Nearest First Traversing Graph for Simultaneous Object Tracking and Recognition

Junya Sakagaito and Toshikazu Wada
Department of Computer and Communication Sciences
Faculty of Systems Engineering, Wakayama University
sakajun@vrl.sys.wakayama-u.ac.jp, twada@ieee.org

Abstract

This paper presents a new method for simultaneous object tracking and recognition using object image database. This application requires two searches: search for object appearance stored in the database and that for pose parameters (position, scale, orientation, and so on) of the tracking object in each image frame. For simplifying this problem, we propose a new method, pose parameter embedding (PPE) that transforms the original problem to an appearance search problem. The nearest neighbor (NN) appearance search in this problem has a special property that gradually changing queries are given. For this problem, graph based NN search is suitable, because the preceding search result can be used as the starting point of the next search. Delaunay graph can be used for this search, however, both the graph construction cost and the degree (number of mean edges connected to a vertex) drastically increase in high-dimensional space. Instead, we propose nearest first traversing graph (NFTG) for avoiding these problems. Based on these two techniques, we successfully realized video-rate tracking and recognition.

1. Introduction

In this paper, we address a problem of simultaneous object tracking and recognition, which estimates both classes and pose parameters of objects in each video frame.

Most methods proposed so far use connotational (parametric) models that represent multiple appearances of an object by changing model parameters. This approach has the following limitations: 1) Designing an object model requires human support, i.e., a model totally depends on the designers thought. 2) Modifying the model using additional samples enforces re-training of whole model using all training samples. 3) A single model cannot represent appearances of multiple objects.

If we use a denotational model of an object, i.e., a set of object images, above limitations can be solved: 1-2) Appearances of an object can be automatically modeled and modified only by storing object images. 3) Appearances of multiple objects can be included in a single model. Because of these advantages, we formalize the problem as an image retrieval problem using a denotational model.

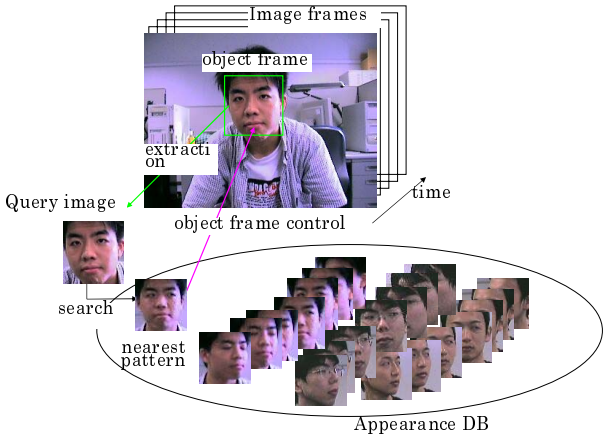


Figure 1. Simultaneous object tracking and recognition

In this formalization, query image is extracted from an image using object frame (See Figure 1), and the nearest neighbor (NN) search is performed for finding the image nearest to the query. The distinguishing properties of this problem from others are 1) the object frame have to follow the moving object, and 2) the NN search query image will not change drastically.

The first property implies that the searches for pose and appearances have to be done simultaneously in the product space of pose parameters and appearances. This requires complicated hybrid search, because the search algorithms in pose parameters and appearances are quite different. For simplifying this problem, we propose pose parameter embedding (PPE) that transforms the search space to the appearance space.

1-4244-1180-7/07/$25.00 ©2007 IEEE

The latter property implies that the NN search result of an image frame can be utilized for next search. This is because the object appearance changes gradually, and hence, the NN search results for consecutive image frames will be close in most cases. For utilizing the previous search result, nearest-first traversing (NstFT) on a graph is suitable. This NN search is a special case of the best-first traversing on a graph, where the best vertex is the nearest vertex to the query. NstFT requires Delaunay graph for accurate NN search, however, the Delaunay graph construction in high dimensional space is impractical, because the computational complexity for the graph construction exponentially increases with the dimensionality of the space. Furthermore, the degree (number of edges connected to a vertex) of the Delaunay graph becomes considerably big. Instead, we propose a new graph, nearest first traversing graph (NFTG). NFTG can be constructed efficiently even in high dimensional space and has low degree. NstFT using NFTG can find NN vertex when the query is close to one of the vertices. As well, complete NN vertex can be found using NFTG, by performing an additional search followed by NstFT.

