Sprawozdanie Etap I

Rozpoznawanie i Przetwarzanie Obrazów

Stanisław Kurzyp - 264477@student.pwr.edu.pl
Michał Korczak - 263923@student.pwr.edu.pl

Prowadzący: dr inż. Tomasz Serafin
Politechnika Wrocławska
Semestr letni 2024

Spis treści

1 Wstęp **5**

2 Syntetyczny opis tematu projektu i planowanego do wykonania systemu **5**
2.1 Analiza istniejących systemów . . . 5

3 Case studies **6**
3.1 Opis obsługiwanych sytuacji . . . 6
3.2 Prezentacja scenariuszy sytuacji . . . 7

4 Opis funkcjonalny systemu **10**
4.1 Wymagania architektoniczne . . . 10
4.1.1 Modularność systemu . . . 10
4.1.2 Wydajność . . . 11
4.1.3 Przenośność . . . 11
4.2 Wymagania biznesowe . . . 11
4.2.1 Zgodność z przepisami . . . 11
4.2.2 Dobór odpowiedniego sprzętu . . . 11
4.3 Wymagania funkcjonalne . . . 11
4.3.1 Wykrywanie obiektu . . . 11
4.3.2 Konfiguracja oprogramowania . . . 11
4.3.3 Informacja zwrotna dla użytkownika . . . 12
4.3.4 Obliczanie odległości od obiektu . . . 12
4.3.5 Wyznaczanie pola wrażliwości . . . 14
4.4 Tabela wymagań . . . 18

5 Architektura wysokopoziomowa systemu **18**
5.1 Komponenty oprogramowania . . . 18
5.1.1 Interfejs użytkownika . . . 18
5.1.2 Moduł przetwarzania obrazu . . . 18
5.1.3 Moduł ostrzeżeń i asystowania kierowcy . . . 19
5.1.4 Moduł integracji z pojazdem . . . 19
5.2 Komponenty sprzętowe . . . 19
5.2.1 Kamera cofania . . . 19
5.2.2 Komputer pokładowy . . . 19

6 Architektura logiczna systemu **20**
6.1 Strukturalny widok architektury . . . 20
6.1.1 Konfigurator . . . 20
6.1.2 Moduły wykonawcze . . . 20
6.2 Behawioralny widok architektury . . . 20
6.2.1 Konfiguracja systemu . . . 20
6.2.2 Współpraca modułów wykonawczych . . . 20
6.3 Implementacyjny widok architektury . . . 21
6.3.1 Modularyzacja . . . 21
6.3.2 Przenośność . . . 21

7 Dobór technologii **21**
7.1 Wymagania techniczne . . . 21
7.2 Wymagania sprzętowe . . . 21
7.3 Kwestie sieciowe i bezpieczeństwa . . . 21
7.4 Wymagania dotyczące kamer . . . 22
7.5 Język programowania + biblioteki . . . 22
7.6 Konteneryzacja . . . 22

8 Plan prac nad projektem **22**

9 Testy projektu **22**
9.1 Test 1 - Martwe strefy . . . 22
9.1.1 Warunki początkowe . . . 23
9.1.2 Inne zagrożenia . . . 23
9.1.3 Efekt działania systemu . . . 23
9.2 Test 2 - Wykrycie obiektu w polu wrażliwości . . . 23
9.2.1 Warunki początkowe . . . 23
9.2.2 Inne zagrożenia . . . 23
9.2.3 Efekt działania systemu . . . 23
9.3 Test 3 - Obiekt poza polem wrażliwości . . . 23
9.3.1 Warunki początkowe . . . 23
9.3.2 Inne zagrożenia . . . 23
9.3.3 Efekt działania systemu . . . 23
9.4 Test 4 - Obiekt widoczny częściowo . . . 23
9.4.1 Warunki początkowe . . . 23
9.4.2 Inne zagrożenia . . . 23
9.4.3 Efekt działania systemu . . . 24
9.5 Test 5 - Wykrycie obiektu w całości . . . 24
9.5.1 Warunki początkowe . . . 24
9.5.2 Inne zagrożenia . . . 24
9.5.3 Efekt działania systemu . . . 24
9.6 Test 6 - Obiekt widoczny w całości, poza polem wrażliwości . . . 24
9.6.1 Warunki początkowe . . . 24
9.6.2 Inne zagrożenia . . . 24
9.6.3 Efekt działania systemu . . . 24
9.7 Żródła materiałów wideo . . . 24

10 Źródła **24**

11 Etap II - efekty pracy **25**
11.1 Rozpoznawanie obiektów . . . 25
11.2 Wyliczanie odległości . . . 25
11.3 Rysowanie linii ograniczających strefę wrażliwości . . . 25

12 Etap III - podsumowanie projektu **26**
12.1 Opis zrealizowanych prac w projekcie . . . 26
12.2 Architektura niskopoziomowa wraz z fragmentami kodu programu . . . 26
12.2.1 Strony użytkownika . . . 27
12.2.2 Strumień wideo we Flasku . . . 27
12.2.3 Analiza wideo . . . 28
12.2.4 Funkcja custom_boxes . . . 28
12.2.5 Funkcja draw_box . . . 28
12.2.6 Wyjaśnienie pozostałej części systemu . . . 29
12.3 Opis wykorzystywanych modeli oraz metod przetwarzania obrazów . . . 29
12.4 Wyniki testów i eksperymentów . . . 30
12.4.1 Test 1 - Martwe strefy . . . 30
12.4.2 Test 2 - Wykrycie obiektu w polu wrażliwości . . . 31
12.4.3 Test 3 - Wykrycie obiektu poza polem wrażliwości . . . 32
12.4.4 Test 4 - Obiekt widoczny częściowo . . . 33
12.4.5 Pozostałe testy . . . 34
12.5 Wnioski z projektu . . . 34
12.5.1 Skuteczność rozwiązania . . . 34
12.5.2 Poprawność dobranych technologii . . . 34
12.5.3 Ocena obsługi sytemu . . . 34
12.5.4 Ocena podejścia . . . 34
12.6 Dokumentacja użytkownika . . . 35
12.6.1 Wymagania . . . 35
12.6.2 Uruchomienie systemu . . . 35
12.6.3 Interfejs użytkownika . . . 35

Spis rysunków

1 Schemat działania systemu Audi Matrix LED . . . 6
2 Umiejscowienie kamery - rzut z tyłu . . . 8
3 Umiejscowienie kamery - rzut z boku. Przykładowe sytuacje . . . 9
4 Umiejscowienie kamery - rzut z góry. Przykładowe sytuacje . . . 10
5 Proporcje na obrazie z wykrytym obiektem . . . 13
6 Rzut boczny sytuacji pomiaru odległości . . . 14
7 Rzut 3D otoczenia kamery . . . 16
8 Rzut z góry . . . 16
9 Rzut 2.5D strefy wrażliwości . . . 17
10 Schemat architektury wysokopoziomowej . . . 19
11 Schemat architektury wysokopoziomowej z uwzględnieniem sprzętu . . . 20
12 Schemat architektury logicznej . . . 21
13 Wykrywanie obiektów . . . 25
14 Rysowanie linii . . . 26
15 Architektura YOLO . . . 29
16 Widok z kamery . . . 30
17 Widok martwej strefy . . . 31
18 Obiekt wykryty w całości . . . 31
19 Obiekt na skraju pola wrażliwości . . . 32
20 Obiekt poza polem wrażliwości . . . 32
21 Obiekt poza polem wrażliwości - naprzeciwko . . . 32
22 Obiekt widoczny częściowo . . . 33
23 System po uruchomieniu . . . 35
24 Strona główna . . . 36
25 Kamera na żywo . . . 36

26 Test wideo 37

1 Wstęp

Poniższy dokument dotyczy projektu oprogramowania do przetwarzania obrazu z kamery cofania samochodu. Projekt realizowany jest w ramach kursu - Rozpoznawanie i Przetwarzanie Obrazów - na Politechnice Wrocławskiej.

2 Syntetyczny opis tematu projektu i planowanego do wykonania systemu

W poniższej dokumentacji wykorzystywane są specyficzne pojęcia. W celu zapewnienia jednoznaczności treści, poniżej zaprezentowane są ich definicje w kontekście tej pracy.
Definicje:

- Pole wrażliwości - ustalony obszar w przestrzeni w którym pojawienie się/wykrycie obiektu uważane jest za zagrożenie.
- Obszar widoczności kamery - obszar w przestrzeni z którego informacje zbierane są przez sensor kamery. Jego wielkość zależna jest od kąta widzenia kamery.
- Obiekt będący zagrożeniem - obiekt znajdujący się w ustalonym polu wrażliwości systemu. Celem systemu jest wykrywanie obiektów-zagrożeń, aby umożliwić ich ominięcie przez kierowcę.

Celem projektu jest przygotowanie systemu obsługującego kamerę cofania umieszczoną z tyłu samochodu. Program ma realizować następujące funkcjonalności:

1. Możliwość konfiguracji systemu.
2. Prezentowanie danych z kamery na wyświetlaczu z nałożonymi przetworzonymi przez program informacjami.
3. Wykrywanie i rozpoznawanie obiektów z pewnej odgórnie ustalonej puli obiektów.
4. Analizowanie położenia obiektów w celu ustalenia czy generują sytuację potencjalnie niebezpieczną.
5. Przekazywanie informacji zwrotnych o wykrytych niebezpiecznych sytuacjach użytkownikowi.

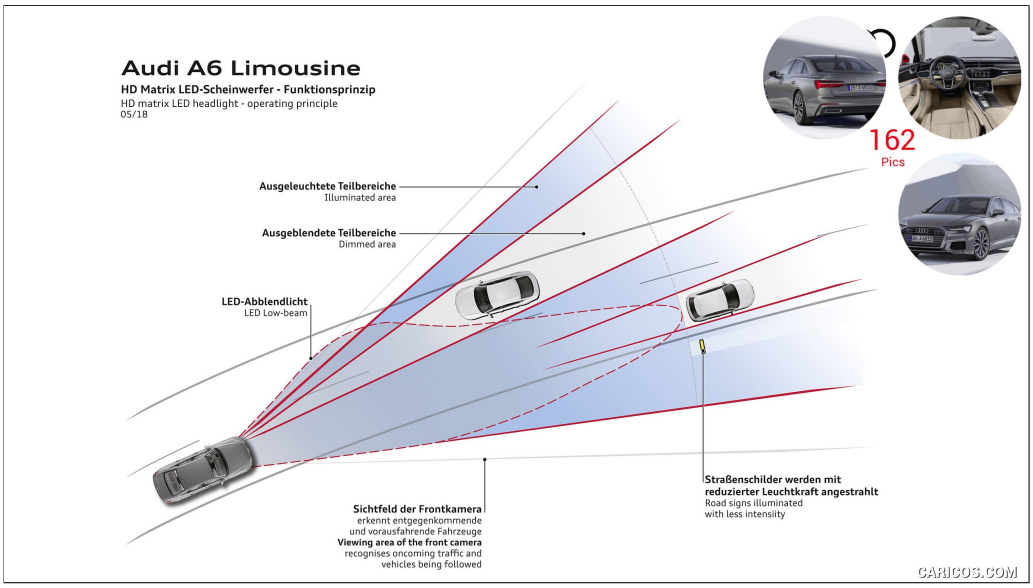
2.1 Analiza istniejących systemów

Spośród obecnie istniejących systemów na rynku można wyszczególnić dwie kategorie:

- Systemy wbudowane - samochód zaopatrzony został w system wspierania cofania przez producenta
- Systemy dodatkowe - zestaw wyświetlacza i kamery do montażu własnego

