Contents

Abstract

Registering 3D point sets subject to rigid body transformations is a common problem in computer vision. It occurs in areas like building 3D model from multiple range images and estimation of motion. In this report we shall present the problem and the knowledge we obtained in implementing, developing, combining and testing an algorithm for simultaneous registration of multiple partially corresponding surfaces represented as point sets.

1. Introduction

To explain the virtue of the problem, let us imagine that we are to obtain a surface model of some complex, free-form rigid object. We may suppose that we have been able to get a set of 3D images, called *point sets*, by means of some range sensor. However, due to the physical limitations of the sensing method, none of the images contains information about the whole surface. We usually have some information associated with each picture, about the mutual position of the object and the sensor but it is rarely accurate enough. As our task is to combine all the images into one model of the entire object surface, the first step is to calculate the exact relationship between the images. This first step is often called *registration* and it will be the main topic of this paper.

It is surprisingly difficult to define what should be considered a good registration. A human being performing the same task, i.e. assembling a picture from a set of partly overlapped sub-pictures, usually keeps shifting the sub-pictures until the overlapped parts visually match, or at least resemble each other. To perform the visual matching, human perception and brain automatically reduce the complex picture on a relatively few simple patterns that are then easy to match.

Unfortunately, computer vision algorithms have not yet achieved the degree of proficiency in pattern finding, recognition and matching needed to effectively simulate the human approach. Instead, a simplified procedure is used that generally comprises the following two steps:

(1) Identify image points in all the images that correspond to a single model point, i.e. *find the correspondences* between the point sets.

(2) Find a set of *rigid body transformations* (i.e. consisting of rotations and translations) to be applied to each image, leading to the best unification of the common image points.

Point (1) is easier to accomplish if there are some significant points on the model we can identify. However, this is rarely the case, as these *feature points* are usually difficult to find. Many different approaches for finding feature points are currently being developed exploiting different properties of the examined surfaces such as structural patterns or curvature. Alternatively the correspondence can be established on the basis of the Iterative Closest Point (ICP) algorithm ([7]) described bellow.

Point (2) can be solved quickly using closed form formula in the case of two point sets ([6]). For more point sets, no such formula seems to exist, we therefore have to employ an iterative algorithm presented bellow ([2]).

2. Iterative Closest Point Algorithm

ICP ([7]) attempts to find correspondences between two point sets, i.e. for each point in one of the sets (called *data*) find a corresponding point (*partner*) from the second set (called *model*). The ultimate goal would be to find a transformation Q_∞ (the subscript denotes the number of iterations, ∞ represents the limit) mapping each data point to its partner. Let us assume for start that such a transformation exists and that we have an initial estimate Q_0 of this transformation. If the estimate is sufficiently good, the transformed data points will be closer to their partners than to any other point. From this observation directly emerges the basic principle of the ICP algorithm, corresponding to point (1) above:

> For each point x from the data set transformed by Q_0 find the closest point y from the model data set. Then label y as x's partner, symbolically $y = P_0(x)$, $P_0(x) = \arg\min_y \|Q_0(x) - y\|$.

Although our initial estimate Q_0 of the transformation Q_∞ is rarely good enough for the labeling to be fully correct, we may still use it to

> Determine a transformation Q_1 minimizing distances between partner points (point (2) above), i.e. $Q_1 = \arg\min_Q e$, $e = \sum_i \|Q(x_i) - P(x_i)\|^2$

To further improve the transformation (Q_1) we shall apply the process *iteratively*, i.e. use Q_1 to obtain a new labeling P_1, use the new labeling P_1 to get Q_2 etc. The iterative process may be stopped after a predetermined number of iterations or better when the average square error falls bellow a prescribed precision limit.

If Q_∞ exists and Q_0 was sufficiently close, the process converges quickly. As Q_0 moves away from Q_∞ the convergence slows down but is usually acceptable. For data coming from measurement Q_∞ does not exist due to noise and resolution limitation. However, by increasing the resolution we improve the likelihood of finding a good approximation.

