Kurs:
Rozpoznawanie i przetwarzanie obrazów - Projekt

Kamera samochodowa z funkcją wykrywania obiektów i panelem konfiguracyjnym

Autor:
Aleksandra Chrustek, 263900
Czw TP 7.30

Prowadzący:
mgr inż. Tomasz Serafin

31.03.2024

1 Syntetyczny opis tematu projektu i planowanego systemu

1.1 Opis projektowanego systemu

Celem projektu jest stworzenie systemu opartego na kamerze samochodowej, który zbiera obraz z otoczenia pojazdu oraz analizuje zebrane dane i wykrywa obiekty. System ten składa się z dwóch elementów: konfiguratora oraz oprogramowania wykonawczego. Konfigurator umożliwia użytkownikowi wybranie własnych ustawień dla oprogramowania wykonawczego poprzez interfejs graficzny: edytowanie rysowanych na ekranie linii oraz wybranie rozpoznawanych obiektów (człowiek, ściana, słupek, inny samochód). Dzięki konfiguratorowi będzie także możliwe wybranie sposobu informowania o wykrytej przeszkodzie: ramki zaznaczające obiekt, alert dźwiękowy, komunikat na ekranie. Kolejnym elementem jest oprogramowanie wykonawcze odpowiedzialne za przetwarzanie obrazu z kamery. Oprogramowanie ma za zadanie wyświetlać obraz, przechwycony za pomocą kamery, w okienku oraz dodać odpowiednie elementy wybrane przez użytkownika: linie, ramki, alerty. Taki system spełnia nie tylko funkcję podstawowej kamery samochodowej, ale także posiada funkcjonalność wykrywania obiektów oraz własnej konfiguracji co sprawi, że będzie użyteczny i pomocny dla szerokiego grona odbiorców.

1.2 Istniejące rozwiązania

Obecnie na rynku dostępne są rozwiązania takie jak:
-czujniki cofania (dźwiękowe),
-systemy oparte na jednej kamerze,
-systemy kamer 360 stopni,
-systemy wykrywające przeszkody oraz znaki podczas jazdy.
Dostępne na rynku możliwości różnią się w zależności od modelu i marki. Poniżej przedstawiono przykładowe wady i zalety dwóch wybranych rozwiązań.

1.2.1 Rozwiązanie zastosowane w Volvo

Wyposażenie auta w przednie i tylne czujniki ultradźwiękowe, kamerę cofania, system wykrywający znaki i przeszkody podczas jazdy. Zaletą takiego rozwiązania jest na pewno graficzne przedstawienie wszystkich odczytów użytkownikowi. Zdecydowaną wadą jest natomiast bardzo mało opcji personalizacji w konfiguratorze.

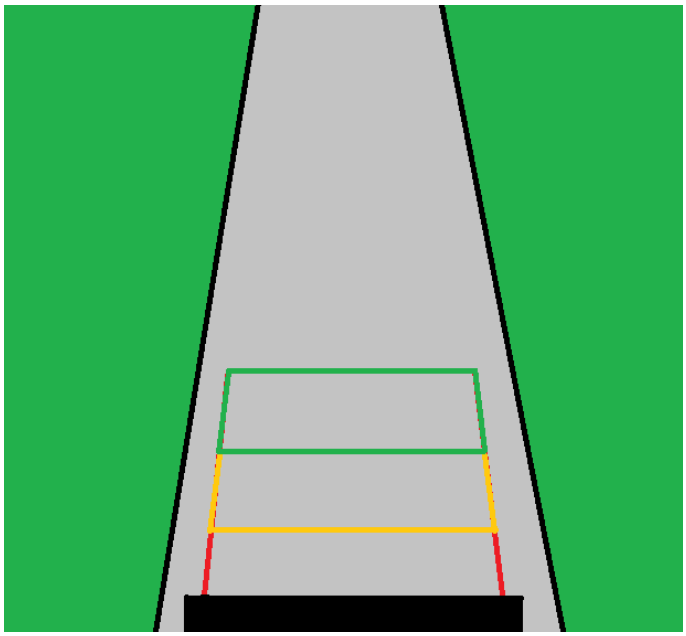
1.2.2 Rozwiązanie kamery BOSCAM K7

BOSCAM K7 to bezprzewodowy system cyfrowej kamery samochodowej znajdujący zastosowanie w cofaniu. Zaletą jest zdecydowanie możliwość wyboru różnych linii pomocniczych w zależności od pojazdu. Dużą wadą jest natomiast często opóźniony, mały i słabej jakości obraz.

2 Case studies

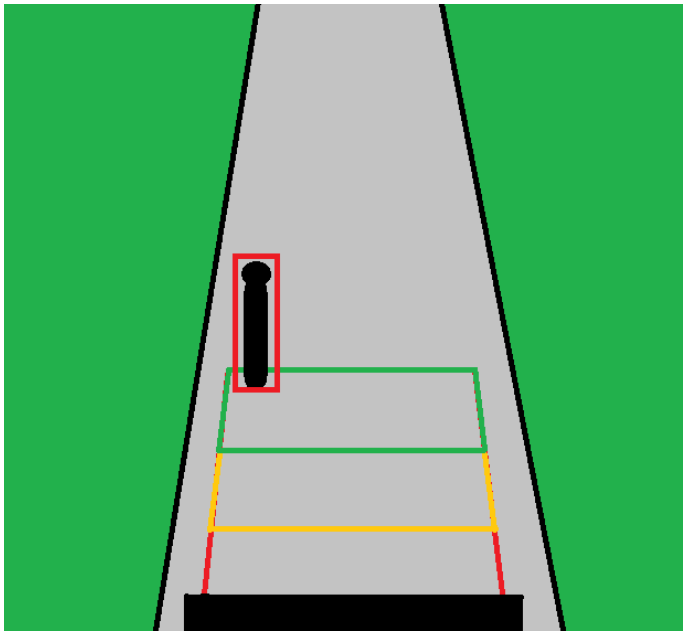
Projektowana kamera ma za zadanie pomóc kierowcy określić położenie samochodu względem obiektów za nim oraz wykryć ewentualne przeszkody. Na ekranie dodatkową pomocą ma być naniesienie na obraz linii pomocniczych w zależności od odległości: czerwona, żółta, zielona oraz zaznaczenie ramką wykrytych przeszkód. O wykrytych przeszkodach będzie informować alert dźwiękowy, alert na ekranie lub oba - w zależności od wybranej przez użytkownika konfiguracji. Poniżej przedstawiono przykładowe przypadki użycia kamery.

