

# Spis treści

|   |           |
|---|-----------|
| <b>1 Wstęp</b>  | <b>3</b>  |
| 1.1 Cel i zakres pracy . . . . .  | 3         |
| 1.2 Detekcja obiektów a szacowanie odległości . . . . .                         | 3         |
| 1.3 Operacje na obrazach . . . . .  | 4         |
| 1.3.1 Konwersja do skali szarości . . . . .                                     | 4         |
| 1.3.2 Filtr Gaussa . . . . .  | 4         |
| 1.3.3 Detektor Canny . . . . .  | 4         |
| 1.3.4 Wyznaczenie obszaru zainteresowań . . . . .                               | 5         |
| 1.3.5 Transformacja Hougha . . . . .  | 5         |
| 1.4 Wskaźnik F1 . . . . .   | 6         |
| <b>2 Przegląd literatury</b>  | <b>6</b>  |
| 2.1 Metody szacowania odległości do obiektów . . . . .                          | 6         |
| 2.1.1 Dist-YOLO . . . . .   | 8         |
| 2.1.2 Wykorzystanie szacowania głębi i klasyfikatorów . . . . .                 | 10        |
| 2.2 Metody utrzymywania pieszego w pasie ruchu . . . . .                        | 11        |
| <b>3 Zadanie szacowania odległości do obiektów</b>                              | <b>12</b> |
| 3.1 Dane . . . . .  | 13        |
| 3.2 YOLOv8 . . . . .  | 14        |
| 3.2.1 Trenowanie modelu YOLOv8 . . . . .  | 16        |
| 3.2.2 Model 1 - 100 epok . . . . .  | 18        |
| 3.2.3 Model 2 - 150 epok . . . . .  | 19        |
| 3.3 MiDaS . . . . .   | 20        |
| 3.4 Monodepth2 . . . . .  | 22        |
| 3.5 Implementacja detekcji obiektów i szacowania odległości w języku Python . . | 23        |
| 3.6 Testowanie metody . . . . .   | 25        |
| 3.6.1 Porównanie pod względem jakości rozwiązania . . . . .                     | 25        |
| 3.6.2 Porównanie pod względem czasu działania . . . . .                         | 28        |

|          |   |           |
|----------|---|-----------|
| 3.6.3    | Podsumowanie testów                                 | 28        |
| <b>4</b> | <b>Zadanie utrzymywania użytkownika na chodniku</b> | <b>28</b> |
| 4.1      | Implementacja metody za pomocą OpenCV               | 31        |
| 4.2      | Testowanie metody                                   | 33        |
| 4.2.1    | Wpływ konwersji do skali szarości                   | 34        |
| 4.2.2    | Wpływ parametrów filtra Gaussa                      | 35        |
| 4.2.3    | Wpływ parametrów detektora Canny                    | 37        |
| 4.2.4    | Wpływ kształtu obszaru zainteresowań                | 40        |
| 4.2.5    | Wpływ parametrów transformacji Hougha               | 43        |
| <b>5</b> | <b>Aplikacja mobilna</b>                            | <b>46</b> |
| 5.1      | Implementacja metod                                 | 46        |
| 5.1.1    | Detekcja obiektów i szacowanie odległości           | 46        |
| 5.1.2    | Utrzymywanie użytkownika na chodniku                | 46        |
| 5.2      | Testowanie aplikacji                                | 47        |
| <b>6</b> | <b>Podsumowanie</b>                                 | <b>50</b> |
| <b>7</b> | <b>Podziękowania</b>                                | <b>51</b> |

# 1 Wstęp

## 1.1 Cel i zakres pracy

Celem pracy będzie zbadanie metod sztucznej inteligencji w zadaniu prowadzenia osoby z niepełnosprawnością wzroku w warunkach poruszania się w mieście. Praca obejmuje stworzenie prototypu aplikacji mobilnej, na której będą prowadzone eksperymenty. Funkcjonalności aplikacji będą obejmować: trzymanie się chodnika w trakcie przemieszczania się, wykrywanie przeszkód, wyliczanie odległości do przeszkody, lokalizacja najbliższego przejścia dla pieszych, naprowadzanie użytkownika na przejście dla pieszych. Należy więc zaimplementować i przetestować metody detekcji obiektów, szacowania odległości do wykrytych obiektów, wskazywanie położenia obiektów.

## 1.2 Detekcja obiektów a szacowanie odległości

Detekcja obiektów (ang. *object detection*) polega na wykonaniu dwóch działań: znalezieniu obiektu na obrazie i jednoczesnym przyporządkowaniu go do określonej klasy. Obiekt oznacza się ramką ograniczającą (ang. *bounding box*), zazwyczaj w kształcie prostokąta. Obecnie do detekcji obiektów na obrazach stosuje się powszechnie głębokie sieci neuronowe. W tym przypadku należy przed użyciem wytrenować model sieci neuronowej na zbiorze danych, który zawiera setki lub tysiące obrazów. Dane dzieli się na treningowe i testowe, dzięki czemu można uzyskać wiarygodne wskaźniki jakości takiego modelu. Po wytrenowaniu, ewaluacji, i ewentualnych poprawkach model jest gotowy do zastosowania.

Jednym z celów aplikacji mobilnej, która będzie zaimplementowana w ramach tej pracy, jest wykrywanie przeszkód mogących zagrozić niewidomemu pieszemu (hydranty, słupy, lampy, rowery, inni piesi, itp.). Jednakże sama detekcja nie wystarczy, potrzebna jest jeszcze informacja o odległości. Zdecydowano się sprawdzić dwa podejścia. W pierwszym z nich wbudowuje się szacowanie odległości w system detekcji, zatem to samo narzędzie wykrywa obiekt i jednocześnie aproksymuje odległość. Przykładem takiego rozwiązania jest Dist-YOLO, które będzie bardziej szczegółowo omówione kolejnym rozdziale. Drugim podejściem jest wykrycie obiektu, a następnie oszacowanie odległości inną metodą.

## 1.3 Operacje na obrazach

W tym podrozdziale omówiono operacje na obrazach wykorzystane w tej pracy.

### 1.3.1 Konwersja do skali szarości

Konwersja do skali szarości jest jedną z najbardziej podstawowych operacji na obrazach. Zamienia obraz kolorowy (najczęściej w formacie RGB lub BGR) do skali szarości. Oznacza to zredukowanie wymiaru obrazu o jeden wymiar. Mając obraz kolorowy, w którym każdy piksel jest zapisany jako trzy liczby  $(r, g, b)$ , wyznacza się średnią wartość każdego piksela i w ten sposób otrzymuje się obraz w skali szarości.

### 1.3.2 Filtr Gaussa

Filtr Gaussa jest filtrem dolnoprzepustowym, który korzysta z dwuwymiarowej funkcji Gaussa [1][2]:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \quad (1)$$

gdzie  $\sigma$  - odchylenie standardowe. Liczby naturalne  $x$  i  $y$  definiują rozmiar jądra filtru. Dla przykładu, jeżeli  $x = y = n \in \mathbf{N}$ , to wartości funkcji Gaussa oblicza się dla  $-n \leq x \leq n$  oraz  $-n \leq y \leq n$ . W ten sposób otrzymuje się maskę filtru. Następnie wykonuje się splot obrazu z maską filtru, otrzymując obraz rozmyty.

### 1.3.3 Detektor Canny

Detektor Canny jest popularnym algorytmem wykrywania krawędzi. Obraz wygładzony (rozmyty) za pomocą filtra Gaussa przetwarza się najpierw filtrem Sobela, którego maski (dla kierunków poziomego i pionowego) można zapisać następująco [3]:

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & -2 \\ -1 & 0 & 1 \end{bmatrix}, K_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \quad (2)$$

Obraz poddaje się operacji splotu z obiema maskami, żeby otrzymać pierwszą pochodną  $G_x$  (horyzontalną) i  $G_y$  (wertykalną). Stąd można obliczyć gradient dla każdego piksela [4]:

$$\text{EdgeGradient}(G) = \tg^{-1} \frac{G_y}{G_x}. \quad (3)$$

Następnie wykorzystuje się niemaksymalne tłumienie, aby uzyskać obraz z cienkimi krawędziami. Po zastosowaniu progowania histerezowego otrzymuje się obraz z wyraźnymi krawędziami. Progowanie usuwa krawędzie o zbyt dużym lub zbyt małym gradiencie [4].

#### 1.3.4 Wyznaczenie obszaru zainteresowań

Wyznaczenie obszaru zainteresowań składa się z trzech kroków:

1. Stworzenie czarnego obrazu o rozmiarze obrazu wejściowego.
2. Narysowanie na czarnym obrazie wielokąta wynaznaczającego obszar zainteresowań - utworzenie maski.
3. Wykonanie operacji sumy bitowej obrazu wejściowego z przygotowaną maską.

#### 1.3.5 Transformacja Hougha

Działanie transformacji Hougha jest szeroko opisane w literaturze [5][6][7]. Algorytm można zapisać następująco [5]:

1. Zbudowanie przestrzeni parametrów z odpowiednim poziomem kwantyzacji dla współczynnika kierunkowego  $m$  i wyrazu wolnego  $c$ .
2. Utworzenie tablicy przechowującej pary parametrów -  $A(m, c)$ .
3. Ustawienie  $\forall_{(m,c)} A(m, c) = 0$ .
4. Wczytanie obrazu z krawędziami wykrytymi za pomocą detektora Canny. Obraz traktuje się jako zbiór pikseli  $(x, y)$ .
5.  $\forall (x_i, y_j) \wedge \forall (m_k, c_l)$  zweryfikowanie równania  $c_l = -x_i m_k + y_j$ . Następnie wykonanie inkrementacji  $A(m_k, c_l)$ .
6. Znalezienie maksimów lokalnych  $A(m, c)$ , które odpowiadają liniom prostym w przestrzeni parametrów.

Pewne rozszerzenia transformacji Hougha umożliwiają też wykrywanie innych kształtów, jak krzywe i okręgi [6]. W tej pracy wykorzystano probabilistyczną transformację Hougha, która minimalizuje ilość pikseli używanych do głosowania, jednocześnie zachowując dokładność standardowej transformacji Hougha [8].

## 1.4 Wskaźnik F1

Wskaźnik F1 jest metryką służącą do oceny jakości sieci neuronowych. Definiuje się go jako średnią harmoniczną precyzji (ang. *precision* - stąd oznaczenie  $p$ ) i czułości (ang. *recall* - stąd oznaczenie  $r$ ) [9]. Precyzję i czułość wyraża się wzorami:

$$p = \frac{t_p}{t_p + f_p}, \quad (4)$$

$$r = \frac{t_p}{t_p + f_n}, \quad (5)$$

gdzie:

- $t_p$  — *true positives* — obiekty wykryte w sposób prawidłowy,
- $f_p$  — *false positive* — obiekty wykryte nieprawidłowo, model wykrywa obiekt, mimo że go nie ma,
- $f_n$  — *false negatives* — obiekty, które nie zostały wykryte, mimo że są na obrazie.

Wskaźnik F1 wyraża się zatem wzorem:

$$F_1 = 2 \frac{pr}{p + r}. \quad (6)$$

## 2 Przegląd literatury

### 2.1 Metody szacowania odległości do obiektów

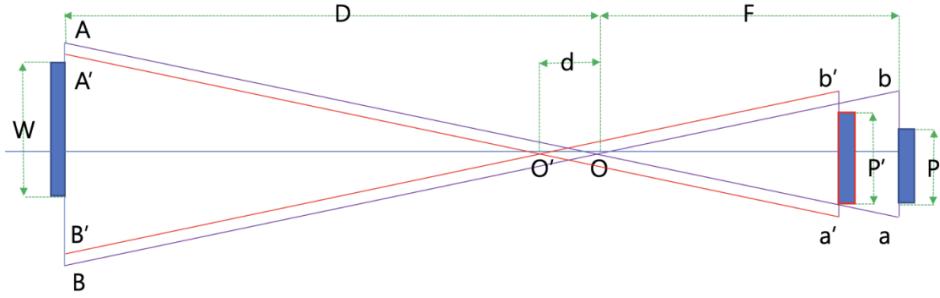
W literaturze i Internecie można znaleźć wiele rozwiązań problemu szacowania odległości. Za najprostsze w implementacji można uznać algorytmy oparte o zależności geometryczne. Jedna z takich zależności jest opisana w artykule [10]. Mierząc wysokość obiektu w kolejnych

krokach oraz wiedząc, o ile przemieściła się kamera, można wyprowadzić wzór (na podstawie podobieństwa trójkątów):

$$D = d \frac{P'}{P - P'}, \quad (7)$$

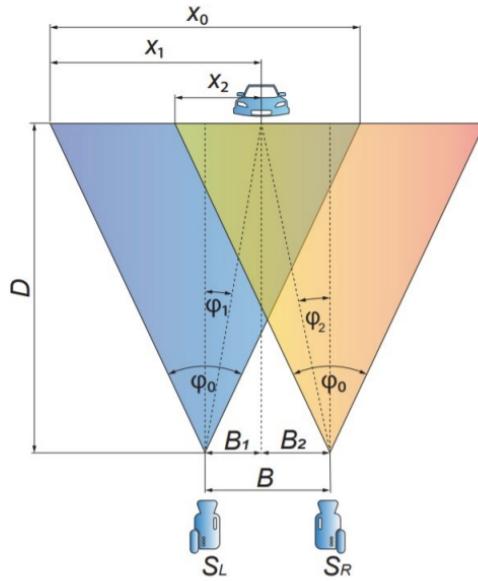
gdzie:

- $D$  - oszacowana odległość,
- $d$  - długość "kroku" kamery,
- $P$  - wysokość obiektu na ekranie na początku,
- $P'$  - wysokość obiektu na ekranie kamery po przesunięciu.

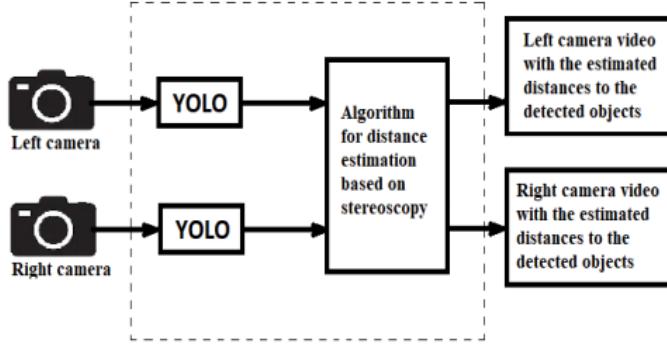


Rysunek 1: Relacja między obrazem w kamerze i obiektem

Dużą wadą tej metody jest to, że trzeba dokładnie znać przesunięcie kamery, co jest trudne do mierzenia w rzeczywistych warunkach. Inna metoda, również bazująca na podobieństwie trójkątów, została opisana w artykule [11]. Również w tym przypadku do oszacowania odległości potrzebna jest znajomość początkowej odległości od obiektu oraz ustalona szerokość obiektu. Oznacza to, że możliwości zastosowania takiej metody w aplikacji mobilnej dla osób niewidomych byłyby mocno ograniczone. Interesującym podejściem może być zastosowanie dwóch kamer. Przykładowo, obrazy z dwóch kamer (ustawionych równolegle do siebie) wykrywają obiekty (np. za pomocą YOLO), a następnie odległość jest szacowana metodą geometryczną, porównując położenie obiektów w dwóch kamerach [12]. Na rysunkach 2, 3 przedstawiono sposób działania takiego systemu.



Rysunek 2: Relacja geometryczna kamer i obiektu [12]



Rysunek 3: Schemat działania systemu z dwoma kamerami [12]

### 2.1.1 Dist-YOLO

Dist-YOLO jest narzędziem do wykonywania zadania detekcji i szacowania odległości od obiektów. Jest siecią neuronową opartą na architekturze YOLOv3, która rozszerza funkcjonalności YOLO o szacowanie odległości [13]. Sieć uczy się rozpoznawać odległość na podstawie etykiet w zbiorze treningowym. Funkcja straty  $l$  została rozszerzona o człon  $l_5$ , który

odpowiada za uczenie się odległości [13]:

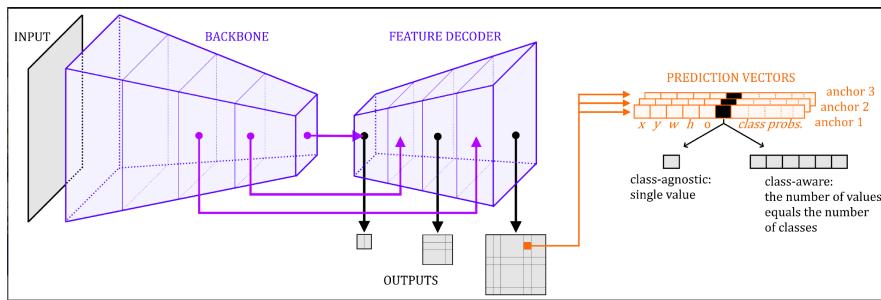
$$l = \sum_{i=0}^{G^W G^h} \sum_{j=0}^{n^a} q_{i,j} [l_1(i,j) + l_2(i,j) + l_3(i,j) + l_5(i,j)] + l_4(i,j), \quad (8)$$

$$l_5(i, j) = \omega \sum_{k=0}^c C_{i,j,k} (\hat{d}_{i,j,k} - d_{i,j})^2, \quad (9)$$

gdzie:

- $\omega$  - współczynnik ważności danej funkcji (powoduje to, że model skupia się na minimizacji danej funkcji straty bardziej lub mniej),
  - $\hat{d}, d$  - odległość oszacowana oraz znana,
  - $C_{i,j,k}$  - prawdopodobieństwo k-tej klasy w i-tej komórce. Literą  $j$  oznaczono indeks prototypowej ramki ograniczającej (ang. *anchor box*).