3. Iterative Reciprocal Closest Point Algorithm

The classical ICP algorithm is well suited for matching two images resulting from transformation of the same original model point set. Such two sets correspond entirely to each other. However, in our case, we have point sets corresponding only in limited overlapping regions. To obtain the correct matching we should calculate the transformation Q only from points within the overlapping region. The key question is, how to determine whether a given point x from the first point set belongs to the overlapping region, i.e. whether it has a partner y from the model set. Even if some point x_1 has in fact no partner y, the ICP algorithm would still find $y = P(x_1)$ as the closest (although not close) point from the model set. The remedy is to temporarily *revert the roles* of the two sets and find the closest x_2 for a given y from the data set. If x_1 and y were real partners, then $x_2 = x_1$, otherwise $x_2 \neq x_1$. Allowing for the inaccuracy, we permit x and x' to be 'close' (not necessarily identical). Ideally, a geodetic distance between x and x' should be measured. However, for close points, Euclidean distance is a good approximation and much easier to compute, so we shall use it instead. The observation we have just described makes the core of the Iterative Reciprocal Closest Point (IRCP) algorithm ([8]):

> y is labeled x's partner if and only if the data point closest to y when transformed is closer to x than some predetermined threshold ε. Symbolically
>
> $$y = P(x) \iff \left\| x - \arg\min_{x'} \left\| Q(x') - \arg\min_{y} \|Q(x) - y\| \right\| \right\| < \varepsilon$$

To determine the optimal transformation Q in the second step we naturally use only those data points that have partners. In this way we eliminate points not having partners in the other set. This helps us to find the correct matching, i.e. matching with respect to the corresponding (overlapping) parts only.

Although we shall go on calling the point sets *data* respective *model*, please note that in the ICRP algorithm both point sets are equivalent as far as results are concerned. So far, data point set is the set being transformed but this will not be the case later on when *both* sets will be transformed.

The ICRP algorithm is generally much less robust than the original ICP algorithm and much more prone to be mislead to an unwanted local minimum by a wrong estimate Q_0 of the ideal transformation. It is therefore strongly recommended, mainly when no good estimate Q_0 is available, to start the algorithm several times with different initial Q_0. When no estimate is available at all, a set of uniformly laid out rotations can be used.

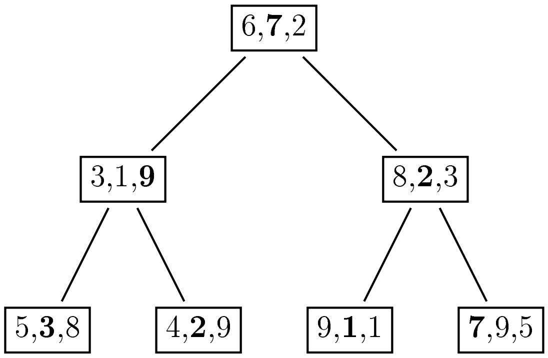
The performance of the ICRP algorithm is also strongly influenced by the choice of the threshold ε. When set too high, too few points will be eliminated and the calculated transformation will be adversely influenced by points not belonging to the common region. When set too low, on the other hand, too many points will be eliminated, resulting in Q being calculated from too few points and therefore suffering from noise and error.

It is generally difficult to find a good criteria as to when to stop the ICRP algorithm as because of the changing number of points considered, the average square error function is not monotonously decreasing with time. A solution might be to compare the average error for an *identical* set of point pairs for successive transformations Q_i and Q_{i+1}. Alternatively the computation can be stopped after the average error has not changed significantly in a certain number of iterations.

The convergence properties of the ICRP algorithm are yet to be studied. The number of iterations depends in a complicated way on the initial transformation and also on the spatial distribution of the two point sets. Therefore, no exact measure of the time consumption of the ICRP algorithm can be given here, however it is evident that the algorithm is inherently slower than the ICP algorithm as the closest point search must be performed twice as many times and also because its convergence is much slower.