2.1 Brak przeszkód za pojazdem



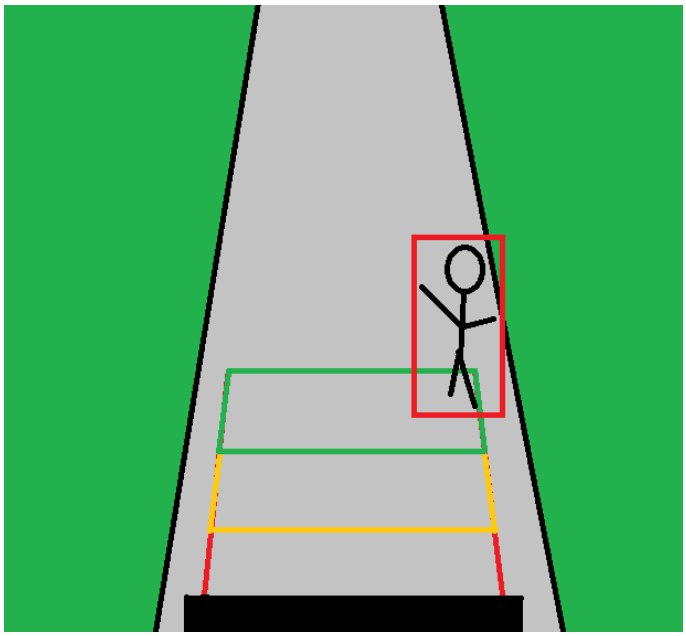
Rysunek 1: Schemat sytuacji

2.2 Wykryto słupek za pojazdem



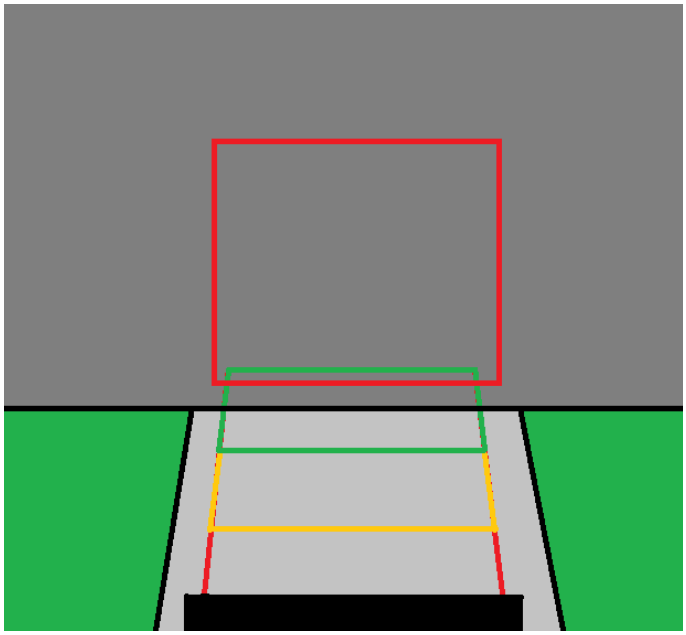
Rysunek 2: Schemat sytuacji

2.3 Wykryto pieszego za pojazdem



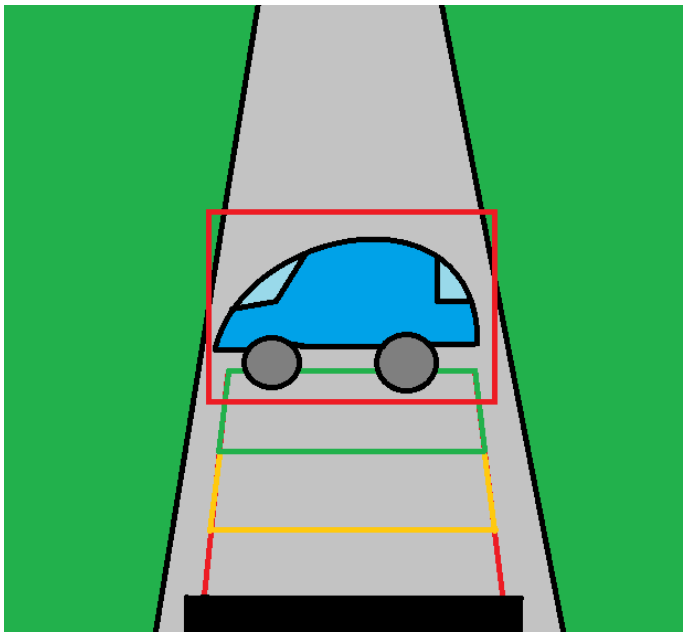
Rysunek 3: Schemat sytuacji

2.4 Wykryto ścianę za pojazdem



Rysunek 4: Schemat sytuacji

2.5 Wykryto samochód za pojazdem



Rysunek 5: Schemat sytuacji

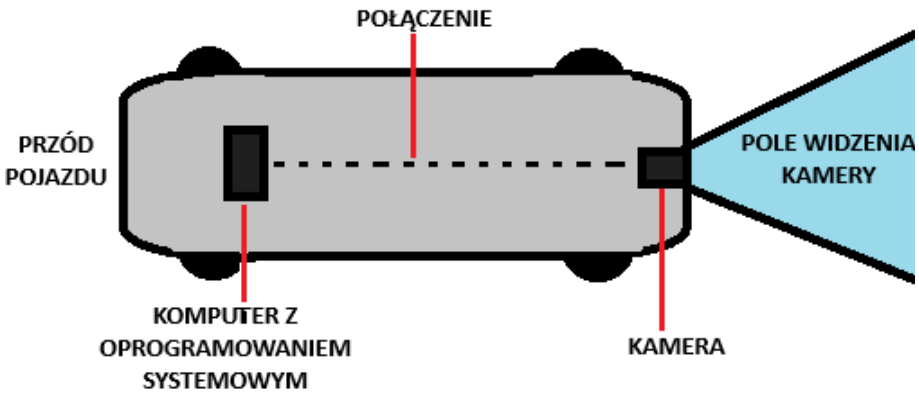
3 Opis funkcjonalny systemu

1. Wyświetlanie obrazu pobranego z kamery (nagranie obrazu wybrane przez użytkownika).
2. Rysowanie na ekranie linii pomocniczych w trzech kolorach: czerwony, żółty, zielony.
3. Rozpoznawanie przeszkody na obrazie: człowiek, słupek, ściana, inny samochód.
4. Rysowanie alertu na ekranie w postaci czerwonej ramki wokół wykrytej przeszkody.
5. Generowanie alertu dźwiękowego w przypadku wykrytej przeszkody.
6. Umożliwienie użytkownikowi personalizacji systemu dzięki konfiguratorowi: możliwość włączenia/wyłączenia linii pomocniczych, alertu wizualnego, alertu dźwiękowego.

4 Architektura wysokopoziomowa systemu

System składa się z:

1. Kamera: rejestracja obrazu za pojazdem, przesłanie zarejestrowanego obrazu do kontrolera.
2. Kontroler: odbiór i przetwarzanie obrazu z kamery, przekazanie informacji do ekranu oraz głośnika.
3. Ekran: wyświetlenie obrazu z kamery, rysowanie linii pomocnicznych i ramek wokół wykrytych przeszkód, umożliwienie personalizacji ustawień systemu.
