Learning Compact Binary Codes for Visual Tracking

Xi Li, Chunhua Shen, Anthony Dick, Anton van den Hengel
Australian Centre for Visual Technologies, The University of Adelaide, SA 5005, Australia

Abstract

A key problem in visual tracking is to represent the appearance of an object in a way that is robust to visual changes. To attain this robustness, increasingly complex models are used to capture appearance variations. However, such models can be difficult to maintain accurately and efficiently. In this paper, we propose a visual tracker in which objects are represented by compact and discriminative binary codes. This representation can be processed very efficiently, and is capable of effectively fusing information from multiple cues. An incremental discriminative learner is then used to construct an appearance model that optimally separates the object from its surrounds. Furthermore, we design a hypergraph propagation method to capture the contextual information on samples, which further improves the tracking accuracy. Experimental results on challenging videos demonstrate the effectiveness and robustness of the proposed tracker.

1. Introduction

As a fundamental problem in computer vision, visual tracking underpins a wide range of applications such as visual surveillance, human-computer interaction, object recognition, event detection, and action recognition. Most state of the art trackers use a sampling approach, in which the object location is selected from a pool of candidate samples at each frame. This provides robustness to unpredicted or ambiguous motion but leads to the question of how these samples are evaluated or scored. Ideally, the highest scoring sample should be the one which best aligns with the object, and sample scores should decrease with the amount of object overlap, while all background samples should be scored lower than any sample containing at least part of the object. This is made more challenging as the scoring function must be robust to object and background appearance changes, and be computed and updated in real time.

Scoring functions are typically based on a model of object appearance (*e.g.*, linear regression [16, 19, 21, 34], principal component analysis [15, 24], discrete cosine transform [17], random forest [25], support vector machine [3, 11], and boosting [2, 9]), which is in turn based on a robust image feature (*e.g.*, attentional regions [6], covariance features [22, 31], feature learning [10, 33], and multi-feature kernels [11, 20]). Many state of the art trackers construct appearance models from a collection of different feature types to cope with object appearance variations. A basic approach to feature fusion is to directly concatenate weighted, normalized features into a unified feature vector. However, the resulting feature vector is often high-dimensional and redundant, making it difficult to separate object and background samples. To alleviate this issue, dimensionality reduction is generally applied, but may result in the loss of the intrinsic structural information from samples [7] or high computational cost [20, 28].

We propose a hashing method to perform feature fusion by reducing multiple feature descriptors to a single binary code vector. The proposed hash function is based on randomized decision trees, each of which is efficiently built by a sequence of simple operations on samples and their associated features. As a result, the problem of feature fusion is converted to that of randomized decision tree growing. Using the learned hash function, we can explicitly formulate the binary code corresponding to a sample by aggregating the posterior distributions of the leaf nodes reached in all randomized decision trees. These binary codes capture hierarchical discriminative information from different feature modalities in a decision-tree-growing manner, leading to a discriminative image representation. Since they are individually learned over randomly sampled subsets, the learned hash functions are almost uncorrelated with each other. Because of this, a compact image representation can be achieved. Due to taking binary values as feature elements, our image representation also has low memory usage.

Given the compact and discriminative binary codes representing samples, an object appearance model is typically required to maximize the separation of foreground and background samples. Among existing appearance models, the linear support vector machine (SVM) has proved to be a simple yet effective choice. Compared to the standard linear SVM using the hinge loss, the linear SVM with least square loss (referred to as LS-SVM) can be a better choice for real-time tracking because of its closed-form solution [32]. In terms of online learning, the LS-SVM can be efficiently up-

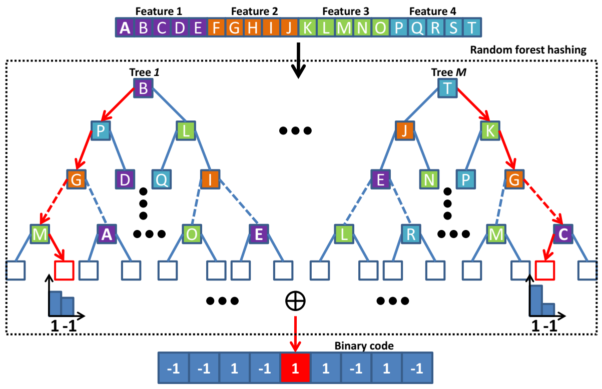

Figure 1: Illustration of our compact binary code learning method for multi-modal feature fusion.

dated by incrementally computing the inverse of its regularized covariance matrix (shown in Sec. 2.2) as new data arrive. Empirical studies [27] have demonstrated that the LS-SVM can achieve comparable generalization performance to the standard linear SVM.

Learning the LS-SVM on the binary codes ensures a max-margin hyperplane separating the foreground from the background. However, due to the precision loss incurred by hashing, the LS-SVM classification scoring function may not be able to accurately localize the object (*i.e.,* determine which of the foreground samples best represents it). To further refine the scoring function, we note that sample confidence scores are not only determined by their own appearance features but also constrained by their contextual dependencies. In other words, if two test image regions have a similar binary code, their confidence scores ought to be close; otherwise, their confidence scores may greatly differ from each other. In order to model such a dependency, we use hypergraph analysis, which is a useful tool for capturing the contextual interaction between graph vertices [13,35]. In hypergraph analysis, the problem of dependency modeling is converted to that of building a set of hyperedges, which correspond to vertex communities. In each vertex community, the vertices have some common properties (*e.g.,* the same weak labels obtained by compact binary code learning in our case), and pass support messages to each other. Therefore, the hypergraph propagation method can refine the sample scores to be consistent with their binary codes, leading to more accurate object localization.

In summary, we propose a robust visual tracker that incorporates three measures to improve the accuracy and robustness of the sample scoring function while maintaining its required computational efficiency. The main contributions of this work are as follows.

1. We propose a novel compact binary code learning method based on random forest hashing, which learns to produce compact and discriminative binary codes representing samples. To our knowledge, *it is the first time that the compact binary code learning method is proposed to build a robust image representation for visual tracking.*
2. We build an appearance model based on these binary codes using an incremental closed-form LS-SVM, which can online learn a hyperplane that separates the foreground samples from the background samples.
3. We present a hypergraph propagation method that further refines the appearance model by capturing contextual similarity information from samples. Using such information, the method is able to obtain more accurate object localization.

2. The proposed visual tracker

The workflow of our tracking system is summarized in Algorithm 1. Like most sampling based trackers, at each frame the method generates a sample set, scores each sample, and updates its estimated target location based on the highest scoring sample. In this section, we focus on the construction and update of the sample scoring function. This is based on three techniques: i) compact binary code learning; ii) incremental LS-SVM learning; and iii) hypergraph propagation.