4. Closest Point Search

A simplified study on the speed of the ICRP algorithm reveals that an efficient implementation of the closest point search, forming the innermost part of the ICRP algorithm, is of a paramount importance. Let us briefly restate the problem:



Binary search tree, key coordinates set in **bold face**

Given a set of points $Y = \{y_1, y_2, \ldots, y_n\} \subset \mathbf{R}^3$ and a point x.
Find $y = \arg\min_{y \in Y} \|x - y\|$. For one set Y many points x will be queried, so it is feasible to use preprocessing.

A number of approaches is available, ranging from the simplest conceivable method going sequentially through all the points in Y (which takes $O(n)$ time) up to some very clever methods based on Voronoi diagram ([3]) (which take only $O(\log n)$ time per query). However, as the task of effectively constructing and applying Voronoi diagram in 3D is by no means an easy one, we have deferred implementing the probably optimal solution and have used simpler *range search tree* ([4]) based method that has a worst-case performance $O(n)$ but on the average takes also $O(\log n)$ time. According to our present knowledge the exact behavior of this algorithm has yet to be fully analyzed and the average case property stated above is only an estimate based on the algorithm structure. An empirical evidence shows great dependence of the processing time on the spatial distribution of Y and the position of x.

During the preprocessing phase an ideally balanced binary search tree is constructed having in each node one point from Y. The number of left descendants of each node differs at most by one from the number of its right descendants. Furthermore, each node has a *key coordinate* and it holds that each of the left descendants is less or equal in this key coordinate, while all the right descendants are greater or equal in this coordinate. (See picture.) The key coordinate can be altered regularly from level to level, we obtained slightly better results by choosing the coordinate with the greatest difference between the maximum and minimum values among all the descendants to be.

Let us begin with a simple problem first. To find a point that is guaranteed to belong to Y in $O(\log n)$ time we can simply traverse the tree starting from the root, comparing the key coordinates of the query point and of the point in the actual node, proceeding to left or right child accordingly.

A more complex problem is to find all points falling in a given rectangle $\{y = (a, b, c) \in Y \mid a \in \langle a_1; a_2 \rangle, b \in \langle b_1; b_2 \rangle, c \in \langle c_1; c_2 \rangle\}$ — *range search*. For this purpose a breadth-first search beginning from the tree root can be used where we shall descend only the branches that may contain some of the points searched. More precisely if a key coordinate of a node falls within the desired interval, we shall search both sub-trees, if it is less than the lower boundary or greater than the upper boundary we shall search only the right or left sub-tree respectively. Experiments show that this algorithm seem to perform about $O(r+\log n)$ operations, where r is the number of points found. Additional small performance gain can be obtained by storing information about the smallest and largest points in key coordinates within left and right sub-trees respectively in each node.

Finally, let us show how to use the tree to speed up the closest point search. The algorithm used is very similar to the range search described above. At the very beginning we consider the rectangle to be infinitely large in all dimensions. Later on, as we proceed with the search, we limit the rectangle to $\{y = (a, b, c) \in Y \mid a \in \langle x_1 - \delta; x_1 + \delta\rangle, b \in \langle x_2 - \delta; x_2 + \delta\rangle, c \in \langle x_3 - \delta; x_3 + \delta\rangle\}$, where $x = (x_1, x_2, x_3)$ is the query point we search for and $\delta = \min_i |x - y_i|$ is the minimum of distances between x and points from Y encountered so far. We can do so, because no points laying *outside* this rectangle can be closer to x than δ. This way, we gradually make the search rectangle shrink and limit the number of points in question.

Ideally, only a small percentage of the points is searched, in some cases it was less than 60 points out of 5000. However, it can also happen that a node is so far away from x that no points can be eliminated. If this happens for all the n nodes, we need to examine all of them. This accounts for the worst case of $O(n)$ time needed.

5. Registration of Multiple Point Sets

In the preceding sections we have described the apparatus used to register two point sets. The fundamental difference of the multiple set problem is that to solve point (2) — finding the optimal rigid body transformation — no analytic method is known and thus numerical solution must be used. However, the numerical method is still faster than finding the mutually corresponding points (point (1)).

