



Politechnika Wrocławska

Wydział Informatyki i Zarządzania

Kierunek studiów: Informatyka

Praca dyplomowa – inżynierska

APLIKACJA MOBILNA DLA NIEWIDOMYCH SŁUŻĄCA DO NAWIGACJI PO MIEŚCIE

Kamil Moszczyc

słowa kluczowe:
aplikacja mobilna, Android, Kotlin,
OpenCV, detekcja przeszkód, lokalizacja,
nawigacja

krótkie streszczenie:

W pracy zawarto projekt i implementację aplikacji mobilnej wspierającej osoby niewidome i słabo widzące w poruszaniu się po mieście. Aplikacja m.in.: dokonuje detekcji przeszkód i krawędzi chodnika, informuje o obecnej lokalizacji oraz kierunku geograficznym.

Opiekun pracy dyplomowej	dr inż. Aleksander Mariański		
	Tytuł/stopień naukowy/imię i nazwisko		
Ostateczna ocena za pracę dyplomową			
Przewodniczący Komisji egzaminu dyplomowego			
	Tytuł/stopień naukowy/imię i nazwisko	ocena	podpis

*Do celów archiwalnych pracę dyplomową zakwalifikowano do:**

- a) kategorii A (akta wieczyste)
- b) kategorii BE 50 (po 50 latach podlegające ekspertyzie)

* niepotrzebne skreślić

pieczętka wydziałowa

Wrocław, rok 2022

Streszczenie

Celem tej pracy jest projekt i implementacja aplikacji mobilnej dla niewidomych służącej do nawigacji po mieście. W ramach pracy przedstawiono problem niewidomych, jakim niewątpliwie jest zachowanie orientacji w mieście i omijanie przeszkód, jak i sama nawigacja. W celu rozwiązania tych problemów wykonano przegląd istniejących rozwiązań, by nie powielać ich, tylko zaimplementować rozwiązanie, którego nie ma aktualnie na rynku, a tym niewątpliwie jest detekcja przeszkód za pomocą samego przetwarzania obrazu.

W ramach tej pracy dokonano projektu aplikacji zawierającego diagram przypadków użycia, scenariusze przypadków użycia oraz diagram pakietów. W ramach tej pracy także zaimplementowano aplikację mobilną na urządzenia Android, w wersji od 6.0 do 10.0. Detekcja przeszkód w tej pracy opiera się na założeniu, że przeszkody mają długie, proste krawędzie i na tej podstawie przeszkody są wykrywane. Po wykryciu przeszkody w zależności od jej położenia na obrazie aplikacja wydaje dźwięk o określonej głośności w stereo. Algorytm detekcji przeszkód nie wykrywa przeszkód o krzywych krawędziach oraz błędnie wykrywa niektóre cienie jako przeszkody. Aplikacja zawiera takie funkcje jak: określenie aktualnej lokalizacji, kompas, data i godzina, oraz możliwość skorzystania z tychże funkcji za pomocą komend głosowych. Za pomocą tej aplikacji osoba niewidoma bądź niedowidząca może włączyć nawigację Google Maps, podając miejsce docelowe. Praca uwzględnia także testy wydajnościowe i akceptacyjne oraz wnioski końcowe i kierunki dalszych prac.

Abstract

The topic of this thesis is a project and implementation of a mobile application for blind to help them navigate through the city. In this thesis the main problem of blind people was presented which is preserving the orientation in the city and obstacle avoidance, and the navigation itself. In the pursuit of solving these problems a review of currently existing solutions was done. In order to not copy existing solutions but to implement a new solution, obstacle detection relies solely on image processing.

As part of the project: use case diagram, use case scenarios and package diagram were produced. Also, as part of this thesis mobile application was implemented,

available for Android devices 6.0 to 10.0. Obstacle detection in this thesis relies on the assumption that obstacles have long, straight borders, and on that basis obstacles are detected. After detection of an obstacle, depending on its position on the image, the application produces sound with according volume in stereo. The Obstacle detection algorithm does not detect obstacles with curved borders and it wrongly detects some shadows as obstacles. The Application offers functions such as: current location, compass, hour and date and possibility of accessing all of those functions by voice commands. With this application a blind or low-vision person can open external Google Maps navigation by providing target location. Thesis contains also performance and acceptance tests and final conclusions and directions of further work.

