SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

Final Report

Face and Object Recognition

Mohammad Asif
8955195

Supervisor:
Dr Hujin Yin

29 April 2016

Abstract

An in-depth analysis has been performed on the different types of face/object detection algorithms available as well as face recognition techniques. These include the Viola-Jones Haar Classifier cascade method to train new objects for detection with examples on how to implement using the OpenCV API. Local Binary Pattern Histogram method (LBP) and Eigen Face Recognition has been inspected for recognition and with the aid of experiments, LBP has been chosen for the industrial project. Further improvements in recognition accuracy have been examined with face alignment being the major focus. Two different types of alignment have been inspected – eye cascade alignment using the Viola-Jones Haar Classifier and facial landmark feature detector using the Flandmark API. These have been applied on the ORL Database and tested for the gain in accuracy. Post-processing images to aid in recognition has also been inspected with the DoG (Difference of Gaussian) method. In the context of the industrial project – a thorough look at the results obtained when training new objects for detection has been presented and numerous algorithms have been looked at to solve various problems such as the training of letter-based logos. Traditional OCR has been investigated for a solution but ultimately a custom cascade-based letter detection has been developed to solve the issue.

Table of Contents

1. Introduction

Machine-assisted image detection/recognition has reached a high level of maturity and is making its way into consumer-level applications such as verification at airports and automatic tagging of faces in images. Implementation on mobile device applications such as on Android has however been sparse with the only exception being unlocking the device with face recognition instead of finger-print/pattern. This has however seen little adoptation as the feature itself (Android SmartLock) warns that it can be temperamental and anyone with similar features can unlock the device (hence low accuracy). It is evident from this that the accuracy of recognition needs to be inspected to further develop the technology. One of the primary factors contributing to the accuracy is the quality of the training images. Variables such as light intensity, facial orientation, image resolution in the training sample can affect the overall recognition accuracy drastically.

The ORL Database[1] has been collected with these variables in mind but the orientation of the faces are not perfect which can hamper recognition. This can be seen in figure 1 where the sample detected face with detected eyes has been rotated automatically to align the eyes on the same *x* coordinates. The methodology behind this process has been discussed in more detail in Section 5.

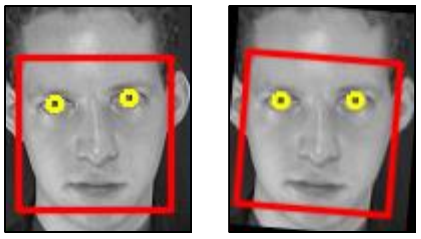


Figure 1 : ORL Database Alignment

The first part of the project has aimed to inspect the different types of image detection / recognition algorithms available (primarily the OpenCV API[2]) and implement them in real-time recognition scenarios. Extra features in the recognition pipeline such as post-processing have been incorporated as additional stages to the algorithms to improve the detection accuracy. Multiple experiments have been performed to investigate the accuracy gains achieved by adding these features to the overall recognition.

Another area the project has focused on is the facial landmark detectors concept. It offers new opportunities over traditional face detection with bounding boxes as the feature coordinates can be used in further verification. In the project's context, it has been implemented into the alignment stage preceding recognition to boost the accuracy. An example of a successful recognition with facial landmarks on an Android device is shown in figure 2.

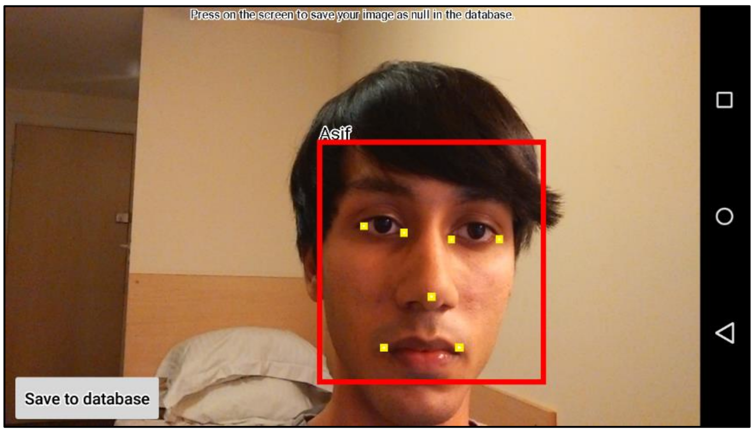

Press on the screen to save your image as null in the database.
Asif
Save to database

Figure 2 : Recognition on Android Device

The second part of the project focused on delivering a robust solution to an industrial company for their database implementation. The details of the work carried out for the industrial company cannot be divulged (due to NDA) but a broader discussion can be undertaken on the methodolgy used.

There are numerous companies and industries that have internal inventories for the products sold. This can range from grocery items in a store to car models in a car show-room. Typically these inventories have to be maintained and recorded by humans which can become challenging when the volume of products increase substantially. Machine-assisted image recognition provides a unique opportunity to maintain the inventory database and sort new items automatically as they are added to the database. The project will focus on recognizing car logos as an example and how they can be utilized to automatically categorize in the database. Along with object recognition, face detection and recognition will also be added as it presents an opportunity for the database to track employees and potentially work as a virtual manager to delegate physical tasks. An example detection of a car logo is shown in figure 3.

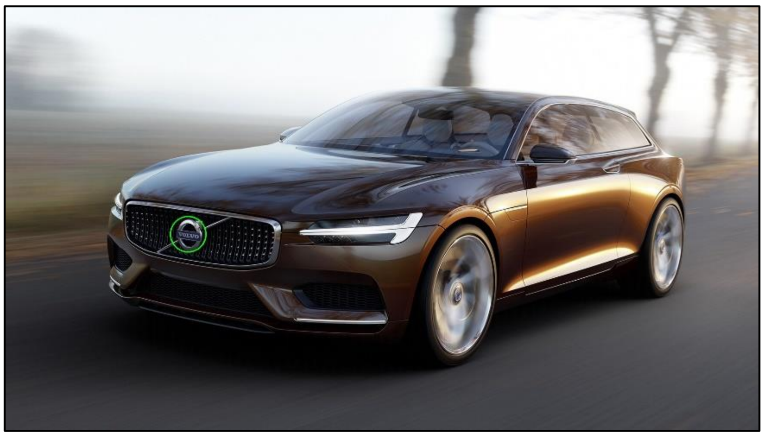


Figure 3 : Object Detection – Logo[6]

2. Aims and Objectives

2.1 Android Requirements

The initial aim for the Android part of the project was to get accustomed to the Java wrapping of the OpenCV API and develop a standard face detection and recognition application. Once that was completed, the main objective was to improve the accuracy by exploring different features that aid in the recognition proccess. The following were planned out –

- Use cascade-style detection with Viola-Jones Haar classifier to detect eye coordinates and then align and crop the faces.
- Implement facial landmark detector using the Flandmark library and extract the coordinates of the features of the face.
- Replace the cascade style eye alignment with the landmark alignment.
- Compare the accuracy and processing time of the two different types of alignment with experiments.
- Utilise post-processing to further increase accuracy by implementing the DoG filter on faces before storing in database.
- Compare the accuracy gains achieved using post-processing.
- Compare the advantages of Eigenface and LBP with varying database sizes.

2.2 Industrial Specification

An abstract view of the industrial specification will be given by considering the training of general objects rather than a specific product. The industrial project was developed on the Linux OS (Ubuntu) as it was aimed for the server platform running CentOS. As such instead of Java, the program had to be written in C++ and native C wrapping of the OpenCV API was used. The output of the program written will be a JSON formatted text file that will contain the details of the recognition (coordinates) so that it can be utilised by any database software that the company is running. Below are the objectives that were planned out for the training of an object–

- Collect sample images of object to be detected.
- Use OpenCV cascade training function to train the object using the collected samples.
- Explore alternatives for letter-based objects (logos, posters) particularly OCR.
- Test the trained detectors and refine the parameters for higher accuracy.
- Implement Viola-Jones Haar Classifier detection of faces and LBP based face recognition.
- Improve the accuracy of the recognition by the features implemented in the Android objectives – i.e image alignment, post processing.

Accomplishing these objectives would meet the aim of the industrial project and would be a successful implementation of machine-assisted detection/recognition in solving real world problems. Additionally the industrial part of the project had strict time dead-lines and the objectives outlined above were scheduled to be carried out in under 2 months time.

3. Literature Review

3.1 Methods for Detection

There are several detection methods available such as SURF[3], SIFT[4] and Viola-Jones[5]. Since the project largely revolved aorund the OpenCV API and the library only supports Viola-Jones detection as open-source, much of the work was focused on this algorithm.

3.11 Viola-Jones Haar Classifier

This is an algorithm which extracts the distribution of pixel intensities within a positive image database and compiles them using machine-learning into a cascade file that can then be used for object detection. The first step to the algorithm is outlining *Haar-like features*[7] within the positive image database. These are features that are common to all samples within the positive database – for example in a Volvo car logo, each sample will have the letter V which can then be presented as a Haar feature. The arrangement of the pixel distibution in each Haar feature is constant within a preset threshold which gives the algorithm the ability to detect these Haar features in a new image. Once the features have been collected, the next step is to calculate the integral image which is the sum of the neighbouring pixel intensities within each Haar feature. Following this, the data collected is sent as an input to the machine-learning algorithm which goes over the information and extrapolates similar features using the negative images provided. This is done to increase the accuracy and the detection rate.

The machine-learning algorithm used for Viola-Jones Classifier is the algorithm AdaBoost[8]. Typical Boost algorithms create several weak classifiers using the positive data provided that can vaguely detect the object and then combines them to form a strong classifier. AdaBoost differs from the Boost algorithms by going through the disagreements in detection made by the weak classifiers and then modifies each of the weak classifier by assigning accuracy weights to be more in line with the neighbouring classifiers. Following which they are combined into a strong classifier (Equation 1[8]) capable of providing high detection and accuracy rates.

$$H(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \qquad [1]$$

In equation 1, *H(x)* is the function representing the strong output classifier and $h_t(x)$ is the function representing the weak classifier. The variable α_t represents the weighted value of each weak classifier and it can be deduced from the equation that the strong output classifier is the sum of the weighted weak classifiers where *T* is the number of rounds of training. The weighted value of each weak classifiers can be represented by the following equation[8] –

$$\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t} \qquad [2]$$

The weighted value is calculated by the error, ϵ_t which is the error generated by the respective tested weak classifiers at each stage. By scaling each classifier with the error generated it allows the strength of that classifier in the overall procedure to be adjusted accordingly. For example a strong sub classifier will have a low error rate and hence will have a higher weight according to *Equation 2* and will hence retains its characteristics in the subsequent training phases. Following which a normalization phase can be carried out if necessary.

3.12 Cascade Training[9]

The OpenCV library provides several useful tools that allow the Viola-Jones Haar Cascade Classifier to be trained from scratch for a given object. There are several key functions required which have to be used in respective order as well as key parameters that have to be set for the cascade to have a satisfactory detection rate. The first step for the training is to collect and crop images into a folder named positive. Once that is done, the next step is to use the bash terminal in Ubuntu to generate a text file which contains the absolute path of all the positive images. This can be done with the following command –

```
$ find positive -name '*.jpg' -exec identify -format '%i 1 0 0 %w %h' \{\} \; > info.txt
```

Another text file has to be created which will contain the absolute path of the negative images. This can be done in a similar fashion to the positives with the following command –

```
$ find negative/ -name '*.jpg' > neg.txt
```

The next step is to combine the positives into a vector format file which the OpenCV function can further process. There are two parts to this – first using the following command, the individual positives are converted to .vec format files.

```
! /bin/bash
        for((a=0; a<=59 ; a++))
                do
                        opencv_createsamples -img positive/$a".jpg" -bg neg.txt -
                        num 100 -vec vecCollection/$a".vec"
                done
```

The above code part assumes that the number of positive images is 60 (since the variable *a* is starting from 0). This command will create 60 .vec format files which have to be merged into a single .vec file which can be done with the following command –

```
python mergevec.py -v /home/zod/Desktop/vecCollection/ -o
/home/zod/Desktop/merged.vec
```

The merging requires the *Python* library as well as the script *mergevec.py*[10] which will produce a single output 'merged.vec' file. This concludes setting up the environment for training and the actual training can be started with the following command –

```
opencv_traincascade -data data -vec "merged.vec" -bg neg.txt -numPos 4400 -
numNeg 2500 -numStages 18 -precalcValBufSize 2560 featureType Haar -w 24 -h -
```

minHitRate 0.995 -maxFalseAlarmRate 0.3 -weightTrimRate 0.95 -maxDepth 1 -maxWeakCount 100 -featSize 1

The function[11] has a total of 16 parameters and a brief description for each is given below along with the justification of the numerical value chosen for each parameter.

