Improved Kernel Correlation Filter (KCF) for Visual Tracking

Student: Haolin Lyu

Supervisor: Jochen Lang

Course number: CSI6900

Abstract:

Visual object tracking is an important topic in computer vision and the topic has been extensively studied. Among various visual tracking methods, correlation filters for long-term visual object tracking have recently seen great interest. However, even though competitive results have come out in the past few years, tracking methods need more improvements to fit the latest requirements. In this project, we present an effective object tracking solution based on the Scalable Kernelized Correlation Filter (sKCF) framework. An adjustable rotated Gaussian window function and a rotation angle estimation based on the existing key-point matching strategy will be introduced to deal with rotation detection. Furthermore, we adapted more expressive and simpler deep learning features in this project to improve the tracking performance. We test our solution using the standard VOT toolkit and the VOT Challenge datasets. Our implementation will be compared with the original sKCF algorithm and the performance evaluation is based on overlap rate, failure time and running speed.
Key words: rotation estimation, deep learning features, kernelized correlation filter

Index

Glossary:
KCF: Kernelized Correlation Filter
sKCF: Scalable Kernelized Correlation Filter
srKCF Scalable Rotated Kernelized Correlation Filter
VOT: Visual Object Tracking
SVM: Support Vector Machine
FPS: Frame Per Second
FHOG: Felzenszwalb's Histogram of oriented Gradient

Introduction

Visual tracking is one of the fundamental research areas in computer vision with various applications such as video compression [10, 26], augmented reality [18], traffic control [6], surveillance and security [11]. Although some settings allow for strong assumptions about the target [2, 5, 8], in most cases, limited prior knowledge is the biggest challenge for a tracker. Model-free tracking requires online learning and adaption of a representation for a given target.

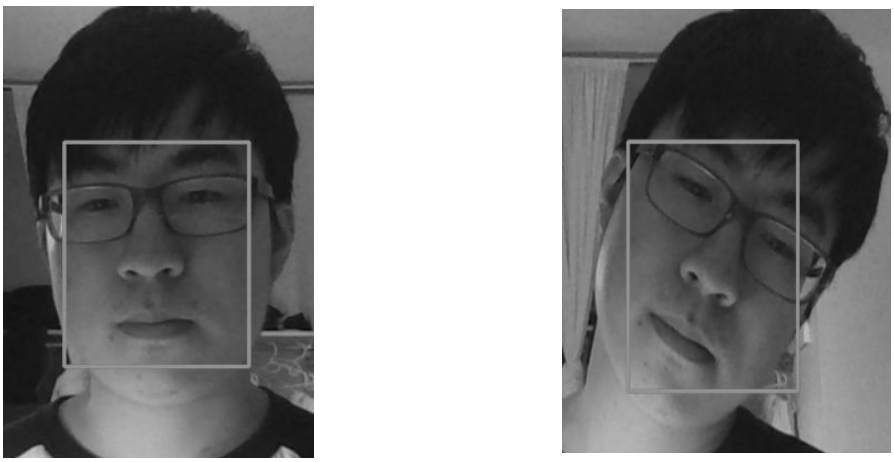
The tracker analyzes sequential video frames and after the initial detection of a target, outputs the successive positions of the object from frame to frame. A very successful approach has been tracking-by-detection [1, 4]. This approach directly makes use of some powerful discriminative methods in machine learning. Many of these machine learning algorithms can be used for online training. For every frame of a sequence, a successful detection can be used to update the model and be used to detect the next frame. Previously, many methods make use of a sparse sampling strategy [4, 15]. Basically, it randomly collects some neighbor samples from the target and builds the model. Since the number of sample is limited because of running speed requirements, these methods are neither fast enough nor accurate enough.

Unlike previous methods, Henriques et al. proposed a dense sampling strategy [3, 7]. He extended the correlation filter by exploiting the circulant structure of the tracking object to quickly incorporate information from all cyclically shifted samples into the Fourier analysis without iterating over all possible samples in the target's neighborhood in the time domain. Furthermore, Henriques et al. incorporated multiple feature channels instead of raw pixel into their correlation filter framework to improve the accuracy and robustness of the tracker.

Even with much improvement, their method uses a fixed size template and hence the tracker is not able to handle scale changes of a target occurring during motion. This means with the scale change of a target, the target window can't match with the target itself very well. After sometime, original background will become part of the target or part of the target will become background.

As a result Montero et al. proposed a Scalable Kernel Correlation Filter (sKCF) [12]. There are two main contributions of that method. First, it has the capability to handle scale change of a tracked target. Second, fast HOG feature also accelerates the tracker and improve the performance. However, there is still space for improvement.

On the one hand, target deformation is much more than just scale change, it may also include target rotation. On the other hand, traditional hand-crafted features such as FHOG are not expressive enough when faced with complex video sequences.



(a) (b)

Figure 1. This is a human face target, original sKCF is not able to handle target rotation angle change. After target rotates for a certain angle as shown in (b), part of target in (a) is out of white bounding box, so the features of the target will be not accurate anymore.

So the purpose of this project is to improve an existing scale-adaptable KCF for short-term visual object tracking. We will both add new functionality and improve the performance of the tracker. There are three major contributions of this project.

1. We extend the functionality of original sKCF tracker. With the rotated Gaussian window and existing key-point pair strategy, the rotation of a target can be handled in our tracker now.
2. We adapt deep learning features to replace traditional hand crafted FHOG features. Deep learning features are much more flexible and they also improve the performance of the tracker while they can still keep the real-time attribute of the tracker.
3. We also explore some potential ways to further improve the performance of the tracker. They include the Opencv+CUDA impact on our tracking speed, some attempts on better video segmentation and a way to deal with occlusion

In the following, we will first review the sKCF method in the second paragraph. Then the detailed major improvements of our tracker will be given. Next, we will list the experimental results based on the standard VOT toolkit with the VOT 2017 challenge dataset. We are going to discuss some potential way to further improve tracker performance. At last, the conclusion and acknowledgement will be given.

2. Scalable Kernel Correlation Filter

In this section, we will give a review of the scalable Kernelized Correlation Filter. sKCF is mainly based on KCF tracker.

2.1 Classifier equation

The tracker model can be considered as a high dimensional function $f(x) =< w, x > +b$. The input x is raw image or features generated from that image, the output will be the response matrix f(x). The largest response location is going to be the target position in a new frame. Once given a set of images and labels, the minimization

problem will be

$$\min_{w,b} \sum_{i=1}^{m} L\left(y_i, f(x_i)\right) + \gamma\|w\|^2 , \qquad \text{……..(1)}$$

where $L\left(y_i, f(x_i)\right)$ is hinge loss function and γ controls the amount of regulization.

We also know that kernel trick can usually make a model more expressive. The kernel $k(x, x) =< \varphi(x), \varphi(x) >$ helps variables to be mapped into high dimensional space. So the parameter w together with x can be transformed into another way: $w = \sum_i \alpha_i \varphi(x_i)$. Based on (1), the only model parameter α is

$$\alpha = (K + \gamma I)^{-1} y, \qquad (2)$$

where K is generated by all the input images X and y contains the labels of all the input images.

2.2 Circulant structure

The above learning model needs plenty of inputs to work but we only get one image. One solution is sparse random sampling, however, this usually makes tracker very slow and samples also can't make a good model. Instead, Henriques et al. use a dense sampling strategy. In other words, all the samples are generated from a circulant matrix. We assume the input image is u. Then the matrix C (u) will be

$$C(\mathbf{u}) = \begin{bmatrix} u_0 & u_1 & u_2 & \cdots & u_{n-1} \\ u_{n-1} & u_0 & u_1 & \cdots & u_{n-2} \\ u_{n-2} & u_{n-1} & u_0 & \cdots & u_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_1 & u_2 & u_3 & \cdots & u_0 \end{bmatrix}.$$

Actually, this matrix consists of all the input sample set and it is a dense sample. At the same time, because convolution of vectors can be computed in Fourier domain, if there is another vector v, then we can get

$$C(u)v = \mathcal{F}^{-1}(\mathcal{F}^{*}(u) \odot \mathcal{F}(v)), \qquad (3)$$

where $\odot$ is the element-wise product, while $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the Fourier transform and its inverse, respectively, and * is the complex-conjugate.

