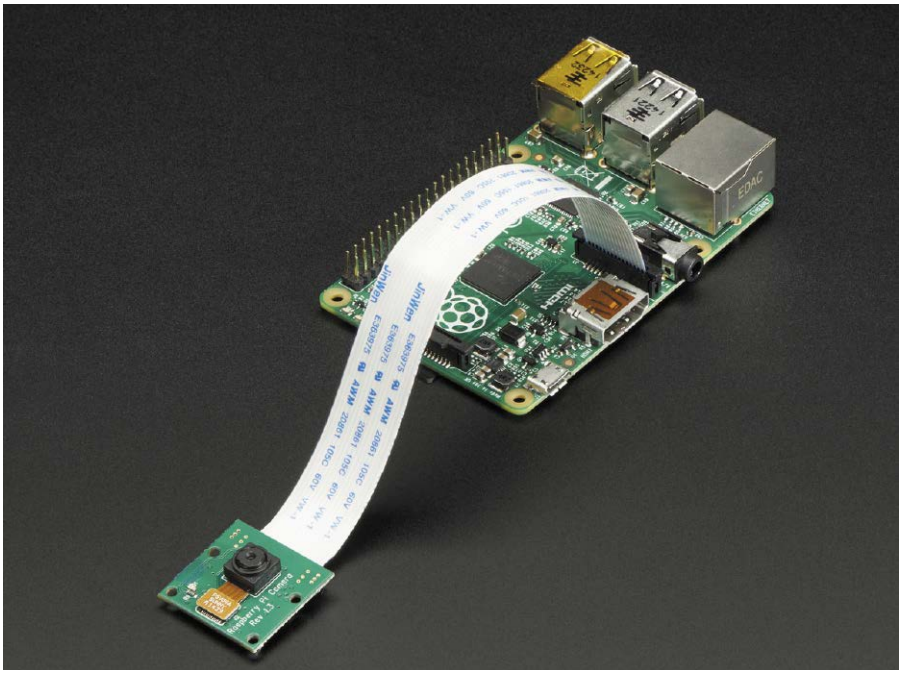
Sejf z rozpoznawaniem twarzy z Raspberry Pi

Moc obliczeniowa Raspberry Pi, choć nie dorównuje nowoczesnym komputerom PC, ani nawet wielu smartfonom, jest wystarczająco duża do realizacji różnych ciekawych zadań. Natomiast możliwość korzystania z gotowych, darmowych, otwartych, zaawansowanych bibliotek i niedrogich, specjalizowanych modułów sprawia, że wykonanie złożonych projektów okazuje się bardzo proste. W artykule pokazujemy taki przykład: sejf z kamerą, który otwiera się po rozpoznaniu odpowiedniej twarzy.

Potrzebne komponenty:
- Pudełko.
- Raspberry Pi.
- Moduł Raspberry Pi Camera.
- Uniwersalna płytka drukowana do wykonania połączeń.
- Serwomechanizm lub zamek z solenoidem.
- Przycisk monostabilny.
- Rezystor 10 kΩ.
- Zasilacz 5 V z wyjściem USB.
- Pakiet 4 baterii AA do zasilania serwomechanizmu.
- Klej, opaski montażowe.
- Przewody do wykonania połączeń.



Fotografia 1. Moduł Raspberry Pi camera, podłączony do Raspberry Pi

Naturalnie samodzielne wykonanie dobrego sejfu nie jest takie łatwe, ale wynika to przede wszystkim z trudności precyzyjnego dopracowania mechaniki. Porządny sejf musi mieć mocny zamek, grube ścianki i brak luk, które umożliwiłyby jego niepowołane otwarcie lub uniemożliwiłyby skorzystanie przez odpowiednie osoby.

