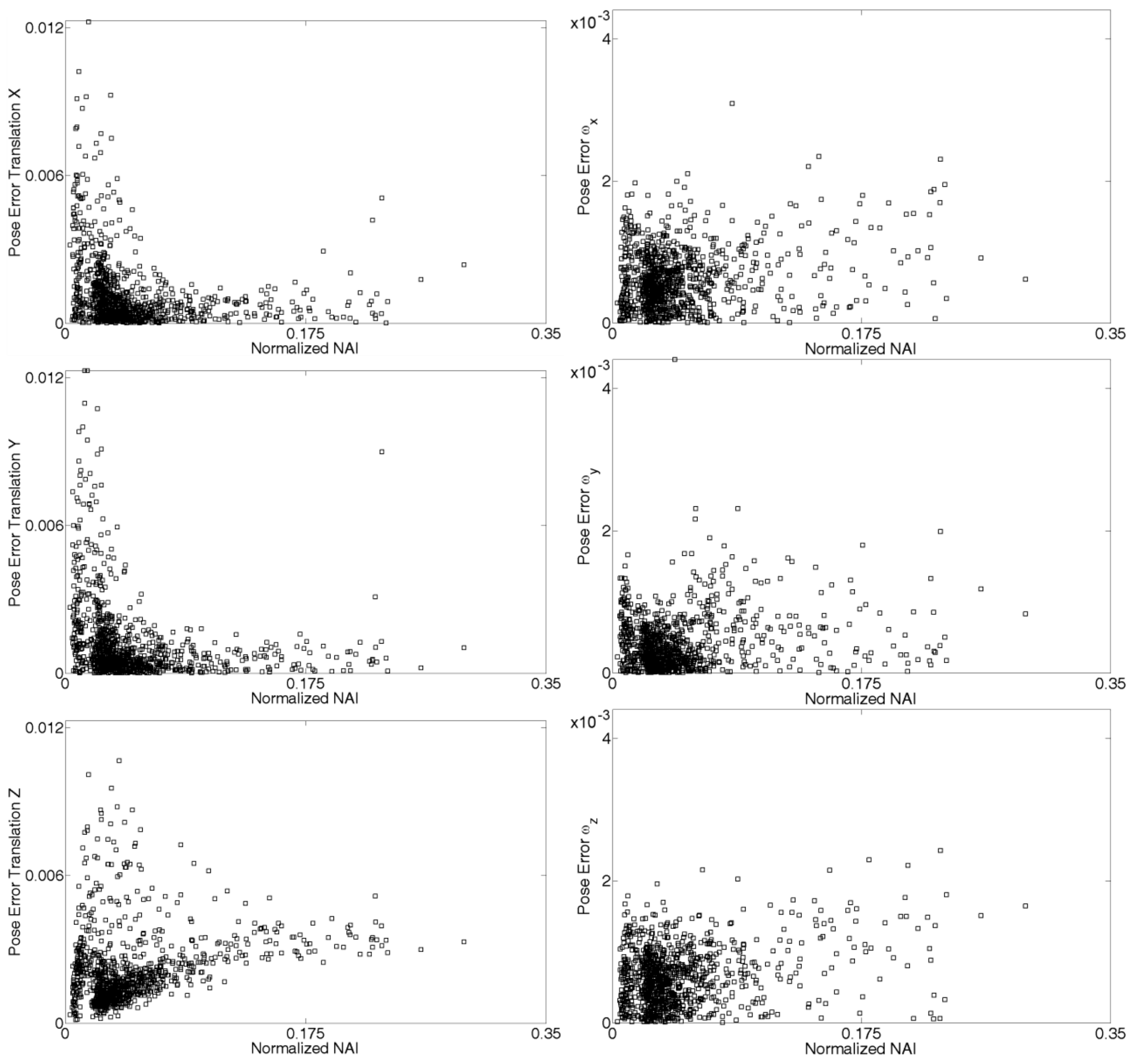


Figure 39 - Individual DOF for Shuttle

As Figure 39 shows, the Shuttle model shows good results with the translation DOFs. However, the rotation DOFs stay level; while it is expected that the error diminishes, it is also expected that the error does not increase. Figure 40 shows a similar study performed with the Hubble Telescope model, which is more in line with the expected results.

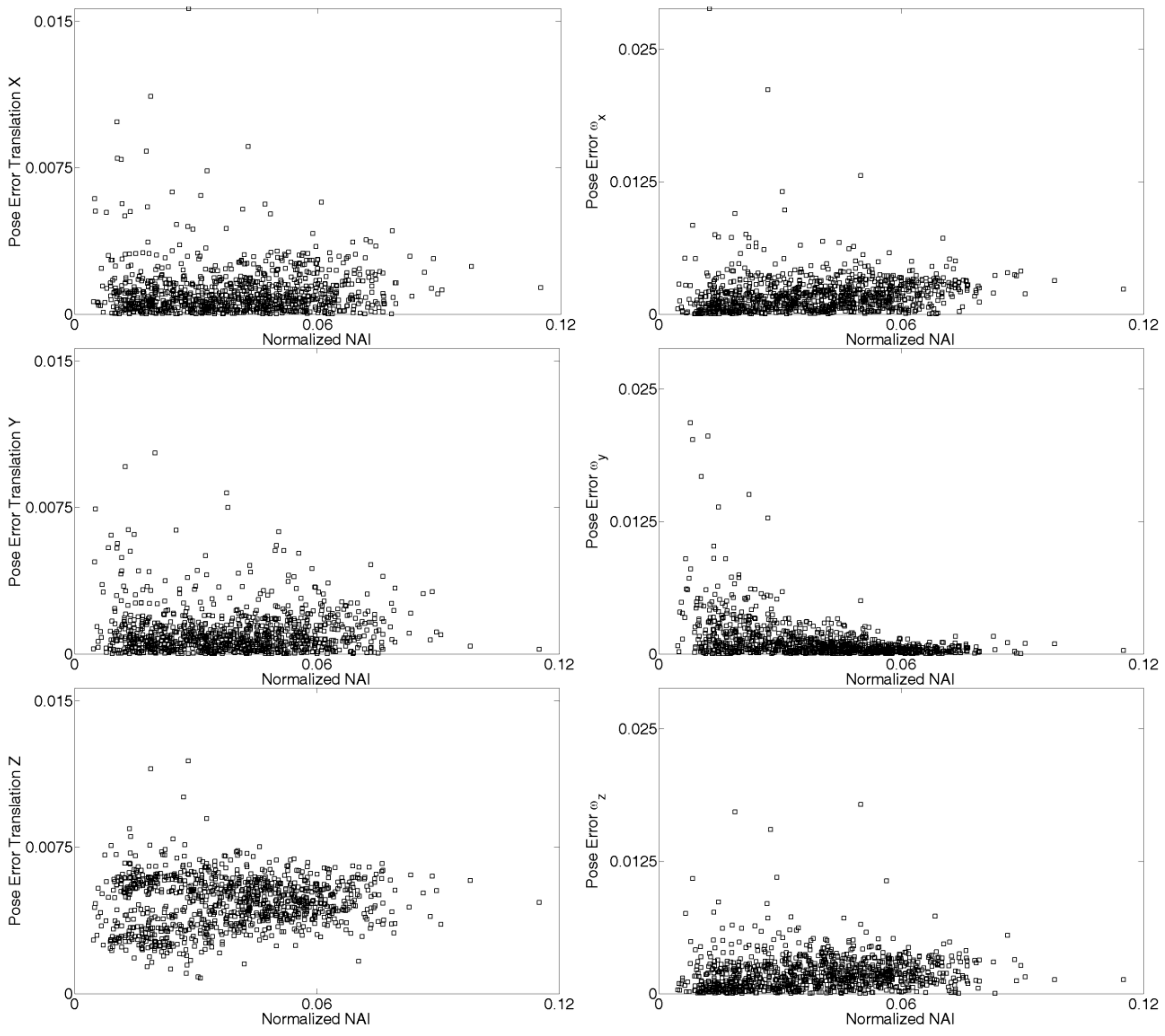


Figure 40 - Individual DOFs for Hubble Telescope

Given the similarities in formulation between the InvCond and NAI indices, both measures can be used to select views with accurate poses. Any rise in geometric constraint is reflected in both measures, as they are tied directly with the maximum and minimum eigenvalues. In simulated data, both measures perform similarly, as seen in Figure 41. Therefore, either index could be used to analyze the individual DOFs as well.

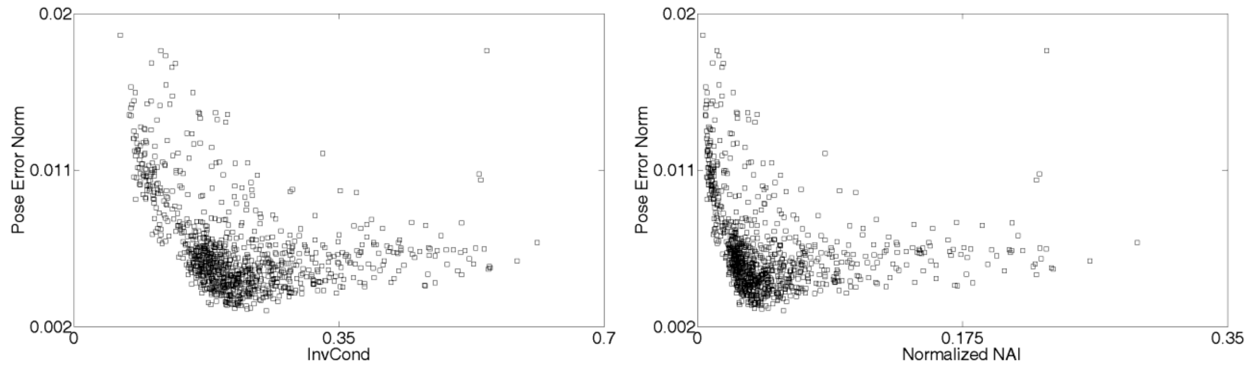


Figure 41 - InvCond and normalized NAI for Shuttle

4.2 Edge Constraint

This work primarily investigates the relation between surface geometry and the accuracy of ICP pose estimation. However, there is may be another factor that comes into effect especially in the presence of weak geometric constraint(s). The edges on the surface of a model can help to constrain points in the point cloud and consequently improve the pose estimation. This is briefly mentioned in McTavish, Okouneva, & English (2010) where it was dubbed *edge constraint*. The edges help to constraint the point cloud by preventing points from moving off of the model. Once a few points begin to move off and away of the model surface, the cost in the ICP algorithm begins to climb due to the MSE and the surface to point distance.

A large portion of the edge constraint is likely contributed from the outline of the model. When the point cloud begins to move away from the ideal registration pose, some points will fall off or move away from the surface of the model and add significantly to the ICP cost metric. If the point cloud continues to move in an erroneous direction, then the number of points that have fallen off the model will increase, as would the MSE; see Figure 42. Effectively, the edge constraint stop points from moving off the surface of a model due to the error cost.

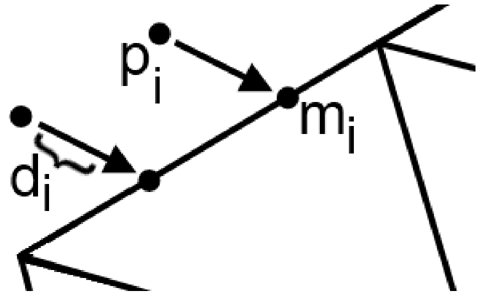


Figure 42 - Points off model surface

The level of edge constraint is dictated by the distance between a point and the outline of a model. If the spread of a point cloud is maximized in one direction, then it is likely that the point cloud will have several points that lie close to the model's outline. A point cloud with edge constraint should result in low pose error and low standard deviation. For example, if two points separated by a fixed distance, are placed on a line, the points will have less room to oscillate if they are placed closer to the edges of that line. This can be seen in Figure 43, if the point cloud is moved against the outline of a model triangle, then it will most likely not move off the surface by continuing in that direction. If the point cloud continues to move, the error cost would be high and would result in Figure 42. Therefore, this edge constrains the movement of the point cloud as the points can only slide along the edge, or away from it. However, if the points were spread further apart, then the number of constraining edges increase and there is less room to manoeuvre.

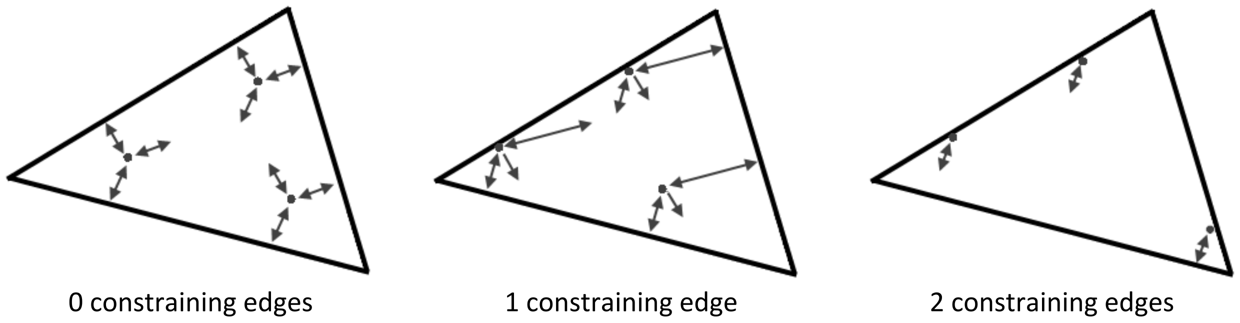


Figure 43 - Edge constraint

The edge constraint is also applicable on interior edges that are not part of the outline; Figure 43 could represent either the outline of a pyramid or a triangular mesh element. There may be many small interior edges available on a highly geometrically constrained shape. Combining the geometric and edge constraint effects can possibly lead to a much lower error.

It is expected that at low geometric constraint, there should be high error. However, this is not always the case; a low geometric constraint view may have a large range of pose error. This may partially be explained by the presence of the edge constraint. Therefore, it is expected that if no edge points or edge constraint is available, then the results of ICP registration over 1000 poses should hold to the generalization of high error-low geometric constraint and low error-high geometric constraint. Ideally, there should be no low error-low geometric constraint regions as only geometric constraints are being considered.

The ability of a point cloud to move around on the model surface before being constrained is called sliding. Maximizing the point cloud spread of the model surface is one way to limit the amount of sliding between two surfaces.

To investigate the effects of the edge constraint; a model can be registered using:

1. All points
2. Edge points
3. Inner points

The edge set contains the points that are very close to the model outline and to the interior edges of the model. The inner set contains the points that reside near the center of a model surface, away from any outline or interior edges. To get the sets of edge points and inner points, it is possible to split up a point cloud as they are mutually exclusive sets. The number of edge points can become very low if the original scan is sparse. Therefore, it may be advantageous to use a dense scan to extract the edge set and a less dense scan for the both the regular point cloud and for the inner set. Gelfand & Rusinkiewicz (2003) perform a similar study of the effect of point placement, but instead used a selection scheme based on sampling from the normal-space to create a more rounded hyperellipsoid.

4.2.1 Inner and Edge points

A simple asymmetric shape was designed to see how the either the presence or absence of edges will impact the ICP registration accuracy. A skewed pyramid is studied as it offers multiple edges and planar surfaces for testing; see Figure 44.

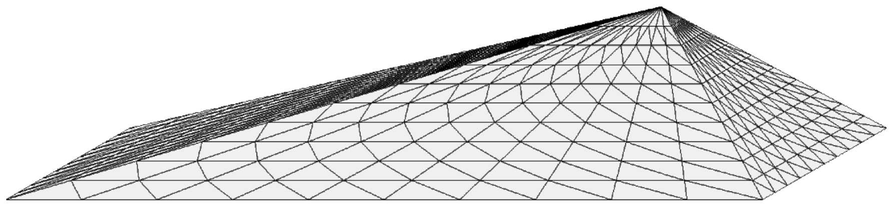


Figure 44 - Pyramid model

The various different types of points are shown in Figure 45. The outline points are those which form the silhouette of an object. Interior edge points are found when the surface changes direction dramatically. The edge point set will comprise both the outline and interior edge points. Inner points refer to the points which are neither outline nor interior edge points.

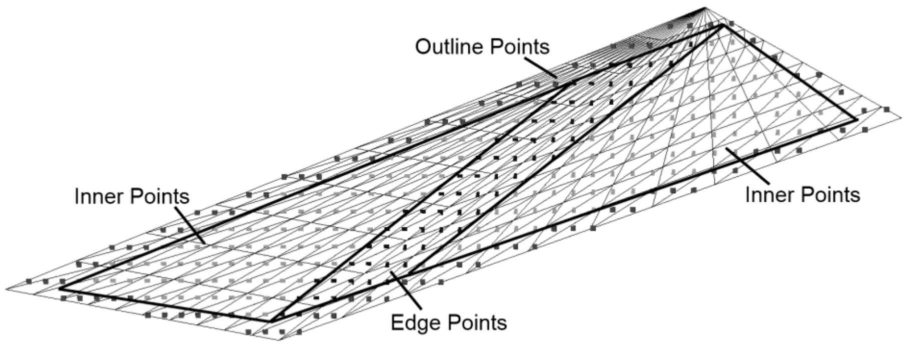


Figure 45 - Pyramid edge and inner points

Over 1000 random poses, the regular case where all points were used was registered. The result of these trials shows the typical trend for geometric constraint, as pose error diminishes with growing geometric constraint. When only the edge sets were registered, the errors at lower geometric constraint were much lower than when the inner sets were used. For the same view, the regular, edge and inner point sets were analyzed twice by employing a high density scan and a lower density scan. Figure 45 shows a pose where the number of edge points roughly equals the number inner points.

The high and low density regular point clouds were split into the edge set and inner set. As seen in Table 4, a denser scan typically had edges points accounting for approximately one tenth of the point cloud. In the less dense scan, the edges were represented by approximately one fifth of the point cloud.