Spis treści

Streszczenie	1
1. Słownik pojęć	5
2. Wprowadzenie	7
3. Przegląd rozwiązań	9
3.1. Przegląd literatury o detekcji krawędzi chodnika i przeszkód	9
3.2. Przegląd aplikacji używanych przez niewidomych	10
3.3. Inne rozwiązania	14
3.4. Podsumowanie	14
4. Założenia projektowe	16
4.1. Przedmiot projektu	16
4.2. Wymagania funkcjonalne	16
4.3. Wymagania niefunkcjonalne	17
4.4. Podsumowanie	17
5. Projekt aplikacji	18
5.1. Historyjki użytkownika	18
5.2. Diagram przypadków użycia	19
5.3. Scenariusze przypadków użycia	21
5.4. Diagram pakietów	26
5.5. Wzorce projektowe	27
5.6. Podsumowanie	27
6. Implementacja aplikacji mobilnej	28
6.1. Środowisko, biblioteki	28
6.2. Struktura projektu	30
6.3. Detekcja przeszkód	31
6.3.1. Algorytm wykrywania krawędzi	31
6.3.2. Efekty zastosowania rozszerzania jasnych pikseli (dilation)	32
6.3.3. Wizualizacja algorytmu detekcji krawędzi	34
6.3.4. Rozmiar pola detekcji	37
6.3.5. Przykłady detekcji przeszkód	40
6.3.6. Detekcja przeszkód - noc	46
6.4. Przykład użycia nawigacji	48
6.5. Przykłady użycia funkcji lokalizacji	49
6.6. Przykłady użycia kompasu	49

6.7. Przykłady użycia funkcji daty i godziny	50
6.8. Interfejs ustawień aplikacji	51
6.9. Podsumowanie	53
7. Testowanie aplikacji	54
7.1. Testowanie wydajności	54
7.2. Testy akceptacyjne	57
7.3. Podsumowanie:	61
8. Zakończenie	62
Bibliografia	64
Spis rysunków	66
Spis tabel	68

1. Słownik pojęć

Przed przeczytaniem pracy należy wpierw zapoznać się z podstawowymi terminami, definicjami, jakie przedstawiono w tabeli 1.1.

Tabela 1.1. Słownik pojęć (źródło: opracowanie własne)

Pojęcie	Opis
API	(ang. application programming interface) - interfejs programowania aplikacji, czyli w uproszczeniu sposób komunikacji pomiędzy programami, np. Google Maps API, Depth API.
Canny	Sposób detekcji krawędzi opracowany przez Johna f. Canny w 1985 roku [14]. Działanie polega na konwolucji obrazu wejściowego za pomocą filtra Gaussa 5x5, by odpowiednio wygładzić i odszumić obraz. Następnie przeprowadzane są 4 konwolucje z 4 filtrami to detekcji krawędzi poziomych i pionowych [13]. W bibliotece OpenCV stosuje się 2 progi gradientu uznawanego za krawędź. Gradienty poniżej minimalnego progu są odrzucone. Gradienty powyżej maksymalnego progu są uznawane za krawędzie. Gradienty pomiędzy minimalnym i maksymalnym progiem mogą być krawędziami, jeśli w sąsiedztwie znajdują się krawędzie.
Depth API	API z biblioteki ARCore od Google'a [1], umożliwiająca stosunkowo prosty dostęp do zdjęć z mapą głębokości obrazu.
Dilation	Operacja przetwarzania obrazu polegająca na rozszerzaniu jasnych pikseli względem ciemniejszych pikseli w ich sąsiedztwie. Operacją przeciwną do dilation (rozszerzanie) jest erosion (erozja), która rozszerza ciemne piksele kosztem jasnych. Z pomocą operacji rozszerzania można skutecznie usunąć ciemne krawędzie pomiędzy kostkami brukowymi, nie tracąc istotnych informacji z obrazu.
Osoba niewidoma	Osoba całkowicie pozbawiona zmysłu wzroku, jednakże w tej w pracy w ramach uproszczeń to pojęcie obejmuje też osoby niedowidzące, ze znacznymi wadami wzroku.
Przetwarzanie obrazu	(Cyfrowe przetwarzanie obrazu) - dziedzina grafiki komputerowej zajmująca się reprezentacją oraz transformacją obrazów posługując się takimi operacjami jak: filtracja, binaryzacja, transformacja pomiędzy przestrzeniami barw (np. RGB do HSV, BGR etc.) [2]
Punkty zainteresowań	(POI - points of interest) - miejsca publiczne zaznaczane na mapach np. sklepy, przystanki, restauracje, zabytki etc.

Tensorflow Lite	Biblioteka do uczenia maszynowego skierowana na urządzenia Android i iOS. Używana głównie do klasyfikacji obrazu. Jako kierunek dalszych prac mogłaby zostać wykorzystana do klasyfikacji obiektów lub do estymacji głębokości obrazu [20].
Transformacja Hougha	Metodyka wykrywania regularnych kształtów w grafice komputerowej, opracowana przez Paula Hougha w 1962 roku [21]. W tej pracy transformacja Hougha będzie wykorzystywana do wykrywania prostych linii, na obrazie po detekcji krawędzi Canny.