W drugim przypadku wykorzystywana jest zazwyczaj łączność bezprzewodowa co znacząco ułatwia samodzielny montaż. Kamery zastosowane w tych systemach zazwyczaj nagrywają obraz w jakości 1080p i spełniają normy wodoodporności IP69 lub lepsze. Warto zwrócić uwagę na kąty widzenia stosowanych kamer. Wahają się one od 140 do 180 stopni. Wiele z systemów nie polega tylko na kamerze, lecz stosuje jeszcze czujniki odległości poprawiające skuteczność na przykład przy gorszej widoczności po zmroku lub w nocy.
W systemach wbudowanych w samochody standardem jest obecnie stosowanie dynamicznych linii pomocniczych cofania. Reprezentują one przewidywany tor ruchu pojazdu w oparciu o dane z czujników informujących o kącie skręcenia przednich kół. Rozwiązanie takie pozwala użytkownikowi w szczególności na wygodniejsze parkowanie.
Rozwiązania komercyjne w zakresie kamer cofania nie wykorzystują modeli uczenia maszynowego w celu rozpoznawania obiektów i wykrywania obiektów. W tej dziedzinie kamera ma najczęściej funkcję poglądową, a czujniki odległości zapewniają wiarygodny pomiar odległości od obiektów. Ponadto często obejmują one większy kąt niż 180 stopni i występują w dużej ilości by zagwarantować bezawaryjność.
Wykrywanie i rozpoznawanie obiektów w motoryzacji częściej można spotkać w kontekście innych kamer niż

kamery cofania. Jednym z przykładów są samochody do weryfikacji kupna biletu w strefach płatnego parkowania. Posiadają one kamery wykrywające rejestracje samochodów obok których przejeżdżają, a następnie sprawdzające w bazie danych czy samochód o danej rejestracji ma opłacony bilet. Podobnym przykładem mogą być kamery analizujące ruch w mieście. Rozpoznają one samochody w celu ich zliczania, ale również identyfikują ich typ co pozwala na ustalenie typu ruchu przepływającego przez badane skrzyżowanie. Powyższe systemy skupiają się na wykrywaniu obiektów, a następnie klasyfikowaniu ich. W tym celu wykorzystywane są modele uczenia maszynowego, które wyszkolone zostały w oparciu o specjalne bazy.
Warto zwrócić uwagę jednak na to, że kamera cofania oprócz wykrywania obiektów powinna być też w stanie określić odległość od nich dzielącą. W przypadku obiektów o nieznanych wymiarach nie da się zastosować wprost skali i proporcji, a zadanie staje się nietrywialne. W ramach projektu zostanie zaproponowane podejście do rozwiązanie tego problemu, jednakże wiąże się ono z pewnymi ograniczeniami i założeniami. Rozwinięcie tego wątku znajduje się w 4.3.4.
Ciekawym przykładem zastosowania wykrywania i rozpoznawania obiektów w czasie rzeczywistym przez kamerę umieszczoną z przodu pojazdu jest nowatorski system świateł zaprojektowany przez Audi. Audi Matrix LED 1 wykorzystuje zaawansowane kamery do wykrywania i rozpoznawania pojazdów na drodze. Dzięki temu inteligentnemu rozwiązaniu, reflektory automatycznie dostosowują kierunek i intensywność światła, minimalizując oślepianie innych kierowców, zapewniając jednocześnie optymalną widoczność i bezpieczeństwo podczas jazdy.



Rysunek 1: Schemat działania systemu Audi Matrix LED

3 Case studies

3.1 Opis obsługiwanych sytuacji

Oprogramowanie przygotowane jest tak by wykrywać potencjalnie niebezpieczne sytuacje zarejestrowane przez kamerę cofania pojazdu. Za niebezpieczne sytuacje uważa się te, w których istnieje znaczące ryzyko, że pojazd poruszając się w tył, po danym torze ruchu, zderzy się z innym obiektem. Obiektem tym może być zarówno statyczny obiekt taki jak mur, słup lub drzewo, ale i również przemieszczający się obiekt taki jak człowiek, samochód lub rower. Oprogramowanie umożliwia wykrycie następujących obiektów: rower, znak dorgowy, człowiek.
By sytuacja została zinterpretowana jako niebezpieczna, obiekt musi znajdować się w ustalonej strefie wrażliwości. Oprogramowanie przetwarzające informacje z kamery wykryje obiekt, a następnie przeprowadzi obliczenia w celu określenia jego położenia względem kamery. Gdy potencjalne niebezpieczeństwo zostanie

potwierdzone, zakomunikuje to kierowcy poprzez wyświetlenie komunikatu na ekranie oraz sygnał dźwiękowy.

Obsługiwane sytuacje można podzielić na kilka poniższych kategorii

1. Brak wykrytych obiektów - w tej sytuacji system pozostaje w stanie wykrywania obiektów.
2. Wykryto obiekt
   (a) Obiekt znajduje się w bezpiecznej odległości - obiekt został wykryty i rozpoznany przez kamerę, lecz znajduje się w bezpiecznej odległości(poza polem wrażliwości), więc nie wywoła informacji zwrotnej dla kierowcy. System pozostaje w stanie wykrywania obiektów.
   (b) Obiekt znajduje się w polu wrażliwości - obiekt został wykryty, rozpoznany oraz znajduje się w wystarczająco blisko by poinformować o nim kierowcę. System pozostaje w stanie wykrywania obiektów.

Dokładny opis sytuacji znajduje się w podrozdziale 3.2

3.2 Prezentacja scenariuszy sytuacji

Do rozpoczęcia analizy dopuszczalnych scenariuszy sytuacji konieczne jest najpierw określenie położenia kamery i zdefiniowanie jej podstawowych parametrów. W przedstawionych poniżej scenariuszach, kamera umieszczona jest z tyłu pojazdu, w połowie szerokości i na wysokości środka wysokości pojazdu. Zobrazowane jest to na rysunku 2.
Kluczowymi parametrami dla analizy scenariuszy sytuacji są:

- Kąt widzenia obiektywu kamery - wraz ze wzrostem kąta widzenia, zwiększa się zakres jaki kamera rejestruje. Większy zakres przekłada się na zwiększenie bezpieczeństwa poprzez wcześniejsze wykrywanie obiektów. Ryzykiem związanym z dużym kątem widzenia jest zniekształcenie obrazu prowadzące do deformacji obiektów znajdujących się na jego skrajach (efekt rybiego oka). Może to negatywnie wpływać na skuteczność stosowanego algorytmu wykrywania i rozpoznawania obiektów.
- Wysokość umieszczenia kamery - system posiadający nieposiadający czujników odległości w celu wyznaczenia odległości od wykrytego obiektu przeprowadza obliczenia matematyczne. Wysokość umieszczenia kamery jest kluczowym parametrem w procesie wyznaczania tej wartości. Ponadto warto zwrócić uwagę na umieszczenie kamery tak by znajdowała się w okolicach środka wysokości bryły samochodu co zapewni jej optymalny zakres widzenia.
- Szerokość umieszczenia kamery - w specjalnych przypadkach umieszczenie kamery po środku osi X bryły samochodu może być niemożliwe. W celu zapewnienia poprawności działania algorytmu wymagana jest znajomość dokładnego położenia (przesunięcia) kamery w osi X względem środka.
- Pole wrażliwości - system powinien wykrywać obiekty w ustalonym polu wrażliwości. Program zaznacza potencjalny tor ruchu pojazdu na obrazie jako prostokąt o długości równej długości samochodu oraz szerokości równej szerokości samochodu. Domyślnie jeśli w tym obszarze pojawi się obiekt, użytkownik zostanie poinformowany - jest to pole wrażliwości. Konfiguracja programu pozwala na zwiększenie tego zakresu tak by użytkownik otrzymywał informację o obiektach również poza jego wyznaczonym torem ruchu. Może być to szczególnie przydatne w przypadku obiektów ruchomych. Dzięki temu system powiadomi o pojawieniu się takiego obiektu wcześniej niż gdy znajdzie się on w torze ruchu pojazdu.

Rysunek 2 przedstawia przykładowe umieszczenie kamery z tyłu samochodu.

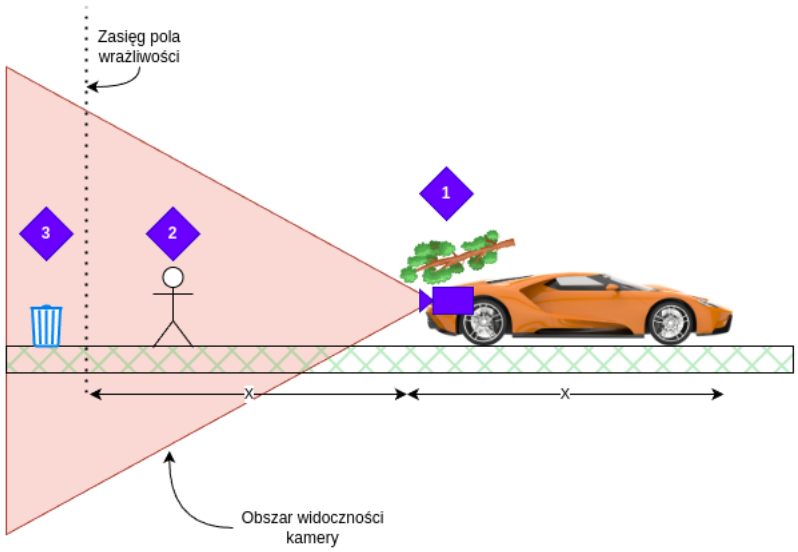
Rysunek 2: Umiejscowienie kamery - rzut z tyłu

Rysunek 3 przedstawia schemat rzutu z boku na samochód korzystający z urządzenia wraz z jego otoczeniem. Zaprezentowane są na nim również 3 sytuacje charakterystyczne dla problemu wykrywania obiektów przez kamerę cofania. Wszystkie możliwe sytuacje z którymi spotka się system, da się sprowadzić do tych zaprezentowanych na rysunkach 3 oraz 4.
Przeanalizujmy sytuacje występujące na rysunku 3.

1. Sytuacja 1 - nad cofającym samochodem znajduje się obiekt stanowiący zagrożenie - w tym przypadku reprezentowany przez gałąź. Obiekt leży w obrysie samochodu, ale poprzez swoje wysokie położenie nie został wykryty przez kamerę - nie mieści się on w kącie widzenia kamery. W takiej sytuacji system nie jest w stanie go wykryć. Analogiczny problem może wystąpić w przypadku niskich obiektów położonych pod kamerą. Są to dwie martwe strefy, które zminimalizować można za pomocą zastosowania kamery o szerszym kącie widzenia. Należy jednak zaznaczyć, że ich występowanie jest praktycznie nieuniknione i bez zastosowania odpowiednich dodatkowych czujników niemożliwe jest rozwiązanie ich problemu.

2. Sytuacja 2 - obiekt znajduje się w obszarze widoczności kamery oraz w zasięgu pola wrażliwości systemu. W tym przypadku widoczna jest wizualizacja w 2D. Zakładając, że obiekt ten znajduje się w zakresie pola wrażliwości również w drugiej jego osi, wywoła on odpowiednią informację zwrotną dla użytkownika systemu.

3. Sytuacja 3 - obiekt znajduje się w obszarze widoczności kamery, lecz poza zasięgiem pola wrażliwości systemu. Wykrycie takiego obiektu nie wywoła informacji zwrotnej dla użytkownika, ponieważ nie jest on uważany za zagrożenie.



Rysunek 3: Umiejscowienie kamery - rzut z boku. Przykładowe sytuacje

Rysunek 4 przedstawia schemat rzutu z lotu ptaka na samochód korzystający z urządzenia wraz z jego otoczeniem. Zaprezentowane są na nim również 5 sytuacji charakterystycznych dla problemu wykrywania obiektów przez kamerę cofania. Warto zwrócić uwagę na poglądowy schemat pola wrażliwości.

1. Sytuacja 1 - obiekt reprezentowany przez krzak znajduje się poza obszarem widoczności i poza polem wrażliwości - nie jest on zagrożeniem.

