TEAM MEMBER: Chen Gang, Ye Xin, Hao De Hong, Lu Hong Qing, Xia Miao Juan

# Machine Vision for a Selfie Flight by Drone

Machine Vision

2017/6/5

# Content

# 1. Project Overview

## 1.1 Background

Multi-rotor unmanned aerial vehicle is a type of popular flying robot. It can be utilized as an excellent platform to fulfil various application scenarios. One of the most common application is to taking aerial photos. Due to UAV's flexible flight capability, it can be used to take videos via a vision-based feedback control system on a desired target. In this work, we are engaged to establish a system to make the UAV take videos of a user like a journalist totally on its own. Whenever the user walks or turns, the UAV should always keep the user in the center of the image and always try to face towards the front side of the user like a photographer.

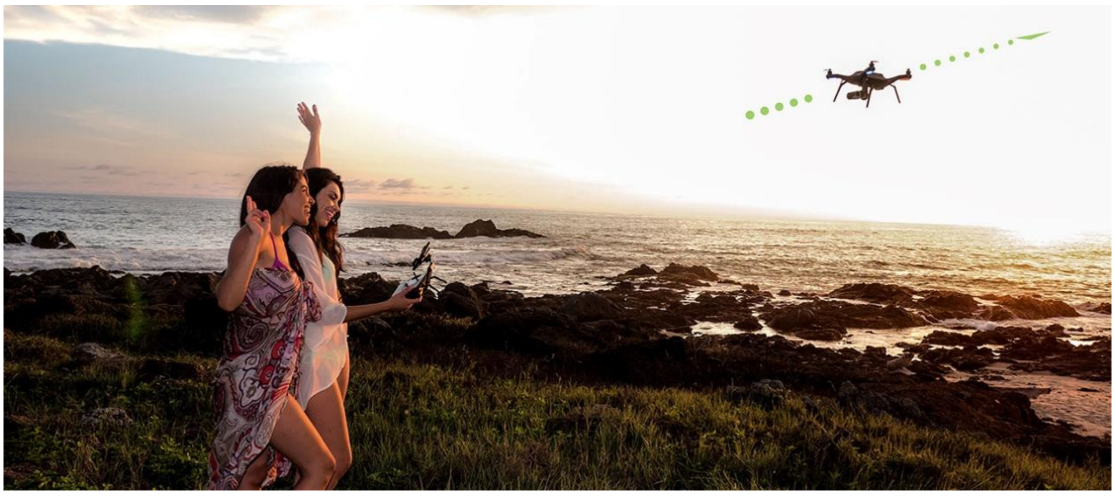


Figure 1.1 Application scenario

The on-sale commercial multi-rotor UAVs having the similar abilities are "Mavic" made by DJI and "Passport" made by hover camera. "Mavic" can follow a pedestrian or any other moving target at a relatively large height. It can capture the target and the scene but misses the details of people because of the long distance. "Passport" can detect human face and follow the face. Also, it can finish the similar work like "Mavic". However, there is no UAV products that can achieve our target, which requires fusion of human detection, target tracking, face detection and face angle measurement.

## 1.2 Project

The project is of great difficulty both in machine vision algorithms and UAV control algorithms. The whole system requires project management, system design, algorithms develop and software programming.

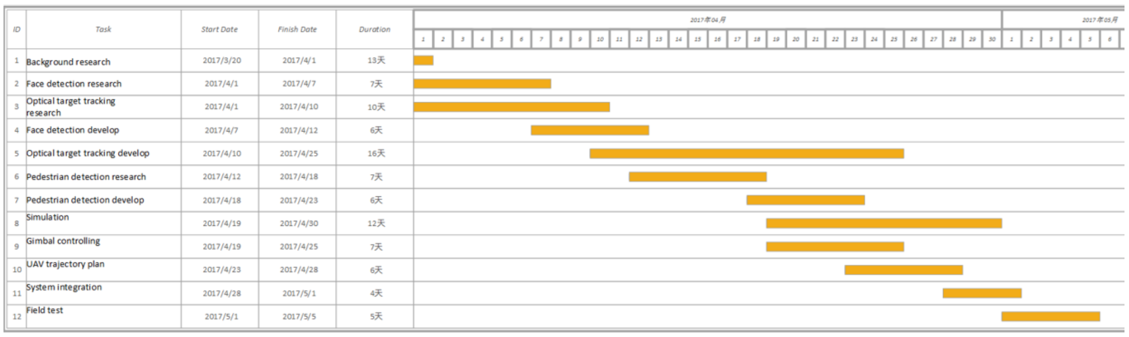
Our target is to keep the UAV

a) *Facing you*
b) *As high as your face*
c) *Keeping a constant distance from you*
d) *Keep tracking you when :*
   1. Another person appears in the screen
   2. When you are turning

The task alignment table is shown as follows

| **Task** | **Details** | **Person in charge** |
|---|---|---|
| Background research | Existing products & Latest research | The whole group |
| Face detection | Front faces and profile faces, distinguish right and left face | Dehong Hao, Hongqing Lu |
| Pedestrian detection | Detect pedestrians | Dehong Hao, Hongqing Lu |
| Optical target tracking | Track a target in pixels | Miaojuan Xia, Xin Ye |
| Gimbal controlling | Move the camera towards the target | Xinye |
| UAV trajectory plan | Fly the UAV to the optimal position smoothly | Xinye |
| System integration | Combine the components | Gang Chen |
| Face angle measure | Face angle measure | Gang Chen |
| Simulation | Simulate the system | The whole group |
| Field test | | The whole group |

The time line is given in the following



| ID | Task | Start Date | Finish Date | Duration |
|---|---|---|---|---|
| 1 | Background research | 2017/3/20 | 2017/4/1 | 13天 |
| 2 | Face detection research | 2017/4/1 | 2017/4/7 | 7天 |
| 3 | Optical target tracking research | 2017/4/1 | 2017/4/10 | 10天 |
| 4 | Face detection develop | 2017/4/7 | 2017/4/12 | 6天 |
| 5 | Optical target tracking develop | 2017/4/10 | 2017/4/25 | 16天 |
| 6 | Pedestrian detection research | 2017/4/12 | 2017/4/18 | 7天 |
| 7 | Pedestrian detection develop | 2017/4/18 | 2017/4/23 | 6天 |
| 8 | Simulation | 2017/4/19 | 2017/4/30 | 12天 |
| 9 | Gimbal controlling | 2017/4/19 | 2017/4/25 | 7天 |
| 10 | UAV trajectory plan | 2017/4/23 | 2017/4/28 | 6天 |
| 11 | System integration | 2017/4/28 | 2017/5/1 | 4天 |
| 12 | Field test | 2017/5/1 | 2017/5/5 | 5天 |

## 1.2. Framework of the System

The hardware system is AR.Drone. It is a remote controlled helicopter built by the French company Parrot. It is equipped with a vertical camera and a front camera. The vertical camera detects the fabric of the ground patterns and thereby tracks the movement of the vehicle. The front camera is utilized to see and follow a person. With a Wi-Fi interface, AR.Drone establishes communication with a PC to send video data and to receive control signals. The machine vision algorithms run on the PC.

The framework of the machine vision software system is built based on Robot Operating System (ROS as abbreviation). In such a framework consisting of multiple threads (nodes) and information channels (messages), it is simple to coordinate the tasks such as human detection, tracking, face detection and vehicle control simultaneously. Beside this basic functionality, ROS also provides the following advantages that facilitate programming, testing and realization.

First, this system minimizes the hardware needs. It requires no more than a mass produced vehicle and a PC. No additional on-board computer, gimbals or customized flight controller are necessary. All machine vision and control tasks are visualized and accomplished on the PC. This makes it easy to debug because the bugs exist nowhere else but on the PC.

Second, ROS features a decentralized system structure. Nodes are executed separately at their own paces. Failure of a single node may not cause a total crash of the system since other nodes are still able to run on. Such a distributed structure also accommodates an easy cooperation between team members, as one takes charge of the programming and debugging of one node. They only need to coordinate the messages communicating between nodes.

Third, ROS offers several other convenient accessories. With the help of YAML files, it is much easier to tune the parameters without recompiling the C++ code. Launch files not only start several nodes at the same time, but also assign the start-up options such as the paths of configuration files or libraries. Thus the whole software system can be transplanted to other computers with minor modifications.

## 1.3. Detailed Design of the Software Framework

In accordance to the functional design, the tasks of the software are distributed as follows. 1. Receive and publish the video data as well as the flight status; 2. Find human figures in the picture and publish its region of interest; 3. Track the movement of the human figure, and then publish its position and height; 4. Find and publish the face within the region of human figure; 5. Decide and publish the orientation of the face; 6. Collect the data of human position, height and orientation, and then control the vehicle to fly to the front at a distance towards the person.

The topology is shown in the following diagram.