Table 4 - Average number of points in edge/inner point sets

	High Density	Low Density
Regular points	4070	1018
Edge points	489	224
Inner points	3580	793

4.2.2 Edge Point Generation

To create the edge point set, the $[\theta_x, \theta_y]$ data used to calculate the direction vector, $\vec{d}$, in the raster scan implementation was required; see Equation 3.19 and Section 3.4.3.1. By keeping track of the values used over a range of $[\theta_{x1}, \theta_{xn}]$ and$[\theta_{y1}, \theta_{yn}]$, the scanned point cloud can be placed into a 2-D matrix. Not all rays will generate an intersection which results the zero value elements. Consider:

$$\begin{bmatrix} (\theta_{x1}, \theta_{y1}) & (\theta_{x1}, \theta_{y2}) & \cdots & (\theta_{x1}, \theta_{yn}) \\ \vdots & \vdots & \ddots & \vdots \\ (\theta_{xn}, \theta_{y1}) & 0 & \cdots & (\theta_{xn}, \theta_{yn}) \end{bmatrix}$$

The outline and silhouette of the point cloud can be found by selecting the first and last non-zero element of each column and row. To find the interior edge set, each element in the 2-D matrix is checked against its immediate neighbours:

$$\begin{tikzcd}
(\theta_{x1},\theta_{y1}) & (\theta_{x1},\theta_{y2}) \arrow[r] \arrow[ddl] \arrow[dd] \arrow[ddrr] & \cdots \arrow[r] & (\theta_{x1},\theta_{yn}) \\
\vdots & \vdots & \ddots & \vdots \\
(\theta_{xn},\theta_{y1}) & 0 & \cdots & (\theta_{xn},\theta_{yn})
\end{tikzcd}$$

Starting at the (1,1) position of the matrix, only elements below and to the right need to referenced, when available. An edge is identified if the normal of the current element differs from that of the element being checked. When the normals from two elements is aligned, the dot product will be $\cong 1$. Conversely, if the dot product of two unaligned normals falls below an arbitrary value of 0.9, then it is assumed that both of these points lie on either side of an edge and are identified as edge points. The inner set is formulated as the complement to the edge set.

It may be possible to also create edge points by looking at small sets of point clouds and using the normals to define planes. By defining a plane, the points can be fitted to a matrix similar to the one above. Alternatively, if the boundaries of the surface is known, then edge points may be selected if they are within a certain distance from a boundary edge. This is a much more complicated process but may be possible for shapes with simple geometry that can be represented as squares or triangles.

The resulting sets contain only either edge points or inner points. However, when the angle between the camera and the surface is small, then the surface is seen as being shallow. It is possible for a very large surface to appear small if a view is sufficiently shallow. A shallow surface will result in the surface having only a few points scattered across it; see Figure 46. The edge generation algorithm is likely to select many or all of the points on these faces as being edge points. However, as the surface is shallow, the projected area is small, any movement of the point cloud will be constrained by the surrounding edges. Therefore, despite designating most of a surface as edge points this is correct behaviour; the intent of this test is to remove points which constrain the model during ICP registration.

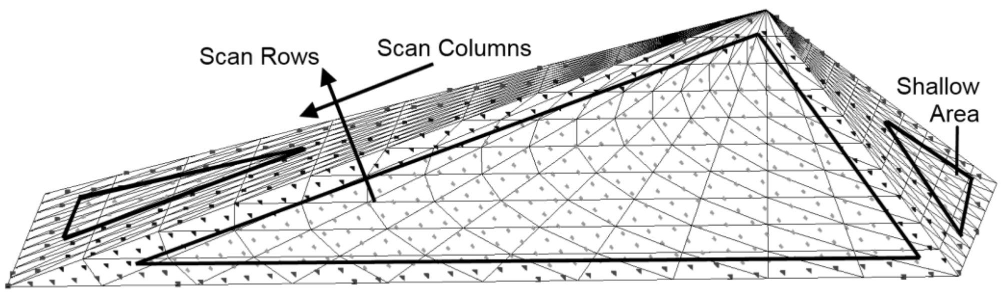


Figure 46 - Shallow scanning

4.2.3 Edge Constraint for a Pyramid

When comparing Figure 47 with Figure 48, one can see that the edge points seem to constrain the model fairly well when there is low geometric constraint. This is reinforced by the higher error that the interior point set experiences with lower geometric constraint. When geometric constraint increases, the overall error level drops for the edge, the inner and the regular point sets. As the baseline case uses all the points, it is constrained similarly to the edge points; see Figure 49. This effect is seen in both the high density and the low density scans.

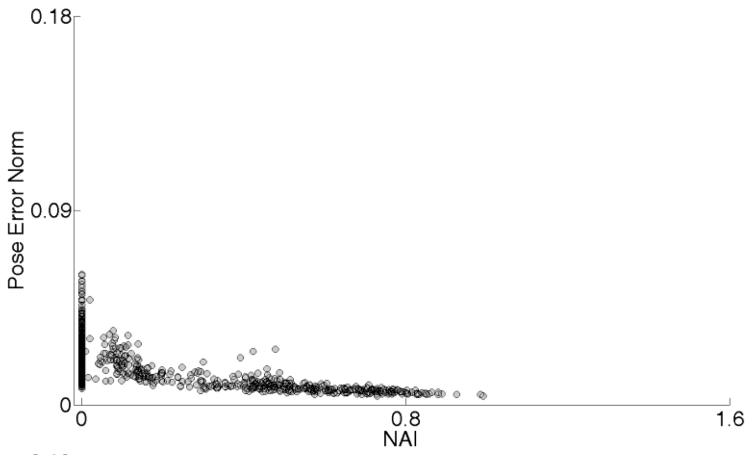


Figure 47 - Pyramid pose error vs. NAI: edge points

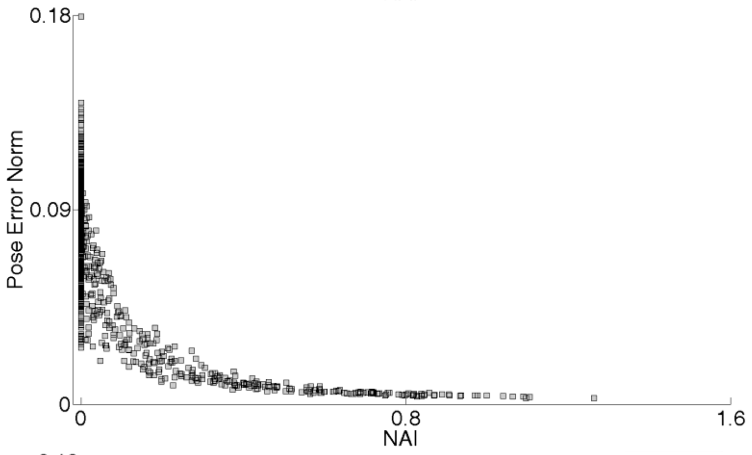


Figure 48 - Pyramid pose error vs. NAI: inner points

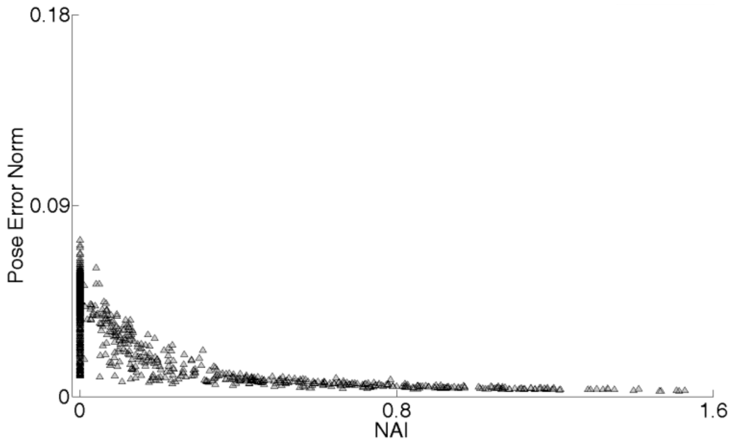


Figure 49 - Pyramid pose error vs. NAI: regular points

Figure 47, Figure 48 and Figure 49 show the un-normalized NAI. This is because the same views on the same object are being assessed with differing point selection strategies. Typically, point normalization is done so that different views are analyzed for geometric constraint on a per-point basis. However, the inner and edge points are expected to provide similar geometric constraint as the same surfaces are used. When NAI is normalized by ($\sqrt{N}$), Figure 50 shows similar results to Figure 47 and Figure 48. As the edge sets have fewer points which provide a similar level of geometric constraint to the inner set, there is an over-exaggeration of the geometric constraint when the edge set is normalized.

Figure 50 - Pyramid: pose error vs. normalized NAI: edge and inner points

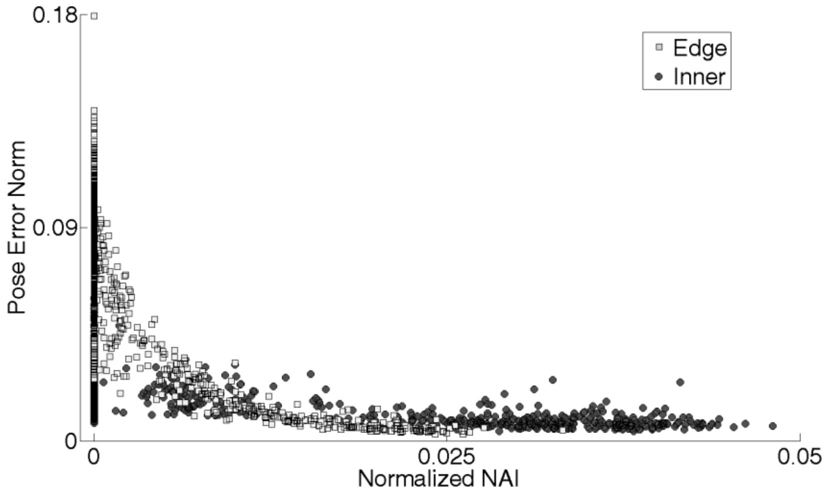


Figure 52 shows the comparison between inner points generated by a high density scan, and edge points generated by a lower density scan. At low geometric constraint, the edge point set still provides much better registration than the inner point set. This means that intelligent point selection can perform better than increasing the scan density as a means for reducing pose errors. However, there is a difference in the pose error when comparing the results of the inner points for the high and low density scans in Figure 48 and Figure 52. This can be explained by the encroachment of the inner point set towards the edges. In the higher density scan, the edge points surrounding the interior edges are much closer together than in a lose scan. As a result, the inner points may lie much closer to the edges and thus be provided with some degree of edge constraint; this can be seen in Figure 51.

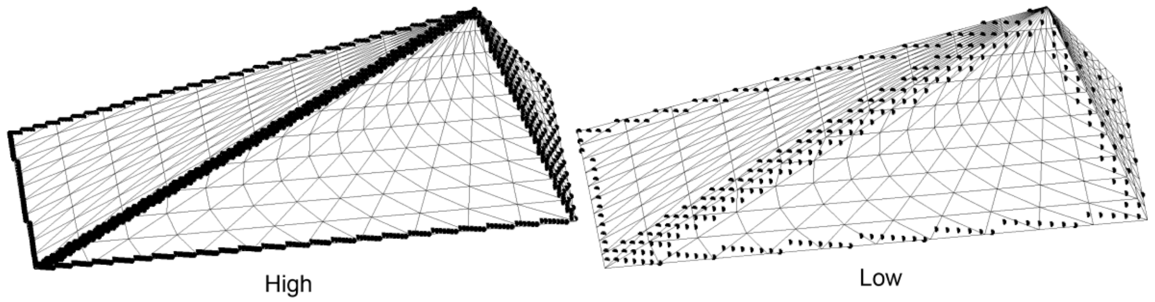


Figure 51 - Comparison of edge selection in high and low density scans

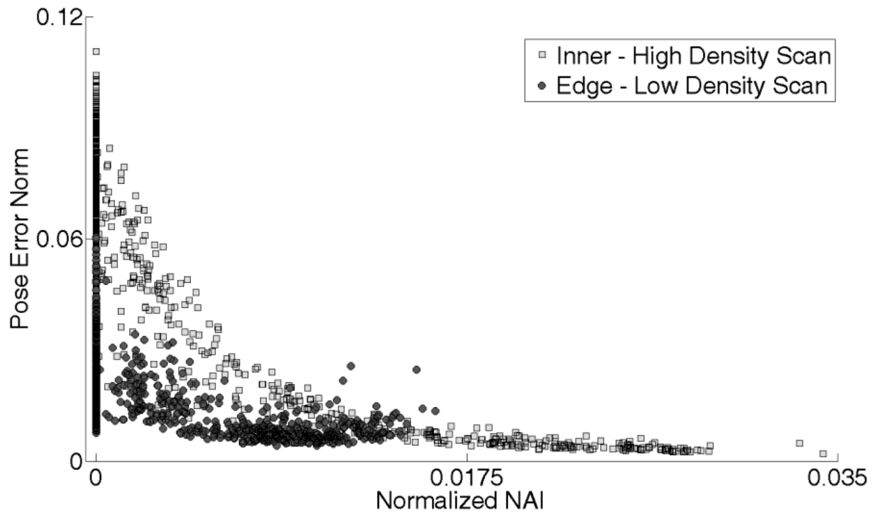


Figure 52 - Pyramid pose error vs. normalized NAI: high and low density scans

4.2.4 Edge Constraint for Space Shuttle

The investigation of edge constraint can also be applied to the Shuttle as well. Each of the large dots in Figure 53 represents a view with similarly low geometric constraint and represent a range of pose errors. These views typically targeted the Space Shuttle's protective thermal tiles; as an over simplification, the underside of the Shuttle can nearly be approximated to a flat plate as it is very slightly rounded. On its own, a flat plate has been proven to only constrain three degrees of freedom, the Z-axis translation and the X-axis and Y-axis rotation, see Gelfand & Rusinkiewicz (2003). When a flat plate is rotated, the rotational constraint can increase, as seen in Section 3.4.5.3.

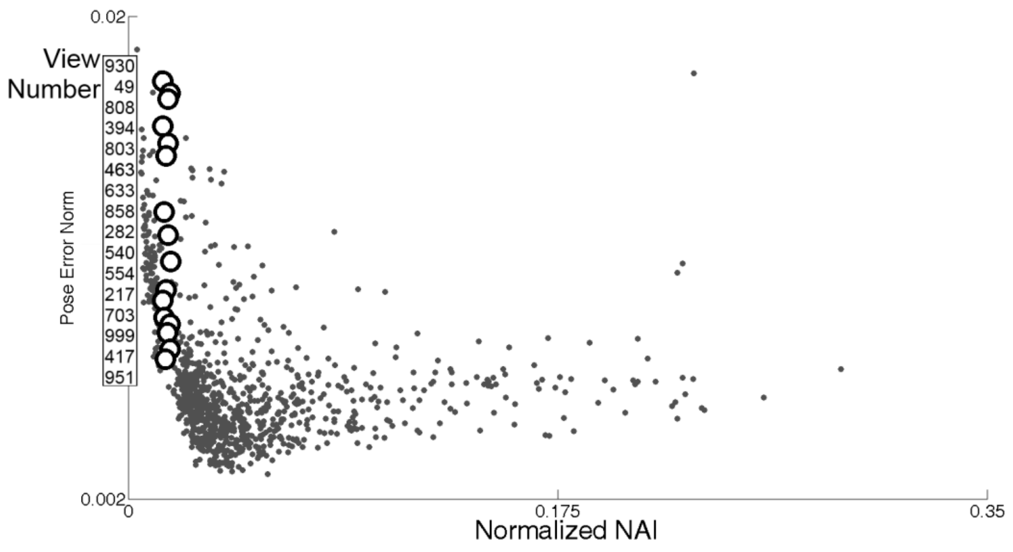


Figure 53 - Space Shuttle: pose error norm vs. normalized NAI

From Figure 53, pose 951 and pose 394 are examined and are shown in Figure 54.

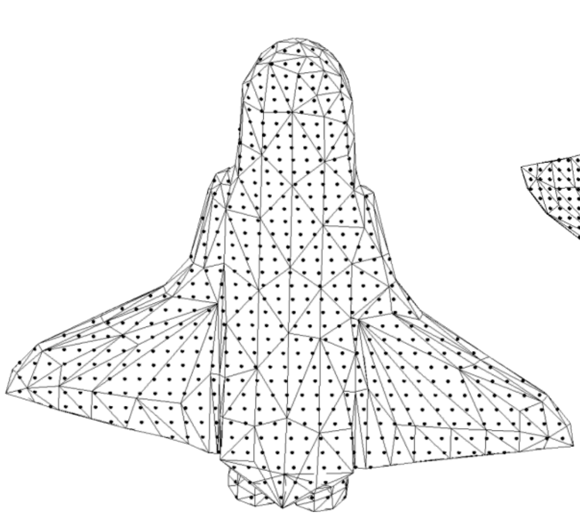


Figure 54 - Low geometric constraint: Shuttle poses 951 and 394

By using the edge removal method previously described, the selected Shuttle poses were analyzed using the edge and inner point sets. Figure 55 shows the results when only the edge points were registered for the selected poses. The vectors in Figure 55 indicate the results of the associated edge point sets for each selected poses. Typically, the use of edge points slightly decreased the overall pose error norm and standard deviation. This may indicates that performing ICP using a smart selection of points can be beneficial.

