Visual Tracking by Dictionary Learning and Motion Estimation

Amin Jourabloo, Behnam Babagholami-Mohamadabadi, Amir H. Feghahati, Mohammad T. Manzuri-Shalmani, Mansour Jamzad
Dept. of Comp. Engineering, Sharif University of Technology
{jourabloo,babagholami,fe@ce.sharif.edu, {manzuri, jamzad}@sharif.edu

Abstract—In this paper, we present a new method to solve tracking problem. The proposed method combines sparse representation and motion estimation to track an object. Recently, sparse representation has gained much attention in signal processing and computer vision. Sparse representation can be used as a classifier but has high time complexity. Here, we utilize motion information in order to reduce this computation time by not calculating sparse codes for all the frames. Experimental results demonstrate that the achieved result are accurate enough and have much less computation time than using just a sparse classifier.

Keywords—Machine Vision, visual tracking, Sparse Representation.

I. INTRODUCTION

Tracking is one of the most famous problems in machine vision which has received much attention from researchers due to its obvious uses in many computer vision applications such as object-based video compression [1], security surveillance systems [2], augmented reality [3], etc. Many tracking algorithms have been developed, but still none of them can produce satisfying results in all environments, thus, new methods of tracking are being developed. Some of the challenges in tracking are occlusion, illumination change, scale change, varying viewpoint, and deterioration [4]. Utilizing sum of squared difference (SSD) as a cost function is an early work in visual tracking [5]. Another approach is Particle Filter framework [6] which considers tracking as an estimation of the states for a time series state space model. In [7] an appearance-adaptive model incorporated in a particle filter to do robust tracking. Covariance tracker [8] is another method which uses a covariance based object description, in which different types of features and modalities are combined to track nonrigid objects efficiently.

In the last decade, many researchers from machine learning community have tried to solve this problem as a classification problem with two classes, the object of the interest class and the background class. In this case, a classifier is trained in advance in order to distinguish between the object and the background. A common approach to incorporate a classifier into tracking problems is to use the tracker and the classifier sequentially. The tracker will find where the object moved to and the classifier will give it a score. This scheme will repeat until the classification score reaches a satisfactory point. The most frequently used classifier in this framework is Support vector machine (SVM) [9]. The disadvantage of such an approach is that the tracker is not guaranteed to move to the best location (the location with the highest classification score) but rather find the best matching image region[10]. Kernel based density estimation techniques are another machine learning methods that have been used in visual tracking [11], [12]. In these methods, the visual observations are the distributions of certain visual features (e.g., color) within a region and are expressed in an analytical form by kernel density estimators [13].

Some of the newly introduced classification methods used sparse representation for their goal. Sparse representation firstly used as a reconstruction method such that the reproduced signal uses a minimum amount of predefined blocks of data. That is, the signal is represented by a sparse set of basis functions. The sparse representation usage in machine vision may inspired by works in neuroscience field. Motion estimation is another well-studied field in computer vision. A motion estimator can act like a confidence measure for the tracker to look for the object in certain areas rather than all the image. Also, candidate blocks which are found by motion estimator may have more importance in the learning process. The main contribution of our work is presenting a method in which sparse code and motion estimation are used jointly to tackle the tracking problem. We tested the proposed approach on some video sequences containing all the challenges in the tracking such as heavy occlusion, large illumination and pose changes. The proposed approach shows excellent performance in comparison with three previously proposed trackers, including the IVT [14], the fragment based tracker [15], and $L_1$ tracker [16] in accuracy. The rest of this paper is organized as follow. In section II, a short review on related works of sparse coding is discussed. In section III, the proposed method is presented. In section IV the experimental results are depicted and at last, we conclude our work in conclusion section.

II. RELATED WORKS

Here, we briefly review some of the works in sparse representation field to find out how we can utilize this method to solve the tracking problem. In [17], a comprehensive study on image denoising has been done using sparse representation. The authors studied two types of dictionaries, a pre-defined dictionary and a learned one, to denoise an image. It was so absorbing that a learned dictionary on a set of corrupted images gave satisfactory results. This achievement provided an incentive for others to propose algorithms for learning/updating a dictionary. In [18], a method of updating a dictionary in an
Online manner is introduced. Online methods do not need all the data i.e. in tracking case means all frames to update the dictionary. Instead, the dictionary is updated based on the current data. In [19] authors used a non-convex relaxation unlike other convexifying approaches to obtain the sparse codes. In [20], authors conducted experiments on a variety of problems to show the use of sparse representation in pattern recognition and computer vision field. Their experiments showed that the results obtained by sparse representation are promising while Rigamonti et al. [21] claimed that the sparse coding did not perform better than a standard filter bank for recognition tasks on their datasets. In [22], the authors learned one dictionary per class and observed that there is a high correlation between the learned dictionaries. They reduced this correlation by adding an incoherence term for each optimization problem.