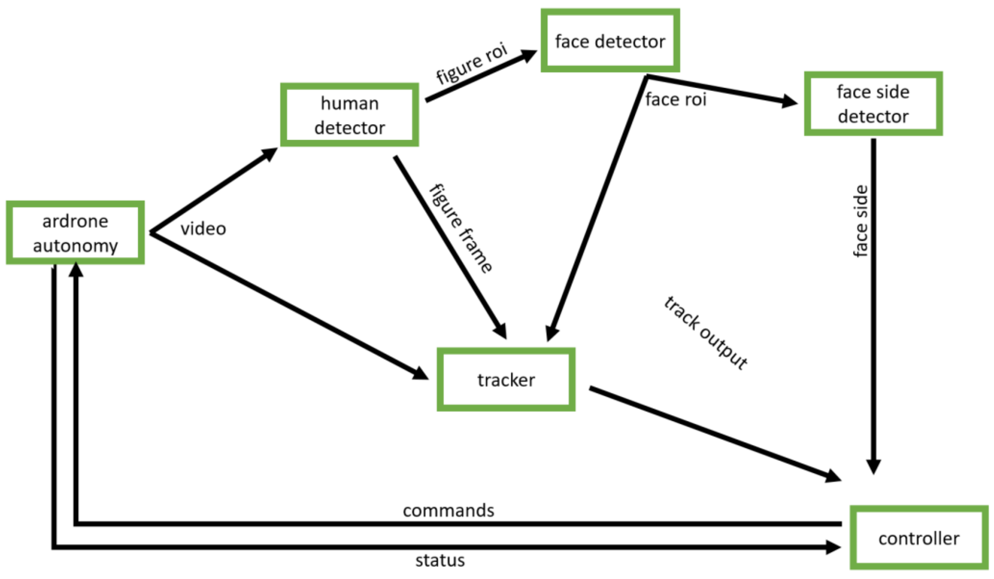


**Fig. Topology of Nodes and Messages**

The detailed functions of each of the nodes are listed in the following table.

**Tab. Detailed Function of Nodes**

| Node Name | Functions | Subscription | Publication | Contents | Other Interfaces |
|---|---|---|---|---|---|
| ardrone autonomy | 2-way link to the vehicle, remote controller of the vehicle, publisher of camera data and flight status | commands, from "controller" | video, to "human detector", "tracker" | original moving pictures | Wi-Fi connection with AR.Drone |
| | | | status, to "controller" | current attitude and position | |
| human detector | enframes the human figure and finds out its position and height in the picture | video from "ardrone autonomy" | figure roi, to "face detector" | the reduced region for face search | a human figure library |
| | | | figure frame, to "tracker" | the frame to initialize and to correct tracking | |
| tracker | track the movement of the human figure, fuse with human detection to get smoothed position, fuse with face detection to get the height | video from "ardrone autonomy" | track output, to "controller" | position of the human figure in the picture, height of the human figure to be understood as distance | N/A |
| | | figure frame from human detection | | | |
| face detector | find out the face inside the given region of human figure | figure rio from "human detection" | face roi, to "face side detector" | the reduced region for face side determination | a face library |
| face side detector | decide if the face taken is front-, left- or right-sided | face roi from "face detector" | face side, to "controller" | the 3 sides of face with 3 discrete digits | a training library |
| controller | adjust the distance towards the person according to height data, adjust yaw to make the figure at | status, from "ardrone autonomy" | commands, to "ardrone autonomy" | x and y velocity commands, yaw rate commands | N/A |

| | | | | | |
|---|---|---|---|---|---|
| | the middle region of the video frames, fly leftwards or rightwards to capture the front-sided face | track output, from tracker | | | |
| | | face side, from “face side detector” | | | |

# 2. Pedestrian detection

## 2.1. Introduction

Human detection in video is important in a wide range of applications that intersect with many aspects of our lives: surveillance systems and airport security, automatic driving and driver assistance systems in high-end cars, human-robot interaction and immersive, interactive entertainments, smart homes and assistance for senior citizens that live alone, and people-finding for military applications. The wide range of applications and underlying intellectual challenges of human detection have attracted many researchers.

In our project, we choose the $C^4$ method, a method detecting human Contour using a Cascade Classifier and the CENTRIST visual descriptor.

## 2.2. Related work

Recent progress in human detection has advanced the frontiers of this problem in many aspects, e.g., features, classifiers, testing speed, and occlusion handling. However, at least two important questions still remain open:

**Real time detection**. The speed issue is very important, because real-time detection is the prerequisite in most of the real-world applications and in a robot in particular.

**Identify the most important information source**. Features like HOG and LBP have been successful in practice. But they do not know clearly yet what is the critical information encoded in these features, or why they achieve high pedestrian detection performance in practice.

After some research in the reference paper, they get the conclusion. For pedestrian detection the most important thing is to encode the contour, and this is the information that HOG is mostly focusing on. One important difference between C$^4$ and existing methods is that they explicitly detect humans from the Sobel image. And signs of comparisons among neighboring pixels are key to encode the contour.

| Method | GPU | qVGA speed | VGA speed | speedup to HOG | Accuracy |
|---|---|---|---|---|---|
| HOG | No | | 0.075 fps | 1x | 74.4% (Sec. V-B) |
| ChnFtrs | No | | 0.5 fps | | 86% |
| HOG-LBP | Yes | | about 4 fps | | about 87% |
| HOG cascade | No | 5-30 fps | | 12-70x | |
| GPU HOG | Yes | 34 fps | 10 fps | 34x | similar to HOG |
| C$^4$ | No | 109 fps | 20 fps | 80x | 83.5% |

Comparing with other method such as HOG and LBP, we could see the C$^4$ method has the highest speed and the best accuracy. Therefore, we choose it.

CENTRIST is very appealing in terms of speed. They describe a method for feature evaluation that does not involve image pre-processing or feature vector normalization. In fact, they show that it is not even necessary to explicitly compute the CENTRIST feature vector, because it is seamlessly embedded into the classifier evaluation, achieving video-rate detection speed. They use a cascade classifier, and call the proposed method C$^4$, since they are detecting humans emphasizing the human Contour using a Cascade Classifier and the CENTRIST visual descriptor.

C$^4$ produces an accurate detector running in real-time (using only one single CPU core (or thread), not involving GPU or other special hardware)

## 2.3. The CENTRIST visual descriptor

CENTRIST means CENsus Transform hISTogram. They then propose to use the CENTRIST visual descriptor to recognize humans, because it succinctly encodes the crucial sign information, and does not require pre- or post-processing.

Census Transform (CT) is originally designed for establishing correspondence between local patches. Census transform compares the intensity value of a pixel with its eight neighboring pixels, as illustrated in Eqn. 3. If the center pixel is bigger than (or equal to) one of its neighbors, a bit 1 is set in the corresponding location. Otherwise, a bit 0 is set.

$$\begin{array}{c|c|c} 32 & 64 & 96 \\ \hline 32 & \mathbf{64} & 96 \\ \hline 32 & 32 & 96 \end{array} \quad \begin{array}{ccc} 1 & 1 & 0 \\ \Rightarrow 1 & & 0 \\ 1 & 1 & 0 \end{array} \Rightarrow (11010110)_2 \Rightarrow CT = 214 \qquad (1)$$

The eight bits generated from intensity comparisons can be put together in any order (they collect bits from left to right, and top to bottom), which is consequently converted to a base-10 number in [0 255]. This is the CT value for the center pixel. The CENTRIST descriptor is a histogram of these CT values.

As shown in Eqn. 1, CT values succinctly encode the signs of comparisons between neighboring pixels. The only thing that seems to be missing from CENTRIST, however, is the power to capture global (or larger scale) structures and contours beyond the small $3 \times 3$ range.

More importantly, if they are given an image *I* with CENTRIST ***h***, then among the small number of images *I'* that has a matching CENTRIST descriptor, they expect that *I'* will be similar to *I*, especially in terms of global structure or contour, which they illustrate in Fig. 1. Fig. 1a shows a 108×36 human contour. They divide this image into 12×4 blocks, thus each block has 81 pixels. For each block *I*, they want to find an image *I'* that has the same histogram and CENTRIST descriptor as *I*. As shown in Fig. 1b, the reconstructed image is similar to the original image. The global characteristic of the human contour is well preserved in spite of errors in the left part of the image.

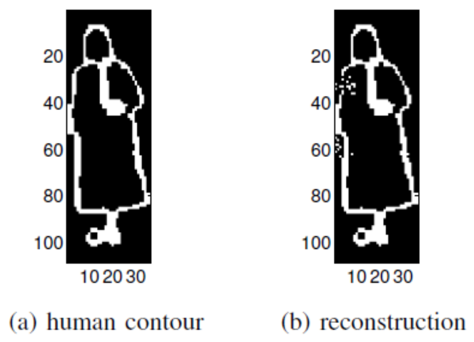


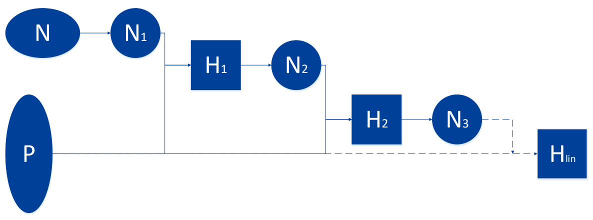
Fig. 1: Reconstruct human contour from CENTRIST.

The fact that CENTRIST not only encodes important information (signs of local comparisons) but also implicitly encodes the global human contour makes them believe that it is a suitable representation for detecting human contours.

