# Rozpoznawanie i Przetwarzanie Obrazów
# Dokumentacja projektu

***Filip Przygoński, 248892***

***Wybrany obiekt: Samochód osobowy***

## Analiza obiektu

### Zdjęcia (licencja Creative Commons)



## Charakterystyka obiektu

- Samochód widoczny z przodu: rejestracja – biały prostokąt z czarnymi literami i cyframi, ciemny wlot powietrza, białe światła po bokach, przezroczysta przednia szyba.
- Samochód widoczny z tyłu: rejestracja – biały prostokąt z czarnymi literami i cyframi, czerwone tylnie światła, przezroczysta tylnia szyba.
- Samochód widoczny z boku: przezroczyste szyby, pomarańczowe kierunkowskazy, koła.

Istnieje prawdopodobieństwo, że analiza samej geometrii nie wystarczy do rozpoznawania samochodów, będzie potrzebna również analiza kolorów, m.in. tylnie światła (czerwień).

## Ryzyka i problemy

- Rejestracja – możliwość trafienia na niecodzienne rejestracje m.in.
    - stare czarne rejestracje z białymi symbolami,
    - rejestracje nowych samochodów z czerwone symbolami,
    - rejestracje innych państw np. tylna brytyjska rejestracja jest żółta.
- Kształt świateł
    - może być bardzo nieregularny między różnymi modelami (jak np. tylnie światła Hondy NSX i Fiata 500 na zdjęciach powyżej).
- Wlot powietrza z przodu
    - samochody elektryczne posiadają znacznie mniejsze wloty lub nie posiadają ich w ogóle.
- Malowanie, kolor samochodu
    - samochody np. dostawcze mogą być obklejone reklamami, sponsorami.
- Szyby
    - mogą być również przyciemniane bądź zasłonięte.

## Segmentacja obrazu

Segmentacja obrazu pomoże znacznie przyspieszyć przetwarzanie obrazów. Wykorzystując np. detekcję krawędzi, będziemy w stanie

podzielić obraz na kilka regionów/zbiorów, w których każdy piksel będzie podobny do innych z tego zbioru pod względem jakiejś właściwości.

### Zabezpieczenie się przed wielokrotną klasyfikacją tego samego obiektu

Aby zabezpieczyć się przed wielokrotną klasyfikacją tego samego obiektu, w pamięci programu powinny być przechowywane dane wykrytych w ostatniej klatce samochodów (np. położenie środka wykrytego obiektu, średni kolor wykrytego obiektu), i w następnej klatce sprawdzać różnicę. Jeśli jest ona zbyt mała, można z dużą dozą prawdopodobieństwa powiedzieć że są to te same obiekty i nie wykrywać go jeszcze raz.

# Dokumentacja programu

## Etap 2

### Wstęp

Proof of Concept aplikacji został napisany w Pythonie (wersja 3.8.2) wraz z biblioteką OpenCV (wersja 4.4.0.46). Aplikacja działa na komputerach osobistych, posiada interfejs konsolowy: wybór albo kamery podpiętej do komputera, albo odczyt pliku o podanej nazwie/ścieżce.

### Omówienie kodu

```
import cv2 as cv
```

Import biblioteki OpenCV.

```
def main():
    while True:
        user_input =
        input("Wczytać obraz z kamery (1) czy z pliku (2)? ")
        try:
            user_input = int(user_input)
        except Exception as e:
            print(e)
            continue
        if user_input == 1:
            camera_capture = cv.VideoCapture(0)
```

```
            # 0 = pierwsze urządzenie - kamera
            footage(camera_capture)
        elif user_input == 2:
            filename = input("Podaj nazwę pliku: ")
            file_capture = cv.VideoCapture(filename)
            footage(file_capture)
```

