Emotion Recogniser

Tianqi (Carl) Liu, *z5019791*
Hanzhang (Roger) Zeng, *z3493047*

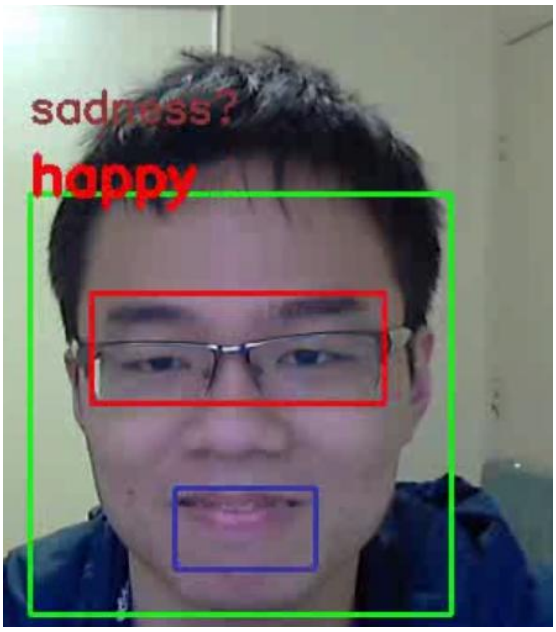
***Abstract*—This is a report on the emotion recognizer project for comp9517 at UNSW. the report includes our goal, reference survey, problem decomposition, detailed specification, design, test and justification, group contribution and conclusion.**

I. Introduction

The goal of the project is to able to recognize human emotions on real time live stream to a degree of accuracy, more details provided in Scope of project section. There are 4 emotions that we are concerned with:

1) *neutral*
2) *happy*
3) *angry*
4) *sadness*

other: when the recognizer is not sure we consider the emotion to be some other emotion



For the final product, as shown in the photo, The face will be framed telling us which components are used for emotion recognition. There are 2 emotions displayed, the larger text on top of the face frame (Green) is the best estimation of the current emotion, the darker red text is the 2nd best predication.

II. Problem Decomposition

The project consists of two major components:

1) ***Face Extraction***
2) ***Emotion Recognition***

A. Face Extraction

In order to perform the emotion recognition algorithm, we need to extract the face from the image. We decided to use the Fisherface recognizer which requires the input image to be of fixed size (detailed explanation will be in survey section) therefore it is essential that we first extract the face.

We chose the Haar Cascade algorithm for facial extraction because it is a really fast algorithm while maintaining a relatively high accuracy. Speed is very important for our application because during live stream, we want the algorithm to spend as little time as possible on the computation or else the stream will lag and the result will be unfavorable. Accuracy is not as important as speed in this context. When it comes to live stream, at every $\frac{1}{60}$ second, a new frame will be captured and computed, it is barely noticeable that one or two frames were not captured during a single second.

B. Emotion Recognition

The emotion recognition phase can be furthur classified into 2 parts.

1) ***Training***
2) ***Prediction***

In order to perform real time recognition we need fast algorithms, the best way to achieve this is to use a trained recognizer. The most common recognizers for the face are Haar Cascade Recognizer, Fisherface and Eigenface (details of each recognizer will be in survey sections)

C. Training

For the training process, we require large amount of data. We found a frontal face database from Cohn Kanade which contains 6000 images on 7 emotions.

1) *neutral*
2) *contempt*

3) *disgust*
4) *fear*
5) *happy*
6) *sadness*
7) *surprise*

We wrote a python code which classifies the emotions to folders named as the emotion, so in the happy folder contains all images that are happy

We train the recognizer by extracting the images from each of the folder, detailed explanation will be in algorithm design section.

D. Prediction

For the prediction phase, we extract the face using Haar Cascade then pass it into the recognizer, the recognizer will output a specific value that corresponds to one of the emotions. The emotion will be displayed on the screen above the head frame.

III. Scope of the project, detailed Specification

The project uses python2.7 + opencv2.4

A. list of goals

There is a list of requirements we need to achieve for our project. Our final goal is to produce an acceptable accurate (70%) multi person emotion recogniser performing on live stream, at the start of the project we set the benchmark to around 70% because for most current object recognition algorithm an accuracy of 70% can be achieved (we will discuss this later in the test and design parts). There will be 2 emotions displayed one is the best estimation while the other is the 2nd best estimation. In order to achieve this we need have the following set of functional parts:

1) A well performing Face extractor that is fast and relatively accurate
   a) The extractor should be able to extract multiple faces
2) A fast emotion recognition system that performs at high speed

A major consideration for this application is speed because if the program lags too much it won't be able to even function at run-time.

