上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

Project Title Real-time illumination Robust Tracker under Ensemble Learning 集成学习下高鲁棒性人脸追踪器

Name1 Huan Zheng ID1 5133709255 Major ECE

Name2 Xuefeng Hu ID2 5133709060 Major ECE

Name3 Siying Li ID3 5133709049 Major ECE

Name4 Shengjie Pan ID4 5133709064 Major ECE

Name5 Zhenren Lu ID5 5133709275 Major ECE

Supervisor 马澄斌 Chengbin Ma

School UM-SJTU Joint Institute

Semester 2016-2017-Summer

上海交通大学

毕业设计（论文）学术诚信声明

本人郑重声明：所呈交的毕业设计（论文），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：

日期： 2017 年 8 月 7 日

上海交通大学
毕业设计（论文）版权使用授权书

本毕业设计（论文）作者同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本毕业设计（论文）。

保密□，在___年解密后适用本授权书。

本论文属于

不保密☑。

（请在以上方框内打“√”）

作者签名：　　指导教师签名：

日期：2017 年 8 月 7 日　　日期：　　年　　月　　日

Abstract

In this paper, we introduce a real-time robust face tracker under different illumination scenarios. Instead of using single model or classical tracking algorithm, we designed a new tracking function that combined the result of face detection and the object tracking, and achieved a better result under variant scenarios. We have implemented two versions with differnet detection model: a tracker with high accuracy Faster-RCNN detector for environment with GPU installed, and a tracker with realtime dlib detector for environment with no GPU installed. The overall test precision is over 95% for GPU vesion and 76% for cpu version. Processing speed is over 24 FPS.

Executive Summary

This project is sponsored by United Automotive Electronic Systems Co.,Ltd (UAES). The company is looking for an innovative in-car system that enables the driver easily control automotive electronics without pressing any button and, thus, improve the driving experience. The company directed our project to track human face in the with the camera.

The specification of the project is that the tracker should be specific object(in this case, face) oriented. It should remain accurate and robust under real-time use and different scenarios. Ultimately, it should also be portable to be installed into a car. More specifically, the processing speed should reach 24 FPS with accuracy over 80% and the size of parameters should be less than 100 MB.

Face detector and an object tracking system(a.k.a tracker) are two most important concepts. Face detector can also be applied to detect face(s) for a video stream with a relatively high accuracy but slow processing speed. While a tracker process faster than detector, but it cannot tell what kind of object it is tracking.

To take the advantage of both detector and tracker, we finally decide to combined them together. We use faster R-CNN as the model of detector and KCF as the tracker. To balance between detecting and tracking, we take the most possible results of both methods with a threshold. Then it adds weight to combine the rankings and outputs the most likely result.

The process of the system consists 4 parts: pre-processing, face detection, KCF tracking and visualization. The input video stream would be pre-processed into a set of images. Then for each image, both detector and tracker would give a predicted face position. The mixed ranking function would then give out the final position of the face. The position then would be visualized as a bonding box on the image and played as an output video stream.

To evaluate the results of our final design, we have tested the two version of our tracker (with *dlib* and with Faster RCNN) and compared the results with the original KCF over five different scenarios from the Video Tracker Benchmark. Through the evaluation, the performance of KCF with faster RCNN beats the performance of original KCF. The performance of KCF with *dlib* does not beat the original KCF but the result is still acceptable for practical use with a computer with no GPU installed.

Though we almost meet the requirements, there is still much space to improve. The weakness of our current model is the structure is too large. Faster R-CNN is designed for general object detection. The structure is not compact enough which leads to the high hardware requirement. Possible solution to make the project run faster on CPU is using a common face detector written with *dlib* as our detector.

The face tracker runs smoothly under real-time video test while remains its high robustness. It also works well under weak illumination background test and face occlusion test. The overall test precision is over 80%, which means this tracker is mature enough to generate a stable object moving trajectory for further use such as an in-car tracking control.

Table of Contents

Table of Contents

1. Introduction

United Automotive Electronic Systems Co.,Ltd (UAES) is a joint venture of Zhong-Lian Automotive Electronics Co., Ltd and Robert Bosch GmbH. The company is looking for an innovative in-car system that enables the driver easily control automotive electronics without pressing any button and, thus, improve the driving experience. The original function of this system is to detect the trajectory of simple gestures captured by a camera. However, due to the lack of data of hand gestures, the company redirect our project slightly to track human face in the video.

For a computer vision system, tracking can be specified as an online learning problem. Given the initial image containing the target, the goal is to learn a classifier to discriminate between its appearance and that of the environment.[3] Different from a detector which is detecting objects in each individual images, tracker is to track the object from the previous images.

Figure 1.1 Face tracking [5]

As a result, a real-time illumination robust face tracker is decided to be built as the core functionality of the system. The tracker should remain robust under certain scenarios, such as focus lose, sudden illumination change, dark background, etc. Our project is highly portable so the project can be utilized to recognize gestures given enough data.

2. Specifications

Our customer, UAES, is looking for a fast accurate reliable tracker that can be installed in a car. However, the proposal from them is too general that they not even define the object of the tracker. After an in-depth discussion with the mentor, we summarize requirements from customers as the following:

- Object: gestures/face/general objects in a car
- Real-time
- Accurate
- Reliable under different cases（ Sudden illumination change, focus lose, etc.）
- Portable/being able to be installed in a car

To translate the customer requirement into engineering specification, we first talked to the professor who teaches Computer Vision, he shared his experience with us that data collection is the most time-consuming part. He also mentioned that there is a huge amount of datasets on the face with great diversity. Considering the limited time, we choose human face as our tracking object. This decision will ease our burden on collecting and labeling data.

Secondly, we did the literature research in order to find the benchmark and leading algorithms in the tracking area. Through reading the papers, we also learn how the performance of an algorithm is evaluated from academic aspects. We find that

· Real-time = Fast speed/low time delay: Frame Rate (>24 frames per second (fps))

· Accurate = high precision: Average Precision (AP) (%)

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

· Reliable under different cases = Robustness: Precision Variance/Recall (%)

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

· Portable/being able to be installed in a car = size of the model/parameters (M)

· Success Rate vs overlap ratio: the percentage of frames that have a certain overlap ratio with the groundtruth result