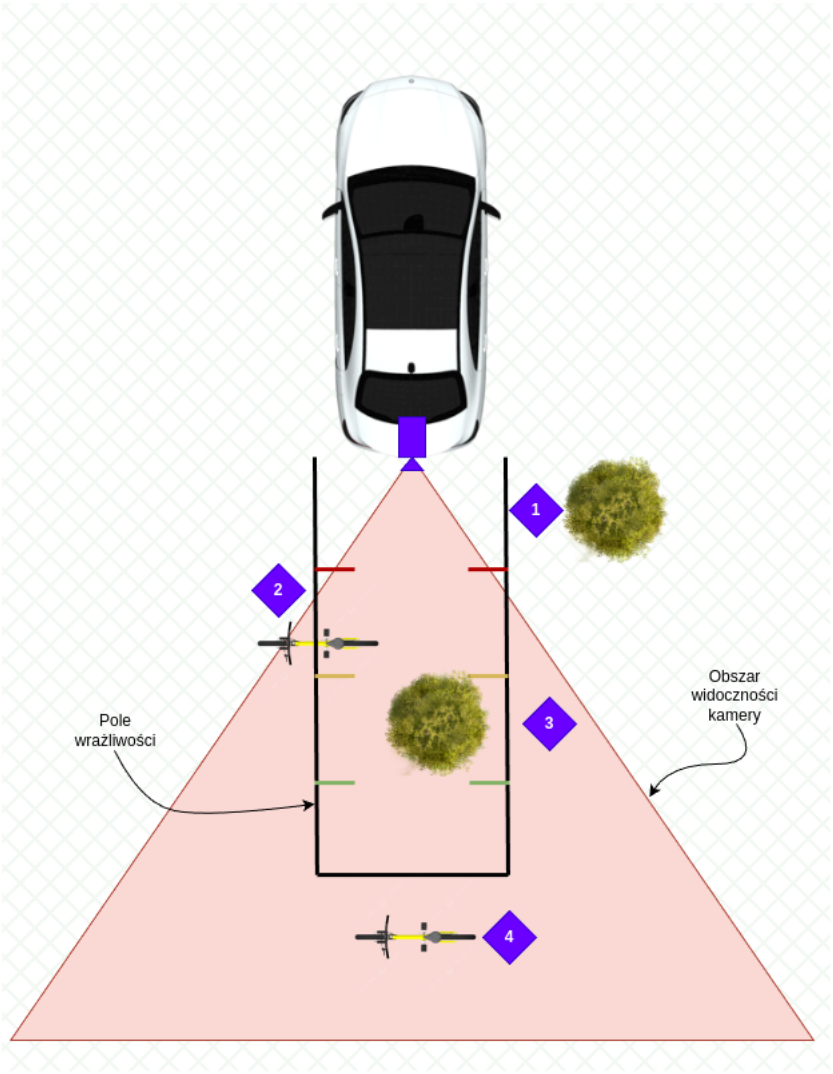
2. Sytuacja 2 - obiekt reprezentowany przez rower znajduje się na skraju pola wrażliwości oraz obszaru widoczności. W tym przypadku rozwój sytuacji zależy od skuteczności modelu rozpoznającego obiekty. Mogą wystąpić dwie następujące sytuacje:

   (a) Obiekt zostanie wykryty i rozpoznany pomimo, że nie jest widoczny w całości. Znajduje się on również w polu wrażliwości więc użytkownik otrzyma informację.

   (b) Obiekt nie zostanie wykryty co może być spowodowane tym, że jest widoczna zbyt mała jego część. Użytkownik nie otrzyma informacji o zagrożeniu.

3. Sytuacja 3 - obiekt reprezentowany przez krzak znajduje się w całości w obszarze widoczności oraz w polu wrażliwości. Zostaje on wykryty i rozpoznany, a użytkownik otrzyma informację.

4. Sytuacja 4 - obiekt reprezentowany przez rower znajduje się w całości w obszarze widoczności kamery, lecz leży poza polem wrażliwości. Użytkownik nie otrzyma informacji, ponieważ zagrożenie nie zostało wykryte.



Rysunek 4: Umiejscowienie kamery - rzut z góry. Przykładowe sytuacje

Istotnym wnioskiem z analizy potencjalnych scenariuszy jest fakt, że pośród powyżej opisanych przypadków występują takie, w których system jest nieskuteczny i poleganie na nim w pełni może prowadzić do niebezpiecznych sytuacji.

4 Opis funkcjonalny systemu

4.1 Wymagania architektoniczne

Dokładny opis wymagań architektonicznych zawarty jest w 5 oraz 6.

4.1.1 Modularność systemu

System powinien być zbudowany w sposób modułowy, umożliwiający łatwą rozbudowę i modyfikację poszczególnych komponentów. Oznacza to, że użytkownik może łatwo wymienić wyświetlacz lub kamerę na inną.

4.1.2 Wydajność

System powinien być zoptymalizowany pod kątem wydajności, aby zapewnić płynne działanie w czasie rzeczywistym. Jest to kluczowe dla jego niezawodności i bezpieczeństwa użytkownika.

4.1.3 Przenośność

System powinien być przenośny, co oznacza możliwość łatwej instalacji i w różnych środowiskach (przeniesienie systemu z jednego samochodu do innego).

4.2 Wymagania biznesowe

4.2.1 Zgodność z przepisami

System musi być zgodny z normami i przepisami regulującymi stosowanie i wykorzystywanie tego typu urządzeń w samochodach. Ponadto warto zaznaczyć, że jest to tylko dodatkowe wsparcie kierowcy o ograniczonej funkcjonalności, a patrzenie w lusterka i zachowanie zasad ruchu drogowego nadal pozostaje obowiązkiem użytkownika.

4.2.2 Dobór odpowiedniego sprzętu

Odpowiedni dobór kamery ma kluczowe znaczenie dla efektywności systemu. Kamera o wąskim kącie widzenia wiąże się z znaczącymi martwymi strefami nie widocznymi dla systemu.
Dobór wyświetlacza wpływa na jakość odbieranego obrazu oraz czytelność nakładanych na niego informacji przez oprogramowanie.
Urządzenie wyświetlające powinno być również wyposażone w głośnik w celu emitowania sygnałów dźwiękowych.

4.3 Wymagania funkcjonalne

4.3.1 Wykrywanie obiektu

Wykrywanie obiektu zaimplementowane będzie z wykorzystaniem biblioteki uczenia maszynowego YOLO - "You Only Look Once". Dokładny opis algorytmu znajduje się w sekcji **??**

4.3.2 Konfiguracja oprogramowania

Konfiguracja oprogramowania ma na celu zapewnienie elastyczności i możliwości dopasowania do różnych przypadków i zastosowań. Program pozwoli również na zmianę parametrów pozwalających na personalizację ustawień ze względu na potrzeby i preferencje użytkownika. W ramach konfiguracji użytkownik otrzyma dostęp do wprowadzenia następujących parametrów:

- Umieszczenie kamery - kamera musi być umieszczona równolegle do podłoża, lecz wysokość na której zostanie umieszczona zależy od użytkownika
- Wybór kamery - użytkownik może zastosować dowolną kamerę, lecz musi podać jej dane techniczne takie jak ogniskowa obiektywu, przekątna matrycy oraz proporcje matrycy.
- Wielkość strefy wrażliwości - strefa wrażliwości domyślnie ustawiona będzie jako prostokąt o długości i szerokości samochodu. Użytkownik może jednakże powiększyć ją o preferowaną wartość co wpłynie na zwiększenie obszaru uznawanego za potencjalnie niebezpieczny.
- Wybór obiektów do wykrywania- użytkownik może wybrać z rozwijanej listy jakie obiekty mają być wykrywane przez oprogramowanie

4.3.3 Informacja zwrotna dla użytkownika

Użytkownik otrzymuje informację zwrotną jako ciągły obraz z kamery. Na obraz nałożone zostaną również prostokąty otaczające wykryte obiekty, a pojawienie się wykrytego obiektu w strefie wrażliwości dodatkowo wywoła sygnał dźwiękowy oraz spowoduje pojawienie się czerwonej obwódki wokół nagrania.

4.3.4 Obliczanie odległości od obiektu

Obliczanie odległości obiektu jest możliwe dzięki wiedzy o dokładnym położeniu kamery oraz jej parametrach. Kluczowe jest posiadanie informacji o ogniskowej obiektywu zastosowanego w kamerze, przekątnej oraz proporcji matrycy (zazwyczaj jest to 4:3, lecz zdarzają się również sensory 5:4 lub 1:1). W celu wyznaczenia własności AFOV (Angular Field Of View) kamery wykorzystuje się następujące równanie:
$AFOV = 2 \cdot \arctan\left(\frac{d}{2f}\right)$
Gdzie:

- AFOV - Angular Field Of View - kąt widzenia kamery po przekątnej
- d - przekątna matrycy
- f - ogniskowa obiektywu

Znając proporcje matrycy oraz jej przekątną możemy wyznaczyć jej dokładne wymiary w oparciu o równanie Pitagorasa oraz równanie proporcji.

$$W^2 + H^2 = d^2 \tag{1}$$

Gdzie

- W to szerokość matrycy,
- H to wysokość matrycy,
- d to przekątna matrycy,

Jeśli stosunek proporcji matrycy wynosi $\frac{r_W}{r_H}$ to:

$$\frac{W}{r_W} = \frac{H}{r_H} \tag{2}$$

Otrzymujemy z tego:

$$H = \frac{r_H}{r_W} \cdot W \tag{3}$$

Podstawiając do równianina Pitagorasa otrzymujemy:

$$W^2 + (\frac{r_H}{r_W} \cdot W)^2 = d^2 \tag{4}$$

Po przekształceniu równania możemy wyznaczyć W:

$$W = \sqrt{\frac{d^2}{1 + \left(\frac{r_h}{r_w}\right)^2}} \tag{5}$$

Następnie ponownie korzystając z proporcji matrycy, wyznaczyć H:

$$H = \frac{r_H}{r_W} * W = \frac{r_H}{r_W} \cdot \sqrt{\frac{d^2}{1 + \left(\frac{r_h}{r_w}\right)^2}} \tag{6}$$

Znając H i W wyznaczamy kolejno:

$$AFOV_h = 2 \cdot \arctan\left(\frac{W}{2f}\right) \tag{7}$$

oraz

$$AFOV_v = 2 \cdot \arctan\left(\frac{H}{2f}\right) \tag{8}$$

Założenia: kamera jest skierowana równolegle względem podłoża. Linia wychodząca równolegle do podłoża z obiektywu kamery wyznacza poziom kąta 0 stopni kamery. Jeśli horyzontalny AFOV wynosi $AFOV_h$ to zakres widzenia kamery względem kąta 0 wynosi $\frac{AFOV_h}{2}$
Znając wertykalne AFOV, wysokość położenia kamery względem ziemi oraz położenie obiektu na przechwytywanym przez kamerę obrazie, można wyznaczyć jego odległość od kamery.

Pierwszym krokiem jest ustalenie pod jakim kątem względem kamery leży obiekt, a konkretnie jego najbardziej wysunięty na dół obrazu punkt. Na rysunku 5 widoczny jest człowiek, a przerywaną linia zaznaczono jego poziom na którym znajduje się jego najbardziej wysunięty na dół punkt. Zauważmy że:

- a - wysokość obrazu można przyrównać do $AFOV_h$.
- $\frac{a}{2}$ - jeśli na środku obrazu ustalimy linię kąta 0 stopni to obraz dzieli się na dwie połowy zawierające się w poszczególnych kątach widzenia kamery równych $\frac{AFOV_h}{2}$.
- x - odległość x jest odległością od dołu wykrytego obiektu do linii połowy obrazu.
- y - odległości y jest odległością od dołu wykrytego obiektu do dołu obrazu.