4. Głośnik: generowanie alertu dźwiękowego.



Rysunek 6: Schemat architektury wysokopoziomowej systemu

5 Architektura logiczna systemu

5.1 Konfigurator

5.1.1 GUI

Graficzny interfejs użytkownika umożliwi użytkownikowi personalizację ustawień systemu.

5.1.2 Moduł konfiguracyjny

Zarządzać będzie ustawieniami systemu i pozwalać na zmianę parametrów takich jak wyświetlanie linii pomocniczych, alerty dźwiękowe i alerty wizualne.

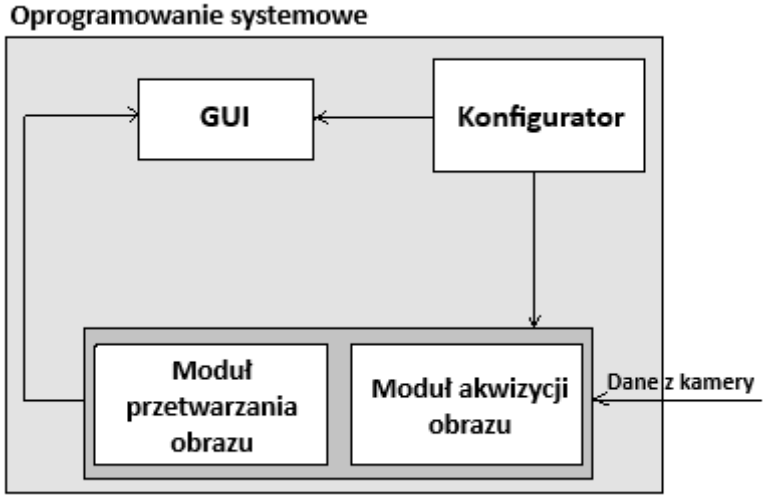
5.2 Oprogramowanie wykonawcze

5.2.1 Moduł akwizycji obrazu

Odpowiedzialny będzie za pobieranie obrazu z nagrania.

5.2.2 Moduł przetwarzania obrazu

Jego zadaniem będzie wykrywanie przeszkód na obrazie i w przypadku znalezienia przeszkody nałożenie na obraz odpowiedniego alertu.



Rysunek 7: Schemat architektury logicznej systemu

6 Dobór technologii

1. System operacyjny: Windows
2. Język programowania i biblioteki: Python, biblioteka OpenCV, NumPy.
3. Kamera: wejście USB w celu połączenia z komputerem, rozdzielczość co najmniej 720p.
4. Predykcje wydajnościowe: minimalne opoźnienie w celu płynnego działania systemu, aktualizowanie alertów i linii w czasie rzeczywistym, alerty generowane z jak najmniejszym opóźnieniem, aby umożliwić użytkownikowi reakcję.
5. Wymagania sprzętowe: procesor Intel Core i5 lub wyższy, pamięć RAM co najmniej 8 GB.
6. Kwestie sieciowe i bezpieczeństwa: zapewnienie poufności i integralności przesyłanych danych, komunikacja przewodowa bez połączenia z Internetem.
7. Konteneryzacja za pomocą technologii Docker.