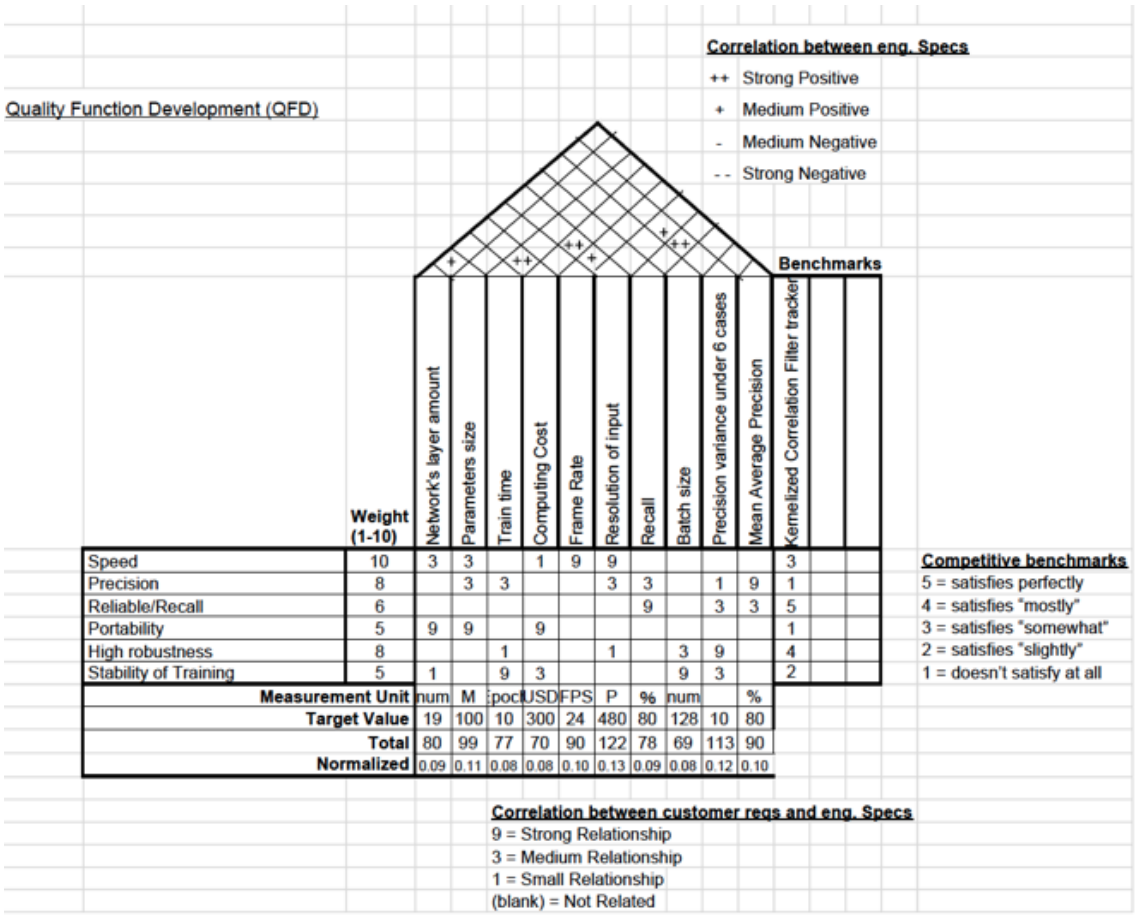

Fig 2.1 QFD

As the requirement are listed above, we find the relative factors may affect the performance including:

· Networks' layer amount: The more layers, the stronger learning ability it has. However, the speed will decrease with increasing amount.

· Parameters' size: The larger it is, the slower the model performs on PC, and while the stronger learning ability it has.

· Training time/computing cost/batch size/resolution of input: it will affect the portability also precision.

· Frame Rate/Frames per second: This should be in the tradeoff with precision.

· MAP/Precision Variances/Missing Frames: it directly measures the performance of the tracking system.

The following table contains all the engineering requirements of our project.

Specifications	Target Value
The amount of networks' layer	19
The size of parameters	100 (MB)
Training time	10 (Epoch)
Computing cost	300 (USD)
Frames per second	24 (FPS)
Resolution of input	480 (P)
Recall	80%
Batch size	128
Precision variance	10
MAP	80%

Table 2.1 *Engineering Requirements*

3. Concept Generation

We generate the concepts from two objectives of our task: locating the position of human face in the video and tracing the movement of the face. There are some tricky cases that need to be considered. For example, many or no face shows in one frame or the movement of face causes a motion blur. Through brainstorming, our team comes up with two solutions: using face detector to classify and localize human faces or using a tracker to find the trace. After further discussion, we propose advanced solutions that combine the previous two methods. Then we derive five general concepts, which will be elaborated below.

The first concept is **face detector system**. We preprocess the video stream and split it frame by frame. Then we run face detection algorithms on each frame separately. Finally we analyze the detection results and generate the output. There are several existing options, the performance of which is list in Fig 3.1.

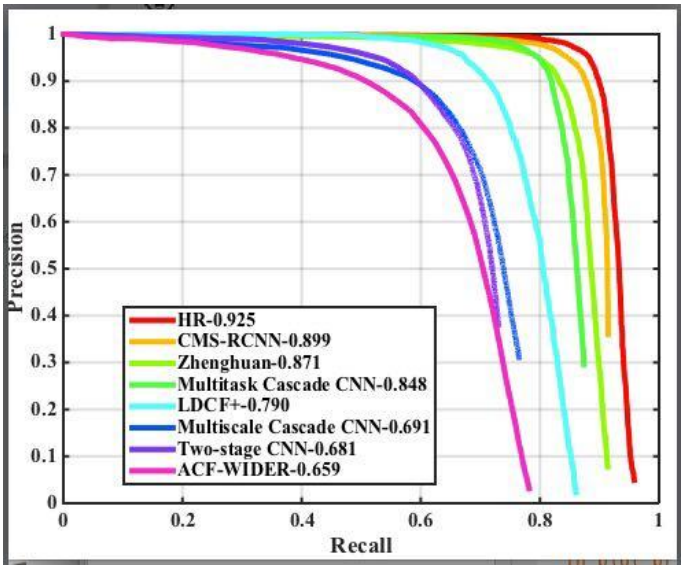

Fig 3.1 *Performance of existing detection methods.*

The second concept is **face tracker system**, which predicts the position of the face comparing to the previous frame. We input the frames consequently and record the movement of the tracking bounding box. The performance of several existing tracking algorithms is shown in Fig 3.2.

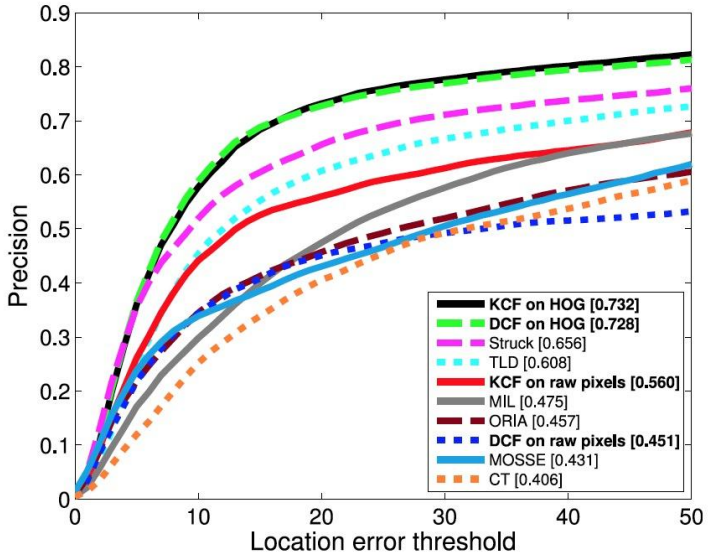

Fig 3.2 *Performance of tracking systems [3].*

To take the advantage of both methods, we try to combine face detector and tracker system. Face detection is slow but accurate, while tracking system is fast but run the risks of losing the object. Also the outputs of two methods may vary a lot with different candidates and scores. We propose different ways to combine the two systems and produce robust results.

