To the Graduate Council:

I am submitting herewith a thesis written by James G. McGreger entitled Data-driven Registration of Laser-range Images. I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Jens Gregor

Dr. Jens Gregor, Major Professor

We have read this thesis
and recommend its acceptance:

Brad Vander Zanden

M. A. Abidi

Accepted for the Council:

Associate Vice Chancellor
and Dean of the Graduate School

DATA-DRIVEN REGISTRATION OF LASER RANGE IMAGES

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

James G. McGreger

December 1998

Acknowledgments

There are many people I have to thank for allowing me the opportunity to further my education. First and foremost is my family: My parents John and Gayle McGreger, my sister and her husband Willie and Tina Schrambeck, and my younger sister Krista McGreger and brother John McGreger II, for giving me all the love, support, and encouragement needed to reach all my academic goals. I would like to thank Dr. J. Gregor and Dr. M. A. Abidi for selecting me for the Graduate Research Assistant Position in Three-Dimensional Scene Modeling that ultimately made obtaining a master's degree possible. A special thank you to my advisor Dr. J. Gregor for the great deal I have learned from him throughout the course of this project, and especially for his patience in guiding me through many uncharted territories. Thanks also to the members of my committee, Dr. J. Gregor, Dr. M. A. Abidi, and Dr. B. Vander Zanden for their willingness to serve.

The work in this thesis was supported by the DOE's University Research Program in Robotics (Universities of Florida, Michigan, New Mexico, Tennessee, and Texas) under grant DOE-DE-FG02-86NE37968.

Abstract

This study focuses on techniques to recover the movement of a laser range finder camera between capturing two different views of the same scene. The two tasks required for determination of this transformation are computing correspondences across the two range images and using these correspondences to compute the motion estimate. Two methods for computing correspondences are the closest point and the point-to-surface which respectively minimize the distance between corresponding points and the distance from a point to a plane; we present a variation of the latter based on the alignment of corresponding planes. Both methods use a least squares technique to recover the motion estimate. The four methods for computing a motion estimate all use virtually the same technique, just with different mathematical representations for the transformation. The effectiveness of these view registration techniques in recovering accurate transformations is tested on ideal synthetic data images as well as synthetic images with zero mean Gaussian noise added. The results of the experiments suggest that these methods are used primarily for fine tuning a motion estimate and not for recovering large motions. Therefore a relatively accurate initial estimate of the transformation must be given as input. The results yielded are highly dependent on the device used to capture the range images and its setup, as in size of the image, field of view of the camera, and the distance from the camera to the scene. The experiments show that both methods are able to recover transformations with acceptable accuracy given a close initial estimate.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

The purpose of this research is to facilitate the construction of a three-dimensional (3D) volumetric model of an unknown environment from data acquired by a laser range finder (LRF) on board a mobile robot. Numerous real world applications exist including object recognition, visual navigation, 3D reconstruction and/or update of architectural or industrial plans into a computer-aided design (CAD) model, and estimating the position and orientation of a recognized object (pose estimation). The data acquired by the LRF is in the form of a range image which contains distances from points in a scene to the viewer and thus provides implicit information about the geometry of the scene. Range images are used in various applications such as determining the geometric integrity of manufactured parts and measuring their precise dimensions.

Multiple views of a scene are required to construct a 3D model since parts of the

scene will not be visible in any one particular view. Three primary steps are required for scene modeling: (1) data acquisition, (2) view registration, and (3) view integration. Regarding data acquisition, the LRF gathers range image information by making a two-dimensional (2D) scan of the object scene. The depth r_{ij} measured at each position is stored in a 2D array that is indexed by the number of passes the LRF makes over the scene and the number of positions measured during each pass; the array indices correspond to the particular ray in the field of view of the LRF which sampled the given point. The 3D coordinate (x, y, z), defined relative to the coordinate system of the LRF, is computed from the (i, j, r_{ij}) tuple. The point sets acquired in this manner are then used as input to the registration process.

The purpose of the view registration is to determine the camera motion (a rigid transformation given by rotation R and translation t) between two or more successive views. There exists a number of possible methods such as those based on the raw data points, features such as line segments, and surface curves and regions. View integration is the process of merging multiple views of a scene into one 3D scene model.

The data acquisition and view integration into a 3D scene model are topics unto themselves and are not discussed further. This paper focuses on those methods for view registration that use point based techniques to recover the transformation between two views.

1.2 Iterative Closest Point (ICP)

1.2.1 Algorithm Outline

The ICP algorithm was originally developed by Besl and McKay [2] with slightly different versions independently developed by Zhang [12] and Chen and Medioni [4]. The ICP algorithm is a general purpose mechanism for the registration of two 3D range images. The two fundamental steps of the algorithm are: (1) establishing point correspondences across range images, and (2) using the established correspondences to compute the transformation between the given images. This process is iterated until the transformation recovered converges to a minimum.

The ICP algorithm requires that an initial estimate of the transformation between the two images is given. The initial estimate is required because the ICP algorithm is primarily used for fine tuning a motion estimate and not for computing estimates of large motion. This initial transformation is applied to the data points in the first image, yielding an intermediate image that is in relatively close alignment with the second image. The next step is to find the correspondences between the intermediate image and the second image. There are numerous methods discussed in the literature for computing correspondences across range images. Once these temporary correspondences have been established, a least squares technique is used to recover the motion estimate that minimizes the distance between the two images. The recovered transformation is now compared with the transformation from the previous iteration, calculating some metric that represents the change in the two transformations to determine when to

halt the iteration. The ICP algorithm is outlined in Figure 1.1. Besl and McKay [2] proved that the ICP algorithm will always converge to a local minimum with their Convergence Theorem. However, whether this local minimum corresponds to the global minimum that is sought is highly dependent on the complexity of the range images and the accuracy of the initial estimate.

1.2.2 Least Squares Formulation

The transformation at each iteration of the ICP algorithm is computed using a least squares technique that minimizes the distance between corresponding points in two images. Given two 3D point sets $P_1 = \{p_{1i}\}$ and $P_2 = \{p_{2i}\}$, $i = 1, 2, ..., N$, that are related by

$$p_{2i} = Rp_{1i} + t + v_i \tag{1.1}$$

where R is a standard 3×3 rotation matrix, t is a 3×1 translation vector and v_i a

obtain initial estimate of the transformation (R^0, t^0)
repeat {
    $I_m = R^{i-1} I_1 + t^{i-1}$
    obtain correspondences between I_m and I_2
    compute (R^i, t^i) from found correspondences
} until ((R^i, t^i) $\approx$ (R^{i-1}, t^{i-1}))

Figure 1.1. Outline of the ICP algorithm. Input is given as two images I_1 and I_2, and an initial estimate (R^0, t^0) of the transformation. I_m represents the intermediate image.

noise vector, the object is to solve for the optimal transformation $[\ \hat{R}, \hat{t}\ ]$ that maps P_1 onto P_2 by minimizing a least squares error criterion given by:

$$\Sigma^2 \ = \ \sum_{i=1}^{N} \ \| \ p_{2i} \ - \ (\ \hat{R} p_{1i} \ + \ \hat{t} \ ) \ \|^2. \tag{1.2}$$

The literature discusses several different approaches to solving Equation (1.2) for $\hat{R}$ and $\hat{t}$. The solution to the least squares minimization problem is complicated by the requirement that R be orthonormal to preserve inner products and thus angles and distances. R is orthonormal if it satisfies $RR^T = R^T R = I$ or equivalently, $R^T = R^{-1}$.

1.2.3 Rotation and Euler Angles

The 3×3 rotation matrix R can be expressed in terms of the three Euler angles θ_x, θ_y, and θ_z which represent the amount of rotation around each of the x, y, and z axes, respectively [9]. The orthonormal matrix R is calculated by

$$R \ = \ R_x \ R_y \ R_z$$

where

$$R_x \ = \ \begin{bmatrix} \cos\theta_x & \sin\theta_x & 0 \\ -\sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

and θ_x denotes the angle of rotation around the x-axis. R_y and R_z are defined similarly to R_x. The resulting orthonormal rotation matrix is:

$$R = \begin{bmatrix} \cos\theta_y \cos\theta_z + \sin\theta_x \sin\theta_y \cos\theta_z & -\cos\theta_y \sin\theta_z + \sin\theta_x \sin\theta_y \cos\theta_z & \sin\theta_y \sin\theta_z + \sin\theta_x \cos\theta_y \cos\theta_z & \sin\theta_y \cos\theta_x \\ \cos\theta_x \sin\theta_z & \cos\theta_x \cos\theta_z & -\sin\theta_x & \\ -\sin\theta_y \cos\theta_z + \sin\theta_x \sin\theta_y \sin\theta_z & -\cos\theta_y \sin\theta_z + \sin\theta_x \sin\theta_y \cos\theta_z & \cos\theta_y \cos\theta_x & \end{bmatrix}. \quad (1.4)$$

The Euler angle representation is just one of several different ways of representing rotation.

1.2.4 Stopping Criteria

A method is needed for determining when the ICP algorithm has converged to a minimum and the iteration can be terminated. One such method is a simple condition where the iteration is terminated after a pre-specified number of iterations. Zhang [12] suggests that this prefixed threshold should be set at 40 for images with surfaces.

Another common approach is to use the relative change in the motion estimate

between two successive iterations. Once this change falls below a preset threshold, τ, the iteration is terminated. The metric used to represent this change is the mean-square error. In order to calculate a more precise termination condition, the mean-square error for the rotation and translation components of the transformation can be calculated separately. The relative difference in the translation, δt, at any given iteration i is defined as

$$\delta t \;=\; \| \; t_i \;-\; t_{i-1} \; \| \;\; / \;\; \| \; t_i \; \|.$$

There are however various opinions on how to calculate the difference in the rotation. Besl and McKay [2] calculate the mean-square error using the individual elements of R, while Zhang [12] uses the rotation-axis representation which is derived from a related quaternion. An alternative approach is to calculate the mean-square error using the angles of rotation around each individual axis. The Euler angles that represent the standard rotation matrix (1.4) can be extracted from R through a simple decomposition of its individual components. The element $R_{23} = -\sin\theta_x$ can be solved for θ_x and subsequently substituted into $R_{13} = \sin\theta_y \cos\theta_x$ and $R_{21} = \cos\theta_x \sin\theta_z$ to recover θ_y and θ_z. The angles of rotation themselves yield a more comprehensible view of the amount of movement that takes place between successive iterations. Hence, letting $r = [\; \theta_x,\; \theta_y,\; \theta_z \;]$, the relative difference in rotation, δr, at any given iteration i is computed as

$$\delta r \;=\; \| \; r_i \;-\; r_{i-1} \; \| \;\; / \;\; \| \; r_i \; \|.$$

The value of τ, the preset threshold for termination, must be carefully chosen as it not only affects the final transformation recovered, but also the amount of processing time needed. If the threshold is set to high, the iteration could halt prematurely, yielding a transformation that is less accurate than desired. On the other hand, if the threshold is set too low, the iteration could continue while not gaining any significant accuracy in the recovered transformation. Zhang [12] recommends that the iteration be terminated when the relative difference in rotation and translation from one iteration to the next are both less than 1%. That is, when

$$\delta t < \tau_t \quad and \quad \delta r < \tau_r$$

where τ_t and τ_r both equal 1%, the ICP algorithm will terminate.

1.3 Computer Information

The programs for the experimentation reported for this research were written in the C programming language and compiled with Gnu compiler gcc version 2.7 using level three optimizations. The system used was a GATEWAY2000 PC running the Red Hat Linux 2.0.35 Operating System. The GATEWAY system has a 233 MHz PentiumII processor and 256 Mbytes of local memory. The Numerical Recipes [8] software package was used for generating random numbers as well as miscellaneous mathematical matrix calculations.

1.4 Outline of Thesis

This paper focuses on several of the available methods for registering 3D range images. Chapters 2 and 3 address the matter of computing correspondences across range images, describing the Closest Point and Point-to-Surface correspondence methods respectively. Chapter 4 introduces three methods for computing the motion estimate between images at each iteration of the ICP algorithm, namely: Singular Value Decomposition, Unit Quaternion, and Dual Number Quaternion. Chapter 5 discusses a new method for computing motion estimation between images given corresponding planes as opposed to corresponding points. Finally, Chapter 6 discusses the experimentation process and the recovered results.

Chapter 2

Point-to-Point Correspondence

The first and perhaps the simplest method for computing point correspondences is the closest point method. It is based on the premise that with two images that are in relatively close alignment, a point in the first image and its corresponding point in the second image should be close together in a Euclidean sense. Given a point in the first image $p_1 = (\ x_1,\ y_1,\ z_1\ )$ and a point in the second image $p_2 = (\ x_2,\ y_2,\ z_2)$, the closest point method uses the Euclidean distance between the 3D points defined as

$$d(\ p_1,\ p_2\ ) \ = \ \sqrt{(x_2 - x_1)^2 \ + \ (y_2 - y_1)^2 \ + \ (z_2 - z_1)^2}$$

to determine the closest point in the second image to the given point in the first image. This method was initially suggested by Besl and McKay [2], with a more comprehensive description given by Zhang [12].

2.1 Brute Force

The most straightforward way of computing closest point correspondences is an exhaustive, brute force search of the second point set. Given two point sets P_1 and P_2, a brute force search computes $d(\ p_{1i},\ p_{2j}\ )$ between each p_{1i} and every point p_{2j} in P_2, keeping only the p_{2j} whose distance to p_{1i} is the smallest. This closest point p_{2j} is then paired with p_{1i} as corresponding points. This method, while simple to understand and code, is computationally very expensive. Letting N_1 and N_2 represent the number of points in P_1 and P_2 respectively, the brute force search computes a closest point for all points in P_1 in $O(N_1N_2)$ time. Considering that N_1 and N_2 can be very large numbers, better search methods are needed to reduce the search time necessary for computing closest points.

2.2 Multidimensional Binary Tree

The multidimensional binary tree (kD tree), discussed in detail by Preparata and Shamos [7], is a method of extending the concept of bisection in one dimension to multiple dimensions. The idea is to store the data points in a binary tree structure in a manner where, upon searching the tree, it is possible to eliminate entire subtrees of points that could not possibly contain the closest point sought. The closer to the root node of the tree a subtree can be removed from consideration, the faster the search will be.

2.2.1 General Algorithms

While the kD tree can be used with dimensions ≥ 2, it is perhaps easier to understand the basic concepts by discussing the 2D case. Given a set P_2 of 2D data points, the first consideration is the construction of the 2D tree. The initial step is to split P_2 into two approximately equal regions and mark the separation with a line segment. The dividing line passes through the midpoint and, depending on which coordinate is used in determining the split, is parallel to either the x or y axis. For example, using the y coordinate, the split is accomplished by computing the median y value over all the data points in P_2. A line segment, passing through the point p_{2j} containing the median y value and parallel to the x axis, divides the 2D space in two with approximately equal number of points on both sides. The points with y coordinates less than the median value are assigned to the left subtree, and those greater than the median value are assigned to the right subtree. The coordinate used to compute the median value is alternated between x and y at each level of the binary tree, and the dividing line segment is alternated accordingly between vertical and horizontal. The splitting process continues until a region is obtained that contains no data points. The node corresponding to this empty region is a leaf of the tree.

There is a definite recursive structure to the construction of a 2D tree. The data stored at each node is mandated by the information necessary for searching the tree. Such data includes the data point the cutting line segment passes through, as well as the orientation (vertical or horizontal) of the line segment. Naturally, there must also be

links to the left and right subtrees associated with each individual node. The choice for the orientation of the initial cutting line segment makes no difference in the construction of a 2D tree. From the preceding description, given a point set P_2, the construction of a kD tree is initiated by a call BUILD(P_2) of the procedure outlined in Figure 2.1.

Continuing with the 2D case, the method of construction is illustrated in Figure 2.2 for a set of $N = 13$ points. To start, the median x value is computed to be a coordinate of p_7. A vertical cutting line, passing through p_7, splits the space in two approximately equal regions. Since some data points remain in the left subregion, the left subtree is built, starting by computing the median y value of the points in the left subregion; in this case, point p_4. Once the left subtree is completed, the process continues by building the right subtree. When at last the construction is complete, the tree will contain one node for every point in the point set.

Preparata and Shamos [7] claim the construction of an N-point kD tree, $d \geq 2$, can be done in $O(dN \log N)$ time utilizing $O(dN)$ storage. Timing experiments were run on an implementation of the build kD tree algorithm for point sets ranging in size from $10 - 1,000,000$, yielding the results shown in Figure 2.3. The results would appear to confirm the required time for construction of a kD tree.

Now that the tree is built, the attention switches to how it can be traversed to find closest points. This leads to the introduction of the concept of a range search. Staying with the 2D example, each node of the tree represents a rectangular region. Letting p_{1i} represent the control point, the search range is computed by taking p_{1i} and creating

```
Initialize T to root
Procedure BUILD( point_set[ ] )
begin
        M(v) := compute median value;
        l(v) := orientation of cutting feature;
        p(v) := p_{2j} containing M(v)
        Set l for next level
        if ( left subregion NOT empty ) then
            left_child := BUILD( left subregion );
        else
            leaf of tree;
        end
        if ( right subregion NOT empty ) then
            right_child := BUILD( right subregion );
        else
            leaf of tree;
        end
end.
```

Figure 2.1. Outline for construction of a kD tree. The input is given as a set of data points, and v represents the current node of the tree.

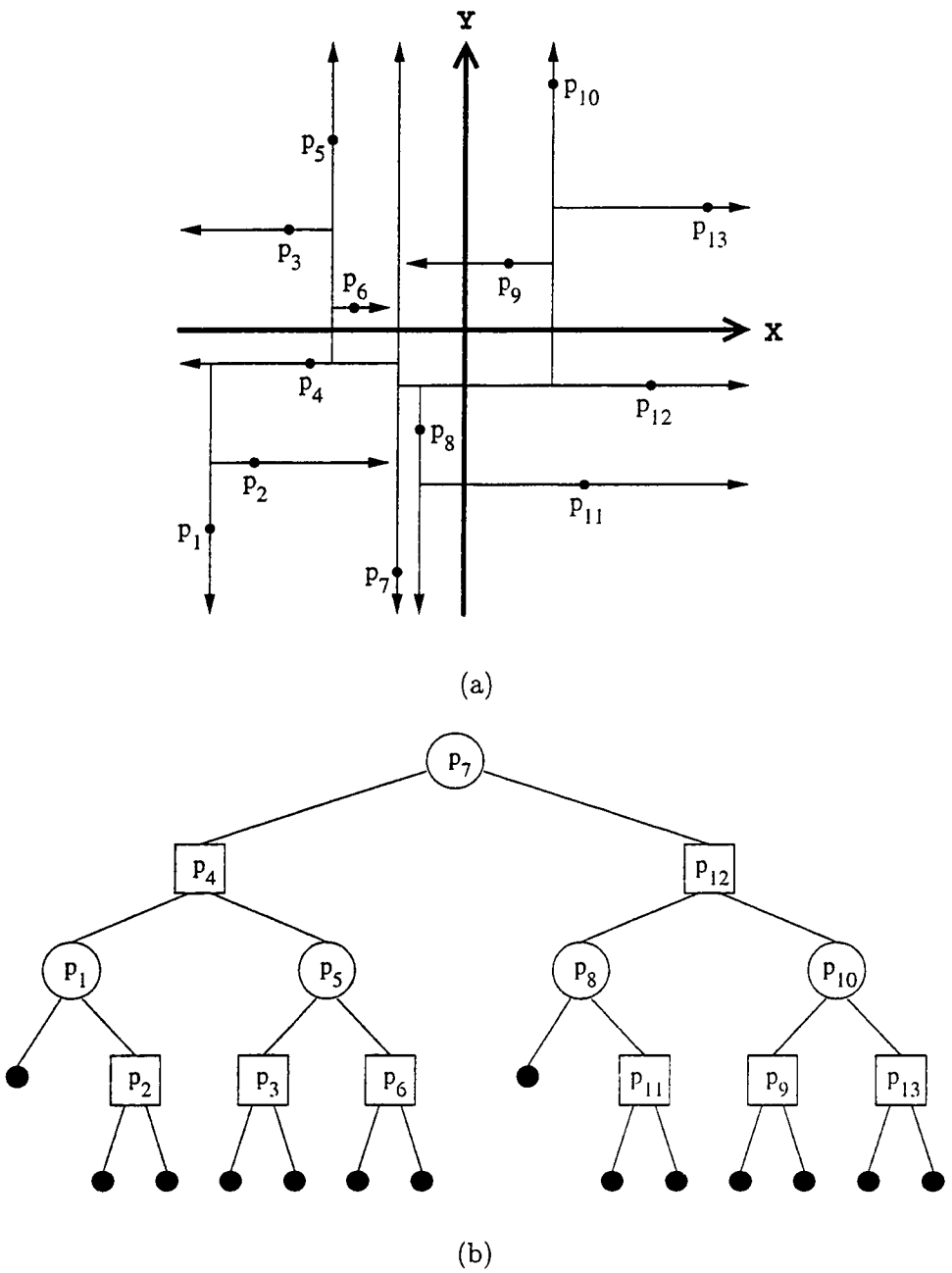


Figure 2.2. Illustration of a 2D binary-tree construction. The partition of the plane in (a) is modeled by the tree in (b). In (b) the following graphic convention is used: circular nodes denote vertical cuts; square nodes denote horizontal cuts; solid nodes are leaves (amended from [7]).

