A Nearest Neighbor Method for Efficient ICP

Michael Greenspan Guy Godin

Visual Information Technology Group
Institute for Information Technology, National Research Council Canada
Bldg. M50, 1500 Montreal Rd., Ottawa, Ontario, Canada, K1A 0R6
email: *firstname.lastname*@nrc.ca

Abstract

A novel solution is presented to the Nearest Neighbor Problem that is specifically tailored for determining correspondences within the Iterative Closest Point Algorithm. The reference point set $\mathbf{P}$ *is preprocessed by calculating for each point* $\vec{p}_i \in \mathbf{P}$ *that neighborhood of points which lie within a certain distance* ϵ *of* $\vec{p}_i$*. The points within each* ϵ*-neighborhood are sorted by increasing distance to their respective* $\vec{p}_i$*. At runtime, the correspondences are tracked across iterations, and the previous correspondence is used as an estimate of the current correspondence. If the estimate satisfies a constraint, called the* Spherical Constraint*, then the nearest neighbor falls within the* ϵ*-neighborhood of the estimate. A novel theorem, the* Ordering Theorem*, is presented which allows the Triangle Inequality to efficiently prune points from the sorted* ϵ*-neighborhood from further consideration.*

The method has been implemented and is demonstrated to be more efficient than both the k-d tree and Elias methods. After ~ 40 *iterations, fewer than 2 distance calculations were required on average per correspondence, which is close to the theoretical minimum of 1. Furthermore, after 20 iterations the time expense per iteration was demonstrated to be negligibly more than simply looping through the points.*

Keywords: *nearest neighbor problem, correspondence, iterative closest point, triangle inequality*

1 Introduction

The Iterative Closest Point Algorithm is overwhelmingly the most widely utilized range data processing method. First formulated by Besl and McKay [1] and extended by Chen and Medioni [2], ICP iteratively improves the registration of two overlapping surfaces by calculating the unique transformation that minimizes the mean square distance of the correspondences (the *mse*) between the two surfaces. At each iteration, new correspondences are determined based upon the current relative position of the surfaces.

Calculating the correspondences has been reputed to account for the bulk of the computational expense of ICP. In their implementation, Besl and McKay reported that determining the correspondences consumed a full 95% of the runtime. If the two surfaces are represented as point sets, then establishing the correspondence for each point is known as the *Nearest Neighbor Problem*. At each iteration, the correspondent of each point in the *floating* (i.e. *data*) surface is defined as that point in the *reference* (i.e. *model*) surface which it is nearest to. Correspondence can be defined equivalently when the surfaces are represented otherwise.

A number of classical general solutions exist to the Nearest Neighbor Problem, including binning methods such as k-d tree [3] and Elias [4], and those which make use of the Triangle Inequality [5, 6, 7]. Implementations of the ICP typically employ one of these general methods, with the k-d tree likely being the most widely used.

We present a solution to the Nearest Neighbor Problem which is specifically tailored to ICP. It is motivated by two observations. The first is that the correspondence distances tend to become smaller with each successive ICP iteration. In the aggregate, this is described by the *convergence theorem* [1], which states that the *mse* will reduce monotonically with each iteration. The second observation is that the Triangle Inequality tends to be a more powerful pruning constraint as the correspondence distances reduce.

The method has been formulated here for the case where both surfaces are represented as point sets, and it is believed that there are generalizations to other surface representations. The reference point set $\mathbf{P}$ is preprocessed by calculating for each $\vec{p}_i \in \mathbf{P}$ that neighborhood of points which lie within a certain distance ϵ of $\vec{p}_i$. The points within each ϵ-neighborhood are sorted by increasing distance to their respective $\vec{p}_i$. At runtime, the correspondences are tracked across iterations. If a floating point ever falls closer than $\epsilon/2$ to the reference point to which it corresponded in the previous iteration, then its new correspondence will lie within the ϵ-neighborhood of this pre-

vious correspondence. The Triangle Inequality is applied to the sorted ϵ-neighborhood to efficiently prune out neighborhood points from further consideration [7].

Despite the understanding that the correspondence calculation is the rate limiting step, there has been little previous work reported on techniques for efficient Nearest Neighbor determination in the context of ICP. An exception is Simon et al. [8] who used a caching method which is similar to the Spherical Constraint presented here. For a given iteration, rather than determining the single nearest neighbor for each query point, they identified small sets of ~ 5 nearest neighbors, which they stored in caches associated with each query point. At the next iteration, assuming that the incremental transformation was small, it was necessary only to search the respective cache to determine the correspondence for each transformed query point. They found this technique to improve the runtime performance of the k-d tree method.

Blais and Levine [9] used the inverse calibration parameters of the acquisition sensor to project the points from one image onto the reference frame of the other image. While this method was efficient, it did not identify the true nearest neighbor correspondences between images.

Other context specific solutions to the Nearest Neighbor problem include Vidal Ruiz et al. [6] which was based upon the Triangle Inequality. Their method was effective for moderately small point sets, and was applied to a speech recognition problem [10]. Nene and Nayar [11] presented a method which was effective mainly for large dimensional spaces.

The paper continues as follows: Section 2 covers the mathematical preliminaries, and the method is described in detail in Section 3. Section 4 reports the results of an example designed to illustrate its performance. The paper concludes in Section 5 with a summary and some future research directions.