Combined score function takes the advantages that both detecting and tracking algorithm use a score function to give or predict the position of objects. Through mathematics transform and normalization, we balance the two algorithms by generating a mixed score function, which takes into consideration the candidates from both algorithms and outputs the most likely position.

Detection assisted tracking comes up with the idea that the tracking algorithm has a faster speed, about 172 frames per second. We take its speed advantage and use face detection algorithm to fix the extreme results, such as the starting point and the lost of focus. Detecting algorithm is used to classify and localize the human face in the first frame and as a reference to adjust the results if the tracking algorithm loses its focus.

Mixed ranking is the last concept that balances between detecting and tracking. It ranks the most possible results of both methods with a threshold value. Then it adds weight to combine the rankings and outputs the most likely result. The benefit of this concept is that it is less math-based and more intuitive in the tracking process.

4. Concept Selection

We implement the five concepts in section 3 and analyze their results quantitatively. They all have strength and weakness. After consideration, we choose the fifth concept: mixed ranking as the robust face tracking algorithm.

The face detection is pretty mature in the academy. According to Fig 3.1, we choose Faster R-CNN, which has high recall and high precision compared to other detection algorithms. However, the speed is limited, which is also the problem of the first concept. As we test the demo on our machine without GPU, the speed reaches only 3-4 frames per second. Also, we consider Dlib as a candidate, which is very fast but less accurate. In our project, we need strong portability and robustness, which pure face detection cannot provide.

For the tracking algorithm, we choose KCF, which is much faster than face detection and has high precision and recall, based on the performance in Fig 3.2. But the problem of KCF is that it needs the tracking object to be located in advance. Also, if it loses the object during tracking, the results will be very poor. Since we use detector to adjust the results and we expect a real-time system, we choose KCF as our candidate.

Combined score function and detection assisted tracking are two potential way to combine tracking and detection. However, they have own weaknesses. The former requires a complex computation of the score function and brings some theoretical problems in the process. The latter is easier to implement by adding detection part to KCF code. The outcome is quick but not robust enough. We are expecting accurate results in an fast-changing environment.

Finally, we implement and test the mixed ranking and find that it is the best among all candidates. It considers the top rankings of two methods and generates the most likely position. It is faster than pure detection and more accurate than tracking. Compared with detection assisted tracking, mixed ranking provides more robust results but is little slower. We use the technique of image compression to accelerate the detection time and control the speed of output to meet the requirement, which is 24 frames per second. This concept is also very close to the ideal case, which produces a smooth trace of the human face in a video and is robust against changing environment, such as motion blur, light changing, and quick black out.

5. Concept Description

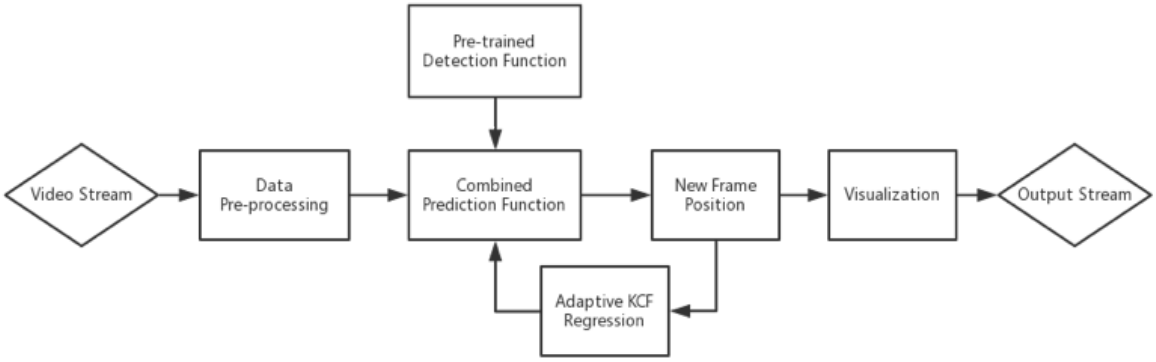

Figure 5.1 *Overview of selected concept*

The whole project is written in Python and applied MXNet framework. The video stream would be first pre-processed into a set of images with 25 FPS. The tracking is realized by combining a face detector based on Fast R-CNN and a KCF regression tracker. KCF regression is a popular algorithm for object-tracking. It can predict the next position of an object based on its current position. The weakness of KCF is that if the face disappears or being blocked from the screen for a while, KCF can not do the tracking when the face comes back. Hence, a robust face detector is used to fix these problems. The detector is based on faster R-CNN(Region-based Convolutional Neural Network) Algorithm. For each frame, the detector would classify and localize possible faces, so when KCF failed to follow the face, it can re-start from the position given by detector. The output position would be visualized by a bounding box on video stream.

5.1 Data preprocessing

To achieve a real-time tracking, a data preprocessing subsystem is needed to connect the computer camera and the tracking system. The major goal of this part is to get access to the camera video stream, capture every new frames and pass the frames into the tracking system.

5.2 Detection function

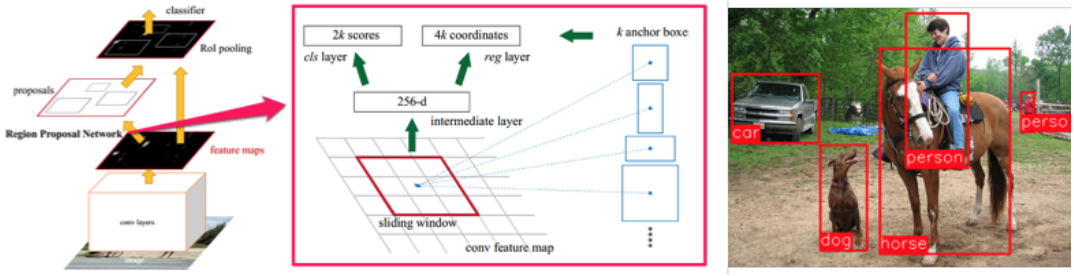

Figure 5.2.1 *Realization of faster R-CNN & Sample result[4]*

The model used for detection is called faster R-CNN(faster Region-based Neural Network). The image will be first process into a feature map which is actually a multidimensional matrix that contains all significant information of the origin image. Then for each position of the feature map, the function would propose several anchor boxes, which are bonding boxes with different sizes and shapes(see Figure.5.3.2.2). Each anchor box then would be classified as a specific object or background. Finally, for all anchor boxes that have been realized as object, function would generate corresponding bounding boxes on the origin image for result visualization. To save time, we choose to use a pre-trained model called “zhenghuan-0.871” named after our team leader Huan Zheng. There is also an alternative detection model, which is the open-source dlib face detection library. The dlib detection have a significantly higher speed compared with Faster-RCNN, and can have real-time detection even on a computer with no GPU. However, the accuracy of dlib is lower than Faster-RCNN.