7 Planowany zakres prac

Termin ukończenia	**Zakres prac**	**Planowany efekt**
02.05	Wstępna wersja oprogramowania systemowego z interfejsem graficznym.	Aplikacja pozwala na odebranie obrazu z nagrania wideo.
16.05	Oprogramowanie systemowe z funkcjami przetwarzania obrazu.	Aplikacja przetwarza obraz i rysuje linie pomocnicze.
30.05	Oprogramowanie systemowe z funkcjami przetwarzania i rozpoznawania obrazu.	Aplikacja przetwarza obraz i rozpoznaje człowieka, słupek i ścianę. Implementacja alertów.
13.06	Testy i wdrożenie systemu.	Aplikacja przeszła testy i jest gotowa w postaci obrazu kontenera Docker.

Tabela 1: Plan prac

8 Plan testów systemu

1. Testy sprawdzające poprawną akwizycję obrazu ze wskazanego przez użytkownika nagrania.
2. Testy poprawnego wykrywania obiektów i umieszczania alertu wizualnego.
3. Testy konfiguratora sprawdzające, czy system odpowiednio stosuje wybrane przez użytkownika ustawienia.

Testy zostaną przeprowadzone na nagraniach znalezionych w Internecie, głównie na YouTube, oraz na materiałach własnych. Testom zostanie poddana wystarczająco duża liczba filmów, aby efektywnie sprawdzić skuteczność wykrywania przeszkód takich jak człowiek, słupek, ściana, inny samochód z uwzględnieniem różnych warunków pogodowych oraz pór dnia.

1. W tym teście za cofającym pojazdem znajduje się inny samochód. Sprawdzana jest poprawność wykrywania innego samochodu za pojazdem oraz zgodność z rzeczywistością rysowanych linii pomocniczych. Video: auto.
2. W tym teście za cofającym pojazdem znajduje się ściana. Sprawdzana jest poprawność wykrywania ściany za pojazdem oraz zgodność z rzeczywistością rysowanych linii pomocniczych. Video: sciana.
3. W tym teście za cofającym pojazdem znajduje się słupek. Sprawdzana jest poprawność wykrywania słupka za pojazdem oraz zgodność z rzeczywistością rysowanych linii pomocniczych. Video: slupek2.
4. W tym teście za cofającym pojazdem znajduje się hydrant. W tym nagraniu padające słońce zmniejsza widoczność obiektu. Sprawdzana jest poprawność wykrywania słupka za pojazdem oraz zgodność z rzeczywistością rysowanych linii pomocniczych. Video: slupek3.
5. W tym teście za cofającym pojazdem znajdują się słupki. Sprawdzana jest poprawność wykrywania słupków za pojazdem oraz zgodność z rzeczywistością rysowanych linii pomocniczych. Video: slupek.
6. W tym teście za cofającym pojazdem przechodzi pieszy. Sprawdzana jest poprawność wykrywania pieszego za pojazdem. Video: pieszy1.
7. W tym teście za cofającym pojazdem przechodzi pieszy. Sprawdzana jest poprawność wykrywania pieszego za pojazdem. Video: pieszy2.
8. W tym teście sprawdzamy poprawność wykrywania obiektu samochód przez model. Na nagraniu widoczne są pojazdy poruszające się drogą szybkiego ruchu. Video: auta.

9. W tym teście sprawdzamy poprawność wykrywania obiektu pieszy przez model. Na nagraniu widoczni są piesi poruszający się na przejściu dla pieszych. Video: piesi.

10. W tym teście sprawdzamy poprawność wykrywania obiektu samochód przez model. Na nagraniu widoczne są pojazdy uczestniczące w kolizji na drodze. Video: auta4.

11. W tym teście sprawdzamy poprawność wykrywania obiektu samochód przez model. Na nagraniu widoczne są pojazdy poruszające się drogą. Nagranie powstało w warunkach nocnych, co dodatkowo sprawdza poprawność modelu w różnych porach dnia. Video: autanoc.

12. W tym teście za cofającym pojazdem znajduje się barierka. Sprawdzana jest poprawność wykrywania ściany za pojazdem oraz zgodność z rzeczywistością rysowanych linii pomocniczych. Video: cofanie1.

13. W tym teście za cofającym pojazdem znajduje się inny samochód. Sprawdzana jest poprawność wykrywania innego samochodu za pojazdem oraz zgodność z rzeczywistością rysowanych linii pomocniczych. Video: cofanie3.

9 Opis zrealizowanych prac w projekcie

Projekt polegał na stworzeniu aplikacji do detekcji obiektów w plikach wideo, wykorzystując biblioteki takie jak OpenCV do przetwarzania obrazu oraz biblioteki tkinter do stworzenia prostego interfejsu użytkownika.
Prace nad projektem przebiegały według następującego planu:

1. Analiza wymagań: Pierwszym krokiem było dokładne zdefiniowanie wymagań projektowych. Określenie funkcjonalności, które aplikacja ma posiadać, takich jak możliwość wyboru pliku wideo, detekcja obiektów, rysowanie linii odległości, odtwarzanie dźwięku w przypadku wykrycia obiektu oraz prosty interfejs użytkownika.