2 Mathematical Preliminaries

In this section, a variation of the Nearest Neighbor Problem is defined which applies directly to ICP. A condition called the *Spherical Constraint* is presented which limits the neighborhood of points which need be considered in determining the nearest neighbor. The *Triangle Constraint* is also presented which, according to the *Ordering Theorem*, further reduces the number of potential points.

2.1 Nearest Neighbor Problem

Let $\mathbf{P} = \{\vec{p}_i\}_1^n$ be a set of n points in a d-dimensional space, and let $\vec{q}$ be a *query point*. A statement of the *Nearest Neighbor Problem* is:

> *Find the point $\vec{p}_c$ in $\mathbf{P}$ which is the minimum distance from $\vec{q}$, i.e.*
>
> $$\|\vec{q} - \vec{p}_c\| \leq \|\vec{q} - \vec{p}_i\| \ \ \forall \ \ \vec{p}_i \in \mathbf{P} \qquad (1)$$

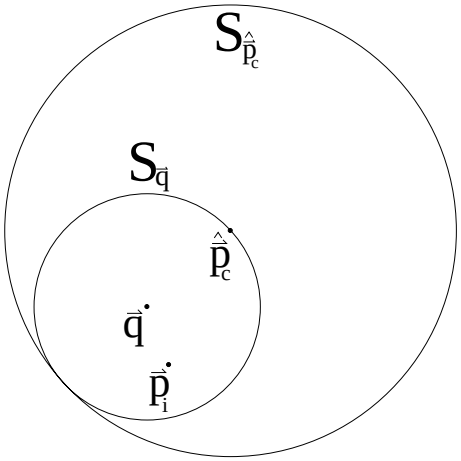


Figure 1: Spherical Constraint

We consider here the following variation of the Nearest Neighbor Problem which, as we shall see, will be useful for improving the time efficiency of ICP:

> *Find the point $\vec{p}_c$ in $\mathbf{P}$ which is the minimum distance from $\vec{q}$, given an initial estimate $\hat{\vec{p}}_c \approx \vec{p}_c$.*

One option is to simply ignore $\hat{\vec{p}}_c$ and solve the conventional Nearest Neighbor Problem using classical methods, such as k-d tree or Elias. To be relevant, any method which makes use of $\hat{\vec{p}}_c$ must improve upon the efficiency of conventional solution methods.

2.2 Spherical Constraint

A property which we shall call the *Spherical Constraint* holds that any point which is closer to $\vec{q}$ than is the current estimate $\hat{\vec{p}}_c$ must lie within the sphere centered on $\hat{\vec{p}}_c$ with radius $2\|\vec{q} - \hat{\vec{p}}_c\|$.

Proposition 1 (Spherical Constraint)

$$\|\vec{q}-\vec{p}_i\| < \|\vec{q}-\hat{\vec{p}}_c\| \Rightarrow \|\hat{\vec{p}}_c-\vec{p}_i\| < 2\|\vec{q}-\hat{\vec{p}}_c\| \qquad (2)$$

Proof*: Let $S_{\vec{q}}$ be the sphere centered at $\vec{q}$ with radius $\|\vec{q}-\hat{\vec{p}}_c\|$. From the LHS of Eq.(2), $\vec{p}_i$ must also lie within $S_{\vec{q}}$;*

$$\|\vec{q}-\vec{p}_i\| < \|\vec{q}-\hat{\vec{p}}_c\| \Rightarrow \vec{p}_i \in S_{\vec{q}} \qquad (3)$$

Now let $S_{\hat{\vec{p}}_c}$ be the sphere centered at $\hat{\vec{p}}_c$ with radius $2\|\vec{q}-\hat{\vec{p}}_c\|$, such that $S_{\hat{\vec{p}}_c}$ completely contains $S_{\vec{q}}$, i.e. $S_{\vec{q}} \subset S_{\hat{\vec{p}}_c}$. Then, $\vec{p}_i \in S_{\vec{q}} \subset S_{\hat{\vec{p}}_c}$, rendering the RHS of Eq.(2).

A 2D diagram of the proof of Proposition 1 is illustrated in Figure 1.

The Spherical Constraint describes a necessary but not sufficient condition for closest point identification: there

may well be points which satisfy only the RHS of Eq.(2). It is nevertheless hypothesized that when the estimate $\hat{\vec{p}}_c$ is sufficiently close to $\vec{p}_c$, the constraint can be used to rule out many contenders.

2.3 Triangle Inequality

Let points $\vec{p}_i$, $\vec{p}_j$, and $\vec{p}_k$ be defined in a metric space, and let $\|\vec{p}_i - \vec{p}_j\|$ denote the Euclidean distance between $\vec{p}_i$ and $\vec{p}_j$. A statement of the Triangle Inequality is:

$$|\|\vec{p}_i - \vec{p}_j\| - \|\vec{p}_j - \vec{p}_k\|| \;\leq\; \|\vec{p}_i - \vec{p}_k\| \;\leq\; \|\vec{p}_i - \vec{p}_j\| + \|\vec{p}_j - \vec{p}_k\| \quad (4)$$

The equality limit on the lower bound holds when the 3 points are collinear, and $\vec{p}_j$ does not lie between $\vec{p}_i$ and $\vec{p}_k$. The equality limit on the upper bound holds when the 3 points are collinear and $\vec{p}_j$ does lie between $\vec{p}_i$ and $\vec{p}_k$.