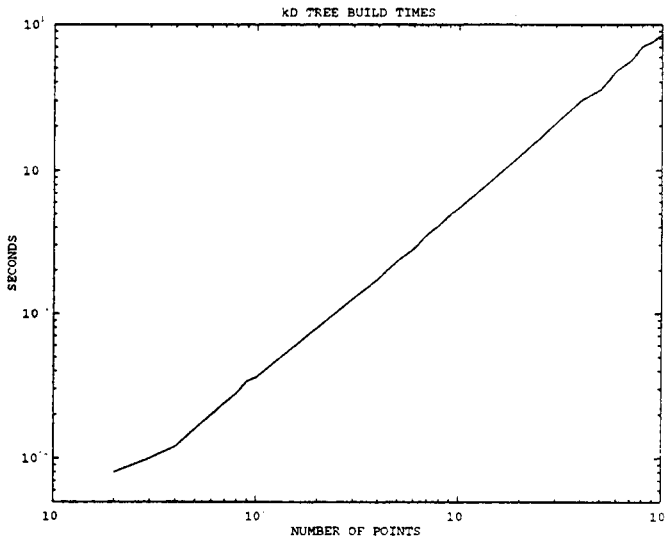


Figure 2.3. Timings for building a kD tree.

a square region around it based on the desired size of the search range, call it D. The search will stop at a node of the tree and include the point p_{2j} associated with that node in its retrieved points list only if p_{2j} lies completely within the search range. The search will continue to the left subtree if the search range lies completely to the left of the cutting line segment $l(v)$, to the right subtree if the search range lies completely to the right of $l(v)$, or will search both subtrees if a portion of the search range lies on both sides of $l(v)$. The search terminates upon reaching the leaf nodes of the subtrees traversed.

As is the case with the construction of the kD tree, the method of searching the kD tree is recursive. All the points p_{2j} found lying within the search range are stored in a list, call it U. Once the kD search is complete, a brute force search is performed for p_{1i} on the points in U, keeping the p_{2j} in U with the smallest distance as the closest

point. The value for the desired size of the search range is passed as a variable to the search procedure to give flexibility for using different search ranges. An alternative to storing all the potential closest points in a list and computing distances after the search is complete, is to compute $d(\ p_{1i},\ p_{2j}\ )$ each time a point p_{2j} is found to lie within the search range, keeping p_{2j} only if it is the closest point found to that instant. This slight change reduces the storage requirements as there is no longer a need to store all the points found within the search range. The change also reduces the execution time by eliminating the need to allocate storage for, and store points in the list. For the 2D case, given $D = [\ x_1,\ x_2\ ] \times [\ y_1,\ y_2\ ]$ as the search range, the search of the tree T is initiated by a call SEARCH(root(T), D) of the procedure outlined in Figure 2.4.

Using the example on the construction of a kD tree, the search procedure is illustrated in Figure 2.5. Part (a) shows the search range within the given space. Part (b) illustrates the visit of nodes of T performed by the search algorithm outlined in Figure 2.4. Note that only at nodes where the search splits does the test occur for inclusion of a point in the search range. The nodes in (b) marked with an asterisk (*) are those where the test for inclusion was successful and a distance calculation for closest point is performed.

The performance of the kD tree search is proportional to the total number of nodes of the tree visited during the search [7]. In particular, range searching of an N-point d-dimensional set can be effected in time $O(dN^{1-1/d})$. Despite the changes made in implementing the kD search algorithm, this worst case time still applies.

```
Procedure SEARCH( v, D )
begin
    if ( l(v) = y-axis ) then
       [ left, right ] := [ x1, x2 ];
    else { l(v) = x-axis }
       [ left, right ] := [ y1, y2 ];
    end
    if ( left ≤ M(v) ≤ right ) then
       if ( p(v) ∈ D ) then
          if ( d( p1i, p(v)  ) < current closest ) then
          begin
             current closest = d( p1i, p(v) );
             closest point = p(v);
          end
    if ( v ≠ leaf ) then
    begin
       if ( left < M(v) ) then
          SEARCH( left_child[v], D );
       if ( M(v) < right ) then
          SEARCH( right_child[v], D );
    end
end.
```

Figure 2.4. Outline for searching a kD tree. v represents the current node of the tree.

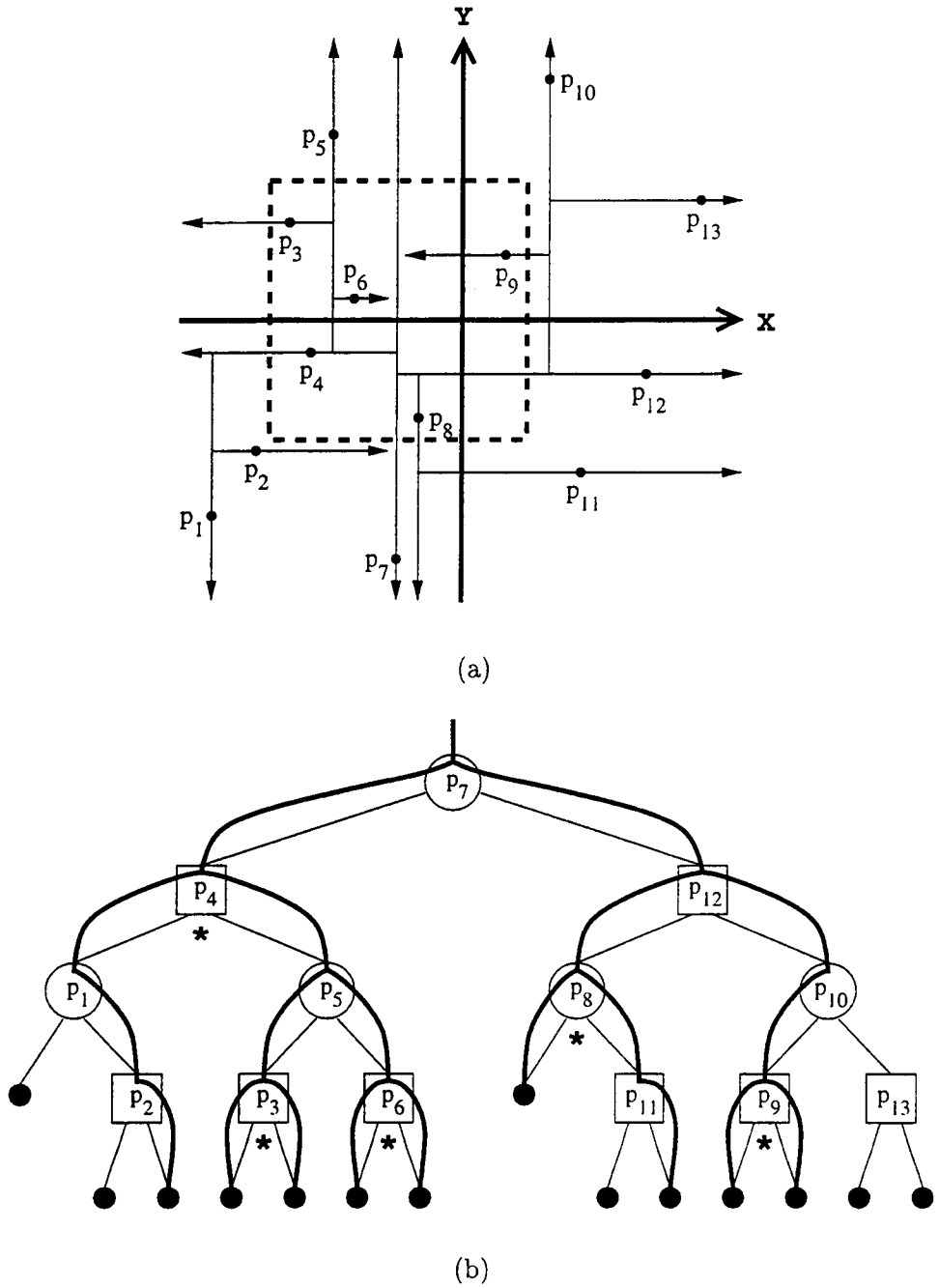


Figure 2.5. Illustration of a range search on the 2D binary-tree constructed in Figure 2.2. Part (b) shows the nodes actually visited by the search (amended from [7]).

While the kD tree is an improvement over the simple brute force method, still other methods exists that claim to perform range searching faster than the kD tree. One such method, the layered range tree [7], partitions the search space into standard intervals and utilizes pointers to enable direct access. Although the search performance supposedly is better than that of the kD tree, there is a large increase in the amount of storage required for the tree itself.

2.2.2 Median Selection

The most time consuming part of building a kD tree is in computing the median value. Preparata and Shamos [7] suggests a method called PICK, developed by Blum et al. [3], for the selection of a specific element in an array. This algorithm selects the ith smallest element of a point set S, where N is the number of points in S and $1 \leq i \leq N$. The PICK algorithm, at each iteration, discards from consideration a subset of S in which the elements are known to be too large or too small to be the ith element sought. Each subset discarded contains a minimum of 25% of the remaining elements under consideration. The discarding halts when left with only the ith element. The PICK algorithm is summarized in Figure 2.6.

It is perhaps easier to understand the algorithm by going through the example given in Figure 2.7. The original point set placed in N/c columns of length c is shown in part (a). Going from part (a) to (b) is simply a matter of sorting the elements in each individual column into descending order. To move from part (b) to (c), the columns are sorted into ascending order based on the dth smallest element in each column; in

(1) Select $m \in S$:

   (a) Arrange the elements of S into N/c columns of length c, and sort each individual column in descending order.

   (b) Select m as the bth smallest element of T, where T is an array of length N/c where its elements are the dth smallest element from each column. The N/c columns are sorted into ascending order based on the elements in T.

(2) Compute m's rank in S by comparing m to every element in S for which it is not already known whether $m < x$ or $m > x$.

(3) Halt the procedure if m is the ith smallest element in S, otherwise

   - Discard $D = \{ x \mid x \geq m \}$ and set $N = N - |D|$ if m's rank in $S > i$, otherwise discard $D = \{ x \mid x \leq m \}$ and set $N = N - |D|$, $i = i - |D|$.

   - Return to Step (1).

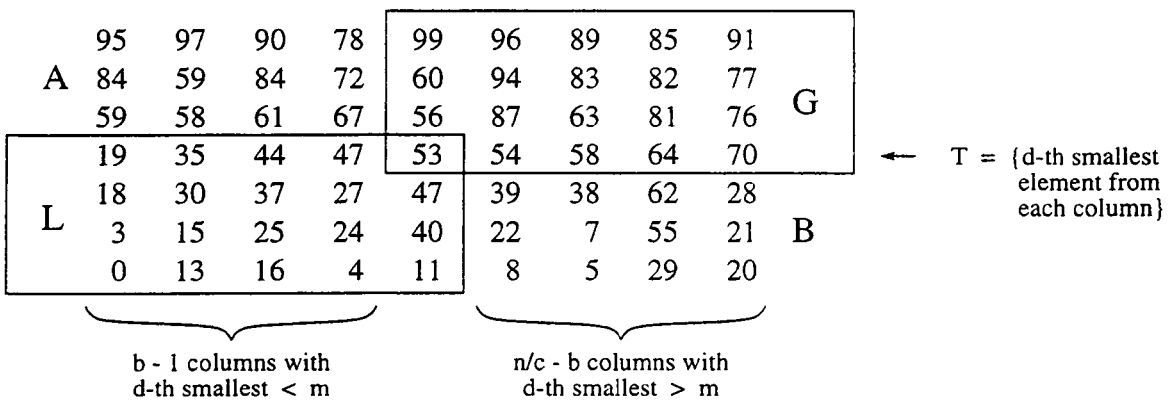
Figure 2.6. Outline for PICK median selection algorithm. The input is given as S representing the array of elements, N the number of elements in S, and i the index of the element sought.

58	81	25	76	3	78	97	99	87
5	64	44	91	19	24	59	53	22
83	55	90	28	0	47	30	40	39
63	82	37	77	84	4	15	56	54
89	62	61	70	95	72	58	60	8
38	29	16	20	59	27	13	47	96
7	85	84	21	18	67	35	11	94

(a)

89	85	90	91	95	78	97	99	96	
83	82	84	77	84	72	59	60	94	
63	81	61	76	59	67	58	56	87	
58	64	44	70	19	47	35	53	54	F
38	62	37	28	18	27	30	47	39	
7	55	25	21	3	24	15	40	22	
5	29	16	20	0	4	13	11	8	

(b)



(c)

Figure 2.7. Example of PICK algorithm for median selection. Part (a) shows the points of S arranged into N/c columns of length c. Part (b) is each column sorted in descending order. Part (c) shows the groups of points that could be discarded.

this case, the 4th smallest element as seen in box F of part (b). Looking at part (c) it is clear that all the elements in box G are greater than m and all the elements in box L are less than m. Only those elements in regions A and B will need to be compared to m in step (2) of Figure 2.6. In this case $m = 53$, the only element contained within both box G and box L. Letting $i = 44$ making the ith smallest element sought 70, the index of $m = 30$, since the index of $m \neq i$ and the index of $m < i$, the 20 points in box L are discarded along with the 10 points from regions A and B whose value is less than m, making $|D| = 30$. Since $N = 63$ to begin with, $N = N - |D|$ leaves $N = 33$ points, and $i = i - |D|$ makes $i = 14$ for the next iteration of the algorithm. The performance of the PICK algorithm is best explained by Blum et al. [3] by stating, "The PICK algorithm will execute a median search in linear time since a significant fraction of S is discarded on each pass, at a cost proportional to the number of elements discarded on each step."

While the PICK algorithm is theoretically a simple and fast method for computing the median value for ideal cases, it does have a few drawbacks. First is what to do when $N \bmod c \neq 0$, i.e., when all the columns are full and there are a few elements remaining. Second is the problem of determining an acceptable and optimal value for the length c of the columns. A concern as it relates to the implementation for the kD tree is that after the median value is determined, the elements of S must be separated into left and right subregions based on the median value. These factors make the implementation of the PICK algorithm impractical for the kD tree median search. The general concept of the PICK algorithm along with the necessity of separating the points into left and right

subregions, led to the idea of using a sorting technique for computing the median value.

The sorting method that best applies to the kD tree median search is the Quicksort algorithm. The use of Quicksort for this application is logical because the partitioning of the point set required for the kD tree is a built in feature of the algorithm. To avoid completely sorting point set P_2 each time a median value is computed, the partitioning value is used to determine when the median value has been found. When the partitioning value equals the median sought, Quicksort will terminate. Upon termination, the elements in P_2 will be appropriately separated into left and right subregions based on the median value. Making a slight modification to the Quicksort algorithm, a fraction of the elements in P_2 can be discarded from consideration at each pass of the partitioning procedure. This is accomplished by comparing the partitioning value to the median value sought. If the partitioning value is less than the median value sought, the lower index is set to the partitioning value + 1, and if the partitioning value is greater than the median value sought, the upper index is set to the partitioning value − 1. While there is no guarantee as to the minimum fraction of elements discarded at each iteration, the search space is guaranteed to decrease at each pass while simultaneously separating the elements into appropriate subregions. This method for median selection has the advantage of not needing to allocate extra storage to assist in the computation of the median value. In fact, the elements of P_2 never have to be moved, as the index into P_2 can be used for the partitioning of each region. Given that there are no special cases to consider, the Quicksort approach is easier to implement for this application.

2.2.3 Static vs. Dynamic

The kD search method described by Preparata and Shamos [7] uses a static search range. This is a general design for finding all points that lie within a specified range from a given point. For the purposes of the ICP algorithm however, the search requires only that the single closest point is located. Hence it is possible to modify the general algorithm in an attempt to make enhancements to its performance for this application. The first change was in keeping only the closest point rather than a list of all the points within a given range, as was mentioned previously. That concept led to the idea that once a point is found inside the search range, there is no need to continue checking points inside the search range that might possibly lie farther from the given point than the previous point already found. Since no point with a distance greater than that of the current closest point will be accepted, the size of the search range can be reset to the smaller distance each time a closer point is discovered. The smaller the search range, the fewer number of regions it will overlap, resulting in a smaller number of subtrees that will need to be traversed. Furthermore, letting the search range dynamically shrink with the distance of each closer point found results in fewer nodes the search will visit and fewer Euclidean distance calculations to perform.

The implementation considerations for changing to a dynamically shrinking search range are trivial. The search range by design is readable as a parameter to the search procedure. Therefore, the only requirement to implement the change is to pass a pointer to the search range to the search procedure so changes at one level of the tree will

propagate to all other levels. Each time a closer point is found, the search range is shrunk to a radius equal to the distance of the new closest point. The following section will demonstrate the gain in performance yielded as a result of this minor change to the searching algorithm.

2.2.4 Experimental Results on kD Tree

Numerous experimentation was done in an attempt to compare the performance of the three methods: brute force, static kD search, and dynamic kD search, for computing the closest point to a given point. Various size point sets ranging from 100 to $1,000,000$ points, with all coordinate values in the range [0 .. 1], were created using a random number generator. The initial seed for the random number generator was different for each size point set generated. An array of 100 points randomly generated were used as the control points for each search. For the timing experiments, the search time required for all 100 points combined was calculated and then averaged to yield the average time to search for a single point. For the productive node experiments, a productive node is counted as any time a Euclidean distance calculation is performed.

