Rozpoznawanie i przetwarzanie obrazów
Rozpoznawanie linii drogowych

Dokumentacja końcowa

Wydział Informatyki i Telekomunikacji	Informatyka Techniczna
Skład grupy	Grupa 26 Piotr Łach 256761 (Wtorek TP 12:15-15:15) Jakub Szpak 252782 (Wtorek TP 9:05-12:05)
Prowadzący	mgr Tomasz Serafin

1. Wstęp

Tematem wybranym przez grupę są linie drogowe. Program ma rozpoznawać linie oddzielające pasy ruchu oraz linie krawędziowe wyznaczającego krawędzie jezdni.

Ich identyfikacja to podstawowa umiejętność samochodu autonomicznego. Bez niej pojazd nie potrafiłby obrać odpowiedniego toru ruchu, co jest priorytetem tego typu rozwiązań. Idąc dalej, rozpoznawanie linii drogowych jest wykorzystywane przez marki samochodowe od wielu lat w oprogramowaniach do stabilizacji toru jazdy.

Duża potrzeba tego typu rozwiązań przekłada się na duże zapotrzebowanie takich detektorów, jednak szczęśliwie problem ten w jego podstawowej wersji, w idealnych warunkach jest uważany za trywialny, ze względu na duży kontrast białych/żółtych linii z czarnym asfaltem. Do wykrycia tych punktów wykorzystuje się detektory krawędzi (segmentacja obrazu - metoda krawędziowa).

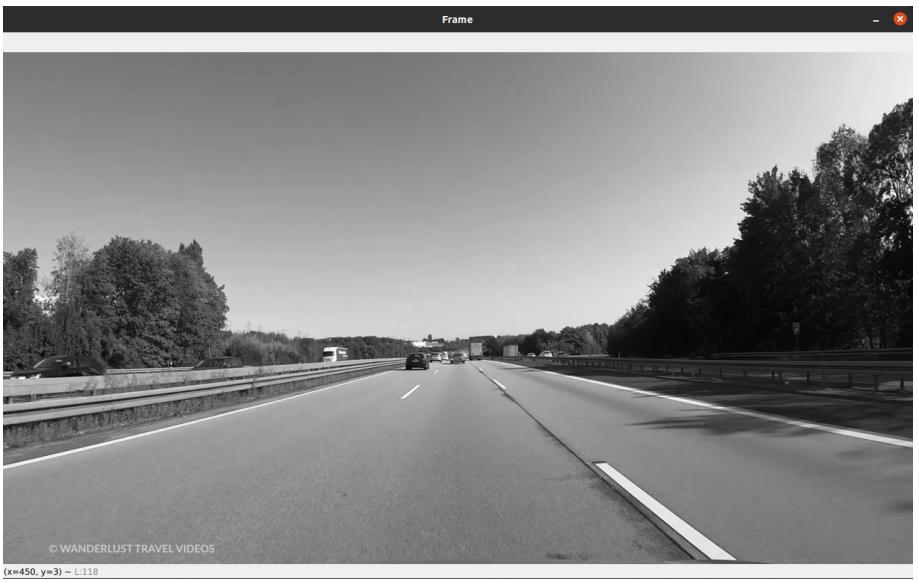
2. Opis oprogramowania

Aplikacja została napisany w Pythonie, wersji 3.8.10. Wykorzystujemy biblioteki opencv oraz numpy.

