##### A MAJOR PROJECT REPORT

ON

FACE RECOGNITION SYSTEM

***In partial fulfilment for the award of the degree***

***Of***

**BACHELOR OF COMPUTER APPLICATIONS**



MAHARAJA SURAJMAL INSTITUTE

##### C-4, JANAK PURI,

**NEW DELHI - 110058**

##### May 2018

Submitted to:

Dr Meenal Dahiya

Assistant Professor

Department of Computer Science

Maharaja Surajmal Institute

Submitted By:  
Divyanshu Sengar

# 

# ABSTRACT

*Face Detection System, is a project created with the use of Raspberry Pi, which is a credit card sized computer, based on a 4 core ARM processor. The motive of this project is to utilize the Raspberry Pi as a surveillance/automation hub, which, in conjuncture with a webcam, would provide wireless face monitoring capabilities over the internet in real-time. All programming for the facilitation of proposed features would be done on Python, as it provides tight integration with the Raspbian OS, the native operating system for Raspberry Pi. The Python IDE in the OS is very flexible, and works at a low level, thus allowing us to expand the system as and when required to accommodate more cameras, other sensors, and creating monitoring hubs for a multi-computer setup.*

Table of Contents

[DECLARATION 3](#__RefHeading___Toc5667_1809370720)

[ACKNOWLEDGEMENT 4](#__RefHeading___Toc5881_1809370720)

[ABSTRACT 5](#__RefHeading___Toc5883_1809370720)

[1. INTRODUCTION 8](#__RefHeading___Toc5477_1809370720)

[1.1 Background Information 9](#__RefHeading___Toc5479_1809370720)

[1.2 Reason to Choose Python and OpenCV 10](#__RefHeading___Toc5485_1809370720)

[1.3 Applications and Software Attributes 10](#__RefHeading___Toc5487_1809370720)

[1.4 Objectives of the Project 1](#__RefHeading___Toc5489_1809370720)2

[2. METHODS AND MATERIALS USED 13](#__RefHeading___Toc5493_1809370720)

[2.1 Haar-Cascade Identifier 14](#__RefHeading___Toc5495_1809370720)

[2.1.1 Software Requirements 14](#__RefHeading___Toc5497_1809370720)

[2.1.1.1 Python 14](#__RefHeading___Toc5499_1809370720)

[2.1.1.2 OpenCV 15](#__RefHeading___Toc5501_1809370720)

[2.1.1.3 Raspbian 16](#__RefHeading___Toc5503_1809370720)

2.1.1.4 MAKE 17

[2.1.2 Hardware Requirements 20](#__RefHeading___Toc5505_1809370720)

[2.1.3 External Libraries used for Development 20](#__RefHeading___Toc5507_1809370720)

[2.1.3.1 NumPy 20](#__RefHeading___Toc5513_1809370720)

[2.2 Principles of Haar-Cascade Identifier 21](#__RefHeading___Toc5515_1809370720)

[3. FUNCTIONING OF APPLICATION 2](#__RefHeading___Toc5517_1809370720)4

[3.1 Installing OpenCV on Raspberry Pi 2](#__RefHeading___Toc5519_1809370720)5

[3.2 Setting up Web Camera 3](#__RefHeading___Toc5521_1809370720)4

[3.3 Face Detection 34](#__RefHeading___Toc5523_1809370720)

3.3.1 Smile Detection 36

3.3.2 Eye Detection 37

3.3.3 Smile and Eye Detection 38

3.4 Face Recognition 39

3.4.1 Data Gathering 39

3.4.2 Trainer 41  
3.4.3 Recognizer 42

[3.5 Screenshots 4](#__RefHeading___Toc5671_1809370720)5

4. FUTURE SCOPE 46

5. CONCLUSION 48

6. REFRENCES 50

**1. INTRODUCTION**

## Background Information

We live in a digital era. In this age, information travels faster than ever before, and that speed translates to us wanting everything to be faster than before. Our project aims to automate a very big part of it, and that is collection of local demographics. Traditional attendance and survey systems implemented in a small area take a major chunk of the available time. For instance, in a 30-minute-long lecture, it becomes impractical if the attendance process takes 5 minutes.

Computer vision allows us to process visual data at a rate never done before, while remaining computationally light. Implemented on a large scale, the system becomes an able platform to perform large scale space audits, producing accurate demographic data even in large scale implementations, like stadiums, auditoriums, concert halls, etc. This application can also be used for security purposes, with the system being able to cross reference facial databases present with law enforcement authorities.

## 1.2 Reason to Choose Python and OpenCV

* Definitely decreases your development time.
* It's easier to debug.
* An Image is basically an nD-array for which Python's NumPy and SciPy.org modules can be used.
* The array manipulations in NumPy module are highly optimized for speed. The Multi-dimensional image processing (submodule in SciPy is concentrated for image processing.)
* You don’t have to worry about memory management directly.
* You can combine the conciseness of Python with high performance powers of C using Python: C-Extensions for Python. It can provide a speed boost of up to 100x or even higher compared to native Python code depending on your application.
  + - *Sometimes in large scale projects, not everyone is comfortable with Python and codes in C++, so one has to conform to the unified coding style.*

## 1.3 Applications and Software Attributes

While providing basic motion detection and recording abilities, the software can be extended to provide full network support, including Dropbox integration, so that the system can be monitored from anywhere across the world. Also, as the software utilizes OpenCV, powerful image processing can be done, and the software can be used to perform facial recognition, which can be then cross referenced to public records to know exactly who is in the camera. This greatly increases the capabilities of the software, without any additional expense on hardware costs.

The entire software is written in Python, and utilizes OpenCV APIs, along with the picamera libraries to use a USB webcam. These libraries are open source and can be freely implemented across any platform. Some of the applications are written as follows:

* **Energy Savings:** An actively monitored system can detect when a room or an area is not in use, and turn off additional appliances consuming electricity. This system can be implemented throughout the area of deployment, and can greatly reduce the carbon footprint of an area.
* **Access Control:** The Raspberry Pi can be given additional biometric scanning abilities, which can allow it to detect when an authorized person is about to enter a room. This ability can be used to prevent unauthorized access, without causing hindrance to authorized personnel, unlike conventional authentication protocols.
* **Pre-emptive actions:** The system can be programmed to perform a certain set of actions when a certain event occurs. This may include dialling a set of numbers, unlocking the garage door, etc.
* **Occupancy Counter:** This system can be effectively used to count the number of people inside a given area. This functionality is useful when data is required regarding how many people attended a certain event is required, additional statistics, etc.

As well as being flexible and expandable, this project is highly marketable, as equally priced offerings in the market don’t have as many features, while those having equivalent features are much more expensive.

## 1.4 Objectives of the Project

The objective of our project is to provide a low cost, extensible surveillance platform, which can be easily implemented by anyone, with zero to minimal operational costs, and a one-time hardware investment. The requirement of such systems are high, and with a cost considerably less than other systems available in market, it’s easily sellable. As there are no recurring costs, more people will be willing to buy it.

The primary reason for building it as a DIY project is the cost advantage. All the necessary hardware can be purchased for under ₹5000. Given the open nature of the Raspberry Pi platform, the build can be further improved, and additional features can be implemented as and when required.

1. Open platform: Raspberry Pi runs on Raspbian, which is an Open Source Linux distribution. Any and all parts of the operating system can be modified to allow for extensibility and features.
2. Network compatibility: As the Raspberry Pi is Wi-Fi enabled, the whole setup can be put anywhere with minimal wiring requirements.

