Boosting-based Visual Tracking using Structural Local Sparse Descriptors

Yangbiao Liu, Bo Ma, Hongwei Hu, and Yin Han

Beijing Laboratory of Intelligent Information Technology,
School of Computer Science and Technology,
Beijing Institute of Technology, Beijing 100081, China

Abstract. This paper develops an online algorithm based on sparse representation and boosting for robust object tracking. Local descriptors of a target object are represented by pooling some sparse codes of its local patches, and an Adaboost classifier is learned using the local descriptors to discriminate target from background. Meanwhile, the proposed algorithm assigns a weight value, calculated with the generative model, to each candidate object to adjust the classification result. In addition, a template update strategy, based on incremental principal component analysis and occlusion handing scheme, is presented to capture the appearance change of the target and to alleviate the visual drift problem. Comparison with the state-of-the-art trackers on the comprehensive benchmark shows effectiveness of the proposed method.

1 Introduction

Visual tracking is an important problem in computer vision and has a wide range of applications in surveillance, robotics, human computer interaction, and medical image analysis. Although steady progress has been made to the speed, accuracy and robustness of object tracking in recent years, it is still a difficult task due to appearance changes of a target object caused by some factors such as illumination variation, occlusion, background clutter, pose variation and shape deformation.

A lot of tracking methods have been proposed to deal with the challenges mentioned above, and readers can refer to the survey papers [1, 2] and a recent benchmark [3]. Most recent tracking algorithms can be roughly categorized as either generative or discriminative approaches. Based on the appearance model of target object, generative tracking methods search the most similar region with the best matching score by some metric. These methods update target appearance model dynamically to make the model fit for the target appearance changes and reduce the drifting problem. Ross *et al.* [4] learned the dynamic appearance of the target via incremental low-dimensional subspace representation to adapt online to changes of target appearance. Recently, numerous tracking algorithms based on sparse representation [5–8] have been proposed due to its robustness to occlusion and noise. In [6], a tracking algorithm was developed with structural

local sparse appearance model, using both partial information and spatial information of the target with alignment-pooling method. Wang *et al.* [9] proposed a least soft-thresold squares tracking algorithm by modeling the error term with the Gaussian-Laplacian distribution. However, background information that is critical for effective tracking isn't considered in these generative models.

Discriminative tracking methods [10–14] usually treat tracking as a binary classification task which separates the object from its surrounding background. These methods first train a classifier in an online manner, then the classifier is applied to candidate targets sampled from next frame. Babenko *et al.* [10] used multiple instance learning (MIL) which put the positive and negative samples into some positive and negative bags respectively to learn a discriminative model to solve ambiguity problem. Kalal *et al.* [12] developed a semi-supervised learning approach in which tracking results were regarded as unlabeled and positive and negative samples were selected with structural constraints. Zhang *et al.* [13] utilized a random sparse compressive matrix to reduce dimensionality of Haar-like features, and then trained a naive Bayes classifier with the low-dimensional compressive features. Discriminative trackers are usually more robust against appearance variations than generative trackers under complicated environments, because discriminative trackers take background information into account. Some hybrid methods that combine generative approach and discriminative approach to get a more robust result were proposed, such as [15–17].

Recently, sparse representation has attracted considerable interest in object tracking due to its robustness to occlusion and image noise etc. Moreover, a large number of experiments suggest that sparse representations are effective models to account for appearance change. In this paper, we present an online visual object tracking algorithm using local sparse appearance representation and an Adaboost classifier. The proposed method samples overlapped local image patches inside the object region and then represents each image patch with its sparse code. Different from [18] which represents a target by concatenating the sparse codes of all image patches, our method represents a target using some local descriptors. Each local descriptor is represented by pooling several sparse codes selected from all sparse codes of the target. And then, an Adaboost classifier can be trained using the local descriptors of positive and negative samples collected in the first several frames. A candidate target has a classification score via the Adaboost classifier, but the classification score is not accurate if the candidate target experiences great appearance variations, so the classification score should be adjusted. The proposed algorithm assigns a weight value to each candidate target to adjust its classification score, and the weight value is calculated by structural reconstruction error of the candidate target. In addition, a template update strategy is applied to capture the appearance change of the target.