We assume that there are m point set $Y^1, \ldots, Y^m$, each forming (after being properly transformed by a rigid body transformation Q^i_∞) a part of the surface of the same object. Let us denote the points belonging to the set $Y^i = \{y^i_1, y^i_2, \ldots, y^i_{n(i)}\}$. Assume an algorithm such as ICRP has been run for each pair Y^i, Y^j, $i \neq j$ to obtain pairwise correspondences P_0 using the initial estimates of the transformations Q^i_0, Q^j_0. Symbolically if $y^i_k \in Y_i$ is a partner of $y^j_l \in Y_j$, we shall write $y^i_k = P^{(i,j)}(y^j_l)$ and $y^j_l = P^{(j,i)}(y^i_k)$, as the partnership is a symmetrical relation.

Now, the task is to find such a set of transformation $Q^1_1, Q^2_1, \ldots, Q^m_1$ that the total sum of weighted Euclidean distances between partners is minimal, i.e.

$$(Q^1_1, \ldots, Q^m_1) = \arg\min_{(Q^1_1, \ldots, Q^m_1)} e_{tot}\,, \qquad e_{tot} = \sum_{i=1}^{m} \sum_{j=i+1}^{m} e(i,j)$$

$$e(i,j) = \sum_{\substack{k=1 \\ P_0^{(j,i)}(y^i_k) \neq \emptyset}}^{n(i)} \left\| Q^i_0\left(y^i_k\right) - Q^j_0\left(P_0^{(j,i)}(y^i_k)\right)\right\|^2 =$$

$$= \sum_{\substack{l=1 \\ P_0^{(i,j)}(y^j_l) \neq \emptyset}}^{n(j)} \left\| Q^j_0\left(y^j_l\right) - Q^i_0\left(P_0^{(i,j)}(y^j_l)\right)\right\|^2$$

where the notation under the summations mean: "Sum over all points from a given set having partners in the other set."

A useful model of the situation appears when we imagine a small spring between each pair of corresponding points. The total difference e_{tot} plays the role of the total *potential energy* the system has. From this physical analogy we know that the equilibrium exists. In order to find it we let the system evolve exactly as if it were a real physical system.

We shall define an useful notation shortcut to denote an average value: $\langle x\rangle = \sum_{x\in X} x \,/\, N$. ($N$ is the number of members of X.) Similarly, the average of points from Y^i having partners in Y^j will be denoted as $\langle y^i\rangle^{(j,i)}$.

Let us define a *center of mass* for each point set. It will later serve us as a turning point around which the set will rotate. It could be set to an arbitrary point in space but an experience shows that good results are obtained by following the physical analogy and setting $y^i_{CM} = \langle y^i\rangle$.

Finally, let us assume that the transformation Q^i can be written as

$$Q^i(x) = R^i(x) + r^i \tag{$*$}$$

where R^i is a 3×3 rotation matrix and is a r^i vector of translation which permits us to express e_{tot} as:

$$e_{tot} = \sum_{i=1}^{m}\sum_{j=i+1}^{m}\sum_{\substack{k=1\\ P_0^{(j,i)}(y_k^i)\neq\emptyset}}^{n(i)} \left\| R^i\left(y_k^i\right) + r^i - R^j\left(P_0^{(j,i)}(y_k^i)\right) - r^j\right\|^2$$

The force acting on the set Y^i due to the set Y^j may be computed as:

$$F^{(i,j)} = -\sum_{k=1}^{n(i)}\frac{\partial e_{tot}}{\partial y_k^i} = -2\left(R^i\langle y^i\rangle^{(j,i)} + R^i y^i_{CM} + r^i - R^j\langle y^j\rangle^{(i,j)} - R^j y^j_{CM} - r^j\right)$$