Based on the proposed methods, PPE and NFTG, we implemented an object tracking and recognition system and confirmed that it successfully tracks and recognizes objects in image sequences and the processing speed for each image frame is less than 10[ms] using a PC with Pentium4, 3GHz. This processing speed enables on-the-fly processing of NTSC and PAL video streams.

2. Related Works

2.1. Object Tracking

Tracking an object in an image sequence is a basic and important technology for Computer Vision. Recently, promising methods have been proposed, e.g., Markov chain Monte Carlo (MCMC) based object tracking[10], mean-shift based object tracking[7], and so on. Object tracking continuously searches a given object in each image frame and determines its position and pose by corresponding the object model with the local image. Tracking methods can be classified into two types: feature based and appearance based approaches. Feature based approach uses local features as tracking cue, such as lines, points, or regions. Such methods are insensitive to local occlusions. On the other hand, appearance-based approach uses an object image as the model. The advantage of this approach is the ability dealing with complex objects that cannot be described by the local features. Because of this advantage, we confine ourselves to the latter approach.

Appearance models for visual tracking can be classified into two types:

- Single appearance based model: Object model is designed based on an object image.
- Multiple appearance model: Object model is designed based on multiple object images. We regard 3D model is equivalent to this model.

The well-known single appearance based object tracking is the template matching[2]. For adapting to the appearance changes, variable templates have been proposed, which deals with affine transform, perspective projection (for planar object), elastic deformation, illumination changes, and so on. As well, updating template matching is widely used, which substitutes the template by the sub-image that matches to the template.

The single appearance based tracking is easily applied to simple tracking problems, i.e., only by specifying the object region on an image frame, object pose on each successive frame can be estimated. However, a drastic appearance change causes erroneous estimation. Especially, template updating suffers from misupdating, which causes tracking failure.

For solving these problems, multiple appearance models have been proposed. These models are classified into three types:

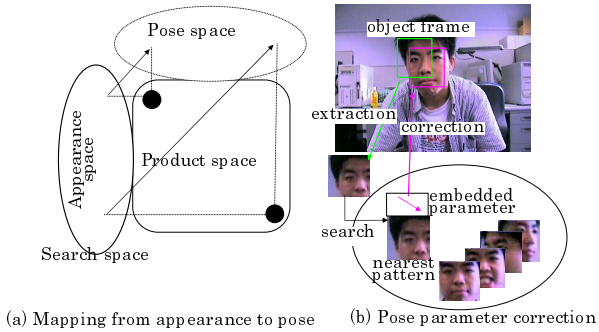
- 3D models: designed or measured geometric models [11].
- Subspace based model: subspaces spanned by the eigenvectors of correlation or covariant matrix [9].
- Parametric eigen-space[12] based model: object models represented by the manifold in the subspace [4].

Tracking methods employing these models are more robust than the single appearance based methods. This is because the models involve possible appearances of the object.

3D model based tracking methods are robust, but it requires special measurement device or design process to obtain the 3D model suitable for the target. Subspace and parametric eigen-space models have a potential that geometric and photometric invariants can be obtained in the subspace. The drawback of these methods is the conflict between the accuracy and the speed. For the real-time processing, we cannot use high-dimensional subspace, which does not guarantee accurate matching. Conversely, for mapping an image into high-dimensional space, it requires many inner product operations between images, which slows down the processing speed.

2.2. Nearest Neighbor Search

The nearest neighbor (NN) search in the appearance space should be done in real-time (video-rate). A brute-force NN search cannot find the nearest pattern in real-time from a huge number of appearances.



(a) Mapping from appearance to pose (b) Pose parameter correction

Figure 2. Pose parameter embedding

NN search based on space decomposition, such as, k-d tree[3] or ANN [1] cannot be applied to this problem, because the appearance (image) vector is a pretty high dimensional vector and the tree construction may fail in these method.