2 Proposed Tracking Algorithm

In this section, the proposed tracking algorithm is described in detail. We first show how the local descriptors of target are represented with local sparse codes.

Next, we give a description of training classifier and calculating weight of candidate target. The update strategy of template and classifier is then introduced.

2.1 Local Descriptors Representation by Local Sparse Codes

Given an object image I, we can extract a set of overlapped local image patches $X = \{x_i | i = 1, 2, \cdots, N\} \in R^{d \times N}$ inside the target region with a sliding window, where x_i is the i-th column vectorized local image patch extracted from image I, d is the dimension of the image vectors and N is the number of local patches. If we have an image set of templates $T = [T_1, T_2, \cdots, T_n]$, we extract local image patches from T in the same way mentioned above, and then a dictionary $D = [d_1, d_2, \cdots, d_{N \times n}] \in R^{d \times (N \times n)}$ used to encode local patches of candidate targets can be obtained, where n is the number of templates. Each item of the dictionary is a d-dimensional vector corresponding to a local patch extracted from T. The process of constructing dictionary is similar to [6] that demonstrates the advantage of constructing dictionary in this way.

The first n (n=8) frames are tracked using other tracking algorithm and the tracking result (normalized to 32×32) of each frame is treated as a template, then we obtain the set of templates T.

Each local image patch x_i in X can be encoded with the dictionary D by solving

$$\min_{\alpha_i} \|x_i - D\alpha_i\|_2^2 + \lambda \|\alpha_i\|_1, \tag{1}$$

where $\alpha_i \in R^{(N \times n) \times 1}$ is the sparse code, corresponding to local patch x_i. λ is a regularization parameter that controls sparsity and reconstruction error. Then the sparse coefficient matrix A of the candidate X can be obtained, i.e. $A = [\alpha_1, \alpha_2, \cdots, \alpha_N] \in R^{(N \times n) \times N}$.

In order to fully describe the candidate X, we generate some local descriptors for X with the sparse coefficients. Detailed process is as follows: Given a candidate object region X, we can extract a set of overlapped local image patches (denoted as $x_1, x_2, \cdots, x_N$), and calculate a sparse code for each local patch according to Eq.1. We select M local patches from all N local patches of the object region randomly (denoted as $x_{i_1}, x_{i_2}, \cdots, x_{i_M}$) and pool the sparse codes of the selected local patches using average-pooling method (see Eq.2). Then, we get a local descriptor of the object region, $f_i \in R^{(N \times n) \times 1}$. If different local patches are selected, we can obtain different local descriptors. Therefore, the object region can generate m ($m = C_N^M$) local descriptors in total.

$$f_i = \frac{1}{M} \sum_{j=1}^{M} \alpha_{i_j} \tag{2}$$

where α_{i_j} is the sparse code of the local patch x_{i_j}.

In [19], in order to represent target with local sparse codes, authors use average pooling method for all sparse codes of the target to generate a vector. However, the strategy ignores the spatial layout of local patches. In our paper, each local descriptor which is generated by several sparse codes with average pooling method can keep spatial information to some extent.

2.2 Classifier Learning with Local Descriptors

Adaboost classifier is selected as our classifier for making the best of local descriptors. To initialize the classifier, we need to get training sample set S which is composed of N_p positive samples and N_q negative samples from the first n frames. We draw p (p=9) positive samples around the tracking result of each frame via perturbation of a pixel around the target position, then N_p ($N_p = n \times p$) positive samples are obtained. We select N_q negative samples from the n-th frame further away from the target location with a Gaussian perturbation. Using the same way as Sec.2.1, each training sample can generate m local descriptors. After that, our Adaboost classifier is learned with the local descriptors of training sample set.