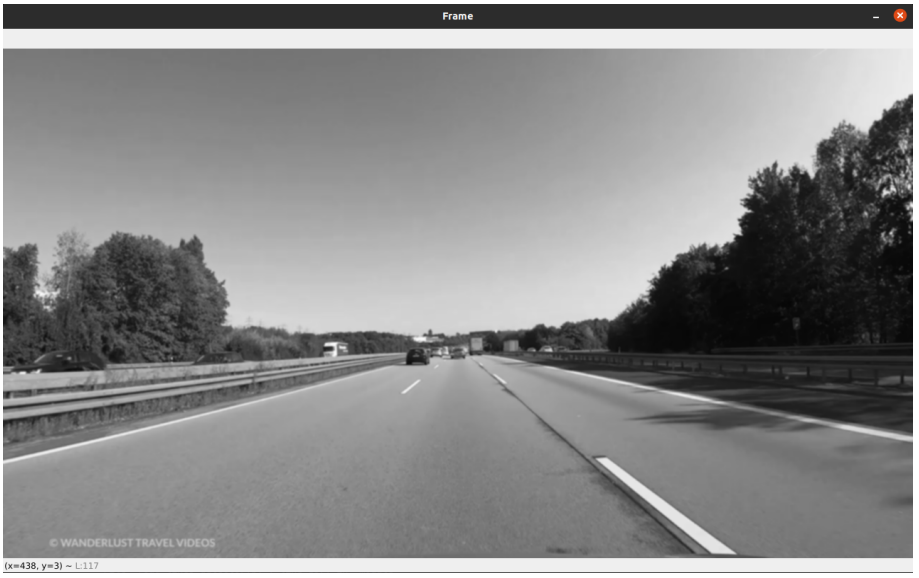
Przetwarza ona wideo klatka po klatce(danymi wejściowymi może być obraz z kamery lub wcześniej zarejestrowane wideo) dokonując przy tym kolejno następujących operacji:

Funkcja canny (rysunki 1-3).

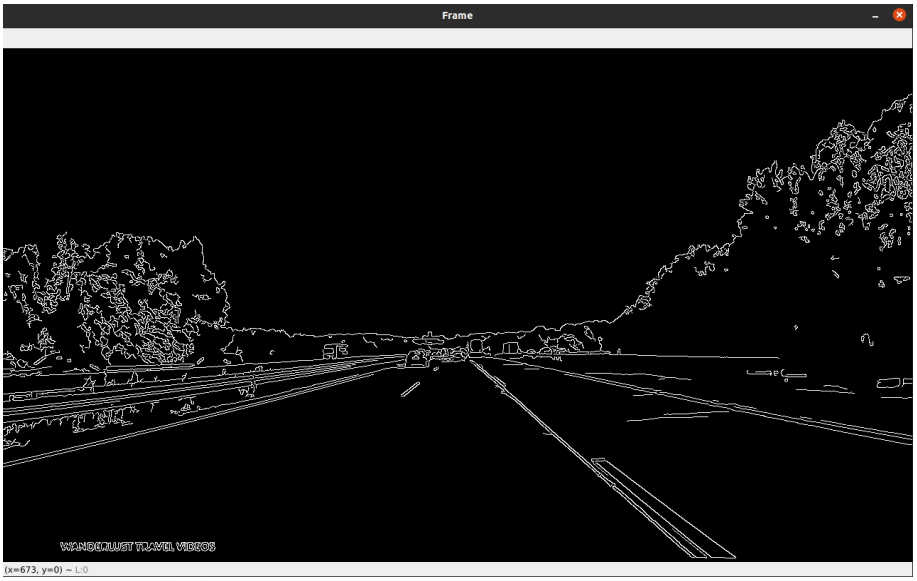
```
def canny(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)# Rysunek 1
    blur = cv2.GaussianBlur(gray, (5,5), 0)# Rysunek 2
    canny = cv2.Canny(blur, CANNY_LOW_THRESHOLD, CANNY_HIGH_THRESHOLD)# Rysunek 3
    return canny
```



