

Raspberry Pi - konfigurowalny inteligentny monitoring

Wykonały: Szymon Janikowski Maryna Savchenko Celem naszego projektu było stworzenie oprogramowania, które będzie mogło zostać uruchomione na płytce Raspberry Pi w celach zapewnienia bezpieczeństwa, a dokładniej inteligentnego monitoringu z przechwytywaniem obrazu oraz możliwością konfiguracji jego działania, z dostępem do zapisywanych przez niego ujęć w Internecie.

Do zrealizowania tego zadania użyliśmy języka Python oraz poszczególnych popularnych bibliotek:

- → OpenCV do przetwarzania obrazu, zapisu i jego analizowania przez sieć neuronową (do działań na macierzy w sieci neuronowej niezbędna była też biblioteka NumPy)
- → Flask do stworzenia serwera, który na podstawie konfiguracji wybranej przez użytkownika uruchamia skrypt odpowiedzialny za przetwarzanie obrazu z kamery
- → Dropbox połączenie z kontem w tym serwisie i wysłanie tam zapisanego obrazu

oraz kilku mniejszych, służących do obsługi argumentów wywołania skryptu, przetwarzania pliku json z konfiguracją lub skalujących grafikę dla optymalizacji działania. Korzystaliśmy z modelu do detekcji obiektów MobileNet.

Działanie aplikacji:

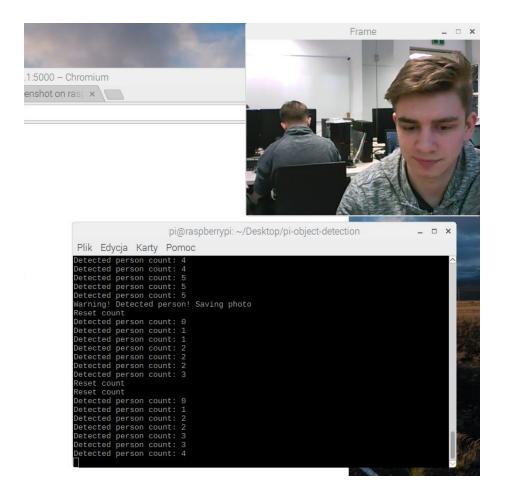
1. Konfiguracja



Kod serwera:

```
real_time_object_detection.py
                          server.py •
      from flask import Flask, request, render template
     app = Flask(__name__)
      ON = 0
      CLASSES = ["aeroplane", "bicycle", "bird", "boat",
         @app.route('/', methods=['GET', 'POST'])
     def index():
         global ON
          if request.method == 'POST':
             class_list = request.form.getlist("class")
             class_list = prepare_classes(class_list)
             if ON == 1:
                 os.system("cd /home/pi/Desktop/pi-object-detection")
                 os.system("pkill -f real_time_object_detection.py")
             os.system("cd /home/pi/Desktop/pi-object-detection")
             os.system("python3 real time object detection.py -cl " + class list)
         return render template('config.html', classes = CLASSES)
      def prepare classes(class list):
         arg = ""
         for c in class list:
             arg += c + " "
         return arg
     if __name__ == '__main ':
         app.run(debug=True, host='0.0.0.0')
```

2. Przetwarzanie obrazu - inkrementowanie licznika w następujących po sobie klatkach

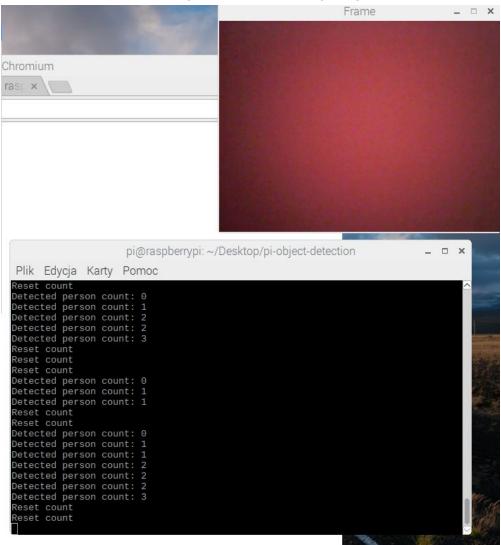


```
for i in np.arange(0, detections.shape[2]):
    # get confidence (probability) associated with the prediction
    confidence = detections[0, 0, i, 2]
    # filter out weak detections
    if confidence > args["confidence"]:
        # get index of the class label from the `detections`
        idx = int(detections[0, 0, i, 1])

for classes[idx] in WARNING:
    print("Detected " + CLASSES[idx] + " count: " + str(actual_count))
    if actual_count == detection_count:
        actual_count += 1
```

Rozróżnienie na zmienne detection_count i actual_count zapewnia, że w przypadku pojawienia się kilku interesujących nas obiektów, licznik następujących po sobie klatek zwiększy się tylko o 1 niezależnie od ilości tych obiektów.





```
if actual_count == detection_count:

detection_count = 0

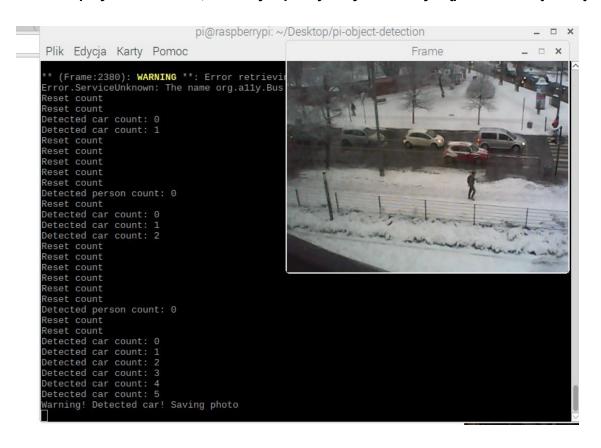
actual_count = 0

detection_count = actual_count

detection_count = actual_count
```

Jeśli licznik nie zmienił się po sprawdzeniu wystąpienia wszystkich klas - resetujemy go.