Główna metoda, odpowiadająca za konsolowy interfejs użytkownika. Gdy użytkownik wpisze 1, aplikacja zacznie przechwytywać obraz z kamery i będzie go wyświetlać na ekranie, gdy natomiast użytkownik wpisze 2, aplikacja otworzy plik z filmem, odtworzy go, a następnie wróci do interfejsu.

```
def footage(capture):
    # sprawdzenie czy capture jest dobrze 'otwarty'
    if not capture.isOpened():
        print("Capture is not opened")
        return

    title = "RiPO"
    cv.namedWindow(title)

    while capture.isOpened():
        # pobranie obrazu z capture
        reading, frame = capture.read()

        # wyświetlenie obrazu
        try:
            cv.imshow(title, frame)
        except Exception:
            break

        key = cv.waitKey(1)

        # wyjście z przechwytywania obrazu
        if key == 27:  # 27 == ESCAPE
            break

    capture.release()
    cv.destroyAllWindows()
```

Metoda pobierająca klatki z obiektu ‘capture’ (który może być równocześnie plikiem lub kamerą), i wyświetlająca je na ekranie. Bardzo łatwo będzie umieścić tutaj metodę analizującą każdą klatkę pobraną z capture. W przypadku gdy użytkownik wyjdzie z przechwytywania obrazu lub gdy skończy się film, aplikacja

poprawnie zamknie wszystkie okna, zwolni plik/kamerę i wróci do interfejsu konsolowego.

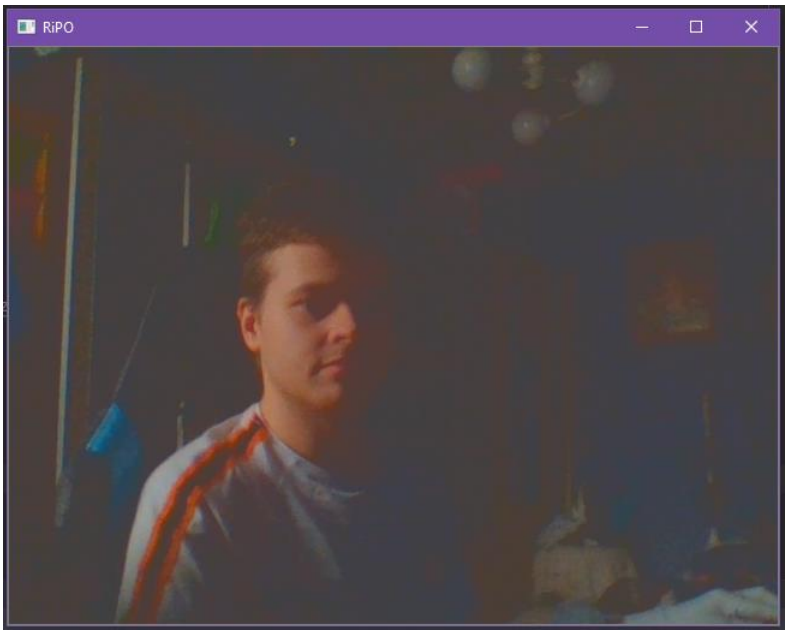
```
if __name__ == "__main__":
    main()
```

Wywołanie głównej metody podczas włączenia modułu.

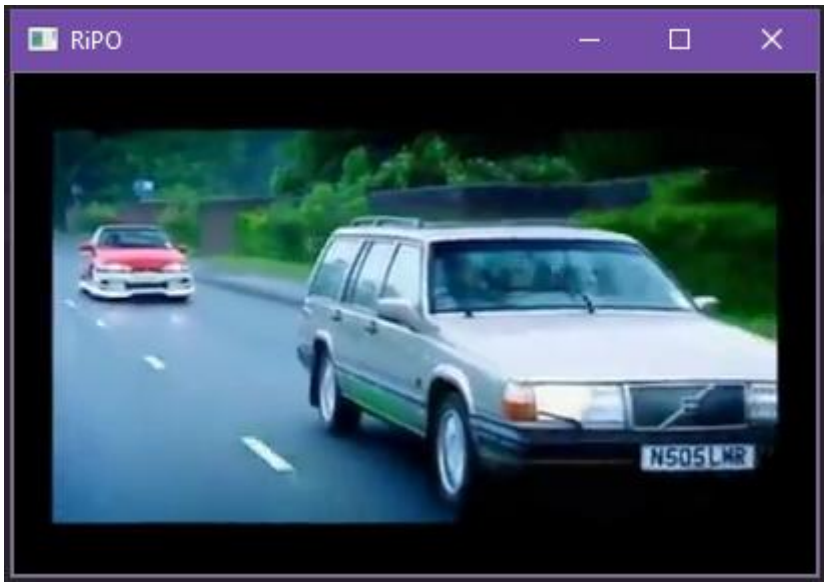
## Interfejs aplikacji

```
Wczytać obraz z kamery (1) czy z pliku (2)? 1
Wczytać obraz z kamery (1) czy z pliku (2)? 2
Podaj nazwę pliku: film2.mp4
```