B. Users and User interaction

We aim to provide equal performance for users of all race and color, the user should interact with the program by sitting in front of the camera where the camera is able to capture the **frontal** face of the user. The program should output a solution in a time that is considered appropriate waiting time for the user and does not cause severe lag, we set the benchmark to 1 second.

C. Limitations

There are a couple of limitations to the project

1) ***Accuracy***
2) ***number of emotions***
3) ***frontal face only***

1) Accuracy: The emotion recognizer has a limitation on the accuracy because all human emotions look different, there is no single classifier that can accurately predict every emotion, the benchmark we set was 70% although, during actual testing the benchmark was never achieved (details in testing).

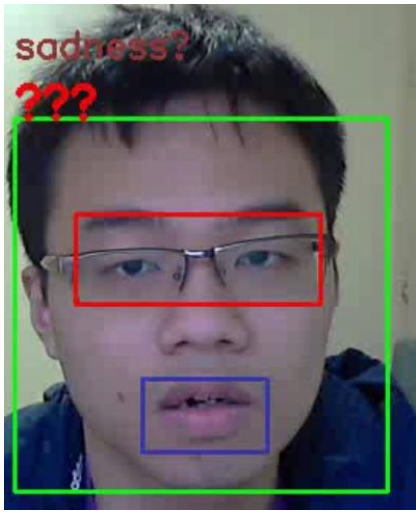


Fig. 1. sad face recognized as other

2) number of emotions: python has a limitation on memory space usage and because it is not a really optimized language we were unable to open large sets of image data at one time, due to this reasons we were only able to train around 4 sets of data. The emotion we chose are happy, angry, neutral and sadness other emotions will not be detected therefore will be classified as other.

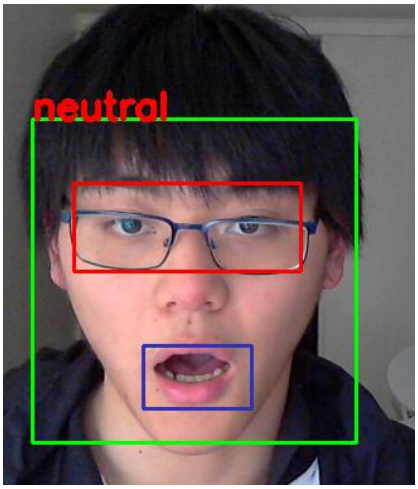


Fig. 2. Surprised face not classified

3) frontal face only: The training data set we acquired has frontal face only, all available data we have found are only dedicated to the frontal face, for this reason we only implement the frontal face detector and recogniser. Any face other than the frontal face will not be detected thus not classified.

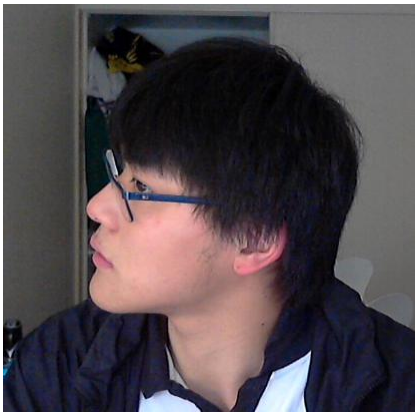


Fig. 3. Side face not considered

IV. LITERATURE SURVEY

A. Haar Cascade Face Extraction

When we are collecting expressions from different images, we want to discard the background and other objects in the photos in order to increase the correctness of training and prediction. We use the Haar-cascade classifier. According to the Viola and Jones's object detection framework, the classifier determines the feature points by calculating the Haar-like feature by looking up to multiple rectangles (feature types) and applying the mask on the region to summarise the characteristics of the specific image.

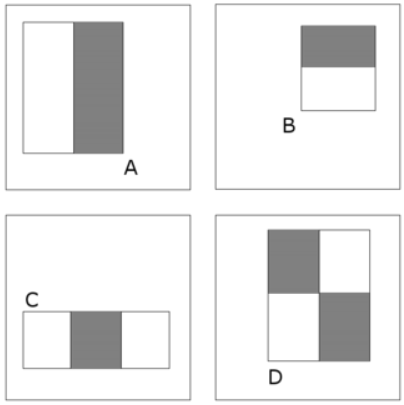


Fig. 4. The Viola detection framework evaluates the characteristics in an image based on four kind of Haar feature

The training of the the Haar Cascade recognizer is done by inputting a large quantity of positive and negative images, as said, the feature detectors will detect points in positive and negative images and based on the results modifiy the threshold for each feature detector, combining all these weak detectors where each has a weighted average it produces an object recognizer. In our case the frontal face recognizer.