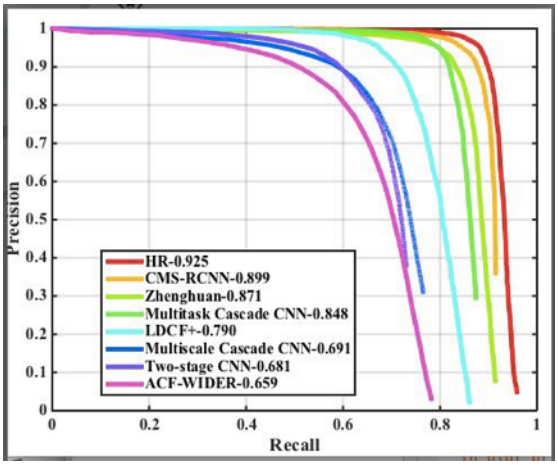

Figure 5.2.2 *Precision ranking of different model*

5.3 The modified KCF tracker

The kernelized correlation filter (KCF) is the most popular tracker model that people using in the recent years. The main advantage of this model is high accuracy tracking with a very high speed (around 200 fps). As mentioned in the paper[3], KCF is an adaptive online update model, which means it keeps update the model each frame and predict the location of target in next frame based on this model as shown in Figure 5.3.1:

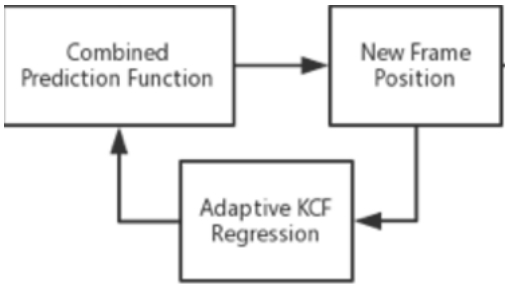

Figure 5.3.1 *KCF flow chart*

The prediction model that KCF are using is called the kernelized correlation regression, which is a common used model in traditional machine learning. "Kernel" is a classic trick that can help people build high dimensional feature by simply compute the kernel matrix which includes the inner product of each pair of samples. Then, with the high dimensional features computed by the kernel matrix, we can have a regression based on those features and predict the best sample. The main contribution of KCF is that it transforms the kernel matrix into fourier space and gets a elegant closed form regression solution that can be computed efficiently (no matrix multiplication but only point wise multiplication). The closed form solution can be fast computed, which gives the KCF the ability to compute and updates its regression model in each frame with very high speed.

However, the KCF is a pure online model and does not store pre-trained model, which means it can track anything without determine what it is tracking. To have a stable tracking system that optimize for human face, the KCF needs to be modified.

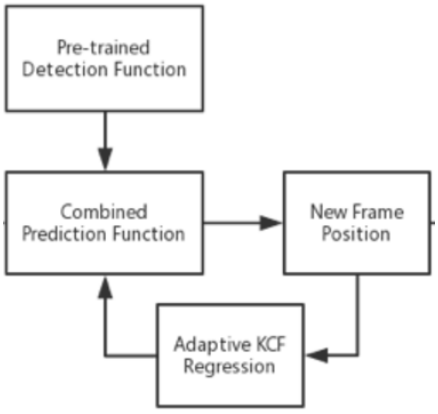

Figure 5.3.2 *Modified KCF flow chart*

- $Score_{Final} = (Score_{KCF} + Score_{Detection}) * \frac{Size_{overlapping}}{Size_{KCF} + Size_{Detection}}$

Figure 5.3.3 *Modified score function*

As shown in figure 5.3.3.3, combined with the score of the detector, we designed a new prediction function to generate the location of target. We selected 10 best results from tracker and 50 best locations from tracker, calculate the score based on the modified prediction function and select the best location.

5.4 Visualization

After get the predicted position of the object, a bounding box would be generated on the current frame. Then the frame would be played through the output video stream as the final result. Numpy and opencv library would be used to realize the visualization.

6. Parameter Analysis

After analysis the specification, we believe that the tradeoff between the accuracy and speed will be our major concern. The required accuracy will not be a major concern since the KCF and Faster R-CNN models we are using guaranteed a basic accuracy, and our design solved the robustness issue under special scenes. The major issue in our project design is how to guarantee the real-time speed (>24 fps) with certain accuracy.

The followings are some specific parameters we picked:

Detector input size: For Faster R-CNN model, speed is a big issue. It will require 1.5~2s to compute a detection in an image with size 720*1280. Since in our task we are caring more about the detection score that help us determine confidentiality of the existence of a face rather than the precise location of the bounding box, we zoom in the image into size of 72*128 and achieve a 0.3s/frame detection speed on regular computer and less than 1/24 s speed on GPU, which guarantee a real-time detection. Thus our detector input size is finalized to be 72*128.

Detector threshold : We choose 0.3 to be our detector threshold. The threshold is used to reduce the output size and only output candidate faces that have score above the threshold. Usually the industry use 0.7 for face detector, however here we decrease it to 0.3 for the following reasons:
1, For in-car situation, the light may not alway be good for detection (considering driving in a tunnel). The score of a face can be lower to 0.4-0.5 when the light condition is not ideal.

2, 0.3 Threshold does not significantly increase the candidate number(still usually less than 10 candidates), and most of the candidates are still faces. Thus a low threshold does not have much drawback here.

Detection interval: The detector can have a real-time detection with the help of GPU, however if the detection is running on regular computer without speedup of GPU, the fps will be only around 2-3. To let our tracking also work on regular computer, we develop an asynchronous scheme that runs the detection once every few frames rather than each frame. To have a smooth output stream, we have tried several detection interval, and finally we think the interval of 50 frames guaranteed smooth tracking on regular computer.

Tracking lost grace period: Due to the construction of our tracking system, the face detector will have lost it detection of face if there is a blocking above the target's face. To keep tracking under this case, we design a grace period to let the tracker keep tracking while the face is lost for a while and wait for the reappear of a face (the remove of the blocking object). Due to our experiment, the blocking usually have time 1-3s for in-car situation, thus we have set the grace period to 100 frames (4~5s).

7. Final Design

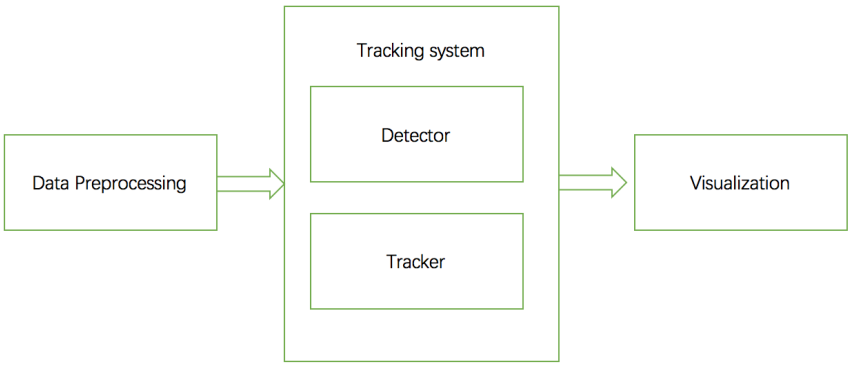

Figure 7.1 *Overview Layout*

Our tracking system can be decomposed into 3 major sub-systems:

1. The data preprocessing system is in charge of connecting the camera stream and the tracking algorithms.
2. The tracking systems is in charge of providing the tracking result (the target location in each new frame). The tracking system can be again divided into 2 sub-systems: the detector and the tracker. The detector will detector the human face in frame and the tracker will track the location of target based on the previous frames. We designed a new score function combines those results together and can make the prediction based on those results.
3. The visualization system will take the tracking results and annotate the target with a box in the output stream.