Proposition 2 (Triangle Constraint)

$$\begin{aligned} &| \, \|\vec{q} - \hat{\vec{p}}_c\| - \|\hat{\vec{p}}_c - \vec{p}_i\| \, | > \|\vec{q} - \vec{p}_j\| \\ &\quad \Longrightarrow \|\vec{q} - \vec{p}_i\| > \|\vec{q} - \vec{p}_j\| \end{aligned} \quad (5)$$

Proof: *Reversing the lower bound of Eq.(4) gives* $\|\vec{q} - \vec{p}_i\| \geq | \, \|\vec{q} - \hat{\vec{p}}_c\| - \|\hat{\vec{p}}_c - \vec{p}_i\| \, |$. *Substituting this into the LHS of Eq.(5) gives:*

$$\|\vec{q} - \vec{p}_i\| \geq | \, \|\vec{q} - \hat{\vec{p}}_c\| - \|\hat{\vec{p}}_c - \vec{p}_i\| \, | > \|\vec{q} - \vec{p}_j\|$$

which yields RHS.

The Triangle Constraint in the form of Eq.(5) has been well exploited in solutions to the Nearest Neighbor Problem [5, 6]. We present here a novel treatment toward this end which is based upon the following Ordering Theorem. The main idea of the Ordering Theorem is to arrange the points by increasing distance to $\hat{\vec{p}}_c$. Any point which violates the Triangle Constraint can be discounted as a possible nearest neighbor. Further, as the points are arranged in increasing order, all subsequent points will necessarily also violate the Triangle Constraint, and therefore need not be considered.

Theorem 1 (Ordering Theorem) *Let* $\mathbf{P} = \{\vec{p}_i\}_1^n$ *be a set of points which are ordered by increasing distance to some point* $\hat{\vec{p}}_c$*:*

$$i < j \iff \|\vec{p}_i - \hat{\vec{p}}_c\| \leq \|\vec{p}_j - \hat{\vec{p}}_c\| \quad (6)$$

Let there be a point $\vec{p}_i$ *which is further to* $\hat{\vec{p}}_c$ *than the query point* $\vec{q}$*, i.e.* $\|\vec{q} - \hat{\vec{p}}_c\| \leq \|\vec{p}_i - \hat{\vec{p}}_c\|$*. Then the following relation holds:*

$$\begin{aligned} &| \, \|\vec{q} - \hat{\vec{p}}_c\| - \|\vec{p}_j - \hat{\vec{p}}_c\| \, | > \|\vec{q} - \vec{p}_i\| \\ &\quad \Longrightarrow \|\vec{q} - \vec{p}_k\| > \|\vec{q} - \vec{p}_i\| \;\; \forall \;\; k \;\geq\; j \end{aligned} \quad (7)$$

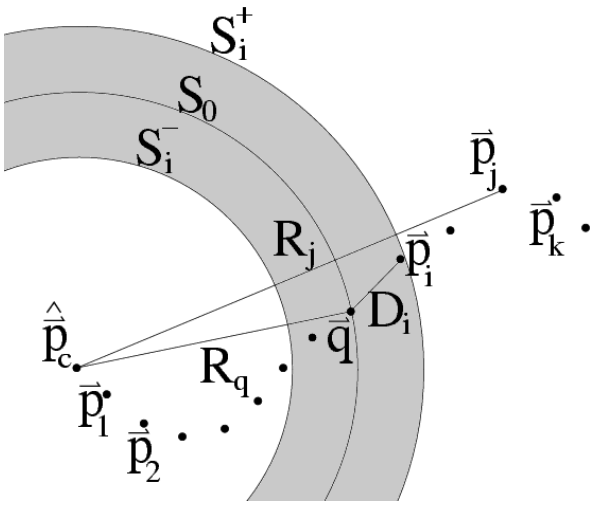


Figure 2: Ordering Theorem

Proof: *From the lower bound of the Triangle Inequality (Eq.(4)):*

$$\|\vec{q} - \vec{p}_k\| \geq | \, \|\vec{p}_k - \hat{\vec{p}}_c\| - \|\vec{q} - \hat{\vec{p}}_c\| \, | \quad (8)$$

The ordering of $\mathbf{P}$ *ensures that* $\vec{p}_k$ *and* $\vec{p}_j$ *are further from* $\hat{\vec{p}}_c$ *than is* $\vec{q}$*, so that:*

$$\begin{aligned} | \, \|\vec{p}_k - \hat{\vec{p}}_c\| - \|\vec{q} - \hat{\vec{p}}_c\| \, | \;&=\; \|\vec{p}_k - \hat{\vec{p}}_c\| - \|\vec{q} - \hat{\vec{p}}_c\| \\ &\geq\; \|\vec{p}_j - \hat{\vec{p}}_c\| - \|\vec{q} - \hat{\vec{p}}_c\| \end{aligned} \quad (9)$$

