Raspberry Pi - konfigurowalny inteligentny monitoring

Wykonali:
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.

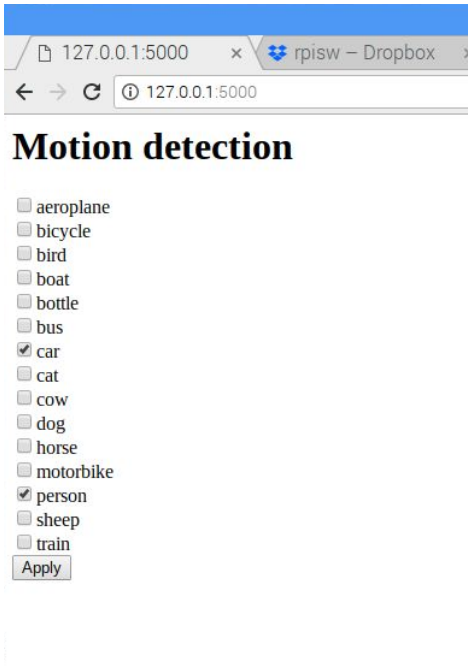
Do zrealizowania tego zadania użyliśmy języka Python oraz poszczególnych popularnych bibliotek:

- ➔ 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**

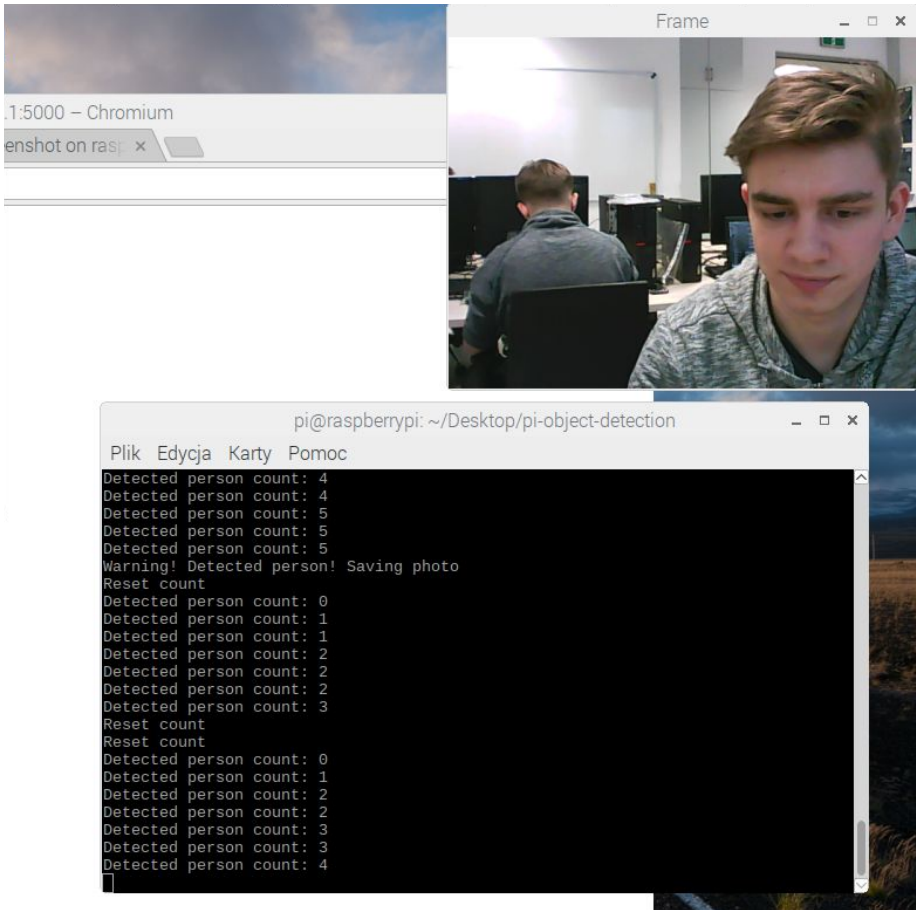


Kod serwera:

real_time_object_detection.py | server.py

```python
from flask import Flask, request, render_template
import os

app = Flask(__name__)
ON = 0
CLASSES = ["aeroplane", "bicycle", "bird", "boat",
    "bottle", "bus", "car", "cat", "cow", "dog",
        "horse", "motorbike", "person", "sheep", "train"]

@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)
        ON = 1
    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')