W oparciu o powyższe spostrzeżenia oraz rysunek **??** można zauważyć, że:

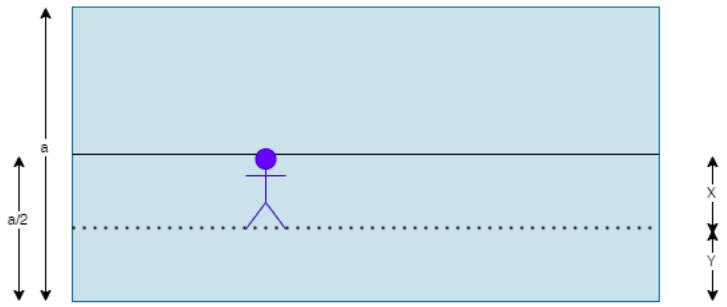
- $\frac{\alpha}{\beta} = \frac{X}{Y}$, kąt kamery w którym mieści się część dolnej połowy (połowy poniżej ustalonej linii kąta 0 stopni) obrazu znajdująca się ponad dołem obiektu jest wprost proporcjonalna do jej wysokości. Analogiczna sytuacja występuje z kątem obejmującym część poniżej obiektu oraz jej wysokość.
- Kąty α, β, γ sumują się do 90 stopni, więc jesteśmy w stanie wyznaczyć kąt γ.
- Kamera znajduje się na wysokości **H** od ziemi, a jej odległość od obiektu wynosi **L**. Formuje to trójkąt prostokątny o przyprostokątnych **H** i **L** oraz jednym z kątów równym γ

Zatem:

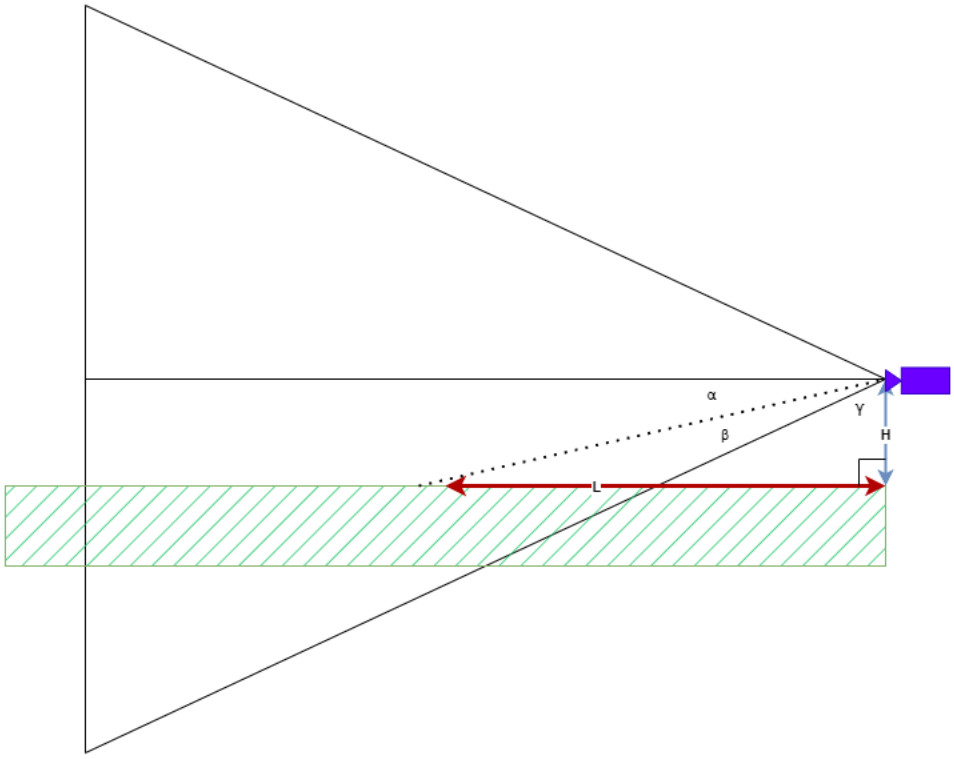
$$\frac{X}{Y} = \frac{\alpha}{\beta}, gdzie\ \alpha + \beta = \frac{AFOV_h}{2} \tag{9}$$

Korzystając z własności trygonometrycznych można wyznaczyć odległość kamery od obiektu jako:

$$L = \tan(\gamma + \beta) \cdot H = \tan(90^\circ - \alpha) \cdot H \tag{10}$$



Rysunek 5: Proporcje na obrazie z wykrytym obiektem



Rysunek 6: Rzut boczny sytuacji pomiaru odległości

4.3.5 Wyznaczanie pola wrażliwości

Pole wrażliwości domyślnie ma wymiary równe statystycznie przeciętnej długości i szerokości samochodu. W ramach konfiguracji programu, użytkownik otrzyma możliwość zwiększenia go o wybraną wartość podaną w centymetrach. Wizualizacją pola będą linie podobne do tych na rysunku 4, wyświetlające się na ekranie.
W celu narysowania linii należy ustalić położenie 4 kluczowych punktów opisujących wierzchołki pola wrażliwości. Do ustalenia położenia w osi Y użyty zostanie algorytm odwrotny do opisanego w 4.3.4. Do wyznaczenia położenia w osi Y wykorzystany zostanie poniższy tok rozumowania.

- **distance_x** - szerokość (oś X) strefy wrażliwości podana w metrach
- **distance_y** - wysokość (oś Y) strefy wrażliwości podana w metrach
- **AFOV_H** - horyzontalny kąt widzenia kamery podany w stopniach
- **IMAGE_WIDTH** - szerokość obrazu w pixelach
- **IMAGE_HEIGHT** - wysokość obrazu w pixelach

1. **Algorytm wyznaczania pola wrażliwości**

(a) **Wyznaczanie odległości w metrach odległości od obiektu**

$$\begin{aligned} &Niech \quad a = \frac{distance_x}{2}, \\ &b = ?, \\ &\alpha = \frac{AFOV_H}{2}, \\ &wtedy \quad b = \frac{a}{\tan \alpha} \end{aligned} \tag{11}$$

(b) **Wyznaczanie odległości pomiędzy linią m a linią horyzontu, w pixelach**
Stosując funkcję odwrotną do funkcji opisanej w 4.3.4, wyznaczamy odległość **l**.

(c) **Wyznaczanie długości odcinka m**
Założeniem funkcjonowania systemu jest to, że kamera nie ma kąta szerszego niż 180 stopni. W takiej sytuacji linie reprezentujące boki strefy wrażliwości zaczynać będą się na obrazie w jego rogach lub powyżej nich (tzn. na bokach obrazu, w osi Y). Założyć zatem można, że długość linii **m** wynosi zawsze **IMAGE_WIDTH**.

(d) **Wyznaczanie położenia linii n**
Linia **n** reprezentuje górę strefy wrażliwości. Znając długość strefy wrażliwości - **distance_y** - wykorzystujemy funkcję odwrotną do 4.3.4.

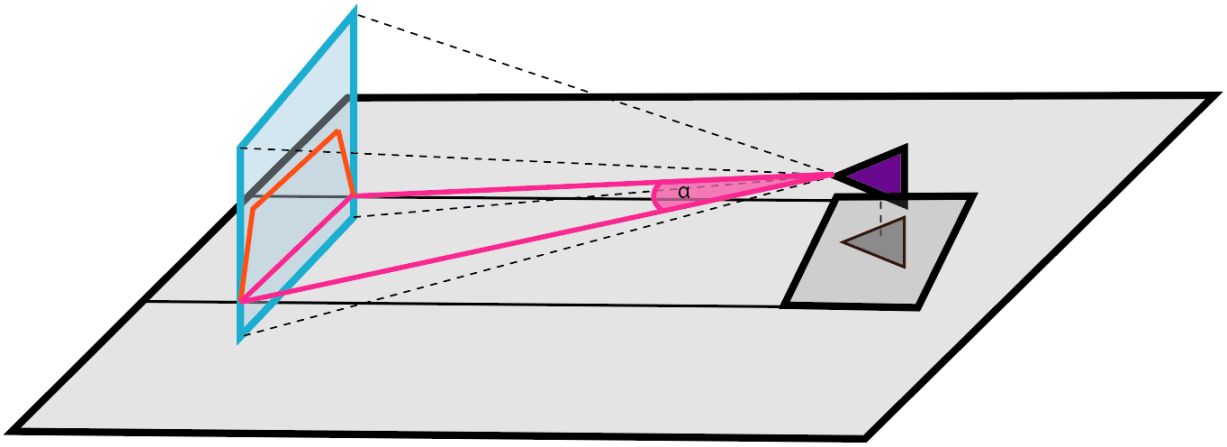
(e) **Wyznaczanie długości linii n**
Korzystając z proporcji wynikających z twierdzenia Talesa, wyznaczamy długość linii **n**.

$$n = m * \frac{k}{l} \tag{12}$$

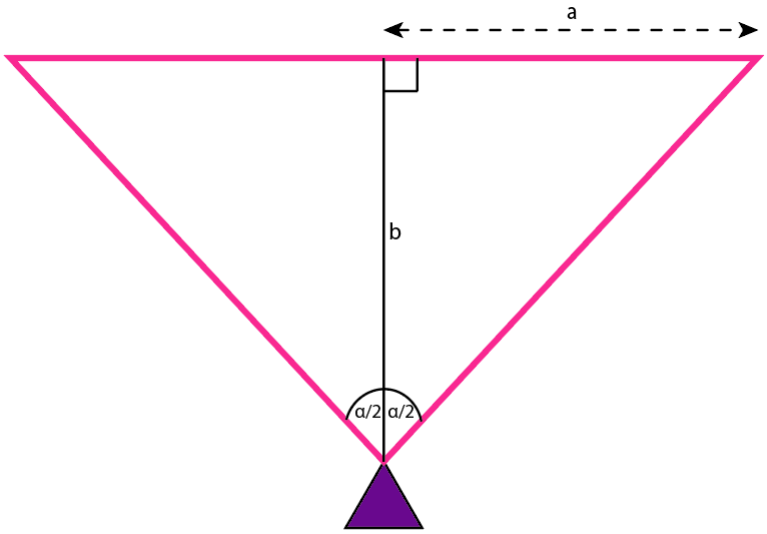
(f) **Określenie położenia punktów**
Na podstawie obliczeń wyznaczamy współrzędne punktów:

$$\begin{aligned} &top_left = (\frac{IMAGE_WIDTH}{2} - \frac{n}{2}, \frac{IMAGE_HEIGHT}{2} + k) \\ &top_right = (\frac{IMAGE_WIDTH}{2} + \frac{n}{2}, \frac{IMAGE_HEIGHT}{2} + k) \\ &bottom_left = (0, \frac{IMAGE_HEIGHT}{2} + l) \\ &bottom_right = (IMAGE_WIDTH, \frac{IMAGE_HEIGHT}{2} + l) \end{aligned} \tag{13}$$

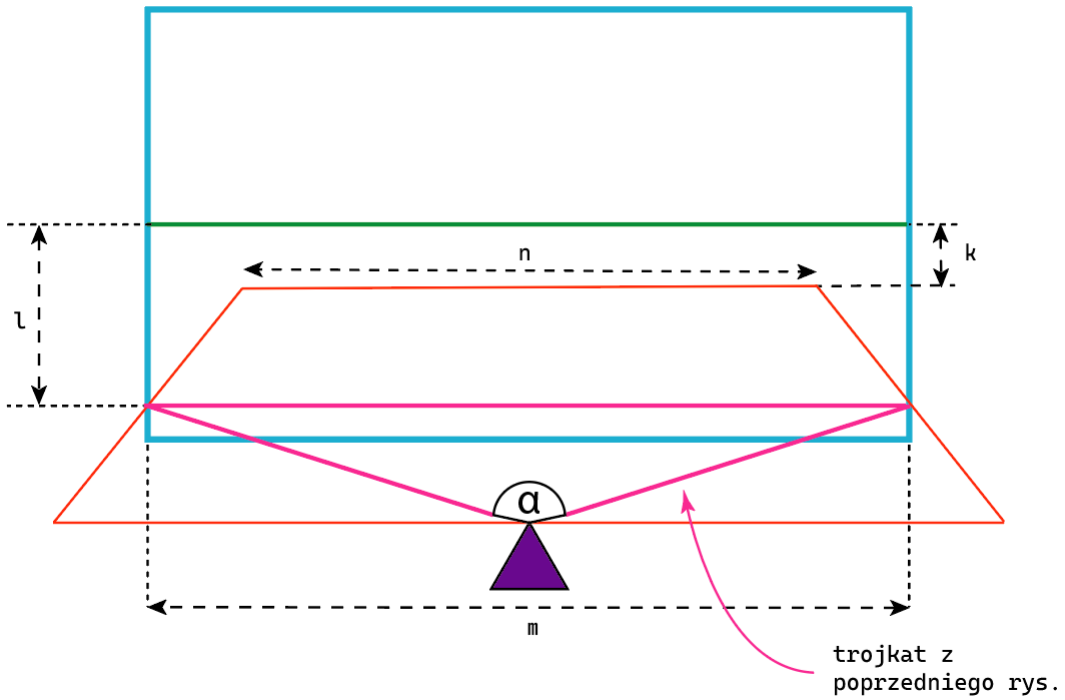
2. **Wizualizacje**



Rysunek 7: Rzut 3D otoczenia kamery



Rysunek 8: Rzut z góry



Rysunek 9: Rzut 2.5D strefy wrażliwości

4.4 Tabela wymagań

Kategoria	Konieczność	Priorytet	Wymaganie	Kryterium sukcesu	Metoda oceny
Funkcja	Konieczne	1	Wykrywanie i rozpoznawanie obiektów	System wykrywa i rozpoznaje zadane obiekty	Testy modułu odpowiedzialnego za funkcjonalność
Funkcja	Konieczne	2	Wyświetlanie linii strefy wrażliwości na ekranie	Linie są wyświetlane poprawnie	Wizualna ocena
Funkcja	Konieczne	1	Przekazywanie wizualnej informacji zwrotnej dla użytkownika o wykrytym zagrożeniu	System wyświetla informację zwrotną po wykryciu obiektu w strefie wrażliwości	Testy w warunkach funkcjonowania
Funkcja	Opcjonalne	3	Przekazywanie dźwiękowej informacji zwrotnej dla użytkownika o wykrytym zagrożeniu	System sygnalizuje dźwiękiem o wykryciu obiektu w strefie wrażliwości	Testy w warunkach funkcjonowania
Funkcja	Konieczne	1	Umożliwienie zmiany pozycji kamery w pionie	System umożliwia konfigurację obliczeń dla wysokości na jakiej znajduje się kamera	Testy w warunkach funkcjonowania
Funkcja	Konieczne	1	Umożliwienie zmiany kamery	System umożliwia konfigurację typu obiektywu i sensora stosowanego w kamerze.	Testy w warunkach funkcjonowania
Funkcja	Konieczne	1	Obliczanie odległości od wykrytych obiektów	Skuteczne obliczanie odległości	Testy w warunkach funkcjonowania

Tabela 1: Opis wymagań funkcjonalnych

5 Architektura wysokopoziomowa systemu

Architektura wysokopoziomowa obejmuje takie elementy jak sprzęt i oprogramowanie, które współdziałają w celu zapewnienia funkcjonalności systemu.