2. Wprowadzenie

Na świecie żyje 285 milionów osób ze znacznie pogorszonym widzeniem, z czego 39 milionów jest całkowicie ślepe. Brakuje rozwiązań, które by pomagały takim osobom poruszać się poza domem, w nieznanym środowisku. Istnieją rozwiązania w formie nawigacji np. Google Maps, nawigujące zakręt po zakręcie. Istnieją też rozwiązania, które informują o położeniu najbliższych punktów zainteresowania (POI), takich jak sklepy, restauracje czy urzędy.

Brakuje rozwiązań, które wykrywałyby przeszkody na chodniku i obok niego. Niewidomi do tego celu zazwyczaj korzystają z pomocy specjalnie wyszkolonych psów, tzw. pies przewodnik, który idzie wzduż krawędzi chodnika, wskazując drogę, na której nie ma przeszkód, zatrzymując niewidomego przed przejściem dla pieszych. Wyszkolenie psa przewodnika trwa około dwóch lat, a ceny w Polsce wahają się w okolicach 30 tys. złotych, co nie jest trywialnym wydatkiem i nie każdy może sobie na to pozwolić.

Niedowidzący do poruszania się poza domem używają także laski i z jej pomocą wykrywają krawędź chodnika i idą wzduż niej. Co w sytuacji, gdy nie ma krawędzi, jeśli osoba taka znajdzie się na placu, skrzyżowaniu chodników? Osoba taka nie ma żadnego punktu odniesienia.

Osoby niedowidzące z pomocą laski wykrywają też większe przeszkody, lecz laska ma ograniczony zasięg i osoby takie zmuszone są iść wolnym tempem, by w porę zareagować na przeszkodę. Laska taka może być niewydajna, gdy na krawędzi chodnika postawione są słupki i osoba niedowidząca stale o nie zahacza laską. Kolejnym problemem jest możliwe uszkodzenie wrażliwych obiektów, które znajdują się na drodze laski. Doniczki naziemne, samochody, rowery, hulajnogi, wózki, stragan na rynku. Niewidomy laską może i nie doprowadzi do całkowitego zniszczenia obiektu przed nim się znajdującego, ale może go porysować, a i siebie przy okazji poturbować.

Celem pracy jest projekt i implementacja aplikacji mobilnej dla niewidomych służącej do nawigacji głosowej po mieście.

Zakres projektu obejmuje przegląd literatury, a także analizę istniejących rozwiązań pod kątem wad i zalet. Praca zawiera także sam projekt w tym m.in. diagramy przypadków użycia, pakietów, scenariusze przypadków użycia, jak i samą implementację aplikacji mobilnej, i testowanie.

W ramach projektu zawarto takie funkcje jak: detekcja krawędzi chodnika i możliwych przeszkód, za pomocą biblioteki OpenCV do obróbki obrazu [4] [16]. Informowanie użytkownika o jego aktualnej lokalizacji, adresie. Problem orientacji

rozwiążany został za pomocą kompasu, który głosowo informuje o aktualnym kierunku geograficznym, w jakim zwrócony jest użytkownik. W projekcie zostaną zawarte także proste funkcje takie jak aktualna data i godzina. Kolejną z funkcji w zakresie projektu jest obsługa komend głosowych. W zakresie projektu jest także włączenie nawigacji Google Maps, jako zewnętrznej aplikacji.

Poza zakresem projektu jest użycie biblioteki Tensorflow Lite do faktycznej detekcji i nazwania obiektów czy przeszkód. Pomimo niemałych zalet, jakie wykrywanie ludzi, samochodów czy rowerów dałoby tego rodzaju aplikacji, to implementacja takiego rozwiązania nie byłaby trywialna, z uwagi na nie przystosowanie biblioteki do Androida. Co prawda istnieją gotowe, wytrenowane modele, ale by detekcja taka była możliwa, trzeba wpierw pobrać obraz z kamery w sposób wydajny, następnie zrównoleglić detekcję, tak by nie zatrzymywała całej aplikacji, gdy w 200ms będzie przetwarzać daną klatkę. W przyszłości, gdy korzystanie z tej biblioteki zostanie uproszczone, stanowiłaby ona dobre rozwinięcie tej aplikacji. Projekt nie obejmuje także funkcji informowania użytkownika o najbliższych punktach zainteresowania (POI).

