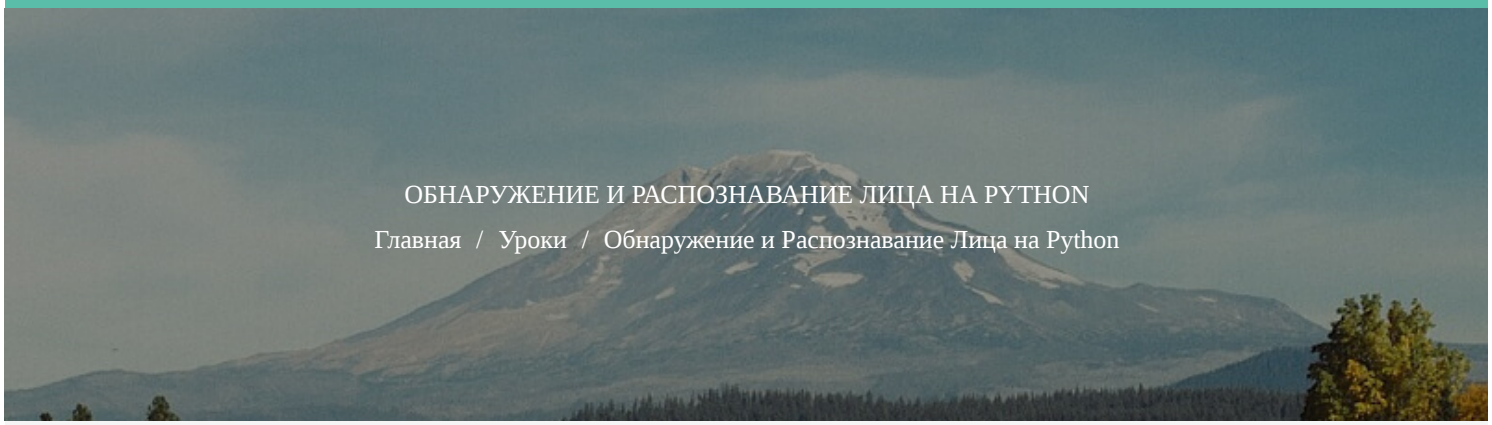


ОБНАРУЖЕНИЕ И РАСПОЗНАВАНИЕ ЛИЦА НА PYTHON

Вторник, 03 апреля 2018 20:31

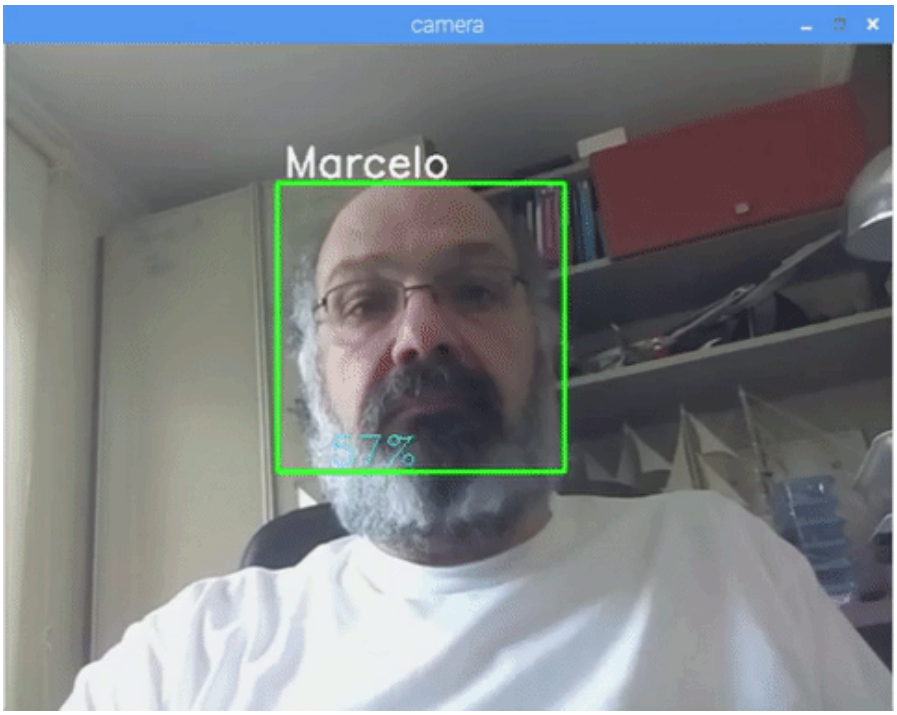
УРОКИ

ОБНАРУЖЕНИЕ И РАСПОЗНАВАНИЕ ЛИЦА НА PYTHON

Bender 2 комментарии

Оцените материал (10 голосов)

В предыдущем уроке, посвященном OpenCV, мы рассказали, как отслеживать цветной объект в реальном времени. Теперь мы будем использовать PiCam для распознавания лица в режиме реального времени с помощью Python, как вы можете видеть ниже:

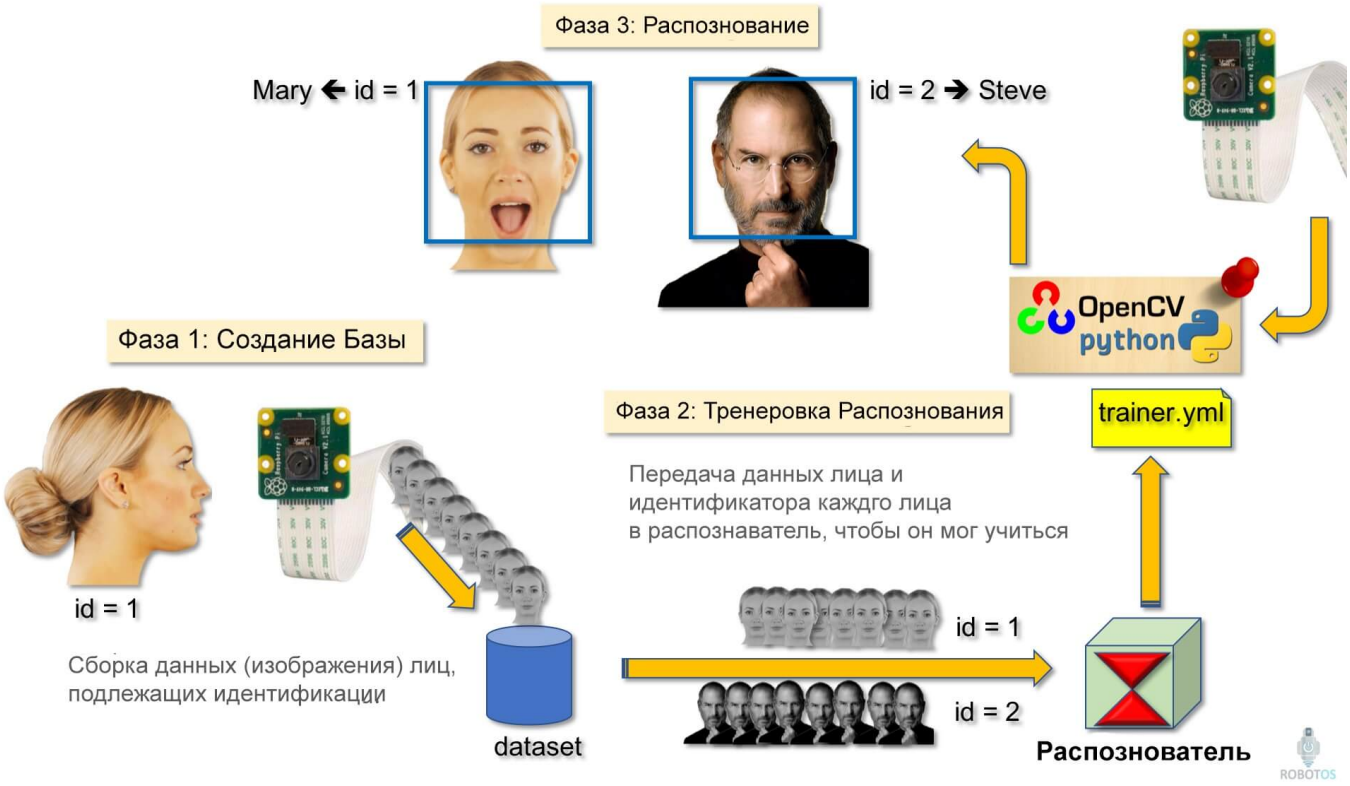


Этот проект был выполнен с помощью «Open Source Computer Vision Library», OpenCV. В этом уроке мы сосредоточимся на Raspberry Pi (так, Raspbian как OS) и Python, но я также проверил код на My Mac, и он также отлично работает. OpenCV был разработан для эффектного вычисления и работы в реальном времени. Таким образом, он идеально подходит для распознавания лиц в реальном времени с помощью камеры.

Чтобы создать полный проект по распознаванию лиц, мы должны работать на 3 очень разных этапах:

1. Обнаружение лиц и сбор данных
2. Обучение распознавателя
3. Распознавание лица

На приведенной ниже блок-схеме показаны этапы:

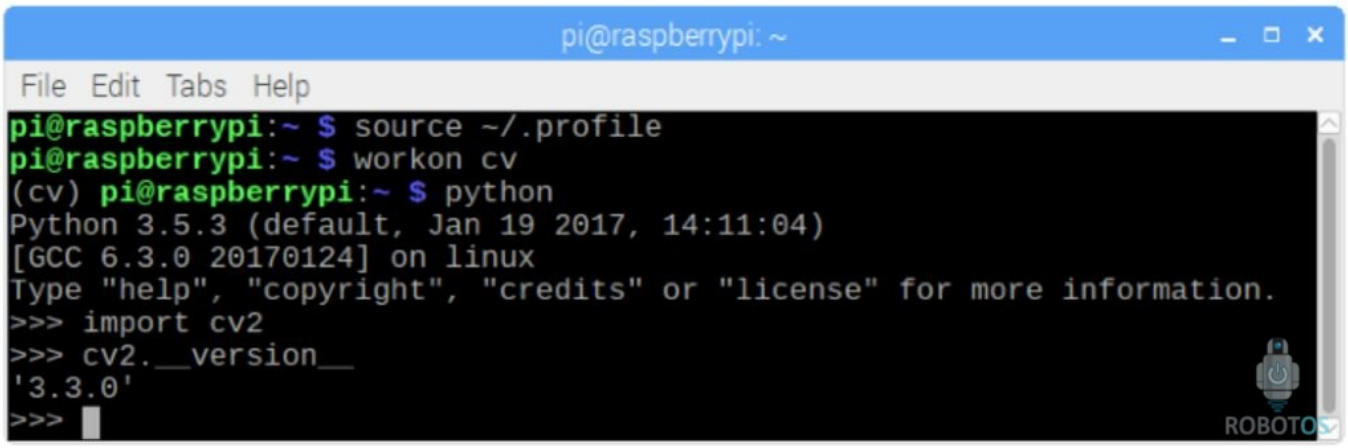


Шаг 1: Список деталей

Raspberry Pi V3 - US$ 32.00 / Raspberry Pi V3 - US$ 36.99

5 Megapixels 1080p Sensor OV5647 Mini Camera Video Module - US$ 13.00 / 5 Megapixels 1080p Sensor OV5647 Mini Camera Video Module - US$ 6.40

Шаг 2: Установка пакета OpenCV 3



Я использую Raspberry Pi V3, обновленный до последней версии Raspbian (Stretch), поэтому лучший способ установить OpenCV - следовать отличному учебному пособию, разработанному Адрианом Розброком: Raspbian Stretch: Установка OpenCV 3 + Python на Raspberry Pi V3.

> Я попробовал несколько разных руководств по установке OpenCV на моем Pi. Учебник Адриана - лучший. Я советую вам сделать то же самое, следуя его указаниям шаг за шагом.

После того, как вы закончите установку по учебнику Адриана, у Вас должна быть виртуальная среда OpenCV, готовая запускать наш эксперимент на Raspberry Pi.

Перейдем к нашей виртуальной среде и убедитесь, что OpenCV 3 правильно установлен.

Адриан рекомендует запускать команду «source» каждый раз, когда вы открываете новый терминал, чтобы обеспечить правильную настройку системных переменных.

```
source ~/.profile
```

Затем перейдем к нашей виртуальной среде:

```
workon cv
```

Если вы видите текст (cv), предшествующий вашему запросу, то вы находитесь в *виртуальной среде cv*:

```
(cv) pi@raspberry:~$
```

> Адриан обращает внимание, что ***виртуальная среда cv Python*** полностью независима и секвестрирована из стандартной версии Python, включенной в загрузку Raspbian Stretch. Таким образом, любые пакеты Python в каталоге глобальных пакетов сайтов не будут доступны для виртуальной среды cv. Аналогично, любые пакеты Python, установленные в сайтах-пакетах cv, не будут доступны для глобальной установки Python.

Теперь введите интерпретатор Python:

```
python
```

и убедитесь, что вы используете версию 3.5 (или выше)

Внутри интерпретатора (появится «>>>») импортируйте библиотеку OpenCV:

```
import cv2
```

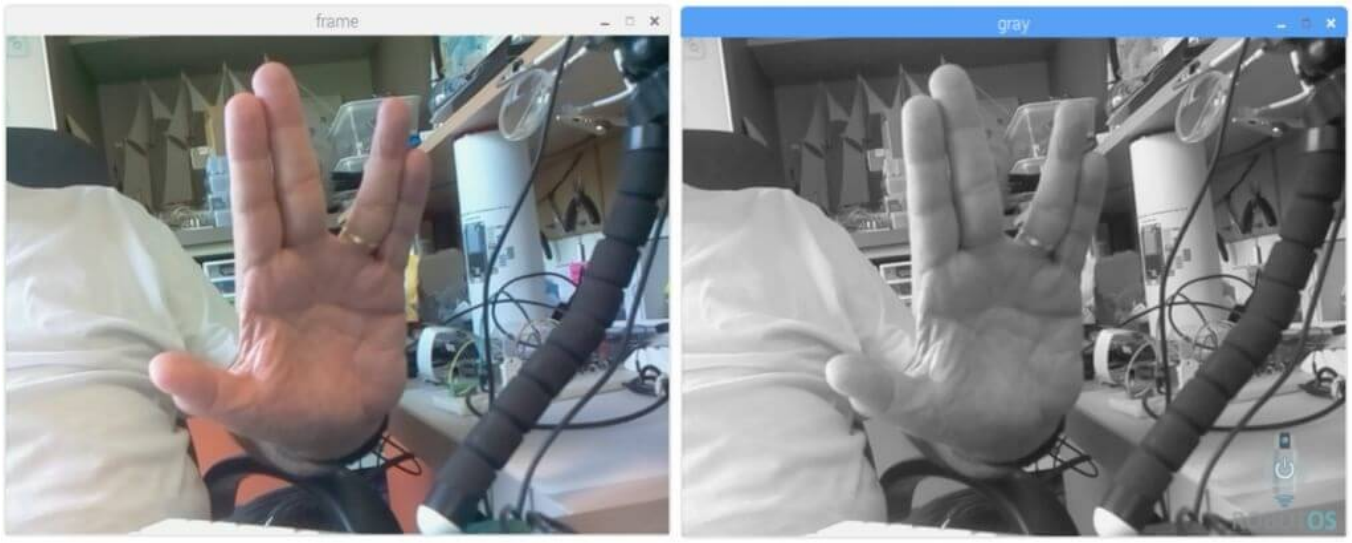
Если сообщений об ошибках не выходят, значит OpenCV правильно установлен на ВАШЕЙ PYTHON ВИРТУАЛЬНОЙ СРЕДЕ.