5.1 Komponenty oprogramowania

5.1.1 Interfejs użytkownika

- Zapewnia możliwość interakcji użytkownika z systemem poprzez wyświetlanie obrazu z kamery, a także interakcji z innymi funkcjami systemu (np. jego konfiguracją)

5.1.2 Moduł przetwarzania obrazu

- Jest odpowiedzialny za komunikację z kamerą i analizę pozyskanego z niej obrazu
- Identyfikuje obiekty za pomocą algorytmów służących do detekcji, segmentacji i klasyfikacji
- Wyznacza zidentyfikowane obiekty na przekazywanym obrazie z kamery

5.1.3 Moduł ostrzeżeń i asystowania kierowcy

- Generuje ostrzeżenia dla kierowcy w oparciu o przetworzony obraz
- Asystuje kierowcy podczas manewrowania pojazdem przy cofaniu

5.1.4 Moduł integracji z pojazdem

- Komunikuje system ze sprzętem pojazdu
- Odbiera dane z czujników samochodu
- Wysyła sygnały kontrolne do sprzętu pojazdu np. w celu kontroli alarmu cofania

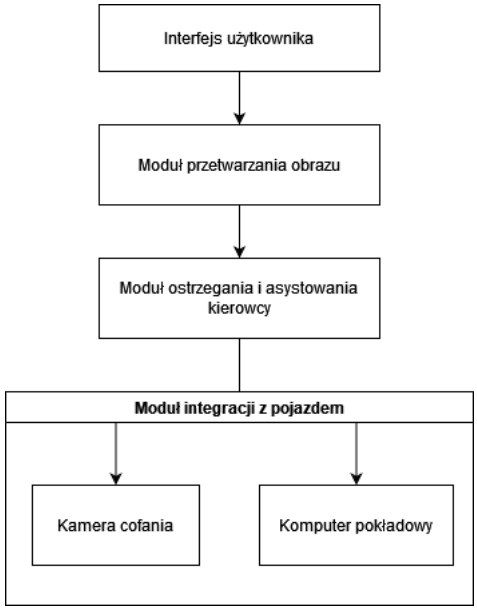
5.2 Komponenty sprzętowe

5.2.1 Kamera cofania

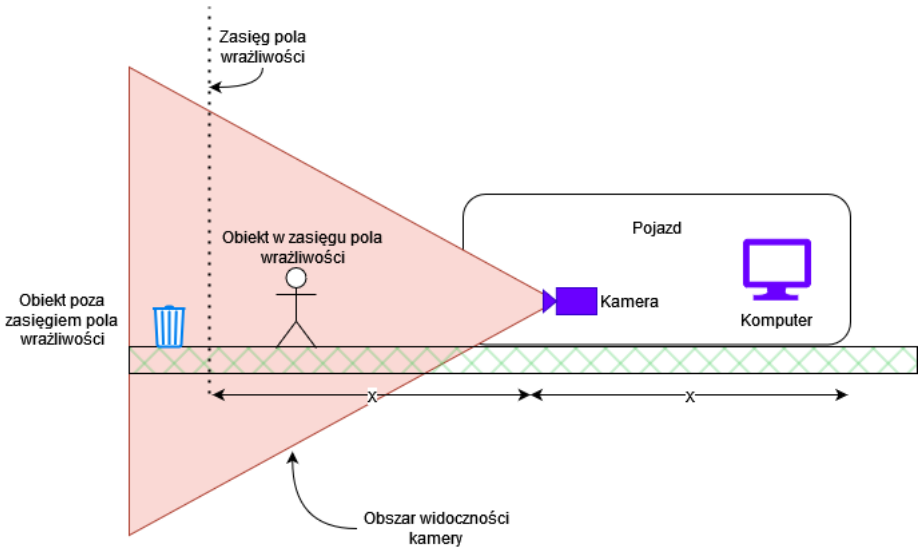
- Fizyczne urządzenie odczytujące obraz zamontowane na tyle pojazdu
- Ułatwia manewrowanie poprzez zapewnienie obrazu z tyłu pojazdu

5.2.2 Komputer pokładowy

- Jest to specjalistyczny komputer wbudowany w pojazd, który odpowiada za analizę danych z kamery zgodnie z zaimplementowanymi algorytmami
- Wyposażony we wszystkie niezbędne interfejsy komunikacyjne z systemem pojazdu (np. magistrala CAN)



Rysunek 10: Schemat architektury wysokopoziomowej



Rysunek 11: Schemat architektury wysokopoziomowej z uwzględnieniem sprzętu

6 Architektura logiczna systemu

Norma ISO/IEC 42010 opisuje architekturę logiczną w taki sposób, że musi ona uwzględniać aspekty strukturalne, behawioralne i implementacyjne.

6.1 Strukturalny widok architektury

6.1.1 Konfigurator

- Odpowiada za zarządzanie konfiguracją systemu
- Umożliwia dostosowanie różnych parametrów działania poszczególnych modułów
- Jest odpowiedzialny za zbieranie danych od użytkownika oraz z działania systemu

6.1.2 Moduły wykonawcze

- To moduły odpowiedzialne za konkretne funkcje systemu
- Każdy moduł skupia się na określonym zadaniu, jak np. analiza obrazu, czy ostrzeganie kierowcy
- Są niezależne i odizolowane od siebie, co ułatwia utrzymanie systemu

6.2 Behawioralny widok architektury

6.2.1 Konfiguracja systemu

- Konfigurator pozwala na zmianę parametrów systemu wedle potrzeb użytkownika
- Umożliwia reakcję na zmienne środowisko i preferencje użytkownika

6.2.2 Współpraca modułów wykonawczych

- Moduły wykonawcze komunikują się wymieniając dane poprzez wspólne interfejsy
- Umożliwia przepływ informacji pomiędzy modułami i koordynację ich działania

6.3 Implementacyjny widok architektury

6.3.1 Modularyzacja

- Architektura jest zorganizowana w sposób modułowy, co ułatwia utrzymanie systemu wraz z rozwojem i jego testowaniem
- Każdy moduł może być rozwijany niezależnie od innych

6.3.2 Przenośność

- Opisany system jest zaprojektowany w taki sposób, że umożliwia łatwe przenoszenie na różne platformy sprzętowe i integrację z różnymi pojazdami

Rysunek 12: Schemat architektury logicznej

7 Dobór technologii

7.1 Wymagania techniczne

- System Ubuntu 22.04

7.2 Wymagania sprzętowe

- Procesor: Intel Core i5 lub wyższy
- RAM: Minimum 8 GB
- Dysk: SSD o pojemności 256 GB
- Karta graficzna: Karta wspierająca OpenGL 3.3 lub nowszy
- Ekran: Monitor o rozdzielczości co najmniej 1920x1080 pikseli (FullHD)
- Predykcje wydajnościowe: Przetwarzanie obrazu w czasie rzeczywistym będzie odbywać się z minimalnym opóźnieniem, co skutkuje płynnym wyświetlaniem

7.3 Kwestie sieciowe i bezpieczeństwa

- Komunikacja pomiędzy modułami odbywa się lokalnie i nie wymaga połączenia z zewnętrzną siecią
- Stosowane są protokoły szyfrowane

7.4 Wymagania dotyczące kamer

- Obsługa różnych formatów obrazu, min. MJPEG, H.264
- Rozdzielczość obrazu: minimum 720p
- System musi być kompatybilny z kamerami o standardowych interfejsach, np. USB

7.5 Język programowania + biblioteki

- Python 3.x
- Flask - do tworzenia interfejsu z użytkownikiem
- YOLOv8 - do przetwarzania obrazu z kamer

7.6 Konteneryzacja

- Zastosowany zostanie Docker w celu zapewnienia gotowego środowiska
- Aplikacja testowa zostanie wystawiona do podglądu na serwerze dostępnym w publicznej sieci

8 Plan prac nad projektem

Tabela 2: Plan prac nad projektem

Temat	Deadline	Przewidziany czas wykonania	Uwagi i rezultat
Stworzenie środowiska	16.04	1 tyg.	Działające środowisko deweloperskie
Implementacja i testowanie wykrywania oraz rozpoznawania obiektów	23.04	1 tyg	Wykorzystanie modelu YOLO v8. Testy skuteczności modelu dla zakładanych obiektów
Przygotowania materiałów do testowania systemu	30.04	1tyg	Nagranie filmów zgodnych z założeniami działania systemu, pozwalających na jego testowanie
Implementacja i testowanie algorytmu wyznaczania odległości od wykrytego obiektu	07.05	1tyg.	Pierwsze podejście do kalibracji mierzonej odległości na podstawie znanych wartości ze zdjęć
Implementacja i testowanie algorytmu obliczającego wymiary i położenie linii pola wrażliwości na ekranie	14.05	1tyg.	Poprawnie działający algorytm
Implementacja interfejsu użytkownika i zapewnienie możliwości konfiguracji systemu	28.05	2tyg	Konfigurator wraz z widokiem z kamery w czasie rzeczywistym
Końcowe testy gotowego systemu	04.06	1tyg	Działająca aplikacja przechodząca przewidziane testy

9 Testy projektu

9.1 Test 1 - Martwe strefy

Obiekt stanowiący zagrożenie (np. rower) znajduje się obok samochodu, poza polem widzenia kamery.

9.1.1 Warunki początkowe

Obiekt jest poza polem widzenia kamery, ale może stanowić zagrożenie dla pojazdu.

9.1.2 Inne zagrożenia

Brak wykrycia obiektu ze względu na jego położenie poza kątem widzenia kamery.

9.1.3 Efekt działania systemu

System nie wykryje obiektu i nie zarejestruje go jako zagrożenie.

9.2 Test 2 - Wykrycie obiektu w polu wrażliwości

Obiekt znajduje się w obszarze widoczności kamery oraz w zasięgu pola wrażliwości systemu.

9.2.1 Warunki początkowe

Obiekt jest widoczny dla kamery i znajduje się w obszarze, który system uznaje za potencjalnie niebezpieczny.

9.2.2 Inne zagrożenia

Brak.

9.2.3 Efekt działania systemu

System wykryje obiekt i zarejestruje go jako zagrożenie, generując odpowiednią informację dla użytkownika.

9.3 Test 3 - Obiekt poza polem wrażliwości

Obiekt znajduje się w obszarze widoczności kamery, ale poza zasięgiem pola wrażliwości systemu.

9.3.1 Warunki początkowe

Obiekt jest widoczny dla kamery, ale znajduje się poza obszarem, który system uznaje za potencjalnie niebezpieczny.

9.3.2 Inne zagrożenia

Brak.