As mentioned before, tracking can be solved by using a two class classifier. Mairal et al. [23] used sparse coding for solving a classification problem by adding a loss function term to generative part of formulation. Authors of [16] used particle filter, a database of templates and sparse coding to solve the tracking and recognition together. The drawbacks of their work are the necessity of existence of a database for recognition task and high computation time.

A. Problem formulation

In recent decade, sparse representation field has got much attention from signal processing community. This attention is due to the fact that many natural signals like natural images are sparse in their nature and can be approximated or even fully recovered by their sparse codes. This reconstruction can be achieved by a linear combination of some bases from a dictionary. This dictionary is defined as \( D = [d_1, \ldots, d_k] \in \mathbb{R}^{n \times k} \) where \( d_i \) is the \( i \)-th basis and \( n \) is the dimension of each basis. Suppose that we want to find some bases from dictionary, such that the reconstructed signal, \( \hat{X} \), has the most resemblance to the original signal, \( X \). The formulation of this problem is demonstrated in equation bellow.

\[
\hat{X} = D\alpha
\]

\[
\hat{\alpha} = \arg\min_{\alpha} \|\alpha\|_0 \quad \text{s.t.} \quad \|\hat{X} - X\|_2^2 \leq T
\]

(1)

Where the vector \( \alpha \) chooses which bases must take part in reconstruction. \( T \) is a predefined threshold and \( \|\cdot\|_0 \) is \( l_0 \)-pseudo norm which is defined in equation 2.

\[
\|x\|_0 = \#\{j \text{ s.t. } x_j \neq 0\} = \lim_{q \to 0} \left( \sum_{j=1}^m |x_j|^q \right)^{1/q}
\]

(2)

Note that \( l_0 \)-pseudo norm means the count of non-zero elements of its input vector. Unfortunately, \( \hat{\alpha} \) cannot be found in an efficient way because of the NP-hard nature of the problem, but it has demonstrated that \( l_1 \) norm is also creates sparse solutions [24]. Thus, the reformulated problem is

\[
\hat{\alpha} = \arg\min_{\alpha} \|X - D\alpha\|_2^2 + \lambda \|\alpha\|_1
\]

(3)

By which both sparseness and minimum energy difference can be achieved. In equation 3, \( \lambda \) is a regularization term which controls the tradeoff between sparseness and reconstruction error. To the best of our knowledge, there is not a theoretical link between \( \lambda \) and the sparseness of the model. Using \( l_1 \) norm rather than \( l_0 \)-pseudo norm has three main benefits. First, it is known that if the solution of the problem at hand is sparse enough, \( l_1 \) norm has the same result as \( l_0 \) pseudo norm. Second, a \( l_0 \) pseudo norm problem has a high time complexity and cannot be solved in a reasonable amount of time. On the other hand, \( l_1 \) norm of this representation converts this nonconvex problem to a convex one. Convex problems are well-studied problems which can be solved by numerous methods such as [25]. The third property which is crucial for the task at hand, tracking, is that the \( l_1 \) norm results are more stable than \( l_0 \) norm. This means that the dictionary is more robust to small perturbations when \( l_1 \) norm is utilized than the situation when \( l_0 \) norm is used [23]. If the dictionary is not a pre-defined one, we have to find both sparse codes and the proper dictionary, thus the general form of the optimization problem changes to

\[
\arg\min_{D, \alpha} \|X - D\alpha\|_2^2 + \lambda \|\alpha\|_1
\]

(4)

which is not fully convex but by fixing one parameter, \( \alpha \) or \( D \), and minimizing the other one, the problem can be treated as a convex problem. Methods such as K-SVD [26] and MOD [27] can be used to find a proper dictionary and a sparse code simultaneously. Both of these methods are iterative approaches and tend to minimize the reconstruction error. In these methods the sparse codes \( a \) and the dictionary \( D \) are updated in two phases. One phase is when the sparse codes are being updated explicitly when \( D \) is fixed. This task can be done by using methods such as greedy orthogonal matching pursuit [28] if we want to use \( l_0 \) norm as measure of sparseness, or by basis pursuit [29] if we want to use \( l_1 \) norm. Another phase is when the atoms of the dictionary are getting updated. In MOD, \( a \) is fixed and all atoms of dictionary are updated using least square criteria, whereas in K-SVD, not only the atoms of the dictionary are updated, but also the non-zero coefficients of the sparse code are updated in the same time.

