Chapter 5

Report on present investigation

5.1 Proposed System

The proposed system involves automated cashless payment system using facial recognition. This system will work as an API between the payment system and any other application in which face of the user can be used for authentication. Using this system, the user can directly pay fares for the cab rides or hotel stay etc. The initial step of the system is the registration process. During the time of registration, the system will collect the facial information of the user by asking user to upload an video of user's face. The system will then extract the frames from that video which will be used stored in the database for further use. Every user has a specific user directory, in which all the frames collected from the video will be saved. This dataset of images will be used to train our facial recognition model powered by OPENCV, Local Binary Patterns Histograms Face Recognizer (LBPH) algorithm. After the complete registration process, the user can login into the system by using facial recognition, and selects the destination. The system will then allocate a nearby cab to the user, and ensure the pickup for that user. Based on the pickup point and the destination, an invoice will be generated and the amount will be shown to the user. After the completion of the ride, the user will be redirected to the payment engine, where user will be prompted to enter his special pin to complete the transaction.

5.1.1 System Architecture

A system architecture or systems architecture is the conceptual model that defines the structure, behaviour, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviours of the system.

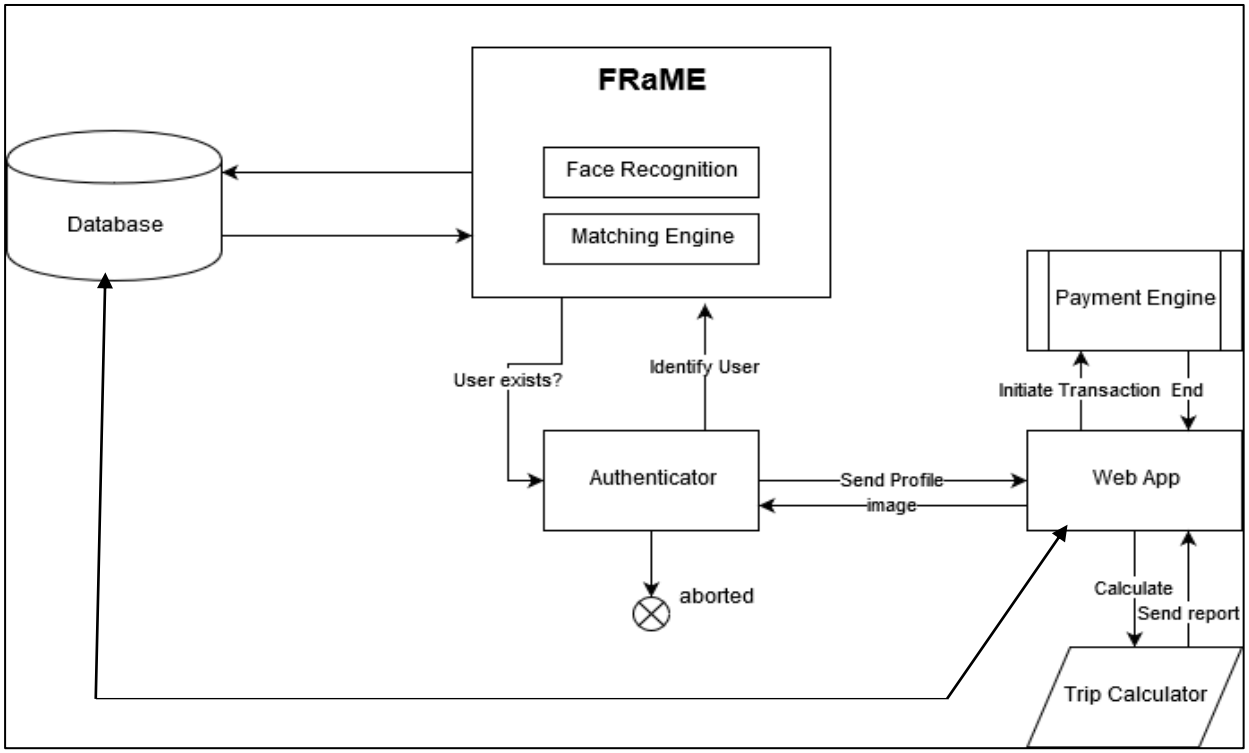

Fig 5.1 : System Architecture

Figure 5.1 shows system is a two tier application, First tier is the webapp and second tier is the facial recognition and matching engine. The web app manages the trip calculator and the payment engine. Authentication for the webapp is done by the face recognition and matching engine which is connected to the database. The dataset used by facial recognition also uses the same database in a specific user directory structure. The facial recognition and matching engine consists of two part which are face recognition and matching engine. The Authenticator is one of the most important part of the system, it is the bridge between the webapp and the frame engine, it identifies whether an particular user already exists or not.

5.2 Implementation

There are three easy steps to computer coding facial recognition, which are similar to the steps that our brains use for recognizing faces. These steps are:

1) Data Gathering: Gather face data (face images in this case) of the persons user want to identify.