The results of the experimentation on the kD tree are shown in the following table and figures. The base reference for productive nodes comes from the brute force method, where the number of productive nodes is equivalent to the number of points in the given set. The base reference for the average search time is also computed using the brute force method, the results of which can be seen in Table 2.1. The figures show the comparison of static kD search and dynamic kD search vs brute force on the percentage

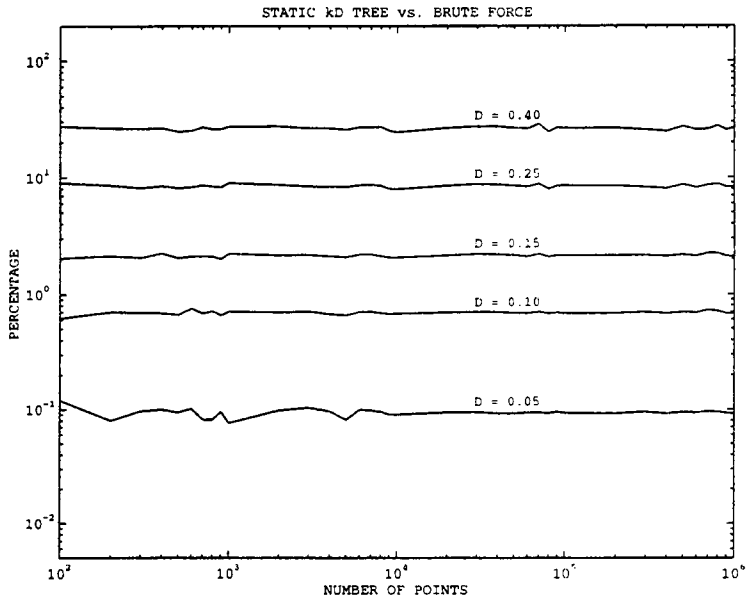
Table 2.1

Time in milliseconds required to compute the closest point, in the size point sets listed, to a single control point using the brute force search method.

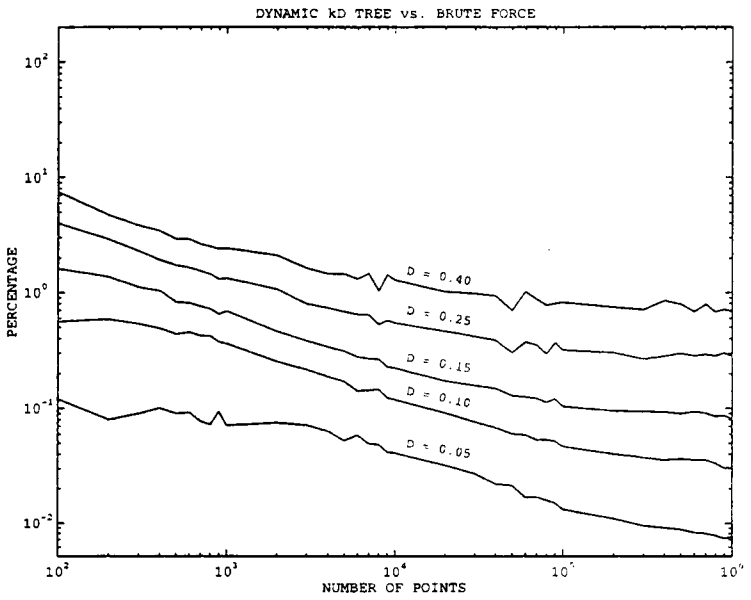
Points	Time	Points	Time
100	0.00	20,000	9.10
200	0.10	50,000	24.10
500	0.20	100,000	49.60
1,000	0.50	200,000	99.10
2,000	0.90	500,000	247.10
5,000	2.30	1,000,000	494.50
10,000	4.50		

of productive nodes and the average time required to search for a single closest point. Looking at Figure 2.8 part (a), the percentage of productive nodes for the static kD tree vs brute force appears linear, irrespective of the size of the search range. In contrast, for part (b), although the dynamic kD tree vs brute force percentages for point sets of size 10^2 are similar to those in part (a), as the number of points increases, the percentage steadily decreases. This is due to the fact that more and more points are eliminated from consideration by being outside the search range as the search range decreases during the searching process.

Although it is clear from Figure 2.8 that both versions of the kD tree perform far fewer Euclidean distance calculations than the brute force method, there is the additional overhead involved in traversing the tree that should be taken into account when discussing the difference in performance between the two methods. The comparison of performance including this overhead can be seen in Figure 2.9, where the kD tree and

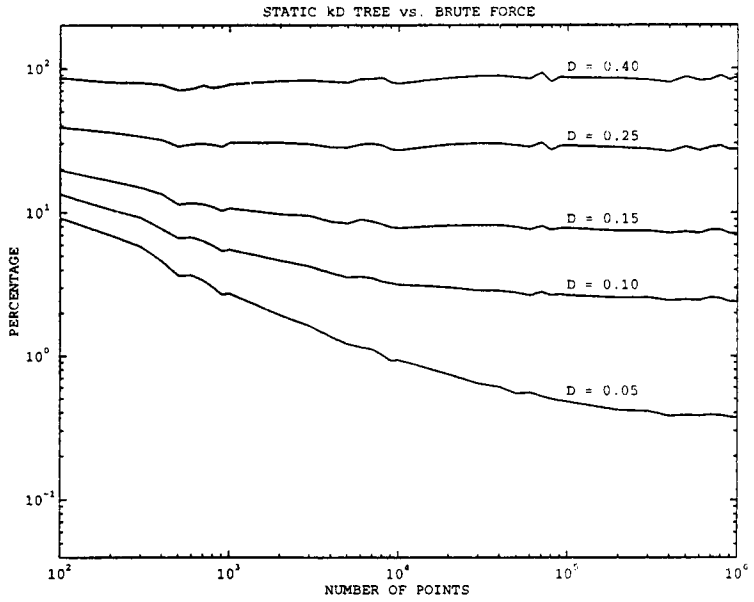


(a)

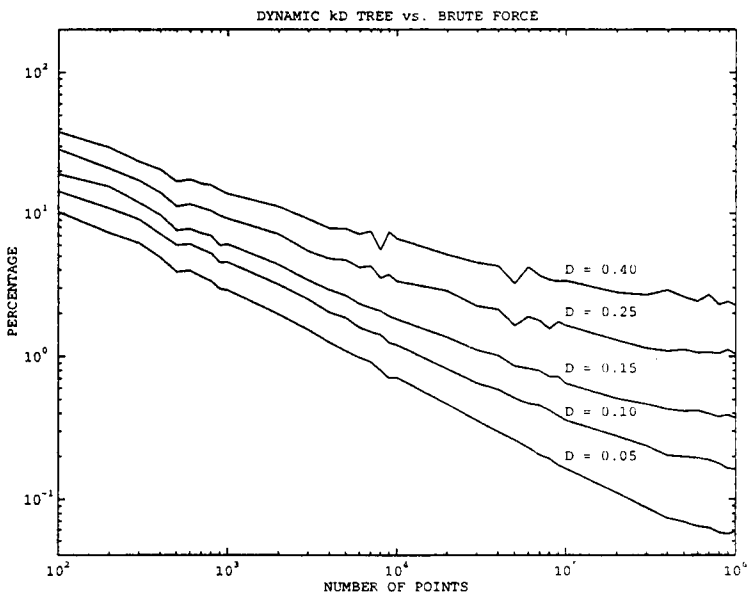


(b)

Figure 2.8. Experimental comparison of brute force, static kD search, and dynamic kD search, on the number of productive nodes. Each graph represents the percentage of productive nodes; static vs. brute force in part (a), and dynamic vs. brute force in part (b).



(a)



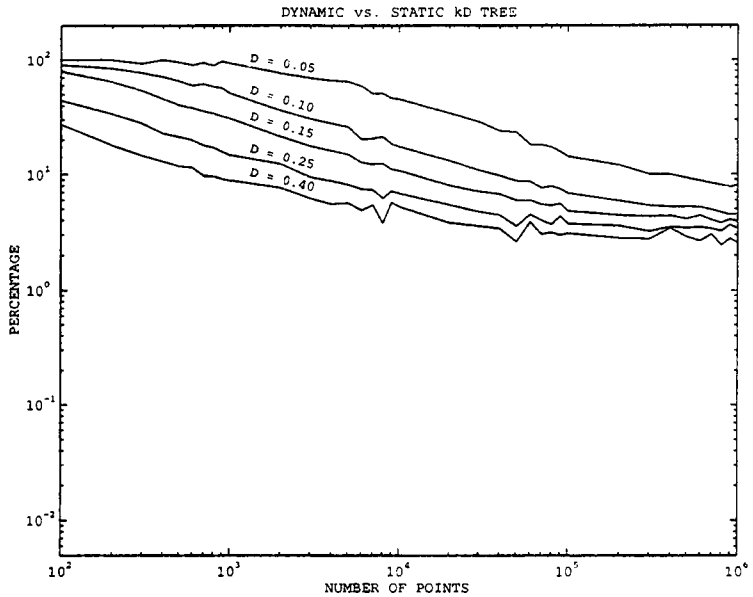
(b)

Figure 2.9. Experimental comparison of brute force, static kD search, and dynamic kD search, on the average time needed to search for a single closest point. Each graph represents the percentage of time required; static vs. brute force in part (a), and dynamic vs. brute force in part (b).

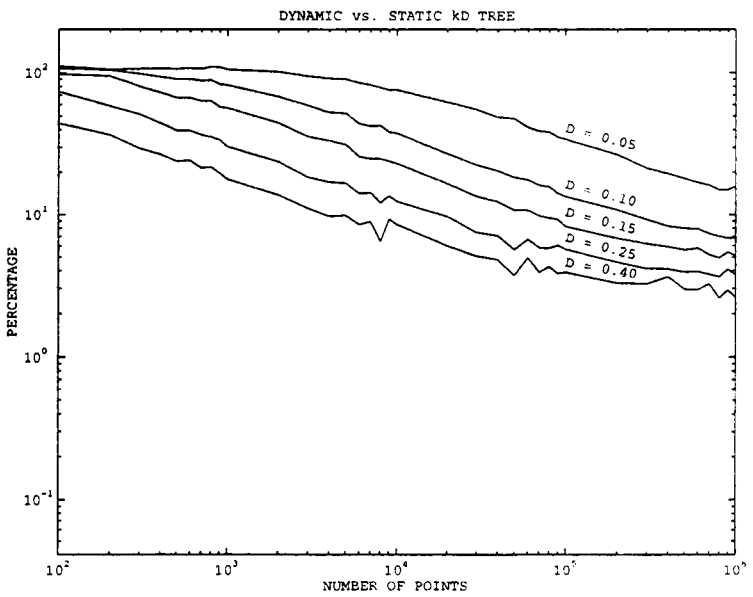
brute force methods are compared based on the average time required to search for the closest point to a single control point. The search times are given in Table 2.1, which shows the time in milliseconds required for a brute force search to compute the closest point to a single control point. As an example, consider a point set with size 10^6, which requires approximately 500 milliseconds to compute the closest point using brute force. From Figure 2.9 part (b), the dynamic kD search method using a search range with $D = 0.15$ requires approximately 0.3% of the time required for a brute force search. Multiplying $500 \times 0.3\%$ yields a search time for computing a single closest point at 1.5 milliseconds for the dynamic kD search.

In Figure 2.9 part (a) comparing the static kD tree vs brute force, for search ranges with $D = 0.40$, the search time is only slightly better for the static kD tree as opposed to the brute force method, regardless of the size of the point set. However, as the size of D decreases, the decrease in search time for the static kD search becomes increasingly significant. For point sets of size 10^6, decreasing the size of D from 0.25 to 0.10 decreases the percentage of time required for the search by more than an order of magnitude. Similar performance is seen in part (b) for the dynamic kD tree vs brute force. As with the productive node comparison of these two methods, for point sets of size 10^2 the percentages are similar to those in part (a). However, the rate of decrease for the search time is higher than that seen in Figure 2.8 part (b) for productive nodes.

The results presented in Figures 2.8 and 2.9 illustrate the superior performance of the kD tree method to that of brute force. Figure 2.10 presents the more interesting



(a)



(b)

Figure 2.10. Experimental comparison of static kD search vs. dynamic kD search on the number of productive nodes in part (a), and the time required to search for a single closest point in part (b).

comparison, showing the performance difference between the dynamic and static versions of the kD tree search. Notice that in parts (a) and (b) the percentages with respect to the value of D are inverted from those in Figures 2.8 and 2.9. This is because the larger the initial size of the search range, the more room there is for the search range to shrink in the dynamic version, as closer and closer points are found. Part (a) illustrates the percentage difference in productive nodes between the dynamic and static versions of the kD search, while part (b) shows the percentage difference in the time required to search for the closest point. In parts (a) and (b) both, the percentages for point sets of size 10^2 are fairly large, but both continue to decrease as the size of the point set increases. The graphs in Figure 2.10 provide visual evidence to the performance gains incurred with the dynamic kD search over the static kD search.

2.3 Outlier Detection

Outlier detection is an important part of establishing correct point correspondences. An outlier is a point in one image that has no corresponding point in the other image because of either noise in the data or the fact that some points that are visible to the camera in one frame are not visible in the other. The particular method considered here for the closest point correspondence method is referred to as the maximum tolerance for distance [12]. For details on other outlier detection techniques such as distance continuity in a neighborhood and orientation consistency of the surface normals for corresponding points, refer to [12]. The idea behind the maximum tolerance is to

calculate the distance between each point p_{1i} in P_1 and its corresponding point p_{2j} in P_2. If the distance $d(\ p_{1i},\ p_{2j}\ )$ is larger than the maximum tolerable distance as given by D_max, than p_{1i} is considered an outlier and is not included in the computation of the motion estimate for the current iteration of the ICP algorithm. This concept is accepted because the motion between the two given images is assumed to be small, and therefore the distance between two true point correspondences should not be very large. The value of D_max is set adaptively for each iteration of the ICP algorithm, and is based on statistics computed for the distances between corresponding points.

2.3.1 Statistical Updating of D_max

The statistical method for setting the value of D_max at each iteration, is derived from the concept that distances between true point correspondences should be similar, with a reasonable margin of error associated with noise in the data. Once the point correspondences have been established and the corresponding distances calculated, each distance is compared to D_max from the previous iteration, and only those correspondences with distances less than D_max are kept. The statistics computed on the remaining distances are the mean μ and standard deviation σ, using the equations

$$\mu \ = \ (1/N) \sum_{i=1}^{N} \ (d_i)$$

and

$$\sigma \ = \ \sqrt{(1/N) \sum_{i=1}^{N} \ (d_i - \mu)^2}.$$

D_max for the current iteration is adaptively set depending on the values of μ and σ, as shown in Figure 2.11.

The previous matching is next updated by removing a correspondence if its distance is larger than the new value of D_max. All the pairings remaining after the second update are considered true correspondences, and are used to compute the motion estimate for the current iteration. The ICP algorithm is claimed to be robust to relatively large motions and to bad outliers because of the manner in which D_max is updated in each iteration based on the statistics of the distances between corresponding points.

2.3.2 Setting of D and ξ

Two values, D and ξ, were introduced in the statistical method for updating D_max outlined in Figure 2.11. The value D is used to determine when the registration between

```
if ( μ < D )          /* the registration is quite good */
   D_max = μ + 3σ;
else if ( μ < 3D )    /* the registration is still good */
   D_max = μ + 2σ;
else if ( μ < 6D )    /* the registration is not too bad */
   D_max = μ + σ;
else                  /* the registration is really bad */
   D_max = ξ;
end.
```

Figure 2.11. Method for Statistical Updating of D_max.

two images is considered good, while the value ξ is used for D_max when the registration is relatively far off [12]. The value of D is also used in setting the initial value D_max^0 for the ICP algorithm as $20D$.

The value of D is crucial to the performance of the ICP algorithm because it has a direct impact on its convergence to the closest minimum. The value of D should correspond to the expected average distance between correspondences when the registration is good [12]. If D is set too small, then several true correspondences could be discarded, which may lead to additional iterations required for the algorithm to converge to a minimum. On the other hand, if D is set too large, than several false correspondences may be accepted, which could cause the algorithm to converge to a local minimum rather than the desired global minimum.

The value of D should be related to the resolution of the data. Letting $\bar{D}$ represent the average distance between neighboring points in a given point set, the mean of the distances between two point sets in perfect registration with one another is equal to $\bar{D}/2$. Since the mean is expected to be greater than $\bar{D}/2$ in general, $\bar{D}$ can be computed and used as the value for D.

In computing $\bar{D}$ for the closest point method, 25% of the points in P_2, uniformly distributed, are chosen as control points. Next, the distance between each of the control points and their eight surrounding neighbors is computed and averaged. Finally, the average distance calculated for each control point is summed, and the overall average distance is computed and assigned as the value of D.

The method described for computing the value for ξ is based on the distribution of the distances between correspondences. When the initial estimate of the transformation is fairly poor, the distribution of the distances as seen in a histogram, will typically be difficult to computationally interpret. The idea is that the largest peak will give some indication of what a reasonable correspondence should be. The value chosen for ξ is the first valley after the largest peak, provided the number of points at the valley is not greater than 60% of the points at the peak. Due to the complexity involved in implementing this heuristic, the value of ξ is arbitrarily set to $20D$, same as for the initial value of D_max.

Chapter 3

Point-to-Surface Correspondence

3.1 General Algorithm

Computing accurate correspondences across range images is a difficult task at best. Chen and Medioni [4] developed a method for computing correspondences based on minimizing the distance from a point to a plane as opposed to the standard minimization of the distance between points. The point to plane minimization only constrains the direction in which the distance can be reduced, leaving two other degrees of freedom in which the point can move in accordance with the constraints imposed by other points and planes. Given two point sets P_1 and P_2, and an initial transformation T^0, the iterative algorithm is outlined in Figure 3.1. The time complexity for each iteration of the algorithm is linear to the number of control points with acceptable smooth surface fits.

(1) Select a set of control points

(2) Perform a smooth surface fit to the neighborhood around each control point and calculate the residual error

(3) At each iteration k, for $k = 1, 2, 3, ...$, repeat the following until the process converges:

   (a) For each control point with an acceptable residual error:
      - Calculate $p'_{1i} = T^{k-1} p_{1i}$ and $n'_{p_{1i}} = T^{k-1} n_{p_{1i}}$
      - Compute the intersection point p_{2j} of surface Q in P_2 with the normal line defined by p'_{1i} and $n'_{p_{1i}}$
      - Compute the tangent plane S_i of Q at p_{2j}

   (b) Use a least squares method to compute the transformation T^k

Figure 3.1. Outline for Point-to-Surface Correspondence Algorithm.

3.2 Control Point Selection

The first step in the process is the selection of a subset of points $\{p_{1k}\}$ from P_1 as control points. A subset of points are used as opposed to all points in P_1 to reduce the computational time required. Two concerns related with control point selection are the number of control points to select and their distribution within P_1. Chen and Medioni [4] recommend that the number of control points needed for the computation of a good registration is generally in the range from 50 to 100, depending on the type of surface and the amount of overlap between the two frames. Rather than selecting a specific number of control points, an alternative technique is letting the number of control points be a percentage of the total number of points in P_1. An appropriate percentage can be determined based on the surface characteristics of the image.

The second more complex issue relating to the selection of control points is their distribution within P_1. The standard method of distribution is a uniform sampling of the points over all of P_1. Another method commonly used is that of random selection where points are randomly selected from P_1. There are advantages and disadvantages to the use of both methods. For the uniform distribution, the variable nature of the size of the point sets, percentage of control points requested, and size of the smooth surface fit, make an algorithm design for uniform selection complicated. Similarly, for random selection, the typical problem exists of keeping track of all points that have previously been selected, to prevent selecting the same point more than once. An advantage of the uniform approach however, is the better odds it gives that some of the points selected

will lie in smooth surface regions. The advantage of the random selection approach for this application is that if necessary, it can continue to select additional points beyond the specified percentage requested.