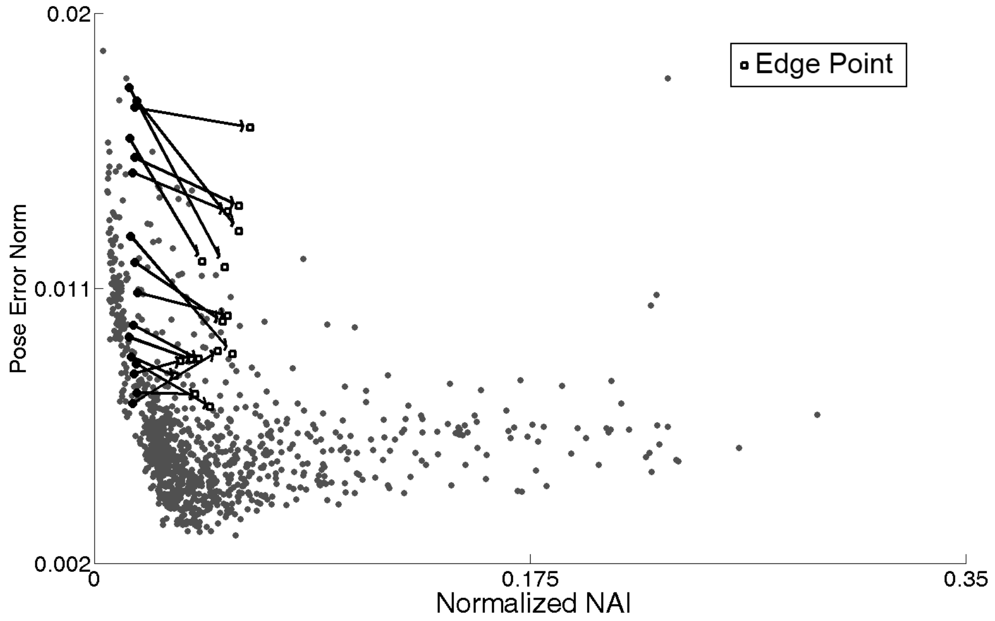


Figure 55 - Shuttle edge results

Initially, the Pose 951 has a much lower registration pose error and a lower pose error standard deviation. However, once the edge points are removed to decrease the edge constraint, the error levels rise substantially. Table 5 shows the average pose error norm and pose error standard deviation for each view shown in Figure 55. In all but three of the cases, the use of edge points gave better alignment and registration. Conversely, the use of the inner point set significantly raised the average pose error and pose error standard deviations. This does not indicate that using inner points will result in erroneous registration, only that it is poor relative to the use of the edge or regular sets.

Table 5 - Pose error norm and standard deviation for select poses

	Pose Error Norm			Pose Error Standard Deviation		
Pose	Regular	Edge	Inner	Regular	Edge	Inner
394	0.0179	0.0073	0.0766	0.0172	0.0036	0.0409
930	0.0165	0.0086	0.0806	0.0160	0.0047	0.0355
808	0.0165	0.0109	0.0680	0.0161	0.0153	0.0385
803	0.0146	0.0071	0.0752	0.0143	0.0036	0.0393
463	0.0132	0.0158	0.0772	0.0149	0.0297	0.0308
49	0.0122	0.0076	0.0785	0.0130	0.0033	0.0410
858	0.0106	0.0059	0.0766	0.0092	0.0027	0.0392
282	0.0093	0.0053	0.0780	0.0076	0.0023	0.0343
554	0.0075	0.0056	0.0733	0.0054	0.0019	0.0362
540	0.0066	0.0057	0.0879	0.0041	0.0024	0.0425
703	0.0064	0.0047	0.0702	0.0052	0.0019	0.0372
417	0.0062	0.0047	0.0776	0.0045	0.0017	0.0412
217	0.0059	0.0048	0.0733	0.0042	0.0020	0.0417
999	0.0055	0.0049	0.0863	0.0043	0.0023	0.0436
951	0.0052	0.0061	0.0790	0.0027	0.0025	0.0450
663	0.0037	0.0038	0.1020	0.0017	0.0015	0.0578

To see the effect increasing the number of points, a high density scan was used to increase the number points in the inner set. The inner set increased from an average of 501 to 4758 points. Similar to how the pose error for the pyramid dropped, the increase in the number of points caused the average pose error norm to drop from 0.0745 to 0.0476. This is still significantly higher when compared to the case when only the edge sets or the regular sets were used. As with the pyramid study, this may be because the encroaching inner points allow for more edge constraint. Another reason for the improved pose estimation is that the ICP algorithm can be affected by the number of points used. If the data is very noisy, then providing more data for the registration covariance matrix will bring the average closer to the expected values. The consequence of using more points is more computational time and memory. The effect of adding more points will be further discussed in Section 4.3.4.

4.2.5 Conclusion

This Section shows that at lower geometric constraint, poses with low NAI, the governing constraint can be attributed to the edge constraint rather than geometric constraint. At higher geometric constraint, high NAI, the contribution of edge constraint diminishes or is outweighed, and the pose estimation accuracy will be depend mainly on variations of the surface geometrical structure.

4.3 Selection of Regions for Efficient Pose Estimation

As previously mentioned, there are several ways in which elements from a point cloud are selected. Some selection algorithms include filling out the normal space and random sampling; however in Shahid & Okouneva (2007), a different point selection scheme is introduced. The subdivision of the model's surface is used to find and locate the area with the highest constraint. Each subdivision is called a window, and each window peeks into a different area of the point cloud and model surface. In Figure 56 each of the dots on the surface of the model represents a scan point, while the square outlines indicate the window areas. Shahid & Okouneva search for one near-optimal scanning region and uses NAI as the measure of geometric constraint. LIDAR scanning can be performed for just one selection region and still achieve acceptable pose accuracy.

Although only one window was previously used, the selection of windows can be extended to combine two or more windows together. This has the potential advantage of pairing several poorly constrained windows to make a window with a well constrained combined point cloud. Using a window rather than the whole scan can possibly lead to increases in speed as the ICP algorithm should work faster when fewer points are being registered. However, the use of a window is valid only if results provide adequate pose estimation accuracy. The expected result of combining two windows should be a good estimate of the true pose. This estimate can then be used as the initial guess for another ICP registration using all points. However, it is desirable to develop an algorithm which selects a combination of windows to generate pose estimates comparable to using the whole point cloud.

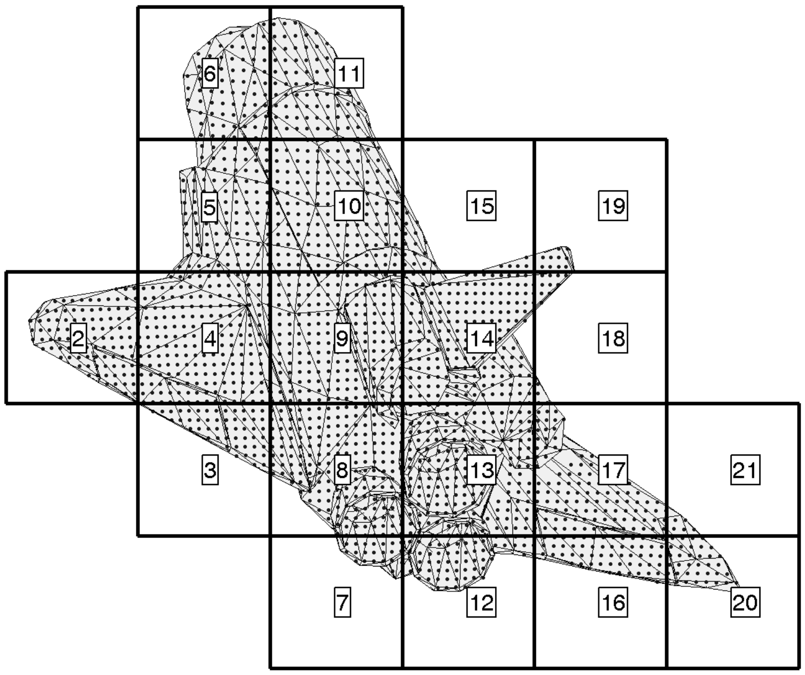


Figure 56 - Shuttle windows

4.3.1 Creating Windows

Windows can be created by simply subdividing a point cloud into a grid based on the X and Y axis values. However, to make the subdivision more accurate, the proper perspective must be used, see Figure 57. A wrong perspective may arise when a scanner projects the point cloud onto a viewing plane.

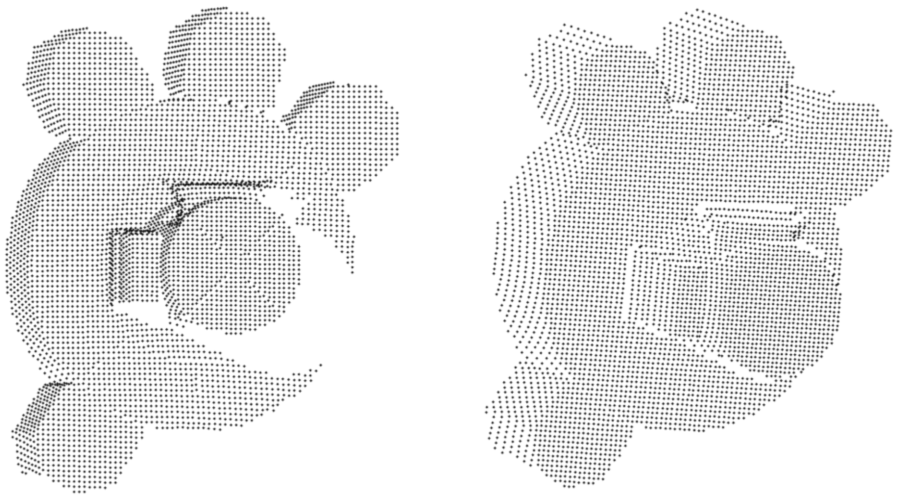


Figure 57 - Airlock wrong and right perspectives

As the camera origin is known, a direct vector from the point cloud to the origin can be defined. Using this vector, the X and Y axis rotation values can be found, which can then be used to create a rotation matrix that will rotate the point cloud so that it lies in X-Y plane and therefore with the correct perspective for subdivision. The Z-axis values are ignored when splitting the point cloud as the Z-axis is orthogonal to the viewing plane.

The square windows are used in this investigation. If square or rectangular window is used, the points can be verified that they fall between the X-axis and Y-axis window boundaries. If a window has a circular shape then the Euclidean distance from the center of the window to the point can be used. Theoretically any window shape or size could be used in conjunction with MATLAB's *inpolygon* function.

This application can be applied to raster, rosette and lissajous scans. However, when the windows are applied, they no longer simulate a LIDAR device focusing on one area of a model. A rosette or lissajous scan focused on one area is quite different than selecting points from a larger scan, as seen in Figure 58.

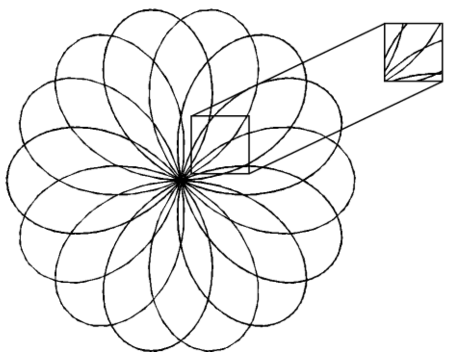


Figure 58 - Rosette subsection

There are two factors that govern the creation of windows: the window size and the step size. A larger window will contain more points but will eventually approach the case where all points are used. Conversely, a smaller a window may contain too few points, which can result in poor ICP performance. A smaller window may miss certain geometric features or surface fluctuations, by not being able to feature all of it. While combining several small windows together should improve the constraint; a smaller window size will increase the number of windows significantly. Akin to the voxel sizing, every time the window size is halved, the number of windows increases by four. Figure 59 shows the effect of decreasing both the window size and step.

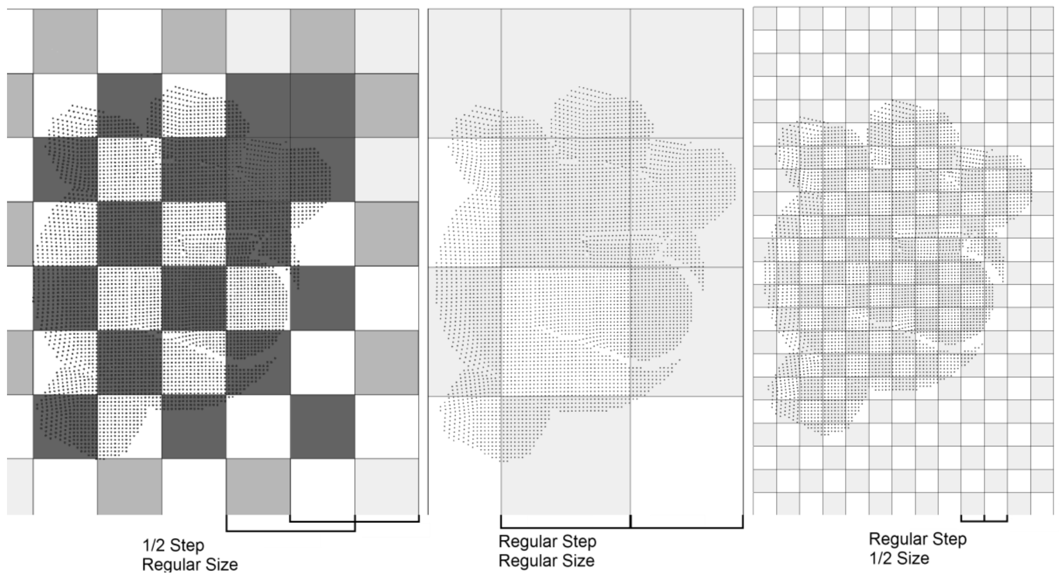


Figure 59 - Effect of window size and step

The checkerboard patterns shown in Figure 59 are for demonstration purposes only and are only used to enhance the distinction between window subsections. The darker tiles shown in the ½ step are the result of multiple windows overlapping. A darker coloured tile corresponds to more overlap; with a ½ step size, it is possible for 1, 2 or 4 windows to feature the same subspace. If the step size is set to be the same as the window size then the windows are mutually exclusive. If the step size is larger than the window size, then there will be gaps in the grid and features will be missed. While it is perfectly valid to overlap the windows it increases the number of windows significantly. If the step size is the half the window size, then the number of windows increases via

$$(2 \times m - 1) \times (2 \times n - 1)$$

where $\begin{cases} \text{m is the number of mutually exclusive windows along the X-direction} \\ \text{n is the number of mutually exclusive windows along the Y-direction} \end{cases}$

Although, this is may be the maximum number of windows, it is more likely that there will be windows which contain no points and can be discarded.

If the number of windows is increased, then there will be more individual window trials and more window combinations to be registered using ICP. Given all the permutations, the total number of trials required increases steeply when selecting more than two windows.

4.3.2 Combining Windows

Once the windows have been created, each window and each window combination is registered using the ICP algorithm. By first registering all the permutations of two windows, an algorithm for selecting windows can be quickly tested. For combinations of three or more, the windows should be selected first and then registered due to the large number of potential cases. There are several possible means of selecting a second window to complement a given first window, and can be based on the following strategies or properties:

1. Random selection
2. Normal space
3. Window distance
4. NAI value – a priori or a posteriori
5. Expectivity value – a priori or a posteriori
6. Maximum eigenvectors
7. Minimum eigenvectors

With regards to the strategies above, the random selection of a second window will most likely be the worst performing pairing algorithm. Alternatively, the second window can be selected by filling out the normal-space. If the normal vectors contained by a window all point in one direction, then several degrees of freedom will be unconstrained. By finding a window with complementary normal vectors, a more balanced normal space is defined, this should allow more for more overall constraint. It may also be possible to select several windows located far apart to help constrain based on the rotational weights. Using windows which are not adjacent to each other allows different features of the model to be targeted. The use of two window distant windows increases the chance that more surface features are selected.

Alternatively, the second window can be selected by considering the eigenvalues, and geometric constraint measures derived from the constraint matrix. The NAI, ME, and EI measures can be calculated *a priori,* before the windows are combined, for each individual window. The largest two values for each of the constraint measures should indicate which individual windows to select. Alternatively, the NAI, ME, and EI measures can be calculated *a posteriori*, after the windows have been combined.

For a window, the maximum eigenvector, v_1^1, associated with the maximum eigenvalue, λ_1^1, shows the direction in which the window's point cloud is the most constrained. However, it is unlikely that v_1^1 will constrain all DOFs. Therefore, a second window should be selected with a v_1^2 that complements v_1^1. The complement to v_1^1 is not $-v_1^1$, an eigenvector in the opposite direction; this is because an eigenvector is equal to its negative value, $-v_i = v_i$. Therefore, a selection algorithm should avoid the case where the angle between the maximum eigenvectors of the first and second windows, v_1^1 and v_1^2, is close to either $0°$ or $180°$. To find the complement to v_1^1, an orthogonal eigenvector is required, this means that the angle between the maximum eigenvectors will be near $90°$ or $270°$ and the dot product between v_1^1 and v_1^2 will approach one.

Similarly for a window, the minimum eigenvector, v_6^1, defines the direction of the weakest geometric constraint. Instead of finding an orthogonal vector, a parallel maximum eigenvector is sought to boost the constraint in that direction. This means that the angle between v_6^1 and v_1^2 should be close to either $0°$ or $180°$, and the dot product will approach zero. The effect is that another window of dissimilar constraint is selected. This method might also be applicable to a combination of two minimum eigenvectors v_6^1 and v_6^2. While the angles between eigenvectors may be used, it is easier to evaluate using the absolute value of the dot product, this removes the cyclic nature of the angles. The dot product of the angles between (v_6^1, v_6^2), (v_6^1, v_1^2) and (v_1^1, v_1^2) can be calculated.

The previous algorithms reference the eigenvectors, v_i, but do not take into account the strength of the constraint given by the eigenvalues, λ_i. New strategies are developed when the eigenvectors are scaled by multiplying them with their associated eigenvalues.

$$scaled\ eigenvector = \lambda_i \cdot \vec{v}_i = \vec{\lambda}_i \tag{4.1}$$

When the largest scaled eigenvectors are combined, $\vec{\lambda}_1^1$ and $\vec{\lambda}_1^2$, then the question of orthogonality becomes which window pair results in the largest triangular area in the eigenspace. Suppose there are two vectors, $\vec{\lambda}_1^1$ and $\vec{\lambda}_1^2$ of specified length λ_1^1 and λ_1^2, then the maximum area they can form will be when they are orthogonal; see Figure 60.

