Real-time 3-D Pose Estimation Using a High-Speed Range Sensor

David A. Simon, Martial Hebert and Takeo Kanade

CMU-RI-TR-93-24

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

November 1993

© 1993 Carnegie Mellon University

Table of Contents

Abstract

This report describes a system which can perform full 3-D pose estimation of a single arbitrarily shaped, rigid object at rates up to 10Hz. A triangular mesh model of the object to be tracked is generated offline using conventional range sensors. Real-time range data of the object is sensed by the CMU high speed VLSI range sensor. Pose estimation is performed by registering the real-time range data to the triangular mesh model using an enhanced implementation of the Iterative Closest Point (ICP) Algorithm introduced by Besl and McKay. The method does not require explicit feature extraction or specification of correspondence. Pose estimation accuracies on the order of 1mm in translation and 1 degree in rotation have been measured.

1.0 Introduction

The problem of determining the 3-D pose of a rigid object at high speed has been approached by a number of researchers [10][12]. However, there are few systems capable of full 3-D pose estimation of arbitrarily shaped objects in real-time. There are three reasons why this goal has been difficult to attain. First, the 2-D data provided by conventional video cameras lacks the sensitivity required for *accurate* 3-D pose estimation of arbitrarily shaped objects. Second, many approaches to 3-D pose estimation require two operations which are difficult to perform: feature extraction and correspondence specification. Third, in order to perform 3-D pose estimation in real-time, each step in the underlying algorithm must be computationally efficient.

Direct use of 3-D data simplifies the pose estimation problem by providing shape structure which would otherwise need to be inferred from 2-D data. As noted in [12], while 2-D data is useful for estimating object motion in planes normal to a camera's optical axis, it is less sensitive to motions which deviate from these planes. Direct use of 3-D data should provide more precise object pose estimates, especially for general 3-D motions.

Many previous approaches to 3-D pose estimation are feature based [8][10][12]. Such approaches, however, suffer from some common difficulties. Typically, the steps in feature based pose estimation are: 1) extract features such as points or lines from the underlying data; 2) specify correspondence between data and model features; 3) compute the pose estimate from the derived correspondence. Unfortunately, the extraction of reliable features from images of real-world objects is difficult. Even when such features can be found, solution of the correspondence problem can be complex and computationally expensive.

In our approach, raw range data points which lie on the surface of the tracked object are matched to the underlying object surface model using an iterative least squares technique (the ICP algorithm). This approach eliminates the need to perform any feature extraction, or to specify feature correspondence.

To our knowledge, no previous approaches have succeeded in combining both high speed acquisition of 3-D data with high speed 3-D pose computation. Several researchers have utilized range data in the 3-D pose estimation problem [8][13]. Yamamoto [13] discusses a system for estimating the shape and pose of deformable objects using a video rate range camera, but the required computations are not performed at high speed.

The remainder of this paper is organized as follows. Section 2.0 describes the Iterative Closest Point algorithm and enhancements which allow it to be used for real-time pose estimation. Section 3.0 outlines the algorithm for real-time pose estimation. Section 4.0 describes the experimental setup used to demonstrate the approach. Section 5.0 contains experimental results, and Section 6.0 contains the conclusion.

2.0 Registration

The registration algorithm used in this system is strongly motivated by the work of Besl and McKay [2]. Their paper describes a general purpose method for the registration of rigid 3-D shapes which they refer to as the Iterative Closest Point algorithm. Zhang [14] has independently developed a similar algorithm which is better at handling outliers and occlusions in the data. Since these were not a major concern in our work, the formulation presented below parallels that of Besl and McKay.

2.1 The ICP Algorithm

Suppose that we have two independently derived sets of 3-D points which correspond to a single shape. We will call one of these sets the *model* set *M*, and the other the *data* set *D*. Assume that for each point in the data set, the corresponding point in the model set is known. The problem is to find a 3-D transformation which when applied to the data set *D*, minimizes a distance measure between the two point sets. The goal of this problem can be stated more formally as follows:

$$\min_{\boldsymbol{R},\,\boldsymbol{T}} \sum_{i} \left\| M_i - (\boldsymbol{R}D_i + \boldsymbol{T}) \right\|^2 \tag{1}$$

where $\boldsymbol{R}$ is a 3x3 rotation matrix, $\boldsymbol{T}$ is a 3x1 translation vector, and the subscript i refers to corresponding elements of the sets M and D as shown in Figure 1. Efficient, non-iterative solutions to this problem, both employing unit quaternions, were presented in two papers, one by Faugeras and Hebert [4] and the other by Horn [7].

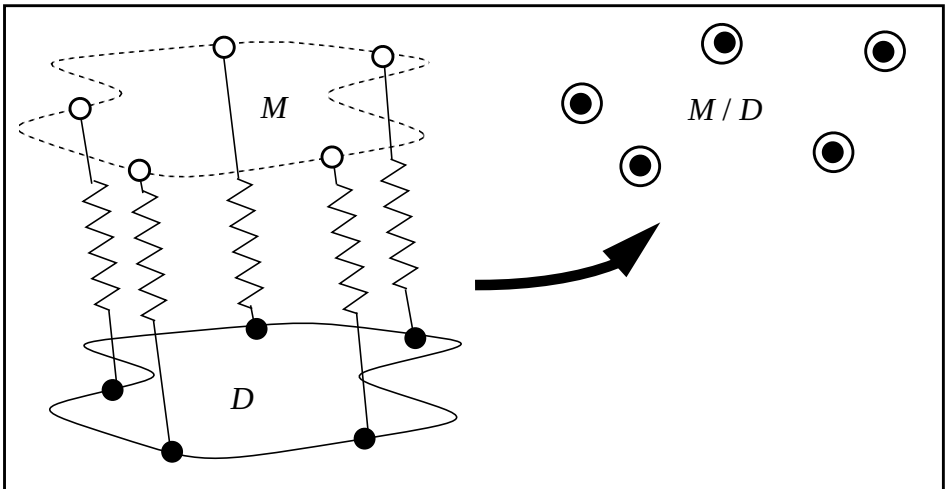

Figure 1: Corresponding Point Set Registration

The general 3-D shape registration problem that we address here, however, differs from the corresponding point set registration problem in two important regards. First, the point correspondence which was assumed to be known in the above problem is unknown in the general

case. Second, general 3-D shapes to be registered are not necessarily represented as point sets [2].