The selection of control points is an important step in the development of a robust registration algorithm. Given that control point selection is a complicated issue relating to global surface properties and evaluation criteria, Chen and Medioni [4] offer no suggestion on which method should be chosen. For this research, a uniform distribution was implemented.

3.3 Smooth Surface Fit

Each control point selected needs to lie in a smooth surface region to assist in the computation of correct correspondences. Two reasons for this are: (1) the corresponding neighborhoods in point set P_2 for each control point will likely be smooth also, and (2) a smooth planar region makes for a more reliable computation of the surface normal at each control point. After the control points $\{p_{1k}\}$ are selected, a smooth surface fit is performed on each point p_{1k} to test whether it is an acceptable point to use in the main iteration. The smooth surface fit is done by fitting a plane to the 5×5 neighborhood surrounding each p_{1k} using a least squares technique. The residual error for each neighborhood and associated plane is then calculated, and accepted only if the residual error is within the range computed based on the percentage of control points desired for the iteration. If acceptable, p_{1k} and its respective plane description (normal

vector and offset) are stored and used in the iterative portion of the correspondence algorithm.

To perform a plane fit, the points in the 5×5 neighborhood surrounding each p_{1k} are stored in a 25×3 matrix for which the singular value decomposition is computed. The next step is computing the normal vector $n_{p_{1k}}$ by solving the linear equation $Ax = b$ for the 3×1 vector x, where A is the matrix resulting from the SVD of the 25×3 matrix above, and b is a 25×1 vector whose elements all equal -1. Prior to solving the linear equation, a check must be performed to ensure that matrix A is not ill-conditioned, meaning that some of the singular values are very small but not zero. The very small singular values must be zeroed in order to yield a meaningful and accurate solution for x (the normal vector $n_{p_{1k}}$ of the plane). This is accomplished by finding the largest singular value, then computing the minimum acceptable singular value as a product of the maximum singular value and the typical constant value 10^{-6}, as given in [8]. The necessary singular values are zeroed by stepping through each of the singular values and setting to zero any one that is less than the minimum acceptable singular value. A back-substitution routine can now be used to solve for x. The final step in the computation of the plane fit involves setting the offset of the plane, o_k, to 1 and normalizing. The resulting plane, $w_k = (n_{p_{1k}}, o_k)$ is used in determining if p_{1k} lies in a smooth enough region.

The smoothness of the region around p_{1k} is determined by calculating the residual error between w_k and the points in the 5×5 neighborhood surrounding p_{1k}. The residual

error, r_e, is computed by

$$r_e = \sum_{i=1}^{25} | (n_{p_{1k}} p_{1i} + o_k) |. \tag{3.1}$$

An acceptable residual error is determined by the ith smallest residual error computed, where i is computed as a percentage of the total number of control points. For example, letting $N = 200$ be the total number of control points and 25% be the percentage of points desired, than $200 \times 25\% = 50$ gives the value of i, and the maximum acceptable residual is the 50th smallest residual error computed.

3.4 Intersection Point Algorithm

The intersection of a surface Q in P_2 and a line l in the direction of $n_{p_{1k}}$ is a crucial part of the correspondence algorithm and is illustrated in Figure 3.2. Chen and Medioni's [4] approach is to compute the intersection of line l with the tangent plane S_i to surface Q in the neighborhood of possible intersection points on Q. This is an iterative procedure with the prospective initial intersection being chosen by projecting p_{1k} orthographically along the z-axis onto Q. This is done by searching a 2D binary tree built from only the x and y coordinates of the points in P_2. In other words, the orthographic projection of p_{1k} along the z-axis onto Q is accomplished by computing the closest (x, y) coordinates in P_2, call it $P_2(x, y)$, to the (x, y) coordinates of p_{1k}, call it $p_{1k}(x, y)$. Once the prospective intersection point p_{2j} is found, the next step is to compute the tangent plane S to Q

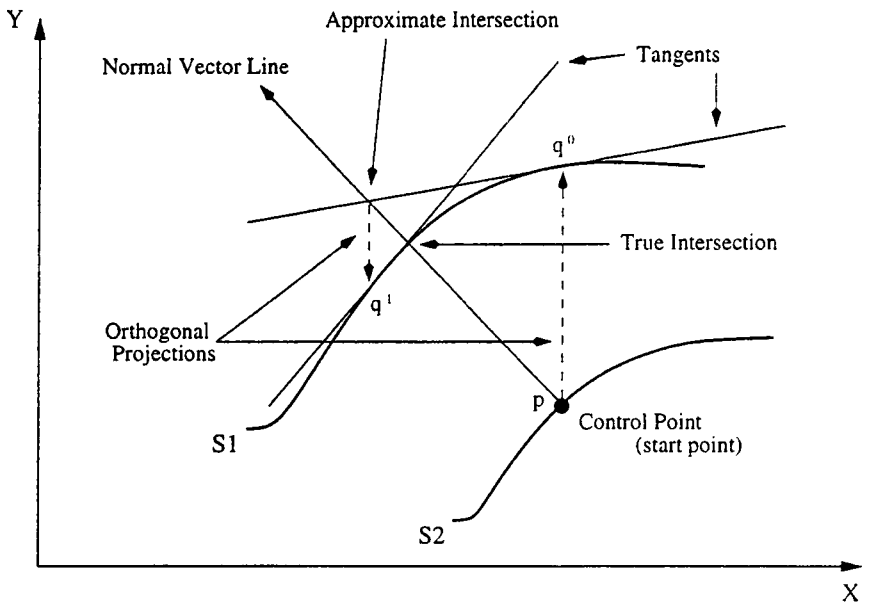


Figure 3.2. Intersecting a line with a surface (amended from [4]).

at point p_{2j}. This is done in the same manner as for the plane fit to the control points described in the previous section. Finally, the actual intersection point is computed between l and S.

The iteration is terminated when at successive iterations $\mid P_2(x,y)^k - P_2(x,y)^{k-1} \mid$ is less than the spatial sampling unit computed for only the x and y coordinates in P_2. The convergence of the intersection point algorithm depends on the prospective intersection points computed at each iteration and the surface characteristics of the neighborhood surrounding the actual intersection point [4]. This is the real reason that the control points selected need to be in a smooth planar region. It is estimated that one intersection will typically require approximately 3 to 5 iterations to converge. Note that the algorithm does not work when the surface normal associated with a control

point is nearly perpendicular to the z-axis. Likewise, if the closest $P_2(x, y)$ to $p_{1k}(x, y)$ lies near enough to the border of P_2 so that a 5×5 neighborhood cannot be constructed around the prospective intersection point, the intersection algorithm will halt without producing any result.

3.5 Correspondence Options

There are numerous possible methods for computing the motion estimate given the correspondences established through the use of the point-to-surface method. From the information computed throughout the course of the algorithm, correspondence data exists for point-to-point, point-to-plane, and plane-to-plane motion estimation.

The point-to-plane correspondences, pictured in Figure 3.3 part (a), are used by Chen and Medioni to compute the motion estimate at each iteration. They claim to minimize the distance between the transformed control point p_{1k} and the tangent plane S used to represent the intersection point on surface Q. however, there is no algorithm given for computing the motion estimate using point-to-plane correspondences.

The point-to-point correspondence information is used by Sequeira et al. [10] to compute the motion estimate. After using the point-to-surface method to compute correspondences, Sequeira et al. project p_{1k} orthogonally onto S, as shown in Figure 3.3 part (b), and use the resulting point, call it p'_{2j}, as the corresponding point. The motion estimate can now be computed same as for the point-to-point correspondence method.

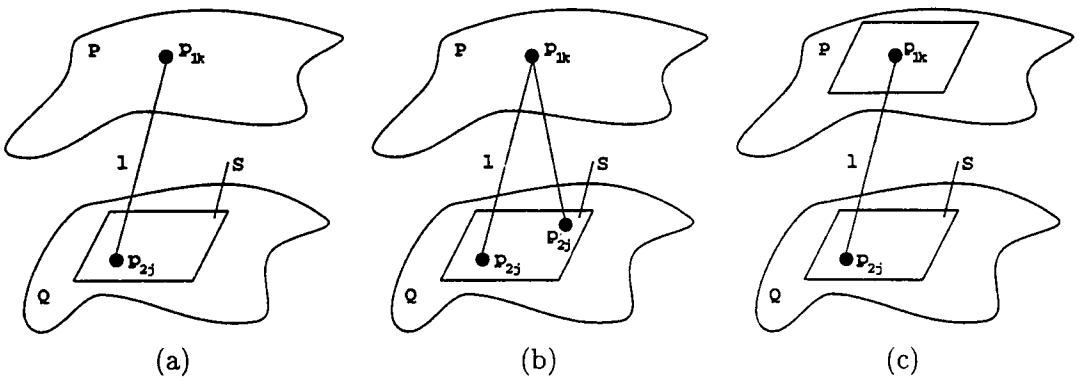


Figure 3.3. Correspondence options available for computing motion estimate with the point-to-surface algorithm. Part (a) is an illustration of a point-to-plane correspondence; (b) a point-to-point correspondence; and (c) a plane-to-plane correspondence.

The third set of correspondence data available is plane-to-plane, as seen in Figure 3.3 part (c). The first plane comes as a result of the plane fit performed on each control point to check for the smoothness of the surrounding region, and the algorithm computes the corresponding plane as the tangent plane S computed for the actual intersection point. All that is needed is a technique for computing the motion estimate by minimizing the distance between corresponding planes. A new method for this task is discussed in detail in Chapter 5.

3.6 Outlier Detection

As seen with the closest point method, outlier detection plays an integral part in keeping only correct correspondences for motion estimation computation. For this application, the focus is on two methods of detecting outliers specifically suited to plane-to-plane correspondences.

3.6.1 Smooth Surface Test on Tangent Plane

The concept of outlier detection is to discard those correspondences that form incorrect matches. Using the point-to-surface algorithm, the task becomes how to tell when p_{1k} is paired with a tangent plane S representing an incorrect intersection point. Given that each control point accepted lies in a smooth planar region, its corresponding tangent plane should also lie in a smooth planar region if it is computed for the correct intersection point. This is the premise for the first outlier detection technique, where the smoothness, of the region in P_2 where the intersection point was found, is computed and used to discard those correspondences with a non-smooth tangent plane.

The implementation of this outlier detection technique is done using the same smooth surface fit performed on each control point. The residual error computed in Equation (3.1) for the tangent plane associated with the final intersection point is computed for each correspondence established. The issue to consider is what is an acceptable residual error for a smooth surface fit on the tangent plane. The easiest answer is to accept a certain percentage of the smallest tangent plane residuals. For example, letting $N = 150$ be the number of tangent plane residuals computed and 67% the percentage desired, than $150 \times 67\% = 100$ is the number of residuals to accept. In general terms, any pair of corresponding planes with a tangent plane residual error greater than the 67th percentile is considered an outlier for that iteration. Eliminating corresponding plane pairs with large residual errors for the smooth surface test of the tangent plane yields a more accurate calculation of the motion estimate.

3.6.2 Angle Between Corresponding Normal Vectors

The second method implemented for outlier detection was first mentioned by Zhang [12]. The method described was to compute the surface normal vectors associated with each corresponding point, and then calculate the angle between them. Although the technique was not intended for this application, it is a logical choice given that the surface normals are part of the correspondence information and therefore have already been computed.

The main idea is that the angle between corresponding normal vectors should not be greater than the maximum amount of rotation between the two images. Hence, a pair of corresponding planes is considered an outlier if the angle between their normal vectors is greater than a certain threshold. The question again becomes how to set and/or compute the value for an acceptable angle. The solution is based on the distribution of the angles between all corresponding plane pairs. The angles are counted based on one degree intervals from 0 to 90°. Next, the interval with the largest number of angles is computed and chosen as the base angle between normal vectors for the current iteration. Finally, any pair of corresponding planes with an angle between normal vectors within a range of ± 5 degrees from the base angle is considered a valid correspondence, while all others are marked as outliers for the current iteration.

Both outlier detection techniques can be used in conjunction with one another. The best results are seen when the angles between normal vectors are filtered first, followed by the tangent plane residual errors.

Chapter 4

Point-to-Point Motion Estimation

Once point correspondences have been established and outliers removed, the motion estimate between two point sets is computed. A motion estimate consists of a 3×3 rotation matrix R and a 3×1 translation vector t. Given two 3D point sets $P_1 = \{p_{1i}\}$ and $P_2 = \{p_{2i}\}$, $i = 1, 2, ..., N$ with each p_{1i} and p_{2i} represented as a 3×1 column vector, the task is to solve for the rotation R and the translation t that minimizes Equation (1.2). This chapter focuses on three methods for computing R and t known as Singular Value Decomposition (SVD), Unit Quaternion, and Dual Number Quaternions.

4.1 Singular Value Decomposition

The SVD method by Arun et al. [1] attempts to decouple the rotation and translation in order to solve for the rotation separately, and then use it to compute the translation. The decoupling is possible because if the least squares solution to Equation (1.1) is

$[\ \hat{R},\ \hat{t}\ ]$, then $\{p_{2i}\}$ and $\{p'_{2i} = \hat{R}_{p_{1i}} - \hat{t}\}$ have the same centroid. Hence, two point sets Q_1 and Q_2 are computed as the original point sets P_1 and P_2 with the centroid of each respective point set subtracted from each point. This is done mathematically by first computing the centroids of P_1 and P_2 as

$$c_1 \ = \ (\ 1\ /\ N\ ) \sum_{i=1}^{N} p_{1i}$$

and

$$c_2 \ = \ (\ 1\ /\ N\ ) \sum_{i=1}^{N} p_{2i}.$$

The point sets Q_1 and Q_2 are then computed by

$$Q_1 \ = \ \{\ q_{1i} \ = \ p_{1i} \ - \ c_1\ \} \tag{4.1}$$

and

$$Q_2 \ = \ \{\ q_{2i} \ = \ p_{2i} \ - \ c_2\ \} \tag{4.2}$$

Equation (1.2) has thus been reduced to the simpler function

$$\Sigma^2 \ = \ \sum_{i=1}^{N} \|\ q_{2i} \ - \ Rq_{1i}\ \|^2.$$

Once a solution for R has been calculated, the translation is computed as

$$t \ = \ c_2 \ - \ Rc_1. \tag{4.3}$$

The problem remaining is to calculate a solution for R. This is accomplished by the singular value decomposition of a 3×3 matrix, namely, the covariance matrix

$$H = \sum_{i=1}^{N} q_{1i} q_{2i}^T,$$

where the superscript T stands for transposition. The singular value decomposition of matrix H yields $U\Lambda V^T$, such that U and V are orthogonal 3×3 matrices whose columns are the normalized eigenvectors of HH^T and H^TH respectively, and Λ is a 3×3 diagonal matrix given by $\Lambda = U^THV$. The elements of Λ represent the singular values of H. Finally, the rotation is computed as

$$R = VU^T. \tag{4.4}$$

The solution calculated for R is not quite complete. If the determinant of $R = +1$, then R from Equation (4.4) is the correct solution. However, a problem arises when the determinant of $R = -1$. In such cases, the SVD algorithm has produced a solution for R that represents a reflection across a single axis due to the points in $\{q_{1i}\}$ being coplanar. However, by simply negating the third column of matrix V if its smallest singular value is zero and forming $V' = [\ v_1,\ v_2,\ -v_3\ ]$, then the correct solution can be computed as

$$R = V'U^T.$$

When the determinant of $R = -1$ and none of the singular values of H is zero, the

method fails. This situation happens when neither $\{q_{1i}\}$ nor $\{q_{2i}\}$ are coplanar, but a reflection solution is produced due to the fact that there is no possible rotation that yields a smaller value for Σ^2. This only occurs when noise levels are extremely large, in which case the least squares solution is not the best approach.

4.2 Unit Quaternion

The standard method for representing rotation is a 3×3 orthonormal matrix computed from the Euler angles as described in Chapter 1. An alternative approach is to use the so-called unit quaternion, which describes rotation around a unit vector that passes through the origin. As can be seen in Figure 4.1, n represents the unit vector and θ the amount of rotation around n between the first and second coordinate frames.

Mathematically, a quaternion can be viewed as a complex number with three different imaginary parts [6]:

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

Given the axis of rotation described by the unit vector $n = [\ n_x,\ n_y,\ n_z\ ]$ and the rotation angle θ_n, the quaternion is computed as

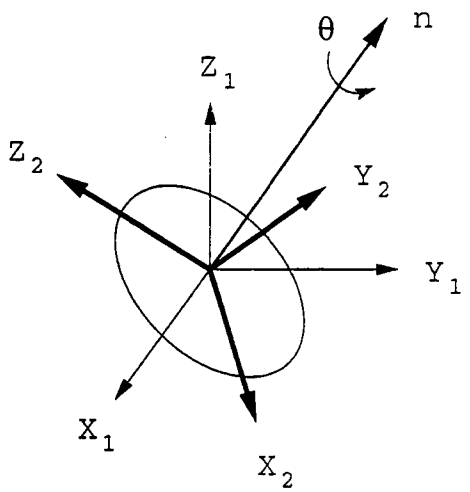


Figure 4.1. Illustration of rotation for a real quaternion (amended from [6]).

$$q_0 = \cos(\theta_n / 2),$$

$$q_x = \sin(\theta_n / 2) n_x,$$

$$q_y = \sin(\theta_n / 2) n_y,$$

$$q_z = \sin(\theta_n / 2) n_z.$$

An advantage to using the unit quaternion representation is the ease of maintaining the constraint that a quaternion be of unit magnitude as opposed to the requirement that a matrix be orthonormal [6].

As for the SVD method, the unit quaternion method makes use of the fact that $\{p_{2i}\}$ and $\{p'_{2i} = Rp_{1i} - t\}$ have the same centroid when R and t solve the least squares problem formulated in Equation (1.1). Therefore, the point sets Q_1 and Q_2 are computed by subtracting from P_1 and P_2 their respective centroids as in Equations

(4.1) and (4.2). The 3×3 covariance matrix M is calculated from the individual points of Q_1 and Q_2 by

$$M = \sum_{i=1}^{N} q_{1i} q_{2i}^T.$$

The resulting matrix contains all the necessary information to solve for R using a least squares technique. The individual elements of M can be viewed as

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

where

$$S_{xx} = \sum_{i=1}^{N} q_{1ix} q_{2ix}, \qquad S_{xy} = \sum_{i=1}^{N} q_{1ix} q_{2iy},$$

and so forth. The 4×4 real symmetric matrix N is computed using sums and differences of the elements of M as follows

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

The next step is to compute the eigenvectors and corresponding eigenvalues of N. Horn [6] shows that the unit quaternion $\dot{q} = [\ q_0,\ q_x,\ q_y,\ q_z\ ]$ that represents the desired rotation from Q_1 to Q_2 is the eigenvector that corresponds to the largest positive eigenvalue of N.

The orthonormal rotation matrix can be computed from the components of a unit quaternion $\dot{q}$ through the following computation [6]:

$$R = \begin{bmatrix} (q_0^2+q_x^2-q_y^2-q_z^2) & 2(q_xq_y+q_0q_z) & 2(q_xq_z+q_0q_y) \\ 2(q_yq_x+q_0q_z) & (q_0^2-q_x^2+q_y^2-q_z^2) & 2(q_yq_z-q_0q_x) \\ 2(q_zq_x-q_0q_y) & 2(q_zq_y+q_0q_x) & (q_0^2-q_x^2-q_y^2+q_z^2) \end{bmatrix}.$$

Likewise, a unit quaternion can be recovered from an orthonormal rotation matrix. Once R has been computed, the translation vector t is computed the same way as for

the SVD method using Equation (4.3).