Rysunek 1. Konwersja obrazu z RGB na odcienie szarości



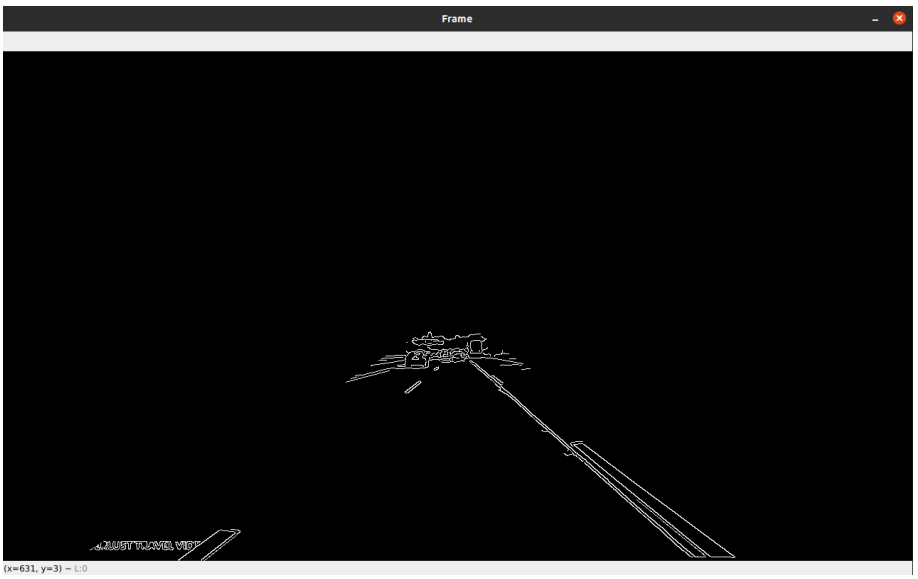
Rysunek 2. Rozmycie Gaussowskie



Rysunek 3. Wykorzystanie algorytmu Canny z biblioteki OpenCV

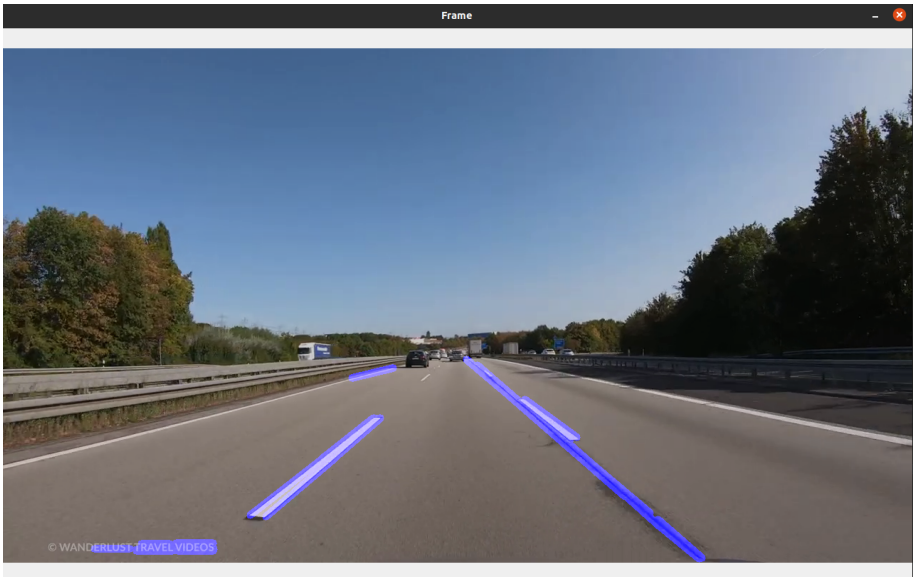
Funkcja region_of_interest przeznaczona jest do wyodrębnienia kluczowego fragmentu wideo. Wykorzystuje ona fakt, że z perspektywy przedniej szyby samochodu układają się one w kształt trójkąta zakończonego w centralnym punkcie na horyzoncie. Wyodrębnienie tego obszaru klatki jest jednym z istotniejszych etapów segmentacji (przykład segmentacji obszarowej). Jego rozmiar jest zależny od rozdzielczości kamery. Konfiguracje do plików multimedialnych, na których przeprowadzane były testy, znajdują się na początku pliku z kodem źródłowym.

```
def region_of_interest(image):
    height = image.shape[0]
    polygons = np.array([
        [(TRIANGLE_VERTEX_X_DOWN_LEFT, height),
        (TRIANGLE_VERTEX_X_DOWN_RIGHT, height),
        TRIANGLE_VERTEX_UPPER]
        ])
    mask = np.zeros_like(image)
    cv2.fillPoly(mask, polygons, 255)
    masked_image = cv2.bitwise_and(image, mask)
    return masked_image
```



