Rigid Registration: The Iterative Closest Point Algorithm

Ziv Yaniv

School of Engineering and Computer Science
The Hebrew University, Jerusalem, Israel.

This lecture continues the subject of point based rigid registration. An important aspect of the algorithms previously presented was the assumption that the pairing between point coordinates in the two coordinate systems is known. In this lecture we present an algorithm which will not require this assumption and furthermore does not require a full pairing between data sets.

Problem Definition
Given the coordinates of points measured in two cartesian coordinate systems, without a known pairing, find the rigid transformation between the two systems.

The summary is divided into four sections: (1) Iterative closest point algorithm (ICP). (2) k-D Trees (3) Accelerated ICP (4) Algorithm initialization and local minima.

1 Iterative Closest Point Algorithm

In our previous lecture we described algorithms which performed rigid registration given two point sets $\{R_i\}$ and $\{L_i\}$ with a known pairing between them. A similar problem arises when we want to perform rigid registration using two point sets without specifying the pairing. An iterative approach to this problem was presented in [3] and [17]. Essentially both articles describe the same algorithm [1].

Given two point sets $\{R_i\}$ and $\{L_i\}$ *without* a known pairing between them we want to compute the rigid transformation between them. If we somehow obtain the missing pairing then we already have a closed form solution to the problem by minimizing the following objective function (details in previous lecture):

$$f(R,t) = \frac{1}{N}\sum_{i=1}^{N} \|p_{r,i} - R(p_{l,i}) - t\| \ , \text{ where } N \text{ is the number of paired points.}$$

The question is how do we find the pairing?

[1] This summary is based on [3]

Iterative Closest Point Algorithm

1. Create a pairing between point sets, closest points are matched.
2. Compute the rigid registration given the pairing.
3. Apply the transformation to the data and compute the mean distance between point sets.
4. If change in mean distance has not decreased below a given threshold $d\mu$ or number of iterations has reached the threshold MAX_ITERATIONS terminate.

Table 1: Iterative Closest Point algorithm.

Let us assume that the transformation is small, nearly identity. This means that the distance between the point in the coordinate system R and its transformed location in L is small. Given this assumption a likely pairing for the point p_r is the closest point in $\{L_i\}$:

$$d(p_r, \{L_i\}) = \min_{p_l \in \{L_i\}} \|p_r - p_l\|$$

Unfortunately there is no guarantee that the transformation is nearly the identity transform, so taking the closest point in $\{L_i\}$ will usually yield a wrong match. A standard approach in such situations is to use iterations in the hope of converging to the correct solution, hence the Iterative Closest Point algorithm, Table 1.

Now that we have defined the general framework of the ICP algorithm we can mention the differences between the two works [3] and [17]:

1. In [3] all point pairs are used when computing the transformation, while in [17] only pairs whose distance is below a certain threshold are retained. This yields greater robustness to the presence of outliers, but makes it impossible to analytically prove convergence to local minima.
2. In [3] the paired point registration is computed using the method due to Horn [8], while [17] uses the dual number quaternion method due to Walker [15].

Looking at the description of the algorithm (Table 1) we can clearly see that the pairing step is the most computationally expensive step with a worst case cost of $O(MN)$ for two point sets of size M and N respectively. Given a naive search for the closest point we incur this worst case cost which makes this algorithm impractical for large data sets. Two immediate improvements are:(a) Not to take the square root when working with the L_2 norm. (b) Partial distance/sum computation, continue computing the distance only if the partial sum is still smaller than the current minimal distance. Unfortunately even with these improvements the algorithm is impractical with large data sets, as its complexity is still $O(MN)$. In the next section we introduce