Some training examples are randomly selected from entire training set S according to their weight. Each selected example has m local descriptors, so we can train m weak classifiers based on these local descriptors of selected examples. Next, the best weak classifier of m weak classifiers is selected (having the lowest classification error) according to the classification error of the weak classifier that is estimated to entire training set S. We denote the best weak classifier as $h_1(x)$. Then, all training samples of S are re-weighted so that samples that are misclassified can get more weight. Carrying out the process repeatedly, we can receive some weak classifiers $h_2(x), \cdots, h_k(x)$. The final strong classifier $H(x)$ is as follow,

$$H(x) = \sum_{i=1}^{k} \rho_i h_i(x), \tag{3}$$

where ρ_i is the weight of $h_i(x)$.

In this paper, we use linear classifier as weak classifier that is calculated by solving the following optimization problem [18],

$$w^* = \arg\min_w \frac{1}{L} \sum_{i=1}^{L} \log\left(1 + e^{-y_i w^T z_i'}\right) + \frac{\eta}{2} \|w\|_2^2, \tag{4}$$

where w is the classifier parameter, L is the number of selected training examples, $z_i' = [z_i^T, 1]^T$ and $z_i \in R^{(N \times n) \times 1}$ is a local descriptor, y_i represents the property of the local descriptor z_i, i.e., $+1$ for positive training example and -1 for negative training example, η is a regularization term.

2.3 Weight Calculation Based on Reconstruction Error

We draw some samples around the target location in the previous frame as the candidate targets of current frame and each candidate target has a classification score with the strong classifier $H(x)$. A simple approach is that the candidate target with the largest score is treated as tracking result of current frame. However, the classification result is not accurate when the target experiences great appearance variations, because the samples used to train classifier are sampled from the previous frames. In order to improve the accuracy of result, we assign

a weight to each candidate target to adjust its classification score. The weight value of a candidate target is calculated based on reconstruction error under the dictionary and reflects the similarity between the candidate target and templates.

From the Sec.2.1, we know that the dictionary can be denoted as

$$D = \left[d_1, \cdots, d_N, d_{N+1}, \cdots, d_{2N}, \cdots, d_{(n-1)N+1}, \cdots, d_{nN}\right] \in R^{d\times(n\times N)}. \quad (5)$$

If the candidate X is perfect, the local image patch x_i in X should be represented well by sub-dictionary $D_i = \left[d_i, d_{N+i}, \cdots, d_{(n-1)N+i}\right] \in R^{d\times n}, 1 \le i \le N$ and the sparse code under D_i is denoted as $\beta^i = \left[\alpha_i^i, \alpha_i^{N+i}, \cdots, \alpha_i^{(n-1)N+i}\right]^T \in R^{n\times 1}$ where α_i^j is the j-th item of α_i. The reconstruction error ε_i of the local image patch x_i under D_i can be calculated by

$$\varepsilon_i = \left\|x_i - D_i\beta^i\right\|_2^2. \quad (6)$$

For purpose of calculating the reconstruction error more conveniently, Eq.6 can be rewritten as follow,

$$\varepsilon_i = \left\|x_i - D\left(\omega_i \otimes \alpha_i\right)\right\|_2^2, \quad (7)$$

where $\alpha_i \in R^{(N\times n)\times 1}$ is the sparse code of x_i under dictionary D, $\otimes$ is the element-wise multiplication,

$$\omega_i = \left[\omega_i^1, \omega_i^2, \cdots, \omega_i^{(N\times n)}\right]^T \in R^{(N\times n)\times 1}, \quad (8)$$

and

$$\omega_i^j = \begin{cases} 1 & , j = i, i+N, \cdots, i+(n-1)N \\ 0 & , others \end{cases}. \quad (9)$$

Our method of calculating reconstruction error is motivated by the paper [6]. The main advantage of the method is that it takes spatial layout between local patches into account. In order to make full use of the sparse code α_i, we add a penalty term $\left\|D\cdot\left((1-\omega_i)\otimes\alpha_i\right)\right\|_1$ to Eq.7, so Eq.7 can be rewritten as follow,

$$\varepsilon_i = \left\|x_i - D\left(\omega_i \otimes \alpha_i\right)\right\|_2^2 + \gamma\left\|D\cdot\left((1-\omega_i)\otimes\alpha_i\right)\right\|_1, \quad (10)$$

where γ controls the strength of the penalty term. If x_i can be represented well by sub-dictionary D_i, the penalty term will be very small, otherwise very large.