Since the project is a pure software project, our implementation is the final design of the project. The difference between our implementation and the design is, on a computer without GPU acceleration, the Faster RCNN detector requires 0.1-0.5s for each detection, which largely delayed the real-time tracking sequence. Therefore in our implementation there are two mode: if the computer have the GPU acceleration, the Faster-RCNN detector will be called and provide the stable and high accuracy detection result. If there is no GPU, the dlib detection will be called and provides a relatively low accuracy but still real-time detection result.

8. Manufacturing Plan

The project is written in python due to the convenience of opencv package and MXNet library.

8.1 Data preprocessing

The connection between the camera and the tracking system is done by using the opencv python library *cv2.VideoCapture(0)* function.he preprocessor is written in Python with functions in opencv package. The core function is *cv2.imwrite()* which would save the current moment of the video into a image. Calling the function during the entire video range and the output would be a set of image.

8.2 Detection function

The detection is based on "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks"[3]. However, we only follow the concept of this paper, and made our modification on the "Faster R-CNN in MXNet with distributed implementation and data parallelization"[6]

The pipeline of Faster R-CNN is shown below:

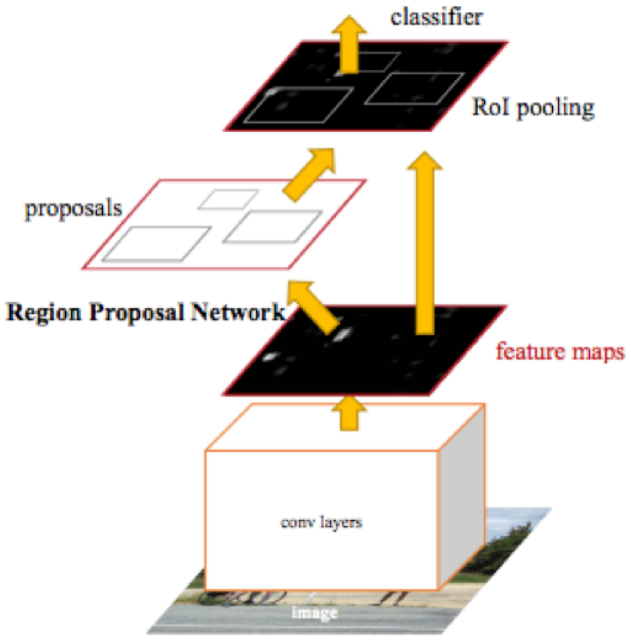

Figure 8.2.1 *Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.[3]*

However, although the Faster R-CNN is very powerful on general object detection, it still needs to be modified for face detection. Faces are much smaller and have various pose. We fine-tune and retraine the Faster R-CNN with Wider Face Dataset.

Figure 8.2.2 *Wider Face Dataset provides us with faces under different cases. It is vital for the model robustness. [7]*

After we change the data interface, write down the new configuration (Network Parameters Initial Setting , and train the model, the detector now has the ability to find faces.

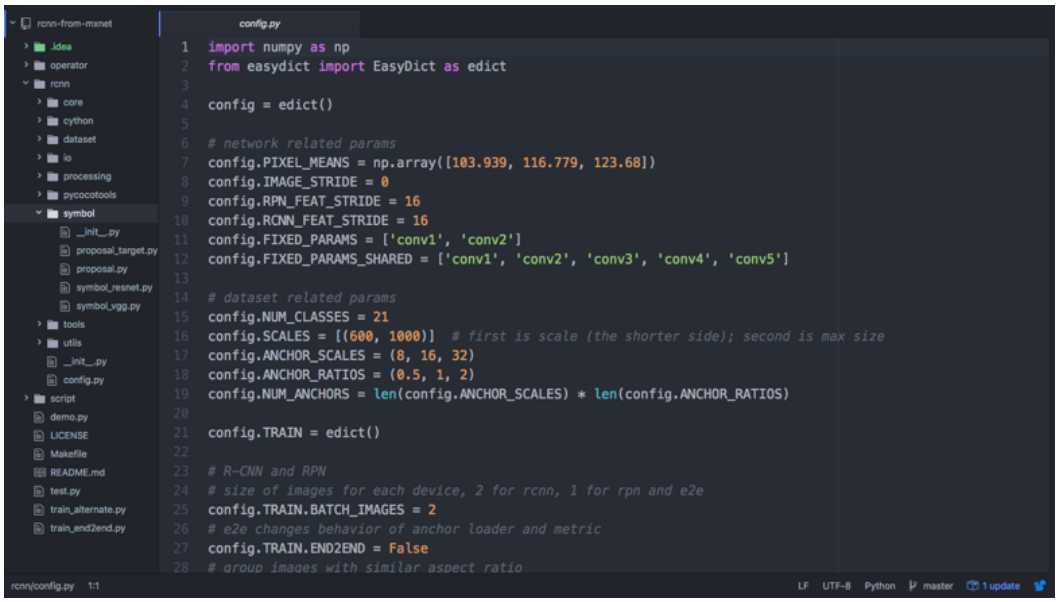

Figure 8.2.3 *All the codes files of detector function and its configuration (Network Parameters).*

8.3 The modified KCF tracker

Our implementation of KCF tracker is based on the uoip's kcf python implementation on github[1]. The pseudo code of uoip's kcf function is shown below:

Algorithm 1: Matlab code, with a Gaussian kernel.
Multiple channels (third dimension of image patches) are supported. It is possible to further reduce the number of FFT calls. Implementation with GUI available at:
`http://www.isr.uc.pt/~henriques/`

Inputs
- `x`: training image patch, $m \times n \times c$
- `y`: regression target, Gaussian-shaped, $m \times n$
- `z`: test image patch, $m \times n \times c$

Output
- `responses`: detection score for each location, $m \times n$

```
function alphaf = train (x, y, sigma, lambda)
  k = kernel_correlation(x, x, sigma);
  alphaf = fft2(y) ./ (fft2(k) + lambda);
end

function responses = detect (alphaf, x, z, sigma)
  k = kernel_correlation(z, x, sigma);
  responses = real(ifft2(alphaf .* fft2(k)));
end

function k = kernel_correlation (x1, x2, sigma)
  c = ifft2(sum(conj(fft2(x1)) .* fft2(x2), 3));
  d = x1(:)'*x1(:) + x2(:)'*x2(:) - 2 * c;
  k = exp(-1 / sigma^2 * abs(d) / numel(d));
end
```