Interfejs konsolowy, wybieranie metody wczytywania obrazu.




Okno aplikacji (obraz z kamery)



Okno aplikacji (film)

## Etap 3

### Rozpoznawanie samochodów na klatce obrazu

```
car_cascade_classifier = cv.CascadeClassifier('cars.xml')

# [...]

# pobranie obrazu z kamery
reading, frame = capture.read()

if frame is None:
    break

frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

cars = car_cascade_classifier.detectMultiScale(frame_gray)

for (x, y, w, h) in cars:
    cv.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)

# wyświetlenie obrazu
cv.imshow(title, frame)
```

Aplikacja została rozszerzona o wykrywanie samochodów za pomocą wytrenowanego klasyfikatora kaskadowego. Aby wykrywać samochody, najpierw należy przekonwertować klatkę obrazu na skalę szarości, a następnie przekazać ją klasyfikatorowi, a on na klatce obrazu zaznaczy prostokąty, w których wykryje samochód.

Wykryte przez klasyfikator samochody

# Testy

## Wstęp

Do testów wykorzystałem nagranie znalezione w Internecie, którego kopię zamieściłem w katalogu studenta. Posiada ono wiele różnych sytuacji pozwalających na sprawdzenie czy aplikacja poprawnie wykrywa samochody w wielu sytuacjach.

## Sprzęt

Testy zostały wykonane na laptopie Lenovo Legion Y520 o specyfikacji:

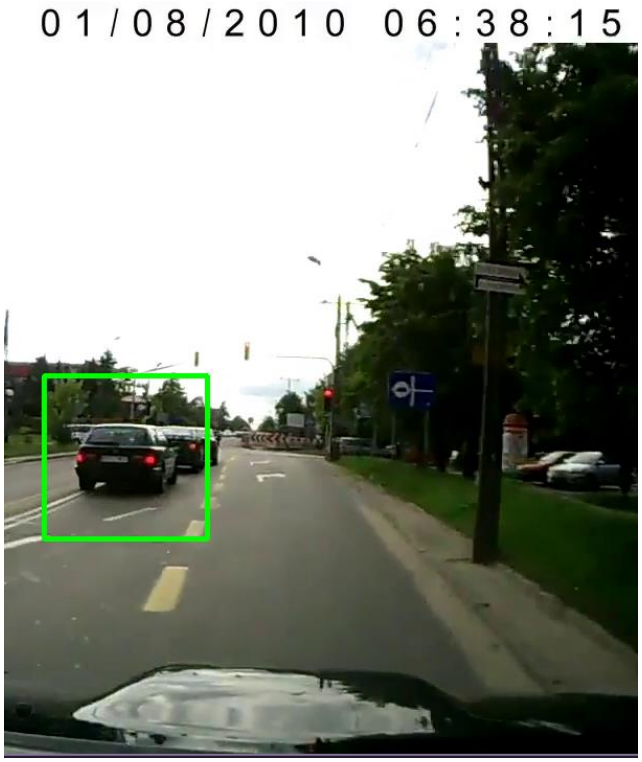
- **System operacyjny:** Windows 10 64-bit
- **Procesor:** Intel Core(TM) i7-7700HQ CPU @ 2.80GHz
- **Karta graficzna:** Intel HD Graphics 630 + NVIDIA GeForce GTX 1060 with Max-Q Design
- **Pamięć RAM:** 16GB
- **Dysk twardy:** Samsung SSD