For i), we focus on learning a set of random forest hash functions for feature fusion, as described in Sec. 2.1. For ii), the LS-SVM classifier (with a closed-form solution) is incrementally learned for object/non-object classification by computing Eq. (8) and Eq. (10), as shown in Sec. 2.2. For iii), a weakly supervised hypergraph is created by exploring a set of hashing-bit-specific communities, as shown in Sec. 2.3. According to Eq. (14), hypergraph propagation is performed to diffuse the LS-SVM classification scores on the weakly supervised hypergraph, resulting in more accurate object localization. After object localization, we collect some new foreground and background training samples using a spatial sampling scheme [2, 33]. These training samples are used at regular intervals to update the random forest and and LS-SVM classifier, as explained in Secs. 2.1 and 2.2.

2.1. Compact binary code learning

Given multiple types of visual features, we design a hashing method to form a compact and discriminative fused feature. In principle, the hashing method needs to satisfy the following two conditions: i) each individual hash function is balanced such that:

$$\int_{h(\mathbf{u})=1} \Pr(\mathbf{u})d\mathbf{u} = \int_{h(\mathbf{u})=-1} \Pr(\mathbf{u})d\mathbf{u} = \frac{1}{2}, \qquad (1)$$

where $\mathbf{u}$ is a test sample (represented by a concatenation of different visual features), $\Pr(\cdot)$ is a probability density function, and $h(\cdot)$ is the hash function; ii) the hash functions are mutually independent. To achieve these two goals,

Algorithm 1 Compact binary code learning based tracker

```
Input: New video frame.
1. Crop out a set of image regions ${\mathbf{u}_i}_{i=1}^K$ using the sliding-window-sampling scheme [2,33] and extract their associated visual features;
2. Compute the corresponding binary codes ${\mathbf{x}_i}_{i=1}^K$ by compact binary code learning (Sec. 2.1);
3. Perform LS-SVM classification on the binary codes to produce the initial confidence score vector $\mathbf{s}^0$ (Sec. 2.2);
4. Perform hypergraph propagation (Sec. 2.3) to obtain the final confidence score vector $\mathbf{s}$;
5. Update the tracker location to $\{\mathbf{u}_k | k = \arg\max_i s_i\}$.
6. Add new foreground and background samples to sample buffer. If the buffer limit is exceeded, discard oldest training samples.
7. Update the random forest hash functions (Algorithm 2) and the LS-SVM classifier (Sec. 2.2) based on sample buffer every few frames.
```

Algorithm 2 Learning random forest hash functions

```
Input: Training sample set $\{\mathbf{u}_i, y_i\}_{i=1}^N$ with $y_i \in \{1, -1\}$, binary code length $\ell$, and randomized tree number $M$.
Output: Random forest hash functions $\{h_{\mathbb{T}_j}(\cdot)\}_{j=1}^{\ell}$.
for $j$ = 1 to $\ell$ do
    for $m$ = 1 to $M$ do
        • Randomly sample $\{\mathbf{u}_i, y_i\}_{i=1}^N$ to generate a training sample subset $\{\mathbf{u}_{j_k}, y_{j_k}\}_{k=1}^F$ with equal representation from positive and negative samples;
        • Use $\{\mathbf{u}_{j_k}, y_{j_k}\}_{k=1}^F$ to construct a randomized tree $t_m$ by random internal node split (Eq. (3)).
    end
    • Obtain random forest $\mathbb{T}_j = \{t_1, \dots, t_M\}$ and output the hash function $h_{\mathbb{T}_j}(\cdot)$ by Eq. (4).
end
```

the training samples for each hash function are randomly selected from the entire training dataset, and equally distributed between positive and negative samples. To capture the discriminative information from inter-class samples, the hash function is typically formulated as a binary classifier:

$$h(\mathbf{u}) = \mathrm{sgn}\left(\kappa(\mathbf{u}, \{\mathbf{u}_i^+\}) - \kappa(\mathbf{u}, \{\mathbf{u}_j^-\})\right), \qquad (2)$$

where sgn$(\cdot)$ is the sign function, $\kappa(\cdot,\cdot)$ is a similarity function, $\{\mathbf{u}_i^+\}$ and $\{\mathbf{u}_j^-\}$ are respectively the positive and negative training samples. Now, the remaining problem is how to design an efficient and discriminative similarity function κ. We make use of randomized trees to construct the similarity function due to their effectiveness and efficiency in discriminative learning. The process of growing each randomized tree enables our hashing method to effectively combine the discriminative information from different feature modalities in a top-down manner, as shown in Fig. 1. Each internal node of these randomized trees contains a binary test that best splits the space of a randomly sampled subset of training data along a randomly chosen feature dimension:

$$\mathrm{sp}(u_i) = \begin{cases} \text{left child}, & \text{if } u_i \geq \gamma, \\ \text{right child}, & \text{otherwise}, \end{cases} \qquad (3)$$

where u_i corresponds to the i-th feature dimension (chosen at random) of $\mathbf{u}$ and γ is a threshold determined by optimizing an entropy-based information gain criterion [4]. Such random sample selection and random internal node split ensure the high efficiency of our hashing method. Using the aforementioned tree growing scheme (3), we construct a random forest $\mathbb{T}$ comprising a set of randomized trees. For descriptive convenience, let $\mathbb{C} \in \{1, -1\}$ be the set of all classes and $\mathbb{L}_t$ be the set of all leaves for a given randomized tree $t \in \mathbb{T}$. The posterior probability $\mathrm{Pr}_{t,l}(c)$ for each class $c \in \mathbb{C}$ at each leaf node $l \in \mathbb{L}_t$ needs to be learned during the training stage. Mathematically, $\mathrm{Pr}_{t,l}(c)$ is calculated as the ratio $\frac{|\mathbb{Q}_{t,l,c}|}{|\mathbb{Q}_{t,l}|}$, where $\mathbb{Q}_{t,l}$ is the set of training samples reaching the leaf node l in the randomized tree t, and $\mathbb{Q}_{t,l,c}$ is the set of class-c training samples in $\mathbb{Q}_{t,l}$.

Based on the random forest $\mathbb{T}$, we define the similarity function (Eq. (2)) as $\kappa(\mathbf{u}, \{\mathbb{Q}_{t,l_{\mathbf{u}}^t,c} | t \in \mathbb{T}\}) = \sum_{t\in\mathbb{T}} \mathrm{Pr}_{t,l_{\mathbf{u}}^t}(c)$, where $l_{\mathbf{u}}^t$ denotes the leaf node reached by the test sample $\mathbf{u}$ in the randomized tree t. For each test sample $\mathbf{u}$, its corresponding binary code $h_{\mathbb{T}}(\mathbf{u}) \in \{1, -1\}$ w.r.t. the random forest $\mathbb{T}$ is obtained by the following three steps. First, pass the test sample down each randomized tree until reaching a leaf node; second, aggregate all the corresponding posterior probabilities of the reached leaf nodes from $\mathbb{T}$; finally, output the binary code such that

$$h_{\mathbb{T}}(\mathbf{u}) = \mathrm{sgn}(\sum_{t\in\mathbb{T}} \mathrm{Pr}_{t,l_{\mathbf{u}}^t}(c=1) - \sum_{t\in\mathbb{T}} \mathrm{Pr}_{t,l_{\mathbf{u}}^t}(c=-1)). \quad (4)$$

Algorithm 2 shows the detailed workflow of learning the random forest hash functions. Suppose that there are ℓ learned random forests (denoted as $\{\mathbb{T}_j\}_{j=1}^{\ell}$) corresponding to ℓ hash functions in our hashing method. As a result, for any test sample $\mathbf{u}$, we have an associated ℓ-dimensional binary feature vector denoted as $\mathbf{x} = (h_{\mathbb{T}_1}(\mathbf{u}), h_{\mathbb{T}_2}(\mathbf{u}), \dots, h_{\mathbb{T}_\ell}(\mathbf{u}))^\top$.

