MULTIMODAL PERSON IDENTIFICATION SYSTEM

A PROJECT REPORT

Submitted in partial fulfillment of the
requirement for the award of the
Degree of

BACHELOR OF TECHNOLOGY
in
ELECTRONICS AND COMMUNICATION ENGINEERING

By

SHRADDHA AGRAWAL - 13BEC1138

PRITHVISH V N – 13BEC1100

Under the Guidance of

Prof. JOHN SAHAYA RANI ALEX

SCHOOL OF ELECTRONICS ENGINEERING
VIT University
CHENNAI. (TN) 600127

(MAY 2017)

MULTIMODAL PERSON IDENTIFICATION SYSTEM

A PROJECT REPORT

Submitted in partial fulfillment of the
requirement for the award of the
Degree of

BACHELOR OF TECHNOLOGY
in
ELECTRONICS AND COMMUNICATION ENGINEERING

By

Shraddha Agrawal - 13BEC1138
Prithvish V N – 13BEC1100

Under the Guidance of

Prof. John Sahaya Rani Alex

SCHOOL OF ELECTRONICS ENGINEERING
VIT University
CHENNAI. (TN) 600127

(MAY 2017)

CERTIFICATE

This is to certify that the Project work titled "***Multimodal Person Identification System***" that is being submitted by "***Shraddha Agrawal" and "Prithvish V N"*** is in partial fulfillment of the requirements for the award of **Bachelor of Technology**, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Prof. John Sahaya Rani Alex
Guide

The thesis is satisfactory / unsatisfactory

Internal Examiner **External Examiner**

Program Chair Dean (SENSE)

ACKNOWLEDGEMENT

We sincerely thank our professor Prof. John Sahaya Rani Alex, SENSE Department, VIT University, Chennai campus for their timely guidance, keen interest and encouragement which have contributed immensely to the evolution of ideas on the project. We acknowledge with thanks the kind of patronage, loving inspiration and valuable guidance we received from her working on the project.

We also thank Dr P. Gunasekaran (Pro VC), Dr. Kanchana Bhaskaran (Dean of Acadmenics), Dr. Prabharan SRS (Dean, SENSE department), Dr. Manoj Kumar (Program Chair, SENSE department) and all my other faculties for their constant support and encouragement throughout this journey.

Shraddha Agrawal -13BEC1138
Prithvish V N – 13BEC1100

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BBB	:	Beagle Bone Black
ARM	:	Advanced RISC Machine
USB	:	Universal Serial Bus
MFCC	:	Mel Frequency Cepstral Coefficient
SVM	:	Support Vector Machine
DCT	:	Discrete Cosine Transform
HMM	:	Hidden Markov Model
LBPH	:	Local Binary Pattern Histogram
ORB	:	Oriented FAST and Rotated BRIEF
BRIEF	:	Binary Robust Independent Element Features
OpenCV	:	Open Source Computer Vision
ALSA	:	Advanced Linux Sound Architecture
API	:	Application Programming Interface
UART	:	Universal Asynchronous Receiver/ Transmitter
SSH	:	Secure Shell
OS	:	Operating System
PIL	:	Python Imaging Library
UTF-8	:	Unicode Transformation Format - 8
BSD	:	Berkeley Software Distribution
GNU	;	GNU’s Not Unix
GPL	:	General Public License
LGPL	:	Lesser General Public License

LIST OF SYMBOLS

E	:	difference between the original and the moved window.
u	:	window's displacement in the x direction
v	:	window's displacement in the y direction
w(x, y)	:	window at position (x, y).
I	:	intensity of the image at a position (x, y)
I(x+u, y+v)	:	intensity of the moved window
I(x, y)	:	intensity of the original
M	:	summed matrix
R	:	score for Harris corner detector
f	:	frequency of the speech signal
m	:	value of frequency in mel scale
s(n)	:	time domain speech signal
$s_i(n)$	:	samples of time domain signal
h(n)	:	N samples long analysis window
K	:	length of the DFT
N	:	number of samples
$P_i(k)$	:	periodogram based power spectral estimate
d_t	:	delta coefficient for frame t
c_t	:	static MFCC coefficient for frame t
$H_m(k)$	:	mel filterbank

ABSTRACT

In this project, we present an audio-visual feature-level fusion for person identification system using a combination of acoustic features, fingerprint images and 2D face images. Person identification is of paramount importance in security, surveillance, human computer interfaces, and smart spaces. There are numerous instances where a single feature, however, sometimes fails to be exact enough for identification. Also, another disadvantage of using only one feature is that the chosen feature is not always readable. Thus, a multimodal approach using three different modalities - face, voice and fingerprint - takes care of all such instances and achieves greater accuracy than single feature systems. The speaker verification is done by extracting Mel Frequency Cepstral Coefficients followed by Support Vector Machine. The facial recognition is done using Local Binary Pattern histogram face recognizer after performing Harr cascade for face detection. The fingerprint verification is done using binarization on a grayscale image, followed by skeletization and then the minutiae points are found using Harris corner detector. All the three above approaches are implemented on hardware using Beagle Bone Black along with audio microphone and USB cameras as sensory modules. The various sensory modalities, speech, fingerprint and faces, are processed both individually and jointly and it has been observed that the performance of the multimodal approach results in improved performance in the identification of the participants. Our system achieves around 90% recognition and verification rates on natural real-time input with 10 registered clients.

CHAPTER - 1
INTRODUCTION

1.1 OBJECTIVE

The objective of the project is to design a system that uses a multimodal approach for person identification. In other words, this approach involves a fusion of more than two human traits or biometrics for improved performance and excludes all the discrepancies of a conventional single-feature person identification system. Persistent efforts are given so as to minimize the computational cost of the system by reducing the dimensionality of data and achieve high processing speed. Also, focus will be given to make this whole system cost effective and portable.

1.2 BACKGROUND

In today's electronically wired society, there are an increasing number of scenarios which require an individual, as a user, to be verified by an electronic device. Traditionally, a user can be verified based on whether he/she is in possession of a certain token such as an ID card and/or whether he/she is in possession of a specific knowledge which nobody else is expected to know such as a password. These approaches have a number of significant drawbacks. Tokens may be lost, stolen, forgotten, forged or misplaced. Passwords may be forgotten or compromised. All these are unable to differentiate between an authorized user and an imposter who fraudulently acquires the token or knowledge of the authorized user, and does not provide sufficient security in many critical applications involving access control and financial transactions [1].

Biometrics refers to the automatic identification of a person based on his/her physiological or behavioral characteristics. It relies on something he/she has or he/she does to make a personal identification. It is inherently more reliable and has a higher discrimination capability than the

previous methods because the physiological or behavioral characteristics are unique to each user.

Currently, nine different biometric indicators are either widely used or under intensive evaluation, including face, facial thermogram, voice-print, signature, fingerprint, hand geometry, hand vein, lip movement, iris, retinal pattern. It is the requirements of an application domain which determine the choice of a specific biometric [2].

All these biometric indicators have their own advantages and disadvantages in terms of the accuracy, user acceptance, and applicability. A single feature, however, sometimes fails to be exact enough for identification. Consider identical twins, for example. Their faces alone may not distinguish them. Another disadvantage of using only one feature is that the chosen feature is not always readable. For example, some five percent of people have fingerprints that cannot be recorded because they are obscured by a cut or a scar or are too fine to show up well in a photograph [3,4] . In order to enable a biometric system to operate effectively in different applications and environments, a multimodal biometric system which makes a person identification based on multiple physiological or behavioral characteristics is preferred [5].

1.3 LITERATURE REVIEW

PAPER/ ARTICLE	APPROACH	PROS AND CONS
Multimodal Person Identification In A Smart Room J.Luque, R.Morros, J.Anguita, M.Farrus, D.Macho, F.Marqués, C.Martínez, V.Vilaplana, J. Hernando	• Mel Frequency Cepstral Coefficients (MFCC) performed but replacing the final Discrete Cosine Transform by the logarithmic filter-bank energies of the speech frame • Eye coordinates are used to determine the face pose for each image. Only frontal faces are used for identification • PCA based approach has been used • The z-score normalized biometric is calculated • Matcher Weighting with equalized scores	• logarithmic filter-bank are used,giving better or same results • normalization process is used to transform different scores into a comparable range • pre-emphasis of speech signal was lacking • using the corner cameras, face images can't be ensured to be all frontal. Mixing frontal and non-frontal faces present complications • variability of the users' appearance is assumed to be low

IV Jornadas en Tecnologia del Habla		• lighting variations are not accounted for
BioID: A Multimodal Biometric Identification System *Robert W. Frischholz Ulrich Dieckmann* Dialog Communication Systems AG	• video sequence : face and lip movement while speaking a word. • audio signal • one-dimensional lip movement feature vector : calculates a vector field representing the local movement of each image part to the next image in the video sequence • calculates the cepstral coefficients, which form the audio feature vector • synergetic computer to classify the optical features, and a vector quantifier to classify the audio feature.	• uses a dynamic feature, lip movement • synergetic phenomena of theoretical physics • the vector quantifier combines matrices for each voice pattern into one matrix (or codebook) that displays the reference voice pattern • perform only face extraction
Multimodal Person Recognition using Unconstrained Audio and Video *Tanzeem Choudhury, Brian Clarkson, Tony Jebara, Alex Pentland*	• identifies largest skin color blob and roughly look for facial features (eyes, nose, mouth) • refines previous estimates by researching for eyes, nose, mouth within a small area around the previous estimate • quantifies images using 35 eigen coefficients • performs MFCC and estimate HMM for each person • considers samples of noisy environments and estimates HMM on it • fusion is done using Baye's net	• it is very important to select reliable video frame and audio clip to use for recognition • uses histogram fitting to normalize illumination • in essence, face recognition is fast • performs weak HPF pre-emphasis to remove DC offset and boost higher frequencies • recognition rate for clean speech is low
A Multimodal Biometric System Using Fingerprint, Face and Speech *Anil Jain, Lin Hong, Yatin Kulkarni*	• performs both minutiae extraction and matching • linear prediction coefficients of 10th order to make a verification • eigenvalue based face recognition	• calculates similarity function for all individual biometrics • fingerprint ensures high accuracy • no pre-emphasis performed on audio signal
Bimodal Person	• Fisher linear discriminant analysis was used for face recognition	• Single mode PCA and Mulimode PCA were compared and

Identification System *C. N. Dinakardas, S. Perumal Sankar, and Nisha George*	• Principal Components Analysis reduces data dimensionality by performing a covariance analysis between factors. • Finger print projection PCA was performed • Matching and Future Fusion was performed to calculate the final result	multimode had better results than single mode PCA

Table 1: Literature review

1.4 APPROACH

The project designs a multimodal biometric system which integrates face, fingerprint and speech to make a personal identification. The choice of these three specific biometrics is based on the fact that they have been used routinely in law enforcement community. Most of the commercial biometric systems currently rely on either fingerprint, face or voice. Further, these biometric indicators complement one another in their advantages and strengths. For example, consider a system which uses these three biometrics for user authentication. If the operating environment is noisy then voice is not a suitable biometric indicator. If the background is cluttered, then the face location algorithm, which is necessary for face recognition, may not work very well. If a user cannot provide good fingerprint images (e.g. due to dry fingers, cuts, etc.), then face and voice may be better biometrics. Our system is targeted for verification applications to authenticate the identity claimed by a user in a multiuser account authentication.

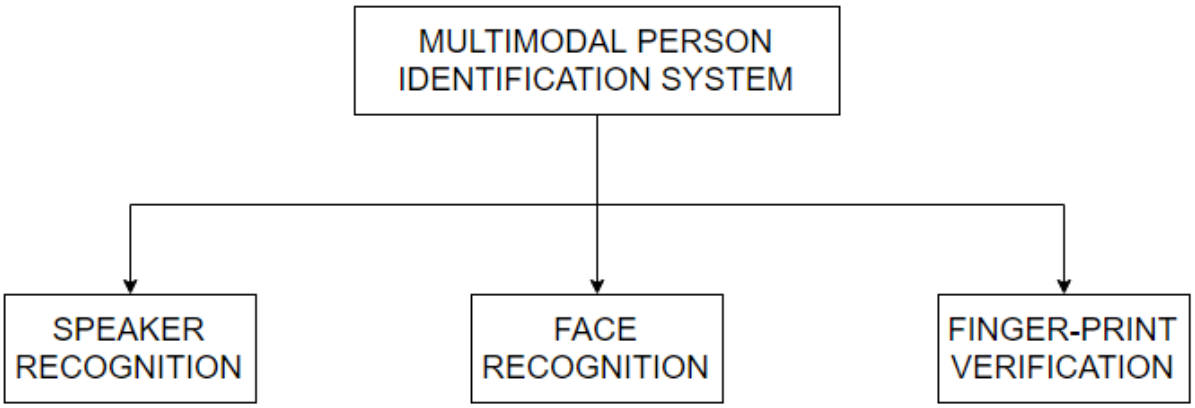


Fig.1 Biometric indicators used in the system

CHAPTER - 2
PROJECT DESCRIPTION

2.1 METHODOLOGY

The project consists of two phases: training and testing. Initially, a database containing template records for face, fingerprint and speech for all the intended users is created. For the offline training, the system then extracts the required feature for each of these biometrics. Further, the high dimensionality of the data is reduced to a lower dimensional data representation and stored with proper labels.

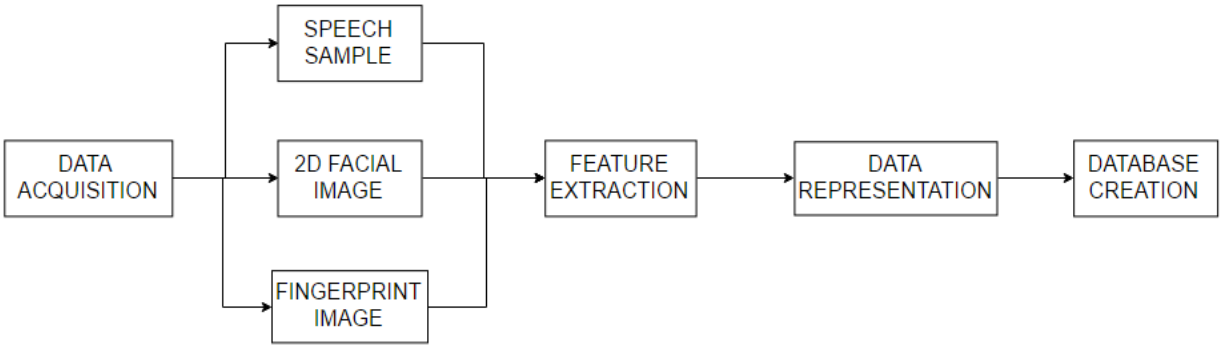


Fig.2 Block diagram for offline database creation

In the testing stage, a real time input for the speech, face and fingerprint is acquired using the same system, required pre-processing is performed and features are extracted. Using the template database created, scores are obtained for speaker recognition, face recognition and fingerprint verification. Further these scores are normalized and by means of a decision fusion algorithm to authenticate a user.