2. Wybór narzędzi i bibliotek: Następnie dokonano wyboru narzędzi i bibliotek niezbędnych do realizacji projektu. Zdecydowano się na użycie biblioteki OpenCV do przetwarzania obrazu, tkinter do tworzenia interfejsu użytkownika oraz biblioteki simpleaudio do odtwarzania dźwięku.

3. Implementacja podstawowych funkcji: Rozpoczęto od implementacji podstawowych funkcji, takich jak wczytywanie pliku wideo, skalowanie obrazu, detekcja obiektów oraz rysowanie linii odległości na obrazie. Te funkcje były kluczowe dla działania całej aplikacji.

4. Testowanie i debugowanie: Po zaimplementowaniu podstawowych funkcji przystąpiono do testowania i debugowania kodu. Skupiono się na identyfikacji i naprawie wszelkich błędów oraz nieprawidłowego zachowania aplikacji.

5. Dodawanie dodatkowych funkcji: Po upewnieniu się, że podstawowe funkcje działają poprawnie, dodano dodatkowe funkcje, takie jak odtwarzanie dźwięku w przypadku wykrycia obiektu oraz dodatkowe opcje interakcji z użytkownikiem, takie jak możliwość wyboru detekcji obiektów i rysowania linii odległości.

6. Optymalizacja i poprawki: Ostatnim etapem było dokładne testowanie, optymalizacja kodu oraz wprowadzenie wszelkich poprawek i ulepszeń, które były konieczne do uzyskania działającej i wydajnej aplikacji.

Podczas prac nad projektem napotkano kilka problemów i wyzwań:

1. Integracja bibliotek: Integracja biblioteki OpenCV z interfejsem tkinter była wyzwaniem. Konieczne było zapewnienie kompatybilności między strukturami danych używanymi przez OpenCV a formatami danych obsługiwanymi przez tkinter.

2. Dostosowanie do różnych rozmiarów obrazu: Konieczne było dostosowanie aplikacji do obsługi plików wideo o różnych rozmiarach. W związku z tym, należało

uwzględnić skalowanie obrazu w celu zachowania proporcji i jakości wyświetlanego obrazu.

3. Wybór i wczytywanie pliku wideo: Implementacja funkcji umożliwiającej wybór i wczytanie pliku wideo z systemu plików użytkownika wymagała odpowiedniej obsługi błędów oraz zabezpieczenia przed próbą otwarcia nieistniejącego pliku lub pliku o nieobsługiwanym formacie.

4. Detekcja obiektów: Implementacja algorytmu detekcji obiektów za pomocą klasyfikatorów była jednym z głównych wyzwań. Wymagała uwzględnienia różnych klas obiektów do detekcji.

5. Zarządzanie wątkami: Aby zapewnić płynność działania aplikacji podczas detekcji obiektów, konieczne było prawidłowe zarządzanie wątkami, zwłaszcza podczas odtwarzania sygnału dźwiękowego w przypadku wykrycia obiektu.

6. Interfejs użytkownika: Stworzenie prostego i intuicyjnego interfejsu użytkownika za pomocą biblioteki tkinter wymagało przemyślanego projektowania układu okna oraz obsługi zdarzeń związanych z interakcją użytkownika.

Ostatecznie, po przezwyciężeniu tych wyzwań, udało się zakończyć projekt, tworząc działającą aplikację do detekcji obiektów w plikach wideo z prostym interfejsem użytkownika.

10 Architektura niskopoziomowa wraz z fragmentami kodu programu

Program został podzielony na kilka kluczowych funkcji:

10.1 Detekcja obiektów

Kod wykorzystuje bibliotekę OpenCV Cascade Classifier do detekcji obiektów w pliku wideo. Możliwe jest wybór detekcji pieszych lub samochodów poprzez interfejs użytkownika.

```
def detect_objects(root, choices, draw_line, play_sound):
    classifiers = []
    for choice in choices:
        if choice == 'Piesi':
            classifiers.append(cv2.CascadeClassifier('pedestrian.xml'))
        elif choice == 'Samochody':
            classifiers.append(cv2.CascadeClassifier('cars.xml'))
        else:
```

```
            print("Nieprawidłowy wybór:", choice)

    file_path = filedialog.askopenfilename()
    cap = cv2.VideoCapture(file_path)

    if not cap.isOpened():
        print("Nie można otworzyć pliku wideo.")
        return

...

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        red_x_left_top, red_y_top, red_x_right_bottom, red_y_bottom =
            draw_distance_line(frame, original_dims, draw_line)

        alarm_played = False
        for classifier in classifiers:
            objects = classifier.detectMultiScale(gray, 1.1, 4)

...
```