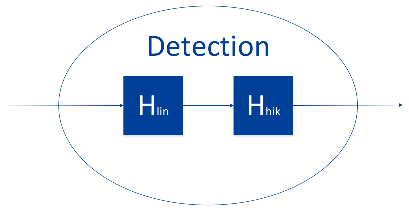
## 2.4. Detection framework

In the training phase, they have a set of 108×36 positive training image patches P and a set of larger negative images N that do not contain any pedestrian. They first randomly choose a small set of patches from the images in N to forma negative training set $N_1$. Using P and $N_1$ they train a linear SVM classifier $H_1$.

A bootstrap process is used to generate a new negative training set $N_2$. $H_1$ is applied to all patches in the images in N. In this bootstrapping process, they also rescale the negative image to examine more patches. They then train $H_2$ using P and $N_2$. This process is repeated until all patches in are classified as negative by at least one of $H_1$, $H_2$, They N then train a linear SVM classifier using P and the combined negative set $N_i$, which they call $H_{\text{lin}}$.



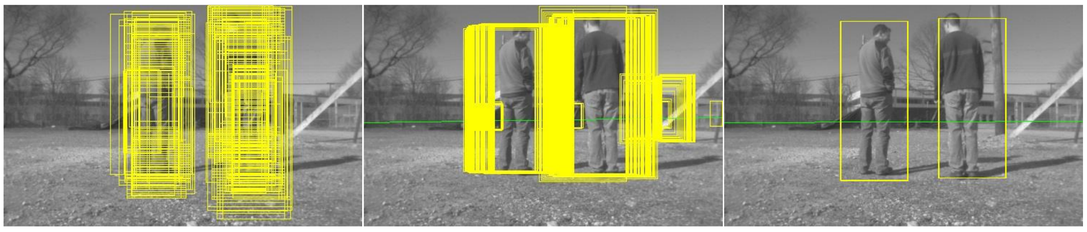
Linear classifiers ensure fast testing speed (and fast boot strapping process). However, it has been shown that HIK achieves higher classification accuracies on histogram features than linear SVM classifiers. They will train a second HIK SVM classifier to achieve higher detection accuracy. They use $H_{\text{lin}}$ on N to bootstrap a new negative training set $N_{\text{final}}$, and train an SVM classifier using the libHIK HIK SVM solver of, which they call $H_{\text{hik}}$. In the testing / detection phase, a cascade with two nodes $H_{\text{lin}}$ and $H_{\text{hik}}$ is used.



## 2.5. Pedestrian detection on-board a robot

They used the raw camera imagery to perform the detection and used the stereo range data to estimate the distance to the pedestrian. They used a particle filter to track the pedestrian between frames and to eliminate outliers. Finally, a following component was implemented to steer the robot chassis and command the neck pan axis.

They compared the basic approach described above with an optimized method that utilized the stereo data. They use the range image to provide hypotheses for where pedestrians may be standing. From the stereo data they use RANSAC to estimate a ground plane, and they sampled the depths along the ground plane's horizon. With the depth and coordinates of the plane they can calculate a box that would contain a pedestrian standing on the plane at the given position on the horizon and given distance. This gives them far fewer windows to test with the detector, which reduces both computation and false positives. Figures 2a, 2b, and 2c show the raw detections from the $C^4$ algorithm, the hypotheses generated from the stereo data, and the results of the $C^4$ classifier evaluated only on these hypotheses.



(a) $C^4$ (b) $C^4$+stereo (c) $C^4$+stereo, final

Fig. 2: On-board detection example. The three images are raw detection results using $C^4$, $C^4$+stereo, and the post-processed result of $C^4$+stereo, respectively. The green line is the ground plane estimated using stereo.

Note that the C4 detector was tailored to work with the robot to a 3-layer cascade instead of 2-

layer for faster speed (but less accurate). By default, the detection procedure is as follows. The detector looks for pedestrians of different projected sizes within the image. Recall that distance to the pedestrian determines the size of the pedestrian in the image: pedestrians that are farther away appear smaller in the image; while closer pedestrians appear bigger. The detection system searches the image at multiple scales, and for each scale runs a classifier on every single possible location of a pedestrian of that size within the image. Lacking any other information about the scene, this is the only reliable approach to detection. However, when other information is available, they should be able to use this to our advantage to decrease the number of windows on which to run the classifier, and to decrease the false positive rate. In particular, they can use information about the ground plane – which they can acquire from the stereo camera. For example, Fig 2 shows the result of the pedestrian detection classifier being run on all possible windows ($C^4$, Fig. 2a), and the many redundancies that are generated despite the fact that in most of the locations there cannot be a pedestrian. Pedestrians are bound to the ground and they can use this fact as a prior to limit the search range ($C^4$+stereo, Fig. 2b). The redundancies are a problem because each of the redundant windows has to be filtered out, increasing algorithm and computational complexity. The results after post-processing are shown in Fig. 2c.

And in our project, we will choose the biggest window as output in node B.

# 3. Human Tracking and Fusion

## 3.1 Tracking Performance summary

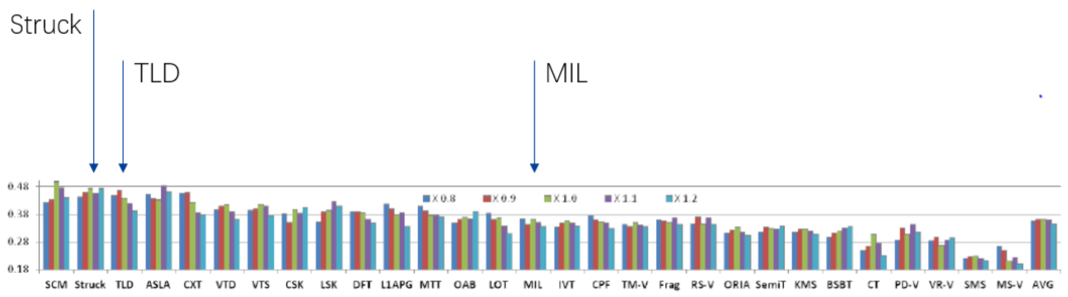
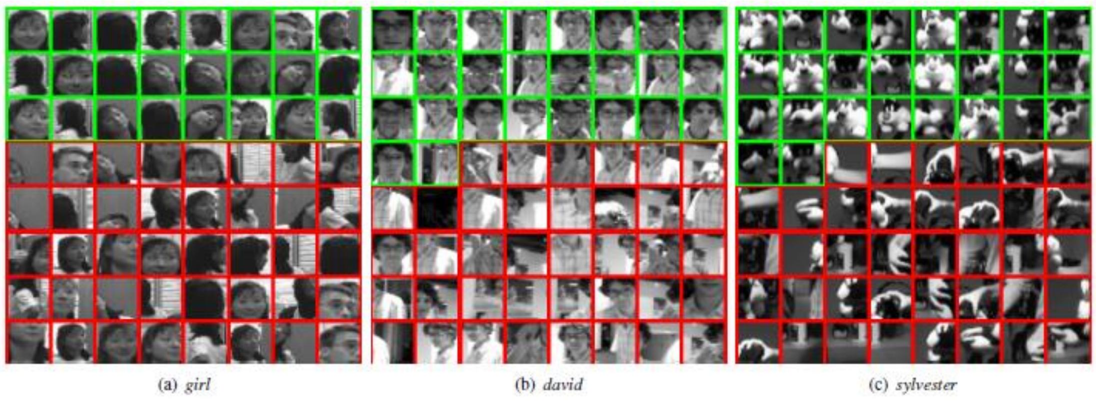


Figure 1. Performance summary for the trackers initialized with different size of bounding box. AVG (the last one) illustrates the average performance over all trackers for each scale.

Figure 1 illustrates the average performance of all trackers for each scale which shows the performance often decreases significantly when the scale factor is large as many background pixels are inevitably included in the initial representations. The performance of TLD, CXT, DFT and LOT decreases with the increase of initialization scale. This indicates these trackers are more sensitive to background clutters. Some trackers perform better when the scale factor is smaller, such as L1APG, MTT, LOT and CPF. One reason for this in the case of L1APG and MTT is that the templates have to be warped to fit the size of the usually smaller canonical template so that if the initial template is small, more appearance details will be kept in the model. On the other hand, some trackers perform well or even better when the initial bounding box is enlarged, such as Struck, OAB, SemiT, and BSBT. This indicates that the Haar-like features are somewhat robust to background clutters due to the summation operations when computing features. Overall, Struck is less sensitive to scale variation than other well-performing methods. In this case, we choose Struck in our project to track human.

