Real-time Visual Tracking

EECS451 Final Project

Chengcan Ye
EECS, University of Michigan
Ann Arbor, Michigan, U.S.
yeccan@umich.edu

***Abstract*—**Visual tracking is one of the most challenging problems in computer vision. Generally speaking, there are mainly two state-of-the-art methods of visual tracking: exhaustive search and neighborhood search. The former one generates candidate bounding boxes of different size and realizes tracking via object detection. The latter one tracks the object through finding the maximum response near the region of last tracked result. This paper uses the method of neighborhood search to achieve the goal of real-time tracking. By adopting the color attributes, this paper has shown its superior performance over multiple datasets.

Keywords—visual tracking; color attributes.

I. Introduction

Visual object tracking, where the objective is to estimate the locations of a target in an image sequence, is one of the most challenging problems in computer vision. It has important applications in multiple fields, especially in human-computer interaction, surveillance, and robotics. Similar to many other problems in computer vision, visual tracking can be affected by factors such as illumination variance, scale variance, occlusions and background clutter.

Most state-of-the-art trackers rely on exhaustive search [1, 2, 3, 4]. Given an initial object to be tracked, either selected manually or acknowledged from ground truth, a model of the object is learned. During the process of tracking, multiple bounding boxes of different size, angle and shape are generated, and then compared with the model in first step. The box with the highest similarity thus be chosen as the tracking result. Exhaustive search has shown its robustness over multiple criterions such as scale variance and occlusions. However, this method calls for expensive computation and memory costs, especially when the model of object is complicated.

Another popular method of object tracking is neighborhood search [5, 6, 7, 8]. Given an initial object or the tracking result of last frame, a model is learned over the local region near the object. During the process of tracking, maximum response of corresponding features will be chosen as tracking result. Intensity or texture information, and color features are state-of-the-art features used. Though neighborhood search does not perform as robust as exhaustive search, especially when heavy occlusions and out-of-frame situation happens, it is superior over computation and memory costs, which makes it competitive over real-time visual tracking.

A neighborhood search method, CSK tracker, is shown to provide high speed among top ten visual trackers. This paper will use the idea of CSK tracker, and adopt color attributes to improve the performance of it. An evaluation against multiple criterions will be presented to show this paper has achieved a good balance between speed and robustness.

II. Related works

A. The CSK Tracker

The CSK tracker [9] learns a kernelized least squares classifier of a target from a single image patch. CSK tracker exploits the circulant structure that appears from the periodic assumption of the local image patch.

A classifier is trained using an single image patch x of size $M \times N$ whose centroid is the target. The tracker considers all cyclic shifts $x_{m,n}, m \in \{0,1,\dots,M-1\}, n \in \{0,1,\dots,N-1\}$ as training examples, they are labelled with a two dimensional hamming window y, such that $y(m,n)$ is the label for $x_{m,n}$. Thus the further the example is away from the centroid, the lower value of the label it will be assigned. The classifier is trained by minimizing the least squares problem over w.

$$\epsilon = \sum_{m,n} |\langle \phi(x_{m,n}), w\rangle - y(m,n)|^2 + \lambda \langle w, w\rangle \qquad (1)$$

Here, ϕ is the mapping to the Hilbert space induced by the kernel κ, defining the inner-product $\langle \phi(x), \phi(y)\rangle = \kappa(x,y)$. λ is the regularization parameter. Equation (1) is minimized by $w = \sum_{m,n} a(m,n)\, \phi(x_{m,n})$, where the coefficients α are:

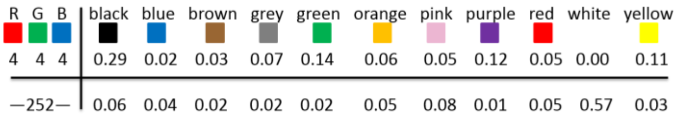
$$A = \mathcal{F}\{\alpha\} = \frac{Y}{U_x + \lambda} \qquad (2)$$

Here $\mathcal{F}$ is the notation of Discrete Fourier Transform(DFT). $Y = \mathcal{F}\{y\}$ and $U_x = \mathcal{F}\{u_x\}$, where $u_x(m,n) = \kappa(x_{m,n}, x)$ is the output of the kernel function. Equation (2) holds if and only if κ is shift invariant, thus Gaussian RBF kernel is chosen.