2) Train the Recognizer: Feed that face data and respective names of each face to the recognizer so that it can learn.
3) Recognition: Feed new faces of that people and see if the face recognizer user just trained recognizes them.
4) OpenCV has three built-in face recognizers and thanks to its clean coding, user can use any of them just by changing a single line of code. Here are the names of those face recognizers and their OpenCV calls:

- EigenFaces – cv2.face.createEigenFaceRecognizer()
- FisherFaces – cv2.face.createFisherFaceRecognizer()
- Local Binary Patterns Histograms (LBPH) – cv2.face.createLBPHFaceRecognizer()

5.2.1 Eigenfaces face recognizer

This algorithm considers the fact that not all parts of a face are equally important or useful for face recognition. Indeed, when user look at someone, user recognize that person by his distinct features, like the eyes, nose, cheeks or forehead; and how they vary respect to each other. In that sense, user are focusing on the areas of maximum change. For example, from the eyes to the nose there is a significant change, and same applies from the nose to the mouth. When user look at multiple faces, user compare them by looking at these areas, because by catching the maximum variation among faces, they help user differentiate one face from the other. In this way, is how EigenFaces recognizer works. It looks at all the training images of all the people as a whole and tries to extract the components which are relevant and useful and discards the rest. These important features are called principal components.

Principal components, variance, areas of high change and useful features indistinctly as they all mean the same.

EigenFaces recognizer trains itself by extracting principal components, but it also keeps a record of which ones belong to which person. Thus, whenever user introduce a new image to the algorithm, it repeats the same process as follows:

1. Extract the principal components from the new picture.
2. Compare those features with the list of elements stored during training.
3. Find the ones with the best match.
4. Return the 'person' label associated with that best match component.

In simple words, it's a game of matching. However, one thing to note in above image is that EigenFaces algorithm also considers illumination as an important feature. In consequence, lights and shadows are picked up by EigenFaces, which classifies them as representing a face. Face recognition picks up on human things, dominated by shapes and shadows: two eyes, a nose, a mouth. Fig 5.2 Shows the variance of shades extracted from various faces in detail.

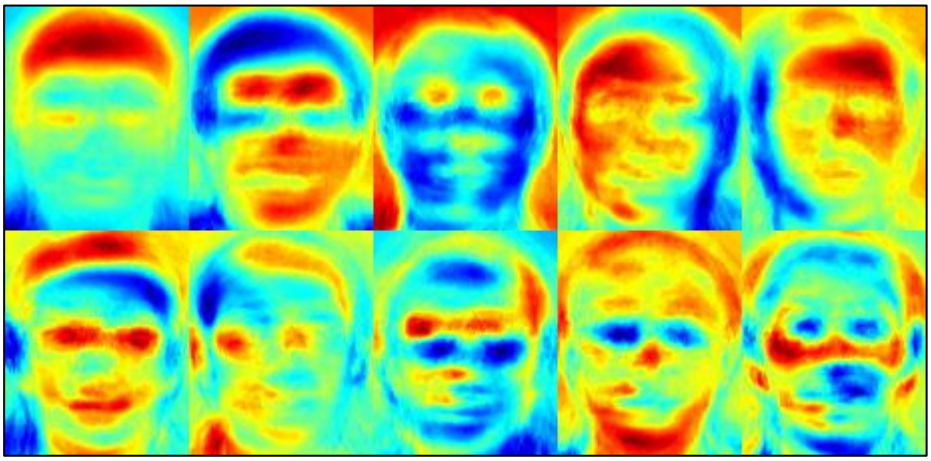

Fig 5.2 : Variance extracted from a list of faces.