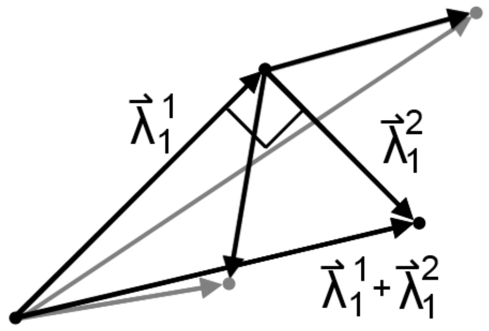


Figure 60 - Triangle area

To calculate the area of a k-dimensional triangle, Heron's formula requires the semiperimeter, s, and the length of each side:

$$area = \sqrt{s(s-A)(s-B)(s-C)} \qquad \textbf{4.2}$$

$$\text{where} \begin{cases} s = \frac{1}{2}(A+B+C) \\ A = \lambda_1^1 \\ B = \lambda_1^2 \\ C = \sqrt{A^2+B^2} \end{cases}$$

A larger area should simultaneously indicate (1) that the combination eigenvectors of the two windows are orthogonal and (2) that eigenvalues have significance. This can be seen with a simple example, illustrated in Figure 61:

- Windows 1 and 2 are similar: $v_1^1 \approx v_1^2$ and $\lambda_1^1 \approx \lambda_1^2$
- Windows 3 and 4 are somewhat similar: $v_1^3 \approx v_1^4$ but $\lambda_1^3 \gg \lambda_1^4$

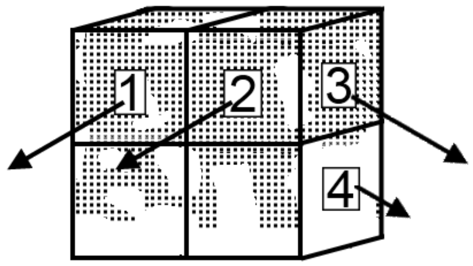


Figure 61 - Area example: box

The area given by the combination of Window 1 with the other windows is seen in Figure 62.

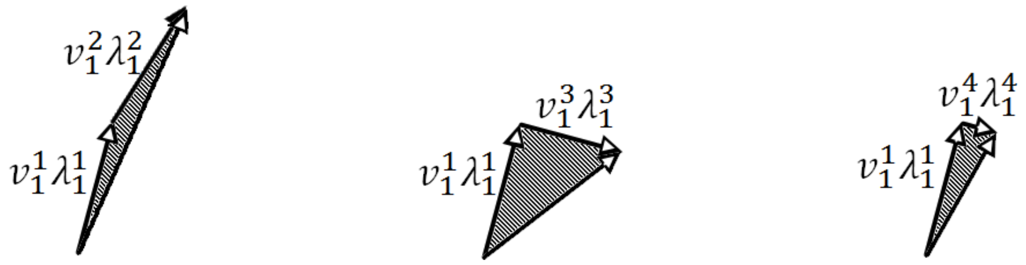


Figure 62 - Area example: eigenspace area

Windows 1 and 2 constrain in similar directions, therefore despite the large eigenvalues, they are eliminated as v_1^1 and v_1^2 are not orthogonal. Conversely, Windows 1 and 4 are orthogonal which is desirable, however, they are eliminated as λ_1^4 is negligible. Windows 1 and 3 are orthogonal and both eigenvalues have significance, $\lambda_1^1 \gg 0$ and $\lambda_1^3 \gg 0$. Therefore, the combination of Windows 1 and 3 will provide the best constraint out of all the cases, and they form the largest eigenspace area in Figure 62.

The same procedure cannot be performed between the minimum eigenvalue and the maximum eigenvalue, this is because $\vec{\lambda}_6^1$ is often relatively very small and will be dominated by $\vec{\lambda}_1^2$. The eigenspace area can be calculated for two minimum eigenvalues, $\vec{\lambda}_6^1$ and $\vec{\lambda}_6^2$, but will result in very small values. This is because the window subdivisions can be very poorly constrained and the minimum eigenvalues represent the weakest constraint. Therefore, λ_6 is very small and occasionally is zero. With the constraint matrix, a zero eigenvalue denotes no constraint in the eigenvector direction and the point cloud can freely rotate and/or translate without affecting the ICP cost metric.

4.3.3 Pose Estimation Results for Combinations of Window

The use of two windows is generally more accurate than using one. Windows 14 and 19 from Figure 56 are combined together to form the point cloud, Cloud 118. Figure 63 depicts two different views of Cloud 118 to provide some depth.

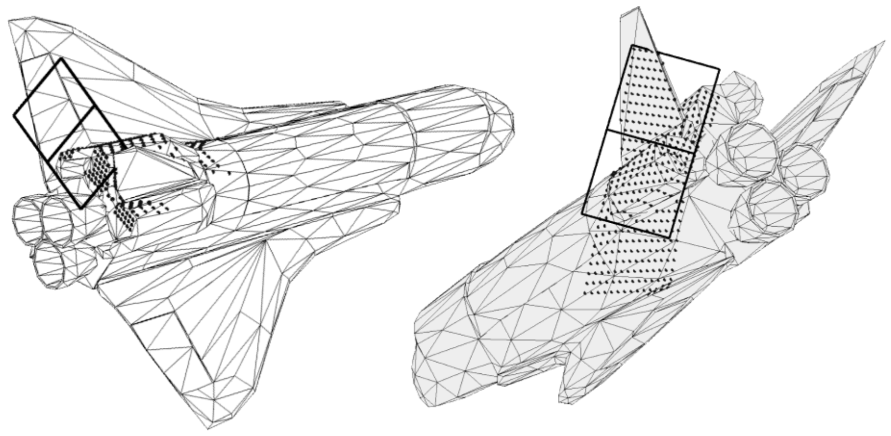


Figure 63 - Cloud 118

Individually, the two windows have relatively good pose error; however, once combined together, the pose error norm for the cloud decreased dramatically, as shown in Table 6. Also seen in Table 6 is the rise in geometric constraint reflected by the NAI and Expectivity values.

Table 6 - Window and cloud results

	Pose Error Norm	NAI	Expectivity	Normalized NAI	Normalized Expectivity
Window 9	0.1054	0.1431	0.9650	0.0102	0.0688
Window 14	0.1774	0.2172	1.0905	0.0154	0.0775
Cloud 118	0.0052	0.4520	1.8784	0.0227	0.0945

The rise in geometric constraint is because each alone window is well constrained in a limited number of directions. Table 7 shows the normalized constraint matrices. By combining Window 9 and 14 which are well constrained in t_x and t_y, respectively, the resultant Cloud 118 becomes well constrained in both. Both Windows 9 and Window 14 are poorly constrained in all rotations, but by combing the windows the rotational constraint is improved, especially in ω_y. However, the rotational constraint remains the weak comparing to the translational constraint.

Table 7 - Window and cloud eigenvectors

λ_i	t_x	t_y	t_z	ω_x	ω_y	ω_z
11.64	-0.78	0.58	0.22	0.02	0.07	0.00
2.16	0.30	0.47	0.09	-0.49	-0.65	0.12
1.60	-0.53	-0.57	-0.21	-0.30	-0.48	-0.15
0.75	0.04	-0.28	0.93	0.13	-0.17	-0.08
0.20	-0.08	-0.06	-0.08	0.52	-0.35	0.77
0.13	0.08	0.19	-0.16	0.62	-0.43	-0.61

Window 9

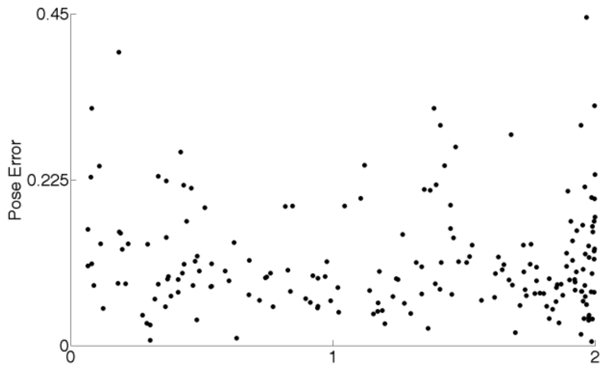
λ_i	t_x	t_y	t_z	ω_x	ω_y	ω_z
7.83	-0.85	0.42	0.28	0.02	-0.04	0.09
3.95	-0.11	-0.69	0.70	-0.16	0.05	0.05
2.52	0.49	0.56	0.66	0.10	0.02	-0.07
0.56	0.07	0.01	0.02	-0.23	-0.96	0.14
0.32	-0.05	-0.19	0.04	0.94	-0.24	-0.11
0.16	0.11	0.01	-0.01	0.15	0.11	0.98

Window 14

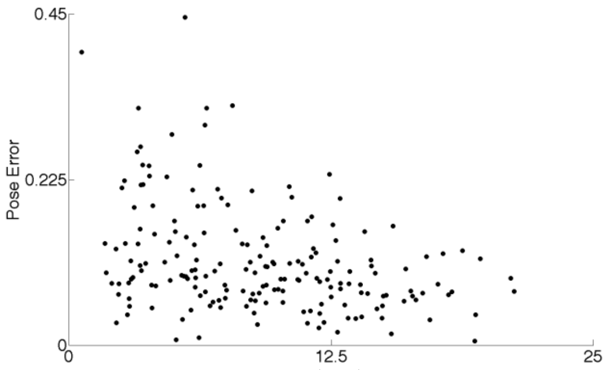
λ_i	t_x	t_y	t_z	ω_x	ω_y	ω_z
13.74	-0.80	0.53	0.23	0.07	0.08	-0.01
3.88	0.32	0.53	-0.42	0.23	0.61	-0.15
3.18	0.28	0.62	-0.13	-0.31	-0.60	0.25
2.73	-0.42	-0.21	-0.86	-0.14	-0.11	0.09
0.51	0.01	-0.03	-0.03	0.70	-0.09	0.71
0.38	-0.01	-0.05	0.11	-0.58	0.49	0.63

Cloud 118

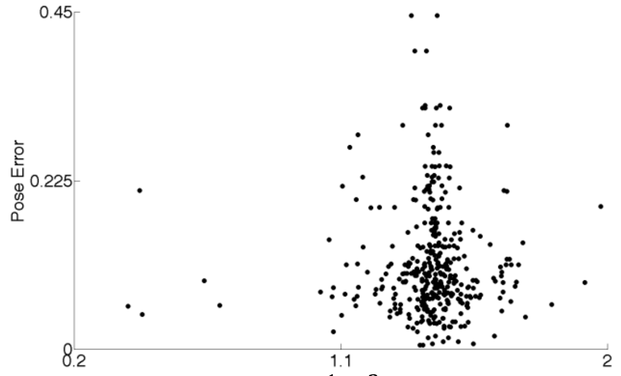
With more variation in the normal vectors and point placement, the geometric constraint, and therefore the eigenvalues λ_i, increase. Shown below in Figure 64 are the strategies from Section 4.3.2 and the resulting pose error norm for Figure 56. *Add* refers to the strategy of addition of the eigenvectors $\vec{v}_i$ or of the scaled eigenvectors $\vec{\lambda}_i$. *Area* refers to the strategy of calculating the area described by two scaled eigenvectors $\vec{\lambda}_i$. *Angle* refers to the strategy of finding the angle between two eigenvectors $\vec{v}_i$. *Dot* refers to the strategy of finding the dot product between two eigenvectors $\vec{v}_i$. The PCA measures are shown for the combined windows.



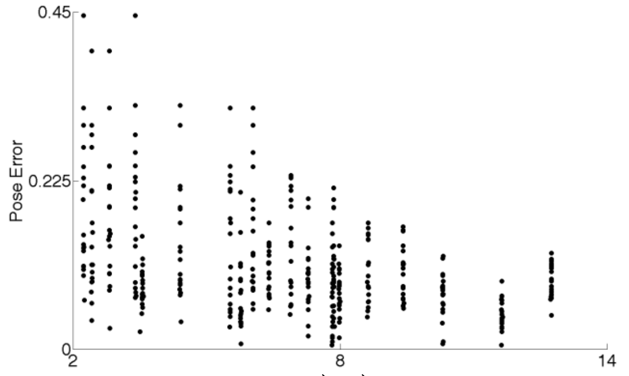
a)Add(v_1^1, v_1^2)



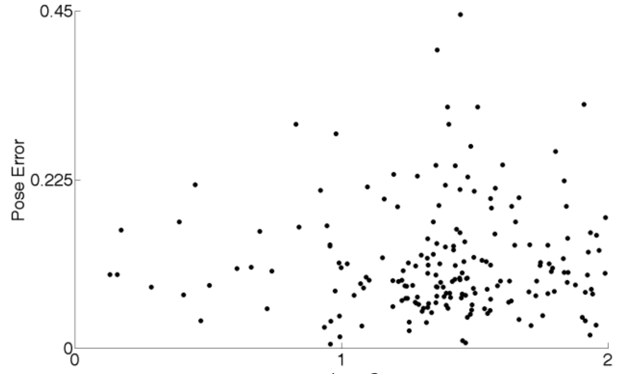
b) Add$(\vec{\lambda}_1^1, \vec{\lambda}_1^2)$



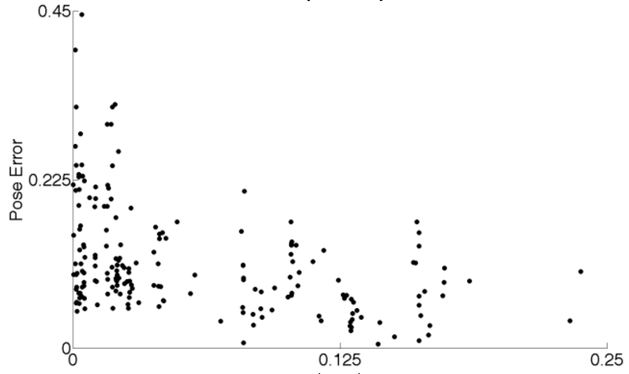
c) Add(v_6^1, v_1^2)



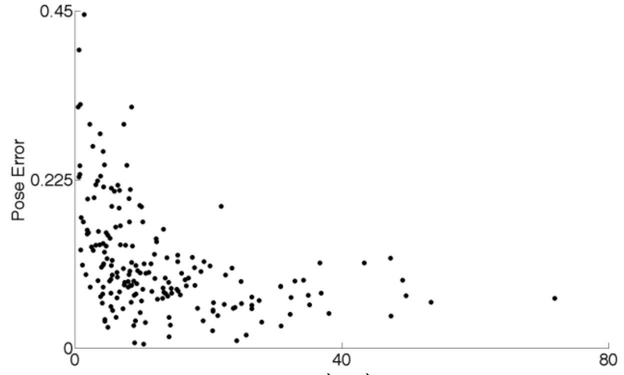
d) Add$(\vec{\lambda}_6^1, \vec{\lambda}_1^2)$



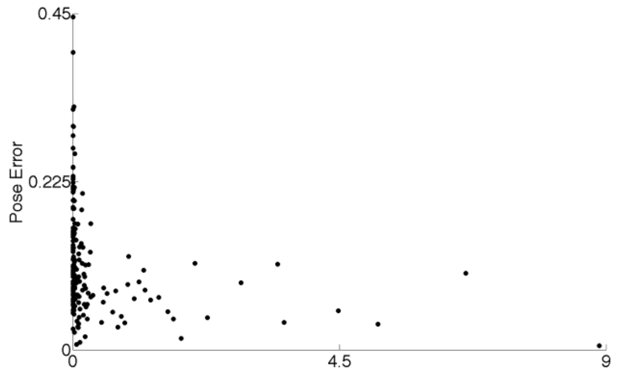
e) Add(v_6^1, v_6^2)



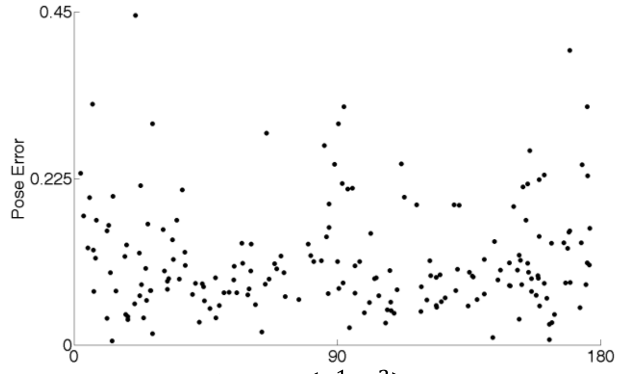
f) Add$(\vec{\lambda}_6^1, \vec{\lambda}_6^2)$



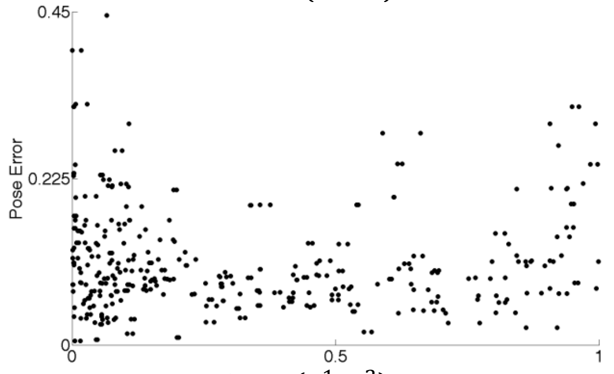
g) Area$(\vec{\lambda}_1^1, \vec{\lambda}_1^2)$



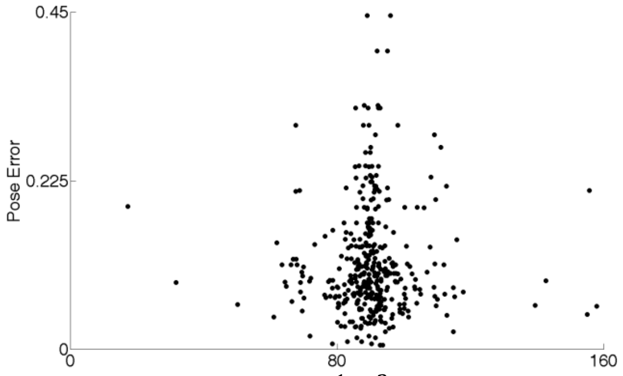
h) Area$(\vec{\lambda}_6^1, \vec{\lambda}_6^2)$



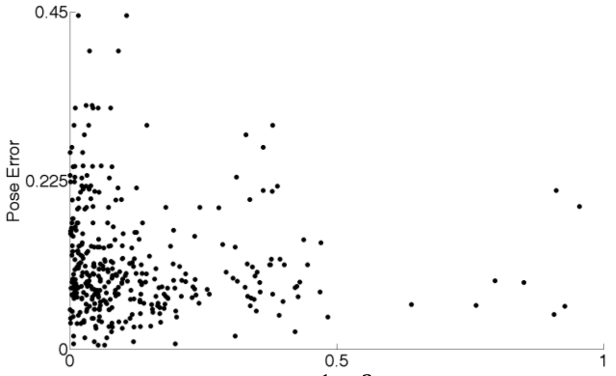
i) Angle (v_1^1, v_1^2)



