Farnell
AN AVNET COMPANY

Machine Vision Door Entry System with Raspberry Pi

Index

Overview 2

Technology overview - Face recognition 3

Project kit 6

What you can learn from this kit 7

Hardware setup 8

Software requirements 8

Instructions 9

References 13

How to scale this kit in real work application 14

Common problems with real work application 14

Overview

Facial recognition is the biggest advancement in security systems since the introduction of CCTV cameras. It's a technology capable of matching a human face from a digital image or a video frame against a database of faces, typically employed to authenticate users through ID verification services, works by pinpointing and measuring facial features from a given image.

Facial recognition technology (FRT) has materialized as a nubile solution to solve many contemporary requirements for verification and identification of identity claims. It assembles the promise of other biometric systems, which ventures to secure identity to individual discrete body features

FRT has demonstrated effectiveness. It is highly successful comparatively smaller populations residing in controlled environments, for verification of identity claims, where an image of a person's face is matched to an "on-file" pre-existing image and tied to the claimed identity (said verification task). FRT, however, betrays lackluster performance in complex attempts to identify persons who are not keen to self-identify. The FRT, in such cases, strives to match a person's face with any "on-file" image (identification task). Such happenstance is the "face in the crowd" scenario, where a face is plucked from a crowd in a chaotic environment. This is unlikely to be an operational reality in the foreseeable future.

FRT only recognizes a face if a particular person's face is advance added (enrolled) in the system. Enrolment conditions—voluntary or not—and the resulting image quality (gallery image) sign notably impacts FRT's final success. Image quality holds high significance in FRT overall performance.

Performance depends on several other ascertained factors, such as:

- **Environment:** Better FRT performance relates to increased similar image environments to be compared (background, head size and its orientation, camera distance, and lighting conditions).
- **Image age:** FRT performs better if less time has elapsed between the compared images.
- **Consistent camera use:** FRT exhibits better performance if the optical characteristics of the said enrolment process camera and the obtained on-site image (light intensity, focal length, and color balance to name a few) are similar.
- **Gallery size:** Since the number of possible images which enter the gallery as near-identical mathematical representations (biometric doubles) and increases with bigger gallery size, limiting the gallery size in "open set" identification applications (like watch list applications) may assist system integrity maintenance and boost overall performance.

Technology overview – Face recognition

Face recognition is the technology which involves the understanding of how the faces are recognized by biological systems and how this can be emulated by computer systems. Its one of the few biometric methods that possess the merits of both high accuracy and low intrusiveness.

This section covers how contemporary face recognition works! To do this, we will push this tech to solve a challenging problem — distinguish **Mark Wahlberg** from **Matt Damon**.

Fig: Mark Wahlberg (left) Matt Damon(Right)

Face recognition is a series of multiple related problems:

1. Look at the picture understudy and find the faces
2. Focus on an individual face and try to comprehend even if that particular face is in a strange weird direction or illuminated by bad lighting, it continues to be the same person.
3. Select the face's unique features. These features differentiate that particular individual from others. These can be big eyes, a longer face, and the like.
4. Finally, compare the face's distinctive features to all known people to determine the individual's name.

There is a need to create a pipeline where each step of face recognition is solved separately and pass the result to the succeeding step. The essence is to chain multiple machine learning (ML) algorithms.

Face recognition steps

We will steadily solve this multiple step problem. We'll learn about a different ML algorithm in each step. You'll learn the principal ideas behind each algorithm and how you can create your facial recognition system in Python.

Step 1: Finding faces

Face detection is the first step in the pipeline. Any modern camera user has seen face detection in action:

Fig: Face detection in phones

Face detection is a winning camera feature. Since the camera automatically determines faces, it ensures that all faces are in focus before the final shot. However, we have a different motive — find the image areas we will pass to the subsequent step in the pipeline.

To locate faces in our image, we'll make it black and white as we don't require color data to ascertain faces:

Fig: Converting image to gray scale

Then we'll study individually every pixel in our image. We will look at the pixels directly surrounding the pixel under observation.

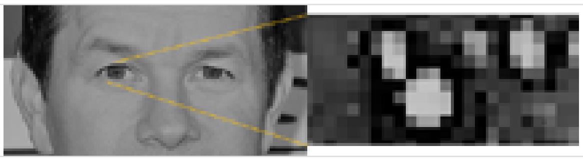


Fig: Pixel observation

We aim to figure out how dim the current pixel is in contrast to the pixels encircling it. Then we will draw an arrow showing the direction where the image gets darker:

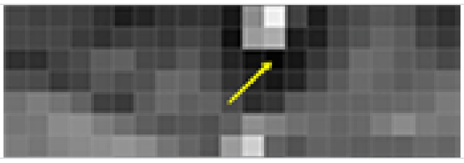


Fig: Gradients

The repetition of this process for every single pixel in the image culminates with every pixel substituted by an arrow. Such arrows are term gradients and they show the flow from light to dark across the complete image.

There's an excellent reason for substituting pixels with gradients. If pixels are directly analyzed, really light images and dark images of the same individual will come to different pixel values. However, if you only consider the direction of changes in brightness, both bright images and dark images will have identical representation. The problem is thus much easier to solve!

Saving the gradient for each pixel provides us excessive detail. We simply need to see the fundamental flow of lightness or darkness at a higher level to ascertain the image's fundamental pattern. We'll splinter the image into small 16x16 pixels squares. We'll count in each square the number of gradients that point in each major direction. We'll then substitute that square in the image with the strongest arrow directions.