4.3 Dual Number Quaternion

For both the SVD and unit quaternion methods discussed above, the translation vector t is computed using the recovered rotation. A perceived drawback to this approach is that any error in the computation of R is propagated in the computation of t. To combat this, Walker et al. [11] devised a method using dual number quaternions to solve for R and t simultaneously by minimizing a single cost function.

The two parts of a dual quaternion, $\ddot{q} = \dot{r} + \varepsilon\dot{s}$, are the real part, $\dot{r}$, and the dual part, $\dot{s}$, which are both real quaternions. As illustrated by Figure 4.2, the transformation using a dual number quaternion is accomplished by first translating the first coordinate system a distance of d along the direction represented by the unit vector n. The second step is to rotate the coordinate system with respect to a line defined as having direction given by the unit vector n and passing through a point p, an amount given by the angle θ. It is possible to convert to and from the dual number quaternion representation and the orthonormal matrix representation of rotation, the details of which can be found in [11].

The mathematics behind this method of motion estimation are quite detailed and somewhat complicated. The following is a basic generalization of the implementation of the method, and details of the derivation can be found in [11]. As with the two previous point-to-point methods, the input is given as two point sets P_1 and P_2 of corresponding

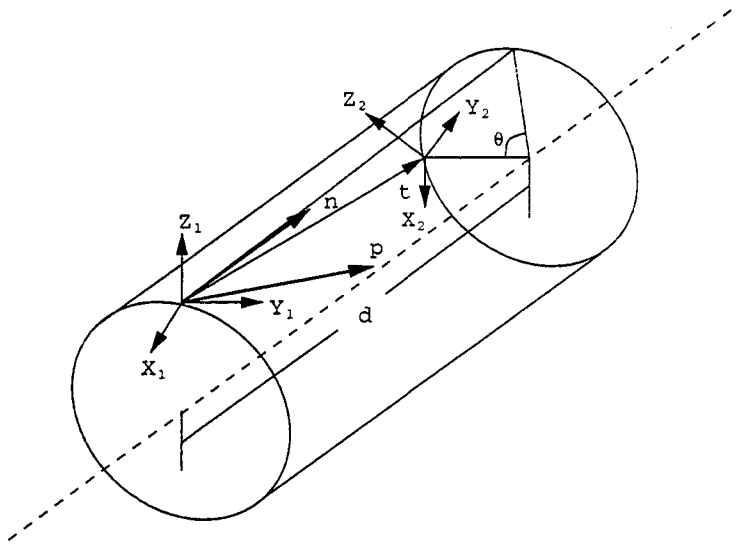


Figure 4.2. Illustration of rotation and translation for a dual number quaternion (amended from [11]).

points. The first step is to compute two 4×4 symmetric matrices C_1 and C_3 from the point sets P_1 and P_2. Matrix C_1 is computed as cross-product-sums of the 3×3 matrix S, which is formed from P_1 and P_2 by multiplying corresponding points and computing their summation over all point correspondences. Matrix C_3 is computed by using the cross-sum-products of two 3×1 vectors sA and sB which are formed as the summation over all point correspondences of the x, y, and z coordinates of each point respectively. Letting N represent the number of point correspondences, the next step is to compute the 4×4 symmetric matrix A from C_1 and C_3 as

$$A = (C_3 \ / \ 4N) + (C_1 \ / \ 2).$$

The quaternion $\dot{r}$ corresponds to the eigenvector of the largest positive eigenvalue of

A. The desired orthonormal rotation matrix is computed from the quaternion $\dot{r} = [\, r_0,\ r_x,\ r_y,\ r_z \,]$ as follows:

$$R = \begin{bmatrix} r_z^2 + r_0^2 - r_x^2 - r_y^2 & 2(r_0 r_x - r_y r_z) & 2(r_x r_z + r_0 r_y) \\ 2(r_y r_z + r_0 r_x) & r_z^2 - r_0^2 + r_x^2 - r_y^2 & 2(r_x r_y - r_0 r_z) \\ 2(r_0 r_y - r_x r_z) & 2(r_0 r_z + r_x r_y) & r_z^2 - r_0^2 - r_x^2 + r_y^2 \end{bmatrix}.$$

The quaternion $\dot{s} = [\, s_0,\ s_x,\ s_y,\ s_z \,]$ of $\ddot{q}$ is computed from $\dot{r}$ and the sA and sB vectors previously calculated. The $\dot{r}$ and $\dot{s}$ quaternions, along with the total number of point correspondences N, are used to compute t as follows:

$$t = \begin{bmatrix} (\, s_0 r_z - s_x r_y + s_y r_x - s_z r_0 \,) \ / \ N \\ (\, s_0 r_y + s_x r_z - s_y r_0 - s_z r_x \,) \ / \ N \\ (\, -s_0 r_x + s_x r_0 + s_y r_z - s_z r_y \,) \ / \ N \end{bmatrix}.$$

4.4 Experimental Results

Given the three different approaches to computing the motion estimate between frames, some method of comparison is needed to determine which method is the most accurate and stable. Eggert et al. [5] devised experiments to compare the accuracy of the various methods under varying levels of noise, as well as the stability (computation of a correct transformation under different data conditions) of the algorithms. The accuracy portion of these experiments has been recreated for the SVD, Unit Quaternion, and Dual Number Quaternion methods, attempting to match the previous results. The

experiments were performed using both single precision and double precision floating point values.

The first task is generating the data for the experiments. Several data point sets of non-degenerately arranged $3D$ points were generated, with sizes ranging from $N = 4$ to $10,000$. The point sets, $\{m_i\}$, were randomly generated from a uniform distribution within a cube of size $2 \times 2 \times 2$ centered about the origin. Each corresponding data set, $\{d_i\}$, was formed by adding uncorrelated, isotropic, Gaussian noise with a zero mean and variable standard deviations of 0, 10^{-10}, 10^{-6}, 0.01, and 1.0 to each component of the individual points in $\{m_i\}$, and then transforming them using the generated transformations.

The rotation and translation components of the transformations were generated separately. The $3D$ translations were generated by randomly selecting numbers uniformly distributed in the range $[\ -10 \ldots 10\ ]$. Each rotation was calculated from a randomly generated unit quaternion subject to certain constraints. The individual components of each quaternion were randomly selected from a uniform distribution and scaled to be within the range $[\ -1 \ldots 1\ ]$. The quaternion generated is accepted only if its magnitude is less than one. If not less than one, random quaternions were continually generated until an appropriate one was found. Once a quaternion is generated that represents a valid rotation, it is normalized, and the orthonormal rotation matrix is extracted using Horn's method [6].

For each data set size/noise level combination, one hundred trials were run, where

a trial consists of new points for data set $\{m_i\}$, along with new noise and transformation values. During each of the one hundred trials, three different error statistics were computed for each of our three methods. The first two were simply the norm of the difference between the actual and estimated translation vectors,

$$t_{err} = \| t_{alg} - t_{act} \|$$

and the norm of the difference between the actual and estimated unit quaternions representing the rotation,

$$\dot{q}_{err} = \| \dot{q}_{alg} - \dot{q}_{act} \|.$$

The third statistic calculated was the root mean square (RMS) error of the distance between corresponding points in the model and adjusted data sets under the found transformation,

$$RMS_{err} = \sqrt{\sum_{i=1}^{N} (\| d_i - R_{alg} m_i - t_{alg} \|^2) / (N - 3)}.$$

For each of the figures on the following pages, the left column represents results obtained using single precision floating point values, while the right column presents results using double precision, as was originally used in [5]. Each figure shows the results for translation, rotation, and RMS errors computed. Figure 4.3 is for the SVD method, Figure 4.4 the Unit Quaternion method, and Figure 4.5 shows the results for the Dual Number Quaternion method.

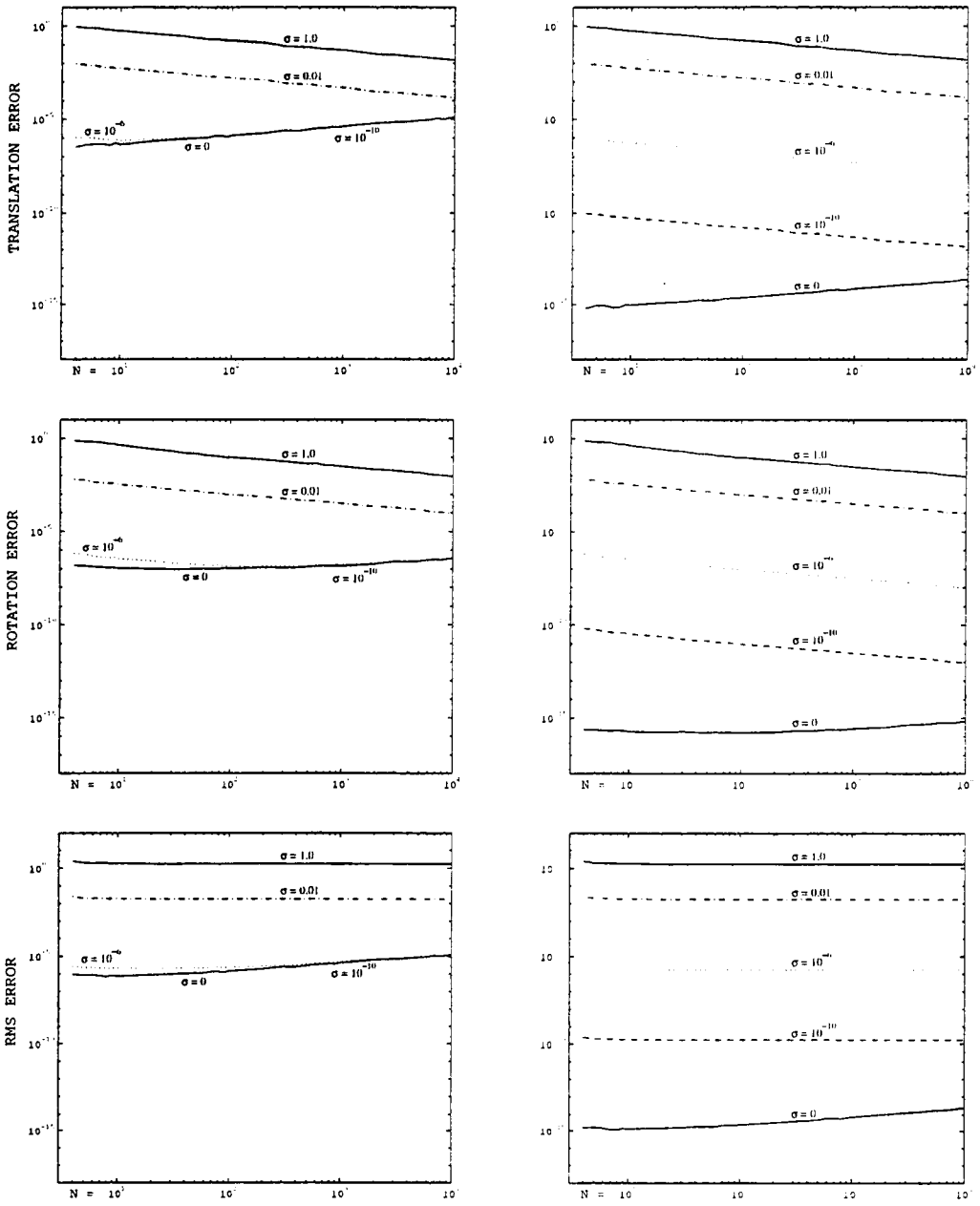


Figure 4.3. SVD Motion Estimation Experiments, with the number of corresponding points N ranging from 3 to 10,000 while σ ranges from 0.0 to 1.0. The graphs on the left are errors for motion estimates computed using single precision floating point representation, while the graphs on the right use double precision.

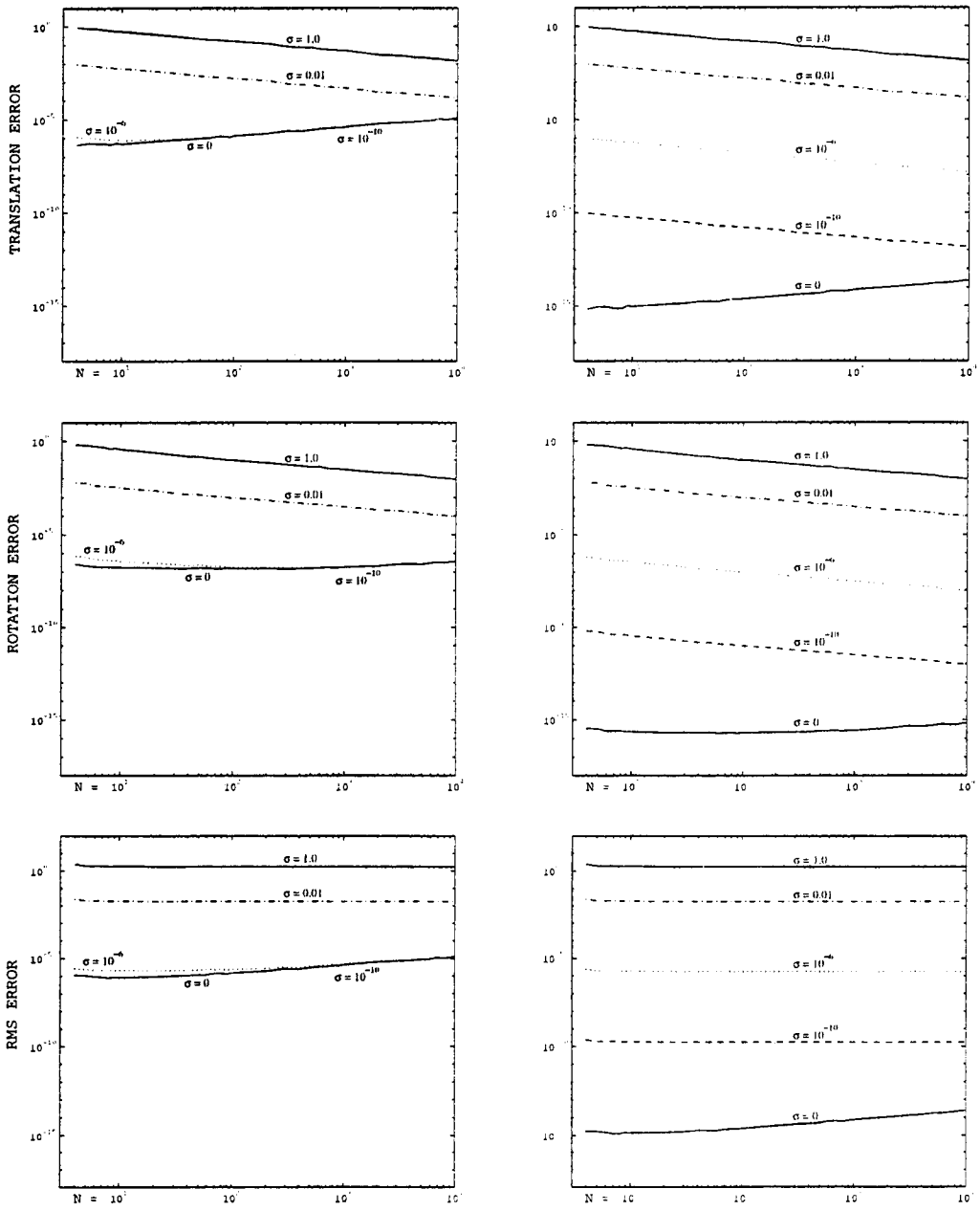


Figure 4.4. Unit Quaternion Motion Estimation Experiments, with the number of corresponding points N ranging from 3 to 10,000 while σ ranges from 0.0 to 1.0. The graphs on the left are errors for motion estimates computed using single precision floating point representation, while the graphs on the right use double precision.

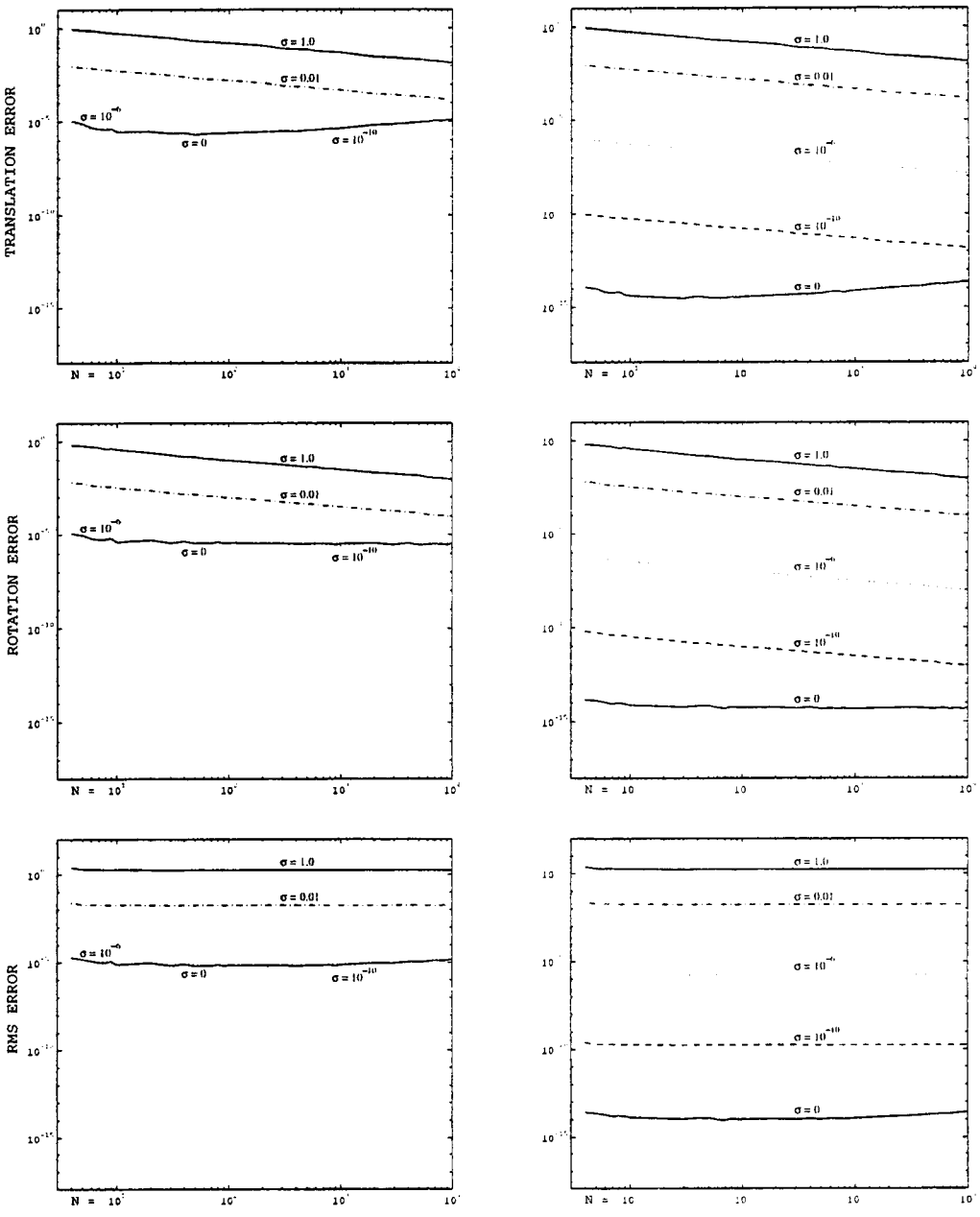


Figure 4.5. Dual Number Quaternion Motion Estimation Experiments, with the number of corresponding points N ranging from 3 to 10,000 while σ ranges from 0.0 to 1.0. The graphs on the left are errors for motion estimates computed using single precision floating point representation, while the graphs on the right use double precision.