Tree based nearest neighbor search in metric space has been proposed, e.g., VP-tree[14] MVP-tree[5] and GNAT[6]. These methods are not suitable for the continuous NN search problem, because every NN search using tree structure starts from the root node and the previous search result cannot be utilized for successive search.

Graph based nearest neighbor search algorithm is suitable for the tracking problem, because the previous NN search result can be used as the starting point of the next search. All graph based nearest neighbor search algorithms employ nearest-first traversing (NstFT) described in 4.1. For accurate NN search by NstFT, we have to use Delaunay graph[13]. However, the graph cannot be constructed in high-dimensional space, because the construction cost of the Delaunay graph with n vertices in D-dimensional space is $O(n^{\lfloor D/2 \rfloor+1})$.

3. Pose Parameter Embedding

Pose parameter embedding defines the mapping from appearances to pose parameters (Figure 2 (a)). This is done by embedding the pose correction parameters to misaligned appearances. In the tracking procedure, by finding the pattern nearest to the sub-image in the object frame, embedded parameters are also found. According to the embedded parameters, misaligned object frame can be corrected (Figure 2 (b)). As a result, object in the image space can be tracked only by searching the appearance space.

The idea of PPE is similar with the idea of "motion template" [9] , which directly represents the changes in brightness induced by the object motion. By using the motion template, object motion parameters can be estimated from the matching result. This means that the search in the pose parameter space is not necessary. The difference of the PPE from the motion template is that PPE can be more robust, because PPE does not "estimate" the object motion. It stores actual pose correction parameters corresponding to the misaligned images, and we can find accurate parameters by the nearest neighbor search.

4. Nearest First Traversing Graph

4.1. NN search using NFTG

By PPE, the problem becomes the NN search among the stored images. For the real-time search of continuous query images, we employ graph based search algorithm: nearest-first traversing (NstFT). Here, we provide a formal description of NstFT.

Algorithm 1 (Nearest-first traversing (NstFT)) **Step1** *Suppose a graph G consisting of vertices (images) V and edges E: $G = (V, E)$. Select a vertex $\mathbf{x}_c$ from V as the current vertex.*

Step2 *Create the vertex set adjacent to $\mathbf{x}_c$: $A(\mathbf{x}_c) \equiv \{\mathbf{x} \mid (\mathbf{x}, \mathbf{x}_c) \in E\}$. For each element $\mathbf{x}_k \in A(\mathbf{x}_c)$, distance between $\mathbf{x}_k$ and query (image) $\mathbf{x}_q$ is computed and find the minimum distance: $d(\mathbf{x}_q, A(\mathbf{x}_c)) = \min_{\mathbf{x}_k \in A(\mathbf{x}_c)} d(\mathbf{x}_q, \mathbf{x}_k)$, where $d(\mathbf{x}_i, \mathbf{x}_j)$ represents the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$.*

Step3 *If $d(\mathbf{x}_q, A(\mathbf{x}_c)) < d(\mathbf{x}_q, \mathbf{x}_c)$, i.e., the nearest vertex in the adjacent vertices is nearer than the current vertex, set the nearest vertex as the current vertex: $\mathbf{x}_c := \arg \min_{\mathbf{x}_k \in A(\mathbf{x}_c)} d(\mathbf{x}_q, \mathbf{x}_k)$, and go to Step2. Otherwise $\mathbf{x}_c$ is nearest to $\mathbf{x}_q$.*

As discussed above, if we use Delaunay graph as G, we can find the NN image by this algorithm. But the computational complexity of the graph construction is very expensive, and the number of distance computations is big, because the number of edges connected to a vertex is at least $D+1$ in D-dimensional space. That is, the Delaunay graph based nearest neighbor search is impractical.

Our idea is to relax the NN criterion, i.e., guaranteeing true NN search only for $\mathbf{x}_q \in V$. This relaxed condition is defined as below.

Definition 1 (quasi-nearest neighbor: QNN)
Let $\mathbf{x}_q$ be the query, $\mathbf{QNN}(\mathbf{x}_q)$ be the NstFT result for $\mathbf{x}_q$, and $\mathbf{NN}(\mathbf{x}_q)$ be the true NN for $\mathbf{x}_q$. Then the condition is $\mathbf{x}_q \in V \Longrightarrow \mathbf{QNN}(\mathbf{x}_q) = \mathbf{NN}(\mathbf{x}_q)$.
We call $\mathbf{QNN}(\mathbf{x}_q)$ quasi-nearest neighbor.

