EMBEDDED SYSTEM DESIGN (ECE-308)

SIXTH SEMESTER E&C MINI PROJECT REPORT

ON

FACE RECOGNITION BASED LOCKING SYSTEM

Submitted by

C Phaneendra Sarma – 120907262
Deep Chakraborty – 120907292
Anusuya Roy - 120907597

Under the guidance of

Name of the Teacher **C Sivananda Reddy**
Designation **Assistant Professor**

Department of Electronics and Communication Engineering
Manipal Institute of Technology, Manipal.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
MANIPAL INSTITUTE OF TECHNOLOGY
(A Constituent College of Manipal University)

ABSTRACT

Password Based Security systems are increasingly common these days in homes and offices. The necessity for secure locks is critical in today's information sensitive world. However, password or key combination based locks are extremely vulnerable as they can be bypassed very easily if someone gets access to the keycode, by hook or by crook. Hence, we propose a new and more secure form of biometric locking system using face recognition. Every human has a distinct biometric footprint that includes facial features, fingerprint, retina, voice, etc. Here, we have exploited one such modality that is the human face to determine whether a person is authorized to lock or unlock a door. The whole application has been made using the Raspberry Pi as an embedded system. It has been implemented as a way to replace traditional calling bells in houses to an automated 'owner unlock' / 'visitor alert' system.

Keywords: Face Recognition, OpenCV, Raspberry Pi, Security System

Contents

CHAPTER 1
INTRODUCTION

1.1 Overview

The project explores the territory of Face Recognition which has experienced an exponential growth in terms of research and integration into main stream technology. These algorithms provide endless ways to integrate from security of high tech facilities to gaming and unlocking your personal devices like laptops. These algorithms are becoming increasingly simple to implement and also requires so less system resources that they are making their way into small gadgets like smartphones. Face recognition is an exciting field of computer vision with many possible applications hardware and devices. OpenCV (Open Source Computer Vision) is a popular computer vision library started by Intel in 1999. OpenCV 2.4 now comes with the very new FaceRecognizer class for face recognition. The currently available algorithms are: Eigenfaces, Fisherfaces and Local Binary Pattern Histograms. Using embedded platforms like the Raspberry Pi and open source computer vision libraries like OpenCV we can now add face recognition to our project.

1.2 Project Description

Our project utilizes the security aspects of Face recognition. There is a camera near the entrance to the house. This camera is always on. It takes pictures of the person who wants to enter the house or wants to talk to the owner. People are classified into three groups with respect to the project: owner, friends, visitors. The door lock/unlock mechanism is implemented using a DC motor.

- **Owner:** The system recognizes the owner of the house and automatically unlocks the door.
- **Friend:** The system recognizes familiar faces and automatically unlocks the door if "DO-NOT-DISTURB" switch is turned off. If not then the system notifies the friend to come back later as the owner is busy.
- **Visitor:** The system recognizes the visitor and notifies the owner about the visitor. If the "DO-NOT-DISTURB" is ON then the system notifies the person to come back later. The main processing of the image and notifications are taken care by the Raspberry Pi.