Вы также можете проверить какая версия OpenCV установлена:

```
cv2.__version__
```

Должен появиться 3.3.0 (или более высокая версия, которая может быть выпущена в будущем). Вышеупомянутый терминал PrintScreen показывает предыдущие шаги.

Шаг 3: Тестирование Вашей Камеры



Как только вы установите OpenCV в свой RPi, давайте проверим, работает ли ваша камера правильно.

> Я предполагаю, что у вас есть PiCam, уже установленный на вашем Raspberry Pi.

Введите ниже код Python в вашу среду IDE:

```
import numpy as np
import cv2
```

```
cap = cv2.VideoCapture(0)

while(True):
    ret, frame = cap.read()
    frame = cv2.flip(frame, -1) # Flip camera vertically
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow('frame', frame)
    cv2.imshow('gray', gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Вышеприведенный код будет захватывать видеопоток, который будет создан вашим PiCam, и отображать его как в цвете BGR, так и в режиме Gray.

> Обратите внимание, что я повернул камеру по вертикали из-за того, что она собрана. Если это не ваше дело, прокомментируйте или удалите командную строку «flip».

Вы также можете загрузить код из GitHub: simpleCamTest.py

Для выполнения введите команду:

```
python simpleCamTest.py
```

Чтобы закончить программу, вы должны нажать клавишу [ESC] на клавиатуре.

> Щелкните мышью на окне видео, прежде чем нажимать [ESC]

На приведенном выше рисунке показан результат.

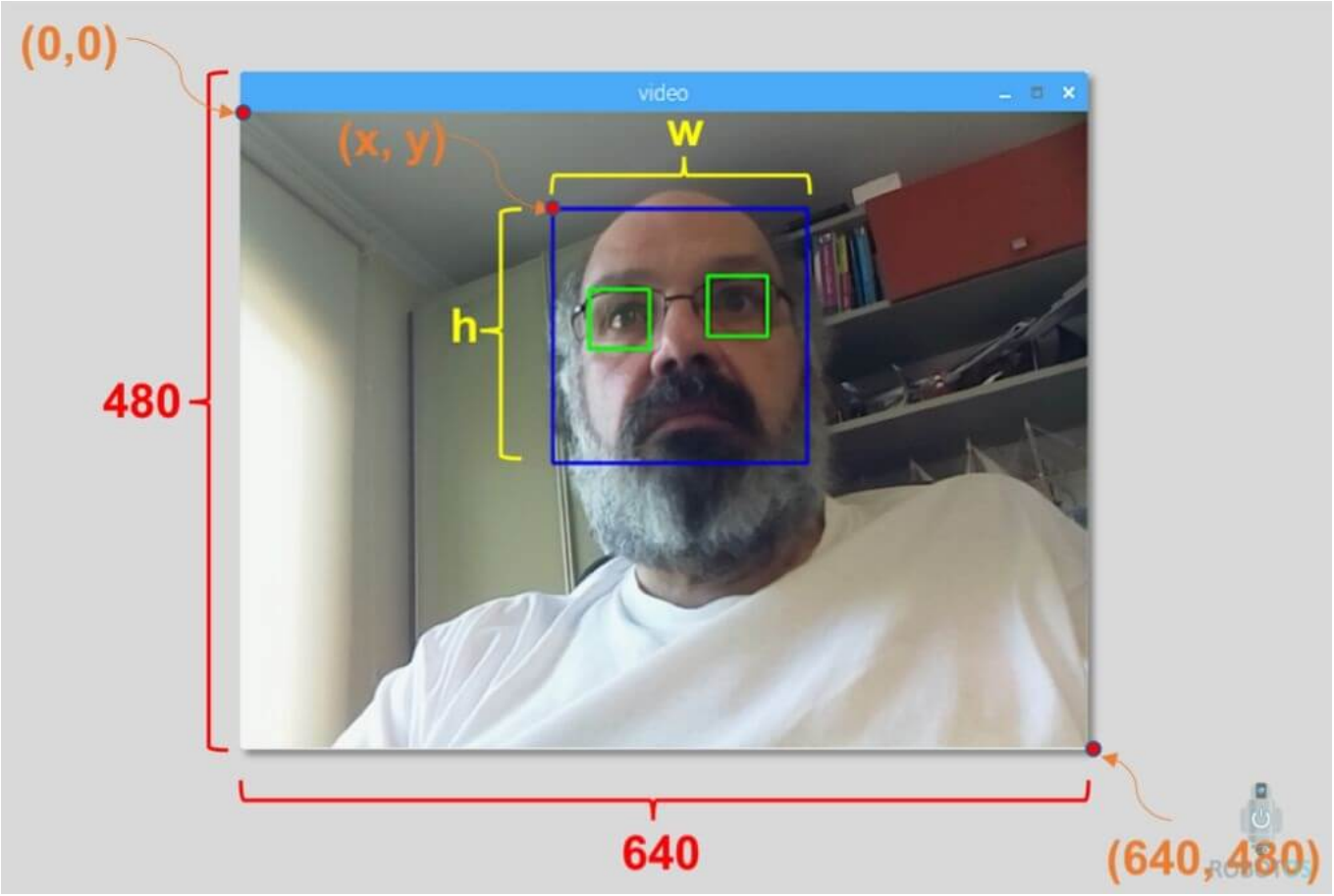
Некоторые разработчики обнаружили проблемы при попытке открыть камеру (сообщение об ошибке «Ошибка подтверждения»). Это может произойти, если камера не была включена во время установки OpenCv, поэтому драйверы камеры не установлены правильно. Чтобы исправить, используйте команду:

```
sudo modprobe bcm2835-v4l2
```

Вы также можете добавить bcm2835-v4l2 в последнюю строку файла / etc / modules, чтобы драйвер загружался при загрузке.

Чтобы узнать больше о OpenCV, Вы можете ознакомитьсяс уроком: загрузка -video-python-opencv-tutorial

Шаг 4: Определение Лица



Самой основной задачей Распознавания Лица является, конечно же, «Определения Лица». Прежде всего, вы должны «захватить» лицо, чтобы распознать его, по сравнению с новым лицом, захваченным в будущем.

Наиболее распространенный способ обнаружения лица (или любых объектов), использует «классификатор каскадов Хаар».

Обнаружение объектов с использованием «классификатора каскадов Хаар» - эффективный метод обнаружения объектов, предложенный Полом Виолой и Майклом Джонсом в своей статье «Быстрое обнаружение объектов с использованием расширенного каскада простых функций» в 2001 году. Это подход, основанный на механизме обучения, каскадная функция обучается из множества положительных и отрицательных изображений. Затем он используется для обнаружения объектов на других изображениях.

В уроке мы будем работать с распознаванием лиц. Первоначально алгоритм требует много положительных изображений (изображений лиц) и негативных изображений (изображений без лиц) для обучения классификатора. Затем нам нужно

извлечь из него функции. Хорошей новостью является то, что OpenCV поставляется с тренером, а также с детектором. Если вы хотите обучить свой классификатор для любого объекта, такого как автомобиль, самолеты и т. Д., Вы можете использовать OpenCV для его создания. Его полная информация приведена здесь: Обучение каскадного классификатора.

Если вы не хотите создавать свой собственный классификатор, OpenCV уже содержит много предварительно подготовленных классификаторов для лица, глаз, улыбки и т. Д. Эти файлы XML можно загрузить из каталога haarcascades.

Достаточно теории, давайте создадим детектор лица с OpenCV!

Загрузите файл: faceDetection.py из GitHub.

```
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()
```

Верьте или нет, несколько строк кода - все, что вам нужно, чтобы обнаружить лицо, используя Python и OpenCV.

Когда вы сравниваете последний код, используемый для проверки камеры, вы поймете, что к нему добавили несколько частей. Обратите внимание на следующую строку:

```
faceCascade = cv2.CascadeClassifier('Cascades/haarcascade_frontalface_default.xml')
```

Это строка, которая загружает «классификатор» (который должен находиться в каталоге с именем «Cascades /» в каталоге вашего проекта).

Затем мы установим нашу камеру и внутри цикла, загрузим входное видео в режиме серого (то же, что мы видели раньше).

Теперь мы должны назвать нашу классификаторную функцию, передав ей некоторые очень важные параметры, такие как масштабный коэффициент, количество соседей и минимальный размер обнаруженного лица.