Suppose that we are again given two sets *M* and *D* corresponding to a single shape, where *D* is a set of 3-D points and *M* is a triangular faceted surface. Assume that the correspondence between points in the two sets is initially unknown. As seen in Figure 2, for each point D_i from the set *D*, there exists at least one point on the surface of *M* which is closer to D_i than all other points in *M*. This is the *closest point*, M_i.

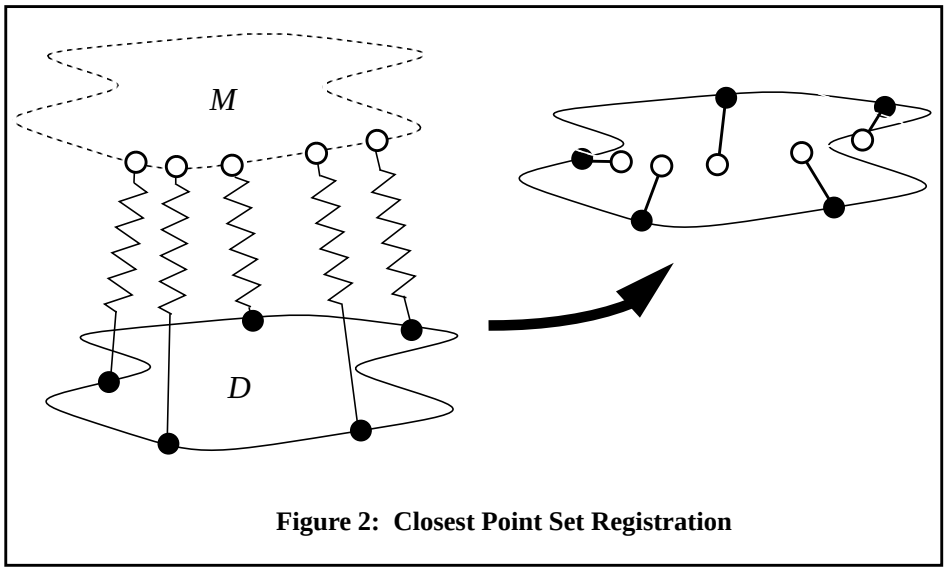

Figure 2: Closest Point Set Registration

The basic idea behind the ICP algorithm is that under certain conditions, the point correspondence provided by sets of closest points is a reasonable approximation to the true point correspondence. Besl and McKay proved that if the process of finding closest point sets and then solving equation (1) is repeated, the solution is guaranteed to converge to a *local* minimum. The ICP algorithm can now be stated:

1. For each point in *D*, compute the closest point in *M*
2. With the correspondence from step 1, compute the incremental transformation (***R***, ***T***) [equation (1)].
3. Apply the incremental transformation from step 2 to the data *D*.
4. Compute the change in total mean square error. If the change in error is less than a threshold, ε, terminate. Else goto step 1.

While the ICP algorithm is only guaranteed to converge to a local minima, there is no guarantee that this local minima will correspond to the actual global minima. How well the algorithm performs is a function of the initial pose estimate and the characteristics of the shape being registered. Besl and McKay discuss in detail the problem of finding the global minimum in situations where initial pose error is large. We have found that the ICP algorithm converges to the global minimum even with fairly large initial pose discrepancies. For the purposes of the system described in this paper, the initial pose discrepancies are usually small.

2.2 Speed Enhancements to ICP

A basic implementation of the ICP algorithm lacks the speed required to perform pose estimation in real-time. We have implemented several enhancements: kd-trees, closest point caching, efficient point to surface computation, and acceleration.

2.2.1 Kd-trees

The most computationally expensive step in the ICP algorithm is finding the closest point sets. In general if there are N_D points in the data set and N_M geometric entities (i.e.: points, lines, triangles) in the model set, then the complexity of the closest point computation is $O(N_D N_M)$. However, as suggested in [2] and demonstrated in [14], this complexity can be reduced to $O(N_D \log N_M)$ by the use of a k-dimensional binary tree, or simply kd-tree [1]. The use of kd-trees for closest point computation allows us at each node of a binary tree to decide which side of a hyperplane the closest point will lie on. Thus, large regions of the search space can be pruned at each level in the search. We have implemented a closest point algorithm based on the kd-tree [5]. We have found that the actual performance improvement approaches that predicted by theory.