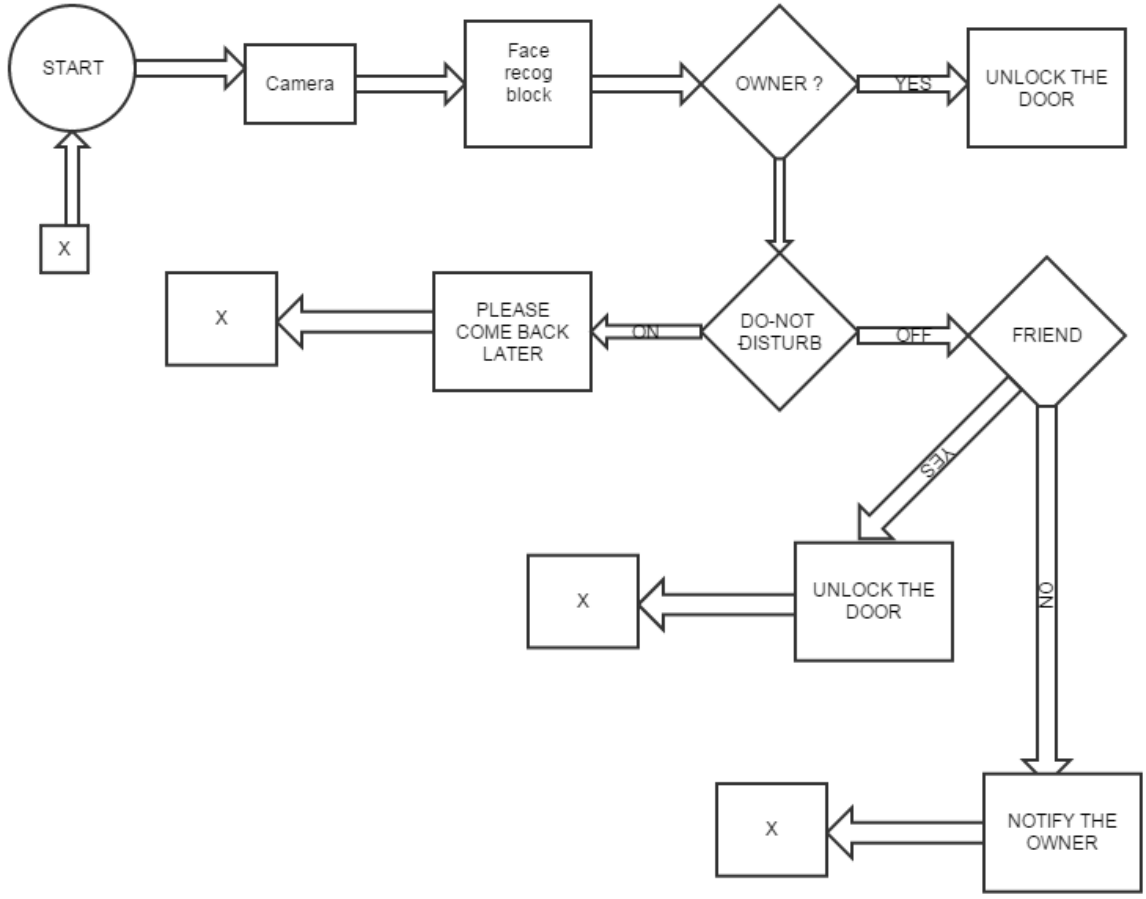
1.3 Hardware Description

Raspberry Pi 2 - Model B Technical Specification:

- Broadcom BCM2836 ARMv7 Quad Core Processor powered Single Board
- Computer running at 900MHz
- 1GB RAM
- 40pin extended GPIO
- 4 x USB ports
- 4 pole Stereo output and Composite video port
- Full size HDMI
- CSI camera port for connecting the Raspberry Pi camera
- DSI display port for connecting the Raspberry Pi touch screen display