In the OpenCV implementation of Haar classifier, an improvement is made by Rainer Lienhart. Rainer took the advantages of a 45° rotation on all the Haar features that Viola had and categorised them into four groups: Edge, Line, Centre-surround and the special diagonal line feature (Viola feature D).

In this project, we use frontal face cascade classifier provided by Intel Corporation. The Haar features of frontal face can be visited in OpenCV repository on GitHub.

B. Eigenface Recogniser

1) Recognition: During the training process the eigenvector of all image are processed and eigenvalues are produced, the eigenvector get's sorted by the

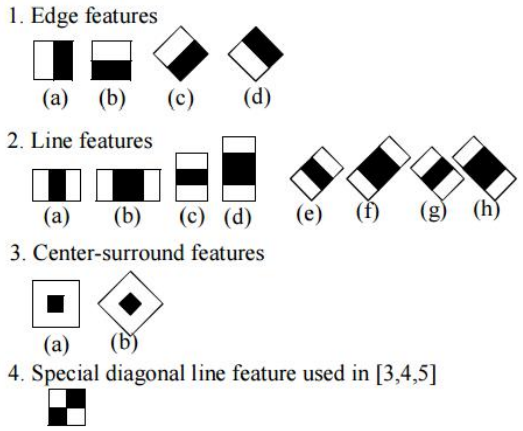


Fig. 5. The Rainer classifier takes advantages of rotation and enhanced line feature detections

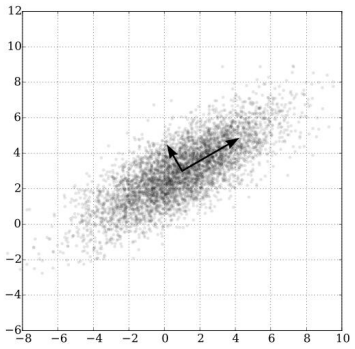


Fig. 6. PCA of a multivariate Gaussian distribution centered at (1,3) with a standard deviation of 3 in roughly the (0.866, 0.5) direction and of 1 in the orthogonal direction. The vectors shown are the eigenvectors of the covariance matrix scaled by the square root of the corresponding eigenvalue, and shifted so their tails are at the mean.

eigenvalue and we take the first k eigenvectors. A pca basis is constructed based on these k component. We map all the traning sample into this space, then we map the input image to the pca space and return the nearest neighbor between the traning sample and the input image.

2) Training: The idea of Eigenface recognition is that a high dimensional dataset is described by correlated variables therefore only a few dimension possess usefulel information, we can only make a decision when there are variance in the data set and inorder to get this variance we perform the PCA which converts possibly correlated data into unrelated data.

in order for this to be accurate we need the eigenvector to be of a large dimension because for small dimension too much information is lost, in order to increase accuracy we need large input of data.

C. Fisherface Recogniser

1) Recognition: The Principal Component Analysis (PCA), which is the core of the Eigenfaces method, finds a linear combination of features that maximizes the total variance in data. The Linear Discriminant Analysis performs a class-specific dimensionality reduction maximizes the ratio of between-classes to within-classes scatter, instead of maximizing the overall scatter.

2) Training: To use the Fisherface recogniser, we prepare two sets of data, one is the training data and the other one is the test data. We feed the classifier with the training data set. The Fisherface algorithm uses LDA(Linear Discriminant Analysis) which basically downgrades the image by mapping the points to a smaller orthogonal matrix which will serve as bases for the class, such that the between class distance is maximised and the within class scatter is minimised.

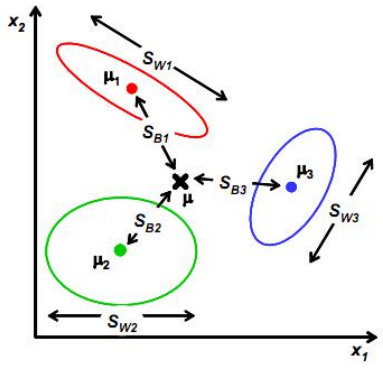


Fig. 7. LDA mapping

V. Design and Implementation

A. Design Overview

There are mainly two components in our project, face extractor and emotion recogniser. The face extractor helps us sampling our own face and extracting faces from Cohn-Kanade database. It builds the training set for emotion recogniser. Our emotion recogniser predicts emotion images into four different groups as they are mentioned before.