Obviously, the original samples X follow circulant structure, but we also need to look into the kernel matrix K.

Henriques et al. also prove that K follows circulant structure. As a result, we can get the equation $K = C(k)$. The small k can be calculated from only original real image. At last from (2) and (3), the model is shown as below

$$\alpha = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(y)}{\mathcal{F}(k)+\gamma}\right) \quad (4)$$

2.3 Kernel Computation

With the model, we can detect the target position based in a frame. We also need to know how we can calculate the kernel fast when we build the model. Henriques et al. give the formula and proof in their paper. There are different kinds of kernels but we usually use the most expressive Gaussian kernel.

$$k^{gaussian} = \exp\left(-\frac{1}{\sigma^2}\left(\|x\|^2 + \|x'\|^2 - 2\mathcal{F}^{-1}\left(\mathcal{F}(x)\odot\mathcal{F}^*(x')\right)\right)\right) \quad (5)$$

So that is how KCF tracks a target

2.4 Gaussian window

The Hann window proposed in KCF trcker is a fixed size filter. It is generated from trigonometric function so when a target changes its size, Hann window may include too much background or just include part of the target. Gaussian window proposed in sKCF is used to solve this problem. If the target center location is (x_0, y_0), the Gaussian window will be

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-((x-x_0)^2+(y-y_0)^2)/2\sigma^2}. \quad (6)$$

(x, y) can be any point in that image and G(x, y) is filter value at that point. At last, σ controls the shape of Gaussian window, this will depend on the ratio between the size of target window and the size of patch window.

2.5 Scale estimation

The scale estimation uses a key-point based strategy. Basically, we will find some key points from frame 1 based on Harris corner detector [16]. More uniform distributed random points will be added to the key points set.

Then, these key points will be tracked into frame 2 based on optical flow algorithm and it is also called forward process. After this process, a set of key points will be generated in frame 2. Next, we will use the optical flow again to track calculated key points in frame 2 into frame 1. It is also called backward process. At last, only the key points which can be tracked both in forward process and backward process will be kept for later calculation. If after optical flow forward-backward strategy frame 1 has key points $K^{p1} = \{K_1^{p1}, K_1^{p2}, K_1^{p3} \ldots K_1^{pi}\}$ and frame 2 has key points

$K^{p2} = \{K_2^{p1}, K_2^{p2}, K_2^{p3} \ldots K_2^{pi}\}$. Then without deformation of a target, we can estimate

the scale change based on these key points. Note that different key point pairs have different weights since key points which are near center should be given a higher weight while key points which are near border should be given a lower weight. So the scale change is

$$\mathrm{scale}(K^{p1}, K^{p2}) = \frac{\sum_i^T \sum_j^T w_i w_j * \frac{\left\| K_1^{pi} - K_1^{pj} \right\|^2}{\left\| K_2^{pi} - K_2^{pj} \right\|^2}}{\sum_i^T \sum_j^T w_i w_j} . \quad (7)$$

This means, we can use the change of distance to estimate the scale change. Weights will determined by the Gaussian window and the key point location in that Gaussian window.

2.6 Summary

In brief, sKCF will proceed as follows: Given the initial selection of a target (i.e., center position and size), a tracked region is created and the initial model will also be built. The tracked region is increased from the target size to provide some context. Features (either raw pixels or other hand crafted features such as FHOG) are extracted from the tracked region and each channel is weighted by a Gaussian window. So for every new frame, its feature will be sent into trained model and key point pair strategy will estimate scale change. After given a new location and scale change, new target and new Gaussian window will be used to train model for the next frame until the end of the sequence.

3. Deep-srKCF Algorithm

3.1 Rotated Gaussian Window.

The original Gaussian window can only handle scale changes of a target. This limitation will cause some error. First, because the filter is not compatible with the target, key points we select may have wrong weights, in that case, the scale and rotation angle estimation will become not accurate. Second, tracker itself always consider target window as real target, with wrong filter, the target will gradually become a mixture of target itself and background. So using a rotated Gaussian window may improve our tracker performance. In a coordinate system, if there is a point p(x, y), then after it rotates around origin point a certain angle α(clockwise). The new coordinate will be $p(x\cos\alpha + y\sin\alpha, -x\sin\alpha + y\cos\alpha)$
So, based on this conclusion, we know that

$$G_\alpha(x, y) = G(x\cos\alpha + y\sin\alpha, -x\sin\alpha + y\cos\alpha), \quad (8)$$

where α means the intersection angle between two Gaussian windows. Based on (6) and (8), we can get the equation of rotated Gaussian window as shown below

$$G_\alpha(x,y) = \frac{1}{2\pi\sigma^2} e^{-((x\cos\alpha + y\sin\alpha - x_0)^2 + (y\cos\alpha - x\sin\alpha - y_0)^2)/2\sigma^2}. \quad (9)$$

The rotated Gaussian window can better fit the target if the object rotates while tracking.

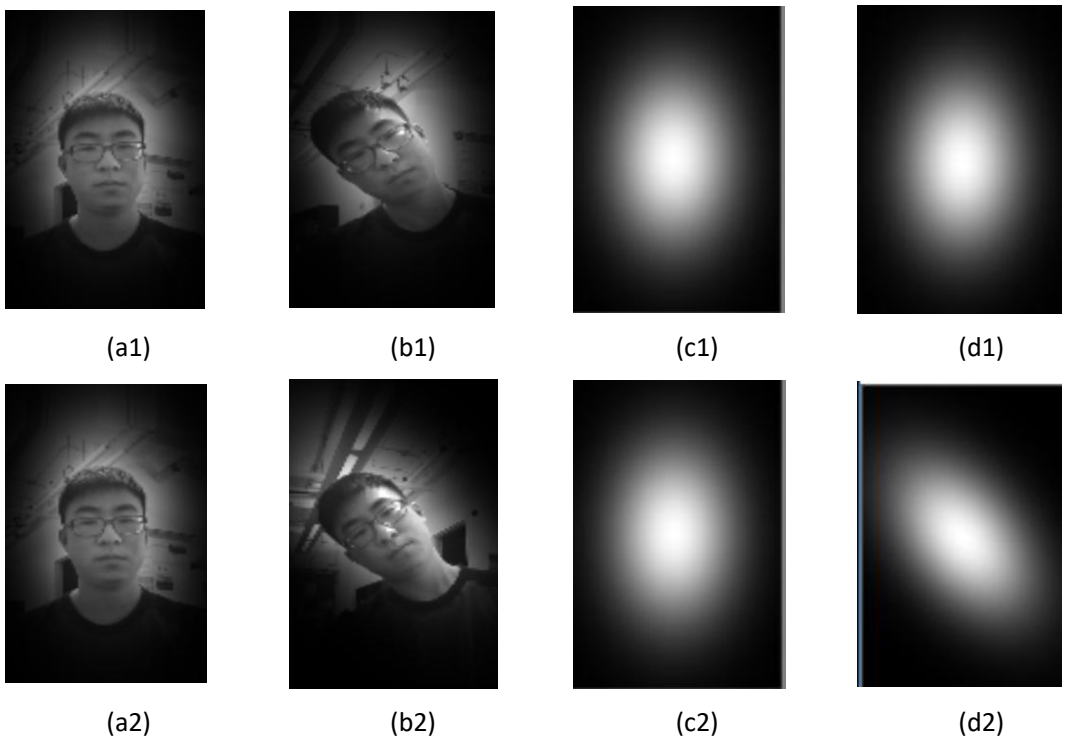


Figure 2. Original Gaussian window(c1)(d1) don't change with the target rotation. So in (b1) some part of object becomes darker because of unchanged filter. However, in our tracker, rotated Gaussian window (c2)(d2) are able to rotate so the target can be still given a high weight in filter and background is given a low weight.