5.2.2 Fisherface face recognizer

This algorithm is an improved version of the last one. As system just saw, EigenFaces looks at all the training faces of all the people at once and finds principal components from all of them combined. It doesn't focus on the features that discriminate one individual from another.

Instead, it concentrates on the ones that represent all the faces of all the people in the training data, as a whole. Since EigenFaces also finds illumination as a useful component, it will find this variation very relevant for face recognition and may discard the features of the other people's faces, considering them less useful. In the end, the variance that EigenFaces has extracted represents just one individual's facial features.

System can do it by tunning EigenFaces so that it extracts useful features from the faces of each person separately instead of extracting them from all the faces combined. In this way, even if one person has high illumination changes, it will not affect the other people's features extraction process. Precisely, FisherFaces face recognizer algorithm extracts principal components that differentiate one person from the others. In that sense, an individual's components do not dominate (become more useful) over the others.

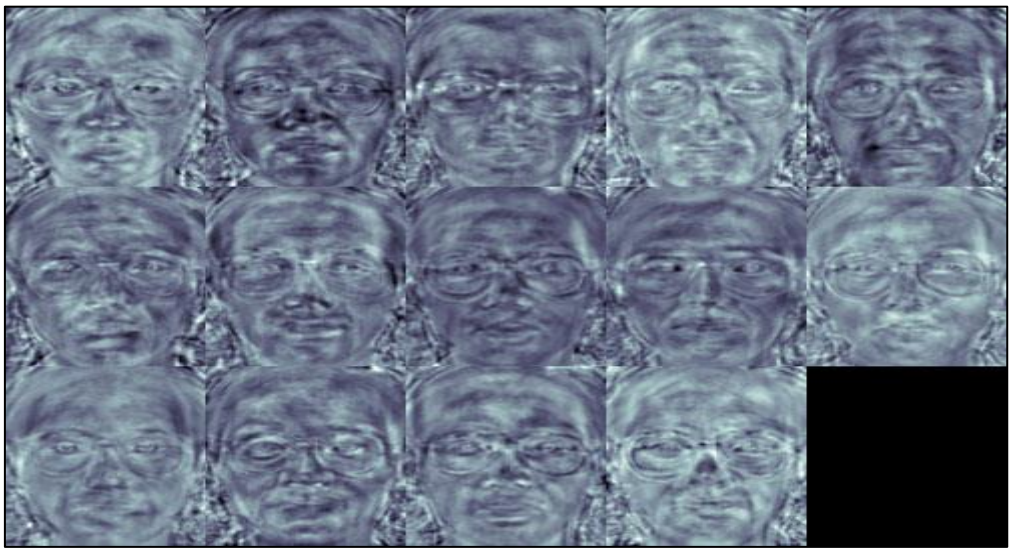

Fig 5.3: principal components using Fisherface algorithm.

Figure 5.3 shows that FisherFaces only prevents features of one person from becoming dominant, but it still considers illumination changes as a useful feature. System know that light variation is not a useful feature to extract as it is not part of the actual face.

5.2.3 Local binary patterns histograms (LBPH) Face Recognizer

The Eigen faces and Fisherface are both affected by light and so cannot guarantee perfect light conditions. LBPH face recognizer is an improvement to overcome this drawback.. Take a 3×3 window and move it across one image. At each move (each local part of the picture), compare the pixel at the centre, with its surrounding pixels. Denote the neighbours with intensity value less than or equal to the centre pixel by 1 and the rest by 0.After user read these 0/1 values under the 3×3 window in a clockwise order, user will have a binary pattern like 11100011 that is local to a particular area of the picture. When user finish doing this on the whole image, user will have a list of local binary patterns.