3. Przegląd rozwiązań

3.1. Przegląd literatury o detekcji krawędzi chodnika i przeszkód

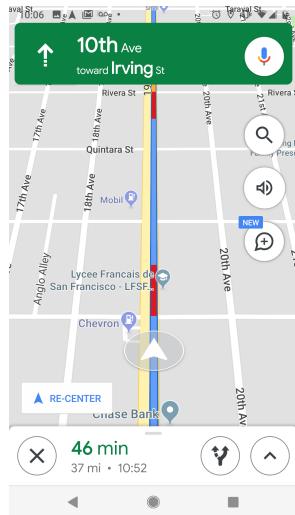
System detekcji ścieżek oraz przeszkód ruchomych dla niewidomych

Praca naukowa [18] prezentuje rozwiązanie problemu poruszania się niewidomych po chodniku za pomocą jednej kamery. System ten wpierw ogranicza obszar detekcji do horyzontu. Tak by niezależnie od rotacji kamery w detekcji krawędzi nie był uwzględniany obszar ponad horyzontem. Następnie taka wycięta klatka jest konwertowana do obrazu szarego, by zmniejszyć czas przetwarzania klatki. Potem nakładany jest filtr wygładzający, który rozmazuje obraz, by zachować jedynie istotne krawędzie. Po czym system stosuje detekcję krawędzi Canny'ego [14]. A na koniec użycie transformacji Hough'a [21] na liniach. Wynikiem powyższych transformacji są 2 linie, jedna po lewej stronie chodnika pochylona w prawo, a druga znajduje się po prawej stronie i pochylona jest w lewo. Na przecięciu tych linii znajduje się punkt wygaszenia (Vanishing Point), estymujący horyzont. Druga część artykułu traktuje o detekcji kluczowych punktów na obrazie, tych zmiennych, jak i tych niezmiennych. Porównując przemieszczenie tychże punktów w stosunku do punktu wygaszenia, można dokonać estymacji czy obiekt porusza się w naszym kierunku, czy w przeciwnym.

Detekcja przeszkód na chodnikach

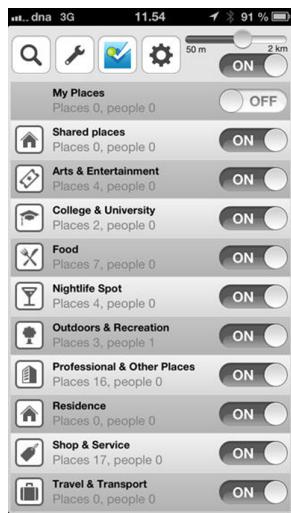
W tej pracy naukowej [3], podobnie jak w poprzedniej wykrywany jest chodnik i jego granice. Wpierw wykonywana jest detekcja krawędzi Canny'ego. Następnie detekcja przeszkód podzielona jest na dwa etapy. Pierwszy etap (szybszy i mniej dokładny), sprawdzenie wariancji w minimalnych i maksymalnych poziomów szarości na chodniku. Drugi etap choć wolniejszy, ale jednocześnie dokładniejszy działa poprzez uzyskanie horyzontalnych iertykalnych histogramów obrazu i tam, gdzie są duże spadki w zawartości białych krawędzi lub gdzie jest ich nadmiar prawdopodobnie jest obiekt / przeszkoda.

3.2. Przegląd aplikacji używanych przez niewidomych



Rys. 3.1. Interfejs graficzny Google Maps (źródło: <https://support.google.com/maps/thread/14375376?hl=en>).

Na rys. 3.1 przedstawiono interfejs aplikacji Google Maps. Jest to najpopularniejsza aplikacja do nawigacji. Zawiera aktualne mapy i dokładną lokalizację łączącą estymacje lokalizacji za pomocą GPS, jak i Internetu. Google Maps jest całkowicie darmowe i nawiguje głosowo zakręt po zakręcie, informując użytkownika o tym, kiedy ma skręcić. Jest to popularna aplikacja wśród niewidomych pomimo tego, że nie jest to aplikacja ściśle do nich dostosowana. Choć Google Maps oferuje tryb chodzenia, to nie mówi o każdym nadchodzącym zakręcie na każdej ścieżce, tylko o tych występujących na drogach. Google Maps, choć daje wskazówki gdzie skręcić przed zakrętem, to nie daje wskazówek, zanim do tego zakrętu się dojdzie. Co więcej, moment, w którym Google Maps mówi, by skręcić, nie zawsze pokrywa się z faktycznym położeniem geograficznym osoby niewidomej. Google wydało dodatek dostępny w USA i Japonii skierowany do osób niewidomych [24], który na celu miał wspomóc osoby niewidome w nawigacji pomiędzy jednym zakrętem a drugim. W praktyce jednak dużo użytkowników zgłaszało błędy w działaniu dodatku i najwyraźniej na tym sprawa się zakończyła, bo od jesieni 2019 roku nie ma żadnych informacji o dalszym rozwijaniu tej funkcji.