3.2 Rotation angle estimation

Rotated Gaussian window updates based on the estimated rotation angle. The rotation angle estimation also uses a key point based strategy. We will find some key points from frame 1 based on Harris algorithm. More uniform distributed random points will be added to the key points set.

Then, these key points will be tracked into frame 2 based on optical flow algorithm and it is also called forward process. After this process, a set of key points will be generated in frame 2. Next, we will use the optical flow again to track calculated key points in frame 2 into frame 1. It is also called backward process. At last, only the key points which can be tracked both in forward process and backward process will be used kept for later calculation. If after optical flow forward-backward strategy, frame 1 has key points $K^{p1} = \{K_1^{p1}, K_1^{p2}, K_1^{p3} \ldots K_1^{pi}\}$ and frame 2 has key points

$K^{p2} = \{K_2^{p1}, K_2^{p2}, K_2^{p3} \ldots K_2^{pi}\}$. Then without deformation of a target, we can estimate the rotation angle change based on these key points. The intersection angle between line $K_1^{pi}K_1^{pj}$ and line $K_2^{pi}K_2^{pj}$ will be rotation angle estimation of one pair of lines. Just like scale change, we need to be more confidence about the points around the center and less confidence about the points around the corner. As a result, we will do the estimation based on all pair of lines with different weights.

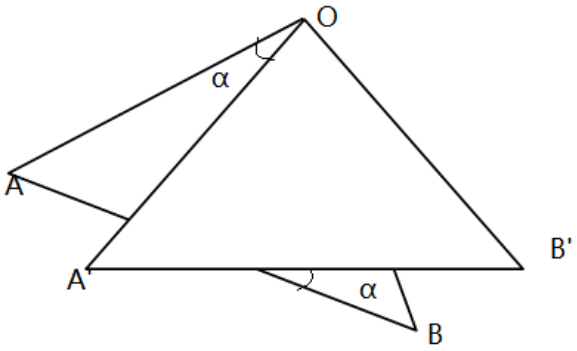


Figure3. A and A' are a pair of key points, B and B' are another pair of key points. If there is no deformation of target (only translation, scale change and rotation), then we know that intersection angle of two lines is the rotation of a target.

The formula for rotation angle will be

$$\text{rotation}\left(K^{p1}, K^{p2}\right) = \frac{\Sigma_i^T \Sigma_j^T w_i w_j * \theta(\text{line}(K_2^{pi}, K_2^{pj}), \text{line}(K_2^{pi}, K_2^{pj}))}{\Sigma_i^T \Sigma_j^T w_i w_j} \qquad (10)$$

line(A,B) means two points A, B form a line. $\theta(l_1, l_2)$ is the intersection angle between two lines. Parameter w is point weights in rotated Gaussian window.

When we calculate the scale change or rotation angle change, every pair of key point will be considered. However, it doesn't mean we need to use all of them. Optical flow can't guarantee 100% accuracy so there might be some wrong key point pairs exist. With incorrect information, the estimated scale or rotation angle change will also be incorrect. To deal with outliers, I will sort the estimation result of every key point pair. Then, 10% of highest value and 10% of lowest value (rounded down) will be deleted. In that case, some abnormal value will not participate in the average value.

At the same time, the variance of remaining values of rotation angle and scale change are also needed to compare with a threshold, because a high variance means the estimation result for this frame is very unstable, so we need to abort the results in that frame. Currently, the variance threshold for scale change is 0.001 and the variance threshold for rotation angle change is 0.06 degree.

3.3 Initialization of bounding box

Given a groundtruth, we can confirm the position of a target in the first frame. We will use this position to initialize our target window and the model. The groundtruth means the real position of a target. It is a rotated rectangle, or four vertex points. sKCF doesn't take rotation into consideration, it will just consider bounding box of key points as the target. This is not true, because, the bounding box will include some background information.Instead, we will build a rotated rectangle from four key points.

If key points are $P_1(x_1, y_1), P_2(x_2, y_2), P_3(x_3, y_3), P_4(x_4, y_4)$, then the center of target will be

$((\max_x + \min_x)/2, (\max_y + \min_y)/2)$. Then we pick up the longer length of rectangle to measure the angle, line=max(P_1P_2, P_2P_3), so the initial angle is

$$\alpha = \theta(\text{line}, \text{axis}_{x_{\text{positive}}}) \tag{11}$$

In this case, we can potentially get a better model at the beginning.

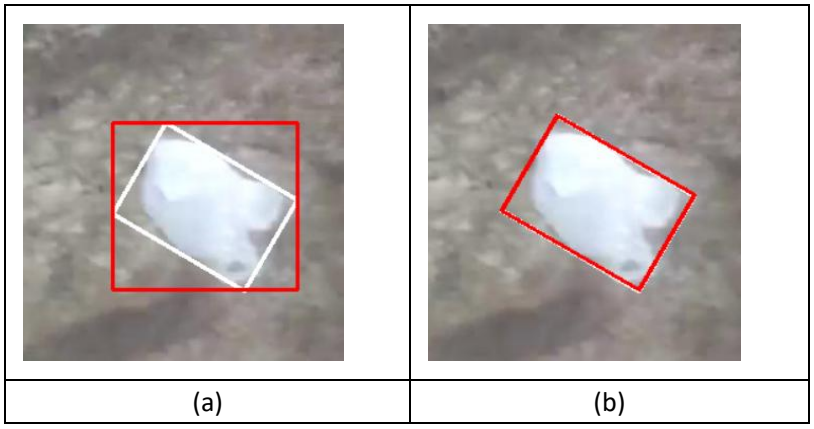


Figure 4. White rectangle represents the groundtruth and the dark rectangle means the target. Origin sKCF will only consider axis aligned bounding box as target, so just like (a) shows, our dark bounding box contains some background information. In figure (b), new initialization way will take rotation into consideration so the dark rotated bounding box will totally be same as white groundtruth.

3.4 Interpolation of maximum response

Every time a new frame comes in, it will be transferred into feature space in the different ways. The size of feature space is not actually the same as original frame. For instance, if original image size is M*N, then the feature space after FHOG process will be M/4*N/4 because of the cell size. As a result, final response matrix is also M/4*N/4.

The final target shift will be estimated by the maximum location in response matrix and the shift result will not be very accurate since the shift is an integer value (0, 4,

8….). So, we can use a interpolation strategy to improve that. Assume the response matrix is R(X) and we are going to use Taylor expansion near original maximum at point $X_0(x_0, y_0)$, then X=(x, y) means a point near (x_0, y_0). So based on Taylor expansion, we can get

$$R(X) \approx R(x_0,\ y_0) + \frac{\partial R^T}{\partial X}(X - X_0) + \frac{1}{2}(X - X_0)^T \frac{\partial^2 R}{\partial X^2}(X - X_0). \quad (12)$$

To get the local maximum, $\frac{\partial R}{\partial X} = 0$. So the shift between original maximum response and the calculated value will be

$$\Delta X = -\frac{\partial^2 R^{-1}}{\partial X^2}\frac{\partial R}{\partial X}, \quad (13)$$

Where R^{-1} means the inverse matrix of original R. When doing the derivative calculation, we are going to use pixels which are near the point X_0.
We add the shift to original local maximum, so the location of target in new frame comes out.

3.5 Deep learning feature

Traditional hand-crafted features have already been researched and used in many places. Features such as FHOG have been used successfully in face detection and pedestrian detection. However, there are limitations for traditional features. On the one hand, they don't perform well in some complex video sequences. On the other hand, they are not flexible enough. It is difficult to find a balance between running speed and performance based on application needs.
Compared with traditional features, deep learning features are much more flexible and easier to use, with good parameter, they will also be more expressive than traditional feature such as FHOG. Deep learning features mean middle level output of a deep neural network. In our project, we use AlexNet to do the work.
AlexNet is the name of a convolution neural network, which competed in the ImageNet Large Scale Visual Recognition Challenge in 2012. The network achieved a top-5 error of 15.3%, more than 10.8 percentage points ahead of the runner up. In our project, we directly use a pre-trained network to do the test.
The most interesting part is that this pre-trained network is actually built from ImageNet which is totally irrelevant to our VOT challenge sequences. However, the network does work with the tracker.