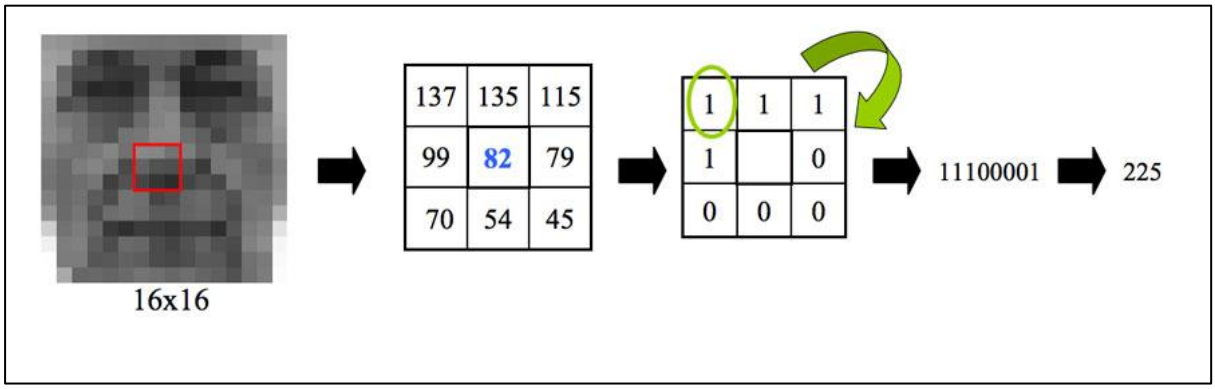

Fig 5.4: LBPH Face recognizer Process

Now, after user get a list of local binary patterns, user convert each one into a decimal number using binary to decimal conversion as shown in figure 5.4 and then user make a histogram of all of those decimal values. Figure 5.5 shows a sample histogram of LBPH.

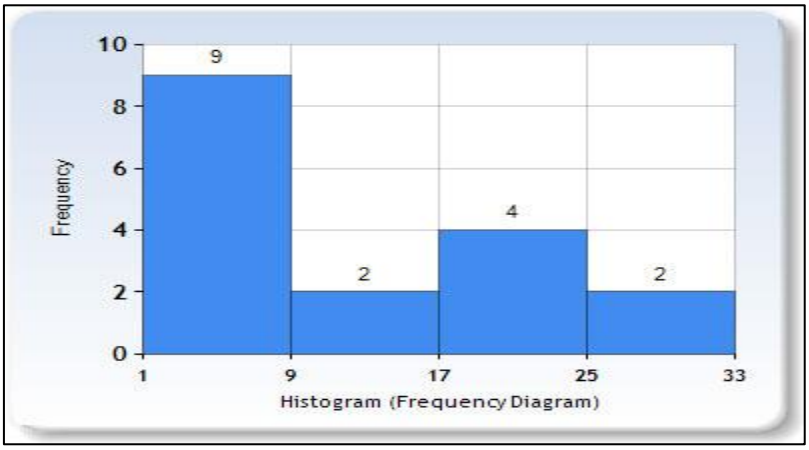

Fig 5.5: LBPH Histogram

In the end, user will have one histogram for each face in the training data set. That means that if there were 100 images in the training data set then LBPH will extract 100 histograms after training and store them for later recognition. Remember, the algorithm also keeps track of which histogram belongs to which person.
Later during recognition, the process is as follows:

1. Feed a new image to the recognizer for face recognition.
2. The recognizer generates a histogram for that new picture.
3. It then compares that histogram with the histograms it already has.
4. Finally, it finds the best match and returns the person label associated with that best match.

Below is a group of faces and their respective local binary patterns images. System see that the LBP faces are not affected by changes in light conditions:

5.2.4 Required Modules

Import the following modules:

- **cv2:** This is the OpenCV module for Python used for face detection and face recognition.
- **os:** System will use this Python module to read our training directories and file names.
- numpy: This module converts Python lists to numpy arrays as OpenCV face recognizer needs them for the face recognition process.

