Visual tracking with randomly projected ferns

Jin Zhang\textsuperscript{a}, Kai Liu\textsuperscript{a,}\textsuperscript{*}, Fei Cheng\textsuperscript{a}, YunSong Li\textsuperscript{b}

\textsuperscript{a} School of Computer Science and Technology, Xidian University, Xi’an, China
\textsuperscript{b} School of Telecommunication Engineering, Xidian University, Xi’an, China

\begin{abstract}
In this paper, a novel visual tracking method based on random ferns and random projection has been proposed. Instead of a binary comparison in the standard random ferns method, a subtraction is used in the proposed method. Different subtraction results are projected into a single real value by the random projection. The real value from the subtraction is more informative than a binary comparison, thus less subtractions are needed. More importantly, the proposed method has a constant memory resource requirement independent from the number of subtractions in each fern, whereas in a standard random ferns method, the requirement is an exponential relationship with the number of binary comparisons. Besides low memory requirement, our approach is also computational efficient. Experimental results demonstrate that the proposed method performs better than the standard random ferns method and some other methods, especially in memory and computing resources requirements, which are of vital importance in some embedded systems.
\end{abstract}

1. Introduction

Visual tracking is one of the classic problems in computer vision. It is the basis for many other high level computer vision tasks such as action recognition, human computer interaction and behavior analysis. Also it is the basis for many applications in embedded systems such as human face recognition and tracking in cameras, and tracking of object for augmented reality in cell phones. Though many researches have been done in this area, it is still a challenging problem due to various challenges in a real world environment such as illumination variation, appearance variation, occlusion, and motion blur. There has been a tendency that object detection methods are adapted to online versions for visual tracking recently, known as tracking by detection. In this perspective, visual tracking is treated as a binary-class (foreground/background) classification problem. Due to the success of object detection, many methods adapted from object detection work well for visual tracking. However, visual tracking is different from object detection. For example, object detection problems usually offer a large dataset with many training samples (e.g., the VOC challenge\cite{1}), while visual tracking problems offer only one image at the beginning of a video sequence specified manually or by an object detector. Object detection methods usually would not worry about the computational complexity too much, while for a practical visual tracking method, both precision and running time are required and on some embedded systems, even memory and computing resources are also limited. Therefore, many visual tracking methods proposed in the literature are not appropriate for these occasions. In this paper, a novel visual tracking method named Randomly Projected Ferns (RPF) is proposed to address the computational complexity and memory usage problems.

\textit{Keywords:}
Visual tracking
Random ferns
Random projection
Resource limited

\textit{Abbreviation:}
VOC, visual object classes; RPF, Randomly Projected Ferns; SSVM, structured output support vector machine; RF, random ferns; RIP, restricted isometry property; MIL, multiple instance learning; CT, compressive tracking; OAB, online adaboost; IIR, infinite impulse response; TLD, tracking learning detection

* Corresponding author. Tel.: +86 13892810532.
E-mail addresses: jinzhang.cv@gmail.com (J. Zhang), kailiu@mail.xidian.edu.cn (K. Liu), chengfei8582@163.com (F. Cheng), ysli@mail.xidian.edu.cn (Y. Li).

http://dx.doi.org/10.1016/j.image.2014.06.009
0923-5965/© 2014 Elsevier B.V. All rights reserved.

Please cite this article as: J. Zhang, et al., Visual tracking with randomly projected ferns, Signal Processing-Image Communication (2014), http://dx.doi.org/10.1016/j.image.2014.06.009
The rest of this paper is organized as follows. In Section 2, some related work in visual tracking and object detection are discussed. Section 3 gives an introduction of random ferns. The proposed method is presented in Section 4. Section 5 shows the experimental results, followed by conclusion in Section 6.

2. Related work

In visual tracking, tracking by detection became popular since the work of Avidan [2], in which the tracking problem was treated as a classification problem that distinguishes foreground from background. The classifier used in [2] was trained offline while more recent methods train a classifier online with the coming frames in order to track arbitrary objects without the requirement of the training samples of the target before tracking.

Grabner et al. [3] adapted the boosting method in an online fashion and proposed an online boosting tracker. In the first frame, they used the image patch in the specified bounding box as a positive sample and image patch in the surrounding of the bounding box as negative samples to train a classifier with the boosting framework. Later, Grabner et al. [4] proposed to use semi-supervised learning to address the drifting problem. The training samples from the first frame are labeled, while the samples from the coming frames are treated as unlabeled ones. Babenko et al. [5] proposed to use multiple instance learning [6] to address the labeling problem in visual tracking and achieve remarkable results. In order to take the instance importance into consideration, Zhang and Song [7] proposed a tracker based on weighted multiple instance learning. However, these boosting methods may suffer from efficient problems, which are of great importance for online visual tracking, especially for embedded systems where memory and computing resources are limited. More recently, Hare et al. [8] used a structured output support vector machine (SSVM) to predict the location of the target. Instead of learning a classifier, they proposed to learn a mapping from feature space to transformation space directly. They showed that the method is more accurate in tracking results. However, the training and the online update of the SSVM are very complex and require a large amount of memory and computing resources. Jiang et al. [9,10] proposed to use adaptive metric learning for visual tracking. Instead of applying a fixed metric in most visual tracking algorithms, their idea was to learn an optimal distance metric. Later, a nonparametric data-driven local metric adjustment method was proposed in [11] to account for the variation of the object appearance during tracking procedure. Zhang et al. [12] used random projection [13] in compressive sensing and proposed a tracking method which is simple and efficient. They used a random projection matrix to map the original high-dimensional image to a low-dimensional space in which the classification was done.