Działanie sejfu

Prezentowany projekt został wykonany przez Tony DiColę i składa się z pudełka i komputera wraz z modułem kamery, a także elementu wykonawczego – np. serwomechanizmu, który rygluje zamek pokrywy pudełka. W pudełku umieszczony jest też pakiet baterii, które służą do zasilania napędu zamka, podczas gdy z założenia, sam komputer zasilany jest z zewnętrznego zasilacza. Oznacza to, że do otwarcia pudełka konieczne jest podłączenie go do prądu, co jest trochę bez sensu w przypadku urządzeń przenośnych, ale modyfikacja projektu tak, by zawierał dodatkowe, przenośne, ładowalne źródło napięcia dla komputera nie jest trudna. Poza tym taką samą konstrukcję pod względem elektroniki i oprogramowania, można by było umieścić w futrynie drzwi, pomijając problem zasilania.

Działanie sejfu jest bardzo proste. Użytkownik chcąc otworzyć pudełko, bierze je w ręce i ustawia tak, by obiektyw skierowany był w stronę twarzy człowieka, a następnie naciska wbudowany przycisk. Komputer robi zdjęcie człowieka z użyciem kamery i zaczyna je analizować. Jeśli na zdjęciu znaleziona zostanie twarz i będzie ona wystarczająco podobna do twarzy wzorcowej, uruchomiony zostanie serwonapęd, by odryglować zamek. W przeciwnym przypadku zamek pozostanie na swojej pozycji i będzie można spróbować wykonać ponowne zdjęcie. Jeśli zamek będzie otwarty, naciśnięcie przycisku spowoduje zaryglowanie.

Trzeba przy tym zaznaczyć, że projekt został przygotowany tak, by działał dla jednej twarzy, której zdjęcia przygotowano i poddano do treningu dla algorytmu rozpoznawania twarzy.

Potrzebne elementy i narzędzia

Do realizacji omawianego projektu nie potrzeba dużej ilości sprzętu. Wystarczy odpowiednio duże pudełko, najlepiej z materiału nieprzewodzącego prądu i takiego, w którym można nawiercić otwory. W nim umieszczamy komputer Raspberry Pi w dowolnej wersji, ale z modułem Raspberry Pi Camera (**fotografia 1**) o wartości około 30 dolarów. W projekcie autor zastosował miniaturowy serwonapęd Tower Pro SG92R (**fotografia 2**), kosztujący 6 dolarów, którego moment siły to 0,16 Nm, a wymiary to 23×11×29 mm. Alternatywnie można użyć solenoidu do zamka (**fotografia 3**), który – jeśli konstrukcja pudełka jest odpowiednia – może być nawet wygodniejszy w zastosowaniu.

Potrzebny będzie też monostabilny przycisk, 10-kiloomowy rezystor podciągający dla przycisku, 5-woltowy zasilacz USB oraz pakiet 4 baterii 1,5-woltowych, najlepiej w formacie AA, którym zasilany będzie serwonapęd. Oczywiście potrzebny będzie klej i różne elementy montażowe oraz sam rygiel, który można będzie przyczepić do serwonapędu, a także przewody do podłączenia wszystkich elementów. Przydatna może okazać się też uniwersalna płytka, która ułatwi wykonywanie połączeń.

Montaż

W pudełku nawiercamy otwory: na kamerę, przycisk i na podłączenie zasilania do komputera. Może się okazać konieczne wykonanie kolejnych otworów, by przy przyczepić pozostałe elementy we wnętrzu pudełka. Warto tak rozmieścić elementy, by moduł

Fotografia 2. Serwonapęd Tower Pro SG92R

z kamerą znajdował się blisko samego komputera. Zasilanie serwonapędu podłączamy do pakietu baterii, aby nie przeciążać linii zasilania komputera. Linię sterującą serwonapędu podłączamy do pinu GPIO 18. Przycisk do GPIO 25 z użyciem rezystora podciągającego napięcie do 3,3 V, podawanego na jednym z wyprowadzeń Raspberry Pi.