9.3.3 Efekt działania systemu

System nie wykryje obiektu i nie zarejestruje go jako zagrożenie, nie generując informacji dla użytkownika.

9.4 Test 4 - Obiekt widoczny częściowo

Obiekt (np. rower) znajduje się częściowo w obszarze widoczności kamery i polu wrażliwości systemu.

9.4.1 Warunki początkowe

Obiekt jest widoczny tylko częściowo, ale znajduje się w obszarze uznawanym przez system za potencjalnie niebezpieczny.

9.4.2 Inne zagrożenia

Brak.

9.4.3 Efekt działania systemu

System może zarejestrować obiekt jako zagrożenie lub nie, zależnie od skuteczności modelu rozpoznającego obiekty.

9.5 Test 5 - Wykrycie obiektu w całości

Obiekt (np. krzak) znajduje się w całości w obszarze widoczności kamery i polu wrażliwości systemu.

9.5.1 Warunki początkowe

Obiekt jest w całości widoczny dla kamery i znajduje się w obszarze uznawanym przez system za potencjalnie niebezpieczny.

9.5.2 Inne zagrożenia

Brak.

9.5.3 Efekt działania systemu

System wykryje obiekt i zarejestruje go jako zagrożenie, generując odpowiednią informację dla użytkownika.

9.6 Test 6 - Obiekt widoczny w całości, poza polem wrażliwości

Obiekt (np. rower) znajduje się w całości w obszarze widoczności kamery, ale poza polem wrażliwości systemu.

9.6.1 Warunki początkowe

Obiekt jest w całości widoczny dla kamery, ale poza obszarem uznawanym przez system za potencjalnie niebezpieczny.

9.6.2 Inne zagrożenia

Brak.

9.6.3 Efekt działania systemu

System nie wykryje obiektu i nie zarejestruje go jako zagrożenie, nie generując informacji dla użytkownika.

9.7 Źródła materiałów wideo

- Nagrania z kamer samochodów o różnych warunkach pogodowych i oświetleniu
- Zbiory danych takie jak KITTI Vision Benchmark Suite, YouTube-8M
- Inne zbiory danych pochodzące z platform Kaggle oraz GitHub
  np. https://github.com/Valdiolus/Rear_view_camera_dataset

10 Źródła

- https://www.vecteezy.com/free-png
- https://toppng.com
- https://pngtree.com

- https://en.wikipedia.org/wiki/Software_requirements_specification
- http://www.iso-architecture.org/42010/cm/
- https://www.cvlibs.net/datasets/kitti/
- https://research.google.com/youtube8m/

11 Etap II - efekty pracy

11.1 Rozpoznawanie obiektów

Wykorzystywany model rozpoznaje przewidziane przez nas obiekty takie jak: rower, samochód oraz człowiek.

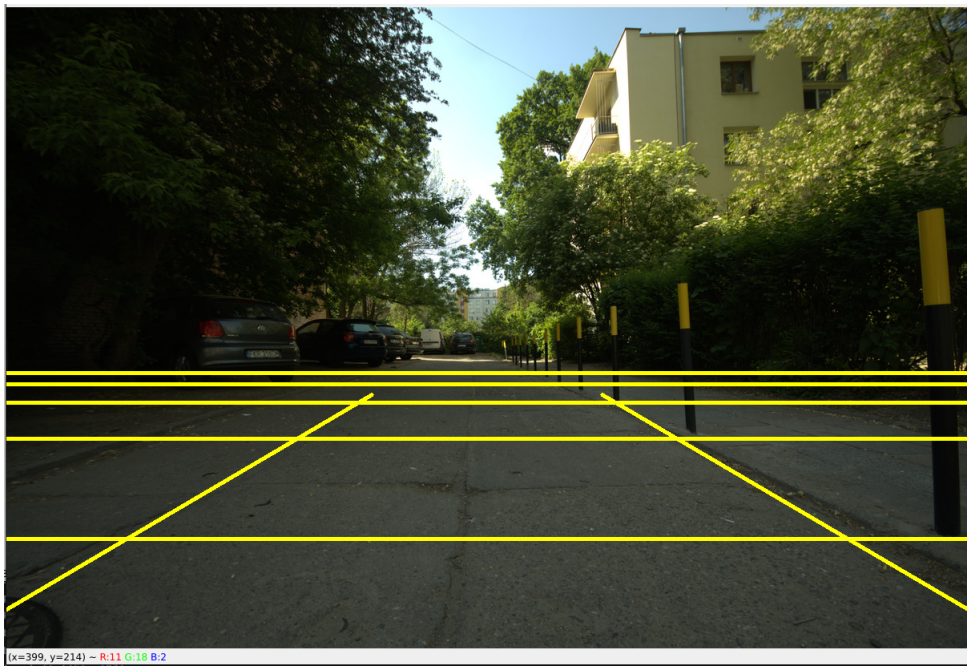
11.2 Wyliczanie odległości

Zaimplementowany algorytm skutecznie wylicza odległość do wykrytych obiektów oraz pozwala na filtrowanie i wyświetlanie tych które znajdują się bliżej niż ustalona odległość. Nie zaimplementowane jest dotychczas jednak sprawdzanie czy obiekty znajdują się w polu wrażliwości, a jedynie czy znajdują się de facto poniżej pewnej prostej na ekranie.

Rysunek 13: Wykrywanie obiektów

11.3 Rysowanie linii ograniczających strefę wrażliwości

Zaimplementowany został algorytm rysowania linii ograniczających strefę wrażliwości.



Rysunek 14: Rysowanie linii

12 Etap III - podsumowanie projektu

12.1 Opis zrealizowanych prac w projekcie

Plan działań nad projektem został ujęty w ramach punktu 8. Pierwszym krokiem było zdefiniowanie założeń, wymagań oraz ograniczeń projektowych. Zostało to zrealizowane w ramach etapu 1 projektu. W kroku tym dokonaliśmy również analizy dostępnych technologii mogących się przydać w projekcie jak i komercyjnych rozwiązań w tej dziedzinie nauki dostępnych na rynku. Powyższa analiza pozwoliła rozpisać plan działania oraz wybrać metodę działania i stos technologiczny. Głównym problemem okazało się być matematyczne wyprowadzenie i udowodnienie metody pomiaru odległości od wykrytego obiektu oraz rysowanie granic strefy wrażliwości wraz z uwzględnieniem perspektywy. Dobór modelu rozpoznawania obiektów ważył się pomiędzy tym czy chcemy sami wyuczyć model czy zastosować model już wyuczony. Zdecydowaliśmy się na drugie rozwiązanie. Następnie zaimplementowana została testowa wersja programu ograniczająca się do wykorzystania modelu YOLO oraz modułu obliczającego odległość od obiektów i rysującego linie w oparciu o informacje o kamerze i dostarczonym zdjęciu. Skuteczność powyższego programu została uznana za zadowalającą w oparciu o wizualną poprawność wyznaczanych odległości od słupków umieszczonych w terenie co 1.5m. Posiadając działający szkielet aplikacji przeszliśmy do realizacji części wizualne wraz z konfiguratorem dla całego systemu. Po implementacji tej części projektu nastąpiła jego konteneryzacja. W międzyczasie przeprowadzone zostały również testy końcowego produktu w oparciu o serie filmów symulujących zdarzenia w których mógłby się znaleźć system. Dokładny opis testów znajduje się w punkcie 12.4.

12.2 Architektura niskopoziomowa wraz z fragmentami kodu programu

Oprogramowanie do rozpoznawania i przetwarzania obrazów zostało zaprojektowane z myślą o efektywności i elastyczności w działaniu. Podzielona jest na dwie główne części. Pierwsza to aplikacja z interfejsem użytkownika napisana we Flasku. Drugi ważny element to moduł akwizycji obrazu wraz z jego przetwarzaniem i analizą rozpoznawania stworzony z wykorzystaniem modelu YOLOv8 oraz OpenCV. Aplikacja domyślnie działa na adresie 127.0.0.1 i porcie 5050.

12.2.1 Strony użytkownika

Po wejściu na wskazany w powyższym punkcie adres ukazuje nam się strona z konfiguratorem. Dwie możliwe drogi wyboru to ścieżka `/webcam` która wyświetla interfejs z widokiem kamery oraz ścieżka `/frontpage`, na której możemy wysłać film za pomocą formularza, a następnie go wyświetlić.

```
@app.route('/webcam', methods=['GET','POST'])
def webcam():
    return render_template('ui.html')


@app.route('/frontpage', methods=['GET','POST'])
def front():
    form = UploadFileForm()
    if form.validate_on_submit():
        file = form.file.data
        file.save(os.path.join(os.path.abspath(os.path.dirname(__file__)), app.config['UPLOAD_FOLDER'],
                              secure_filename(file.filename)))

        session['video_path'] = os.path.join(os.path.abspath(os.path.dirname(__file__)), app.config['UPLOAD_FOLDER'],
                                             secure_filename(file.filename))
    return render_template('new-video.html', form=form)
```

Listing 1: Ścieżki stron użytkownika

12.2.2 Strumień wideo we Flasku

Poniższe ścieżki `/video` oraz `/webapp` służą za odpowiedź serwera ze strumieniem wideo. Główna różnica między nimi znajduje się w listingu 2. w linijce 21 dla funkcji `webapp()`. Kiedy podana ścieżka do pliku jest równa liczbie *0*, to ładowany jest obraz z kamery, zamiast z pliku wideo.
Na listingu 3. znajduje się funkcja odpowiedzialna za generowanie ramek otrzymanych od modułu analizy wideo. Wygenerowane ramki zostają przekształcone do formy odpowiedzi HTTP.

```
1  @app.route('/video')
2  def video():
3      path = session.get('video_path', None)
4      if path is None:
5          return render_template('new-video.html')
6
7      if session.get('config', None) is None:
8          session['config'] = app.config['DEFAULT_CONFIG']
9
10     sys.load_video_capture(path)
11
12     sys.configure(session['config']['sensor_width'], session['config']['sensor_height'], session['config']['focal_length'], session['config']['camera_position_height'], session['config']['box_height'], session['config']['box_width'])
13     return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')
14
15
16 @app.route('/webapp')
17 def webapp():
18     if session.get('config', None) is None:
19         session['config'] = app.config['DEFAULT_CONFIG']
20
21     sys.load_video_capture(0)
22     sys.configure(session['config']['sensor_width'], session['config']['sensor_height'], session['config']['focal_length'], session['config']['camera_position_height'], session['config']['box_height'], session['config']['box_width'])
23     return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')
```

Listing 2: Ścieżki strumienia wideo

```
def generate_frames():
    yolo_output = sys.analyse_video()
    for detection in yolo_output:
        ret, buffer = cv2.imencode('.jpg', detection)
        frame = buffer.tobytes()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

Listing 3: Generowanie ramek

12.2.3 Analiza wideo

Proces analizy wideo po odczytaniu strumienia wideo rozpoczyna się od załadowania modelu przetrenowanego YOLOv8. Na listingu 4. widać to w linijce 2. Cała funkcja wykonuje się w nieskończonej pętli, a w przypadku zakończenia filmu zapętla go (linijki 7-9). Dokonywana jest analiza za pomocą modelu.
W linijce 15 dodany został warunek wyłączenia przetwarzania obrazu na wypadek braku posiadania GPU w komputerze - wynika to z dokładnie tego problemu u jednego z naszych programistów. Finalny obraz zostaje zwrócony do funkcji generującej ramki w formie generatora `yield`, co pozwala na proste zapętlenie programu bez potrzeby dodatkowych warunków obsługujacych pobieranie kolejnych ramek wideo.

