Tracking the Pose of Objects through Subspace

Simon Léonard and Martin Jägersand

University of Alberta, Edmonton AB, Canada
sleonard,jag@cs.ualberta.ca

Abstract. Tracking the pose of an object is a fundamental operation in computer vision. Yet, achieving this task for arbitrary objects without requiring *a priori* knowledge remains a major stumbling block. This paper introduces a method for tracking the pose of a moving object without requiring its 3D model or textured surfaces. In the first step, a sequence of images-poses pairs is obtained and PCA coefficients are derived from the image sequence. Then, a piecewise linear observation mapping is build between the poses and the PCA coefficients. The mapping is then used in the observation model of a Kalman filter that tracks the pose of the object.

1 Introduction

Over the years, there has been many methods that successfully used 2D tracking to recover the 3D pose [10]. One drawback of such methods lies in the robustness of the 2D tracking. In order to track 2D patches in a sequence of images each patch must be discriminating enough, which is a characteristic offered by textured surfaces. However, for objects without salient features (low textured surfaces) these methods become unreliable.

When tracking an object fails because of the lack of small visual attributes, one can often resort to the overall look of the object. Such "look" can be provided from various sources such as a 3D CAD model [6] or based on typical 2D views [8, 1].

The present introduces a method for tracking the pose of an object by using an imaged based representation. The representation is obtained by representing different views of the object in a low dimensional subspace. Such a procedure is known as principal component analysis (PCA). Also, during the data acquisition, the actual state of the object (3D pose) is recorded by a magnetic tracker apparatus, such that a coefficients-state pair is obtained for each view.

It has been shown that under varying points of view, the PCA coefficients vary on a smooth manifold [7, 8]. Therefore, similar views will have similar decompositions in the subspace. Furthermore, this implies that the set of coefficients of all views can be clustered into subsets representing similar views. By assuming local linearity, a linear mapping between coefficients and states can be obtained for each subset. Finally, these mappings are used in a Kalman filter that is modified to accommodate all the different observation models.

J. Bigun and T. Gustavsson (Eds.): SCIA 2003, LNCS 2749, pp. 371–378, 2003.

2 Previous Work

Subspace methods have been widely used in computer vision. In motion analysis, it is often used to constraint measurements such as optical flow [2] or motion [10] into a low dimensional subspace. As mentioned above, these methods often suffer from the ambiguity of the 2D correspondence.

Subspace methods were also used for object recognition [8] and for face recognition [11]. They also accounted for possible translations of the target object or face. These recognition systems have also inspired many tracking methods such as the one presented in [1]. The tracking is performed by robustly matching an input image to an image of the training sequence. The motion is recovered by introducing parameterized motion and simultaneously minimizing a robust objective function over both a set of motion parameter and PCA coefficients. That is, it searches for the PCA coefficients that represent a warping of an image region. In this method, the subspace is only used for encoding image regions and there is no process that is carried within the subspace.

Another method was proposed in [8], where tracking is achieved by decomposing the input image in the subspace and then finding the closest known view (with L_2 norm) in the subspace. The method assumes that consecutive views are strongly correlated and hence their decomposition in the subspace represent a continuous function. This method is akin to object recognition, were the similarity between two views is reflected by similarity in the subspace.

It is important to note that these methods lack the concept of dynamic systems, that is time and state varying models are not taken into consideration. Yet, dynamic system have played an increasing role in computer vision especially in tracking.

An analogous, but inverse application was demonstrated in [3]. Instead of estimating the state of an object given a view, it estimates the view given a state. First the coefficients are obtained by approximating the function that maps the state to the subspace. Then, the coefficients are projected on the subspace basis and the estimate of the image is obtained. The present paper aims at exactly the inverse. That is, to track the state given changes in the observation manifold. Moreover, it integrates a Kalman filter to account for the dynamics of the system, with the particularity that the tracking is performed in the subspace.

3 Subspace Representation

In a dynamic system framework, let a $p \times q$ image be represented by a $pq \times 1$ column vector $\mathbf{I}$ and let $E : \Re^{pq} \rightarrow \Re^k$ where $k \ll pq$ such that $\mathbf{z} = E(\mathbf{I})$. Furthermore, let $\mathbf{x}$ be the d-dimensional state of the object (i.e. pose) and let $H : \Re^d \rightarrow \Re^k$ such that the *observation function* is given by $\mathbf{z} = H(\mathbf{x})$. Therefore, by finding the two functions E and H, it is possible to relate the state $\mathbf{x}$ of an object to its view $\mathbf{I}$. This section shows how to derive the function E from a set of training images, while the next section deals with H.