CHAPTER 2
METHODOLOGY

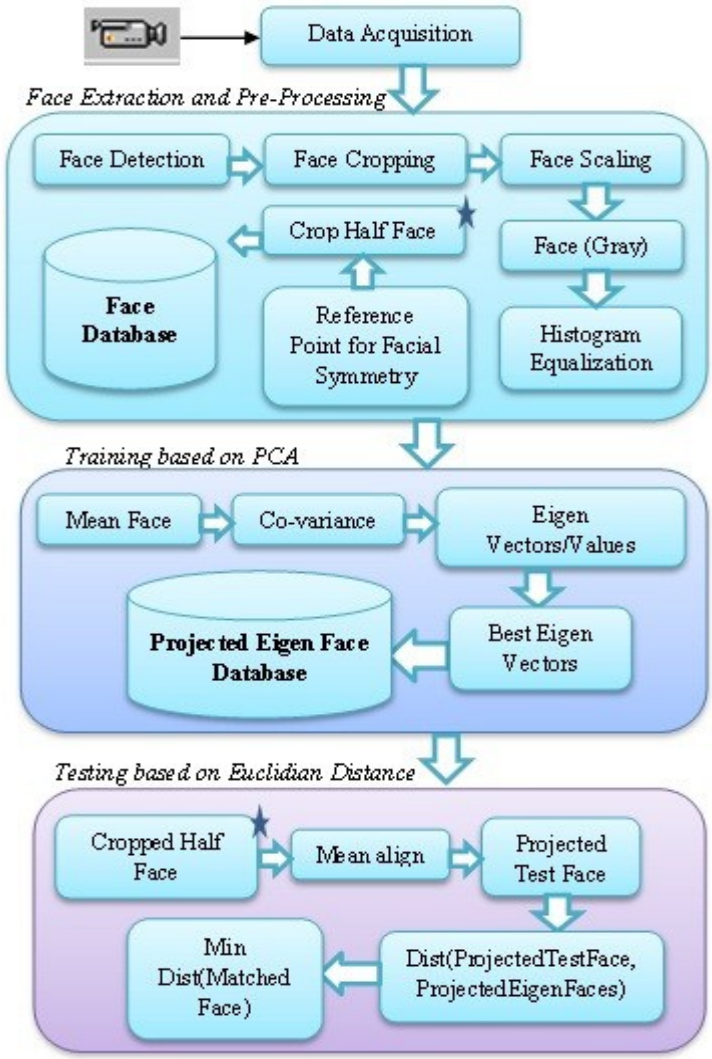
2.1 Block Diagram



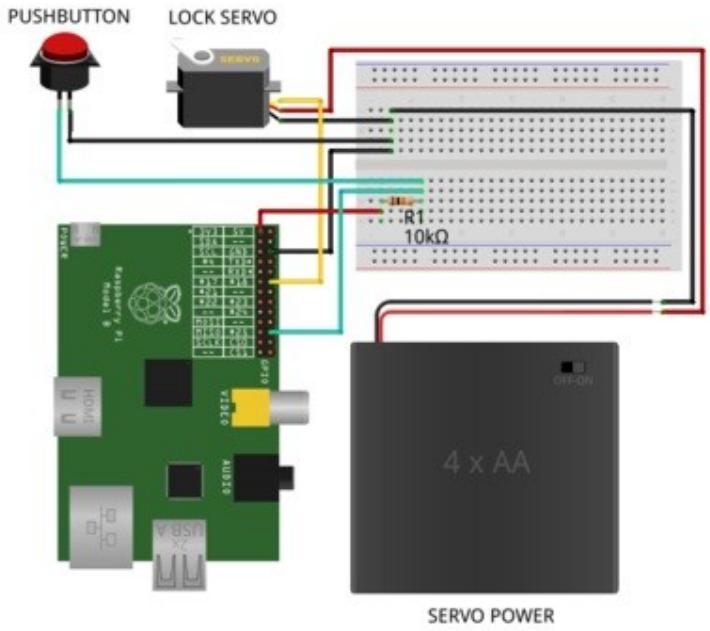
2.2 Eigenfaces Algorithm for Face Recognition

The face image representation has high dimensionality. For example, An image with 100x100 pixels lies in a 1000-dimensional image space. But all dimensions are not equally useful for us. We can only make a decision if there's any variance in data, so what we are looking for are the components that account for most of the information. The Principal Component Analysis (PCA) was independently proposed by Karl Pearson (1901) to turn a set of possibly correlated variables into a smaller set of uncorrelated variables. The idea is, that a high-dimensional dataset is often described by correlated variables and therefore only a few meaningful dimensions account for most of the information. The PCA method finds the directions with the greatest variance in the data, called principal components.