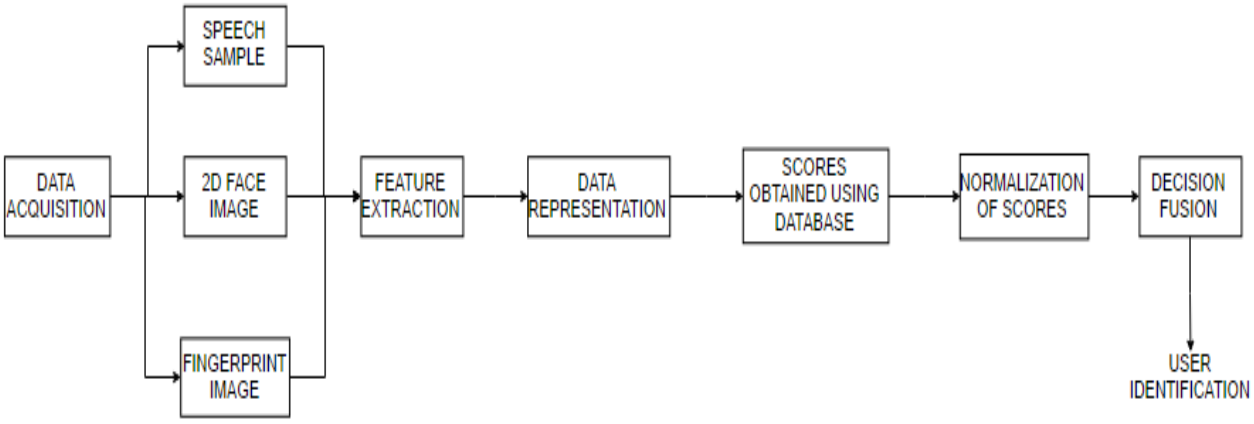


Fig. 3 Block diagram for online testing

2.2 DESIGN APPROACH

The verification process essentially consists of four stages :

1) Fingerprint verification
2) Face recognition
3) Speaker verification
4) Decision fusion

2.2.1 FACE RECOGNITION

Face recognition is responsible for matching the input face against the face template to obtain the face matching score.

In person identification, face recognition refers to static, controlled full frontal portrait recognition. There are two major tasks involved for this: i) face detection and ii) face recognition. Face detection finds whether there is a face in the input image and if so, the location of the face in the image. After this, some normalization should be performed on the image, e.g. applying some grayscaling, histogram equalization, some alignment based on eye and mouth detection, etc. finally

the data needs to be passed to a face recognizer interface. It finds the similarity between the located face and the stored templates to determine the identity of the user.

The face recognition method is divided in two stages: i) training stage and ii) operational stage. In the training stage, a set of orthonormal images that best describe the distribution of the training facial images in a lower dimensionality is computed. Then, the training facial images are projected onto this set to generate the representations of the facial images. In operational stage, a detected facial image is projected onto the same set and the similarity between the input facial image and the template is thus computed [6].

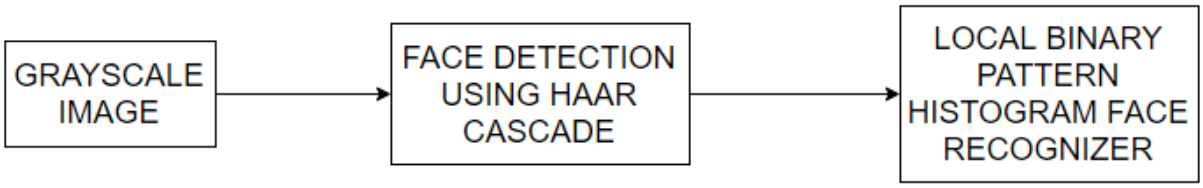


Fig. 4 Flow for face recognition

2.2.1.1 FACE DETECTION

The first step is to detect the face in each image. Once, we get the region of interest containing the face in the image, we will use it for training the recognizer. For the purpose of face detection, we will use the Haar Cascade provided by OpenCV.

OpenCV uses machine learning algorithms to search for faces within a picture. For something as complicated as a face, there isn't one simple test that will tell you if it found a face or not. Instead, there are thousands of small patterns/features that must be matched. The algorithms break the task of identifying the face into thousands of smaller, bite-sized tasks, each of which is easy to solve. These tasks are also called classifiers [7].

For something like a face, you might have 6,000 or more classifiers, all of which must match for a face to be detected (within error limits, of course). But therein lies the problem: For face detection, the algorithm starts at the top left of a picture and moves down across small blocks of data, looking at each block, constantly asking, *"*Is this a face? … Is this a face? … Is this a face*?"*

Since there are 6,000 or more tests per block, you might have millions of calculations to do, which will grind your computer to a halt [8].

To get around this, OpenCV uses cascades. Like a series of waterfalls, the OpenCV cascade breaks the problem of detecting faces into multiple stages. For each block, it does a very rough and quick test. If that passes, it does a slightly more detailed test, and so on. The algorithm may have 30-50 of these stages or cascades, and it will only detect a face if all stages pass. The advantage is that the majority of the pictures will return negative during the first few stages, which means the algorithm won't waste time testing all 6,000 features on it. Instead of taking hours, face detection can now be done in real time. Figure 5 shows the example of a Haar cascade with our subject.

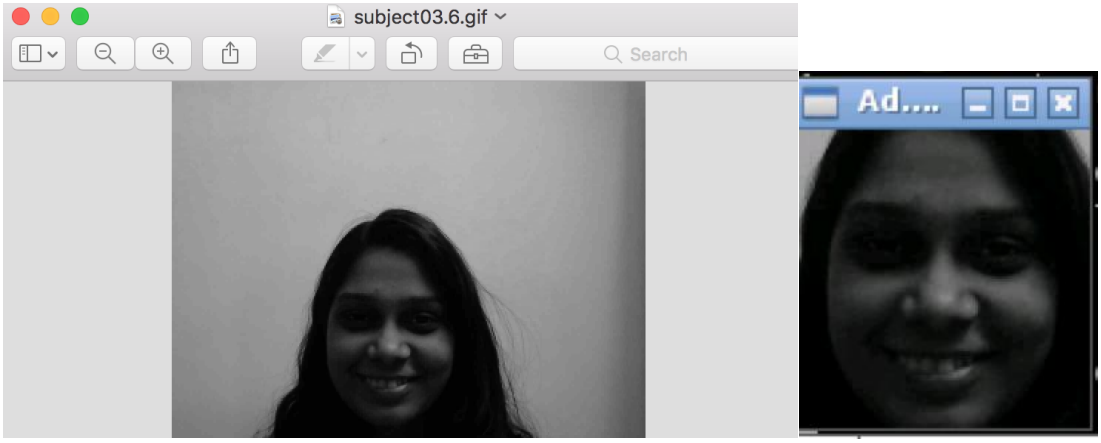


Fig. 5 Face detection from the subjects' image

The haar cascades that come with OpenCV are located in the /data/haarcascades directory of your OpenCV installation. We will use *haarcascade_frontalface_default.xml* for detecting the face. So, we load the cascade using the *cv2.CascadeClassifier* function which takes the path to the cascade xml file.

```
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
```

2.2.2.2 FACE RECOGNIZER INTERFACE

An intermediate state of face representation is needed we can apply face recognition. Based on pure pixel research has proven that faces are not unique enough unless you gather a tremendous

amount of facial features, which makes it even more complex. Therefore, we will make a derivate face representation, based on a single feature descriptor, for each person represented in a specific feature space, so that it is easier to separate different from each other. There are three possibilities, namely: three possibilities, namely:

- Eigenfaces: Eigenvalue decomposition through PCA.
- Fisherfaces: Linear discriminant analysis using the Fisher criterion.
- LBPH: Local Binary Pattern Histogram comparison

The basic idea behind each approach is that reduce the high dimensionality of the facial data to a lower-dimensional data representation, which aids us in better retrieving a correct match to our pre-created database.

While using Eigenfaces, increasing the number of eigenvectors used for projecting increases accuracy, but it also makes the algorithm slower. Finding a good trade-off is very important for our application. To avoid fraud getting in by the best match from the database principle, you can use the confidence scores to threshold out matches that are not secure enough. LDA, or linear discriminant analysis, based on the Fisher criterion, minimizes variance within a single class, while maximizing variance between classes at the same time, which makes the technique more robust in the long run[6].

Instead of simply reducing the dimensionality to universal axes, another approach is the use of local feature extraction. By looking at local features rather than a complete global description of the feature, researchers have tried to cope with problems like partial occlusion, illumination, and small sample size. The use of local binary pattern intensity histograms is a technique that looks at local face information rather than looking at global face information for a single individual. Local binary patterns have their origin in texture analysis and have proven to be efficient at face recognition by focusing on very specific local textured areas. This measure is more prone to changing lighting conditions than the previous techniques.

```
recognizer = cv2.createLBPHFaceRecognizer()
```

The face recognizer object has functions like `FaceRecognizer.train` to train the recognizer and `FaceRecognizer.predict` to recognize a face.

2.2.2 FINGERPRINT VERIFICATION

Fingerprint verification is responsible for matching the input fingerprint against the fingerprint template(s) stored in the database to obtain the fingerprint matching score.

A fingerprint is the pattern of ridges and furrows on the surface of a fingertip. The uniqueness of a fingertip is exclusively determined by the local ridge characteristics and their spatial relationships. The two most prominent ridge characteristics, called minutiae are: i) ridge ending and ii) ridge bifurcation.

Fingerprint verification depends on the comparison of minutiae and their relationships to make a personal identification. It usually consists of two stages: i) minutiae extraction and ii) minutiae matching. The minutiae extraction module extracts minutiae from input fingerprint images and the minutiae matching module determines the similarity of two minutiae patterns.

Fig. 6 Flow for fingerprint verification

2.2.2.1 BINARIZATION

The first step is to grab a grayscale image from the fingerprint system and to apply binarization. This enables to remove any noise from the image, as well as help to make the contrast better between the skin and the wrinkled surface of the finger. Binarization of the image is done through local thresholding. The Otsu thresholding will automatically choose the best generic threshold for the image to obtain a good contrast between foreground and background information. This is because the image contains a bimodal distribution (which means that we have an image with two

peak histograms) of pixel values. For that image, we can take an approximate value in the middle of those peaks as the threshold value (for images that are not bimodal, binarization won't be accurate). Otsu allows to avoid using a fixed threshold value, making the system more compatible with capturing devices. However, if only one capturing device is used, then playing around with a fixed threshold value may result in a better image for that specific setup. The result of the thresholding can be seen below:

Fig. 7 Comparison between grayscale and binarized image

2.2.2.2 SKELETONIZATION

Once a binary image is obtained, next step is to calculate feature points and feature point descriptors. However, in order to improve the process a bit more, it is better to skeletize the image. This will create more unique and stronger interest points. The skeletization is based on the Zhang-Suen line-thinning approach.

Fig. 8 Comparison of binarized and thinned fingerprint images using skeletization technique

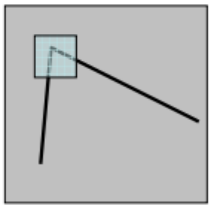
When this skeleton image is obtained, the next step is to look for crossing points on the ridges of the fingerprint, called the minutiae points. This can be done with a keypoint detector that looks for large changes in local contrast, like the Harris corner detector. Since the Harris corner detector is able to detect strong corners and edges, it is ideal for the fingerprint verification problem, where the most important minutiae are short edges and bifurcations – the positions where edges come together.

2.2.2.3 HARRIS CORNER DETECTOR

The Harris Corner Detector is a mathematical operator that finds features in an image. It is simple to compute, and is fast enough to work on computers. Another reason for choosing Harris corner detector is that it is rotation, scale and illumination variation independent.

The basic idea behind Harris corner detector is:

- Obtaining an easily recognizable point by looking through a small window
- Shifting a window in any direction should give a large change in intensity



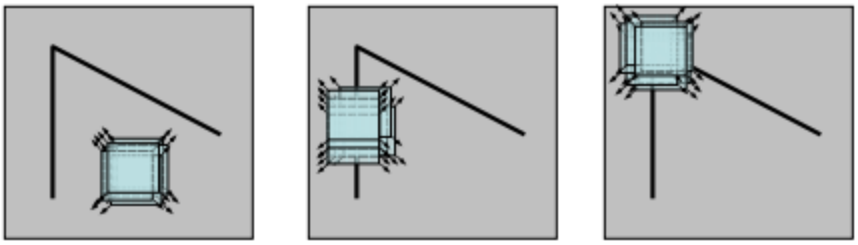


Fig. 9 Basic idea for Harris corner detector

Harris corner detector gives a mathematical approach for determining which case holds out of the above three.

Change of intensity for the shift [u,v] :

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$$

- E is the difference between the original and the moved window.
- u is the window's displacement in the x direction
- v is the window's displacement in the y direction
- w(x, y) is the window at position (x, y). This acts like a mask. Any desired window can be used (rectangular or guassian)
- I is the intensity of the image at a position (x, y)
- I(x+u, y+v) is the intensity of the moved window
- I(x, y) is the intensity of the original

For nearly constant patches, the term in the equation $[I(x+u, y+v)- I(x, y)]^2$ will be near 0. For very distinctive patches, this will be larger. Hence to find corner, aim is to find patches where E(u,v) is large[9].

Average intensity change in direction [u,v] can be expressed as a bilinear form:

$$E(u,v) \approx [u \quad v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$

M: summed matrix

$$M = \sum w(x,y) \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix}$$

A score, R, is calculated for each window:

$$R = \det M - k(trace\, M)^2$$

$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$

All windows that have a score R greater than a certain value are corners. They are good tracking points.

Calling the Harris Corner operation on a skeletonized and binarized image in OpenCV is quite straightforward. The Harris Corners are stored as positions corresponding with their cornerness response value in the image. If we want to detect points with a certain cornerness, then we should simply threshold the image [10].