The result is that we turn the original image into an extremely simple representation that successfully captures the primary face structure:

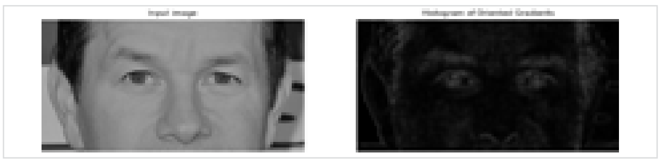
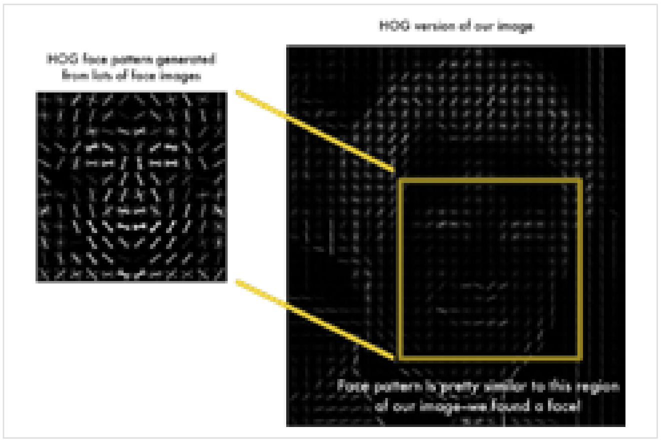


Fig: HOG representation of image

The original image is made into a HOG representation that expresses the major features of the image independent of image brightness.


HOG version of our image
HOG face pattern generated from lots of face images
Face pattern is pretty similar to this region of our image—we found a face!

Step 2: Posing and projecting faces

You have isolated the faces in the image. Now you must tackle the problem of faces turned in different directions which appear different to a computer:

To compensate, you must warp each picture so the eyes and lips always remain in the image sample place. Face comparison will now be much easier in the next steps. An algorithm named **face landmark estimation** is used to achieve this.

The fundamental idea is to arrive with 68 specific points (aka landmarks) that every face has — outside edge of each eye, and top of the chin. Then you will train an ML algorithm to locate these 68 face specific points:

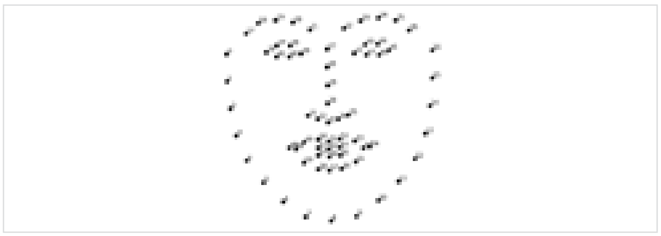


Fig: 68 Landmarks on face

Here's the result of finding the 68 face landmarks on the test image:

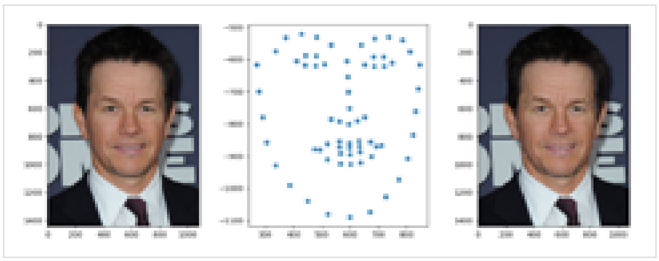


Fig: 68 Landmarks on actual image

Step 3: Encoding faces

Now we are in the crux of the problem — actually figuring faces apart. This is the exciting part!

The simplest approach to recognize a face is to directly compare the unknown face we discovered in Step 2 with all the pictures of people already tagged. Finding a previously tagged face that appears similar to the unknown face establishes it to be the same person. Seems legit, right?

That approach brings an insurmountable problem. A site like Facebook with billions of subscribers and trillions of photos cannot possibly swing through every previous-tagged face to compare it to any newly uploaded picture. This would take a lot of time. Faces must be recognized in milliseconds, not days.

What we require is a technique to obtaining a few basic measurements from each face. We may then measure the unknown face the same way and locate the known face having the closest measurements. To give an example, we may measure individual ear size, nose length, and spacing between eyes.

Researchers discovered that the best approach is to allow the computer to figure out the measurements which it will itself collect. Deep learning (DL) executes a better job compared to humans when it comes to figuring out the parts of a face that should be measured.

The solution? Train a Deep Convolutional Neural Network. The trick is to train it to produce 128 measurements for each face.

The training process functions by simultaneously analyzing 3 face images:

1. Load training face image of any known person
2. Load another picture of the same known person
3. Load a picture of a completely different person

The neural network, after repeating this step millions of times corresponding to millions of images of thousands of individuals, the neural network subsequently learns to reliably generate 128 measurements for an individual. Any ten different pictures of the same person should give approximately identical measurements.

ML professionals term a face's 128 measurements as **embedding**.

