A Long-term Tracking Model Based on Tracking Failure Detection Strategy and Weighted Random Forest

Tao Zhu, Jun Chu, Jun Miao

Institute of Computer Vision, School of Software, Nanchang Hangkong University, FengHe South Road No.696, Nanchang City, Jiangxi Province, China

*992383884@qq.com

KeyWords: long-term; tracking; multi-scale; weighted; failure

Abstract: Compared to traditional visual tracking, long-term tracking appears to be more challenging since the target is likely to suffer more severe deformation, occlusion, scale change or move out of view scenarios. It is challenging to develop a robust and efficient target model. In this paper, we propose a robust model for long-term tracking in complex scenes. In order to achieve this goal, firstly, we extract multi-scale feature based on the illumination invariant color space to solve scale and illumination change of the target. For the purpose of reducing time consumption caused by the multi-scale feature, we adopt a random measurement matrix to project the high-dimensional multi-scale features onto a low-dimensional subspace. Secondly, we introduce a tracking Failure Detection Strategy (FDS) to decide whether the tracking is a failure which cause by occlusion, illumination change and situations when the target moves out of camera view. Finally, we proposed a Weighted Random Forest (WRF) classifier to retrieve the target position after the tracking failure situation, and the classifier is updated online, so that the performance of the model improves over time. Our proposed model performs favorably in complex scenes against conventional models in terms of robustness and time consumption.

1 Introduction

Despite its importance in computer vision, visual tracking faces many challenges, such as deformation, occlusion, illumination etc. Creative target models have proven to be effective in addressing these concerns, utilizing discrimination features to create target models and improve the performance by updating these models over the tracking sequence. Numerous online modeling methods have been proposed that can locate targets accurately\cite{1-7}. However, conventional online modeling methods are usually time consuming, prohibiting real-time application. There are also many algorithms that have been proposed to achieve the purpose of real-time\cite{7-12} tracking, but those methods sacrifice target tracking performance. There is often a trade-off between speed and robustness.

Different online modeling methods have been adopted based on different situations. Zdenek Kalal et al.\cite{1} utilize robust object detectors to enhance the performance of the online models. Allan D. Jepson et al.\cite{2} use robust online appearance models that mix stable image structures to improve the performance of the algorithm. The methods utilized in Ref.\cite{5} combine online models with continuous confidence of pedestrian detectors for human target tracking. Helmut Grabner et al.\cite{6} apply the “semi-supervised boosting” method to the online models to handle drifting problems. During tracking, partial occlusion is likely to occur and causes significant difficulty. Yang Hua et al.\cite{13} proposed a motion-reasoning method that is based on labeled object trajectories to solve the occlusion issue. The face-TLD (tracking-learning-detection)\cite{3} algorithm responds well to some
extended occlusion and the strategy is able to identify the tracking failures caused by fast occlusion or full occlusion during the tracking process. However, almost all the online target models of complex scenes face a critical issue: The training set that includes incorrect tracking results is used to update the online models. Errors then accumulate over time and eventually lead to a fatal failure in long-term tracking. Therefore, we need a strategy to detect tracking failure situation so that we can prevent the incorrect tracking result from degrading the target model when there are used to update the classifier.

In order to retrieve the target position after the tracking failure situation, an effective classifier is required to detect the target. Since the target is likely to change its scale during tracking, multi-scale is usually extracted to represent the target, and this will obviously increase the computational complexity, making it difficult for the algorithm to run in real-time. To reduce the computational load as possible, Martin Danelljan el al.[8] proposed a low-dimensional adaptive color attribute. Chen Qian el al. [11] modeled the target using spheres and a fast cost function, which made the algorithm fast enough for real-time tracking. In recent years, many new methods have been proposed. For instance, the works in [9,12,15,16] use Compressive Sensing[9] and Sparse Representation[17] for dimensionality reduction. These methods randomly project high-dimensional features onto a low-dimensional subspace field. Compared to conventional approaches like the PCA (principal component analysis), the random projection is less time consumption[15], as shown in Reference [12]. By utilizing the random projections that preserve the image feature structure to project the high-dimensional feature space onto a low-dimensional subspace to reduce the time consumption of target tracking.