The detection step is performed by cropping out a patch z of size $M \times N$ in the new frame, the detection scores are calculated as $\hat{y} = \mathcal{F}^{-1}\{AU_z\}$, where $U_z = \mathcal{F}\{u_z\}$, and $u_z(m,n) = \kappa(z_{m,n}, \hat{x})$. Here $\hat{x}$ denotes the patch of target appearance, which will be updated frame by frame. The tracking result in the new frame will be the maximum response in $\hat{y}$.

B. *Color Attributes*

Recently, color attributes obtained excellent results for object recognition, object detection and action recognition. Color attributes [10] are linguistic color labels assigned by human to represent colors in the world. In a linguistic study by Berlin and Kay [11], it was concluded that English language contains eleven basic color terms: black, blue, brown, grey, green, orange, pink, purple, red, white and yellow. In the field of computer vision, color naming is an operation that associate RGB values with linguistic color labels. According to [10], a $32 \times 32 \times 32$ look-up table which maps RGB values to a probabilistic color representation is learned via Google Image. As seen in Fig. 1, each row of color names sums up to 1.



R	G	B	black	blue	brown	grey	green	orange	pink	purple	red	white	yellow
🟥	🟩	🟦	⬛	🟦	🟫	⬜	🟩	🟧	⬜	🟪	🟥	⬜	🟨
4	4	4	0.29	0.02	0.03	0.07	0.14	0.06	0.05	0.12	0.05	0.00	0.11
—252—			0.06	0.04	0.02	0.02	0.02	0.05	0.08	0.01	0.05	0.57	0.03

Fig. 1. Example of color names feature

Intuitively, RGB values with lower intensity have greater possibility over dark colors and vice versa.

Apart from the gray scale information used in CSK tracker, now we can combine color attributes as features to train the model. However, the dimension of color attributes is 11, which is computation expensive, thus we introduce PCA to compress the dimension of features.

C. *Principal Component Analysis*

PCA was invented by Karl Pearson [12], it is a statistical procedure the uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables, thus can be used to compress the dimension of features.

Given a $N \times p$ observation matrix X, each row x_i denotes an observation. First, we can extract the mean vector m using:

$$m = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3}$$

Then compute the covariance matrix C using:

$$C = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - m)^T(x_i - m) \tag{4}$$

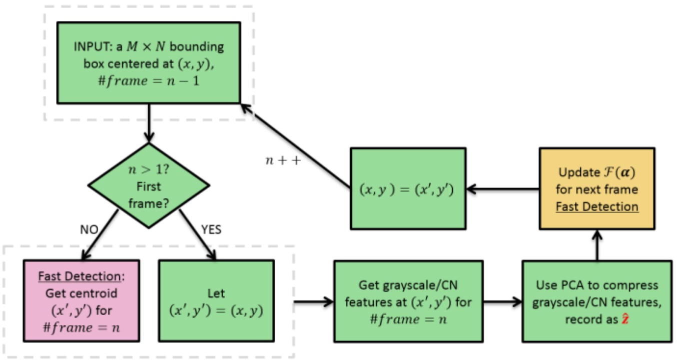
After that we can obtain the $p \times p$ eigenvector matrix of C via eigenvalue decomposition. Each eigenvector represents a principal component. By taking the first L columns, we will get a $p \times L$ matrix P, called projection matrix.

$$Y = XP \tag{5}$$

Using the above equation, we receive a compressed matrix Y which has N rows but only L columns. Such dimension reduction steps can be very useful in visualizing and processing high-dimensional datasets.

III. The Algorithm

The flow chart of the algorithm will be shown in Fig. 2, and the algorithm will be illustrated generally. Further details are presented in the following parts.



INPUT: a $M \times N$ bounding box centered at (x, y), $\#frame = n-1$
$n > 1$? First frame?
NO
YES
Fast Detection: Get centroid (x', y') for $\#frame = n$
Let $(x', y') = (x, y)$
Get grayscale/CN features at (x', y') for $\#frame = n$
Use PCA to compress grayscale/CN features, record as $\hat{z}$
Update $\mathcal{F}(\boldsymbol{\alpha})$ for next frame Fast Detection
$(x, y) = (x', y')$
$n++$

Fig. 2. Flow chart of the algorithm

Given the target's coordinates of frame $n-1$, the goal is to find the target's coordinates of frame n. If $n = 1$, we will select the target manually or get the target's information from ground truth. Otherwise, we need to compute the coordinates through Fast Detection, which has been introduced in Part II.A.