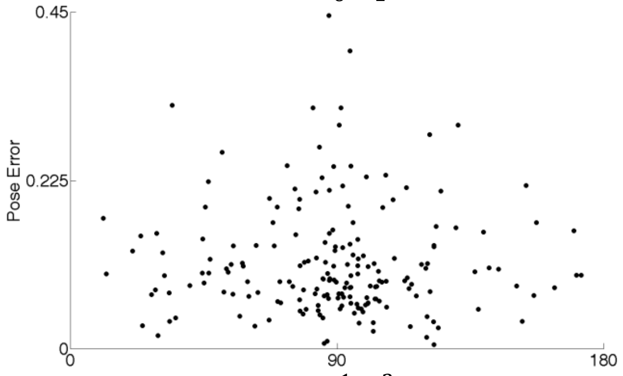
j) Dot (v_1^1, v_1^2)



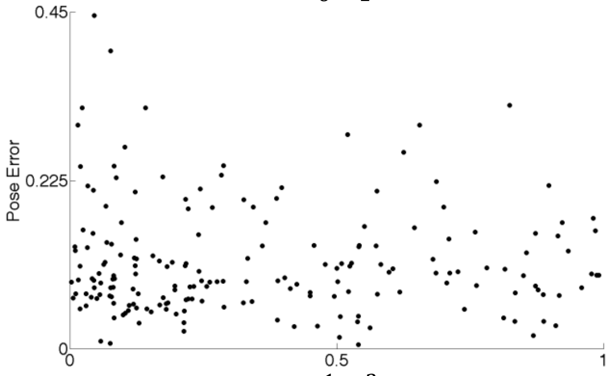
k) Angle (v_6^1, v_1^2)



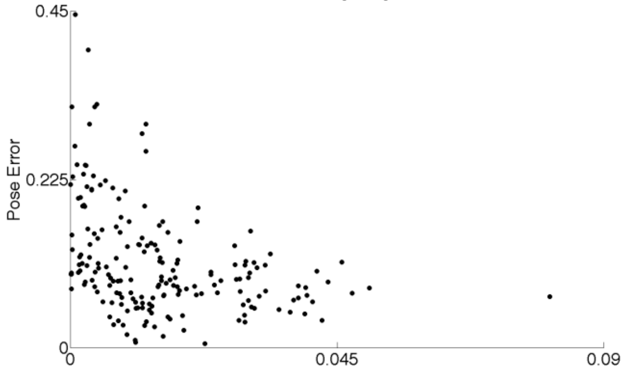
l) Dot (v_6^1, v_1^2)



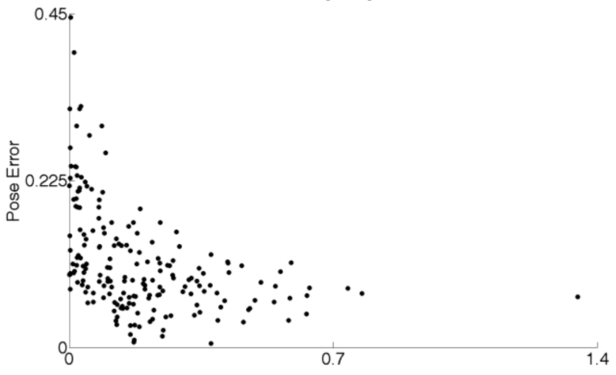
m) Angle (v_6^1, v_6^2)



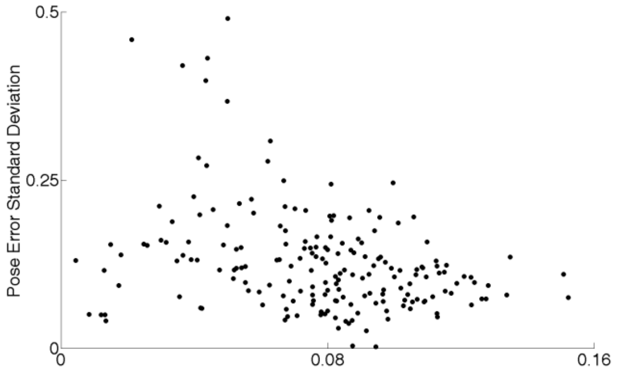
n) Dot (v_6^1, v_6^2)



o)A posteriori normalized NAI



p) A posteriori normalized λ_6



q) A posteriori normalized Expectivity

Figure 64 - Selection strategies

Some of the strategies perform as expected and offered promising results. The addition of the maximum eigenvectors $v_1^1 + v_1^2$ (a) and the addition of the maximum-minimum eigenvectors $v_6^1 + v_1^2$ (c) were not beneficial. It is expected that an orthogonal v_1^1, v_1^2 would result in a complementary constraint. Two orthogonal unit vectors should result in a hypotenuse of $\sqrt{2} \cong 1.41$; however, the minimum pose error appears to be centred around 1 instead. On the other hand, the majority of the high pose errors for $v_6^1 + v_6^2$ (e) were clustered around 1.41. This means that the eigenvectors are orthogonal and that the two windows with different weak constraint directions are selected. However, the low pose errors also appeared around 1.41, which makes this strategy unusable. The orthogonality problem carried over to the angle (i, k, m) based and dot product (j, l, n) based strategies as well, rendering them all unusable. The strategies based upon the eigenvectors alone did not perform very well.

The addition of the maximum-maximum eigenvalue $\lambda_1^1 + \lambda_1^2$ (b), minimum-maximum eigenvalue $\lambda_6^1 + \lambda_1^2$ (d), and minimum-minimum eigenvalue $\lambda_6^1 + \lambda_6^2$ (f) all showed potential. The maximization of any of these measures can possibly lead to an increase in overall constraint. The $\lambda_1^1 + \lambda_1^2$ addition showed the expected decrease in pose error, however, this focuses upon the direction and strength of the maximum constraint; meanwhile, λ_6, which can cause larger errors, is ignored. The $\lambda_6^1 + \lambda_1^2$ addition was dominated by the λ_1^2 as it was comparably much larger than λ_6^1, this too strategy also ignores λ_6^1 by outweighing it. The $\lambda_6^1 + \lambda_6^2$ addition focuses on the weak constraint direction and strength of the two windows. By maximizing the weak constraint $\vec{\lambda}_6$ by adding two similar λ_6 together, there should be an increase in the strength of weak constraint.

The calculated a priori scaled eigenvector measures based on the ares of $\vec{\lambda}_1^1, \vec{\lambda}_1^2$ (g) and the $\vec{\lambda}_6^1, \vec{\lambda}_6^2$ (h) gave very good results. By focusing on the minimum eigenvectors, the $\vec{\lambda}_6^1, \lambda_6^2$ area strategy selected the window pairing with the lowest pose error out of all the possibilities. The implication of this is that by having two large and orthogonal $\vec{\lambda}_6$ is that the weakest direction of constraint is not compounded. This may mean that one of the other $\vec{\lambda}_{1-5}^2$ may coincide with $\vec{\lambda}_6^1$ which would help to increase the constraint in that direction. Additionally, the maximum area implies that two windows are well constrained in two different directions and that the values of $\vec{\lambda}_6^1, \vec{\lambda}_6^2$ are high relative compared to other $\vec{\lambda}_6$.

The PCA based indices (o,p,q) all performed as expected.

Figure 65 and Figure 66 show the values of NAI, EI and ME, both normalized and regular, and for the windows individually before they have been combined. Generally speaking, there is not very much to choose from between the PCA measures. Typically, when the value of ME, λ_6, rises, so do the values of Expectivity and NAI. The non-normalized versions of the geometric constraints are included as this is not comparing the constraint given by two different views of the same object or two different views of two different objects. This is comparing the selection of two different areas on the same object. In this case, there is a distinct advantage for a window to be selected where more points are available. Most windows residing on the outline of the view contain very few points and once normalized the relatively little geometric constraint becomes over-exaggerated on a per-point basis. This can be seen if Window 3 and Window 20 from Figure 56 are examined in Figure 65 and Figure 66.

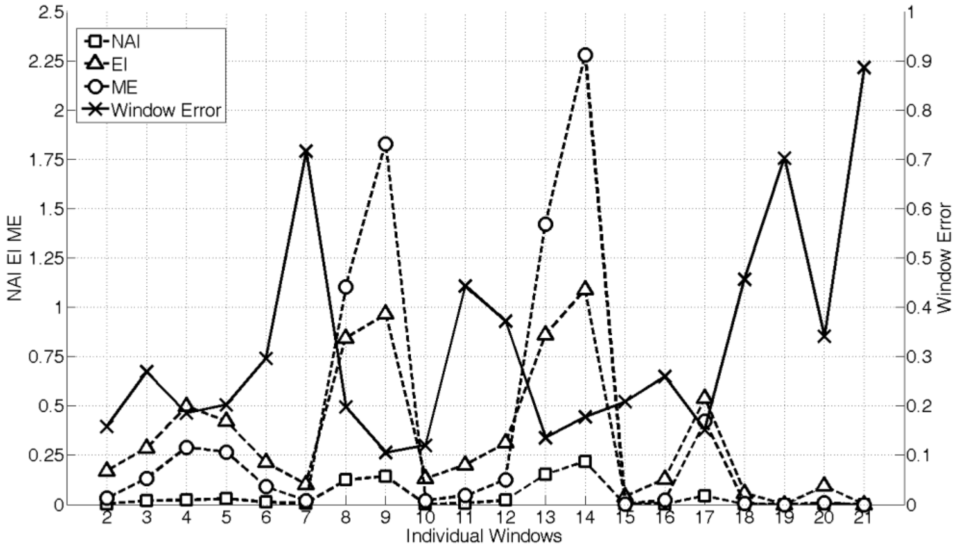


Figure 65 - Individual windows and PCA measures

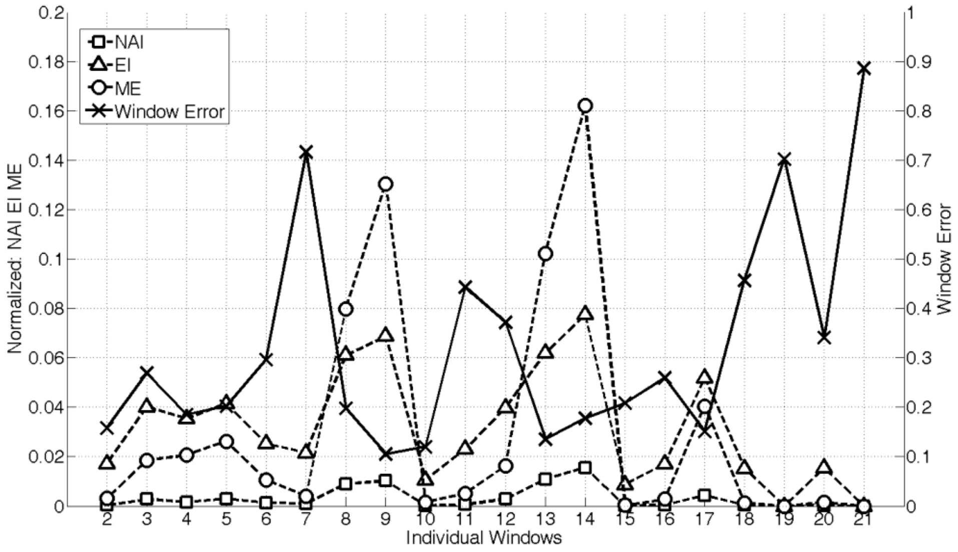
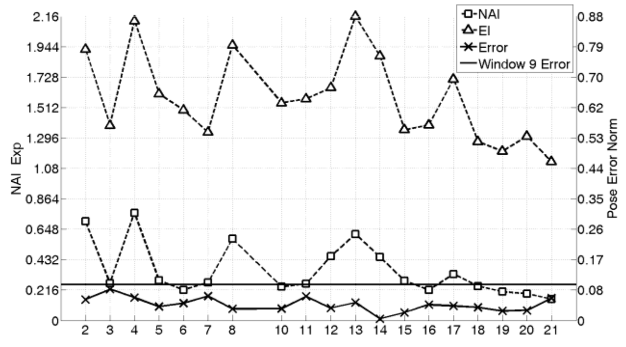
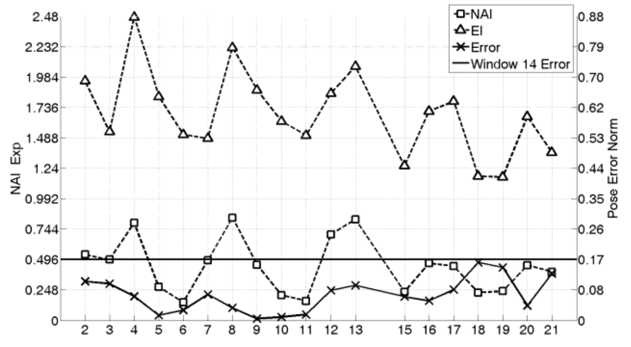


Figure 66 - Individual windows and point normalized PCA measures

The a posteriori measures, calculated after the windows have been combined, include the NAI and Expectivity indices. Figure 67 a) and b) show the results of combining Window 9 and Window 14 with all other windows. The NAI and Expectivity values for the combinations are plotted on the left Y-axis, while the pose error norm is plotted on the right Y-axis. The X-axis indicates which window is being combined with Window 9 or Window 14.



a) Combined with Window 9



b) Combined with Window 14

Figure 67 - Window 9 and Window 14 NAI and EI results

It is evident that in Figure 65 through Figure 67, the range of each PCA index will vary. The value of the Expectivity index is almost always higher than the NAI value for the same window. However, it is not the values of the PCA measures that are being compared, but rather the relative values and where the peaks are located.

When there is a small amount of geometric constraint, the calculation of the Expectivity index is modified to ignore eigenvalues that are zero or very small to avoid dividing by zero. Negligible eigenvalues result in the Expectivity index dropping to zero.

$$Expectivity = \begin{cases} \left(\sqrt{\sum \frac{1}{\lambda_i}}\right)^{-1} & \lambda_i > 10^{-5} \\ 0 & \lambda_i < 10^{-5} \end{cases} \qquad \textbf{4.3}$$

Other than the a priori PCA measures, the selection strategies Area$(\vec{\lambda}_6^1, \vec{\lambda}_6^2)$, Add$(\vec{\lambda}_6^1, \vec{\lambda}_6^2)$ and $\vec{\lambda}_6^{1+2}$ all provide interesting results, as seen in Figure 68 and Figure 69 for Window 9 and Window 14, respectively. The Area$(\lambda_6^1, \lambda_6^2)$ strategy resulted in very small values, to display it alongside the other strategies, it was scaled by

$$scaled\ strategy = \ strategy/\max\,(strategy) \times axis\ limit$$

This results in the strategy spanning the full range of the left Y-axis and makes the identification of peaks much easier.

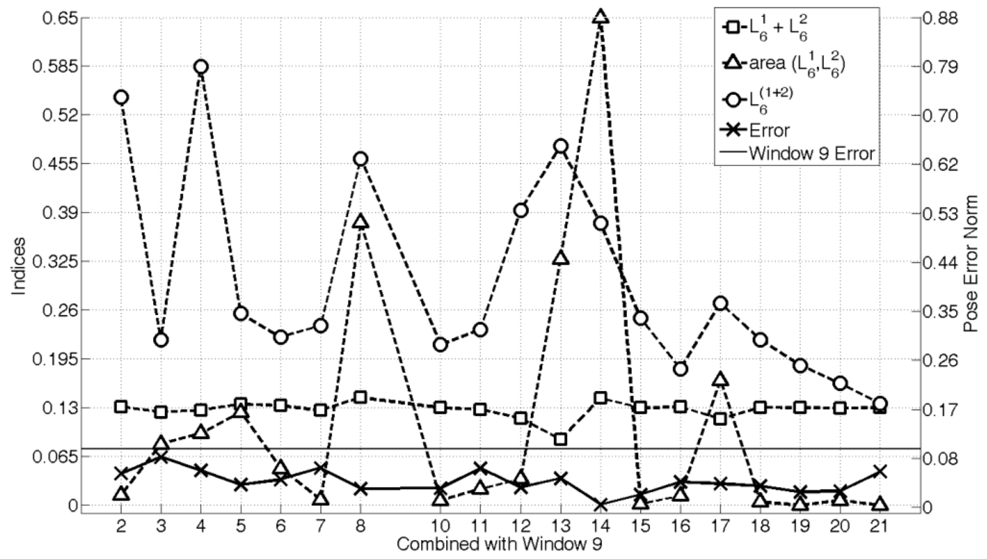


Figure 68 - Window 9 results

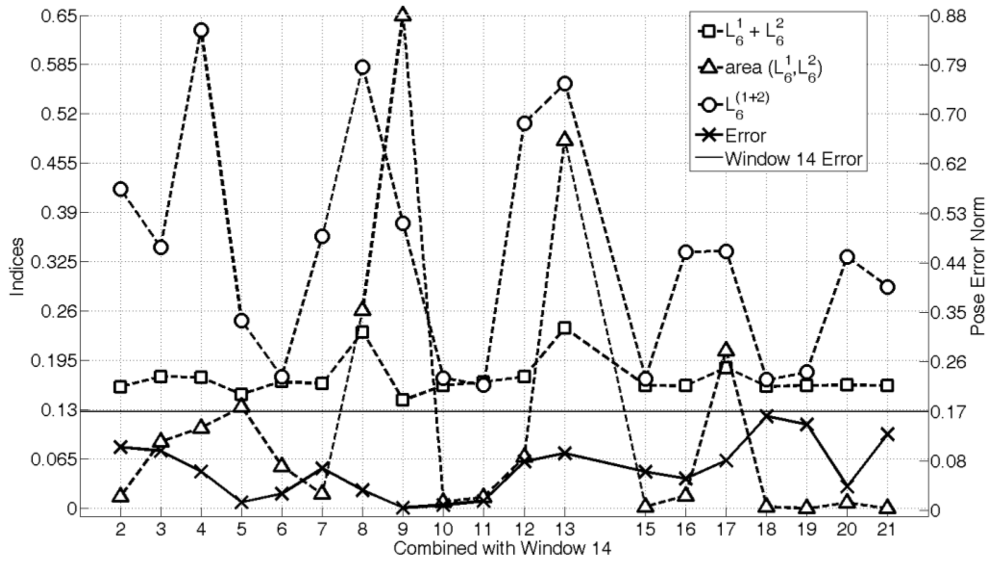


Figure 69 - Window 14 results

A few of the selection strategies described above have been summarized in Table 8. To rank each strategy, the selected window's pose error was used. From Figure 56, Window 9 and Window 14 provided the lowest pose error of 0.0052, and was selected by several strategies. Some strategies selected window combinations with larger, undesirable errors. When entire view was registered using the complete point cloud, the ICP algorithm gave a pose error of 0.0022, which is comparable to the best window combination results.