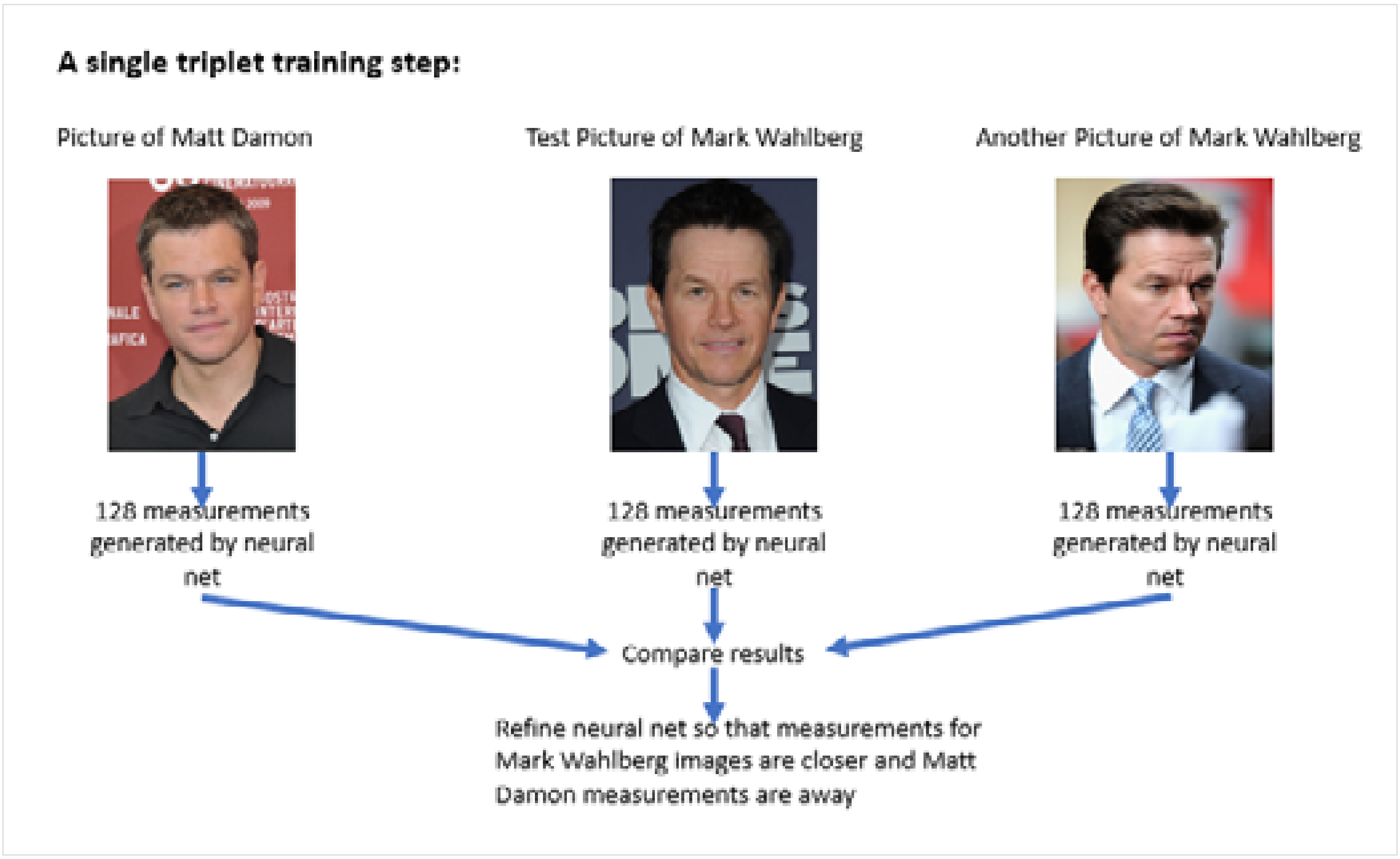
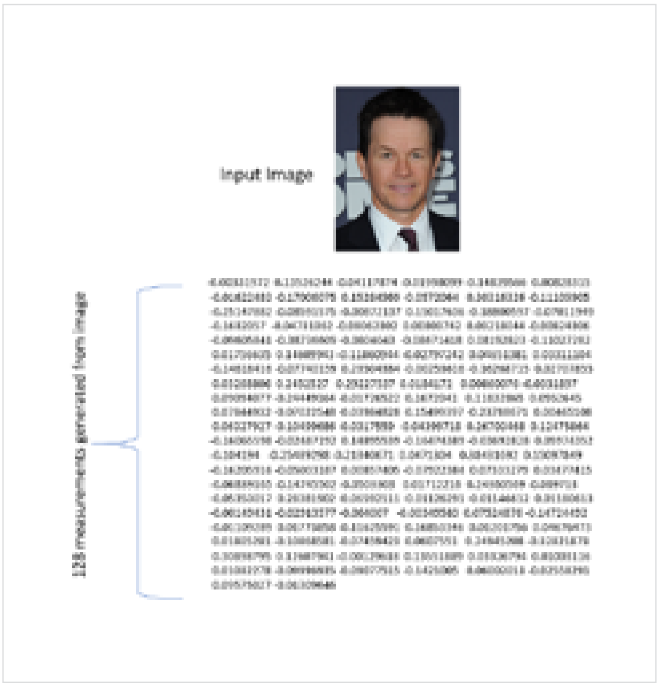

A single triplet training step:
Picture of Matt Damon
Test Picture of Mark Wahlberg
Another Picture of Mark Wahlberg
128 measurements generated by neural net
128 measurements generated by neural net
128 measurements generated by neural net
Compare results
Refine neural net so that measurements for Mark Wahlberg images are closer and Matt Damon measurements are away

Fig: Triplet training

Encoding our face image

Training a convolutional neural network to generate face embedding needs substantial data and plentiful computer power.

You simply have to run the face images through the pre-trained network to obtain the 128 measurements fora face. The measurements for our test image are:



```
[illegible]
```

Fig: 128 measurements of face

Step 4: Finding the person’s name from the encoding

This last step is the easiest in the complete process. All we simply have to do is find the person in our database of known people who enjoy the closest measurements to our test image.

Project kit

We have put a solution together to help implement face recognition technology with the help of Raspberry Pi. It detects the faces in an image, identify key facial features, and get the contours of detected faces.

This solution offers facial recognition in real-time which makes it an ideal solution for an indoor security applications.

The images must be processed before actual recognition. An example of processing is converting an image to grey. We will use Open CV to process an image in this project.

OpenCV (Open Source Computer Vision Library) was built to accelerate machine perception use in commercial products and offer a common computer vision applications infrastructure.

OpenCV targets real-time computer vision. It finds principal use in image-related operations and assists in the following functions:

- Face detection and its features.
- Detecting shapes like Circles and rectangles in an image. For example, finding a picture of a coin in an image.
- Read images and Write images.
- Recognizing text in images, like reading number plates

OpenCV use brings a few advantages:

- Easy to learn since a large number of tutorials are readily available
- Works with nearly all major languages.
- Can be used free of cost

The solution requires Raspberry Pi 4, Raspberry Pi High-Quality Camera, and PIFACE DIGITAL 2 board. The Raspberry Pi 4 is a series of small single-board computers with a 1.5 GHz 64-bit quad-core ARM Cortex-A72 processor, on-board 802.11ac Wi-Fi, Bluetooth 5, full gigabit Ethernet, two USB 2.0 ports, two USB 3.0 ports, and dual-monitor support via a pair of micro Type D HDMI ports for up to 4K resolution. The Pi 4 is powered via a USB-C port.