In object recognition, Ozuysal et al. [14] proposed a very simple yet effective method named random ferns for image description. They used simple intensity comparison on randomly selected pixel pairs to hash an image patch into a binary code, then a probability distribution is assigned to the binary codes extracted from the training samples. With the probability distribution, a new sample can be classified with Bayes’ formula. Random ferns was used by Kalal et al. [15] as the classifier in P–N learning, and showed excellent performance. Rao et al. [16] proposed to use online random ferns for the tracking problem. However, there exist some potential problems with random ferns. First, the memory requirement is enormous, having an exponent relationship with the number of pixel pairs in one fern, since each comparison of pixel pair results in one bit feature value. Second, the comparison of each pixel pair produces only two possible outputs, 0 or 1, leading to lot of other information loss. Thus more pixel pairs are needed to compensate for the loss, leading to an exponential memory requirement. In this paper, we propose a tracking method based on random ferns which requires constant memory independent from the number of pixel pairs in a fern. The method produces real value feature for a fern based on subtraction and random projection as shown in Fig. 1. The real value feature rather than integer composed of 0, 1 binary code requires less pixel pairs in a fern and less memory and computing resources.

3. Preliminaries

In this section, some reviews of random ferns [14] are presented. Let \( f_j, j = 1, 2, \ldots, N \) be the binary features generated from an image patch. The class \( c \) for this image patch can be predicted by

\[
\hat{c} = \arg \max_c p(c|f_1, f_2, \ldots, f_N).
\]

where \( C \) is the set of all possible classes. In visual tracking, usually \( C \) equals to \( \{bg, fg\} \), where \( bg \) and \( fg \) denote foreground and background respectively. With Bayes’ formula, the posterior can be rewritten as

\[
p(c|f_1, f_2, \ldots, f_N) = \frac{p(f_1, f_2, \ldots, f_N|c)p(c)}{p(f_1, f_2, \ldots, f_N)}.
\]

Fig. 1. Illustration of random projection of pixel pair differences in a fern. The differences of intensity values in pixel pairs \( f_{ij}, j = 1, 2, 3, 4 \) are weighted and summed by weights \( w_j, j = 1, 2, 3, 4 \), resulting in the real value \( F_i \).
where the denominator is a normalization constant that can be omitted. Assuming that the probability \( p(c) \) is uniform, (1) is identical to

\[
c = \arg \max \frac{p(f_1, f_2, \ldots, f_N | c)}{p(f_1, f_2, \ldots, f_N)}.
\]

(3)

In random ferns, the binary features are chosen as intensity comparisons between pixel pairs, which are very simple. As mentioned before, a large number of \( N \) is needed to compensate for the information loss in utilizing the simple binary features, leading to enormous even infeasible memory requirement (e.g., for \( N=25,32 \text{ MB} \) memory is needed). One possible solution is naive Bayes assumption that assumes \( f_1, f_2, \ldots, f_j \) are independent of each other, however, this assumption is too strong to be satisfied. Ozuysal et al. [14] proposed to use semi-naive Bayes approach to address the problem, by grouping several pixel pairs into a group and assuming that different groups are independent from each other.

Formally,

\[
p(f_1, f_2, \ldots, f_N | c) = \prod_{i=1}^{N/N_f} p(F_i | c),
\]

(4)

where \( N_f \) is the number of pixel pairs in each fern and \( F_i = \{f_{11}, f_{12}, \ldots, f_{N_f} \} \) is a group of features, which is called a fern. \( N_f = N/N_p \) is the total number of ferns. In practice, \( N_f \) cannot be too small, thus the memory occupation is also very enormous.

4. Proposed method

In this section, the proposed method will be introduced to address the problems encountered in the standard random ferns method.

4.1. Feature representation

In the standard random ferns method, the binary feature \( f_{ij} \) is generated by comparison of randomly generated pixel pair

\[
f_{ij} = \begin{cases} 
0 & \text{if } l(x^1(i,j)) > l(x^2(i,j)) \\
1 & \text{otherwise}
\end{cases},
\]

(5)

where \( x^1(i,j) \) and \( x^2(i,j) \) denote the coordinates of the first and second pixels of the randomly generated pixel pair \( j \) of fern \( i \). \( l(x) \) denotes the intensity of an image at \( x \). This approach is very simple and easy to implement, but only the relative intensities between two points are considered. Many information about how the intensity of these two pixels is different from each other is lost. For example, when pixel at \( x^1(i,j) \) has an intensity equals to 0, pixel at \( x^2(i,j) \) having an intensity of 1 is treated equally with that having an intensity of 255.

We propose a method to address the above problem. Instead of generating binary feature between pixel pair, we define the real value feature as

\[
\tilde{f}_{ij} = l(x^1(i,j)) - l(x^2(i,j)).
\]

(6)

The difference between (5) and (6) is that \( f_{ij} \in \{0,1\} \) while \( \tilde{f}_{ij} \in \mathbb{R} \). Thus \( \tilde{f}_{ij} \) keeps more information about the intensity difference between two pixels. It will be demonstrated in Section 5 that the calculation of (6) is as efficient as that of (5).

4.2. Random projection

In Section 4.1, real value feature \( \tilde{f}_{ij} \) is extracted from pixel pairs \( j \) of fern \( i \). In a standard random ferns method, the extracted binary value features \( f_{ij} \) in a fern are combined into a binary code. However, in the proposed method, the feature for each pixel pair is a real value. These real values can be handled simultaneously, but here we seek to “encode” these real values like the way processed in the standard random ferns. If the distances between points in the high-dimensional space can be preserved in their low-dimensional representations, then we can build classifiers based on these low-dimensional representations.