Table 8 - Results of selection strategies for View 3

Strategy	Cloud Number	Pose Error	Overall Rank
a priori Expectivity	118	0.0052	1
a priori normalized NAI	118	0.0052	1
Area $(\lambda_6^1, \lambda_6^2)$	118	0.0052	1
a priori normalized λ_6^1	118	0.0052	1
Area $(\lambda_1^1, \lambda_1^2)$	43	0.0669	5
a posteriori normalized Expectivity	47	0.0683	6
a posteriori normalized NAI	47	0.0683	6
a posteriori normalized λ_6^{1+2}	47	0.0683	6
Addition $(\lambda_1^1, \lambda_1^2)$	51	0.0730	10
Addition $(\lambda_6^1, \lambda_6^2)$	156	0.1024	12

From Figure 68 and Figure 69, every combination of window gave some form of improvement. However, this is not always the case. A window can be classified as being well, average or poorly constrained based on the resulting pose error. It is beneficial if there is an improvement in pose error regardless of the two window classifications. It is expected that any combination including a well constrained window will give better results. However, if an average window and a poor window, or two poor windows are combined, then there should also be an improvement. This was seen with by checking the results of combination of Windows 3, 14 and 19, as seen in Figure 70. Each dashed line represents a Window's error when combined with the window on the X-axis; the horizontal lines represent the individual window's error.

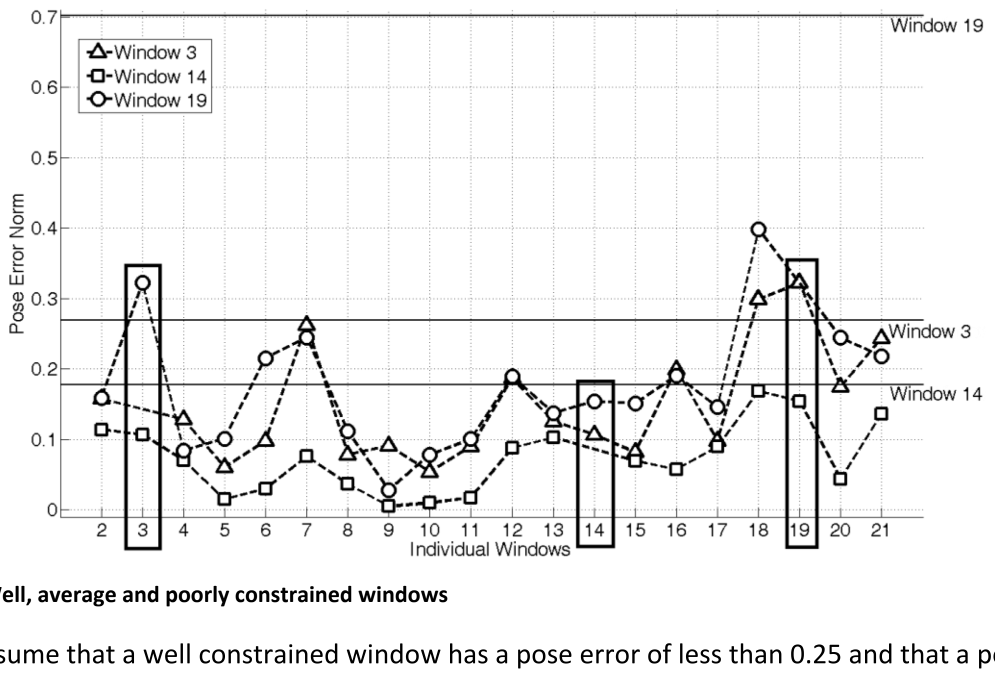


Figure 70 - Well, average and poorly constrained windows

Assume that a well constrained window has a pose error of less than 0.25 and that a poorly constrained window has a pose error greater than 0.50; then the pose error of an average window will be between 0.25 and 0.50. From Figure 70, Window 14 is considered to be a well constrained window , Window 3 an average window and Window 19 a poorly constrained window.

The pose error decreased when Window 14 or Window 19 was combined with either a well constrained, average or poorly constrained window. When Window 3 was combined with a well constrained window, the pose error dropped. However, when Window 3 was combined with a poorly constrained window, the pose error increased slightly.

Of all the windows from Figure 56, Windows 7, 19 and 21 had the highest individual pose errors of 0.71, 0.70 and 0.89, respectively, and are designated as poorly constrained or bad windows. When Windows 19 and 7, and Windows 19 and 21 were combined with each other, the error dropped to 0.24 and 0.22, respectively. This shows a marked improvement on the individual window results and Figure 70 confirms that a combination of a poor-poor, poor-average or poor-well constrained windows will improve with respect to the poorly constrained window. The flip-side of this is that combining with a poorly constrained window can sometimes be detrimental with respect to the other window.

In general, this indicates that regardless of the windows classification, at least one window will find improvement. Of the 190 cases, only 16 combined windows caused an increase in the pose error with respect to one of the windows. The relative rise in pose error for these detrimental combinations is small; on average, the increase was 0.01208. There have not been any cases where combining windows was detrimental to both windows.

4.3.4 Increasing Point Cloud Size

While adding more points via a higher scan density can help registration by introducing extra edge constraint and giving the ICP algorithm more data to smooth out the noise, this effect is not as apparent in windows. When the Shuttle, Figure 56, was registered, the average pose error norm was 0.3191 with an average of 94 points in each window. The average number of points was increased by using a much higher scan density so that the average number of points in each window was 2333. This caused the pose error to drop to an average of 0.2586. Figure 71 shows a comparison of each window's pose error and several different scan densities. Except for Window 21, the performance of a low density scan is comparable to that of a high density scan. The low density scan placed 5 points in Window 21, while the high density scan placed 155 points. A better option would have been to move the scanner target to Window 17 or Window 20 instead.

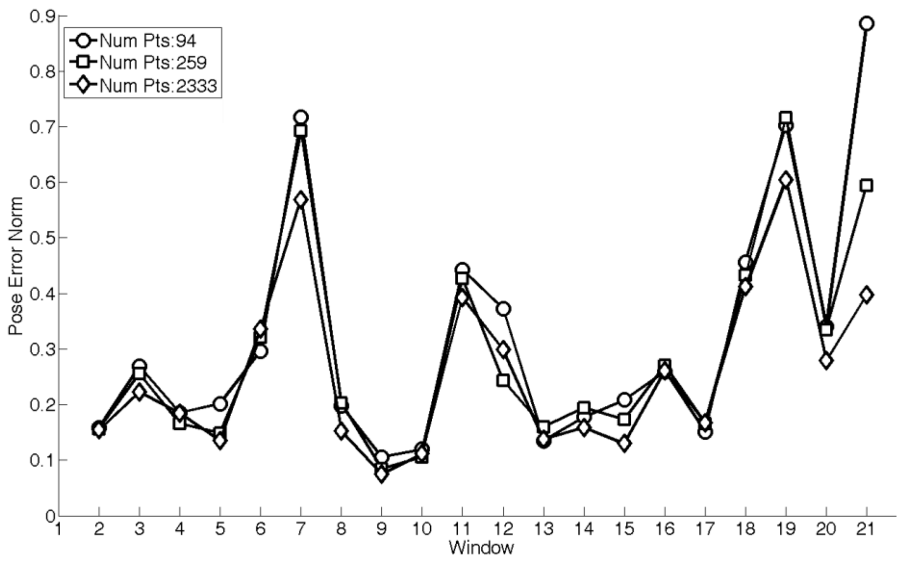


Figure 71 - Shuttle: pose error with varying number of points

Despite increasing the scan density, the pose accuracy never improves as effectively as in combining pairs of windows. This is most likely because increasing the scan density in an area does not increase the types of available constraint. Scanning the same region does not suddenly provide a wider variety of constraining points or normals; if the region being scanned is planar then adding more points will only increase the strength of the constraint in one direction, likely growing λ_1 at the cost of the other eigenvalues. If there are very small features, smaller than the initial scan density, which are registered during a higher density scan, then this might possibly improve the registration accuracy. However, most surface features usually are scanned once or twice at the very least.

Even though a simulated scanner can theoretically apply any scan density, in reality, a LIDAR has a limit to the maximum density. In a real scenario it may not be feasible to increase the scan density any further.

4.3.5 Windowed Cuboctahedron

A real point cloud of a reduced-ambiguity cuboctahedron was taken at the Neptec Design Group using their *TriDar* system. Using this data, the windows were created and registered; Figure 72 shows that the point cloud includes some edge effects which are different from the edge constraint examined in Section 4.2. The edge effects can occur when a LIDAR beam scans a shallow surface which creates noisy points surrounding the edges.

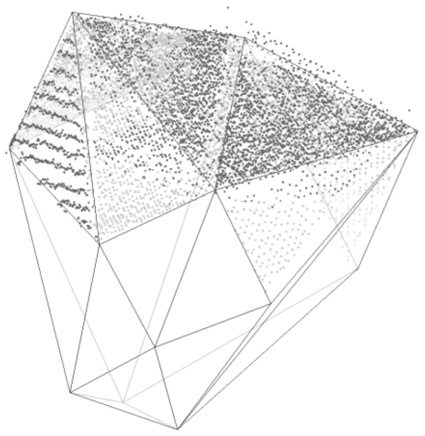
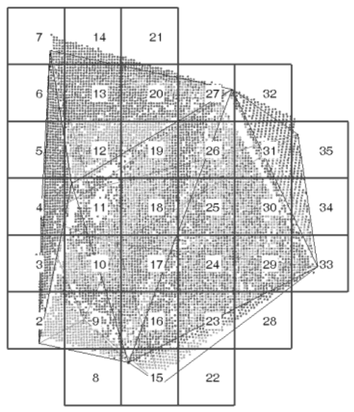



Figure 72 – Cuboctahedron point cloud and windows

The point clouds created by a simulated scan and the real scan of the cuboctahedron were subdivided into windows and analyzed. The normal reconstruction method discussed in Section 3.4.5.5 was used to generate normal vectors. The use of windows magnifies the problem of trying to calculate PCA constraint measures using the reconstructed normals. The cuboctahedron is comprised of several planar surfaces, for which the simulated and theoretical weakest geometric constraint is zero. However, the reconstructed normals are not perfectly parallel and offer a greater variation for the constraint matrix. Therefore, the noisy, real point cloud gives the illusion of having more geometric constraint than is actually available. Figure 73 shows a comparison of normalized NAI between the real data and two sets of simulated data; the first simulated set has the same average number of points in each window, the second simulated set has nearly 3 times the average number of points when compared to the other cases. This is done to show how similar geometric constraint is regardless of the number of points, and the similarity between the real and simulated cases.

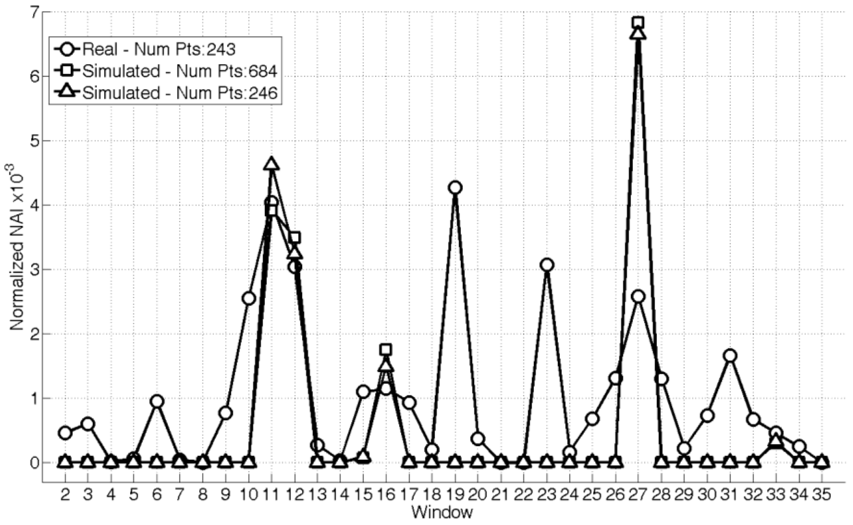


Figure 73 - Cuboctahedron real and simulated NAI

When the real and simulated point windows were registered, the pose error norms in Figure 74 were very close. With the major exception appearing in Window 7, this window targeted a corner of the cuboctahedron with noisy edge effects present.

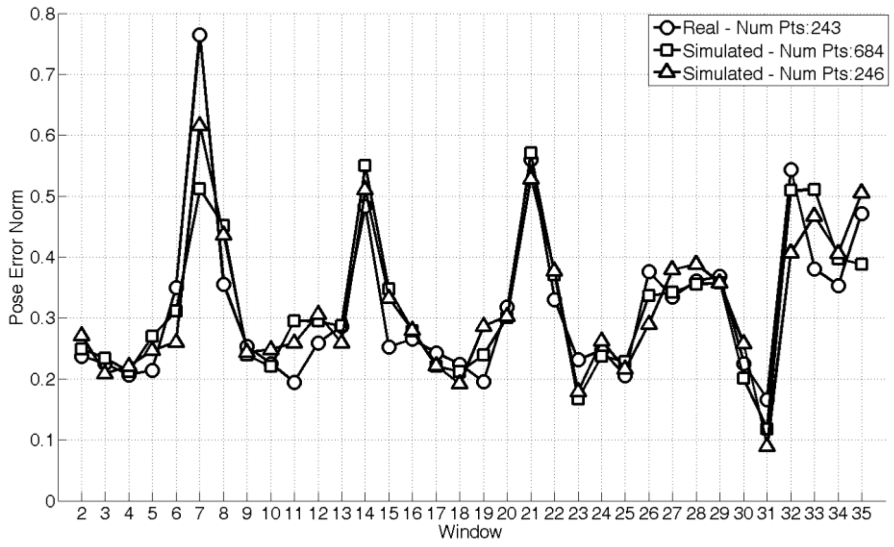


Figure 74 - Cuboctahedron real and simulated pose error

When the selection strategies from Section 4.3.2 was applied to the real cuboctahedron data, the a posteriori normalized Expectivity, λ_6, and the a priori Expectivity strategies, selected Windows 4 and 31. This window combination represented the best registration selected out of all the strategies. However, the combination of Windows 15 and 31 registered the lowest overall error but was not selected by any strategy. The error for window 4 and 31 was on par with window 15 and 31.

4.3.6 Path Planning

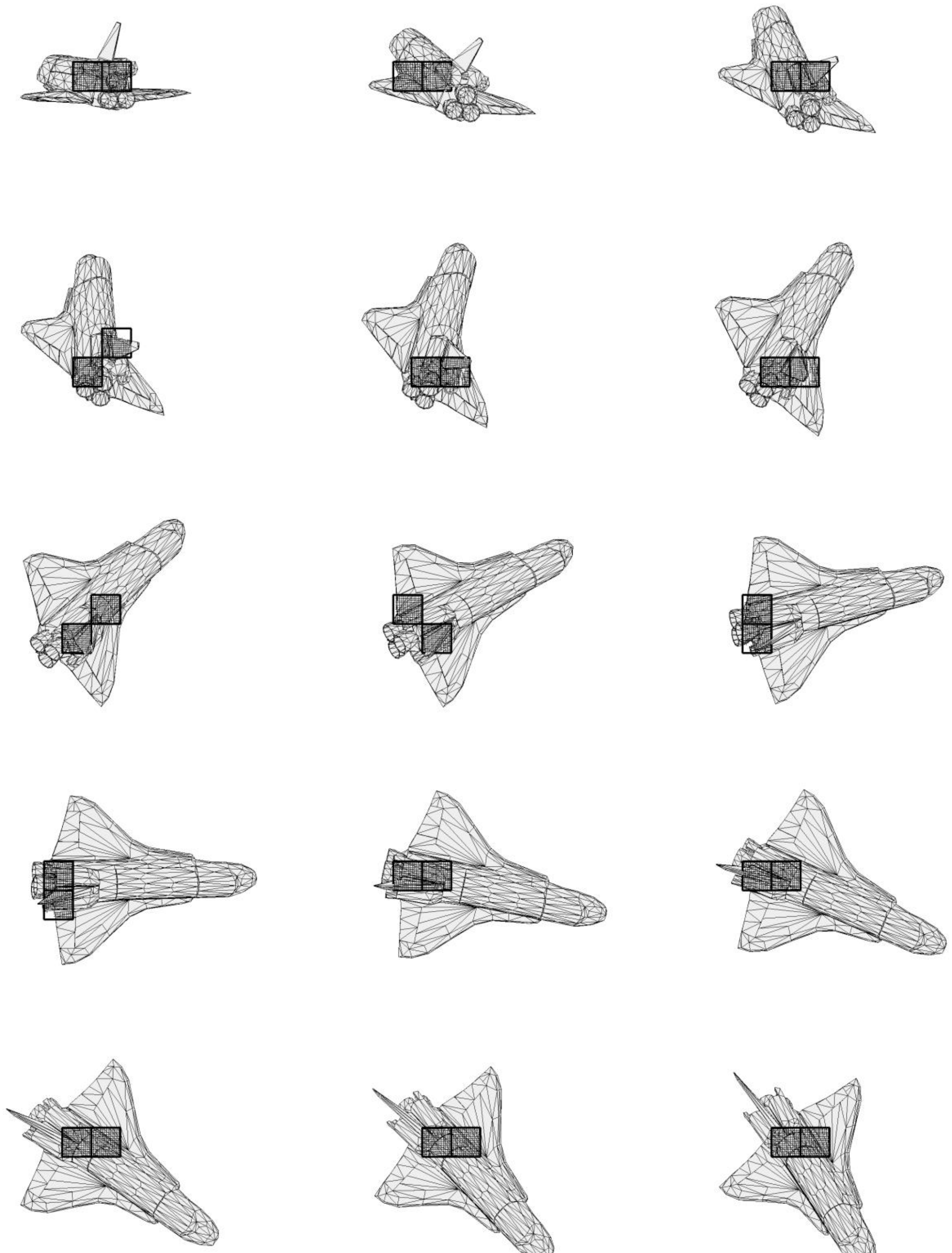
By selecting and combining several windows over a variety of views, a database of window combinations can be formed. Using this database, the best combination of windows can always be identified quickly and effectively. This would allow for quick results with reasonable pose estimation. If, during a scanning process, the current view is known to provide poor registration, then the LIDAR should be moved on a vector that will result in more geometric constraint. To simulate a spinning object or a flyby, multiple views of an object were taken. For each view in the path, a pair of windows was selected by each selection strategy defined in Section 4.3.2. Instead of finding a window to complement another preselected window, the strategies are searching for what it considers to be the top window pairings.