Rys. 3.2. Interfejs graficzny BlindSquare (źródło: <https://www.blindsight.com/>).

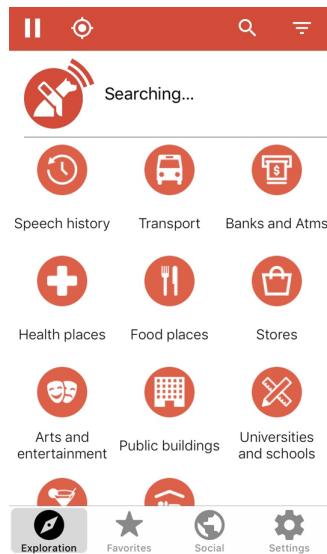
Na rys. 3.2 przedstawiono interfejs aplikacji BlindSquare. BlindSquare to aplikacja mobilna dla niewidomych, która opisuje najbliższe punkty zainteresowań (POI) mówiąc o ich odległości i kierunku, w stosunku do aktualnego położenia użytkownika. Aplikacje tego typu pełnią ważną funkcję, gdyż opisują miejsca wokół użytkownika, takie jak sklepy, apteki, przystanki etc. Jeśli osoba niewidoma jest w nieznanej okolicy i nie ma przewodnika, to nie wie, gdzie znajduje się sklep, gdzie może zrobić zakupy.

BlindSquare ma interfejs w pełni graficzny, bez komend głosowych. Często używany jest przez osoby niewidome, jako dodatek do nawigacji w Google Maps. BlindSquare umożliwia także filtrowanie punktów zainteresowań, jak i maksymalny zasięg informowania o punktach zainteresowań. Istotną wadą BlindSquare jest cena, wynosząca 40\$ oraz brak informacji o aktualnej lokalizacji użytkownika. Aplikacja mówi jedynie o lokalizacji punktów zainteresowań, a także kierunku, w jakim one się znajdują względem użytkownika.



Rys. 3.3. Interfejs graficzny Be My Eyes (źródło: <https://www.bemyeyes.com/>).

Na rys. 3.3 przedstawiono interfejs aplikacji Be My Eyes. Be My Eyes to aplikacja oparta na relacji wolontariusz-osoba niewidoma. Obecnie w aplikacji zarejestrowanych jest 4,333,957 wolontariuszy oraz 266,725 osób niedowidzących. Wolontariusze pochodzą ze 150 różnych krajów, oferując tym samym pomoc w ponad 180 językach. Sama aplikacja zaś działa tak, że zweryfikowani wolontariusze dobrowolnie pomagają osobom niewidomym za pomocą wideo rozmowy. Pomoc oferowana przez wolontariuszy w tej aplikacji jest szeroka, począwszy od identyfikacji etykiet na produktach, daty ważności po wskazanie drogi na trudnym skrzyżowaniu. Aplikacja ta jest całkowicie darmowa, lecz w dłuższym użytkowaniu niepraktyczna w kontekście nawigacji. W wypadku zerwania połączenia internetowego na środku skrzyżowania lub zwyczajnie będąc na nieznanym terenie, osoba niewidoma będzie w niekomfortowej sytuacji.

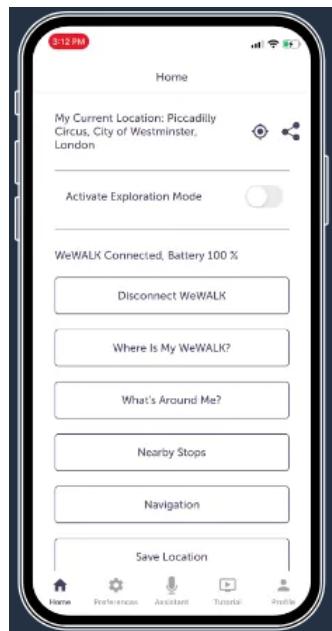


Rys. 3.4. Interfejs graficzny Lazarillo GPS for Blind (źródło: <https://lazarillo.app/>).

Na rys. 3.4 przedstawiono interfejs aplikacji Lazarillo GPS for Blind. Lazarillo GPS for Blind to aplikacja mobilna dla niewidomych i tak jak BlindSquare, lokalizuje najbliższe punkty zainteresowań (POI). Podczas testowania tej aplikacji lokalizacja najbliższych POI, o ile pod względem odległości jest dokładna, to pod względem kierunku rozróżnia jedynie prawo, lewo, naprzód i z tyłu i to z pewnym opóźnieniem, i dozę niedokładności. Lazarillo jednak mówi o aktualnej lokalizacji użytkownika (adresie), najbliższych skrzyżowaniach, a także jest za darmo, co jest istotną zaletą w stosunku do BlindSquare. Co więcej, Lazarillo zawiera wbudowaną nawigację na bazie Google Maps API. Lazarillo w serwisie Google Play ma 1000 opinii i ponad 50 tys. pobrań.