2.2. Incremental LS-SVM

To classify binary features as object/non-object, we build an online discriminative appearance model based on incremental LS-SVM learning. Given a set of training samples $\{\mathbf{x}_i, y_i\}_{i=1}^N$ with $\mathbf{x}_i \in \mathcal{R}^\ell$ and $y_i \in \{-1, +1\}$, the LS-SVM optimizes the following objective function [32]:

$$\min_{\mathbf{w},\mathbf{b}} \sum_{i=1}^{N} \|f(\mathbf{x}_i) - y_i\|_2^2 + C\|\mathbf{w}\|_2^2, \qquad (5)$$

where $\|\cdot\|_2$ is the L2 norm, $f(\mathbf{x}) = \mathbf{w}^\top\mathbf{x} + \mathbf{b}$ is the classifier to learn, and C is the trade-off control parameter. For convenience, let $\mathbf{1}$ be an all-one vector, $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$ be the data matrix, N_+ (N_-) be the positive (negative) sample size such that $N_+ + N_- = N$, $\boldsymbol{\mu}_+$ ($\boldsymbol{\mu}_-$) be the sample mean of the foreground (background) class, and $\boldsymbol{\mu}$ be the mean of all the training samples such that $\boldsymbol{\mu} = \frac{N_+}{N}\boldsymbol{\mu}_+ + \frac{N_-}{N}\boldsymbol{\mu}_-$. Then, the closed-form solution to (5) is formulated as:

$$\begin{array}{l}\mathbf{w} = \frac{2N_+N_-}{N^2}\left(\mathbf{S} + \frac{C}{N}\mathbf{I}\right)^{-1}(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-),\\ \mathbf{b} = \frac{N_+ - N_-}{N} - \boldsymbol{\mu}^\top \mathbf{w},\end{array} \tag{6}$$

where $\mathbf{I}$ is an identity matrix and $\mathbf{S}$ is the covariance matrix defined as: $\mathbf{S} = \frac{1}{N}(\mathbf{X} - \boldsymbol{\mu}\mathbf{1}^\top)(\mathbf{X} - \boldsymbol{\mu}\mathbf{1}^\top)^\top$. In terms of online learning, the key factor is the incremental computation of $\left(\mathbf{S} + \frac{C}{N}\mathbf{I}\right)^{-1}$ with respect to new samples $\delta\mathbf{X} = (\delta\mathbf{x}_1, \delta\mathbf{x}_2, \ldots, \delta\mathbf{x}_{\delta N})$. Let $\delta\boldsymbol{\mu}$ be the sample mean of $\delta\mathbf{X}$, N' be the total number of samples such that $N' = N + \delta N$, and $\boldsymbol{\mu}'$ be the sample mean of $(\mathbf{X}, \delta\mathbf{X})$ such that $\boldsymbol{\mu}' = \frac{N}{N'}\boldsymbol{\mu} + \frac{\delta N}{N'}\delta\boldsymbol{\mu}$. Then, the updated covariance matrix S' can be calculated as:

$$\begin{array}{l}\mathbf{S}' = \frac{N}{N'}\mathbf{S} + \frac{1}{N'}[\sum_{i=1}^{\delta N}(\delta\mathbf{x}_i - \delta\boldsymbol{\mu})(\delta\mathbf{x}_i - \delta\boldsymbol{\mu})^\top \\ \quad + N(\boldsymbol{\mu} - \boldsymbol{\mu}')(\boldsymbol{\mu} - \boldsymbol{\mu}')^\top + \delta N(\delta\boldsymbol{\mu} - \boldsymbol{\mu}')(\delta\boldsymbol{\mu} - \boldsymbol{\mu}')^\top].\end{array} \tag{7}$$

Defining $\mathbf{A}$ as $\frac{1}{N'}[\sum_{i=1}^{\delta N}(\delta\mathbf{x}_i - \delta\boldsymbol{\mu})(\delta\mathbf{x}_i - \delta\boldsymbol{\mu})^\top + N(\boldsymbol{\mu} - \boldsymbol{\mu}')(\boldsymbol{\mu} - \boldsymbol{\mu}')^\top + \delta N(\delta\boldsymbol{\mu} - \boldsymbol{\mu}')(\delta\boldsymbol{\mu} - \boldsymbol{\mu}')^\top]$, we obtain the following relation:

$$\mathbf{S}' + \frac{C}{N'}\mathbf{I} = \frac{N}{N'}(\mathbf{S} + \frac{C}{N}\mathbf{I} + \frac{N'}{N}\mathbf{A}). \tag{8}$$

As a result, we have $(\mathbf{S}' + \frac{C}{N'}\mathbf{I})^{-1} = \frac{N'}{N}(\mathbf{S} + \frac{C}{N}\mathbf{I} + \frac{N'}{N}\mathbf{A})^{-1}$. Therefore, the key factor of efficiently computing $(\mathbf{S}' + \frac{C}{N'}\mathbf{I})^{-1}$ is to incrementally update $(\mathbf{S} + \frac{C}{N}\mathbf{I} + \frac{N'}{N}\mathbf{A})^{-1}$ when $(\mathbf{S} + \frac{C}{N}\mathbf{I})^{-1}$ is given. Since $\frac{N'}{N}\mathbf{A}$ is formed by the sum of rank-one matrices, it can be decomposed as: $\frac{N'}{N}\mathbf{A} = \sum_{k=0}^{\delta N+2}\mathbf{q}_k\mathbf{q}_k^\top$ such that

$$\mathbf{q}_k = \begin{cases} \mathbf{0}, & k = 0, \\ \frac{1}{\sqrt{N}}(\delta\mathbf{x}_k - \delta\boldsymbol{\mu}), & 1 \le k \le \delta N, \\ \boldsymbol{\mu} - \boldsymbol{\mu}', & k = \delta N + 1, \\ \sqrt{\frac{\delta N}{N}}(\delta\boldsymbol{\mu} - \boldsymbol{\mu}'), & k = \delta N + 2. \end{cases} \tag{9}$$

According to the theory of [12,23], $(\mathbf{S} + \frac{C}{N}\mathbf{I} + \frac{N'}{N}\mathbf{A})^{-1}$ can be recursively computed by:

$$(\mathbf{J}_m + \mathbf{q}_{m+1}\mathbf{q}_{m+1}^\top)^{-1} = \mathbf{J}_m^{-1} - \frac{\mathbf{J}_m^{-1}\mathbf{q}_{m+1}\mathbf{q}_{m+1}^\top\mathbf{J}_m^{-1}}{1 + \mathbf{q}_{m+1}^\top\mathbf{J}_m^{-1}\mathbf{q}_{m+1}}, \tag{10}$$

where $\mathbf{J}_m = \mathbf{S} + \frac{C}{N}\mathbf{I} + \sum_{k=0}^{m}\mathbf{q}_k\mathbf{q}_k^\top$ and $0 \le m \le \delta N + 1$.