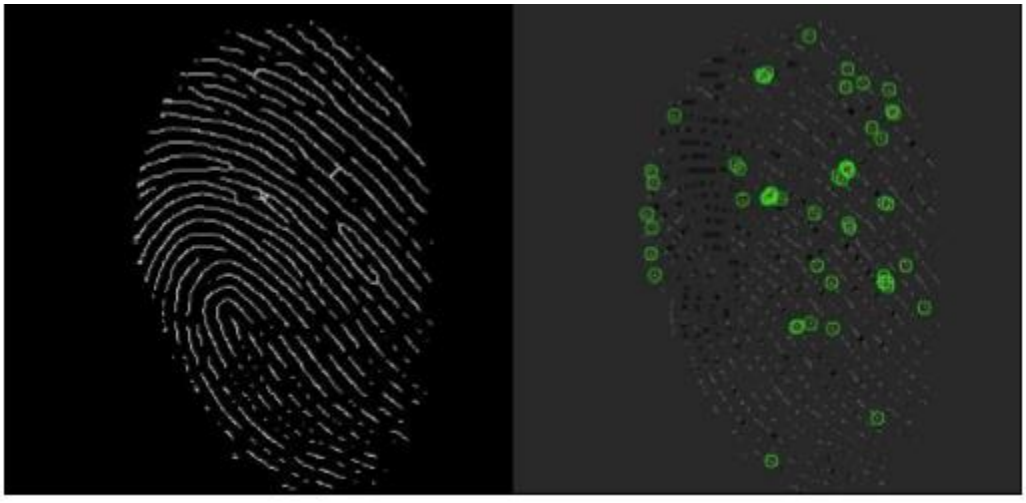


Fig. 10 Comparison of thinned image and Harris corner detector response

2.2.2.4 ORB DESCRIPTOR

Since we have an application where the orientation of the thumb can differ (since it is not in a fixed position), we want a keypoint descriptor that is good at handling these slight differences. Thus, using ORB as a feature descriptor is a good option; it being rotation invariant. However, it is required to calculate only the descriptors using the ORB approach, since the location of the keypoints using the Harris corner approach.

Oriented FAST and rotated BRIEF (ORB) is a fast robust local feature detector based on the FAST keypoint detector and the visual descriptor BRIEF (Binary Robust Independent Elementary Features)[11,12].

A binary descriptor is composed out of three parts:

1) A sampling pattern: where to sample points in the region around the descriptor.
2) Orientation compensation: some mechanism to measure the orientation of the keypoint and rotate it to compensate for rotation changes.
3) Sampling pairs: the pairs to compare when building the final descriptor.

There are two main differences between ORB and BRIEF:

1) ORB uses an orientation compensation mechanism, making it rotation invariant.
2) ORB learns the optimal sampling pairs, whereas BRIEF uses randomly chosen sampling pairs[13,14].

2.2.2.5 TESTING

At this point, a descriptor can be retrieved for each detected keypoint of any given fingerprint. The descriptors matrix contains a row for each keypoint containing the representation. Consider the example in which there is just one reference image for each fingerprint. We then have a database containing a set of feature descriptors for the training persons in the database. We have a single new entry, consisting of multiple descriptors for the keypoints found at registration time. We now have to match these descriptors to the descriptors stored in the database, to see which one has the

best match. The simplest way to do this is to perform brute-force matching using the hamming distance criteria between descriptors of different keypoints.

For each matching couple, we will have the original keypoint, the matched keypoint, and a floating point score between both matches, representing the distancethe matched points. This seems pretty straightforward. Looking at the number of matches that have been returned by the matching process and weigh them by their Euclidean distance in order to add some certainty. We then look for the matching process that yielded the biggest score. This will be our best match, and the match we want to return as the selected one from the database. If you want to avoid an imposter getting assigned to the best matching score, a manual threshold can be added on top of the scoring to avoid matches and ignore those that are not good enough. However, it is possible that, if you increase the score too much, people with little change will be rejected from the system, if, for example, someone cuts their finger and thus changes their pattern drastically.

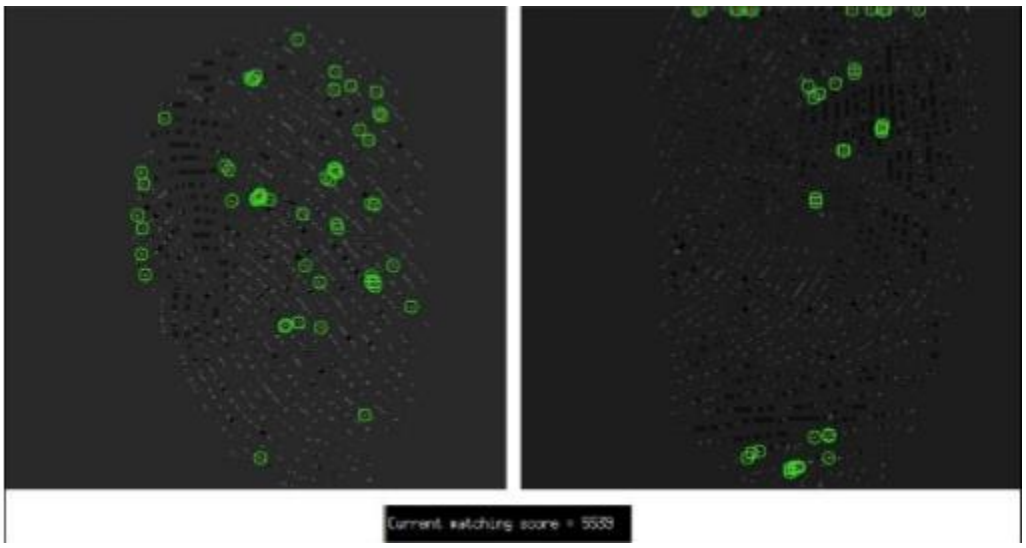


Fig. 11 Visualization of testing process

2.2.3 SPEAKER VERIFICATION

Speaker recognition is responsible for obtaining the matching score of the input speech signal. Anatomical variations that naturally occur amongst different people and the differences in their learned speaking habits manifest themselves as differences in the acoustic properties of the speech

signal. By analyzing and identifying these differences, it is possible to discriminate among speakers.

The project implements a text independent speaker recognition system which uses Mel Frequency Cepstral Coefficients (MFCC) followed by a Support Vector Machine (SVM).

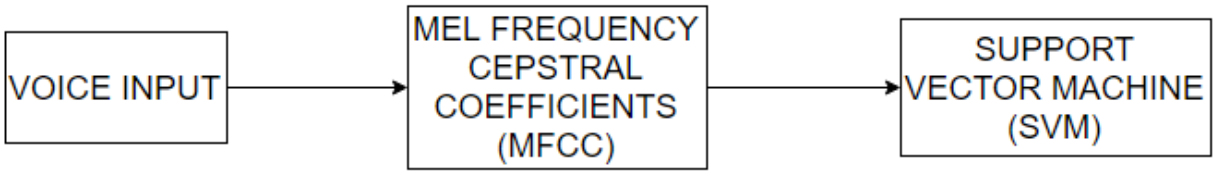


Fig. 12 Flow for speaker verification

2.2.3.1 MEL FREQUENCY CEPSTRAL COEFFICIENTS (MFCC)

The first step in any automatic speech recognition system is to extract features i.e. identify the components of the audio signal that are good for identifying the linguistic content and discarding all the other stuff which carries information like background noise, emotion etc.

The main point to understand about speech is that the sounds generated by a human are filtered by the shape of the vocal tract including tongue, teeth etc. This shape determines what sound comes out. If we can determine the shape accurately, this should give us an accurate representation of the phoneme being produced. The shape of the vocal tract manifests itself in the envelope of the short time power spectrum, and the job of MFCCs is to accurately represent this envelope.

Mel Frequency Cepstral Coefficients (MFCCs) are a feature widely used in automatic speech and speaker recognition. They were introduced by Davis and Mermelstein in the 1980's, and have been state-of-the-art ever since [15, 16].

2.2.3.2 STEPS FOR CALCULATING MFCC

STEP 1: Frame the signal into short frames.

An audio signal is constantly changing, so to simplify things we assume that on short time scales the audio signal doesn't change much (i.e. statistically stationary, obviously the samples are constantly changing on even short time scales). This is why the signal is framed into 20-40ms frames. If the frame is much shorter we don't have enough samples to get a reliable spectral estimate, if it is longer the signal changes too much throughout the frame.

STEP 2: For each frame calculate the periodogram estimate of the power spectrum.

The next step is to calculate the power spectrum of each frame. This is motivated by the human cochlea (an organ in the ear) which vibrates at different spots depending on the frequency of the incoming sounds. Depending on the location in the cochlea that vibrates (which wobbles small hairs), different nerves fire informing the brain that certain frequencies are present. Our periodogram estimate performs a similar job for us, identifying which frequencies are present in the frame.

STEP 3: Apply the Mel filterbank to the power spectra, sum the energy in each filter.

The periodogram spectral estimate still contains a lot of information not required for Automatic Speech Recognition. In particular the cochlea cannot discern the difference between two closely spaced frequencies. This effect becomes more pronounced as the frequencies increase. For this reason we take clumps of periodogram bins and sum them up to get an idea of how much energy exists in various frequency regions. This is performed by our Mel filterbank: the first filter is very narrow and gives an indication of how much energy exists near 0 Hertz. As the frequencies get higher our filters get wider as we become less concerned about variations. We are only interested in roughly how much energy occurs at each spot. The Mel scale tells us exactly how to space our filterbanks and how wide to make them. See below for how to calculate the spacing.

STEP 4: Take the logarithm of all filterbank energies.

Once we have the filter bank energies, we take the logarithm of them. This is also motivated by human hearing: we don't hear loudness on a linear scale. Generally to double the percieved volume of a sound we need to put 8 times as much energy into it. This means that large variations in energy may not sound all that different if the sound is loud to begin with. This compression operation makes our features match more closely what humans actually hear. Why the logarithm and not a cube root? The logarithm allows us to use cepstral mean subtraction, which is a channel normalization technique.

STEP 5: Take the DCT of the log filterbank energies

The final step is to compute the DCT of the log filterbank energies. There are 2 main reasons this is performed. Because our filterbanks are all overlapping, the filterbank energies are quite correlated with each other. The DCT decorrelates the energies which means diagonal covariance matrices can be used to model the features in e.g. a HMM classifier. But notice that only 12 of the 26 DCT coefficients are kept. This is because the higher DCT coefficients represent fast changes in the filterbank energies and it turns out that these fast changes actually degrade performance, so we get a small improvement by dropping them [17].

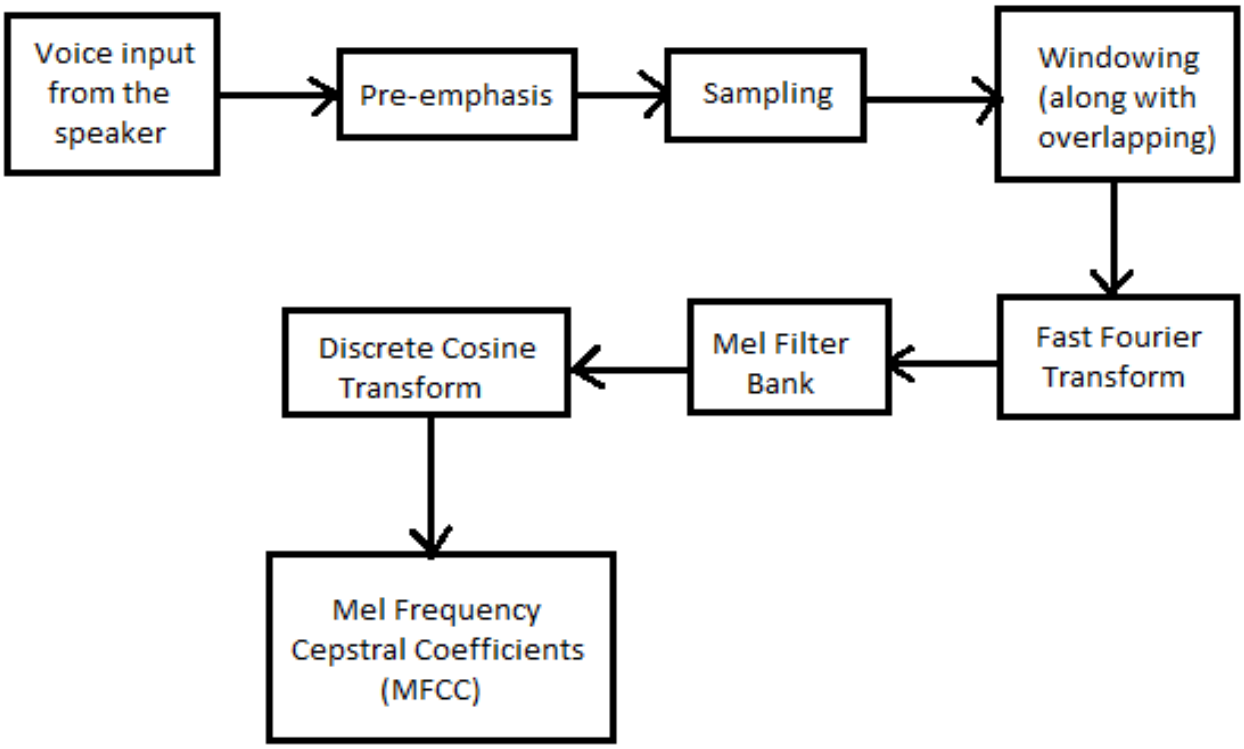


Fig. 13 Computing MFCC

2.2.3.3 MEL SCALE

The Mel scale relates perceived frequency, or pitch, of a pure tone to its actual measured frequency. Humans are much better at discerning small changes in pitch at low frequencies than they are at high frequencies. Incorporating this scale makes our features match more closely what humans hear.

The formula for converting from frequency to Mel scale is:

$$M(f) = 1125 \ln(1 + f/700)$$

To go from Mels back to frequency:

$$M^{-1}(m) = 700(\exp(m/1125) - 1)$$

2.2.3.4 IMPLEMENTATION STEPS

Sampling rate = 16 kHz

Frame duration = 25 ms

Frame length = 0.025*16000 = 400 samples

Frame step = 10 ms

The first 400 sample frame starts at sample 0, the next 400 sample frame starts at sample 160 etc. until the end of the speech file is reached. If the speech file does not divide into an even number of frames, pad it with zeros so that it does.

Time domain signal is denoted as s(n) and after sampling, $s_i(n)$ is obtained where n ranges over 1-400 (if our frames are 400 samples) and i ranges over the number of frames.

To take the Discrete Fourier Transform of the frame, perform the following:

$$S_i(k) = \sum_{n=1}^{N} s_i(n)h(n)e^{-j2\pi kn/N} \qquad 1 \leq k \leq K$$

where h(n) is an N sample long analysis window (e.g. hamming window), and K is the length of the DFT.

The periodogram-based power spectral estimate for the speech frame is given by:

$$P_i(k) = \frac{1}{N}|S_i(k)|^2$$

This is called the Periodogram estimate of the power spectrum. We take the absolute value of the complex Fourier transform, and square the result. We would generally perform a 512 point FFT and keep only the first 257 coefficients [15].

The Mel-spaced filter bank is a set of 20-40 (26 is standard) triangular filters that are applied to the periodogram power spectral estimate from step 2. The filterbank comes in the form of 26 vectors of length 257 (assuming the FFT settings fom step 2). Each vector is mostly zeros, but is

non-zero for a certain section of the spectrum. To calculate filterbank energies, multiply each filterbank with the power spectrum, then add up the coefficents. Once this is performed, left are 26 numbers that give an indication of how much energy was in each filterbank[16].