Assuming that \( u_1, u_2, \ldots, u_d \in \mathbb{R}^m \) are \( d \) column vectors in a space of dimension \( m \), \( v_1, v_2, \ldots, v_d \in \mathbb{R}^n \) are \( d \) column vectors mapped from \( u_1, u_2, \ldots, u_d \) by a mapping function \( f: \mathbb{R}^m \rightarrow \mathbb{R}^n \)

\[
v_j = f(u_j), \quad j \in \{1, 2, \ldots, d\},
\]

(7)

then the Johnson–Lindenstrauss (JL) lemma [17] states that the mapping function \( f \) exists so that the following holds:

\[
(1 - \epsilon) \| u_i - u_j \|^2 \leq \| v_i - v_j \|^2 \leq (1 + \epsilon) \| u_i - u_j \|^2,
\]

(8)

for any \( 0 < \epsilon < 1 \) and \( n > 4 \ln d/(\epsilon^2/2 - \epsilon^2/3) \). This means that the distance of two points \( i \) and \( j \) in the original dimension \( m \) space is preserved in the mapped dimension \( n \) space. Thus it provides a theoretical basis to classify high-dimensional features with low-dimensional representation. Next problem is how to find the mapping function \( f \). Assuming further that the vectors \( u_i \) and \( u_j \) are \( k \)-sparse (with at most \( k \) nonzero elements) and a matrix \( A \in \mathbb{R}^{n \times m} \), if

\[
(1 - \sigma) \| u_i - u_j \|^2 \leq \| A(u_i - u_j) \|^2 \leq (1 + \sigma) \| u_i - u_j \|^2,
\]

(9)

for all \( u_i, u_j \in \mathbb{R}^m \), then the matrix \( A \) is called \( (k, \sigma) \)-RIP (restricted isometry property) in compressive sensing [18,19]. It has been proved that for \( k \)-sparse data (e.g., image and audio signal) and some random matrix such as Gaussian, Bernoulli and Fourier matrix, (9) holds with high probability. Hence, we can use random projection (linear transformation with these random matrices) to project vectors from dimension \( m \) to dimension \( n \)

\[
[v_1, v_2, \ldots, v_d] = A[u_1, u_2, \ldots, u_d].
\]

(10)

while the distances between these vectors can be preserved with high probability. As a special case, we propose to use random projection to project feature values of different pixel pairs into a single real value. Formally,

\[
F_i = \sum_{j=1}^{N_f} w_{ij} \tilde{f}_{ij},
\]

(11)

where \( w_{ij} \) is a real value generated randomly according to a Gaussian distribution. The Gaussian vector \( w = [w_1, w_2, \ldots, w_M] \) satisfies restricted isometry property [18], thus the projection result \( F_i \) keeps most of the information of the original features \( \tilde{f}_{ij} \). Given real value features
Algorithm 1. Randomly projected ferns.

Input: Image patch $P$, Target width $w$ and height $h$, number of ferns $N_f$ and number of pixel pairs in a fern $N_p$.

Output: Projected feature value $F_i$, $i \in \{1, 2, \ldots, N_f\}$.

1: for $i = 0$; $i < N_f$; $i++$ do
2: \hspace{1em} $p(x[i]) = \text{ii}(0, w-1)$
3: \hspace{1em} $p(y[i]) = \text{ii}(0, h-1)$
4: end for
5: for $j = 0$; $j < N_p$; $j++$ do
6: \hspace{1em} $w[i] = \mathcal{N}(0, 1)$
7: end for
8: for $i = 0$; $i < N_f$; $i++$ do
9: \hspace{1em} $s = 0$
10: for $j = 0$; $j < N_p$; $j++$ do
11: \hspace{2em} $x^i = (\text{ii}(N_f \times i \times 2 + j \times 2) - p(x[i] \times i \times 2 + j \times 2))$
12: \hspace{2em} $x^j = (\text{ii}(N_f \times i \times 2 + j \times 2 + 1) - p(y[i] \times i \times 2 + j \times 2 + 1))$
13: \hspace{2em} $s = s + ((x^i - x^j) \cdot w[i])$
14: end for
15: $F_i = s$
16: end for

4.3. Joint probability

With the feature $F_i$ calculated as described in the previous section, the next problem is how to calculate the joint probability $p(F_i|c)$. In standard random ferns, the joint probability can be calculated by counting the frequency of a specific binary pattern since $F_i$ is a integer composed of binary code that can be enumerated. Formally,

$$p(F_i = x(c)) = \frac{\sum_{x \in \mathcal{C}(c)} \delta(F_i^k - x)}{|\mathcal{S}(c)|},$$

(12)

where $F_i^k$ is the feature value of fern $i$ in sample $k$, $\mathcal{S}(c)$ is the set of training samples that belong to class $c$ and $\delta(\cdot)$ is the Dirac function.

In the proposed method, since the output $F_i$ is a real value, the probability distribution can be modeled with parametric or nonparametric estimation. For simplicity, we model the joint probability as a Gaussian distribution with mean $\mu_i$ and standard deviation $\sigma_i$ for fern $i$ of class $c$.

$$p(F_i = x(c)) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp \left( -\frac{(x - \mu_i)^2}{2\sigma_i^2} \right).$$

(13)

The parameter $(\mu_i, \sigma_i)$ can be estimated from the training samples

$$\mu_i = \mathbb{E}[F_i|c],$$

$$\sigma_i = \sqrt{\mathbb{V}[F_i|c]} = \sqrt{\mathbb{E}[F_i|c] - (\mathbb{E}[F_i|c])^2}. \tag{14}$$

4.4. Online update

Different from object detection, it is needed in visual tracking to update the classifier online to follow the object with appearance variations. In the standard random ferns method, the online update can be handled by a LIR-like filter [20]. The frequency count of a specific binary pattern in a fern is then updated by the weighted sum of the count at the previous frame and the count is calculated by training samples in current frame. In our approach, the value of each fern $F_i$ is modeled as a parametric distribution, specifically, Gaussian distribution with parameter $(\mu_i, \sigma_i)$. Thus the update of the classifier can be simplified as a weighted parameter update

$$\mu_i \leftarrow (1 - \lambda)\mu_i + \lambda \mu_i^{\text{new}},$$

$$\sigma_i \leftarrow (1 - \lambda)\sigma_i + \lambda \sigma_i^{\text{new}}, \tag{15}$$

where $\lambda$ is the learning rate, $\mu_i^{\text{new}}$ and $\sigma_i^{\text{new}}$ are estimated from the training samples at current frame with (14).