The problem is how to define the graph for $\mathbf{QNN}$. This graph is named Nearest First Traversing Graph (NFTG) having the following properties:

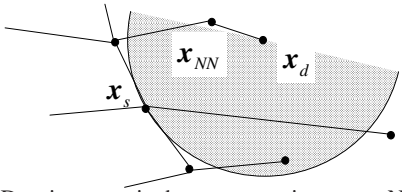


Figure 3. Despite $\mathbf{x}_{NN}$ is the nearest point to $\mathbf{x}_d$, NstFT starting from $\mathbf{x}_s$ to $\mathbf{x}_d$ is stuck at $\mathbf{x}_s$.

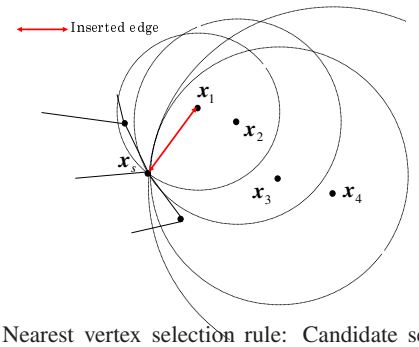


Figure 4. Nearest vertex selection rule: Candidate set is $C = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ for vertex $\mathbf{x}_s$. By inserting an edge $(\mathbf{x}_s, \mathbf{x}_1)$, non-stuck condition stands for all vertices in C.

Connectivity: There must be at least a path between arbitrary pair of vertices in the graph.

Monotonicity: NstFT can find NN vertex $\mathbf{NN}(\mathbf{x}_q)$ for any $\mathbf{x}_q \in V$. In other word, in NstFT procedure, the distance $d(\mathbf{x}_c, \mathbf{x}_q)$ between the current vertex $\mathbf{x}_c$ and query image $\mathbf{x}_q$ must decrease monotonically and finally $\mathbf{x}_c = \mathbf{x}_q$ stands.

If the monotonicity stands by adding an edge to a non-monotonic graph, then the connectivity as well as the monotonicity stand at this moment. Then the problem becomes how to guarantee the monotonicity.

4.2. Monotonic Graph

For guaranteeing the monotonicity, we first investigate non-monotonic graph, which contains stuck points, i.e., points satisfying the following condition:

Definition 2 (Stuck condition)
If a vertex $\mathbf{x}_s \in V$ *does not have adjacent vertex* $\mathbf{x} \in A(\mathbf{x}_s)$ *satisfying* $d(\mathbf{x}, \mathbf{x}_d) < d(\mathbf{x}_s, \mathbf{x}_d)$*, NstFT from* $\mathbf{x}_s$ *to* $\mathbf{x}_d$ *is stuck* at $\mathbf{x}_s$*. We denote this condition by a predicate* $Stuck(\mathbf{x}_s, \mathbf{x}_d)$.

An example of stuck point is illustrated in Figure 3. As shown in this figure, since $\mathbf{x}_s$ does not have neighboring vertex nearer to $\mathbf{x}_d$ than $\mathbf{x}_s$, NstFT is stuck at $\mathbf{x}_s$.

By negating the predicate, we can represent *non-stuck condition* $\neg Stuck(\mathbf{x}_s, \mathbf{x}_d)$. Note, this condition does not guarantee that NstFT from $\mathbf{x}_s$ to $\mathbf{x}_d$ is not stuck, i.e., this local condition is just a necessary condition for the global property; monotonicity. However, if the non-stuck condition stands for any combinations of $\mathbf{x}_s$ and $\mathbf{x}_d$, then NstFT will not be stuck at any vertex. This leads the following Lemma.

Lemma 1 (Monotonic condition) *Those graphs having no stuck point are monotonic. Formally, a monotonic graph satisfies* $^{\forall}\mathbf{x}_s, \mathbf{x}_d \in V \left\{^{\exists}\mathbf{x} \in A(\mathbf{x}_s) \left\{d(\mathbf{x}, \mathbf{x}_d) \leq d(\mathbf{x}_s, \mathbf{x}_d)\right\}\right\}$.