III. Proposed Method

The bottleneck of sparse representation algorithms is their relatively high computation time for computing sparse codes. To overcome this problem, we present a new method to estimate these sparse codes. Also, by incorporating a motion estimator, the computation time can be significantly reduced. In this method, we compute the sparse codes for i-th frame and estimate the sparse code for the (i+1)-th frame. The assumption behind our method is that two successive frames are very similar, thus, their sparse codes must be similar.

In our method, we consider tracking problem as a two class classification problem. We also create one dictionary for object
Table 1. Dictionary Learning Algorithm \[ z = 1, -1 \]

\[
D^t = [d_1, \ldots, d_k] \in \mathbb{R}^{n \times k} \\
A^1 = [\alpha_1, \ldots, \alpha_k] \in \mathbb{R}^{k \times k} \\
B^t = [b_1, \ldots, b_k] \in \mathbb{R}^{n \times k} \\
1. A^t = \sum_i \alpha_i^2 \alpha_i^T, B^t = \sum_i x_i \alpha_i^T \\
2. Repeat \\
3. for \( j = 1 \) to \( k \) \\
4. Update the \( j^{th} \) column \\
\[
u^j = \frac{1}{\sum_j} (d^j - D_z \alpha^j - \eta (D_z D^T \alpha) d^j + d^j) \\
\]
5. endfor \\
6. Until convergence

class and another one for background class. The outline of our method is depicted in figure 1. In the first frame, a dictionary is learnt based on the object and its background. For other frames, either the sparse code is fully calculated or approximated. The proposed method consists of three main steps, exact sparse coding, approximate sparse coding and, dictionary learning which are going as follows.

A. Exact Sparse Coding

The exact sparse codes are obtained by solving

\[
\arg\min_{\alpha} \|X - DA\|_2^2 + \lambda \|\alpha\|_1 \tag{5}
\]

Where \( \lambda \) is a factor which controls the tradeoff between sparseness and reconstruction error. This model will be solved by \( l_1 \)-regularized least square method [30]. It is known that \( l_1 \) norm holds a sparse solution for coefficient vector \( \alpha \) and also, there is no theoretical relation between \( \lambda \) and the sparseness which is achieved by this model.

B. Approximate Sparse Coding

One assumption which can be used for motion estimation is that a pixel in a frame and its correspondence in the next frame are expected to be the same with respect to their intensity values [31]. Block region estimator solve the ill-posedness of this problem by defining a support region. This support region defined as the neighborhood of the pixel of interest. The exact motion vector is the one that minimizes the intensity difference between the pixels in the original block and the pixels in the estimated block.

Thus, the predicted motion vector, \( \hat{v}_i = (v_x, v_y) \), can be estimated by

\[
\hat{v}_i = \arg\min_{v} AD_i(v) \\
AD_i(v) = \sum_{x, y \in B_i} |f(x, y) - f^-(x - v_x, y - v_y)| \tag{6}
\]

In which \( AD_i(v) \) is the absolute difference between pixel's intensity in block \( B_i \) and \( f \) and \( f^- \) are the current and previous frame data respectively.

Approximating sparse codes of a frame needs the sparse codes of its previous frame. So, with the assumption that the dictionary is fixed and the exact sparse codes of patches of the previous frame are already calculated, the sparse code of each patch of current frame is approximated by the sparse code of its most similar patch from the previous frame. To have a more robust approximation, we also consider the reconstruction error, so, the sparse codes in approximation phase are

\[
\hat{\alpha}_{i,t} = \arg\min_{\alpha} \|x_{i,t} - DA\|_2^2 + \lambda \|\alpha - \alpha_{i,t-1}\|_2^2 \tag{7}
\]

In which \( x_{i,t} \) is the \( i^{th} \) patch of frame \( t \). \( \hat{\alpha}_{i,t} \) is the approximate sparse code for patch \( i \) of frame \( t \) and \( \alpha_{i,t-1} \) is the exact sparse code of the most similar patch of frame \( t - 1 \) to patch \( i \) of frame \( t \) which is determined using equation 6. The first term of the right hand side of above equation is reconstruction error and the second term is the similarity between the sparse codes of the previous and current frames. Hence, the closed form solution of (7) is

\[
\hat{\alpha}_{i,t} = (D^T D + \lambda I_{k \times k})^{-1} (D^T x_{i,t} + \lambda \alpha_{i,t-1}) \tag{8}
\]

In which \( I \) is Identity Matrix and \( k \) is the dimension of each sparse code. Note that the eq. (7) can be computed efficiently.