Eigenfaces Block Diagram



2.3 Circuit Diagram



2.4 Training

This project uses the eigenfaces algorithm in OpenCV to perform face recognition. To use this algorithm you'll need to create a set of training data with pictures of faces that are and are not allowed to open the box. Included in the project is a large set of face images that are tuned for training with the face recognition algorithm. This is the database of faces published by research AT&T Laboratories Cambridge in the mid 90's. These faces make up the set of negative images which represent faces that are not allowed to open the box. These images exist in the training/negative subdirectory of the project. To generate images of the owner, or positive training images, we use the capture.py script. This script will take pictures with the RPi hardware and write them to the training/positive sub-directory (which will be created by the script if it does not exist). If the script detects a single face, it will crop and save the image in the positive training images sub-directory. If the script can't detect a face or detects multiple faces, an error message will be displayed.

capture.py script

```
import glob
import os
import sys
import select
import cv2
import hardware
import config
import face


# Prefix for positive training image filenames.
POSITIVE_FILE_PREFIX = 'positive_'


def is_letter_input(letter):
      # Utility function to check if a specific character is
available on stdin.
      # Comparison is case insensitive.
      if select.select([sys.stdin,],[],[],0.0)[0]:
            input_char = sys.stdin.read(1)
            return input_char.lower() == letter.lower()
      return False


if __name__ == '__main__':
      camera = config.get_camera()
      door = hardware.Door()
      # Create the directory for positive training images if it
doesn't exist.
      if not os.path.exists(config.POSITIVE_DIR):
            os.makedirs(config.POSITIVE_DIR)
      # Find the largest ID of existing positive images.
      # Start new images after this ID value.
      files = sorted(glob.glob(os.path.join(config.POSITIVE_DIR,
            POSITIVE_FILE_PREFIX + '[0-9][0-9][0-9].pgm')))
      count = 0
      if len(files) > 0:
            # Grab the count from the last filename.
            count = int(files[-1][-7:-4])+1
      print 'Capturing positive training images.'
      print 'Press button or type c (and press enter) to capture an
image.'
      print 'Press Ctrl-C to quit.'
      while True:
            # Check if button was pressed or 'c' was received, then
capture image.
            if door.is_button_up() or is_letter_input('c'):
                  print 'Capturing image...'
                  image = camera.read()
```

```
                # Convert image to grayscale.
                image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
                # Get coordinates of single face in captured image.
                result = face.detect_single(image)
                if result is None:
                        print 'Could not detect single face!  Check
the image in capture.pgm' \
                                ' to see what was captured and try
again with only one face visible.'
                        continue
                x, y, w, h = result
                # Crop image as close as possible to desired face
aspect ratio.
                # Might be smaller if face is near edge of image.
                crop = face.crop(image, x, y, w, h)
                # Save image to file.
                filename     =     os.path.join(config.POSITIVE_DIR,
POSITIVE_FILE_PREFIX + '%03d.pgm' % count)
                cv2.imwrite(filename, crop)
                print  'Found  face  and  wrote  training  image',
filename
                count += 1
```

Once we've captured a number of positive training images, you can run the train.py script to perform the training. We will see a message that the training images were loaded, and the text `'Training model...'` to indicate the training calculations are being performed. Training the face recognition model on the Pi will take about 10 minutes. Once the training is complete we will see the message `'Training data saved to training.xml'`. The training data is now stored in the file training.xml, which will be loaded by the software to configure the face recognition model.

Face.py script

```
import cv2
import numpy as np
import config


haar_faces = cv2.CascadeClassifier(config.HAAR_FACES)


def detect_single(image):
      """Return bounds (x, y, width, height) of detected face in
grayscale image.
        If no face or more than one face are detected, None is returned.
      """
      faces = haar_faces.detectMultiScale(image,
                        scaleFactor=config.HAAR_SCALE_FACTOR,
                        minNeighbors=config.HAAR_MIN_NEIGHBORS,
                        minSize=config.HAAR_MIN_SIZE,
```

```
                        flags=cv2.CASCADE_SCALE_IMAGE)
        if len(faces) != 1:
                return None
        return faces[0]

def crop(image, x, y, w, h):
        """Crop box defined by x, y (upper left corner) and w, h
(width and height)
        to an image with the same aspect ratio as the face training data.
Might
        return a smaller crop if the box is near the edge of the image.
        """
        crop_height = int((config.FACE_HEIGHT / float(config.FACE_WIDTH))
* w)
        midy = y + h/2
        y1 = max(0, midy-crop_height/2)
        y2 = min(image.shape[0]-1, midy+crop_height/2)
        return image[y1:y2, x:x+w]

def resize(image):
        """Resize a face image to the proper size for training and
detection.
        """
        return cv2.resize(image,
                                (config.FACE_WIDTH, config.FACE_HEIGHT),
                                interpolation=cv2.INTER_LANCZOS4)
```