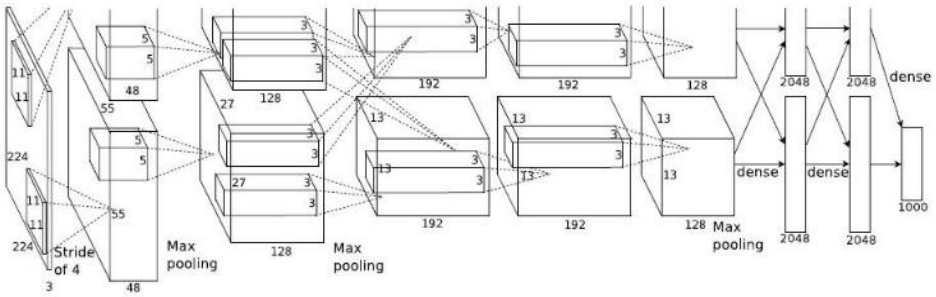


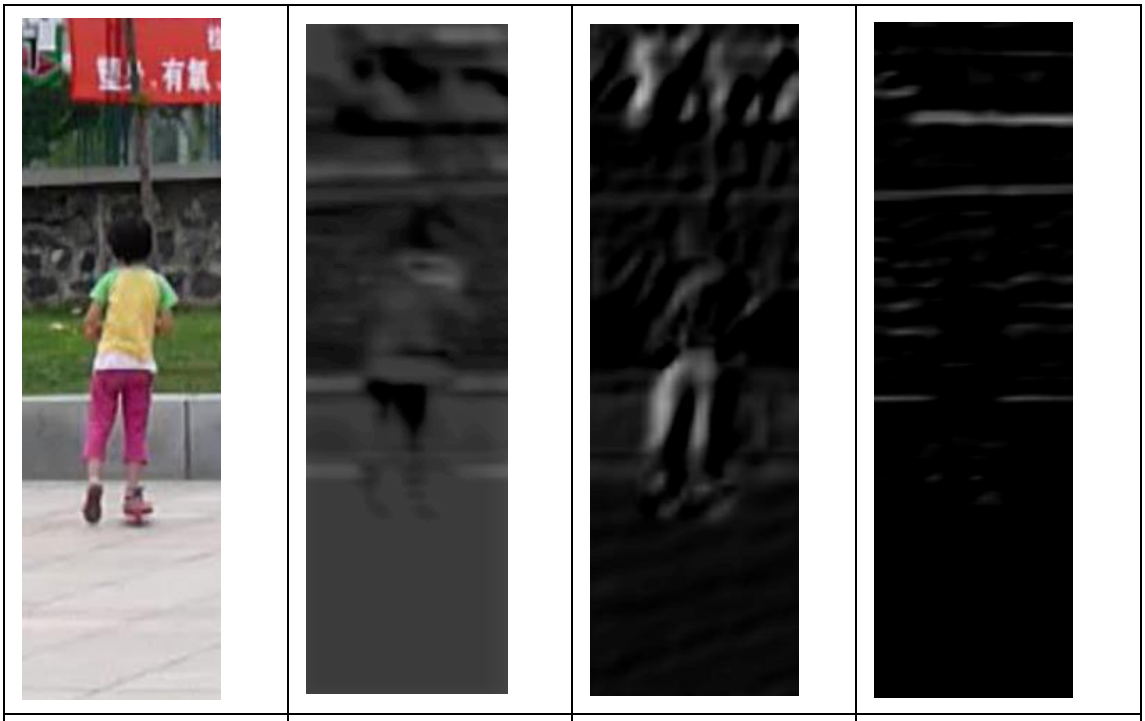
Figure 5. The structure of AlexNet. Taken from: Alex Krizhevsky, Ilya Sutskever, Geoffery E.Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." 2012.© 1987-2018 Neural Information Processing Systems Foundation, Inc.

There are many middle layers in AlexNet, we can use any one of it in theory. However, we are going to use the first layer in our test. First, convolution is a very time consuming calculation, as a result, using other layer means we have to do much more convolution with many layers. Second, first middle layer is closest to original input image, so we can potentially obtain features that suitable for our tracker.

Assume the size of input image is $A \times B$ and it has C channels. In the meanwhile, kernel dimension is $C \times K \times K$ and the number of kernel is N, then the output features can be

$$\text{Convolve}(A \times B \times C, C \times K \times K \times N) = \text{Output}_{A \times B \times N} \tag{14}$$

It means that deep learning features have the same size as input image and the features have N channels. In actual test, we can set different number of kernels to use based on the capacity of computer. More kernels may increase the tracker performance but it will also decrease the running speed.



(a) (b) (c) (d)

Figure 6. (a) is the original input image and (b)(c)(d) are the convolution results between input and three different kernels. As we can see, different kernels give different results based on the same image. Once given more kernels, the features can be built.

Just like other feature descriptor, deep learning features also have two important parameters.

First one is rescale ratio. Assuming the original image size is $M \times N$, at the same time, our final feature descriptor size is $M/4 \times N/4$. Then there are many choices before we start convolution process. We can directly do convolution on original image and then resize the output to $M/4 \times N/4$. In that case, we can get more information from original image but it will cost us more time. We can also rescale the original image to another size, for instance, $M/2 \times N/2$. In this case, rescale ratio is 0.5, we will save some time in the convolution phase, however, resize the image also means loss of information, so we will potentially get a worse test result.

Second parameter is the kernel number. AlexNet contains 96 channels in the first hidden layer. However, based on my current C++ implementation, it is unrealistic for us to use all of the kernels in our tracker. So we can set a reasonable amount of kernels to use based on our own situation.

Usually, more kernels mean we can get a more stable tracker. Stable means lower failure rate in our tracker. More kernels also mean we need to spend more time in convolution. Note that, currently, we just select our kernels starting from first kernel, this is not an optimized way to do it since we don't know which kernel is better for our tracker. Further research can be done here.

Both parameters will also be discussed in section 4.

3.6 Process of Deep-srKCF

In conclusion, Deep-srKCF mainly has two differences compared with original sKCF. First, it has the ability to handle object rotation. Second, it can use expressive deep learning feature to build the tracker model.
Below are the basic steps of this tracker.

1. Read the groundtruth from the disk or manually set an initial target bounding box.
2. Based on first frame of groundtruth or bounding box, we will initialize the window size, target size and the rotation angle.
3. Calculate the initial tracker model, the Gaussian window filter and the key points from first frame.
4. If we use deep learning features, the kernels parameter will also be read into our tracker.
5. Every time a new frame comes in, we will convert the original frame into the feature descriptor. Then the feature descriptor will be put into tracker model so that we can get the location of the target in this new frame.
6. Based on last frame and this frame, the key point pairs will be calculated. With the key point pairs, we can calculate the scale and rotation angle change of the target.
7. The Gaussian window filter will be updated after scale and rotation angle change.
8. Update the model and extract new key point pairs using new target location and Gaussian window filter.
9. If this is the end of the sequence, quit the program. Or we jump back to step 5 and track the target in the new frame.