Let $\mathbf{I}_1, \ldots, \mathbf{I}_M$ be a set of training images representing M points in $\Re^{pq}$. The goal of principal component analysis it to reduce the dimensionality of the set

while preserving as much as possible of the variation inherent to the set. These principal components are defined in terms of eigenvectors of a covariance matrix and are sorted in decreasing order such that the first ones capture most of the variation of the data set [4].

The usual and efficient way of creating the subspace was made popular by [11]. First the average image is computed by $\overline{\mathbf{I}} = \sum_{m=1}^{M} \mathbf{I}_m / M$. Then, for each image $\mathbf{I}_m$, calculate its deviation from the mean by $\Delta\mathbf{I}_m = \mathbf{I}_m - \overline{\mathbf{I}}$ and put the result in a matrix $A = [\Delta\mathbf{I}_1 \ \Delta\mathbf{I}_2 \cdots \Delta\mathbf{I}_M]$. The principal components are given by the eigenvectors of the covariance matrix $C = AA^T$. However, the size of C $(pq \times pq)$ is impractical. For computational efficiency, the eigenvectors of $L = A^T A$ are instead computed and denoted by $V = [\mathbf{v}_1 \ldots \mathbf{v}_M]$. Then, the first k eigenvectors of the subspace are given by normalizing the columns of $U = A[\mathbf{v}_1 \ldots \mathbf{v}_k]$ for $k \leq M$.

Finally, the coefficients $\mathbf{z}$ of an image $\mathbf{I}$ are obtained from

$$\mathbf{z} = E(\mathbf{I}) = U^T(\mathbf{I} - \overline{\mathbf{I}}) \tag{1}$$

4 Observation Model

As described in the previous section, the other step is to find a function $H(\mathbf{x})$ that maps the state to the subspace. Before going any further it is important to look ahead and see what kind of function is sought. First, there is little indication on the shape of the function H and without very good insight it would be unwise to commit to any. As mentionned in [7], H is smooth and thus a reasonable assumption is that H is locally linear, that is a small variation in the state $\mathbf{x}$ will cause a small variation in $\mathbf{z}$. Such an assumption is the foundation of the methods presented in [3, 8].

Given that H is locally linear, the next step is to look at how to incorporate this in a mathematical tracking framework like Kalman filtering. The standard Kalman filtering requires a globally linear observation function, which H is not. The extended Kalman filter accommodates non linear functions by using the Jacobian of the partial derivatives $\mathbb{H}$, hence using local linearity. Unfortunately, to compute the Jacobian $\mathbb{H}$, the function H must be know, which is not the case here.

Instead, a standard Kalman filter can be used with a set of linear observation functions $\mathcal{S} = \{H_1, H_2, \ldots, H_N\}$ where H_i and H_j $(i \neq j)$ represent observation functions over mutually exclusive intervals of the domain of $\mathbf{x}$. This can be understood as covering the state space $\Re^d$ with piecewise linear functions, where each function represent how certain views of the object are related to certain observations.

To achieve this, the subspace must be either tessellated or clustered. Because the observation functions must be build with the available data (measured states and coefficients) and there is no way to ensure that enough points are present in each cell of a tessellation to build reliable functions, the Linde-Buzo-Gray (LBG) vector quantization [5] is used instead for clustering.

4.1 Vector Quantization

Given a set of points $\mathcal{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_M\}$ with $\mathbf{z}_m \in \Re^k$, a set of codevectors (or codebook) $\mathcal{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_N\}$ with $\mathbf{c}_n \in \Re^k$ and a function $K(\mathbf{z}_m) = \mathbf{c}_n$ that assigns a codevector to each point, the LBG algorithm divides the space by calculating $\mathcal{C}$ and K in such a way that it minimizes the average distortion between each point and its codevector, that is