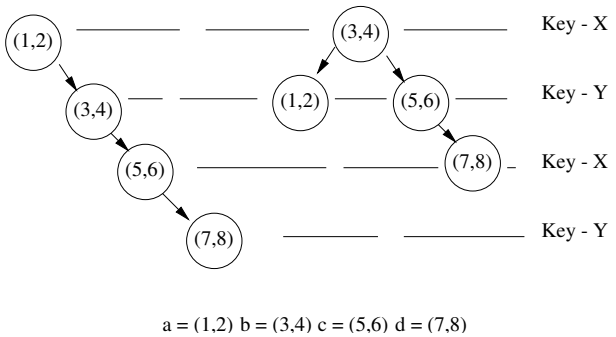

Figure 1: Two examples of inserting four points into the naive k-D tree: (1) Left tree is the result of insertion order 'abcd'. (2)Right tree is the result of insertion order 'bcad'.

a spatial data structure which allows us to perform nearest neighbor searches in less than linear time per search.

2 k-D Trees

A k-D tree, k-Dimensional binary tree, is a spatial data structure originally proposed by [2]. Given d-dimensional points this data structure imposes a spatial decomposition which allows for efficient search on orthogonal range queries and nearest neighbor queries. This efficiency is due to the spatial decomposition which prunes much of the the search space. Most of the following is found both in the cited articles and in two books [12] and [5]. In the following subsections we give a brief description of several types of k-D trees.

2.1 Naive k-D Tree

This data structure is the original k-D tree proposed by [2]. Unlike the other variants it does not require the whole set of points to be available prior to its construction. On the other hand its structure is affected by the order in which the points are inserted.

Points are inserted just as if we were inserting data into a regular binary search tree with the exception that the key we use changes between levels of the tree. At each level i the key is the (i *mod* d) coordinate where d is the dimensionality of the points.

The point data is held in all nodes of the tree. This type of tree can result in different trees for the same data, depending on the insertion order (Figure 1).

2.2 Median k-D Tree

When referring to k-D trees this is probably what most people are referring to. This data structure is due to [6].

Given a set $\{P_i\}$ of d-dimensional points create the k-D tree.

Median k-D Tree Creation

1. If cardinality of $\{P_i\}$ is less than bucket size create leaf node containing point data.
2. Else
   (a) Choose key coordinate (two options):
      i. If at level i key is the (i *mod* d) coordinate.
      ii. Find axis with maximal spread and choose this as the key (requires additional information held in internal tree nodes).
   (b) Split data according to median of the 'key' coordinate and apply recursion with two point sets $\{P_i\}_{=<median}$, left subtree, and $\{P_i\}_{>median}$, right subtree.

The cost of construction is $O(nlogn)$.
The most computationally expensive step is finding the median according to the given key. Median finding can be done in $O(n)$ complexity. Given this complexity for each tree level and that there are $logn$ levels the total construction time is $O(nlogn)$. Linear time median selection algorithms are a bit tedious from an implementation standpoint. We can solve this by preprocessing the point data. Sort the data according to all d-dimensions, a complexity of $O(nlogn)$. At each recursive call pass left and right sorted sub lists to the calls, median is given in $O(1)$ for each partition and sub list construction is an $O(n)$ operation.

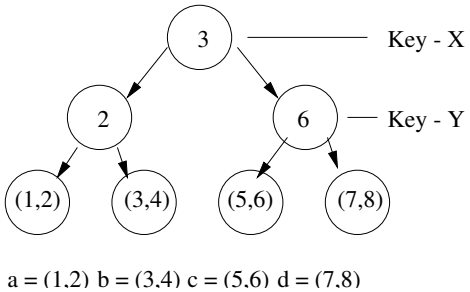

Figure 2: A tree created with the points a,b,c,d with bucket size set to one, key chosen according to (*level mod* 2) and partition plane is the median.

After constructing the tree we would like to perform nearest neighbor queries using it: Given a d-dimensional point and the root of the k-D tree as input find the nearest neighbor.

Median k-D Tree Nearest Neighbor

1. If k-D tree node is a leaf perform an exhaustive search on points contained in the node.
2. Else
   (a) Key is coordinate ($i \ mod \ d$), where i is current level in the tree.
   (b) If $point[key] > partition\_value$
      i. Recurse on right sub tree.
      ii. If $currentMinimalDistance > distance(point, partitionPlane)$ recurse on left sub tree.
   (c) Else
      i. Recurse on left sub tree.
      ii. If $currentMinimalDistance > distance(point, partitionPlane)$ recurse on right sub tree.