4. Experiments

4.1 Experiment setup and Methodology

The original sKCF is mainly developed with C++ implementation. We apply our changes which are also developed in C++ language to the original sKCF algorithm. First, we apply the ability to handle target rotation to the sKCF algorithm. Second, we change the FHOG descriptor to deep learning features. Other things remain the same as before. All experiments are conducted on an Intel i7-4970 at 3.6GHz with 16GB of memory. We use the same parameter configurations for all implementations as described in [7]: interpolation factor = .02, Gaussian kernel correlation σ = .5, regulation term λ = 1e $-$ 4, HoG cell size is 4 $\times$ 4 and there are 9 orientations bins. We use the VOT challenge 2017 datasets as our test datasets.
In the test reports, there are three main performance indicators.
First is overlap rate. For one single frame, if the overlap area between groundtruth and our detection bounding box is A, the area of groundtruth is B, then the overlap rate for this frame will be A/B. The average overlap rate for all frames in one video sequence will be the overlap rate for that sequence.
Second one is failure times, it only exists in baseline mode. For a single frame, if the

overlap area between groundtruth and our detection bounding box becomes 0, then we fail in this frame. In that case, the tracker will be initialized again in the next frame. The total failure number in one video sequence will be the failure number of that video sequence.

Last one is running speed. We use frame per second (FPS) to evaluate the running speed of the tracker. Note that the first two indicators are not relevant to the hardware capacity. However, we will have different running speed test result in different computers. The running speed in our report will only apply to our computer configurations.

Every aspect is important for the tracker. First one indicates the tracking capacity of the tracker. Higher overlap rate means it can work better in different kinds of sequences. Lower failure number means the tracker is more stable. The tracker will especially perform better in complex video sequences. High running speed means a tracker can keep real-time.

4.2 Experimental results

Experiment involves two kinds of tests which are both from the standard VOT Challenge approach.

First one is the baseline test. In this mode, every time a track fails in one frame, the toolkit will initialize the tracker again. We will have overlap rate, failure time and running speed performance indicators in baseline mode.

Second one is the unsupervised test. In this mode, the tracker will just start run from the beginning to the end of a video sequence and the toolkit will not interrupt the tracker. As a result, we will only have overlap rate and running speed indicators.

There are three kinds of trackers in below tables. They are the original sKCF, srKCF and Deep-srKCF. Note that deep learning features will be tested in multiple parameters. There are two parameters as explained before, rs means rescale ratio, kernel means number of kernels we use.

Table 1. VOT 2017 baseline test results

Tracker \ Results	Overlap rate	Failure times	Running speed
sKCF	0.39	4.09	**102.21**
srKCF	0.41	4.86	99.94
DeepsrKCF rs=0.5 kernel=15	0.42	4.69	94.13
DeepsrKCF rs=0.5 kernel=30	0.40	3.84	69.49
DeepsrKCF rs=0.5 kernel=45	0.43	4.02	55.32
DeepsrKCF rs=0.5 kernel=60	0.41	3.82	45.67
DeepsrKCF	0.39	4.47	97.31

rs=0.25 kernel=30			
DeepsrKCF rs=0.75 kernel=30	0.42	3.97	48.74
DeepsrKCF rs=1 kernel=30	**0.44**	**3.74**	31.24
DeepsKCF rs=0.5 kernel=30	0.40	4.10	70.36

Table 2. VOT 2017 unsupervised test results

Tracker \ Results	Overlap rate	Running speed
sKCF	0.22	**109.07**
srKCF	0.22	108.82
DeepsrKCF rs=0.5 kernel=15	0.23	95.17
DeepsrKCF rs=0.5 kernel=30	0.24	75.01
DeepsrKCF rs=0.5 kernel=45	0.25	57.15
DeepsrKCF rs=0.5 kernel=60	0.25	50.03
DeepsrKCF rs=0.25 kernel=30	0.22	105.40
DeepsrKCF rs=0.75 kernel=30	0.26	51.93
DeepsrKCF rs=1 kernel=30	**0.27**	34.22
DeepsKCF rs=0.5 kernel=30	0.24	77.23

4.3 Analysis

First, from the test result, we know that deep learning feature is a flexible feature because the change of parameter will directly affect the performance and running time. So we can use this feature based on our own needs.

For overlap rate, we will mainly refer to unsupervised test since this test correspond to the situation when we run tracker on a previously unseen video.

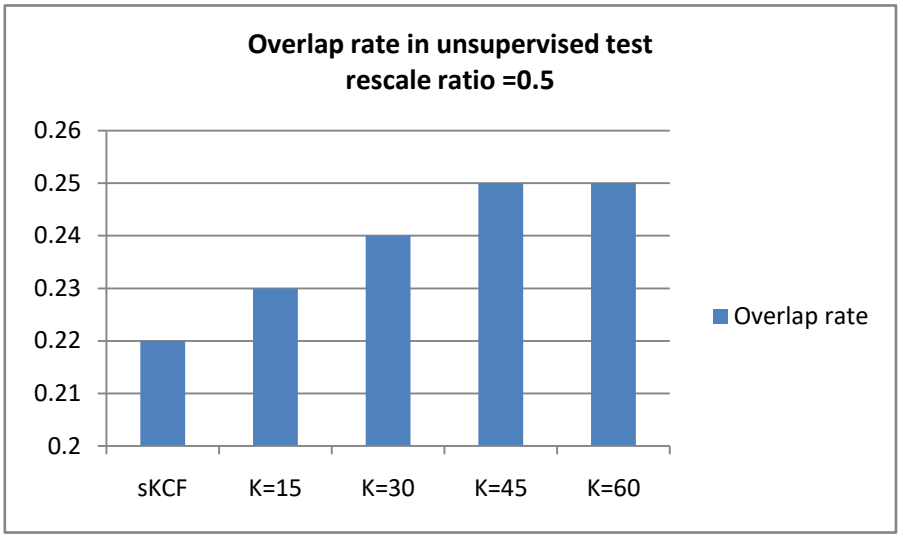


Figure 7. Overlap rate increases with the kernel number

From figure above, we can see that a higher kernel number increases the overlap rate.

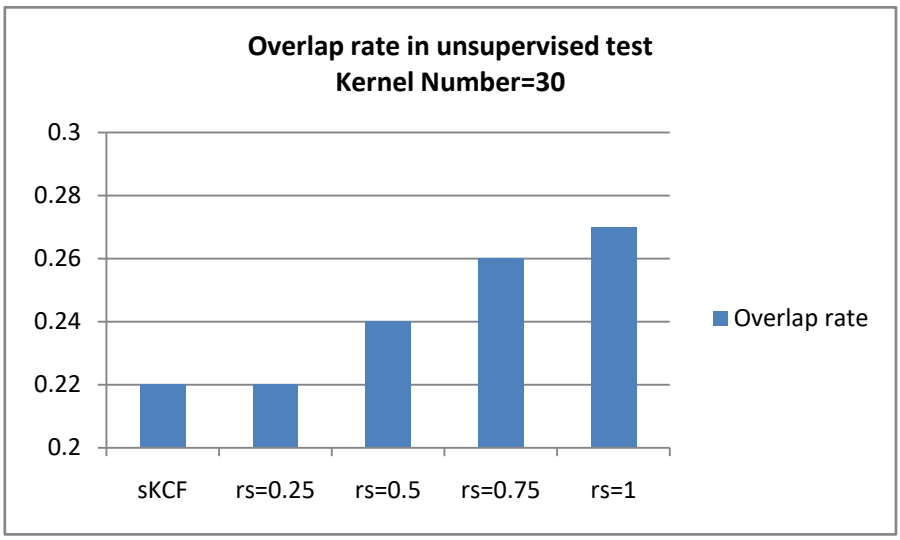


Figure 8. Overlap rate increases with a higher rescale ratio

From figure above, we can see that a higher rescale ratio can increase the overlap rate. The overlap rate become maximum when we use original image to do convolution Next indicator is failure time which we will refer to baseline test data.