In this paper, we propose an long-time tracking model based on tracking Failure Detection Strategy and Weighted Random Forest. First, we use the a sparse random measurement matrix that we pre-calculated to reduce the dimensionality of the multi-scale feature space that we acquired for the image frame. Since this matrix is calculated before modeling the target and data-independent, so the training samples cropped from the previous frames are not required. Considering that the process of tracking may encounter tracking failures caused by occlusion or illumination change etc., we improve our model by introducing a Failure Detection Strategy(FDS). By calculating the Forward-Backward Error[18] of the feature points while tracking, our model is able to identify the tracking failure situation. In order to retrieve the target position of the next frame we proposed a Weighted Random Forest(WRF) classifier to detect the target. The results of the experiment show that our algorithm performs favorably in long-term tracking and runs almost in real-time.

2 Preliminaries

2.1 Random projection. The high-dimensional space is projected to a lower-dimensional subspace by using a random projection, therefore the m-dimensional data is projected to an n-dimensional subspace where $n \ll m$

$$v = Rx$$

(1)

According to the Johnson-Lindenstrauss lemma[20], the distances between the points are preserved with high probability if the points in a vector space are randomly projected onto a subspace with suitably high dimensions. In Ref. [20], the random matrix is proved to satisfy the Johnson-Lindenstrauss lemma and the restricted isometric property of compressive sensing. For instance, suppose there is a random matrix $R$ that satisfies the Johnson-Lindenstrauss lemma. Then, the lower-dimensional vector $v$ can be used to reconstruct the original high-dimensional
vector $x$ ($x$ is a compressive piece information such as an image) with minimum error and high probability. Subsequently, the reconstructed $x$ can preserve almost all the information of the original vector. Therefore, the random projection theory can be used to convert the original high-dimensional feature to a low-dimensional feature and still keep almost all the feature information.

2.2 Random Measurement Matrix. The choice of the matrix $R$ is critical. The values of $r_{ij}$ are the elements of the matrix $R$, which are Gaussian distributed. However, the Gaussian distribution is not suitable for this case because of the high density of the Gaussian matrix. Instead, a very sparse measurement matrix is adopted in this paper.

$$r_{ij} = \sqrt{s} \times \begin{cases} 1 & \text{ with probability } \frac{1}{2s} \\ 0 & \text{ with probability } \frac{1}{s} \\ -1 & \text{ with probability } \frac{1}{2s} \end{cases}$$

(2)

This matrix $R \in \mathbb{R}^{m \times n}$ can satisfy the Johnson-Lindenstrauss lemma when $s=2$ or $3$. This will make the matrix very sparse, so it can be easily implemented by a uniform random generator. When we choose $s=3$, the computation will reduce to $1/3$ as the probability is $2/3$ when $r_{ij} = 0$, which is significant. Our algorithm sets $s = m/4$, and the random projections are almost the same accuracy as the conventional random projections where $r_{ij} \sim N(0,1)$.[12] For each row of the matrix $R$, there are only $c$ entries ($c \leq 4$) that need to be calculated which means the computational complexity is only $O(cn)$ instead of $O(mn)$. Therefore, the randomly generated matrix is extremely simple and easy to compute.