B. Cohn-Kanade Database Processing

Cohn-Kanade contains different kinds of information for each emotion. The first group of data is the original images which describe the change in emotion of each candidate in multiple animated frames. The second

group of data is the emotion encoding[1] of images set for each candidate. The database also contains FACS encoding and face geometry. We used frontal face Haar-cascade instead of facial geometry in order to keep the consistency of training and prediction while we never used FACS encoding for trading the accuracy on video streamed emotion detection for program performance and simplicity.

C. Naive Face Extractor

In our first attempt, we implement our facial extractor to walk through the database and simply crop out faces using frontal face Haar-cascade classifier. We dispatched the cropped out faces with their emotion encoding and used them as training set.

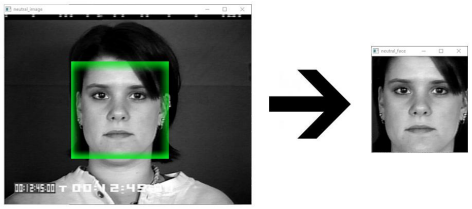


Fig. 8. Naive face extractor using frontal face Haar-cascade classifier to crop out a face

D. Naive Emotion Recogniser

We started with Fisherface recogniser which is used for general classification on overall face detection due to its regression-liked training module. Unfortunately, such recogniser has so many drawbacks and makes our program barely usable.

1) Inaccuracy: The result is unsatisfying and it can barely detect the happy emotion. The Fisherface recogniser takes care about too much information on the faces. Even a small portion of the background set behind the face will affect its behaviour. Also, the live camera capture introduces vary inconsistencies[2] during emotion prediction.

2) Computational Heavy: The computation time of Fisherface recogniser is heavier compared to Eigenface because it does regression analysis for the face classification.

3) Illumination: The illumination on the face will affect recognition process. It introduces a large number of unwanted feature points on our faces which leads to incorrect prediction.

4) Bad User Experience: Our program is not user friendly due to the unstable recognition on each frame during live streaming. The emotion indicator changes too rapidly which makes the prediction result hard to focus.

VI. Justification and Improvement

A. Problem Analysis

Inaccuracy is the main problem in our implementation. To improve the correctness of prediction, we need to refine our training set and calibrate our recogniser. We recreate the face extractor[3] and emotion recogniser[4].

B. Face Extractor Improvement

With further research and understanding on emotion recognition. We conclude that as for human, most of our emotions leads to a significant change on our eye region (i.e. eye brows and eye lips) and mouth region (i.e. the angle formed between lips and cheeks). For instance, when we are happy, our lips are open and teeth are exposed with our eye brows lift up a bit. While we are sad, our eye lips stay closer with the mouth corner drops. Such details improve the accuracy on emotion detection.

1) Extraction Region Calibration: We decided to restrict our sampling region. We use Haar-cascade for eyes and mouth to crop out the face components we need. For the eye region, the descriptor detects each eye independently. To find the region of both eyes, we need to calculate the minimum and maximum on x-axis and y-axis to find the boundary of our eyes. Unfortunately, the mouth descriptor often picks eyes due to their shape. Therefore, we separate upper half and lower half of the faces and do eyes and mouth detections respectively.

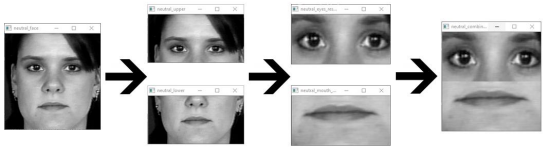


Fig. 9. Improved extractor forms training image from eyes and mouth region

[1]The motion encoding list from 0 to 7 would be ”neutral”, ”anger”, ”contempt”, ”disgust”, ”fear”, ”happy”, ”sadness” and ”surprise” according to Cohn-Kanade+ documentation

[2]In live demo, the face position, background change, illumination and noise are main uncertainties

[3]Face extractor implementation sits in Processor.py and Sampler.py in our code base

[4]Emotion recogniser implementation sits in Recogniser.py, Sampler.py and Trainer.py

2) Image Processing: We combined eyes and mouth into one image to form the training set. The difference in skin colour leads to large variance in training process. We overcome it by using histogram averaging on the last stage of processing. Such procedure effectively takes out background from the original images and concludes a highly compact eyes-mouth training data set with less redundant information.