# 

# 2. METHODS AND MATERIALS USED

In order to fulfil the objectives that have been spelled out in the previous chapter, an exploratory research was carried out. To successfully execute Face Detection and Recognition, we need to use the principle of Haar-Cascade analysis implemented using Python and OpenCV on the Raspberry Pi and a USB webcam.

## 2.1 To utilise the software to implement Haar-Cascade analysis

### 2.1.1 Software Requirements

The following is the software requirements for the project:

* Python
* OpenCV
* Raspbian OS
* MAKE

##### 2.1.1.1 Python

Python is a high-level, interpreted and general-purpose dynamic programming language that focuses on code readability. The syntax in Python helps the programmers to do coding in fewer steps as compared to Java or C++.

The Python language has diversified application in the software development companies such as in gaming, web frameworks and applications, language development, prototyping, graphic design applications, etc. This provides the language a higher plethora over other programming languages used in the industry. Some of its advantages are:

* ***Extensive Support Libraries***

It provides large standard libraries that include the areas like string operations, Internet, web service tools, operating system interfaces and protocols. Most of the highly used programming tasks are already scripted into it that limits the length of the codes to be written in Python.

* **Integration Feature**

Python integrates the Enterprise Application Integration that makes it easy to develop Web services by invoking COM or COBRA components. It has powerful control capabilities as it calls directly through C, C++ or Java via Python. Python also processes XML and other mark-up languages as it can run on all modern operating systems through same byte code.

* ***Improved Programmer’s Productivity***

The language has extensive support libraries and clean object-oriented designs that increase two to tenfold of programmer’s productivity while using the languages like Java, VB, Perl, C, C++ and C#.

* ***Productivity***

With its strong process integration features, unit testing framework and enhanced control capabilities contribute towards the increased speed for most applications and productivity of applications. It is a great option for building scalable multi-protocol network applications.

##### 2.1.1.2 OpenCV

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it’s free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or robotics.

**Advantages:**

* First and foremost, OpenCV is available **free** of cost
* Since OpenCV library is written in C/C++ it is quite **fast**
* **Low RAM usage** (approx. 60–70 MB)
* It is **portable** as OpenCV can run on any device that can run C

##### 2.1.1.3 Raspbian OS

Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make your Raspberry Pi run. However, Raspbian provides more than a pure OS: it comes with over 35,000 packages, pre-compiled software bundled in a nice format for easy installation on your Raspberry Pi.

The initial build of over 35,000 Raspbian packages, optimized for best performance on the Raspberry Pi, was completed in June of 2012. However, Raspbian is still under active development with an emphasis on improving the stability and performance of as many Debian packages as possible.

Raspbian uses PIXEL, Pi Improved Xwindows Environment, Lightweight as its main desktop environment as of the latest update. It is composed of a modified LXDE desktop environment and the Openbox stacking window manager with a new theme and few other changes. The distribution is shipped with a copy of computer algebra program Mathematica and a version of Minecraft called Minecraft PI as well as a lightweight version of Chromium as of the latest version. The initial build was completed in June 2012. The operating system is still under active development. Raspbian is highly optimized for the Raspberry Pi line's low-performance ARM CPUs.

**2.1.1.4 MAKE**

In software development, Make is a build automation tool that automatically builds executable programs and libraries from source code by reading files called Makefiles which specify how to derive the target program. Though integrated development environments and language-specific compiler features can also be used to manage a build process, Make remains widely used, especially in UNIX and Unix-like operating systems.

Besides building programs, Make can be used to manage any project where some files must be updated automatically from others whenever the others change.  
  
***Origin***

There are now a number of dependency-tracking build utilities, but Make is one of the most widespread, primarily due to its inclusion in Unix, starting with the PWB/UNIX 1.0, which featured a variety of tools targeting software development tasks. It was originally created by Stuart Feldman in April 1976 at Bell Labs. Feldman received the 2003 ACM Software System Award for the authoring of this widespread tool.

Feldman was inspired to write Make by the experience of a co-worker in futilely debugging a program of his where the executable was accidentally not being updated with changes:

Make originated with a visit from Steve Johnson (author of yacc, etc.), storming into my office, cursing the Fates that had caused him to waste a morning debugging a correct program (bug had been fixed, file hadn't been compiled, cc \*.o was therefore unaffected).

“As I had spent a part of the previous evening coping with the same disaster on a project I was working on, the idea of a tool to solve it came up. It began with an elaborate idea of a dependency analyzer, boiled down to something much simpler, and turned into Make that weekend. Use of tools that were still wet was part of the culture. Makefiles were text files, not magically encoded binaries, because that was the Unix ethos: printable, debuggable, understandable stuff."

— Stuart Feldman, The Art of Unix Programming, Eric S. Raymond 2003

Before Make's introduction, the Unix build system most commonly consisted of operating system dependent "make" and "install" shell scripts accompanying their program's source. Being able to combine the commands for the different targets into a single file and being able to abstract out dependency tracking and archive handling was an important step in the direction of modern build environments.  
  
  
***Derivatives***

Make has gone through a number of rewrites, including a number of from-scratch variants which used the same file format and basic algorithmic principles and also provided a number of their own non-standard enhancements. Some of them are:

Sun DevPro make was a rewrite of the UNIX make program that appeared in 1986 with SunOS-3.2. With SunOS-3.2, DevPro make was delivered as optional program; with SunOS-4.0, SunPro make was made the default make program. In December 2006, Sun DevPro make was made OpenSource as part of the efforts to open-source Solaris, in order to regain lost market share.

Dmake or Distributed Make that came with Sun Solaris Studio as its default make, but not the default one on the Solaris Operating System (SunOS). It was originally required to build OpenOffice, but in 2009 the build system was rewritten to use GNU make. While Apache OpenOffice still contains a mixture both build systems, the much more actively developed LibreOffice only uses the modernized "gbuild" now.

BSD Make (pmake, bmake or fmake), which is derived from Adam de Boor's work on a version of Make capable of building targets in parallel, and survives with varying degrees of modification in FreeBSD, NetBSD and OpenBSD. Distinctively, it has conditionals and iterative loops which are applied at the parsing stage and may be used to conditionally and programmatically construct the makefile, including generation of targets at runtime.

GNU Make (short gmake) is the standard implementation of make for Linux and OS X. It provides several extensions over the original make, such as conditionals. It also provides many built-in functions which can be used to eliminate the need for shell-scripting in the makefile rules as well as to manipulate the variables set and used in the makefile. For example, the foreach function sets a variable to the list of all files in a given directory. GNU Make has been required for building gcc since version 3.4. It is required for building the Linux kernel, Apache OpenOffice and LibreOffice, Mozilla Firefox and many others.

Rocky Bernstein's remake is a fork of GNU make and provides several extensions over GNU make, such as better location and error-location reporting, execution tracing, execution profiling, and it contains a debugger.

Glenn Fowler's nmake is unrelated to the Microsoft program of the same name. Its input is similar to make, but not compatible. This program provides shortcuts and built-in features, which according to its developers reduces the size of makefiles by a factor of 10.

Microsoft nmake, a command-line tool which normally is part of Visual Studio. It supports pre-processor directives such as includes and conditional expressions which use variables set on the command-line or within the makefiles. Inference rules differ from make; for example they can include search paths. The make tool supplied with Embarcadero products has a command-line option that "Causes MAKE to mimic Microsoft's NMAKE."

Mk replaced Make in Research Unix, starting from version 9. A redesign of the original tool by Bell Labs programmer Andrew G. Hume, it features a different syntax. Mk became the standard build tool in Plan 9, Bell Labs' intended successor to Unix.