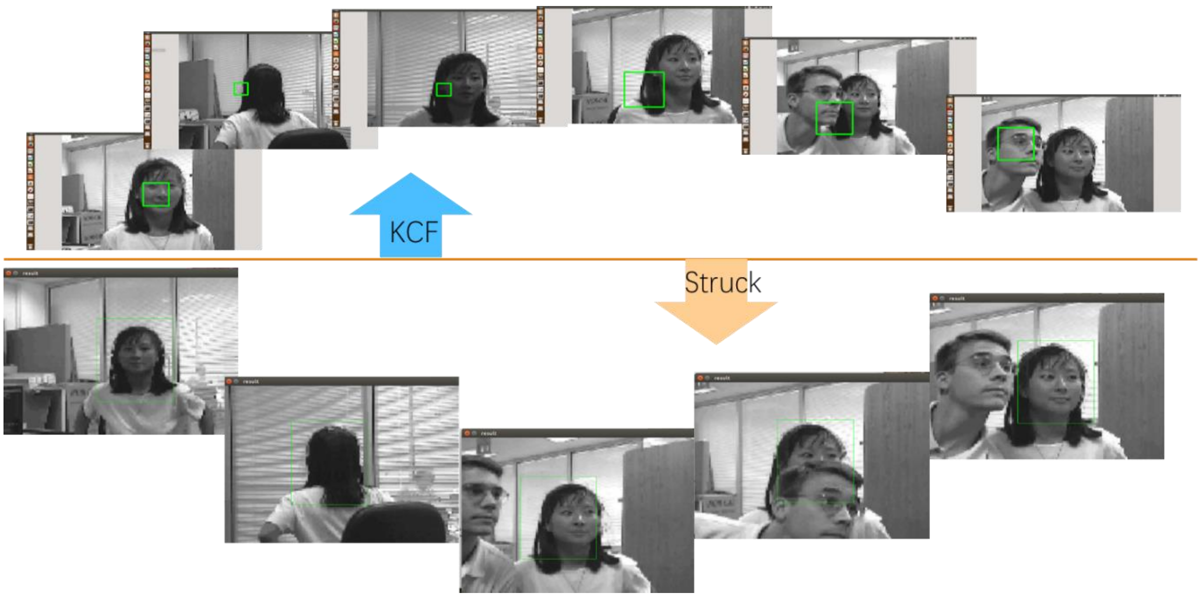
## 3.2 Human Tracking using Struck

Struck is introduced by Sam Hare from Department of Computing ,Oxford Brookes University, Amir Saffari ,Stuart Golodetz and so on. It relates to tracking-by-detection, structured output SVMs, budget maintenance, GPU-based tracking. Adaptive tracking-by-detection methods are widely used in computer vision for tracking arbitrary objects. Current approaches treat the tracking problem as a classification task and use online learning techniques to update the object model. Struck presents a framework for adaptive visual object tracking based on structured output prediction. By explicitly allowing the output space to express the needs of the tracker, we are able to avoid the need for an intermediate classification step. Struck uses a kernelized structured output support vector machine (SVM), which is learned online to provide adaptive tracking. To allow for real-time application, Struck introduce a budgeting mechanism which prevents the unbounded growth in the number of support vectors which would otherwise occur during tracking.


(a) *girl* (b) *david* (c) *sylvester*

In the fig, the green and red rectangles are treated as support vectors to train the classifier, and this training set is updated real-time. The green ones are good ones and red ones are bad. An issue with Struck is that the number of support vectors is not bounded and in general will increase over time. In this case a number of approaches have been proposed for online learning of classification SVMs on a fixed budget meaning the number of support vectors cannot exceed a specified limit. If the budget is already full and a new support vector needs to be added, these approaches identify a suitable support vector to remove and potentially adjust the coefficients of the remaining support vectors to compensate for the removal. We now propose an approach for incorporating

## 3.3 Comparison between Struck and KCF



Here is a comparison between tracking method of Struck and KCF, using the dataset of Girl. In the top, we focus the green rectangle in the center of the girl and use KCF to track the girl, it performs well even when the girl turns around, but when there is another person appears in the screen, the tracking rectangle will be stolen. In the same time, if we use Struck to tracking the girl, in the same pictures of sequence, the girl will always be tracked, no matter turning around or another person appears.

## 3.4 Compensation Filter for Fusion of Human Detection and Tracking

Even though the algorithms of human detection and tracking promise high performance, it is not always the case in practice. Our experiment will be carried out in outdoor environment with the variation of sunshine direction, with strong vibration, and with disruption by other passer-byes. Therefore, a method must be put forward to enhance the robustness of the detecting and tracking algorithm.

Before the design of such an algorithm, the characteristics of detector and tracker are studied through offline tests. In the test video, the main figure is the female student. The camera shoots her as she walks on the lawn. The camera may turn as she walks on, and may not always be in front of her. In addition, some other people may pass beside her and may even block the camera for short period of time.

The above described disruptions cause different kinds of failures of either the human detector or the tracker. On the one hand, when other people go into the scope of the camera, the detector may sometimes give the result of that passer-by, but not of the desired person, and the result often jumps between these human figures (as shown by the purple frame in Figure 1). On the other hand, the tracker may not be able to track the whole body of the desired person when the camera changes its shooting direction suddenly, later the tracker may drift and lost the object completely (as shown by the yellow frame in Figure 2).

Although these problems sometimes appear in practice, each one of the two vision algorithms features some preferred characteristics that compensates the drawbacks of the other. As the output of the detector may jump all of a sudden, the result of the tracker always stays smooth with respect to the last few frames and is thus precise for short period of time. However, the tracker sometimes suffers from drift, but the detector, on the contrary, has no problem of this kind and stays accurate for a long time.

A compensation filter combines the advantages of the two algorithm and prevents their drawbacks. The filter regards the tracker result as the base of estimation because of its smoothness. Then the output of the detector is used to correct the drift of the tracker. The difference of the two results is acquired by subtraction, and then the difference is multiplied by a weight factor, and this will be added to the estimation output of the tracker. The final result of each frame is given back to the tracker to execute the next tracking iteration.