2.2.2 Closest Point Caching

A second small speed improvement was realized by caching closest points. Points in the sets *M* and *D* which are proximal at time *k,* are highly likely to be proximal at time *k*+1. Thus, rather than finding the single closest point in *M* for a given point $D_i[k]$, we can find *n* closest points in *M* and cache these points together with the point $D_i[k]$. Note that there is little overhead involved in finding *n* closest points when *n* is a small number like 5. On the next iteration, since the point $D_i[k+1]$ is likely to be close to the point $D_i[k]$, it is also likely that the closest point in *M* to $D_i[k+1]$ will be one of the points cached on the previous iteration. It is possible to determine conclusively whether the closest point is contained in the cached set by performing a simple test. This test compares the magnitude of the previous incremental transformation to the distance between the closest cached point and the *n*th closest cached point (where *n* is the number of cached points). A variation on this test can also determine whether the closest point at time *k*+1 is the *same* as the closest point at time *k.* The overall result of caching is that closest points can often be found without requiring a full search of the kd-tree. Rather, only the points in the cached set must be tested.

A similar caching technique can be applied to *spatially* (rather than *temporally*) adjacent points. If two data points $D_i[k]$ and $D_{i+1}[k]$ are proximal, then it is likely that their corresponding closest points $M_i[k]$ and $M_{i+1}[k]$ will also be proximal. An analogous caching technique can be applied for this situation, however we have not yet implemented caching for spatially adjacent points.

2.2.3 Closest *Surface* Point Computation

When *M* is a triangular faceted surface, computation of the closest point requires an additional step. The output of the kd-tree based closest point algorithm will return the closest *vertex* V_i on the surface of *M*, as shown in Figure 3. Given V_i, the closest point M_i will lie within, or on the border of, one of the triangles to which the vertex belongs[1]. In order to find M_i, D_i is

1. This is not strictly true, as there are pathological cases for which M_i will lie in a totally different triangle. In our experience, we found that we can ignore such cases.

projected into the plane of each triangle, and the closest point between D_i and that triangle is computed. This is repeated for all triangles containing V_i, and the overall closest point is selected. In order to perform these computations quickly, once D_i is projected into the plane, all computations are performed in 2-D rather than 3-D. Thus, during initialization each triangle must be saved in both its 2-D and 3-D representations.

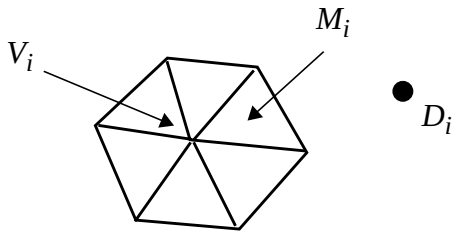

Figure 3: Closest Facet Point Computation

2.2.4 Acceleration

A final speed improvement was realized using a modified version of the *accelerated* ICP algorithm described in [2]. The accelerated ICP algorithm adds the following step to the basic algorithm (after step 2):

2b. If the incremental transformations (***R***, ***T***) at times *k*-1, *k*-2, and *k*-3 are *well aligned*, extrapolate the current incremental transformation.

The well aligned condition above tests that the solution has been moving in an approximately constant direction. Extrapolation is performed by scaling the current incremental transformation. The scale factor is a function of the mean square error and the magnitude of the incremental transformations at the previous three iterations.

Besl and McKay calculate a single acceleration scale factor for both translation and rotation. We achieved better results by decoupling the acceleration of translation and rotation. There are two reasons for doing this. First, in Besl's approach, the well aligned condition above is tested once for both rotation and translation. Thus, for example, if rotation was well aligned but translation was not, no acceleration would be performed. However, an acceleration on rotation alone seems desirable in this situation. A second reason for decoupling is related to the scale factor used in extrapolation. Besl and McKay used the same scale factor to extrapolate both rotation and translation components. This scale factor is designed to extrapolate the solution as much as possible in a single step without overshoot. In the coupled version, the size of the scale factor is governed by the component (translation or rotation) which would cause the solution to overshoot first. The other component could usually be accelerated further. By decoupling, translation and rotation are independently accelerated as much as possible without overshoot.

2.2.5 Enhancement Results

Four speed enhancements were described in this section: closest point computation via kd-trees, closest point caching, efficient computation of closest facet points, and decoupled acceleration. The results of applying each of these enhancements to a single registration problem are summarized in Table 1. In this problem, *D* was a point set containing 2432 points and

M was a triangular mesh containing 4860 facets. The initial pose error was roughly 10 degrees of rotation about each axis, and about 10% of object size in each translation. The ICP termination threshold, ε, was small.[1]

Type	Time	%T	Iter	R-Acc	T-Acc
none	908.8	100.0	122	0	0
a	261.2	28.7	35	11	11
kd	62.2	6.8	122	0	0
kd/a	18.0	2.0	35	11	11
kd/a/d	13.1	1.4	25	13	7
kd/a/d/c	11.9	1.3	25	13	7
kd/a/d/c/2d	8.3	0.9	25	13	7

Table 1: Enhancement Comparisons

In the table, *Type* indicates the enhancements used: a - coupled acceleration; kd - kd-tree search; d - decoupled acceleration; c - closest point caching; 2d - 2d calculation of closest facet points. *Time* is the total ICP execution time in seconds. *%T* is percentage of time relative to the slowest time. *Iter* is the number of ICP iterations. *R-Acc* and *T-Acc* are the number of accelerations for rotation and translation respectively.

The speed improvements shown in Table 1 give an idea of the relative utility of each of the described enhancements. The actual relative utility is a function of the underlying data, the initial pose, and the termination threshold. Acceleration and kd-tree search are always the two most important enhancements. The relative utility of kd-tree search increases with the number of points in the data set. Caching is useful when the termination threshold is small, since the number of cache hits will be large during fine-tuning.

3.0 The Tracking Algorithm

An outline of the tracking algorithm is shown in Figure 4. Each box in the diagram represents a processing step, and the processing sequence is indicated by the large-headed arrows. Inputs to a processing step are indicated by the quantities to the left of each box, while outputs are indicated by the quantities to the right.