10.2 Rysowanie linii odległości

Program rysuje na obrazie linie odległości, które są wykorzystywane do monitorowania przestrzeni na drodze. Linie te są rysowane w kolorach czerwonym, żółtym i zielonym, aby wizualnie reprezentować różne strefy odległości.

```
def draw_distance_line(frame, original_dims, draw_line):
    # Obliczenie współrzędnych linii na podstawie rozmiaru ramki wideo
    original_height, original_width = original_dims
    height, width, _ = frame.shape

    scale_x = width / original_width
    scale_y = height / original_height

    yellow_trap_width = 380 * scale_x
    green_trap_width = 220 * scale_x

    # Współrzędne środka samochodu
    car_center = width // 2
```

```
red_y_top = int(height * 0.8)
red_y_bottom = height
red_x_left_top = int((original_width // 2 - 450) * scale_x)
red_x_right_top = int((original_width // 2 + 450) * scale_x)
red_x_left_bottom = int((original_width // 2 - 600) * scale_x)
red_x_right_bottom = int((original_width // 2 + 600) * scale_x)

yellow_y_top = int(height * 0.6)
yellow_y_bottom = int(height * 0.8)
yellow_x_left_top = car_center - int(yellow_trap_width) + int(80 * scale_x)
yellow_x_right_top = car_center + int(yellow_trap_width) - int(80 * scale_x)
yellow_x_left_bottom = red_x_left_top
yellow_x_right_bottom = red_x_right_top

green_y_top = int(height * 0.5)
green_y_bottom = int(height * 0.6)
green_x_left_top = car_center - int(green_trap_width)
green_x_right_top = car_center + int(green_trap_width)
green_x_left_bottom = yellow_x_left_top
green_x_right_bottom = yellow_x_right_top

if draw_line:
    cv2.line(frame, (red_x_left_top, red_y_top),
        (red_x_left_bottom, red_y_bottom), (0, 0, 255), 2)
    cv2.line(frame, (red_x_left_top, red_y_top),
        (red_x_right_top, red_y_top), (0, 0, 255), 2)
    cv2.line(frame, (red_x_right_top, red_y_top),
        (red_x_right_bottom, red_y_bottom), (0, 0, 255), 2)
    cv2.line(frame, (red_x_right_bottom, red_y_bottom),
        (red_x_left_bottom, red_y_bottom), (0, 0, 255), 2)
    cv2.line(frame, (yellow_x_left_top, yellow_y_top),
        (yellow_x_left_bottom, yellow_y_bottom), (0, 255, 255), 2)
    cv2.line(frame, (yellow_x_left_top, yellow_y_top),
        (yellow_x_right_top, yellow_y_top), (0, 255, 255), 2)
    cv2.line(frame, (yellow_x_right_top, yellow_y_top),
        (yellow_x_right_bottom, yellow_y_bottom), (0, 255, 255), 2)
    cv2.line(frame, (yellow_x_right_bottom, yellow_y_bottom),
        (yellow_x_left_bottom, yellow_y_bottom), (0, 255, 255), 2)
```

```
        cv2.line(frame, (green_x_left_top, green_y_top),
            (green_x_left_bottom, green_y_bottom), (0, 255, 0), 2)
        cv2.line(frame, (green_x_left_top, green_y_top),
            (green_x_right_top, green_y_top), (0, 255, 0), 2)
        cv2.line(frame, (green_x_right_top, green_y_top),
            (green_x_right_bottom, green_y_bottom), (0, 255, 0), 2)
        cv2.line(frame, (green_x_right_bottom, green_y_bottom),
            (green_x_left_bottom, green_y_bottom), (0, 255, 0), 2)

    return (red_x_left_top, red_y_top, red_x_right_bottom, red_y_bottom)
```

10.3 Odtwarzanie sygnału dźwiękowego

W przypadku wykrycia obiektu w strefie niebezpieczeństwa, aplikacja odtwarza sygnał dźwiękowy przy pomocy biblioteki simpleaudio, informując użytkownika o potencjalnym zagrożeniu.

```
def play_alarm():
        wave_obj = sa.WaveObject.from_wave_file('alarm.wav')
        play_obj = wave_obj.play()
        play_obj.wait_done()

        alarm_played = False
        for classifier in classifiers:
            objects = classifier.detectMultiScale(gray, 1.1, 4)
            for (x, y, w, h) in objects:
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
                if (red_x_left_top <= x <= red_x_right_bottom
                    or red_x_left_top <= x + w <= red_x_right_bottom)
                    and (red_y_top <= y <= red_y_bottom
                    or red_y_top <= y + h <= red_y_bottom):
                    if not alarm_played and play_sound:
                        threading.Thread(target=play_alarm).start()
                        alarm_played = True
```