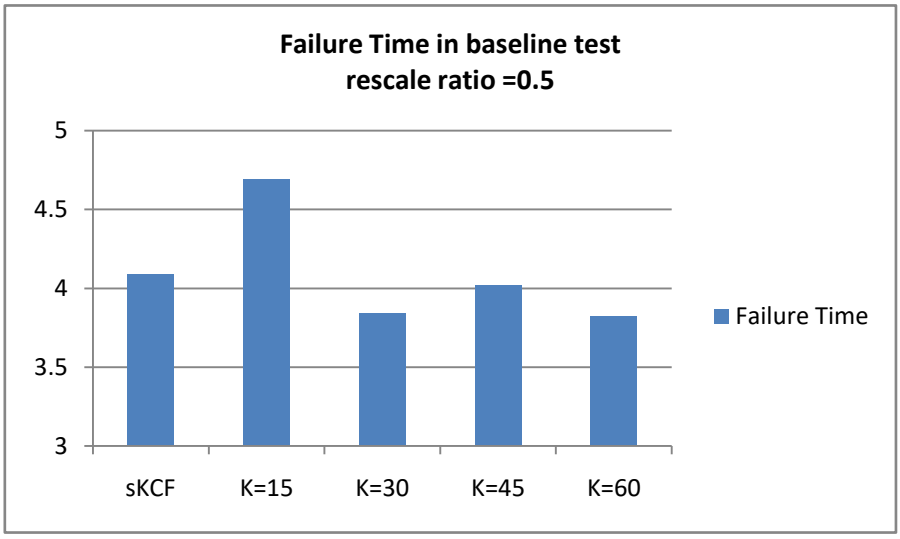


Figure 9 . Low kernel number will result a bad test result while higher kernel number can give a better test result compared with sKCF.

From the figure above, we can see that a low kernel number such as k=15 will result in high failure rate. But if kernel number is bigger than 30, we can get a better failure time compared with original sKCF.

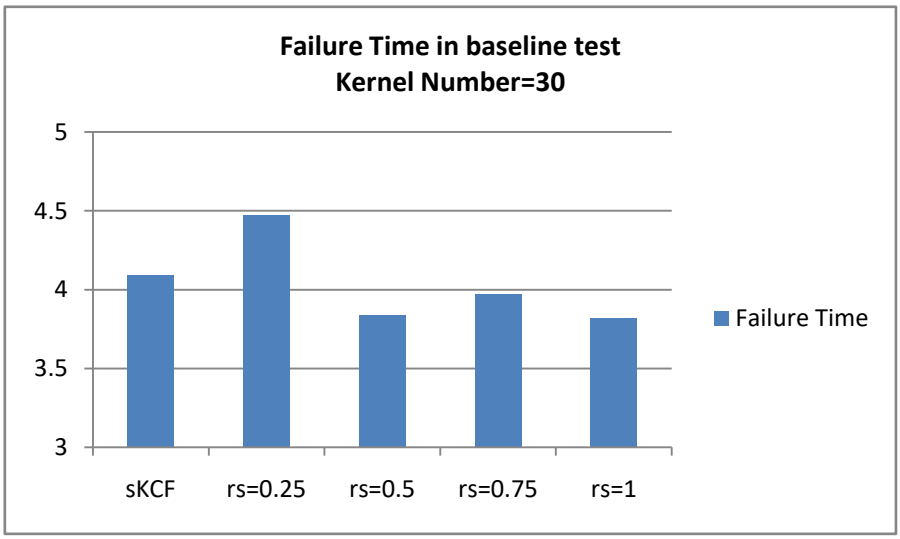


Figure 10. Low rescale ratio will result in high failure time while higher rescale ratio can give a better test result compared with sKCF.

From figure above, we can see that a low rescale ratio such as 0.25 gives a high failure rate.

But if the rescale ratio is bigger than 0.5, we can obtain a better test result compared with original sKCF.

Third indicator is running speed. Both of the parameters have inverse proportion to the running speed.

So based on the discussion above, we know the influence of the two parameters. This will help us set the deep learning feature parameters based on our needs. With a

reasonable parameter setting, we can get a better performance using deep learning feature while keeping the real-time attribute of the tracker. Based on our test results, the recommended configuration for the two parameters are k=30 and rs=0.5.

The impact of rotation functionality is interesting. From the test result, the rotated Gaussian window doesn't make positive impact on the performance. Original sKCF works better than srKCF. However, this doesn't mean rotated Gaussian window is meaningless.

First, the rotated Gaussian window helps the tracker to get the rotation angle of target. Rotation detection is an useful and interesting functionality in itself.

Secondly, when we use deep learning features, rotated Gaussian window has a positive impact on test result. From the test, when we set rs=0.5 and kernel number=30, the rotated Gaussian window improved failure rate. So we know that the rotated Gaussian window is useful and it should be used together with deep learning features.

5. Disadvantages and Potential Improvements

5.1 Calculation

The new tracker proposed in this project is not perfect. As we can see from last paragraph, the performance has already improved compared with original sKCF, however the running speed is slower than before.

We notice that original sKCF project is actually a CPU project. At the same time, the power of the GPU has been recognized by many researches [14]. So a potential way is to change the project implementation from Opencv with CPU to Opencv with GPU support.

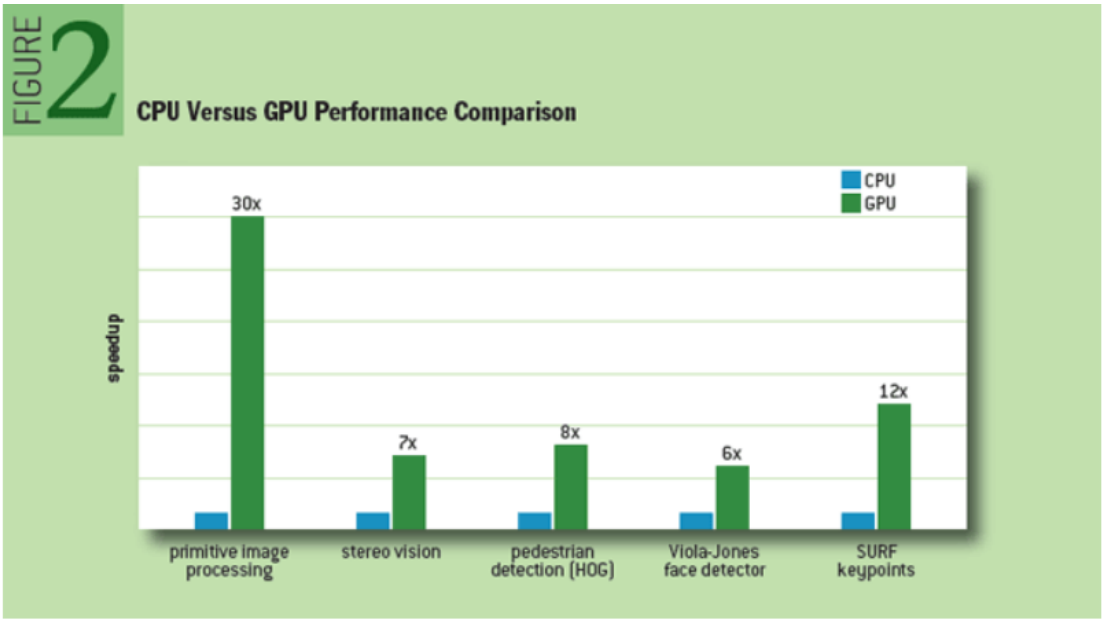


Figure 11. The performance comparison between CPU and GPU. Taken from: Pulli, K., Baksheev, A., Kornyakov, K., & Eruhimov, V. "Realtime computer vision with OpenCV." 2012. Queue 10(4), 40.© 2018 ACM, Inc.

The use of GPU needs us to use CUDA toolkit and compile it with Opencv source code. After some work, we will get the Opencv3.4+CUDA9.1 library and it can be used in our tracker.

Every time we use GPU to calculate, we need to upload our data from the CPU to the GPU, then, data will be calculated in the GPU. After all the calculation, we download the data from the GPU back to the CPU.

However, the test result shows the use of Opencv+CUDA in our tracker is not a success. The GPU convolution actually takes much more time to finish its work. So in our tracker, we still use CPU to do our work. There might be multiple reasons. For instance, we don't fully make use of parallel computing ability of GPU in our configuration and coding. This topic needs much more research in the future and I will not cover this topic in this report further.

5.2 Handle occlusions with failure detection.

Occlusion is a difficult problem in visual tracking. If we just update our tracker based on model and current frame, occlusion is likely to happen since the track doesn't know what foreground and background are.