$$D_{ave} = \frac{1}{Mk} \sum_{m=1}^{M} \|\mathbf{z}_m - K(\mathbf{z}_m)\|^2$$

The recursive algorithm starts with one codevector $\mathcal{C} = \{\mathbf{c}_1\}$ that is set to the average of all the points (all points are assigned to $\mathbf{c}_1$). Then, $\mathbf{c}_1$ is split into two codevectors $\{\mathbf{c}_1, \mathbf{c}_2\}$ by adding a small variation to create $\mathbf{c}_1 = (1+\epsilon)\mathbf{c}_1$ and $\mathbf{c}_2 = (1-\epsilon)\mathbf{c}_1$. The next step consists of adjusting $K(\mathbf{z}_m)$ such that each point is assigned to the closest codevector. Then, each codevector is recomputed according to the average of all the points associated to each of them. Finally, the recursion takes place and each codevector is split in two.

Given a codebook and $K(\mathbf{z}_m)$, the next step is to determine the linear observation functions that represent all the members of each codevector. This requires the pairs $p_m = (\mathbf{z}_m, \mathbf{x}_m)$ where $\mathbf{z}_m$ are the coefficients representing the mth image and $\mathbf{x}_m$ is the respective state of the object.

For the pairs $p_1, \ldots, p_f$ associated with the codevector $\mathbf{c}_i$ the corresponding linear observation function is obtained by solving the following system of equations for H_i.

$$\begin{bmatrix} \mathbf{z}_1 \cdots \mathbf{z}_f \end{bmatrix} = H_i \begin{bmatrix} \mathbf{x}_1 \cdots \mathbf{x}_f \end{bmatrix} \tag{2}$$

In order to have reliable observation functions, the choice of the number of training frames, codevectors and the dimensionality of the state must be considered. For example, by assuming that the dimensionality of the tracked pose is six, equation 2 can be written as

$$\begin{bmatrix} \mathbf{z}_1^T \\ \vdots \\ \mathbf{z}_f^T \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_f^T \end{bmatrix} \begin{bmatrix} \mathbf{h}_1 \cdots \mathbf{h}_k \end{bmatrix} \tag{3}$$

where $\mathbf{z} \in \Re^k$ and $\mathbf{h}_i \in \Re^6$. It should be noted that in order to have an overdetermined system f must be greater than 6. Therefore, each codevector should have at least six points assigned to it. However, since there is very little control on how to distribute the data set with the LBG algorithm, a practical way to avoid underdetermined systems is to use more training images. If, as in the following experiments, the codebook contains 128 codevectors, the minimum number of training images would be $128 \times 6 = 768$, such that experiences had us set 1024 as a safe number of training images.

5 Kalman Filtering

Setting up the Kalman filter requires to create a dynamic function and observation function [12].

The dynamic function of the pose of an arbitrary object, i.e. kinematic chain like an arm, is not linear. Therefore, a first order differential dynamic model is used instead of a linear model. That is, using $\delta \mathbf{x}_i = \hat{\mathbf{x}}_i - \hat{\mathbf{x}}_{i-1}$ (where $\hat{\mathbf{x}}$ represent an *a posteriori* estimate), the *a priori* estimate at time $i+1$ becomes $\mathbf{x}_{i+1}^+ = \hat{\mathbf{x}}_i + \delta \mathbf{x}_i$. Incidentally, the process noise covariance matrix Q can be tailored to reflect the uncertainty due to the magnitude of the motion by $Q_{i+1} = \delta \mathbf{x}_i \delta \mathbf{x}_i^T$.

The other required modification is that once the observation $\mathbf{z}_i$ is available, the appropriate observation function H_n must be used. The selection is simply done by choosing the closest codevector $\mathbf{c}_n^*$ and selecting the function H_n associated with it. The cost of performing the exhaustive search is negligible for the size (N) of the codebook used in the experiments.

To summarize, the system can be divided in two parts the training part and the tracking part.

5.1 Training

1. Acquire M images $\mathbf{I}_m$ and M corresponding states $\mathbf{x}_m$.
2. Find the k first eigenvectors of A.
3. For each $\mathbf{I}_m$ compute $\mathbf{z}_m$ with equation 1.
4. Clustering of the coefficients $\mathbf{z}_m$ (find the codebook $\mathcal{C}$ of N codevectors).
5. For each codevector $\mathbf{c}_n$ build an observation function H_n using equation 3.