Na rysunku 4 przedstawiono schemat architektury Dist-YOLO:

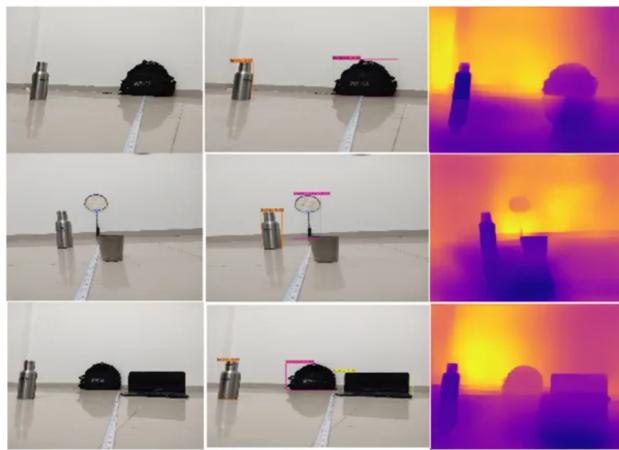


Rysunek 4: Architektura Dist-YOLO [13]

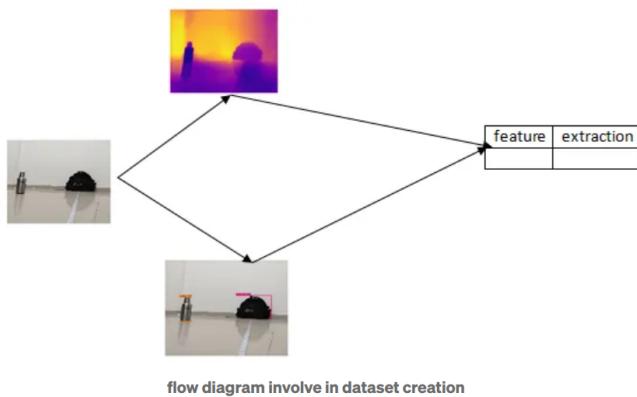
Dużą zaletą tego rozwiązania jest możliwość nauczenia sieci neuronowej rozpoznawania odległości do obiektów jedynie za pomocą etykiet, bez żadnych danych geometrycznych. Ta metoda wymaga jednak odpowiednio przygotowanego i dużego zbioru treningowego. Poza tym jedyna dostępna implementacja [14] jest zależna od nieaktualnych wersji wielu bibliotek, co utrudnia stworzenie i ewaluację własnego modelu, a co za tym idzie — integrację z całością docelowej aplikacji.

### 2.1.2 Wykorzystanie szacowania głębi i klasyfikatorów

Jeszcze inny sposób na oszacowanie odległości jest opisany w artykule [15]. Polega na ekstrakcji cech z obrazu, a następnie poddaniu ich procesowi uczenia takimi metodami jak XGBoost i lasy losowe. Najpierw przeprowadza się detekcję obiektów i wykonuje się mapę głębi obrazu (wykorzystując gotowe modele). Następnie wykonuje się ekstrakcję cech takich jak wysokość i szerokość obiektu, średnia wartość pikseli w mapie głębi, itp. Klasyfikator oparty na XGBoost lub lasach losowych uczy się następnie szacowania odległości na podstawie cech oraz przygotowanych etykiet.



Rysunek 5: Od lewej do prawej: obrazy oryginalne, obrazy z wykrytymi obiektami, mapy głębi [15].



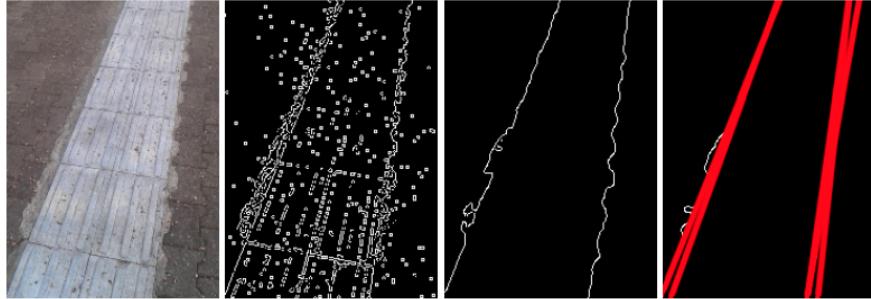
Rysunek 6: Schemat ekstrakcji cech [15].

W tej pracy zdecydowano się na użycie zbliżonej metody, wykorzystującej dwa modele: model detekcyjny i model szacowania głębi. Model detekcyjny wykrywa obiekty, następnie za pomocą modelu szacowania głębi można określić ich względną odległość.

## 2.2 Metody utrzymywania pieszego w pasie ruchu

Istnieją dwa główne podejścia do wykrywania chodnika, które mają na celu wspomagać poruszanie się po mieście osobom niewidomym. Pierwszym z nich jest skorzystanie z uczenia maszynowego, a drugim zastosowanie klasycznych algorytmów przetwarzania obrazu. Przykład użycia uczenia maszynowego do tego zadania można znaleźć w artykule [16], gdzie autorzy opisali próby wykrywania chodnika za pomocą różnego rodzaju klasyfikatorów, a najlepszym z nich okazał się las losowy z dokładnością na poziomie 83,11%. W innym artykule można znaleźć opis metod wykorzystujących operacje takie, jak: wygładzenie filtrem Gaussa, wykrywanie krawędzi detektorem Canny, zastosowanie transformacji Hougha do znajdowania linii prostych [17]. Jednakże ta i inne prace [18][19] skupiają się na wykrywaniu specjalnie oznaczonych ścieżek dla niewidomych (przykład z artykułu [17] znajduje się na rysunku 7), a nie na ogólnym wykrywaniu chodnika. W tych podejściach ma znaczenie również, że autorzy tych artykułów chcieli znaleźć sposób na wykrywanie konkretnego wzoru ścieżki, charakterystycznego dla ich kraju - odpowiednio Chin [18] oraz Korei Południowej [19]. W Polsce takie ścieżki nie są jeszcze powszechnne, zwłaszcza na osiedlach zbudowanych kilkadziesiąt lat temu. Potrzeba zatem bardziej ogólnego rozwiązania.

Podobnym problemem do utrzymywania pieszego na chodniku jest utrzymywanie samochodu autonomicznego na pasie ruchu. W tej pracy wybrano właśnie to podejście, opierając się na dostępnych przykładach [20][21].



Rysunek 7: Przykład wykrywania specjalnie oznaczonej ścieżki dla osób niewidomych [17]. Od lewej do prawej: obraz oryginalny, obraz przetworzony detektorem Canny, obraz przetworzony filtrem medianowym i detektorem Canny, obraz z liniami wykrytymi za pomocą transformacji Hougha.

### 3 Zadanie szacowania odległości do obiektów

Zdecydowano się na rozwiązywanie, które polega na wykryciu obiektów na obrazie za pomocą modelu detekcji i jednoczesnym stworzeniu mapy głębi. Następnie wartość piksela na mapie głębi w środkowym punkcie ramki ograniczającej danego obiektu służy do oszacowania odległości. Wartości pikseli w mapie głębi mają wartości między 0 a 255. Można uznać, czy dany punkt jest blisko lub daleko, wyznaczając kilka przedziałów na skali wartości pikseli. Należy przy tym przeprowadzić wiele prób, dobierając granice przedziałów tak, aby jak najbardziej odpowiadały rzeczywistości. Trzeba też pamiętać o tym, że wartości na mapie głębi mogą zależeć od rodzaju perspektywy albo liczby szczegółów na zdjęciu, co wynika z faktu, że tworzenie mapy głębi polega na ustalaniu względnych odległości [22].

Model detekcji wytrenowano za pomocą architektury YOLOv8 na zbiorze treningowym dostarczonym przez opiekuna pracy. Wybrano YOLOv8 ze względu na jakość i szybkość, jest to jedno z najbardziej zaawansowanych dostępnych narzędzi do detekcji i klasyfikacji. Model wytrenowany za pomocą YOLOv8 może też być łatwo przekonwertowany do formatu TF Lite, który jest niezbędny przy budowie aplikacji mobilnej.

Przetestowano dwa modele szacowania głębi: Monodepth2 oraz MiDaS. Skorzystano z gotowych, wytrenowanych modeli. Porównano je następnie pod względem jakości i szybkości. Następnie zdecydowano się wybrać model MiDaS w formacie TF Lite przy budowie aplikacji

mobilnej.

W praktycznym zastosowaniu ta metoda może służyć zarówno do wykrywania przeszkód, jak i do naprowadzania użytkownika na przejście dla pieszych, czy do informowania o obecności sygnalizacji świetlnej.

### 3.1 Dane

Zbiór danych został stworzony na podstawie zbioru obrazów dostarczonych przez opiekuna pracy. Zbiór zawiera zdjęcia ulic, chodników, czy przejścia dla pieszych. Znaczna część zdjęć została wykonana we Wrocławiu. Na początku odrzucono obrazy, na których nie zaobserwowano żadnych interesujących obiektów. Następnie oznaczono obiekty z następujących klas: śmiertnik, samochód, przejście dla pieszych, osoba, hulajnoga elektryczna, znak drogowy, sygnalizacja świetlna, tramwaj. Nie oznaczono obiektów typu drzewa, lampy, czy słupy, ponieważ ważniejszym celem w tej pracy było przetestowanie metody, niż przygotowanie modelu wykrywającego obiekty każdej możliwej klasy. W związku z tym ograniczono się do powyżej wymienionych klas.

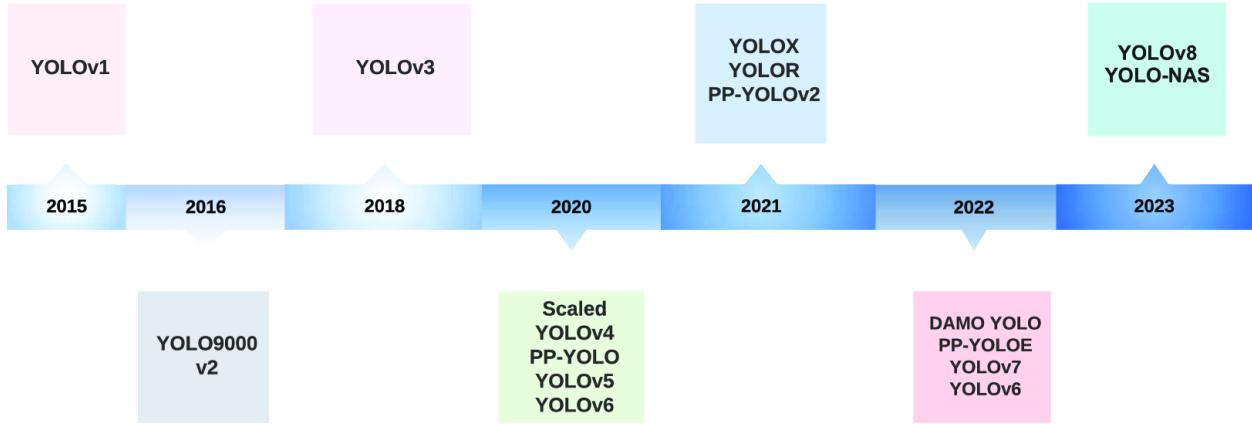
Do oznaczenia zdjęć i podziału zbioru na część treningową, testową, i walidacyjną posłużono się platformą Roboflow. Ostatecznie wygenerowano zbiór, który liczy 716 obrazów treningowych, 102 obrazy testowe, 205 obrazów walidacyjnych.



Rysunek 8: Przykładowe obrazy ze zbioru danych.

### 3.2 YOLOv8

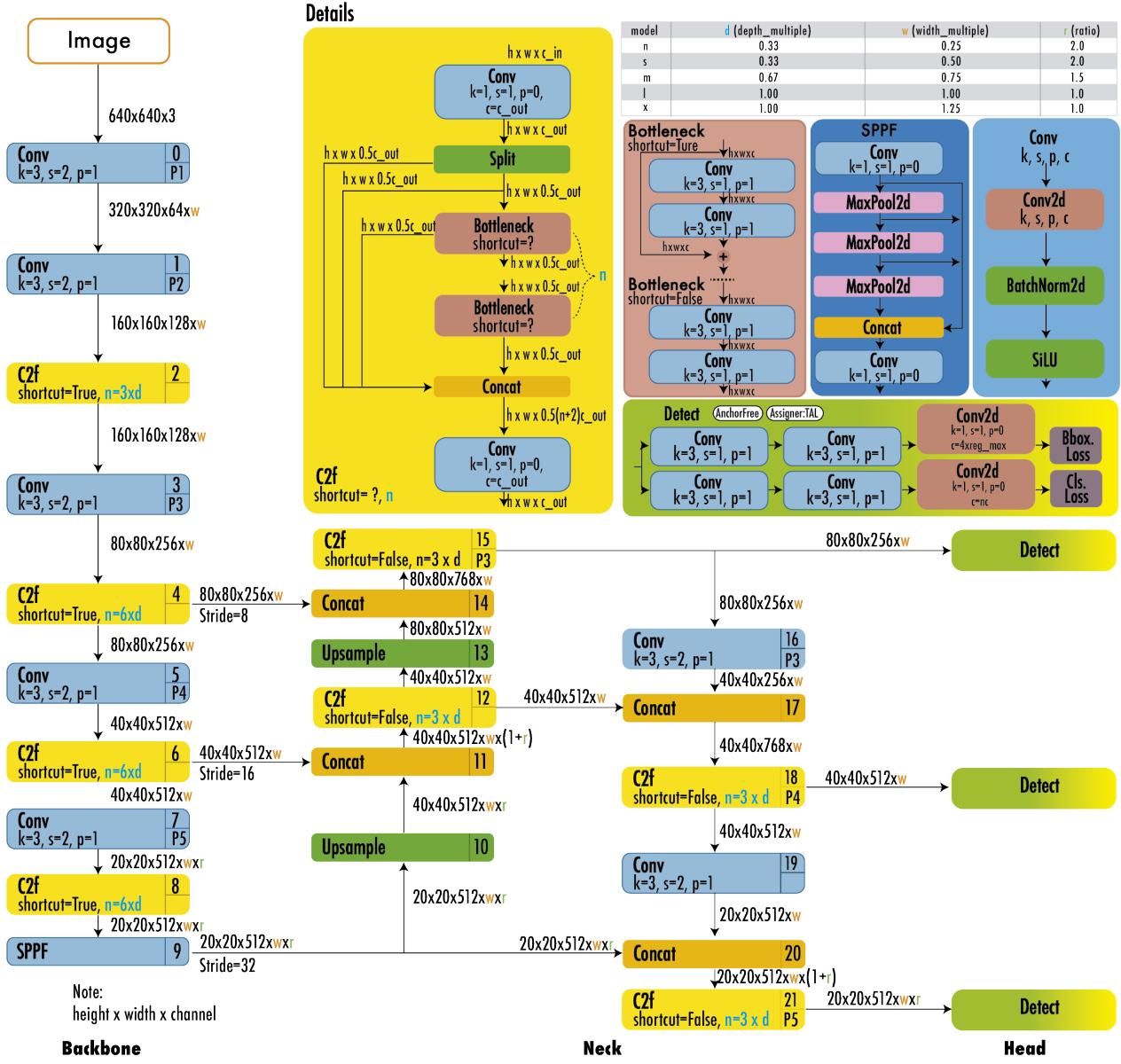
YOLO (*You Only Look Once* — Patrzysz tylko raz) jest zaawansowanym narzędziem opartym na głębokich sieciach neuronowych, które służy do detekcji i klasyfikacji obiektów na obrazach. Jest jednym z najszybszych i najdokładniejszych narzędzi tego typu [23] [24] [25].



Rysunek 9: Kolejne wersje YOLO od YOLOv1 do YOLOv8 [26]

W tej pracy skorzystano z wersji YOLOv8, która posiada kilka nowych funkcjonalności. Za pomocą YOLOv8 można rozwiązywać zadania detekcji (ang. *object detection*), segmentacji (ang. *instance segmentation*), klasyfikacji (ang. *classification*), śledzenia (ang. *tracking*), czy szacowania orientacji (ang. *pose estimation*) [27].

YOLOv8 posiada złożoną architekturę, którą należy pokróćce omówić. Po pierwsze, jest to model typu *anchor-free* z rozdzieloną częścią wyjściową (*head*), co pozwala na niezależne rozwiązywanie zadań detekcji, klasyfikacji, czy regresji [26]. *Anchor-free* oznacza tutaj, że model bezpośrednio wykrywa środek obiektu, bez potrzeby używania tzw. *anchor boxes*, czyli prototypowych ramek ograniczających [28]. Korzysta z funkcji straty *Complete IoU (IoU — Intersection over Union)* [29] oraz DFL (*Distribution Focal Loss*) [30] dla ramek ograniczających i z binarnej entropii krzyżowej (ang. *binary cross-entropy*) dla zadania klasyfikacji [26]. Na rysunku 10 przedstawiono szczegółową architekturę YOLOv8:



Rysunek 10: Architektura YOLOv8 [26]

Jedną z metryk, z których korzysta YOLOv8 jest zaufanie (ang *confidence*), które definiuje się następująco [23]:

$$S_{conf} = P(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}, \quad (10)$$

gdzie:

- $S_{conf}$  — zaufanie,
- $P(\text{Class}_i)$  — prawdopodobieństwo klasifikacji obiektu i-tej klasy,

- $\text{IOU}_{\text{pred}}^{\text{truth}}$  — IoU (*Intersection over Union*) — metryka wyznaczająca dokładność przewidywanej przez model ramki ograniczającej w stosunku do wzorcowej ramki ograniczającej.