4.5. Visual tracking with randomly projected ferns

Before the tracking procedure, $N$ random pixel pairs are generated in a rectangle, the size of the specific bounding box around the target. Together with random pixel pairs, random weight $w_i$ is also generated for each pixel pair. Positive samples are generated from a set of patches in an annular area near the original bounding box in the first frame specified manually. Similarly, negative samples are generated from a set of patches in an annular area far away from the original bounding box. With the positive and negative samples generated, the parameter $(\mu_i, \sigma_i)$ of fern $i$ can be estimated by (14). In visual tracking, the classifier is required to distinguish foreground from background. The discriminative function (1) for an image patch can be converted to

$$S(F) = \sum_{c=1}^{N_c} \log(p(F|c = fg)) - \sum_{c=1}^{N_c} \log(p(F|c = bg)),$$

(16)

where $F = (F_1, F_2, \ldots, F_{N_f})$ is a set containing the value of all ferns for an image patch.

When a new frame comes, the tracker samples all possible locations in a radius $r_s$ with the center of previous bounding box as the origin. The value $F$ on each sampled image patch is calculated and the one gives the highest score of (16) is predicted as the target. The predicted target is used to learn a new Gaussian probability distribution with parameter $(\mu_i^{\text{new}}, \sigma_i^{\text{new}})$, and then the classifier is updated by (15). The proposed method is summarized in Algorithm 2. For a bounding box $b, l, b(b)$ outputs the image patch inside box $b$. Line 1 generates all possible target image patches in an annular area. The features of these image patches are calculated in line 2. Target position is
determined by the image patch with the highest classifier score in line 3. To follow the variation of target appearance, line 4 generates new positive and negative training samples. In line 4, \( r_{\text{pos}} \) and \( r_{\text{neg}} \) denote the minimum and maximum radii, respectively, for positive samples. Similarly, \( r_{\text{neg}} \) and \( r_{\text{neg}} \) denote the minimum and maximum radii, respectively, for negative samples. The training samples are then be used to train new classifiers with new parameters in line 5. Finally, the tracker parameters are updated in line 6.

**Algorithm 2.** Visual tracking with randomly projected ferns.

**Input:** Frame \( I_t \), position of previous bounding box \( b_{t-1} \)

**Output:** Predicted bounding box \( b_t \)

1. Sample from frame \( I_t \) to get candidate image patches set \( I_{\text{cand}} = \{ I(b) \mid b - b_{t-1} \leq r_{\text{min}} \} \).
2. Calculate values of each fern \( F_{k} = (r_{1}^{k}, r_{2}^{k}, \ldots, r_{N}^{k}) \) of \( k \)-th candidate image patch in \( I_{\text{cand}} \) with Algorithm 1, where \( k = 1, 2, \ldots, I_{\text{cand}} \).
3. Target image patch is predicted by maximizing (16)
   \[ \hat{k} = \arg \max_i S(f_{k}^{*}) \], and the position of the bounding box is updated by \( b_t \leftarrow b^* \).
4. Sample from frame \( I_t \) to get positive and negative sample sets
   \( I_{\text{pos}} = \{ I(b) \mid |b - b_{t} - c| < r_{\text{pos}} \} \),
   \( I_{\text{neg}} = \{ I(b) \mid |b - b_{t} - c| > r_{\text{neg}} \} \).
5. Calculate the parametric distribution of the positive and negative sample sets with (14), resulting in \( \mu_{\text{pos}} \) and \( \sigma_{\text{pos}} \),
   where \( c \in [b_{\text{fg}}, b_{\text{bg}}] \).
6. Update the parameters of the classifier with (15).

### 4.6. Verification of projected ferns

The theoretical part of the proposed approach was introduced in Section 4.2. This section presents experimental results of the random projection to verify the effectiveness of the proposed approach. The David video sequence is used as the test video sequence. The mean \( \mu \) and standard deviation \( \sigma \) of the projected value \( F_{1} \) are shown in Tables 1 and 2 for ferns #1 and #2 respectively. It can be found that in both ferns, the mean of foreground \( \mu_{\text{fg}} \) and background \( \mu_{\text{bg}} \) are different. This is reasonable since the appearance of foreground and background is different and the random projection can preserve the distance of high-dimensional image signals with their low-dimensional representation. Thus we can build classifiers based on the low-dimensional representation to distinguish between foreground and background. It can also be found that the standard deviation of foreground \( \sigma_{\text{fg}} \) is significantly smaller than that of background \( \sigma_{\text{bg}} \). This is because that the appearance of the background is more diverse than the foreground.