Substituting Eqs.(9) and (8) into the LHS of Eq.(7) gives:

$$\begin{aligned} \|\vec{q} - \vec{p}_k\| &\geq \|\vec{p}_k - \hat{\vec{p}}_c\| - \|\vec{q} - \hat{\vec{p}}_c\| \\ &\geq \|\vec{p}_j - \hat{\vec{p}}_c\| - \|\vec{q} - \hat{\vec{p}}_c\| > \|\vec{q} - \vec{p}_i\| \\ &\Longrightarrow \textit{RHS} \end{aligned}$$

Corollary 1

$$\begin{aligned} &| \, \|\vec{q} - \hat{\vec{p}}_c\| - \|\vec{p}_j - \hat{\vec{p}}_c\| \, | > \|\vec{q} - \vec{p}_k\| \\ &\quad \Longrightarrow \|\vec{q} - \vec{p}_i\| > \|\vec{q} - \vec{p}_k\| \;\; \forall \;\; i \;\leq\; j \end{aligned}$$

The above corollary was used in [7] to accelerate the performance of binning Nearest Neighbor methods. It is not applied in the method presented here, and is included for completeness.

A geometric interpretation of the Ordering Theorem is illustrated in 2D in Figure 2. The points are shown ordered by increasing distance to $\hat{\vec{p}}_c$, i.e. $\{\vec{p}_1, \vec{p}_2 \ldots \vec{p}_i \ldots \vec{p}_j, \vec{p}_k \ldots\}$. For convenience we denote the distance to $\hat{\vec{p}}_c$ as $R_j = \|\vec{p}_j - \hat{\vec{p}}_c\|$ and $R_q = \|\vec{q} - \hat{\vec{p}}_c\|$, and the distance to $\vec{q}$ as $D_i = \|\vec{p}_i - \vec{q}\|$. Note that $\vec{p}_i$ and $\vec{p}_j$ are not necessarily consecutive, as in Eq.(6). Also, for the sake of illustration the points are shown emanating from $\hat{\vec{p}}_c$ fairly linearly. It is,

however, only their distance from $\hat{\vec{p}}_c$ that is important: they may equivalently be distributed otherwise, as long as their distance from $\hat{\vec{p}}_c$ remains the same.

The three spheres illustrated in Figure 2 are all centered at $\hat{\vec{p}}_c$. Sphere S_0 has radius R_q. Sphere ${S_i}^-$ has radius $R_q - D_i$, and sphere ${S_i}^+$ has radius $R_q + D_i$. By the Ordering Theorem and its Corollary, all points which lie outside of the shaded annulus defined by ${S_i}^+ - {S_i}^-$ will be further from $\vec{q}$ than is $\vec{p}_i$. That is, all points which lie either outside of ${S_i}^+$ or inside of ${S_i}^-$ need not be considered further.

3 Application to ICP

The above results can be used to improve the efficiency of ICP. The main idea is to store the value of the correspondences within an iteration so that they are available at the next iteration. In preprocessing, a neighborhood is calculated for each point $\vec{p}_j$ in the reference set $\mathbf{P}$. At runtime, for each point $\vec{q}_i$ in the floating set $\mathbf{Q}$, the correspondence which was determined at the previous iteration is used as an initial estimate of its current nearest neighbor. The Spherical Constraint is applied to determine whether or not the nearest neighbor falls within the neighborhood of this estimate, and if so, the Triangle Constraint and the Ordering Theorem are applied to the neighborhood to quickly identify the correspondence. A pseudocode representation of the method, which we call *Spherical Triangle Constraint Nearest Neighbor* (*STCNN*), is listed in Figure 3.

3.1 Preprocessing

For each $\vec{p}_j \in \mathbf{P}$ a neighborhood is calculated comprising those points within $\mathbf{P}$ that fall within a distance ϵ of $\vec{p}_j$. This can be accomplished using a k-d tree, Elias, or other conventional Nearest Neighbor method. We call this neighborhood the ϵ-neighborhood of $\vec{p}_j$, and store it in a list which is sorted by increasing distance to $\vec{p}_j$, denoted by the array $\vec{p}_j[\cdot]$. Each element k stores both the point identity $\vec{p}_j[k]$ and its distance $|\vec{p}_j[k]|$ to $\vec{p}_j$. For example, if the 5^{th} element of $\vec{p}_j[\cdot]$ is $\vec{p}_i$, then $\vec{p}_j[5] = \vec{p}_i$ and $|\vec{p}_j[5]| = \|\vec{p}_j - \vec{p}_i\|$.

In summary, the properties of an ϵ-neighborhood are:

$$\text{(membership)} \quad \vec{p}_i \in \vec{p}_j[\cdot] \iff \|\vec{p}_j - \vec{p}_i\| \le \epsilon$$
$$\text{(order)} \quad |\vec{p}_j[a]| < |\vec{p}_j[b]| \iff a < b$$

3.2 Runtime

At runtime, for each $\vec{q}_i \in \mathbf{Q}$, the nearest neighbor which was found for $\vec{q}_i$ in the previous iteration is used as the initial estimate $\hat{\vec{p}}_c$. As the location of $\vec{q}_i$ has changed since the last iteration, the value $\|\vec{q}_i - \hat{\vec{p}}_c\|$ must be recalculated.