-data **:** This is the directory in which the output cascade file and the intermediate stage files will be saved at. The intermediate files are useful in the event an interruption occurs in training the cascade at which point they can be used to resume from the last completed stage.

-vec **:** This is the absolute path of the combined vector file generated from the positives/negatives wich contain the information regarding the directory of the positive/negative images.

-bg **:** The absolute path of the text file generated from the negative images folder which contains the decoded information of the negative images.

-numPos **:** This represents the number of positives generated in the .vec file along with the machine-generated samples using the negatives. The value should be lower than the total samples available since at each stage a higher amount of samples are used until the false alarm rate is fixed for each stage. Since 6000 samples (including machine-generated) were available, 4400 images were used in the first stage.

-numNeg **:** The number of negative samples available.

-numStages **:** The filtering of false positives within the training procedure occurrs in several stages, at each stage the false alarm rate is tuned to be more accurate. Too many stages will set the false-alarm rate too high and no detection will be made while conversely a lower value will increase the amount of false positives detected. Though trial and error it was found that 18 stages provided a satisfactory result with minimum false positives.

-precalcValBufSize **:** This is the size of the RAM used for the haar-like feature calculations in megabytes. Higher values would naturally make the whole process faster. For the project since a virtual machine with 4 gigabytes of RAM was used, the cascade training was given 2.5 gigabytes of memory.

featureType **:** This the type of features extracted in the first step which in this case is *Haar* as the function implements Viola-Jones classifier.

-w **:** The width of each image in the positive sample have to be constant and this parameter represents that value in pixels. Increasing the size costs performance speed so a value of 24 pixels was used.

-h **:** Similar to the width, the height has to be constant in the positive sample as well. A value of 24 pixels was used.
-minHitRate **:** This is the maximum confidence score desired by the cascade training out of 1. Values close to 1 will make the detector very strict and will not tolerate any rotation or orientation change in the positive sample. Since the project was trained for real-word objects with low variance (compared to face cascade training which exhibits high variance) this parameter was set high (0.995).
-maxFalseAlarmRate **:** As mentioned in the number of stages parameter, the cascades are trained until they satisfy the false alarm rate at each stage and this represents the maximum value that can reach out of 1.
-weightTrimRate **:** This is the rate at which the weight is adjusted on weak classifiers in subsequent stages. The default value of 0.95 out of 1 is used.
-maxDepth **:** This is the maximum number of lower branches that the binary tree structure generated by the algorithm can have. Limiting the value makes the process faster at the cost of accuracy. The default value of 1 is used.
-maxWeakCount **:** This is the maximum number of weak classifiers that the algorithm will allow to to be retained at each stage in the Adaboost algorithm (Section 3.11).

The training took around 4 to 5 hours with the parameters specified and at the end, the data folder will contain a file called cascade.xml which has the information necessary for a successful detection. The cascade file can then be supplied to the OpenCV API for the detection of the object.

The next step is to apply the generated cascade file for a successful detection. Since the cascade file has to be loaded from memory and parsed to extract the data, it is recommended to execute on a separate thread. OpenCV API provides the function required for detection –

cvHaarDetectObjects (image, cascade, buffer, scale factor, neighbours, type);

The first three paramters of the function are the image to be tested, the trained cascade file and the pointer to the memory buffer in which to perform the operation. The parameter scale factor is the ratio by which the bounding box window for the detection will increase at every step so for a value of 1.1, the window will increase by 0.1 or 10 percent at every step. Lower values as a result make the process slower but more accurate. The parameter neighbours is the number of overlapping detection to be made so a lower value will make the detector more prone to false positives.

3.2 Methods for Recognition

A brief overview of the different types of recognition was carried out in the progress report (Section 9 - Appendix). The methods were analysed in more detail in the hopes that the inner working would afford some clues for further optimisation. This was successful in the case of LBP and led to significant improvements which will be discussed with the experiments. Below is the detailed description of the method analysis.

3.21 Local Binary Pattern Histogram (LBP) Recognition

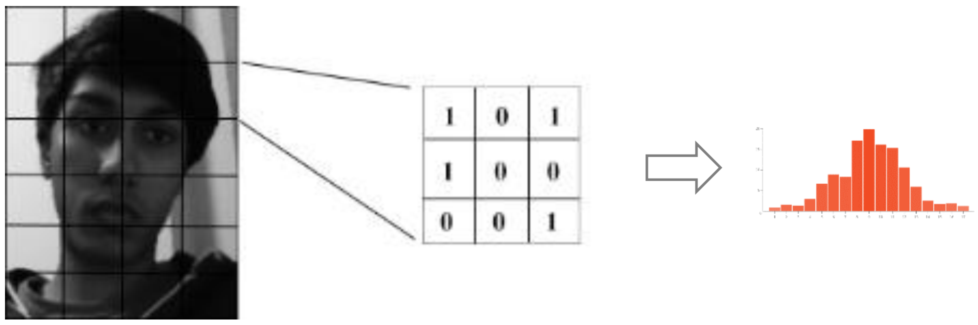
As the name suggests, this algorithm deals with the histogram of pixel distribution by assigning the individual pixels, binary values according to their location respective of each other. This has the benefit of filtering out any light variance since a higher intensity light on a picture would affect the whole image equally and hence cause the values of all pixels to rise which will keep the histogram the same (the distribution of the pixel values relative to each other is still constant). The thresholding can be shown in the equation -

$$F\ (x, y) = \sum_{P=0}^{P-1} 2^P S(i_p - i_c) \qquad [3]^{12}$$

In the above equation, the variables *x,y* are the coordinates of the central pixel while *P* is the point representing the neighbouring pixels. The variable i_c is the intensity of the central pixel while the variable i_p is the intensity of the neighbouring pixel. *S* represents the function to denote *x* has to be greater than or equal to zero to keep the function value over zero. It can be deduced from the equation that the intensity values of the pixels relative to each other is being summed around the central pixel. Once the value for the particular pixel region has been generated, it is repeated over all the regions throughout the image following which a histogram is created with the following equation –

$$H\ (i) = \sum_{x,y} I\ (F\ (x, y)\ ) \qquad [4]^{13}$$

The histogram is generated by summing the function *I (A)* which takes a value of 1 if the function *F (x , y)* exists for that region and a value of 0 if it does not. The relationship is shown in figure 4.



Step 1 Step 2 Step 3

Figure 4: Local Binary Pattern Histogram[12,14]

In step 1, the image is divided into blocks comprising of similar number of pixels. In the figure 4 example, the picture has been divided into 24 blocks. In step 2, one of these blocks is shown and it has been further segmented to show the individual pixels within that block. The figure example has been shown to have 9 pixels. If the neighbouring pixels around a center has a intensity greater than or equal to the intensity of the center, then those pixels are allocated the binary value of 1 or 0 if the value is lower. In this way each block gets a binary value called the *LBP Code* which is then concatenated with the other blocks to form the histogram as shown in the step 3. As a result the information stored in the image (pixel arrangement) has been recorded and by comparing another image's histogram, a recognition can be made if it is above a accuracy threshold. This method works as intended but if the same image were to be scaled then the histogram data would not be able to recognize the scaled image. To counteract this, a varying radius is used inside which all neighbouring pixels are encoded into LBP code. Since the number of pixels can change within the variable radius, therefore features that are scaled still retain the same histogram and are hence recognizable.

The OpenCV library provides a function which allows the user to specify the different settings involved and also manually set the threshold for a successful recognition. The prototype of the function is as follows:

createLBPHFaceRecognizer (radius, x, y, neighbours, threshold)[15]

radius : This is the size of the circle around which the center pixel thresholds the neighbouring pixels. Larger value will make the algorithm slower as the number of pixels to threshold increases for the central pixel but provides higher accuracy.

x : This represents the number of blocks the image will be divided in horizontally. Higher number of blocks will increase the algorithm time as it will increase the number of *LBP Codes* to be calculated but it will yield better accuracy.
y : Similar to grid x but in this case it is the number of blocks vertically. Higher values will once again slow down the algorithm but affords better recognition.
threshold : This is the number which defines how close a sample image has to be to the database historgram to be regarded as a successful prediction. A value of zero will mean no image will be regarded as a successful prediction while higher values will progressively reduce the similarities that pictures need to have to be regarded as recognized. Testing has shown that a value of 150 is ideal as it limits false positives.

3.22 Eigen Face Recognition

This algorithm has been covered in extensive detail in the Progress Report (Appendix) along with its mathematic derivation so a brief description will be given here with more focus towards the OpenCV implementation. The algorithm utilises the Principal Component Analysis concept where a group of images is broken down into reduced dimension vectors called eigen vectors. Each vector represents the higher dimension features with the vectors being sorted into descending order – i.e the eigen faces at the start represent the features that appear the most in the dataset. The eigen faces at the end can be discarded as they contain random noise and background information. When a new image is given for detection, it is broken down into reduced dimension and reconstructed using the eigen faces after which it is checked if it matches with the dataset within a given threshold.

$$\lambda_k = \sum_{n=1}^{M}(u_k^{2T}\ \phi_n^2) \qquad [5][16]$$

In equation 5, *M* represents the number of images in the database. The eigenvalue (λ) of the *k*th vector is calculated by squaring the product of u_k which is the set of orthonormal vectors generated by PCA (as discussed in Appendix) and the difference of each face from the mean of the database, ϕ. The OpenCV library provides a function which allows these parameters to be set manually –

createEigenFaceRecognizer (numComponents, threshold)[15]

The function takes two parameters, the first of which is the number of principal components to be kept. The recommended value is 80 as after that, the important features have already appeared in the preceding eigen faces so there is no point in taking extra random features. The second parameter is the threshold which is the value that the new detection has to overcome to be regarded as a successful recognition. A value of zero will mean that no image will be recognized (no matter how similar) with higher values making it more likely to be recognized (also increasing false positives).

The primary fault with Eigen Face recognition is that it is affected by background light and takes longer to train while taking more memory (storing eigen faces to speed up recognition). A number of tests were carried out to compare Eigenface recognition and LBP recognition to decide the suitable candidate for the industrial project. The degree of performance provided has been examined in Section 6.2.

4. Technical Challenges

The project featured work on a diverse range of platforms from mobile systems (Android) to enterprise-level server systems (running CentOS[17]). As such the different systems had different processing capabilities and different operating systems which presented a major challenge in fine-tuning the application for each platform to obtain the best results.

4.1 Android Limitations

4.11 Aggressive Thread Policy

The Android Operating System enforces a strict thread management policy and forbids any long time consuming tasks to be carried out on the main thread of the application (UI thread). This is in contrast to a typical desktop application which would have the main thread load the cascade (Section 3.12) files from memory and additionally load the image database for recognition (Section 3.2). To circumvent this, the cascade files were loaded asynchronously by putting them on a separate thread (by inheriting the *AsyncTask* class) following which the result was sent back to the main thread. A side-effect of this method was that since the main thread cannot be blocked while waiting for the separate thread, the first few frames of the video captured by the front-camera would be devoid of face detection/recognition as they would not have finished loading yet (3 seconds). To reduce this loading time, another thread was opened which was delegated the task of loading the database. This resulted in the loading time being cut down to less than 1 second which was deemed acceptable in a typical consumer use-case scenario .

4.12 Hardware limitation

Image detection and recognition depend largely on the quality of the camera taking the picture, that is the image resolution. If the pixel arrangement and intensities are disrupted by the camera quality (blurred image in low-light conditions) then none of the algorithms discussed can adapt and detect as they all depend to varying extent on the pixel details. Additionally the processing power of the smartphone is another bottle-neck for image detection/recognition. Real-time recognition depends on the processing time of the algorithm per frame and as seen by the experiment carried out in Section 6.3 later, the improvements in the algorithm pipeline have a negative effect on the frames per second. A solution to this is to have the algorithms utilise the GPU instead of the CPU and have the processing done concurrently in smaller blocks but the OpenCV implementation on Android does not yet provide these features making it not viable at present.

4.2 Alignment of Perspective Rotation

Cascade training is resilient to image rotation in the *x* and and *y* axis as it machine generates the different degrees of variations automatically using which the classifier is trained. This means the cascade for eye detection is able to accommodate different rotations of the face and able to perform satisfactory detection using which face alignment can be carried out. Rotation on the *z* axis (perspective rotation) however is difficult for the cascade to detect as these variations cannot be generated automatically during the training proccess. In figure 5 a sample rotated image in *x, y* axis is shown while a perspective rotated image is shown in figure 6.