4. Zapisywanie obrazu, na którym podejrzany obiekt wystąpił określoną liczbę razy



```
if actual_count > conf["min_frames"]:

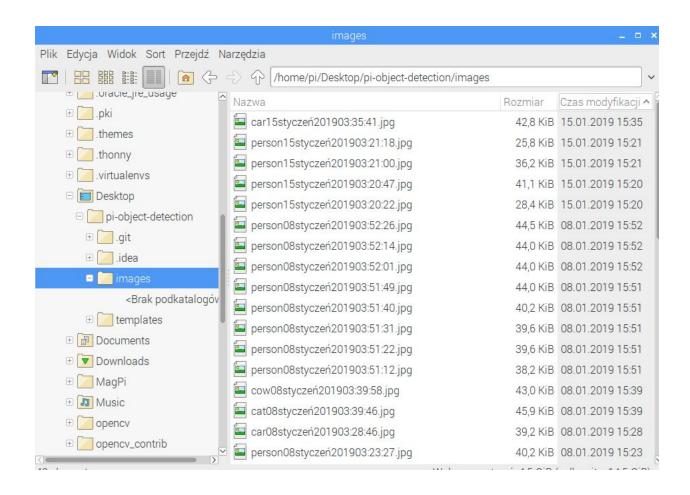
print("Warning! Detected " + CLASSES[idx] + "! Saving photo")

detection_count = 0

actual_count = 0

ts = timestamp.strftime(|"%d%B%Y%I:%M:%S%p")
```

5. Zapisane obrazy na pamięci Raspberry



```
ts = timestamp.strftime(["%d%B%Y%I:%M:%S%p"[]]

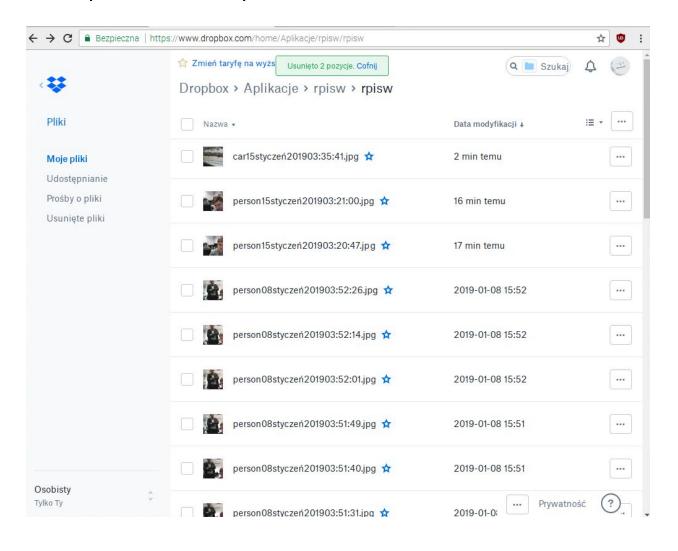
write_name = str(CLASSES[idx] + ts) + '.jpg'

abs_path = os.path.join(os.getcwd(), path, write_name)

cv2.imwrite(abs_path, frame)
```

Odpowiednio formatujemy czas wykonania zdjęcia i zapisujemy klatkę przetwarzaną przez openCV do określonego pliku.

6. Zapisanie obrazu na Dropboxie



```
conf = json.load(open("/home/pi/Desktop/pi-object-detection/conf.json"))
client = dropbox.Dropbox(conf["dropbox_token"])
```

Najpierw musimy połączyć się z danym kontem na dropboxie.

```
path_drop = "/{base_path}/{name}".format(base_path=conf["dropbox_path"], name=write_name)
client.files_upload(open(abs_path, "rb").read(), path_drop)
```

Następnie wysyłamy tam zapisany obraz.

Podsumowanie

Udało nam się spełnić postawione przez siebie wymagania. Podczas realizacji projektu napotkaliśmy na różne problemy:

- Instalacja openCV pomimo kilkukrotnych prób instalacji na własnej wersji systemu Raspbian, musieliśmy skorzystać z obrazu uzyskanego od innej grupy;
- Uruchamianie skryptu przetwarzającego obraz z poziomu serwera (skrypt wywołujący inny skrypt);
- Powtórne włączanie skryptu w przypadku kiedy już działa zamykamy działającą instancję i uruchamiamy go z nową konfiguracją;
- Problem z przekazywaniem argumentów przez serwer do uruchamianego skryptu wymagana była odpowiednia obróbka przekazywanych danych;
- Dostosowanie poziomu granicy kwalifikowania poszczególnych klas;