2.3. Hypergraph propagation

The goal of hypergraph propagation is to refine the confidence score s_i^0 obtained from the LS-SVM for each sample, taking into account contextual information from surrounding samples. This is necessary because the score is based on binary codes which have sufficient precision to distinguish foreground from background, but not to locate the object reliably among foreground samples.

Given a set of samples $\{\mathbf{u}_i\}_{i=1}^K$, we have computed the corresponding binary codes $\{\mathbf{x}_i\}_{i=1}^K$ such that $\mathbf{x}_i = (h_{\mathbb{T}_1}(\mathbf{u}_i), \ldots, h_{\mathbb{T}_\ell}(\mathbf{u}_i))^\top$. Based on $\{\mathbf{x}_i\}_{i=1}^K$, we create a weakly supervised hypergraph $G = (\mathbb{V}, \mathbb{E}, w)$, where $\mathbb{V} = \{v_i\}_{i=1}^K$ is the vertex set corresponding to $\{\mathbf{u}_i\}_{i=1}^K$, $\mathbb{E}$ is the hyperedge set comprising a family of subsets of $\mathbb{V}$ such that $\bigcup_{e\in\mathbb{E}} = \mathbb{V}$, and w is the hyperedge weight [13, 35]. These hyperedges are obtained by using the bit-specific binary code to weakly classify $\{\mathbf{u}_i\}_{i=1}^K$ into bit-specific positive and negative communities. If $h_{\mathbb{T}_k}(\mathbf{u}_i) = 1$, $\mathbf{u}_i$ is added to the bit-k positive community; otherwise, it belongs to the bit-k negative community. In other words, each bit-specific community corresponds to a hyperedge.

Algorithm 3 Hypergraph propagation

```
Input: Binary codes $\{\mathbf{x}_i\}_{i=1}^K$ and maximum iteration number $\tau$.
Output: Confidence score vector $\mathbf{s}$ for object localization.
• $n \leftarrow 0$;
• Compute the LS-SVM classification score vector $\mathbf{s}^0$ for $\{\mathbf{x}_i\}_{i=1}^K$;
repeat
    • Construct the hypergraph $G = (\mathbb{V}, \mathbb{E}, w)$ in Sec. 2.3;
    • Calculate the transition probability matrix $\mathbf{P}$ in Eq. (13);
    • Perform the hypergraph propagation procedure in Eq. (14);
    • $n \leftarrow n + 1$;
until $\mathbf{s}^n$ converges or $n \ge \tau$;
• $\mathbf{s} \leftarrow \mathbf{s}^n$.
```

The hypergraph G is represented by a $|\mathbb{V}| \times |\mathbb{E}|$ incidence matrix $\mathbf{H} = (H(v_i, e_j))_{|\mathbb{V}|\times|\mathbb{E}|}$:

$$H(v_i, e_j) = \begin{cases} 1, & \text{if } v_i \in e_j, \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

The degree of any vertex $v \in \mathbb{V}$ is defined as $d(v) = \sum_{e\in\mathbb{E}} w(e)H(v, e)$ where $w(e)$ is a positive weight ($w(e) = 1$ in our case). Correspondingly, the degree of any hyperedge $e \in \mathbb{E}$ is defined as $\delta(e) = \sum_{v\in\mathbb{V}} H(v, e)$. Let $\mathbf{W}$, $\mathbf{D}_v$, and $\mathbf{D}_e$ denote the diagonal matrices whose diagonal elements are associated with $w(e)$, $d(v)$, and $\delta(e)$, respectively. The process of random walk on the hypergraph G [35] is governed by the following transition probability matrix $\mathbf{P} = (p_{ij})_{K\times K}$ whose entry is defined as:

$$p_{ij} = \sum_{e\in\mathbb{E}} w(e)\frac{H(v_i, e)}{d(v_i)}\frac{H(v_j, e)}{\delta(e)}. \tag{12}$$

Its corresponding matrix form can be written as:

$$\mathbf{P} = \mathbf{D}_v^{-1}\mathbf{H}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{H}^\top. \tag{13}$$

Based on this transition probability matrix, the process of hypergraph propagation is formulated as follows:

$$\mathbf{s}^{n+1} = \alpha\mathbf{P}\mathbf{s}^n + (1 - \alpha)\mathbf{s}^0, \tag{14}$$

where α is a trade-off control factor such that $0 < \alpha < 1$, $\mathbf{s}^0 = (s_1^0, s_2^0, \ldots, s_K^0)^\top$ is the initial LS-SVM confidence score vector (*i.e.,* $s_i^0 = f(\mathbf{x}_i)$) and $\mathbf{s}^n$ is the confidence score vector after propagation at the n-th iteration. Through a sequence of transformations, Eq. (14) is further converted to:

$$\mathbf{s}^n = (\alpha\mathbf{P})^n\mathbf{s}^0 + (1 - \alpha)\sum_{j=0}^{n-1}(\alpha\mathbf{P})^j\mathbf{s}^0. \tag{15}$$

Since $p_{ij} \ge 0$ and $\sum_j p_{ij} = 1$, the spectral radius of $\mathbf{P}$ is not greater than one (see the theorem of Perron-Frobenius [8]), leading to the fact that $\lim_{n\to\infty}(\alpha\mathbf{P})^n = 0$

and $\lim_{n\to\infty}(1-\alpha)\sum_{j=0}^{n-1}(\alpha\mathbf{P})^j\mathbf{s}^0 = (1-\alpha)(\mathbf{I}-\alpha\mathbf{P})^{-1}\mathbf{s}^0$. Therefore, Eq. (14) will converge to $(1-\alpha)(\mathbf{I}-\alpha\mathbf{P})^{-1}\mathbf{s}^0$ after a sufficient number of iterations. When K is large, the cost of computing $(\mathbf{I}-\alpha\mathbf{P})^{-1}$ is high. For computational efficiency, the final confidence score vector $\mathbf{s} = (s_1, s_2, \ldots, s_K)^\top$ is obtained by iterating Eq. (14) until convergence. The complete procedure of hypergraph propagation is summarized in Algorithm 3.