5.2.5 Prepare training data

The premise here is simple: The more images used in training, the better. Being thorough with this principle is important because it is the only way for training a face recognizer so it can learn the different ‘faces’ of the same person; for example: with glasses, without glasses, laughing, sad, happy, crying, with a beard, without a beard, etc. So, our training data consists of total two people with 12 images of each one. All training data is inside the folder: training-data.

This folder contains one subfolder for every individual, named with the format: sLabel (e.g. s1, s2) where the label is the integer assigned to that person. For example, the subfolder called s1 means that it contains images for person 1.

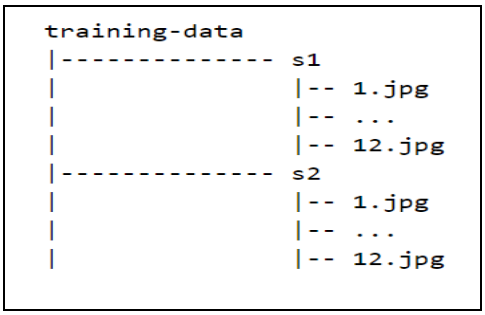

Fig 5.6 : Directory structure tree for training data

Figure 5.6 shows the folder test-data contains images that system will use to test our face recognition program after system have trained it successfully. Considering that the OpenCV face recognizer only accepts labels as integers, system need to define a mapping between integer tags and the person's actual name.

5.3 Data Preparation for Face Recognition

To know which face belongs to which person, OpenCV face recognizer accepts information in a particular format. That means that if there were 100 images in the training data set then LBPH will extract 100 histograms after training and store them for later recognition. It doesn't focus on the features that discriminate one individual from another. Instead, it concentrates on the ones that represent all the faces of all the people in the training data, as a whole. Since EigenFaces also finds illumination as a useful component, it will find this variation very relevant for face recognition and may discard the features of the other people's faces, considering them less useful. Remember that all the sub folders containing images of a person following the format "sLabel" where Label is an integer representing each person.

In fact, it receives two vectors:

- One is the faces of all the people.

- The second is the integer labels for each face.

For example, if system had two individuals and two images for each one. Then the system will make the file structures as shown in the figure 5.7

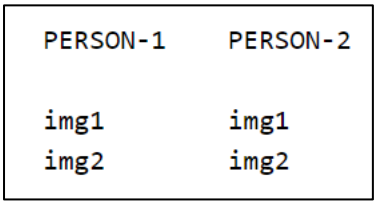

Fig 5.7: Data preparations for face recognition

After making directories the system will label the images according to the person .The data preparation step will produce following face and label vectors as shown in the figure 5.8

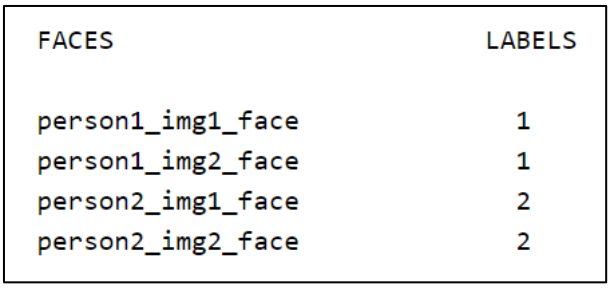

Fig 5.8 : Data preparations for face recognition

In detail, system can further divide this step into the following sub-steps:

1. Read all the sub folders names provided in the folder training-data. In this tutorial; system have folder names:s1, s2.
2. Extract label number. Remember that all the sub folders containing images of a person following the format:sLabel where Label is an integer representing each person. So for example, folder name: s1 means that the person has label 1, s2 means the person's label is 2, and so on. System will assign the integer extracted in this step to every face detected in the next one.
3. Read all the images of the person, and apply face detection to each one.

4. Add each face to face vectors with the corresponding person label (extracted in above step).