Na rysunkach 13, 16 widać przykład zastosowania tej metryki w analizie modelu — zmieniając próg zaufania można obserwować, jak zmienia się wskaźnik F1.

Poza tym YOLOv8 dostarcza model umożliwiający segmentację obiektów (YOLOv8-Seg), jednak nie korzystano z tej funkcjonalności w tej pracy.

### 3.2.1 Trenowanie modelu YOLOv8

Trenowanie modelu opartego na YOLOv8 można wykonać za pomocą prostego skryptu lub odpowiedniej komendy z linii komend. Skorzystano ze skryptu przedstawionego na rysunku 11:

```

1  #!/usr/bin/env python3
2
3  import os
4  from ultralytics import YOLO
5
6  # building the model
7
8  model = YOLO('yolov8n.pt')
9
10 DATA_DIR = os.path.abspath('./dataset/data.yaml')
11
12 model.train(data=DATA_DIR, epochs=150, imgsz=640)
13
14 metrics = model.val()
15
16 model.export('tflite')
```

Rysunek 11: Skrypt służący do trenowania YOLOv8.

Sieć zaczyna trening z zainicjalizowanymi wagami, które są wczytywane z pliku `yolov8n.pt`. Informacje o zbiorze danych znajdują się w pliku `data.yaml`, który przedstawia rysunek 12:

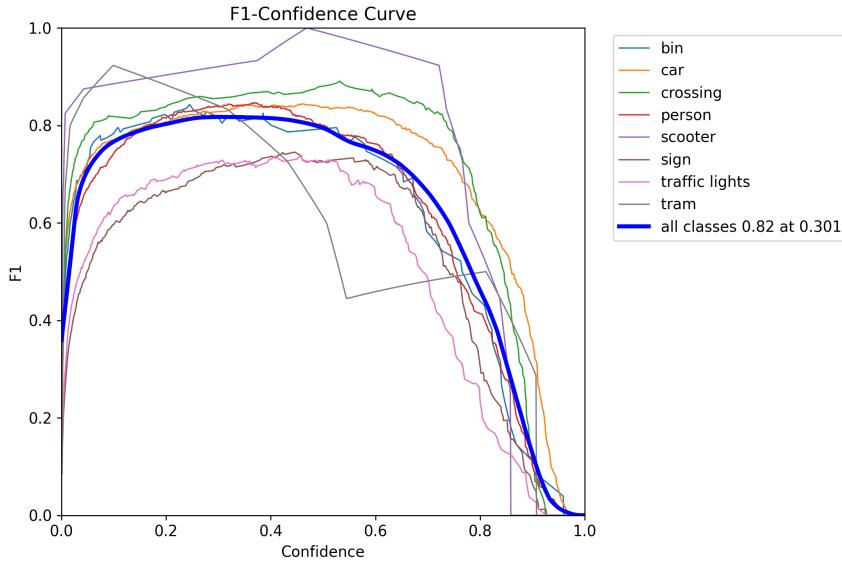
```
1 train: ../train/images
2 val: ../valid/images
3 test: ../test/images
4
5 nc: 8
6 names: ['bin', 'car', 'crossing', 'person', 'scooter', 'sign', 'traffic lights', 'tram']
7
8 roboflow:
9   workspace: magisterka-zdo71
10  project: streetobjects-evdjw
11  version: 1
12  license: CC BY 4.0
13  url: https://universe.roboflow.com/magisterka-zdo71/streetobjects-evdjw/dataset/1
```

Rysunek 12: Plik zawierający informacje o zbiorze danych.

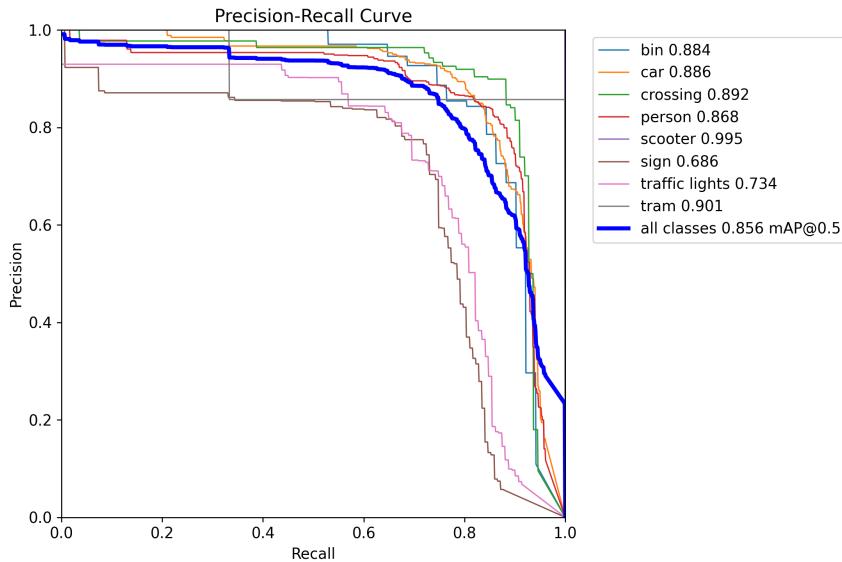
Po zakończeniu treningu od razu przeprowadzana jest walidacja na wyselekcjonowanej części danych, a następnie również konwersja do formatu TF Lite. Wyniki walidacji pozwalały ocenić jakość modelu za pomocą różnych metryk. Zaprezentowano poniżej kilka wykresów obrazujących efekty uczenia dwóch modeli. Pierwszy z nich był trenowany przez sto epok, a drugi przez sto pięćdziesiąt.

Można zauważyc, że metryki obu modeli są zbliżone. W przypadku drugiego modelu pięćdziesiąt epok uczenia więcej nie poprawiło znacząco wskaźnika F1 — jest większy o zaledwie 3 punkty procentowe (w optymalnym przypadku). Oznacza to, że nie należy się spodziewać wyraźnie widocznych różnic w trakcie korzystania z modelu w warunkach rzeczywistych (tzn. w trakcie korzystania z niego w aplikacji mobilnej).

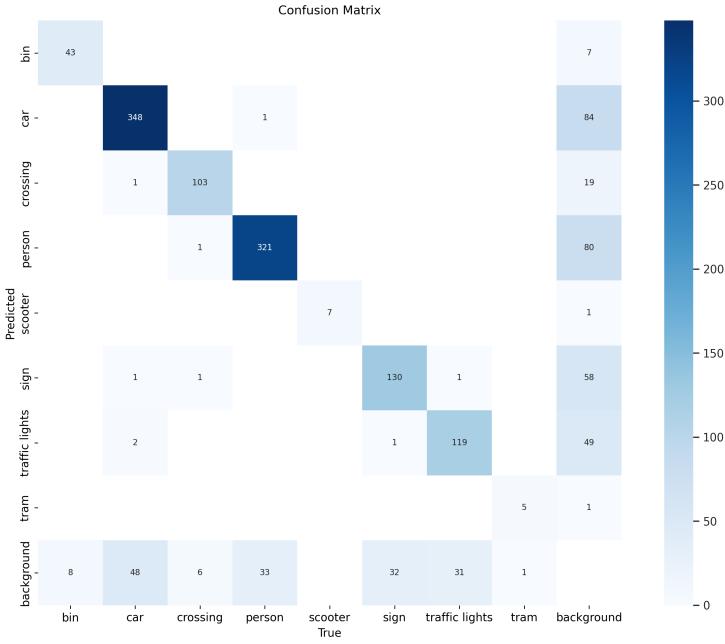
### 3.2.2 Model 1 - 100 epok



Rysunek 13: Wykres zależności wskaźnika F1 w zależności od progu zaufania (*confidence*) [31]. W legendzie zaznaczono, że model osiągnął najlepsze F1 przy progu zaufania równym 30,1%.

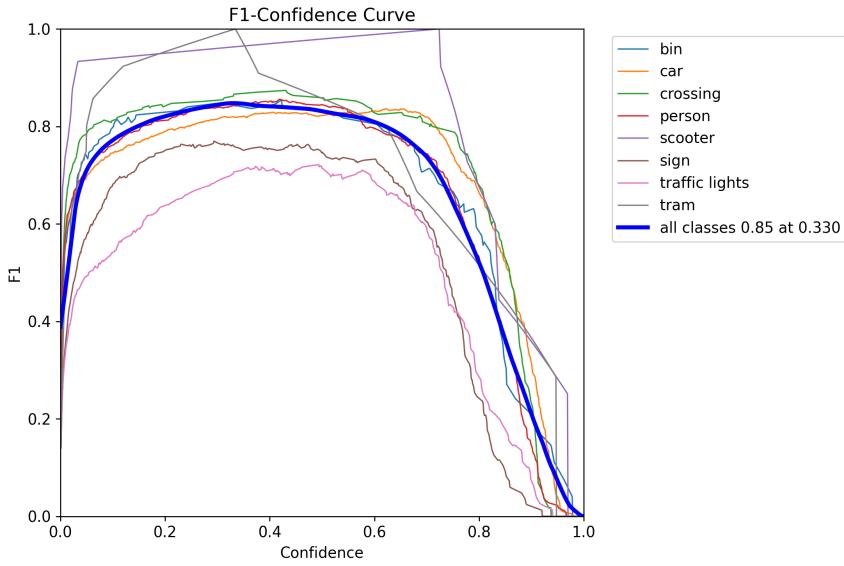


Rysunek 14: Wykres zależności czułości modelu (*recall*) od precyzji (*precision*) [31].

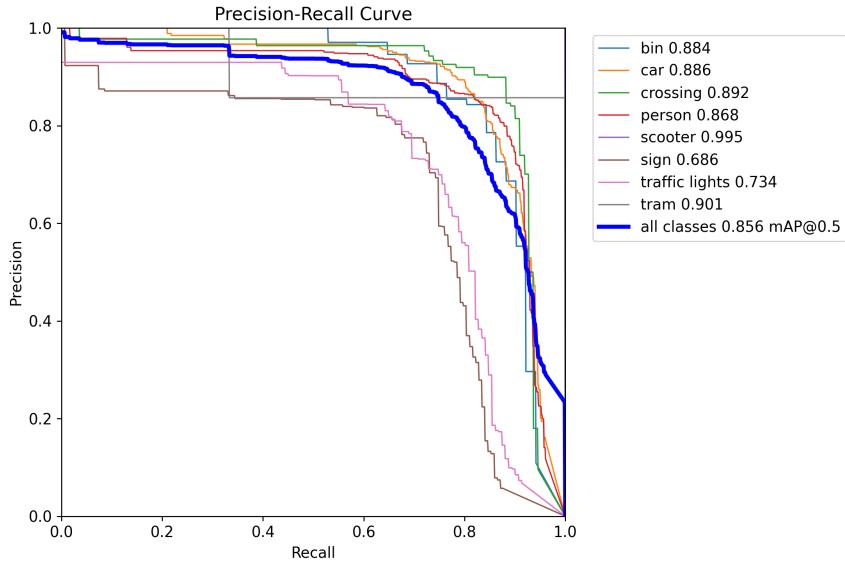


Rysunek 15: Macierz pomyłek.

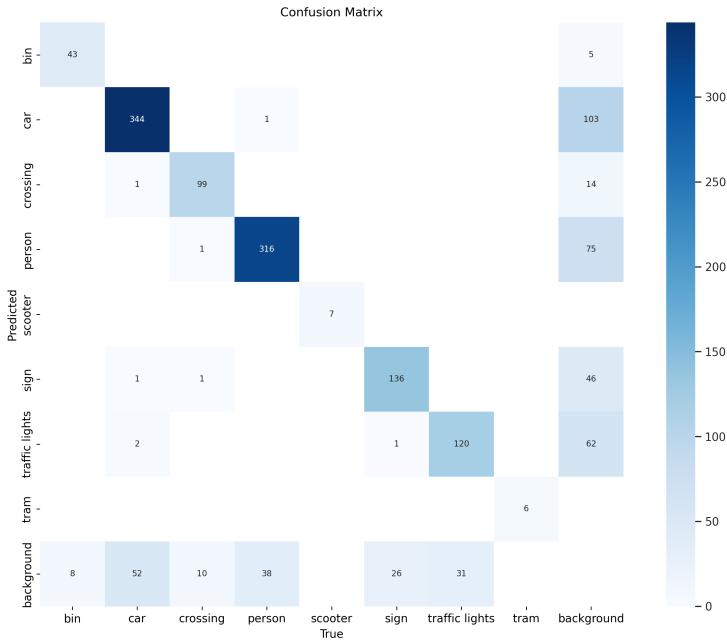
### 3.2.3 Model 2 - 150 epok



Rysunek 16: Wykres zależności wskaźnika F1 w zależności od progu zaufania (*confidence*) [31]. W legendzie zaznaczono, że model osiągnął najlepsze F1 przy progu zaufania równym 33%.



Rysunek 17: Wykres zależności czułości modelu (*recall*) od precyzji (*precision*) [31].



Rysunek 18: Macierz pomyłek.

### 3.3 MiDaS

MiDaS jest modelem szacowania głębi typu *Monocular Depth Estimation*, co oznacza, że tworzy mapę głębi na podstawie pojedynczego obrazu. Trenowanie modeli MiDaS polegało

na podejściu *zero-cross dataset transfer*. Oznacza to, że modele były trenowane na jednych zbiorach danych, a testowane na innych, przez co obrazy testowe bardziej różniły się od treningowych niż zazwyczaj [32]. W tej pracy użyto gotowych modeli, które zostały stworzone przez autorów tej metody [33].



Rysunek 19: Przykładowe efekty działania modelu MiDaS. Jako obrazy wejściowe zostały tu użyte klatki z filmów ze zbioru 3D Movies [32].

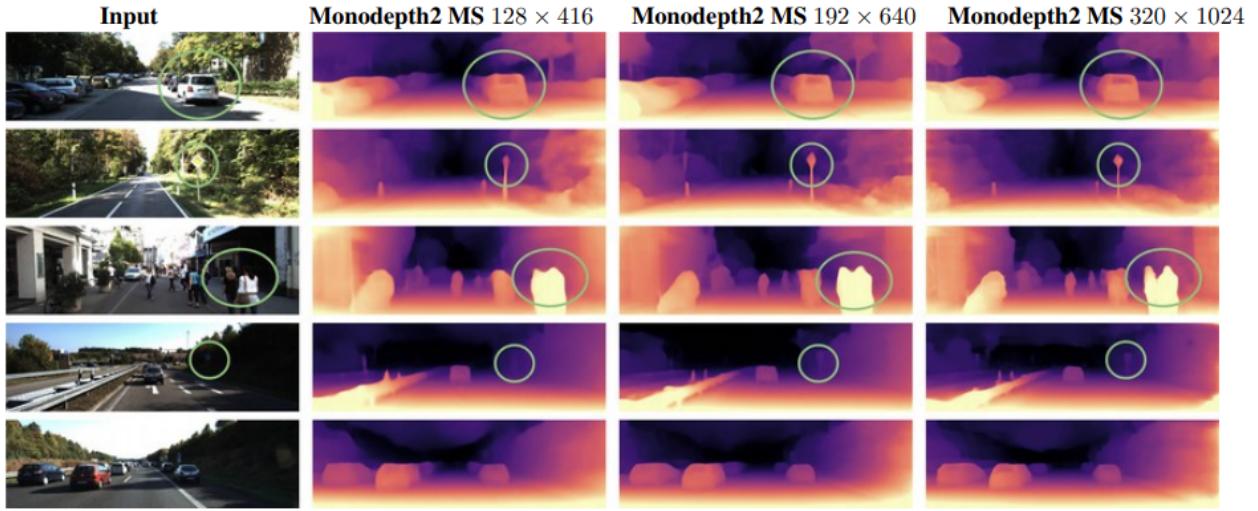


Rysunek 20: Przykładowe efekty działania modelu MiDaS na obrazach i rysunkach [32].

### 3.4 Monodepth2

Monodepth2 również jest modelem typu *Monocular Depth Estimation*. Wykorzystuje uczenie samonadzorowane, czyli podejście, w którym model w trakcie treningu wykorzystuje dane do tworzenia sygnałów nadzorujących (np. etykiet). W porównaniu do innych modeli uczonych w ten sposób, autorzy tej metody wprowadzili kilka ulepszeń [34]:

- Wprowadzenie funkcji straty minimalizującej błąd reprojekcji, dzięki czemu model dobrze radzi sobie z występowaniem okluzji.
- Użycie metody wieloskalowego próbkowania w pełnej rozdzielczości, co zmniejsza liczbę artefaktów.
- Wprowadzenie dodatkowej funkcji straty, dzięki której w trakcie treningu model ignoruje piksele, które naruszają założenia ruchu kamery. Tutaj warto zaznaczyć, że takie modele zakładają statyczną scenę i poruszającą się kamerę.



Rysunek 21: Przykładowe efekty działania Monodepth2 dla różnych rozdzielczości [34].