3. Experiments

3.1. Experimental setup

In the experiments, eighteen publicly available video sequences are used for tracking performance evaluation. Captured in different scenarios, these video sequences contain diverse events such as occlusion, object pose variation, lighting changes and out-of-plane rotation. Like the multi-instance tracker [2], the proposed tracker performs object localization using a sliding-window-search scheme with a search radius of 30 pixels. The average running time of our Matlab implementation is about 0.1 second per frame on a workstation with an Intel Core 2 Duo 2.66GHz processor and 3.24G RAM. For each image region, we extract four types of visual features: intensity histogram, local binary pattern (LBP), histogram of gradient (HOG), and Haar-like wavelets. More specifically, both the intensity histogram and LBP features are extracted by dividing an image region into 4×4 cells, each of which is associated with a 16-dimensional histogram vector. The HOG feature is composed of 3×3 cells, with each cell represented by a 9-dimensional histogram vector, in five spatial block-division modes (like [18]), resulting in a 405-dimensional feature vector. The Haar-like feature is extracted in the same way as the CT tracker [33], resulting in a 100-dimensional feature vector. By concatenating the above four visual features, we have in total a 1017-dimensional real-valued feature vector for each image region. After random forest hashing, the binary code length ℓ in Algorithm 2 is chosen as 100. The training sample sizes N and F in Algorithm 2 are set to 1000 and 100, respectively. Each random forest hash function is associated with a random forest comprising 10 randomized decision trees (*i.e.,* $M = 10$ in Algorithm 2) with 25 layers. The random forest hash functions in Algorithm 2 are updated every 10 frames. The LS-SVM classifier is incrementally updated every 5 frames. The trade-off control factor α in Eq. (14) is set to 0.1. The maximum iteration number τ in Algorithm 3 is chosen as 50. Note that the aforementioned parameters are fixed throughout all the experiments.

For quantitative performance comparison, two popular evaluation criteria are used: center location error (CLE) and VOC overlap ratio (VOR) between the predicted bounding box B_p and ground truth bounding box B_{gt} such that

	Ours	CT	ORF	Struck	Frag	MIT	OAB	IVT	L1T
Cyclist	**4.19**	91.79	34.22	5.22	11.09	93.74	84.04	76.24	65.85
distortion	**2.49**	3.39	5.14	3.07	12.71	3.66	4.82	23.93	5.95
football	**3.36**	11.60	9.48	3.67	16.89	6.59	7.31	42.50	48.63
PedCross	**3.74**	64.54	73.78	138.07	137.73	126.88	69.31	73.65	70.50
animal	**3.18**	10.54	4.06	3.84	50.38	55.60	38.14	5.60	140.54
Jumper	**9.90**	36.24	26.13	35.26	24.74	66.49	36.83	76.49	101.76
Trellis	**3.32**	45.67	36.10	5.67	29.84	60.85	68.72	30.92	89.89
Walker	**5.05**	31.79	53.30	32.27	88.03	32.33	34.05	54.56	89.67
cubicle	**2.94**	17.90	19.09	19.96	30.28	12.06	35.77	49.28	22.73
David	**4.79**	9.51	5.92	5.26	24.70	21.69	25.37	8.44	43.54
Tiger1	6.41	8.27	9.45	**6.35**	33.22	9.20	68.73	38.89	36.58
Tiger2	**6.72**	9.95	26.52	9.18	37.15	8.44	30.25	47.89	32.42
Girl	12.24	29.46	16.13	**10.80**	21.75	35.44	36.95	23.58	18.83
Coke	6.85	22.20	9.93	**5.75**	32.33	18.94	56.55	61.51	62.32
Faceocc1	5.54	19.86	16.22	9.91	**4.63**	19.21	52.63	16.30	6.26
Faceocc2	**6.67**	16.58	14.84	8.64	18.66	17.53	16.48	13.15	37.50
Sylv	**4.89**	6.44	8.85	6.23	20.43	10.70	35.02	48.79	76.88
Surfer	**4.82**	48.88	9.27	7.23	21.27	5.03	31.57	10.09	45.15

Table 1: The quantitative comparison results of the nine trackers over the eighteen video sequences. The table reports their average CLEs over each video sequence.

	Ours	CT	ORF	Struck	Frag	MIT	OAB	IVT	L1T
Cyclist	**0.85**	0.24	0.56	0.82	0.67	0.21	0.22	0.23	0.28
distortion	**0.86**	0.82	0.76	0.84	0.52	0.82	0.75	0.32	0.73
football	**0.80**	0.46	0.67	0.77	0.38	0.64	0.62	0.30	0.11
PedCross	**0.82**	0.37	0.36	0.36	0.30	0.35	0.32	0.35	0.36
animal	**0.92**	0.76	0.89	0.90	0.37	0.40	0.52	0.85	0.07
Jumper	**0.76**	0.43	0.54	0.49	0.56	0.27	0.49	0.24	0.11
Trellis	**0.86**	0.31	0.46	0.79	0.40	0.28	0.16	0.48	0.25
Walker	**0.79**	0.54	0.38	0.53	0.09	0.54	0.53	0.43	0.10
cubicle	**0.81**	0.59	0.55	0.54	0.40	0.56	0.33	0.27	0.57
David	**0.88**	0.77	0.84	0.86	0.55	0.56	0.53	0.79	0.45
Tiger1	**0.70**	0.64	0.61	**0.70**	0.27	0.61	0.13	0.21	0.23
Tiger2	**0.66**	0.57	0.23	0.56	0.15	0.59	0.22	0.13	0.24
Girl	0.78	0.55	0.72	**0.79**	0.65	0.52	0.48	0.69	0.72
Coke	0.64	0.30	0.53	**0.68**	0.16	0.34	0.06	0.10	0.16
Faceocc1	0.90	0.74	0.76	0.83	**0.92**	0.72	0.40	0.76	0.89
Faceocc2	**0.86**	0.70	0.72	0.81	0.68	0.69	0.68	0.74	0.55
Sylv	**0.80**	0.75	0.67	0.76	0.58	0.62	0.39	0.43	0.30
Surfer	**0.72**	0.06	0.52	0.62	0.31	0.71	0.25	0.49	0.06

Table 2: The quantitative comparison results of the nine trackers over the eighteen video sequences. The table reports their average VORs over each video sequence.

$$\text{VOR} = \frac{area(B_p \bigcap B_{gt})}{area(B_p \bigcup B_{gt})}.$$

3.2. Empirical comparison of trackers

We compare the proposed tracker with several state-of-the-art trackers both qualitatively and quantitatively. These trackers are referred to as ORF (online random forest tracker [25]), CT (compressive tracker [33]), Struck (structured learning tracker [11]), Frag (Fragment-based tracker [1]), MIT (multiple instance tracker [2]), OAB (online AdaBoost tracker [9] that is implemented with the same configuration as OAB1 in [2]), IVT (incremental subspace tracker [24]), L1T (ℓ_1 minimization tracker [21]). In the experiments, the following trackers are implemented using their publicly available source code: ORF, CT, Struck, Frag, MIT, OAB, IVT, and L1T.