From the description of the nearest neighbor query algorithm it is evident that the search space is pruned steps b(ii), c(ii) as the search is conducted on the other side of the partition plane only if the points distance from its current nearest neighbor is greater than its distance from the partition plane (Figure 3).

2.3 Sproull/Eigen k-D Tree

This is a refinement on the median k-D tree. Instead of the partition planes being parallel to the coordinate axis they are chosen based on the data [14]. The separation plane is chosen to be perpendicular to the eigenvector corresponding to the largest eigenvalue of the covariance matrix. This is perpendicular to the direction of most variance found in the data. The points are projected onto this direction and the median is chosen as the separating value. This results in more balanced trees, but costs $O(d^3)$ for each eigenvector computation.

This eigenvector based technique was recently rediscovered in [9], constructing a tree structure similar to the k-D tree. Instead of a binary partition at each level the partition has a constant size (when the partition size is two we return to [14]).

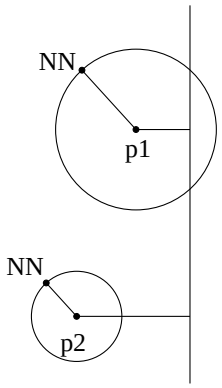

Figure 3: Relationship between query points $p1$,$p2$ an their nearest neighbor. Because the distance between $p1$ and the partition plane is smaller than from its NN a search in the right half plane is required. The point $p2$ does not require search in the right half plane.

3 Accelerated ICP

During the iterations the ICP algorithm produces a set of transformations, 7D points:

$$T_i = [q|t] = [s, x, y, z|dx, dy, dz]$$

We now look at the direction these points define:

$$\Delta T_k = T_k - T_{k-1}$$

Given three points we have two directions as defined above ΔT_k and ΔT_{k-1}. If it turns out that we are heading in approximately the same direction (Figure 4) in the 7D space then we can try to *improve our solution by extrapolation.* The difference in direction is given by:

$$\theta_k = \cos^{-1}\left(\frac{\Delta T_k \Delta T_{k-1}}{\|\Delta T_k\|\|\Delta T_{k-1}\|}\right) \tag{1}$$

Given an angular tolerance of $\delta\theta$ we say that the directions are approximately the same if

$$\theta_k < \delta\theta \qquad and \qquad \theta_{k-1} < \delta\theta$$

Let T_k, T_{k-1} and T_{k-2} be three consecutive transformations and d_k, d_{k-1} and d_{k-2} their respective mean square errors. We define the following approximate arc lengths:

$$v_k = 0, \quad v_{k-1} = -\|\Delta T_k\|, \quad v_{k-2} = -\|\Delta T_{k-1}\| + v_{k-1}$$

Next linear and parabolic interpolants for the last three points are computed:

$$d_1(v) = a_1 v + b_1 \qquad d_2(v) = a_2 v^2 + b_2 v + c_2$$

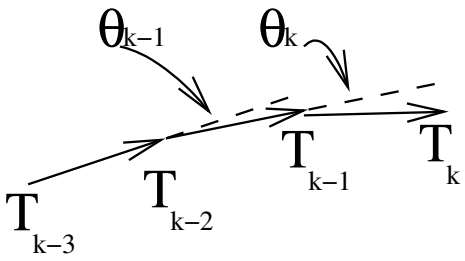

Figure 4: Extrapolation is done when we are heading in approximately the same direction.