5.2 Tracking

1. Find the coefficients $\mathbf{z}_i$ of the current frame with equation 1.
2. Find the nearest codevector $\mathbf{c}_j^*$ to $\mathbf{z}_i$.
3. Time update: find the *a priori* estimate $\mathbf{x}_i^+$ and error covariance P_i^+
4. Measurement update: using H_j, find the *a posteriori* estimates $\hat{\mathbf{x}}_i$ and error covariance P_i.

6 Experiments

The state measurements $\mathbf{x}_i$ were obtained with a Polhemus Ultratrak Pro and were sampled at the same rate than the images. The images size is 160x120. The online tracking at 30 fps runs easily on a 1.5GHz laptop with 256MB, while computing the subspace on 1024 frames takes about 10 minutes to compute. The camera is an off the shelf IEEE1394 webcam. In both experiments, the size of the codebook was 128, therefore there was 128 observation models to chose from.

Two experiments were conducted. The first one involves a human circling around a center and the second one is a moving arm. Both experiments used a

6D state space. In the first experiment, one sensor was place on the chest and the state was composed of $[x, y, z, yaw, pitch, roll]^T$ and in the second experiment one sensor (S_1) was placed above the elbow and one sensor (S_2) was placed at the wrist and the state was $[x_{S_1}, y_{S_1}, z_{S_1}, x_{S_2}, y_{S_2}, z_{S_2}]^T$. In both experiments, measurements from the Polhemus were also used as a reference for comparing the estimated states.

Whereas the motion in the first experiment was periodic (a person walking around), this was not the case in the second experiment where the motion was improvised and relatively fast. A few samples of frames are shown in figure 1 and 2. The sequences are available at www.cs.ualberta.ca/~sleonard/track

Figure 3 shows the tracking results for the arm sequence. The number of training images was 1024 and the number of eigenvectors used was 6 ($\mathbf{z} \in \Re^6$). The left image shows the Eucledian error for the elbow and the right image shows the results for the fist. On average, the error was about 11cm for the elbow and 14cm for the fist. The randomness of the error reflects well the uncorrelated and fast motion of the arm.

Figure 4 shows the errors obtained for the circling sequence. The top left image shows the error in position (x, y, z), while the three other images show the error in orientation. The trajectory was on a circle of about 1.2m of diameter. The number of training images was 1024 and the number of eigenvectors used was 4. The shape of the curves clearly reveal the periodicity of the motion. They indicate that the system was less accurate when the subject was moving closer to the camera and magnetic tracker transmitter. At those points is shows that the yaw is off by 15 degrees compared to the Polhemus. Part of the error in orientation can be attributed to the lack of calibration for roll, pitch and yaw of the sensors in our facility. However, the sensors were properly calibrated for position (x, y, z) measurements.

7 Conclusion

This paper has introduced a method for tracking the pose of objects without relying on textured surfaces and salient features. Instead, it uses an image based representation of the overall appearance of the object. This representation has the form of a low dimensional subspace and is obtained from PCA. Using the result that similar views will have a similar representations in the subspace allows to use a set of linear functions to approximate the mapping between the image space and the subspace. These mappings are then included in a Kalman filtering and they account for the observation model of the system.

The results have shown that the system tracks the pose of the object and is accurate enough for many applications, especially in virtual reality. The goal was to show through the experiments how tracking the pose of the human body could be used for applications such as rendering a virtual environment (i.e. in a head mounted display) according to the position of the body or simple virtual arm manipulations.

References

1. Black, M.J., Jepson, A.D.: EigenTracking: robust matching and tracking of articulated objects using a view-based representation. IJCV **25** (1998) 63–84
2. Irani, M.: Multi-frame optical flow estimation using subspace constraints. ICCV (1999) 626–633
3. Jagersand, M.: Image based view synthesis of articulated agents. CVPR (1997) 1047–1053
4. Jolliffe, I.T.: Principal Component Analysis Springer, New York, (2002)
5. Linde, Y., Buzo, A., Gray, R.M.: An algorithm for vector quantizer design. IEEE Transactions on Communications (1980) 702–710
6. Lowe, D.G.: Fitting parameterized three-dimensional models to images. PAMI **13** (1991) 441–450
7. Murase, H., Nayar, S.K.: Visual learning and recognition of 3D objects from appearance. IJCV **14** (1995) 5–24
8. Nayar, S.K., Nene, S.A., Murase, H.: Subspace methods for robot vision. RA **12** (1996) 750–758
9. Shi, J., Tomasi, C.: Good features to track. CVPR (1994) 593–600
10. Tomasi, C., Kanade, T.: Shape and motion from image streams under orthography: a factorization method. IJCV **9** (1992) 137–154
11. Turk, M., Pentland, A.P.: Eigenfaces for recognition. CogNeuro **3** (1991) 71–96
12. Welch, G., Bishop, G.: An introduction to the Kalman filter SIGGRAPH (2001) short course

Fig. 1. Sample images (arm sequence).