During initialization, a precomputed triangular mesh model, *M*, is loaded into memory, and a kd-tree is built from *M*. For our experiments, *M* is constructed offline using a technique based on deformable surfaces [3]. This technique can fuse range data collected from multiple views into a single triangular mesh surface model. The range data used to create *M* is provided by several commercially available light-stripe range finders [11]. These sensors have been calibrated so that all data points are expressed in a single, world-centered coordinate frame.

To initialize the tracking algorithm, the transformation between the model, *M*, and the initial object pose *D*[0], must be calculated. This transformation, ${}^{M}\boldsymbol{T}_{D}[0]$, can be found in several seconds using the ICP algorithm with a starting transformation provided by the user[2]. In prac-

1. The magnitude of ε determines the amount of "fine-tuning" performed by the ICP algorithm. Smaller values of ε result in pose estimates closer to the local minima.
2. A fully automated initialization which does not require user input would be possible by applying one of the techniques for solving the global pose estimation problem discussed in [2].

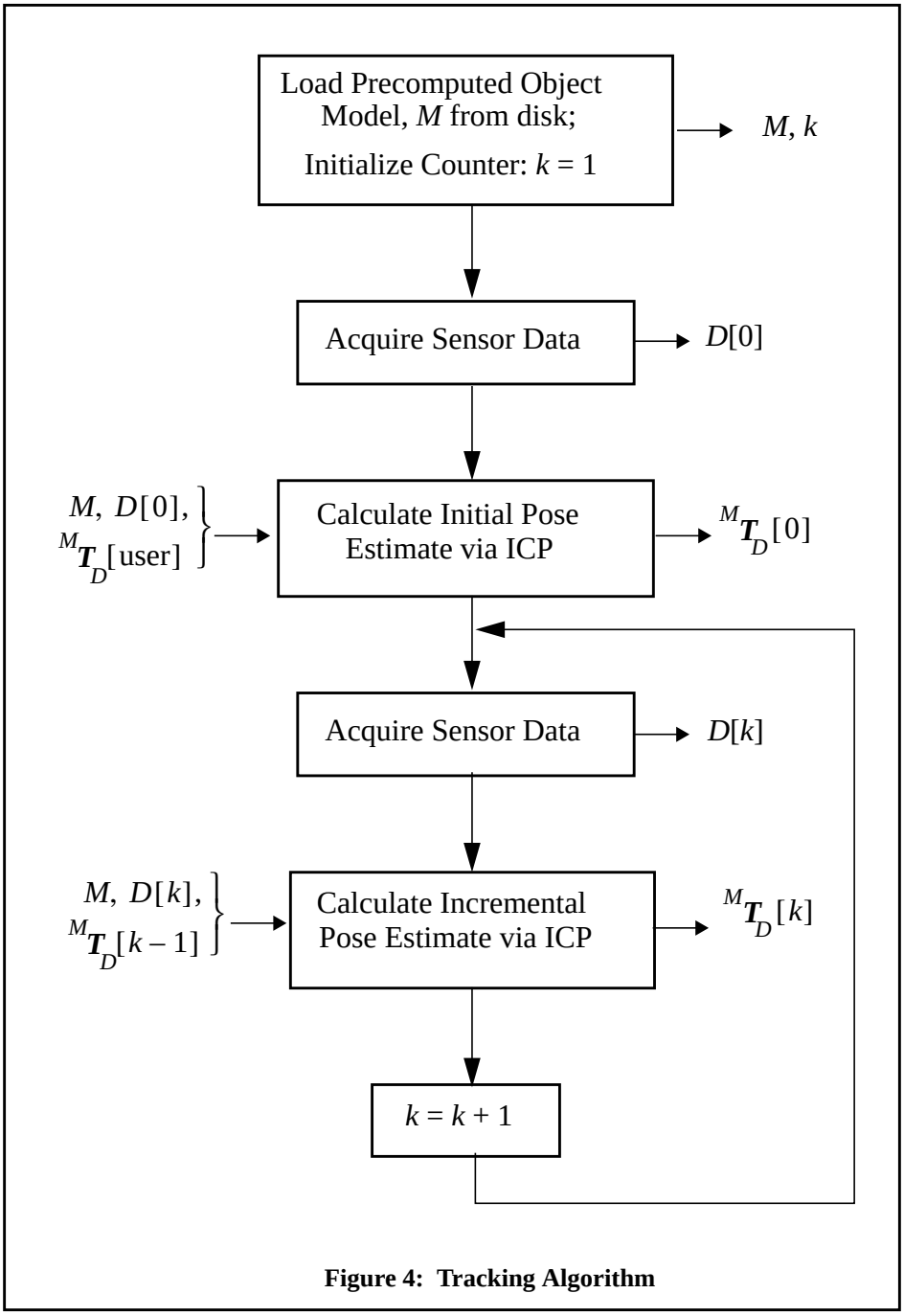

Figure 4: Tracking Algorithm

tice, we have found that initial pose errors as large as 15 degrees of rotation about each axis, and 50% of the object size in any translation will typically converge to the global minimum. Once ${}^{M}\boldsymbol{T}_{D}[0]$ has been calculated, it is used to transform the *model*, *M* to the initial object position. Thus, all future pose estimates are measured with respect to this initial starting position.