Raspberry Pi camera consists of 12.3 megapixel Sony IMX477 sensor which is based on back-illuminated sensor architecture, with adjustable back focus and support for C- and CS-mount lenses.

PiFace Digital 2 is designed to plug on to the GPIO of your Raspberry Pi B+, allowing you to sense and control the real world. With PiFace Digital 2 you can detect the state of a switch. You can drive outputs to power motors, actuators, LEDs, or anything you can imagine.

Hardware Required

I. Raspberry Pi 4 model B 4 GB

II. Raspberry Pi 4 power supply

III. Micro SD card

IV. Raspberry Pi high-quality camera

V. Solenoid lock, 12VDC, 18W

VI. CLB-JL53 connector adapter

VII. PEL00398 AC/DC power supply

VIII. Relay board (PiRelay board)

What you can learn from this kit

This solution is ideal for enthusiasts and hobbyists interested in Machine Learning (ML) applications on a single board computer (SBC) Raspberry Pi with Raspbian operating system. From a hardware point of view, it elaborates on how a relay works and interface with an SBC. You will understand the technology behind facial recognition and can use this technology for your future projects.

The solution offers various easy to use ML capabilities and will help you to develop various AI apps.

The Kit sample includes face detection and you can perform the following using this kit:

- **Image classification**
  Elements in the image classification service are classified into intuitive categories (people, activities, artwork, objects, or environments) to specify image themes and their respective application scenarios. This service is used to create apps that identify, manage, and classify images.

- **Object detection and tracking**
  This service can detect multiple objects and track them in an image so that they can be located and classified in real-time. Images are recognized and examined using this feature. The ML Kit detects objects in each image, and acquire in real-time their respective locations within that image. Apps can be developed to filter away unwanted objects.

- **Face detection**
  This service discerns 3D and 2D face contours. The 2D face detection skillfully detects aspects of the user's face, inclusive of facial expression, wearing, age, and gender. The 3D face detection ability extracts information like face keypoint coordinates, face angle, and 3D projection matrix. Apps that dynamically beautify a users' face during video calls use this feature.

- **Text recognition**
  This text recognition service acquires text from images of documents, receipts, and business cards. It is widely used in transit, office, and education apps. You can extract text from a photo in a translation app and subsequently translate the text, thus improving the user experience.

  Such a service can run on a device or the cloud. The supported languages, however, differ in these two scenarios. The on-device API recognizes text in Korean, Simplified Chinese, Japanese, and Latin-based languages (including English, German, Spanish, Russian, Portuguese, and special characters). In contrast, the on-cloud API recognizes text in Spanish, Indonesian, Portuguese Simplified Chinese, English, Finnish, Norwegian, Swedish, Italian, German, French, Russian, Turkish, Thai, Arabic Japanese, Korean, Polish, Danish, and Hindi.

- **Document recognition**
  This service recognizes text in document images with paragraph formats. Paper documents can thus be changed into electronic copies, with considerable improvement in the information input efficiency and reduction of labor costs.

Hardware setup

Connect the Raspberry Pi camera to the Raspberry Pi board. Mount PiFace Digital 2 board to the Raspberry Pi 4.