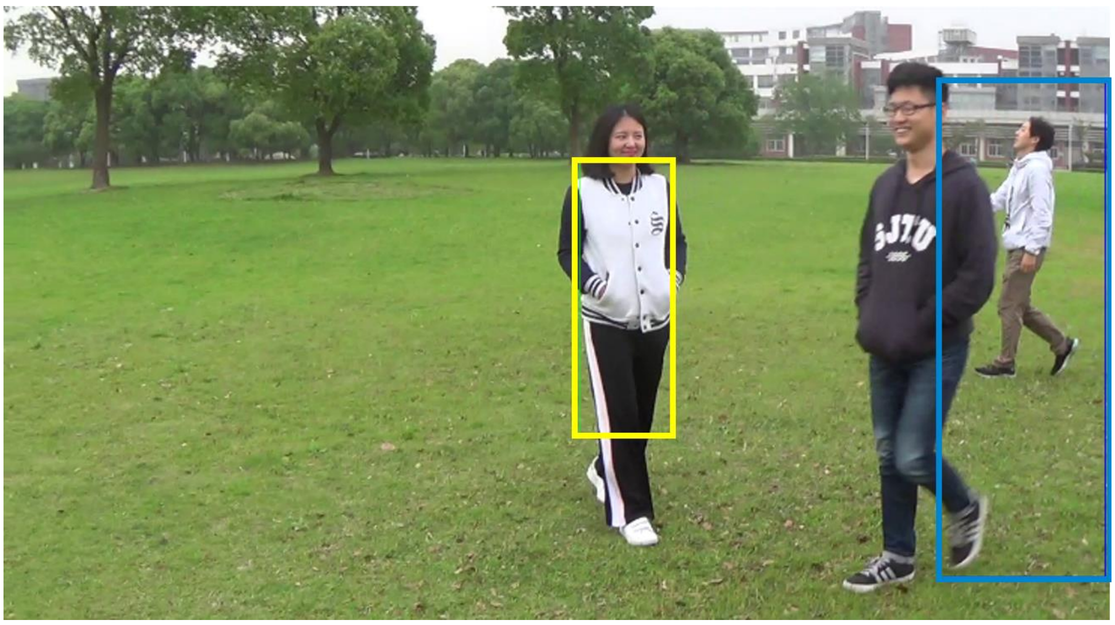


Figure 1. Human Detector Jumps

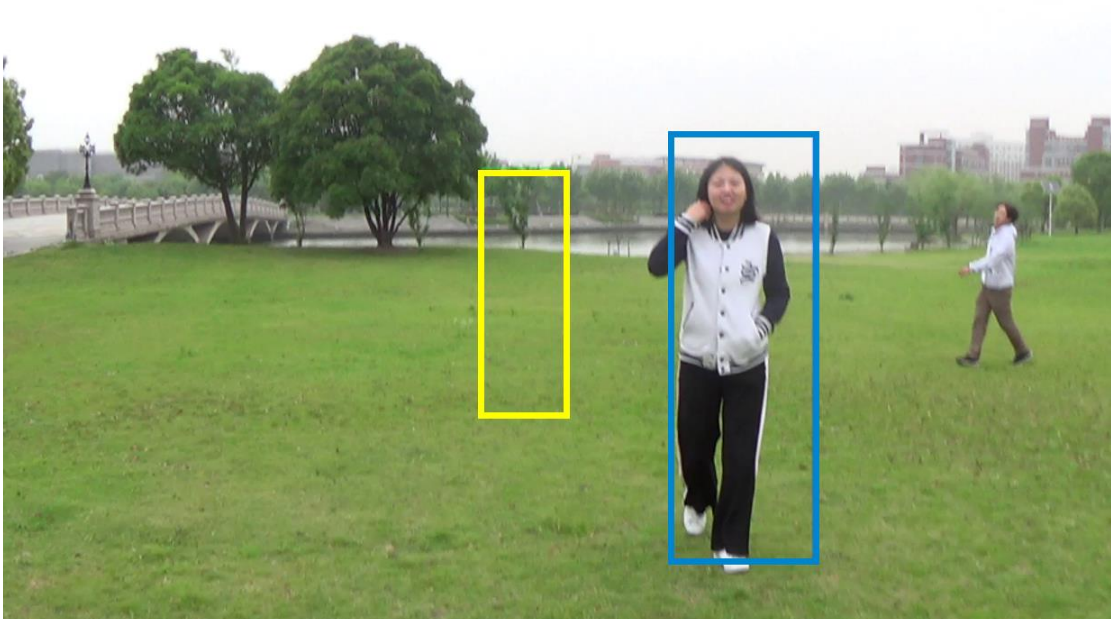
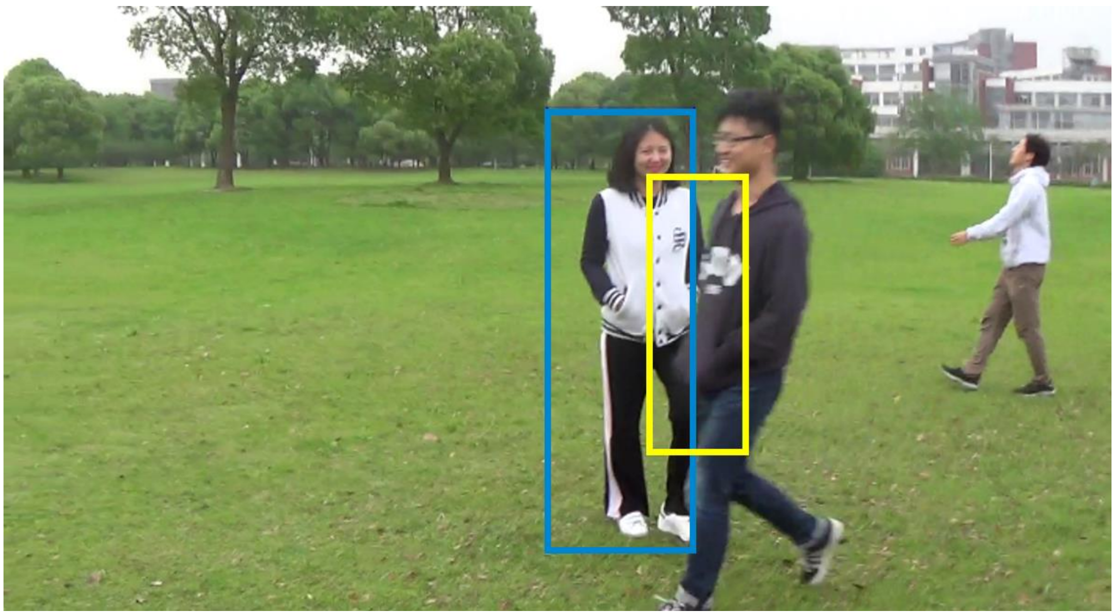


Figure 2. Tracker Drifts

The result of compensation filter with a suitable set of parameter is shown as follows. The output suffers no more from drift or jumping. Even when someone walks through the frame in front of the desired object (the female student), the result is hardly influenced. The robustness is realized as shown in the following two figures, in which the purple frame is the detector output and the yellow one the filtered result.



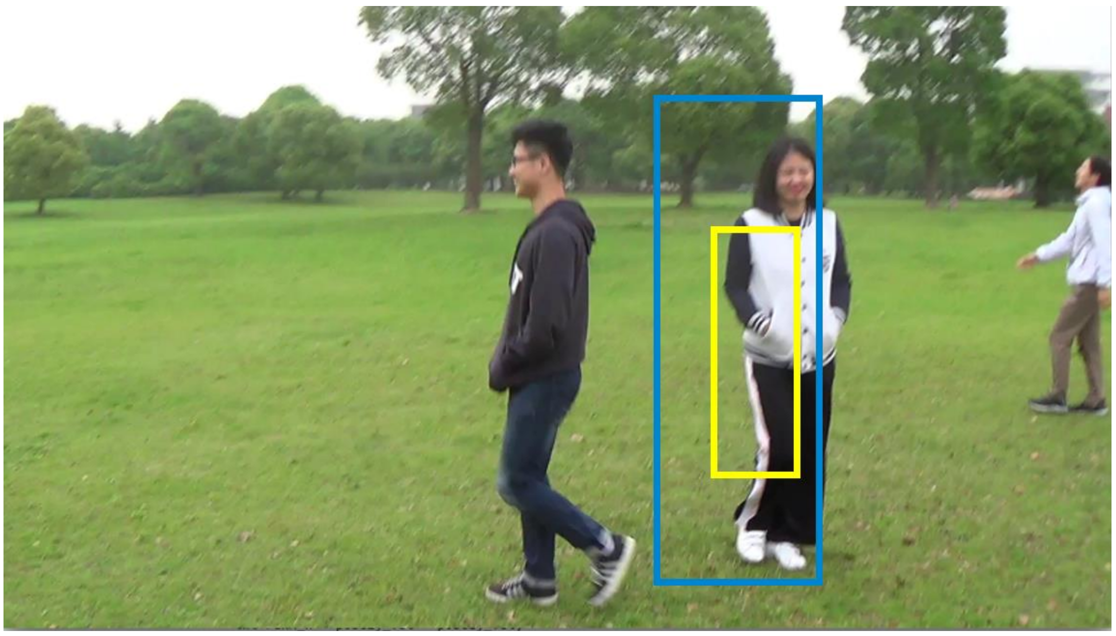


Figure 3 and 4. Fusion through Compensation Filter Prevents Disturbances

# 4. Face Detection and Angle Measurement

## 4.1 Introduction of face detection

Face detection is a computer technology being used in a variety of applications that identifies human faces in digital images. Face detection also refers to the psychological process by which humans locate and attend to faces in a visual scene. Face detection can be regarded as a specific case of object-class detection. In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class. Examples include upper torsos, pedestrians, and cars.

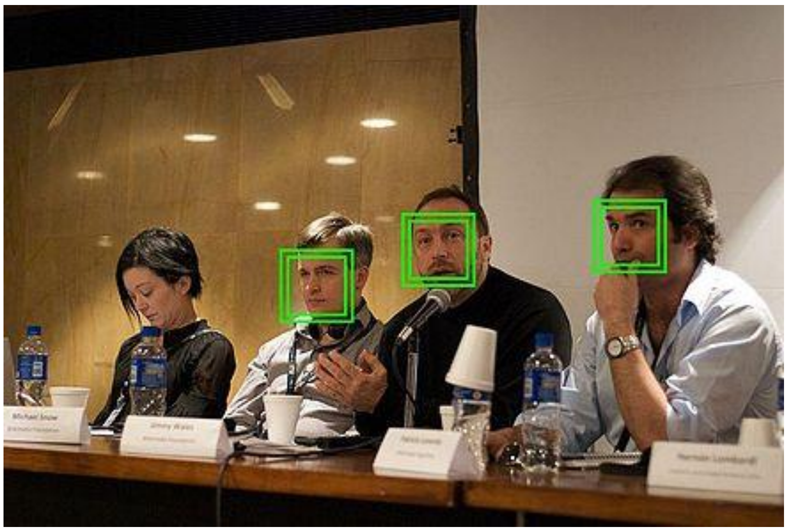


Fig 1 Automatic face detection with OpenCV

## 4.2 History of Face Detection

### 4.2.1 VJ Face Detection

Paul Viola and Michael Jones proposed The Viola–Jones object detection framework，which based on Cascade Classifier, and this is the first framework to The Viola–Jones object detection framework. They used Haar like feature and created an integral image to enhance the calculation of Haar feature. Adaboost (adaptive boost) were also utilized into this framework to enhance the feature extraction. Since then, researchers proposed more method to improve the accuracy or accelerate the speed of detection. Some focus on the modification of feature and others pay more attention on the cascade structure.

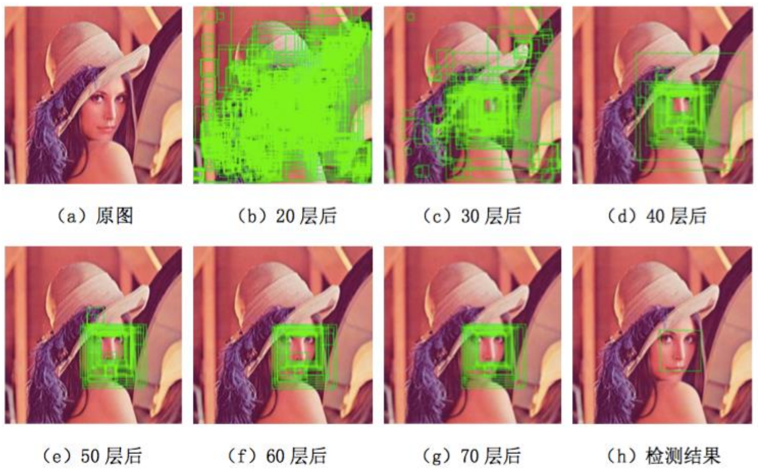


Fig 2 VJ face detection on Lena

The characteristics of Viola–Jones algorithm which make it a good detection algorithm are:

1. Robust – very high detection rate (true-positive rate) & very low false-positive rate always.
2. Real time – For practical applications at least 2 frames per second must be processed.
3. Face detection only (not recognition) - The goal is to distinguish faces from non-faces (detection is the first step in the recognition process).