Potrzebne oprogramowanie

Sprawa oprogramowania jest w tym przypadku dosyć skomplikowana. Wynika to z faktu, że do rozpoznawania obrazów stosujemy bibliotekę OpenCV, ale moduł odpowiedzialny za rozpoznawanie twarzy dostępny jest dopiero od wersji 2.4, podczas gdy w repozytoriach Raspbiana znajduje się OpenCV tylko w wersji 2.3. Oznacza to, że konieczne będzie samodzielne skompilowanie źródeł na wybraną platformę. Oprócz tego użyjemy Pythona i jego pakietów. W tym celu instalujemy menedżer pakietów Pythona **PIP**, korzystając z polecenia **APT**:

```
sudo apt-get install python-pip
```

Instalujemy też pakiet **python-dev** oraz dwa moduły pythona: **picamera**, odpowiedzialny za obsługę kamery i **rpio**, który pozwala na sterowanie wyprowadzeniami GPIO z obsługą serwonapędów przez PWM:

```
sudo apt-get install python-dev
sudo pip install picamera
sudo pip install rpio
```

Przejdźmy teraz do kompilacji bibliotek OpenCV. Aktualna najnowsza stabilna wersja bibliotek OpenCV dla Linuksa to 3.0.0 (*http://goo.gl/4nbi68*). Jednakże wystarczy użycie dowolnej wersji 2.4.x (*http://goo.gl/EdOsrv*). Pobieramy wybraną z nich poleceniem **wget**, a następnie rozpakowujemy pobrany plik komendą **unzip**. W celu kompilacji biblioteki OpenCV ze źródeł konieczne jest też zdobycie odpowiednich narzędzi, najlepiej poleceniami:

```
sudo apt-get update
```

Listing 1. Kod programu pobierającego zdjęcia z kamery, konwertującego je do skali szarości, a następnie wykrywającego twarz i po przycięciu obrazu, zapisującego go na dysk do odpowiedniego katalogu

```
„””Raspberry Pi Face Recognition Treasure Box
Positive Image Capture Script
Copyright 2013 Tony DiCola
„””
„”
import glob
import os
import sys
import select
import cv2
import hardware
import config
import face

POSITIVE_FILE_PREFIX = ‚positive_’

def is_letter_input(letter):
    if select.select([sys.stdin,],[],[],0.0)[0]:
        input_char = sys.stdin.read(1)
        return input_char.lower() == letter.lower()
    return False

if __name__ == ‚__main__’:
    camera = config.get_camera()
    box = hardware.Box()
    # Tworzenie katalogu, jeśli nie istnieje
    if not os.path.exists(config.POSITIVE_DIR):
        os.makedirs(config.POSITIVE_DIR)
    # Znajdź ID najnowszego obrazu i zwiększ ID o 1
    files = sorted(glob.glob(os.path.join(config.POSITIVE_DIR,
        POSITIVE_FILE_PREFIX + ‚[0-9][0-9][0-9].pgm’)))
    count = 0
    if len(files) > 0:
        count = int(files[-1][-7:-4])+1
    print ‚Capturing positive training images.’
    print ‚Press button or type c (and press enter) to capture an image.’
    print ‚Press Ctrl-C to quit.’
    while True:
        # Sprawdź, czy naciśnięto przycisk lub wydano polecenie z linii komend
        if box.is_button_up() or is_letter_input(‚c’):
            print ‚Capturing image...’
            # pobierz obraz z kamery
            image = camera.read()
            # Przekonwertuj do skali szarości
            image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
            # Wykryj twarz w pobranym obrazie
            result = face.detect_single(image)
            if result is None:
                print ‚Could not detect single face!  Check the image in capture.pgm’ \
                ‚ to see what was captured and try again with only one face visible.’
                continue
            x, y, w, h = result
            # przytnij obraz do rozmiaru twarzy
            crop = face.crop(image, x, y, w, h)
            # zapisz obraz do pliku
            filename = os.path.join(config.POSITIVE_DIR, POSITIVE_FILE_PREFIX + ‚%03d.pgm’ % count)
            cv2.imwrite(filename, crop)
            print ‚Found face and wrote training image’, filename
            count += 1
```

Źródło projektu
Autorem omawianego projektu jest Tony DiCola, a komplet zdjęć, kod źródłowy i anglojęzyczny opis można znaleźć pod adresem https://goo.gl/g4hvQG.