After obtaining the target's coordinates, we need to update the model of the target. Specifically, updating the coefficients α in Part II.A. First, cropping out a patch near the target's centroid (x', y'), take the pixels within it as examples. Second, we compute the gray scale feature and color attributes as introduced in Part II.B. The next step is to compress the dimension of the features extracted using PCA, and record the appearance as $\hat{z}$. After that, the coefficients α can be updated by combining the new appearance with the old observation. The visual tracking algorithm operates by doing this frame by frame. The procedure of compressing feature dimension and updating coefficients α will be introduced in detail.

A. *Feature Dimension Compression*

As introduced before, we will use PCA to compress feature dimension. Take the examples in a patch as input, a projection matrix needs to be computed. However, since the appearance of the target in each frame is not independent, we developed a strategy to update the projection matrix frame by frame, rather than create a brand new projection matrix. The procedure is summarized in Algorithm 1.

Algorithm 1 Projection Matrix Computation

```
Input:
    Frame number $n$; Object appearance $\hat{z}^n$
    Previous covariance matrix $Q_{n-1}$; Parameter $\mu$
```

```
OutPut:
    Projection Matrix $P_n$; Covariance Matrix $Q_n$

1: Calculate covariance $C$ using $\hat{z}^n$
2: Set $R = (1-\mu)Q_{n-1} + \mu C$
3: Do Eigenvalue decomposition $R = ESE^T$
4: Take the first 2 column of $E$ as projection matrix $P_n$
5: Take the largest 2 singular values as diagonal components
   of $\Lambda$
6: Update covariance matrix $Q_n = (1-\mu)Q_{n-1} + P_n \Lambda P_n^T$
```

With the use of Algorithm 1, we do not need to store all the previous covariance matrices, but only the last covariance matrix using a learning parameter μ. Thus, the memory cost of the algorithm is greatly decreased, but the consistence of target's appearance is preserved.

B. *Coefficients α update*

To achieve visual tracking that is robust to appearance changes, it is necessary that the target model is updated frame by frame. As in Eq. (2), coefficients α are represented via DFT, and α are the closed form solution to Eq. (1). By updating α, we can update the solution to least squares problem in each frame, thus it is equivalent to update the target model. The procedure is summarized in Algorithm 2.

```
Algorithm 2 Coefficients $\alpha$ update
Input:
    Frame number $n$; Object appearance $\hat{z}^n$; Parameter $\gamma$
    DFT of previous $\alpha$, denotes $A_{n-1} = \frac{A_{n-1}^N}{A_{n-1}^D}$
OutPut:
    DFT of $\alpha$, called $A_n = \frac{A_n^N}{A_n^D}$

1: Calculate circulant kernel $\boldsymbol{\kappa}$ using $\hat{z}^n$
2: Build a 2-d hamming window $\boldsymbol{y}$
3: Set $\frac{A_n^N}{A_n^D} = \frac{\mathcal{F}(\boldsymbol{y})\odot\mathcal{F}(\boldsymbol{\kappa})}{\mathcal{F}(\boldsymbol{\kappa})\odot(\mathcal{F}(\boldsymbol{\kappa})+\lambda)}$, where $\mathcal{F}$ denotes DFT and $\odot$
   denotes point-wise production
4: Update $A_n^N = (1-\gamma)A_{n-1}^N + \gamma A_n^N$
5: Update $A_n^D = (1-\gamma)A_{n-1}^D + \gamma A_n^D$
6: Set $\mathcal{F}(\alpha) = A_n = \frac{A_n^N}{A_n^D}$
```

With the use of Algorithm 2, we only need to store the current model $\{A_n^N, A_n^{\boldsymbol{D}}, \hat{z}^n\}$. This also contributes to the high speed and low memory cost of the algorithm.

IV. EXPERIMENTS

The code is implemented in Python, *numpy, pylab, and matplotlib* library is needed. A public dataset[1] and a commercial dataset are used, both including ground truth information.

Due to the time limitations, control experiments are not designed. Thus, we will only discuss the superiority and inferiority of this paper.