However, there are some disadvantages of this framework :

1. Detector is most effective only on frontal images of faces
2. It can hardly cope with 45 °face rotation both around the vertical and horizontal axis.
3. Sensitive to lighting conditions
4. We might get multiple detections of the same face, due to overlapping sub-windows.

### 4.2.2 DPM: Deformable Parts Model

DPM takes histogram of oriented gradients as a descriptor and support vector machine as the classifier. This method has advantages on solving the problem of blocking, expression and changes of face angle.

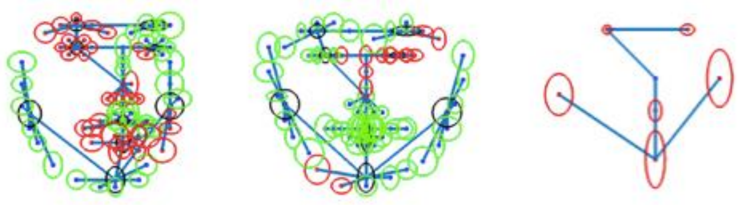


Fig 3 DPM face detection

### 4.2.3 CNN And DL

With the development of performance of computer, and the boom of neural network, Convolutional Neural Network gradually become the mainstream of classifier, especially in the field of the image detection.

## 4.3 Face Detection in OpenCV

The VJ face detection framework were implemented in the OpenCV as CascadeClassfirer. This module based on Haar-like feature or LBP (local binary pattern) features, and Haar-like feature as default. Method *CascadeClassifier::Load* can be called to load a cascade classifier model, which was generated after training. The method *CascadeClassifier::detectMultiScale* is the core part of face detection in this module, to Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles.

## 4.4 Haar-like Feature

Haar-like features are digital image features used in object recognition. They owe their name to their intuitive similarity with Haar wavelets and were used in the first real-time face detector.

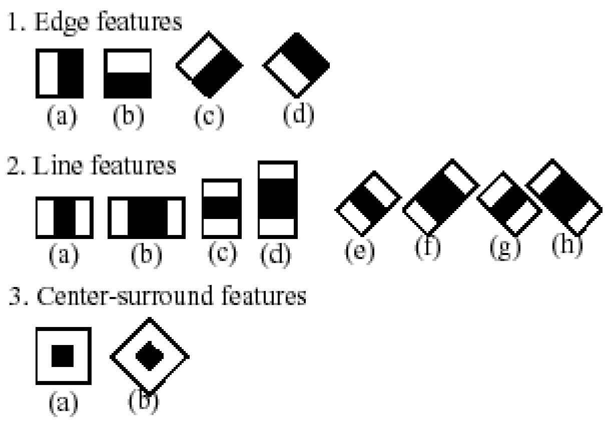
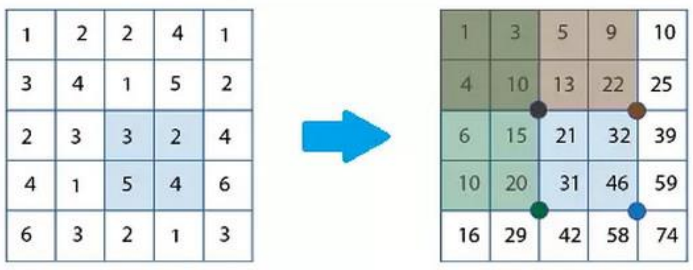


Fig 4 Haar-like Features

In the detection phase of the Viola–Jones object detection framework, a window of the target size is moved over the input image, and for each subsection of the image the Haar-like feature is calculated. This difference is then compared to a learned threshold that separates non-objects from objects. Because such a Haar-like feature is only a weak learner or classifier (its detection quality is slightly better than random guessing) many Haar-like features are necessary to describe an object with sufficient accuracy. In the Viola–Jones object detection framework, the Haar-like features are therefore organized in something called a classifier cascade to form a strong learner or classifier.



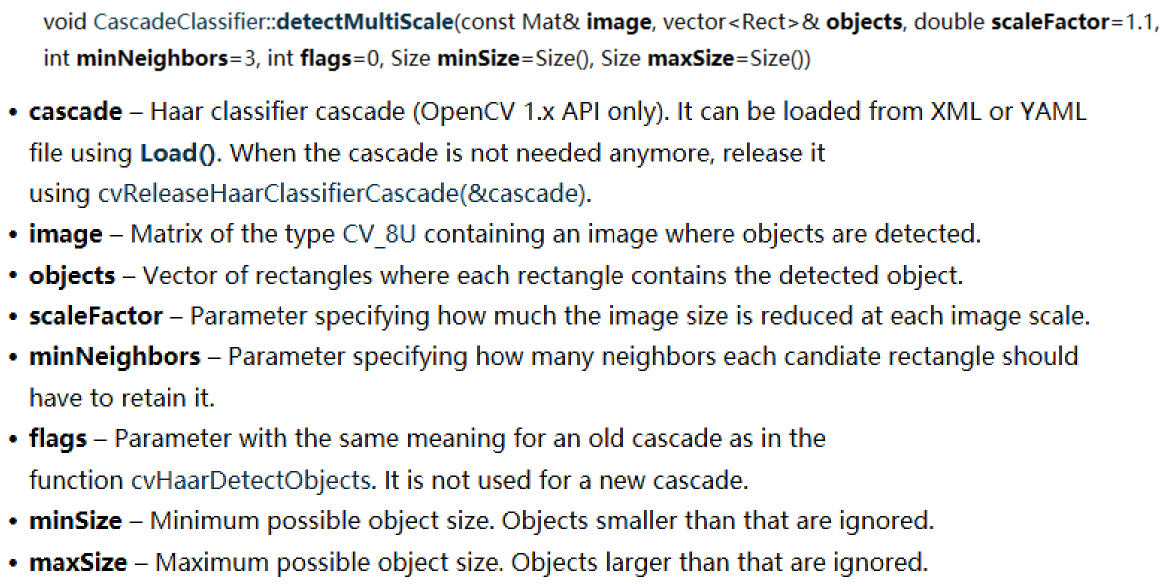
| | | | | |
|---|---|---|---|---|
| 1 | 2 | 2 | 4 | 1 |
| 3 | 4 | 1 | 5 | 2 |
| 2 | 3 | 3 | 2 | 4 |
| 4 | 1 | 5 | 4 | 6 |
| 6 | 3 | 2 | 1 | 3 |

| | | | | |
|---|---|---|---|---|
| 1 | 3 | 5 | 9 | 10 |
| 4 | 10 | 13 | 22 | 25 |
| 6 | 15 | 21 | 32 | 39 |
| 10 | 20 | 31 | 46 | 59 |
| 16 | 29 | 42 | 58 | 74 |

Fig 5 Haar calculation with an Integral Image

### 4.4.1 CascadeClassifier::detectMultiScale

In the new C++ interface it is also possible to use LBP (local binary pattern) features in addition to Haar-like features.



void CascadeClassifier::**detectMultiScale**(const Mat& **image**, vector<Rect>& **objects**, double **scaleFactor**=1.1, int **minNeighbors**=3, int **flags**=0, Size **minSize**=Size(), Size **maxSize**=Size())

- **cascade** – Haar classifier cascade (OpenCV 1.x API only). It can be loaded from XML or YAML file using **Load()**. When the cascade is not needed anymore, release it using cvReleaseHaarClassifierCascade(&cascade).
- **image** – Matrix of the type CV_8U containing an image where objects are detected.
- **objects** – Vector of rectangles where each rectangle contains the detected object.
- **scaleFactor** – Parameter specifying how much the image size is reduced at each image scale.
- **minNeighbors** – Parameter specifying how many neighbors each candiate rectangle should have to retain it.
- **flags** – Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade.
- **minSize** – Minimum possible object size. Objects smaller than that are ignored.
- **maxSize** – Maximum possible object size. Objects larger than that are ignored.

## 4.5 Face Detection in SeetaFace

SeetaFace Engine is an open source C++ face recognition engine, which can run on CPU with no third-party dependence. It contains three key parts, i.e., SeetaFace Detection, SeetaFace Alignment and SeetaFace Identification, which are necessary and sufficient for building a real-world face recognition application system.

This face recognition engine is developed by Visual Information Processing and Learning (VIPL) group, Institute of Computing Technology, Chinese Academy of Sciences. The codes are written in C++ without dependence on any 3rd-party libraries.

### 4.5.1 SeetaFace Detection Method

SeetaFace Detection is an implementation of Funnel-Structured cascade, which is designed for real-time multi-view face detection. FuSt aims at a good trade-off between accuracy and speed by using a coarse-to-fine structure. It consists of multiple view-specific fast LAB cascade classifiers at early stages, followed by coarse Multilayer Perceptron (MLP) cascades at later stages. The final stage is one unified fine MLP cascade, processing all proposed windows in a centralized style.

The open source version of FuSt includes codes for face detection as well as a model for detecting near-frontal faces (also with reasonable capability to detect non-frontal faces), which is trained with approximately 200K face images. Please note that the implementation is slightly different from that described in the corresponding paper: (1) The fine MLP cascade uses SURF feature instead of SIFT; (2) Non-Maximal Suppression (NMS) is added; (3) Landmark prediction is replaced by bounding box regression (For facial landmark localization, see SeetaFace Alignment).