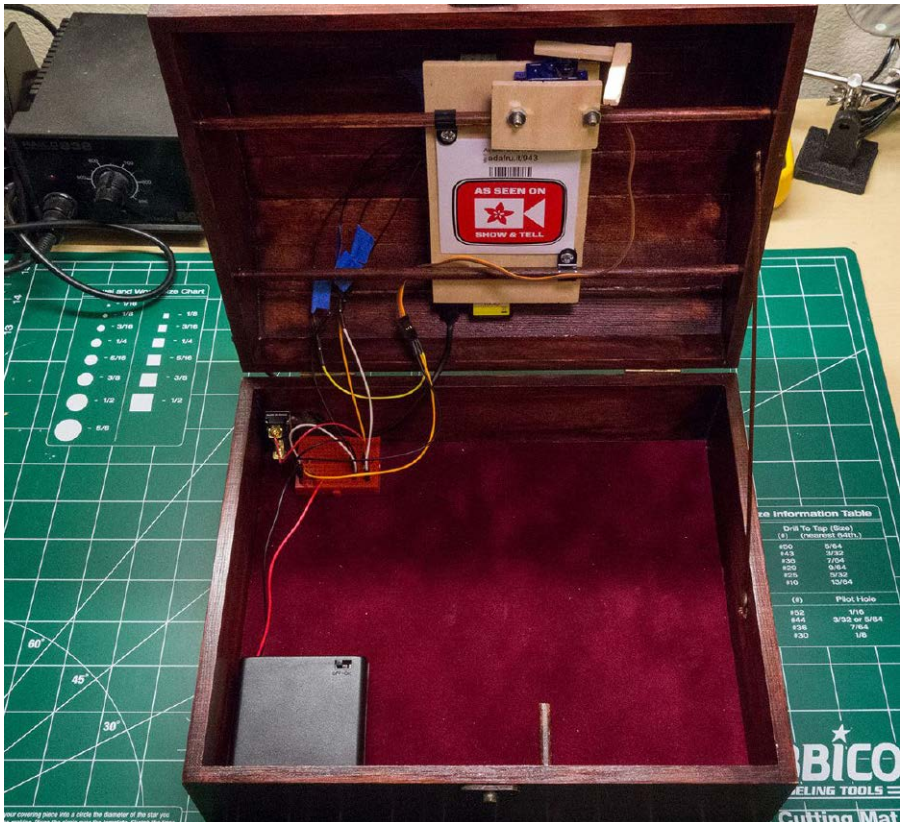
```
sudo apt-get install build-
essential cmake pkg-config python-
dev libgtk2.0-dev libgtk2.0 zlib1g-
dev libpng-dev libjpeg-dev libtiff-
dev libjasper-dev libavcodec-dev
swig unzip
```

W celu przygotowania pliku **make**, który będzie potrzebny do kompilacji, wchodzimy do katalogu z wypakowanym OpenCV i korzystamy z polecenia **cmake**, stosując pewne dodatkowe parametry, które wyłączą testowanie wydajności. Pozostawienie włączonego testowania wydajności sprawiłoby, że czas kompilacji mógłby być zbyt długi, albo operacja w ogóle by się nie udała, ze względu na ograniczony rozmiar dostępnej pamięci. Dlatego polecenie **cmake** należy wywołać w następujący sposób:

Fotografia 3. Zamek z solenoidem

Listing 2. Program trenujący algorytm rozpoznawania twarzy, w oparciu o zbiór fotografii twarzy, która ma odblokowywać zamek i o zbiór różnych innych zdjęć twarzy

```
„””Raspberry Pi Face Recognition Treasure Box
Face Recognition Training Script
Copyright 2013 Tony DiCola
„””
”
import fnmatch
import os
import cv2
import numpy as np
import config
import face

MEAN_FILE = ,mean.png’
POSITIVE_EIGENFACE_FILE = ,positive_eigenface.png’
NEGATIVE_EIGENFACE_FILE = ,negative_eigenface.png’

def walk_files(directory, match=’*’):
    for root, dirs, files in os.walk(directory):
        for filename in fnmatch.filter(files, match):
            yield os.path.join(root, filename)

def prepare_image(filename):
    # wczytywanie i przeskalowanie obrazu
    return face.resize(cv2.imread(filename, cv2.IMREAD_GRAYSCALE))

def normalize(X, low, high, dtype=None):
    # normalizacja macierzy na potrzeby rozpoznawania twarzy
    X = np.asarray(X)
    minX, maxX = np.min(X), np.max(X)
    # normalize to [0...1].
    X = X - float(minX)
    X = X / float((maxX - minX))
    # scale to [low...high].
    X = X * (high-low)
    X = X + low
    if dtype is None:
        return np.asarray(X)
    return np.asarray(X, dtype=dtype)

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

    # właściwe szkolenie
    print ,Training model...’
    model = cv2.createEigenFaceRecognizer()
    model.train(np.asarray(faces), np.asarray(labels))
    # zapisanie modelu
    model.save(config.TRAINING_FILE)
    print ,Training data saved to’, config.TRAINING_FILE
    # zapisanie obrazów pomocniczych twarzy która ma być pozytywnie rozpoznawana
    mean = model.getMat(„mean”).reshape(faces[0].shape)
    cv2.imwrite(MEAN_FILE, normalize(mean, 0, 255, dtype=np.uint8))
    eigenvectors = model.getMat(„eigenvectors”)
    pos_eigenvector = eigenvectors[:,0].reshape(faces[0].shape)
    cv2.imwrite(POSITIVE_EIGENFACE_FILE, normalize(pos_eigenvector, 0, 255, dtype=np.uint8))
    neg_eigenvector = eigenvectors[:,1].reshape(faces[0].shape)
    cv2.imwrite(NEGATIVE_EIGENFACE_FILE, normalize(neg_eigenvector, 0, 255, dtype=np.uint8))
```



Fotografia 4. Gotowy sejf

```
cmake -DCMAKE_BUILD_TYPE=RELEASE
-DCMAKE_INSTALL_PREFIX=/usr/
local -DBUILD_PERF_TESTS=OFF
-DBUILD_opencv_gpu=OFF
-DBUILD_opencv_ocl=OFF
```

Gdy plik **make** będzie przygotowany, należy go uruchomić (czas jego pracy wyniesie około 5 godzin), po czym by zainstalować skompilowane OpenCV w systemie, konieczne jest użycie polecenia:

```
sudo make install.
```

Szkolenie algorytmu

Bardzo ważnym elementem projektu jest trenowanie systemu rozpoznawania twarzy tak, by był w stanie odróżnić pożądanego użytkownika od innych osób. Szkolenie odbywa się poprzez przedstawienie do analizy zarówno przykładów twarzy, które nie mają otwierać sejfu, jak i przykładów jednej twarzy upoważnionego użytkownika. Te pierwsze nazywamy negatywnymi, a drugie – pozytywnymi i umieszczamy je w podkatalogach *training/negative* i *training/positive* (odpowiednio). Auto projektu pobrał zdjęcia twarzy obcych ludzi z udostępnionej bezpłatnie bazy twarzy, opracowanej w latach 90. przez AT&T Laboratories Cambridge. Baza ta dostępna jest pod adresem: *http://goo.gl/s4vfUq* i może się przydać do innych projektów. Bazę fotografii pozytywnych można stworzyć ładując wcześniej przygotowane zdjęcia lub wykorzystać do tego przygotowane pudełko z kamerą i skrypt **capture-positives.py** (**listing 1**). Warto w tym kodzie zwrócić uwagę na użycie funkcji:

- *camera.read()* – pobierającej obraz z kamery,
- *cv2.cvtColor()* – konwertującej obraz do zadanej palety barw (w tym przypadku odcieni szarości),
- *face.detect_single()* – wykrywającej twarz na zadanym obrazie,
- *face.crop()* – przycinającej obraz do wymiarów twarzy,
- *cv2.imwrite()* – zapisującej obraz do pliku.