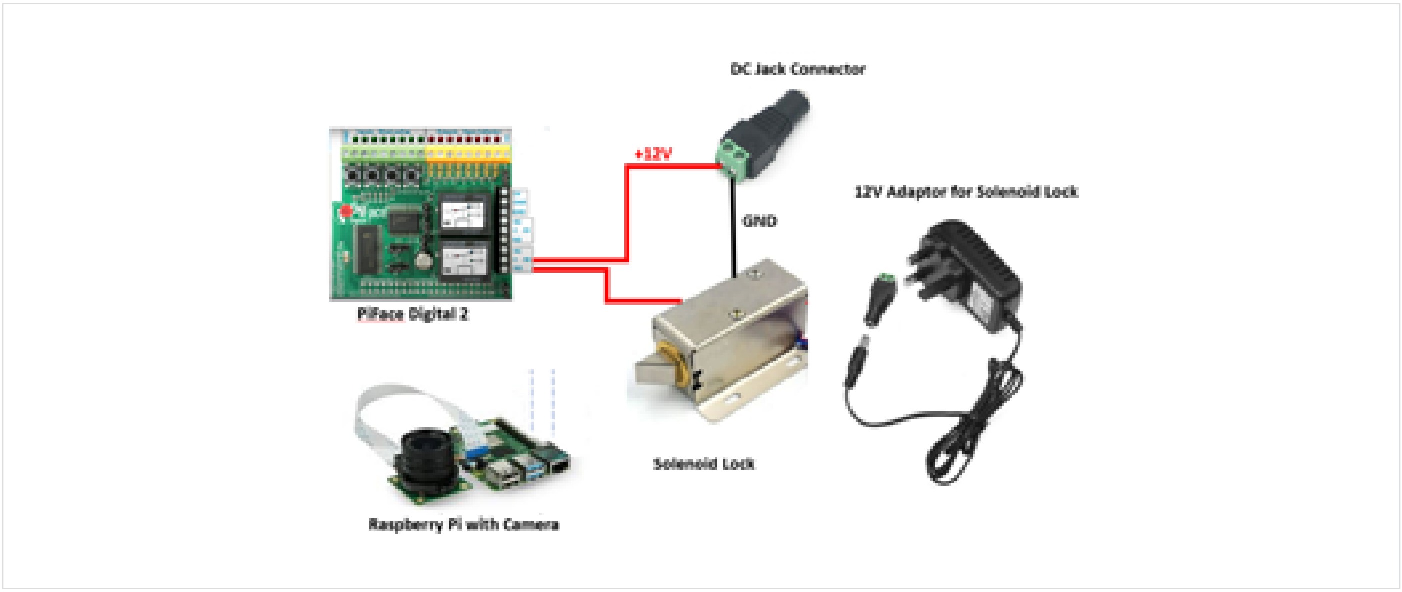


Fig: Hardware setup

The Raspberry Pi output is either 5V or 3.3V, but our solenoid Lock require 12V to operate So we need relay to operate the Solenoid Lock . The relay input is 5V and it is mounted on PiFace Digital2 board, so there is no need for external power supply once the PiFace Digital 2 is mounted on it. After mounting the board, connect red wire from DC jack connector to the common Pin (C) of the relay 0 (R0) mounted on the PiFace digital 2 board as shown in the figure. Connect red wire of the Solenoid Lock to the normally open pin (NO) of the relay. Connect black wire from the solenoid lock which is a ground pin (GND) to the DC jack connector. Finally connect DC jack connector to the 12V Adaptor.

Required cables & tools:

To connect load with relay, hook up wire are used as per load requirement and to make connections, tools like wire cutter and screwdriver are used. Click the following links to make your choice.

Hook up wire
grh.premierfarnell.com/pageredir.aspx?c=EU1&u=/c/cable-wire-cable-assemblies/hook-up-wire?ost=cable

Wire cutter
grh.premierfarnell.com/pageredir.aspx?c=EU1&u=/c/tools-production-supplies/tools-hand-workholding/cutters/prl/results?st=wire%20cutter

Stripping tools
grh.premierfarnell.com/pageredir.aspx?c=EU1&u=/c/tools-production-supplies/tools-hand-workholding/stripping-tools/prl/results?st=wire%20cutter

Screw drive
grh.premierfarnell.com/pageredir.aspx?c=EU1&u=/c/tools-production-supplies/tools-hand-workholding/screwdrivers/phillips-screwdrivers

Network requirements: Wi-Fi with internet

Software requirements

Raspbian operating system for Raspberry Pi Board. The operating system can be downloaded from the below link.

www.raspberrypi.org/downloads/

Required libraries:

OpenCV, Facial_recognition, Dlib

Instructions

Preparing the Raspberry Pi board:

Load a bootable image of Raspbian OS in SD card which should be at least 16GB. Follow the below link to prepare a bootable SD card with Raspbian OS.

www.raspberrypi.org/documentation/installation/installing-images/README.md

Installing necessary libraries:

Open terminal by pressing Ctrl+Alt+T. A new terminal window will open type below commands to install the libraries

Now let us first update the repository and update our operating system to the latest.

```
sudo apt-get update && sudo apt-get upgrade
```

We will now install Cmake and build-essential. CMake is an open-source and free cross-platform software tool for managing the software build process using the compiler-independent method. Build-essential comprises a package containing references to multiple packages required for building general software. Such libraries are a must to build openCV from source.

```
sudo apt-get install build-essential cmake pkg-
config
```

There are many different image files format we generally use for image files most common are JPEG, TIFF, PNG, BMP etc. we will install libraries which helps in reading and writing images to the disk.

```
sudo apt-get install libjpeg-dev libtiff5-dev
libjasper-dev libpng-dev
```

We will now install libraries for reading and writing video files. The libavcodec-dev encode and decode audio and video data and Libavformat- a demuxer library –supports a majority of the existing file formats. The libswscale-dev finds use in scaling video files. libv4l is a library collection adding a thin abstraction layer on top of the video4linux2 devices. The function of such a (thin) layer is to help application writers so that they can easily support diverse devices without the need to write separate code for separate devices within the same class. The libxvidcore refers to an open-source MPEG-4 video codec, implementing the MPEG-4 Simple Profile. The libx264 denotes an advanced encoding library used for the creation of H. 264 (MPEG-4 AVC) video streams.

```
sudo apt-get install libavcodec-dev libavformat-
dev libswscale-dev libv4l-dev
```

```
sudo apt-get install libxvidcore-dev libx264-dev
```

Now let us install libraries which supports graphical user function. GTK+ is a multi-platform toolkit for creating graphical user interfaces

```
sudo apt-get install libgtk2.0-dev libgtk-3-dev
```

Fonts libraries will now be installed. Fontconfig refers to font customization, configuration, and c library, independent of the X Window System. It is specifically designed to find fonts within the system and subsequently selects them as per application-specific requirements. The multi-platform library Cairo offers anti-aliased vector-based rendering for sundry target backends.

```
sudo apt-get install libfontconfig1-dev
libcairo2-dev
```

We will now install libraries that help to render text. Pango is a layout library and rendering text, with a priority on internationalization. It can be used anywhere where the text layout is a must.

```
sudo apt-get install libgdk-pixbuf2.0-dev
libpango1.0-dev
```

Libraries are installed to bank scientific data. Large model files applied in ML are stored in libraries. HDF5 is a library and file format for amassing scientific data.

```
sudo apt-get install libhdf5-dev libhdf5-serial-
dev libhdf5-103
```

Qt libraries will now be installed. It is a cross-platform C++ application framework with rich widgets set its primary feature. The Qt offers standard GUI functionality.

```
sudo apt-get install libqtgui4 libqtwebkit4
libqt4-test python3-pyqt5
```

Coming next is the linear algebra library. The Automatically Tuned Linear Algebra Software

(ATLAS) is a method for automatic numerical software generation and optimization. ATLAS presently supplies optimized versions of Basic Linear Algebra Subroutines (BLAS), a complete linear algebra kernels set. The BLAS is a linear algebra routine subset inside the LAPACK library.

```
sudo apt-get install libatlas-base-dev gfortran
```

We will install libraries which helps in building modules for python 2 and python3

```
sudo apt-get install python2.7-dev python3-dev
```

The ImageTk module holds support to first create and then modify Tkinter, PhotoImage, BitmapImage, and objects from the PIL images. This is optional and only required when tkinter module is used.

```
sudo apt-get install python3-pil.imagetk
```

Let us now install pip. pip is the standard package manager for Python. It allows you to install and manage additional packages that are not part of the Python standard library.

```
wget https://bootstrap.pypa.io/get-pip.py
sudo python3 get-pip.py
```

We will install numpy, Raspberry Pi camera, and the needed libraries post pip installation for the PiFace digital board.