3 Proposed Algorithm

3.1 Model Framework. In this section, the framework of our tracking algorithm is in Figure 1. Our proposed model uses the initial frame that contains the manually selected target area (as shown in the red rectangular area in Figure 1) to generate the required positive and negative samples. Both positive samples and negative samples are extracted by scanning the grids of the initial frame and then these samples are converted into multi-scale samples by using the filter bank. The multi-scale samples are utilized to generate high-dimensional, multi-scale features that require large computation. In order to reduce the computation time, a sparse random measurement matrix is adopted to compress the high-dimensional features, and then the low-dimensional features are utilized to initialize the cascaded classifier of the online models. The results of the successful tracking and detection are used to update the classifier and make the online models more robust for future tracking. We add a tracking Failure Detection Strategy (FDS) to the online models that allows our models to deal with failures related to full occlusion and targets moving out of view.
3.2 Feature Representation. In complex scenes, the illumination change may affect the tracking process and eventually lead to a tracking failure. In order to deal with the illumination variation problem, we introduce a illumination invariant $YC_rC_b$ color space to our target model, the $Y$ component represents the intensity value, the $C_r, C_b$ components represent the different from the red component and the blue component of the $RGB$ color space to the intensity. There is not direct relevancy between the $Y$ and the $C_r, C_b$ components, therefore we have illumination invariant color space in $C_r$ and $C_b$ components. The computational process from $RGB$ to $YC_rC_b$ color space is defined as

$$
\begin{align*}
Y &= 0.299*R + 0.587*G + 0.114*B \\
C_r &= (R-Y)*0.713 + 128 \\
C_b &= (B-Y)*0.564 + 128
\end{align*}
$$

Multi-scale Feature: the features extracted from the frames are converted to multi-scale features by convolving a multi-scale filter bank $\{f_{i,j}\}$, each filter is defined as

$$
f_{i,j}(x, y) = \begin{cases} 
1, & 1 \leq x \leq i, 1 \leq y \leq j \\
0, & \text{otherwise}
\end{cases}
$$

For each input image frame $F \in \mathbb{R}^{wh}$, the $i$ and $j$ represent the width and the height of a filter from the filter bank, respectively. Then, the filtered frames are represented by vectors in $\mathbb{R}^m$ where $m = (wh)^2$ and then we put all the vectors of the filtered frames in one high-dimensional vector $X=(x_1, x_2, \cdots, x_m)^T \in \mathbb{R}^m$, the dimensionality of the vector can be up to $10^{16}$ which means the computational complexity is extremely high. Therefore, a very sparse measurement matrix $R$ in (2)
with $s = m / 4$ is pre-computed and stay fixed throughout the whole tracking process and then adopted to project the vector $X$ onto a low-dimensional subspace $V \in \mathbb{R}^n$ where $n \ll m$, which means the vector is very simple and easy to compute as we shown in Figure 2. Only the nonzero elements in $R$ need to be stored and the nonzero elements in each row are corresponding to the locations of the rectangles in the image frame. Then we have a compressed vector $V$ which is computed by using the sparse matrix to measure the feature in the rectangles randomly.

$$v_i = \sum_j r_j x_j$$

Figure 2. shows the specific steps of compressing the high-dimensional vector to a lower dimensional vector.

### 3.3 Weighted Random Forest (WRF) Classifier

In this paper, the initial data that have been used to train the classifier are generated by using the first frame and then the trained classifier is used to classify the detected patches of the next frame to locate the new position of the target. The classifier that we used to detect the target is constructed by combining $n$ random trees (weak classifier). In our classifier each random tree $T_k$ is composed of a Haar-like feature $F_k$. We represent each input frame as a randomly generated vector of Haar-like features. Each Haar-like feature consists of 2 to 4 rectangles, and each rectangle has a real valued weight. The feature value is computed by summing the pixels in all the rectangles. Then we use the integral image trick [14] to compute the pixels value of the rectangles efficiently.

**Decision Tree:** Each decision tree in the random forest can be consider as a weak classifier, we propose a Weighted Random Forest (WRF) to improved the performance of the weak classifier based on the theory of Adaboost. The WRF classifier performs better than the traditional random forest classifier since we use the classification results to update the weight and make the classifier more robust.

**Selector:** In each selector $H^{\text{weak}} = \{h_1^{\text{weak}}, \ldots, h_M^{\text{weak}}\}$, there are $M$ weak classifiers (decision tree) and the selector selects the weak classifier $h^{\text{w},m}(x) = h_m^{\text{weak}}(x)$, based on the Minimum Wrong Rate (MWR) which means the selector is able to choose the best weak classifier. The selector is a cluster of $M$ weak classifiers, in the cluster every weak classifier is updated during the training process and then we have a weak classifier with the Minimum Wrong Rate (MER).
The construction of the Weight Random Tree is shown in Figure 3.