The first step is to determine whether or not the nearest neighbor of $\vec{q}_i$ resides within the ϵ-neighborhood $\hat{\vec{p}}_c[\cdot]$, which is accomplished by applying the Spherical Constraint. If $\epsilon > 2\|\vec{q}_i - \hat{\vec{p}}_c\|$ then the nearest neighbor is guaranteed to be a member of $\hat{\vec{p}}_c[\cdot]$. Alternately, if $\epsilon \le 2\|\vec{q}_i - \hat{\vec{p}}_c\|$ then the initial estimate was not tight enough to restrict the search only to $\hat{\vec{p}}_c[\cdot]$, and it is necessary to apply a conventional method to find the nearest neighbor of $\vec{q}_i$. It is also necessary to apply a conventional method in the first iteration where no estimates from previous iterations exist. We call this conventional method the *companion* method.

```
// initialization
for all p_j ∈ P
{
    construct p_j[·]
}

// runtime for iterations > 1
for all q_i ∈ Q
{
    p̂_c = p_c for q_i from last iteration
    minD = ‖q_i − p̂_c‖

    if ( 2 minD < ε )        // Spherical Constraint satisfied
    {
        k = 1, p_d = p̂_c, Done = FALSE

        while ( Done == FALSE and k ≤ size of p̂_c[·] )
        {
            if ( |‖q_i − p̂_c‖ − |p̂_c[k]|| > minD )
            {                // Triangle Constraint not satisfied
                p_c = p_d
                Done = TRUE
            }
            else             // Triangle Constraint satisfied
            {
                if ( ‖q_i − p̂_c[k]‖ < minD )
                {   minD = ‖q_i − p̂_c[k]‖
                    p_d = p̂_c[k]
                }
            }
            k++
        }
        p_c = p_d
    }
    else                     // Spherical Constraint not satisfied
    {
        find p_c with conventional nearest neighbor method
    }
}
```

Figure 3: *STCNN* Pseudocode

Once the Spherical Constraint has been satisfied, one option is to simply calculate the distance to each point in $\hat{\vec{p}}_c[\cdot]$. As ϵ is relatively small, the cardinality of the ϵ-neighborhood will be small compared to that of $\mathbf{P}$, and so this will be quite efficient.

The efficiency can be further improved upon by applying the Triangle Constraint to the ordered points of $\hat{\vec{p}}_c[\cdot]$. The nearest neighbor of $\vec{q}_i$ is initially set to be $\hat{\vec{p}}_c$, and we store its identity in the temporary value $\vec{p}_d \leftarrow \hat{\vec{p}}_c$. Using the information precomputed and stored in $\hat{\vec{p}}_c[\cdot]$ we then test the Triangle Constraint for successive elements, starting at

the first element. If for any k

$$| \|\vec{q}_i - \hat{\vec{p}}_c\| - |\hat{\vec{p}}_c[k]| | > \|\vec{q}_i - \vec{p}_d\|$$

then by Eq.(5)

$$\|\vec{q}_i - \hat{\vec{p}}_c[k]\| > \|\vec{q}_i - \vec{p}_d\|$$

and so the nearest neighbor cannot be $\hat{\vec{p}}_c[k]$. Furthermore, as the elements of $\hat{\vec{p}}_c[\cdot]$ are sorted by increasing distance to $\hat{\vec{p}}_c$, from the Ordering Theorem the Triangle Constraint will also be violated for all subsequent elements ($> k$) of $\hat{\vec{p}}_c[\cdot]$. The nearest neighbor of $\vec{q}_i$ has therefore been identified as $\vec{p}_d$.

Alternately, if for any element the Triangle Constraint is satisfied, i.e.

$$| \|\vec{q}_i - \hat{\vec{p}}_c\| - |\hat{\vec{p}}_c[k]| | \leq \|\vec{q}_i - \vec{p}_d\|$$

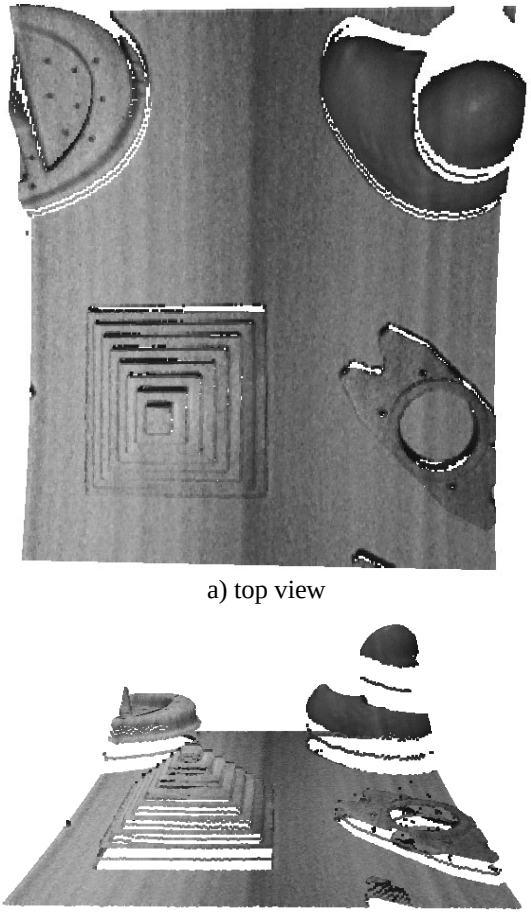
then $\hat{\vec{p}}_c[k]$ may indeed be closer to $\vec{q}_i$ than is $\vec{p}_d$. Its distance is calculated, and if it is less than the current value of $\|\vec{q}_i - \vec{p}_d\|$, then the value of $\vec{p}_d$ is updated to $\vec{p}_d \leftarrow \hat{\vec{p}}_c[k]$. The test then continues with the next list element.