4.3. NFTG construction

The monotonic condition defines the class of monotonic graphs. Then the next problem is how to create an instance of monotonic graph for given vertices V.

By iterating the edge insertion between those vertices$\mathbf{x}_s$ and $\mathbf{x}_e$ satisfying $Stuck(\mathbf{x}_s, \mathbf{x}_e)$, an NFTG can be created. The NFTG created by this procedure depends on the order of edge insertions, i.e., if we change the order, different NFTG will be created.

However, finding the optimal edge insertion is obviously time consuming. Instead, we employ a simple rule, nearest vertex selection. This is because most of the spheres centered at $\mathbf{x} \in C$, $(C = \{\mathbf{x} \in V | Stuck(\mathbf{x}_s, \mathbf{x})\})$ that pass $\mathbf{x}_s$ are nested and the smallest one has the maximum possibility that other spheres include it as shown in Figure 4. This implies that if we insert an edge between $\mathbf{x}_s$ and its nearest vertex in C, the stuck condition between $\mathbf{x}_s$ and other vertices in Cwill be relaxed. For a vertex $\mathbf{x}_s$ in this figure, candidate set is $C = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$. By inserting an edge $(\mathbf{x}_1, \mathbf{x}_2)$ selected by the nearest vertex selection rule, non-stuck condition stands for all vertices in C.

Based on the discussion above, we propose Algorithm 2 for NFTG construction.

Algorithm 2 (Nearest-first traversing Graph construction)

Step1 *Let* V *be a given vertex (image) set, and* $E = \phi$.

Step2 *For each* $\mathbf{x}_s \in V$,
$C = \{\mathbf{x} \in V | Stuck(\mathbf{x}_s, \mathbf{x})\}$, $\mathbf{x}_d = \arg\min_{\mathbf{x} \in C} d(\mathbf{x}_s, \mathbf{x})$
$E = E \cup \{(\mathbf{x}_s, \mathbf{x}_d)\}$

Step3 *Check whether* $G = (V, E)$ *is monotonic or not. If monotonic then the graph construction completed. Otherwise, go to* **Step2**.

An example of the NFTG construction by Algorithm2 is shown in Figure 5. The graph is constructed after three iterations of Step2 in Algorithm2.

4.4. Properties of NFTG

In the case of Figure 5, the mean degree (number of edges connected to vertex) is 3.2, which is greater than the

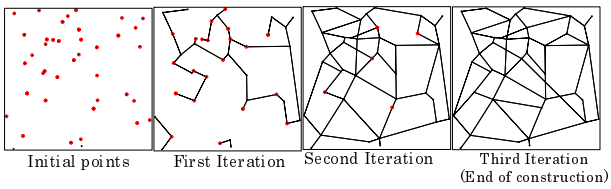


Figure 5. NFTG construction for 40 vertices in 2D space. (Edges 64, Mean number of branches 3.2,)

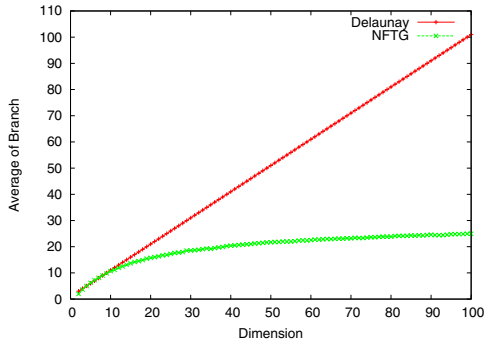


Figure 6. Mean degree (number of edges connected to a vertex) against dimensions. In the case of Delaunay graph, theoretical lower bound of degree is shown, because the graph construction in high dimensional space is impossible. The error bars represent the standard deviations. Each graph has 1000 vertices.

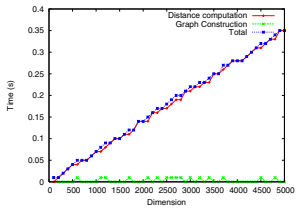


Figure 7. Elapsed time for graph construction against dimensions. Each graph has 5000 vertices generated randomly.