After initialization, the algorithm enters the tracking loop. Within the loop, data are acquired by the high speed range sensor, and the object pose is estimated via the ICP algorithm in roughly 0.1 - 0.3 sec. These high speeds are possible for two reasons. First, the difference in

object position at time *k* and time *k*-1 is typically small. For example, translational velocities of 10cm per second and rotational velocities of 20 degrees per second lead to incremental object pose discrepancies of roughly 2cm and 4 degrees. Thus, since the ICP algorithm uses ${}^{M}\boldsymbol{T}_{D}[k-1]$ as the starting point when finding ${}^{M}\boldsymbol{T}_{D}[k]$, the algorithm can perform the registration in a small number of iterations, typically 3-10. Second, the resolution of the range data used in the tracking loop, usually 16x16, is less than the full sensor resolution of 32x32. The reduced number of data points in the set $D[k]$ results in a faster calculation of the pose estimate.

During each data acquisition cycle, two simple preprocessing steps are performed on the range data. The first step eliminates noisy range data. For the CMU high speed range sensor, noisy data is associated with poor reflection of the projected light from the object. Thus, noisy range data can be eliminated by thresholding the reflected intensity values. Since each cell in the range sensor has circuitry for measuring intensity, this is a trivial operation. The second preprocessing step determines which range data points lie on the surface of the object to be tracked. Since our experiments are performed in an uncluttered environment, range data on the object surface can be distinguished by thresholding the Z component of the range data. While this simple operation works well for our experiments, a more sophisticated approach would be required if the object were in a cluttered environment.

Using ${}^{M}\boldsymbol{T}_{D}[k-1]$ as the starting point for incremental pose estimation works well when object motion is erratic and unpredictable. In some situations, however, object motion may be smooth, continuous and thus easier to predict. For such motions, improved results are possible using an extrapolation scheme such as a Kalman filter. While we have not implemented a Kalman filter for this purpose, we have implemented both first and second order extrapolation. Since the extrapolated pose is often closer to the true pose than ${}^{M}\boldsymbol{T}_{D}[k-1]$, the time required to compute the pose is reduced.

4.0 Experimental Setup

The experimental setup is shown in Figure 5. The CMU high speed VLSI range sensor developed by Gruss, Tada and Kanade [6] consists of two primary components: the sensor head and the light stripe generator. The tracked object, in this case a small bust of the Greek goddess Venus, is mounted on the end effector of a Microbot robot. The CCD imager is not a primary component of the system, but is used for display purposes only. Not shown is a Sparc-10 workstation used for computing the pose estimate, and for graphically displaying a 3-D model of the tracked object. The pose of the graphical 3-D model is updated at high speed to reflect the current object pose estimate.

The CMU high speed range sensor is based on a modified version of the traditional light-stripe range imaging technique known as the cell-parallel light-stripe method. The primary advantage of the cell-parallel method is that range image acquisition time is made independent of the number of data points in each frame.

The current version of the CMU range sensor can acquire a complete 32x32 cell range image in as little as one millisecond. The range data is acquired at 10 bits of resolution, and is accurate to 0.1% or better (0.5mm at 500mm). The sensor workspace is shaped like a four sided pyramid. As currently configured, at a distance of 55cm from the sensor along the optical axis, a cross section of the workspace is an 11.5cm square. Thus, the sensor resolution at this distance is about 2.8 range measurements per cm in each direction.

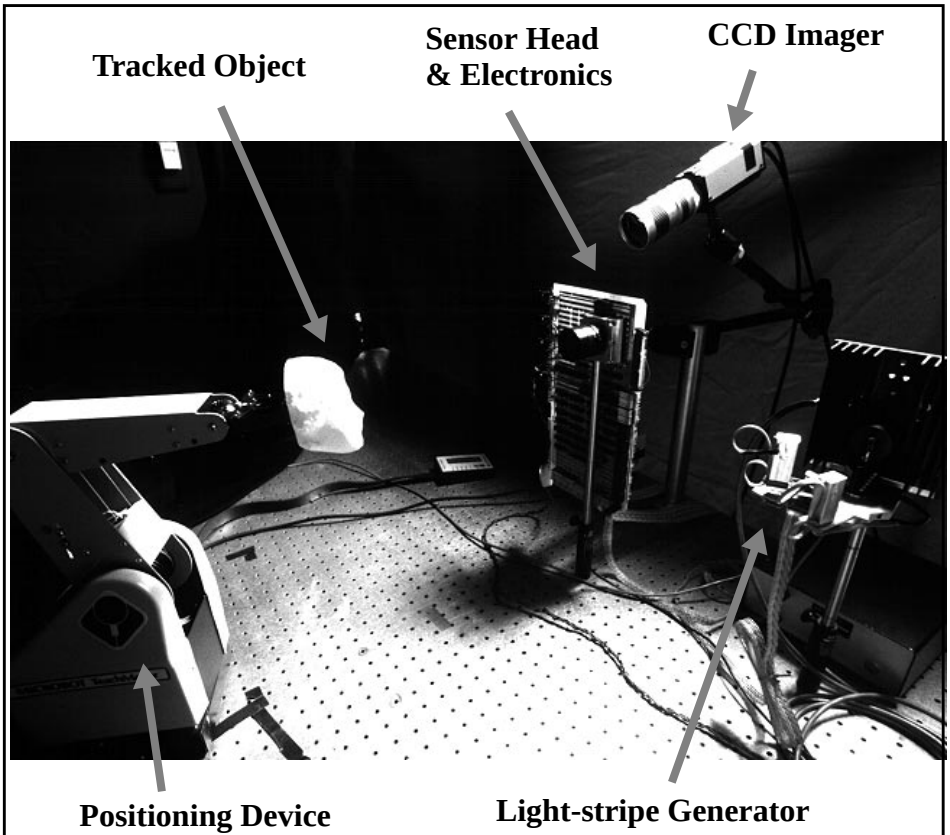

Figure 5: System Components

All of the results presented below were collected using the face object shown in Figure 6. This object was manufactured directly from a triangular mesh CAD model using a stereolithographic process [9]. The advantage of this approach is that the physical object is very accurately represented by the corresponding CAD model. Thus, for purposes of characterizing system accuracy, errors caused by differences between the physical object and the CAD model are minimized.