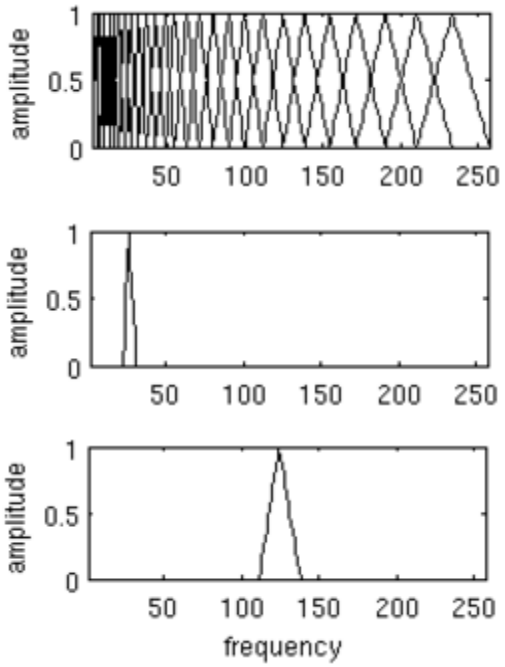


Fig. 14 The full Mel filterbank, filter 7 and 18 from filterbank

Take the log of each of the 26 energies from step 3. This leaves us with 26 log filterbank energies.

Take the Discrete Cosine Transform (DCT) of the 26 log filterbank energies to give 26 cepstral coefficents. Only the lower 12-13 of the 26 coefficients are kept. The resulting features (12 numbers for each frame) are called Mel Frequency Cepstral Coefficients.

The MFCC feature vector describes only the power spectral envelope of a single frame, but it seems like speech would also have information in the dynamics i.e. what are the trajectories of the MFCC coefficients over time. It turns out that calculating the MFCC trajectories and appending them to the original feature vector increases speech recognition performance by quite a bit (if we have 12 MFCC coefficients, we would also get 12 delta coefficients, which would combine to give a feature vector of length 24) [17].

To calculate the delta coefficients, the following formula is used:

$$d_t = \frac{\sum_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^{N} n^2}$$

where d_t is a delta coefficient, from frame t computed in terms of the static coefficients c_{t+N} to c_{t-N}. A typical value for N is 2. Delta-Delta (Acceleration) coefficients are calculated in the same way, but they are calculated from the deltas, not the static coefficients.

2.2.3.5 COMPUTING MEL FILTER BANKS

Step 1: To get the filter banks, first choose a lower and upper frequency.

Step 2: Using the above equation, convert the upper and lower frequencies to Mels.

Step 3: To do 26 filterbanks, 28 points are needed. This means 10 additional points are needed which are spaced linearly between the upper and lower Mel frequencies.

Step 4: Using the above equation, convert these back to Hertz.

Step 5: We don't have the frequency resolution required to put filters at the exact points calculated above, so we need to round those frequencies to the nearest FFT bin. This process does not affect the accuracy of the features. To convert the frequncies to fft bin numbers we need to know the FFT size and the sample rate,

```
f(i) = floor((nfft+1)*h(i)/samplerate)
```

Step 6: Create filterbanks. The first filterbank will start at the first point, reach its peak at the second point, then return to zero at the 3rd point. The second filterbank will start at the 2nd point, reach its max at the 3rd, then be zero at the 4th etc. A formula for calculating these is as follows:

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \le k \le f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) \le k \le f(m+1) \\ 0 & k > f(m+1) \end{cases}$$

2.2.3.6 SUPPORT VECTOR MACHINE (SVM)

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. The operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. Twice, this distance receives the important name of margin within SVM's theory. Therefore, the optimal separating hyperplane maximizes the margin of the training data [18].

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, the method is likely to give poor performances.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

This maximum margin solution allows the SVM to outperform most nonlinear classifiers in the presence of noise, which is one of the longstanding problems in ASR. Also, SVMs don't have the convergence and stability problems typical of other classifiers as Neural Networks (NNs) [19,20].

2.2.4 DECISION FUSION

The final decision made by the system is based on the integration of the decisions made by the fingerprint verification module, the face recognition module, and the speaker verification module.

If the output of each module is only a category label, i.e whether the claimed identity is true or not true, which is not associated with any confidence value, then the integration of these multiple

decisions can only be performed at an abstract level, in which a majority rule can be employed to reach a more reliable decision. If the output of each module is a similarity value/score, then a more accurate decision can be made at a rank level or a measurement level by accumulating the confidence associated with each individual decision

In a multimodal biometric system that uses several characteristics, fusion is possible at three different levels: feature extraction level, matching score level or decision level. Fusion at the feature extraction level combines different biometric features in the recognition process, while decision level fusion performs logical operations upon the monomodal system decisions to reach a final resolution. Score level fusion matches the individual scores of different recognition systems to obtain a multimodal score. Fusion at the matching score level is usually preferred by most of the systems.

Matching score level fusion is a two-step process: normalization and fusion itself. Since monomodal scores are usually non-homogeneous, the normalization process transforms the different scores of each monomodal system into a comparable range of values. After normalization, the converted scores are combined in the fusion process in order to obtain a single multimodal score.

2.2.4.1 CONSENSUS ALGORITHM

The consensus problem requires agreement among a number of processes (or agents) for a single data value. Some of the processes (agents) may fail or be unreliable in other ways, so consensus protocols must be fault tolerant or resilient. The processes must somehow put forth their candidate values, communicate with one another, and agree on a single consensus value.

The consensus problem is a fundamental problem in control of multi-agent systems. One approach to generating consensus is for all processes (agents) to agree on a majority value. In this context, a majority requires at least one more than half of available votes (where each process is given a vote). However one or more faulty processes may skew the resultant outcome such that consensus may not be reached or reached incorrectly.

A consensus protocol tolerating halting failures must satisfy the following properties.

- Termination

  Every correct process decides some value.
- Validity

  If all processes propose the same value v, then all correct processes decide v.
- Integrity

  Every correct process decides at most one value, and if it decides some value v, then v must have been proposed by some process.
- Agreement

  Every correct process must agree on the same value.

In evaluating the performance of consensus protocols two factors of interest are running time and message complexity. Running time is the number of rounds of message exchange as a function of some input parameters (typically the number of processes and/or the size of the input domain). Message complexity refers to the amount of message traffic that is generated by the protocol. Other factors may include memory usage and the size of messages.

2.3 STANDARDS

A technical standard is an established norm or requirement in regard to technical systems. It is usually a formal document that establishes uniform engineering or technical criteria, methods, processes and practices. In contrast, a custom, convention, company product, corporate standard, and so forth that becomes generally accepted and dominant is often called a de facto standard. Following are the two categories of standards used:

- International standards
- Open standards

2.3.1 INTERNATIONAL STANDARDS

Such standards are developed by international standards organizations. International standards are available for consideration and use worldwide.

- **PEP-8 - Styling standard for python coding**

  This specific standard covers coding conventions for the Python code comprising the standard library in the main Python distribution. The main python distribution includes standard libraries used in our project such as OS, PIL etc. The coding conventions cover coding layouts such as indentations, quotes, comments, imports, functions and many more used in Python 2.7. The standard source file encoding in which python 2.7 is distributed is UTF-8. The standard includes variable naming conventions too avoiding package or module names.

- **UART**

  A universal asynchronous receiver/transmitter (UART /ˈjuːɑːrt/), is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. The electric signaling levels and methods (such as differential signaling, etc.) are handled by a driver circuit external to the UART. A UART is usually an individual (or part of an) integrated circuit (IC) used for serial communications over a computer or peripheral device serial port.

- **SSH**

  Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network. SSH provides a secure channel over an unsecured network in a client-server architecture, connecting an SSH client application with an SSH server. Common applications include remote command-line login and remote command execution, but any network service can be secured with SSH.

- **IEEE 802.3**

  Ethernet is a family of computer networking technologies commonly used in local area networks (LAN), metropolitan area networks (MAN) and wide area networks (WAN). It was first standardized in 1983 as IEEE 802.3, and has since been refined to support higher

bit rates and longer link distances. Over time, Ethernet has largely replaced competing wired LAN technologies such as token ring, FDDI and ARCNET. Ethernet stations communicate by sending each other data packets: blocks of data individually sent and delivered. In our application, Ethernet is being used to provide internet access to BBB through a LAN connection.

2.3.2 OPEN STANDARDS

An open standard is a standard that is publicly available and has various rights to use associated with it, and may also have various properties of how it was designed (e.g. open process). There is no single definition and interpretations vary with usage.

- **OpenCV - Standard library and package used for manipulating images**

  OpenCV (*Open Source Computer Vision*) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel's research center in Nizhny Novgorod (Russia), it was later supported by Willow Garage and is now maintained by Itseez. The library is cross-platform and free for use under the open-source BSD license. OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE. OpenCV runs on a variety of platforms. Windows, Linux, macOS, FreeBSD, NetBSD, OpenBSD; Mobile: Android, iOS, Maemo, BlackBerry 10.

- **ALSA - Advanced Linux Sound Architecture**

  Advanced Linux Sound Architecture (ALSA) is a standard software framework and part of the Linux kernel that provides an application programming interface (API) for sound card device drivers. Some of the goals of the ALSA project at its inception were automatic configuration of sound-card hardware and graceful handling of multiple sound devices in

a system. ALSA is released under the GNU General Public License (GPL) and the GNU Lesser General Public License (LGPL).[4] Some frameworks such as JACK use ALSA to allow performing low-latency professional-grade audio editing and mixing.

- **MFCC - Mel-frequency cepstral coefficients**

MFCC**s** are coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound, for example, in audio compression.

- **BEAGLEBONE BLACK**

The BeagleBoard is a low-power open-source hardware single-board computer produced by Texas Instruments in association with Digi-Key and Newark element14. The BeagleBoard was also designed with open source software development in mind, and as a way of demonstrating the Texas Instrument's OMAP3530 system-on-a-chip.[8] The board was developed by a small team of engineers as an educational board that could be used in colleges around the world to teach open source hardware and software capabilities. It is also sold to the public under the Creative Commons share-alike license. The board was designed using Cadence OrCAD for schematics and Cadence Allegro for PCB manufacturing; no simulation software was used.

- **IMAGEMAGICK**

ImageMagick is a free and open-source software suite for displaying, converting, and editing raster image and vector image files. It can read and write over 200 image file

formats. ImageMagick is cross-platform, and runs on Microsoft Windows and Unix-like systems including Linux, macOS, iOS, Android, Solaris, and FreeBSD. The software mainly consists of a number of command-line interface utilities for manipulating images. ImageMagick does not have a robust graphical user interface to edit images as do Adobe Photoshop and GIMP, but does include – for Unix-like operating systems – a basic native X Window GUI (called IMDisplay) for rendering and manipulating images and API libraries for many programming languages.

- **ORB**

Since we have an application where the orientation of the thumb can differ (since it is not in a fixed position), we want a keypoint descriptor that is good at handling these slight differences. Algorithms like SIFT can be used, however, since SIFT is not under a BSD license, which can pose problems when used in commercial software. A good alternative in OpenCV is the ORB descriptor. Thus, using ORB as a feature descriptor is a good option. Oriented FAST and rotated BRIEF (ORB) is a fast robust local feature detector based on the FAST keypoint detector and the visual descriptor BRIEF (Binary Robust Independent Elementary Features).

2.4 CONSTRAINTS, ALTERNATIVES AND TRADE OFFS

- Face Detection using Haar cascade is most effective only on frontal images of faces. It can hardly cope with 45° face rotation both around the vertical and horizontal axis. Face detection alone is sensitive to lighting conditions. We might get multiple detections of the same face, due to overlapping sub-windows.
- In case of noisy environments the accuracy of our speaker verification system might be less.
- Finger orientation while acquiring the input for finger print verification module might create an issue. However the use of ORB descriptor takes care of rotation variance.

- While acquiring the database for the face recognition system only 2D images where captured with a plane white background considering only one face in the frame at a time. Leading to issues when more than one face come into the frame while testing.
- Unable to capture 3D depth information while taking input for face and finger print verification might encourage some fraudulent activities.
- Scars, bruises on faces and finger, disruption in voice data captured due to throat illness might not be accounted by our system. However multimodal approach takes care if there is a problem in any such module

2.4 TECHNICAL SPECIFICATION

Our system combines the USB based camera and microphone acquisition modules with the BeagleBone black as the central processing unit. The beagle bone black comes with a AM335x ARM Cortex-A8 processor supporting a processing speed of 1 GHz sufficient to perform the acquisition, processing and decision making for the individual modules. The provision of SD Card slot on the SoC gives us disk extendibility over the 4GB 8-bit eMMC on-board flash storage options providing feasibility to work with large sized packages such as OpenCV, etc. Open Source Debian OS that runs over the hardware eases the programmability of the user. BeagleBone black being pocket sized gives portability features to our system. 3D graphics accelerator and NEON floating-point accelerator improve the precision as well as improve the processing efficiency of the system. USB client for power and secure shell communications gave easy connectivity to the device [21].

Our system integrated the Logitech c270 USB based camera modules for facial and fingerprint verification system. The USB camera comes with image and video support of up to 1280x720 pixels and 3 MP software enhancement on the images captured. The USB based cameras support USB 2.0 Hi-speed communication standard. The added advantage of using these cameras is that they have hardware encryption supported on them which gives the user an option to either get raw images and jpeg images. Getting the JPEG images directly reduces the processing at the centralized processor [22].

For acquiring audio samples from the user our system uses a combination pair of USB based sound adapters and 3.5 mm audio jack based standard microphones were used. The adapter specifications include USB 2.0 Full-Speed (12Mbps), compatible with USB 1.0/1.1, 12 channel equalizer with a separate jack for stereo output and one for mono microphone jack. The adapter requires no specific driver to run on any Linux OS. The microphone used in our system comes with a 360° Flexible gooseneck holder, convenient for use. The microphone features also include widescreen protector to help reduce wind noise as well as prevents 'P' & 'B' consonants distorting. The polar pattern of the microphone is omnidirectional. Figure 15 shows the sub-systems combined to make a complete multimodal person identification system with an innovative segment of the project shown in pictorial format in Figure 16.