After reconstruction errors of all patches in candidate X are obtained, the weight W of X can be calculated by

$$W = \sum_{i=1}^{N} \exp\left(-\beta\varepsilon_i\right), \quad (11)$$

where β is a constant and N is the number of local patches in X.

When partial occlusion happens to target, the occluded patches may have large reconstruction errors, but the other patches still have small reconstruction errors, so the weight of the target still keep a relative big value. If a candidate is bad, its weight is smaller than target because each patch of the bad candidate has large reconstruction error.

2.4 Template and Classifier Update

Templates should be updated dynamically to adapt to appearance changes. In our work, each template T_i has a weight a_i and its initial value is 1. After getting the target of each frame, we update the weight of each template via $a_i = a_i \cdot e^{-\theta}$, where θ is the angle between T_i and target. Template update is carried out every t (t=5) frames and we choose the template with the least weight to be replaced by a new template. The process mentioned above is similar to [5] to some extent.

In [6], authors use sparse representation and incremental subspace learning to reconstruct a new template and then exploit it to replace an old template. It is efficient but it still has a problem. Before a new template is reconstructed, the tracking results are employed to incrementally update the eigenbasis vectors. If noise or occlusion exists in tracking results, the updated eigenbasis vectors will degenerate gradually. Therefore, the occlusion in tracking results should be handled firstly. In this paper, we use the same way as [9] to handle the occlusion in tracking results.

After getting the target of each frame, we reconstruct the target by

$$[\hat{z}, \hat{s}] = \arg\min_{z,s} \frac{1}{2} \|\bar{y} - Uz - s\|_2^2 + \lambda_1 \|s\|_1, \tag{12}$$

where $\bar{y} = y - \mu$, y represents the observation vector, U is composed of PCA basis vectors, μ is the mean vector, z denotes the coefficients of $\bar{y}$ under U and s is the noise term. Then y is reconstructed by

$$y_r^i = \begin{cases} y^i \ , s^i = 0 \\ \mu^i \ , s^i \neq 0 \end{cases} , \tag{13}$$

where y_r^i denotes the i-th item of y_r which is the reconstructed observation vector. The reconstructed observation vector is collected and then we incrementally update U and μ.

When template needs to be updated (every t frames), we firstly compute the coefficient $\hat{z}$ of current observation vector by Eq.12, then reconstruct a new template by

$$T^* = U\hat{z} + \mu, \tag{14}$$

where T^* is the new template used for updating the template with the least weight.

We draw p (p=9) positive samples around the tracking result of each frame via perturbation of a pixel, then replace some old positive samples, and update the negative samples every t frames via drawing some samples further away from the target location. The Adaboost classifier is then retrained by the updated training sample set.

2.5 Object Tracking by Particle Filter

Our tracking algorithm is implemented based on particle filter framework. Let x_t represents the target state variable and $z_{1:t} = \{z_1, z_2, \cdots, z_t\}$ denotes the

observations up to time t. x_t can be estimated by $\hat{x}_t = \arg\max_{x_t} p(x_t|z_{1:t})$, where $p(x_t|z_{1:t})$ is the posterior probability and can be computed by Bayesian theorem,

$$p(x_t|z_{1:t}) \propto p(z_t|x_t) \int p(x_t|x_{t-1})\, p(x_{t-1}|z_{1:t-1}) dx_{t-1}, \tag{15}$$

where $p(x_t|x_{t-1})$ is a dynamic model and $p(z_t|x_t)$ is an observation model. In our algorithm, the target motion is modeled by the affine transformation with six parameters. We apply a Gaussian distribution $p(x_t|x_{t-1}) = \mathrm{N}(x_t; x_{t-1}, \Sigma)$ to represent the dynamic model, where Σ is a diagonal covariance matrix. The observation model is constructed by

$$p(z_t|x_t) \propto W \cdot H(x), \tag{16}$$

where $H(x)$ is the classification score of a candidate and W is the weight.