### 4.7. Related to other work

The proposed method is perhaps the most similar to the work of Zhang et al. [12], but different from that. Firstly, a Haar-like feature is used in [12], while in the proposed method, the framework of random ferns is used, in which features are generated directly from the difference between the intensities within pixel pairs. Secondly, the number of Haar-like features in [12] is enormous and randomly projecting these features directly is computationally intensive. Thus a very sparse projection matrix [13] was used to approximate in [12]. The proposed method is based on random ferns in which the number of pixel pairs in a fern is limited, thus the standard Gaussian projection matrix can be used directly without worrying about the computational complexity. Thirdly, the tracker in [12] used a single projection matrix to project high-dimensional data into a low-dimensional representation. In our approach, the projection is inside a fern. Different pixel pairs in a fern are projected into a real value. Thus a lot of small projections are applied instead of a single large projection.

The presented approach is based on the framework of random ferns [14] and different from random ferns in feature representation and encoding. Firstly, the raw feature representation in the original random ferns approach was a pixel value comparison while in the proposed approach, the pixel value difference is used. Secondly, the comparison in [14] was encoded into a binary code while in our method, the differences of pixel pairs are encoded into a single real value with random projection. Thirdly, the class conditional probability density of different classes in the original random ferns was estimated by the frequency of binary codes and the online updating was achieved with a IIR-like filter. In the presented method, the classifier is built upon Gaussian distribution of foreground and background and the classifier is updated by the updating of the parameters of the Gaussian distribution (Eq. (15)).

### 5. Experiments

In this section, comparison of the proposed Randomly Projected Ferns (RPF) method with the standard random ferns method (RF) and several other trackers proposed in the literature will be presented. These trackers are multiple instance learning tracking (MIL) [5], compressive tracking (CT) [12], online adaboost (OAB) [3], structured output tracking (Struck) [8] and tracking-learning-detection (TLD) [15]. All the experiments ran on a machine with a 3.0 GHz CPU and 8 GB RAM.

---

**Table 1** Statistics parameters of the projected feature value of fern #1.

<table>
<thead>
<tr>
<th>Frame number</th>
<th>( \mu_{\text{fg}} )</th>
<th>( \mu_{\text{bg}} )</th>
<th>( \mu_{\text{fg}} )</th>
<th>( \mu_{\text{bg}} )</th>
<th>( \mu_{\text{fg}} )</th>
<th>( \mu_{\text{bg}} )</th>
<th>( \mu_{\text{fg}} )</th>
<th>( \mu_{\text{bg}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>65.57</td>
<td>62.76</td>
<td>64.66</td>
<td>44.86</td>
<td>44.23</td>
<td>39.22</td>
<td>39.42</td>
<td>40.36</td>
</tr>
<tr>
<td>2</td>
<td>24.74</td>
<td>28.88</td>
<td>22.13</td>
<td>30.43</td>
<td>32.08</td>
<td>28.88</td>
<td>24.24</td>
<td>20.95</td>
</tr>
<tr>
<td>3</td>
<td>10.15</td>
<td>2.18</td>
<td>14.80</td>
<td>2.40</td>
<td>19.46</td>
<td>5.80</td>
<td>34.81</td>
<td>22.72</td>
</tr>
<tr>
<td>4</td>
<td>6.33</td>
<td>65.33</td>
<td>59.39</td>
<td>79.32</td>
<td>76.43</td>
<td>80.40</td>
<td>83.51</td>
<td>72.52</td>
</tr>
</tbody>
</table>

---

1. The number of pixel pairs in random ferns [14] is 14 for planar object detection. The number of Haar-like features is in the order of 10^6 to 10^10 as claimed in [12].
2. We implement RPF and RF in C++ with OpenCV. The implementation of other trackers are offered by the corresponding authors.
values 1 (OAB1) and 5 (OAB5) are tested and the results are generating only one positive sample. In this experiment, the total number of ferns is set to $N_f = 150$ (samples), leading to totally $K = 150$ training samples. The setting generates many image patches in the next frame are training targets in adjacent frames and thus with high possibility of foreground, we need more background training (since the appearance of background is more diverse than that of foreground). Positive and negative samples are sampled from different annular areas. Annular areas for positive and negative samples are set to $r_{pos}^{min} = 0$, $r_{pos}^{max} = 4$ and $r_{neg}^{min} = 8$, $r_{neg}^{max} = 30$ respectively. $r_{pos}^{max}$ controls the sensitivity of the classifier. A small $r_{pos}$ makes the classifier more sensitive but more likely to over fitting since with small $r_{pos}$, positive samples are less diverse. $r_{neg}^{min}$ is set slightly larger than $r_{pos}$ since we want the differences between positive and negative samples become obvious. $r_{neg}^{max}$ is set close to $r_s$ as the limited moving distance of the target in adjacent frames and thus with high possibility, many image patches in the next frame are training samples in the current frame. The setting generates approximately 50 positive samples and a large number of negative samples. Among the large number of negative samples, 100 negative samples are selected randomly (since the appearance of background is more diverse than that of foreground), we need more background training samples), leading to totally $K = 150$ training samples. The total number of ferns is set to $N_f = 50$ and the number of pixel pairs in a fern is set to $N_p = 2$. In [14], these parameters are set to $N_f = 20, N_p = 14$ for planar objects and $N_f = 50, N_p = 11$ for 3D objects. We observe that the above parameters give good results on both accuracy and efficiency (computational complexity and memory footprint). The learning rate is set to $\lambda = 0.1$. A larger $\lambda$ would make the tracker quickly adapt to the new target appearance while a smaller $\lambda$ would make the tracker more stable and less likely to drift. Experimentally, we find that 0.1 is a suitable learning rate with the ability to follow the object appearance variation and to overcome drifting. The parameters of standard random ferns tracker are set to the same as the proposed method, except for the number of pixel pairs $N_p$ in a fern and the number of ferns $N_f$, which are set to $N_f = 8$ (RF8), $N_f = 12$ (RF12) and $N_f = 50$ respectively, since the standard random ferns method needs more pixel pairs in a fern to compensate for the loss of information using only intensity comparison between pixels. In the original OAB tracker, $r_{pos}^{max}$ was set to 1, generating only one positive sample. In this experiment, values 1 (OAB1) and 5 (OAB5) are tested and the results are reported. The parameters of other trackers are the default ones offered by the corresponding authors.