Rys. 3.5. Laska z sensorem ultradźwiękowym (źródło: <https://wewalk.io/en/>).



Rys. 3.6. Interfejs graficzny aplikacji WeWalk (źródło: <https://wewalk.io/en/>).

WeWalk to startUp z 2017 roku oferujący laski z sensorem ultradźwiękowym (600\$, rys. 3.5) wykrywającym obiekty przed osobą niewidomą, a także darmową aplikację z nawigacją i kierunkami zegarowymi przedstawioną na rys. 3.6. Co więcej, aplikacja ta informuje o najbliższych przystankach tramwajowych i autobusowych, sklepach czy restauracjach. Do tych obiektów aplikacja nawiguje użytkownika, informując go o odległości do następnego zakrętu, a także o obecnym kierunku zegarowym użytkownika w stosunku do lokalizacji kolejnego punktu. Należy jednak zaznaczyć, że aplikacja wykrywa jedynie sklepy wielkopowierzchniowe, także nie wszystkie restauracje. Wykrywa zaś wszystkie przystanki komunikacji miejskiej. WeWalk stanowi połączenie wszystkich powyższych rozwiązań. Aktualnie aplikacja WeWalk została pobrana tylko kilka tysięcy razy i oceniona 44 razy, a widnieje ona na serwisie

Google Play od 2018 roku. Zakres funkcji jest bardzo szeroki, ale tak niewielka ilość użytkowników jest co najmniej zastanawiająca.

3.3. Inne rozwiązania

Nowy projekt od Google, opublikowany 17 listopada 2020 roku [22]. Projekt ten ma na celu pomóc osobom niewidomym w swobodnym bieganiu poza domem, bez przewodnika. Projekt jest w fazie beta, ale wiadomo, że zastosowano w nim techniki uczenia maszynowego do detekcji linii, wzduż której osoba niewidoma biegnie.

Istnieje również Depth API z biblioteki ARCore stworzonej przez Google [1]. Umożliwia ono tworzenie obrazów z uwzględnieniem głębokości przy użyciu dwóch kamer telefonu. Jednak Depth API wspiera niewielką ilość telefonów, z czego cena większości to wydatek ponad 2 tys. złotych. Co jest stosunkowo zaskakujące z uwagi na fakt, że większość współczesnych smartfonów, nawet tych za 500 zł często ma kilka kamer z tyłu. Użycie tej biblioteki stanowiłoby dobre rozwinięcie rozwiązania zaproponowanego w tej pracy.

Istnieją również rozwiązania z działu konwolucyjnych sieci neuronowych używających jedynie jednej kamery do estymacji mapy głębokości obrazu [20]. Rozwiązanie to było testowane na Raspberry PI 3 (Arm Cortex A53) z wynikiem 4 klatek na sekundę. Choć praca naukowa ta implementowana była w Pythonie przy użyciu biblioteki Tensorflow, to istnieje również wersja tejże biblioteki na Androidzie pod nazwą Tensorflow Lite. Istnieje zatem możliwość implementacji tej pracy na urządzenia Android.

Ciekawą opcją jest także sensor ToF (Time of Flight), który wysyła podczerwoną wiązkę światła i w zależności od czasu odpowiedzi estymuje odległość. Sensor ToF występuje głównie we flagowych smartfonach o cenach 2 tys. zł i wzwyż, choć sama technologia nie jest droga. ToF używany jest do ulepszenia ustawiania ostrości obrazu. Niestety producenci nie dają dostępu do tego sensora. W przyszłości, jeśli producenci smartfonów zaczną udostępniać dane z sensora ToF deweloperom, można by go użyć do wykrywania przeszkołd.

3.4. Podsumowanie

Wyżej wymienione rozwiązania oferują szeroką gamę funkcji. Stosunkowo dokładna nawigacja za pomocą Google Maps, wysoka niezawodność i niewielka konkurencja na rynku sprawia, że jest najczęściej używaną aplikacją przez niewidomych do poruszania się poza domem. Aplikacje takie jak BlindSquare, Lazarillo czy WeWalk oferując nawigację do najbliższych punktów zainteresowań, również używane są przez osoby niewidome, jednak ich liczba nie jest już liczona w milionach, lecz w tysiącach. Be My Eyes tworząc medium do wideo rozmów między wolontariuszami a niewidomymi, również umożliwia rozwiązywanie problemów, które tylko drugi, widzący człowiek

może rozwiązać. Na świecie jednak jest 285 mln osób niedowidzących i potrzeba by wielokrotnie więcej wolontariuszy by pomoc była zawsze dostępna.