POSIX includes standardization of the basic features and operation of the Make utility, and is implemented with varying degrees of completeness in Unix-based versions of Make. In general, simple makefiles may be used between various versions of Make with reasonable success. GNU Make, BSD Make and Makepp can be configured to look first for files named "GNUmakefile", "BSDmakefile" and "Makeppfile" respectively, which allows one to put makefiles which use implementation-defined behaviour in separate locations.

### 2.1.2 Hardware requirements

The following is the hardware requirements for the project:

* Raspberry Pi 3 with inbuilt Wi-Fi
* Generic Webcam

### 2.1.3 External Libraries Used in Development

# 2.1.3.1 NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

* a powerful N-dimensional array object
* sophisticated (broadcasting) functions
* tools for integrating C/C++ and Fortran code
* useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the BSD license, enabling reuse with few restrictions.

**2.2 Principle of Haar-Cascade Identifier to implement Face Detection**

The Open Source Computer Vision Library (OpenCV) is used to implement the Haar-Cascade classifier. It was written by Paula Viola and Michael Jones. For the detection of faces, Haar features are the main part of the Haar-Cascade classifier. Haar features are used to detect the presence of feature in given image. Each feature result in a single value which is calculated by subtracting the sum of pixels under white rectangles from the sum of pixels under black rectangles as shown below. Haar like features are the rectangle features for rapid face detection. Some Haar-like features are also shown below.

**P(x) = Sumblack rectangle – SumWhite rectangle**

****

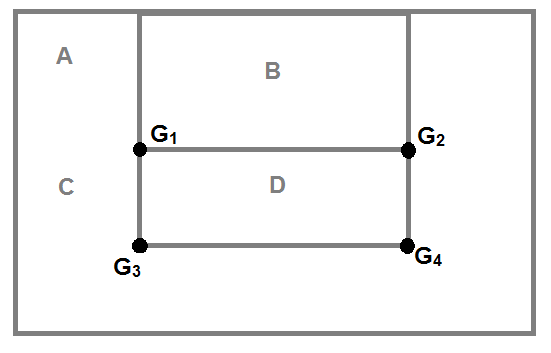
**Fig 1. Haar features**

The Haar feature starts scanning the image for the detection of the face from the top left corner and ends the face detection process at the bottom right corner of the image given. The image is scanned several times through Haar-like features in order to detect the face from an image.

To compute the rectangle features rapidly, integral image concept is used. It need only four values at the corners of the rectangle for the calculation of sum of all pixels inside any given rectangle. In an integral image, the value at pixel (x,y) is the sum of pixels above and to the left of (x,y). Sum of all pixel value in rectangle D is shown in fig. 2.

G1=A, G2=A+B, G3=A+C, G4=A+B+C+D

G1+ G4 - G2- G3=A+A+B+C+D-A-B-A-C=D



**Fig 2. Calculation of integral image**

Viola Jones Algorithm uses a 24×24 window as the base window size to start evaluating these features in any given image. If we consider all the possible parameters of the Haar features like position, type and scale, we would have to calculate upwards of 1,60,000 features in this window but this is practically impossible. The solution of this problem is to use the Adaboost algorithm, which is a machine learning algorithm which helps us to find the best features in a sample set. These features are the weak classifiers and Adaboost construct a strong classifier as a linear combination of these weak classifiers as shown below.

**F(x) =** α**1F1(x) +** α**2F(x)**

The face detection can be performed by cascade using Haar-like features as shown in figure below**.** In that cascade, an image will be a human face if it passes all the stages. If it is not passed any one of the stage it means the image is not a human face.

# 

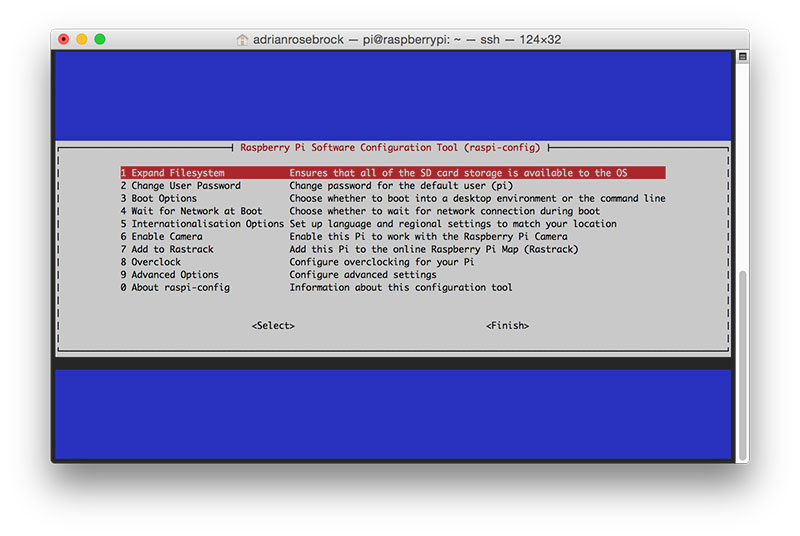
# 3. FUNCTIONING OF THE APPLICATION

## 3.1 Installing OpenCV and Python on the Raspberry Pi

### *Step #1: Expand filesystem*

Expand the filesystem to include the entire space of the SD card if using the latest version of Raspbian Jesse.

**$sudo raspi-config**



**Figure 1:** Expanding the filesystem on your Raspberry Pi 3.

Once prompted, select the first option, ***“1. Expand File System”***, ***hit Enter*** on your keyboard, arrow down to the ***“<Finish>”*** button, and then reboot the Pi:

|  |  |
| --- | --- |
|  | **$sudo reboot** |

After rebooting, the file system should have been expanded to include all available space on your micro-SD card. We can verify that the disk has been expanded by executing df-h  and examining the output:

**$df-h**

**Filesystem      Size  Used Avail Use%Mounted on**

**/dev/root       7.2G  3.3G  3.6G  48%/**

**devtmpfs        459M     0  459M   0%/dev**

**tmpfs           463M     0  463M   0%/dev/shm**

**tmpfs           463M  6.4M  457M   2%/run**

**tmpfs           5.0M  4.0K  5.0M   1%/run/lock**

**tmpfs           463M     0  463M   0%/sys/fs/cgroup**

**/dev/mmcblk0p1   60M   20M   41M  34%/boot**

**tmpfs 93M 0 93M 0%/run/user/1000**

### *Step #2: Install dependencies*

The first step is to update and upgrade any existing packages:

**$sudo apt-getupdate**

**$sudo apt-getupgrade**

We then need to install some developer tools, including [CMake](https://cmake.org/), which helps us configure the OpenCV build process:

**$sudo apt-getinstall build-essential cmake pkg-config**

***Timing: 40s***

We will then be installing some image I/O packages that allow us to load various image file formats from disk. Examples of such file formats include JPEG, PNG, TIFF, etc.:

**$sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev**

***Timing: 32s***

Just as we need image I/O packages, we also need video I/O packages. These libraries allow us to read various video file formats from disk as well as work directly with video streams:

**$sudo apt-getinstall libavcodec-dev libavformat-dev libswscale-dev libv4l-dev**

**$sudo apt-getinstall libxvidcore-dev libx264-dev**

***Timing: 34s***

The OpenCV library comes with a sub-module named highgui  which is used to display images to our screen and build basic GUIs. In order to compile the highgui  module, we need to install the GTK development library:

**$sudo apt-getinstall libgtk2.0-dev**

***Timing: 3m 6s***

Many operations inside of OpenCV (namely matrix operations) can be optimized further by installing a few extra dependencies:

**$sudo apt-getinstall libatlas-base-dev gfortran**

***Timing: 46s***

These optimization libraries are especially important for resource constrained devices such as the Raspberry Pi.