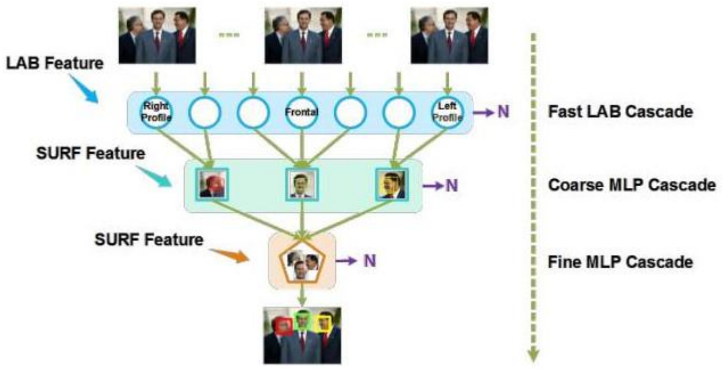


Fig 6 Structure of FuSt

## 4.6 Performance Evaluation

The discrete ROC curve on FDDB of the SeetaFace Detector is shown below (curves of other methods are obtained from FDDB official site). To achieve the results in this figure with SeetaFace Detector, the minimum size of faces to detect should be set to 20, step of sliding window is set to 2 or 4 (as shown in the legend), and the scaling factor is set to 0.8 (or 1.25 if defined as that used in OpenCV).

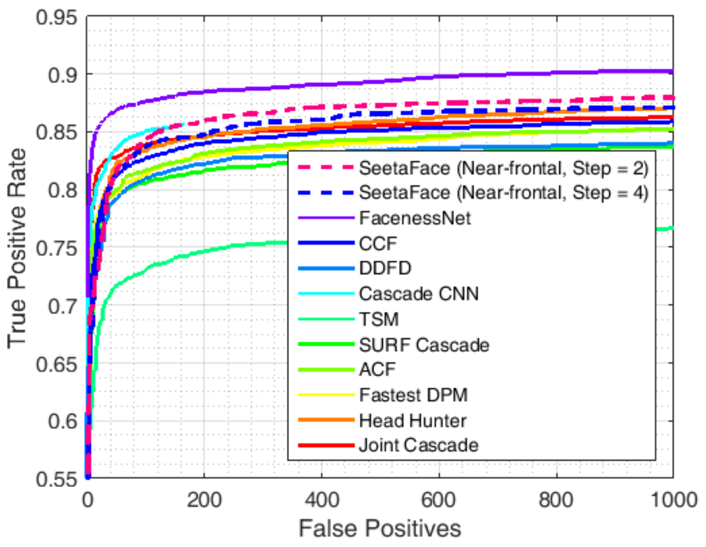


Fig 7 performance of SeetaFace Detection

## 4.7 Face Detection in node_c

Per the design of whole structure of this project, node_c is for face detection. Node_c subscribes to the client to get the raw frame of the camera, and subscribe to the node_b to get the result of human detection, and then combine these two message to get new image for the face detection. The main function for the detection is *detectAndDraw ()* and the main parts of this function is as follows:

```
seeta::FaceDetection
detector("/home/rover/catkin_ws/src/fdetection/model/seeta_fd_frontal_v1.0.bin");
detector.SetMinFaceSize(10);
detector.SetScoreThresh(2.f);
detector.SetImagePyramidScaleFactor(0.8f);
detector.SetWindowStep(4, 4);
  …
std::vector<seeta::FaceInfo> faces = detector.Detect(img_data);
```

Fig 8 some code in *detectAndDraw ()*

This node publishes the result as a ROI Rect for the part of face orientation detection. The result as shown on the below.

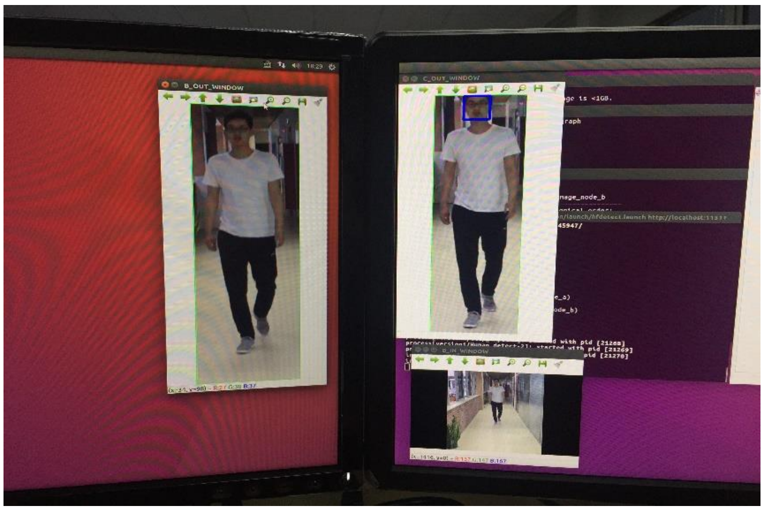


Fig 9 the result of face detection

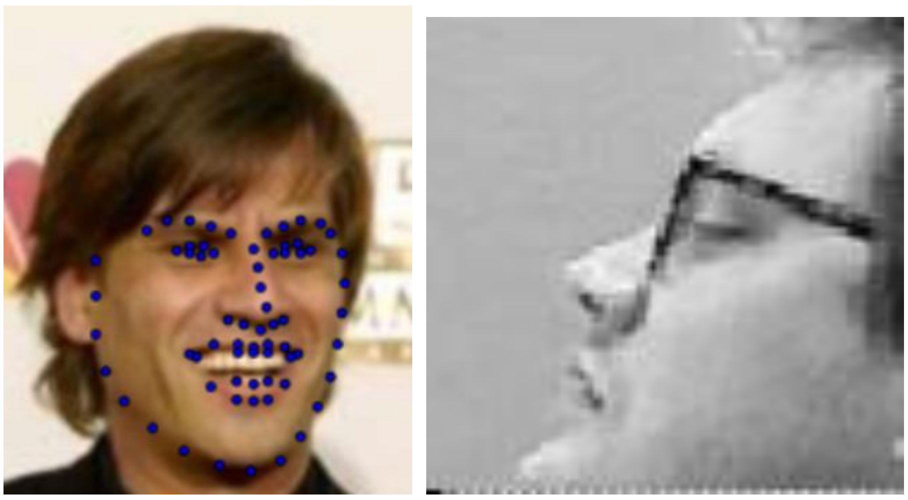
## 4.8 Face angle measure

In order to keep the UAV shooting towards the front of users, face angle requires to be measured.

When the user makes a turn, the UAV needs linear motion to adjust position and angular motion to maintain the right direction to the front of the user. Measure of the front direction of user may be based on body or face. Measure by body is nearly impossible because the features of many clothes are very similar towards all directions and the styles of clothes change from people to people and time to time. Therefore, we measure the angle of the face in this work. Face angle measure is a difficult work due to the complicated features on face and various skin color and wild light environment. Besides, the application scenario of face angle measure is very few. Hence, few works on face angle measure are found. Here we just introduce our own method.

In a real application scenario to keep shooting in front of someone, the angle of the user's face is changing all the time. It is of great difficulty and of none necessity to measure a relatively high accuracy angle. Thus we only consider three categories of face angles, namely left side, front face and right face. Each categories have a relatively wide tolerance.

An easy method is to find the landmarks of a face and then classify faces is to extract the landmarks of the face and then use some traditional classifier like K-Nearest Neighbors(KNN) and support vector machine(SVM). Like Figure 5.1(a) shows, blue points represent the landmarks of the face.



(a) Landmarks (b)Profile face with large angle

Figure 5.1 Example faces

However, in the real environment, the angle of the face may be very large. As Figure 5.1(b) shows, landmarks can not be extracted through current techniques. Therefore, we decide to utilize convolutional neural networks(CNN) to measure the angle of the faces directly from the pixels of faces.

An important part of machine learning methods like CNN is the dataset. Here a face dataset with angle labels is required. The most famous dataset with such angle labels is Multi-pie dataset made by CMU. However, it costs 350$ and then we decide to use a much smaller but free dataset, QMUL multi-view face dataset, as is shown in Figure 5.2.