The torque acting on set Y^i due to the set Y^j may be computed as:

$$\begin{aligned}\tau^{(i,j)} &= -\sum_{k=1}^{n(i)} y_k^i \times \frac{\partial e_{tot}}{\partial y_k^i} = \\ &= -2\left(R^i\langle y^i\rangle^{(j,i)}\right)\times\left(R^i y^i_{CM} + r^i - R^j y^j_{CM} - r^j\right) + 2\langle R^i y_k^i \times R^j P^{(j,i)}(y_k^i)\rangle^{(j,i)}\end{aligned}$$

Denoting the covariance matrix between sets Y^i and Y^j as

$$\Theta^{(i,j)} = \sum_{\substack{k=1\\ P_0^{(j,i)}(y_k^i)\neq\emptyset}}^{n(i)} y_k^i\left(P^{(j,i)}(y_k^i)\right)^T \quad\text{and}\quad R^i = \begin{pmatrix} R^i_1\\ R^i_2\\ R^i_3\end{pmatrix}$$

an identity holds:

$$\langle R^i y_k^i \times R^j P^{(j,i)}(y_k^i)\rangle^{(j,i)} = \begin{pmatrix} R^i_2\Theta^{(i,j)}R^{j\,T}_3 - R^i_3\Theta^{(i,j)}R^{j\,T}_2\\ R^i_3\Theta^{(i,j)}R^{j\,T}_1 - R^i_1\Theta^{(i,j)}R^{j\,T}_3\\ R^i_1\Theta^{(i,j)}R^{j\,T}_2 - R^i_2\Theta^{(i,j)}R^{j\,T}_1\end{pmatrix}$$

By summing over all the sets Y^j, $j \neq i$ we obtain the total force and torque acting upon set Y^i:

$$F^i_{tot} = \sum_{\substack{j=1 \\ j\neq i}}^{m} F^{(i,j)} \quad \text{and} \quad \tau^i_{tot} = \sum_{\substack{j=1 \\ j\neq i}}^{m} \tau^{(i,j)}$$

It is noteworthy to observe that it suffices to know only the mean values $\langle y^i \rangle^{(j,i)}$, $\langle y^j \rangle^{(i,j)}$ and the covariance matrix $\Theta^{(i,j)}$ to calculate the total force F^i_{tot} and torque τ^i_{tot} for any set of rigid body transformation $Q^1, Q^2, \ldots, Q^m$. Consequently, after the preprocessing has been done (in $O(mn)$ time), we can perform this calculation very quickly, in *constant time*, regardless of the number of points in the sets.

Let us return now to the physical analogy. The physical system (rigid bodies connected with springs) moves according to the laws of motion, depending on the total force and torque acting on it, unless it reaches a (local) potential minimum or it starts to oscillate. To avoid oscillation we add *friction*, so much friction that the equations of motion becomes:

$$\gamma \frac{\mathrm{d}y^i_{CM}}{\mathrm{d}t} = F^i_{tot}\ , \quad \Gamma\omega^i = \tau^i_{tot}$$

where γ and Γ are constant parameters resembling mass and moment of inertia. ω denotes angular velocity, its direction represents the axis of rotation and its magnitude the speed in rad/s.

Given the angular and linear velocities and a time difference $\mathrm{d}t$ we want to update the rotation R^i and translation t^i. First of all we convert $\omega^i = (\omega_1, \omega_2, \omega_3)$ to a quaternion q:

$$q = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \cos(\|\omega\|\mathrm{d}t/2) \\ \sin(\|\omega\|\mathrm{d}t/2)\,\omega_1/\|\omega\| \\ \sin(\|\omega\|\mathrm{d}t/2)\,\omega_2/\|\omega\| \\ \sin(\|\omega\|\mathrm{d}t/2)\,\omega_3/\|\omega\| \end{pmatrix}, \quad q = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \frac{q'}{\|q'\|}$$

From the quaternion we proceed to a rotation matrix:

$$R_{\omega\mathrm{d}t} = \begin{pmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & q_0^2+q_2^2-q_1^2-q_3^2 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & q_0^2+q_3^2-q_1^2-q_2^2 \end{pmatrix}$$