3 Experimental Results

Our tracking algorithm is tested on 51 challenging videos provided with the recent benchmark [3], and compared with 12 state-of-the-art trackers which show the best performance on the benchmark. The trackers used for comparison are: Struck [11], SCM [15], TLD [12], ASLA [6], CXT [20], VTD [21], VTS [22], CSK [23], LSK [19], DFT [24], LOT [25], OAB [26]. For convenience, we directly use the results of these trackers provided with [3] to conduct comparative experiments with our results.

We use the precision plot and success plot [3] to measure the overall performance. The precision plot indicates the percentage of frames whose center location error (the distance between center location of tracking result and that of ground truth) is less than a given threshold distance. The precision score of each tracker is represented with the score under the threshold = 20 pixels. The success plot demonstrates the radios of successful frames whose bounding box overlap is larger than a given threshold. The AUC (area under curve) of each success plot is used to measure the trackers.

3.1 Experiments Setup

The proposed algorithm is implemented in MATLAB R2012b and runs at 1.1 frames per second on an Intel Core i7 3.4GHz with 4G memory. The number of templates is 8, training samples and candidate targets are all normalized to 32×32 pixels, and then 9 overlapped 16×16 local patches are extracted within the region with 8 pixels as step length. When representing the local descriptors with local sparse codes, we select 3 local sparse codes from 9 local sparse codes to carry out average pooling, then can get 84 local descriptors. For learning classifier, we collect $N_p = 72$ (9 positive samples per frame and 8 consecutive frames) positive samples and $N_q = 150$ negative samples. We select 2/3 samples randomly from all training samples to get 84 weak classifiers, and then select

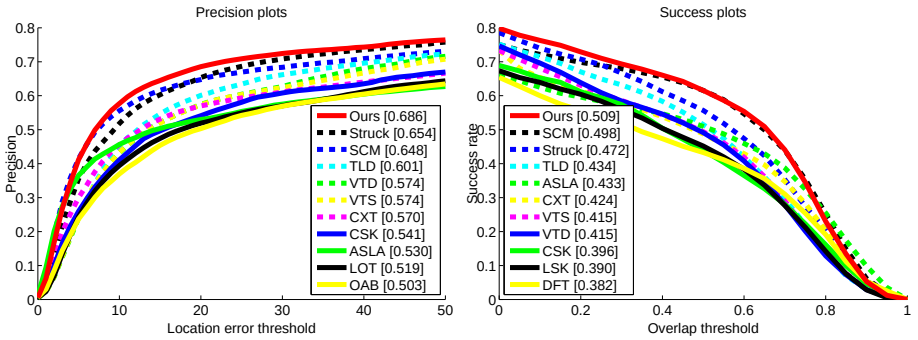

Fig. 1. Precision plots and success plots over all 51 video sequences. The legends in left Fig and right Fig shows the precision scores and AUC scores for each tracker, respectively.

the best weak classifier. After repeat 100 times, we obtain 100 candidate weak classifiers and the number of chosen weak classifier is set to 45. The templates and the Adaboost classifier are updated every 5 frames.

The other parameters are set as follows. The variable λ in Eq.1, η in Eq.4, γ in Eq.10, β in Eq.11 and λ_1 in Eq.12 are set to 0.01, 0.1, 0.01, 5 and 0.1 respectively. The affine transformation with six parameters is fixed to [8, 8, 0.005, 0, 0, 0]. The number of particles is set to 600. All the parameters mentioned in this section are fixed for all sequences.

3.2 Overall Performance

Fig.1 shows the precision plots and success plots which illustrate the overall performance of our tracker and the competing trackers on 51 videos. For precision plots, we rank the trackers according to the result at error threshold of 20 pixels. For success plots, the trackers are ranked as the AUC scores. The precision scores and AUC scores for each tracker are shown in the legend of Fig.1. Only the top 10 of the competing trackers and our tracker are displayed for clarity.