Fig. 2. Sample images (circle sequence).

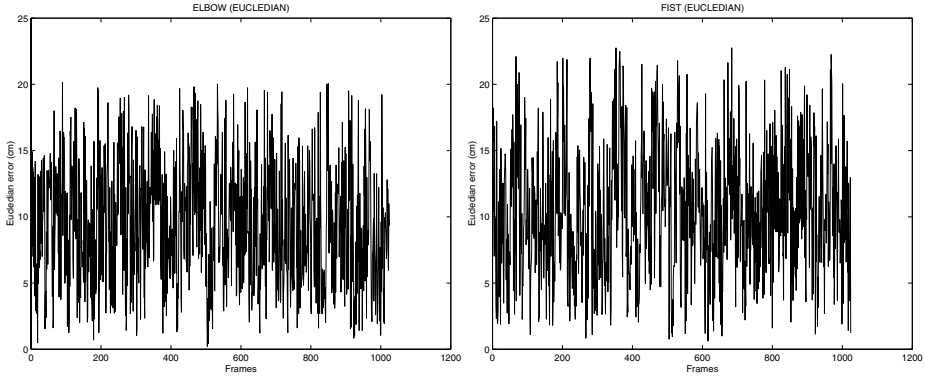

Fig. 3. Arm sequence (relative error).

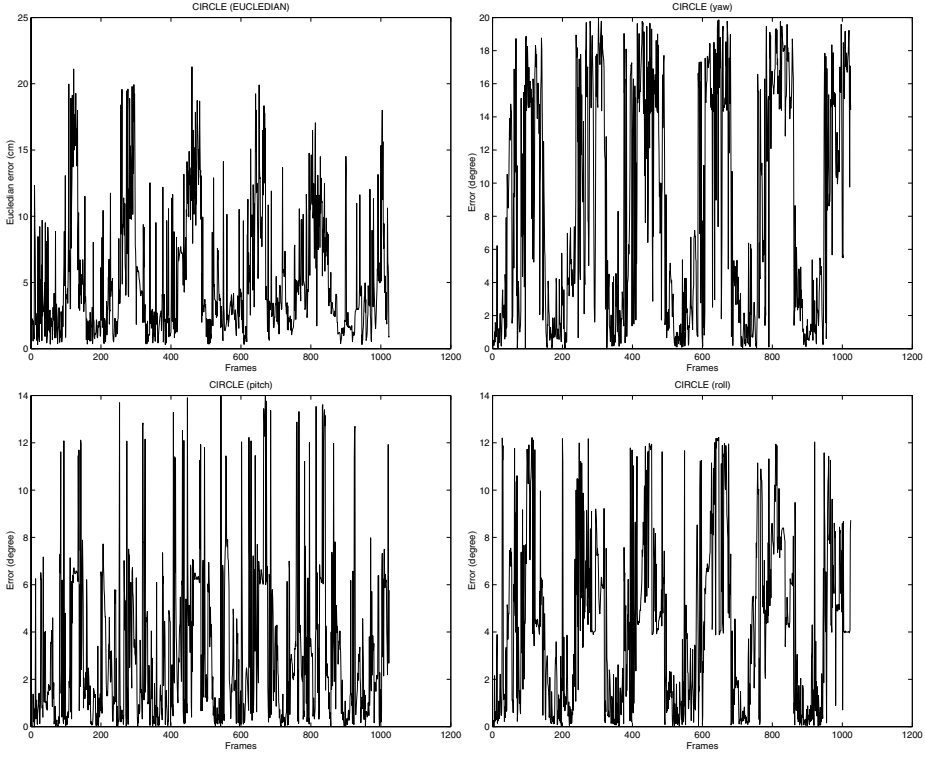

Fig. 4. Circle sequence (relative error).