Wszystkie z powyższych rozwiązań korzystają głównie z interfejsu graficznego, który w przeciwnieństwie do interfejsu głosowego nie wymaga połączenia internetowego, by funkcjonować. Inną sprawą jest to, że większość aplikacji dla niewidomych Internetu wymagają. Nawigacja w Google Maps, wyszukiwanie punktów zainteresowań w BlindSquare, Lazarillo czy video-rozmowa w Be My Eyes. Zastosowanie samego interfejsu głosowego byłoby nierozważne z uwagi nie tylko na niemożność korzystania z funkcji aplikacji, gdy przerwane zostanie połączenie internetowe, ale także z uwagi na możliwy hałas i rozmowy wokół użytkownika. Choć rozpoznawanie mowy za pomocą API Speech-to-Text oferowane przez Google jest stosunkowo dokładne i wspierane przez większość telefonów z Google Services, to przy ruchliwej ulicy może mieć problem z identyfikacją komendy głosowej. Jednakże zastosowanie interfejsu głosowego, choćby do tych najczęściej używanych funkcji mogłoby ułatwić używanie aplikacji. Osoba niewidoma używająca czytnika ekranu musi przejść przez każdy element interfejsu po kolej, z góry na dół, aż dojdzie do tego pożądanego. Z interfejsem głosowym wystarczy zaś jedynie wydać komendę, a aplikacja zwróci adekwatną odpowiedź. Toteż rozsądny jest implementacja obu interfejsów, graficznego do wszystkich funkcji i głosowego do tych najczęściej używanych.

Najważniejszą funkcją, której wszystkim wyżej wymienionym rozwiązaniom brakuje, pomijając laskę za 600\$ z WeWalk jest detekcja przeszkód. Choć wszystkie osoby niewidome dysponują zwykłą laską, czasem nawet psem przewodnikiem, to w powyższym przeglądzie istniejących rozwiązań brak praktycznego rozwiązania na telefony Android, które by wykrywało przeszkody i jednocześnie nie wymagało dodatkowych urządzeń peryferyjnych, takich jak sensor ultradźwiękowy.

Rozwiązanie proponowane w tej pracy zaś będzie podobne do dwóch podanych publikacji naukowych [18] [3], a mianowicie jest to podejście z działu przetwarzania obrazów [2], z wygładzaniem i detekcją krawędzi z założeniem, że przeszkody i krawędzie chodnika są proste, a nie krzywe. Rozwiązanie to choć niedoskonałe, to przy odpowiedniej implementacji stanowi niewielki narzut na procesor telefonu i nawet te starsze, 5-letnie Android'y 6.0 z niskiej półki są w stanie wykrywać przeszkody z 15 klatkami/s.

Podsumowując, przegląd istniejących rozwiązań na tle literatury oraz istniejących aplikacji pomógł wybrać, jakie cechy oraz funkcje powinna zawierać aplikacja dla niewidomych. Pokazało to, jakich funkcji brakuje, a jakich jest pod dostatkiem i nie ma potrzeby ich powielania. W następnym rozdziale - Założenia Projektowe, funkcje aplikacji, której projekt i implementacja to cel tej pracy zostaną wymienione oraz opisane.

4. Założenia projektowe

Na podstawie przeglądu istniejących rozwiązań oraz opisu problemu przedstawiono poniższe założenia projektowe. W ramach założeń projektowych zawarto przedmiot projektu, wymagania funkcjonalne i niefunkcjonalne.

4.1. Przedmiot projektu

Przedmiotem projektu jest implementacja aplikacji mobilnej wspomagającej osoby niewidome w poruszaniu się poza domem. Aplikacja powinna wykrywać obiekty oraz krawędzie chodnika, przed użytkownikiem za pomocą tylnej kamery telefonu. Aplikacja powinna także wykrywać aktualną lokalizację użytkownika (utrudnione wewnętrz budynków) oraz kierunek geograficzny, w jakim użytkownik jest aktualnie zwrócony np. północ, południe. Aplikacja pozwoli także na modyfikowanie szerokości i wysokości pola detekcji krawędzi, w zależności od potrzeb użytkownika np. zmniejszenie pola detekcji wewnętrz budynków. Aplikacja implementowana będzie jedynie na urządzenia z systemem Android.

4.2. Wymagania funkcjonalne

Na podstawie przedmiotu projektu przedstawiono poniższe wymagania funkcjonalne aplikacji.