From Fig.1, we can see that our tracker, Struck and SCM perform well, but our tracker achieves the best performance. In precision plot, our algorithm performs 3.2% better than Struck, 3.8% better than SCM. When the error threshold is reduced to 10 pixels, the SCM performs better than Struck but our method still performs best. If the error threshold is set to 5 pixels, our tracker and the SCM perform favorably compared to other trackers. In success plot, our tracker outperforms SCM by 1.1% and Struck by 3.7%. When given a specific overlap threshold (e.g. 0.5), our method still achieves the best performance. We can also observe that SCM achieves higher precision when the error threshold is relatively small and higher success rate when the overlap threshold is relatively large. This is because SCM integrates holistic templates and local representations based on sparse code to handle appearance variations. Struck achieves higher precision

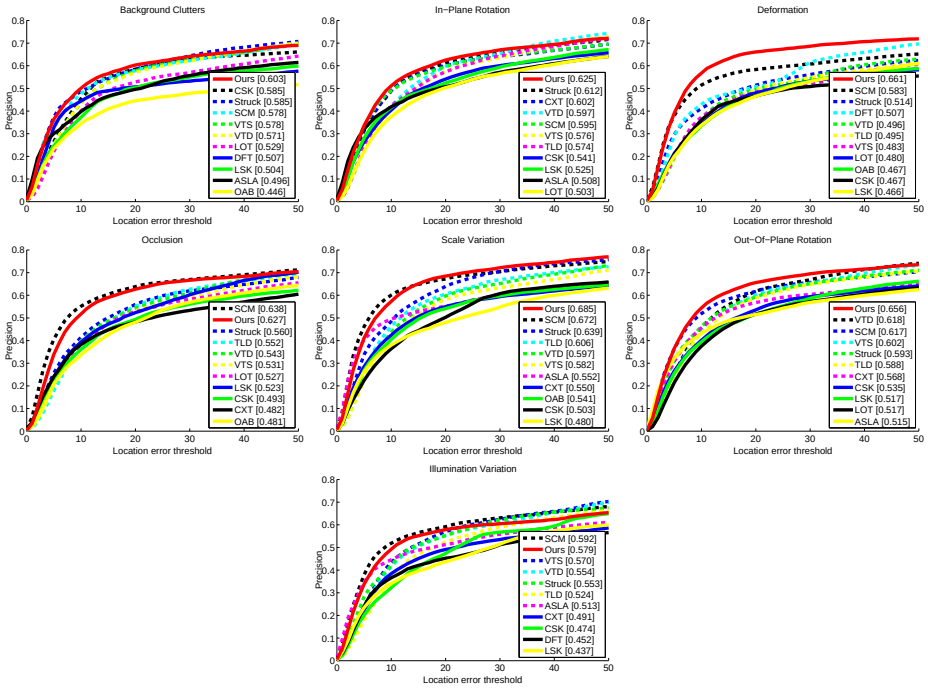

Fig. 2. Attribute based performance analysis using precision plots. These attributes are: background clutters, in-plane rotation, deformation, occlusion, scale variation, out-of-plain rotation, illumination variation.

scores than SCM, but lower AUC scores than SCM. The main reason is that Struck only predicts the location of target and ignores scale variation.

Overall, our tracker performs favorably compared to other trackers. The main reasons are explained as follows. First, the proposed method can generate some structural local descriptors for target which possess good discrimination. Even if the target is partial occluded or contaminated, some local descriptors generated by the parts which are not contaminated still possess good discrimination. Our method can select some discriminative local descriptors to train classifier, which ensures the accuracy of the classifier. Second, the weight value is calculated by structural reconstruction error of the candidate target to adjust the classification score, and a small weight value is assigned to the bad candidate. Therefore, the weight model improves the robustness of our tracker. Third, the update scheme doesn't introduce heavy occlusion and alleviates the drift problem to some extent.