We evaluate the CLE and VOR performance of all the nine trackers on eighteen video sequences. Fig. 2 shows the qualitative tracking results of the nine trackers over several representative frames of eight video sequences. Fig. 3 plots the frame-by-frame CLEs (highlighted in different colors) obtained by the nine trackers for the fifteen video sequences. Tabs. 1-2 report the average CLEs and VORs of the nine trackers on each of all the eighteen video sequences. From Fig. 3 and Tabs. 1-2, we observe that the proposed tracker achieves the best tracking performance on most video sequences. In particular, the proposed tracker obtains the more robust tracking results in the presence of complicated appearance changes (caused by occlusion, drastic pose variation, background clutter, image distortion and blurring, etc.). An example of severe occlusion and background distraction is the "PedCross" sequence, shown bottom left of Fig. 2. The tracked pedestrian, moving right to left, is lost by all other trackers at frame 78 as he overlaps

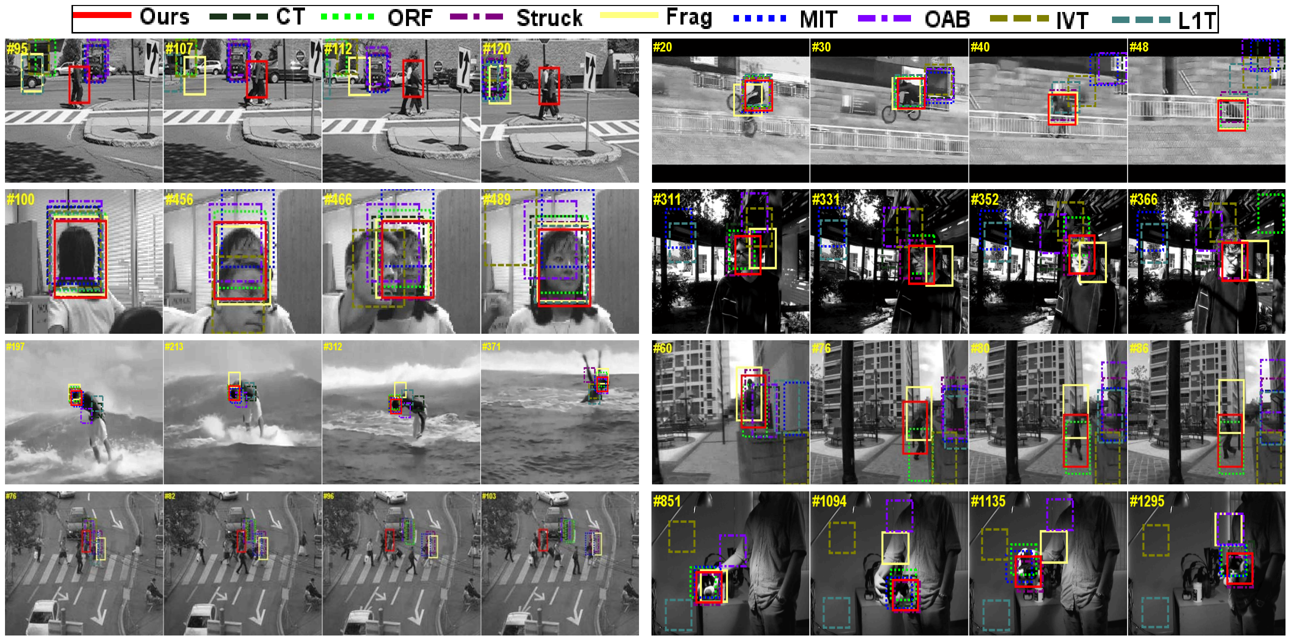

Figure 2: Qualitative tracking results of the nine trackers over several representative frames of the eight video sequences (*i.e.,* "Walker", "Cyclist", "Girl", "Trellis", "Surfer", "Jumper", "PedCross", and "Sylv") that are respectively aligned from left to right and from up to down.

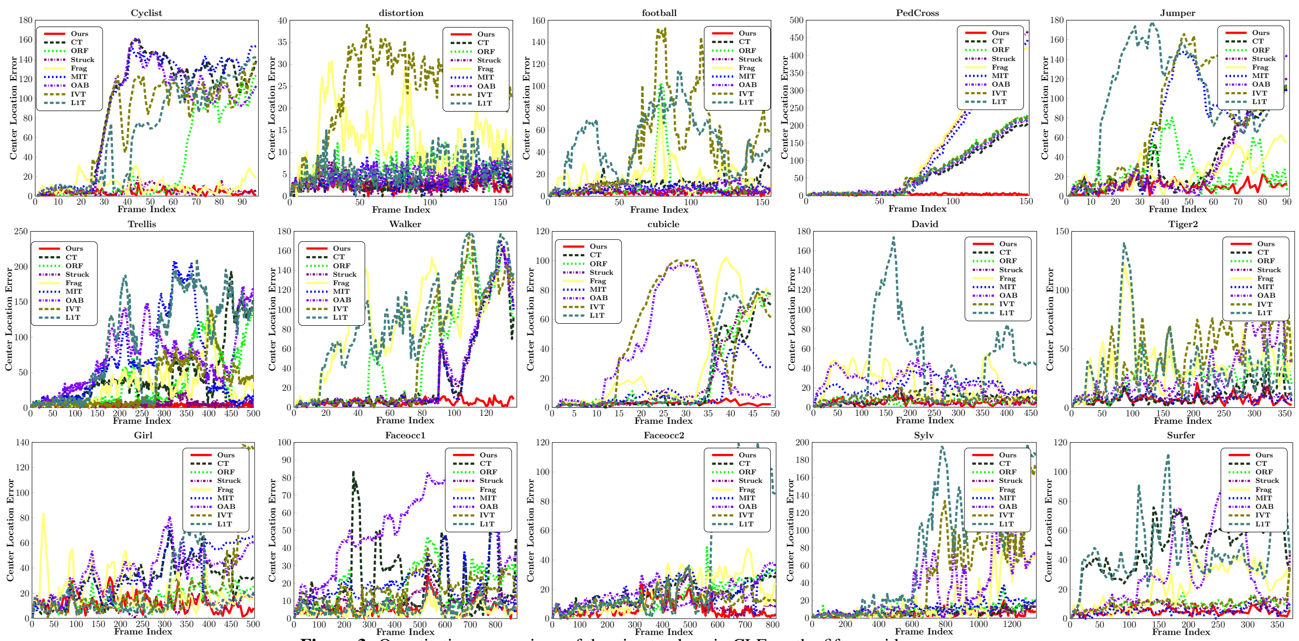

Figure 3: Quantitative comparison of the nine trackers in CLE on the fifteen video sequences.

with other pedestrians. The other trackers lock onto different pedestrians, so their error increases for the rest of the sequence, as shown in Fig. 3. The "Cyclist" video sequence, top right of Fig. 2, contains drastic pose variation followed by partial occlusion. MIT, IVT, OAB, and CT break down after the 30th frame due to the change in body pose. L1T and ORF lose the target after the 47th frame and the 69th frame, respectively, due to occlusion by the railing. Frag is able to keep track of the target, but also includes some background. In contrast, both Struck and our tracker are robust to the body pose variations and partial occlusions encountered throughout the entire video sequence. The "Trellis" video sequence, second row right of Fig. 2, contains significant illumination changes. L1T, OAB, and MIT begin to drift away from the target after the 195th frame because of the changing lighting conditions. Due to the combination of lighting and changing head pose, IVT, ORF, Frag, and CT fail to track the target after the 367th frame. Both our tracker and Struck successfully track the target across the whole video sequence, although our tracker locates the