The search terminates for this $\vec{q}_i$ when either the Triangle Constraint is violated, or all elements of $\hat{\vec{p}}_c[\cdot]$ have been visited.

There is a property of ICP which works in favour of the above conditions being satisfied. The *Convergence Theorem* [1] states that the mean square distance of all correspondences (*mse*) will monotonically reduce with successive iterations. On average, the values of $\|\vec{q}_i - \vec{p}_c\|$ and $\|\vec{q}_i - \hat{\vec{p}}_c\|$ will therefore tend to reduce with each iteration. Furthermore, it is the nature of ICP that the first few iterations converge very quickly, resulting in large transformations of $\mathbf{Q}$, whereas later iterations converge more gradually, resulting in smaller transformations. When a highly accurate result is desired (as often is the case) the majority of the time is spent on iterations where the transformations are small, which is exactly the situation where *STCNN* becomes most efficient.

4 Example

We have implemented the *STCNN* method and tested it on a number of data sets. We present here details of the results on one representative example, which we believe typified the performance. The example reference point set $\mathbf{P}$ was a range image containing 94884 points that was acquired with a Biris range sensor. This image is illustrated in Figure 4. The floating point set $\mathbf{Q}$ was an exact copy of $\mathbf{P}$ which had been perturbed to a slightly different initial position. The rotational offset was 5 degrees about all 3 axes, and the translational offset in all 3 directions was 10 mm, which was approximately 1%, 7.5%, and 3% of the extent of the image along the x-, y-, and z-axes respectively.



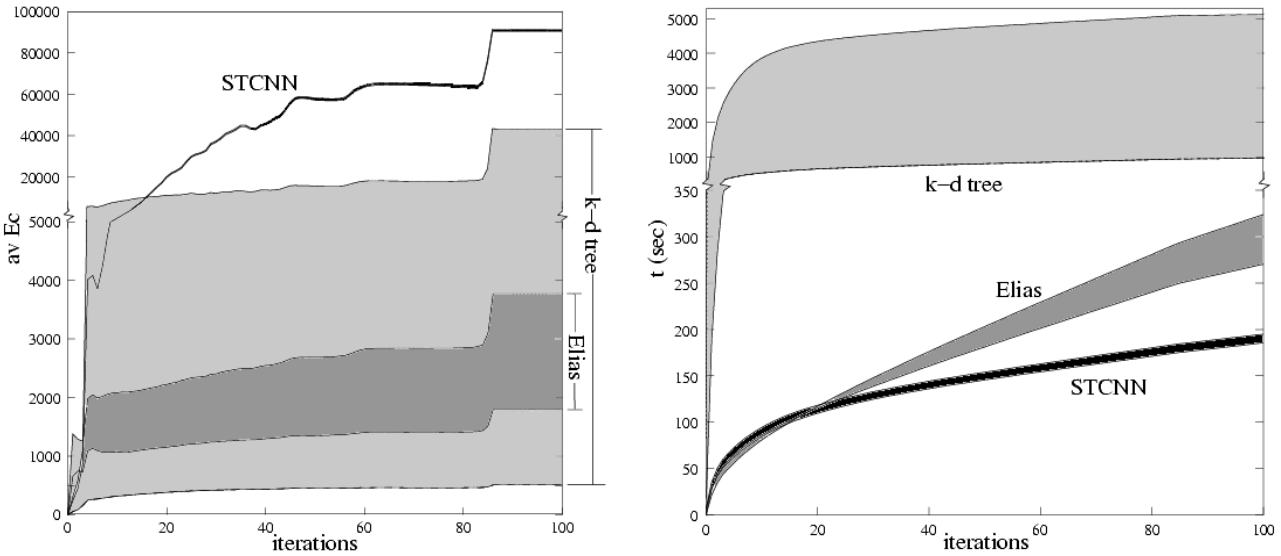
a) top view

b) rotated view

Figure 4: Test Image (~ 95k points)

The performance of *STCNN* was compared with implementations of both the k-d tree and Elias methods. The k-d tree is a strictly binary tree in which each node represents a partition of the k-dimensional space. The root of the tree represents the entire space, and the leaf nodes represent subspaces containing mutually exclusive small subsets of $\mathbf{P}$. At any node, only one of the k dimensions is used as a discriminator, or *key*, to partition the space. When the tree is traversed, the single scalar value of the key of $\vec{q}$ is compared with the node key value, and the appropriate branch is followed. When a leaf node is encountered, all of the B points resident in the leaf's bin are tested against $\vec{q}$. Depending on the closeness of the match at a leaf node, and the boundaries of the space partition, the traversal may backtrack to explore alternate branches.