From the figures it is evident that little to no difference exists between the performance of the three algorithms. For smaller standard deviations, the errors reach the level of machine precision. This is best displayed by the differences between the results when using single precision vs. double precision floating point representation. For single precision, it appears the error values for $\sigma = 0.0$, 10^{-10}, and 10^{-6} are relatively the same, whereas the same statistics show a vast difference for those standard deviations when using double precision. This is a result of the machine precision for floating point representation being approximately 10^{-6}. The experimentation was performed for both floating point representations because the implementation of the algorithms for this research all use single precision floating point values. The improved accuracy provided through the use of double precision values is not necessary considering most real data images will likely contain noise levels greater than the machine precision for floating point representation.

Timing experiments were performed during these accuracy experiments to determine if there was a noticeable difference in the time required to execute each algorithm. Timings were captured for each of the five hundred trials run for a particular data set size, and then averaged. The final results were that the SVD and Unit Quaternion methods require the same amount of execution time, while the Dual Number Quaternion method executes just slightly faster. In general, there is no significant gain in execution time with one method over another.

Chapter 5

Plane-to-Plane Motion Estimation

This new technique was developed by Dr. Jens Gregor to compute the motion estimate between two frames given corresponding planes. The technique uses a least squares method to solve for rotation and translation separately. This chapter gives a description of the underlying mathematics, and experimental evidence showing the accuracy of the technique.

Consider a set of points p in frame 1 which define a plane (η, o), and a set of points q in frame 2 that define a plane (ν, ω), where η and ν are normal vectors and o and ω the respective offsets. The objective is to isolate the rotation and translation components and solve for each separately. First, p can be transformed into the coordinate frame of

q by

$$q = Rp + t. \tag{5.1}$$

Since R is orthogonal meaning $RR^T = I$, Equation (5.1) can be solved for p, resulting in

$$p = R^T(q - t). \tag{5.2}$$

Now, if p lies in the plane given by (η, o), then p satisfies

$$\eta^T p + o = 0. \tag{5.3}$$

Substituting the result in Equation (5.2) derived for p into Equation (5.3), gives

$$\eta^T(R^T(q - t)) + o = 0. \tag{5.4}$$

Using a transpose property for matrices where $(AB)^T = B^T A^T$ and multiplying through by R^T in Equation (5.4) gives

$$(R\eta)^T q - (R\eta)^T t + o = 0. \tag{5.5}$$

The final step in isolating R and t is to let Equation (5.5) be equivalent to

$$\nu^T q + \omega,$$

such that

$$\nu \;=\; R\eta$$

and

$$\omega \;=\; -\nu^T t \;+\; o.$$

The rotation and translation components have been isolated as single unknowns in two separate equations, and can be solved for independently of one another.

Letting N represent the total number of corresponding plane pairs, solving for R is a simple matter of forming the covariance matrix given by

$$\sum_{i=1}^{N} \eta_i \, \nu_i^T,$$

and taking the SVD of the resulting matrix. The SVD produces the usual $U\Lambda V^T$, and the desired rotation matrix is $R = U^T V$, same as seen for the point-to-point SVD motion estimation method in Chapter 4. Lastly, the determinant of R is checked to insure that the result is a proper rotation and not a reflection.

To solve for the translation t, the object is to minimize the squared error given by

$$\min_t \sum_{i=1}^{N} (\omega_i \;+\; \nu_i^T t \;-\; o_i)^2.$$

Multiplying and pulling the constant out front gives

$$\min_t \sum_{i=1}^{N} \nu_i^T t \nu_i^T t \; + \; 2(\; \omega_i \; - \; o_i \;) \nu_i^T t$$

which can be rewritten as

$$\min_t \; t^T Q t \; + \; 2 m^T t,$$

where

$$Q \; = \; \sum_{i=1}^{N} \nu_i \; \nu_i^T \tag{5.6}$$

and

$$m \; = \; \sum_{i=1}^{N} \; (\; \omega_i \; - \; o_i \;) \nu_i. \tag{5.7}$$

This leaves the equation $Qt = -m$, which is solved for t by using SVD inversion. Taking the SVD of matrix Q from Equation (5.6) yields the familiar $U \Lambda V^T$. Using these matrices and m computed in Equation (5.7), the solution for t is given by

$$t \; = \; - V diag(\; 1 \; / \; \omega_j \;) U^T m.$$

Hence, a nice clean method exists to solve for rotation and translation given pairs of corresponding planes for input as opposed to points.

The experiments described in Section 4.4 to test the performance of the three point-to-point techniques were also performed on the plane-to-plane motion estimation method. Although there is an insufficient basis to make a direct comparison between

the results for the plane-to-plane and point-to-point methods, the translation, rotation, and RMS errors seen in Figure 5.1 are similar to the results shown in Chapter 4 for the point-to-point methods. The findings suggest that this new method produces accurate motion estimates under varying levels of noise. Also, the timing experiments performed on this method show no significant differences in execution time between the plane-to-plane and point-to-point methods.

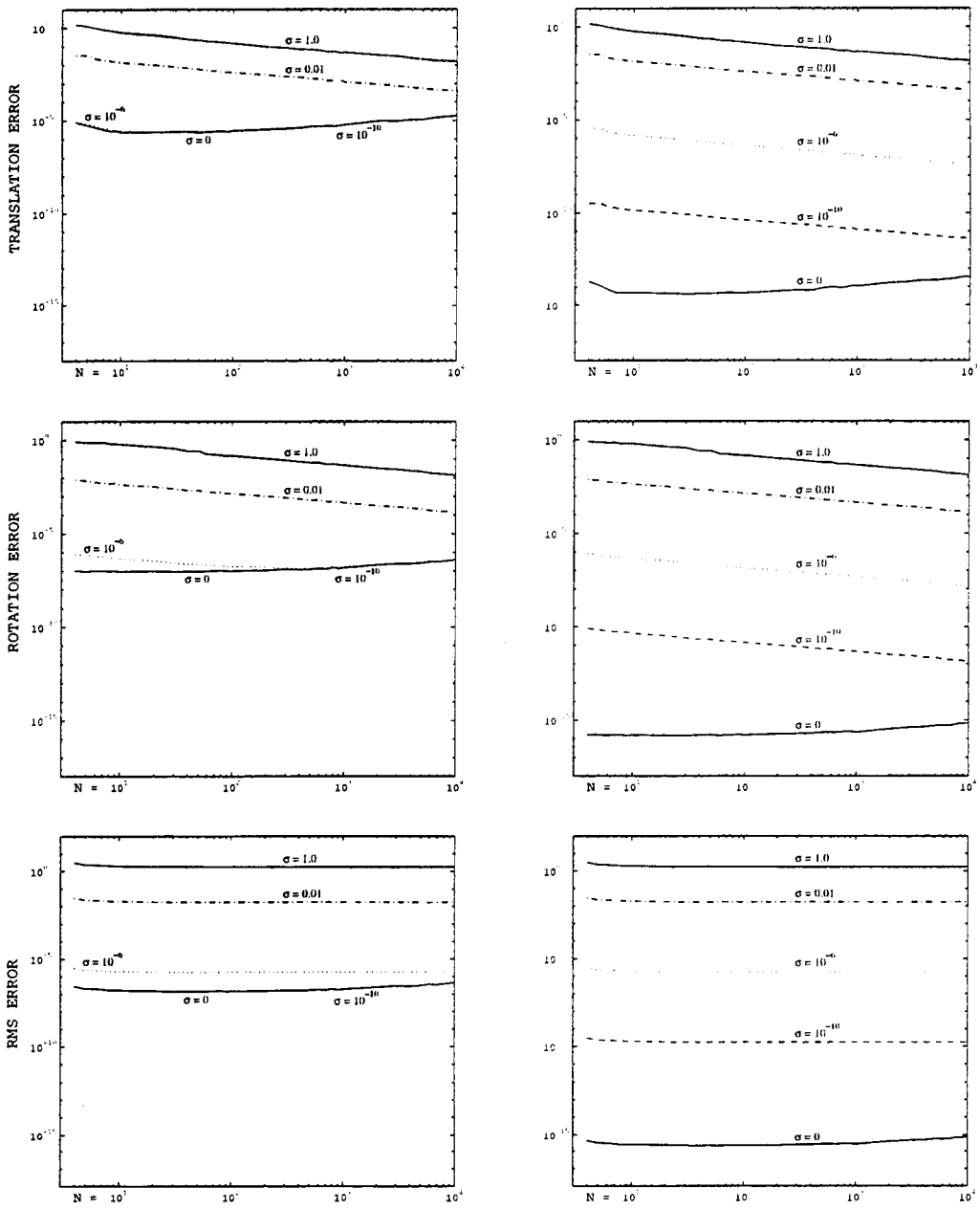


Figure 5.1. Plane-to-Plane Motion Estimation Experiments, with the number of corresponding planes N ranging from 3 to 10,000 while σ ranges from 0.0 to 1.0. The graphs on the left are errors for motion estimates computed using single precision floating point representation, while the graphs on the right use double precision.

Chapter 6

Experimentation

6.1 Synthetic Data

This section presents some examples of the closest point and point-to-surface range image registration algorithms. The images used were captured using a simulator imitating a spherical coordinate system. The scene scanned was of a simple box in a corner, as seen in Figure 6.1. The images were created using a field of view of 20° with size 256 × 256 before they were sub-sampled to a size of 128 × 128. The rectangular coordinates (x, y, z) are stored using single precision floating point values. The actual transformation between the two images used for the experimentation is $r = [\ -17.5°\ \ -18.5°\ \ 15.8°\ ]$ and $t = [\ -92.3\ \ 79.0\ \ 38.0\ ]$, where the vector r represents θ_x, θ_y, and θ_z, the amount of rotation around the x, y, and z axes respectively. The results presented are for ideal data containing no noise.

The results produced using the closest point algorithm and the point-to-surface

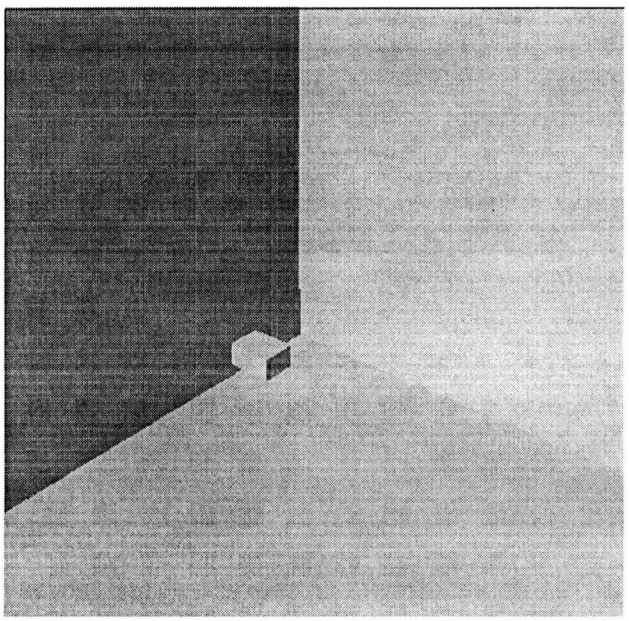


Figure 6.1. Scene for Synthetic data images.

method are shown in Figures 6.2 and 6.3 respectively. The two graphs at the top of each figure represent the rotation and translation errors for tests performed on the two images where the initial transformation is computed by rotating around the x and y axes, the amount shown for the angle of rotation, away from the θ_x and θ_y angles in r. Hence, an angle of rotation of 0 means the actual transformation is given as the initial estimate, while an angle of 4 means the initial estimate is computed by rotating the θ_x and θ_y angles 4° off from the actual transformation, i.e., the r used to compute the initial estimate is $r = [\ \theta_x - 4°,\ \theta_y - 4°,\ \theta_z\ ]$. Similarly, the middle graphs represent transformations where the x and y components of the translation are moved off by an amount given by the units of translation, and the bottom graphs show results where both the θ_x and θ_y angles as well as the x and y components of the translation are

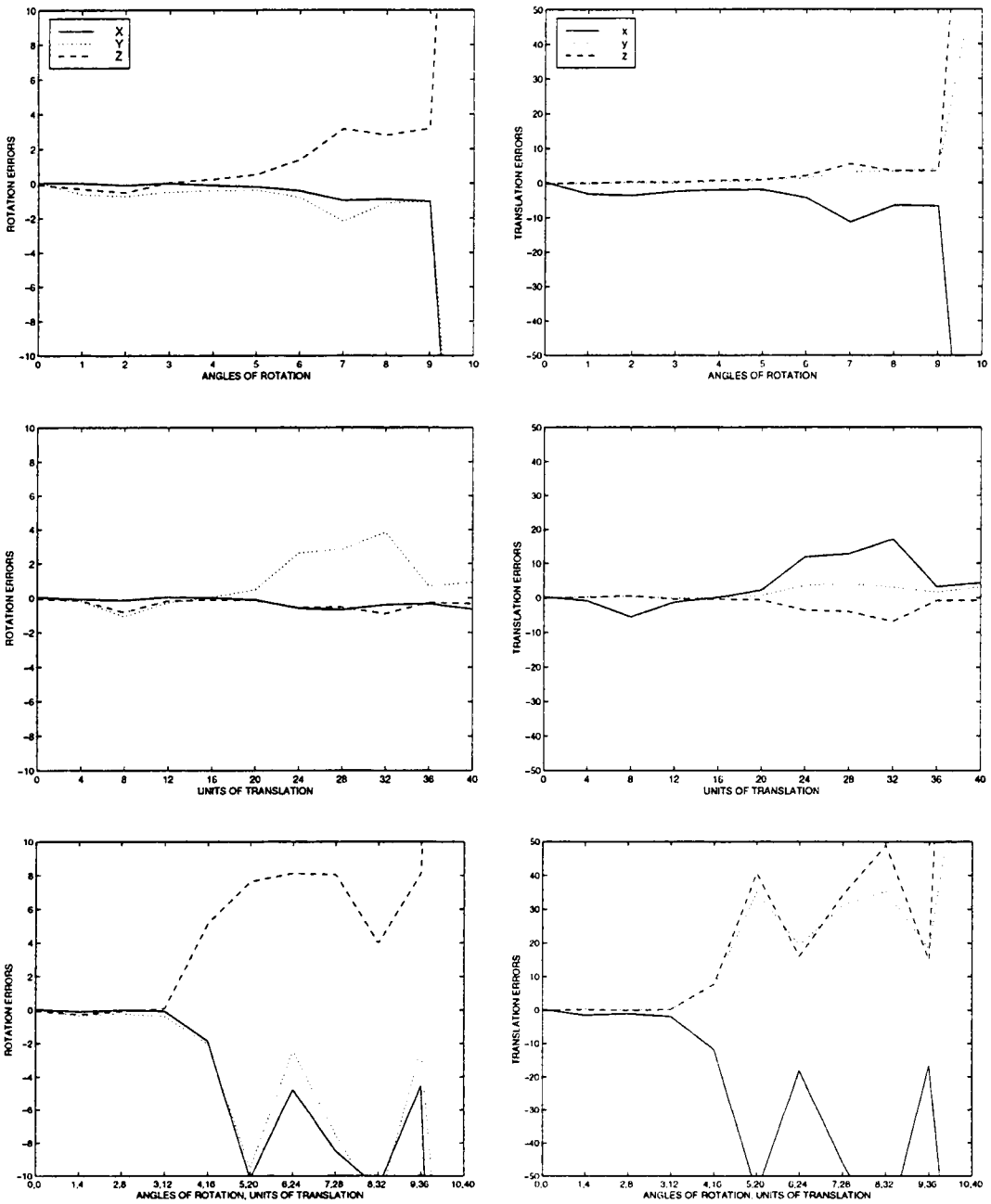


Figure 6.2. Results for closest point correspondence method. The graphs in the left column show the rotation errors for the transformation produced by the algorithm minus the actual transformation. The right column shows the translation errors.

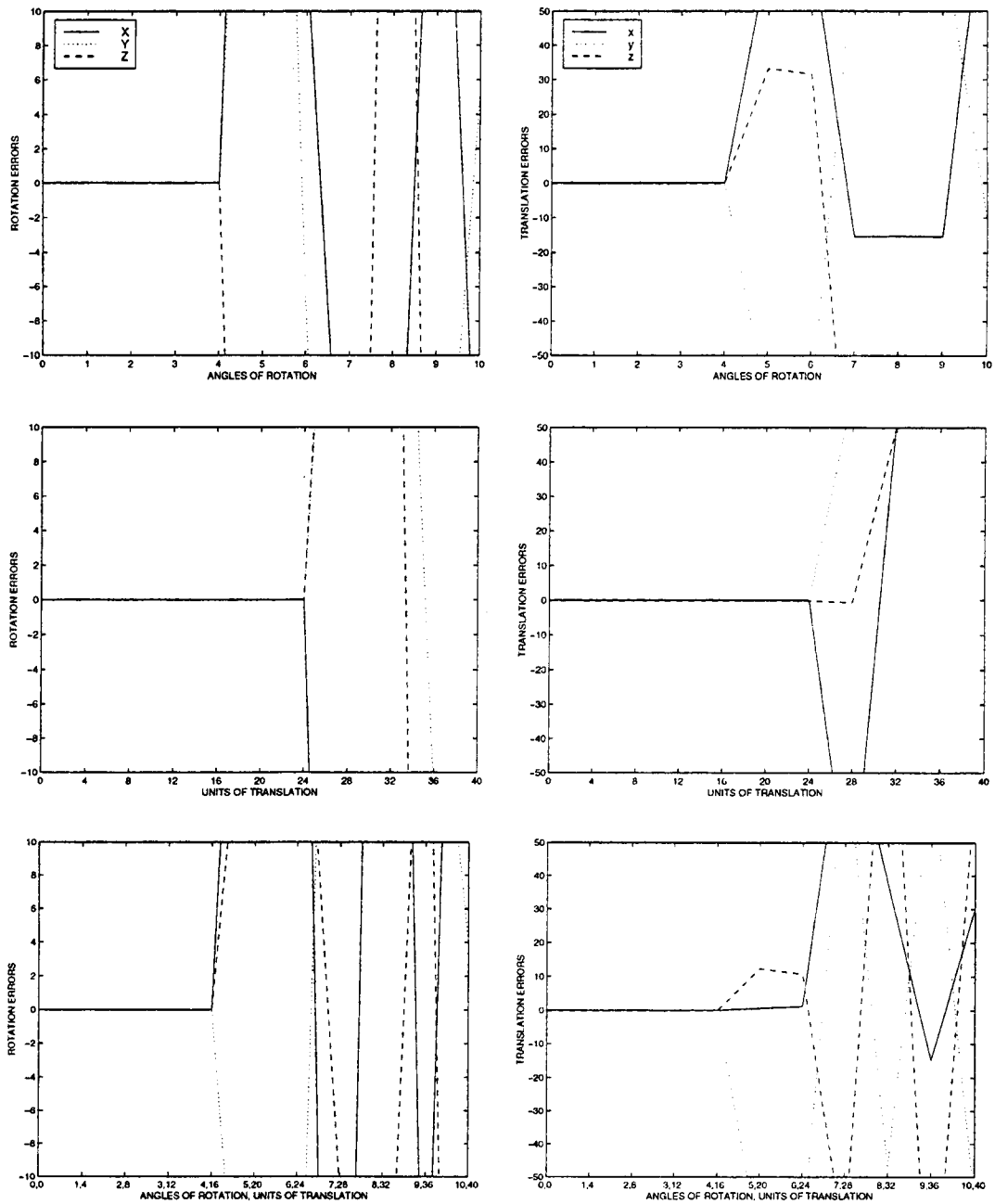


Figure 6.3. Results for point-to-surface correspondence method. The graphs in the left column show the rotation errors for the transformation produced by the algorithm minus the actual transformation. The right column shows the translation errors.