```
1  def analyse_video(self):
2      model = YOLO("yolov8n.pt")
3
4      while True:
5          success, img = self.capture.read()
6
7          if not success:
8              self.capture.set(cv2.CAP_PROP_POS_FRAMES, 0)
9              continue
10
11         self.image = img
12
13         self.results = model(img, stream=True)
14
15         if not DEBUG_CPU:
16             self.custom_boxes()
17             # self.draw_test_lines()
18             self.draw_box()
19
20         yield self.image
```

Listing 4: Analiza wideo

12.2.4 Funkcja custom_boxes

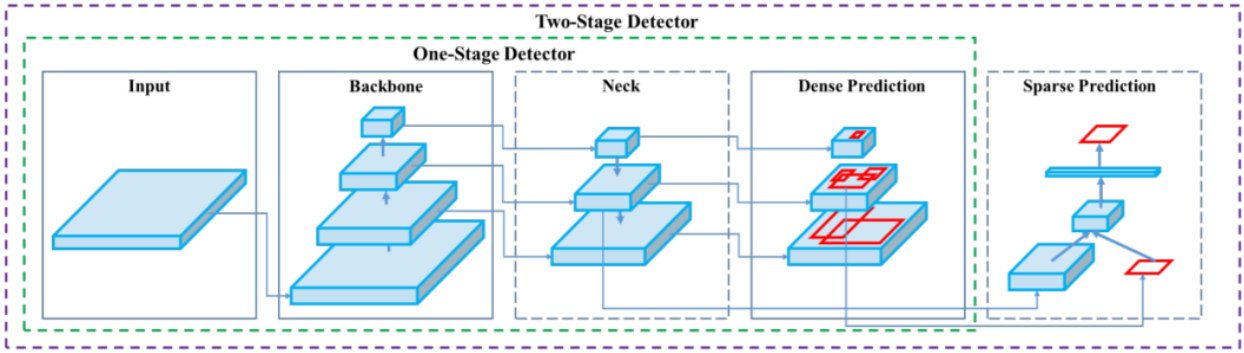
Funkcja przedstawiona na listingu 5. pełni najważniejszą funkcję - rysuje ramki wokół obiektów znajdujących się w strefie zagrożenia.

```
def custom_boxes(self):
    for r in self.results:
        boxes = r.boxes
        for box in boxes:
            if self.dangerZone.is_object_inside(box):
                x, y, x2, y2 = box.xyxy[0]
                x, y, x2, y2 = int(x), int(y), int(x2), int(y2)
                cv2.rectangle(self.image, (x, y), (x2, y2), (255, 128, 255), 3)
```

Listing 5: Funkcja custom_boxes

12.2.5 Funkcja draw_box

Funkcja ta rysuje granice strefy wrażliwości od dolnych rogów obrazu do wyznaczonej odległości.



Rysunek 15: Architektura YOLO

```
def draw_box(self):
    # we need four points
    # top_left, top_right
    # bottom_left, bottom_right

    top_left, top_right, bottom_left, bottom_right = self.dangerZone.get_zone_borders()