Whereas the k-d tree method partitions space hierarchically and irregularly, the Elias method groups $\mathbf{P}$ into subsets that form regular (congruent and non-overlapping) subregions. The nearest point to the query point $\vec{q}$ is found by searching the subregions in order of their proximity to $\vec{q}$, until the distance to any remaining region is greater than



a) average E_c vs. iterations

b) time vs. iterations

Figure 5: Comparison of *STCNN*, k-d tree, and Elias: $B \in [1, 300]$, $V \in [40, 100]$, $\epsilon \in [2, 6]$

the distance to the nearest point found, or until all subregions have been processed. Points in each searched subregion are exhaustively examined. Simple modifications to the method allow the search for the K nearest neighbors, or for all points closer than a threshold distance. The partitions are axis-aligned, forming squares in 2-D, cubes in 3-D, or hypercubes in d-D space for $d \geq 4$. The only parameter of the Elias method is the number of bins V_t along each axis t. For our implementation we assume an equal number of bins per axis, (i.e. $V_x = V_y = V_z = V$) so that the total number of bins is equal to V^3.

The *STCNN* uses Elias as its *companion* method, which is required both for initialization of *STCNN* and whenever the Spherical Constraint is not satisfied.

The comparison of the implemented methods was based upon two metrics. The first metric is denoted E_c, and is a measure of the number of distance computations which are explicitly executed. For $\mathbf{P}$ of cardinality n and query point $\vec{q}$, $E_c = \frac{n}{m}$, where m is the number of points for which $\|\vec{q} - \vec{p}_i\|$ was actually computed.

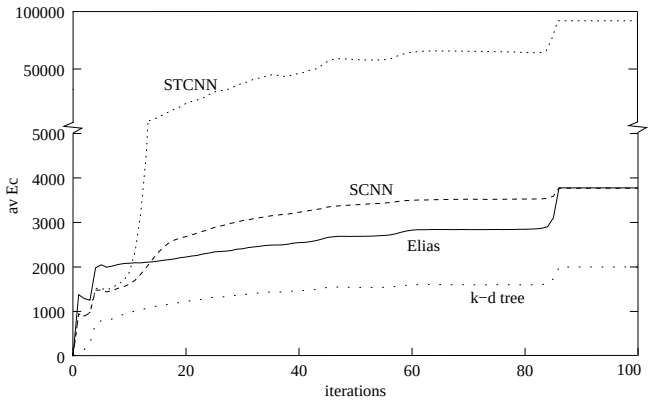
The second metric is the measured runtime to complete a number of iterations. While E_c is a direct measure of computational complexity, the runtime is dependent upon implementation specifics. Each implementation was executed on the same platform (a 400 MHz Pentium II) and compiled with the same compiler options, and a moderate and equivalent amount of attention was afforded to the optimization of each. For example, square root calculations were avoided wherever possible, but no assembly level coding or other machine specific techniques were used. It is believed that these implementations (and therefore the measured time values) are representative of what a practitioner in the field would typically reproduce. It is also likely that these results would scale equivalently with any other implementation optimizations, so long as they were applied equally across all methods.

That the same data set was used for both $\mathbf{P}$ and $\mathbf{Q}$ did not bias the performance of the ICP either for or against any of the methods. It did produce the effect that exact correspondences existed for each point, so that it was possible to identify exactly when the ICP had converged. This occurred at the iteration where each correspondence remained unchanged and the *mse* vanished. In the test case, complete convergence occurred at iteration 86.

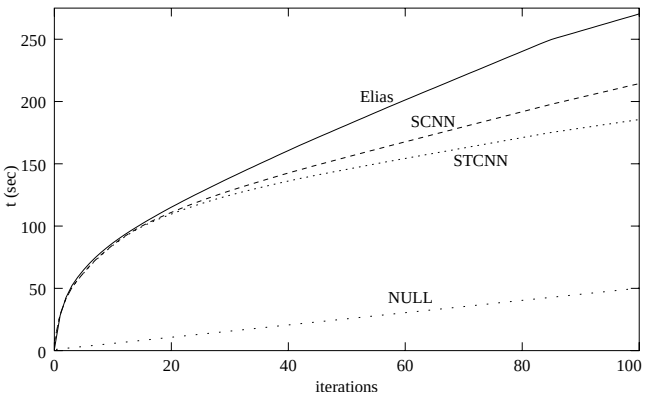
Each method was tested over a range of parameter values. For the k-d tree, the number of points per leaf node was varied over $B \in [1, 300]$. The Elias V parameter was varied over a range of $V \in [40, 100]$. The value of the *STCNN* neighborhood epsilon was varied over a range of $\epsilon \in [2, 6]$ mm. For these values, the average cardinalities of the ϵ-neighborhoods varied respectively between 13 and 117 points.

The range of results of these tests were plotted in Figure 5. The time to initialize each method is indicated on iteration 0 of the time plot in part b). To facilitate visualization over large ranges of values, each plot comprises two distinct linear regions on their y-axes; $[0, 5000]$ and $[5000, 100000]$ for E_c in part a), and $[0, 350]$ and $[350, 5000]$ for runtime in part b).

Two benefits of *STCNN* can be observed from this



a) average E_c vs. iterations



b) time vs. iterations

Figure 6: Comparison of Best Results

graph. The first is that, after ~ 20 iterations, *STCNN* is more efficient than either k-d tree or Elias in both E_c and time. The second benefit is that *STCNN* is less sensitive to the selection of the parameter values V and ϵ. Indeed, most of the variation between the *STCNN* trials occurred in the earlier iterations, which were more dependent upon the Elias companion method. In later iterations, when most or all points fell within the ϵ-neighborhoods, all of the *STCNN* trials had similar performance.