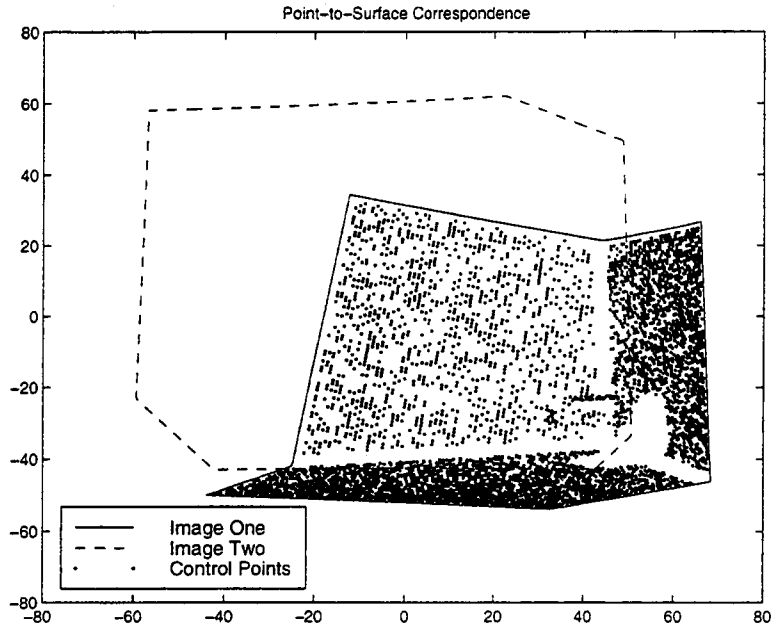
moved off simultaneously. The results presented represent the error in the registration as the transformation produced by the algorithm minus the actual transformation, or mathematically,

$$r_{res} = r_{alg} - r_{act},$$

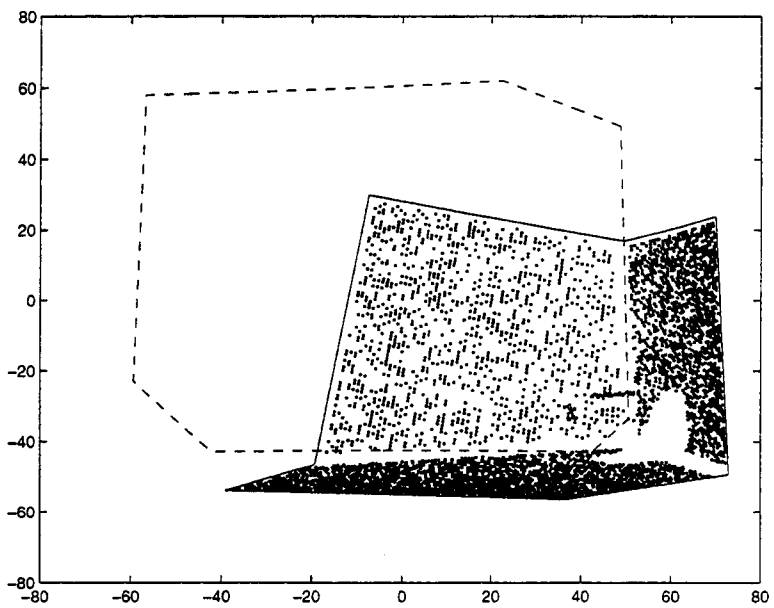
and

$$t_{res} = t_{alg} - t_{act}.$$

As seen in figure 6.2, the closest point method produces acceptably accurate transformations for small amounts of rotation and translation, before gradually breaking down. In comparison, the results shown in Figure 6.3 reveal that the transformations recovered by the point-to-surface method on the same images are very accurate for small rotations and translations, and then breakdown suddenly. The results seen after the breakdown of the algorithm are a result of the correspondence method finding no corresponding planes between the images, leaving no data to compute a transformation. As a result, the transformation produced by the algorithm is simply the motion estimate computed for the previous iteration of the algorithm, and therefore offers no meaningful information. Figure 6.4 is an illustration of the breaking point for the point-to-surface results shown in the graphs at the top of Figure 6.3. Each graph in the figure depicts the border for each image, as well as the location of all control points selected. Part (a) shows the circumstances for the test case where θ_x and θ_y have been rotated 4° off from the actual transformation; a transformation recovered by the algorithm. Part (b)



(a)



(b)

Figure 6.4. Illustration of breakdown of point-to-surface method. The first graph shows the two images and control points for the case where θ_x and θ_y are rotated 4° off from the actual rotation. The second graph is for a rotation of 5° off.

shows the relative positioning for the two images and control points for the test case where the initial estimate has been rotated 5° off from the actual transformation; a transformation the algorithm does not recover. In Part (a), the overlap of the images clearly contains control points lying on all three surfaces in the scene, the two walls and the floor. In part (b) however, the overlap contains control points on only one surface. The overlap of the two images has lost all 3D information from the first image, resulting in the algorithm computing false correspondences that move the transformation further off at each iteration until, ultimately, no correspondences can be found.

Returning to the closest point method, the results shown in Figure 6.5 represent experiments performed on the original 256×256 images captured by the simulator, before they were sub-sampled. The first thing to note is the scaling of the graphs in Figure 6.5 as opposed to that of the graphs in Figure 6.2. The graphs for the 256×256 images are scaled from $[\ -60° \ldots 60°\ ]$ for rotation errors and $[\ -300 \ldots 300\ ]$ for translation errors. These are six times the scales necessary for the results using the 128×128 images. The closest point algorithm using the 256×256 images is not able to recover the transformation when given the exact transformation as the initial estimate. The graphs in Figure 6.5 do show that the algorithm does recover certain transformations fairly accurately, but when viewing the results overall, these recovered transformations are merely a case of random chance. The closest point algorithm does not work for the 256×256 images because the points are too densely sampled within too narrow a field of view. As a result, it is more likely that the algorithm will compute

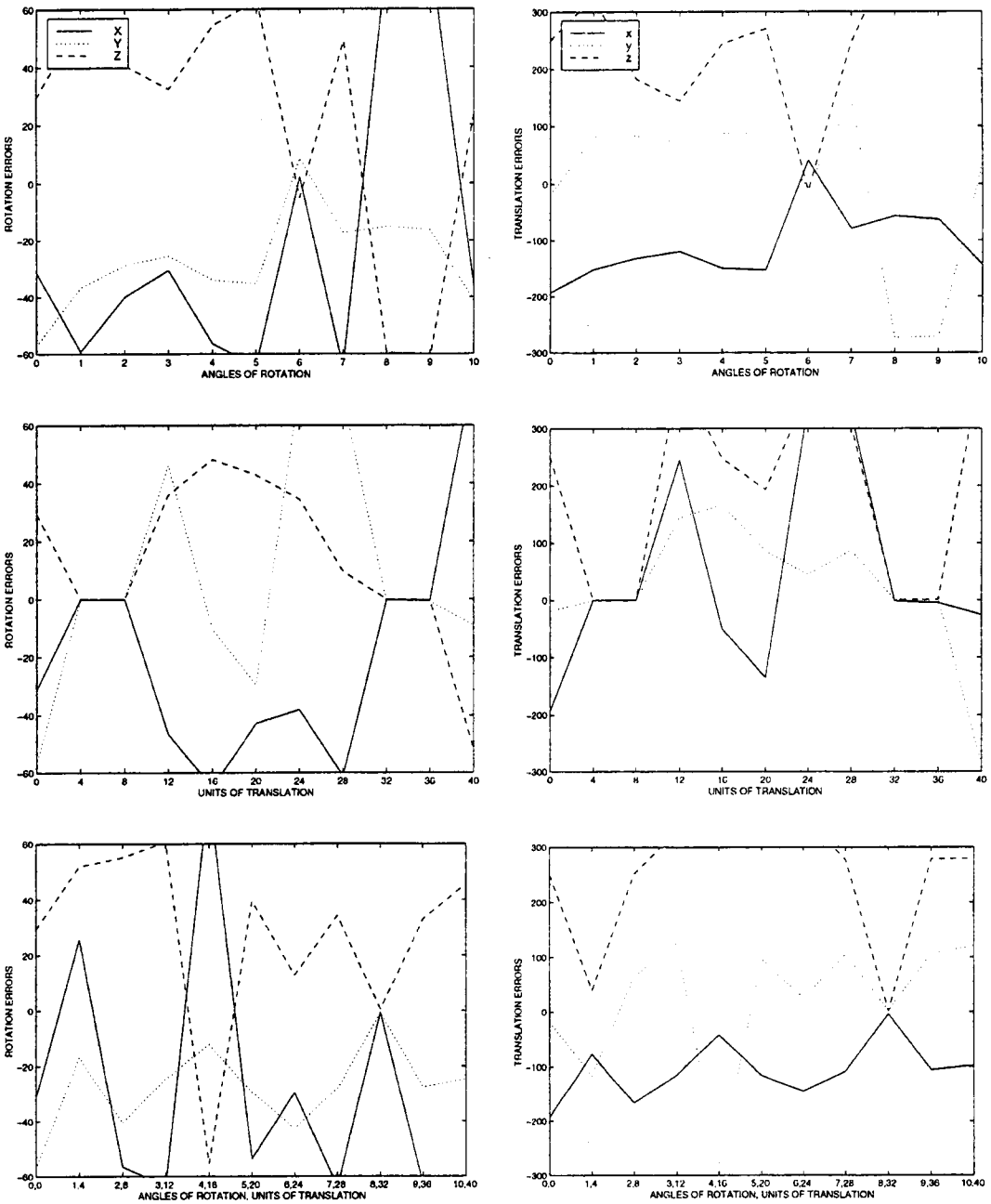


Figure 6.5. Results for closest point correspondence on images of size 256 × 256. The graphs in the left column show the rotation errors for the transformation produced by the algorithm while the right column shows the translation errors.

false matches given so many points in such a small area. For this reason, the captured images were uniformly sub-sampled by taking every other point from every other row, creating the 128×128 images used for these experiments.

All the results presented to this point have been computed by letting the algorithms iterate to a maximum of 40 iterations. The results shown in Figure 6.6 represent experiments where $\tau = 1\%$, meaning the iteration was terminated when the relative change in rotation and translation between successive iterations was less than 1%. The use of τ as an iteration termination condition was described in Section 1.2.4. The results are very different from those seen in Figure 6.2, where the tests were all carried out to 40 iterations. In most cases, the iteration halted prematurely and produced a less accurate transformation than those resulting from continuing to 40 iterations. The 1% value for τ was suggested by Zhang, only he was calculating the relative difference from the rotation axis representation derived from a related quaternion in contrast to the angles of rotation used in this application. Further experimentation should be performed to possibly determine a value for τ appropriate to using the angles of rotation to compute the relative difference at each iteration.

Two separate images of the scene in Figure 6.1, with views similar to the previous two, were captured using a field of view at 45° as opposed to the previous 20°. These images were also created at a size of 256×256 and subsequently sub-sampled to 128×128. The results recovered by the closest point and point-to-surface methods on these images are given in Figures 6.7 and 6.8 respectively. The idea for using the larger field of view

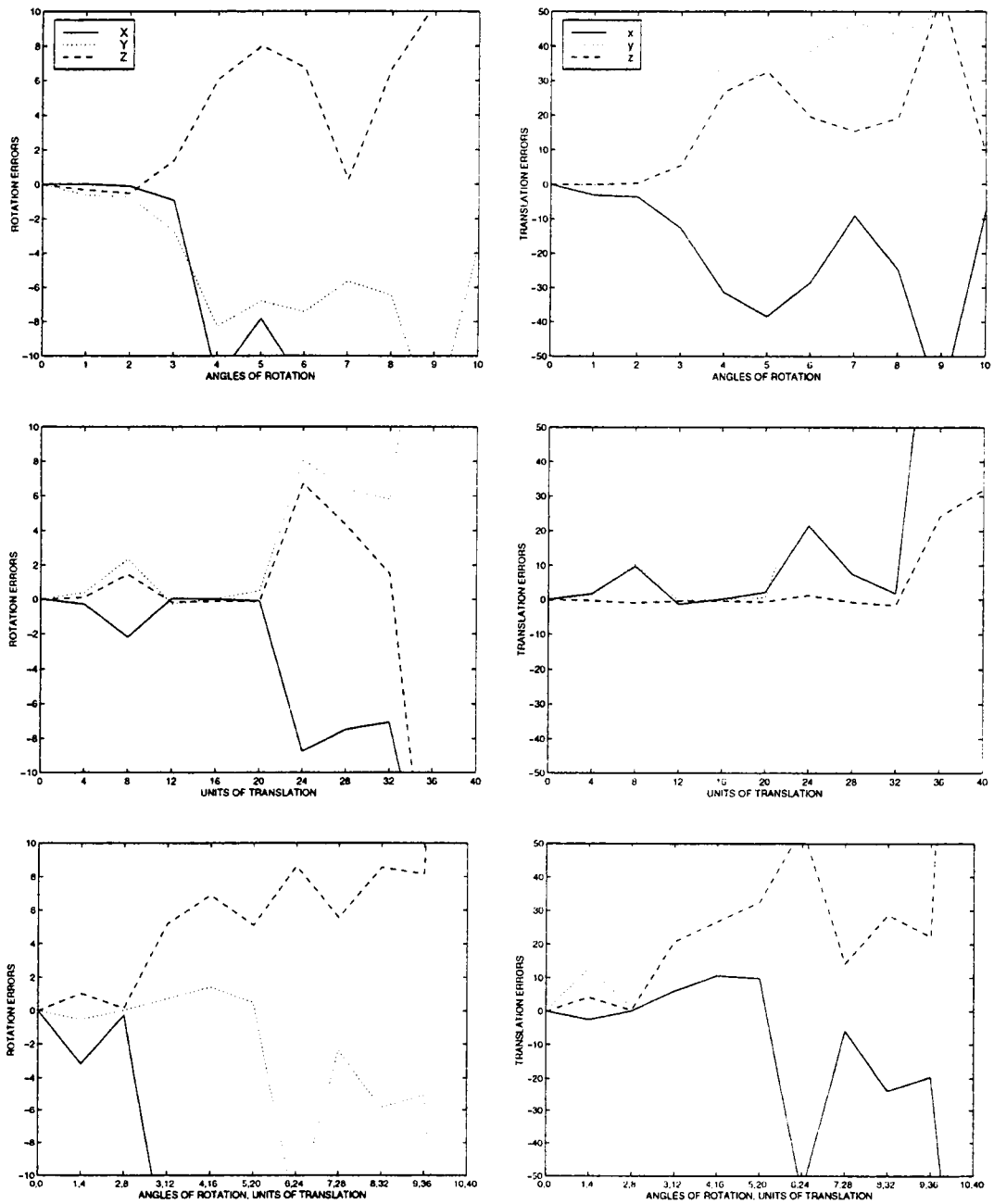


Figure 6.6. Results for closest point correspondence using $\tau = 1\%$. The graphs in the left column show the rotation errors for the transformation produced by the algorithm while the right column shows the translation errors.

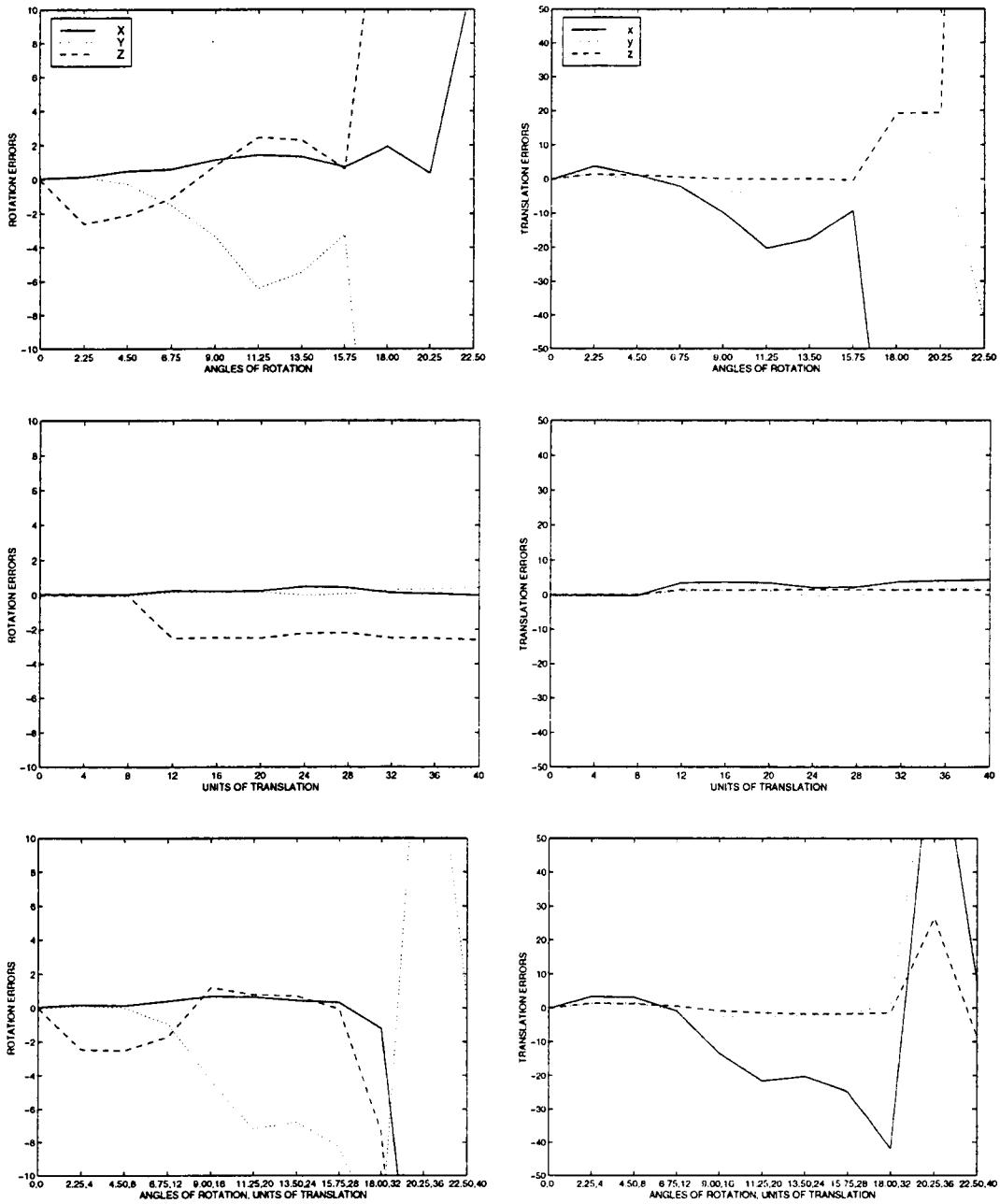


Figure 6.7. Results for closest point correspondence method on images with a field of view of 45°. The graphs in the left column show the rotation errors for the transformation produced by the algorithm while the right column shows the translation errors.