5.2. Quantitative results

Two evaluation criteria are applied to the quantitative evaluation. One criterion is center error which is measured by the Euler distance between the center of the bounding box generated by the trackers and the ground truth. The other criterion, success rate, is measured by the percentage of successfully tracked frames. To judge whether a frame is successfully tracked by a tracker, a cover rate is used. The cover rate is defined as $S(T \cap S(G)) / (S(T) \cup S(G))$, where $S(T)$ is the area of the bounding box given by a tracker and $S(G)$ is that given by the ground truth. A bounding box given by a tracker for a frame with a cover rate larger than 50% is treated as the frame being successfully tracked. Tables 3 and 4 show the center error and success rate of various trackers respectively.

5.3. Qualitative results

Qualitative comparisons have also been conducted in the experiment. Fig. 2 plots the center error of different trackers. Some snapshots of the tracking results are shown in Fig. 3.

5.4. Discussion

5.4.1. Performance analysis

From Table 3, it can be found that the proposed method produces lower center errors than the standard random ferns method (both $N_f = 8$ and $N_f = 12$) on most sequences. Compared with other trackers, the proposed method produces lower center error than CT, MIL, OAB, and TLD on most video sequences and is comparable with Struck. The average center error of the proposed method is lower than others. From Table 4, it can be found that the proposed method produces higher success rate than RF, CT, MIL, OAB, and TLD on most video sequences and is comparable with Struck. On average, the proposed method also performs better than others. The conclusion can also be observed from Fig. 2. Note that there exist some video sequences in which the proposed method performs not as well as others. In fact no tracker can perform better than others on all video sequences. The purpose of this paper is to propose an efficient visual tracking method that requires very low memory and computing resources, which will be discussed in the following sections.

Table 2

<table>
<thead>
<tr>
<th>Frame number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_{fs}$</td>
<td>-89.63</td>
<td>-73.13</td>
<td>-70.19</td>
<td>-83.85</td>
<td>-84.00</td>
<td>-86.46</td>
<td>-96.12</td>
<td>-108.66</td>
</tr>
<tr>
<td>$\sigma_{fs}$</td>
<td>24.91</td>
<td>21.88</td>
<td>23.84</td>
<td>21.26</td>
<td>25.22</td>
<td>27.35</td>
<td>28.82</td>
<td>19.23</td>
</tr>
<tr>
<td>$\mu_{hs}$</td>
<td>-5.73</td>
<td>-8.83</td>
<td>-12.00</td>
<td>-7.73</td>
<td>-7.47</td>
<td>-9.18</td>
<td>-7.57</td>
<td>-24.06</td>
</tr>
<tr>
<td>$\sigma_{hs}$</td>
<td>51.15</td>
<td>58.80</td>
<td>54.97</td>
<td>62.09</td>
<td>58.18</td>
<td>58.49</td>
<td>65.41</td>
<td>65.72</td>
</tr>
</tbody>
</table>
5.4.2. Memory resource usage analysis

The random ferns method is famous for its simplicity and efficiency, but the memory requirement is enormous. Assume that the number of classes is \( r \) (in visual tracking, usually \( r = 2 \) denoting foreground and background) and the counter used to save the frequency of a specific binary pattern occupies 4 Bytes (e.g., \texttt{int} in C/C++). Then the memory requirement is

\[
\text{MEM}_{\text{rf}} = \frac{N_f}{C^2} \times \frac{r}{C^2} \times 4
\]  
(17)

In the experiment, the parameters are set as \( N_f = 8, N_F = 50, r = 2 \), thus the memory requirement is \( \text{MEM}_{\text{rf}} = 2^8 \times 50 \times 2 \times 4 = 102,400 \) Bytes. The proposed method projects values calculated from pixel pairs to a real value. Assuming that the real value is stored in a single precision floating point number which occupies 4 Bytes (e.g., \texttt{float} in C/C++), the memory requirement is

\[
\text{MEM}_{\text{rpf}} = \frac{N_f}{C^2} \times \frac{r}{C^2} \times 4
\]  
(18)

It is clear that the standard random ferns method needs memory \( 2^N \) times more than the proposed method. In the experiment, \( \text{MEM}_{\text{rpf}} = 50 \times 2 \times 4 = 400 \) Bytes.

Compared with other trackers, CT, MIL, OAB and Struck need to calculate the integral image of the original image, thus in a matrix the dimension of the original image is needed. Assuming that each pixel in the integral image occupies 4 Bytes, the memory requirement for an image of size 320 \times 240 is \( \text{MEM}_{\text{integral}} = 320 \times 240 \times 4 = 307,200 \) Bytes, which is higher than the standard random ferns method and much higher than the proposed method. TLD contains a 1-NN classifier and the calculation of optical flow. Thus the memory requirement is also enormous.