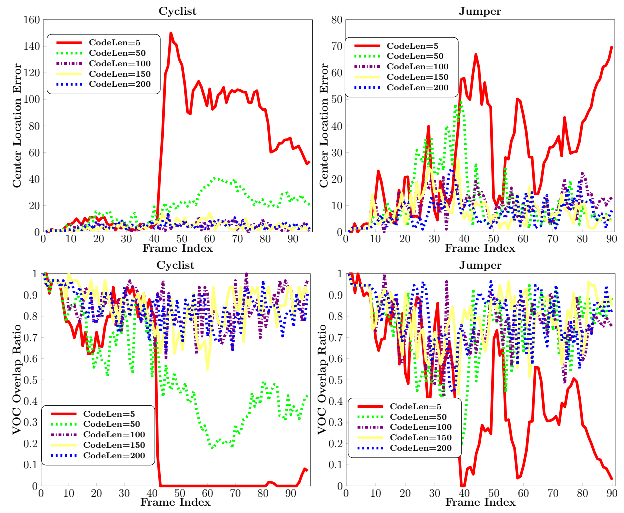

Figure 4: Quantitative evaluation of using different binary code lengths in CLE and VOR on the "Cyclist" and "Jumper" video sequences.

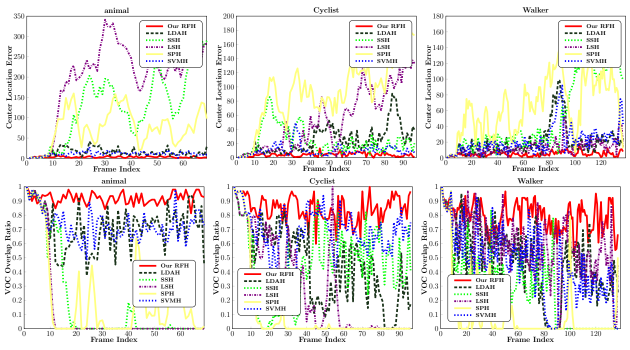

Figure 5: Quantitative evaluation of using different hashing methods in CLE and VOR on the "animal", "Cyclist", and "Walker" video sequences.

head more accurately. The "Girl" video sequence, second row left of Fig. 2, contains out-of-plane rotations followed by partial and severe occlusions. As shown in Fig. 3, both Struck and our tracker obtain more accurate tracking results than the other trackers.

3.3. Discussion and analysis

In this section, we evaluate each component to show its contribution to the overall performance of the tracker and its sensitivity to parameter settings. The effect of each component varies for each sequence, so we show a different but representative subset for each evaluation.

Binary code length We quantitatively evaluate the performance of the proposed tracker for five different binary code lengths. Fig. 4 shows the quantitative CLE and VOR performance on two video sequences. Clearly, it is seen from Fig. 4 that the CLE (VOR) performance improves as code length increases, and plateaus with approximately more than 100 hashing bits. This is a desirable property because we do not need a high-dimensional binary feature to achieve promising tracking performance.

Comparison of hashing methods Different hashing methods generate different binary codes, encoding various

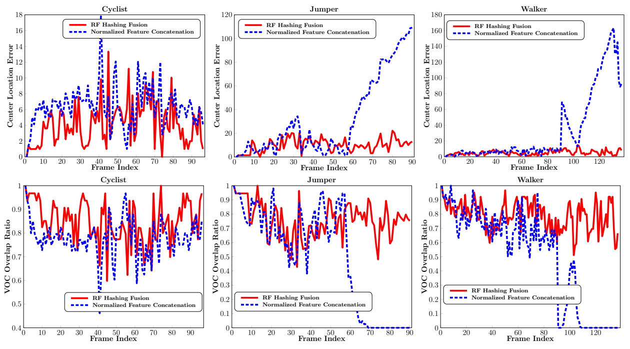

Figure 6: Quantitative evaluation of using different feature fusion methods in CLE and VOR on the "Cyclist", "Jumper", and "Walker" video sequences.

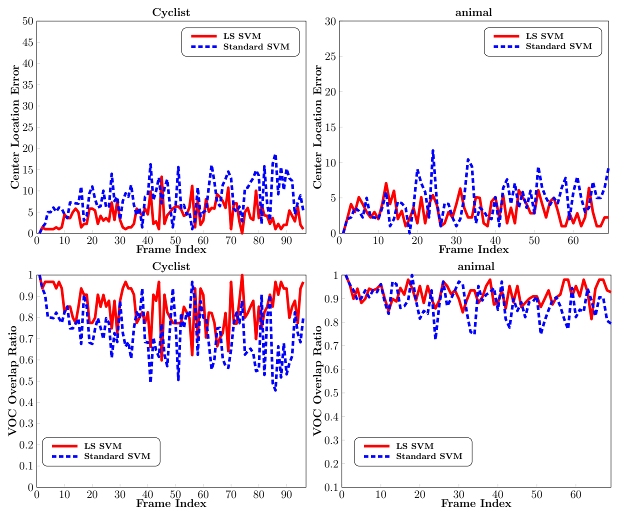

Figure 7: Quantitative evaluation of the proposed tracker with different SVMs in CLE and VOR on the "Cyclist" and "animal" video sequences.

discriminative information on object appearance. Here, we aim to evaluate different hashing methods applied to the task of feature fusion, including LDAH (linear discriminant analysis hashing [26]), SSH (semi-supervised hashing [29]), LSH (locality sensitive hashing [5]), SPH (spectral hashing [30]), SVMH (support vector machine hashing [14]), and our RFH (random forest hashing). As shown in Fig. 5, our RFH used in the proposed tracker achieves the better CLE and VOR performance than the other hashing methods in most cases.

Evaluation of feature fusion methods In order to verify the effectiveness of our feature fusion method, we compare it to a direct concatenation of normalized feature vectors into a unified feature vector. Fig. 6 displays the quantitative CLE and VOR performance of the proposed tracker with different feature fusion methods. Clearly, we see that our feature fusion method outperforms the standard feature fusion method in most cases.

Comparison of SVMs To justify the effectiveness of the LS-SVM, we compare it to the standard SVM. Fig. 7 shows the quantitative CLE and VOR tracking results on two video sequences. From Fig. 7, we clearly see that the proposed

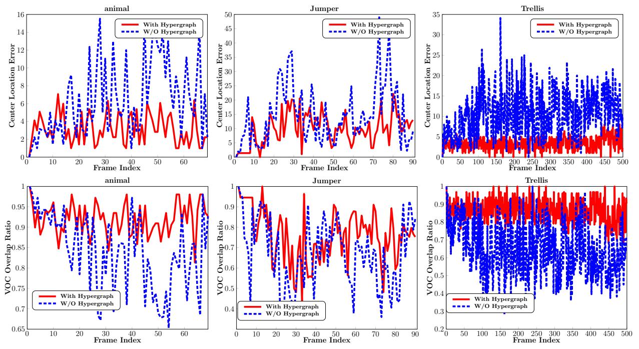

Figure 8: Perform comparison of the proposed tracker with and without hypergraph propagation in CLE and VOR on the "animal", "Jumper", and "Trellis" video sequences.

tracker using the LS-SVM achieves close but slightly superior tracking performance to the standard SVM.