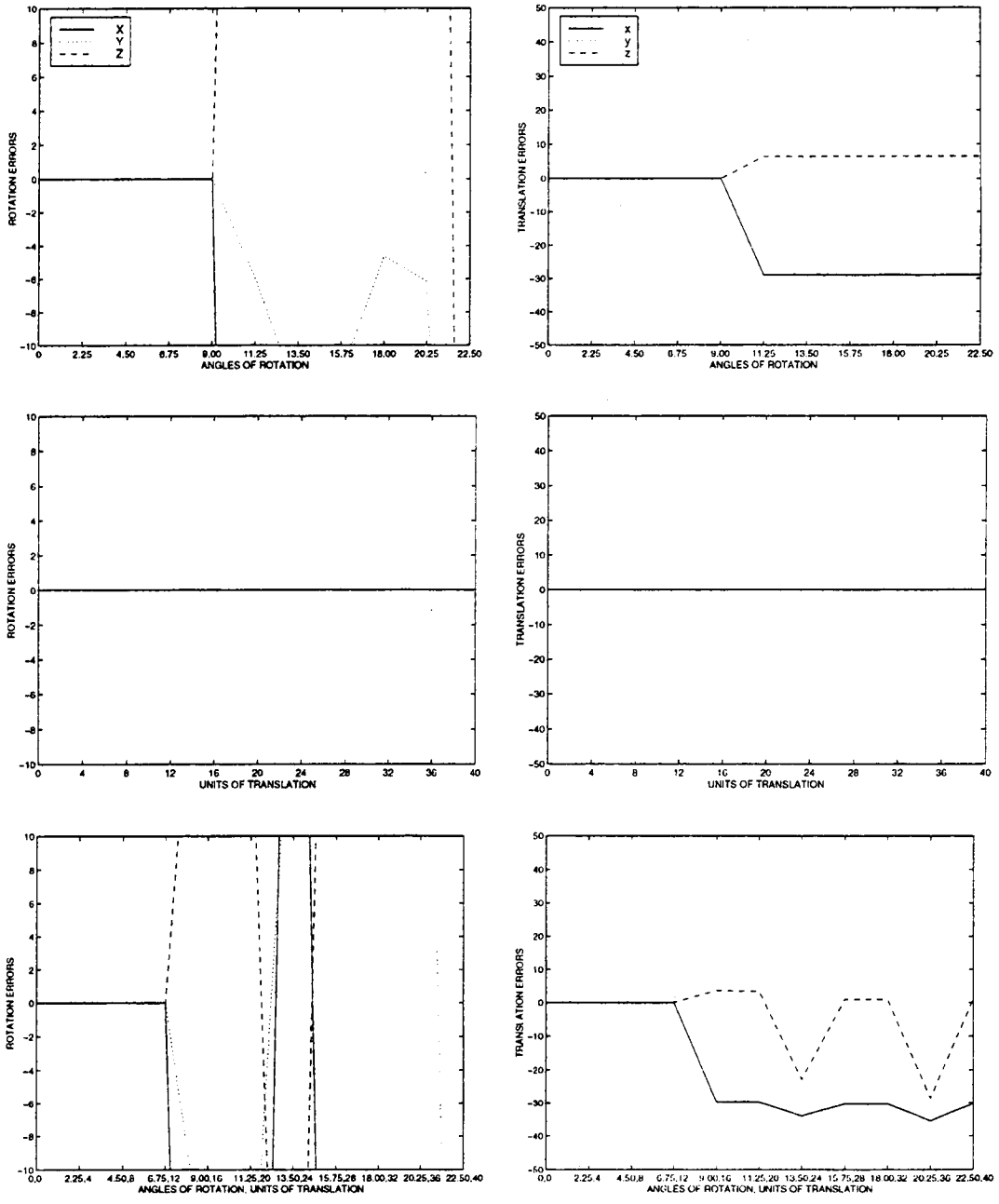


Figure 6.8. Results for point-to-surface correspondence method on images with a field of view of 45°. The graphs in the left column show the rotation errors for the transformation produced by the algorithm while the right column shows the translation errors.

with the closest point algorithm is that the less dense the sampling is, the less likely the algorithm will compute an incorrect correspondence by finding a point closer than the true correspondence. For the point-to-surface algorithm, the hypothesis is that the solution to the least squares fit of each 5×5 plane will be more accurate with the points lying farther apart. The second thought is that the algorithm will recover rotations of up to 20% of the field of view used to capture the image. For these two examples, that seems like a reasonable hypothesis, as the 20° field of view recovered up to 4° of rotation, and as seen in Figure 6.8 the 45° field of view recovered rotations up to 9°. Given that all the literature on the subject suggest that these methods are for fine tuning motion estimates, 20% of the field of view is reasonable to consider within the scope of fine tuning.

6.2 Synthetic Data with Added Noise

To move towards more realistic data conditions, zero mean Gaussian noise with variable standard deviation was added to each image. The noise was randomly generated using the Box_Muller method. The noise was added to the individual data points in the images by making use of the fact they were originally generated by a simulator imitating a spherical coordinate system. Thus, each rectangular coordinate (x, y, z) was transformed to its equivalent spherical coordinate (ρ, ϕ, θ), and a randomly generated amount of noise was added to each ρ coordinate. Lastly, each spherical coordinate is transformed back into a rectangular coordinate, yielding a fairly accurate representation

of noise within an image.

Experiments were performed on images perturbed by noise with a standard deviation, σ, ranging from 10^{-5} to 1.0, using both the closest point and point-to-surface methods. Given that the r_{ij}'s (distance from camera to scene) for both images are in the range of [175 .. 330], even $\sigma = 1.0$ is considered a negligible amount of noise. For the point-to-surface method, a noise level with $\sigma = 10^{-5}$ did not alter the results seen in Figure 6.3 for the no noise case. However, for noise levels with $\sigma = 10^{-4}$ or greater, the point-to-surface method failed to produce meaningful results, even when given the exact transformation as the initial estimate. There are many possible reasons for this, including the sensitivity of the least squares method used to compute each plane description and the difficulty in setting the outlier detection parameters to eliminate false correspondences.

The closest point method performed much better on the noisy data, as can be seen in Figure 6.9 showing the results for a noise level of $\sigma = 1.0$. The results are comparable to those seen in Figure 6.2 for the no noise case. Both sets of results show the closest point method breaking down gradually as the initial estimate is moved farther and farther off. For this case, using these images of a box in a corner, this level of noise does not appear to significantly alter the transformations produced by the algorithm.

The experiments for the images with noise were next performed on the images captured with the 45° field of view. The closest point algorithm results were similar to those seen for the images using a 20° field of view. The results for the point-to-surface

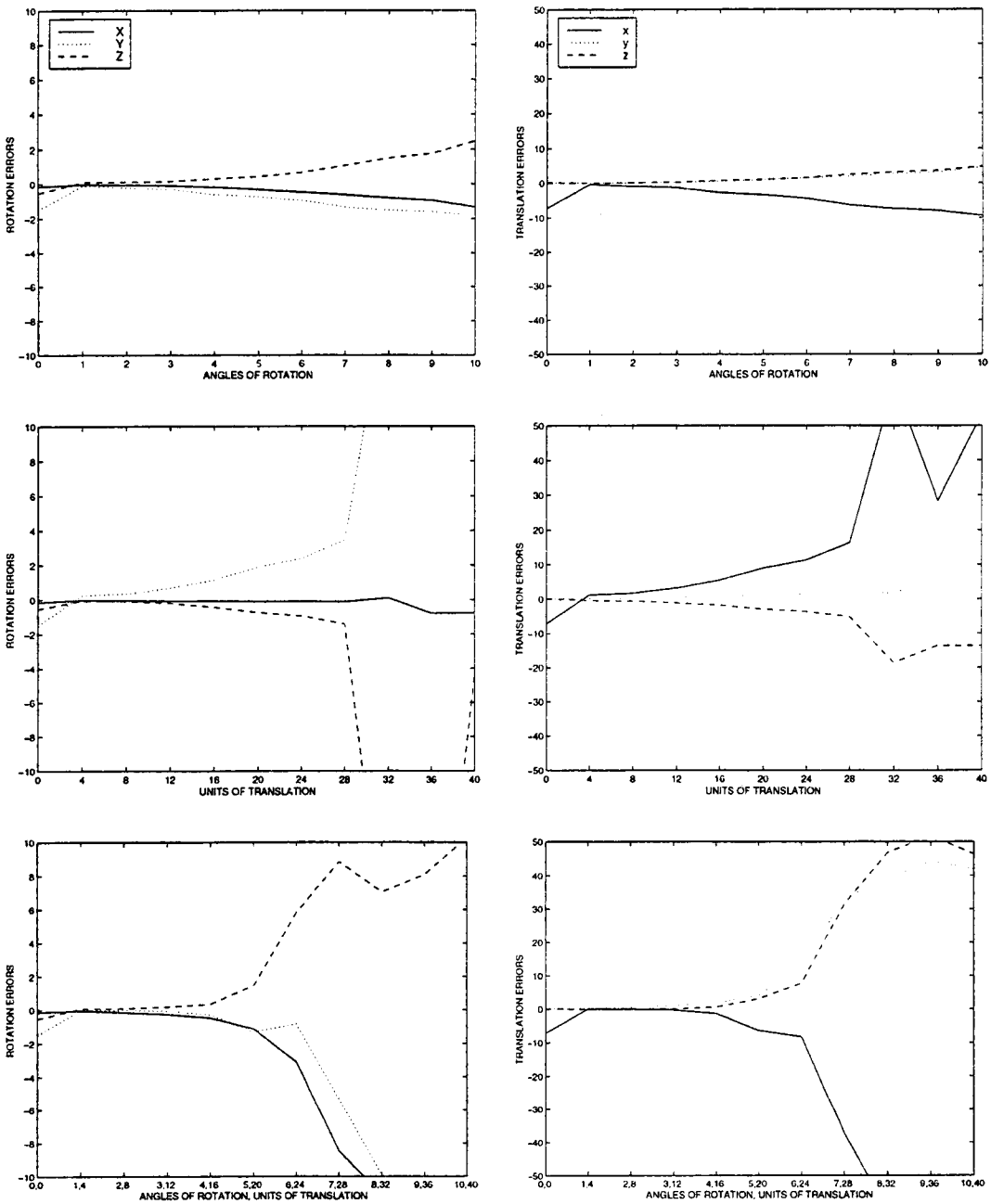


Figure 6.9. Results for closest point correspondence on images perturbed by noise with $\sigma = 1.0$. The graphs in the left column show the rotation errors for the transformation produced by the algorithm while the right column shows the translation errors.

algorithm however were much improved over those seen with the smaller field of view. For these images, the algorithm recovered transformations with noise levels of up to $\sigma = 0.01$. These results better relate to the data presented in Figure 6.10 which shows contour plots of the normal vectors and histograms of the offsets for the 20° field of view images. The top graphs show the normal vectors and offsets for the images with no noise, while the middle graphs are for noise levels with $\sigma = 0.001$ and the bottom graphs with $\sigma = 1.0$. The problem is raised from the middle graphs of Figure 6.10. The data depicted does not explain why the transformations with this level of noise are not recovered using the 20° field of view images. From the graphs at the bottom of Figure 6.10 it is understandable that the point-to-surface method would have a difficult time recovering the correct transformation given the spreading out of the normal vectors and offsets.

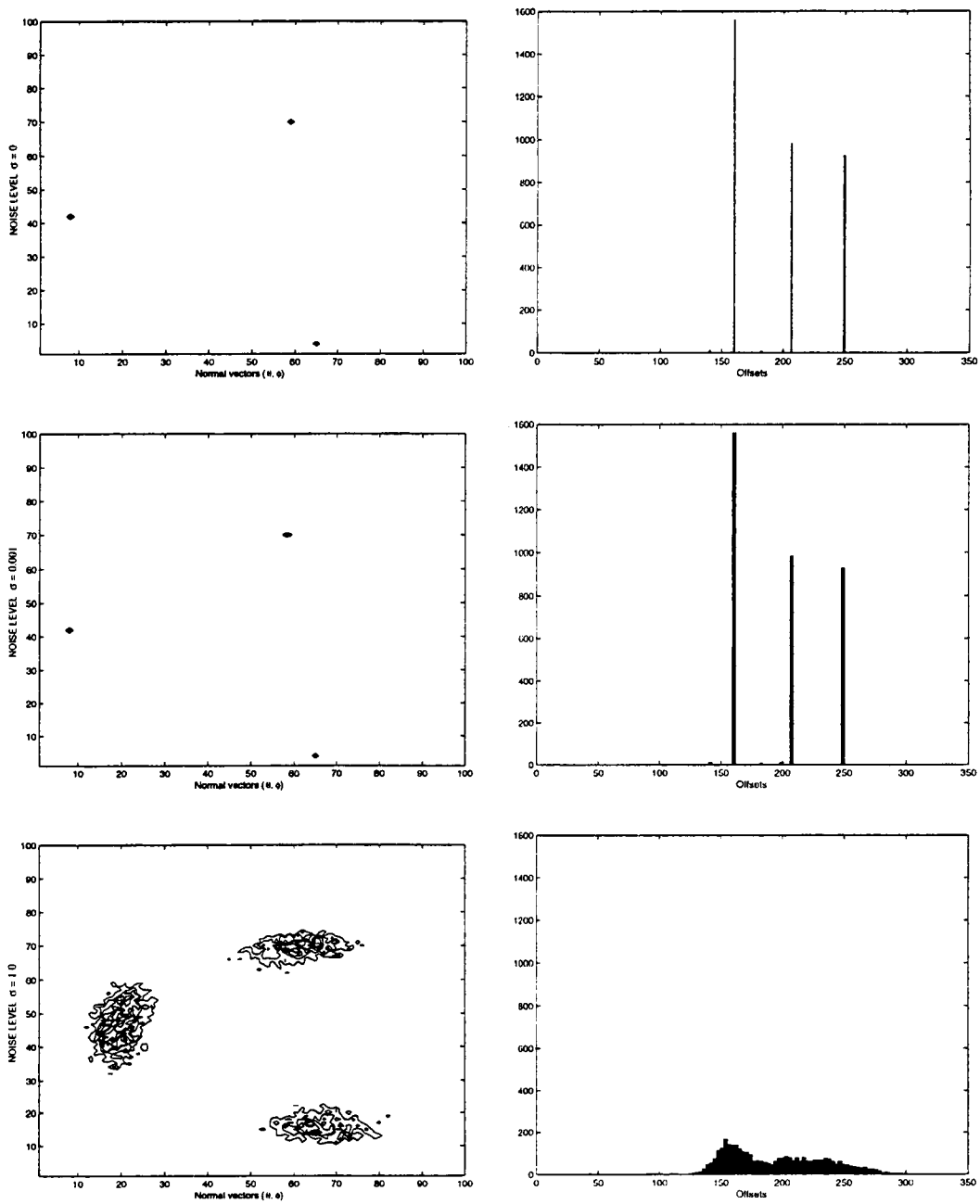


Figure 6.10. Results for normal vectors and offsets on images perturbed by noise with $\sigma = 0.001$ and $\sigma = 1.0$. The graphs in the left column show a contour plot of the normal vectors while the right column shows a histogram of the offsets.

Chapter 7

Conclusions and Future Research

This paper has described several algorithms used to register two sets of raw 3D data points obtained through the use of a laser range finder. The registration is accomplished by first computing correspondences and then computing motion estimates based on those correspondences. For the closest point method, the adaptation of the dynamic kD tree significantly reduces the time required to search for closest points, making the closest point method more feasible for use with larger size point sets. There has been a connection established between how densely the point set is sampled and the ability of the closest point method to produce accurate transformations. The relationship is between the field of view and the size of the image. The development of the plane-to-plane motion estimation technique allows for the near exact recovery of transformations for images without noise and with small amounts of noise. The closest point method appears to yield more consistent results than the point-to-surface method, but that is

only for the particular images reported. For two different views of the same scene, the closest point could not recover the correct transformation even when given the exact transformation as the initial estimate, while the point-to-surface method performed better than on the two views presented in Chapter 6. At this stage of the experimentation, neither correspondence method can be proven more or less reliable than the other.

The establishment of correct correspondences is the most difficult of the two tasks and is where most of the problems are found. There are a number of parameters whose settings greatly affect the transformation produced by the registration. Among these, the termination condition for the ICP algorithm, which as seen can produce vastly different results depending on the value assigned. Within the area of establishing correspondences, there are many possible areas for improvements. Future work should focus on improving the described outlier detection techniques as well as developing others. Outliers have a significant impact on the registration process, as just a handful thereof is enough to distort the least squares solution. Additional experimentation needs to be performed to discover dynamic methods for setting outlier detection parameters such as D for closest point and the angle between normal vectors for the point-to-surface method, based on the data of each scene. The more accurate the outlier detection, the more accurate the registration process will be.

The motion estimate techniques described in this paper are all robust methods given accurate correspondence information. There are other motion estimation techniques such as least median squares that are not as sensitive to incorrect correspondences,

although they are far more complex and require more computational time. The real problem lies in the accurate computation of correspondences across range images, hence further research is necessary to improve upon the outlier detection process.

Bibliography

Bibliography

[1] K S Arun, T S Huang, and S D Blostein. Least-squares fitting of two 3-D point sets. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 9:698–700, 1987.

[2] P J Besl and N D McKay. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 14:239–256, 1992.

[3] M Blum, R W Floyd, V Pratt, R L Rivest, and R E Tarjan. *Time Bounds for Selection.* Science - Engr., 1973.

[4] Y Chen and G Medioni. Object modeling by registration of multiple range images. *Int. J. Image and Vision Computing*, 10:145–155, 1992.

[5] D W Eggert, A Lorusso, and R B Fisher. Estimating 3D rigid body transformations: A comparison of four major algorithms. *Machine Vision and Applications*, 9:272–290, 1997.

[6] B K P Horn. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Amer. A*, 4:629–642, 1987.

[7] F Preparata and M Shamos. *Computational Geometry, An Introduction.* Springer-Verlag, 1986.

[8] W H Press, S A Teukolsky, W T Vetterling, and B P Flannery. *NUMERICAL RECIPES in C: The Art of Scientific Computing.* Cambridge University Press, 1992.

[9] B Sabata and J K Aggarwal. Estimation of motion from a pair of range images: A review. *CVGIP: Image Understanding*, 54:309–324, 1991.

[10] V Sequeira, J G M Goncalves, and M I Ribeiro. 3-D environment modeling using laser ranger sensing. *Robotics and Autonomous Systems*, 16:81–91, 1995.

[11] M W Walker, L Shao, and R A Voltz. Estimating 3-D location parameters using dual number quaternions. *CVGIP: Image Understanding*, 54:358–367, 1991.

[12] Z Zhang. Iterative point matching for registration of free-form curves and surfaces. *Int. J. Computer Vision*, 13:119–152, 1994.

Vita

James G. McGreger was born in Riverside, California on Tuesday, October 13, 1970. He graduated from Riverside Poly High School on June 23, 1988 as a lifetime member of the California Scholastic Federation. He entered Riverside Community College in September 1988 where, after changing majors six times, he received an Associate in Science Degree on June 16, 1992. After taking a year off of school in which his family moved to Savannah, Georgia, he entered Georgia Southern University in September 1993 as a Computer Science major, where he received the Computer Science Outstanding Senior Award from the Department of Mathematics and Computer Science at Honors Day on May 1, 1996. He graduated Georgia Southern with a Bachelor of Science in Computer Science and a minor in Mathematics with Cum Laude Honors on June 9, 1996. After working that summer at Gulfstream Aerospace, he entered the Master's program in Computer Science at the University of Tennessee, Knoxville in August 1996 working as a Teaching Assistant. He worked as a Research Assistant for Dr. Jens Gregor studying Three-Dimensional Scene Modeling from January 1997 through December 1998, officially receiving the Master of Science Degree on December 20, 1998.