Taking into account that the rotation is around the center of mass, the update equations are straightforward:

$$\begin{aligned} R^i(t+\mathrm{d}t) &= R^i_{\omega\mathrm{d}t} R^i(t) \\ r^i(t+\mathrm{d}t) &= R^i_{\omega\mathrm{d}t}\left(r^i(t) - y^i_{CM}\right) + y^i_{CM} + F^i_{tot}\mathrm{d}t \end{aligned}$$

The above mentioned equations can be in fact combined into one:

$$\left(R^1, \ldots, R^m, r^1, \ldots, r^m\right)(t+\mathrm{d}t) = \mathcal{F}\Big(\left(R^1, \ldots, R^m, r^1, \ldots, r^m\right)(t), \mathcal{P}\Big)$$

where $\mathcal{P}$ is a vector of time-independent parameters

$$\mathcal{P} = \left(\gamma, \Gamma, y^1_{CM}, \ldots, y^m_{CM}, \langle y^1\rangle^{(2,1)}, \langle y^1\rangle^{(3,1)}, \ldots, \langle y^{m-1}\rangle^{(m,m-1)}, \Theta^{(2,1)}, \Theta^{(3,1)}, \ldots, \Theta^{(m,m-1)}\right)$$

The equation can be readily transformed into a set of first order ordinary differential equations and the form above represents directly the Euler method to solve it. The equation is guaranteed to have a *steady point* that is a local minimum of the cost function e_{tot} we were to minimize. An adaptive step-size modification of the Euler method is recommended ([5]), however a constant step-size variety will do for most cases when small steps are used, at the expense of some speed. We experimentally used a simple feedback step-size control — when $\mathcal{E}_1 = \sum_i \|F^i_{tot}\|^2 + \|\tau^i_{tot}\|^2$ increased, step-size was reduced ten times and the last step was redone, otherwise the step-size was increased by 20%, unless the step-size left the bounds $10^{-10} < \mathrm{d}t < 10^2$ (to prevent pathological cases). This method speeded up the process but led to some undesirable glitches in the time behaviour of the estimated parameters. Apparently more elaborate method should be used.

All that rests is to give some guidance on the choice of parameters. The following values are known to give good results but experimenting is encouraged:

$$\gamma = 1\,, \quad \Gamma = \frac{\gamma}{2} \max_i \left(\sum_{k=1}^{n(i)} \left\| y^i_k - y^i_{CM} \right\|^2 \right)\,, \quad \mathrm{d}t = 0.1$$

6. Complete Multiple Point Sets Registration Algorithm

In the preceding sections we have described all the important parts of the algorithm. Here we have it in full:

begin
Compute $y^1_{CM}, \dots, y^m_{CM}$, set γ and Γ.
Set $R^1, \dots, R^m, r^1, \dots, r^m$ to initial values supplied by user.
repeat
    Transform all points from all sets using the appropriate transformations see $(*)$.
    for each pair of point sets (i, j), $i < j$ **do**
        begin
        Calculate the correspondences $P^{(j,i)}$ using the *Closest Point Search* algorithm
        Calculate the mean values $\langle y^i \rangle^{(j,i)}$,$\langle y^j \rangle^{(i,j)}$ and the covariances $\Theta^{(i,j)}$, $\Theta^{(j,i)}$.
        end
    repeat
        for each point set i **do**
            begin
            Calculate F^i_{tot},$\tau^i_t ot$.
            Update R^i, r^i.
            end
    until $|\mathcal{E}_1(k-1) - \mathcal{E}_1(k)| < \varepsilon_1$
until $|\mathcal{E}_2(k-1) - \mathcal{E}_2(k)| < \varepsilon_2$

The symbols $\mathcal{E}$ represent some suitable quantities representing the quality of the temporary solution. Ideally we would use e_{tot} but it is slow to compute. Instead, we have used

$$\mathcal{E}_1 = \mathcal{E}_2 = \sum_i \left\| F^i_{tot} \right\|^2 + \left\| \tau^i_{tot} \right\|^2$$

that is easily computed as a by-product of the inner loop calculations. $\varepsilon_1 = 10^{-2}$, $\varepsilon_2 = 10^{-4}$ is fine for the first experiments and can be decreased to get more accurate results.