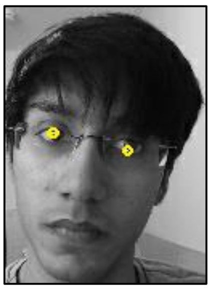


Figure 5: Normal Rotation

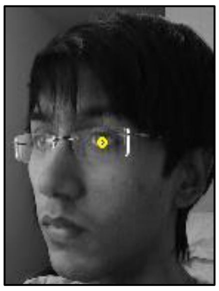


Figure 6: Perspective Rotation

It can be seen from the figure that the eye cascade was able to detect both eyes in figure 5 while only eye was able to be detected in figure 6. This presents a problem in recognition training as the perspective rotated images cannot be aligned and are bad samples that do not contribute all the necessary features. To solve this issue, a further filtering stage was added before training where images that do not contain both eyes were removed from the training data set.

5. Technical Achievments

5.1 Image Alignment (Cascade Style)

The image alignment techinique used in conjunction with eye cascade detection has already been covered in extensive detail in the Progress Report (Appendix). Using the Viola-Jones Haar Classifier method instead of detecting the face, the individual eyes were detected. The cascade classifier returned a bounding box from which the center of the box could be taken as the center of the eye coordinate. Following this, a line was drawn between the two eye coordinates and using the Pythogoras theorem the angle at which the eyes were oriented against the plane was calculated. Figure 7 (*taken from Progress Report Appendix) shows a sample detected non-aligned face while figure 8 (*taken from Progress Report Appendix) shows the calculation.

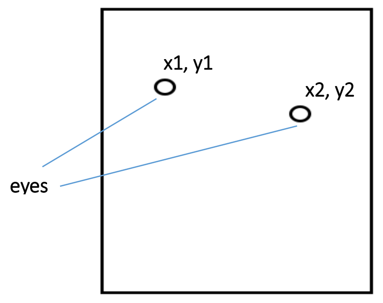


Figure 7 – Example tilted face*

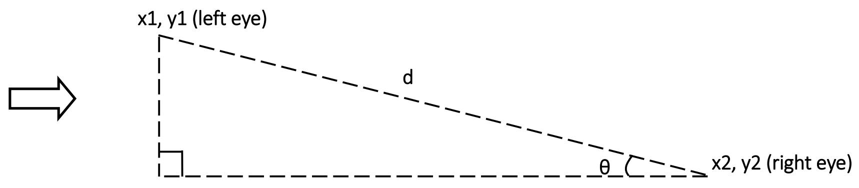


Figure 8 – Enlarged version which shows the eye coordniates.*

The main problem associated with this method was that during alignment, black bars would appear in the rotated image which would hamper the LBP face recognition later on. This is because if a high number of faces that were saved in the database had black bars, it would become a ‘feature’ in the histogram and any image that did not have black bars (for example an image that does not require alignment) would fail to be detected. An example of the black bar encountered is shown in figure 9.

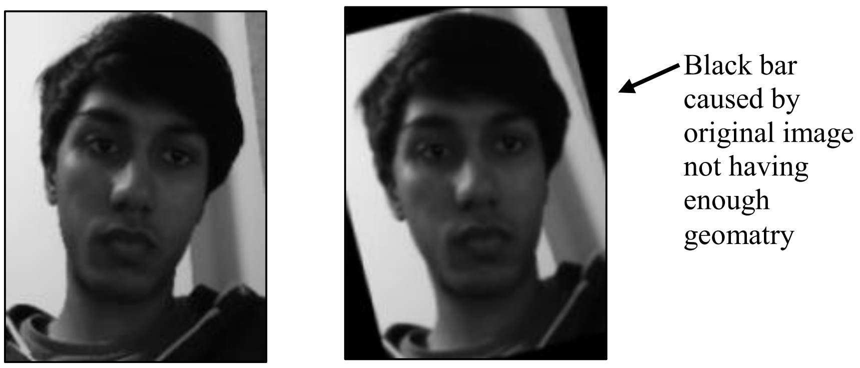

Black bar caused by original image not having enough geomatry

Figure 9: Non-aligned and aligned sample

To solve this problem, a new method was devised where the image was rotated *before* the face was cropped. This would mean that background details would persist in the image after rotation and the image could be cropped without black bars if the face was not near the edge of the image. The rotation can no longer be done from the center of the image and an additional coordinate is required around which the whole image can rotated. This method so far had not been using face cascade detection (only eye detection from which rest of face was calculated and cropped) but in order to get the center of the face for the detection, the face cascade detection was added back in using which the center coordinate was detected and the whole image rotated. Following which the face was cropped as shown in figure 10 devoid of the black bars on the sides. Additional processing time was incurred due to the face cascade detection but it was acceptable as it removed the black bars satisfactorily.

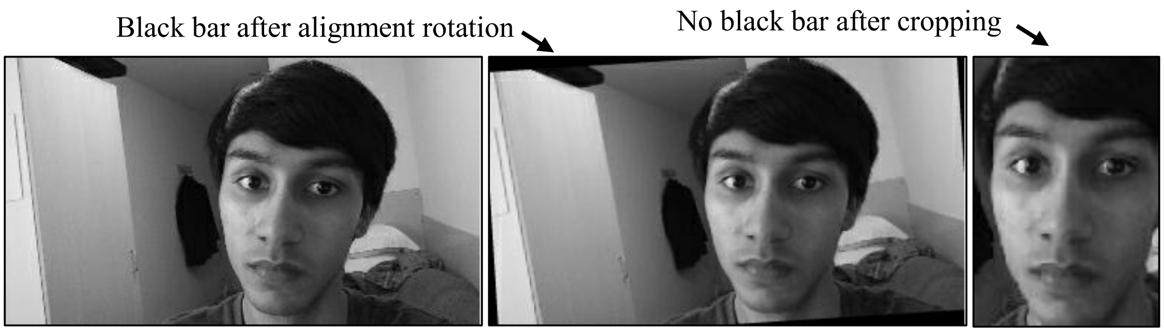

Black bar after alignment rotation
No black bar after cropping

Figure 10: Cropped Face after Alignment

5.2 Image Processing

While LBP method of detection is light invariant it is still affected heavily by spatial background information in the training images. By taking a large sample size over different backgrounds, it is possible to filter out the face only for LBP training. In a typical user case scenario, this is not feasible as there is only a limited number of training pictures that can be obtained. This was the case while working on the industrial project recognition database – only a small number of user images were provided with one static background in a video. In order to boost the recognition accuracy percentage, alternative ways were investigated to reduce spatial information from training images.

5.21 Difference of Gaussian (DoG) Post Processing[26]

This is a technique by which an image is blurred to different levels and then convoluted with itself to filter out the less prominent edges and reduce grainy, random noise in the image. The amount of Gaussian blur for each image is decided by the radius around which the edges are processed. If the Gaussian blurred output of the image is denoted by *H(x)* and *I(y)* then it can be respresented by the following equation[18] –

$$G\ (x,y) = H(x)\ I(y) \qquad [5]$$

G(x,y) represents the DoG output image and is the convolution of the two blurred Gaussian functions as seen in equation 5. The two Gaussian blurs can be represented by the following equations –

$$H(x) = e^{-\frac{x^2}{2{r_1}^2}} / \sqrt{2\pi r_1^2} \qquad [6]$$

$$I(y) = e^{-\frac{y^2}{2{r_2}^2}} / \sqrt{2\pi r_2^2} \qquad [7]$$

The two radii representing the different blur amounts for each image is denoted by r_1 and r_2 while *x,y* represent the preset standard deviation. By changing the radii, different amounts of edge filtering can be achieved in the final image. Figure 10 shows a sample cropped face being DoG processed with radii of 9 and 11 respectively.

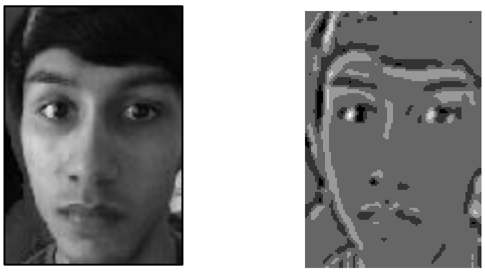


Figure 10: DoG filter

From figure 10, it can be seen that the spatial information present in the original image has been reduced to leave only the primary features such as eyes, nose and mouth. Additonally the texture of the hair – i.e the hair lines have been converted to a single uniform colour which will prevent it from becoming a feature in the dataset. The degree of performance gain achieved by the filter has been tested thoroughly in Section 6.3.

5.3 Implementing Facial Landmark Detector (Flandmark)

The main challenge associated with image alignment by detecting eyes is that in a large number of cases only one eye is detected due to bad face orientation/lighting condition. In these cases the coordinates of the other eye has to be predicted which does not lead to proper alignment. Additionally since the eye locations are being detected by cascades, the exact center of the eye (pupil) is difficult by the detector as it outputs a square box coordinates around the general location of the eye. A solution to these problems is to use more features for alignment such as the nose and the mouth. The facial landmark detector (Flandmark)[19] provides these features and was hence investigated thoroughly.

The open-source Flandmark library differs from the traditional cascade style of detection as it uses the Deformable Parts Model[20]. The Deformable Parts Model (DPM) records several features and their distances from each other in a trained file which allows the detector during run-time to understand where the other features should be once one feature is detected. The first step to this algorithm is similar to the LBP method of recognition as the image is divided into blocks[21] following which the histogram of each block is calculated (HOG – Histogram of Gradient). This gives the light invariant information in the image following which linear SVM[22] is used. This technique creates a regression model that filters out the image according to the labelled negative images provided during the training process. Once a feature is detected, the subsequent features are searched for in a bounding box around the first detection according to preset information. The distance at which the other features can be is variable within a threshold which is why the algorithm is deformable. In practice the deformable characteristic make it possible for the arms of a human to be trained following which any pose that the arms can make is able to be detected.

The derivation of the DPM model works by assigning a particular score to the trained image which is the sum of the smaller individual weighted features. When a new image is tested for detection, it is assigned a score in the same away by looking for the features and assigning scores depending on their location. If the score is within a threshold, then a succesful detection is made. The score can be represented by the following equation[23] –

$$score(\omega) = m_0(\omega) + \sum_{i=1}^{n} score_i(a_i(\omega)) \qquad [8]$$

The first quantity $m_0(\omega)$ is the score of the primary roots of the model – in this case the eye location, nose and mouth in terms of the variable ω which represnts the location coordinates. This is summed with the non-primary roots of the model which consists of other features such as skin intensity, size and shape of overall model. The variable n represents the number of these non-primary features while a_i is the score of each. The resulting output gives an indication of the score of the model and can be compared with the score of a new image for detection.

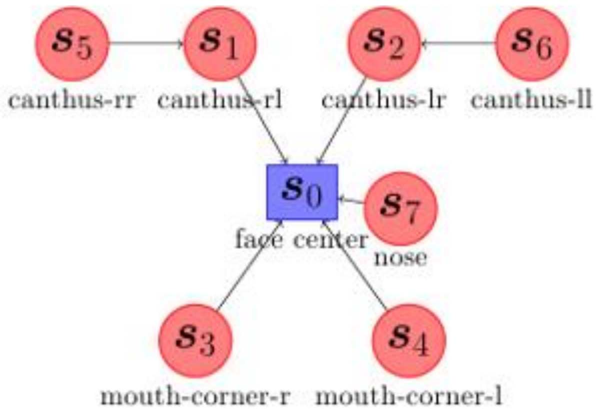


Figure 11: Flandmark Model[19]

The Flandmark library provides a DPM trained file of the human head which is shown in figure 11. It provides 8 features which are the 2 corners for each eye, 1 for the nose, 1 for the mouth and 1 for the center of the face. In order to have a higher degree of accuracy, the training for the DPM file assumes that the image supplied would be the cropped face alone with minimum background. This means that before the Flandmark algorithm can be applied, the face has to be detected and cropped using the Viola-Jones Haar Cascade Classifier method. In figure 12, a sample image of the flandmark feature detection is shown.