### 3.5 Implementacja detekcji obiektów i szacowania odległości w języku Python

Opisaną już wcześniej metodę wykrywania obiektów i ich położenia zaimplementowano w języku Python. Stworzono dwa warianty — jeden wykorzystujący model Monodepth2 i drugi wykorzystujący model MiDaS. Poniżej rozpisano szczegółowo poszczególne kroki tej metody:

1. Detekcja obiektów za pomocą YOLOv8 i jednoczesne przetworzenie obrazu wejściowego za pomocą modelu szacowania głębi.
2. Normalizacja wyniku szacowania głębi tak, aby wartości mieściły się między 0 a 255 (w celu możliwości wyświetlenia wyniku). Następnie skalowanie na wartości między 0 a 100 (w celu lepszego wyznaczenia zakresów odległości).
3. Dla każdego wykrytego obiektu:
  - wyznaczenie współrzędnych środkowego punktu ramki ograniczającej -  $(x, y)$ ,
  - ustalenie etykiety położenia,
  - ustalenie etykiety odległości.

Ustalenie etykiety położenia dokonuje się na podstawie współrzędnej  $x$  środka ramki ograniczającej ( $w$  - szerokość obrazu w pikselach). Pogrubioną czcionką zaznaczono nazwy etykiet.

- $0 \leq x \leq \frac{1}{3}w$  — obiekt w lewej części obrazu (**LEFT**),
- $\frac{w}{3} < x \leq \frac{2}{3}w$  — obiekt w środkowej części obrazu (**CENTER**),
- $\frac{2}{3}w < x \leq w$  — obiekt w prawej części obrazu (**RIGHT**).

Ustalenie etykiety odległości odbywa się na podstawie wartości obrazu mapy głębi w punkcie  $(x, y)$  (tą wartość oznaczono poniżej jako  $v$ ). Pogrubioną czcionką zaznaczono nazwy etykiet.

- $v > 50$  — bardzo blisko (**VERY CLOSE**),
- $30 < v \leq 50$  — blisko (**CLOSE**),
- $15 < v \leq 30$  — średnia odległość (**MEDIUM**),
- $5 < v \leq 15$  — daleko (**FAR**),
- $v \leq 5$  — bardzo daleko (**VERY FAR**).

W programie testującym metodę zastosowano powyższą procedurę wobec wszystkich obrazów ze zbioru walidacyjnego. Przykładowe wyniki działania programu przedstawia rysunek 22.



Rysunek 22: Przykładowe efekty działania programu

## 3.6 Testowanie metody

W trakcie badań tej metody uwzględniono cztery kombinacje użytych modeli:

- model YOLOv8 wytrenowany przez 100 epok i MiDaS,
- model YOLOv8 wytrenowany przez 100 epok i Monodepth2,
- model YOLOv8 wytrenowany przez 150 epok i MiDaS,
- model YOLOv8 wytrenowany przez 150 epok i Monodepth2.

### 3.6.1 Porównanie pod względem jakości rozwiązania

Przetestowano metodę pod względem jakości rozwiązania. Testowano tutaj bardziej modele szacowania głębi. Modele oparte na YOLOv8 zostały wcześniej zbadane podczas etapu walidacji, co opisano wyżej. Zaobserwowano zbliżoną jakość rozwiązań dla obu modeli. Na rysunkach 23, 24, 25, 26 porównano efekty działania na trzech obrazach ze zbioru walidacyjnego dla czterech przypadków.



Rysunek 23: Trzy obrazy wyjściowe przy zastosowaniu modelu MiDaS do estymacji głębi i modelu opartego na YOLOv8, wytrenowanego przez 100 epok, do detekcji obiektów.



Rysunek 24: Trzy obrazy wyjściowe przy zastosowaniu modelu Monodepth2 do estymacji głębi i modelu opartego na YOLOv8, wytrenowanego przez 100 epok, do detekcji obiektów.



Rysunek 25: Trzy obrazy wyjściowe przy zastosowaniu modelu MiDaS do estymacji głębi i modelu opartego na YOLOv8, wytrenowanego przez 150 epok, do detekcji obiektów.



Rysunek 26: Trzy obrazy wyjściowe przy zastosowaniu modelu Monodepth2 do estymacji głębi i modelu opartego na YOLOv8, wytrenowanego przez 150 epok, do detekcji obiektów.

W tabelach 1, 2, 3 przedstawiono wartości pikseli na mapach głębi w środkach ramek ograniczających konkretne obiekty. W tabeli 1 znajdują się wartości dla obrazu po lewej z rysunku 25, w tabeli 2 — dla obrazu środkowego z rysunku 25, w tabeli 3 — dla obrazu po prawej z rysunku 25. Obiekty zostały wykryte za pomocą modelu opartego na YOLOv8, wytrenowanego przez 150 epok. Można zauważyć, że choć w niektórych przypadkach wartości są dość rozbieżne, to jednak w wielu przypadkach oba modele dały zbliżone wartości.

|            | osoba | samochód 1 | samochód 2 |
|------------|-------|------------|------------|
| MiDaS      | 14    | 26         | 29         |
| Monodepth2 | 1     | 23         | 32         |

Tabela 1: Wartości dla pierwszego obrazu. Liczby oznaczają wartości pikseli na mapie głębi w środku ramki ograniczającej obiekt.

|            | osoba 1 | osoba 2 | osoba 3 | przejście | samochód | śmiertnik | sygnalizacja 1 | sygnalizacja 2 | sygnalizacja 3 | sygnalizacja 4 |
|------------|---------|---------|---------|-----------|----------|-----------|----------------|----------------|----------------|----------------|
| MiDaS      | 58      | 39      | 46      | 129       | 15       | 50        | 29             | 27             | 10             | 27             |
| Monodepth2 | 56      | 48      | 43      | 116       | 55       | 46        | 39             | 38             | 41             | 21             |

Tabela 2: Wartości dla drugiego obrazu. Liczby oznaczają wartości pikseli na mapie głębi w środku ramki ograniczającej obiekt.

|            | samochód 1 | samochód 2 | samochód 3 | samochód 4 | samochód 5 | osoba | znak | hulajnoga 1 | hulajnoga 2 |
|------------|------------|------------|------------|------------|------------|-------|------|-------------|-------------|
| MiDaS      | 83         | 47         | 8          | 13         | 31         | 13    | 7    | 55          | 30          |
| Monodepth2 | 69         | 51         | 17         | 31         | 44         | 25    | 18   | 78          | 41          |

Tabela 3: Wartości dla trzeciego obrazu. Liczby oznaczają wartości pikseli na mapie głębi w środku ramki ograniczającej obiekt.

### 3.6.2 Porównanie pod względem czasu działania

Sprawdzono czas działania programu, który przetwarzał wszystkie obrazy ze zbioru walidacyjnego. Procedurę dla każdego przypadku przeprowadzono 5 razy. W tabeli 4 przedstawiono wyniki. Zmierzono tutaj wartość FPS (ang. *frames per second*), czyli liczbę klatek, którą program przetworzył w sekundę.

|                | MiDaS + YOLOv8 (100 epok) | Monodepth2 + YOLOv8 (100 epok) | MiDaS + YOLOv8 (150 epok) | Monodepth2 + YOLOv8 (150 epok) |
|----------------|---------------------------|--------------------------------|---------------------------|--------------------------------|
| 1              | 7,19                      | 5,98                           | 6,63                      | 5,57                           |
| 2              | 7,28                      | 6,08                           | 6,69                      | 5,92                           |
| 3              | 6,93                      | 5,84                           | 6,94                      | 5,75                           |
| 4              | 6,61                      | 6,08                           | 6,47                      | 5,82                           |
| 5              | 7,26                      | 5,88                           | 6,77                      | 5,94                           |
| <b>Średnia</b> | <b>7,05</b>               | <b>5,97</b>                    | <b>6,70</b>               | <b>5,80</b>                    |

Tabela 4: Porównanie czasu działania dla różnych kombinacji użytych modeli. Wyniki przedstawiono posługując się metryką FPS (ang. *frames per second* - klatki na sekundę).

### 3.6.3 Podsumowanie testów

Osiągnięto zbliżone efekty pod względem jakości rozwiązania dla obu modeli szacowania głębi. Zdecydowano się użyć modelu MiDaS przy budowie aplikacji mobilnej ze względu na krótszy czas działania. Do wykonywania zadania detekcji w aplikacji mobilnej przeznaczono model wytrenowany przez 150 epok ze względu na lepsze metryki.

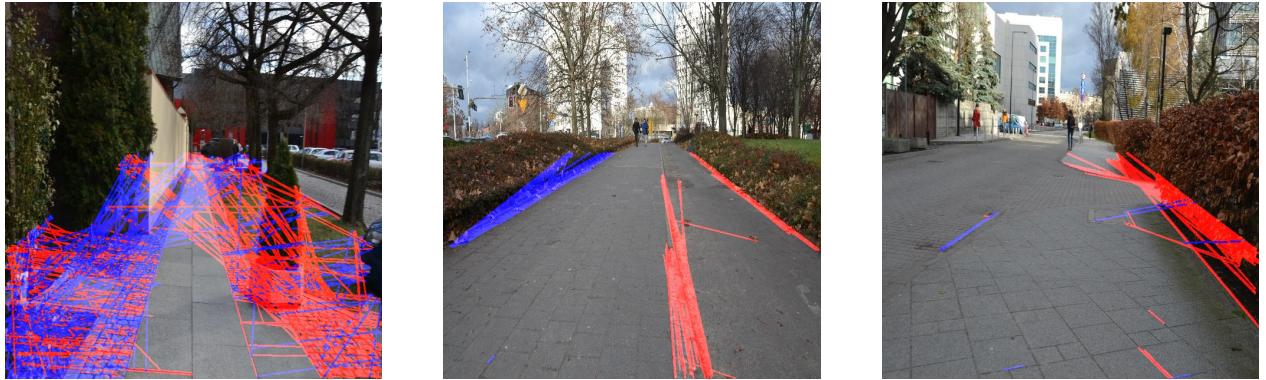
## 4 Zadanie utrzymywania użytkownika na chodniku

W tej pracy postarano się rozwiązać zadanie utrzymania użytkownika na chodniku w sposób zbliżony do zadania utrzymywania autonomicznego samochodu na pasie ruchu. Implementację oparto na dostępnych przykładach [20][21], a następnie dobrano hiperparametry, aby

uzyskać jak najlepsze wyniki. Metoda polega na przetworzeniu obrazu wejściowego w kilku krokach, a następnie podjęciu decyzji na podstawie wyników. Poniżej rozpisano kolejne kroki tej metody:

1. Wczytanie obrazu kolorowego jako trójwymiarowej macierzy.
2. Rozmycie obrazu filtrem gaussowskim.
3. Wykrycie krawędzi za pomocą detektora Canny.
4. Wyznaczenie obszaru zainteresowań.
5. Znalezienie linii prostych za pomocą transformaty Hougha.
6. Podzielenie linii na dwie grupy ze względu na znak współczynnika kierunkowego, wyznaczenie linii wypadkowych — jednej z dodatnim, a drugiej z ujemnym współczynnikiem kierunkowym (poziome linie są odrzucane).
7. Wyznaczenie punktu przecięcia prostych. Na tej podstawie wnioskuje się, czy użytkownik porusza się po chodniku w prawidłowy sposób.

O ile operacje na obrazach zostały omówione w tej pracy wcześniej, to należy dokładniej opisać dwa ostatnie punkty metody. Po wyznaczeniu układu współrzędnych w sposób charakterystyczny dla komputerów i telefonów — początek układu jest w lewym górnym rogu, oś  $X$  biegnie od lewej do prawej, a oś  $Y$  z góry na dół — można zauważyc, że lewa krawędź chodnika będzie prostą o ujemnym współczynniku kierunkowym, a prawa — o dodatnim. Na tej podstawie linie wykryte transformacją Hougha grupuje się ze względu na znak współczynnika kierunkowego, co zobrazowano na rysunku 27.



Rysunek 27: Linie proste wykryte za pomocą transformacji Hougha na trzech różnych obrazach. Kolorem niebieskim oznaczono proste o ujemnym współczynniku kierunkowym, a kolorem czerwonym proste o dodatnim współczynniku kierunkowym.

Następnie liczy się średnie wartości współczynnika kierunkowego i wyrazu wolnego w każdej grupie, by oszacować rzeczywiste położenie krawędzi chodnika na obrazie.

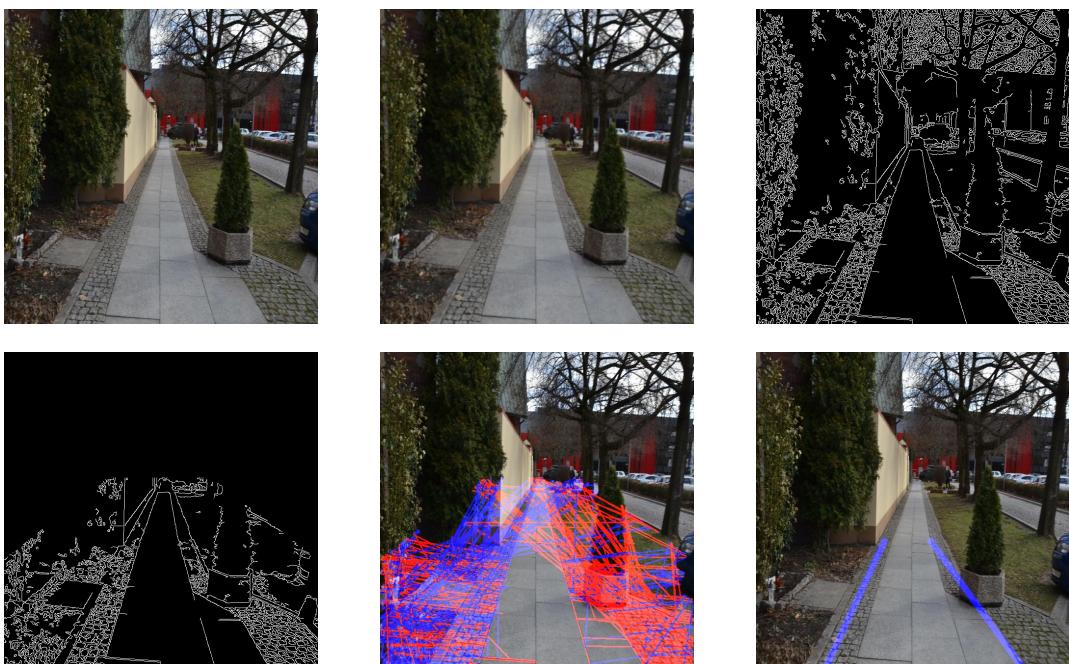
Mając wyznaczone równania kierunkowe obu krawędzi chodnika można obliczyć ich punkt przecięcia. Od położenia tego punktu zależy decyzja, czy użytkownik porusza się po chodniku w prawidłowy sposób. Punkt nie może leżeć zbyt nisko — to by oznaczało, że krawędź chodnika jest pod niewłaściwym kątem względem użytkownika. Może to też oznaczać, że użytkownik w ogóle nie znajduje się na chodniku. Punkt też powinien leżeć w środkowej części obrazu. W przeciwnym razie oznacza to, że użytkownik zboczył za bardzo na prawo lub na lewo. Mając punkt przecięcia  $(x, y)$  nałożono ograniczenia:

$$\begin{aligned} \frac{w}{4} \leq x \leq \frac{3w}{4}, \\ y \leq \frac{3h}{5} \end{aligned} \quad (11)$$

gdzie:

- $h$  - wysokość obrazu w pikselach,
- $w$  - szerokość obrazu w pikselach.

Na rysunku 28 pokazano kolejne kroki przetwarzania obrazu w tej metodzie:



Rysunek 28: Kolejne kroki metody. Górnny rzad od lewej do prawej: obraz wejściowy, obraz wygładzony filtrem Gaussa, krawędzie wykryte detektorem Canny. Dolny rzad od lewej do prawej: krawędzie w obszarze zainteresowań, obraz z prostymi wykrytymi transformacją Hougha, obraz wyjściowy.

#### 4.1 Implementacja metody za pomocą OpenCV

Metoda została zaimplementowana w języku Python przy pomocy biblioteki OpenCV. Zrobiono to w celach testowych, przed wbudowaniem algorytmu w strukturę aplikacji mobilnej. Przy poniższym opisie wspomagano się oficjalną dokumentacją [35]. Wykorzystano następujące funkcje do kolejnych kroków metody:

1. Konwersja do skali szarości (w wariancie algorytmu, który ją uwzględnia) - `cvtColor(src, code)`, gdzie:
  - `src` — obraz wejściowy,
  - `code` — kod zmiany kolorów, w tym przypadku: `code=cv2.BGR2GRAY`.
2. Wygładzenie filtrem Gaussa — `GaussianBlur(src, ksize, sigma)`, gdzie:

- **src** — obraz wejściowy,
- **ksize** — rozmiar jądra filtru Gaussa podawany jako para liczb,
- **sigma** — odchylenie standardowe.

3. Wykrycie krawędzi detektorem Canny — `Canny(image, threshold1, threshold2)`, gdzie:

- **image** — obraz wejściowy,
- **threshold1** — pierwszy próg (dolne ograniczenie),
- **threshold2** — drugi próg (górnego ograniczenie).

4. Wyznaczenie obszaru zainteresowań. Ten krok obejmuje stworzenie czarnego obrazu, narysowanie wielokąta, i wykonanie sumy bitowej obrazu wejściowego z maską. Wykorzystano tutaj:

- **zeros\_like(a)** — funkcja z biblioteki NumPy, która tworzy macierz wypełnioną zerami o takich samych wymiarach jak obraz wejściowy. Argument funkcji **a** to obiekt typu *array\_like*, na przykład tablica wielowymiarowa [36].
- **fillPolly(img, pts, color)** do narysowania wielokąta na czarnym obrazie (utworzenie maski), gdzie:
  - **img** — obraz wejściowy,
  - **pts** — zbiór punktów określający wielokąt, podany w postaci tablicy,
  - **color** — kolor wielokąta, podany jako trójką liczb  $(r, g, b)$ .
- **bitwise\_and(src1, src2)** — funkcja wykonująca sumę bitową na obrazach **src1** i **src2**.