Fig. 15 Sub-systems integrated to make the multimodal person identification system

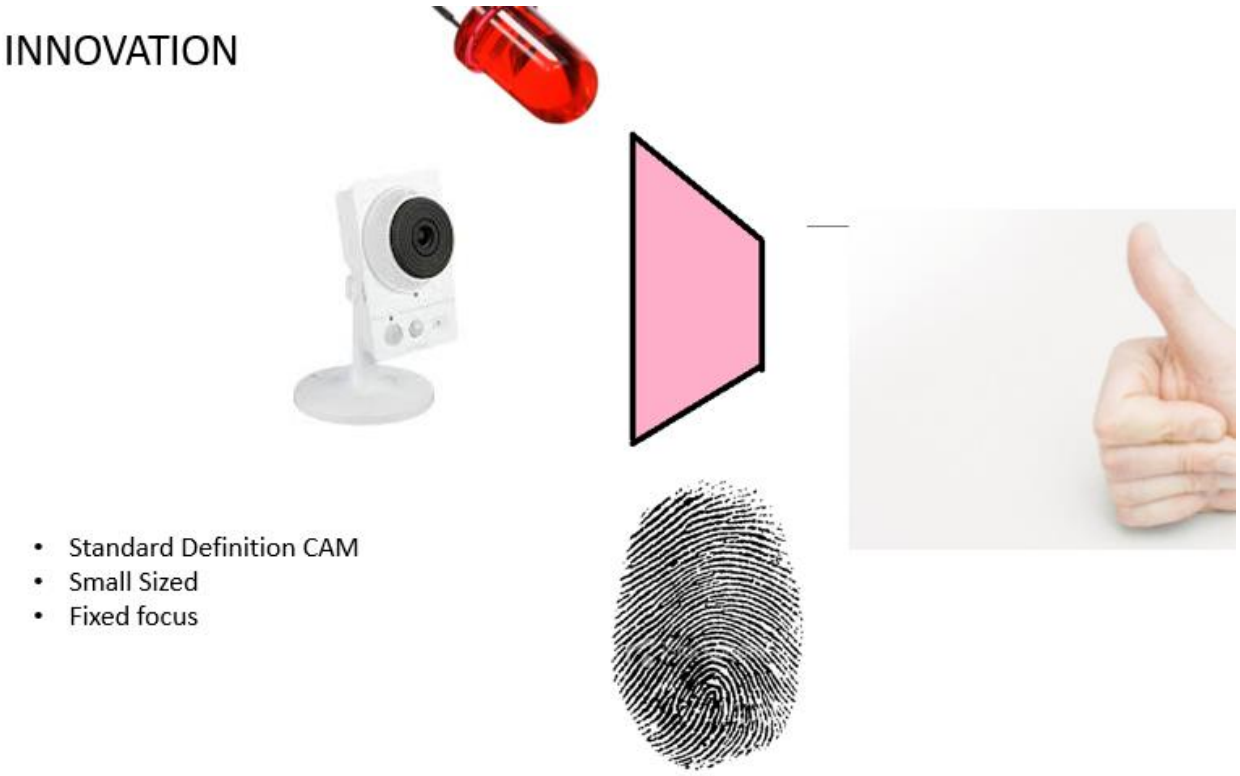


Fig. 16 Fingerprint biometric acquisition as an innovative segment in our project

CHAPTER – 3
PROJECT DEMONSTRATION

3.1 DATABASE CREATION

For the system to work seamlessly without much error the system, i.e the voice, facial and fingerprint, must been trained with data acquired under certain strict environment conditions.

In case of facial recognition the images taken must have a single face in background, must be taken in proper lighting conditions and the person volunteering for the database must not tilt his/her face while capturing images for the database. Our system only supports 2D facial recognition so side faces will not be used for training as well as testing. For creating a database for facial recognition the subjects were asked to sit in front of the same system facing the camera and 20 facial images per subject were taken and stored in the system for training. The subjects were asked to make different facial expressions (sad, happy, surprised, normal, with glasses, no glasses, angry etc.) for the images used to create database. Initial testing were done dividing the database into sections for training and testing further testing it in live scenario. Figure 17 shows our small sample database. Figure 18,19 show the training and testing phase on Beagle Bone Black.

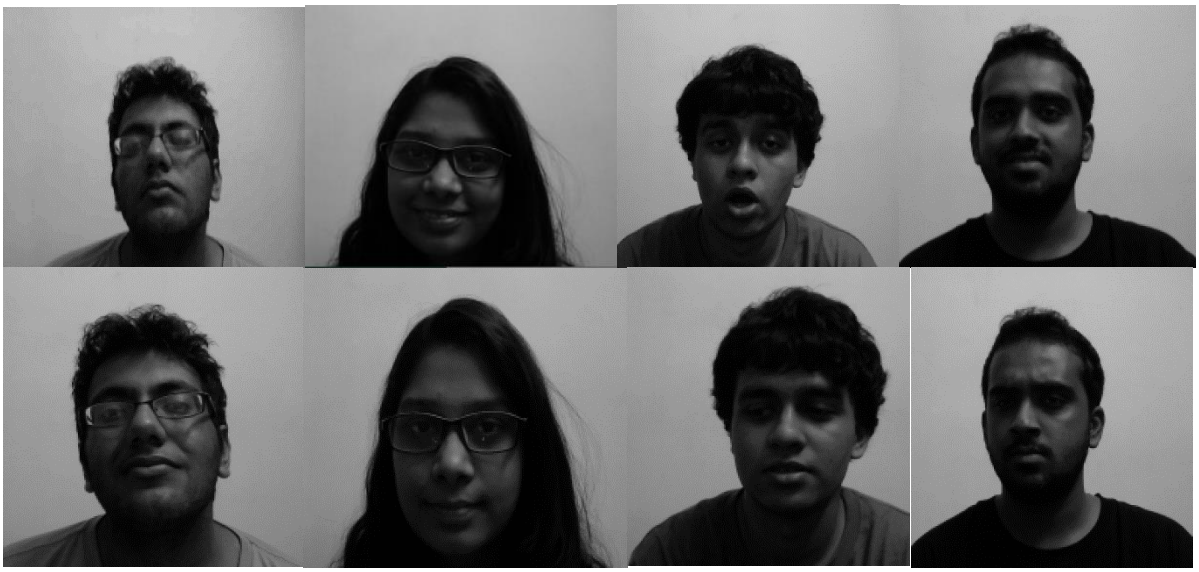


Fig. 17 A sample database for testing face recognition

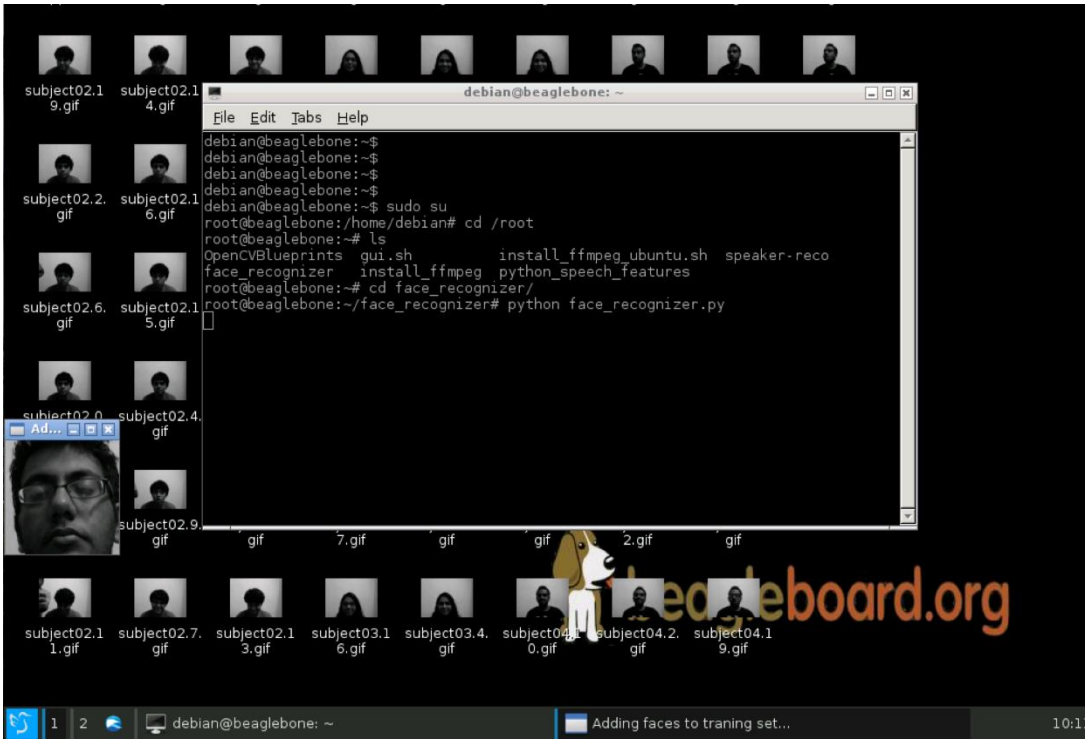


Fig. 18 Training on BBB with the database

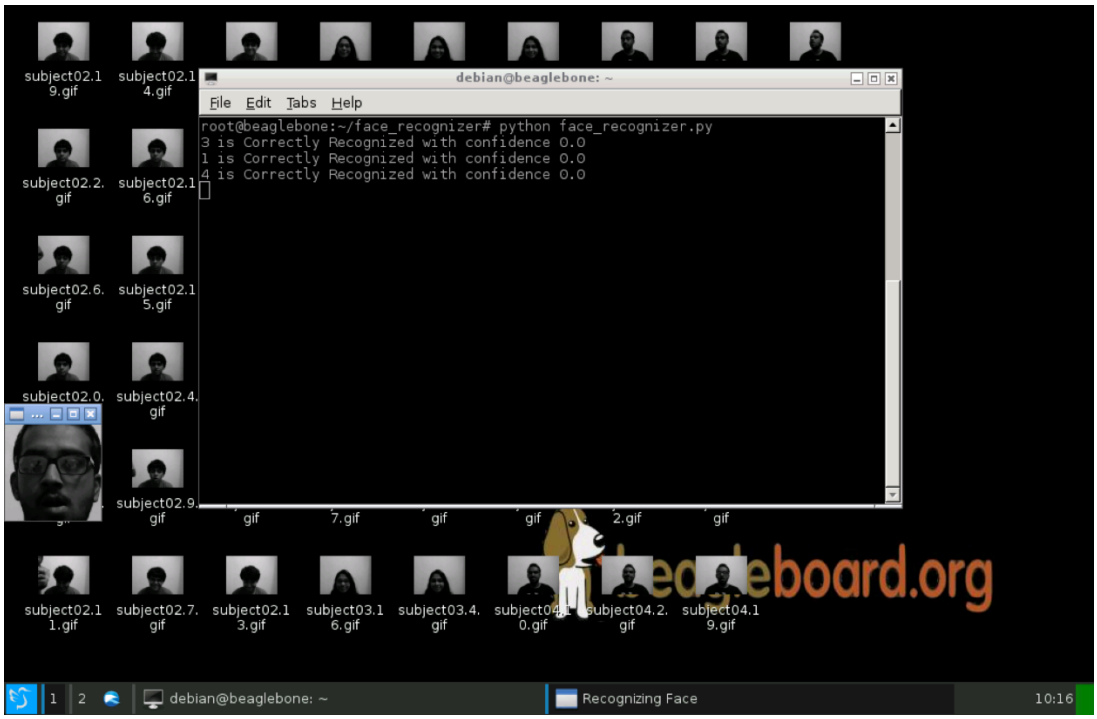


Fig. 19 Testing the subject over the trained database and generating the score

In case of speaker verification system the database creation was again done in low noise environments and the recording parameters were also kept constant. The recording parameters were

- Rate – 16000 Hz
- Duration – 5 sec
- Format – 16 bit little endian
- Channel - Mono

The subject were asked to record 20 voice data samples with a standard set of sentences (selected from TIMIT database). For each voice data sample the subject were asked to record a sentence from a set given below:

1 The empty flask stood on the tin tray
2 It's easy to tell the depth of a well
3 These days a chicken leg is a rare dish
4 Fairy tales should be fun to write
5 Know closely the size of the gas tank
6 Wipe the grease off his dirty face
7 Add salt before you fry the egg
8 Slide the box into that empty space
9 Nine rows of soldiers stood in line
10 The coffee stand is too high for the couch
11 We admire and love a good cook
12 Live wires should be kept covered
13 the large house had hot water taps
14 To make pure ice you freeze water
15 let's all join as we sing the last chorus
16 The pencils have all been used
17 The stray cat gave birth to kittens
18 Try to trace the fine lines of the painting
19 The boy was there when the sun rose

20 We tried to replace the coin but failed

Since the dataset chosen is text independent the subject may not necessarily speak a sentence from the dataset while testing. Our system will handle text independent data. Figure 20 shows the options given to a user while training and testing the speaker recognition sub-system. Figure 21 shows the MFCC coefficients of the sample database trained.

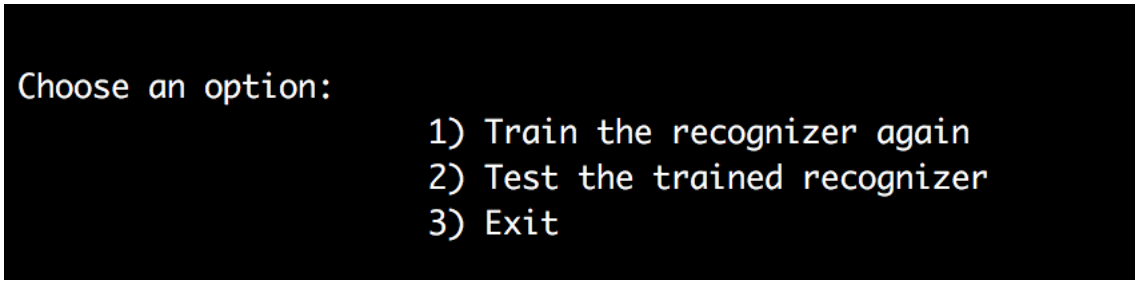


Fig. 20 Set of options provided to the user in debugging and training mode

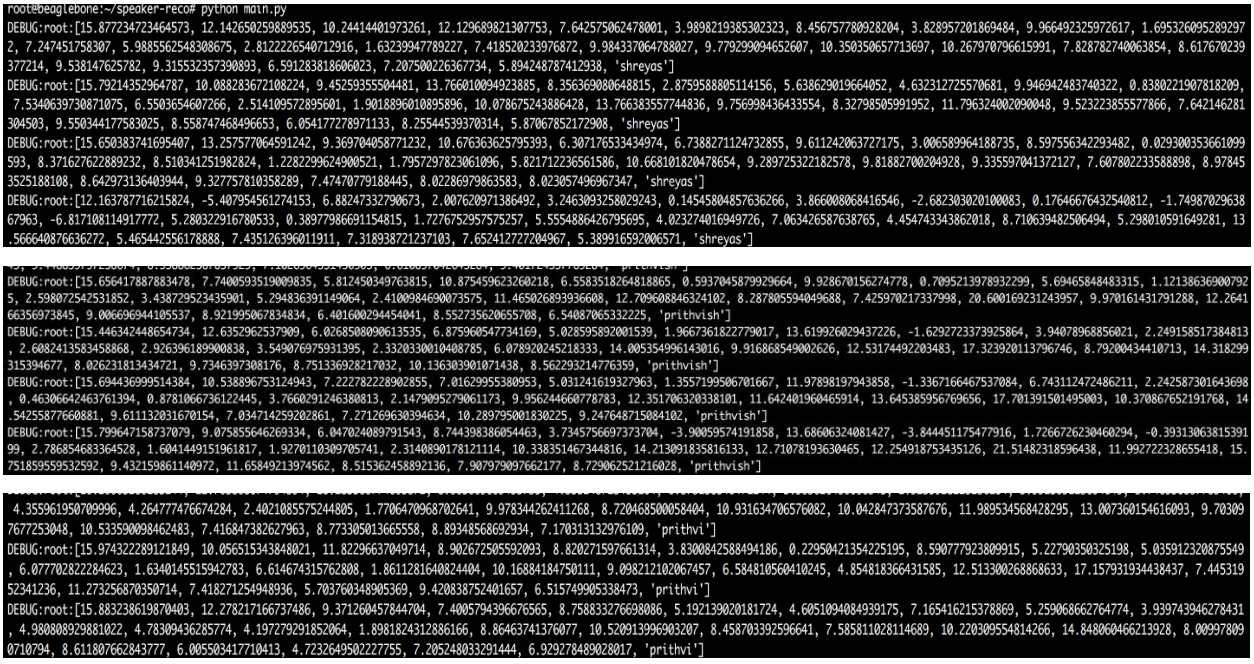


Fig. 21 MFCC for different subjects used for training

In-case of fingerprint database the database was recorded keeping the orientation of the finger in mind as well as the lighting condition near the camera module and the screen over which the finger can be placed. Considering the above conditions the dataset of 20 samples for a single subject were acquired and tested with. In the following system, though we can add as many fingers for a single

individual for the proof of concept the count is kept as 1 and the recommended finger is a right hand thumb. In case a right hands thumb is not an appropriate choice (in case of cut/injury) the subject can use other fingers to create the database. Figure 22 to 26 shows the complete demonstration of fingerprint verification on BBB.