As shown in Figure 3. Firstly, we have \( N \) selectors defined as \( h_{1}^{sel}, \ldots, h_{n}^{sel} \) and each selector contains \( M \) decision trees. When we input the training sample \( <x, y> \) to train all the selectors with an initial weight \( \lambda \). The decision tree with the MWR is selected to represent the selector, defined as:

\[
\text{arg min}_{m}(e_{n,m}), e_{n,m} = \frac{\lambda_{n,m}^{\text{wrong}}}{\lambda_{n,m}^{\text{wrong}} + \lambda_{n,m}^{\text{correct}}}
\]  

(5)

the \( n \) and \( m \) of the \( e_{n,m} \) represent the number of the selector and the decision tree respectively, the \( \lambda_{n,m}^{\text{correct}} \) and \( \lambda_{n,m}^{\text{wrong}} \) the represent the number of the correct and wrong decisions that the decision trees have made. After the decision tree with MER is selected, the weight of the selector \( \alpha_{n} \) and the weight of the sample \( \lambda \) are then updated and passed to the next selector. There is our Weighted Random Forest(WRF) when all the selectors are trained, the WRF is defined as:

\[
h_{\text{strong}}(x) = \text{sign} \left( \sum_{n=1}^{N} \alpha_{n} \cdot h_{n}^{sel}(x) \right)
\]

(6)

3.4 Failure Detection Strategy. During target tracking process, tracking failures are tend to occur when the target is occluded or moved out of the view of the camera (shown in Figure 4). In this paper, we adopted a Failure Detection Strategy(FDS) to decide whether the tracking is a failure. This strategy is implemented by computing the Forward-Backward error\[^{18}\] of the consecutive frames. In order to handle the illumination change, the feature points are evenly selected from the current frame in invariant \( YC_rC_b \) color space.
Forward-Backward error: We set $S = (I_1, I_{t+1}, \ldots, I_{t+m})$ to be the image frame and $X_t$ is the location of a point in the $t$-th frame. A median-flow tracker is used to track forward for $m$ steps (as shown in Figure 5). Therefore we have a forward trajectory $T^m_f = (X_t, X_{t+1}, \ldots, X_{t+m})$, and subsequently track the point backward with an opposite trajectory $T^m_b = (X'_{t+m}, X'_{t+m-1}, \ldots, X'_t)$, where $X_t$ and $X'_t$ represent the same point. According to (7), we define the tracking to be a failure if the median value of $\left| \frac{FB^i_{error}}{FB^{median}_{error}} \right|$ is larger than a threshold $\delta$ (in this paper $\delta = 10$ pixels), $FB^i_{error}$ represents the Forward-Backward error of one single point, $FB^{median}_{error}$ represents the median value of Forward-Backward error of all points.

$$FB_{error} = \tan \text{dis}ce(T^m_f, T^m_b) = ||X'_{t+m} - X_{t+m}||$$

4 Experiments

Experiments are implemented to demonstrate the performance of our algorithm in terms of robustness, long-term tracking and real-time capability. Our algorithm is compared with different methods through the benchmark video sequences that are generally applied in the literature that have occluded targets, as well as targets that move in and out of camera view.

The location of the target that needs to be tracked is manually selected in the initial frame. The configuration is as follows. The minimum size of the initial bounding box of the target is $15 \times 15 (\text{pixels})$, which indicates the size of the manually labeled target cannot be smaller than this minimum. In order to guarantee the performance of the experiment, the number of the random trees used in our system is 10. Our algorithm is implemented in C++, and on an Intel (R) C 3.2GHz 4.0
GB it runs at around 20fps. The sizes of the testing sequences are all 320×240. System performance is evaluated on standard sequences, including the sequences “David”, “Jumping”, “Pedestrian 1”, “Pedestrian 2”, “Pedestrian 3” and “Carchase”.