3.3 Attribute based Performance Analysis

The performance of a tracker is affected by many factors which can be divided into 11 attributes [3]. The 51 videos are annotated with the 11 different at-

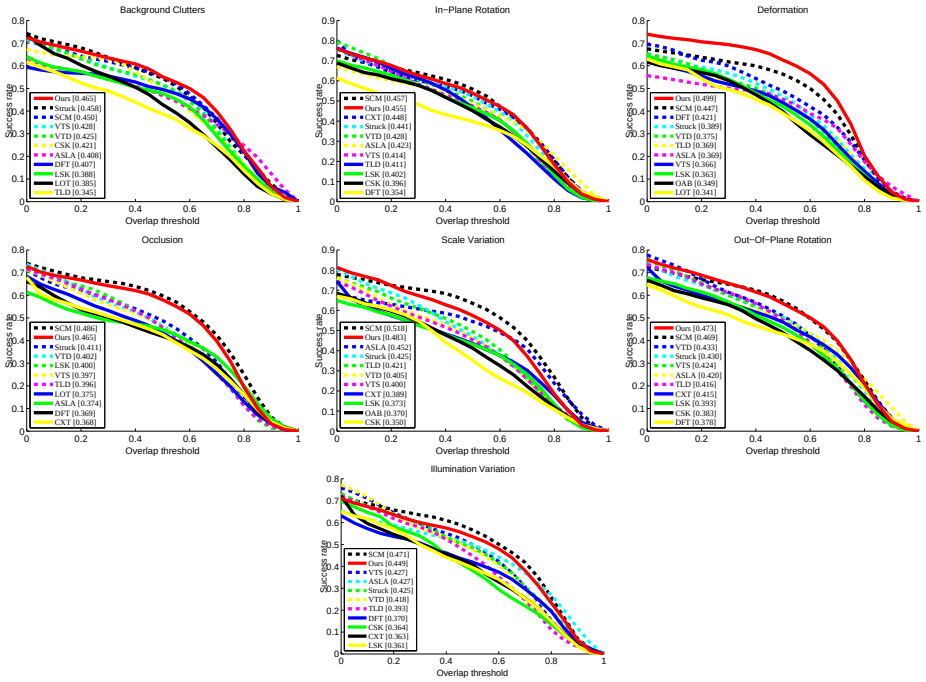

Fig. 3. Attribute based performance analysis using success plots. These attributes are: background clutters, in-plane rotation, deformation, occlusion, scale variation, out-of-plain rotation, illumination variation.

tributes and one sequence may be annotated with several attributes, then we can construct 11 subsets based on these attributes. Each subset can be utilized to evaluate the performance of trackers to deal with a specific challenging factor.

We compare our tracker with other methods on the 51 video sequences with respect to the 11 attributes mentioned above. Our tracker performs well in 7 of the 11 video subsets: background clutters, in-plane rotation, deformation, occlusion, scale variation, out-of-plain rotation and illumination variation. Fig.2 and Fig.3 show the precision plots and success plots of our tracker and the competing trackers in these 7 attributes, respectively. These results show that our tracker is robust to appearance changes of a target object caused by some factors. At present, the proposed algorithm can't handle motion blur well.

On the occlusion subset, the SCM and our method perform favorably compared to other trackers. Our some structural local descriptors which are generated by the local patches that are not occluded still possess good discrimination; therefore, our tracker can avoid much influence of occlusion. Meanwhile, the weight model focuses more on the uncontaminated local patches, which makes our tracker more accurate. On the background clutters subset, the SCM, Struck and our method provide much better results. The reason that our method performs well is that our tracker considers the background information and selects

some discriminative local descriptors to train classifier, which ensures the accuracy of the classifier. On the illumination variation subset, the SCM and our method perform much better than others. The reason is that the template update strategy, based on incremental PCA and occlusion handing scheme, is able to capture the appearance change due to illumination variation and alleviate the visual drift problem. On the deformation subset, our method provides superior results than others. It may be due to the proposed structural local descriptors that are robust to the deformation of target. However, for the blurry target, the proposed method may fail. The main reason is that the blurry local patches can't be well represented by the dictionary. Therefore, effectiveness of the sparse codes is restricted and the pooled local descriptors may lose discrimination.

4 Conclusion

In this paper, we employ sparse codes of local patches to generate local descriptors of object and then an Adaboost classifier is learned with the local descriptors of training sample set. The classifier is applied to candidate targets to separate the object from its surrounding background. In order to adapt the classifier to appearance change of the target, we assign a weight to each candidate target to adjust its classification score. The weight is computed based on reconstruction error under generative model and it reflects the similarity between the candidate target and templates. In addition, a robust template update scheme is applied. Comparison with the state-of-the-art trackers on the comprehensive benchmark shows effectiveness of the proposed method.