5. Probabilistyczna transformacja Hougha — `HoughLinesP(image, lines, rho, theta, threshold, minLineLength, maxLineGap)`, gdzie:

- **image** — obraz wejściowy,
- **lines** — wyjściowy wektor linii prostych, które są reprezentowane w postaci pary punktów  $(x_1, y_1, x_2, y_2)$ .

- `rho` — rozdzielczość odległościowa w pikselach,
- `theta` — rozdzielczość kątowa w radianach.
- `minLineLength` — minimalna długość linii,
- `maxLineGap` — maksymalna przerwa między punktami na linii.

## 4.2 Testowanie metody

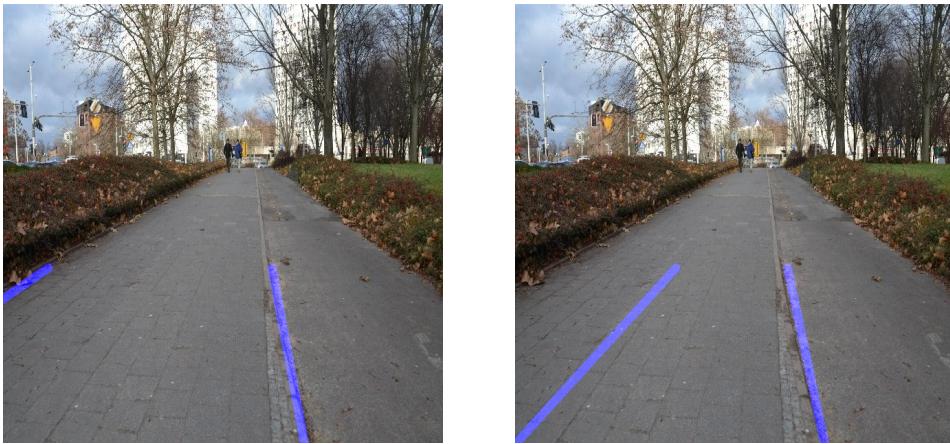
Metoda wykrywania pasa ruchu użyta w tej pracy jest zależna od wielu parametrów — w każdym etapie przetwarzania obrazu należy dobrać odpowiednie wartości, aby osiągnąć odpowiedni efekt. Z tego względu przeprowadzono badania każdego etapu metody osobno. Badając dany etap parametry w pozostałych funkcjach były ustalone. Poniżej podano kontrolne wartości argumentów w poszczególnych funkcjach:

- Filtr Gaussa: `ksize=(3,3)`, `sigma=0`,
- Detektor Canny: `threshold1=50`, `threshold2=150`,
- Obszar zainteresowań: sześciokąt o wierzchołkach  $(0, h), (0, \frac{2}{3}h), (\frac{1}{3}w, \frac{2}{5}h), (\frac{2}{3}w, \frac{2}{5}h), (w, \frac{2}{3}h), (w, h)$ .  $w$  — szerokość obrazu w pikselach,  $h$  — wysokość obrazu w pikselach,
- Transformacja Hougha:
  - `rho=1`,
  - `theta=\pi/180`, `threshold=20`,
  - `minLineLength=10`,
  - `maxLineGap=100`.

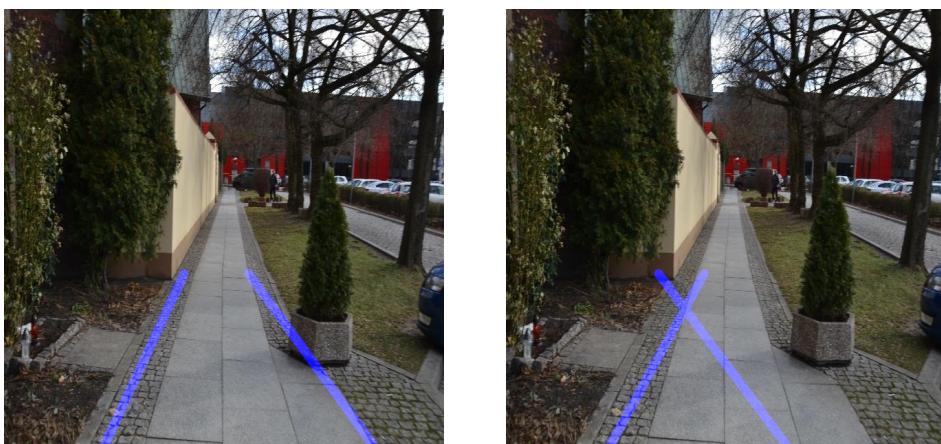
Poniższe badania przeprowadzono, aby pokazać wpływ poszczególnych etapów na rozwiązanie. Podczas dalszego etapu prac, w budowie aplikacji mobilnej, należało powtórnie dobrać parametry na podstawie działania metody w warunkach rzeczywistych (w trakcie spaceru chodnikiem).

#### 4.2.1 Wpływ konwersji do skali szarości

Przed wykonaniem wygładzenia filtrem Gaussa przekonwertowano obraz do skali szarości, żeby sprawdzić wpływ na efekt końcowy. Okazało się, że metoda działa lepiej, kiedy detektor Canny dostaje na wejście obraz kolorowy. Na rysunkach 29, 30 przedstawiono dwa przykłady działania algorytmu w dwóch wariantach - z konwersją do skali szarości i bez.



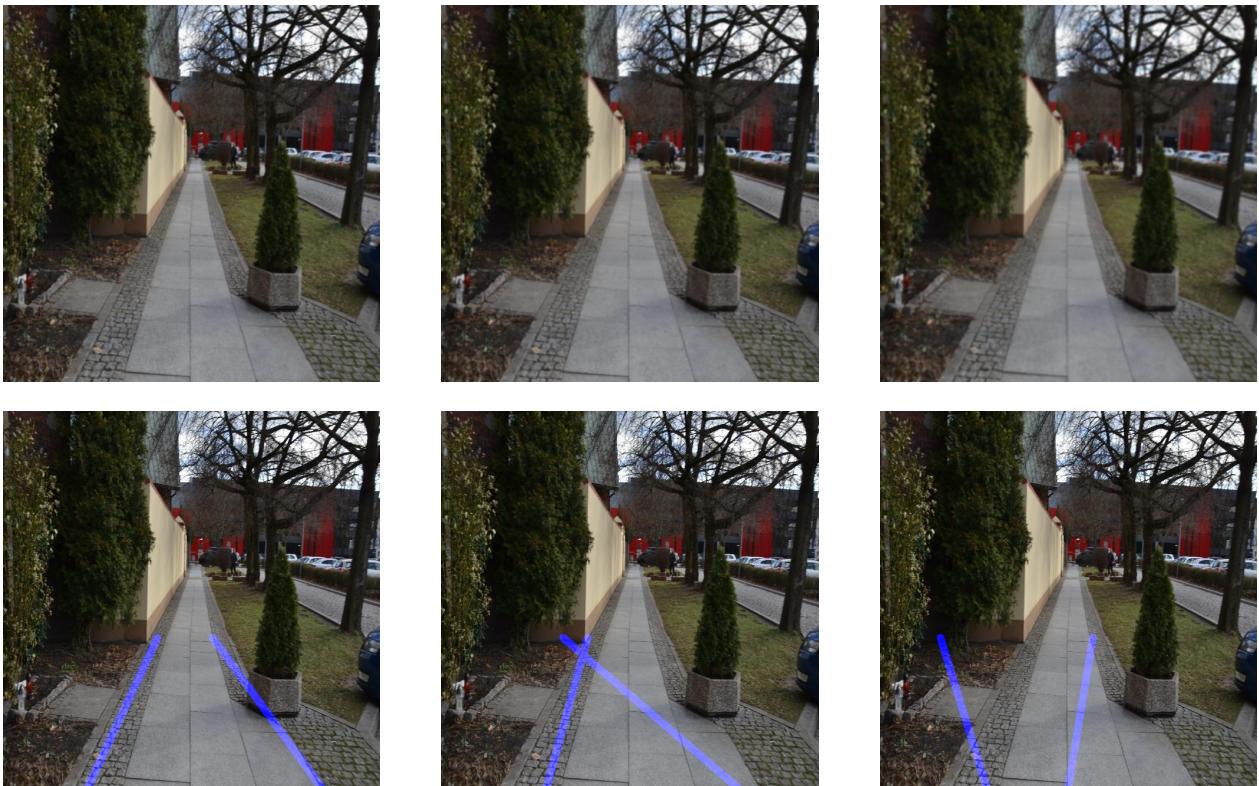
Rysunek 29: Obraz po lewej - algorytm bez konwersji do skali szarości - poprawny wynik. Obraz po prawej - algorytm z konwersją do skali szarości - poprawny wynik, ale lewa krawędź wykryta z mniejszą dokładnością



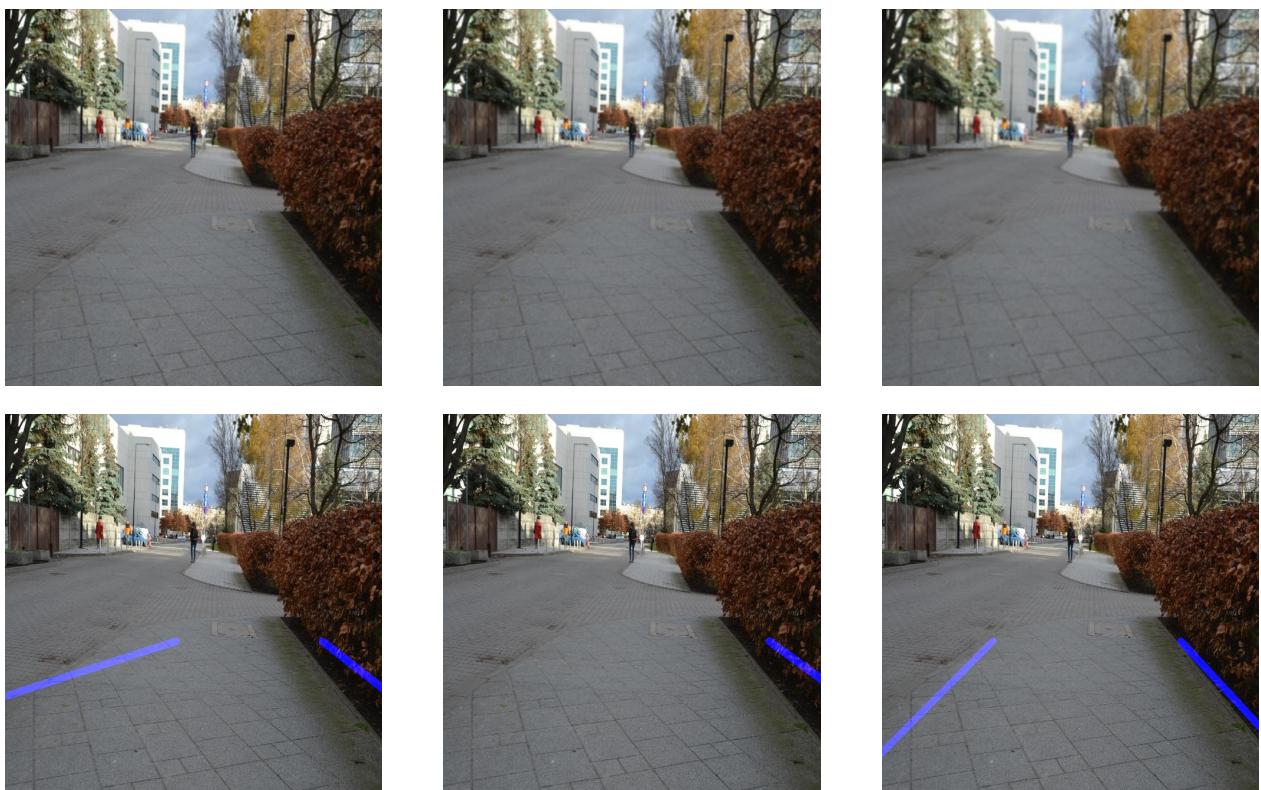
Rysunek 30: Obraz po lewej - algorytm bez konwersji do skali szarości - poprawny wynik. Obraz po prawej - algorytm z konwersją do skali szarości - niepoprawny wynik (wynik wskazał, że użytkownik nie znajduje się na chodniku)

#### 4.2.2 Wpływ parametrów filtru Gaussa

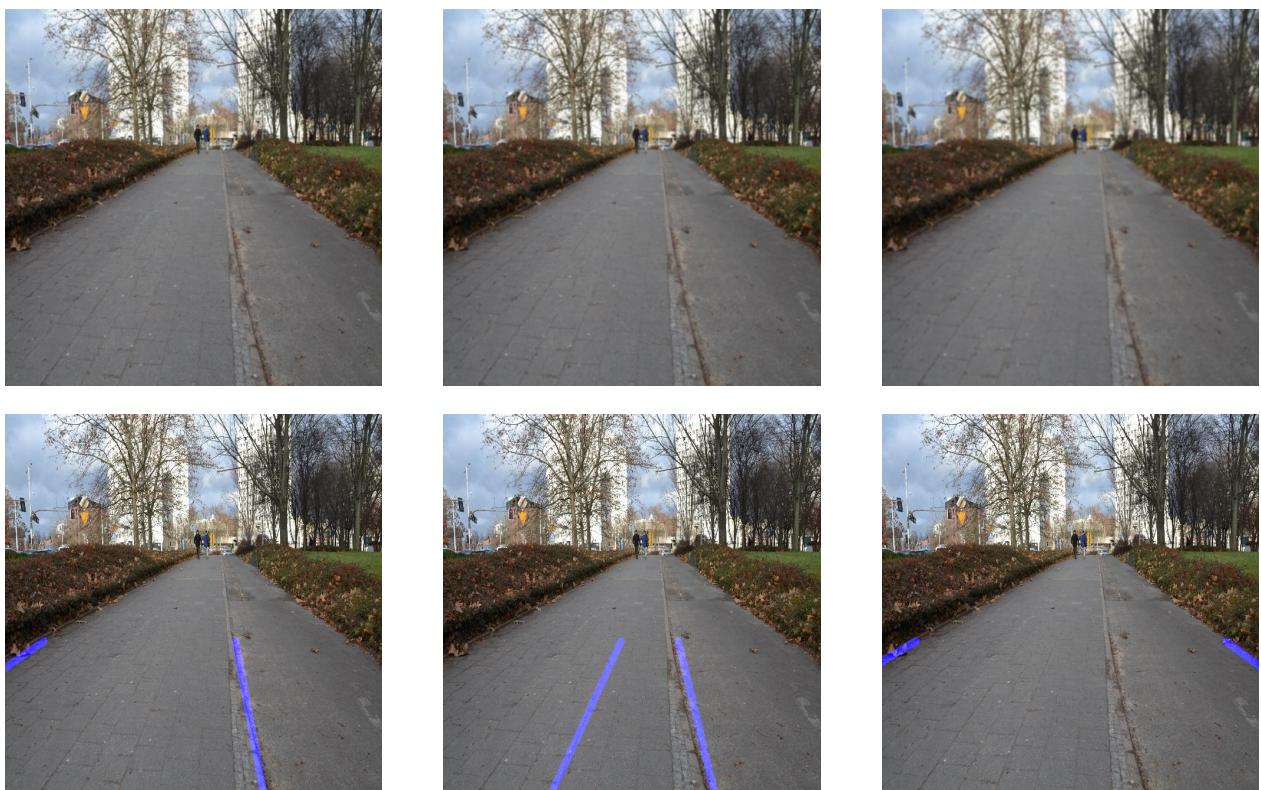
Badanie wykonano dla następujących wartości parametru `ksize`: (3,3), (5,5), (7,7). Na rysunkach 31, 32, 33 pokazano wpływ zmiany rozmiaru jądra filtru Gaussa dla trzech obrazów.



Rysunek 31: Wyniki dla pierwszego obrazu. Górnny rzząd: Obrazy rozmyte za pomocą filtru Gaussa o rozmiarach (3,3), (5,5), (7,7). Dolny rzząd: Obrazy wyjściowe przy zastosowaniu jądra filtru o rozmiarach (3,3), (5,5), (7,7). Metoda zadziałała najlepiej przy użyciu filtru o najmniejszym rozmiarze. W pozostałych przypadkach krawędzie chodnika nie zostały wystarczająco dokładnie oszacowane, wyznaczone linie proste nie spełniły nałożonych ograniczeń, przez co otrzymano rezultat niezgodny z rzeczywistością — program wskazał, że użytkownik nie znajduje się na chodniku.



Rysunek 32: Wyniki dla drugiego obrazu. Górnny rząd: Obrazy rozmyte za pomocą filtru Gaussa o rozmiarach (3,3), (5,5), (7,7). Dolny rząd: Obrazy wyjściowe przy zastosowaniu jądra filtru o rozmiarach (3,3), (5,5), (7,7). W przeciwnieństwie do pierwszego obrazu, w tym przypadku metoda najlepiej wykryła krawędzie chodnika, kiedy użyto najmocniejszego rozmycia. Osiągnięto też poprawny wynik dla najsłabszego rozmycia. Jedynie przy użyciu filtra o rozmiarze jądra (5,5) metoda zdołała wykryć tylko jedną krawędź, co uniemożliwia stwierdzenie, czy użytkownik znajduje się na chodniku.