This gives us a possible linear update, the zero crossing of the line, and a parabolic update, the extremum point of the parabola:

$$v_1 = -b_1/a_1 \ > 0 \quad v_2 = -b_2/2a_2$$

We define a maximal scale value v_{max} to remove the problem of overshooting. Having computed v_1 and v_2 apply the following extrapolation strategy:

1. If $0 < v_2 < v_1 < v_{max}$ or $0 < v_2 < v_{max} < v_1$

$$T'_k = T_k + v_2 \frac{\Delta T_k}{\|\Delta T_k\|}$$

2. If $0 < v_1 < v_2 < v_{max}$ or $0 < v_1 < v_{max} < v_2$

$$T'_k = T_k + v_1 \frac{\Delta T_k}{\|\Delta T_k\|}$$

3. If $v_1 > v_{max}$ and $v_2 > v_{max}$

$$T'_k = T_k + v_{max} \frac{\Delta T_k}{\|\Delta T_k\|}$$

If applying T'_k results in a mean square error smaller then d_k then we have improved our registration without iterating. In [3] v_{max} was set empirically to $25\|\Delta T_k\|$ (the phrase *magic number* comes to mind).

This extrapolation strategy has a certain deficiency due to coupling of rotation and translation which restrains the extrapolation. For instance if the rotation is well aligned but the translation is not, there will be no extrapolation. Applying the extrapolation scheme both to rotations and translations separately yields better results [13].

At this point in the discussion it should be pointed out that *applying the extrapolation is not always beneficial*. There are cases where the acceleration may cause the

minimization to be trapped in a local minimum which would not have been entered had we not performed the extrapolation.

We conclude this section with an important observation for those interested in implementing the acceleration. When checking if two transformations are approximately in the same direction, Equation 1, we are comparing two vectors whose elements are a quaternion q and a translation t. Remembering that the rotations defined by q and $-q$ are equivalent it is clear that we should not compute the dot product of two consecutive transformations directly. Instead we arbitrarily decide that the first entry of all our quaternions should be positive/negative. We then modify the quaternion yielded by our closed form solution accordingly. This modification does not change the rotation but now we can check if we are proceeding in the same direction in our seven dimensional search space.

4 Algorithm Initialization and Local Minima

The main drawback of the ICP algorithm is that it only guarantees convergence to a *local* minimum, while we want to arrive at the *global* minimum (this is a feature of many iterative algorithms). In our quest for the global minimum we should look into two issues, algorithm initialization and our optimization search strategy.

For the ICP algorithm to converge to the global minimum we have to initialize it near this minimum. The initialization is done by obtaining/computing an estimate of the desired transformation. Obtaining an initial transformation can be done by manipulating a 3D model on the computer and aligning it to the acquired data. The manipulation yields the desired initialization. Another initialization strategy is to acquire a small set of paired points, where the pairing does not have to be accurate. The initial transformation is computed from this set of points using Horn's algorithm.

In [16] and in [4] the second strategy is applied. In [16] an in vitro study is conducted on a phantom. An exact a priori pairing is given, noise is added to the measurements and an initial transformation is computed from this data. Given this initial transformation the phantom surface is sampled and the ICP algorithm is applied. Empirical results show that the combination of the inaccurate pairing combined with the surface data yields better results than just the original pairing. In [4] anatomical data is used, CT scans of the lungs. An initial coarse point pairing is done by searching the two scans we want to align for known anatomical landmarks. This is done using templates of these landmarks and the Normalized Cross Correlation ratio as a distance function. Once the landmarks are located in both scans the transformation is computed using the landmark centroids as the corresponding paired points. Finally the ICP algorithm is applied using the segmented surface of the lungs and the initial transformation.

Unfortunately initialization near the global minimum still does not guarantee convergence to it, if there are local minima near the global one. This is why we have to

consider the search strategy. At each step in our algorithm we would like to produce an update to the current transformation, which will move us towards the global minimum. There exist several popular search heuristics which increase the chances of this happening: simulated annealing, genetic algorithms and tabu search. For a concise introduction to these heuristics see chapter one of [11].