```
faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20, 20)
    )
```

Где,

- **grey**- это входное изображение в оттенках серого.

- **scaleFactor** - это параметр, определяющий размер изображения при каждой шкале изображения. Он используется для создания масштабной пирамиды.
- **minNeighbors** - параметр, указывающий, сколько соседей должно иметь каждый прямоугольник кандидата, чтобы сохранить его. Более высокое число дает более низкие ложные срабатывания.
- **minSize** - минимальный размер прямоугольника, который считается лицом.

Функция будет определять лица на изображении. Затем мы должны «маркировать» лица на изображении, используя, например, синий прямоугольник. Это делается с помощью этой части кода:

```
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]
```

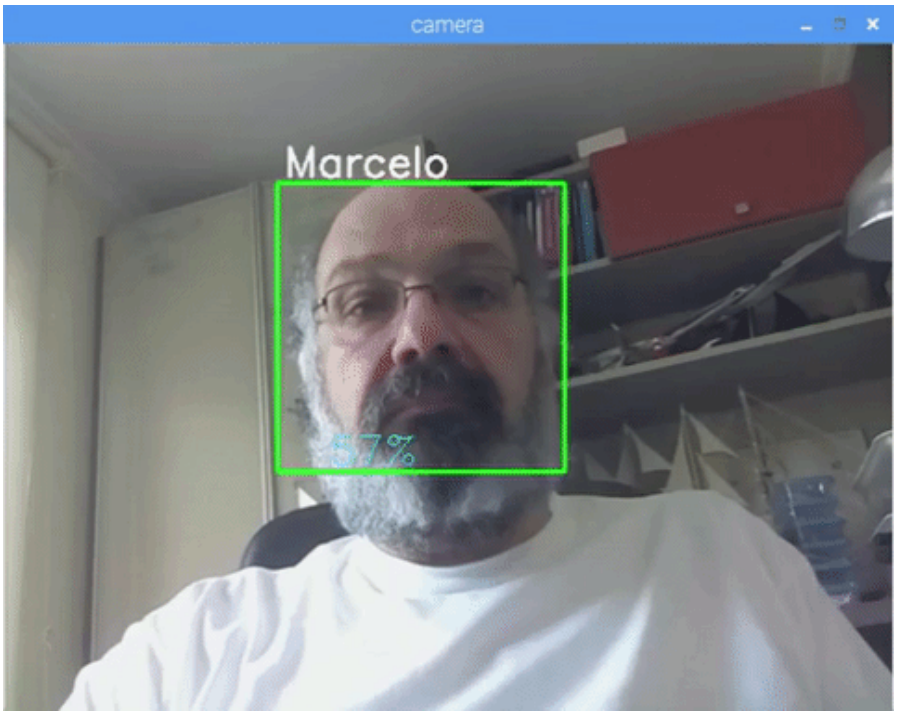
Если грани найдены, они возвращают позиции обнаруженных лиц в виде прямоугольника с левым углом (x, y) и имеют «w» в качестве его ширины и «h» как его высоту ==> (x, y, w, час). См. Приведенную выше картинку.

Как только мы получим эти координаты, мы можем создать «ROI» (рисованный прямоугольник) для лица и представить результат с помощью функции *imshow* ()

Запустите вышеупомянутый скрипт python в вашей среде python, используя терминал Rpi:

```
python faceDetection.py
```

Результат:



Вы также можете включить классификаторы для «обнаружения глаз» или даже «обнаружения улыбки». В этих случаях вы будете включать функцию классификатора и рисовать прямоугольник внутри контура лица, потому что не было бы смысла обнаруживать глаз или улыбку за пределами лица.

> Обратите внимание, что на Pi, имея несколько классификаторов в одном коде, будет замедляться обработка, как только этот метод обнаружения (HaarCascades) использует большую вычислительную мощность. На системном блоке легче запустить его.

На GitHub вы найдете другие примеры:

faceEyeDetection.py

faceSmileDetection.py

faceSmileEyeDetection.py

И в приведенной выше картине вы можете увидеть результат.

Вы также можете изучить приведенный ниже учебник, чтобы лучше понять функцию как происходит распознавания лиц:

Обнаружение объекта Haar Cascade Face & Eye OpenCV Python

Шаг 5: Сбор данных

Прежде всего, я должен поблагодарить Рамиза Раджи за его великую работу над Распознаванием Лица на фотографиях:

РАСПОЗНОВАНИЕ ЛИЦА С ИСПОЛЬЗОВАНИЕМ OPENCV И ПИТОНА: РУКОВОДСТВО ДЛЯ НАЧИНАЮЩИХ

а также Anirban Kar, который разработал очень всеобъемлющий учебник с использованием видео:

РАСПОЗНОВАНИЕ ЛИЦА – 3 ЧАСТИ

Я очень рекомендую Вам взглянуть на оба учебника.

Давайте начнем первый этап нашего проекта. То, что мы будем делать здесь, начинается с последнего шага (Определение Лица), мы просто создадим набор данных, где мы будем хранить для каждого идентификатора, группу фотографий в сером цвете с той частью, которая использовалась для обнаружения лица.

Сначала создайте каталог, в котором вы разрабатываете свой проект, например, FacialRecognitionProject:

```
mkdir FacialRecognitionProject
```

В этом каталоге, помимо 3 сценариев python, которые мы создадим для нашего проекта, мы должны будем сохранить на нем классификатор лица. Вы можете скачать его с моего GitHub:

haarcascade_frontalface_default.xml

Затем создайте подкаталог, в котором мы будем хранить наши образцы лица и назовите его «dataset»:

```
mkdir dataset
```

И загрузите код из GitHub: 01_face_dataset.py

```
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.x
ml')

# For each person, enter one numeric face id
face_id = input('\n enter user id end press  ==>  ')

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()
```

Код очень похож на код, который мы писали для обнаружения лиц. Мы добавили, что это была «команда ввода» для захвата идентификатора пользователя, который должен быть целым числом (1, 2, 3 и т. Д.),

```
face_id = input('\n enter user id end press ==> ')
```

И для каждого из захваченных кадров мы должны сохранить его как файл в каталоге «dataset»:

```
cv2.imwrite("dataset/User." + str(face_id) + '.' + str(count) + ".jpg",
 gray[y:y+h,x:x+w])
```

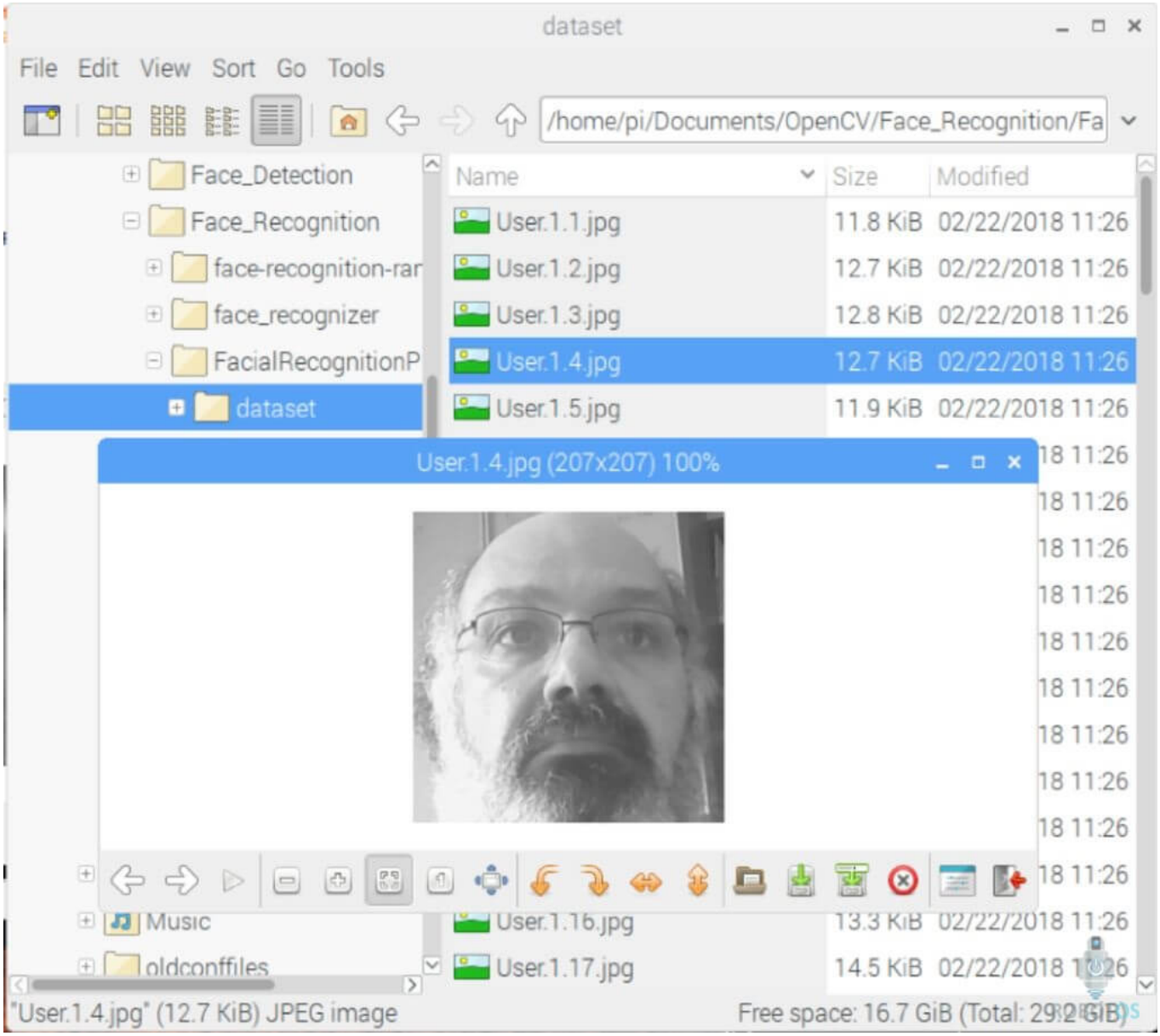
Обратите внимание, что для сохранения вышеуказанного файла необходимо импортировать библиотеку «os». Имя каждого файла будет следовать за структурой:

```
User.face_id.count.jpg
```

Например, для пользователя с face_id = 1, 4-й пример файла в каталоге dataset / будет выглядеть примерно так:

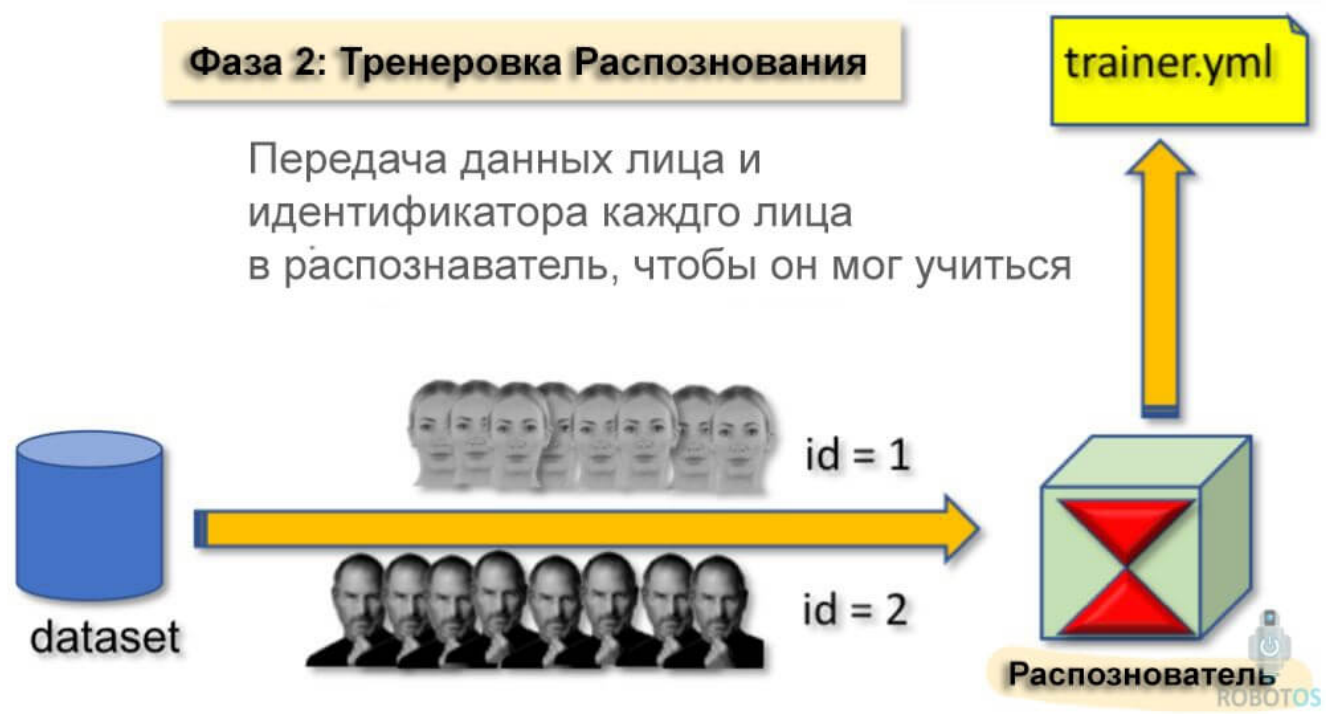
```
User.1.4.jpg
```

Как показано на фотографии моего Pi. В моем коде я собираю 30 образцов из каждого идентификатора. Вы можете изменить его на последнем «elif». Число выборок используется для разрыва цикла, в котором захватываются образцы лица.



Запустите скрипт Python и запишите несколько идентификаторов. Вы должны запускать скрипт каждый раз, когда хотите сгенерировать нового пользователя (или изменить фотографии для уже существующего).

Шаг 6: Тренировка



На этом втором этапе мы должны взять все пользовательские данные из нашего набора данных и «инструктор» в OpenCV Recognizer. Это делается непосредственно с помощью определенной функции OpenCV. Результатом будет файл .yml, который будет сохранен в каталоге «trainer /».

Итак, давайте начнем создание подкаталога, в котором мы будем хранить подготовленные данные:

```
mkdir trainer
```

Загрузите с GitHub второй скрипт python: 02_face_training.py

```
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 gra
yscale
        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 Ma
c, but not on Pi

# Print the numer of faces trained and end program
print("\n [INFO] {0} faces trained. Exiting Program".format(len(np.uniqu
e(ids))))
```

Убедитесь, что у вас установлена библиотека PIL на вашем Raspberry Pi. Если нет, запустите следующую команду в терминале:

```
pip install pillow
```