Figure 8.3.1 *Pseudo code for kcf[3]*

The modification of the KCF is focused on the score function shown below in figure 5.4.3.2 as described in section section 5.3.3.

```
class FRCNNDetector:
    def __init__(self):
        #result array
        self.result = []
        #best score
        self.best_result = 0
        #best score location
        self.best_location = [0,0,0,0]
        #threshhold to be face
        self.th = 0
        self.result_num = 0
        self.overlap_th = 0.5
        self.detector = demo.detector()

    def update(self, image):
        #faster rcnn result
        #image = image[0:-1:10,0:-1:10,:]
        #print np.shape(image)
        self.result = self.detector.detect(image)
        print(self.result)
        #sort the position in decending order
        result = self.result
        self.result_num = len(self.result)
        #print self.result_num
        if (self.result_num > 0):
            self.result = sorted(result, key=lambda result:result[4],reverse=True)


    def exist_face(self):
        if (self.result_num != 0):
            return True

    def best_face(self):
        if(self.exist_face()):
            return [int(self.result[0][0]),int(self.result[0][1]),int(self.result[0][2]-self.result[0][0]),int(self.result[0][3]-self
        else:
            return 0

    #detector box, tracker box
    def overlape_ratio(self,box1,box2):
        #print box1
        #print box2
        x_overlap = float(box1[2] + box2[2] - (max(box1[0]+box1[2],box2[0]+box2[2]) - min(box1[0],box2[0])))
        y_overlap = float(box1[3] + box2[3] - (max(box1[1]+box1[3],box2[1]+box2[3]) - min(box1[1],box2[1])))
        if(x_overlap < 0):
            x_overlap = 0
        if(y_overlap < 0):
            y_overlap = 0
        ratio = (x_overlap * y_overlap) / (float(box1[2]) * float(box1[3]) + float(box2[2]) * float(box2[3]))
        #print ratio
        return ratio
    #determine whether face or not
    def is_face(self,box):
        for i in range(self.result_num):
            if(self.overlape_ratio( [int(self.result[i][0]),int(self.result[i][1]),int(self.result[i][2]-self.result[i][0]),int(self.
                return 1
        return 0
```

Figure 8.3.2

The full access of our code will be appended in the appendix.

8.4 Visualization

After getting the predicted position from the combined tracking function, the position would be draw as a bounding box using the python numpy package. Then the image with the bounding box then will be played as a video stream using opencv package. The trajectory is also shown as a series of circles linked by a white line.

Figure 8.4.1 sample results of tracking box and trajectory

9. Test Result

9.1 Test procedure

As we mentioned in the first DR, the evaluation is done abode the Visual Track Benchmark [2]. Here is the standard protocol to evaluate on this benchmark according to their website[2]:

- Tracking starts from one initial bounding box in a start frame.
- The start frame may not be the first frame of the sequence.
- DO NOT use manually-tuned, different parameters for individual sequences.
- Any additional information, such as sequence name or attributes, may not be used in determining the parameters.
- It is allowed for a tracker to automatically adapt its parameters only using features from input images.
- The tracking result is a sequence of bounding boxes at each frame.
- Affine or similarity parameters are converted into bounding boxes for comparison.
- Use the AUC (area under curve) to show the tracker's overall performance.
- The success plots are preferred over the precision plots, since precision only uses the bounding box locations, and ignores the size or overlap.

Based on this protocol, we have made several modification in our evaluation according to the purpose of our project:

- Since the benchmark is designed for general object tracking, and our project only focuses on the fact tracking, we only picked several face tracking videos inside the benchmark for our evaluation.
  The selected videos includes the following scenarios:

OCC: Occlusion – the target is partially or fully occluded.
FM: Fast Motion – the motion of the ground truth is larger than tm pixels (tm=20).
MB: Motion Blur – the target region is blurred due to the motion of target or camera.
OV: Out-of-View – some portion of the target leaves the view.
BC:Background Clutters – the background near the target has the similar color or texture as the target.

- As suggested in the protocol, we have used the success rate vs. overlap ratio to show our accuracy. The modification over the overlap ratio formula is made to serve better for our project purpose. Since here we care more how well the tracking is followed rather than how accurate the rectangle is close to the hand-labeled groundtruth rectangle/ The modification is, here we use a modified overlap formula:
    - Overlap ratio = overlap(predict region, groundtruth)/smaller(predict region, groundtruth)

9.2 Test result

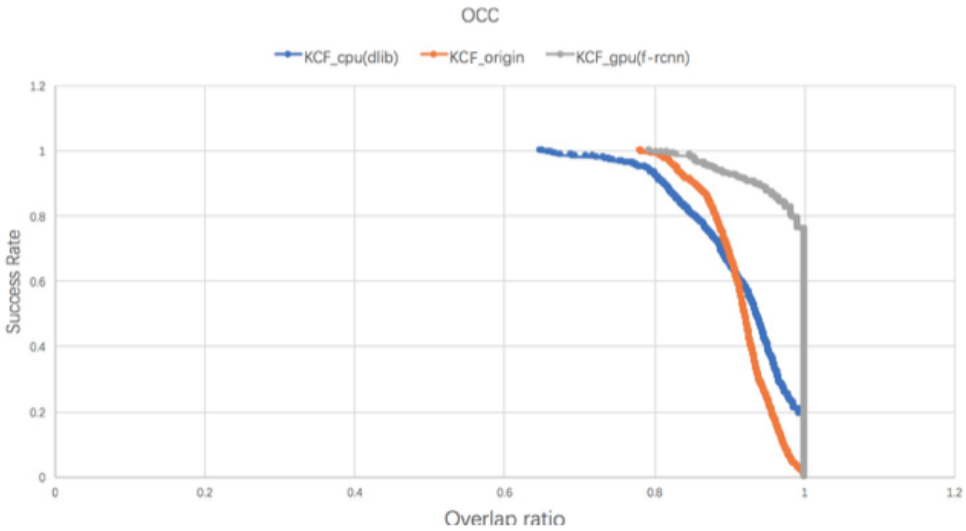

Figure 9.1.1 Success rate vs. overlap ratio in OCC test

OCC: Occlusion – the target is partially or fully occluded.

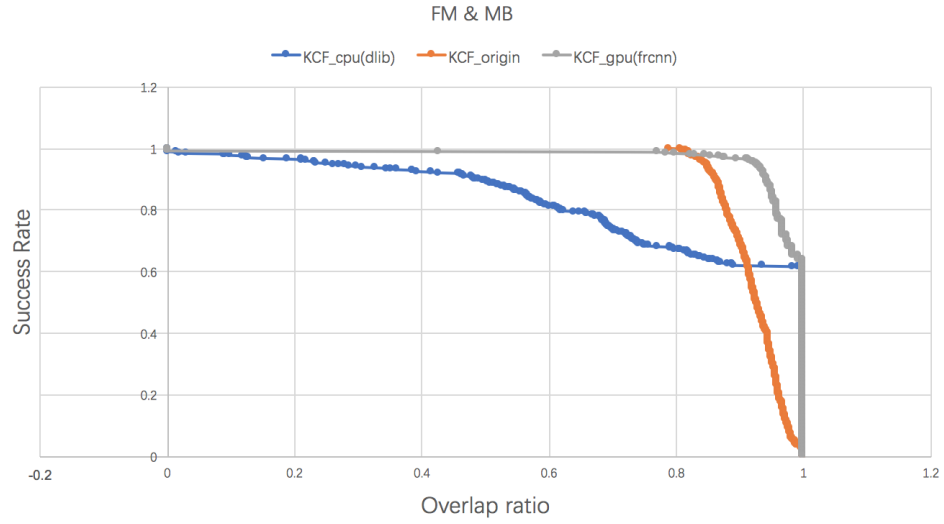

Figure 9.1.2 Success rate vs. overlap ratio in FM & MB test

FM: Fast Motion – the motion of the ground truth is larger than tm pixels (tm=20).