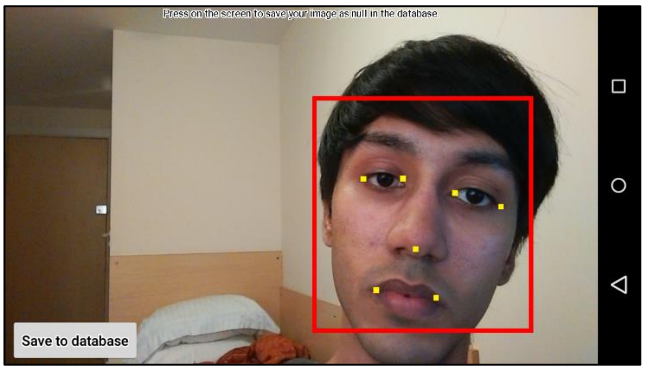

Press on the screen to save your image as null in the database
Save to database

Figure 12: Feature detector on Android

5.31 Android Implementation

Implementing the algorithm on Android as shown in figure 12 is straightforward as the OpenCV API provides a sample function with which the data file containing the DPM model can be interfaced to allow for feature detection. The first step is to initialise the flandmark model with the following code –

```
File internalFile = Environment.getExternalStorageDirectory();
File file = new File (internalFile , “flandmark_model.dat”);
Final flandmark.FLANDMARK_MODEL model = flandmark_init (
file.getAbsolutePath( ) );
```

Since this is a memory intensive process occuring over several seconds, it cannot be done on the UI thread and hence has to be carried out asynchronously on a different thread with a handler notifying when it is complete. The next step is to supply the flandmark model loaded with the image that needs to have its feature detected along with the coordinates for the boundaries of the face.

```
bbox[0] = r.x( ); bbox[1] = r.y( );                    //face boundaries
bbox[2] = r.x( ) + r.width( ); bbox[3] = r.y( ) + r.height( );
IplImage img = grayImage.clone( );                     //camera photo
flandmark_detect ( img , bbox , model , landmarks)     //detection
```

The output coordinates for the features detected will be stored in the variable *landmarks* from which the individual coordinates for the eyes, nose and mouth can be extracted. The alignment of the face can then be carried out using the method outlined in the previous section (Image Alignment). A sample aligned image using flandmark is shown in figure 13.

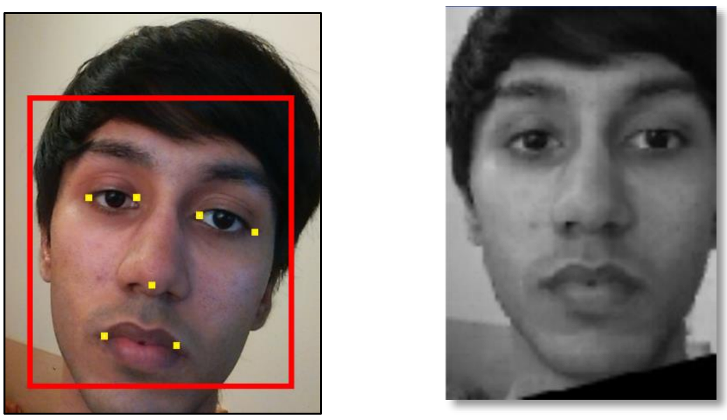


Figure 13: Alignment using Flandmark

5.4 Cascade Trained Objects

There are several steps that must be followed to train an object for detection. As an example car logos will be trained. The first step to training was to collect an adequate sample of images for each logo. Since the placement of the car logos can differ by quite a large margin, attention was paid to collecting from different places such as the front, back and from the front of the steering wheel. By collecting from different places the aim was to filter out the background details since they will not match from different places. The logo from these places will be the same and will hence result in a strong detector. Another crucial aspect was the maximum angle at which the logos were collected from. If the angle was too high then the geomatry of the logos would become too distorted for successful training. In figure 14 and figure 15 a few of the samples collected for Volvo and Seat are shown.

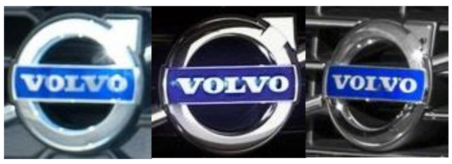


Figure 14: Volvo samples

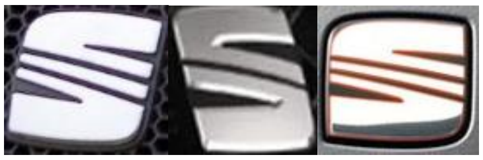


Figure 15: Seat samples

Once the logos were collected, the OpenCV function *opencv_traincascade* was used to to rotate each image by several degrees on either side to generate tens of thousands of images.

This is because collecting images by hand is not sufficient to produce all the permutations that are required for a successful detector. In addition, considerable number of negative images are needed as well. The time required to train the detector increases with the number of negative images so a thousand was selected as some testing showed that it would train each logo within 5 hours which was suitable for the time-frame of the project. The negatives had to be images which could potentially contain the logo such as the front of an arbitrary car without any logo or the side of the wheel also without any logo. Unfortunately the negatives could not be generated artificially like in the case of the positives so they were taken from public background image databases such as the Stanford Background Dataset[24]. Figure 16 shows two sample negatives that are important for adequate training.

Figure 16: Negative samples[25]

Once the resources for the training had been collected they were given as input to the cascade training function and the system was left to train for approximately 5 hours. The initial detectors trained this way (Volvo, Peugeot) were not giving satisfactory results and was hence diagnosed to identify the problem. It was found through testing that while the detector managed to detect rotated logos, it was also giving high false positives as can be seen in figure 17.

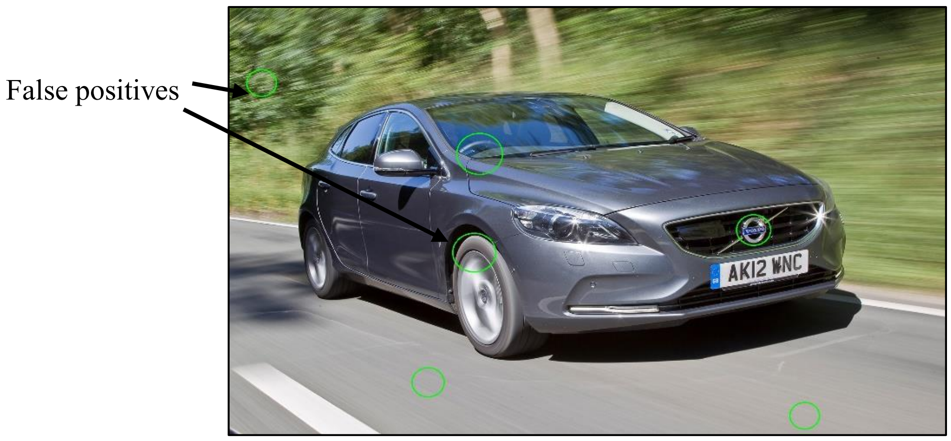


Figure 17: Low accuracy detector

This is because the similarity of the positive training images is not sufficient due to the angle of rotation being to large. Another potential reason was that the size of the positive training images were too small. To counter these problems a new set of positive samples were collected – this time with lower maximum angle of rotation and double the previous size. The disadvantage of this was that the training time increased by 1 hour but by using one more machine to train concurrently this issue was resolved. In figure 18, two succesful detection are shown with no false positives generated.

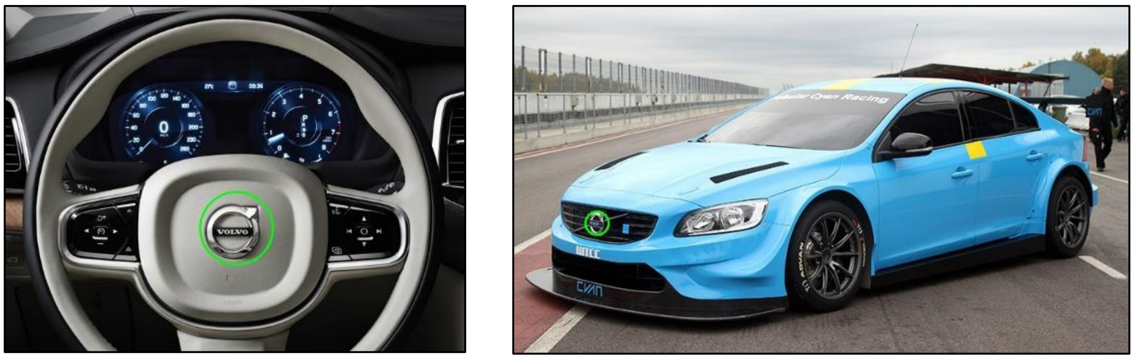
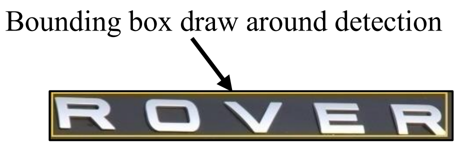


Figure 18: High accuracy detection

5.5 Cascade based OCR

While cascade method of detection works well for objects containing a good amount of features such as human faces, car logos (Volvo, Seat) – it does not work well with images that have low distinctive features and high amount of space within them. An example of this is letter-based logos such as the Range Rover logo.

Alternative ways were explored to detect objects such as these and one of the solutions found was to use OCR (Optical Character Recognition) instead of cascade detection. The methodology of OCR employs edge detection instead of cascade based training which makes it naturally very strong at detecting letters on a clear background as shown in figure 19 but against a detailed background – figure 19, traditional OCR fails to detect.



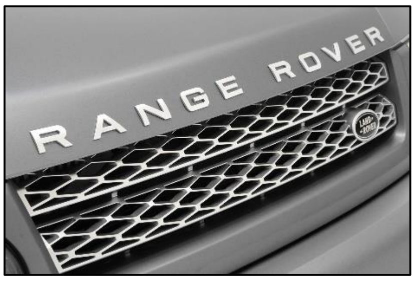


Figure 19: Succesful OCR

Figure 20: Failed Detection

Since car-based logos always have background edges consisting of the car features, it was not suitable to use this method. A different methodolgy was developed to detect these features which employs cascades but in different stages to compensate for the space between the letters. The first step to the new method is to train each letter seperately using the Viola-Jones Classifier training method. Using this classifier a cascade is generated for each letter so for the Range Rover logo, the letters R, O, V, E were trained. One option at this point is to search for each letter in the whole image and if all 4 letters are detected, it can be safely said that the image contains the Rover logo. While this works, it takes 4 times the amount of time taken by a traditional object detector since 4 cascade-detectors are working on the whole image consecutively. To solve this, the method was modified so that in the first stage only one letter is searched on the whole image – for example the letter ‘R’. Once the letter ‘R’ is detected it can be predicted that if the image contains the ‘Rover’ logo, the letter ‘R’ should have the letter ‘O’ beside it and then subsequent letters. Using this prediction, a 100x100 space (pixels) can be cropped on the left of the ‘R’ detection and then tested for the subsequent letters. In figure 21, a successful detection of the logo is shown which has been tested with the letters R, O and V.

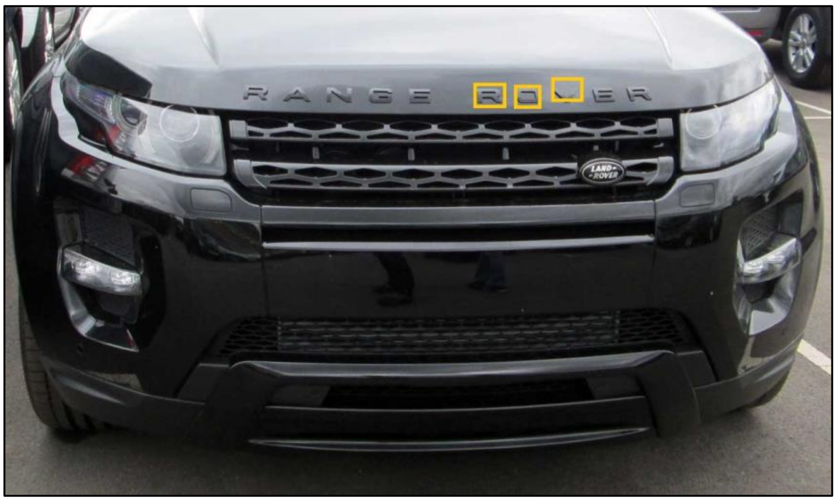


Figure 21: Cascade OCR Detection

Since the cascades are not running on the whole image, it cuts down the detection time drastically and is closer to 1.4 times the taken for a traditional cascade rather than the 4 times incurred previosuly. Since the time taken is lower, each of the subsequent letter cascades can be given a stricter false positive parameter (1.1 instead of 1.2) to make them more accurate (smaller cropped region allows this parameter without slowing down the whole process). Additionally since each of the letters have been cascade trained seperately, their false positives are different which causes the method to automatically filter out the false positives generated by each cascade as the letters are verifying each other through their position.

This interesting side-effect makes this method more accurate and resilient than the traditional cascade detectors which leads to the conclusion that if traditional detectors trained for human faces, car logos are trained with a secondary set of characteristics such as the skin tone for faces or the front shape of the car – it will verify the first set of detection and filter out any false positives and result in more accurate detectors.

6. Testing, Evaluation and Analysis

Choosing the correct algorithm for the industrial project face recognition module required some testing as their specification outlined that each database would have 20 subjects. Therefore the first experiment was designed to compare LBP and Eigenface recognition with varying database sizes. The second experiment is benchmarking the features implemented into the recognition pipeline and quantifying the accuracy gains achieved.