Now, to install Python 2.7 and Python 3 so we can compile OpenCV with Python bindings:

**$sudo apt-getinstall python2.7-dev python3-dev**

***Timing: 45s***

### *Step #3: Download the OpenCV source code*

After installing the necessary dependencies,we have to install the latest archive of OpenCV (3.1.0,in this case) from the official OpenCV repository.

**$cd~**

**$wget-Oopencv.ziphttps://github.com/Itseez/opencv/archive/3.1.0.zip**

**$unzip opencv.zip**

***Timing: 1m 26s***

To have access to features such as SIFT and SURFin OpenCV 3, we also need to grab the [opencv\_contrib](https://github.com/itseez/opencv_contrib) repository as well:

**$wget Oopencv\_contrib.ziphttps://github.com/Itseez/opencv\_contrib/archive /3.1.0.zip**

**$unzip opencv\_contrib.zip**

***Timing: 43s***

You might need to expand the command above using the “<=>” button during your copy and paste. The .zip  in the 3.1.0.zip  may appear to be cutoff in some browsers. The full URL of the OpenCV 3.1.0 archive is:

https://github.com/Itseez/opencv\_contrib/archive/3.1.0.zip

### *Step #4: Python 2.7 or Python 3?*

Before we can start compiling OpenCV on our Raspberry Pi 3, we first need to install pip, a Python package manager:

**$wget https://bootstrap.pypa.io/get-pip.py**

***Timing: 20s***

Now we need to install both virtualenv and virtualwrapper :

**$sudo pip install virtualenv virtualenvwrapper**

**$sudo rm-rf~/.cache/pip**

***Timing: 9s***

Now that both virtualenv and virtualenvwrapper have been installed, we need to update our ~/.profile file to include the following lines at the bottom of the file:

**# virtualenv and virtualenvwrapper**

**export WORKON\_HOME=$HOME/.virtualenvs**

**source/usr/local/bin/virtualenvwrapper.sh**

After updating our~/.profile, we need to reload it to make sure the changes take effect. You can force a reload of your ~/.profile file by:

1. Logging out and then logging back in.
2. Closing a terminal instance and opening up a new one
3. Or our personal favourite, ***just use the source command:***

$source~/.profile

**Creating the Python Virtual Environment**

Next, let’s create the Python virtual environment that we’ll use for computer vision development:

**$mkvirtualenv cv-ppython2**

This command will create a new Python virtual environment named cv using ***Python 2.7***.

If you instead want to use ***Python 3***, you’ll want to use this command instead:

**$mkvirtualenv cv-ppython3**

|  |  |
| --- | --- |
|  |  |

### Installing NumPy on your Raspberry Pi

Our only Python dependency is [NumPy](http://www.numpy.org/), a Python package used for numerical processing:

**$pip install numpy**

***Timing: 9m 39s***

### *Step #5: Compile and Install OpenCV*

OpenCV can finally be compiled together. Double-check that you are in the cv\_virtual environment by examining your prompt (you should see the (cv) text preceding it), and if not, simply execute workon:

**$workon cv**

Once you have ensured you are in the cv virtual environment, we can setup our build using CMake:

**$cd~/opencv-3.1.0/**

**$mkdirbuild**

**$cdbuild**

**$cmake-DCMAKE\_BUILD\_TYPE=RELEASE\**

**-DCMAKE\_INSTALL\_PREFIX=/usr/local\**

**-DINSTALL\_PYTHON\_EXAMPLES=ON\**

**-DOPENCV\_EXTRA\_MODULES\_PATH=~/opencv\_contrib-3.1.0/modules\**

**-DBUILD\_EXAMPLES=ON..**

***Timing: 1m 57s***

OpenCV compilation can finally be done :

**$make-j4**

***Timing: 1h 12m***

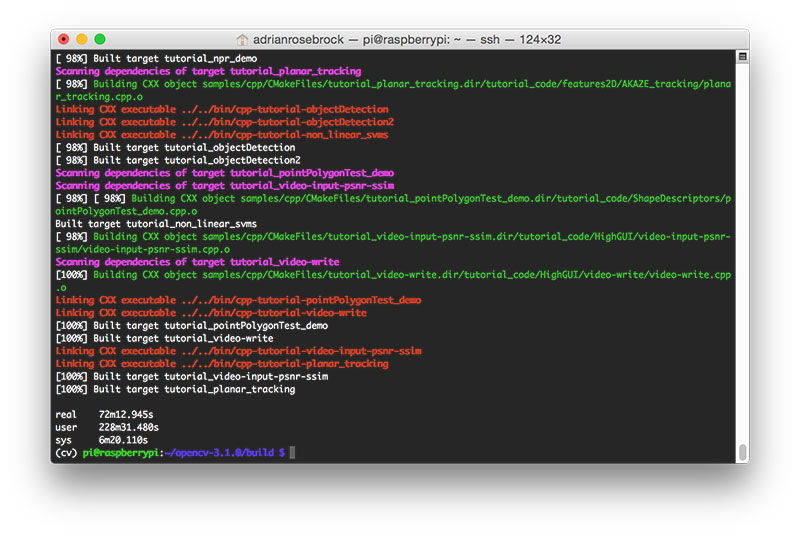
The -j4  command controls the number of cores to leverage when compiling OpenCV 3. The Raspberry Pi 3 has four cores, thus we supply a value of 4 to allow OpenCV to compile faster.

However, due to race conditions, there are times when make errors out when using multiple cores. If this happens to you, I suggest starting the compilation over again and using only one core.

**$makeclean**

**$make**

Once OpenCV 3 has finished compiling, the output should look:



**Figure 5:** Our OpenCV 3 compile on Raspbian Jessie has completed successfully.

Now all that’s left is to install OpenCV 3 on your Raspberry Pi 3:

**$sudo makeinstall**

**$sudo ldconfig**

***Timing: 52s***

### *Step #6: Finish installing OpenCV on your Pi*

We’re almost done — just a few more steps to go and you’ll be ready to use your Raspberry Pi 3 with OpenCV 3.

#### For Python 2.7:

Provided your **Step #5** finished without error, OpenCV should now be installed in /usr/local/lib/python2.7/site-pacakges . You can verify this using the ls  command:

**$ls-l/usr/local/lib/python2.7/site-packages/**

**total1852**

**-rw-r--r--1root staff1895772Mar2020:00cv2.so**

Our final step is to sym-link the OpenCV bindings into our cv  virtual environment for Python 2.7:

**$cd~/.virtualenvs/cv/lib/python2.7/site-packages/**

**$ln-s/usr/local/lib/python2.7/site-packages/cv2.socv2.so**

|  |  |
| --- | --- |
|  |  |

#### For Python 3:

After running make install , the OpenCV + Python bindings should be installed in /usr/local/lib/python3.4/site-packages . Again, you can verify this with the ls  command:

**$ls-l/usr/local/lib/python3.4/site-packages/**

**total1852**

**-rw-r--r--1root staff1895932Mar2021:51cv2.cpython-34m.so**

we can sym-link our OpenCV bindings into the cv  virtual environment for Python 3.4:

**$cd~/.virtualenvs/cv/lib/python3.4/site-packages/**

**$ln-s/usr/local/lib/python3.4/site-packages/cv2.socv2.so**

### *Step #7: Testing your OpenCV 3 install*

Open up a new terminal, execute the source  and workon  commands, and then finally attempt to import the Python + OpenCV bindings:

**$source~/.profile**

**$workon cv**

**$python**

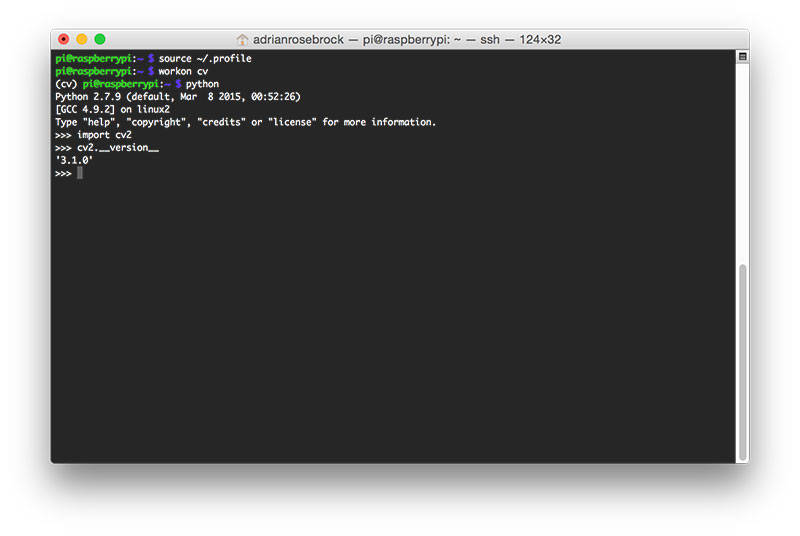
**>>>import cv2**

**>>>cv2.\_\_version\_\_**

**'3.1.0'**

**>>>**

From the below shot, we can see that, **OpenCV 3 has been successfully installed on our Raspberry Pi 3 + Python 2.7 environment:**



**Figure 5:** Confirming OpenCV 3 has been successfully installed on my Raspberry Pi 3 running Raspbian Jessie.

Once OpenCV has been installed, you can remove both the opencv-3.1.0  and opencv\_contrib-3.1.0  directories to free up a bunch of space on your disk:

**$rm-rf opencv-3.1.0opencv\_contrib-3.1.0**

## 3.2 Setting up the Web Camera using Python

## Instead of a PiCamera, which interfaces via the onboard 15 pin camera port on the board, a USB webcam is used for this build. The advantage of a USB camera is that it can be mounted away from the main board, and that it saves CPU processing power since the camera directly sends a video stream to the board, unlike the PiCamera, which sends an un-encoded RAW feed. The USB webcam can be initialized with just one line of code, given below:

**sudo apt-get install fswebcam**

## 3.3 Face Detection

**import numpy as np**

**import cv2**

**faceCascade = cv2.CascadeClassifier('Cascades/haarcascade\_frontalface\_default.xml')**

**cap = cv2.VideoCapture(0)**

**cap.set(3,640) # set Width**

**cap.set(4,480) # set Height**

**while True:**

**ret, img = cap.read()**

**img = cv2.flip(img, -1)**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**faces = faceCascade.detectMultiScale(**

**gray,**

**scaleFactor=1.2,**

**minNeighbors=5,**

**minSize=(20, 20)**

**)**

**for (x,y,w,h) in faces:**

**cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)**

**roi\_gray = gray[y:y+h, x:x+w]**

**roi\_color = img[y:y+h, x:x+w]**

**cv2.imshow('video',img)**

**k = cv2.waitKey(30) & 0xff**

**if k == 27: # press 'ESC' to quit**

**break**

**cap.release()**

**cv2.destroyAllWindows()**

Note the line below:

**faceCascade = cv2.CascadeClassifier('Cascades/haarcascade\_frontalface\_default.xml')**

This is the line that loads the "classifier" (that must be in a directory named "Cascades/", under your project directory).

Then, we will set our camera and inside the loop, load our input video in grayscale mode (same we saw before).

Now we must call our classifier function, passing it some very important parameters, as scale factor, number of neighbours and minimum size of the detected face.

**faces = faceCascade.detectMultiScale(**

**gray,**

**scaleFactor=1.2,**

**minNeighbors=5,**

**minSize=(20, 20)**

**)**

Where,

* **gray** is the input grayscale image.
* **scaleFactor** is the parameter specifying how much the image size is reduced at each image scale. It is used to create the scale pyramid.
* **minNeighbors** is a parameter specifying how many neighbors each candidate rectangle should have, to retain it. A higher number gives lower false positives.
* **minSize** is the minimum rectangle size to be considered a face.

The function will detect faces on the image. Next, we must "mark" the faces in the image, using, for example, a blue rectangle. This is done with this portion of the code:

**for (x,y,w,h) in faces:**

**cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)**

**roi\_gray = gray[y:y+h, x:x+w]**

**roi\_color = img[y:y+h, x:x+w]**

If faces are found, it returns the positions of detected faces as a rectangle with the left up corner (x,y) and having "w" as its Width and "h" as its Height ==> (x,y,w,h). Please see the picture.

Once we get these locations, we can create an "ROI" (drawn rectangle) for the face and present the result with *imshow()* function.

Run the above python Script on your python environment, using the Rpi Terminal:

python faceDetection.py

Note that on a Pi, having several classifiers at same code will slow the processing, once this method of detection (HaarCascades) uses a great amount of computational power. On a desktop, it is easier to run it.

**3.3.1 Smile Detection**

**import numpy as np**

**import cv2**

**# multiple cascades: https://github.com/Itseez/opencv/tree/master/data/haarcascades**

**faceCascade = cv2.CascadeClassifier('Cascades/haarcascade\_frontalface\_default.xml')**

**smileCascade = cv2.CascadeClassifier('Cascades/haarcascade\_smile.xml')**

**cap = cv2.VideoCapture(0)**

**cap.set(3,640) # set Width**

**cap.set(4,480) # set Height**

**while True:**

**ret, img = cap.read()**

**img = cv2.flip(img, -1)**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**faces = faceCascade.detectMultiScale(gray,scaleFactor=1.3,minNeighbors=5,minSize=(30, 30))**

**for (x,y,w,h) in faces:**

**cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)**

**roi\_gray = gray[y:y+h, x:x+w]**

**roi\_color = img[y:y+h, x:x+w]**

**smile = smileCascade.detectMultiScale(roi\_gray,scaleFactor= 1.5,minNeighbors=15,minSize=(25, 25),)**

**for (xx, yy, ww, hh) in smile:**

**cv2.rectangle(roi\_color, (xx, yy), (xx + ww, yy + hh), (0, 255, 0), 2)**

**cv2.imshow('video', img)**

**k = cv2.waitKey(30) & 0xff**

**if k == 27: # press 'ESC' to quit**

**break**

**cap.release()**

**cv2.destroyAllWindows()**

**3.3.2 Eye Detection**

**import numpy as np**

**import cv2**

**# multiple cascades: https://github.com/Itseez/opencv/tree/master/data/haarcascades**

**faceCascade = cv2.CascadeClassifier('Cascades/haarcascade\_frontalface\_default.xml')**

**eyeCascade = cv2.CascadeClassifier('Cascades/haarcascade\_eye.xml')**

**cap = cv2.VideoCapture(0)**

**cap.set(3,640) # set Width**

**cap.set(4,480) # set Height**

**while True:**

**ret, img = cap.read()**

**img = cv2.flip(img, -1)**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**faces = faceCascade.detectMultiScale(gray,scaleFactor=1.3,minNeighbors=5, minSize=(30, 30))**