Acknowledgement. This work is supported in part by the National Natural Science Foundation of China (No. 61472036) and the Major State Basic Research Development Program of China (No. 2012CB720003).

References

1. Yilmaz, A., Javed, O., Shah, M.: Object tracking: A survey. ACM Computing Surveys **38** (2006)
2. Li, X., Hu, W., Shen, C., Zhang, Z., Dick, A., Hengel, A.: A survey of appearance models in visual object tracking. ACM Transactions on Intelligent Systems and Technology **4** (2013)
3. Wu, Y., Lim, J., Yang, M.: Online object tracking: A benchmark. In CVPR. (2013) 2411–2418
4. Ross, D., Lim, J., Lin, R., Yang, M.: Incremental learning for robust visual tracking. IJCV **77** (2008) 125–141
5. Mei, X., Ling, H.: Robust visual tracking using l1 minimization. In ICCV. (2009) 1–8
6. Jia, X., Lu, H., Yang, M.: Visual tracking via adaptive structural local sparse appearance model. In CVPR. (2012) 1822–1829
7. Zhang, T., Ghanem, B., Liu, S., Ahuja, N.: Robust visual tracking via multi-task sparse learning. In CVPR. (2012) 2042–2049

8. Wang, N., Wang, J., Yeung, D.: Online robust non-negative dictionary learning for visual tracking. In ICCV. (2013) 657–664
9. Wang, D., Lu, H., Yang, M.: Least soft-thresold squares tracking. In CVPR. (2013) 2371–2378
10. Babenko, B., Yang, M., Belongie, S.: Robust object tracking with online multiple instance learning. PAMI **33** (2011) 1619–1632
11. Hare, S., Saffari, A., Torr, P.H.: Struck: Structured output tracking with kernels. In ICCV. (2011) 263–270
12. Kalal, Z., Matas, J., Mikolajczyk, K.: P-n learning: Bootstrapping binary classifiers by structural constraints. In CVPR. (2010) 49–56
13. Zhang, K., Zhang, L., Yang, M.: Real-time compressive tracking. In ECCV. (2012) 864–877
14. Yao, R., Shi, Q., Shen, C., Zhang, Y., A.Hengel: Part-based visual tracking with online latent structural learning. In CVPR. (2013) 2363–2370
15. Zhong, W., Lu, H., Yang, M.: Robust object tracking via sparsity-based collaborative model. In CVPR. (2012) 1838–1845
16. Dinh, T.B., Medioni, G.G.: Co-training framework of generative and discriminative trackers with partial occlusion handling. In WACV. (2011) 642–649
17. Liu, R., Cheng, J., Lu, H.: A robust boosting tracker with minimum error bound in a co-training framework. In ICCV. (2009) 1459–1466
18. Wang, Q., Chen, F., Xu, W., Yang, M.: Online discriminative object tracking with local sparse representation. In WACV. (2012) 425–432
19. Liu, B., Huang, J., Yang, L., Kulikowsk, C.: Robust tracking using local sparse appearance model and k-selection. In CVPR. (2011) 1313–1320
20. Dinh, T.B., Vo, N., Medioni, G.: Context tracker: Exploring supporters and distracters in unconstrained environments. In CVPR. (2011) 1177–1184
21. Kwon, J., Lee, K.: Visual tracking decomposition. In CVPR. (2010) 1269–1276
22. Kwon, J., Lee, K.: Tracking by sampling trackers. In ICCV. (2011) 1195–1202
23. Henriques, J., Caseiro, R., Martins, P., Batista, J.: Exploiting the circulant structure of tracking-by-detection with kernels. In ECCV. (2012) 702–715
24. Sevilla-Lara, L., Learned-Miller, E.G.: Distribution fields for tracking. In CVPR. (2012) 1910–1917
25. Oron, S., Bar-Hillel, A., Levi, D., Avidan, S.: Locally orderless tracking. In CVPR. (2012) 1940–1947
26. Grabner, H., Grabner, M., Bischof, H.: Real-time tracking via on-line boosting. In BMVC. (2006)