To evaluate the performance of the selection strategies over all the views in the path, a ranking system based on the pose error norm was used. Ideally, a strategy should always pick the window combination with the lowest pose error, which is given the rank of 1. A perfect strategy should sum to the number of views, i.e. always selects the number one ranked pair of windows. Alternatively, the average difference between the lowest error and the window error, or the average error norm of the selected windows may be used. Using either of these methods is preferable to the window ranking as using a rank implies more difference between pose errors that are close in value. The results of the a priori Expectivity and the $\lambda_6^1 + \lambda_6^2$ strategies are shown in Table 9.

Table 9 - Shuttle path selection

		a priori Expectivity			$\lambda_6^1 + \lambda_6^2$		
View	Lowest Error	Pose #	Pose Error	Rank	Pose #	Pose Error	Rank
1	0.0058	55	0.0099	4	80	0.0058	1
2	0.0065	29	0.0468	17	95	0.0137	2
3	0.0052	48	0.0052	1	156	0.1024	105
4	0.0113	3	0.0187	3	134	0.0187	3
5	0.0216	36	0.0845	71	132	0.0845	71
6	0.0095	6	0.0721	67	272	0.0632	49
7	0.0058	25	0.0058	1	89	0.0149	7
8	0.0061	61	0.0083	3	190	0.0381	22
9	0.0058	9	0.0091	3	104	0.0091	3
10	0.0063	9	0.0372	22	120	0.0372	22
11	0.0332	41	0.0389	3	191	0.0747	45
12	0.0280	32	0.0742	66	176	0.0742	66
13	0.0091	107	0.0651	34	137	0.0733	58
14	0.0098	62	0.0727	56	299	0.0985	143
15	0.0110	51	0.0582	34	242	0.0582	34
16	0.0071	107	0.0631	35	263	0.0695	50
17	0.0087	68	0.0582	38	142	0.0838	70
18	0.0070	25	0.1080	77	71	0.1080	77
		average	0.0464	29.72		0.0571	46

The windows selected using the a priori Expectivity strategy are shown in Figure 75. The strategy often selects a window near where the tail fin meets the engines; this is intuitively the area with the greatest geometric constraint. A smart LIDAR scanner may choose to focus upon this area, and then concentrate upon another area by using a database of predetermined combinations. If one window performs consistently well over a range of motion, then focus should be put on always selecting that window and then another. This likely means that the window has a wide variety of constraining factors.



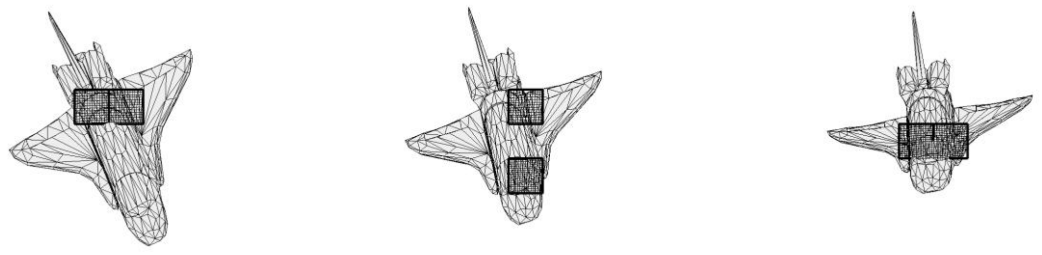


Figure 75 - Shuttle path

A similar study was performed on the ISS airlock model and on the Quicksat satellite; the results of these are shown alongside the results of Shuttle in Table 10. In Table 10, *Pri* refers to the a priori and *Pos* refers to the a posteriori cases.

Table 10 - Results of window selection

		Shuttle		Airlock		Quicksat		Overall	
	Algorithm	Mean Error	Rank	Mean Error	Rank	Mean Error	Rank	Total	Rank
1	Pos Norm NAI	0.0631	11	0.0180	13	0.0205	4	0.1016	11
2	Pos NAI	0.1968	19	0.0772	16	0.1850	19	0.4590	18
3	Pos Norm Exp	0.0846	12	0.0090	4	0.0264	11	0.1199	12
4	Pos Exp	0.0596	7	0.0097	8	0.0238	8	0.0930	8
5	area$(\lambda_1^1, \lambda_1^2)$	0.1057	14	0.0237	14	0.0399	14	0.1693	14
6	Pri NAI	0.0556	4	0.0091	6	0.0100	2	0.0748	1
7	Pri Exp	0.0464	1	0.0097	9	0.0208	5	0.0769	2
8	Pri Norm NAI	0.0480	2	0.0091	5	0.0249	10	0.0819	4
9	Pri Norm Exp	0.0596	8	0.0093	7	0.0244	9	0.0932	9
10	$(v_6^1 + v_1^2)$	0.0920	13	0.0082	1	0.0566	15	0.1568	13
11	$(\lambda_1^1 + \lambda_1^2)$	0.1189	15	0.0246	15	0.1538	17	0.2973	15
12	Area$(\lambda_6^1, \lambda_6^2)$	0.0598	10	0.0089	3	0.0084	1	0.0771	3
13	Pri Norm (λ_6)	0.0577	6	0.0086	2	0.0312	12	0.0975	10
14	Pri Norm (λ_1)	0.4076	20	0.3183	20	0.2821	20	1.0079	20
15	Pos Norm(λ_6^{1+2})	0.0596	9	0.0134	12	0.0169	3	0.0899	6
16	$(\lambda_6^1 + \lambda_6^2)$	0.0571	5	0.0126	11	0.0223	6	0.0920	7
17	dot$(\lambda_1^1, \lambda_1^2)$	0.1271	16	0.1173	17	0.0744	16	0.3187	16
18	dot$(\lambda_6^1 + \lambda_1^2)$	0.1405	17	0.1646	18	0.0387	13	0.3439	17
19	dot$(\lambda_6^1 + \lambda_6^2)$	0.1547	18	0.2145	19	0.1576	18	0.5268	19
20	Pri (λ_6)	0.0554	3	0.0102	10	0.0223	6	0.0878	5

The following strategies performed the best when compared using the window ranks and error differences from Table 10:

- A priori NAI
- A priori EI
- addition $\lambda_6^1 + \lambda_6^2$
- area $(\lambda_6^1, \lambda_6^2)$
- A posteriori ME λ_6^{1+2}
- A priori normalized NAI
- A priori ME
- A posteriori EI

Figure 76 shows the best window combinations and view for the ISS Airlock and Quicksat models. The strategy area $(\lambda_6^1, \lambda_6^2)$ made the selection for ISS Airlock; while both the a prior NAI and a prior EI made the selection for Quicksat. The selections in Figure 76 consider only one view and not the entire path.

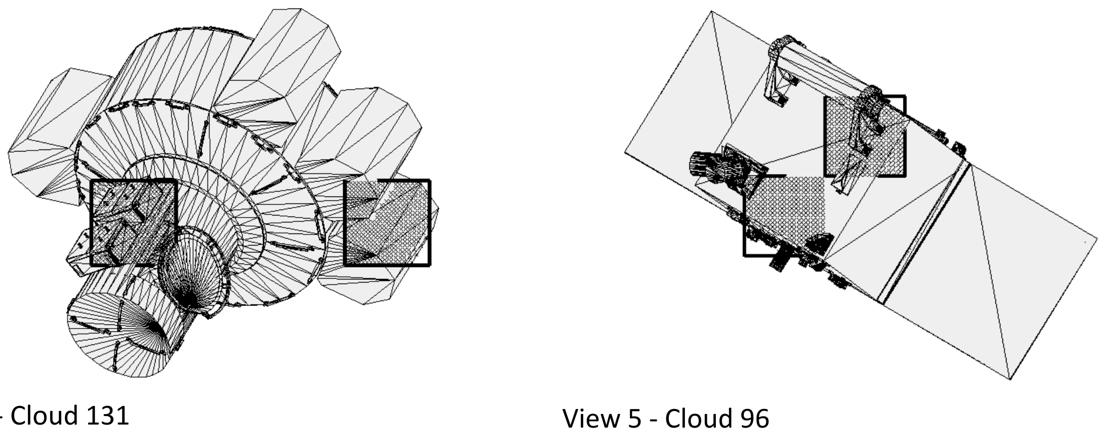


Figure 76 - Airlock and Quicksat optimal windows and views

4.3.7 Quicksat Windows and Path Planning

A 6-view set of real point clouds was taken at the Canadian Space Agency using their Quicksat replica and Neptec's *Laser Camera System* (LCS). Using this data, a set of views were established and the windows were created and registered. Unfortunately, the scans were not complete raster scans as seen in Figure 77; there is a considerable amount of the model which was not scanned. This poses a problem as this makes it difficult to compare the real and simulated geometric constraint in each window. This will also make the ICP algorithm react differently due to the point cloud distribution and the features covered. Figure 77 shows both the simulated and real scans.

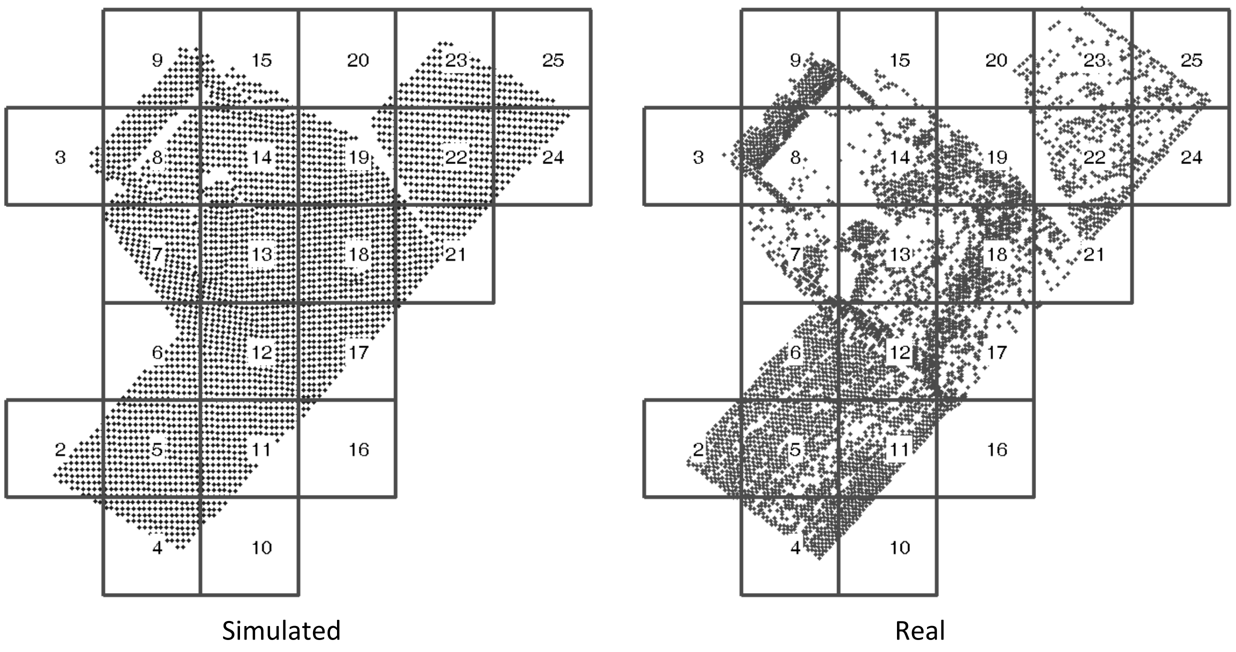


Figure 77 - Quicksat simulated and real scans

As this is a real scan, the true pose itself is subject to some error. For this reason, a real point cloud fitted onto a CAD model does not always completely align. This was the case for the cuboctahedron point cloud as well.

Based on the real data, the results of the selection strategies are shown in the Table 11. The results of the selection strategies for the simulated Quicksat in Table 10 have been repeated for comparison purposes. To tie together the simulated and real results, real windows were selected using based on the simulation results. This was done by ensuring that the real windows were positioned at roughly the same location as the simulated windows.

From Table 11, the top three performing strategies for the real Quicksat were $(v_6^1 + v_1^2)$, $(\lambda_6^1 + \lambda_6^2)$ and a priori NAI. The top three performing strategies for the simulated Quicksat were Area$(\lambda_6^1, \lambda_6^2)$, a priori NAI and normalized (λ_6^{1+2}). The simulation selected real windows gave Norm(λ_6^{1+2}), Area$(\lambda_6^1, \lambda_6^2)$ and a posteriori Expectivity as the top three strategies. This shows that using the simulation selections provided very good windows for the real data to be registered with. It is likely that because of the incomplete raster scans, that some of the results did not translate from the simulated case to the real case. The incomplete raster scans caused changes in the geometric constraint matrix and features to be missed.

Table 11 - Results of window selection for real Quicksat

		Quicksat Real		Simulation Selected		Quicksat Simulated	
	Algorithm	Mean Error	Rank	Mean Error	Rank	Mean Error	Rank
1	Pos Norm NAI	0.1652	8	0.1416	7	0.0205	4
2	Pos NAI	0.4009	18	0.4009	19	0.1850	19
3	Pos Norm Exp	0.1774	10	0.1383	5	0.0264	11
4	Pos Exp	0.1513	7	0.1116	3	0.0238	8
5	area$(\lambda_1^1, \lambda_1^2)$	0.1315	5	0.1481	8	0.0399	14
6	Pri NAI	0.1294	4	0.1408	6	0.0100	2
7	Pri Exp	0.1777	11	0.1653	10	0.0208	5
8	Pri Norm NAI	0.2198	15	0.2002	14	0.0249	10
9	Pri Norm Exp	0.2029	14	0.1840	13	0.0244	9
10	$(v_6^1 + v_1^2)$	0.1139	1	0.1328	4	0.0566	15
11	$(\lambda_1^1 + \lambda_1^2)$	0.2480	17	0.2350	18	0.1538	17
12	Area$(\lambda_6^1, \lambda_6^2)$	0.1454	6	0.1061	2	0.0084	1
13	Pri Norm (λ_6)	0.1862	13	0.2054	17	0.0312	12
14	Pri Norm (λ_1)	0.7131	20	0.4689	20	0.2821	20
15	Pos Norm (λ_6^{1+2})	0.1779	12	0.0988	1	0.0169	3
16	$(\lambda_6^1 + \lambda_6^2)$	0.1254	2	0.1694	11	0.0223	6
17	dot$(\lambda_1^1, \lambda_1^2)$	0.6929	19	0.2045	16	0.0744	16
18	dot$(\lambda_6^1 + \lambda_1^2)$	0.1744	9	0.1626	9	0.0387	13
19	dot$(\lambda_6^1 + \lambda_6^2)$	0.2280	16	0.2032	15	0.1576	18
20	Pri (λ_1)	0.1257	3	0.1694	11	0.0223	6

With regard to the calculation of pose error, the true pose for the real scan is an estimate. As previously mentioned, there is an uncertainty in the measurement of the true pose which can cause misalignment between the CAD model and the real point cloud. By manually rotating or translating the CAD model, it is possible to find a closer match. Full alignment of the real point cloud can be difficult due to the measurement noise and the edge effects. Despite the misalignment, the relative error rankings can still be used to evaluate the windows and combination results.

Figure 78 shows the real scans with the window selected by the simulated Area$(\lambda_6^1, \lambda_6^2)$. Meanwhile Figure 79 shows the real scans with the windows selected by the a priori NAI. Both of these strategies performed well under the real, the simulation and the simulation selected cases.