**for (x,y,w,h) in faces:**

**cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)**

**roi\_gray = gray[y:y+h, x:x+w]**

**roi\_color = img[y:y+h, x:x+w]**

**eyes = eyeCascade.detectMultiScale(roi\_gray,scaleFactor= 1.5,minNeighbors=10,minSize=(5, 5),)**

**for (ex, ey, ew, eh) in eyes:**

**cv2.rectangle(roi\_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)**

**cv2.imshow('video', img)**

**k = cv2.waitKey(30) & 0xff**

**if k == 27: # press 'ESC' to quit**

**break**

**cap.release()**

**cv2.destroyAllWindows()**

**3.3.3 Smile and Eye Detection**

**import numpy as np**

**import cv2**

**# multiple cascades: https://github.com/Itseez/opencv/tree/master/data/haarcascades**

**faceCascade = cv2.CascadeClassifier('Cascades/haarcascade\_frontalface\_default.xml')**

**eyeCascade = cv2.CascadeClassifier('Cascades/haarcascade\_eye.xml')**

**smileCascade = cv2.CascadeClassifier('Cascades/haarcascade\_smile.xml')**

**cap = cv2.VideoCapture(0)**

**cap.set(3,640) # set Width**

**cap.set(4,480) # set Height**

**while True:**

**ret, img = cap.read()**

**img = cv2.flip(img, -1)**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**faces = faceCascade.detectMultiScale(gray,scaleFactor=1.3,minNeighbors=5,minSize=(30, 30))**

**for (x,y,w,h) in faces:**

**cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)**

**roi\_gray = gray[y:y+h, x:x+w]**

**roi\_color = img[y:y+h, x:x+w]**

**eyes = eyeCascade.detectMultiScale(roi\_gray,scaleFactor= 1.5,minNeighbors=5,minSize=(5, 5),)**

**for (ex, ey, ew, eh) in eyes:**

**cv2.rectangle(roi\_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)**

**smile = smileCascade.detectMultiScale(roi\_gray,scaleFactor= 1.5,minNeighbors=15,minSize=(25, 25),)**

**for (xx, yy, ww, hh) in smile:**

**cv2.rectangle(roi\_color, (xx, yy), (xx + ww, yy + hh), (0, 255, 0), 2)**

**cv2.imshow('video', img)**

**k = cv2.waitKey(30) & 0xff**

**if k == 27: # press 'ESC' to quit**

**break**

**cap.release()**

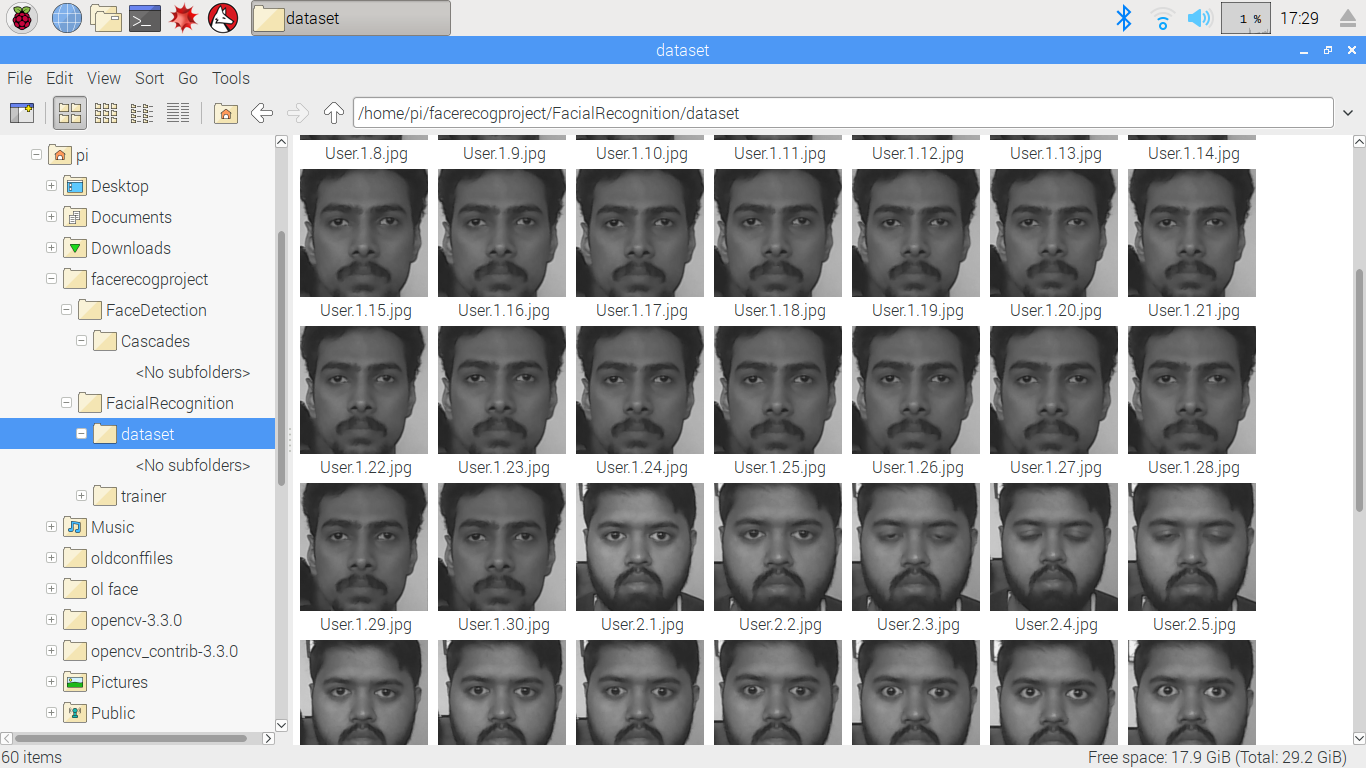
**cv2.destroyAllWindows()**

**3.4 Face Recognition**

Now that we’ve learnt how to detect faces, we shall now implement it to achieve facial recognition for multiple people. This will happen in 3 Phases:

**3.4.1 Data Gathering**

What we will do here, is starting from last step (Face Detecting), we will simply create a dataset, where we will store for each id, a group of photos in grey with the portion that was used for face detecting.

****

First, create a directory where you develop your project, for example, FacialRecognitionProject:

**mkdir FacialRecognitionProject**

In this directory, besides the 3 python scripts that we will create for our project, we must have saved on it the Facial Classifier.

Next, create a subdirectory where we will store our facial samples and name it "dataset":

**mkdir dataset**

**import cv2**

**import os**

**cam = cv2.VideoCapture(0)**

**cam.set(3, 640) # set video width**

**cam.set(4, 480) # set video height**

**face\_detector = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')**

**# For each person, enter one numeric face id**

**face\_id = input('\n enter user id end press <return> ==> ')**

**print("\n [INFO] Initializing face capture. Look the camera and wait ...")**

**# Initialize individual sampling face count**

**count = 0**

**while(True):**

**ret, img = cam.read()**

**img = cv2.flip(img, -1) # flip video image vertically**

**gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**faces = face\_detector.detectMultiScale(gray, 1.3, 5)**

**for (x,y,w,h) in faces:**

**cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)**

**count += 1**

**# Save the captured image into the datasets folder**

**cv2.imwrite("dataset/User." + str(face\_id) + '.' + str(count) + ".jpg", gray[y:y+h,x:x+w])**

**cv2.imshow('image', img)**

**k = cv2.waitKey(100) & 0xff # Press 'ESC' for exiting video**

**if k == 27:**

**break**

**elif count >= 30: # Take 30 face sample and stop video**

**break**

**# Do a bit of cleanup**

**print("\n [INFO] Exiting Program and cleanup stuff")**

**cam.release()**

**cv2.destroyAllWindows()**

The code is very similar to the code that we saw for face detection. What we added, was an "input command" to capture a user id, that should be an integer number (1, 2, 3, etc)

**face\_id = input('\n enter user id end press ==> ')**

And for each one of the captured frames, we should save it as a file on a "dataset" directory:

**cv2.imwrite("dataset/User." + str(face\_id) + '.' + str(count) + ".jpg", gray[y:y+h,x:x+w])**

Note that for saving the above file, you must have imported the library "os". Each file's name will follow the structure:

**User.face\_id.count.jpg**

For example, for a user with a face\_id = 1, the 4th sample file on dataset/ directory will be something like:

**User.1.4.jpg**

Run the Python script and capture a few Ids. You must run the script each time that you want to aggregate a new user (or to change the photos for one that already exists).