NumPy is the basic scientific computing package in Python. It is essentially a Python library that offers a multidimensional array object, assortment for fast operational routines, and various derived objects (like masked arrays and matrices). The routines relate to arrays, including mathematical, logical, sorting, selecting, I/O shape manipulation, fundamental statistical operations discrete Fourier transforms, basic linear algebra, random simulation among others.

```
pip3 install numpy
pip3 install "picamera[array]"
pip3 install pifacecommon
pip3 install pifacedigitalio
```

Now we will install openCV from source. You can skip this step if you want to directly install from repository.

Install OpenCV from source:

```
cd ~
wget -O opencv.zip https://github.com/opencv/
opencv/archive/4.2.0.zip
unzip opencv.zip
wget -
O opencv_contrib.zip https://github.com/opencv/
opencv_contrib/archive/4.2.0.zip
unzip opencv_contrib.zip

cd ~/opencv-4.2.0/
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
      -D CMAKE_INSTALL_PREFIX=/usr/local \
      -D OPENCV_EXTRA_MODULES_PATH=~/
         opencv_contrib-4.1.1/modules \
      -D ENABLE_NEON=ON \
      -D ENABLE_VFPV3=ON \
      -D BUILD_TESTS=OFF \
      -D INSTALL_PYTHON_EXAMPLES=ON \
      -D OPENCV_ENABLE_NONFREE=ON \
      -D CMAKE_SHARED_LINKER_FLAGS=-atomic \
      -D BUILD_EXAMPLES=ON...
make -j4
sudo make install
sudo ldconfig
```

OpenCV alternate install (Directly through PiP)

```
pip3 install opencv-python==4.2.0.34
```

Facial recognition libraries

ML Applications libraries will now be installed. The Dlib open-source library targets both research scientists and engineers and aims to offer a thriving ML software development environment. The Face_recognition library is built on a dlib that Recognizes and also manipulate faces taken from Python.

```
pip3 install dlib
pip3 install face_recognition
pip3 install imutils
```

Gather images for encodings

Gather some images which are used to actuate our door lock. Create a new folder by name `**actuatorImage**'. Save four images (Mark Wahlberg, Matt Damon, Keanu Reeves, and Hugh Jackman) in total by their respective names. This will help us to automatically get the names.

Code

Now we will write the code to get encodings from each image. Create a new file and name it dataEncode.py and save it to the home folder. Open the file in any file editor and type the code which we will see step by step.

Import the necessary libraries which is required for the project which we installed earlier.

```
import os
import pickle
import cv2
import face_recognition
```

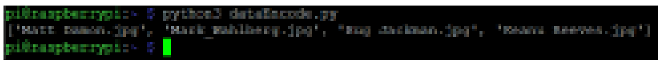
We will create a list that will automatically import images from our folder `**actuatorImage**'. **Images =[]** will create a list of all imported images. Instead of manually writing the name of each person, we will obtain the names of the particular person from the image itself. When we output the result the names of the corresponding person are displayed.

We will grab a list of all images in the folder so we will write **mylist= os.listdir(path)**

This will give us the path.

```
path = `actuatorImage'
images = []
classNames = []
mylist = os.listdir(path)
#print(mylist)
```

Let us now print the list and we can see the names of the files in the folder. Run the file dataEncode.py by typing python3 dataEncode.py in terminal. The terminal window can be opened by typing Ctrl+Alt+T



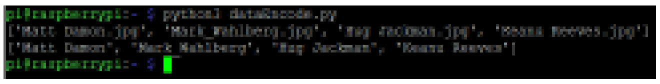
```
pi@raspberrypi:~ $ python3 dataEncode.py
['Matt Damon.jpg', 'Mark_Wahlberg.jpg', 'Hug Jackman.jpg', 'Keanu Reeves.jpg']
pi@raspberrypi:~ $
```

Fig: Terminal window

The below code will loop over all the images and filter out the extension .jpg from the image names.

```
for img in mylist:
 current_img = cv2.imread(f'{path}/{img}')
 images.append(current_img)
 classNames.append(os.path.splitext(img)[0])
print(classNames)
```

We will now print out the classNames by running dataEncode.py. We see it prints without .jpg extension.



```
pi@raspberrypi:~ $ python3 dataEncode.py
['Matt Damon.jpg', 'Mark_Wahlberg.jpg', 'Hug Jackman.jpg', 'Keanu Reeves.jpg']
['Matt Damon', 'Mark Wahlberg', 'Hug Jackman', 'Keanu Reeves']
pi@raspberrypi:~ $
```

Fig: Terminal window

We will now save the list consisting of names to a file so that can be used later by the Raspberry Pi. Data encodings are done on faster machines since most of the time as it takes more time to process on Raspberry Pi. Hence it is better to save on a disk and move it to the different machines.

```
with open(`classNames.name’, `wb’) as f:
  pickle.dump(classNames, f)
```

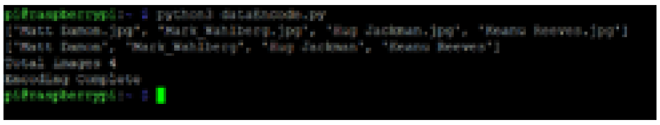
We will now begin the encoding process. We have to find 128- embeddings of each image to be loaded. For this, we will create a function by name findEncodings and loop over each image to get the encodings. The **face_recognition.face_encodings(img)[0])** function will find out all the encodings which are defined in the facial_recognition library. We will append all these encodings and save it to **encodeList[]**

```
def findEncodings(images):
  encodeList = []
  for img in images:
      img = cv2.cvtColor(img, cv2.COLOR_
      BGR2RGB)
      encode = face_recognition.face_
      encodings(img)[0]

      encodeList.append(encode)

  return encodeList
```