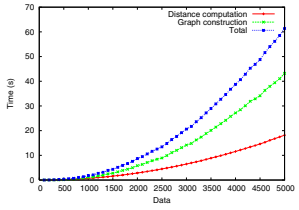


Figure 8. Elapsed time for graph construction against number of vertices. Vertices are generated randomly in 100 dimensional space.

lower bound of Delaunay graph ($D+1=3$), where D is the dimension of the space. This fact implies that the NstFT using Delaunay graph is faster than NstFT using NFTG in 2D case. For clarifying this point, we constructed NFTG for randomly generated 1000 vertices by changing the dimension of the space. Figure 6 shows the mean degrees of NFTG and Delaunay Graph. From this figure, we can notice that NFTG has smaller degree in higher dimensional space than 10, and it saturates to 25 when the dimensionality increases.

The computational complexity of the graph construction for n images in D dimensional space can be evaluated as

- $O(Dn^2)$ for distance table computation,
- $O(B(D)n)$ for graph construction, where $B(D)$ represents the mean degree. This is because the number of distance table lookup and comparison depends on the number of iterations of Step2 in Algorithm2 that is proportional to $B(D)$.

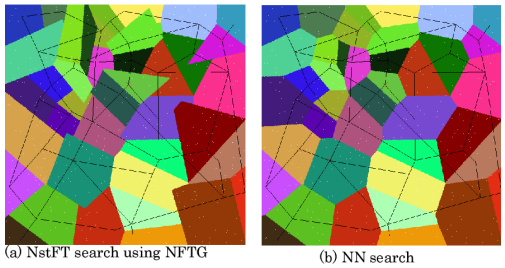


Figure 9. Comparison between NFTG based search and NN search: 2000000 query points are randomly generated and these points are colored by "nearest" vertex colors.

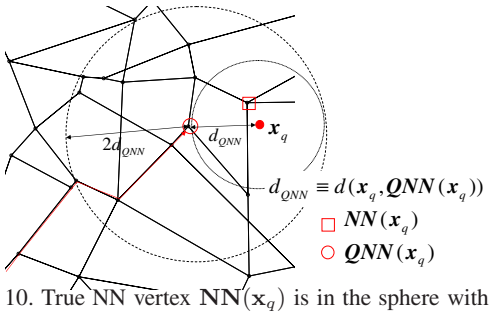


Figure 10. True NN vertex $\mathbf{NN}(\mathbf{x}_q)$ is in the sphere with radius $2d_{QNN}$ centered at $\mathbf{QNN}(\mathbf{x}_q)$.

Figure 7 and 8 show the elapsed time against space dimensions and number of vertices (images), respectively. These results are obtained by using a Linux (kernel-2.2.6) PC with Xeon 3.06 GHz dual CPU and 4GB memory. From these figures, we can notice the following facts: 1) Distance computation is dominant in the total elapsed time, 2) NFTG can be constructed in high-dimensional space.

4.5. Refinement for NN search

As described before, NstFT search using NFTG is not equivalent to NN search for those queries $\mathbf{x}_q \notin V$. We call this search quasi-nearest-neighbor QNN search. The difference between NN and QNN is shown in Figure 9. From this figure, we can confirm that NFTG based search can find correct result for queries close to vertices.

For the true NN search, we can utilize the NFTG search result. Let $\mathbf{x}_q$ be the query, $\mathbf{QNN}(\mathbf{x}_q)$ be the NstFT result for $\mathbf{x}_q$ using NFTG, and $\mathbf{NN}(\mathbf{x}_q)$ be the true NN vertex. As shown in Figure 10, the following inequation stands:

$$d(\mathbf{QNN}(\mathbf{x}_q), \mathbf{NN}(\mathbf{x}_q)) \leq 2d(\mathbf{QNN}(\mathbf{x}_q), \mathbf{x}_q).$$

By using this constraint, we can find $\mathbf{NN}(\mathbf{x}_q)$ starting from $\mathbf{QNN}(\mathbf{x}_q)$. We can use breadth first graph search algorithm with the above bounding condition. Actually, Figure 9 (b) is drawn by the graph-search based on this refinement algorithm.