Rysunek 4. Wyodrębnienie kluczowego fragmentu obrazu

Następny krok to zastosowanie probabilistycznego algorytmu transformacji Hough'a. Jest to metoda wykrywania regularnych kształtów w tzw. widzeniu komputerowym. Wykorzystujemy go w celu detekcji występujących przed kamerą linii drogowych (wykorzystuje się w tym miejscu fakt, znany z segmentacji obrazu - kształt szukanego obiektu).

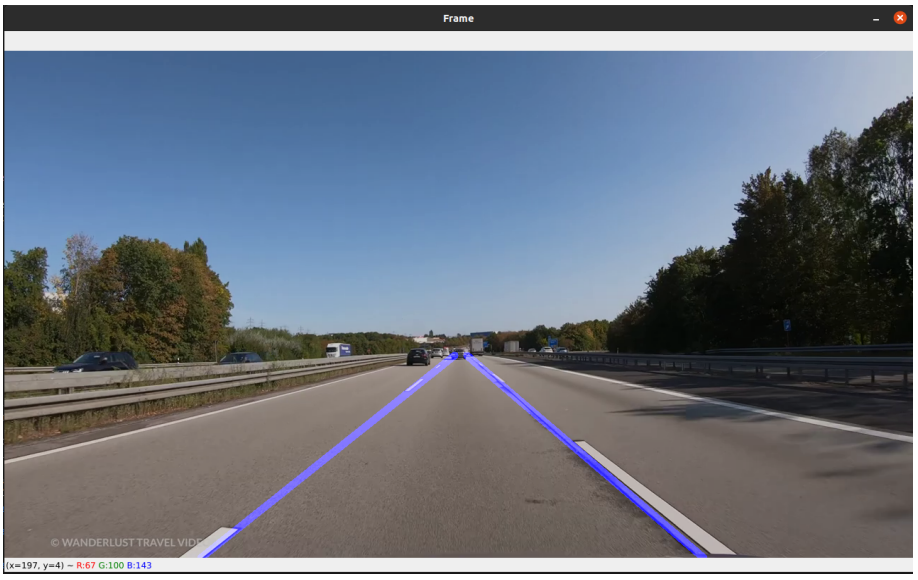


Rysunek 5. Probabilistyczny algorytm transformacji Hough'a do detekcji linii z OpenCV

Ostatnim etapem jest uśrednienie wszystkich znalezionych linii z wykorzystaniem funkcji average_slope_intercept w dwie wyznaczające aktualny pas ruchu. Jedna o ujemnym współczynniku nachylenia do osi X, a jedna dodatnim co przekłada się na prawą i lewą linie z perspektywy samochodu. Ze względu na charakter tej operacji niemożliwe jest podanie

prawdopodobieństwa klasyfikacji obiektu (ilość linii pasujących do szablonu nie przekłada się na prawdopodobieństwa poprawnego wykrycia).

