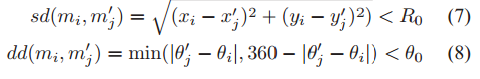
**Minutiae points match**

**A. Problem Formulation**

Let T and I be the template and input fingerprint we are trying to match. I and T may not have the same number of feature points: let *m* and *n* be the number of point for T and I.

Two minutiae points mi and m’j are considered to match if their position and orientation are close. This can be written according to (7) and (8), using the spatial distance *sd*, direction difference *dd.* [6]

R0 and are two tolerance parameters e can adjust to improve matching accuracy.

Let *md* be the function that says if two minutiae points are close or not i.e. md(mi,m’j)=1 if sd(mi,m’j)<R0 and dd(mi,m’j)<, 0 otherwse.

**B. Finding correspondences**

As a preprocessing step of Ransac, we should find for each point in the input image which points in the template are matching point possibly. We don’t use conventional solution SIFT because a minutiae point looks like another point; then distance between them would be not accurate. We present a method called geometric hashing [7] which compute for each minutiae point a feature vector. And the smaller the feature distance, the more likely will be the points to match.

First, we need to consider a minutiae point mi in the template set and three closest points from template set: we number them mi,1, mi,2, mi,3. T make feature vector independent of the rotation in the image, we rotate those points of an angle equal to the olocal orientation of mi.

After we have a rotation invariant feature vector in the template and input set, we can compute Euclidean distance between every permutation of the points. Then we combine those numbers. To make it more robust, we allow one point to be wrong and require two points to match as in (12)

Finally, for each mi in the template, we choose the 5 points m’j in the input set having smallest d2 distance and assign them as being the correspondences.

**C. Ransac**

It’s imposible to try every affine tansformation due to combinatorial complexity. Thus, we use Ransac to try at random transformations and expect to find a good one.

Let *k* be number of iterations required, *d* be number of maximum matching points and *Score* be threshold of matching degree. As shown before, R0 and are thresholds used to identify that a point fits well.

The algorithm is based on three key processes. (1) Take three random points both n template and input set. (2) Find the affine transformation mapping the template random points on the input points. (3) Apply the transformation to all template points. Count the number of points which match (using *md* function) and give a score of match. If score is over *Score*, then we decide these two images are from same fingerprint

**D. Experimental Results**

In the test, we use *R*0 = 20, = 10, *k* = 2000, *Score* = 0.30. We choose FVC2002\_DB1 as database which give us 16 prints and eight times in different conditions for each print. We take first image as input dataset and 80 images as template dataset; therefore. Among 80 comparisons, 60 of them are matched correctly and 20 of them are decided wrongly; 75% accuracy is obtained.

Due to limited time for project, accuracy and running time still need to be improved. For further research, we hope we can improve our approach in following ways: use fingerprint classification to speed up the algorithm; allow no-linear transformation in matching process; compute the local ridge frequency in Gabor filters.

**Reference**

[6] D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar. *Handbook of Fingerprint Recognition*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003. 2, 3

[7] R. S. Germain, A. Califano, and S. Colville. Fingerprint matching using transformation parameter clustering. IEEE COMPUTATIONAL SCIENCE and ENGINEERING. 5