### Table 3
Center error (in pixels) of trackers with respect to the ground truth.

<table>
<thead>
<tr>
<th>Video name</th>
<th>RPF</th>
<th>RF8</th>
<th>RF12</th>
<th>CT</th>
<th>MIL</th>
<th>OAB1</th>
<th>OAB5</th>
<th>Struck</th>
<th>TLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boy</td>
<td>5</td>
<td>5</td>
<td>25</td>
<td>27</td>
<td>6</td>
<td>7</td>
<td>18</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>CarDark</td>
<td>2</td>
<td>8</td>
<td>23</td>
<td>42</td>
<td>22</td>
<td>12</td>
<td>13</td>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>David</td>
<td>27</td>
<td>10</td>
<td>3</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>24</td>
<td>27</td>
<td>7</td>
</tr>
<tr>
<td>David2</td>
<td>5</td>
<td>22</td>
<td>3</td>
<td>38</td>
<td>5</td>
<td>5</td>
<td>57</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>David3</td>
<td>4</td>
<td>96</td>
<td>95</td>
<td>34</td>
<td>7</td>
<td>43</td>
<td>41</td>
<td>4</td>
<td>65</td>
</tr>
<tr>
<td>Deer</td>
<td>26</td>
<td>67</td>
<td>16</td>
<td>60</td>
<td>59</td>
<td>43</td>
<td>36</td>
<td>4</td>
<td>28</td>
</tr>
<tr>
<td>Dudek</td>
<td>15</td>
<td>12</td>
<td>13</td>
<td>19</td>
<td>10</td>
<td>59</td>
<td>41</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>Facecoc1</td>
<td>5</td>
<td>13</td>
<td>22</td>
<td>16</td>
<td>15</td>
<td>31</td>
<td>53</td>
<td>8</td>
<td>33</td>
</tr>
<tr>
<td>Fish</td>
<td>8</td>
<td>6</td>
<td>13</td>
<td>6</td>
<td>11</td>
<td>40</td>
<td>15</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>Football</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>6</td>
<td>7</td>
<td>96</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>Girl</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>15</td>
<td>3</td>
<td>17</td>
</tr>
<tr>
<td>Mhyang</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>7</td>
<td>10</td>
<td>5</td>
<td>40</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>Singer1</td>
<td>30</td>
<td>30</td>
<td>29</td>
<td>31</td>
<td>30</td>
<td>28</td>
<td>36</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>Singer2</td>
<td>8</td>
<td>16</td>
<td>18</td>
<td>7</td>
<td>39</td>
<td>84</td>
<td>85</td>
<td>88</td>
<td>163</td>
</tr>
<tr>
<td>Tiger2</td>
<td>13</td>
<td>31</td>
<td>31</td>
<td>14</td>
<td>20</td>
<td>36</td>
<td>29</td>
<td>11</td>
<td>41</td>
</tr>
<tr>
<td>Walking</td>
<td>9</td>
<td>110</td>
<td>118</td>
<td>8</td>
<td>6</td>
<td>11</td>
<td>7</td>
<td>7</td>
<td>109</td>
</tr>
<tr>
<td>Average</td>
<td>11</td>
<td>28</td>
<td>27</td>
<td>21</td>
<td>16</td>
<td>27</td>
<td>38</td>
<td>13</td>
<td>36</td>
</tr>
</tbody>
</table>

### Table 4
Success rate (%) of trackers.

<table>
<thead>
<tr>
<th>Video name</th>
<th>RPF</th>
<th>RF8</th>
<th>RF12</th>
<th>CT</th>
<th>MIL</th>
<th>OAB1</th>
<th>OAB5</th>
<th>Struck</th>
<th>TLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boy</td>
<td>94</td>
<td>95</td>
<td>21</td>
<td>36</td>
<td>91</td>
<td>84</td>
<td>13</td>
<td>99</td>
<td>92</td>
</tr>
<tr>
<td>CarDark</td>
<td>100</td>
<td>32</td>
<td>14</td>
<td>1</td>
<td>35</td>
<td>69</td>
<td>67</td>
<td>100</td>
<td>53</td>
</tr>
<tr>
<td>David</td>
<td>34</td>
<td>97</td>
<td>100</td>
<td>95</td>
<td>99</td>
<td>100</td>
<td>55</td>
<td>36</td>
<td>97</td>
</tr>
<tr>
<td>David2</td>
<td>100</td>
<td>42</td>
<td>100</td>
<td>0</td>
<td>83</td>
<td>89</td>
<td>8</td>
<td>100</td>
<td>81</td>
</tr>
<tr>
<td>David3</td>
<td>100</td>
<td>19</td>
<td>21</td>
<td>51</td>
<td>100</td>
<td>45</td>
<td>51</td>
<td>34</td>
<td>51</td>
</tr>
<tr>
<td>Deer</td>
<td>51</td>
<td>7</td>
<td>77</td>
<td>14</td>
<td>14</td>
<td>43</td>
<td>36</td>
<td>100</td>
<td>77</td>
</tr>
<tr>
<td>Dudek</td>
<td>98</td>
<td>100</td>
<td>100</td>
<td>97</td>
<td>100</td>
<td>55</td>
<td>59</td>
<td>100</td>
<td>83</td>
</tr>
<tr>
<td>Facecoc1</td>
<td>100</td>
<td>100</td>
<td>98</td>
<td>100</td>
<td>100</td>
<td>66</td>
<td>10</td>
<td>100</td>
<td>64</td>
</tr>
<tr>
<td>Fish</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>97</td>
<td>34</td>
<td>74</td>
<td>74</td>
<td>100</td>
<td>61</td>
</tr>
<tr>
<td>Football</td>
<td>99</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>100</td>
<td>17</td>
<td>78</td>
<td>75</td>
</tr>
<tr>
<td>Girl</td>
<td>63</td>
<td>83</td>
<td>83</td>
<td>59</td>
<td>64</td>
<td>99</td>
<td>26</td>
<td>100</td>
<td>35</td>
</tr>
<tr>
<td>Mhyang</td>
<td>100</td>
<td>100</td>
<td>79</td>
<td>100</td>
<td>84</td>
<td>100</td>
<td>3</td>
<td>100</td>
<td>85</td>
</tr>
<tr>
<td>Singer1</td>
<td>100</td>
<td>100</td>
<td>99</td>
<td>100</td>
<td>100</td>
<td>49</td>
<td>100</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td>Singer2</td>
<td>100</td>
<td>85</td>
<td>70</td>
<td>100</td>
<td>51</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Tiger2</td>
<td>85</td>
<td>24</td>
<td>29</td>
<td>84</td>
<td>64</td>
<td>33</td>
<td>59</td>
<td>88</td>
<td>15</td>
</tr>
<tr>
<td>Walking</td>
<td>98</td>
<td>21</td>
<td>11</td>
<td>100</td>
<td>100</td>
<td>68</td>
<td>88</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>Average</td>
<td>89</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>79</td>
<td>68</td>
<td>39</td>
<td>84</td>
<td>63</td>
</tr>
</tbody>
</table>
Fig. 4 shows the memory requirements of different methods. Classifiers in which the Haar-like feature is used as the feature representation need at least $10^5$ bytes to store the integral image for an image of size $320 \times 240$. When the number of pixel pairs in a fern is set to $N_f = 8$, the standard random ferns method needs memory as many as that of an...
integral image. For $N_f = 12$, the standard random ferns method needs memory more than 1 MB. The proposed method requires a constant and much lower memory than other methods (400 Bytes).