Save these encodings into the dataset_faces.dat file with the help of the pickle library so we can use the face encodings on any machine.

```
encodeListKnown = findEncodings(images)
print(“Total Images”,len(encodeListKnown))
print(`Encoding Complete’)
#print(encodeListKnown)
with open(`dataset_faces.dat’, `wb’) as f:
  pickle.dump(encodeListKnown, f)
```

Save the dataEncode.py file and run it by typing python3 dataEncode.py in the terminal.



```
pi@raspberrypi:~ $ python3 dataEncode.py
['Matt Damon.jpg', 'Mark_Wahlberg.jpg', 'Hug Jackman.jpg', 'Keanu Reeves.jpg']
['Matt Damon', 'Mark_Wahlberg', 'Hug Jackman', 'Keanu Reeves']
Total Images 4
Encoding Complete
pi@raspberrypi:~ $
```

Fig: Terminal window

We have created face encodings and we will use it to compare it with real time faces using a camera and actuate the door when it matches with the images in the previously saved list.

Let us now create a new file and name it dataActuate.py and import the necessary libraries.

```
import time
import cv2
import numpy as np
from imutils.video import VideoStream
import threading
import imutils
import face_recognition
import pickle
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
raspCam=True
cnt=0
import pifacedigitalio
```

Here we define our main function where we will load previously generated **dataset_faces.dat** and **classNames.name** file.

We will capture the image from the camera and match the current image with the previously encoded image. Whenever the current images match with the previously saved image. The Raspberry Pi send signal to Relay pins on which Magnetic Actuator is connected and the door will open. There are few lines commented out in the code which will show the name on the display with the bounding box on the image.

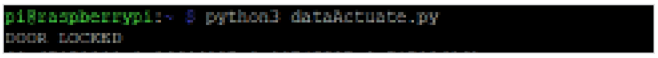
```
if __name__ == `__main__’:

  with open(`dataset_faces.dat’, `rb’) as f:
      encodeListKnown = pickle.load(f)
  pfd = pifacedigitalio.PiFaceDigital()
  with open(`classNames.name’, `rb’) as f:
      classNames = pickle.load(f)
  count()
  # Are we using the Pi Camera?
  usingPiCamera = True
  # Set initial frame size.
  frameSize = (320, 240)

  # Initialize mutithreading the video stream.

  vs = VideoStream(src=0,
       usePiCamera=usingPiCamera,
       resolution=frameSize,
       framerate=32).start()
  # Allow the camera to warm up.
  time.sleep(2.0)

  timeCheck = time.time()
  while True:
      img = vs.read()
      # success, img = cap.read()
      imgS = cv2.resize(img, (0, 0), None,
             0.25, 0.25)
      # img = cv2.cvtColor(imgS,cv2.COLOR_
             BGR2RGB)
      # img = cv2.cvtColor(imgS, cv2.COLOR_
               RGB2BGR)
      # img=imgS
      facesCurFrame = face_recognition.
                      face_locations(imgS)
      encodeCurFrame = face_recognition.
                       face_encodings(imgS,
                       facesCurFrame)
      for encodeFace, faceLoc in
      zip(encodeCurFrame, facesCurFrame):
        matches = face_recognition.
        compare_faces(encodeListKnown,
        encodeFace)
        faceDis = face_recognition.face_
        distance(encodeListKnown,
                 encodeFace)
        print(faceDis)
        matchIndex = np.argmin(faceDis)
        if matches[matchIndex]:
           name = classNames[matchIndex].
                  upper()
           print(name)
           print(“DOOR UNLOCKED”)
           pfd.relays[0].value = 1
           #GPIO.output(17, GPIO.HIGH)
           cnt = 3
           # time.sleep(5);
           # matches=False
           # y1,x2,y2,x1=faceLoc
           # cv2.
           rectangle(img,(x1,y1),(x2,y2),
          (0,255,0),2)
           # cv2.rectangle(img,(x1,y2-
           35),(x2,y2),(0,255,0),cv2.FILLED)
      # cv2.putText(img,name,(x1+6,y26),
     cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)


    # Get the next frame.


  if cnt == 0:
      #GPIO.output(17, GPIO.LOW)
      pfd.relays[0].value = 0
      print("DOOR LOCKED")
      cnt = -1

# Show video stream
        #cv2.imshow(`orig', img)
        #key = cv2.waitKey(1) & 0xFF

# if the `q` key was pressed, break from
    the loop.
     #if key == ord("q"):
        #break



# Cleanup before exit.
     #cv2.destroyAllWindows()
     #vs.stop()
```

When the door opens, we need to set the timer to close after sometime and release the relay, for this let us define the function.

```
def count():
     global cnt
     threading.Timer(1.0,count).start()
     if cnt>0:
        cnt=cnt-1
```

After writing the code, save the file dataActuate.py. Open terminal by typing Ctrl+Alt+T. Type python3 dataActuate.py.

The codes runs and prints Door Locked as shown in below figure.



```
pi@raspberrypi:~ $ python3 dataActuate.py
DOOR LOCKED
```

Fig: Terminal window

The code then starts the videostream from raspberry pi camera and wait for the pictures to be detected. When there is no valid image detected, the solenoid valve will remain in closed position as shown in below figure.

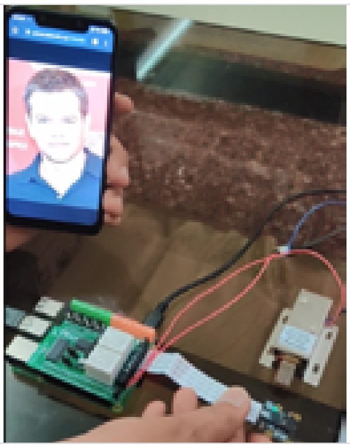


Fig: Undetected image

When the valid image detected the solenoid Lock will open as shown in the below figure.