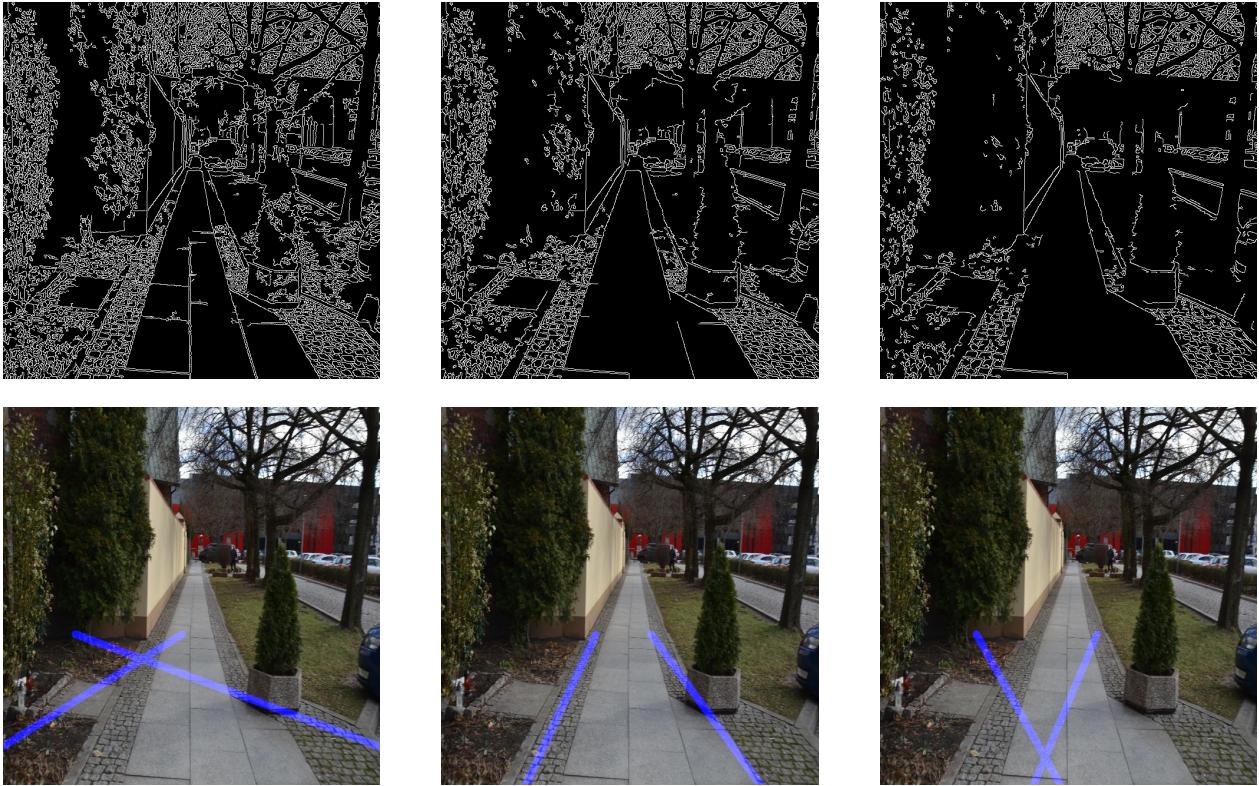
Rysunek 33: Wyniki dla trzeciego obrazu. Górný rząd: Obrazy rozmyte za pomocą filtru Gaussa o rozmiarach (3,3), (5,5), (7,7). Dolny rząd: Obrazy wyjściowe przy zastosowaniu jądra filtru o rozmiarach (3,3), (5,5), (7,7). Przy każdym dobiorze parametrów metoda wskazała poprawny wynik (użytkownik znajduje się na chodniku), ale krawędzie zostały wykryte w różnych miejscach. Przy najsłabszym rozmyciu wykryto krawędzie chodnika, a przy najmocniejszym krawędzie całego ciągu pieszo-rowerowego. W przypadku użycia jądra filtru o rozmiarze (5,5) wykryto prawą krawędź chodnika, a lewa krawędź nie została dobrze oszacowana. Jednak nie przeszkodziło to w spełnieniu nałożonych ograniczeń.

#### 4.2.3 Wpływ parametrów detektora Canny

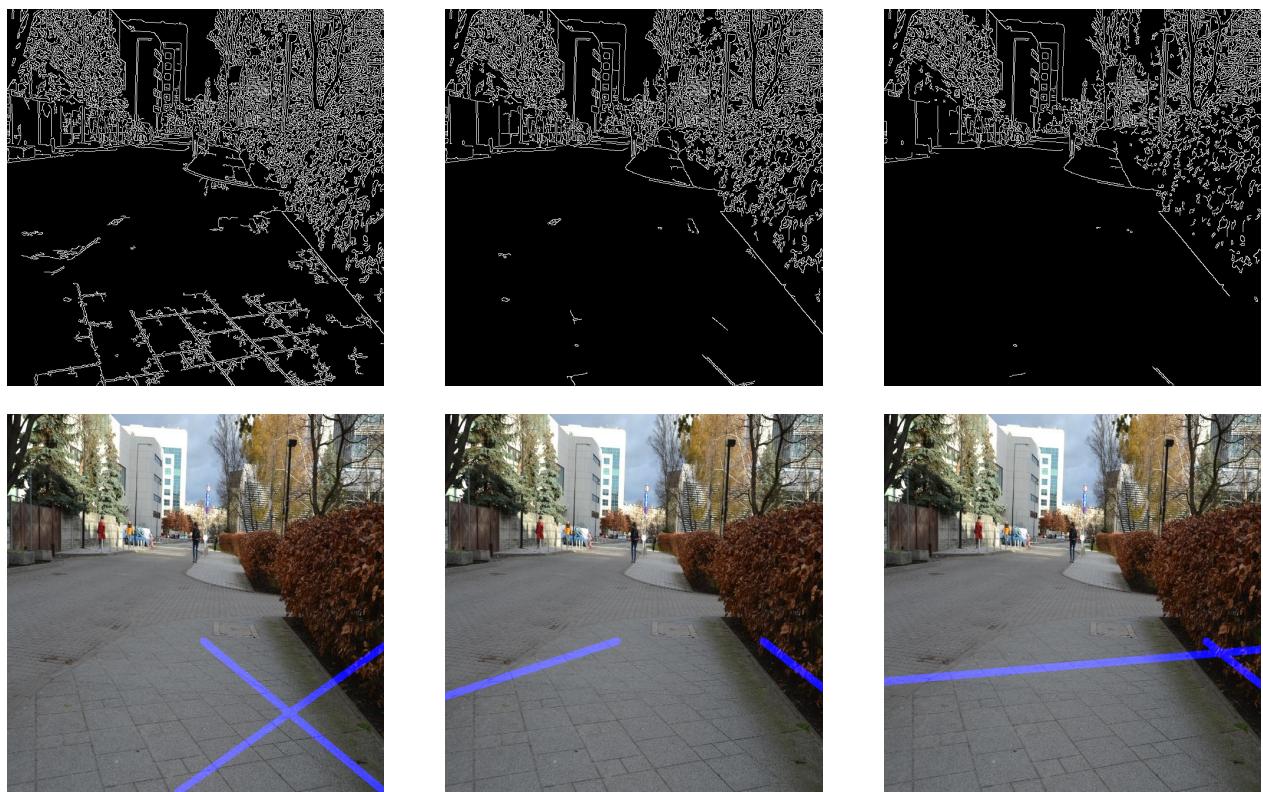
Sprawdzono trzy różne zakresy progów na trzech obrazach:

- `threshold1=20, threshold2=120,`
- `threshold1=50, threshold2=150,`
- `threshold1=80, threshold2=180.`

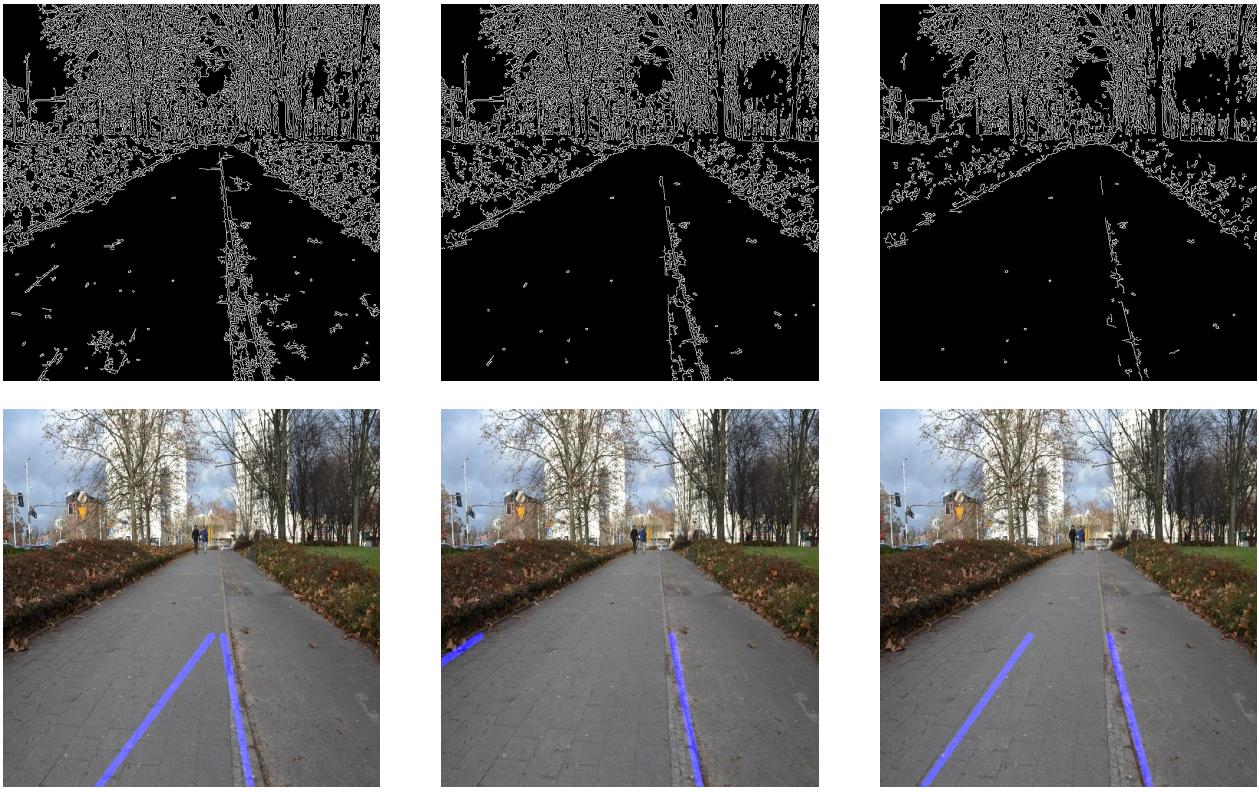
Rysunki 34, 35, 36 przedstawiają wyniki dla poszczególnych obrazów.



Rysunek 34: Wyniki dla pierwszego obrazu. Górný rząd od lewej do prawej: Krawędzie wykryte detektorami przy progach (20,120), (50,150), (80, 180). Dolny rząd od lewej do prawej - obrazy wyjściowe przy użyciu w detektorze Canny kolejnych par progów (20,120), (50,150), (80,180). Poprawny wynik uzyskano tylko przy użyciu wartości (50,150). Można zauważyć, jak krawędzie wykryte detektorem Canny wpływają na ostateczne położenie linii określających krawędzie chodnika.



Rysunek 35: Wyniki dla drugiego obrazu. Górný rzad od lewej do prawej: Krawędzie wykryte detektorami przy prograch (20,120), (50,150), (80, 180). Dolny rzad od lewej do prawej - obrazy wyjściowe przy użyciu w detektorze Canny kolejnych par progów (20,120), (50,150), (80,180). Tak jak dla pierwszego obrazu, poprawny wynik uzyskano tylko dla wartości (50,150).



Rysunek 36: Wyniki dla trzeciego obrazu. Górný rzad od lewej do prawej: Krawędzie wykryte detektorami przy progach  $(20,120)$ ,  $(50,150)$ ,  $(80, 180)$ . Dolny rzad od lewej do prawej - obrazy wyjściowe przy użyciu w detektorze Canny kolejnych par progów  $(20,120)$ ,  $(50,150)$ ,  $(80,180)$ . W każdym przypadku uzyskano poprawny wynik (użytkownik znajduje się na chodniku), lecz położenie wyznaczonych krawędzi chodnika było różne. Można zauważyć, jak zmiana parametrów detektora Canny wpływa na położenie linii wyznaczających krawędzie chodnika.

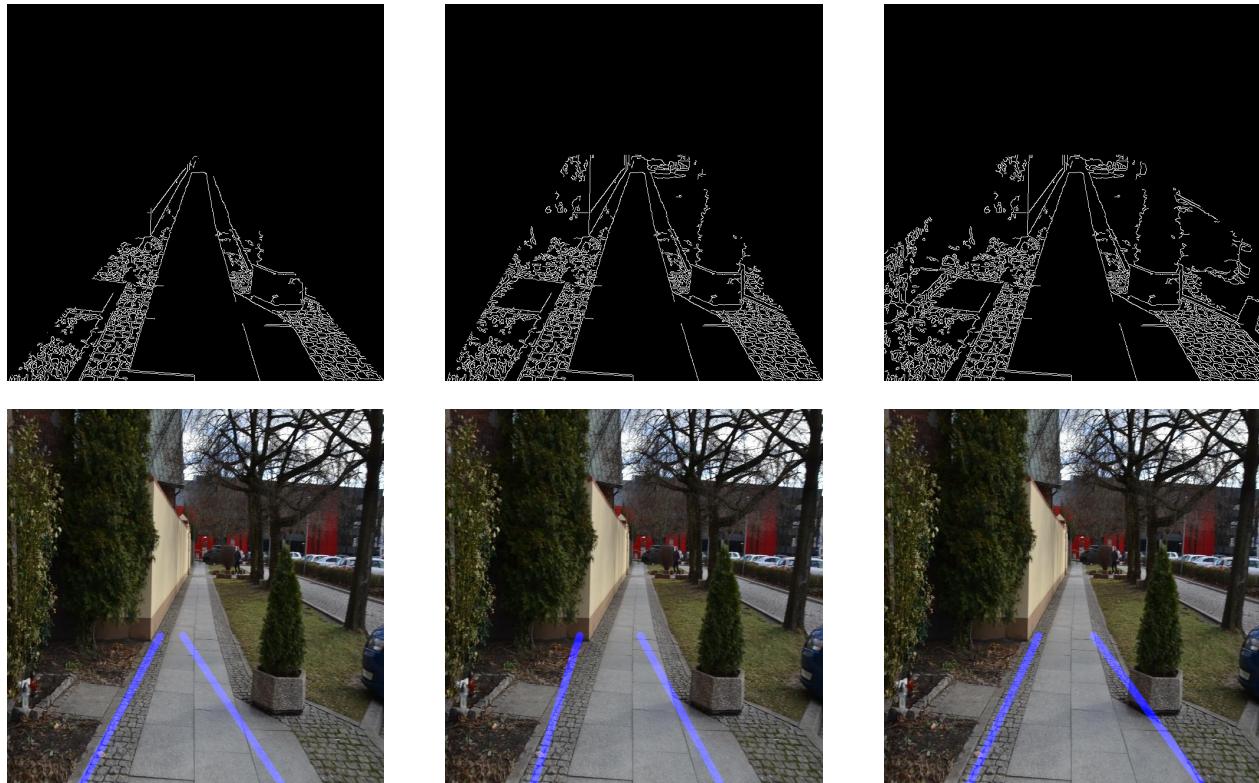
#### 4.2.4 Wpływ kształtu obszaru zainteresowań

Przetestowano wpływ kształtu obszaru zainteresowań na działanie metody. Wybrano trzy figury o następujących wierzchołkach ( $w$  — szerokość obrazu w pikselach,  $h$  — wysokość obrazu w pikselach):

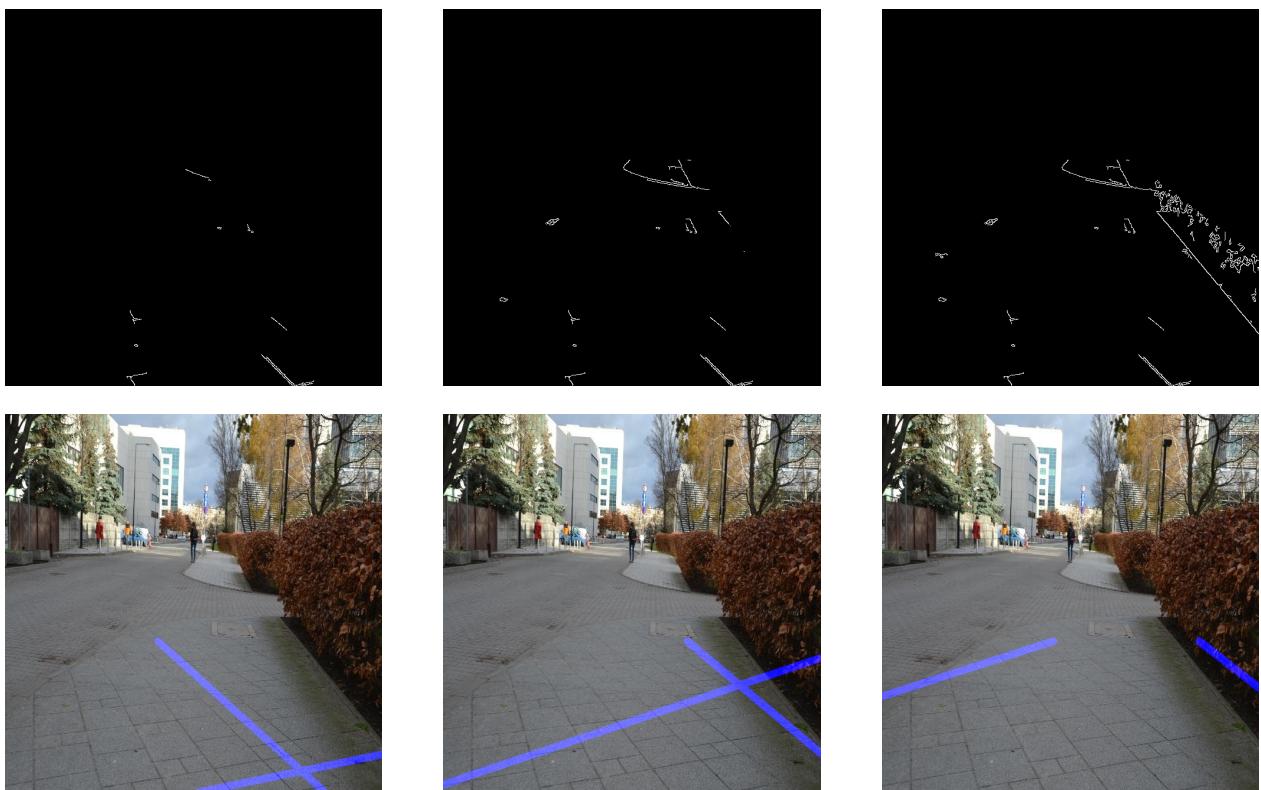
- trójkąt równoramienny:  $(0, h), (\frac{1}{2}w, \frac{2}{5}h), (w, h)$ ,
- trapez równoramienny  $(0, h), (\frac{1}{3}w, \frac{2}{5}h), (\frac{2}{3}w, \frac{2}{5}h), (w, h)$ ,

- sześciokąt o wierzchołkach:  $(0, h), (0, \frac{2}{3}h), (\frac{1}{3}w, \frac{2}{5}h), (\frac{2}{3}w, \frac{2}{5}h), (w, \frac{2}{3}h), (w, h)$ .