### 3.4.2 Data Trainer

On this second phase, we must take all user data from our dataset and "trainer" the OpenCV Recognizer. This is done directly by a specific OpenCV function. The result will be a .yml file that will be saved on a "trainer/" directory.

So, let's start creating a subdirectory where we will store the trained data:

**mkdir trainer**

**import cv2**

**import numpy as np**

**from PIL import Image**

**import os**

**# Path for face image database**

**path = 'dataset'**

**recognizer = cv2.face.LBPHFaceRecognizer\_create()**

**detector = cv2.CascadeClassifier("haarcascade\_frontalface\_default.xml");**

**# function to get the images and label data**

**def getImagesAndLabels(path):**

**imagePaths = [os.path.join(path,f) for f in os.listdir(path)]**

**faceSamples=[]**

**ids = []**

**for imagePath in imagePaths:**

**PIL\_img = Image.open(imagePath).convert('L') # convert it to grayscale**

**img\_numpy = np.array(PIL\_img,'uint8')**

**id = int(os.path.split(imagePath)[-1].split(".")[1])**

**faces = detector.detectMultiScale(img\_numpy)**

**for (x,y,w,h) in faces:**

**faceSamples.append(img\_numpy[y:y+h,x:x+w])**

**ids.append(id)**

**return faceSamples,ids**

**print ("\n [INFO] Training faces. It will take a few seconds. Wait ...")**

**faces,ids = getImagesAndLabels(path)**

**recognizer.train(faces, np.array(ids))**

**# Save the model into trainer/trainer.yml**

**recognizer.write('trainer/trainer.yml') # recognizer.save() worked on Mac, but not on Pi**

**# Print the numer of faces trained and end program**

**print("\n [INFO] {0} faces trained. Exiting Program".format(len(np.unique(ids))))**

Confirm if you have the PIL library installed on your R-Pi. If not, run the below command in Terminal:

**pip install pillow**

We will use as a recognizer, the LBPH (LOCAL BINARY PATTERNS HISTOGRAMS) Face Recognizer, included on OpenCV package. We do this in the following line:

**recognizer = cv2.face.LBPHFaceRecognizer\_create()**

The function "getImagesAndLabels (path)", will take all photos on directory: "dataset/", returning 2 arrays: "Ids" and "faces". With those arrays as input, we will "train our recognizer":

**recognizer.train(faces, ids)**

As a result, a file named "trainer.yml" will be saved in the trainer directory that was previously created by us.

Every time that you perform Phase 1, Phase 2 must also be run.

### 3.4.3 Data Recognizer

Now, we reached the final phase of our project. Here, we will capture a fresh face on our camera and if this person had his face captured and trained before, our recognizer will make a "prediction" returning its id and an index, shown how confident the recognizer is with this match.

**import cv2**

**import numpy as np**

**import os**

**recognizer = cv2.face.LBPHFaceRecognizer\_create()**

**recognizer.read('trainer/trainer.yml')**

**cascadePath = "haarcascade\_frontalface\_default.xml"**

**faceCascade = cv2.CascadeClassifier(cascadePath);**

**font = cv2.FONT\_HERSHEY\_SIMPLEX**

**#iniciate id counter**

**id = 0**

**# names related to ids: example ==> Shrey: id=1, etc**

**names = ['None', 'Shrey', 'Vaishakh', 'Divyanshu', 'Sahil']**

**# Initialize and start realtime video capture**

**cam = cv2.VideoCapture(0)**

**cam.set(3, 640) # set video widht**

**cam.set(4, 480) # set video height**

**# Define min window size to be recognized as a face**

**minW = 0.1\*cam.get(3)**

**minH = 0.1\*cam.get(4)**

**while True:**

**ret, img =cam.read()**

**img = cv2.flip(img, -1) # Flip vertically**

**gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)**

**faces = faceCascade.detectMultiScale(**

**gray,**

**scaleFactor = 1.2,**

**minNeighbors = 5,**

**minSize = (int(minW), int(minH)),**

**)**

**for(x,y,w,h) in faces:**

**cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)**

**id, confidence = recognizer.predict(gray[y:y+h,x:x+w])**

**# Check if confidence is less them 100 ==> "0" is perfect match**

**if (confidence < 100):**

**id = names[id]**

**confidence = " {0}%".format(round(100 - confidence))**

**else:**

**id = "unknown"**

**confidence = " {0}%".format(round(100 - confidence))**

**cv2.putText(img, str(id), (x+5,y-5), font, 1, (255,255,255), 2)**

**cv2.putText(img, str(confidence), (x+5,y+h-5), font, 1, (255,255,0), 1)**

**cv2.imshow('camera',img)**

**k = cv2.waitKey(10) & 0xff # Press 'ESC' for exiting video**

**if k == 27:**

**break**

**# Do a bit of cleanup**

**print("\n [INFO] Exiting Program and cleanup stuff")**

**cam.release()**

**cv2.destroyAllWindows()**

We are including here a new array, so we will display "names", instead of numbered ids:

**names = ['None', 'Shrey', 'Vaishakh', 'Divyanshu', 'Sahil']**

So, for example: Shrey will the user with id = 1; Vaishakh: id=2, etc.

Next, we will detect a face, same we did before with the Haar-Cascade classifier. Having a detected face we can call the most important function in the above code:

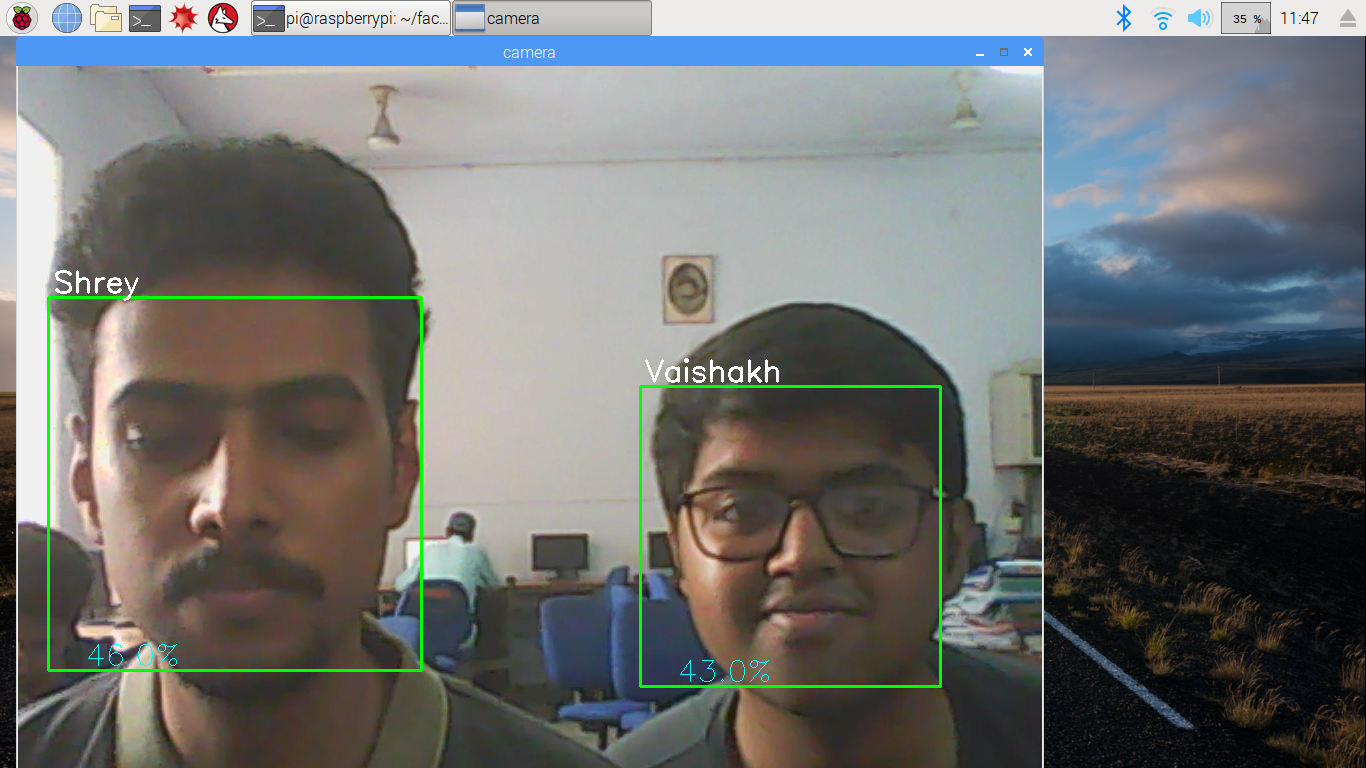
**id, confidence = recognizer.predict(gray portion of the face)**

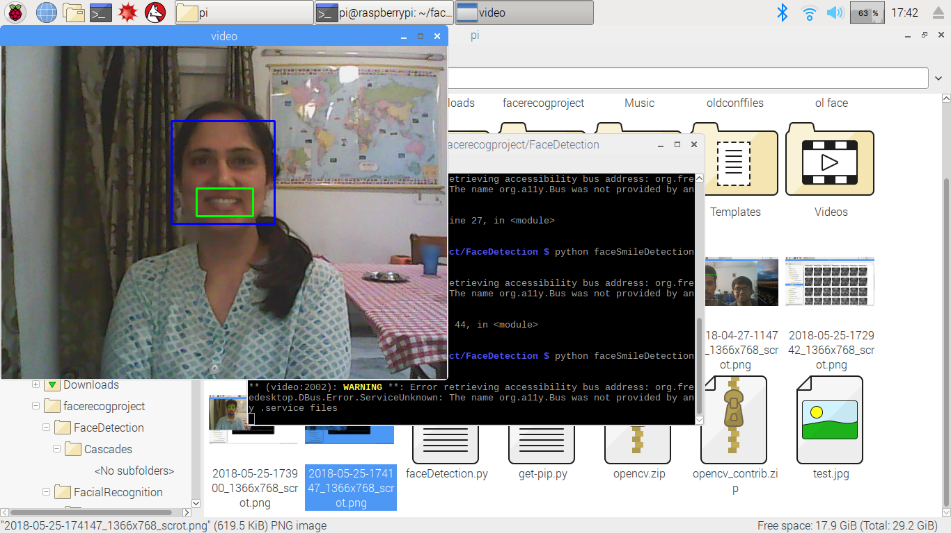
The recognizer.predict (), will take as a parameter a captured portion of the face to be analyzed and will return its probable owner, indicating its id and how much confidence the recognizer is in relation with this match.

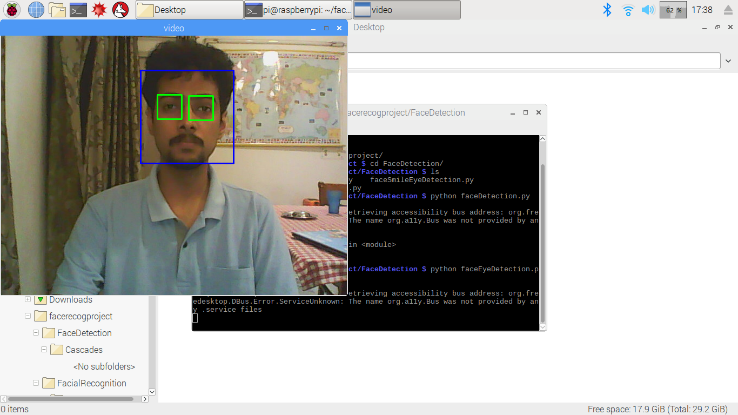
Note that the confidence index will return "zero" if it will be considered a perfect match

And at last, if the recognizer could predict a face, we put a text over the image with the probable id and how much is the "probability" in % that the match is correct ("probability" = 100 - confidence index). If not, an "unknown" label is put on the face.

**3.5 Screenshots:**

****

****

****

1. **FUTURE SCOPE**

The USP of the product is that it is based on open source, freely available code, which can be modified in any way possible. While the current functions of the product are limited to "track and report" functions only, by means of OpenCV and Dropbox, OpenCV can be implemented at a deeper level to provide features like cross referencing face recognition data with official records to perform highly advanced attendance monitoring, and audience mapping functions.

These can be further integrated with embedded functions of the Raspberry Pi, which can run protected web servers, which further reduces its dependence on third party applications like Dropbox and can effectively serve as a standalone solution for surveillance, with no subscription fees, top-of-the line features, all in the same, credit-card sized form factor. The current implementation of the project is a proof-of-concept. If the project needs to be scaled up, higher quality cameras can be used with improved resolution. Moreover, different types of cameras, like infrared cameras can be used to widen the operating capabilities of the system.

1. **CONCLUSION**

This project achieves the goal of creating a low-cost face recognition, which is open source, extendible, and easy to implement. The product, if brought to market, will cost less than the cheapest offerings by established companies, while offering features that can be compared to the mid-high range of products in the market. Because all the code of the project is open source or developed in-house, it can easily be modified to handle advanced features, customisations, personalised user requirements, etcetera.

As the product is self-contained, it is easy to implement, and can be easily brought to everyday households, corporate buildings, and in public spaces where face recognition may be required. If implement in a major scale, the product can be instrumental in acquiring demographics in large audience spaces, like sporting events, festivals, etc. The product can also be used by law enforcement agencies.

**5. REFERENCES**

* Makinen ,E., Raisamo,R.,2002 Real time face detection for Kiosk interfaces. In :Proc .APCHI 2002, Beijing, China,pp.528-539.
* Hjelmas, E.,Low,B.K.,2001 Face detection: A survey. Computer vision and image understanding 83, 236-274.
* P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Computer* Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, 2001, vol. 1, pp. I-511-518.
* Erno and Roope “An experimental comparision of gender classification methods,” Pattern recognition 2008,pp:1544-1556.
* Derek Molloy, *Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux*
* Warren Gay, *Mastering the Raspberry Pi*
* Prateek Joshi, *OpenCV with Python By Example*