C. Dictionary Learning/Updating

The dictionary learning method which is used in this work is introduced in [22]. The learning algorithm to find two dictionaries for object and background is shown in table 1, where \( z \) is the label of the classes and \( \eta \) controls the correlation of the two dictionaries. This method of dictionary learning uses warm restart to update it bases, thus it has faster computation.
time.

Labels for patches are assigned by using $S$.

$$\text{minimize}_S S(x, \alpha, D_z, \lambda) = \|X - D_z \alpha\|_2^2 + \lambda \|\alpha\|_1 \quad (9)$$

The result of (9) is calculated using both object dictionary and background dictionary. The label which minimizes the $S$ is used as the patch label. Calculating sparse code method with is used in this work is presented in [32]. The learning process can be repeated in i-th frames to update the dictionary. Updating the dictionary add robustness to the method against shape change or occlusion.

IV. EXPERIMENTAL RESULTS

In this section we report the performance of our algorithm and compare its results with three other trackers, IVT [14], Fragment based Tracker [15] and $L_1$ tracker [16] in which the $L_1$ tracker is a sparse coding based tracker. These algorithms are tested on some well-known videos in tracking field such as the sequence named "Red Team". Our method is implemented in MATLAB 7.10 on a 2.0 GHz Intel Corei7 machine with 4.0GB of main memory.

The initial condition for our tracker is defined as follow. Object and its background from first frame are given from user by two bounding boxes. Size of each patch is $9 \times 9$ and there are 20 bases in each dictionary. Patches of each region have 50% overlap. We used RGB and HoG features i.e. 243 RGB values and 80 HoG features. We have used two dictionaries, one for background and one for object. In our expriments, the parameter $\lambda$ was set to $\frac{1}{2}$.

We also should mention that we feed our tracker the even frames rather than all frames to get better computation time. Figures 2 and 3 show the qualitative and quantitative results of our method in comparison with three other methods. As it can be seen, our method has more accurate results. In Fig 2. We compare the Euclidian error between our method and other methods. In the left panel, we conducted an experiment on "David outdoor" sequence. Our method shows promising accuracy except for frames $i = 44$ till 60. The poor performance of our method roots in the fact that our method tries to find similar patches in a small neighborhood. On the right panel, our method has superior performance because of the fact that our method uses a patch-based representation and updating the dictionary, thus, it can adapt itself to the severe occlusion in the sequence.

Since $L_1$ tracker is a sparse coding based method, we compared our method with this method in computation time. Table 2 shows the runtime comparison of our proposed method versus $L_1$ tracker method. It can be seen from table 2 that our method is faster than $L_1$ tracker method because our method does the optimization in equation 3 only for even frames and uses equation 9 to calculate the sparse codes of odd frames.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Proposed Method</th>
<th>L1 tracker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surfer</td>
<td>2.3</td>
<td>4.1</td>
</tr>
<tr>
<td>David outdoor</td>
<td>3.9</td>
<td>3.5</td>
</tr>
<tr>
<td>Red Team</td>
<td>1.2</td>
<td>5.8</td>
</tr>
<tr>
<td>Woman</td>
<td>1.8</td>
<td>5.5</td>
</tr>
</tbody>
</table>

Table 2. average of computation time (in seconds per frame)

The novelty of the proposed method is that it combines a motion estimator and a sparse code classifier to achieve proper results. Also, the dictionary learning process of this work has two properties, it is online and supervised which achieves better results than a pure supervised dictionary learning method.

REFERENCES

Fig. 2. The Euclidian distance error for each tracker. Left:"David Outdoor” sequence, Right:"Surfer” sequence.

Fig. 3. Each section shows the result of applying tracking algorithms. Top Left:"Surfer” sequence. The challenge in here is occlusion. Top Right:"David Outdoor” sequence. The challenge in here is its varying in illumination and movements of the camera. Bottom Left:"Red Team” sequence. The challenge in here is zooming in and out of the camera. Bottom Right:"Woman” sequence. The challenge in here is changes in object shape. (Blue=Proposed Tracker, Red=IVT, Green=Fragment Tracker, Magenta=L1 Tracker).