(a) (b) (c)

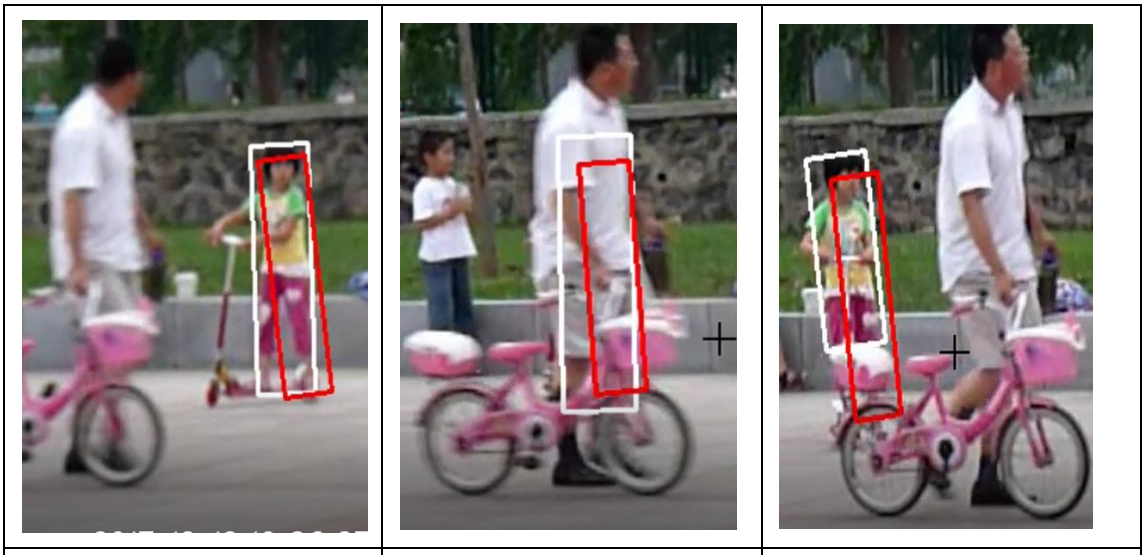
Figure 12. (b) shows that a man is in front of a girl. The model updated based on the man's information so we lost track as shown in (c)

One solution is we depend more on prior knowledge, that means we may store a set of models rather than single model, so when an occlusion happens, it can still work fine because there are more prior models to handle with this. But this will decrease our speed and we believe it is not so feasible to store multiple models in our tracker.

Another solution is that we stop model updating and stop target motion when we detect a failure. Once failure solved, we reinitialize our tracker and continue our tracking. The major problem here is how we can detect a failure. A simple way is look at the maximum value of the response matrix. When a failure happens, the maximum value becomes small at the failure frame.

The simple solution is not good actually as "small" is not a good threshold. A strict way is to use a machine learning model such as SVM to train a model to detect the

response matrix, so we have more confidence to determine if a frame is a failure frame. This failure detection research is out of my project scope so I just mention it here.



(a) (b) (c)

Figure 13. When occlusion happens, the maximum response value becomes very low. So base on this indicator, we stop model and motion update in frame (b), once the response value becomes larger, we reinitialize our tracker and continue tracking object.

5.3 Video segmentation

As we stated above, the window filter is very important in our tracker because it can block the information from background and just leave the foreground information.

From the initial hann window to rotated Gaussian window, we have already made much progress in the window function. If we take one further step to make the filter window more accurate, the result may potentially become better.

Currently, there are several attempts in our project to perform video segmentation in our tracker. First way is to use mix of Gaussian segmentation algorithm. However, this way can only work in static background. Or instead, we try to find the difference between foreground and background based on the different motion of pixels. In this case, optical flow is a choice, but it still remains problematic. One problem is that it is extremely slow, another problem is that the segmentation is not accurate in many cases.

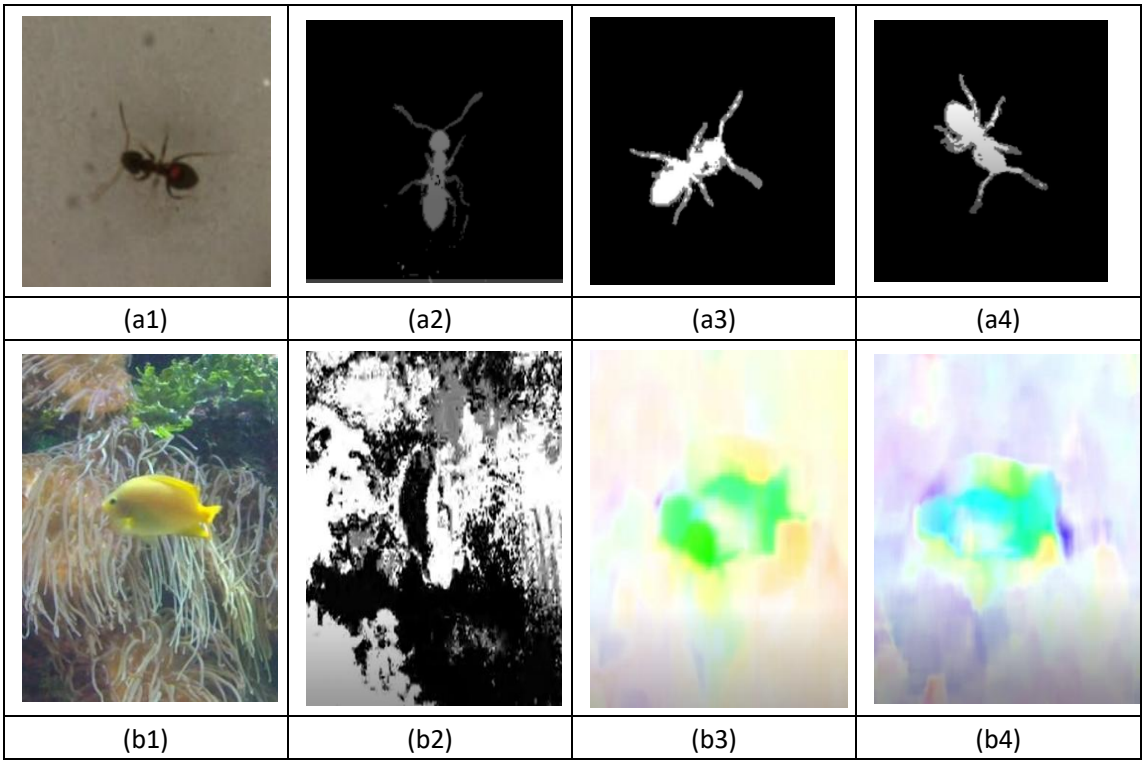


Figure 14. (a2)(a3)(a4) show a segmentation based on mixture of Gaussian. It works fine because the background is static. Mixture of Gaussian fails in (b2) since foreground and background both move. (b3)(b4) shows dense optical flow performance in dynamic background sequence. We see some differences between foreground and background but it is not good enough, at the same time, it is extremely slow.

Recently, many researches have shown that convolutional neural networks are powerful visual models that yield hierarchies of features. The end-to-end, pixel-to-pixel neural network model has shown great performance in semantic segmentation [19]. We believe the semantic segmentation network such as FCN may also potentially help us with our video segmentation.

Unlike AlexNet or VGG, the last few layers of FCN are not fully connected layers. Instead, FCN uses deconvolution strategy to transfer middle convolution layers to a three dimensional pixel score layer. The score layer can classify every pixel based on the scores in different channels.

Based on the introduction above, there are three problems if we try to use current FCN network in our video segmentation:

1. The number of classes for pixel. If the classes only contain major objects in daily life, some images will not be segmented well since we track all kinds of different objects which may not be included in pixel classes.
2. The combination of objects. If we track a man who is on a bicycle, the FCN will fail since the man and the bicycle belong to different kinds of pixels.
3. Part of an object. If we track a hand, the segmentation may fail since the pixels belong to hand and belong to arm are all part of human pixels.