Performance with and without hypergraph propagation The task of hypergraph propagation is to refine the confidence scores by random walk on the object/non-object community hypergraph. Fig. 8 exhibits the quantitative CLE and VOR tracking results of the proposed tracker with and without hypergraph propagation on three video sequences. It is clearly seen from Fig. 8 that hypergraph propagation gives rise to performance gain.

4. Conclusion

In this paper, we have proposed a robust visual tracker that learns compact and discriminative binary codes for an effective image representation. To obtain this representation, we develop a random forest hashing method, which efficiently constructs a set of hash functions by learning several randomized decision trees. To perform object/non-object classification, we build a discriminative appearance model based on incremental LS-SVM, which can be solved extremely efficiently in closed-form. Compared with the standard linear SVM, we have empirically shown that incremental LS-SVM has a simpler implementation with comparable accuracy. To further improve the accuracy of object localization, we present a hypergraph propagation method to capture the interaction information from samples and their contexts. Compared with several state-of-the-art trackers on eighteen challenging sequences, we empirically show that our tracker is able to achieve more accurate and robust tracking results in challenging conditions.

Acknowledgments This work was in part supported by ARC grants DP1094764 and FT120100969.

References

[1] A. Adam, E. Rivlin, and I. Shimshoni. Robust fragments-based tracking using the integral histogram. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 798–805, 2006.

[2] B. Babenko, M. Yang, and S. Belongie. Visual tracking with online multiple instance learning. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 983–990, 2009.

[3] Y. Bai and M. Tang. Robust tracking via weakly supervised ranking svm. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1854–1861, 2012.

[4] A. Bosch, A. Zisserman, and X. Muoz. Image classification using random forests and ferns. In *Proc. IEEE Int. Conf. Comp. Vis.*, pages 1–8, 2007.

[5] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. ACM Ann. Symp. Comp. Geometry*, pages 253–262, 2004.

[6] J. Fan, Y. Wu, and S. Dai. Discriminative spatial attention for robust tracking. In *Proc. Eur. Conf. Comp. Vis.*, pages 480–493, 2010.

[7] Y. Fu, L. Cao, G. Guo, and T. Huang. Multiple feature fusion by subspace learning. In *Proc. ACM Int. Conf. Content-based Image & Video Retrieval*, pages 127–134, 2008.

[8] G. Golub and C. Van Loan. *Matrix computations*, volume 3. Johns Hopkins University Press, 1996.

[9] H. Grabner, M. Grabner, and H. Bischof. Real-time tracking via on-line boosting. In *Proc. British Machine Vis. Conf.*, pages 47–56, 2006.

[10] M. Grabner, H. Grabner, and H. Bischof. Learning features for tracking. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1–8, 2007.

[11] S. Hare, A. Saffari, and P. Torr. Struck: Structured output tracking with kernels. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2011.

[12] A. S. Householder. *The theory of matrices in numerical analysis*. Blaisdell Publishing Co.: New York, 1964.

[13] Y. Huang, Q. Liu, S. Zhang, and D. Metaxas. Image retrieval via probabilistic hypergraph ranking. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 3376–3383, 2010.

[14] A. Joly and O. Buisson. Random maximum margin hashing. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 873–880, 2011.

[15] J. Kwon and K. M. Lee. Visual tracking decomposition. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1269–1276, 2010.

[16] H. Li, C. Shen, and Q. Shi. Real-time visual tracking with compressive sensing. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2011.

[17] X. Li, A. Dick, C. Shen, A. van den Hengel, and H. Wang. Incremental learning of 3d-dct compact representations for robust visual tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2013.

[18] X. Li, W. Hu, Z. Zhang, X. Zhang, M. Zhu, and J. Cheng. Visual tracking via incremental log-euclidean riemannian subspace learning. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1–8, 2008.

[19] X. Li, C. Shen, Q. Shi, A. Dick, and A. van den Hengel. Non-sparse linear representations for visual tracking with online reservoir metric learning. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 1760–1767, 2012.

[20] H. Lu, W. Zhang, and Y. Chen. On feature combination and multiple kernel learning for object tracking. *Proc. Asian Conf. Comp. Vis.*, pages 511–522, 2011.

[21] X. Mei and H. Ling. Robust visual tracking and vehicle classification via sparse representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2011.

[22] F. Porikli, O. Tuzel, and P. Meer. Covariance tracking using model update based on lie algebra. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, volume 1, pages 728–735, 2006.

[23] M. J. D. Powell. A theorem on rank one modifications to a matrix and its inverse. *Computer Journal*, 12(3):288–290, 1969.

[24] D. A. Ross, J. Lim, R. Lin, and M. Yang. Incremental learning for robust visual tracking. *Int. J. Comp. Vis.*, 77(1):125–141, 2008.

[25] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In *Proc. IEEE Int. Conf. Comp. Vis. Workshops*, pages 1393–1400, 2009.

[26] C. Strecha, A. Bronstein, M. M. Bronstein, and P. Fua. LDAHash: Improved matching with smaller descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(1):66–78, 2012.

[27] T. Van Gestel, J. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, and J. Vandewalle. Benchmarking least squares support vector machine classifiers. *Machine Learn.*, 54(1):5–32, 2004.

[28] M. Varma and D. Ray. Learning the discriminative power-invariance trade-off. In *Proc. IEEE Int. Conf. Comp. Vis.*, pages 1–8, 2007.

[29] J. Wang, S. Kumar, and S. Chang. Semi-supervised hashing for large scale search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2012.

[30] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proc. Adv. Neural Inf. Process. Syst.*, 2008.

[31] Y. Wu, J. Cheng, J. Wang, H. Lu, J. Wang, H. Ling, E. Blasch, and L. Bai. Real-time probabilistic covariance tracking with efficient model update. *IEEE Trans. Image Proc.*, 21(5):2824–2837, 2012.

[32] J. Ye and T. Xiong. Svm versus least squares svm. In *Proc. Int. Conf. Artificial Intelligence & Stat.*, pages 640–647, 2007.

[33] K. Zhang, L. Zhang, and M. Yang. Real-time compressive tracking. In *Proc. Eur. Conf. Comp. Vis.*, 2012.

[34] T. Zhang, B. Ghanem, S. Liu, and N. Ahuja. Robust visual tracking via multi-task sparse learning. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 2042–2049, 2012.

[35] D. Zhou, J. Huang, and B. Scholkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Proc. Adv. Neural Inf. Process. Syst.*, volume 19, 2007.