    cv2.line(self.image, top_left, bottom_left, (0, 255, 255), 3)
    cv2.line(self.image, top_right, bottom_right, (0, 255, 255), 3)
```

Listing 6: Funkcja draw_box

12.2.6 Wyjaśnienie pozostałej części systemu

Powyżej pokazany kod należy do plików `webapp.py` i `System.py`. Ten ostatni zawiera w sobie sporo funkcji pomocniczych realizujących obliczenia matematyczne opisane w poprzednich etapach. W celu modularności kodu podzieliliśmy jego funkcje na dodatkowe klasy `Camera` oraz `DangerZoneBox`. Ich zadaniem jest obsłużyć część obliczeń w formie prostej w użyciu klasy nie przejmując się pozostałą logiką systemu.

12.3 Opis wykorzystywanych modeli oraz metod przetwarzania obrazów

W naszym projekcie wykorzystaliśmy dwa główne narzędzia do przetwarzania i rozpoznawania obrazów: bibliotekę YOLOv8 oraz OpenCV. YOLOv8 (You Only Look Once, wersja 8) jest zaawansowanym modelem głębokiego uczenia, który stosuje techniki konwolucyjnych sieci neuronowych (CNN) do detekcji obiektów w czasie rzeczywistym. YOLO jest znany z wyjątkowej szybkości i dokładności, ponieważ przetwarza cały obraz jako pojedynczą jednostkę, co pozwala na jednoczesne przewidywanie klasy i lokalizacji obiektów.

Architektura składa się ze szkieletu, szyi i głowy. Szkielet jest wstępnie wytrenowaną konwolucyjną siecią neuronową (CNN), która wyodrębnia mapy cech niskiego, średniego i wysokiego poziomu z obrazu wejściowego. Szyja łączy te mapy funkcji za pomocą bloków agregacji ścieżek, takich jak Feature Pyramid Network (FPN). Przekazuje je do głowy, klasyfikując obiekty i przewidując obwiednie. Głowa może składać się z jednoetapowych lub gęstych modeli predykcji, takich jak właśnie tu użyte YOLO lub Single-shot Detector (SSD). Alternatywnie, może zawierać dwuetapowe lub rzadkie algorytmy predykcji, takie jak seria R-CNN. OpenCV (Open Source Computer Vision Library) to wszechstronna biblioteka programistyczna, która oferuje szeroki zakres funkcji do przetwarzania obrazów i wizji komputerowej. OpenCV wspiera różnorodne operacje, takie jak konwersje kolorów, wykrywanie krawędzi, transformacje geometryczne oraz wiele innych.

W projekcie OpenCV jest wykorzystane do załadowania filmu wideo lub kamery w celu poddania go analizie przez model YOLO i dalszej obróbce obrazu. Model YOLO przetwarza otrzymaną klatkę wideo tworząc listę wykrytych obiektów różnego rodzaju (takie jak człowiek, rower czy słupek). Nasze oprogramowanie

wykorzystuje te wyniki i rysuje ramki wokół obiektów, które znajdują się w strefie zagrożenia. Dzięki temu pozbyliśmy się niepotrzebnego szumu w postaci ramek wokół wielu obiektów widocznych w oddali obrazu.

Jako zestaw uczący wykorzystaliśmy COCO (Common Objects in Context) dostępny w przetrenowanym modelu `yolov8n.pt`.

Podjęliśmy tę decyzję ze względu na łatwość użycia - trenowaliśmy testowo własny model podczas pracy nad oprogramowaniem, jednak dużo lepiej działa model gotowy do pobrania z internetu. W naszym przypadku wystarczy model *n*, ponieważ większe modele wymagają większej mocy obliczeniowej, a sama ich wyższa dokładność nie jest zalecana dla systemów czasu rzeczywistego. Wybrane biblioteki spełniają dobrze swoje zadanie, choć były z nimi na początku trudności. Jednym z napotkanych problemów było uruchomienie kamery, które nie działało ze względu na błędy w najnowszych wersjach `torch` oraz `opencv-python`. Po obniżeniu wersji problemy zanikły. Innym błędem było uruchomienie YOLO na komputerze bez GPU, o czym nie ma nic w dokumentacji (choć samo YOLO posiada możliwość uruchomienia wersji dla CPU z użyciem Dockera). W tym przypadku rozwiązaniem jest testowanie na komputerze, które posiada dedykowane GPU.

12.4 Wyniki testów i eksperymentów

Testy systemu wykonane zostały na podstawie nagrań symulujących rzeczywiste sytuacje w których mógłby się znaleźć system. Do wykonania nagrań wykorzystany został aparat Sony Alpha 6100 z obiektywem Sigma 10-20mm, a kamera umieszczona została na wysokości około 1m. Ze względu na brak możliwości przyczepienia kamery do samochodu oraz brak gimbala testy ograniczają się do statycznie umieszczonej kamery i ruchomych obiektów w jej polu widzenia.

12.4.1 Test 1 - Martwe strefy

Test martwych stref jest możliwy do przeprowadzenia z trzecioosobową perspektywą na całe zdarzenie, ponieważ z definicji martwej strefy, obiekty się w niej znajdujące nie zostaną uchwycone na nagraniu z kamery. Matematycznie udowodnione zostało, że czym mniejszy kąt widzenia kamery (większa ogniskowa obiektywu oraz czym mniejszy sensor kamery) tym większe te strefy będą i tym większe ryzyko, że znajdzie się w nich obiekt niewidoczny na wyświetlaczu systemu. Trzecioosobowa perspektywa widoku sytuacji pozwala dowieść tego, że ten obiekt faktycznie się tam znajduje co wynika również z założeń tego testu i udowadnia brak potrzeby go przeprowadzania.

Efekt działania systemu:
System nie wykrył obiektu i nie zarejestrował go jako zagrożenie.

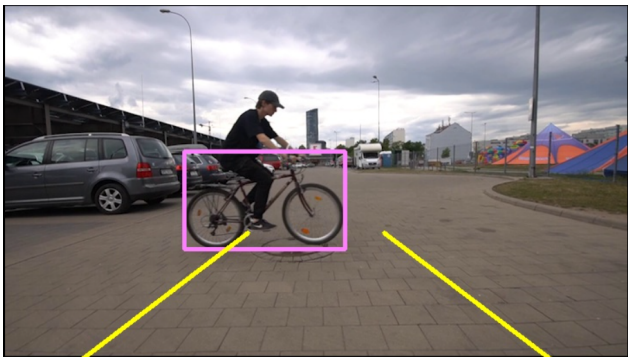
Rysunek 16: Widok z kamery

Rysunek 17: Widok martwej strefy

12.4.2 Test 2 - Wykrycie obiektu w polu wrażliwości

Test został przeprowadzony z wykorzystaniem nagrania na którym człowiek na rowerze wjeżdża oraz wyjeżdża ze strefy wrażliwości. Gdy znajduje się on w strefie otoczony zostaje kwadratem w celu zaznaczenia jego wykrycia. Gdy opuści strefę, otaczający go kwadrat znika co dowodzi poprawnego działania systemu w tym kontekście.

Efekt działania systemu:
System wykrył obiekt i zarejestrował go jako zagrożenie, generując odpowiednią informację dla użytkownika.

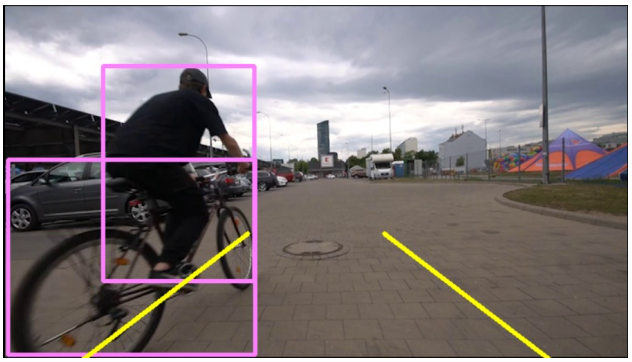


Rysunek 18: Obiekt wykryty w całości

12.4.3 Test 3 - Wykrycie obiektu poza polem wrażliwości

Test przeprowadzony został na nagraniu, gdzie początkowo rowerzysta wraz z rowerem znajdują się na granicy pola wrażliwości, żeby następnie wyjechać poza pole wrażliwości i tak kontynuować trasę. Zgodnie z oczekiwaniami kwadrat wokół obiektu znika gdy obiekt opuści strefę wrażliwości.

Efekt działania systemu:
Obiekt jest widoczny dla kamery, ale znajduje się poza obszarem, który system uznaje za potencjalnie niebezpieczny.



Rysunek 19: Obiekt na skraju pola wrażliwości

Rysunek 20: Obiekt poza polem wrażliwości

Rysunek 21: Obiekt poza polem wrażliwości - naprzeciwko

12.4.4 Test 4 - Obiekt widoczny częściowo

Test został przeprowadzony na nagraniu gdzie rower stoi blisko kamery i tylko fragment jest widoczny. Model w większości przypadków radzi sobie z wykryciem i rozpoznaniem obiektu. Dokładne określenie jaka część obiektu musi być widoczna wymagałoby jednak osobnych badań wraz z uwzględnieniem wielu dodatkowych czynników takich jak oświetlenie czy kontrast pomiędzy kolorem roweru a jego tłem.

Efekt działania systemu:
System może zarejestrować obiekt jako zagrożenie lub nie, zależnie od skuteczności modelu rozpoznającego obiekty. W naszym przypadku system zarejestrował obiekt jako zagrożenie.

Rysunek 22: Obiekt widoczny częściowo

12.4.5 Pozostałe testy

Pozostałe testy to wykrycie obiektu widocznego w całości oraz wykrycie obiektu widocznego, ale poza obszarem wrażliwości. Testy te zostały przeprowadzone na testowej wersji programu, ponieważ końcowa wersja programu nie prezentuje użytkownikowi takich informacji, gdyż są one niepotrzebne i kłócą się z założeniami systemu.

12.5 Wnioski z projektu

12.5.1 Skuteczność rozwiązania

Zaimplementowanie rozwiązanie problemu jest skuteczne, lecz niepraktyczne do zastosowania w czasie rzeczywistym ze względu na czas wymagany na przetworzenie każdej klatki. W celu ulepszenia rozwiązania należałoby zredukować liczbę klatek w nagraniu oraz zastosowanie lżejszej, szybszej wersji modelu. Kolejnym problemem są bardzo rygorystyczne zasady co do umieszczenia kamery oraz duża wiedza o sprzecie wymagana do konfiguracji programu. Warto zaznaczyć jednak, że z matematycznego punktu widzenia działanie systemu jest poprawne i zapewnia wymagane bezpieczeństwo poprzez poprawny pomiar w zakresie od 0 do 7.5m.

12.5.2 Poprawność dobranych technologii

Zastosowany model jest zbyt powolny do przetwarzania testowanych nagrań. W zakresie rozpoznawania i wykrywania obiektów model oferuje znacznie wybiegające nad wymagania i założenia projektu możliwości. Wytrenowanie własnego modelu do zadania rozpoznawania wybranych obiektów przyspieszyłoby zadanie, lecz kierunek rozwoju naszego projektu bardziej skłonił nas do implementacji skutecznej metody obliczania odległości od wykrytych obiektów w celu oceny realnego zagrożenia.

12.5.3 Ocena obsługi sytemu

Zaimplemetowany system działa intuicyjnie, pozwala na przetworzenie zarówno nagrania wczytanego z pamięci urządzenia na którym jest uruchomiony jak i wykorzystanie jego kamery. Na przetwarzane nagranie, klatka po klatce, nakładane są informację takie jak linie wyznaczające strefę niebezpieczeństwa oraz prostokąty (bounding boxes) wokół wykrytych obiektów znajdujących się w niej. Powyższe informacje przedstawione są w sposób intuicyjny i jednoznaczny.

12.5.4 Ocena podejścia

W naszej opinii zastosowane podejście wydaje się być jedyną słuszną drogą do otrzymania uniwersalnego oprogramowania do kamery cofania biorą za główne wymaganie możliwość dostosowania obszaru strefy wrażliwości oraz możliwość wykorzystania różnych kamer. Kwestia wykrywania i rozpoznawania obiektów, która wpływa na wydajność systemu mogłaby jednak być rozwiązana w inny lepszy sposób jak to zostało poruszone w powyższych punktach. Ilość sztywnych zasad, których musi trzymać się użytkownik systemu pokazuje jednak, że nie jest to dobrze podejście do tematu kamer cofania. Inne pomysły rozwiązania problemu wybiegające poza ramy projektu: Naturalnym wnioskiem po realizacji projektu jest krytyczne spojrzenie na zastosowane techniki, ale również przedstawienie alternatyw. Naszym zdaniem wykorzystanie czujników cofania rozwiązałoby wiele z zaistniałych problemów przy utrzymaniu uniwersalności całego systemu. Dowodem potwierdzającym tą tezę jest popularność i łatwa dostępność takich systemów (ocena na podstawie sklepów internetowych Amazon.com oraz Allegro.pl). Innym podejściem bez wykorzystania czujników odległości jest przygotowanie sprecyzowanego systemu o konkretnej kamerze. Wtedy kwestia pomiaru odległości mogłaby zostać znacząco uproszczona, a nawet zakodowana jako jedna stała tablica bez potrzeby wykonywania za każdym razem obliczeń. Warto jednak zaznaczyć, że takie rozwiązanie nie sprawdzi się w przypadku gdy teren na którym znajduje się pojazd będzie nierówny. Potencjalnie przy jednolicie nachylonej powierzchni dałoby się to rozwiązać przy wykorzystaniu żyroskopu lub gimbala, lecz pierwsze z tych rozwiązań wiązałoby się ze znaczącą komplikacją obliczeń (tu zastosowanie sztywnej tabeli mogłoby być problematyczne), a drugie z rozwiązań wydaje się być zbyt skomplikowane, mało praktyczne i niosące ze sobą spore ryzyko

uszkodzeń mechanicznych. Reasumując powyższe przemyślenia zastosowanie czujników pomiaru odległości jest najlepszym podejściem.

12.6 Dokumentacja użytkownika

12.6.1 Wymagania

- Docker Compose w najnowszej wersji
- Dedykowana karta graficzna lub laptop z procesorem min. M1

12.6.2 Uruchomienie systemu

Po wejściu do katalogu projektu należy uruchomić Dockera za pomocą polecenia:
`docker compose up`
Po udanym uruchomieniu kontenera z systemem naszym oczom ukaże się poniższy widok:

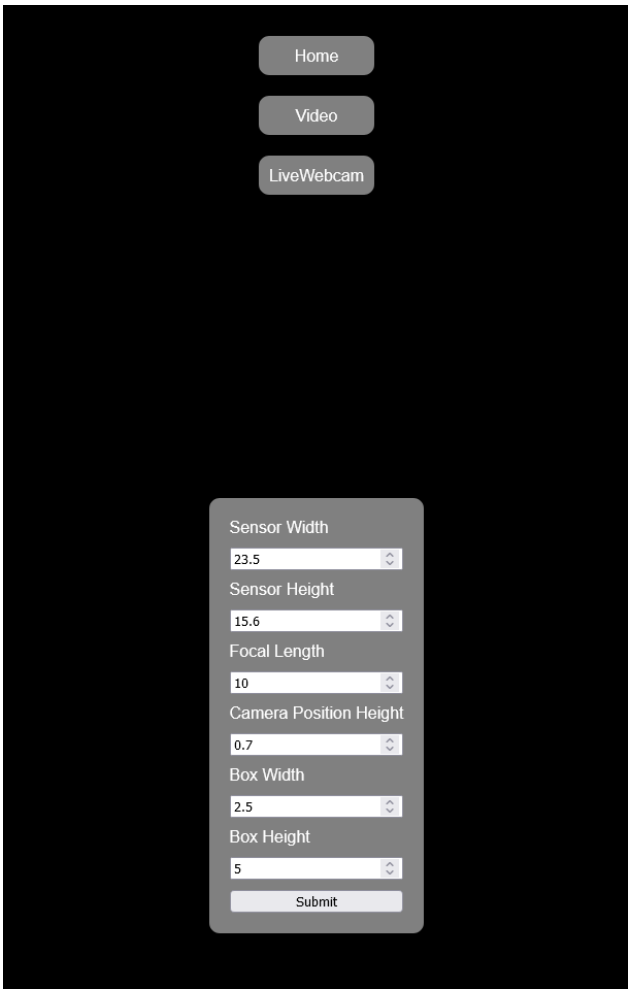
```
 * Serving Flask app 'webapp'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5050
 * Running on http://192.168.28.60:5050
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 948-150-360
```

Rysunek 23: System po uruchomieniu

Oznacza to, że w celu wejścia do systemu należy uruchomić stronę pod adresem `127.0.0.1:5050` bądź innym podanym przez system.

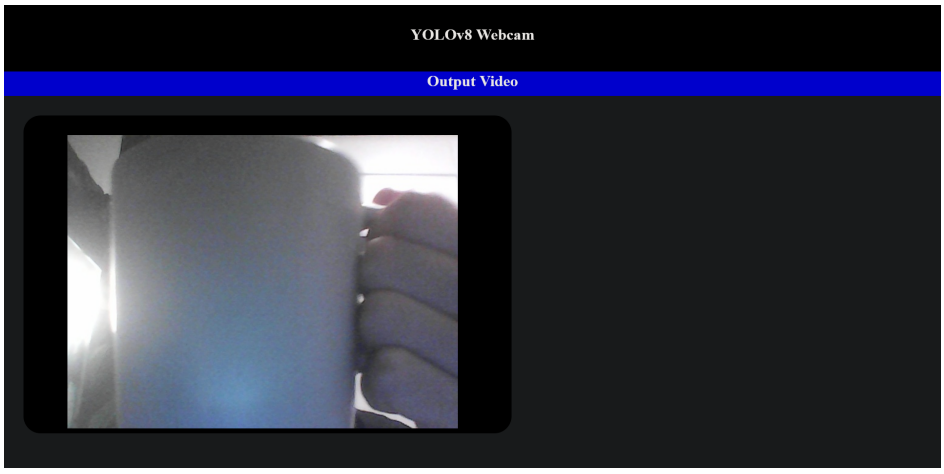
12.6.3 Interfejs użytkownika

Po wejściu do systemu wyświetli nam się główny widok aplikacji. Z niego możemy skonfigurować system przetwarzania lub przejść do odtwarzania wideo, bądź kamery na żywo (LiveWebcam).



Rysunek 24: Strona główna

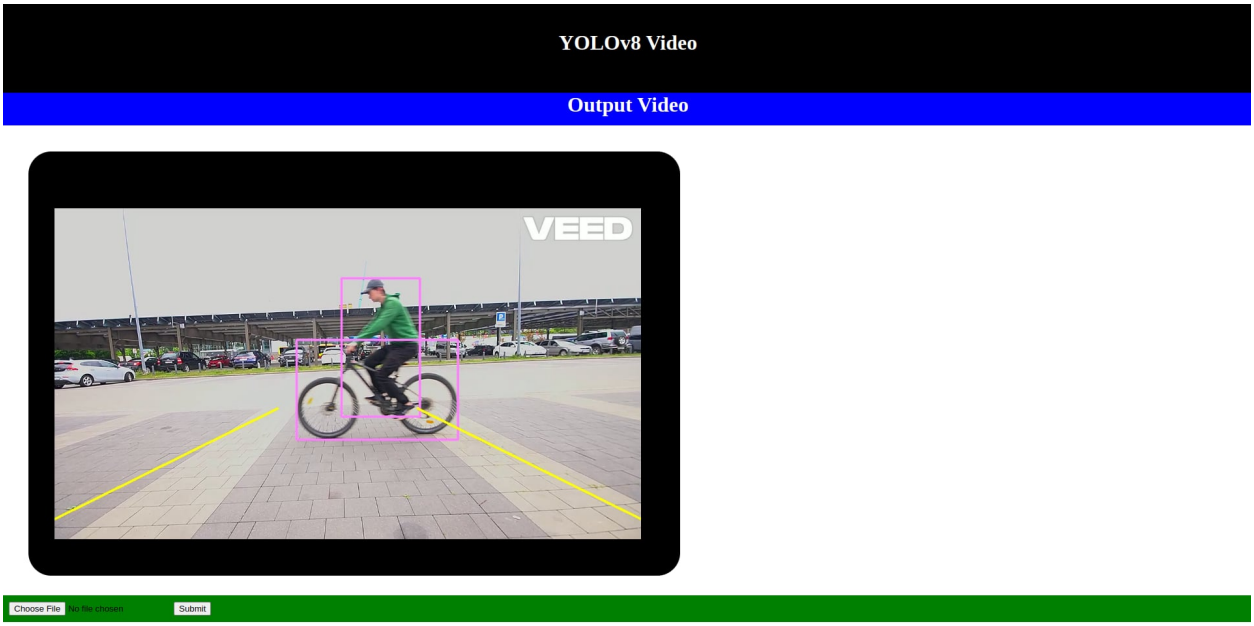
Interfejs kamery na żywo prezentuje się następująco. Na załączonym obrazku system został uruchomiony bez funkcji YOLO, dlatego też nie są widoczne ramki obiektów w teoretycznym polu kamery.



Rysunek 25: Kamera na żywo

Aby uruchomić test wideo należy załączyć plik wideo w formularzu na zielonym pasku, a następnie potwierdzić. Użytkownik zostanie przekierowany do tej samej strony po przesłaniu filmu na serwer.

Poniżej jest widoczny test wideo, na którym zostały narysowane ramki dla obiektu w strefie zagrożenia (człowiek oraz rower).



Rysunek 26: Test wideo

Przedstawiony powyżej interfejs jest prosty i ukierunkowany na natychmiastowe zrozumienie jego działania. Poprawnie uruchomiony program przetwarza obraz z zadanego źródła oraz ustalonej konfiguracji i wyświetla obiekty w strefie zagrożenia. W dodatku wideo zostaje zapętlone, dzięki czemu możemy oglądać daną sytuację wiele razy bez potrzeby włączenia ponownie filmu. Jeśli użytkownik nie posiada kamery - to oczywiście nie zobaczy jej obrazu w aplikacji.