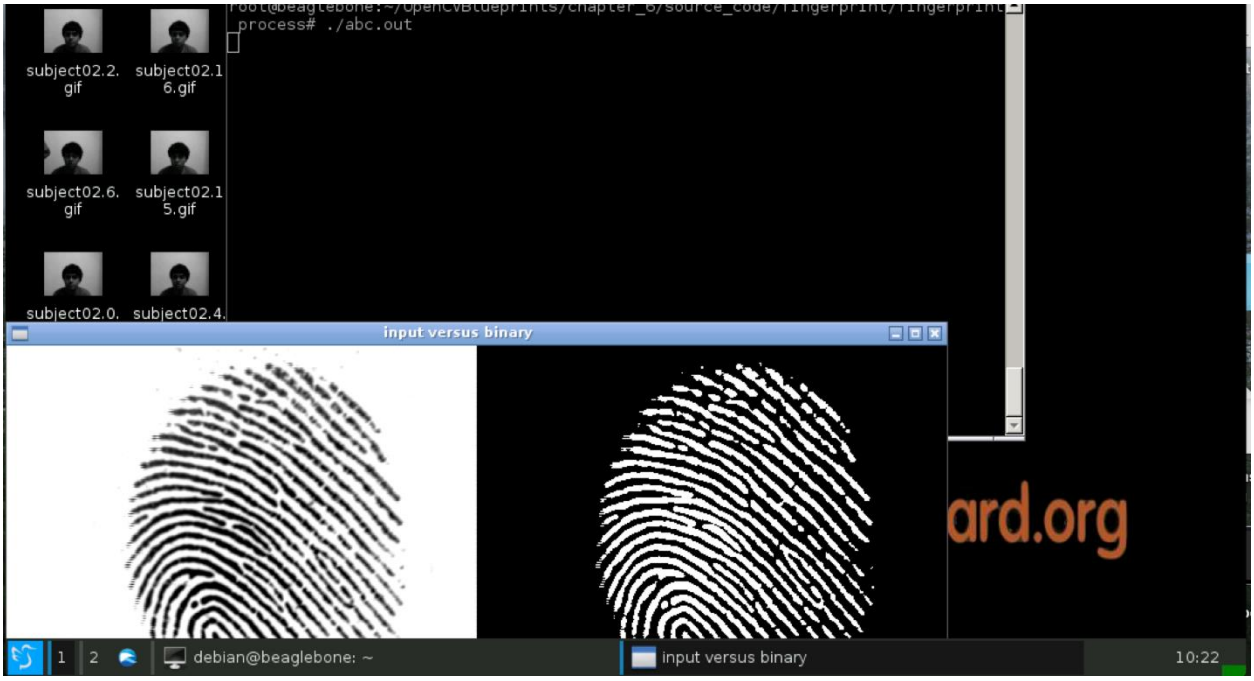


Fig. 22 Conversion of fingerprint image into binarized image

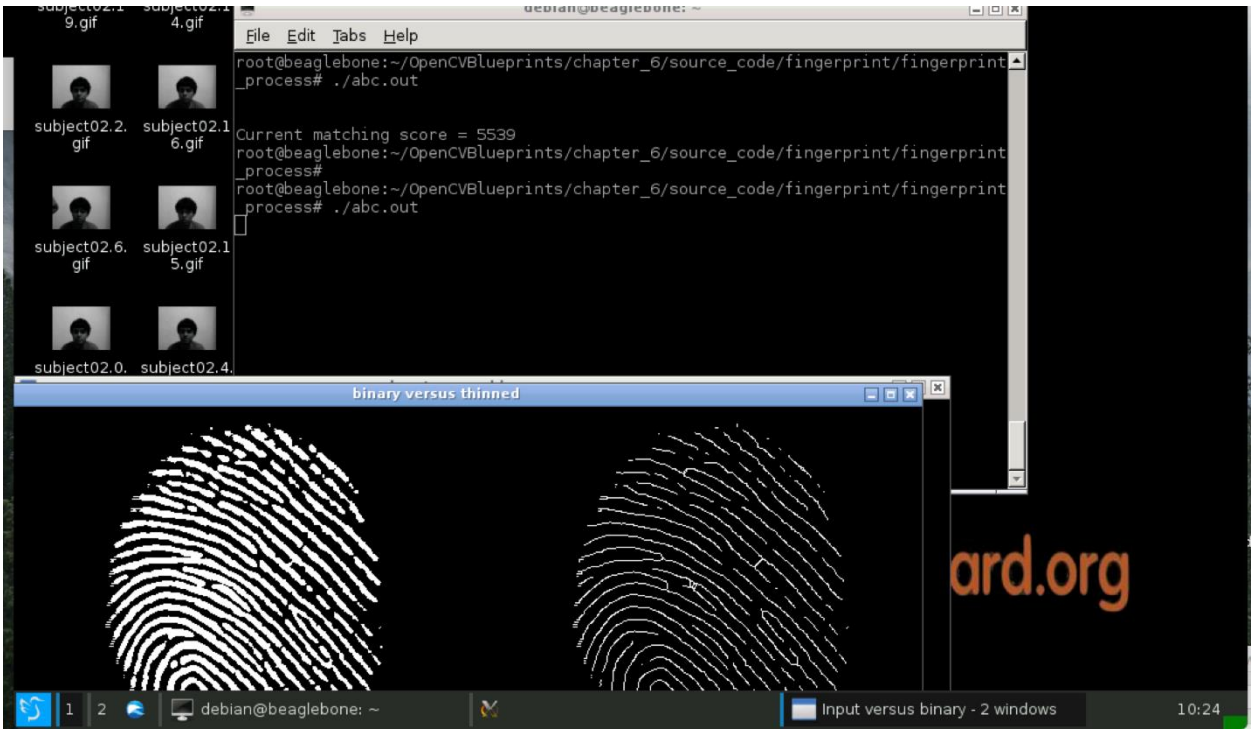


Fig. 23 Thinning of the binarized image

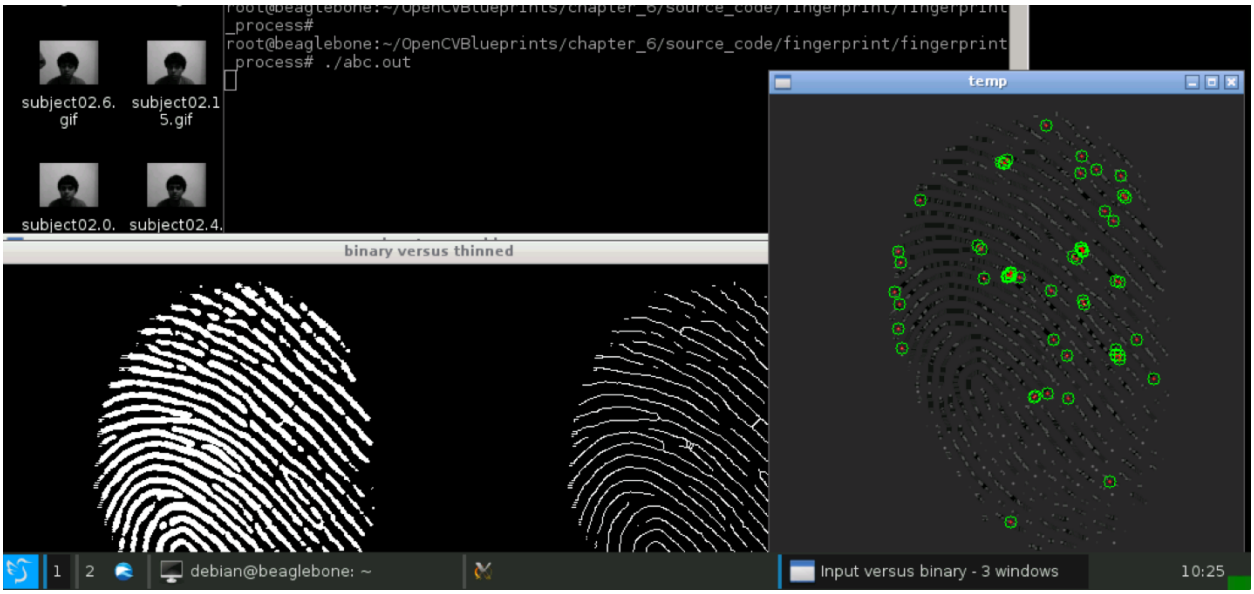


Fig. 24 Extracting corners using Harris corner detector

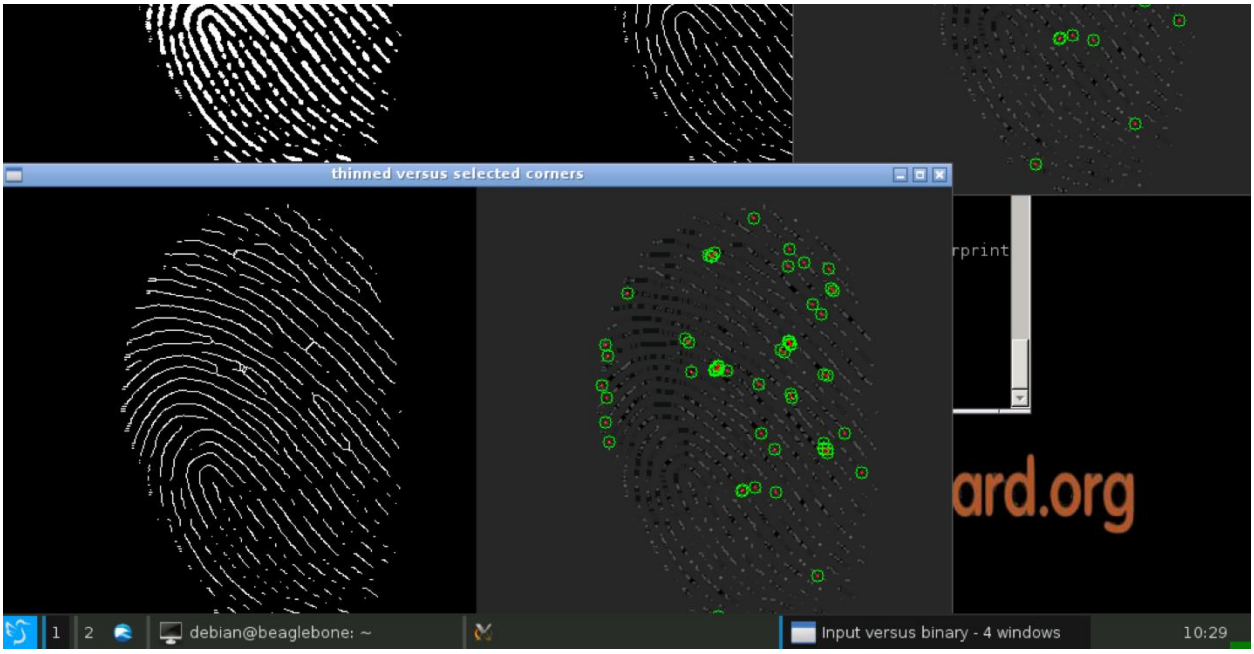


Fig. 25 Generating features using ORB descriptor

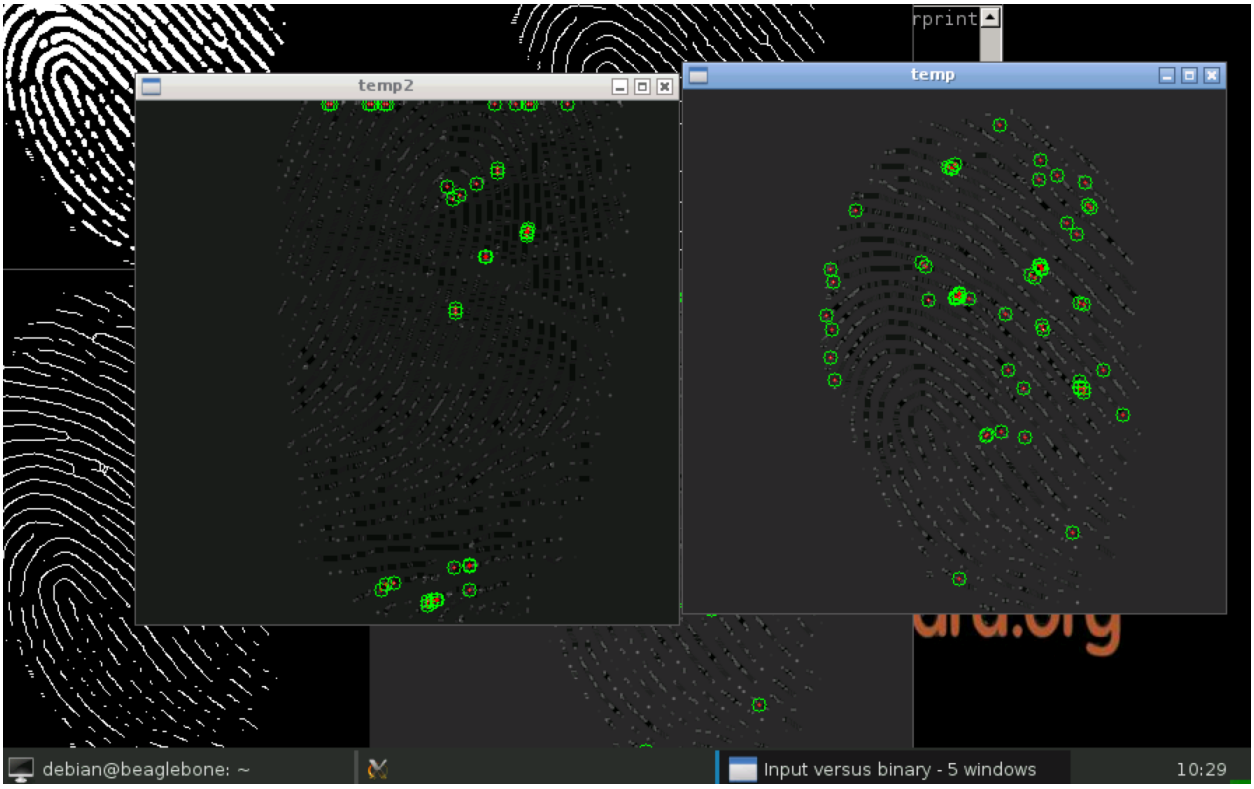


Fig. 26 Testing the fingerprint and generating the score

3.2 HARDWARE POSITIONING

For the proof of concept a modular portable structure is created and, modules a mounted in positions making it convenient for the subjects to give samples as well as use the same framework for testing.