Train.py script

```
import fnmatch
import os
import cv2
import numpy as np
import config
import face

MEAN_FILE = 'mean.png'
POSITIVE_EIGENFACE_FILE = 'positive_eigenface.png'
NEGATIVE_EIGENFACE_FILE = 'negative_eigenface.png'

def walk_files(directory, match='*'):
        """Generator function to iterate through all files in a
directory recursively
        which match the given filename match parameter.
        """
        for root, dirs, files in os.walk(directory):
                for filename in fnmatch.filter(files, match):
                        yield os.path.join(root, filename)

def prepare_image(filename):
        """Read an image as grayscale and resize it to the appropriate
size for
```

```
        training the face recognition model.
        """
        return face.resize(cv2.imread(filename, cv2.IMREAD_GRAYSCALE))

def normalize(X, low, high, dtype=None):
        #Normalizes a given array in X to a value between low and
high.
        X = np.asarray(X)
        minX, maxX = np.min(X), np.max(X)
        # normalize to [0...1].
        X = X - float(minX)
        X = X / float((maxX - minX))
        # scale to [low...high].
        X = X * (high-low)
        X = X + low
        if dtype is None:
                return np.asarray(X)
        return np.asarray(X, dtype=dtype)

if __name__ == '__main__':
        print "Reading training images..."
        faces = []
        labels = []
        pos_count = 0
        neg_count = 0
        # Read all positive images
        for filename in walk_files(config.POSITIVE_DIR, '*.pgm'):
                faces.append(prepare_image(filename))
                labels.append(config.POSITIVE_LABEL)
                pos_count += 1
        # Read all negative images
        for filename in walk_files(config.NEGATIVE_DIR, '*.pgm'):
                faces.append(prepare_image(filename))
                labels.append(config.NEGATIVE_LABEL)
                neg_count += 1
        print 'Read', pos_count, 'positive images and', neg_count,
'negative images.'

        # Train model
        print 'Training model...'
        model = cv2.createEigenFaceRecognizer()
        model.train(np.asarray(faces), np.asarray(labels))

        # Save model results
        model.save(config.TRAINING_FILE)
        print 'Training data saved to', config.TRAINING_FILE

        # Save mean and eignface images which summarize the face
recognition model.
        mean = model.getMat("mean").reshape(faces[0].shape)
        cv2.imwrite(MEAN_FILE, normalize(mean, 0, 255, dtype=np.uint8))
        eigenvectors = model.getMat("eigenvectors")
```

```
        pos_eigenvector = eigenvectors[:,0].reshape(faces[0].shape)
        cv2.imwrite(POSITIVE_EIGENFACE_FILE, normalize(pos_eigenvector,
0, 255, dtype=np.uint8))
        neg_eigenvector = eigenvectors[:,1].reshape(faces[0].shape)
        cv2.imwrite(NEGATIVE_EIGENFACE_FILE, normalize(neg_eigenvector,
0, 255, dtype=np.uint8))
```

Eigenfaces generated from my training data:


Mean Image


Negative Eigenface Image


Positive Eigenface Image

2.5 Usage

We should see the door.py script run and load and the message 'Loading training data...' appear. It should take a few minutes for the training data to load and the following message to appear:

```
Running box...
Press button to lock (if unlocked), or unlock if the correct face is
detected.
Press Ctrl-C to quit.
```