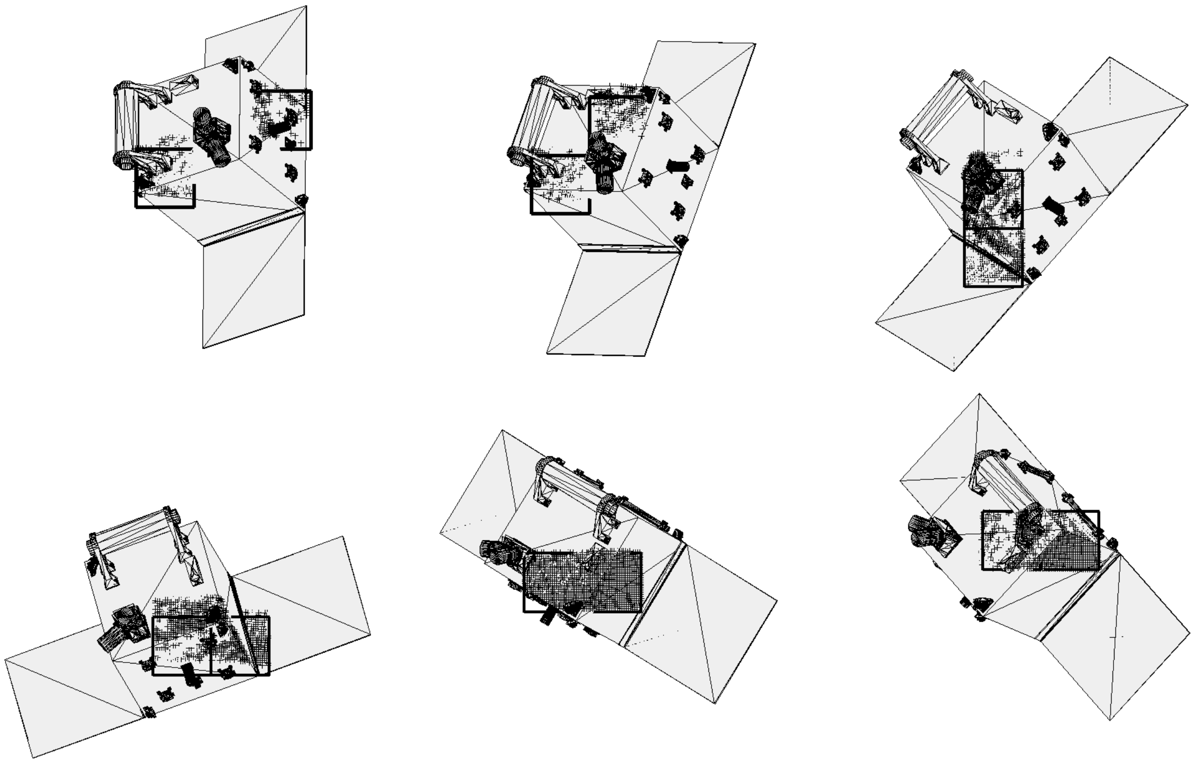


Figure 78 - Real Quicksat window selections made by simulated Area(λ_6^1, λ_6^2)

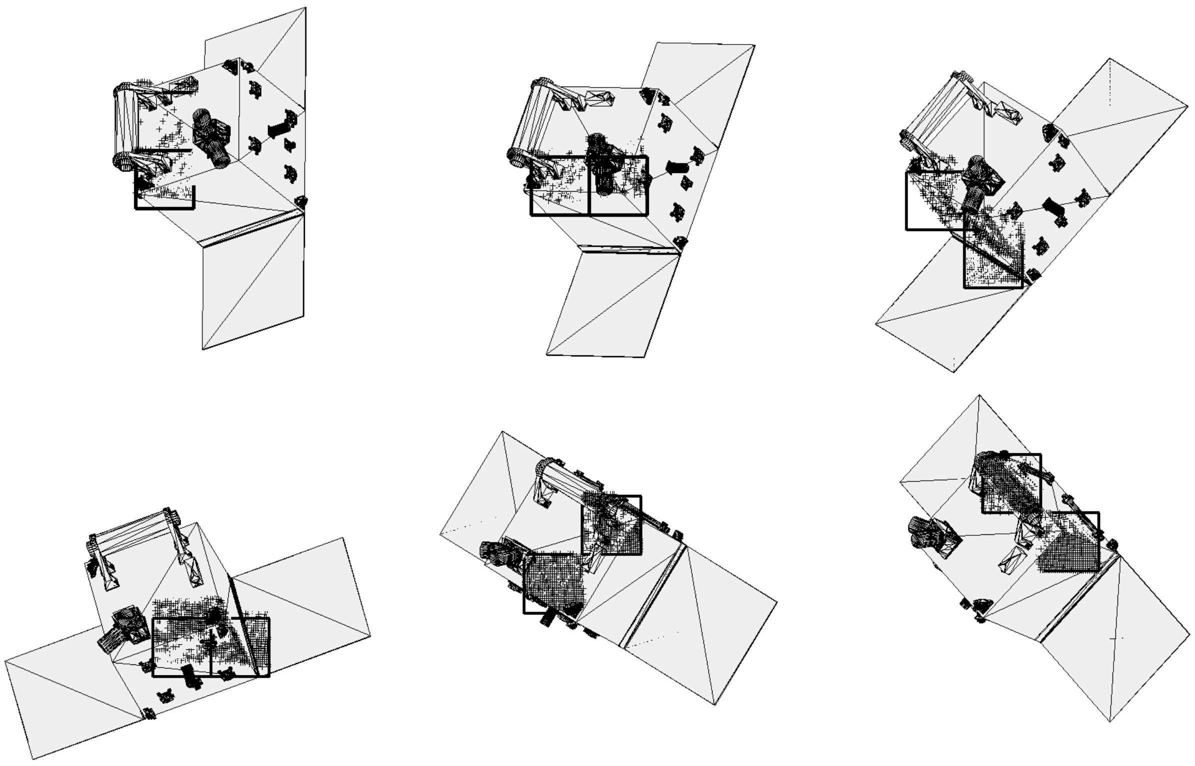


Figure 79 - Real Quicksat window selections made by simulated a priori NAI

Both the selection strategies Area$(\lambda_6^1, \lambda_6^2)$ and a priori NAI perform consistently well for all models and views. As seen in Figure 78 and Figure 79, they also target the same windows. The common property with the two strategies is that they are directly tied to λ_6. While NAI measures the value of the minimum eigenvalue, the area strategy tries to take into account the eigenvectors as well. From the development of the geometric constraint indices in Section 3.4.5.3, when the ME is larger, any movement in the direction of minimum constraint, v_6, will cause a larger error. When more error cost is detected, it is dissipated by the ICP algorithm rotating and translating to minimize it.

A database of window locations is advantageous for real applications as the window selection strategies have been precalculated and the normals do not have to be reconstructed. If the current view of the target is known to provide poor ICP registration, then the camera should be moved to another location where better ICP registration is possible. Knowing which views to target or avoid can be applied to planning the path of the camera.

4.3.8 Conclusion

While the use of one window area provides a rough pose estimation, the use of two windows is much better. Identifying two feature rich locations or complementary windows can lead to results comparable to when the entire point cloud is used. Several strategies have been developed for selecting a complementary window to a preselected first window. These strategies have been extended to finding the top performing combination of windows over a range of poses. The strategies combining windows based on the individual window non-normalized NAI values or the area defined by the minimum eigenvalues have performed consistently well. The database of window locations generated from simulation results is shown to provide accurate results for real scan data. These results have been verified using real point cloud data of a cuboctahedron and the Quicksat satellite.

5 Conclusion

The use of a LIDAR for autonomous rendezvous and docking procedures is vital in space applications. Not only are LIDAR accurate measurement tools, but they are also independent of lighting scenarios. To register a LIDAR point cloud, the Iterative Closest Point algorithm has been shown to be a simple and effective tool. From the ICP algorithm, a pose estimate is given as the rotation and translation relating the point cloud to a CAD model. Capable of operating on the six degrees of freedom, the combination of LIDAR and ICP can be highly effective.

The current view or pose of an object can affect the registration using the ICP algorithm. If the current view is poorly-constrained, then ICP registration can suffer. As shown throughout the thesis, a higher amount of geometric constraint indicates lower ICP pose estimation error. To relate the geometric constraint as a measure to the overall ICP registration error, Principal Component Analysis was performed on a constraint matrix. The constraint matrix was built to relate a point cloud's distribution and normal orientations to geometric constraint. Using the results of PCA, several geometric constraint indices were considered: the Noise Amplification Index, the Minimum Eigenvalue, the Inverse Condition Number and the Expectivity Index. These indices were used to assess the selection of views on several different objects. The InvCond and NAI indices were found to be equally good at predicting the pose error for the individual degrees of freedom in the system.

By simulating a LIDAR, several simulated point clouds were created. In addition to the simulated data, real point clouds were provided by Neptec Design Group and by the Canadian Space Agency. By comparing the results of ICP pose estimation of the real and simulated scans, it is shown that simulation provides a strong indication of how the ICP algorithm will react with real data.

In addition to the geometric constraint in the point cloud, edge constraint may arise when the LIDAR samples points close to an edge of a surface. If two views with similarly low geometric constraint, the expected result is a high pose estimation error. However, edge constraint can aid in ICP pose estimation and can allow for a much lower error. With a simple pyramid model, it was easily seen that with a point cloud distributed along the surface edges will provide better results than points distributed along the inner surfaces. The effect of edge constraint was also investigated with the Shuttle model.

As a way to potentially lower the amount of data needed to register the pose, the concept of windows have been extended. In a real scenario, as a camera approaches a target, the viewing area will become smaller and more focused upon a specific area. When focusing on smaller areas fewer geometric features are available. Therefore, certain areas should be identified and targeted for the ICP algorithm. While one window has the possibility of providing good results, the use of two windows is much better. The combination of nearly any two windows offers a better result than either of them alone. If a database of good window positions is generated for a range of poses, then it may be applied to path planning. The path of the camera should be planned such that good views for ICP registration are always maintained.

Several different strategies are examined to combine windows together, including measures taken before and measures taken after the window combinations. Of the geometric constraint measures listed above, the selection of the two highest NAI values gave the best results. However, selection strategies which involved the ME and the minimum eigenvector have been also shown to work very well. Combining two scaled eigenvectors, either to find the largest distance or the eigenspace area, very good results have been achieved.

As a side effect of the previous investigations, it was found that increasing the scan density above a certain threshold did not reduce pose errors. This was seen when entire views or when smaller window areas were compared. The PCA indices increase with the number of points, however, once normalized, there is little difference between high and low density scans.

Future considerations can include simulating the scanning of windows in a real test. While splitting a real point cloud is viable, it may not recreate all effects of scanning small, targeted areas. Although this work dealt with the combination of two windows, the combination of more windows is also possible. However, the number of window combinations grows very large, selecting even three windows results in approximately 8 to 10 times the number of possible combinations. In addition to performing real window scanning, it would be beneficial to see if a LIDAR can be directed by the simulated ICP registration results. The use of lissajous or rosette scan patterns should also be investigated for windows.

While the minimum eigenvalue has been selected was often used as part of the main selection metric, the second smallest eigenvalue is sometimes of the same magnitude. This means that a point cloud is actually weakly constrained in two directions as opposed to just one. By correcting λ_6, v_6 only, no consideration is applied to λ_5, v_5; new selection strategies may take λ_5, v_5 into account to resolve all unconstrained directions.

6 References

Arun, K. S., Huang, T. S., & Blostein, S. D. (1987). Least-squares fitting of two 3-D point sets. *IEEE Trans. Pattern Anal. Mach. Intell. , 9* (5), 698-700.

Becker, B., & Ortiz, E. (2009). Evaluation of face recognition techniques for application to facebook. *8th IEEE International Conference on Automatic Face & Gesture Recognition.* IEEE.

Beraldin, J.-A., Blais, F., & El-Hakim, S. (1995). Performance evaluation of three active vision systems built at the National Research Council of Canada. *Proc. Optical 3D Measurement Techniques III.*

Besl, P., & McKay, N. (1992). A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence , 14*, 239-256.

Billon, R., Nédélec, A., & Tisseau, J. (2008). Gesture recognition in flow based on PCA analysis using multiagent system. *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, (pp. 139-146).

Birk, H., Moeslund, T., & Madsen, C. (1997). Real-time recognition of hand Alphabet gestures using Principal Component Analysis. *In Proceedings of The 10th Scandinavian Conference on Image Analysis.*

Blais, F., Beraldin, J.-A., & El-Hakim, S. (2000). Range Error Analysis of an Integrated Time-of-Flight, Triangulation, and Photogrammetry 3D Laser Scanning System. *Society of Photo-Optical Instrumentation Engineers.*

Braido, P., & Zhang, X. (2004). Quantitative analysis of finger motion coordination in hand manipulative and gestic acts. *Human Movement Science , 22* (6), 661-678.

Chen, Y., & Medioni, G. (1992). Object Modeling by Registration of Multiple Range Images. *Int. J. of Image and Vision Computing* (10), 145-155.

Draper, B., Baek, K., Bartlett, M., & Beveridge, J. (2003). Recognizing faces with PCA and ICA. *Computer Vision and Image Understanding , 91* (1-2), 115-137.

Ellzey, M. L., Kreinovich, V., & Peña, J. (1993). Fast rotation of a 3D image about an arbitrary line. *Computers & Graphics , 17* (2), 121-126.

Ericson, C. (2004). *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology).* Morgan Kaufmann Publishers Inc.

Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Trans. Math. Softw. , 3* (3), 209-226.

Friend, R. (2008). Orbital express program summary and mission overview. *Proceedings of SPIE - The International Society for Optical Engineering.* SPIE.

Gelfand, N., & Rusinkiewicz, S. (2003). Geometrically Stable Sampling for the ICP Algorithm. *Proc. International Conference on 3D Digital Imaging and Modeling*, (pp. 260-267).

Godin, G., Rioux, M., & Baribeau, R. (1994). Three-dimensional registration using range and intensity information. *Videometrics III. 2350*, pp. 279-290. SPIE.

Greenspan, M., & Godin, G. (2001). A Nearest Neighbor Method for Efficient ICP. *3D Digital Imaging and Modeling, International Conference on , 0*, 161.

Greenspan, M., & Yurick, M. (2003). Approximate K-D Tree Search for Efficient ICP. *3D Digital Imaging and Modeling, International Conference on , 0*, 442.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., & Stuetzle, W. (1992). Surface reconstruction from unorganized points. *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques* (pp. 71-78). ACM.

Horn, B. K. (1987). Closed-Form Solution of Absolute Orientation using Unit Quaternions. *Journal of the Optical Society of America , 4* (4), 629-642.

Horn, B. K., Hilden, H., & Negahdaripour, S. (1988). Closed-Form Solution of Absolute Orientation using Orthonormal Matrices. *Journal of the Optical Society of America , 5* (7), 1127-1135.

Jeong, D. H., Ribarsky, W., & Chang, R. (2009). Designing a PCA-based Collaborative Visual Analytics System. *IEEE Visualization Workshop on Collaborative Visualization 2009.*

Jolliffe, I. T. (2002). *Principal Component Analysis* (Second ed.). Springer-Verlag.

Jost, T., & Hügli, H. (2003). A Multi-Resolution ICP with Heuristic Closest Point Search for Fast and Robust 3D Registration of Range Images. *3D Digital Imaging and Modeling, International Conference on , 0*, 427.

Ling, A. C., Singh, D. P., & Brown, S. D. (2007). Incremental placement for structured ASICs using the transportation problem. *VLSI-SoC*, (pp. 172-177).

Masuda, T. (2002). Registration and integration of multiple range images by matching signed distance fields for object shape modeling. *Comput. Vis. Image Underst. , 87* (1-3), 51-65.

McTavish, D., Okouneva, G., & Choudhuri, A. (2009). CSCA-Based Expectivity Indices for LIDAR-based Computer Vision. *Mathematical Methods and Applied Computing , 1*, 54-62.

McTavish, D., Okouneva, G., & English, C. (2010). Continuum Shape Constraint Analysis: A class of Integral Shape Properties Applicable for LIDAR/ICP-based Pose Estimation. *To appear in International Journal of Shape Modeling* .

McTavish, D., Okouneva, G., & Okounev, O. (2009). Selection of Regions on a 3D surface for Efficient LIDAR-based Pose Estimation. *Vision, Modeling, and Visualization Workshop*, (pp. 387-389). Germany.

Menq, C.-H., Yau, H.-T., & Lai, G.-Y. (1992). Automated precision measurement of surface profile in CAD-directed inspection. *IEEE Transactions on Robotics and Automation , 8* (2), 268-278.

Moler, C. (2004). *Numerical Computing with MATLAB.*

Moon, H., & Phillips, P. (2001). Computational and performance aspects of PCA-based face-recognition algorithms. *Perception , 30* (3), 303-320.

Mount, D., & Arya, S. (2010). *ANN Programming Manual* (1.1 ed.).

Murase, H., & Nayar, S. (1995). Visual learning and recognition of 3-d objects from appearance. *International Journal of Computer Vision , 14* (1), 5-24.

Nahvi, A., & Hollerbach, J. M. (1996). The noise amplification index for optimal pose selection in robot calibration. *Proc. IEEE Intl. Conf. Robotics and Automation*, (pp. 22-28).

NASA. (2010). *NASA - 3D Resources*. Retrieved June 6, 2010, from http://www.nasa.gov/multimedia/3d_resources/

Ohba, K., & Ikeuchi, K. (1997). Detectability, uniqueness, and reliability of eigen windows for stable verification of partially occluded objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence , 19* (9), 1043-1048.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine , 2* (6), 559-572.

Ruel, S., English, C., Anctil, M., & Church, P. (2005). 3DLASSO: real-time pose estimation from 3D data for autonomous satellite servicing. *International Symposium on Artificial Intelligence for Robotics and Automation in Space.* Neptec.

Rusinkiewicz, S., & Levoy, M. (2001). Efficient Variants of the ICP Algorithm. *Proceedings of the Third Intl. Conf. on 3D Digital Imaging and Modeling*, (pp. 145-152).

Shahid, K. (2007). *Intelligent LIDAR scanning region selection for satellite pose estimation.* Ryerson University. Elsevier Science Inc.

Shahid, K., & Okouneva, G. (2007). Intelligent LIDAR Scanning Region Selection for Satellite Pose Estimation. *Computer Vision and Image Understanding* , 203-209.

Shoemake, K. (1985). Animating rotation with quaternion curves. *SIGGRAPH Comput. Graph. , 19* (3), 245-254.

Simon, D. A. (1996). *Fast and accurate shape-based registration.* Carnegie Mellon University.

Strang, G. (1976). *Linear Algebra and its Applications.* New. York: Academic Press.

The Celestia Motherlode. (2010). *The Celestia Motherlode: Satellites*. Retrieved June 6, 2010, from http://www.celestiamotherlode.net/catalog/satellites.php

Turk, M., & Pentland, A. (1991). Eigenfaces for recognition. *Journal of Cognitive Neuroscience , 3* (1), 71-86.

United Nations. (1999). *Technical Report on Space Debris.* New York: United Nations.

Weik, S. (1997). Registration of 3-D partial surface models using luminance and depth information. *NRC '97: Proceedings of the International Conference on Recent Advances in 3-D Digital Imaging and Modeling* (p. 93). IEEE Computer Society.

Wu, H., & Sutherland, A. (2001). Dynamic gesture recognition using PCA with multiscale theory and HMM. *SPIE International Symposium on Multispectral Image Processing and Pattern Recognition*, *4550*, pp. 132-139.

Zhang, Z. (1994). Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision , 13* (2), 119-152.