```

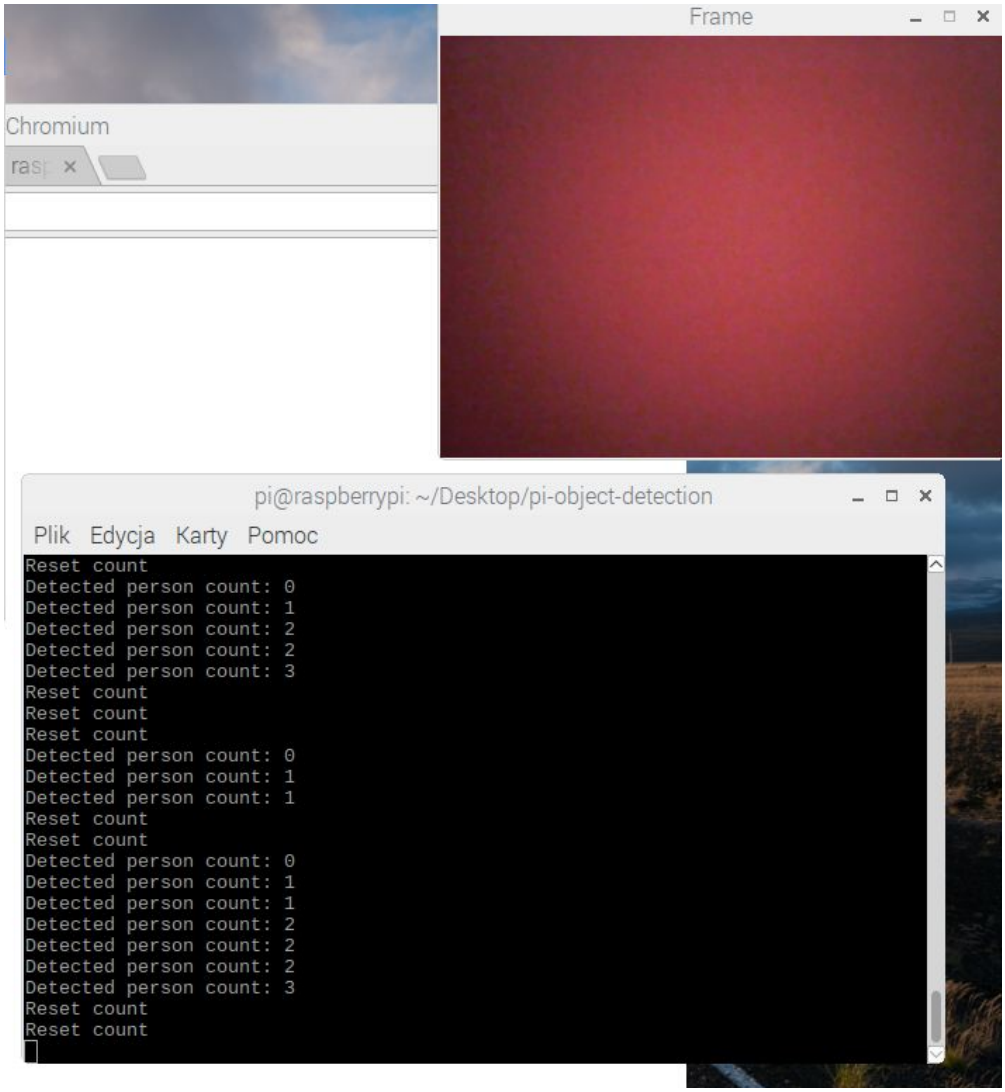
2. **Przetwarzanie obrazu - inkrementowanie licznika w następujących po sobie klatkach**




```
62      # loop over the detections
63      for i in np.arange(0, detections.shape[2]):
64          # get confidence (probability) associated with the prediction
65          confidence = detections[0, 0, i, 2]
66          # filter out weak detections
67          if confidence > args["confidence"]:
68              # get index of the class label from the `detections`
69              idx = int(detections[0, 0, i, 1])
70
71              if CLASSES[idx] in WARNING:
72                  print("Detected " + CLASSES[idx] + " count: " + str(actual_count))
73                  if actual_count == detection_count:
74                      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.

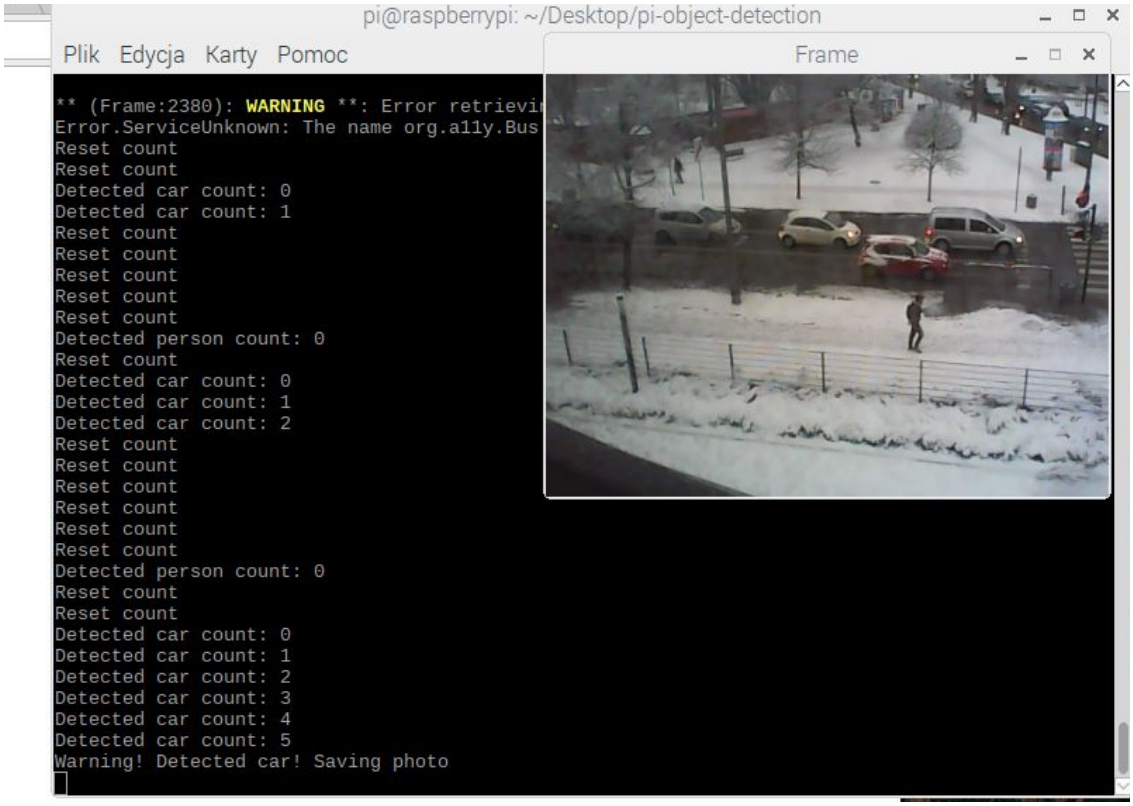
3. **Resetowanie licznika w przypadku braku podejrzanych obiektów**