As we aim the pi camera at our face, and press the button, we should see a message that the button was pressed, and after a moment if the face is recognized we will see a message like:

```
Predicted POSITIVE face with confidence 1321.35253959 (lower is more
confident). Recognized face!
```

If the face is not recognized you will see a message like:

```
Predicted NEGATIVE face with confidence 3987.76625152 (lower is more
confident). Did not recognize face!
```

If a single face couldn't be detected (because none is visible, or there are multiple faces detected) an error message will be displayed like with the training script. When a face is found, we'll notice the prediction message includes how the face was recognized (i.e. matching either the positive or negative training data), and the confidence of the recognition. The confidence is a value like 1321.35 or 3987.77 above and represents the distance of the captured face from the predicted face – a lower distance value means the captured face is more similar to the predicted face. To unlock the door, a captured face must be a positive

prediction with a confidence value below the POSITIVE_THRESHOLD configuration value. By default the POSITIVE_THRESHOLD is 2000, so the positive prediction with confidence 1321.35 was considered a match and would have unlocked the door.

Door.py script

```
import pygame.mixer
from time import sleep

import cv2
import select
import sys
import config
import face
import hardware

pygame.mixer.init(48000, -16, 1, 1024)
soundA = pygame.mixer.Sound("/home/pi/my_code/welcome.wav")
soundB = pygame.mixer.Sound("/home/pi/my_code/wait.wav")
soundC = pygame.mixer.Sound("/home/pi/my_code/later.wav")

soundChannelA = pygame.mixer.Channel(1)
soundChannelB = pygame.mixer.Channel(2)
soundChannelC = pygame.mixer.Channel(3)

def is_letter_input(letter):
      # Utility function to check if a specific character is
available on stdin.
      # Comparison is case insensitive.
      if select.select([sys.stdin,],[],[],0.0)[0]:
            input_char = sys.stdin.read(1)
            return input_char.lower() == letter.lower()
      return False

def main():
      # Load training data into model
      print 'Loading training data...'
      model = cv2.createEigenFaceRecognizer()
      model.load(config.TRAINING_FILE)
      print 'Training data loaded!'
      # Initialize camer and box.
      camera = config.get_camera()
      door = hardware.Door()
      # Move box to locked position.
      door.lock()
      print 'Running Lock...'
```

```
      print 'Press button to lock (if unlocked), or unlock if the
correct face is detected.'
      print 'Press Ctrl-C to quit.'
      while True:
            try:
                  # Check if capture should be made.
                  # TODO: Check if button is pressed.
                  if door.is_button_up() or is_letter_input('l'):
                        if not door.is_locked:
                              # Lock the door if it is unlocked
                              door.lock()
                              print 'Door is now locked.'
                        else:
                              print 'Button pressed, looking for
face...'
                              # Check for the positive face and unlock
if found.
                              image = camera.read()
                              # Convert image to grayscale.
                              image = cv2.cvtColor(image,
cv2.COLOR_RGB2GRAY)
                              # Get coordinates of single face in
captured image.
                              result = face.detect_single(image)
                              if result is None:
                                    print 'Could not detect single
face!  Check the image in capture.pgm' \
                                          ' to see what was captured
and try again with only one face visible.'
                                    soundChannelC.play(soundC)
                                    sleep(.01)
                                    continue
                              x, y, w, h = result
                              # Crop and resize image to face.
                              crop = face.resize(face.crop(image, x, y,
w, h))
                              # Test face against model.
                              label, confidence = model.predict(crop)
                              print 'Predicted {0} face with confidence
{1} (lower is more confident).'.format(
                                    'POSITIVE' if label ==
config.POSITIVE_LABEL else 'NEGATIVE',
                                    confidence)
                              if label == config.POSITIVE_LABEL and
confidence < config.POSITIVE_THRESHOLD:
                                    print 'Recognized face! Unlocking
Door Now...'
                                    door.unlock()
```

```
                                    soundChannelA.play(soundA)
                                    sleep(.01)
                              else:
                                    print 'Did not recognize face!'
                                    soundChannelB.play(soundB)
                                    sleep(.01)
            except KeyboardInterrupt:
                  door.clean()
                  sys.exit()

if __name__ == '__main__':
      main()
```