So from the analysis and the figures below, we believe that semantic segmentation such as FCN is a feasible way for video segmentation, however, much more research

is also needed in this area.

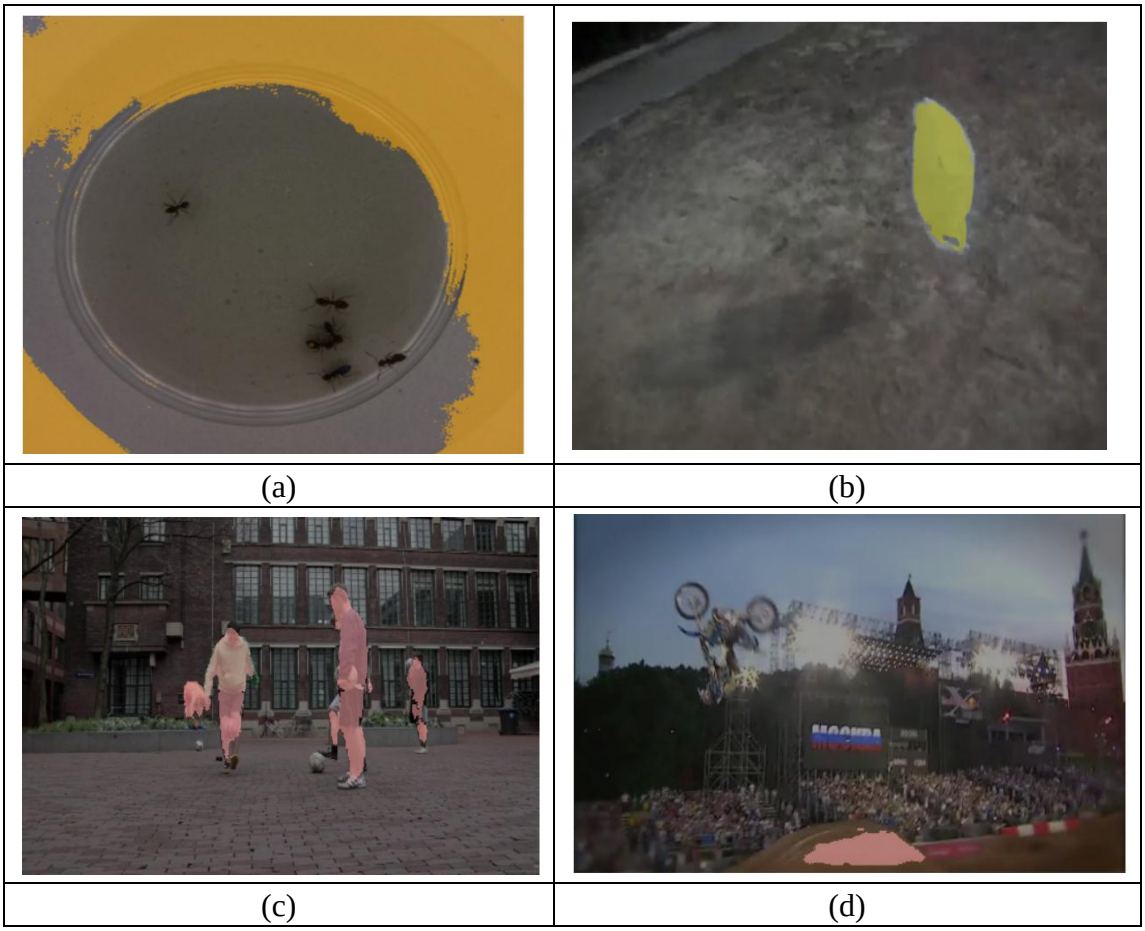


Figure 15. (a) shows the lack of ants class result in a wrong segmentation. (b) is a successful segmentation. (c) shows pixels belong to football are falsely classified into human. (d) shows a failure when the tracked object is a man sitting on a motorcycle.

6. Conclusion

In this project, we present a improved scalable kernel correlation filter visual tracker. This improved tracker is able to handle object rotation based on key-point strategy and rotated Gaussian window. At the same time, we proposed a new feature descriptor based on deep neural network. The new feature descriptor is more flexible, easy to implement and more suitable for complex video sequences. We observed better overlap, failure time in VOT challenge test while this new tracker can still stay “real-time”.

reference

1. A. Roshan Zamir, A. Dehghan, and M. Shah. "Gmcp-tracker: Global multi-object tracking using generalized minimum clique graph." *Proceedings of the European Conference on computer vision.* 2012.
2. A. S. Montero, J. Lang, and R. Laganière. "A general framework for fast 3d object detection and localization using an uncalibrated camera." *2015 IEEE Winter Conference on Applications of Computer Vision.* Waikoloa, HI, USA, January 5-9, 2015. 884-891.
3. Avidan, S. "Support vector tracking." *TPAMI.* 2004. 33(8): 1064-1072.
4. B. Babenko, M.-H. Yang,and S. Belongie. "Robust object tracking with online multiple instance learning." *TPAMI.* August 2011. (33)8: 1619-1632.
5. H. Grabner, C. Leistner, and H. Bischof. "Semi-supervised on-line boosting for robust tracking." *ECCV.* 2008.
6. I. Hwang, H. Balakrishnan, K. Roy, and C. Tomlin. "Multiple target tracking and identity management in clutter for air traffic control ." *Proceedings of the AACC American Control Conference.* 2004.
7. J. ao F. Henriques, R. Caseiro, P. Martins, and J. Batista. "Exploiting the circulant structure of tracking-by-detection with kernels." *Proceedings of the European Conference on Computer Vision - Volume Part IV, ECCV'12.* Berlin, Heidelberg: Springer-Verlag, 2012. 702-715.
8. J. F. Henriques, R. Caseiro, and J. Batista. "Globally optimal solution to multi-object tracking with merged measurements." *ICCV.* D. N. Metaxas, L. Quan, A. Sanfeliu, and L. J. V. Gool: IEEE, 2011. 2470-2477.
9. J. F. Henriques, R. Caseiro, P. Martins, and J. Batista. "Highspeed tracking with kernelized correlation filters." *Pattern analysis and Machine Intelligence.* IEEE Transactions on, 2015.
10. L. Dong, I. Zoghlami,S. Schwartz. "Object tracking in compressed video with confidence measure." *IEEE International Conference on Multimedia and Expo.* July 2006. 753-756.
11. M. Shah, J. Omar. "Tracking and object classification for automated surveillance." *Proceedings of the European Conference on Computer Vision-Part IV.* London, UK, UK: Springer-Verlag, 2002. 343-357.
12. Montero, Andres Solis, Lang, Jochen, Laganiere, Robert. "Scalable kernel correlation filter with sparse feature integration." *Computer Vision Workshop (ICCVW), 2015 IEEE International Conference on.* IEEE, 2015. 587-594.
13. P. F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan. "Object detection with discriminatively trained part based models." *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 2010. 32(9): 1627-1645.
14. Pulli, K., Baksheev, A., Kornyakov, K., & Eruhimov, V. "Realtime computer vision with OpenCV." 2012. Queue 10(4), 40.
15. S. Hare, A. Saffari, and P. Torr. "Struck: Structured output tracking with kernels." *ICCV.* 2011.
16. Stephens, Chris Harris and Mike. "A Combined Corner and Edge Detector." *Alvey Vision Conference.* 1988.
17. V. N. Boddeti, T. Kanade, and B. V. K. V. Kumar. "Correlation filters for object alignment." *In Proceedings of CVPR.* IEEE, 2013. 2291-2298.
18. Y. Park, V. Lepetit, E. Cvlab, and W. Woo. "Multiple 3d object tracking for augmented reality."

Proceedings International Symposium on Mixed and Augmented Reality. 2008. 117-120.

19. J. Long, E. Shelhamer, T. Darrell. "Fully Convolutional Networks for Semantic Segmentation." *In Proceedings of CVPR*. IEEE, 2015. 3431-3440