```
88      if actual_count == detection_count:
89          detection_count = 0
90          actual_count = 0
91
92      detection_count = actual_count
93
```

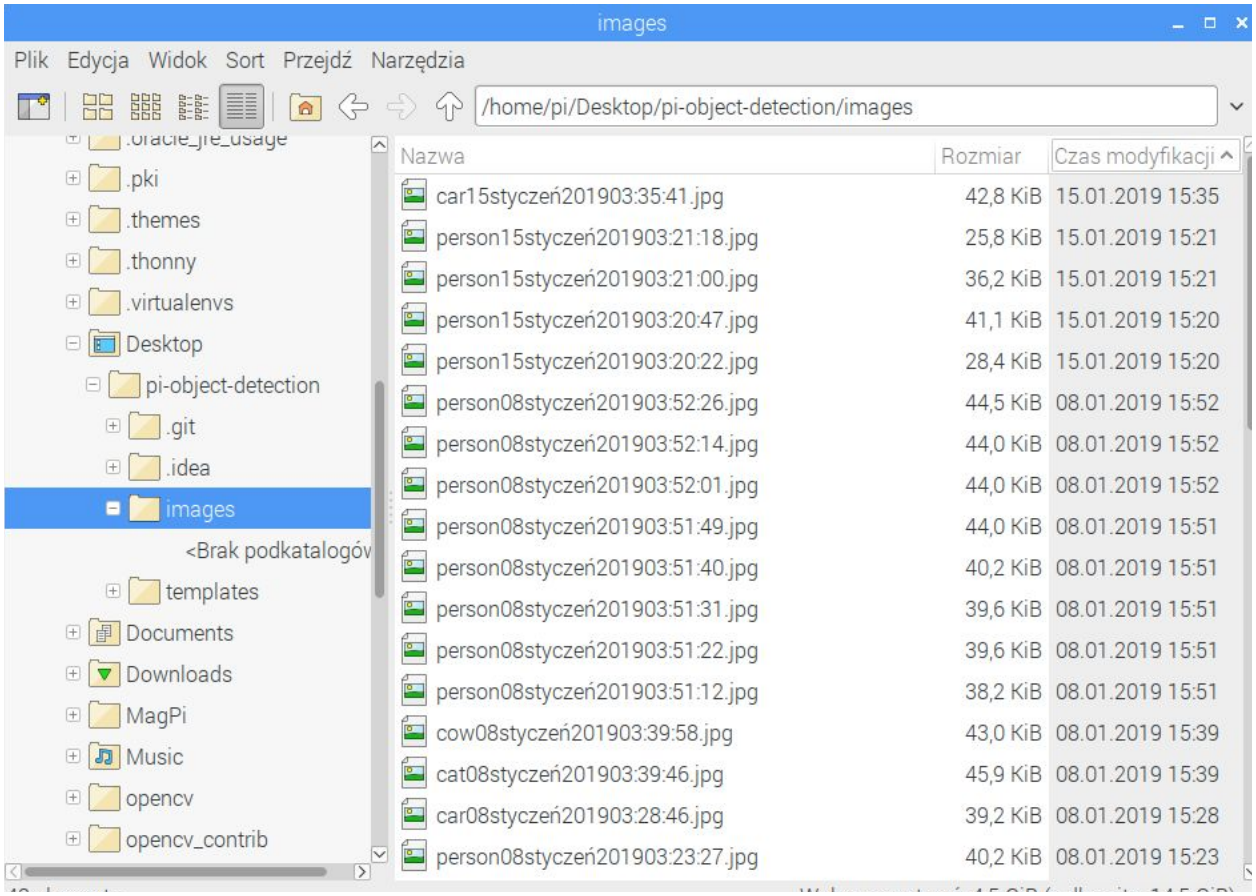
Jeśli licznik nie zmienił się po sprawdzeniu wystąpienia wszystkich klas - resetujemy go.

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