Further for testing it acts as a standalone system, LED can be added making it simpler for the user to authenticate the identity for a user and grant him access. In the testing framework LEDs will also be added indicating the proper run of sub-systems making it easier for the user to maintain a flow.

In our framework, the USB camera is mounted 30 cm above the base level for facial recognition. The camera for the facial recognition can also be adjusted over its rotatable axial points. The microphone over the framework is kept as close as the user mounted around 6 cm above the base level. The camera for the fingerprint is mounted flat on the base level with its lens pointing to the top where the screen is mounted around 6 cm above the base level. The central processing device which is BeagleBone Black in our case is placed in the box with a transparent opening where all the modules get connected. To test the working of every module the system can also be connected over the USB host provided by Beagle bone black. The testing is done creating a desktop over the VNC X11 connection and the processing and workflow is tested. Figure 27 and 28 show the hardware structure to test the multimodal person identification system

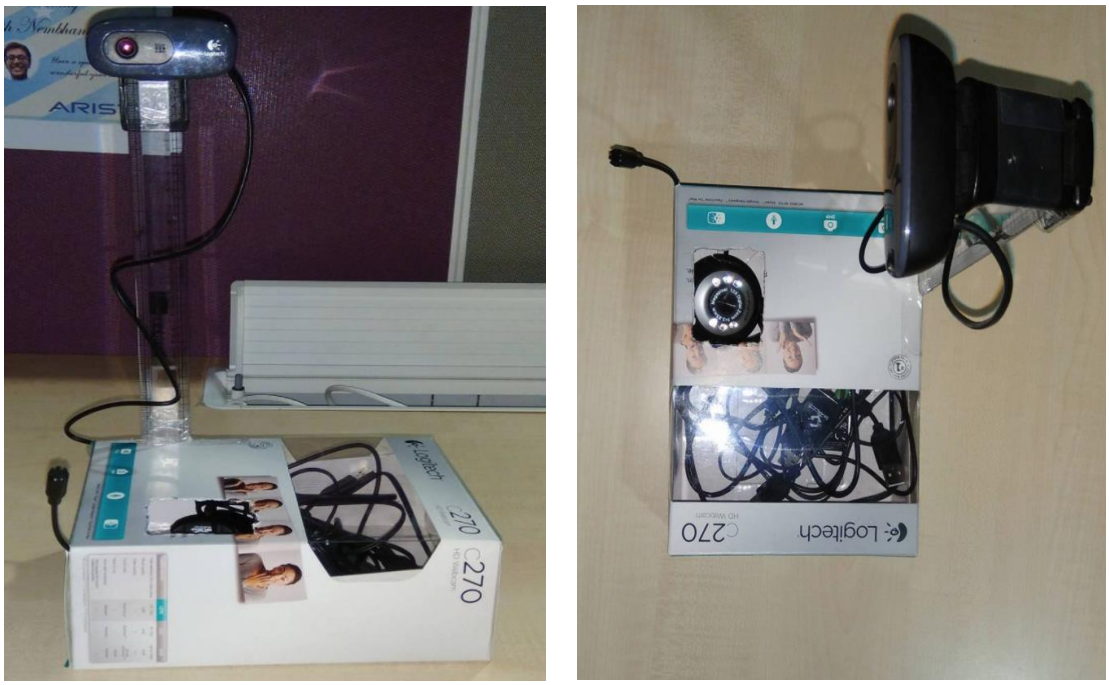


Fig. 27 Front and top view of the multi-modal person identification system's structure

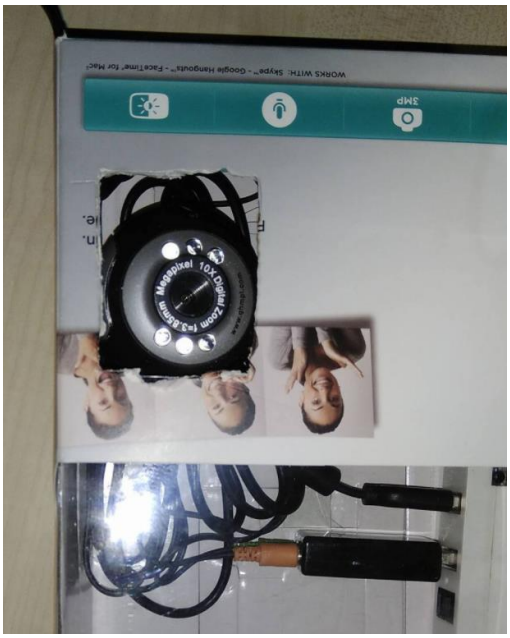


Fig. 28 Fingerprint camera module in the system

3.3 TESTING FLOW

In our system all the modules run in a sequential fashion. As our system is booted, for the verification to trigger a hot key (button) is required. On the trigger of the hot-key, the LEDs placed on the system guide the user through the procedure starting from facial recognition followed by voice recognition followed by fingerprint verification. On the central processor, after the end of every subsystem, a score is calculated. At the end of the whole process the consensus algorithm calculated using the score generated by each module and gives the final decision which is shown on the LED, indicating that the access is given to the user being identified.

CHAPTER – 4

COST ANALYSIS

Sr No.	Component/module	Price (INR)
1	BeagleBone Black	4000
2	USB Camera (Logitech) X 2	1250 X 2 = 2500
3	USB MUX	200
4	USB based audio adapter (Sound card)	200
5	Microphone	100
6	LEDs and jumper wires	50
7	Power Bank	1000
		7950

NOTE

- The major cost reduction in the system is the implementation of USB camera based fingerprint extraction module instead of the fingerprint module available in the market.
- Further reduction in the cost can be made if cheaper USB based cameras are used for the face and finger verification. The logitech cameras have a hardware for jpeg compression on the module itself which provides faster transfer of image data to the central device and in many cases also saves computation for the central device.

CHAPTER – 5
RESULTS AND DISCUSSIONS

The above approach when implemented in real life and tested gave us around 95 % accuracy with facial recognition, 85 % percent with speaker verification and around 80 % with fingerprint recognition system for the database. The combined approach increases the efficiency to a decent level making the system ready to implement in real life. The efficiency of sub-systems speaker recognition system can be increased by adding more features other than Mel Frequency Cepstral Coefficients (MFCC) such as Linear Prediction Cepstral Coefficients (LPCC) and Perceptual Linear Prediction Coefficients (PLP). Another factors one should consider to increase the efficiency of the system are the computation time and scalability. The project scope lies to cost reduction and implementation section over BBB making the whole system portable. Further increasing the efficiency of the process is covered under the future scope of the project.

CHAPTER – 6
CONCLUSIONS

A multimodal biometrics technique, which combines multiple biometrics in making a personal identification, can be used to overcome the limitations of individual biometrics. We have developed a multimodal biometric system which integrates decisions made by face recognition, fingerprint verification and speaker verification to make a personal identification. In order to demonstrate the efficiency of such an integrated system, experiments which simulate the operating environment on a small data set which is acquired in a strict environment were performed. The experimental results show that our system performs very well. However, the system needs to be tested on a large dataset in a real operating environment. We were successful in making our system portable. Our system supports both training and testing, make it a ready to deploy system in the market with no dependencies.

APPENDIX

Pseudocode for face recognition :

creating database

```
import cv2
import os
import numpy

dim=(320,243)

# taking basic parametres to create a subjectwise database
name=raw_input('Enter the name of the subject')
id=raw_input('enter the ID number of the subject')
x=raw_input('enter the number of captures')
os.system("echo "+name+":"+id+">> map_name_id.txt")
path_to_database="./my_database/"

for i in range(0,int(x)):
raw_input("Press any key to continue ...")
print "Taking capture ..."
os.system('./grabber')
im=numpy.array(Image.open('grabber000.ppm'))
print "Pre-processing ..."
im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
im = cv2.resize(im, dim, interpolation = cv2.INTER_AREA)
im = Image.fromarray(numpy.uint8(im))
im.save('temp.ppm')
os.system('convert -delay 2 temp.ppm '+path_to_database+'subject'+id+'.'+str(i)+'.gif')
os.system('rm temp.ppm')
```

Training and testing

```
#!/usr/bin/python

# Import the required modules
import cv2, os
import numpy as np
from PIL import Image

dim=(320,234)

# For face detection we will use the Haar Cascade provided by OpenCV.
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath)

# For face recognition we will the the LBPH Face Recognizer
recognizer = cv2.face.createLBPHFaceRecognizer()


def get_images_and_labels(path):
    # Append all the absolute image paths in a list image_paths
    # We will not read the image with the .sad extension in the training set
    # Rather, we will use them to test our accuracy of the training
    image_paths = [os.path.join(path, f) for f in os.listdir(path) if not f.endswith('.sad')]
    # images will contains face images
    images = []
```

```
    # labels will contains the label that is assigned to the image
    labels = []
    for image_path in image_paths:
        # Read the image and convert to grayscale
        image_pil = Image.open(image_path).convert('L')
        # Convert the image format into numpy array
        image = np.array(image_pil, 'uint8')
        # Get the label of the image
        nbr = int(os.path.split(image_path)[1].split(".")[0].replace("subject", ""))
        # Detect the face in the image
        faces = faceCascade.detectMultiScale(image)
        # If face is detected, append the face to images and the label to labels
        for (x, y, w, h) in faces:
            images.append(image[y: y + h, x: x + w])
            labels.append(nbr)
            cv2.imshow("Adding faces to traning set...", image[y: y + h, x: x + w])
            cv2.waitKey(50)
    # return the images list and labels list
    return images, labels

# Path to the Yale Dataset
path = './my_database'
# Call the get_images_and_labels function and get the face images and the
# corresponding labels
images, labels = get_images_and_labels(path)
cv2.destroyAllWindows()

# Perform the tranining
recognizer.train(images, np.array(labels))

raw_input("Please enter to capture the image ....")
os.system('./grabber')
im=np.array(Image.open('grabber000.ppm'))
print "Pre-processing ..."
im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
im = cv2.resize(im, dim, interpolation = cv2.INTER_AREA)
im = Image.fromarray(np.uint8(im))
im.save('temp.ppm')
os.system('convert -delay 2 temp.ppm temp.gif')
image_path = "./temp.gif"
predict_image_pil = Image.open(image_path).convert('L')
predict_image = np.array(predict_image_pil, 'uint8')
faces = faceCascade.detectMultiScale(predict_image)
for (x, y, w, h) in faces:
result = cv2.face.MinDistancePredictCollector()
recognizer.predict(predict_image[y: y + h, x: x + w],result, 0)
nbr_predicted = result.getLabel()
conf = result.getDist()
        for nbr_actual in range(1,5):
         if nbr_actual == nbr_predicted:
            print "{} is Correctly Recognized with confidence {}".format(nbr_actual,conf)
         else:
            print "{} is Incorrect Recognized as {}".format(nbr_actual, nbr_predicted)
        cv2.imshow("Recognizing Face", predict_image[y: y + h, x: x + w])
        cv2.waitKey(1000)
```