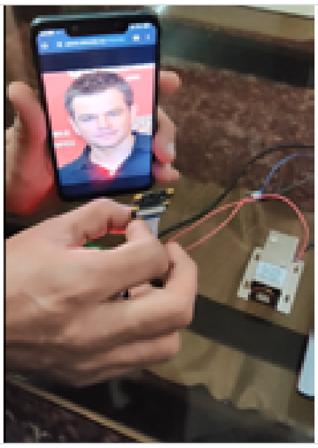
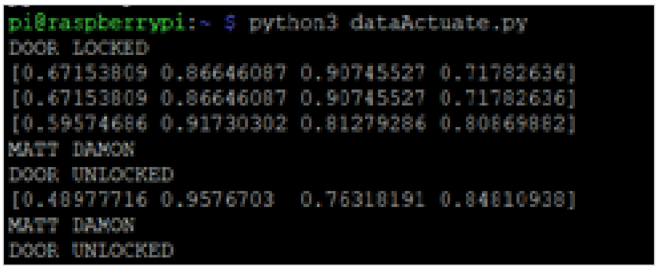


Fig: Detected image

In the terminal it will print the name of the person whose image gets detected and the door unlocked status.

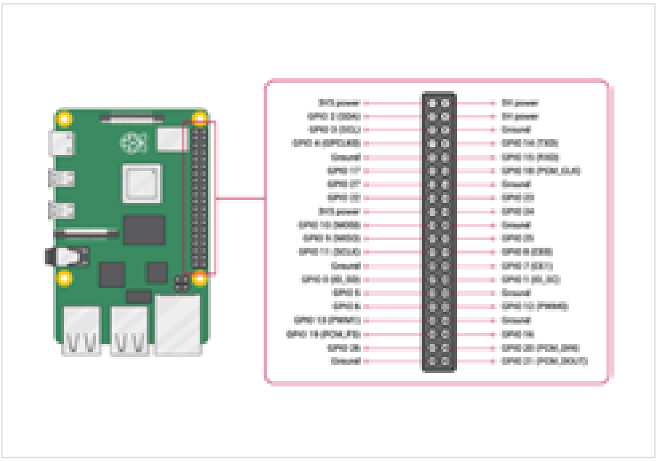


```
pi@raspberrypi:~ $ python3 dataActuate.py
DOOR LOCKED
[0.67153809 0.86646087 0.90745527 0.71782636]
[0.67153809 0.86646087 0.90745527 0.71782636]
[0.59574686 0.91730302 0.81279286 0.80869882]
MATT DAMON
DOOR UNLOCKED
[0.48977716 0.9576703  0.76318191 0.84810938]
MATT DAMON
DOOR UNLOCKED
```

Fig: Terminal window

References

Raspberry Pi 4 pin configuration



Datasheet:
www.farnell.com/datasheets/2819352.pdf

Raspberry Pi 4 power supply

Datasheet:
www.farnell.com/datasheets/2875936.pdf

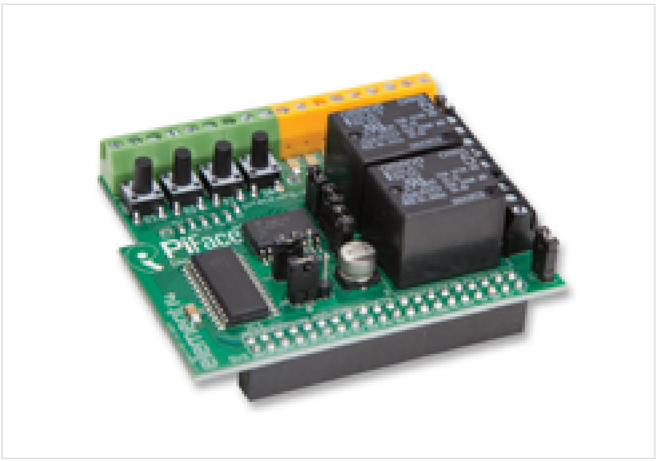
Raspberry Pi high quality camera

Raspberry Pi high quality camera – product brief:
static.raspberrypi.org/files/product-briefs/
Raspberry_Pi_HQ_Camera_Product_Brief.pdf

Solenoid lock

Datasheet:
www.farnell.com/datasheets/2865757.
pdf?_ga=2.69730000.1158994131.1604400308-
1482936714.1603999527

Relay board (PiFaceDigital2)



Datasheet:
www.farnell.com/datasheets/1881551.pdf

How to scale this kit into real work application?

The facial recognition technology can be used in various real world applications few of them can be:

Validate identity at ATMs
Face scans will, in all probability, replace ATM cards as face recognition is a proven identity authentication tool. However, until such a time comes, face recognition can be deployed to ensure that persons using ATM cards are who they claim to be. Face recognition is at present being used at Macau ATMs to secure peoples' identities

Driver recognition
Several car companies continue to experiment with the many uses of face recognition. One use is using a face to substitute the ignition key. Face recognition can be used to change seat preferences and radio stations based on the driver. Face recognition protect drivers safer by recognizing and alerting drivers if they drift off or lose focus while driving.

Smart advertising
Face recognition makes possible targeted advertising by making educated guesses when it comes to people's gender and age. Companies (like Tesco) plan to install integrated face recognition screens at gas stations with face recognition technology built-in.

Track school attendance
Face recognition has the potential to track students' attendance. Tablets can scan students' faces and then match their photographs against a database to confirm their identities.

Common problems with real work application

The face detection complication is a tough one as it must account for all possible appearance variations due to lighting conditions, facial features color of the face, illumination change, and occlusions.

A relay is generally characterized by its rated coil voltage for operating its contacts, and its maximum switching voltage and current. The manufacturer prints this well-intended warning to prevent contact destruction.

The effects that happen at a relay contact primarily depend on the type and size of the load, operation time, the current, the contact size and material, and the contact bounce.