The best performing trials from Figure 5, were replotted in Figure 6. This occurs at $B = 80$ for k-d tree, $V = 80$ for Elias, and $V = 60$ and $\epsilon = 3$ for *STCNN*. Note that the y-axis in part a) comprises two regions; $[0, 5000]$ and $[5000, 100000]$. That the optimal value of the number of bins is less for *STCNN* than for Elias is attributed to the role of the companion method. In *STCNN* initialization, Elias is used to generate the ϵ-neighborhoods, an operation which is slightly more efficient for larger bins. The average cardinality of the ϵ-neighborhood for the value $\epsilon = 3$mm was 28 points.

After about 40 iterations, $E_c > 47442$. Given that the image contained 94884 points, this meant that on average the distance was explicitly calculated for fewer than 2 points per $\vec{q}$. This is significantly close to the theoretical minimum of 1.

Two other trials were also included in this plot, *SCNN* and *NULL*. The *SCNN* method was the same as *STCNN*, with the exception that it applied only the Spherical Constraint, and not the Triangle Constraint. When the Spherical Constraint was satisfied, the distance was therefore tested for every point in the ϵ-neighborhood. In part a), this variation is observed to be only slightly more efficient than Elias by the E_c metric. In part b), it can be seen that the time efficiency of *SCNN* is between that of *STCNN* and Elias. This confirms that the Triangle Constraint serves to practically improve the time performance of the method.

The final curve on the plot of Figure 6b), the *NULL* method, provides an explanation as to why the difference in the time performance between *STCNN* and Elias is not as great as the orders of magnitude difference observed in E_c. For each iteration, the *NULL* method simply loops through all points $\vec{q}_i \in \mathbf{Q}$ and assigns the correspondence of $\vec{q}_i$ as $\vec{p}_i$. As $\mathbf{P} = \mathbf{Q}$, these are in fact the true correspondences, so that the correct transformation is calculated on the first iteration, and the identity transformation thereafter. The significance of the *NULL* method is that there is no search for the correspondences, so that the computational expense is only the overhead of simply looping through the points in $\mathbf{Q}$. After about 20 iterations, it can be seen that the slope of *STCNN* is roughly the same as the slope of *NULL*. This means that, within that range, the correspondence computation of *STCNN* is contributing a negligible expense over that of simply looping through the points.

5 Conclusion

We have presented a novel solution to the Nearest Neighbor Problem which is specifically tailored for use with ICP. The method is based upon the application of the Spherical and Triangle Constraints and uses the correspondences determined in the previous iteration as initial estimates. While the Triangle Inequality has been used in many other Nearest Neighbor methods, to the authors' knowledge the Ordering Theorem and its application are a novel contribution.

The *STCNN* method has been implemented and demonstrated to be faster than fairly lean versions of the k-d tree and Elias methods. At full convergence, which occurred in the presented example at iteration 86, the time to complete the k-d tree, Elias and *STCNN* were 943, 251 and 176 seconds respectively. This represented a time savings of $\sim 30\%$ over Elias, and greater than 500% over k-d tree. A

comparison of the E_c metric, which is a measure of the average number of times the distance was actually calculated, weighed even more in favour of *STCNN*. After ~ 40 iterations, fewer than 2 distance calculations where required per point, which is close to the theoretical minimum of 1. Further, after 20 iterations, the time expense per iteration was demonstrated to be only trivially greater than simply looping through the points without executing any correspondence calculations. We conclude that the cost of the correspondence calculation in later iterations has been reduced so as to be negligible.

In future work, we plan to analyze the complexity expression of the method. We would also like to generalize the results to other surface representations, particularly triangles.

References

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

[2] Y. Chen and G.G. Medioni. Object modeling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, 1992.

[3] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[4] Ronald L. Rivest. On the optimality of Elias's algorithm for performing best-match searches. In *Information Processing 74*, pages 678–681, 1974.

[5] Keinosuke Fukunaga and Patrenahalli M. Narendra. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transactions on Computers*, pages 750–753, July 1975.

[6] Enrique Vidal Ruiz. An algorithm for finding nearest neighbors in (approximately) constant time. *Pattern Recognition Letters*, 4:145–157, July 1986.

[7] Michael Greenspan, Guy Godin, and Jimmy Talbot. Acceleration of binning nearest neighbor methods. In *Vision Interface 2000*, pages 337–344, Montreal, Canada, May 14-17 2000.

[8] David A. Simon, Martial Hebert, and Takeo Kanade. Real-time 3-D pose estimation using a high-speed range sensor. In *IEEE Intl. Conf. Robotics and Automation*, pages 2235–2241, San Diego, California, May 8-13 1994.

[9] Gérard Blais and Martin D. Levine. Registering multiview range data to create 3D computer objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):820–824, August 1995.

[10] Enrique Vidal, Hector M. Rulot, Francisco Casacumerta, and Jose-Miguel Benedi. On the use of a metric-space search algorithm (AESA) for fast DTW-based recognition of isolated words. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(3):651–660, May 1988.

[11] Sameer A. Nene and Shree K. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):989–1003, September 1997.