Część z nich została napisana przez autora na potrzeby projektu, ale jest bardzo uniwersalna i można je z powodzeniem wykorzystać także w innych miejscach. Umiejętność posługiwania się nimi pozwala na robienie zdjęć i zapisywanie ich z wykrywaniem twarzy.

Zdjęcia twarzy powinny być tak przygotowane, by obejmowały ujęcia z różnych kątów, pod którymi ma być ona rozpoznawana. Gdy wszystkie zdjęcia będą zebrane, należy uruchomić program trenujący algorytm. Został on pokazany na **listingu 2**. Jego działanie polega na wczytaniu kolejnych plików i tworzeniu modelu (funkcja *cv2.createEigenFaceRecognizer()*), który następnie jest trenowany (funkcja *model.train()*), po czym zapisywany do pliku (funkcja *model.save()*). Na koniec zapisywane są: uśredniony obraz twarzy pozytywnych, obraz cech typowych dla twarzy pozytywnych i obraz cech typowych dla twarzy negatywnych.

Działanie programu

Po przeprowadzeniu szkolenia można uruchomić kompletny program. Jego fragment prezentujemy na **listingu 3**. W kodzie tym istotne jest użycie funkcji *model.predict()*, która pozwala sprawdzić, czy obraz pasuje do wytrenowanego wcześniej, pozytywnego wzorca. Zwraca ona nie tylko informację o tym, czy obraz pasuje bardziej do wzorca pozytywnego niż do wzorców negatywnych, ale też z jakim stopniem pewności dokonano porównania. Sejf otwiera się wtedy, gdy stopień pewności jest odpowiednio dobry, tzn. gdy zwracana wartość nie przekracza arbitralnie ustalonej wcześniej wartości granicznej.

Do zaryglowania i odryglowania zamka autor użył poleceń *box.lock()* i *box.unlock()*. Polecenia te odnoszą się bezpośrednio do komend przesunięcia serwomechanizmu do zadanych pozycji (funkcja *servo.set_servo()*).

Podsumowanie i ocena projektu

Zaprezentowany projekt jest bardzo ciekawy, ponieważ pokazuje, jak w prosty sposób zrealizować – wydawałoby się – niełatwe zadanie. Pewną trudność może stanowić konieczność kompilacji bibliotek OpenCV na Raspberry Pi, ale w praktyce, oprócz tego, że jest to czasochłonny proces, nie wymaga wielkich umiejętności i powinien się udać każdemu.

Naturalnie projekt jest niedopracowany, jeśli chodzi o część mechaniczną, gdyż tak zaprojektowane pudełko łatwo otworzyć czy zatrzasnąć, a sam mechanizm rozpoznawania twarzy jest daleki od idealności. Dobrze by było wprowadzić też możliwość rozpoznawania wielu różnych twarzy, tak by nie tylko jeden użytkownik mógł otworzyć sejf.

Marcin Karbowniczek, EP

Listing 3. Fragment programu głównego sejfu z kamerą i mechanizmem rozpoznawania twarzy, odpowiadający za rozpoznawanie twarzy i otwieranie pudełka

```
print ,Button pressed, looking for face...'
image = camera.read()
image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
result = face.detect_single(image)
if result is None:
    print ,Could not detect single face!'
    continue
x, y, w, h = result
crop = face.resize(face.crop(image, x, y, w, h))
# porównywanie zdjęcia twarzy z danymi zapisanymi w modelu
label, confidence = model.predict(crop)
print ,Predicted {0} face with confidence {1} (lower is more
confident).'.format(
    ,POSITIVE' if label == config.POSITIVE_LABEL else ,NEGATIVE',
    confidence)
if label == config.POSITIVE_LABEL and confidence < config.POSITIVE_THRESHOLD:
    print ,Recognized face!'
    box.unlock()
else:
    print ,Did not recognize face!'
```