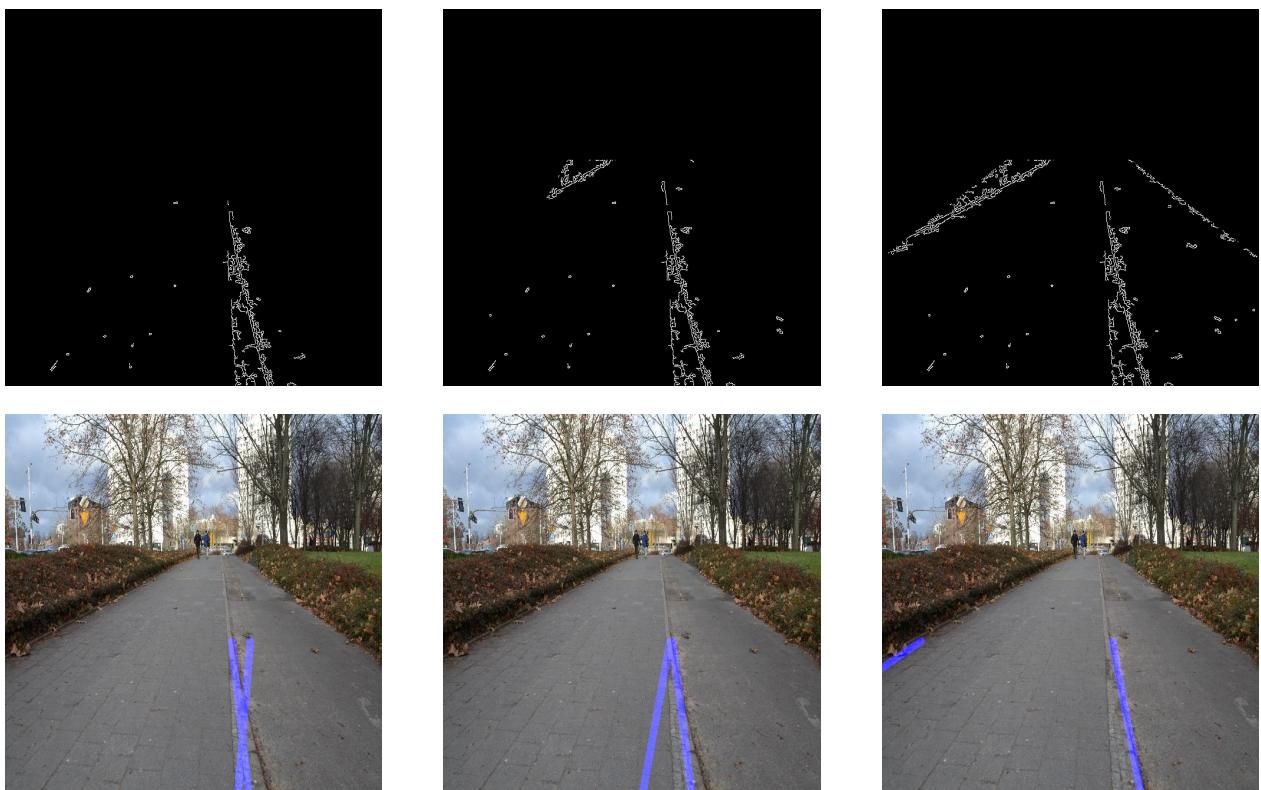
Na rysunkach 37, 38, 39 przedstawiono wpływ kształtu obszaru zainteresowań dla trzech obrazów. Można zauważać, że opisany wyżej sześciokąt jest najlepszym kształtem obszaru zainteresowań ze wszystkich sprawdzonych, ponieważ tylko przy jego użyciu udało się poprawnie wyznaczyć krawędzie chodnika na wszystkich obrazach.



Rysunek 37: Wyniki dla pierwszego obrazu. Górnego rzędu od lewej do prawej: obszary zainteresowań o różnych kształtach. Domyrzdu od lewej do prawej: efekty działania metody kolejno dla obszaru zainteresowań o kształcie trójkątnym, trapezoidalnym, i sześciokątnym. Można zauważać, że metoda zadziałała poprawnie w każdym przypadku, jednak wynik najbliższy rzeczywistości uzyskano przy zastosowaniu obszaru zainteresowań w kształcie sześciokąta.



Rysunek 38: Wyniki dla drugiego obrazu. Górný rzad od lewej do prawej: obszary zainteresowań o różnych kształtach. Dolny rzad od lewej do prawej: efekty działania metody kolejno dla obszaru zainteresowań o kształcie trójkątnym, trapezoidalnym, i sześciokątnym. Metoda dała wyniki zbliżone do rzeczywistości tylko w przypadku zastosowania obszaru zainteresowań w kształcie sześciokąta.



Rysunek 39: Wyniki dla drugiego obrazu. Górný rzad od lewej do prawej: obszary zainteresowań o różnych kształtach. Dolny rzad od lewej do prawej: efekty działania metody kolejno dla obszaru zainteresowań o kształcie trójkątnym, trapezoidalnym, i sześciokątnym. Metoda w każdym przypadku wskazała, że użytkownik znajduje się na chodniku. Należy jednak zauważyć, że wynik najbliższy rzeczywistości uzyskano przy zastosowaniu obszaru zainteresowań w kształcie sześciokąta.

#### 4.2.5 Wpływ parametrów transformacji Hougha

Przetestowano wpływ dwóch parametrów probabilistycznej transformacji Hougha — minimalną długość linii i maksymalną przerwę między punktami. Sprawdzono, czy metoda działa lepiej, kiedy te parametry są do siebie zbliżone, czy kiedy znacznie się od siebie różnią. Rozpatrzone trzy przypadki:

- `minLineLength=10, maxLineGap=100,`
- `minLineLength=30, maxLineGap=80,`

- `minLineLength=50, maxLineGap=60.`

Badanie przeprowadzono na trzech obrazach. Rezultaty pokazano na rysunkach 40, 41, 42. Należy zaznaczyć, że dokładne zbadanie wpływu konkretnych parametrów transformacji Hougha (również takich jak rozdzielczość kątowa) wymagałoby znacznie bardziej kompleksowych testów na wielu obrazach. W tej pracy skupiono się bardziej na działaniu całej metody wykrywania krawędzi chodnika, niż na badaniu samej transformacji Hougha. Wyniki pokazały, że dla obrazów testowych lepiej było stosować łagodniejsze ograniczenia.



Rysunek 40: Wyniki dla pierwszego obrazu. Od lewej do prawej umieszczone obrazy wyjściowe przy zastosowaniu transformacji Hougha o różnych wartościach parametrów, kolejno: (`minLineLength=10, maxLineGap=100`), (`minLineLength=30, maxLineGap=80`), (`minLineLength=50, maxLineGap=60`). W każdym przypadku wynikiem metody było orzeczenie, że użytkownik znajduje się na chodniku, jednak warto zauważyć, że krawędzie zostały najlepiej wykryte przy najlagodniejszych ograniczeniach (10, 100).



Rysunek 41: Wyniki dla drugiego obrazu. Od lewej do prawej umieszczone obrazy wyjściowe przy zastosowaniu transformacji Hougha o różnych wartościach parametrów, kolejno: (`minLineLength=10, maxLineGap=100`), (`minLineLength=30, maxLineGap=80`), (`minLineLength=50, maxLineGap=60`). Metoda dała dobre wyniki w dwóch przypadkach. W trzecim przypadku, gdzie wartości obydwu testowanych parametrów były najbardziej zbliżone, metoda dała niepoprawne wyniki.



Rysunek 42: Wyniki dla trzeciego obrazu. Od lewej do prawej umieszczone obrazy wyjściowe przy zastosowaniu transformacji Hougha o różnych wartościach parametrów, kolejno: (`minLineLength=10, maxLineGap=100`), (`minLineLength=30, maxLineGap=80`), (`minLineLength=50, maxLineGap=60`). Metoda dała we wszystkich przypadkach bardzo zbliżone rezultaty.

## 5 Aplikacja mobilna

Aplikacja została stworzona w środowisku Android Studio i napisana za pomocą języka Kotlin. Należy zaznaczyć, że aplikacja nie analizuje wszystkich klatek zebranych z kamery po kolei. Analizuje bieżącą klatkę, a następnie pobiera (za pomocą API) najnowszą klatkę z kamery, aby uniknąć znaczących opóźnień w działaniu.

### 5.1 Implementacja metod

#### 5.1.1 Detekcja obiektów i szacowanie odległości

Za szacowanie głębi w aplikacji mobilnej odpowiada model MiDas w formacie TensorFlow Lite, który jest dostępny na platformie Kaggle. Model przeznaczony do detekcji obiektów został oparty na YOLOv8 i wytrenowany na zbiorze danych, co opisano wcześniej. Następnie został przekonwertowany do formatu TensorFlow Lite. Implementację użycia tych modeli oparto na dostępnych przykładach [37][38], a następnie połączono w spójny system tak, aby możliwe było wykrywanie obiektów z jednoczesnym szacowaniem odległości i położenia. Należy również zaznaczyć, że zdecydowano się na prezentowanie użytkownikowi jedynie trzech najbliższych obiektów. Zwiększa to przejrzystość rozwiązania, przy jednoczesnym wykrywaniu najbliższych przeszkód, czy przejść dla pieszych.

W ostatecznej wersji aplikacji wprowadzono funkcjonalność symulującą naprowadzanie użytkownika na przejście dla pieszych. Jeżeli zostanie wykryte przejście dla pieszych w średniej lub bliskiej odległości od kamery, to na dole ekranu pojawi się informacja o pobliskim przejściu dla pieszych i po której stronie się ono znajduje.

#### 5.1.2 Utrzymywanie użytkownika na chodniku

Metodę zaimplementowano bezpośrednio w oparciu o implementację w Pythonie, którą opisano wcześniej. Jedyną różnicą jest dodatkowe ograniczenie, które polega na ignorowaniu linii poziomych i pionowych przy wyznaczaniu krawędzi chodnika. Takie linie często oznaczają obecność słupów lub pasów na drodze, zatem zakłócają wyznaczanie rzeczywistych krawędzi. Proste, dla których wartość bezwzględna współczynnika kierunkowego jest wysoka, to proste zbliżone do prostych pionowych. Natomiast proste zbliżone do prostych poziomych

mają współczynnik kierunkowy, którego wartość bezwzględna jest bliska zeru. Ograniczając dozwolone wartości współczynnika kierunkowego można odrzucić proste zbliżone do prostych pionowych i poziomych. Mając wyznaczony współczynnik kierunkowy  $a$  nałożono ograniczenia:

$$a \in \left(-15, -\frac{1}{2}\right) \cap \left(\frac{1}{2}, 15\right). \quad (12)$$

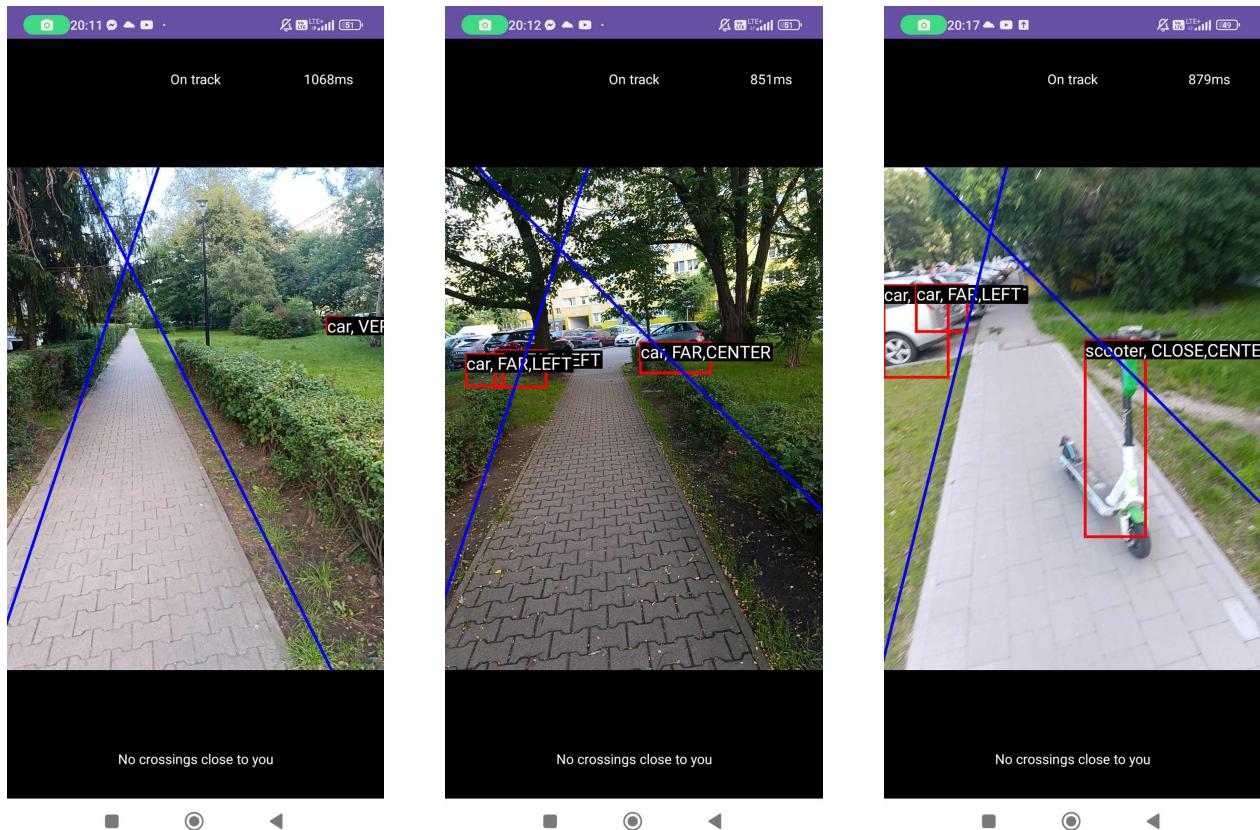
## 5.2 Testowanie aplikacji

Aplikację mobilną testowano wielokrotnie w warunkach rzeczywistych, to znaczy w trakcie poruszania się po chodniku. Ustalono, że zadanie detekcji obiektów z szacowaniem odległości jest wykonywane ze zbliżoną efektywnością do tej, którą obserwowano w testach programu napisanego w języku Python. W przypadku wykrywania krawędzi chodnika sprawdzano, czy efekty działania są zgodne z rzeczywistością przy danym zestawie parametrów, a następnie korygowano je według potrzeby. Przykładowy zestaw parametrów, który daje zadowalające efekty:

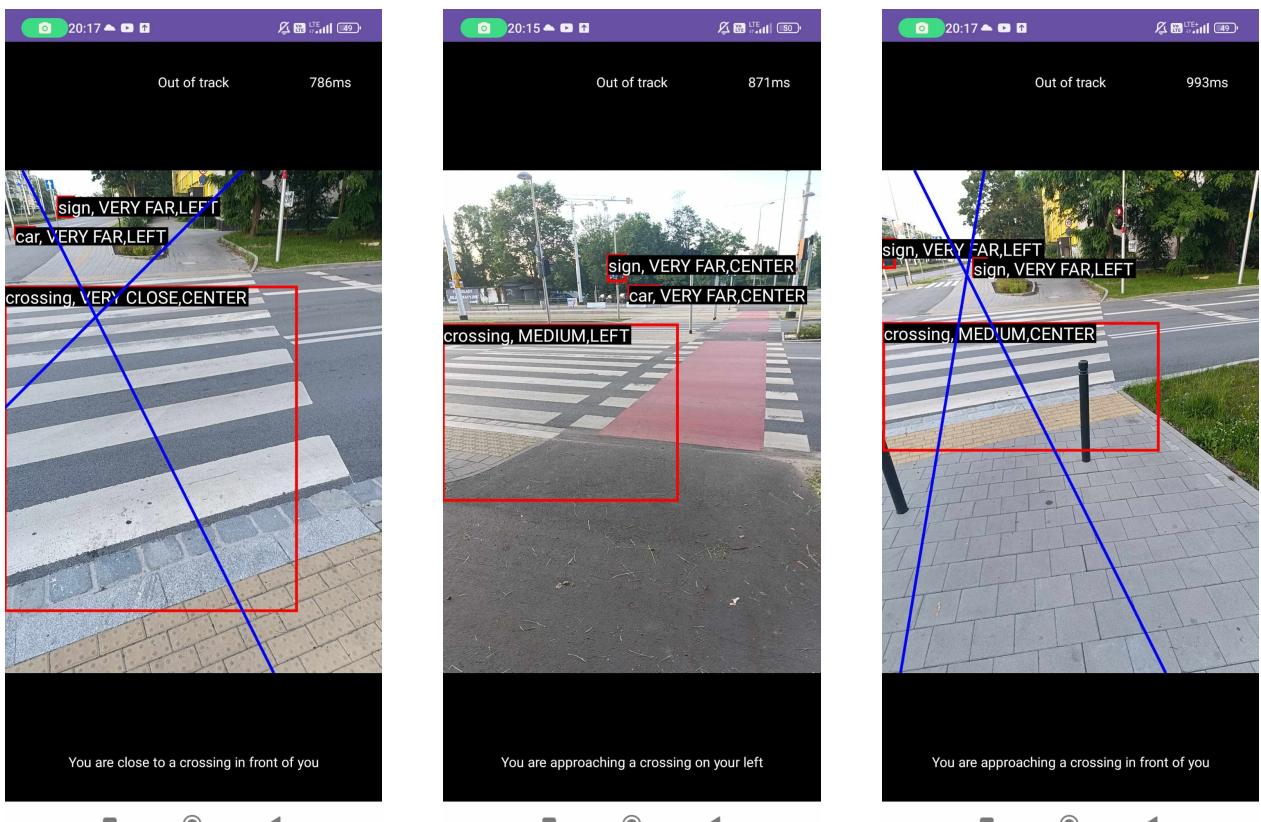
- Filtr Gaussa: `ksize=(5,5), sigma=0,`
- Detektor Canny: `threshold1=50, threshold2=150,`
- Obszar zainteresowań: sześciokąt o wierzchołkach  $(0, h), (0, \frac{3}{5}h), (\frac{1}{3}w, \frac{2}{5}h), (\frac{2}{3}w, \frac{2}{5}h), (w, \frac{3}{5}h), (w, h)$ .  $w$  — szerokość obrazu w pikselach,  $h$  — wysokość obrazu w pikselach,
- Transformacja Hougha:
  - `rho=1,`
  - `theta=π/180, threshold=20,`
  - `minLineLength=50,`
  - `maxLineGap=40.`

Największą różnicę w stosunku do implementacji w Pythonie zaobserwowano w wartościach parametrów transformacji Hougha — w aplikacji mobilnej należało wprowadzić bardziej restrykcyjne ograniczenia, aby uzyskać dobre efekty. Na rysunkach 43, 44, 45 przedstawiono rzuty ekranów z działania aplikacji, gdzie:

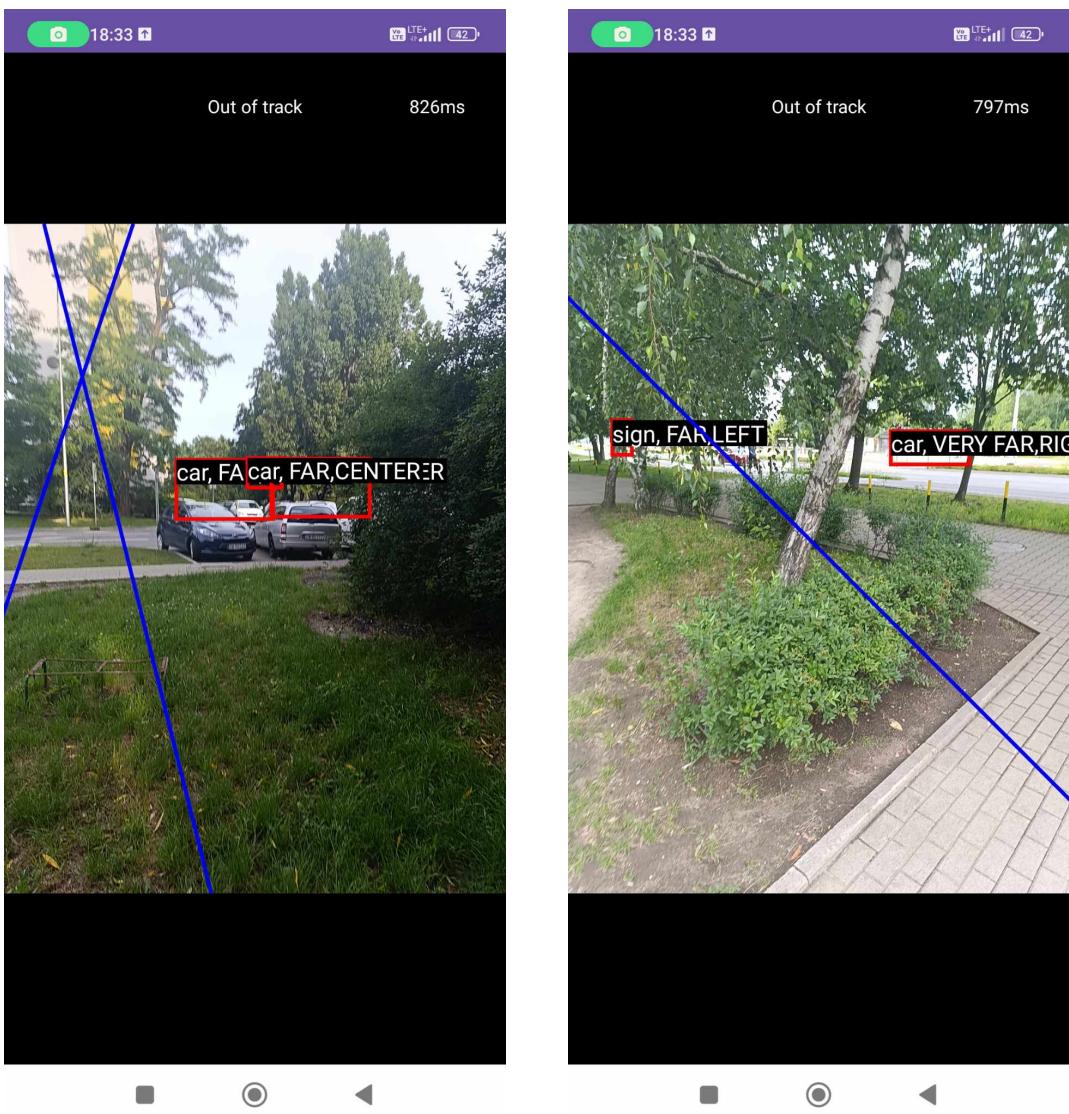
- niebieskie linie oznaczają wykryte krawędzie chodnika,
- napis "On track" na górze ekranu oznacza, że użytkownik porusza się po chodniku, a napis "Out of track", że jest poza chodnikiem,
- w prawym górnym rogu można odczytać całkowity czas analizy poprzedniej klatki.



Rysunek 43: Przykłady poprawnego wykrycia krawędzi chodnika i obiektów znajdujących się w kadrze kamery.



Rysunek 44: Przykład wykrycia przejścia dla pieszych i innych obiektów. Na dole ekranu widać komunikat o położeniu przejścia dla pieszych.



Rysunek 45: Po skierowaniu kamery poza chodnik aplikacja wskazuje, że użytkownik nie znajduje się na chodniku.

## 6 Podsumowanie

W pracy zaimplementowano i przetestowano metody sztucznej inteligencji, które mogą być pomocne przy budowie aplikacji wspomagającej poruszanie się po chodniku osobom niewidomym. Skupiono się głównie na dwóch zagadnieniach: na detekcji obiektów z szacowaniem odległości oraz na wykrywaniu krawędzi pasa ruchu.

Detekcja obiektów została wykonana za pomocą modelu, który został oparty na architekturze YOLOv8 i wytrenowany na dostarczonym zbiorze danych. Przetestowano dwa modele szacowania głębi: MiDaS oraz Monodepth2. Następnie użyto modelu MiDaS przy budowie prototypu aplikacji mobilnej.

Wykrywanie krawędzi oparto na klasycznych metodach przetwarzania obrazu i zbadano wpływ wartości parametrów na efekt działania metody. Następnie zbudowano aplikację mobilną z wykorzystaniem powyższych metod i przetestowano ją w warunkach rzeczywistych, to jest w trakcie spaceru chodnikiem. Metody zaimplementowane i sprawdzone w tej pracy dały zadowalające wyniki, jednakże dla efektywnego działania w warunkach rzeczywistych należałoby przeprowadzić więcej badań. W przypadku detekcji obiektów można by zoptymalizować proces trenowania modelu, sprawdzić inne popularne narzędzia, czy rozszerzyć zbiór danych treningowych o nowe obrazy. Istnieje również możliwość rozpatrzenia innych modeli szacowania głębi. Metoda wykrywania krawędzi chodnika jest bardzo wrażliwa na wartości parametrów w poszczególnych krokach metody, zatem dla uzyskania lepszych wyników można by przeprowadzić poszukiwanie optimów lokalnych dla danego zbioru danych. Należałoby dobrać sposób znajdowania najlepszych zestawów parametrów tak, aby metoda wykrywania krawędzi dawała zadowalające wyniki dla jak największej liczby obrazów w konkretnym zbiorze danych.

Aplikacja mobilna działa dobrze, jeśli traktować ją jako prototyp. W ramach dalszej pracy można by dodać komunikaty audio dla użytkownika, ulepszyć interfejs graficzny, zwiększyć równoległość wykonywanych zadań, czy rozszerzyć o dodatkowe funkcjonalności.

## 7 Podziękowania

Dziękuję Panu Kamilowi Bajusowi za udostępnienie zbioru danych.

## Literatura

- [1] Aryaman Sharda. Image Filters: Gaussian Filter. <https://aryamansharda.medium.com/image-filters-gaussian-blur-eb36db6781b1>. Dostęp: 2024-05-05.

- [2] Siddharth Misra, Yaokun Wu, Jiabo He. *Machine Learning For Subsurface Characterization*. Elsevier Inc., 2020.
- [3] Sofiane Sahir. Canny Edge Detection Step by Step in Python — Computer Vision. <https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123>. Dostęp: 2024-05-06.
- [4] OpenCV Documentation. Canny Edge Detection. [https://docs.opencv.org/4.x/d22/tutorial\\_py\\_canny.html](https://docs.opencv.org/4.x/d22/tutorial_py_canny.html). Dostęp: 2024-05-06.
- [5] Allam Shehata Hassanein, Sherien Mohammad, Mohamed Sameer, and Mohammad Ehab Ragab. A survey on Hough transform, theory, techniques and applications. *arXiv preprint arXiv:1502.02160*, 2015.
- [6] Sylwia Sikorska. Application of the Hough transform to digital image analysis. *Czasopismo Techniczne. Mechanika*, 108(4-M/2):457–463, 2011.
- [7] Priyanka Mukhopadhyay and Bidyut B Chaudhuri. A survey of Hough transform. *Pattern Recognition*, 48(3):993–1010, 2015.
- [8] Jiri Matas and Charles Galambos. Progressive probabilistic Hough transform. In *Proceedings of the British Machine Vision Conference, Southampton, UK*, pages 14–17, 1998.
- [9] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European conference on information retrieval*, pages 345–359. Springer, 2005.
- [10] Jae Moon Lee et al. Real distance measurement using object detection of artificial intelligence. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12(6):557–563, 2021.
- [11] Adrian Rosebrock. Find distance from camera to object/marker using Python and OpenCV. <https://pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/>. Dostęp: 2024-01-30.

- [12] Bojan Strbac, Marko Gostovic, Zeljko Lukac, and Dragan Samardzija. YOLO multi-camera object detection and distance estimation. In *2020 Zooming Innovation in Consumer Technologies Conference (ZINC)*, pages 26–30. IEEE, 2020.
- [13] Marek Vajgl, Petr Hurtik, and Tomáš Nejezchleba. Dist-YOLO: Fast object detection with distance estimation. *Applied Sciences*, 12(3):1354, 2022.
- [14] Marek Vajgl. YOLO with distance. <https://gitlab.com/EnginCZ/yolo-with-distance>. Dostęp: 2024-04-29.
- [15] Shivam Khandelval. Distance estimation of various objects present in an image from camera when they were captured using depth image and object detection mechanism. <https://khandewalshivam.medium.com/distance-estimation-of-various-objects-present-in-an-image-from-camera-when-they-were-captured-64453847a3a1>. Dostęp: 2024-01-30.
- [16] Swati Shilaskar, Mugdha Dhopade, Janhvi Godle, and Shripad Bhatlawande. Machine learning-based pavement detection for visually impaired people. In *International Conference on Communications and Cyber Physical Engineering 2018*, pages 383–395. Springer, 2023.
- [17] Marcelo C Ghilardi, Rafael CO Macedo, and Isabel H Manssour. A new approach for automatic detection of tactile paving surfaces in sidewalks. *Procedia computer science*, 80:662–672, 2016.
- [18] Xu Jie, Wang Xiaochi, and Fang Zhigang. Research and implementation of blind sidewalk detection in portable eta system. In *2010 International Forum on Information Technology and Applications*, volume 2, pages 431–434. IEEE, 2010.
- [19] Byung-Seok Woo, Sung-Min Yang, and Kang-Hyun Jo. Brick path recognition using image shape pattern and texture feature. *Journal of Korea Multimedia Society*, 15(4):472–484, 2012.

- [20] Priyanka Kumari. Real-Time Lane Detection for Self-Driving Cars using OpenCV. <https://www.labellerr.com/blog/real-time-lane-detection-for-self-driving-cars-using-opencv/>. Dostęp: 2024-04-29.
- [21] Dhruv Pandey. Lane detection for a self-driving car using OpenCV. <https://medium.com/analytics-vidhya/lane-detection-for-a-self-driving-car-using-opencv-e2aa95105b89>. Dostęp: 2024-04-29.
- [22] Armin Masoumian, David GF Marei, Saddam Abdulwahab, Julian Cristiano, Domenec Puig, and Hatem A Rashwan. Absolute distance prediction based on deep learning object detection and monocular depth estimation models. In *CCIA*, pages 325–334, 2021.
- [23] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [24] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.
- [25] Jakub Szymkowiak, Marek Bazan, Krzysztof Halawa, and Tomasz Janiczek. Detection of objects dangerous for the operation of mining machines. In *International Conference on Computational Science*, pages 128–139. Springer, 2023.
- [26] Juan Terven, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. A comprehensive review of YOLO architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS. *Machine Learning and Knowledge Extraction*, 5(4):1680–1716, 2023.
- [27] Muhammad Rizwan Munawar, Glenn Jocher, Ayush Chaurasia. YOLO: A Brief History. <https://docs.ultralytics.com/#yolo-a-brief-history>. Dostęp: 2024-05-15.
- [28] Jacob Solawetz. What is YOLOv8? The Ultimate Guide. <https://blog.roboflow.com/whats-new-in-yolov8/>. Dostęp: 2024-05-15.

- [29] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-IoU loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 12993–13000, 2020.
- [30] Xiang Li, Wenhai Wang, Lijun Wu, Shuo Chen, Xiaolin Hu, Jun Li, Jinhui Tang, and Jian Yang. Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection. *Advances in Neural Information Processing Systems*, 33:21002–21012, 2020.
- [31] Honggang Wang. Detection of personal protective equipment (PPE) using an anchor free-convolutional neural network. *International Journal of Advanced Computer Science & Applications*, 15(2), 2024.
- [32] Katrin Lasinger, René Ranftl, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *arXiv preprint arXiv:1907.01341*, 2019.
- [33] Intelligent Systems Lab. MiDaS. <https://github.com/isl-org/MiDaS>. Dostęp: 2024-05-21.
- [34] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3828–3838, 2019.
- [35] OpenCV. OpenCV Documentation. <https://docs.opencv.org/4.x/index.html>. Dostęp: 2024-05-13.
- [36] NumPy. NumPy Documentation. <https://numpy.org/doc/stable/index.html>. Dostęp: 2024-05-13.
- [37] Surendra Maran. YOLOv8 TFLite Object Detector. <https://github.com/surendramaran/YOLOv8-TfLite-Object-Detector>. Dostęp: 2024-06-02.
- [38] Shubham Panchal. Realtime MiDaS Depth Estimation - Android. [https://github.com/shubham0204/Realtime\\_MiDaS\\_Depth\\_Estimation\\_Android](https://github.com/shubham0204/Realtime_MiDaS_Depth_Estimation_Android). Dostęp: 2024-06-02.