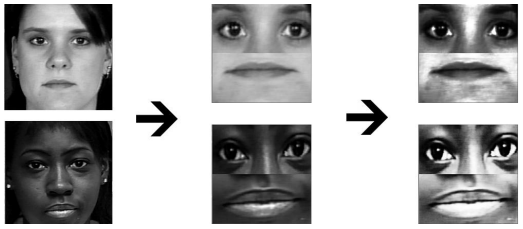


Fig. 10. Comparison on different skin colour with histogram averaging

C. Emotion Recogniser Improvement

We improve our programś performance and user experience during the second half of our project.

1) Eigenface vs Fisherface: During dimension downgrading, the Eigenface algorithm throws away some discriminant points and retains the largest variance of the image. The Eigenface performs much better in feature compacted image (which is our case) compared to Fisherface. Fisherface takes in much more features from the image which leads to over classifying and more redundant computation. Therefore, we pick Eigenface recogniser in our final implementation.

2) Eyes and Mouth tracking: We used three Haar-cascade descriptors for frontal face (outlined with green), eye (outlined with red) and mouth (outlined with blue) detection. The descriptors sometimes lose track on our eyes and mouth, which makes the emotion indicator flicker a lot. To keep it consistent, we use an estimation to calculate the eye region (outlined with pink) and mouth region (outlined with light-blue) depends on the face position in video[5].

3) Prediction Confidence Calibration: Instead of displaying emotion indicator just in time, we accumulate the confidence of emotions in a histogram in second interval. We weigh the confidence with quadratic co-efficiency. Then, the emotion indicator will refresh the display every second. If the variance between each emotion is larger than prediction threshold[6], the first two most likely predictions will be displayed. Otherwise, question marks will be displayed to indicate unclassified emotion in the indicator.

[5]Parameters of region estimation are listed in config.py

[6]Threshold for emotion prediction value is stored in config.py

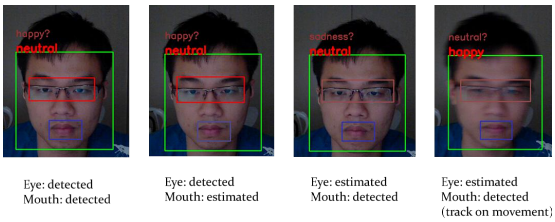


Fig. 11. Facial components position estimation

D. Testing and Evaluation

we tested our program on 2 set's of data, one is static data set where we only test the classifier on data from the Cohn Kanade library that was not used in the training process, the static testing code is located in Recognizer class. The other is live stream testing.

As mentioned early, we implemented both the Fisherface and Eigenface recognizer, for each of them we did a static test and live test

1) Static Data: the static data test is tested using a snippet of python code which can be found in the Recognizer class, we count the number of correct and incorrect cases and output a confusion matrix displaying the result, based on the result an accuracy is displayed in percentage

```
('Rate of success: ', 0.68)
0 miss
[[7, 1, 2, 0, 0],
 [0, 9, 0, 1, 0],
 [4, 0, 5, 1, 0],
 [1, 0, 0, 8, 0],
 [2, 2, 1, 1, 5]]
```

Fig. 12. static Fisherface confusion matrix, formatted in text editor, actual printing format is different

```
('Rate of success: ', 0.612244897959)
1 miss
[[6, 0, 3, 0, 1],
 [0, 8, 0, 1, 0],
 [3, 0, 7, 0, 0],
 [0, 2, 0, 8, 0],
 [3, 0, 2, 4, 1]]
```

Fig. 13. static Eigenface confusion matrix

the order for the confusion matrix from left to right is neutral, anger, sadness, happy and other, we used 10 images for each emotion, which sums to 50 total.

for static data Fisherface has an accuracy around 70% which is similar to Eigenface, with an accuracy also around 70%

2) Live Stream: for live stream it is not possible to do code testing because the change in frames is too fast and we don't know which frame is the face we actually want to test our result on. For live image we base our testing on observation, basically what we do is we accumulate the number of emotion during a second since we believe the human can not change emotion that rapidly. We press a number key which corresponds to a emotion encoding during run time to tell the computer that we are making this emotion. The program would count the accumulated emotions and use the highest occurring emotion as the prediction, it will compare the emotion with our input and update the confusion matrix. We did about 10 samples for each emotion group, which adds up to 50 in total.

for the live testing we had 2 issues

1) **Lighting**
2) **Camera quality**

As stated before, noise is crucial. We tested our data set on 4 different combination. We did the above testing procedure on each of the 4 cases. We used 2 set of cameras, one is 1080P which is the good quality camera and the other is 360P which is a poor resolution. For the lighting, good lighting is when we close the light during daytime and bad lighting is when we have a light bulb right above our head.