All pose estimates presented below are specified in an object centered coordinate system as shown in Figure 6. The object itself is roughly 8cm x 10cm x 6cm in the X, Y, and Z directions respectively.

5.0 Pose Estimation Results

There are two results presented in this section. The first demonstrates the ability of our system to *accurately* estimate the pose of stationary, or slowly moving objects. The second demonstrates the ability to track complex motions in a highly *repeatable* manner. Currently, we do not have the ability to generate complex and accurately calibrated dynamic trajectories which are precisely known at each point along the trajectory. Therefore, we can not currently demonstrate that our system can *accurately* track *high-speed* motions.

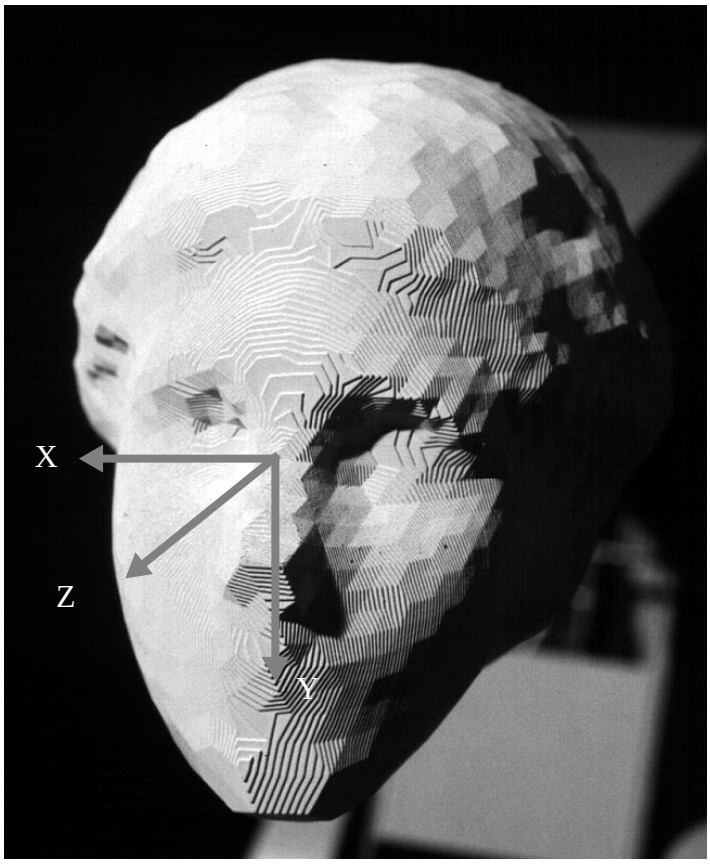

Figure 6: Face Object

5.1 Static Accuracy Results

The graphs in Figure 7 demonstrate the absolute accuracy of the system when the object is assumed to be stationary. To collect this data, the object was manually positioned to selected points along a trajectory using a high precision positioning device. At each point, 100 pose estimates were computed, and corresponding mean and standard deviation values were calculated. Each data point in the graphs compares the object's ground truth position to the mean of the corresponding estimated position. The solid line represents the zero error case, and vertical deviations from this line can be interpreted as error.

The object trajectory for these experiments consisted of coupled translations along each axis, and rotations about the Y axis. We were unable to generate rotations about the X and Z axes due to limitations in our apparatus. The average error between ground truth and estimated positions is 0.93mm in the translation components and 1.4 degrees in the rotation components. The standard deviation of each position estimate is less than 0.06mm in translation and 0.1 degree in rotation.

The results of Figure 7 demonstrate that the system can generate accurate pose estimates for stationary or slowly moving objects. In these experiments, the full resolution of the sensor was used, and the ICP termination threshold, ϵ, was small. In the current implementation, the system is only capable of tracking very slowly moving objects using these parameter settings. When tracking faster motions, such as those described in Section 5.2, the sensor resolution is