10.4 Interfejs użytkownika

Program posiada prosty interfejs użytkownika zbudowany przy użyciu biblioteki tkinter. Pozwala użytkownikowi wybrać typ detekcji (piesi lub samochody), kontrolować rysowanie linii odległości oraz decydować o odtwarzaniu sygnału dźwiękowego.

```
def main():
```

```
root = tk.Tk()
root.title("Wybór detekcji obiektów")

# Ukrycie okna do momentu gotowości
root.withdraw()
root.update_idletasks()

# Ustawienie położenia okna na środku ekranu
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
root.geometry('+{}+{}'.format((screen_width - root.winfo_reqwidth()) // 2,
    (screen_height - root.winfo_reqheight()) // 2))
root.deiconify()

# Funkcja obsługująca kliknięcie przycisku "Start"
def on_button_click():
    selected_choices = [choice.get() for choice in checkboxes if choice.get()]
    draw_line = draw_line_var.get()
    play_sound = play_sound_var.get()
    detect_objects(root, selected_choices, draw_line, play_sound)

# Opcje wyboru detekcji obiektów
choices = ['Piesi', 'Samochody']
checkboxes = []
for choice in choices:
    var = tk.StringVar()
    checkbox = tk.Checkbutton(root, text=choice, variable=var,
        onvalue=choice, offvalue="")
    checkbox.pack()
    checkboxes.append(var)

# Przycisk uruchamiający detekcję obiektów
start_button = tk.Button(root, text="Start", command=on_button_click)
start_button.pack()

# Opcja rysowania linii na wideo
draw_line_var = tk.BooleanVar()
draw_line_var.set(True)
draw_line_checkbox = tk.Checkbutton(root, text="Rysuj linie",
```

```
        variable=draw_line_var)
    draw_line_checkbox.pack()

    # Opcja odtwarzania sygnału dźwiękowego
    play_sound_var = tk.BooleanVar()
    play_sound_var.set(True)
    play_sound_checkbox = tk.Checkbutton(root, text="Odtwarzaj sygnał dźwiękowy",
        variable=play_sound_var)
    play_sound_checkbox.pack()

    root.mainloop()

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

11 Opis wykorzystanych modeli i metod przetwarzania obrazów

Biblioteka Haar Cascade wbudowana w OpenCV to technika wykorzystywana w przetwarzaniu obrazów do wykrywania obiektów, takich jak twarze, oczy, samochody itp. Metoda ta została zaproponowana przez Viola i Jonesa w 2001 roku i jest szeroko stosowana ze względu na swoją skuteczność i wydajność. Teoretycznie modele te składają się z zestawu cech, które są klasyfikowane jako pozytywne lub negatywne. Cechy te są wykorzystywane do budowy klasyfikatora, który jest w stanie ocenić, czy dany obszar obrazu zawiera poszukiwany obiekt.
Proces detekcji składa się z kilku kroków:

1. Przygotowanie zestawu cech: Na podstawie analizy zbioru treningowego wybierane są najlepsze cechy, które mają zdolność do rozróżniania obiektów od tła.
2. Budowa kaskady klasyfikatorów: Wybrane cechy są wykorzystywane do stworzenia kaskady klasyfikatorów, która jest sekwencją prostych klasyfikatorów binarnych (np. drzewa decyzyjne).
3. Skanowanie obrazu: Okno przesuwne jest przesuwane po obrazie, a na każdym etapie kaskady są stosowane kolejne klasyfikatory. Jeśli obszar obrazu nie spełnia warunków klasyfikatora na danym etapie, proces skanowania przechodzi do następnego obszaru.
4. Filtrowanie fałszywych pozytywów: Po wykryciu potencjalnych obiektów przeprowadzane są dodatkowe testy w celu eliminacji fałszywych pozytywów.

Metody przetwarzania oparte na kaskadach są efektywne ze względu na ich szybkość działania i możliwość wykrywania obiektów w czasie rzeczywistym. Są one szeroko stosowane w aplikacjach do rozpoznawania twarzy, detekcji ruchu, monitoringu bezpieczeństwa itp. Pomimo swojej skuteczności, mogą być wrażliwe na zmienne warunki oświetleniowe i skomplikowane tła. Jednak w połączeniu z innymi technikami przetwarzania obrazu mogą stanowić potężne narzędzie do detekcji obiektów.

12 Wyniki testów

Testy zostały przeprowadzone na nagraniach znalezionych w Internecie, głównie na YouTube, oraz na materiałach własnych. Testom zostala poddana wystarczająco duża liczba filmów, aby efektywnie sprawdzić skuteczność wykrywania przeszkód takich jak człowiek czy inny samochód z uwzględnieniem różnych warunków pogodowych oraz pór dnia.

1. W teście za cofającym pojazdem znajduje się inny samochód. Sprawdzana jest poprawność wykrywania innego samochodu za pojazdem oraz zgodność z rzeczywistością rysowanych linii pomocniczych. Video: auto. Samochód został poprawnie wykryty.

2. W teście za cofającym pojazdem przechodzi pieszy. Sprawdzana jest poprawność wykrywania pieszego za pojazdem. Video: pieszy1. W przeprowadzonym teście pieszy nie został właściwie wykryty.