4.1 Occlusion and out-of-view scenarios. The Figure.6 illustrates 4 frames of experimental results of 3 different representative methods in three columns. These methods are CT(Compressive Tracking), CoGDT(Co-trained Generative and Discriminative Trackers) and our method, respectively. The target is manually selected in frame #0, with the target continuously moving and finally occluded in frame #37. The CT method comes to a tracking failure while the CoGDT method and our method are able to identify this tracking failure. When the target moves out of the occluded area our method and the CoGDT method are able to detect the position of the target precisely as shown in frame #39. The comparison of the experiments indicates that our method and the CoGDT method can solve the full occlusion effectively.

![Figure 6. The illustration of the target from tracked to fully occluded then tracked again.](image)

The comparison experiments are demonstrated in Figure.7 that show the process of the three algorithms in dealing with the target that moves out of the camera view. From frame #50 to frame #53, the target moved out of the camera view. The CT method tracked the wrong position and led to a tracking failure. When the target moved back to the view of the camera, the target remained lost. Our method tracked the most precise position of the target in frame #79.
4.2 Scale change of the target. The Figure 8 shows that the movement of the camera view changes the scale of target. Our online model is based on multi-scale features and is able to handle occlusion. This allows our algorithm to track multi-scale targets while partial occlusion exists at the same time. The CT and the CoGDT methods cannot track the complex situation properly, leading to a tracking failure in frame #61.
4.3 Comparative Analysis. The Table.I shows the results of our algorithm compared with other different methods (presented in Refs. [12],[24],[25],[26],[27] on 6 different video sequences. We assess the performance by calculating the number of the frames that have been successfully tracked. The target can only be successfully tracked in valid frames, because full occlusion and out-of-view situations exist in some frames. Our method scored 760, 313, 140, 238, 140, and 416 in the six sequences. These are significantly better than the results of the IVT(Incremental learning for visual tracking), CT (Compressive Tracking), ET(Ensemble Tracking) and MIL(Multiple Instance Learning) methods. The CoGDT(Co-trained Generative and Discriminative Trackers) method performs similar to our algorithm according to the data of Table I, however the processing speed is only 2 fps compared to 20 fps for our algorithm. Furthermore, CoGDT requires more than one initial frame for the initialization where our algorithm only requires one. This demonstrates the accuracy and speed of our algorithm.

Table I. The total number of successfully tracked frames in six different video sequences. Bold fonts indicate the best performance. Comparison of different methods IVT (Incremental learning for visual tracking)\cite{25}, CT(Compressive Tracking)\cite{12}, ET (Ensemble Tracking)\cite{24}, CoGDT (Co-trained Generative and Discriminative Trackers)\cite{26}, MIL (Multiple Instance Learning)\cite{27} and the method we proposed in this paper.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Total Frames</th>
<th>Occlusion</th>
<th>IVT</th>
<th>CT</th>
<th>ET</th>
<th>CoGDT</th>
<th>MIL</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
<td>761</td>
<td>0</td>
<td>17</td>
<td>761</td>
<td>94</td>
<td>759</td>
<td>135</td>
<td>760</td>
</tr>
<tr>
<td>Jumping</td>
<td>313</td>
<td>0</td>
<td>75</td>
<td>38</td>
<td>44</td>
<td>313</td>
<td>313</td>
<td>313</td>
</tr>
<tr>
<td>Pedestrian1</td>
<td>140</td>
<td>0</td>
<td>11</td>
<td>81</td>
<td>22</td>
<td>140</td>
<td>101</td>
<td>140</td>
</tr>
<tr>
<td>Pedestrian2</td>
<td>338</td>
<td>93</td>
<td>33</td>
<td>32</td>
<td>118</td>
<td>240</td>
<td>37</td>
<td>238</td>
</tr>
<tr>
<td>Pedestrian3</td>
<td>184</td>
<td>30</td>
<td>50</td>
<td>135</td>
<td>53</td>
<td>154</td>
<td>49</td>
<td>154</td>
</tr>
<tr>
<td>Carchase</td>
<td>630</td>
<td>212</td>
<td>166</td>
<td>358</td>
<td>39</td>
<td>390</td>
<td>167</td>
<td>416</td>
</tr>
</tbody>
</table>