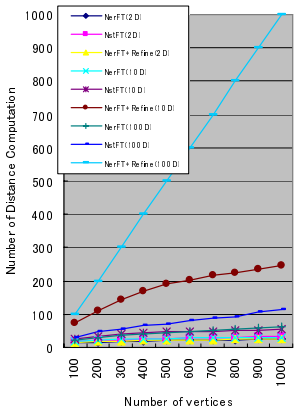


Figure 11. Number of distance computation against number of vertices.

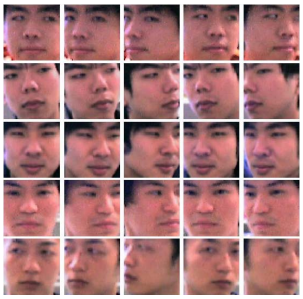


Figure 12. Prototype Images without translation and scaling.

4.6. Further Acceleration

It is clear if a graph can be used for NstFT (Nearest First Traversing), the graph can be used for Nearer First Traversing (NerFT). By slightly modifying the current vertex selection rule in Algorithm 1, NerFT can be realized. That is, if a vertex $\mathbf{x}_k$ adjacent to current vertex $\mathbf{x}_c$ satisfies $d(\mathbf{x}_q, \mathbf{x}_k) < d(\mathbf{x}_q, \mathbf{x}_c)$, then $\mathbf{x}_k$ is selected as next $\mathbf{x}_c$. NerFT is faster than NstFT while keeping the QNN property, i.e., NerFT using NFTG can find NN vertex for $\mathbf{x}_q \in V$. However the search result of NerFT differs from that of NstFT for $\mathbf{x}_q \notin V$ using the same NFTG.

5. Experiments

In the experiment, we first examine the speed of proposed search algorithms, and show the simultaneous tracking and recognition results.

5.1. Experiment 1: Search speed

Figure 11 shows the number of distance computations under different conditions and algorithms. In this experiment, vertices are grouped into several clusters using k-means clustering, and one of the cluster center nearest to the query is selected as the starting point of the search. In 2D space, NerFT, NerFT with refinement, and NstFT require very small number of distance computations (NerFT $<$ NerFT+ Refinement $<$ NstFT). In 10D space, the order is changed (NerFT $<$ NstFT$<<$NerFT+Refine). In 100D space, NerFT+Refine method degenerates to brute force search. This fact shows that pure NN search is impractical in high dimensional space. If we can use densely sampled prototypes and/or the tracking task does not require accurate but fast NN search, we should use NerFT or NstFT.

Table 1. Elapsed time and Number of distance computation for each image frame

	Elapsed time[ms]		Distance computation[times]	
	NstFT	NerFT	NstFT	NerFT
Sequence1	5.7	3.98	27.922	16.971
Sequence2	6.25	4.00	29.324	17.529
Sequence3	5.7	3.89	24.487	16.186
Sequence4	6.5	4.21	29.540	18.163
Sequence5	6.14	3.92	28.887	15.657

5.2. Experiment 2: Object detection, tracking and recognition

This experiment demonstrates the results of object detection, tracking, and recognition. These three tasks are done by a single algorithm; NstFT (or NerFT) using NFTG with Pose Parameter Embedding.

We applied our algorithm to face detection, tracking, and recognition problem. Each prototype is 40x40 RGB color image, i.e., 4800 dimensional vector. The number of persons is 5, and 5 images are taken from different directions for each. Number of horizontal and vertical translation is 49 (=7x7), and the number of scaling is 3 (1.25, 1, 0.75). Then the number of prototype image is 3675=(5x5x7x7x3). The prototype images without translation and scaling is shown in Figure 12.

On a Linux (kernel-2.2.6) PC with Xeon 3.06 GHz dual CPU and 4GB memory, elapsed time for distance table computation is 521 [sec] and for graph construction 5.0[sec] without SSE2 acceleration.

Compared with Figure 6, the resulted NFTG has less mean degree; 14.105345 even in 4800 dimensional space. This is because the distribution of the prototype images is not uniform. This non-uniform distribution is also convenient for NFTG construction, i.e., the number of Step2 iteration in Algorithm 2 is only 22.