3. W teście za cofającym pojazdem przechodzi pieszy. Sprawdzana jest poprawność wykrywania pieszego za pojazdem. Video: pieszy2. W teście pieszy został wykryty jedynie na kilku klatkach.

4. W teście sprawdzano poprawność wykrywania obiektu samochód przez model. Na nagraniu widoczne są pojazdy poruszające się drogą szybkiego ruchu. Video: auta. W przeprowadzonym teście zdecydowana większość pojazdów została wykryta zapewniając bardzo zadowalający wynik.

5. W teście sprawdzamy poprawność wykrywania obiektu pieszy przez model. Na nagraniu widoczni są piesi poruszający się na przejściu dla pieszych. Video: piesi. W przeprowadzonym teście zdecydowana większość pieszych została wykryta zapewniając zadowalający wynik.

6. W teście sprawdzamy poprawność wykrywania obiektu samochód przez model. Na nagraniu widoczne są pojazdy uczestniczące w kolizji na drodze. Video: auta4. W przeprowadzonym teście pojazdy zostały wykryte, widoczna jednak była większa łatwość wykrywania pojazdów jasnych niż ciemnych.

7. W teście sprawdzamy poprawność wykrywania obiektu samochód przez model. Na nagraniu widoczne są pojazdy poruszające się drogą. Nagranie powstało w warunkach nocnych, co dodatkowo sprawdza poprawność modelu w różnych porach dnia. Video: autanoc. W teście pojazdy zostały wykryte jedynie na kilku klatkach.

13 Wnioski z projektu

W projekcie udało się zaimplementować wykrywanie pieszego oraz samochodu. Zaimplementowany jest także konfigurator z interfejsem użytkownika, rysowanie linii pomocniczych oraz sygnał ostrzegawczy dźwiękowy. Nie udało się zrealizować wykrywania słupka oraz ściany, głównie ze względu na ograniczone możliwości istniejących modeli do detekcji obrazów. Zaimplementowana kamera cechuje się większą skutecznością wykrywania obiektów w warunkach dziennych i na filmach o lepszej jakości. Natomiast uzyskuje mniej satysfakcjonujące wyniki w warunkach nocnych czy dla pojazdów w ciemnych kolorach.

13.1 Analiza procesu detekcji obiektów

Metoda kaskad Haara w połączeniu z klasyfikatorami jest skuteczną techniką do detekcji obiektów na obrazach. Skuteczność detekcji jest uzależniona od odpowiedniego doboru cech oraz parametrów klasyfikatora.

13.2 Optymalizacja detekcji

Dostosowanie parametrów detekcji do konkretnego zastosowania może być kluczowe dla uzyskania optymalnych wyników.

13.3 Wrażliwość na warunki otoczenia

Detekcja obiektów za pomocą kaskad Haara może być podatna na zmienne warunki oświetleniowe oraz skomplikowane tła. W celu poprawy wydajności detekcji warto rozważyć zastosowanie technik pre- i postprocessingowych, takich jak wyrównywanie histogramu czy usuwanie szumów.

13.4 Potrzeba dalszych badań

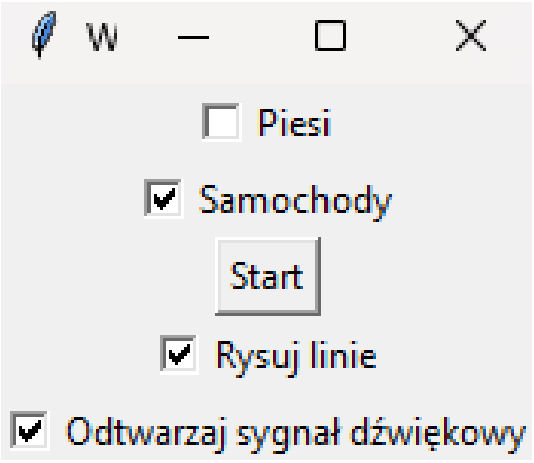
Pomimo osiągniętych wyników istnieje potrzeba dalszych badań nad doskonaleniem detekcji obiektów, zwłaszcza w kontekście złożonych scenariuszy i warunków pracy. Dodatkowe testy na różnorodnych zbiorach danych mogą przyczynić się do poprawy skuteczności algorytmu.

Podsumowując, prace nad stworzonym oprogramowaniem zakończyły się w większości sukcesem, ponieważ stworzona kamera poprawnie działa i posiada prosty, przyjazny dla użytkownika interfejs oraz spełnia swoje zadanie.

14 Instrukcja obsługi programu

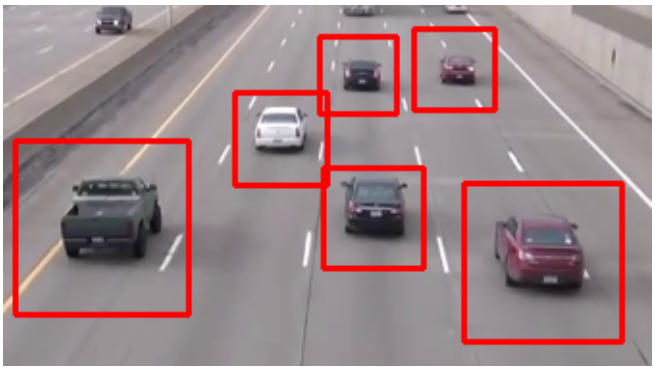
1. Należy zainstalować wymagane biblioteki (OpenCV, Tkinter, Numpy) oraz upewnić się, że pliki klasyfikatorów Haar Cascades są dostępne. W folderze z programem powinny znajdować się następujące pliki: cars.xml, pedestrian.xml oraz video.mp4 które będziemy przetwarzać. Program należy uruchomić w terminalu za pomocą komendy: python3 project.py.

2. Interfejs użytkownika składa się z pól wyboru jaki rodzaj obiektu chcemy wykrywać, decyzji czy rysować linie pomocnicze oraz decyzji czy chcemy słyszeć sygnał dźwiękowy. Można wybrać dowolną liczbę opcji. Następnie należy kliknąć przycisk "Start"aby rozpocząć detekcję.



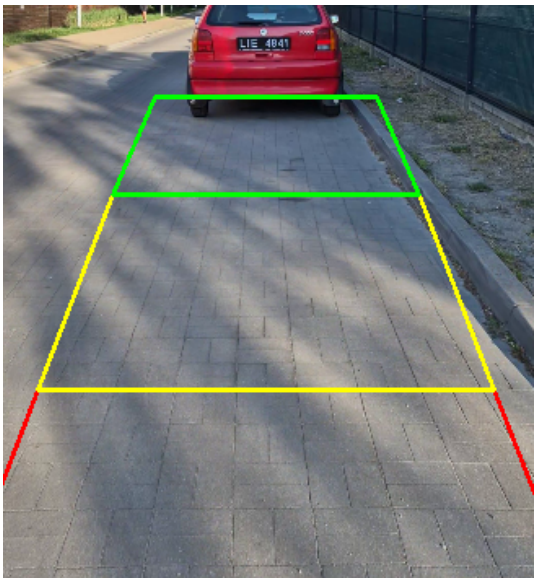
Rysunek 8: Interfejs użytkownika

3. Program wykrywa i zaznacza wykryte obiekty czerwonym prostokątem.



Rysunek 9: Wykryte obiekt

4. Dodatkowo rysowane są linie pomocnicze w trzech kolorach: czerwonym, żółtym i zielonym. Umożliwiają one interpretację odległości za pojazdem.



Rysunek 10: Linie pomocnicze

5. Program wydaje ostrzegawczy sygnał dźwiękowy, gdy wykryty obiekt znajdzie się w zakresie czerwonych linii pomocniczych.

6. Aby zakończyć działanie programu należy wcisnąć przycisk 'q' na klawiaturze.