5.4.3. Computing resource usage analysis

The standard random ferns method can be computed very efficiently, in which the binary features come from the comparisons of pixel values. In the proposed method,
the real value features come from the subtraction of pixel intensity values, which is also very efficient. To confirm this conclusion, a supplementary experiment is conducted. The comparison and subtraction statements are both looped for 999,999,999 times and the time consumed by them is 2573 ms and 2592 ms respectively. Thus the proposed subtraction is as efficient as the comparison in the standard random ferns method.

**Training stage:** To train a classifier with \( K \) training samples (containing both foreground and background), the standard random ferns method learns a class conditional probability density function \( p(F_i = x(c)) \) by (12), which needs \( 2^{N_F+1} \) floating point multiplications and \( K \) additions for the counting of frequency. In the proposed method, (14) is used to learn the classifier. The calculation of parameter \( \mu_i^c \) needs 2 floating point multiplications and \( K-2 \) additions. The calculation of \( \sigma_i^c \) needs \( K+6 \) floating point multiplications and \( K \) additions. In the experiment, 512 floating point multiplications and 150 additions are needed for the standard random ferns method whereas 156 floating point multiplications and 298 additions are needed for the proposed method.

**Testing stage:** In the testing stage, the classifier is required to classify different image paths. The standard random ferns method is very efficient since it only needs to get the probability from a look up table. In order to avoid product of many small numbers, which would cause numerical problems, logarithm calculation is usually applied to transform the product into summation. In the experiment, this would require \( 2N_F \) - 100 logarithm calculations and \( 2N_F-2 \) = 98 additions. The proposed method needs to calculate (13) and (16). However, by plugging (13) into (16), it can be simplified as

\[
S(F) \propto - \sum_{i=1}^{N_i} \left( \frac{F_i - \mu_i^{bg}}{\sigma_i^{bg}} \right)^2 \log \sigma_i^{bg} + \sum_{i=1}^{N_i} \left( \frac{F_i - \mu_i^{fg}}{\sigma_i^{fg}} \right)^2 \log \sigma_i^{fg}.
\]

For a testing sample, the proposed method requires \( 6N_F \) floating point multiplications, \( 4N_F-1 \) additions and \( 2N_F \) logarithm calculations. In the experiment, 300, 199 and 100 floating point multiplications, additions and logarithm calculations are needed respectively.

**Update stage:** In the update stage, the standard random ferns method needs to apply an IIR-like filter to the previous frequency to avoid saturation effects, which needs another \( 2^{N_F} \) floating point multiplications for each class and \( 2^{N_F} \) floating point multiplications for the update of the posterior. In the experiment, \( 3 \times 2^{N_F} = 768 \) floating point multiplications are needed. The proposed method needs only 4 floating point multiplications and 4 additions (for constant \( \lambda \), this could be reduced to 2) for each class by (15). In total, the standard random ferns method needs 1280 floating point multiplications, 248 additions and 100 logarithm calculations. The proposed method needs 464 floating point multiplications, 502 additions and 100 logarithm calculations. Thus the proposed method is more efficient than the standard random ferns method. When there exist a large number of test samples, the proposed method becomes less efficient. But the proposed method can also perform more efficiently than the standard random ferns method as shown in Table 5. This can be explained by the fact that the proposed method needs less pixel value differences, which would cause memory access. Memory access is more time consuming in a computer program compared with the executing of CPU instructions. From Table 5, it can also be found that the proposed method runs over 100 FPS on all video sequences and faster than all other trackers. The second fastest tracker is RF8, due to the simple calculation of probability. Struck performs very well in accuracy, but it is too slow for a real time online tracking application. The proposed method runs much more faster than Struck while the accuracy is comparable with it.

6. Conclusion

In this paper, we have proposed a novel method based on random ferns and random projection for visual tracking. The proposed method is very simple and requires very

---

3 The experiment is implemented in C++ and runs in a debug mode with default optimization options. The running time is measured with the standard C++ function `clock()`.
few memory and computing resources, thus is very appropriate for embedded systems or high frame rate occasions. Though the proposed method is simple, experimental results show that it performs better than the standard random ferns method and some other methods proposed in the literature on most video sequences in accuracy, efficiency, memory requirement and computational complexity.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant no. 61350110239, the Fundamental Research Funds for the Central Universities under Grant no. BDY191420, and the Open Research Funds of State Key Lab. for novel software technology under Grant no. KFKT2012B16.

References