The most time consuming part of the algorithm is the closest point search. Assuming one query can be resolved in $O(\log n)$ time with $O(n \log n)$ time for preprocessing, the algorithm as a whole takes $O(m^2 n \log n)$ time.

7. Results

We have conducted several groups of tests. In the first group of test we tried to register very small point sets consisting of just two or three points to test the basic functionality of the algorithm. For these sets the algorithm always converged to a suitable, albeit sometimes sub-optimal, solution. The number of outer loop iterations, i.e. the number of registrations needed was usually less than 5. The number of inner loop iterations varied depending on the exact configuration usually between 10-100 but could be certainly reduced, if need be, by using a better adaptive step-size integration method.

The input for the second group of tests was based on real scanned data of an object, containing about 5000 points. This data was mathematicaly transformed using a predetermined rigid-body transformation to obtain the point sets we later used for registration. For these sets the algorithm rarely converged if the initial estimate of the transformation yeilded points further apart from their true partners than the resultion of the image. We assume that when the initial closest points search gives an incorrect correspondence and many points are eliminated for not fulfilling the reciprocity condition, the remaining points do not provide enough information for improvement and the algorithm almost randomly changes the transformations *ad infimum*.

The third group of test was based on real data obtained by scanning an object from multiple directions. As we could guess from the results from the second group of test, no succesful registration was found.

8. Conclusion

We have implemented an algorithm for registering multiple point sets by combining the ICRP algorithm ([1]) and the Multiple Point Sets registration algorithm ([2]). Although both algorithms separately perform satisfactorily, the combined algorithm suffered from convergence problems that are yet to be resolved.

Other areas of improvement could be using 3D Voronoi diagrams for closest point search and faster (higher order) integration method of solving the differential equations from the algorithm ([2]).

9. References

[1] Pavel Kučera. Registrace prostorových ploch, Diplomová práce, FEL ČVUT, Praha 1996

[2] A. Stoddart, A. Hilton. Registration of Multiple Point Sets, Proceedings of ICPR'96, p. 40-44, IEEE 1996

[3] ???. Geometric Algorithms, ???

[4] Robert Sedgewick. Algorithms, Addison-Wesley 1988

[5] ???. Numerical Recepies in C, ???

[6] B. Horn, Closed-form solution of absolute orientation using unit quaternions. J. Opt. Soc. Amer. A,4(4): p. 629-642, April 1987

[7] P. Besl, N. McKay. A method for registration of 3D shapes. IEEE Trans. Pattern Analysis and Machine Intell., 14(2), p. 239-256, 1992

[8] Tomáš Pajdla, Luc Van Gool. Matching of 3D curves using semidifferential invariants, 5th International Conference on Computer Vision, Cambridge, MA, July 1995

10. Appendix — Description of the program MREG

To evaluate the algorithms presented, an experimental program called *mreg* has been written in C. The program expects at its standard input the information about what to register formatted as follows:

On the first line there should be the number of views to register. Then, for each view there should be a block of two lines — one line containing the name of the file to read and number of points in this file. The second line contains seven real numbers representing the initial transformation. The first four represent the quaternion of rotation, the remaining three the translation. There can be blank lines between blocks. An example of the input data follows:

```
2
cuthar1 5182
1.0 0 0 0 0 0 0
cuthar2 5372
0.99 0.01 0 0 1 0 0
```

The files to read for each view are standard text files with one line per point, each containing three real numbers — the coordinates of the points.

Besides the parameters expected at the standard input, there are also parameters to be changed in the source code, namely `MATLABOUT`, `REGISTER`, `DEBUG`, `MAXNVIEWS`, `DTINIT`, `eliminate`, tt elimdist, `INNERLIM`, `OUTERLIM`, `NITIN`, `NITOUT`. Their description can be found in the source code.