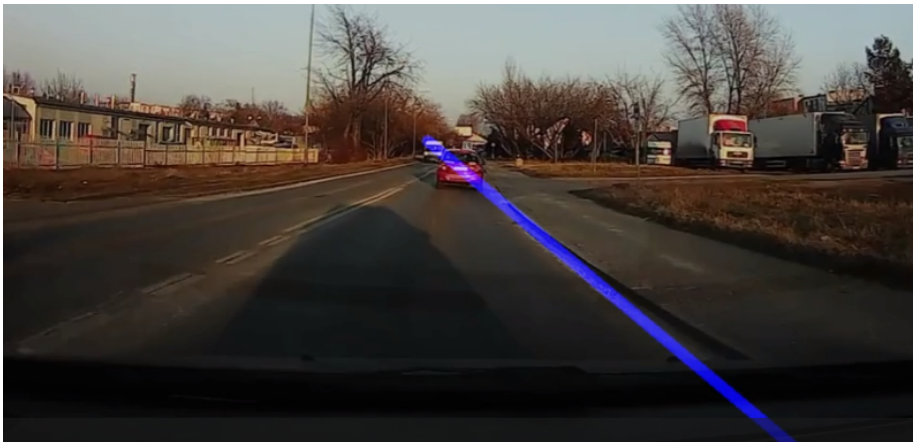
```
def average_slope_intercept(image, lines):
    left_fit = []
    right_fit = []
    for line in lines:
        x1, y1, x2, y2 = line.reshape(4)
        parameters = np.polyfit((x1, x2), (y1, y2), 1)
        slope = parameters[0]
        intercept = parameters[1]
        if slope < MAX_NEGATIVE_SLOPE:
            left_fit.append((slope, intercept))
        elif slope > MIN_POSITIVE_SLOPE:
            right_fit.append((slope, intercept))

    result = []

    if len(left_fit):
        left_fit_average = np.average(left_fit, axis=0)
        left_line = make_coordinates(image, left_fit_average)
        result.append(left_line)
    if len(right_fit):
        right_fit_average = np.average(right_fit, axis=0)
        right_line = make_coordinates(image, right_fit_average)
        result.append(right_line)
    return np.array(result)
```



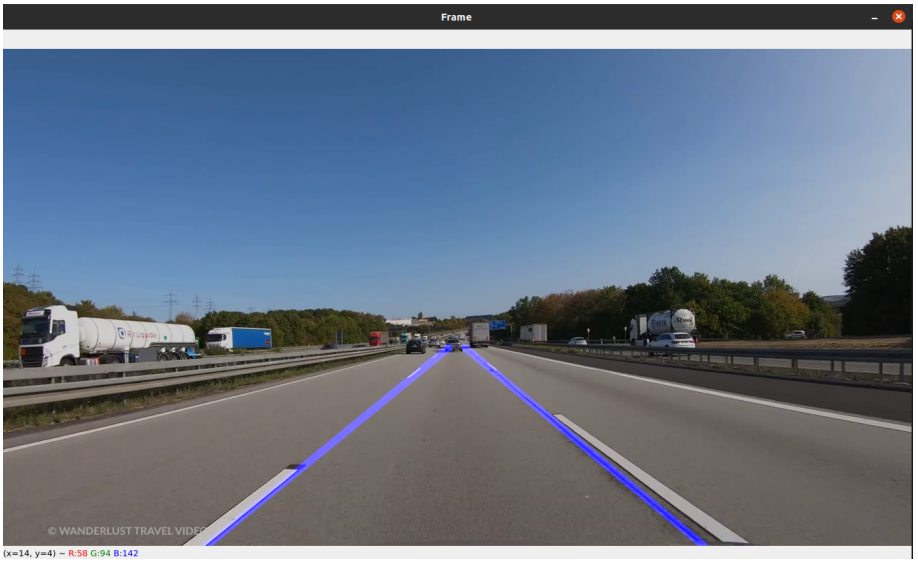
Rysunek 6. Uśrednienie rozpoznanych linii

3. Testy funkcjonalne, wyniki testów

Aplikacje testowaliśmy w kontekście warunków atmosferycznych, jakości wideo oraz jakości jezdni. Testy polegały na zgromadzeniu materiałów wideo oraz analizie zachowania programu w różnych przypadkach.