6.1 Dataset – ORL Face Database

The ORL face database was used to perform the experiments. It was chosen as it is a homogenous database with neutral backgrounds and a good range of emotions depicted in the different images of each subject. The database contains 40 subjects with 10 different variations of each and all of them were used to perform the experiments. Figure 22 shows 3 different subjects from the ORL database.

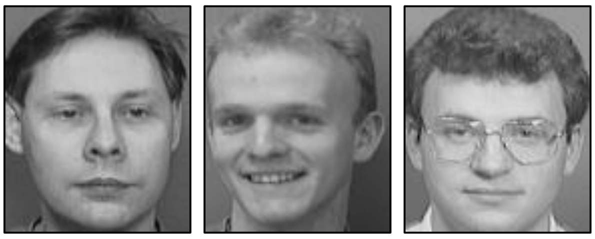


Figure 22: ORL Database

6.2 Experiment 1: Comparing LBP and Eigenface Recognition

6.21 Aim

As the number of subjects in a database increases, the likelihood of similar combination of features appearing on different subjects increases which naturally decreases the accuracy of most recognition algorithms. This is because the algorithms make predictions based on a threshold and if different subjects with one or two similar features fall within the threshold, then they are regarded as being the same. By testing LBP recognition and Eigenface recognition on varying numbers of the same dataset, it would give an indication on the accuracy expected in the industrial project database.

6.22 Design

The first subject in the ORL Database was chosen to be trained in the recognition engine following which the algorithm would have to correctly predict that subject from within the database. Since each subject has 10 samples, therefore the first five was selected to be trained while the other five would have to be correctly predicted. The algorithm would then go through the other subjects and any recognition made that was not unknown would be a mistake since these subjects have not been trained. The number of total subjects tested was started with 10 and then increased by 5 until the whole database of 40 subjects were covered. The number of mistakes made at each phase was recorded along with the total number of tests done, following which the percentage accuracy for that phase was calculated with ((number of correct recognition/number of tests)*100) %.

6.23 Results

The results of the tests with the two algorithms have been shown in table 1. Due to the limited size of the ORL Database, the tests could not be extrapolated further but the results achieved were sufficient to make a decision in choosing the correct algorithm algorithm for the project.

Number of Subjects	Total Samples	Eigen	LBP
10	100	75.79 %	83.263 %
15	150	70.34 %	84.83 %
20	200	70.77 %	83.07 %
25	250	75.918 %	80.18 %
30	300	76.95 %	82.71 %
35	350	78.55 %	81.44 %
40	400	78.99 %	81.77 %

Table 1 : Varying subjects accuracy

A graph has been plotted (figure 23) with the results in table 1 to inspect the general trend for each algorithm as the database size increases. The first point of interest is that the general percentage accuracy for each algorithm is above 70 % regardless of database size which is primarily because the ORL Database has neutral backgrounds as has already been mentioned.

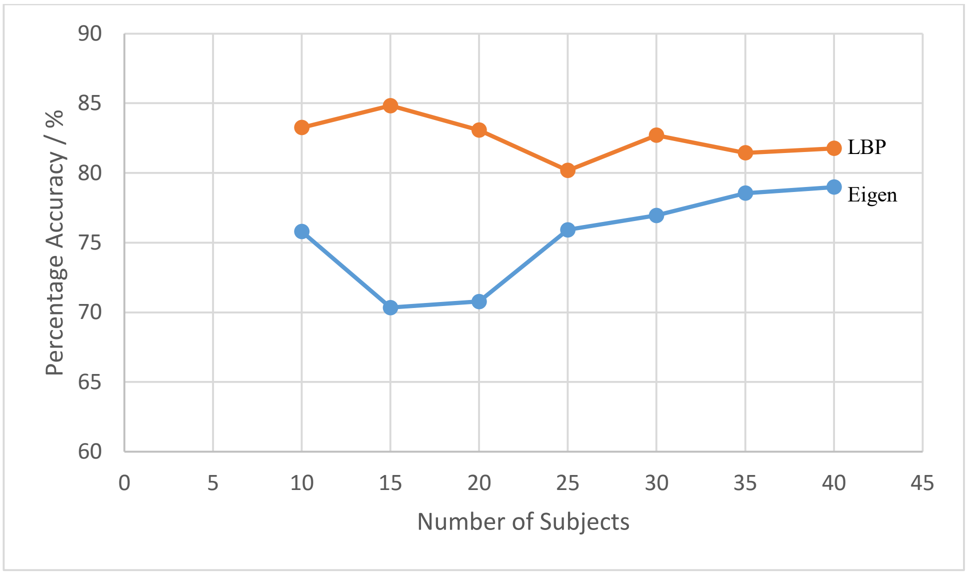


Figure 23: Eigenface and LBP comparison

LBP recognition performed better than Eigenface on all the different sample size which is expected as Eigenface recognition relies on PCA (Principal Component Analysis) which requires a high number of training images to reduce the important features into eigenfaces. The 5 training samples supplied is insufficient for adequate Eigenface recognition as a result. Conversely since LBP is histogram based and recognizes with the average of the training sample, it fared much better with the limited sample size.

As the size of the tested sample increased from 10 to 20, Eigenface recognition saw a sharp dip to 70.34 % before recovering and steadily increasing in accuracy as the size of the testing sample increased. This behaviour is again expected as the reconstruction of Eigenfaces during recognition fares better with more features available in the training set. LBP recognition on the other hand, performed better at lower sample sizes (<20) with a percentage accuracy of 84 % but as the size of the database increased, it gradually fell down in accuracy. This can be attributed to the fact that the probability of similar faced individual increases with higher number of dataset which is why LBP had sharp dips and rises depending on how similar the added faces were to the trained database. This is the primary problem with LBP as the histogram does not filter features depending on location unlike Eigenface.

Since the industrial project would have a database size ranging from 20 to 25 subjects, the obvious choice is LBP face based recognition as it provides a higher percentage accuracy according to graph. As a result LBP faced recognition was the main focus and the subsequent experiments have been based on this algorithm.

6.3 Experiment 2: Comparing Face Recognition Improvements

6.31 Aim

The goal of this experiment is to quantify the amount of improvements made by the different features that have been added to the face recognition pipeline. These are the two different types of alignment stages – eye cascade and facial landmark as well as the post-processing stage using DoG filter. The testing will be done by examining how well each algorithm feature can recognize a person given the same number of images in the database.

6.32 Design

The input to the experiment is the ORL Database but instead of changing the total number of subjects similar to the previous experiment, the number of images trained per subject will be changed. This will allow to gain an insight into how the different features react with different sizes of trained images. Since each subject has 10 samples therefore the features will be tested with 1 sample and then increased successively until 7 samples have been used. 3

samples will be left for the actual testing of the recognition along with the other subjects in the database. The percentage accuracy will be calculated the same way as the previous experiment by taking the ratio of the successful recognition to the number of total tests done.

The first test dealt with the standard LBP recognition (provided by the OpenCV API) with no added features. This will be the control of the experiment and the performance gains from the subsequent tests can be found from this control. The second and third test consisted of the standard LBP recognition but this time with alignment of the training database using eye cascades and facial landmark feature detector respectively. The fourth and final test consisted of the database being proccessed using DoG filter to filter the number of edges in the database which would leave only the major features such as eyes, nose, mouth remaining as explained in Section 5.21. Following which the filtered image was aligned using the facial landmark detector. Additionally the time taken to proccess a single image and perform a recognition was recorded in table 2 as the speed of the overall process has to be kept in mind while trying to boost the percentage accuracy. This is because in real-time recognition such as in videos, a high recognition time per image will make the frames per second low which will cause the video to lag. In applications such as sorting large image databases automatically which is similar to the work done for the industrial project, the time to proccess per image is again important as it will increase the total time taken proportional to the size of the database. In figure 24, sample images of the 4 different tests are shown. The first is standard LBP with no added features while the second has been aligned with eye cascades. The third and fourth has both been aligned by facial landmark detection with the fourth being DoG filtered in addition.

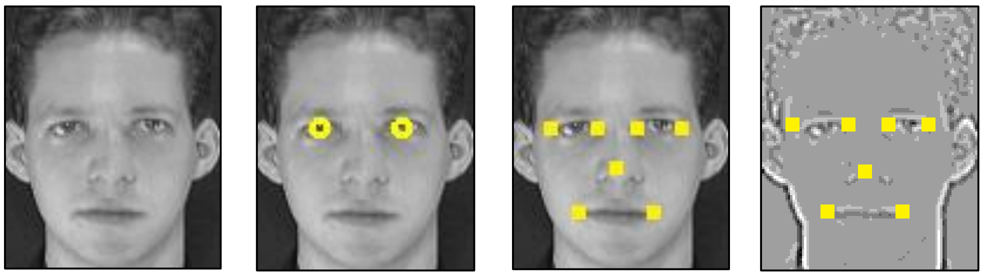


Figure 24: Output of the 4 tests

6.33 Results

Trained Samples	Standard LBP	LBP + Eye Cascade Alignment	LBP + Facial Landmark Alignment	LBP + DoG + Facial Landmark Alignment
1	81.9%	83.8%	85.8%	85.8%
2	82.9%	85.8%	88.4%	89.5%
3	83.1%	87.4%	90.7%	91.7%
4	83.8%	87.9%	90.4%	91.9%
5	84.7%	87.5%	91.2%	93.4%
6	84.6%	87.8%	91.7%	93.5%
7	84.6%	87.7%	92.4%	92.8%
Time per image	216 ms	240 ms	264 ms	270 ms

Table 2: Varying training size

The first point of interest similar to the previous experiment is the unusually high percentage accuracy achieved over all the features. This is again primarily due to the nature of the ORL Database and the background along with the neutral expressions of the subjects that output the high accuracy. A graph was plotted with the data from table 2 which is shown in figure 25.

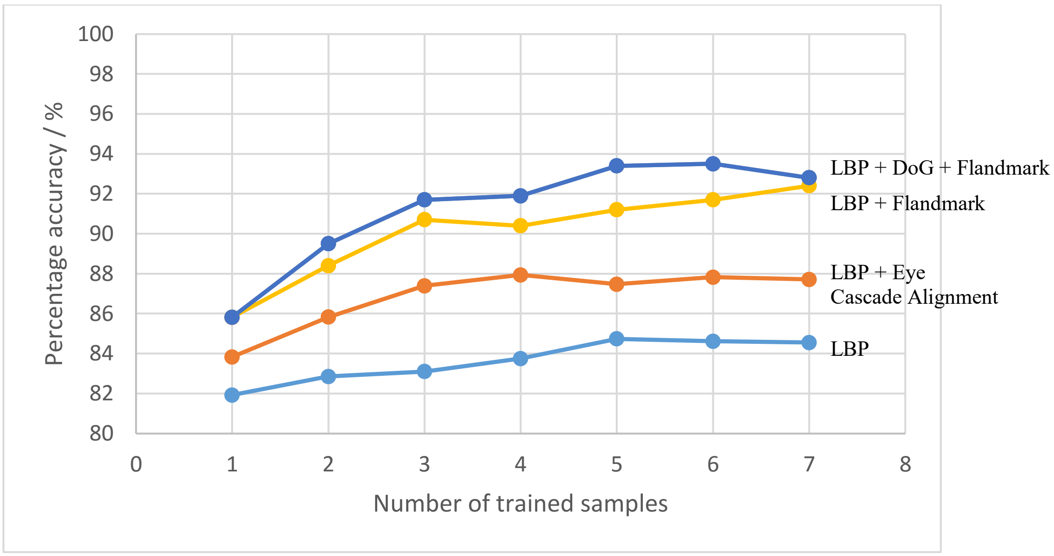


Figure 25: Cascade OCR Detection

From the graph it can be seen that the general trendline is that as the number of trained samples increases, the overall percentage accuracy increases but not at the same rate. From 1-3 trained samples, the percentage accuracy increases sharply after which it becomes more or less stable across all the tested features. This is because having too many trained samples introduces a higher amount of variance in the dataset which has a negative effect on the overall recognition procedure.