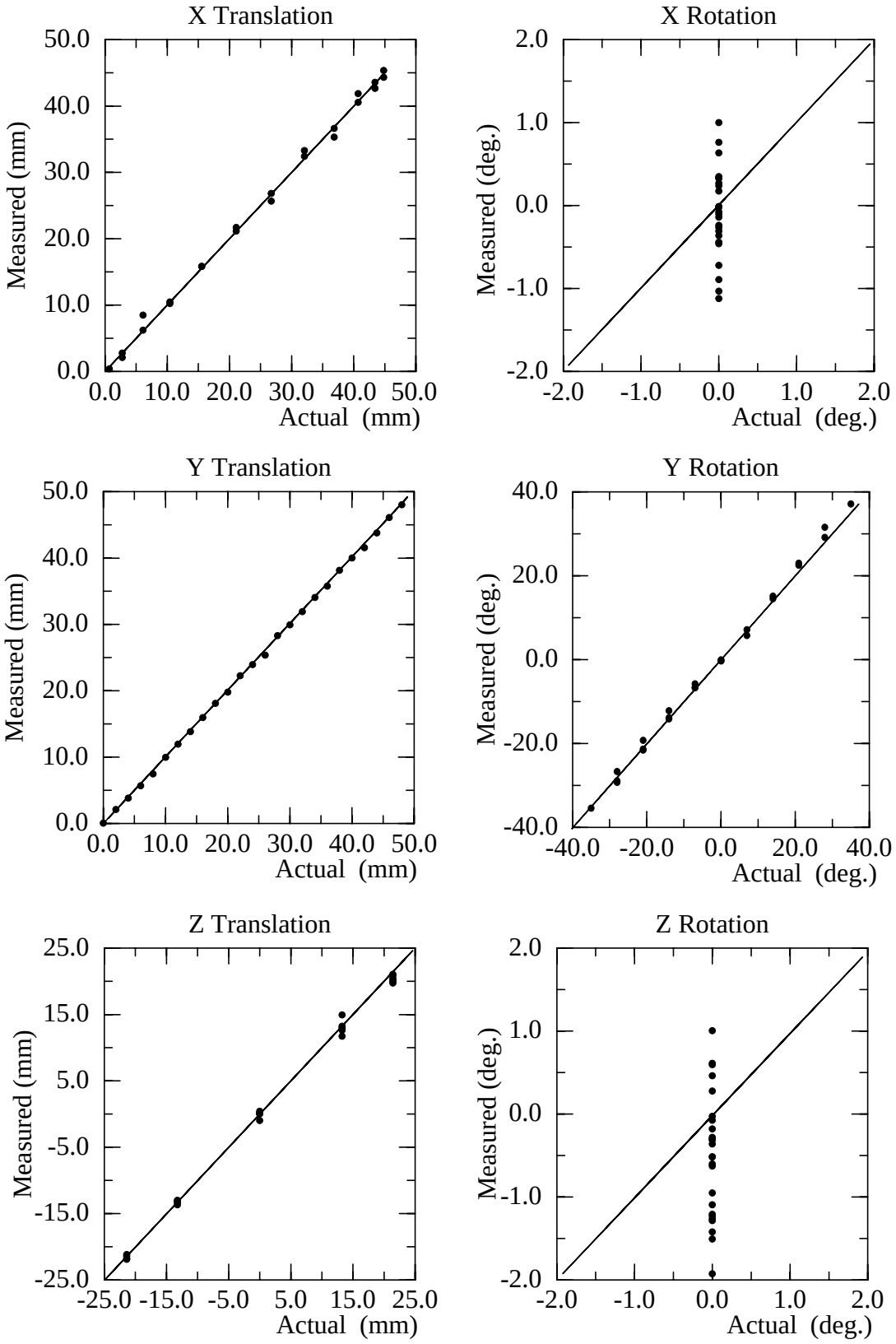

Figure 7: Static Accuracy Measurements

5.2 Dynamic Tracking Results

Figure 8 contains plots of estimated pose as the object is moved through a complex trajectory by the Microbot. Pose estimates are specified with respect to the object's initial pose at time 0. Maximum object velocities are roughly 100 mm/sec in translation and 22 degrees/sec in rotation.

Each graph in these figures actually contains 2 overlaid data sets corresponding to 2 different executions of the trajectory. Furthermore, each single execution of the trajectory is periodic with a period of 2. It is evident from these graphs that the *repeatability* of the pose estimation system is quite good. These results also demonstrate that the system can perform pose estimation fast enough to track object motion at the velocities specified above. The average cycle time in these experiments was about 0.3 seconds (3.3Hz), with variation between about 0.1 seconds (10Hz) and 0.5 seconds (2Hz). This variation in cycle time reflects the variation in the initial pose estimate ${}^{M}\boldsymbol{T}_{D}\,[k-1]$ relative to the actual pose. Large transformations between initial and actual pose result in an increased number of cycles required by the ICP algorithm, and thus a longer overall cycle time. Thus, faster object velocities typically lead to longer cycle times, while slower velocities lead to shorter cycle times.

6.0 Conclusions

We have described and demonstrated an approach for performing full 3-D pose estimation of arbitrarily shaped rigid objects at speeds up to 10Hz. The approach utilizes a high speed VLSI range sensor capable of acquiring 32x32 cell range images in 1 millisecond or less.

Three fundamental difficulties in real-time pose estimation have been addressed by the current work. First, the direct use of 3-D range data circumvents the need to infer depth information from 2-D data. Second, direct matching of object surface data avoids the need to solve the feature extraction and correspondence problems. Third, computationally efficient algorithms allow fast computation of the 3-D pose.

Real-time 3-D pose estimation would be useful in a variety of situations. In manufacturing environments, it could be used in feedback control loops to allow a mechanism (i.e.: a robot) to perform an operation (i.e.: grasping) on a moving part. In the area of Human Computer Interaction (HCI), real-time pose estimation could be useful for tracking movements of a body part for subsequent interpretation as input to a computer. In medicine, a variety of problems involve the need to register pre-operative, volumetric data with the corresponding anatomy of the actual patient. The approach described in this paper may be useful in these cases.

Acknowledgments

The authors would like to thank Kazunori Higuchi for supplying the triangular mesh CAD models, Mark Wheeler for providing the code to implement the kd-tree search, Andy Gruss and Shige Tada for providing assistance with the high speed range sensor, and Lee Weiss and Kevin Hartmann for producing the stereolithographed object.