Hardware Script

```
import time
import numpy as np
import cv2
import RPi.GPIO as GPIO

import picam
import config
import face

class Door(object):
      """Class to represent the state and encapsulate access to the
hardware of
      the treasure box."""
      def __init__(self):
            # Initialize lock servo and button.
            GPIO.setmode(GPIO.BCM)
            GPIO.setup(config.LOCK_SERVO_PIN, GPIO.OUT)
            GPIO.setup(config.BUTTON_PIN, GPIO.IN)
            self.servo = GPIO.PWM(config.LOCK_SERVO_PIN, 50)


            # Set initial box state.
            self.servo.start(2.5)
            self.button_state = GPIO.input(config.BUTTON_PIN)
            self.is_locked = None

      def lock(self):
            """Lock the box."""
            self.servo.ChangeDutyCycle(2.5)
            self.is_locked = True

      def unlock(self):
```

```
            """Unlock the box."""
            self.servo.ChangeDutyCycle(12.5)
            self.is_locked = False

      def clean(self):
            self.servo.stop()
            GPIO.cleanup()

      def is_button_up(self):
            """Return True when the box button has transitioned from
down to up (i.e.
            the button was pressed)."""
            old_state = self.button_state
            self.button_state = GPIO.input(config.BUTTON_PIN)
            # Check if transition from down to up
            if old_state == config.BUTTON_DOWN and self.button_state ==
config.BUTTON_UP:
                  # Wait 20 milliseconds and measure again to debounce
switch.
                  time.sleep(20.0/1000.0)
                  self.button_state = GPIO.input(config.BUTTON_PIN)
                  if self.button_state == config.BUTTON_UP:
                        return True
            return False
```

CHAPTER 3
RESULTS

3.1 Success and Limitations

We have successfully constructed a face recognition based security system / door locking/unlocking mechanism which unlocks to the owner of the house and friends, whereas notifies the owner of any unknown visitors. Also a relatively complex computer vision system has been deployed with ease on a low cost, low power, and high efficiency embedded system such as the Raspberry Pi and has therefore cut costs compared to commercial security systems whilst maintaining comparable accuracy and open source design.

The Eigenfaces algotithm used here provides an easy and cheap way to realize face recognition in that:

- Its training process is completely automatic and easy to code.
- Eigenface adequately reduces statistical complexity in face image representation.
- Once eigenfaces of a database is calculated, face recognition can be achieved with real time.
- Eigenface can handle large databases.

However, the deficiencies of eigenface method are also obvious:

- Very sensitive to lighting, scale and translation, requires a highly controlled environment.
- Eigenface has difficulty to capture expression changes.
- The most significant eigenfaces are mainly about illumination encoding, doesn't provide useful information regarding the actual face.

3.2 Future Work

The following possibilities can be looked into in future iterations of this project:

- Investigate using other face recognition algorithms in OpenCV such as Fisherfaces which is more robust to varied lighting and other conditions.
- Have the Pi send you an email with the picture of whoever is trying to unlock the door.
- Add a microphone and look at adding speech recognition as a means of unlocking the door.

REFERENCES

- http://www.adafruit.com/learn
- Face Recognition with OpenCV, a tutorial: http://docs.opencv.org/modules/contrib/doc/facerec/facerec_tutorial.html
- http://makezine.com/category/electronics/raspberry-pi/
- http://www.face-rec.org
- Adrian Rosebrock – Practical Python and OpenCV, Case Studies
- OpenCV Python Tutorials – https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_tutorials.html
- Maik Schmidt – Raspberry Pi: A Quick Start guide

Student details:

Name	Deep Chakraborty	C Phaneendra Sarma	Anusuya Roy
Reg.No & section	120907292 - D	120907262 - D	120907597 - D
Contact No	7406170450	9573379533	7411135408
Email ID	deepc94@gmail.com	suphalasarma@gmail.com	anushkaaroy@gmail.com