1) good lighting + high quality
2) bright lighting + high quality
3) good lighting + low quality
4) bright lighting + low quality

```
('Rate of success: ', 0.446808510638)
3 miss
[[5, 2, 0, 0, 2],
 [1, 8, 0, 0, 1],
 [2, 0, 3, 1, 4],
 [2, 1, 1, 5, 1],
 [6, 0, 2, 0, 0]]
```

Fig. 14. Confusion matrix for good lighting + high quality

```
('Rate of success: ', 0.375)
2 miss
[[4, 1, 2, 0, 2],
 [1, 6, 2, 0, 1],
 [2, 1, 4, 1, 2],
 [3, 1, 1, 4, 1],
 [4, 1, 1, 3, 0]]
```

Fig. 15. Confusion matrix for bright lighting + high quality

```
('Rate of success: ', 0.3333333333333333)
1 miss
[[3, 7, 0, 0, 0],
 [2, 4, 0, 4, 0],
 [0, 1, 4, 5, 0],
 [4, 3, 0, 2, 0],
 [0, 0, 0, 0, 0]]
```

Fig. 16. Confusion matrix for good lighting + low quality

```
('Rate of success: ', 0.222222222222)
5 miss
[[2, 1, 4, 0, 1],
 [3, 3, 2, 1, 1],
 [2, 3, 1, 1, 2],
 [2, 2, 3, 3, 0],
 [2, 0, 2, 3, 1]]
```

Fig. 17. Confusion matrix for bright lighting + low quality

VII. Conclusion

Our final application is an individual frontal face emotion recognizer with accuracy between 30% to 40% on live stream video.

A. Issues

It still has low accuracy in comparison to the original benchmark

B. Extensions

There are a couple of extensions that could be performed

1) Increase accuracy via noise cancelling mask
2) side face recognition
3) more emotions

1) Noise Cancelling: we concluded that the major issue with our application is that there is too much noise due to lighting and low camera quality. We think that we could use a modified average distance mask to cancel the noise, the issue right now is that the normal mask takes too much time and also diminishes some feature.

2) Side face recognition: we can train a data set based on the side face and also implement a side face detector by using the Haar Cascade recognizer. We can make 2 classifier one for the frontal face and the other for the side face, if the frontal face was not detected we try to detect the side face and based on what is detected we use a different classifier.

3) More Emotions: Currently due to memory limitation we can't process too many emotions, we could modify this by either decreasing the training set for each emotion and increasing the total amount of emotion, this will decrease accuracy because the outlets are taking more weight, or we can use C++ to rewrite the application.

VIII. Subdivision of task

A large amount of the code is done under peer programming where one person types the code while the other one corrects any error the other makes. Peer programming is a tested method which is introduced to us in $COMP2041$, it greatly reduces error rate caused by typo which is the most common error in all programs and since we are using python a typo for variable name can be devastating and very hard to detect.

Before starting peer programming, we decided that each of us should create a template for the classes we are going to use.

A. Hanzhang (Roger) Zeng

Roger is assigned the classes for

1) Processor: crops and dispatches the images
2) Trainer: the trainer program for training the image sets

B. Tianqi (Carl) Liu

Carl is assigned the classes for

1) Recognizer: the recognition program using the trainer
2) Sampler: the sample that samples our own face to add to the data set

We schedule time to do peer programming and on friday each week we would meet up to do a code review and display what we have come up with and what problems we have.

References

[1] L. Rainer, K. Alexander, P. Vadim
Intel Co.
Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection

[2] opencv.org
http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html
Face Detection using Haar Cascades

[3] opencv.org
http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html
Face Recognition with opencv

[4] T. Shakunaga, K. Shigenari
Dept. of Inf. Technol., Okayama Univ., Japan
Decomposed eigenface for face recognition under various lighting conditions

[5] P.N. Belhumeur, J.P. Hespanha, D.J. Kriegman
Dept. of Electr. Eng., Yale Univ., New Haven, CT, USA
Eigenfaces vs. Fisherfaces: recognition using class specific linear projection

[6] G. Hyoun-Joo, C.K. Keun, K. Sung-Suk, C. Myung-Geun
School of Electrical and Computer Engineering, Chungbuk National University Cheongju, Korea
Face Recognition for Expressive Face Images