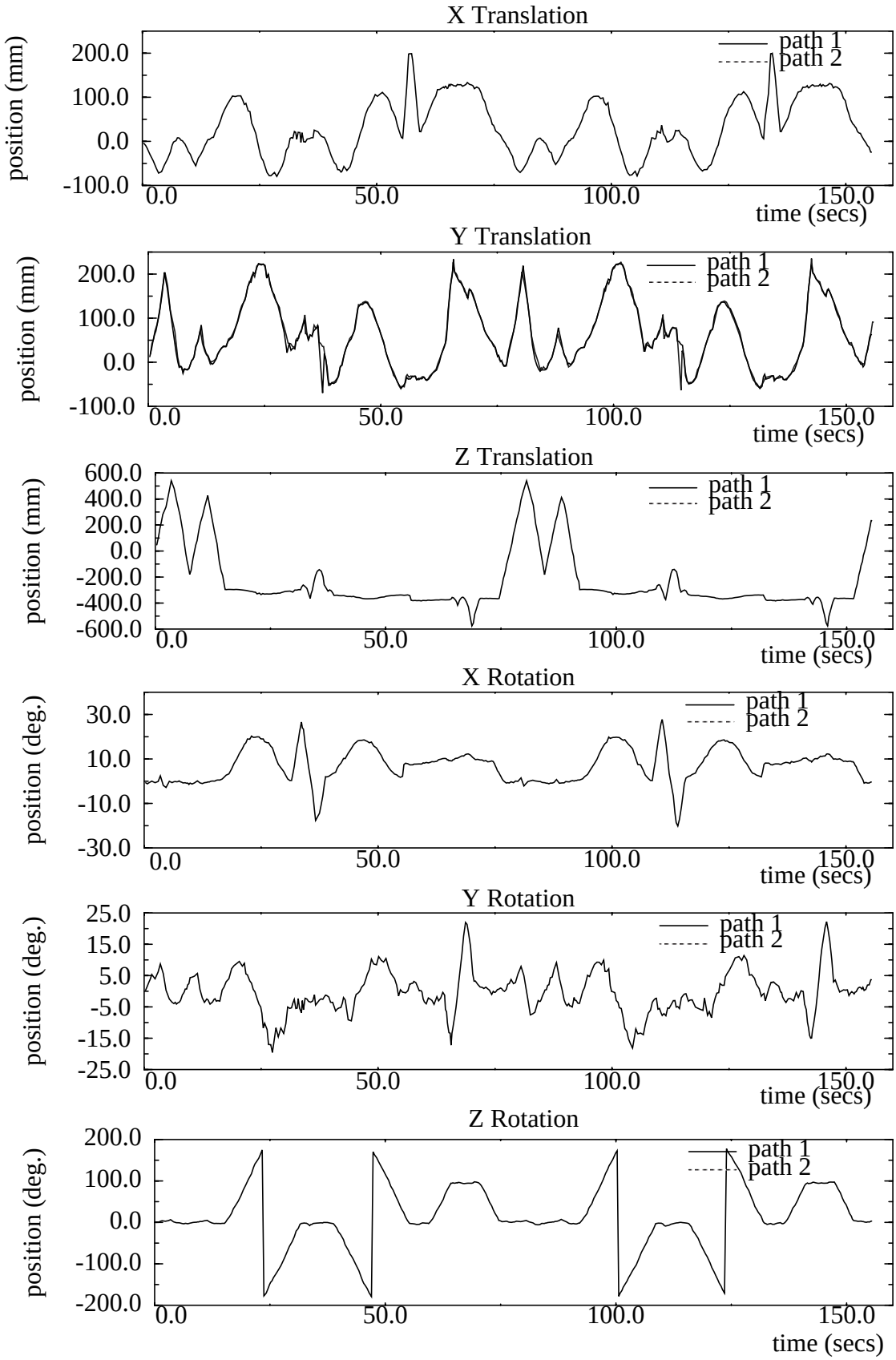

Figure 8: Dynamic Repeatability Measurements

References

[1] Bentley, J.L. Multidimensional binary search trees used for associative searching. *Communications of the ACM.* 18(9):509-517, September, 1975.

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

[3] Delingette, H., Hebert, M. and Ikeuchi, K. Shape representation and image segmentation using deformable surfaces. *Image and Vision Computing.* 10(3):132-144, April, 1992.

[4] Faugeras, O.D. and Hebert, M. The representation, recognition, and locating of 3-D objects. *The International Journal of Robotics Research.* 5(3):27-52, Fall, 1986.

[5] Friedman, J.H., Bentley, J.L. and Finkel, R.A. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software.* 3(3):209-226, 1977.

[6] Gruss, A., Tada, S. and Kanade, T. A VLSI smart sensor for fast range imaging. *International Conference on Intelligent Robots and Systems (IROS '92)*, pages 349-58. IEEE, Raleigh, NC, July, 1992.

[7] Horn, B.K.P. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A.* 4(4):629-642, April, 1987.

[8] Kehtarnavaz, N. and Mohan, S. A framework for estimation of motion parameters from range images. *Computer Vision, Graphics, and Image Processing.* 45(1):88-105, January, 1989.

[9] Marcus, H. and Bourell, D. Solid free form fabrication. *Advanced Materials and Processes.* 144(3):28-35, September, 1993.

[10] Papanikolopoulos, N.P., Nelson, B and Khosla, P.K. Full 3-D tracking using the controlled active vision paradigm. *Proceedings of the International Symposium on Intelligent Control*. IEEE, Glasgow, Scotland, U.K., August, 1992.

[11] Sato, K. and Inokuchi, S. Range-imaging system utilizing nematic liquid crystal mask. *Proc. ICCV,* pages 657-661. London, UK, 1987.

[12] Wang, J. and Wilson, W.J. 3D relative position and orientation estimation using Kalman filter for robot control. *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2638-2645. IEEE, Nice, France, May, 1992.

[13] Yamamoto, M. Direct estimation of range flow on deformable shape from a video rate range camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 15(1):82-89, January, 1993.

[14] Zhang, Z. Iterative point matching for registration of free-form curves and surfaces. *The International Journal of Computer Vision.* To Appear.