In [7] a simulated annealing approach is used while in a later work by the same group [1] a genetic algorithm is applied (this algorithm or a similar one is used in the commercial system SurgiGATE). Both works initialize the ICP stage using a coarse paired point scheme. The ICP minimization function is also updated to incorporate a penalty term based on the coarse point pairing. This term constrains the transformation so that the distance between the coarse pairing does not get too large. In [10] Gaussian noise is added to the data and the ICP algorithm is performed. As the algorithm progresses the standard deviation of the noise is reduced. This approach is very similar to simulated annealing. According to the experimental results it allowed the algorithm to escape from local minima found around the global one.

A k-D Tree Implementations

The following implementations are available on the web:

1. "ANN: Library for Approximate Nearest Neighbor Searching" from University of Maryland.
   http://www.cs.umd.edu/ mount/ANN/

2. "Ranger - Nearest Neighbor Search in Higher Dimensions" from Stony Brook State University of New York.
   http://www.cs.sunysb.edu/ algorith/implement/ranger/implement.shtml

3. "CGAL: Computational Geometry Algorithms Library" a consortium of academic institutes developing a common library.
   http://www.cgal.org

References

[1] Bächler R., Bunke H., Nolte L.P., "Restricted Surface Matching–Numerical Optimization and Technical Evaluation", Computer Aided Surgery, Vol.6(3), pp. 143–152, 2001.

[2] Bentley J., "Multidimensional binary search trees used for associative searching", Commun. ACM, Vol.18, pp. 509–517, 1975.

[3] Besl P.J., McKay N.D., "A Method for Registration of 3D Shapes", IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI), Vol.14(2), pp. 239–255, 1992.

[4] Betke M., Hong H., Ko J.P., "Automatic 3D Registration of Lung Surfaces in Computed Tomography Scans", Medical Image Computing and Computer Assisted Intervention (MICCAI), 2001.

[5] de Berg M., van Kreveld M., Overmars M., Schwarzkopf O., *Computational Geometry, Algorithms and Applications*, Springer-Verlag, 1997.

[6] Friedman J., Bentley J., Finkel R., "An algorithm for finding best matches in logarithmic expected time", ACM Transactions on Mathematical Software, Vol.3, pp. 209–226, 1977.

[7] Gong J., Bächler R., Sati M., Nolte L.P., "Restricted Surface Matching, A New Approach to Registration in Computer Assisted Surgery", CVRMed-MRCAS, pp. 597–605, 1997.

[8] Horn B.K.P., "Closed-form solution of absolute orientation using unit quaternions", Journal of the Optical Society of America, Vol. 4(4), pp. 629–642, 1987.

[9] McNames J., "A Fast Nearest-Neighbor Algorithm Based on a Principal Axis Search Tree", IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), Vol. 23(9), pp. 964–976, 2001.

[10] Penney G.P., Edwards P.J., King A.P. et al., "A Stochastic Iterative Closest Point Algorithm (stochastICP), Medical Image Computing and Computer Assisted Intervention (MICCAI), 2001.

[11] Rayward-Smith V.J., Osman I.H., Reeves C.R., Smith G.D. Editors, *Modern Heuristic Search Methods*, Wiley, 1996.

[12] Samet H., *The Design and Analysis of Spatial Data Structures*, Addison Wesley, 1990.

[13] Simon D.A., Hebert M., Kanade T., "Techniques for Fast and Accurate Intra-Surgical Registration" Journal of Image Guided Surgery, Vol.1(1), 1995.

[14] Sproull R.L., "Refinements to nearest-neighbor searching", Algorithmica, Vol.6, pp. 579–589, 1991.

[15] Walker M.W., Shao L., Volz R.A., "Estimating 3-D location parameters using dual number quaternions, Computer Vision, Graphics, and Image Processing (CVGIP), Vol.54(3), pp. 358–367, 1991.

[16] Yaniv Z., Sadowsky O., Joskowicz L., "In-vitro accuracy study of contact and image-based registration: materials, methods, and experimental results", Computer-Assisted Radiology and Surgery (CARS), Elsevier pp. 141–146, 2000.

[17] Zhang Z., "Iterative Point Matching for Registration of Free-Form Curves and Surfaces", International Journal of Computer Vision (IJCV), Vol.13(2), pp. 119–148, 1994.