Compared to standard LBP, the eye cascade aligned LBP saw a general increase in accuracy of 2-3 %. It can be predicted that in real-world scenarios where the alignment issue is far more prominent, the percentage increase in accuracy will be even higher. The eyes are a major part of the facial feature, having many different pixel colour intensities which when converted to histogram data, results in one of the more distinctive features between different subjects. By aligning the eyes, the pixel intensities can be compared more closely and accurately which results in the boost in percentage accuracy. LBP aligned with facial landmark features saw a further rise in accuracy of around 2-3 percent. This can be attributed to the fact that as mentioned in Section 4.2, eye cascade alignment is unable to work on perspective rotation of faces. These are the face samples that require the alignment the most which facial landmark alignment solves by using other coordinates such as the mouth to align with the one detected eye. Finally adding a processing filter to the facial landmark feature saw a modest increase in accuracy of 0.5% to 1.2%. Decreasing the edges of pictures filtered out the details that vary between different images of the same person such as texture of hair (hair lines) which led to a higher percentage. At the same time, some images were affected negatively since important features were filtered out which resulted in the graph line being more jagged compared to the others. The net result is positive so the filter can be deemed a success.

The timing of the different filters is another important parameter which is listed in Table 2. Eye cascade alignment saw a rise in processing time from 216 milliseconds to 240 milliseconds which is an increase of 11.1 %. This is because eye cascade alignment requires an additional haar cascade file to be loaded and parsed which increases the overall processing time. Facial landmark detection uses the Flandmark API which packs the training data into a more dense dat format file which makes it load slower and it saw an increase in time of 22.2 %. For older devices with limited proccessing power, eye cascade alignment would be the better choice. Lastly the DoG filter saw only a 6 ms rise in processing time compared to LBP with facial landmark alignment which is expected as it is a CPU intensive function with no external file to load. This makes the filter suitable for both old and modern devices alike.

7. Conclusion

The different features implemented into the recognition pipeline all achieved favourable results but it has to be remarked that the ORL database is not representative of actual real-world data. The technical challenges outlined in the project such as perspective rotation and blurred images due to camera sensor quality will degrade the performance of the recognition considerably in real-world scenarios. In the case of the DoG filter, if the real-world image has the face out of focus then the filter will remove the face and keep the prominent background edges which is detrimental for the recognition. Nevertheless in controlled scenarios where the quality of the image received is monitored, the DoG filter can be used to good effect as shown in the experiments.

Facial landmark feature detection provides unique opportunities to perform tasks that no other algorithms are currently able to. By providing as output, 7 coordinates of the different locations of the face, it allows other algorithms to extract and use the feature data for their own use. In the context of the project, the data has been used for alignment of the face but it can be extended to emotion detection by comparing the distance between the features among a variety of other possibilities. While eye cascade alignment is successful in its results it fails to align images as accurately as facial landmark due to having only 2 coordinates to align with. However since it is faster than facial landmark, therefore mobile devices with low proccessing would be better served by implementing this form of alignment.

Cascade training of objects has provided good results with low number of training images by machine-generating more varieties by using transformations such as rotation. The only limiting factor to the method is the time taken to train as with even moderate databases, the classifier can take upto weeks to train which makes it unsuitable for training highly accurate detectors in sensitive applications. One solution is to have the cascade trained adaptively instead of the static way it is done currently – static because it is trained once and then no training data added to it. The adaptive classifier would mean that after every successful detection, the cascade would use it as a training image and become more accurate. This will require major changes to the OpenCV API however as the current functions do not offer this feature.

8. References

[1] "ORL image database" from *Cambridge University Computer Laboratory Database of faces*, http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

[2] OpenCV [2014] https://github.com/itseez/opencv

[3] **Bay, H., Tuytelaars, T. and Van Gool, L,** [2006] "*Surf Speeded-Up Robust Features*"

[4] **D.Lowe**, University of British Columbia, [2004], University of British Columbia ***"****Distinctive Image Features from Scale-Invariant Keypoints"*

[5] **Viola, Jones** "Pages 1,3,11 ", *Robust Real-Time Object Detection*, Vancouver, Canada , July, 13, 2001.

[6] Image, "Volvo Concept Car" from *Car Magazine,* http://www.carmagazine.co.uk/spy-shots/volvo/volvo-s90-and-v90-big-volvos-are-back---carrying-on-where-the-940-left-off/

[7] **Haar A**. *Zur Theorie der orthogonalen Funktionensysteme*, Mathematische Annalen, **69**, pp. 331–371

[8] **Freund; Schapire**. *"A Short Introduction to Boosting"* : Introduction to AdaBoost, *1999*

[9] **Natoshi Seo**. *"Tutorial: OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers based on Haar-like Features"*
http://note.sonots.com/SciSoftware/haartraining.html

[10] **Wulfebw** [2014], https://github.com/wulfebw/mergevec/blob/master/mergevec.py

[11] Cascade Classifier Training
http://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html

[12] OpenCV Docs 2.4 "*Local Binary Pattern Histogram*"
http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#local-binary-patterns-histograms

[13] **Caifeng Shan; Shaogang Gong; Pter W. McOwan**. *"Facial expression recognition based on Local Binary Patterns" :* Image and Vision Computing, *2009*

[14] Image, "Histogtram" from *Dataviz Catalogue,*
http://www.datavizcatalogue.com/methods/images/top_images/histogram.png

[15] OpenCV Docs 2.4 "*FaceRecognizer*"
http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_api.html

[16] **Mathew Turk; Alex Pentland.** *"Eigenfaces for Recognition"* : Vision and Modelling Group, *1991,* p 74-75

[17] Operating System, *"The CentOS Project",* https://www.centos.org/ , 2016

[18] Gaussian Blur, *"15.17 Filter primitive 'feGaussianBlur'", https://www.w3.org/TR/SVG/filters.html#feGaussianBlurElement*

[19] **Michal Uricar, Vojtech Franc** "*flandmark*", Open-source implementation of facial landmark detector, http://cmp.felk.cvut.cz/~uricamic/flandmark/

[20] **Felzenszwalb, P.F., Girshick, R.B., McAllester, D., Ramanan**, "*Object detection with discriminately trained part-based models".* In PAMI 2010. http://www.cs.berkeley.edu/

[21] **Pedro F.Felzenszwalb;** *"Object Detection with Deformable Parts Model (DPM)",* http://cs.brown.edu/courses/cs143/2011/lectures/DPM.pdf

[22] **A. Zisserman;** *"Lecture 2: The SVM Classifier",* Hillary 2015, http://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf

[23] **Pedro F.Felzenszwalb, Ross B.Girshick, David McAllester;** *"Cascade Object Detection with Deformable Parts Model",* https://www.cs.berkeley.edu/~rbg/papers/Cascade-Object-Detection-with-Deformable-Part-Models--Felzenszwalb-Girshick-McAllester.pdf

[24] **S. Gould, R. Fulton, D. Koller**. Decomposing a Scene into Geometric and Semantically Consistent Regions. *Proceedings of International Conference on Computer Vision (ICCV),* 2009.

[25] Image, *"Traffic template",* http://static6.businessinsider.com/image/521f48c669bedde80a7c8807/the-10-most-traffic-clogged-cities-in-the-us.jpg

[26] **P. Kalaiselvi; S.Nithya,** *"Face Recognition under Varying Light Conditions",* Journal of Computer Engineering, 2013

9. Appendix

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

Progress Report

Face Detection & Recognition on Mobile Systems

Mohammad Asif
8955195

Supervised by:
Dr. Hujun Yin

1. Introduction

The advent of technology has made it increasingly clear that face detection/recognition is going to play an integral part in soceity within the next decade. Self-driving cars, medical record keeping, airport security are some of the major industries that have implemented image recognition algorithms to automate and improve their services. These can be considered as the motivation behind the project. While the usage of image recognition algorithms by these industries is impressive, the world of smartphones and mobile devices have been left behind in terms of accuracy and features. This is primarily because mobile devices have to be small in order to be portable which in turn limits the quality of the camera sensors as well as the processor capabilities. Additionaly mobile devices work on the java/python platform which again limits the highly specialised libraries (C++) that have been developed for the industry counterparts. For these reasons, the project aims to update face detection and recognition on mobile devices by using the established algorithms and also to optimise (align) them where necessary to boost their accuracy.

1.1Context

There are different types of algorithms that make face recognition possible (some will be discussed shortly) but the core principles remain the same for all of them. A database of faces previously assembled is used to extract information (nature of information depends on algorithm) following which this information is used to identify a new face. This puts a bottle-neck on face recognition as it is dependant on the image database and the quality of recognition can only be as good as the database used to train it. In order to maintain high accuracy, researchers have assembled faces and have manually aligned them by cropping/rotating where

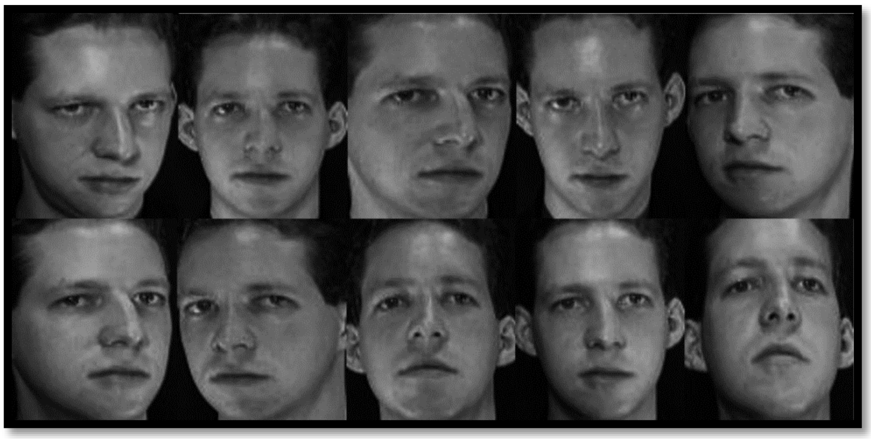


Figure 1 – Manually Aligned ORL database sample [1]

necessary. One such database is the ORL database shown in figure 1 which was manually aligned. It is widely regarded as a good face database, primarily because the eyes in each face are roughly at the same location and there is very little extra background in the images which would hamper successful recognition. If a new image were to be added to this database then it would have to be manually aligned which is not an option for users using face recognition on smartphones and mobile devices. The typical user experience would be to take a few camera shots of the face and then expect facial recognition of the user’s face without too much delay. For this reason it is important to write an algorithm that can automatically align the face as soon as the camera shot is taken which will fall more in-line with user expectation. This is one of the goals the project aims to accomplish. Following this there are plans on improving the actual face recognition algorithm by using LBP (Local Binary Histogram) which is not affected by light intensity differences in the image database. These algorithms will have to be optimised so that they can run on mobile systems which is another area that the project aims to focus on.

1.2 Aims and Objectives

The main aim for the project is to implement robust face recognition on mobile devices such as on Android and Raspberry Pi. To accomplish this, several smaller objectives have to be completed in respected order which are as follows -

- Use Viola-Haar Cascade to detect eyes/face.
- Use eye coordinates to calculate degree of rotation and then crop/rotate according to specification (automatic alignment).
- Improve face recognition by using the aligned images and replace Eigenface recognition with modified LBP.
- Go through the Raspberry Pi tutorials to get accustomed to the Pi interface and the camera.
- Port OpenCV 3.00 to Python and then implement face detection/recognition similar to the Android system.

2. Literature Review

The first part of the project deals with face detection/recognition on Android devices which is why time was allocated in researching the Android ecosystem[2]. An android application is comprised of several activities with the first activity shown to the user being known as the launcher activity. These activities hold functions and data and also '*views'* which are typically buttons, text boxes, image boxes among a variety of other things. In figure 2, an android activity

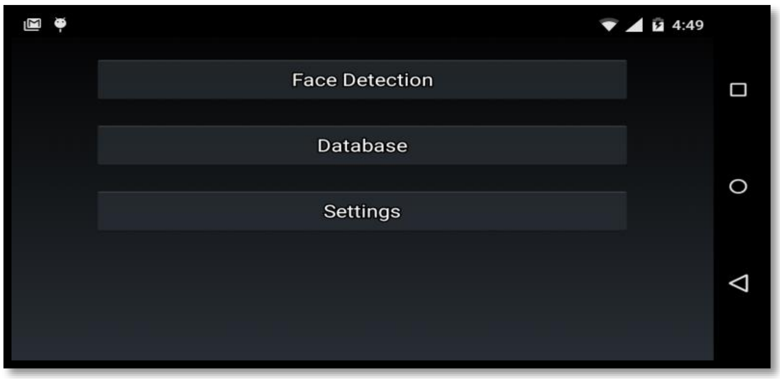


Figure 2 – Android Activity with multiple views

for the project is shown with three buttons (views) that lead to different areas of the android application. Each activity has its own UI (user interface) thread on which the cosmetic changes of the activity is displayed such as animations while lengthy calculations have to be performed on a background synchronous/asynchronous thread.