In the detection phase at initial frame, 24 object frame positions are examined for 640x480 image to find initial position. It consumes 0.130 [sec].

In the tracking phase, NstFT (or NerFT) is performed for each image frame using vertex obtained in the previous search. The elapsed time and the number of distance computation for each image frame are summarized in Table 1.

From this table, we can notice that both NstFT and NerFT can track and recognize the object within the NTSC video cycle (33[ms]). NerFT is faster than NstFT.

Figure 13 shows snap shots of the tracking, and recognition results. Since NerFT and NstFT produces almost the same results, NstFT results are shown in this figure. Green rectangle represents object frame and the search results are shown at right bottom corner of each image. From this figure, we can confirm the accuracy of the search.

Figure 14 shows the correspondence between image

Figure 13. Snap shots of the tracking and recognition.(Sequence 5 is omitted.)

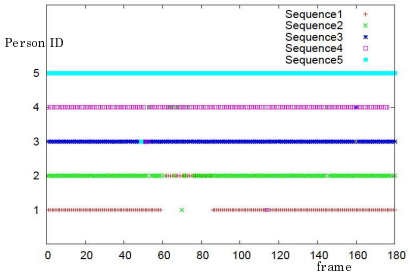


Figure 14. Recognition Result.

Figure 15. Comparison between NstFT and NstFT +Refinement(NN search). Top: NstFT (QNN), Bottom: NstFT+Refinement (NN).

frame and person ID. Since this recognition can be regarded as 1-NN classification, it has enough generalization property when we use many prototype images[8]. From this figure, we can clearly classify the person in image sequences based on the majority of the recognition results. The advantage of simultaneous object tracking and recognition is that we can accumulate the series of recognition results for each object frame to obtain a reliable recognition result.

In the case of true NN search by NstFT+Refinement, mean elapsed time and distance computation are 3100[ms] and 1587.8[times], respectively. From this result, it is clear that true NN search cannot be applied to real-time application. Figure 15 shows the comparison between NstFT(QNN search) and NstFT+Refinement(NN search) for Person ID:2. From this figure, we can notice the difference between them can be negligible.

6. Conclusion

In this paper, we proposed PPE and NFTG for simultaneous object tracking and recognition. By PPE technique, the problem is transformed to the appearance search problem. NFTG is the data structure for continuous and approximate NN search in high-dimensional space. Based on both methods, we successfully realized real-time tracking and recognition.

Since this paper mainly focuses on the algorithms for simultaneous object tracking and recognition, the current system can be easily affected by illumination change. This problem will be addressed in the future work.

References

[1] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998. 3

[2] D. Barnea and H. Silverman. A class of algorithms for fast digital image registration. *IEEE Trans. Computers*, 21:179–186, 1972. 2

[3] J. Bentley. Multidimensional binary search trees used for associative searching. *Comm. of ACM*, 18:509–517, 1975. 3

[4] M. J. Black and A. Jepson. Eigentracking: Robust matching and tracking of articulated objects using a view-based representation. *IJCV*, 26:63–84, 1998. 2

[5] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high dimensional metric spaces. *1997 ACM SIGMOD*, 1997. 3

[6] S. Brin. Near neighbor search in large metric spaces. *21st Conf. on very large database*, pages 574–584, 1995. 3

[7] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. *CVPR*, 2:142–149, 2000. 2

[8] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, IT-13:21–27, 1967. 7

[9] G. Hager and P. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:1125–1139, 1998. 2, 3

[10] M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. *ECCV*, 1:343–356, 1996. 2

[11] I. Matthews and S. Baker. Active appearance models revisited. *IJCV*, 60:135–164, 2004. 2

[12] H. Murase and S. Nayar. Visual learning and recognition of 3-d objects from appearance. *IJCV*, 14:5–24, 1995. 2

[13] A. Okabe, B. Boots, K. Sugihara, and S. Chiu. *SPATIAL TESSELATIONS: Concepts and Applications of Voronoi Diagrams*. JOHN WILEY & SONS, Chichester England, 2nd edition, 2000. 3

[14] P. Y. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. *Fourth Annual ACM-SIAM Symp. on Discrete Algorithms*, 1993. 3