```
75
76                  if actual_count > conf["min_frames"]:
77                      print("Warning! Detected " + CLASSES[idx] + "! Saving photo")
78                      detection_count = 0
79                      actual_count = 0
80                      ts = timestamp.strftime("%d%B%Y%I:%M:%S%p")
```

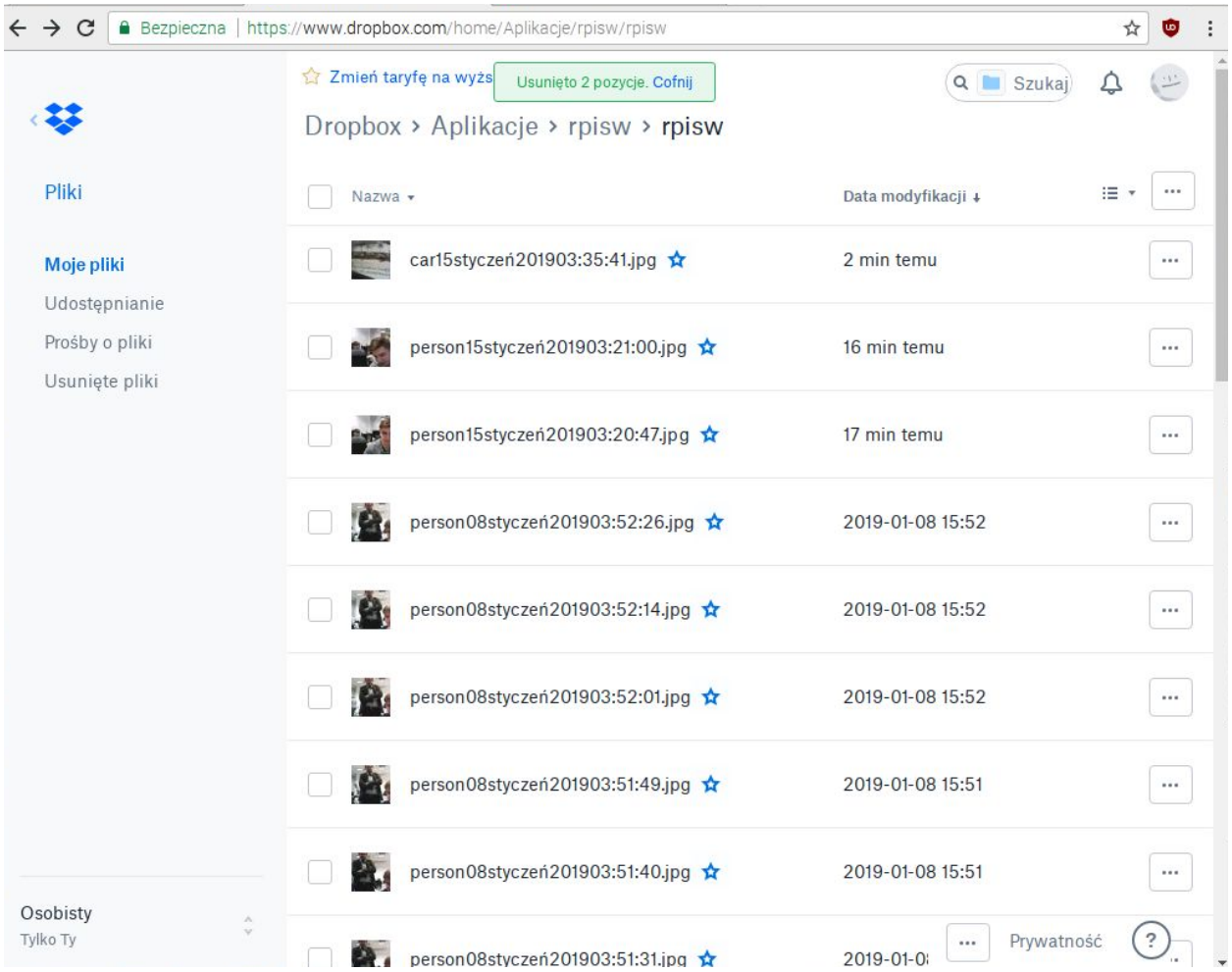
5. **Zapisane obrazy na pamięci Raspberry**




```
80      ts = timestamp.strftime("%d%B%Y%I:%M:%S%p")
81      write_name = str(CLASSES[idx] + ts) + '.jpg'
82      abs_path = os.path.join(os.getcwd(), path, write_name)
83      cv2.imwrite(abs_path, frame)
```

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

6. Zapisanie obrazu na Dropboxie




```
21  conf = json.load(open("/home/pi/Desktop/pi-object-detection/conf.json"))
22
23  client = dropbox.Dropbox(conf["dropbox_token"])
```

Najpierw musimy połączyć się z danym kontem na dropboxie.

```
84          path_drop = "/{base_path}/{name}".format(base_path=conf["dropbox_path"], name=write_name)
85          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;