A. *Tracking Results Format*

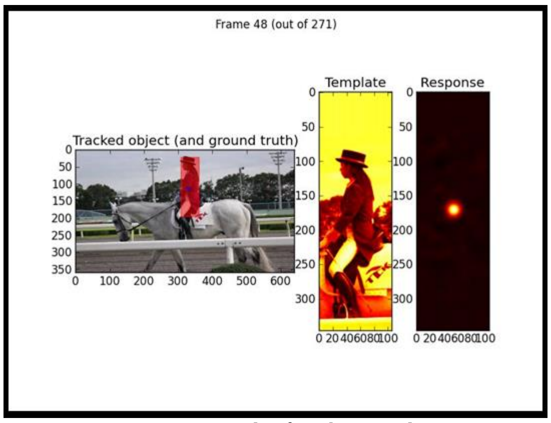


Fig. 3. Example of tracking result

As in Fig. 3, frame number is shown at the top of the figure. Within the leftmost figure, the red box denotes the tracking result, and the bot dot denotes the ground truth information. The middle figure shows the template used to extract grayscale feature and color attributes. The rightmost figure is the local response of the algorithm, and the brightest point denotes the target's coordinates.

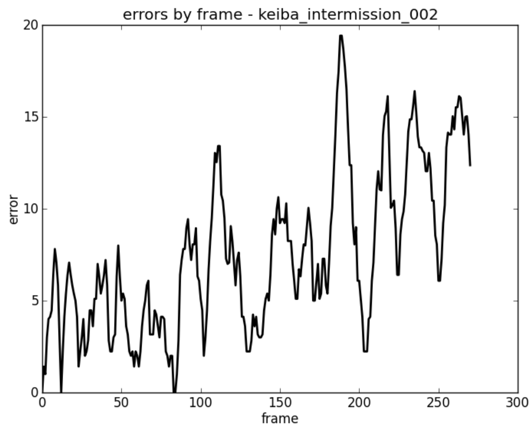


Fig. 4. Error by frame figure

[1] dataset available at: https://sites.google.com/site/trackerbenchmark /benchmarks/v10

Fig. 4 shows the error by frame figure. The horizontal axis denotes the frame number, and the vertical axis denotes the Euclidean distance between the tracking result and the ground truth. This figure is used to detect where this tracker lost its target.

B. *Tracking Results*

The following figure shows this tracker can track its target with high performance. Due to the paper's length limitation, not all the examples in the dataset will be presented, but videos of tracking results will be attached when submitting.

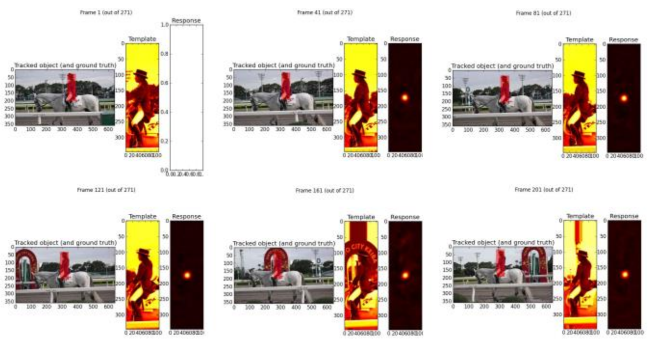


Fig. 5. Example of tracking results

From Fig. 5, we can see that this tracker has excellent performance, especially when the target does not has significant variance. However, in the real world, the environmental changes can affect the tracker greatly, thus we will test the tracker on more challenging sequences.

C. *Partial Occlusions*

Occlusion is one of the most important factor that can affect the performance of a tracker, especially for those using neighborhood search. However, this paper updates the model of the target frame by frame, which greatly offsets the effects of occlusions.

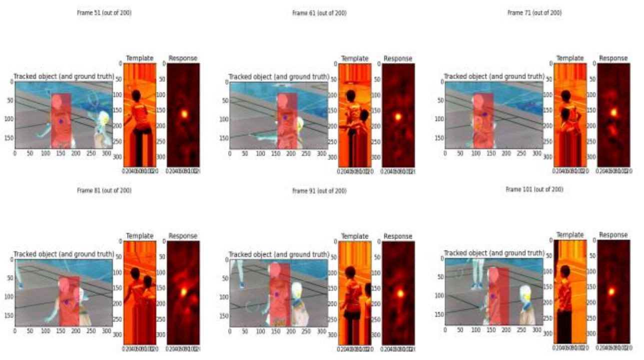


Fig. 6. Example of partial occlusions

For this example, there are heavy occlusions over the target from frame 60 through frame 90, however, the target is still well tracked. From the error by frame figure(not presented due to length limitations), the greatest error is less than 40 pixels. However, similar to most trackers, this tracker is not robust to full occlusions if lasts for a few frames, because it only saves the appearance of last frame to achieve the goal of real time.