MB: Motion Blur – the target region is blurred due to the motion of target or camera.

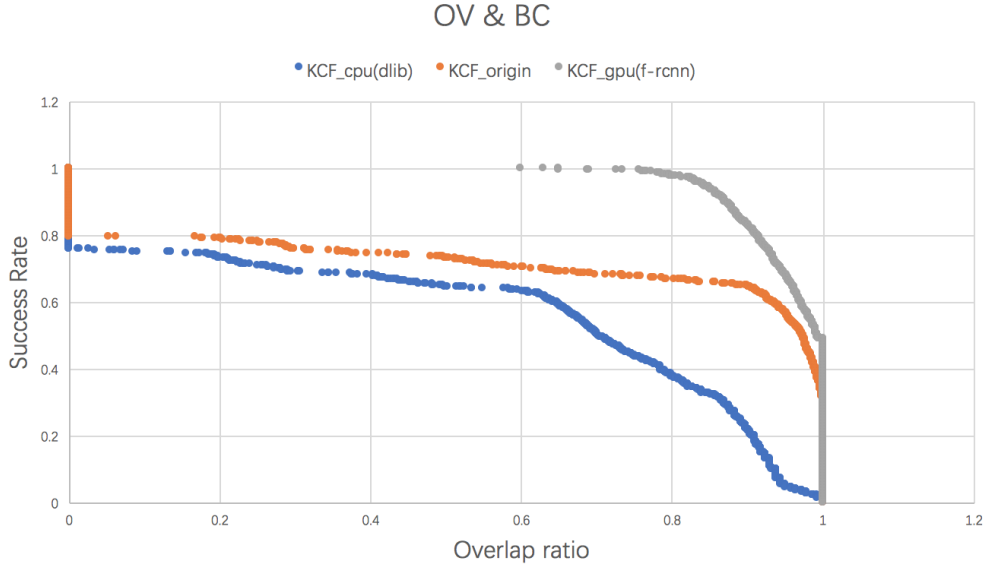

Figure 9.1.3 Success rate vs. overlap ratio in OV & BC test

OV: Out-of-View – some portion of the target leaves the view.

BC:Background Clutters – the background near the target has the similar color or texture as the target.

	KCF_cpu	KCF origin	KCF_gpu
FPS (24)	~50	172	~50 with gpu; ~1 with cpu
Frame Loss rate	7%	6%	1%
Precision (70%) (under 60% overlap ratio)	76%	86%	95%
Variance (10%)	3.4%	3%	0.3%

Table 9.1 Result Summary

9.3 Discussion

The plots show the difference performance between the three tracker. In a success rate-overlap ratio diagram, the curve that close to the right upper corner is better (since a better tracker should have more frames to have higher overlap ratio).

Therefore, from the plots we can tell that in the test of five scenarios (OCC, FM, MB, OV, BC), the KCF with faster RCNN (gpu version) outperform the original KCF. The performance of KCF with dlib (cpu version) does not beat the original KCF, although they are not too far. However, since the benchmark is designed for pure tracking task, pure KCF does not solve the problem on how to initial a tracking target. The KCF with dlib solve the this by adding the detection function. Therefore, for the purpose of practical use, the KCF with dlib should be more prefered than the original KCF.
In conclusion, through the evaluation, the performance of KCF with faster RCNN beats the performance of original KCF. The performance of KCF with dlib does not beat the original KCF but the result is still acceptable for practical use with a computer with no GPU installed.

10. Engineering Change Notice (ECN) - Change in Design since Design Review #3

During the test, we find that when the system runs on the CPU, the output video stream did not respond smoothly since the faster R-CNN is supposed to run on the GPU while the calculation speed of CPU is not enough to run the model at full-speed. Hence, instead of change the structure of the whole system, we decide to change our detector model to *dlib-based* when the system is deployed on an CPU. Unlike faster R-CNN, the dlib detector process faster with relatively low accuracy as the trade-off.

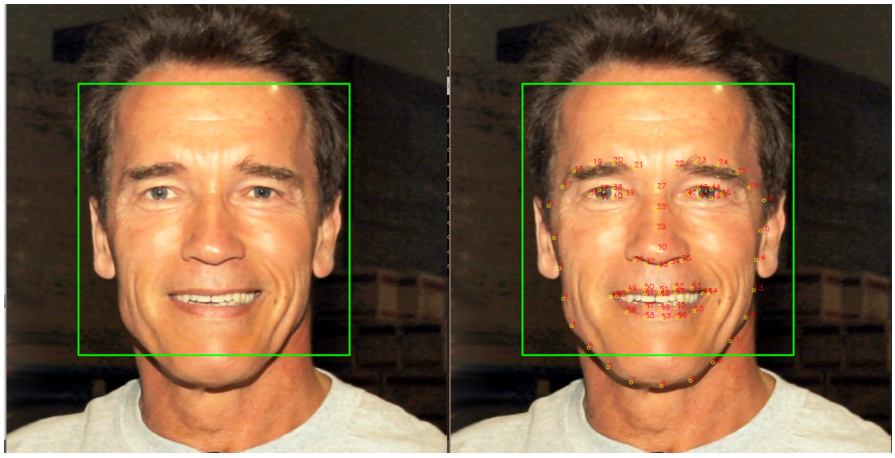

Figure 10.1 Sample output of *dlib* face detection model[8]

11.Discussion

Our project aims to offer a new in-car interaction approach. A gesture tracker will be the perfect solution for this task, which is the original idea from the sponsors. However, after the literature research and further discussion with sponsors, we reach a consensus that figure out a face tracker instead of gesture tracker due to the limitation of time, hardware and data. Now, a Real-time Illumination Robust Face Tracker under Ensemble Learning works well and fit almost all the requirement from sponsors. Furthermore, it has great potential for general object tracking and is portable on a car.

In this project, we made several innovative progresses. First, redefine the pipeline of the system. Therefore, we propose a new method that combining Detector and Tracker together following the concept of Ensemble Learning. When we are talking about tracking, we are always only focusing on tracker. However, considering the inside environment of a car, we realize the motion blur and sudden illumination change will significantly lower the stability of tracker. Tracker is running risks of losing target; and once the target is lost, then nothing will be tracked in the rest time. We make a great difference from tradition tracker system, we use detector to assist the tracking. It helps a lot especially under some extreme cases such as occlusion, sudden illumination change, disappearance and reappearance of target. While the tracker will keep the system stable when the object is in dark environment or motion blur occurs. Under these cases, detector is insensitive but tracker has great performance. Thanks to the combination of Detector and Tracker, our project has high robustness for in-car environment.