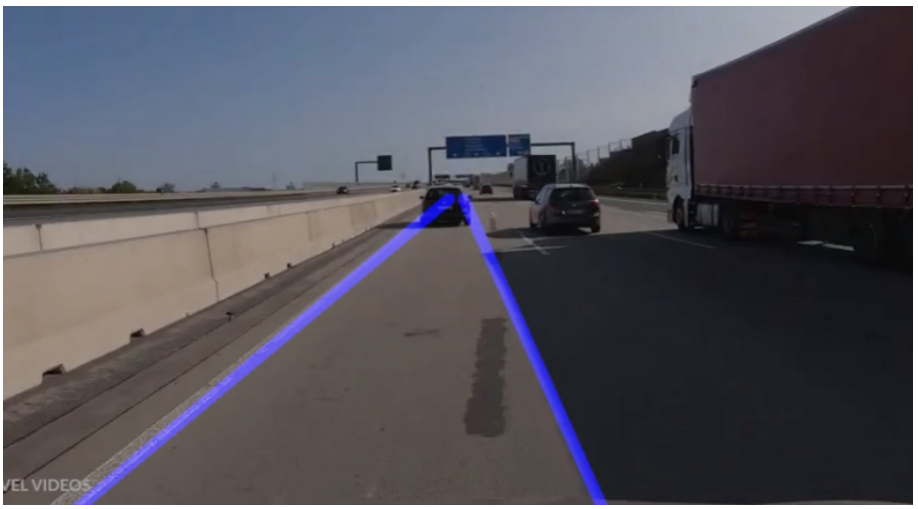


Rysunek 7. Słaba rozdzielczość rejestrowanego obrazu



Rysunek 8. Ryzyko błędnego rozpoznania obiektu w przypadku uszkodzonej nawierzchni

Uszkodzona nawierzchnia (spękania na jezdni) wpływa w sposób negatywny na rozpoznawanie linii drogowych. Jak widać na rysunku 8 program błędnie rozpoznał linie drogowe w uszkodzonej nawierzchni.



Rysunek 9. Błędna klasyfikacja cienia pojazdu jako linii drogowej

Wyniki:

Niestety aplikacja nie działa idealnie. Największe problemy zdają się sprawiać filmy o słabej jakości, wykrywanie linii drogowych w tym przypadku pozostawia wiele do życzenia. Mając jednak na uwadze, że w dzisiejszych czasach jakość wideo nie jest problemem, ponieważ rejestratory, a nawet smartfony dostarczają obraz o wysokiej rozdzielczości, największym przeciwnikiem zdają się być obiekty o podobnym kształcie, klasyfikowane przez program jako linie drogowe co prowadzi do błędów w uśrednianiu (np. cienie, dziury w drodze, inne linie poziome przecinające tor ruchu).

4. Uruchomienie aplikacji w systemie Linux

```
$ python3 -m venv venv/

$ . venv/bin/activate

$ pip install -r requirements.txt

$ python3 road_lines_detector.py
```

5. Wnioski

Efektem końcowym wykonanego projektu jest aplikacja wykrywająca linie drogowe. Wykonanie jej pozwoliło nam zapoznać się z biblioteką OpenCV, wykorzystywaną do przetwarzania obrazu.

Najbardziej podstawowym scenariuszem testowym jest obraz drogi szybkiego ruchu w dobrych warunkach atmosferycznych, z idealnie oznakowaną jezdnią znakami poziomymi (największy kontrast między liniami drogowymi, a asfaltem) nagrany kamerą o wysokiej rozdzielczości.

Z takimi przypadkami po odpowiednim skonfigurowaniu kluczowego trójkąta zakończonego w centralnym punkcie na horyzoncie, czyli obszaru, w którym występują linie drogowe program radzi sobie bez problemu.

Problemy zaczynają się jednak w momencie, w którym pojawia się jakaś niezgodność. Wtedy, w zależności od wpływu danego czynnika na obraz z kamery, zaczynamy zauważać błędy po uśrednianiu wykrytych punktów kontrastowych, co przekłada się na coraz większe rozbieżności od upragnionego rezultatu.

Podsumowując, na samym początku błędnie założyliśmy, że wykrywanie linii drogowych powinno być czymś stosunkowo prostym. Powtarzalność linii w różnych obiektach, uszkodzona nawierzchnia czy słaba jakość zarejestrowanego obrazu to problemy, których rozwiązanie okazało się na tyle nieoczywiste, że niemożliwa była ich zupełna eliminacja.