D. *Scale Variance*

Scale variance, including size variance, shape variance and angle variance, is another important factor that affects a tracker. The following figure shows this tracker has little robustness when occuring size changes if the target becomes bigger and bigger. It is reasonable because we use neighborhood search, and only take the local region into consideration. Thus when the target becomes bigger, the new appearance is just a part of the target.

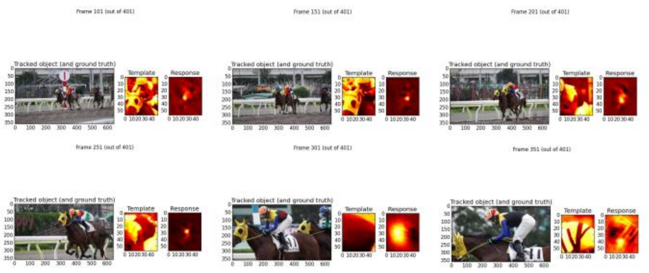


Fig. 7. Example of scale variance

In this example, the rider comes closer to the camera, thus the size is variant. The shape is also variant because of the sports of riding. From the error by frame figure(not presented due to length limitations), we can see the tracker lost its target around frame 150, where the target's size increases greatly. I tried to implement local exhaustive search, that is, generate multiple bounding box around the target but not within the whole image. However, it makes little differences when this kind of scale variance happens. Unfortunately, I will leave this problem for further research.

The good news is that the tracker is robust to scale variance if the target becomes smaller. Because during the process of shrinking, the target's appearance is always within the bounding box.

E. *Illumination Variance*

Unfortunately, the datastes do not contains examples with strong illumination changes. However, according to [9], the CSK tracker is brightness invariant. Since this paper is based on the idea of CSK tracker, it is highly prossible that this strength is inherited.

CONCLUSIONS

A real-time visual tracker is developed based on the CSK tracker. A 50+ fps speed is achieved using QVGA images. The tracker proves its robustness over a public dataset and a commercial dataset, and performs well against partial occlusion and illumination variance. Though, it has many works to do to improve its invariance against scale changes.

I used DFT (Part II.A), hamming window (Part II.A) and circulant convolution (Part II.A) as in-class tools. I used PCA (Part II.C), kernel skills (Part II.A) as out-class tools.

The coolest thing about this project is I learned a new programming language – Python. This language show its strength over data preprocessing and programming easiness. However, it is difficult to debug.

The most painful thing is dealing with the scale invariance of the algorithm. The problem is common to most visual tracker, after reading tons of paper, I realized a compromise between robustness and performance should be made.

References

[1] Fulkerson, Brian, Andrea Vedaldi, and Stefano Soatto. "Class segmentation and object localization with superpixel neighborhoods." *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009.

[2] Vijayanarasimhan, Sudheendra, and Kristen Grauman. "Efficient region search for object detection." *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011.

[3] Wang, Shu, et al. "Superpixel tracking." Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE, 2011.

[4] Kalal, Zdenek, Krystian Mikolajczyk, and Jiri Matas. "Tracking-learning-detection." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 34.7 (2012): 1409-1422.

[5] Toshev, Alexander, Ben Taskar, and Kostas Daniilidis. "Object detection via boundary structure segmentation." *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010.

[6] Jiang, Hao, and S. X. Yu. "Linear solution to scale and rotation invariant object matching." *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009.

[7] Hinterstoisser, Stefan, et al. "Gradient response maps for real-time detection of textureless objects." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 34.5 (2012): 876-888.

[8] Van de Sande, Koen EA, et al. "Segmentation as selective search for object recognition." *Computer Vision (ICCV), 2011 IEEE International* Conference on. IEEE, 2011.

[9] Henriques, João F., et al. "Exploiting the circulant structure of tracking-by-detection with kernels." Computer Vision–ECCV 2012. Springer Berlin Heidelberg, 2012. 702-715.

[10] Van De Weijer, Joost, et al. "Learning color names for real-world applications."Image Processing, IEEE Transactions on 18.7 (2009): 1512-1523.

[11] Berlin, Brent. Basic color terms: Their universality and evolution. Univ of California Press, 1991.

[12] Pearson, Karl. "LIII. On lines and planes of closest fit to systems of points in space." The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2.11 (1901): 559-572.