Pseudocode for speaker verification :

database creation

```
#!/usr/bin/env bash
# This script is used to record training data for a speaker

if [[ -z ${1} ]]; then
echo 'Please specify the data directory as the first argument.'
exit 1
fi

DATA_DIR=${1}
saved_dir=$(pwd)

echo -n 'Enter the name of the Speaker: '
read speaker
echo -n 'Enter the number of files to record: '
read fcount

DEST_DIR="${DATA_DIR}/${speaker}"
mkdir ${DEST_DIR}
cd ${DEST_DIR}

for i in $(seq 1 ${fcount}); do
echo -n "Press any key to start recording."
read tmp
echo "Recording file #${i}..."
#arecord --device=plughw:0,0 --format S16_LE --rate 44100 -c1 test.wav
arecord --device=plughw:1,0 -d 5s --format S16_LE -t wav -r 16000 "clip${i}.wav"
done

cd ${saved_dir}
```

extracting MFCC and training

```
#!/usr/bin/env python2

from __future__ import division

import csv
import os.path
import logging
import numpy as np
import scipy.io.wavfile as wavfile

from sklearn import svm
from features import mfcc

logging.basicConfig(level=logging.DEBUG)

class RecoBlock():
    """Recognizer [If there is a word like that] Block. This class
    keeps the state required to recognize a set of speakers after it has
    been trained.
    """

    def __init__(self, data_dir, out_dir="_melcache"):
```

```
        self.data_dir = os.path.abspath(data_dir);

        # make folder for storing the csv file holding training data
        melcache_dir = os.path.join(os.getcwd(), "_melcache")
        train_file = os.path.join(melcache_dir, "training_data.csv")
        try:
            os.mkdir(melcache_dir)
            # generate training dataset csv file and get data for training
            self._gen_features(self.data_dir, train_file)
        except OSError:
            logging.debug("_melcache already exists. Assuming training_data.csv exists too.")

        self.recognizer = svm.LinearSVC()
        melv_list, speaker_names = self._get_tdata(train_file)

        # generate speaker_ids from speaker_names
        self.spkr_ntoi = {}
        self.spkr_iton = {}

        i = 0
        for name in speaker_names:
            if name not in self.spkr_ntoi:
                self.spkr_ntoi[name] = i
                self.spkr_iton[i] = name
                i += 1

        speaker_ids = map(lambda n: self.spkr_ntoi[n], speaker_names)

        logging.debug(speaker_ids)

        # train a linear svm now
        self.recognizer.fit(melv_list, speaker_ids)

    def _mfcc_to_fvec(self, ceps):
        # calculate the mean
        mean = np.mean(ceps, axis=0)
        # and standard deviation of MFCC vectors
        std = np.std(ceps, axis=0)
        # use [mean, std] as the feature vector
        fvec = np.concatenate((mean, std)).tolist()
        return fvec

    def _gen_features(self, data_dir, outfile):
        """ Generates a csv file containing labeled lines for each speaker """

        with open(outfile, 'w') as ohandle:
            melwriter = csv.writer(ohandle)
            speakers = os.listdir(data_dir)

            for spkr_dir in speakers:
                for soundclip in os.listdir(os.path.join(data_dir, spkr_dir)):
                    # generate mel coefficients for the current clip
                    clip_path = os.path.abspath(os.path.join(data_dir, spkr_dir, soundclip))
                    sample_rate, data = wavfile.read(clip_path)
                    ceps = mfcc(data, sample_rate)

                    # write an entry into the training file for the current speaker
                    # the vector to store in csv file contains the speaker's name at the end
                    fvec = self._mfcc_to_fvec(ceps)
```

```
                fvec.append(spkr_dir)

                logging.debug(fvec) # see the numbers [as if they make sense ]

                # write one row to the csv file
                melwriter.writerow(fvec)

    def _get_tdata(self, icsv):
        """ Returns the input and output example lists to be sent to an SVM
        classifier.
        """
        melv_list = []
        speaker_ids = []

        # build melv_list and speaker_ids lists
        with open(icsv, 'r') as icsv_handle:
            melreader = csv.reader(icsv_handle)

            for example in melreader:
                melv_list.append(map(float, example[:-1]))
                speaker_ids.append(example[-1])

        # and return them!
        return melv_list, speaker_ids

    def predict(self, soundclip):
        """ Recognizes the speaker in the sound clip. """

        sample_rate, data = wavfile.read(os.path.abspath(soundclip))
        ceps = mfcc(data, sample_rate)
        fvec = self._mfcc_to_fvec(ceps)

        speaker_id = self.recognizer.predict(fvec)[0]

        return self.spkr_iton[speaker_id]

if __name__ == "__main__":
    recoblock = RecoBlock("train_data")

    test_dir = os.path.abspath("test_data")
    testset_size = 0
    testset_error = 0

    for spkr_dir in os.listdir(test_dir):
        for soundclip in os.listdir(os.path.join(test_dir, spkr_dir)):
            clippath = os.path.abspath(os.path.join(test_dir, spkr_dir, soundclip))
            prediction = recoblock.predict(clippath)

            testset_size += 1
            if prediction != spkr_dir:
                testset_error += 1
                print "%s %s " % (prediction, u"[\u2717]")
            else:
                print "%s %s " % (prediction, u"[\u2713]")

    if testset_size == 0:
        print "No test data available."
    else:
        print "Error on test data: %.2f%%\n" % (testset_error / testset_size * 100)
```

testing

```
#!/usr/bin/env python2

from __future__ import division
import subprocess
from reco import RecoBlock

recognizer = RecoBlock("train_data")

def menu():
    global recognizer

    # print menu
    print "\n"
    menu_str = """Choose an option:
                    1) Train the recognizer again
                    2) Test the trained recognizer
                    3) Exit
                """
    print menu_str

    # get the user's choice now
    choice = int(raw_input())

    if choice == 1:
        # train a new recognizer
        recognizer = Recoblock("train_data")
    elif choice == 2:
        # record a voice now and find out the speaker
subprocess.call(["arecord","--device=plughw:1,0","--format","S16_LE","-d", "5s", "-r", "16000",
"_melcache/__test.wav"])
        spkr = recognizer.predict("_melcache/__test.wav")
        print "The speker must be: %s" % spkr
    elif choice == 3:
        exit(0)
    else:
        print "Invalid choicec.\n\n"
        menu()

if __name__ == "__main__":
    menu()
```

Pseudocode for fingerprint verification :

```
// Perform a single thinning iteration, which is repeated until the skeletization is finalized
void thinningIteration(Mat& im, int iter)
{
    Mat marker = Mat::zeros(im.size(), CV_8UC1);
    for (int i = 1; i < im.rows-1; i++)
    {
        for (int j = 1; j < im.cols-1; j++)
        {
            uchar p2 = im.at<uchar>(i-1, j);
            uchar p3 = im.at<uchar>(i-1, j+1);
            uchar p4 = im.at<uchar>(i, j+1);
```

```
            uchar p5 = im.at<uchar>(i+1, j+1);
            uchar p6 = im.at<uchar>(i+1, j);
            uchar p7 = im.at<uchar>(i+1, j-1);
            uchar p8 = im.at<uchar>(i, j-1);
            uchar p9 = im.at<uchar>(i-1, j-1);

            int A  = (p2 == 0 && p3 == 1) + (p3 == 0 && p4 == 1) +
                     (p4 == 0 && p5 == 1) + (p5 == 0 && p6 == 1) +
                     (p6 == 0 && p7 == 1) + (p7 == 0 && p8 == 1) +
                     (p8 == 0 && p9 == 1) + (p9 == 0 && p2 == 1);
            int B  = p2 + p3 + p4 + p5 + p6 + p7 + p8 + p9;
            int m1 = iter == 0 ? (p2 * p4 * p6) : (p2 * p4 * p8);
            int m2 = iter == 0 ? (p4 * p6 * p8) : (p2 * p6 * p8);

            if (A == 1 && (B >= 2 && B <= 6) && m1 == 0 && m2 == 0)
               marker.at<uchar>(i,j) = 1;
        }
    }

    im &= ~marker;
}

// Function for thinning any given binary image within the range of 0-255. If not you should first make sure that
your image has this range preset and configured!
void thinning(Mat& im)
{
 // Enforce the range tob e in between 0 - 255
    im /= 255;

    Mat prev = Mat::zeros(im.size(), CV_8UC1);
    Mat diff;

    do {
        thinningIteration(im, 0);
        thinningIteration(im, 1);
        absdiff(im, prev, diff);
        im.copyTo(prev);
    }
    while (countNonZero(diff) > 0);

    im *= 255;
}

int main( int argc, const char** argv )
{
    // Read in an input image - directly in grayscale CV_8UC1
    // This will be our test fingerprint
    Mat input = imread("/root/OpenCVBlueprints/chapter_6/datasets/fingerprints/DB1_B/101_1.tif",
IMREAD_GRAYSCALE);

    // Binarize the image, through local thresholding
    Mat input_binary;
    threshold(input, input_binary, 0, 255, CV_THRESH_BINARY_INV | CV_THRESH_OTSU);

    // Compare both
    Mat container(input.rows, input.cols*2, CV_8UC1);
    input.copyTo( container( Rect(0, 0, input.cols, input.rows) ) );
    input_binary.copyTo( container( Rect(input.cols, 0, input.cols, input.rows) ) );
    imshow("input versus binary", container); waitKey(0);
```

```
    // Now apply the thinning algorithm
    Mat input_thinned = input_binary.clone();
    thinning(input_thinned);

    // Compare both
    input_binary.copyTo( container( Rect(0, 0, input.cols, input.rows) ) );
    input_thinned.copyTo( container( Rect(input.cols, 0, input.cols, input.rows) ) );
    imshow("binary versus thinned", container); waitKey(0);

    // Now lets detect the strong minutiae using Haris corner detection
    Mat harris_corners, harris_normalised;
    harris_corners = Mat::zeros(input_thinned.size(), CV_32FC1);
    cornerHarris(input_thinned, harris_corners, 2, 3, 0.04, BORDER_DEFAULT);
    normalize(harris_corners, harris_normalised, 0, 255, NORM_MINMAX, CV_32FC1, Mat());

    // Select the strongest corners that you want
    int threshold_harris = 125;
    vector<KeyPoint> keypoints;

    // Make a color clone for visualisation purposes
    Mat rescaled;
    convertScaleAbs(harris_normalised, rescaled);
    Mat harris_c(rescaled.rows, rescaled.cols, CV_8UC3);
    Mat in[] = { rescaled, rescaled, rescaled };
    int from_to[] = { 0,0, 1,1, 2,2 };
    mixChannels( in, 3, &harris_c, 1, from_to, 3 );
    for(int x=0; x<harris_normalised.cols; x++){
      for(int y=0; y<harris_normalised.rows; y++){
        if ( (int)harris_normalised.at<float>(y, x) > threshold_harris ){
          // Draw or store the keypoint location here, just like you decide. In our case we will store the location of
the keypoint
          circle(harris_c, Point(x, y), 5, Scalar(0,255,0), 1);
          circle(harris_c, Point(x, y), 1, Scalar(0,0,255), 1);
          keypoints.push_back( KeyPoint (x, y, 1) );
        }
      }
    }
    imshow("temp", harris_c); waitKey(0);

    // Compare both
    Mat ccontainer(input.rows, input.cols*2, CV_8UC3);
    Mat input_thinned_c = input_thinned.clone();
    cvtColor(input_thinned_c, input_thinned_c, CV_GRAY2BGR);
    input_thinned_c.copyTo( ccontainer( Rect(0, 0, input.cols, input.rows) ) );
    harris_c.copyTo( ccontainer( Rect(input.cols, 0, input.cols, input.rows) ) );
    imshow("thinned versus selected corners", ccontainer); waitKey(0);

    // Calculate the ORB descriptor based on the keypoint
    Ptr<Feature2D> orb_descriptor = ORB::create();
    Mat descriptors;
    orb_descriptor->compute(input_thinned, keypoints, descriptors);

    // You can now store the descriptor in a matrix and calculate all for each image.
    // Since we just got the hamming distance brute force matching left, we will take another image and calculate
the descriptors also.
    // Removed as much overburden comments as you can find them above
    Mat input2 = imread("/root/OpenCVBlueprints/chapter_6/datasets/fingerprints/DB1_B/105_1.tif",
IMREAD_GRAYSCALE);
```

```
    Mat input_binary2;
    threshold(input2, input_binary2, 0, 255, CV_THRESH_BINARY_INV | CV_THRESH_OTSU);
    Mat input_thinned2 = input_binary2.clone();
    thinning(input_thinned2);
    Mat harris_corners2, harris_normalised2;
    harris_corners2 = Mat::zeros(input_thinned2.size(), CV_32FC1);
    cornerHarris(input_thinned2, harris_corners2, 2, 3, 0.04, BORDER_DEFAULT);
    normalize(harris_corners2, harris_normalised2, 0, 255, NORM_MINMAX, CV_32FC1, Mat());
    vector<KeyPoint> keypoints2;
    Mat rescaled2;
    convertScaleAbs(harris_normalised2, rescaled2);
    Mat harris_c2(rescaled2.rows, rescaled2.cols, CV_8UC3);
    Mat in2[] = { rescaled2, rescaled2, rescaled2 };
    int from_to2[] = { 0,0, 1,1, 2,2 };
    mixChannels( in2, 3, &harris_c2, 1, from_to2, 3 );
    for(int x=0; x<harris_normalised2.cols; x++){
      for(int y=0; y<harris_normalised2.rows; y++){
        if ( (int)harris_normalised2.at<float>(y, x) > threshold_harris ){
          // Draw or store the keypoint location here, just like you decide. In our case we will store the location of
the keypoint
          circle(harris_c2, Point(x, y), 5, Scalar(0,255,0), 1);
          circle(harris_c2, Point(x, y), 1, Scalar(0,0,255), 1);
          keypoints2.push_back( KeyPoint (x, y, 1) );
        }
      }
    }
    imshow("temp2", harris_c2); waitKey(0);
    Mat descriptors2;
    orb_descriptor->compute(input_thinned2, keypoints2, descriptors2);

    // Now lets match both descriptors
    // Create the matcher interface
    Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create("BruteForce-Hamming");
    vector< DMatch > matches;
    matcher->match(descriptors, descriptors2, matches);

    // Loop over matches and multiply
    // Return the matching certainty score
    float score = 0.0;
    for(int i=0; i < matches.size(); i++){
       DMatch current_match = matches[i];
       score = score + current_match.distance;
    }
    cerr << endl << "Current matching score = " << score << endl;

    return 0;

}
```

REFERENCES

[1] Tanzeem Choudhury, Brian Clarkson, Tony Jebara, Alex Pentland - "Multimodal Person Recognition using Unconstrained Audio and Video", ISSN 8115-2334.2014

[2] (Online) http://www.lsi.upc.edu/~nlp/papers/luque06b.pdf , Accessed May 1,2017

[3] (Online) ftp://tlp.limsi.fr/public/CHIL_book_pid-springer.pdf , Accessed May 1,2017

[4] (Online) https://link.springer.com/chapter/10.1007%2F3-540-44887-X_98?LI=true , Accessed May 1,2017

[5] (Online) http://ieeexplore.ieee.org/abstract/document/820041/ ,Accessed May 1,2017

[6] http://docs.opencv.org/3.0-beta/modules/face/doc/facerec/index.html , Accessed May 1,2017

[7] (Online) http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html , Accessed May 1,2017

[8] (Online) http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html , Accessed May 1,2017

[9] (Online) https://en.wikipedia.org/wiki/Corner_detection , Accessed May 1,2017

[10] (Online)http://docs.opencv.org/2.4/doc/tutorials/features2d/trackingmotion/harris_detector/harris_detector.html , Accessed May 1,2017

[11] (Online) http://www.willowgarage.com/sites/default/files/orb_final.pdf , Accessed May 1,2017

[12] (Online)http://docs.opencv.org/3.0beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html , Accessed May 1,2017

[13] (Online) https://en.wikipedia.org/wiki/Oriented_FAST_and_rotated_BRIEF , Accessed May 1,2017

[14] (Online) https://gilscvblog.com/2013/10/04/a-tutorial-on-binary-descriptors-part-3-the-orb-descriptor/ , Accessed May 1,2017

[15] (Online) https://en.wikipedia.org/wiki/Mel-frequency_cepstrum , Accessed May 1,2017

[16] (Online) http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/ , Accessed May 1,2017

[17] (Online)http://mirlab.org/jang/books/audioSignalProcessing/speechFeatureMfcc.asp?title=12-2%20MFCC , Accessed May 1,2017

[18] (Online) https://en.wikipedia.org/wiki/Support_vector_machine , Accessed May 1,2017

[19] (Online) http://scikit-learn.org/stable/modules/svm.html , Accessed May 1,2017

[20] (Online)http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html , Accessed May 1,2017

[21] (Online) https://beagleboard.org/black , Accessed May 1,2017

[22] (Online) https://www.logitech.com/en-in/product/hd-webcam-c270 , Accessed May 1,2017