Table II. Success rate of the valid frames that have been tracked (SR)(%)\[score = \frac{\text{area}(ROI_t \cap ROI_g)}{\text{area}(ROI_t \cup ROI_g)}\], where the \(ROI_t\) is the tracking box and the \(ROI_g\) is the ground truth box. The tracking result is considered as a success if the value of the score is larger than the Threshold (we set the Threshold = 0.5). Bold fonts indicate the best performance. Table.II shows the percentage of the successful tracked frames, where the highest score indicates the best performance. Our method scored highest in 4 video sequences. The average SR, IVT, CT, ET, and MIL methods scored less than 60 in these instances, which means these methods were not able to solve the full occlusion or out-of-camera-view situations properly. Our method performed comparably or better than the CoGDT method in both single video sequence SR and average SR. However, our method has the running-speed advantage over the CoGDT method.
<table>
<thead>
<tr>
<th>Sequence</th>
<th>Valid Frames</th>
<th>IVT</th>
<th>CT</th>
<th>ET</th>
<th>CoGDT</th>
<th>MIL</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
<td>761</td>
<td>2.24</td>
<td>100</td>
<td>12.35</td>
<td>99.74</td>
<td>11.74</td>
<td>99.97</td>
</tr>
<tr>
<td>Jumping</td>
<td>313</td>
<td>23.97</td>
<td>12.14</td>
<td>14.06</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Pedestrian1</td>
<td>140</td>
<td>7.86</td>
<td>57.86</td>
<td>15.71</td>
<td>100</td>
<td>72.14</td>
<td>100</td>
</tr>
<tr>
<td>Pedestrian2</td>
<td>245</td>
<td>9.80</td>
<td>13.06</td>
<td>48.16</td>
<td>97.96</td>
<td>15.10</td>
<td>97.14</td>
</tr>
<tr>
<td>Pedestrian3</td>
<td>154</td>
<td>32.47</td>
<td>87.67</td>
<td>34.42</td>
<td>100</td>
<td>31.82</td>
<td>100</td>
</tr>
<tr>
<td>Carchase</td>
<td>418</td>
<td>39.71</td>
<td>85.65</td>
<td>9.33</td>
<td>93.30</td>
<td>39.95</td>
<td>99.52</td>
</tr>
<tr>
<td>AverageSR</td>
<td>--</td>
<td>19.34</td>
<td>59.40</td>
<td>22.39</td>
<td>98.50</td>
<td>45.13</td>
<td>99.44</td>
</tr>
</tbody>
</table>

5 Conclusion

In this paper, we propose a robust and long-term tracking model based on the tracking Failure Detection Strategy (FDS) and Weighted Random Forest (WRF). We use the multi-scale feature to handle the target scale change, and the illumination invariant $YC,C_b$ color space that we adopted is able to solve illumination change problem. The multi-scale feature is compressed by using a sparse random measurement matrix to reduce the time consumption, and then applied to train an online updated classifier. The classifier is based on a Weight Random Forest classifier that we proposed to retrieve the target location. Then, a tracking Failure Detection Strategy (FDS) is introduced to keep our model from tracking the incorrect target. By comparing the experiments results that we presented, we conclude that our method performs well in long-term tracking in complex scenes, and the experiments show that our algorithm runs almost in real-time.

Acknowledgment

This research was supported by the National Natural Science Foundation of China (No.61263046,61462065), Natural Science Foundation of Jiangxi Province (No.20122BAB201037) and the Innovation Fund Designated for Graduate Students of Jiangxi Province (No.YC2014-S408).

References