In the context of the project, the camera preview that gets shown to the user is the '*view'* (surface view to be precise) and it is running on top of the launcher activity. Since it is a view, it is considered to be running on the UI thread and hence face detection/recognition cannot be carried out on this thread. To solve this problem, a class called the *Asynchronous Background Task* was used which spawns a new background thread and has a built-in handler by which information can be sent to the main UI thread (in this case the camera preview). This allows the face coordinates to be calculated in the background thread (along with alignment/recognition) which is then sent to the camera preview and drawn on the view as a rectangle box. Since the backgroud thread is asynchronous there is a slight time cost before it notifies the main thread. There was some concern that this would generate lag on the camera preview but in practice it only dropped 1-2 frames which was barely noticeable and hence acceptable for the project.

Along with the Android documentation, considerable time was spent on going through the OpenCV documentation**[3]**. The core component that the OpenCV library works on is the data construct called Mat. Mat is a variable declared by OpenCV that is used to represent an image. It can be described as a matrix representation of the different pixels and their intensities that make up an image. From this matrix, the OpenCV functions can extract data and hence apply the algorithms on it. A few of the essential algortihms is discussed below.

2.1 Viola-Jones Detection

The algorithm used to detect faces and eye is the Viola-Jones**[4]** Detection classifier that is available as a function inside OpenCV. It uses Haar-like**[5]** features to detect the different features in an image. This works by breaking the image into many smaller regions of interest from which the difference of pixel intensity sums between each region of interest is used to classify the Haar-like features. For example the hair in a face image is relatively darker than the face so this difference in pixel intensity allows it to be classified. Similarly eyes, nose, mouth can also be diagnosed as Haar-like features. Adaboost**[6]** which is a type of machine-learning, is used to boost the detection of haar-like features so that a large number of iterations can be carried out in a short period of time.

These technical details are hidden away inside OpenCV. One of the fundamentals of designing a good programming library is to allow abstraction to make the code simpler. OpenCV provides a good level of abstraction and it takes only a few lines of code to set the cascade classifier for detection. The pseudocode is given below:

```
Extract the cascade xml file to cache memory
Initialise the cascade classifier variable with data from cache
Apply the cascade classifier on image
If classifier successful
    Return  detected faces as an array [ ]
    For each detected face inside array [ ], retrieve coordinates
        Draw rectangle around face from retrieved coordinates
```

2.2 Eigenface Recognition

Eigenface recognition**[7]** is based on the mathematical model of Principal Component Analysis**[8]**. This is a method by which a collection of images is reduced in dimension to create a set of orthogonal principal components called eigenfaces. As a result each image in the database can be represented by a combination of these eigenfaces. In practice, the features which do not match between the images (such as random background or different clothes) in the database are filtered out of the eigenfaces (this is done by discarding the eigenfaces at the end which contain mostly noise). The features which are always present between different images such as eyes, nose, mouth are seen as major features (*principal*) and show up in the first few eigenfaces.

As mentioned a database of facial images is needed for the Eigenface recognition. Let the number of images in a database called *S* be *M*. The size of each image in the database is N x N. The database can be represent as a set –

$$S = \{ T_1 , T_2 , T_3 , T_4 \dots T_M \}$$

Since the size of each is NxN therefore the dimension of each image is N^2 pixels. The next step is to normalise the image database. This is done by averaging all the images and then subtracting the average from each image in the database so that every image in the database is left behind with only its unique features. If the average face is **Ψ** and each of the normalised face image from the database is **ϕi** then the normalisation can be depicted as –

$$\phi_i = T_i - \Psi$$

Since each image is being normalised therefore a new matrix is created which can be denoted as A and it is shown below –

$$A = \{\phi_1, \phi_2, \phi_3, \phi_4 \cdots \phi_M\}$$

The next step is to calculate the eigenvectors from the covariance matrix. The covariance matrix is shown below represented by C –

$$C = A^T A$$

This means that the covariance matrix is of size M (number of images in the database) or in other words, M number of eigenfaces from which the first K eigenfaces is selected (this is because later ones are comprised of noise only). Each image in the database can now be represented by a combination of these weighted K eigenvectors added to the mean image (it was previously subtracted for normalisation). The weight is to signify how strongly each eigenface depicts itself in that image. If an unknown input image is given, it is broken down to its own weight vector and then the euclidian distance of this vector to each of the weight vectors in the image database is measured. If the distance is very high(above a previously set threshold) then the unknown input image is comprised of similar proportions of eigenfaces and hence a successful match has been found which results in face recognition.

2.3 Local Binary Pattern (LBP) Recognition

Local Binary Pattern**[9]** is a method by which an image is divided into a grid with each cell containing a set number of pixels. Each pixel is compared to each of its neighbours and when the pixel in the center has a higher pixel value, 1 is put in the binary representation and conversely if the center pixel is lower than its neighbour, 0 is put in the representation. In this way for each pixel a binary representation is created depending on the pixel values surrounding it. Following this, a histrogram is carried out in each cell to determine how often a binary representation repeats which shows the pixel combination that is present the most in that cell. Joining together the respective histograms for each cell gives a vector that describes the features present within the image. If a new image is presented, its feature vector can be found the same way and if the new vector is similar (threshold set previously) to the previous feature vector then a successful recognition is carried out. Since LBP is light independent, it makes it an attractive choice over Eigenface recogntion.

4. Technical Risk Analysis

4.1 Software Design

Perhaps the primary and most dangerous risk that any piece of technical software faces is the design paradigm chosen in writing it. Code written without proper comments or documentation along with poor data encapsulation can be difficult to maintain as time goes by. Additionally if object orientated patterns are not followed by ignoring abstraction/inheritance then it will lead to scaling problems and it will be difficult to upgrade the piece of software in the future. Since the code being written is an iterative process by constantly exploring new ways of solving the problem at hand, it became evident that the software was getting cluttered. To counter this a new strategy was devised. After every 4-5 days of coding, a few hours would be taken to go through the software in question and to look at the design methodology used and if proper documentation was being done. Appropriate comments were added where they were missing and a strict no repetition rule was maintained. This led to the software becoming much smaller in size and hence easier to maintain. A by-product of this maintenance of the code led to smaller bugs being discovered which will undoubtedly save time in the future.

4.2 Data loss

A common risk that all computer programs in general share is data loss and corruption. Old hard drives may fail and there is always the danger of damage by liquid spillage/dropping. As a result it is prudent to regularly back-up data and maintain a version history. This has the benefit of being able to roll back on changes done if they somehow introduce unseen bugs. For this project, cloud storage has been chosen to be used as the backup medium. The project is worked on a Google Drive folder so that the technical risk of losing data is safely mitigated.

4.3 Failure of planned methods

The objectives laid out to accomplish the aims of the project such as using LBP to achieve face recognition might not work or might prove to be less robust than originally intended. If this happens, alternate algorithms will have to implemented. Since Eigenface recogntion has already been tried out before, implementing Fisherface algorithm would be the next viable option. Alternatively the LBP could be modified and a custom algorithm could be used. There is a further risk that setting up OpenCV for Raspberry Pi might fail due to bugs or the general unfamiliar platform of the Pi. In the event of this happening, a python matrix library would be used to simulate matrices and matrix operations similar to Matlab. The face detection and recognition algorithms would then be written from scratch using the matrix representation of the data/image.

Aside from these identified risks, many more techincal challenges and risks will show up during development. Constant vigilance will have to be excercised to make sure all these new future risks are mitigated properly.

5. Practical Progress

The opening few weeks were devoted to developing the image alignment module. For the first prototype, a non real-time system was considered. Using face detection with the Viola-Jones haar cascade classifier in OpenCV, faces were detected and stored in a database. These faces had varying degrees of rotation and sizes depending on if the user was closer to the camera or further away. Below in figure 3 is an example database without any image alignment or processing done on it.

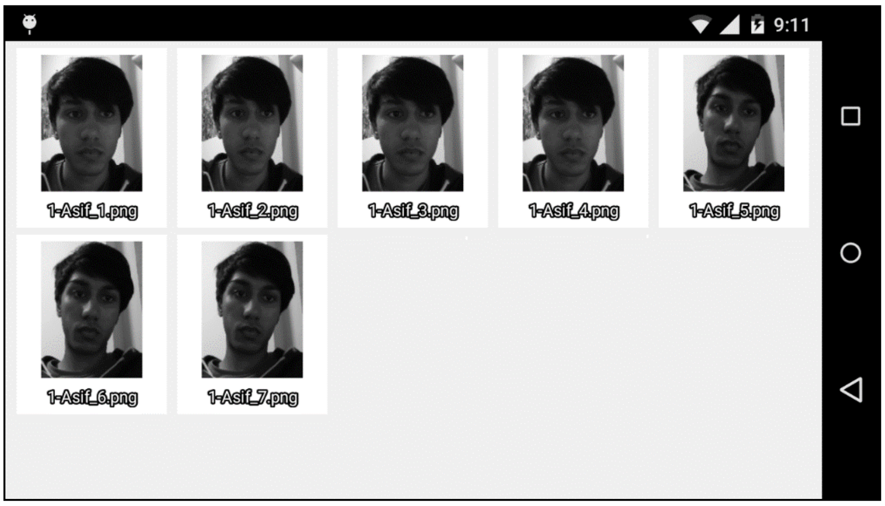


Figure 3 – Non-aligned image database

The four images from 1-Asif_1 to 1-Asif_4 are comparatively good images for recognition as they have minimum rotation and most of the image is covered with the face. Conversely the three images from 1-Asif_5 to 1-Asif_7 are not very good samples as the head is tilted and

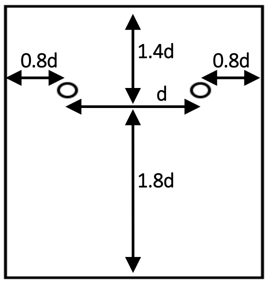


Figure 4 – Ratio of Ideal Face Dimensions

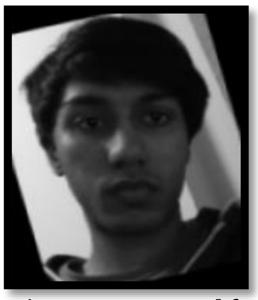


Figure 5 – Rotated face before cropping

about a third of the image is dominated by features not related to the face. To solve this, the supervisor provided some feedback on how to use eyes to calculate the coordinates of a face. The key parameter that is needed to calculate the dimensions of the face is the distance (d) between the eyes. In figure 4 above, the ratio of the dimensions are outlined and in figure 5 a rotated face is shown before it is cropped according to the figure 4 dimensions. To achieve the rotation, the android program needs to carry out some trignometry which has been shown in figure 6 and an enlarged version in figure 7.

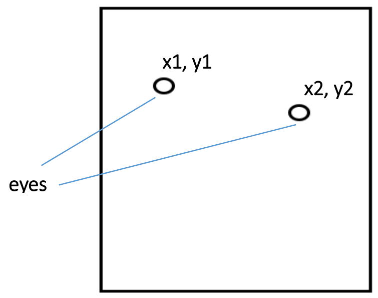


Figure 6 – Example tilted face

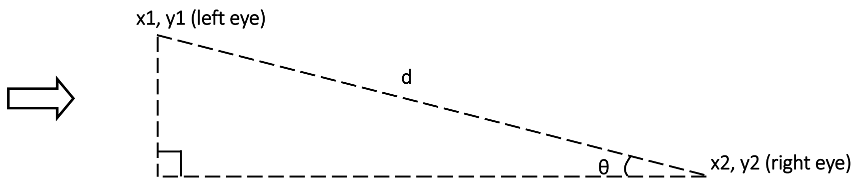


Figure 7 – Enlarged version which shows the eye coordniates.

The right eye (x2, y2) is below the left eye (x1, y1) in figure 6 which resembles a face tilted clockwise. To fix this, the angle by which the eyes are tilted has to be calculated and this angle is denoted in figure 7 by the symbol ‘θ’. The angle θ can be calculated by the following -

$$\theta = tan^{-1}\frac{y2 - y1}{x2 - x1}$$

Following this, the OpenCV library has a image rotating function called *warpAffline* which is used to rotate the image by θ. The rotated image is then cropped according to the dimensions outlined in figure 4. In this case ‘d’ is calculated with the following –

$$d = \sqrt[2]{(x2 - x1)^2 - (y2 - y1)^2}$$

After completing the non real-time alignment, the next part was to move onto a real-time system. Instead of using Viola-Jones haar cascade classifier to detect the face, it was used to detect the two eyes seperately. From the two eyes, the face size and angle of tilt was calculated using the automatic alignment method discussed above. This meant that the face could not only be detected but its degree of rotation followed as well which is shown in figure 8. In figure 9 it can be seen that the alignment algorithm was able to follow high amounts of head rotation as well.