Another strength of our project is its fast speed. It can work in Real-time on one's personal computer. We have two pattern of this face tracking system. One is that we use Faster R-CNN, which have high precision and high robustness but large size and relatively low speed. This also can work in real-time but not so fast and has higher hardware requirement. We also apply DLib Face detection to assist tracking, the speed is extremely fast (100x to Fast R-CNN) and it loses some precision. We can choose from these two patterns and make the tradeoff according to industrial application requirement.

However, even though we almost meet the entire requirements from sponsors, there is

still much space to improve. The weakness of our current model is the structure is too large. Faster R-CNN is designed for general object detection. The structure is not compact enough which leads to the high hardware requirement. The possible improvement is that we use some common face detector instead such as DLib or the model from paper “Finding Tiny Face” issued by CMU students. If we have GPUs and much time, change the feature network of Fast R-CNN from VGG16 to Inception v3. It will definitely reduce the size of the network and the parameters.

12.Recommendations

FIrst, due to the limitation of time and hardwares, we only employed our team workers’ personal computer (Mac) to implement both detector and tracker for training, validating, testing, and demoing. We strongly recommend that use GPUs to conduct all the process, especially for training. It will let the model become more robust by learning more data, save much time when validating and testing (fine-tuning), and make demo faster.

Second, our model is based on Faster R-CNN, which is a general object detector. It is designed for multiple objects instead of human faces. We just fine-tuned it using Wider Face Dataset. Although the result turns out to be quite amazing, we believe it will be even more surprising and robust if we use a delicate and compact model designed for face.

Third, the data collection is a hard task. in this project, all of our teammates have to simulate the environment, become a model and take videos by ourselves. Although we use some common dataset and benchmark to train detector, tracker still lack huge amount of data. The variety of dataset will largely affect the robustness of system. We strongly recommend the sponsors or next team collecting more data.

Last but not least, the pipeline of our team is detector plus tracker. There are still many ways to combine this two components. Not only the score function but only the loss function of the system can be improve. It will largely affect the efficiency of training.

13. Conclusions

In this project, we are required to design a face tracker which has high robustness under some extreme cases such as sudden illumination change, motion blur, and focus lose, etc. The tracker is also supposed to be able to work in Real time. Furthermore, it should have great potential and portable for different tracking targets. Taking all three requirements into account, we propose a system under Ensemble Learning, which it combines Face Detector based on Fast R-CNN and the Object Tracker based on KCF. The Detector will classify and localize the face position at the very first frame and when the occlusion, motion blur, disappearance and reappearance, and sudden illumination change occurs. The Tracker will keep tracking and show the path the target (face), even if the environment is quite dark and the detector cannot work normally. The robustness is assured by the combination. We tested our system under several extreme cases. From the video and the graph in the previous part, both the precision and recall is high. In addition to the robustness, we offer two different tradeoff plan by applying two different detector. Faster R-CNN has high precision and can work in Real-time. DLib has 100x faster speed, smaller size but a relatively lower precision. These two options will largely benefit the portability of the system. If we change the detector, the tracking system will be easily switched to tracking different targets along with the detector. Our design has high robustness, fast speed and great portability. In such as limited time, it reaches or even beyond our expectations.

14.Acknowledgements

Sponsor: Yuzhu Wang, Riguang Bai, Jiawei Liu from UAES
Instructor: Chengbin Ma, Yunlong Guo from UM-SJTU Joint Institute
KCF model: Henriques J F, Caseiro R, Martins P from University of Oxford
Faster RCNN: Ross Girshick From Facebook
KCF python implementation: uoip from MIT
Benchmark: Y. Wu, J. Lim, and M. H. Yang from University of California, Merced

15. Information Sources and Reference List

[1] https://github.com/uoip/KCFpy
[2] Y. Wu, J. Lim, and M. H. Yang, “Online object tracking: A benchmark,” in Proc. Comput. Vis. Pattern Recognit., 2013, pp. 2411–2418.
[3] Henriques J F, Caseiro R, Martins P, et al. High-speed tracking with kernelized correlation filters[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015, 37(3): 583-596.
[4] Ren, Shaoqing, et al. "Faster R-CNN: Towards real-time object detection with region proposal networks." *Advances in neural information processing systems*. 2015.
[5]Yanxia, Hongren Zhou, and Zhongliang Jing. "Visual tracking and recognition based on robust locality preserving projection." Optical Engineering 46.4 (2007): 046401-046401.
[6]Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. In Neural Information Processing Systems, Workshop on Machine Learning Systems, 2015
[7] Yang, Shuo and Luo, Ping and Loy, Chen Change and Tang, Xiaoou.” WIDER FACE: A Face Detection Benchmark” 2015
[8]http://www.codesofinterest.com/2016/10/getting-dlib-face-landmark-detection.html

Appendixes

Concepts: All of five concepts we have mentioned in the concept generation section so that there is no more concepts here.
Code: Our code can be access through github repo:
https://github.com/michiganleon/CapstoneTracking
Component Bill: Totally software project. The only hardwares are the computer and GPUs, which can be determined by the users.

Component	Bill
A computer with camera	5000-50000 RMB
A Gpu (optional)	500-20000 RMB

Change since DR3
During the test, we find that when the system runs on the CPU, the output video stream did not respond smoothly since the faster R-CNN is supposed to run on the GPU while the calculation speed of CPU is not enough to run the model at full-speed. Hence, instead of change the structure of the whole system, we decide to change our detector model to *dlib-based* when the system is deployed on an CPU. Unlike faster R-CNN, the dlib detector process faster with relatively low accuracy as the trade-off.

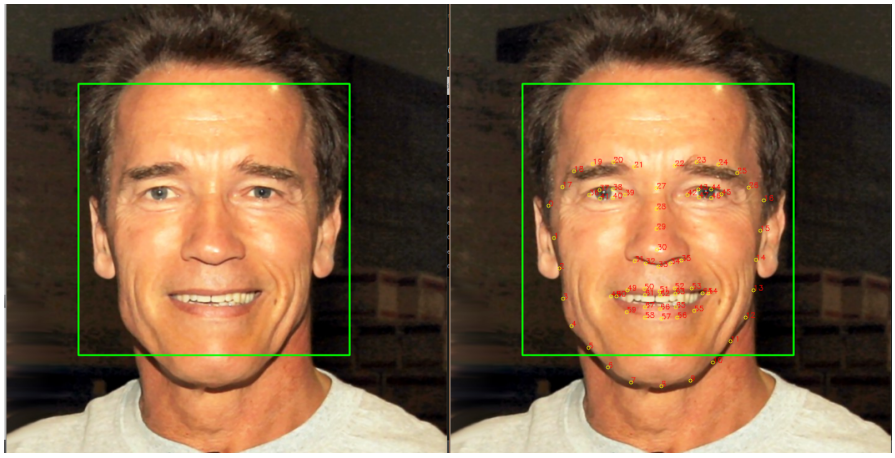

Figure 10.1 Sample output of *dlib* face detection model[8]