## Założenia testów

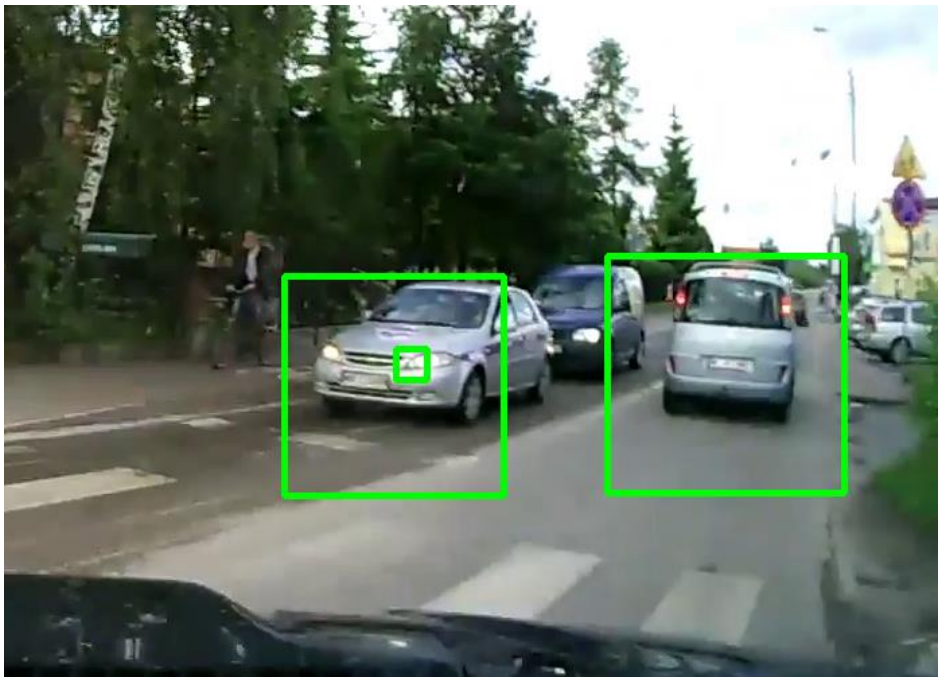
Testy mają na celu sprawdzenie, czy aplikacja poprawnie wykrywa samochody w różnych sytuacjach, takich jak:

- widok samochodu z przodu,
- widok samochodu z tyłu,
- widok samochodu z boku,
- wszystkie sytuacje wyżej, pod różnymi kątami etc.

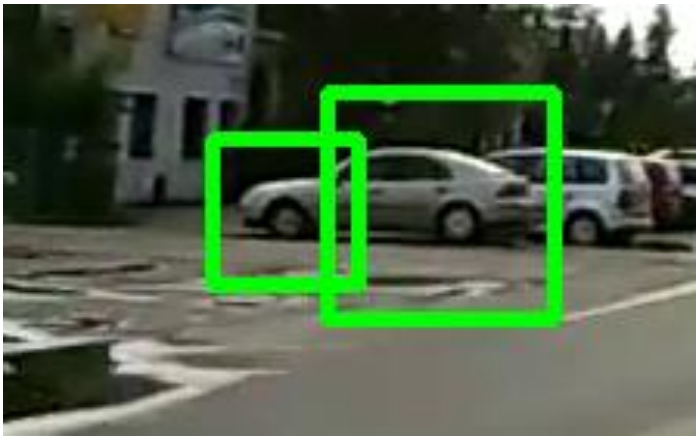

Wykrycie samochodów na podstawie ich przodu oraz tyłu




Wykrycie tyłu samochodu pod pewnym kątem



Wykrycie przodu i tyłu samochodu pod kątem



Wykrycie samochodu z boku

## Wnioski

Klasyfikator ogólnie działa dobrze, potrafi wykrywać samochody w wielu różnych sytuacjach, jednakże należałoby go jeszcze nieco dopracować, ponieważ zwłaszcza przy samochodach z boku, nie potrafi on idealnie ich rozpoznać.