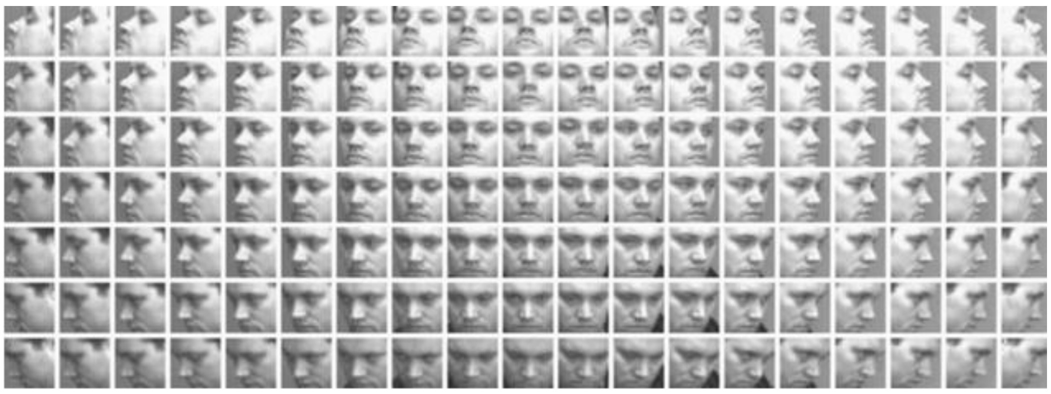


Figure 5.2 QMUL Multiview face dataset. 37 people are in greyscale of 100x100 image size. Each person has 133 facial images covering a viewsphere of +/-90 degrees in yaw and +/-30 degrees in tilt at 10 degrees increment

The CNN is developed based on keras environment with python code. Keras is a very compact deep learning environment built on the basis of theano or tensorflow. The convolutional neural network we use has convolutional layers, max pooling layers and fully connected layers. Figure 5.3 shows a basic architecture of our CNN based face angle measure module. After trial and error, the architecture we designed is shown as follows:

**Network architecture:**

- Input: 100 x 100, output dimension: 100x100
- Convolutional Layer: 8 kernels (5x5), output dimension: 96x96x8
- Pooling Layer: max pooling (3x3), output dimension: 32x32x8
- Convolutional Layer: 16 kernels (5x5), output dimension: 28x28x16
- Pooling Layer: max pooling (2x2), output dimension:14x14x16
- Convolutional Layer: 32 kernels (5x5), output dimension: 10x10x32
- Pooling Layer: max pooling (2x2), output dimension: 5x5x32
- Convolutional Layer: 128 kernels (5x5), output dimension: 4x4x128
- Fully Connected Layer, output dimension: 1x1

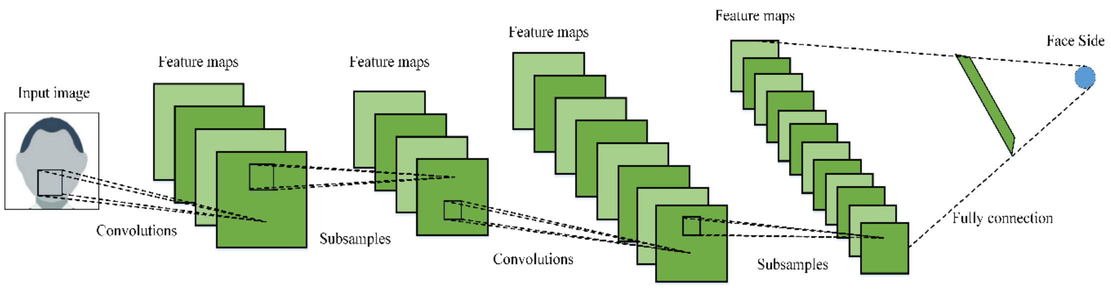


Figure 5.3 A basic architecture of our CNN based face angle measure module

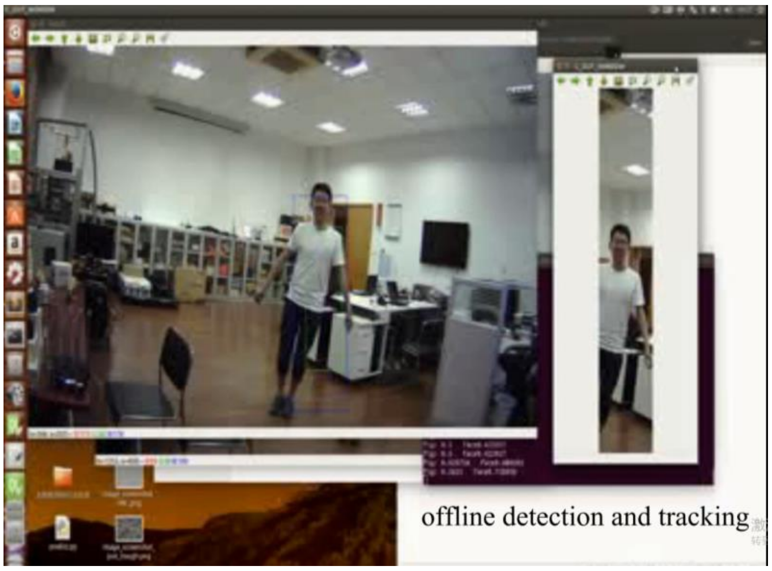
1539 images of 27 people are used as training data and 570 images of 10 people are treated as testing data. Change the size of mini-batch. When mini-batch size is 100, we get an accuracy of nearly 93%.

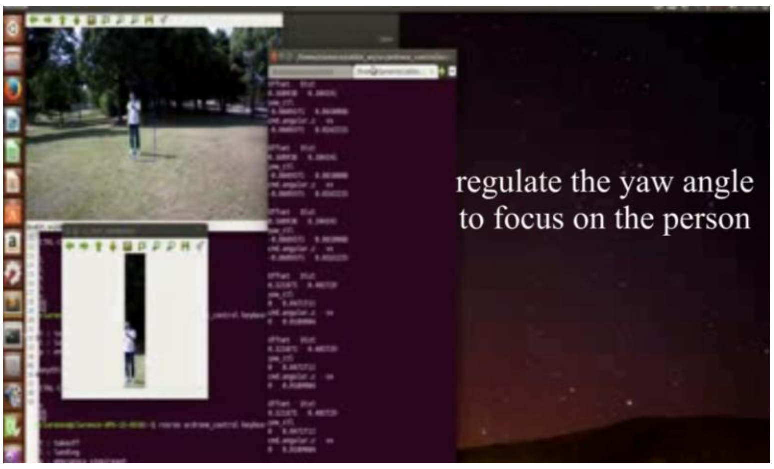
# 5. Result:

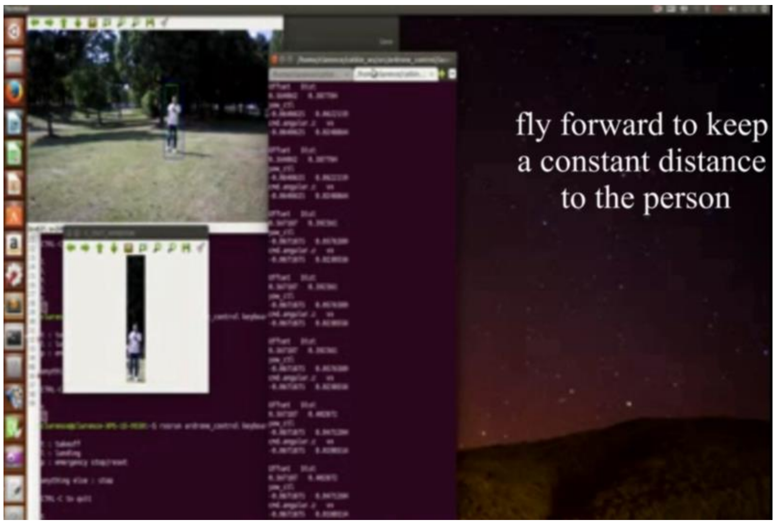
The results of offline human detection and online flying are shown in the video. The following 3 figures are the screen shoots of that video with some explanatory text.

The offline detection and tracking was taken indoors. The shelves, desks and electrical apparatus increase the difficulty of accomplishing the functions of the system. However, as indicated by the frames, as the tester walks and jumps and turns, the system performs just well enough.

The online experiment includes all kinds of situations that the system might confront in practice. As given in the video one after another, the vehicle regulates the yaw angle to focus on the person, flies forward to keep a constant distance to the person, controls yaw and distance at the same time, flies sideward to keep shooting the front face, and can even recover the angle and distance control and sideward control after losing the object for a short time. The performance proved that all the algorithms possess enough robustness.







## 6. Reference:

Wu J, Geyer C, Rehg J M. Real-time human detection using contour cues[C]// IEEE International Conference on Robotics and Automation. IEEE, 2011:860-867.

Sam Hare, Amir Saffari, And Philip H. S. TorrInternational Conference On Computer Vision (ICCV), 2011

Y Wu, J Lim, MH Yang "Online Object Tracking: A Benchmark", Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on

S. Avidan. Support Vector Tracking. IEEE Trans. on PAMI, 26:1064–1072, 2004.

B. Babenko, M. H. Yang, and S. Belongie. Visual Tracking with Online Multiple Instance Learning. In Proc. CVPR, 2009.

M. B. Blaschkoand C. H. Lampert. Learning to Localize Objects with Structured Output Regression. In Proc. ECCV, 2008.

A. Bordes, L. Bottou, P. Gallinari, and J. Weston. Solving multiclass support vector machines with LaRank. In Proc. ICML, 2007.

K. Crammer, J. Kandola, R. Holloway, and Y. Singer. Online Classification on a Budget. In NIPS, 2003.

Z. Wang, K. Crammer, and S. Vucetic. Multi-Class Pegasoson a Budget. In Proc. ICML, 2010. 2

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large Margin Methods for Structured and Interdependent Output Variables. JMLR, 6:1453–1484, Dec. 2005.

P. Viola and M. J. Jones. Robust real-time face detection. IJCV, 57:137–154, 2004.