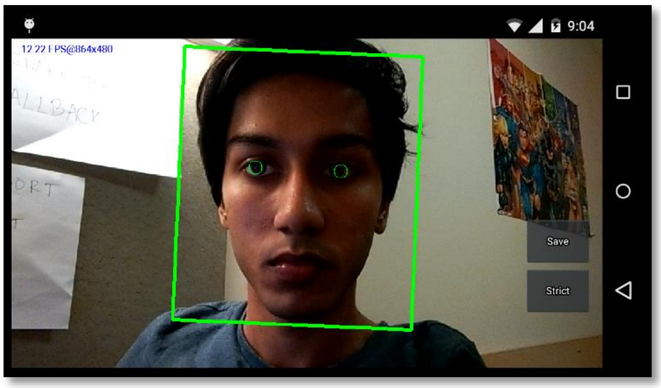


Figure 8 – Real-time alignment example 1

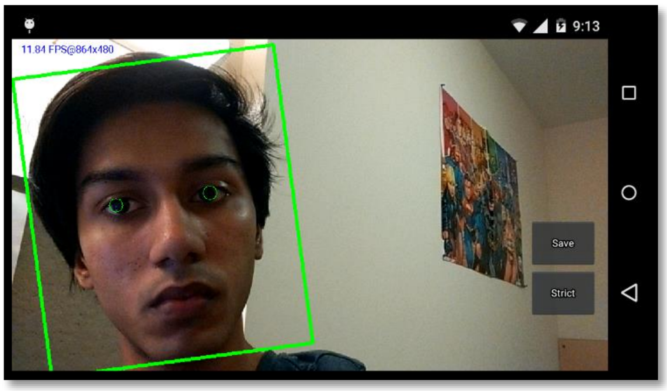


Figure 9 – Real-time alignment example 2

While the application worked well in normal circumstances, it still faced some problems in situations where the eyes were made hard to detect. This included wearing spectacles that reflected a lot of light and having long hair covering the forehead. Another layer of detection will need to be added to make the application more robust using a separate feature such as the nose. The second layer will help in cases where the eyes are not being detected or one eye is being detected. The save button shown in figure 8 and 9 saves the last face detected in an image database (shown in figure 10) for later viewing. The strict button is an experimental function which allows the user to alternate between strict and predictive mode. In predictive mode the program tries to guess the eye position from previous detected frames while in strict mode it always tries to find both eyes in the frame.

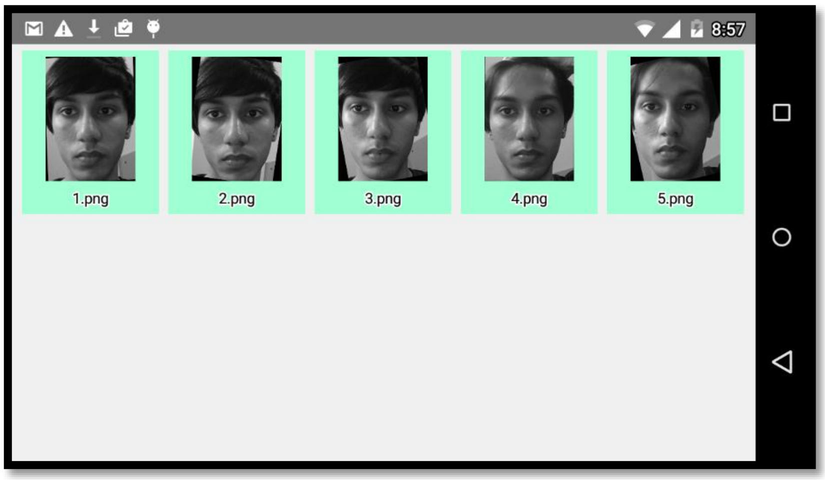


Figure 10 – Automatically-aligned image database

6. Conclusion

Automatic alignment has successfully been implemented which was one of the initial aims of this project. The next step is to use the aligned database to train the face recognition algorithm (LBP). It can be concluded that the project is advancing at the required pace and will likely be able to deliver the project aims in the allocated time.

7. References

[1] Photo "ORL image database" from *Cambridge University Computer Laboratory Database of faces*, http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

[2] Documentation"Android Application fundamentals" from *Google API Guide*, https://developer.android.com/guide/components/fundamentals.html

[3] Documentation"OpenCV tutorials" from *OpenCV 3.0.0-dev documentation*, http://docs.opencv.org/3.0-beta/doc/tutorials/tutorials.html

[4] **Viola, Jones** "Pages 1,3,11 ", *Robust Real-Time Object Detection*, Vancouver, Canada , July, 13, 2001.

[5] **Haar A**. *Zur Theorie der orthogonalen Funktionensysteme*, Mathematische Annalen, **69**, pp. 331–371

[6] ***Freund; Schapire***. *"A Short Introduction to Boosting"* *:* Introduction to AdaBoost, *1999*

[7] ***M. Turk and A. Pentland*** *(1991).* "Face recognition using eigenfaces" *Proc. IEEE Conference on Computer Vision and Pattern Recognition. pp. 586–591.*

[8] ***Mahvish Nasir*** . "Principal Component Analysis" *fewtutorials/bravesite EmguCV* https://www.youtube.com/watch?v=SaEmG4wcFfg

[9] **Ahonen, T., Hadid, A. and Pietikäinen, M.** (2006), Face Description with Local Binary Patterns: Application to Face Recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence* 28(12):2037-2041.

8. Appendix

8.1 Health and Risk Assessment

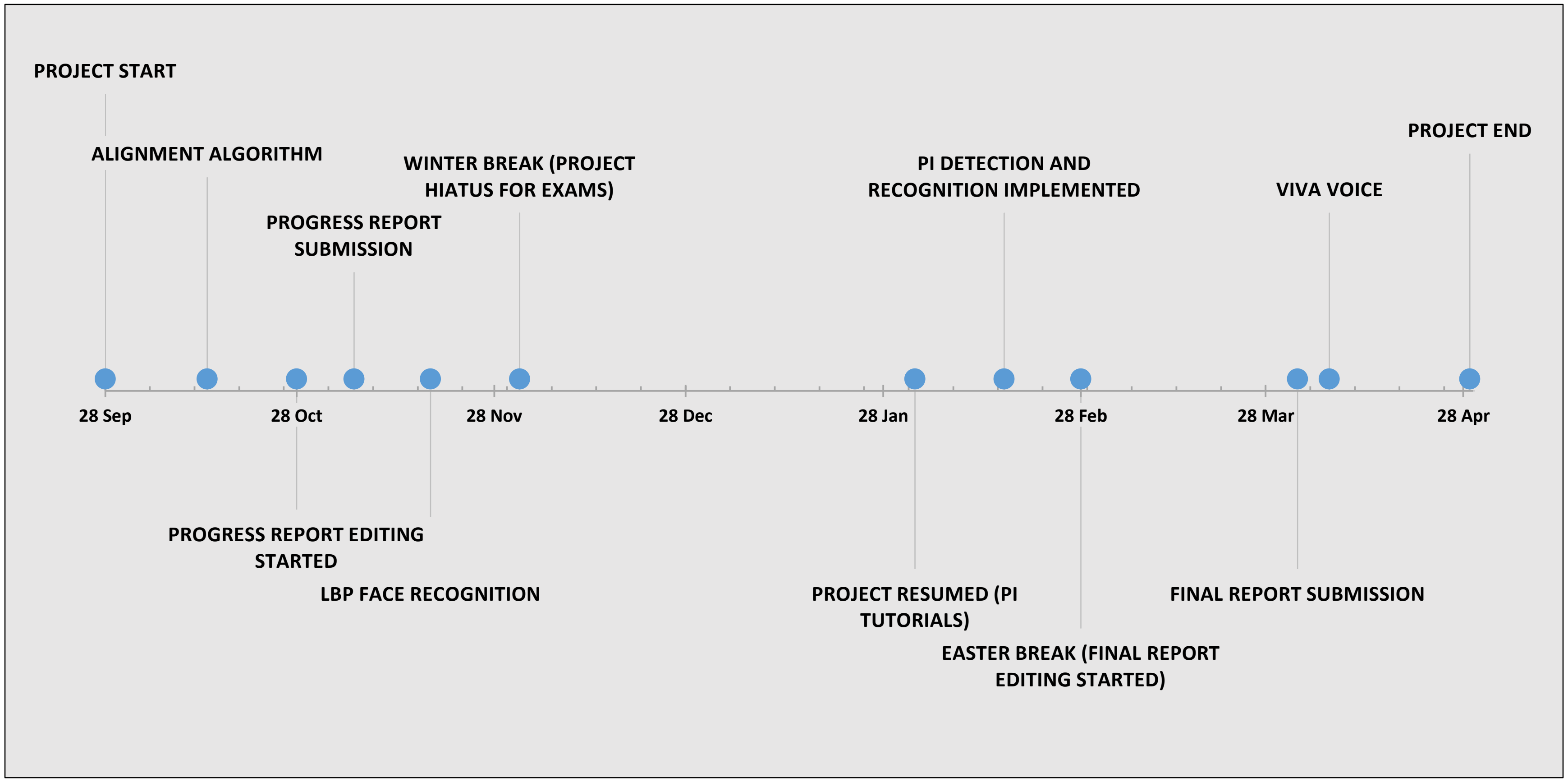
LOCATION/ACTIVITY: Development, Barnes Wallis Computer Cluster

WORK ACTIVITY/ WORKPLACE (WHAT PART OF THE ACTIVITY POSES RISK OF INJURY OR ILLNESS)	HAZARD (S) (SOMETHING THAT COULD CAUSE HARM, ILLNESS OR INJURY)	LIKELY CONSEQUENCES (WHAT WOULD BE THE RESULT OF THE HAZARD)	WHO OR WHAT IS AT RISK (INCLUDE NUMBERS AND GROUPS)	EXISTING CONTROL MEASURES IN USE (WHAT PROTECTS PEOPLE FROM THESE HAZARDS)	WITH EXISTING CONTROLS				MEASURE REQUIRED TO PREVENT OR REDUCE RISK (WHAT NEEDS TO BE DONE TO MAKE THE ACTIVITY AS SAFE AS POSSIBLE)	PERSON RESPONSIBLE FOR ACTIONS AND AGREED TIMESCALES TO ACHIEVE THEM	WITH NEW CONTROLS			
					SEVERITY		RISK RATING	RISK ACCEPTABLE			SEVERITY		RISK RATING	RISK ACCEPTABLE
Charging and power cables for laptop, android phone. Raspberry Pi	Trip on cables.	Slight Minor injury / illness – immediate 1st Aid only / slight loss	Mohammad Asif Other students using computer cluster.	Zip-ties to bundle up extra cable length.	2	2	3	No	Keep cables away from walking zone.	Mohammad Asif must remain vigilant at all times.	2	1	2	yes
Carrying heavy components such as laptop up stairs	Slip down stairs due to the heavy components.	Moderate Injury / illness of 3 days or more absence (reportable category) / Moderate	Mohammad Asif	Do not carry heavy things upstairs.	3	2	3	No	Use ground floor computer cluster.	Mohammad Asif, commencing the month of November 2015	2	1	2	yes
Screwing the Pi to the screen interface with screwdriver and screws.	Prick finger on sharp edges.	Negligible No injury or trivial injury / illness / loss	Mohammad Asif	Handle with care.	1	1	1	Yes						

RISK ASSESSOR	**NAME: Stephen Duffy School Safety Advisor**	**SIGNED:**	**Revised 02/11/2015**	**THIS RISK ASSESSMENT WILL BE SUBJECT TO A REVIEW NO LATER THAN: (12 months)**
RISK ASSESSOR	**NAME: Mike Ash Technical Services Manager**	**SIGNED:**	**DATE:**	

8.2 Project Plan

The key dates for the project such as holidays, exams and submission dates are listed in the project time-line below along with tentative dates of the major aims and when it is expected to finish them. Since the project is in its early stages, a lot of the variables and factors that will come into play is still unknown so the project timeline provides a brief look currently. A more comprehensive timeline will be developed when all the tasks have been decided for the final report.


PROJECT START
ALIGNMENT ALGORITHM
PROGRESS REPORT SUBMISSION
WINTER BREAK (PROJECT HIATUS FOR EXAMS)
PI DETECTION AND RECOGNITION IMPLEMENTED
VIVA VOICE
PROJECT END
28 Sep
28 Oct
28 Nov
28 Dec
28 Jan
28 Feb
28 Mar
28 Apr
PROGRESS REPORT EDITING STARTED
LBP FACE RECOGNITION
PROJECT RESUMED (PI TUTORIALS)
EASTER BREAK (FINAL REPORT EDITING STARTED)
FINAL REPORT SUBMISSION