1. Użytkownik otrzymuje informacje dźwiękowe o przeszkodach znajdujących się przed nim.
2. Użytkownik otrzymuje informacje dźwiękowe o krawędzi chodnika, ściany znajdującej się przed nim.
3. Użytkownik może otrzymać aktualną lokalizację w formie adresu.
4. W przypadku braku adresu odpowiadającego danej lokalizacji, aplikacja informuje o aktualnej szerokości i długości geograficznej użytkownika.
5. Użytkownik może dowiedzieć się o aktualnym kierunku geograficznym, w jakim jest ustawiony telefon (kompas).
6. Użytkownik może dowiedzieć się o aktualnej dacie i godzinie.
7. Użytkownik może włączyć zewnętrzną nawigację Google Maps z poziomu aplikacji, podając adres docelowy.
8. Użytkownik może używać komend głosowych dostępnych dla niektórych funkcjonalności, jako alternatywę dla interfejsu graficznego.
9. Użytkownik może zmieniać szerokość i wysokość pola detekcji krawędzi.
10. Użytkownik może włączać i wyłączać detekcję przeszkód.

11. Użytkownik może zmieniać wersje językowe aplikacji: angielski i polski.

4.3. Wymagania niefunkcjonalne

Na podstawie przedmiotu projektu przedstawiono także wymagania niefunkcjonalne aplikacji.

1. Aplikacja dostępna jest w dwóch wersjach językowych: polskiej i angielskiej.
2. Aplikacja wspiera urządzenia Android od wersji 6.0 do 10.0 (API 23 do API 29).
3. Do działania komend głosowych (rozpoznawanie mowy) i nawigacji potrzebne jest połączenie internetowe.
4. Do działania detekcji krawędzi potrzebne jest zezwolenie na używanie kamery.
5. Do określenia aktualnej lokalizacji geograficznej potrzebny jest włączony GPS i zezwolenie do używania go.
6. Do określenia aktualnego adresu potrzebne jest połączenie internetowe.

4.4. Podsumowanie

Podsumowując, na podstawie opisu problemu oraz przeglądu istniejących rozwiązań przedstawiono powyższe założenia projektowe. Określono przedmiot projektu, wymagania funkcjonalne i niefunkcjonalne. Na tej podstawie w następnym rozdziale przedstawiony zostanie projekt aplikacji.

5. Projekt aplikacji

Na podstawie założeń projektowych dokonano poniższego projektu aplikacji. W ramach projektu zawarto:

1. historyjki użytkownika
2. diagram przypadków użycia
3. scenariusze przypadków użycia
4. diagram pakietów
5. wzorce projektowe

5.1. Historyjki użytkownika

Jako rozwinięcie wymagań funkcjonalnych przedstawiono poniższe historyjki użytkownika.

H-01 Jako użytkownik chcę otrzymywać informacje o zbliżaniu się do krawędzi chodnika.

H-02 Jako użytkownik chcę otrzymywać informacje o przeszkodach znajdujących się przede mną.

H-03 Jako użytkownik chcę, by detekcja przeszkód rozróżniała obiekty znajdujące się blisko i daleko.

H-04 Jako użytkownik chcę mieć możliwość otrzymania informacji o mojej aktualnej lokalizacji w formie adresu.

H-05 Jako użytkownik chcę móc dowiedzieć się o kierunku geograficznym, w jakim zmierzam.

H-06 Jako użytkownik chcę móc otrzymać informacje o aktualnej dacie i godzinie.

H-07 Jako użytkownik chcę mieć możliwość włączenia nawigacji Google Maps na określony przeze mnie adres.

H-08 Jako użytkownik chcę móc komunikować się z aplikacją za pomocą komend głosowych.

H-09 Jako użytkownik chcę mieć możliwość używania aplikacji jedynie za pomocą interfejsu graficznego.

H-10 Jako użytkownik chcę mieć możliwość wyboru pomiędzy polską a angielską wersją aplikacji.

H-11 Jako użytkownik chcę mieć możliwość zmiany ustawień detekcji przeszkód.

H-12 Jako użytkownik chcę móc włączać i wyłączać detekcję przeszkód, by zmniejszyć zużycie baterii.

H-13 Jako użytkownik chcę, aby odpowiedzi aplikacji były prezentowane w formie głosowej, z pomocą syntezatora mowy.

5.2. Diagram przypadków użycia

Na podstawie wymagań funkcjonalnych oraz historyjek użytkownika zaprojektowano diagram przypadków użycia 5.1.

Spis przypadków użycia

PU-01 Otrzymywanie informacji o przeszkołach

PU-02 Ustalenie lokalizacji

PU-03 Ustalenie kierunku geograficznego

PU-04 Informacja o godzinie

PU-05 Informacja o dacie

PU-06 Wydawanie komend głosowych

PU-07 Nawigacja Google Maps

PU-08 Głosowe otrzymanie informacji

PU-09 Wybór ustawień aplikacji

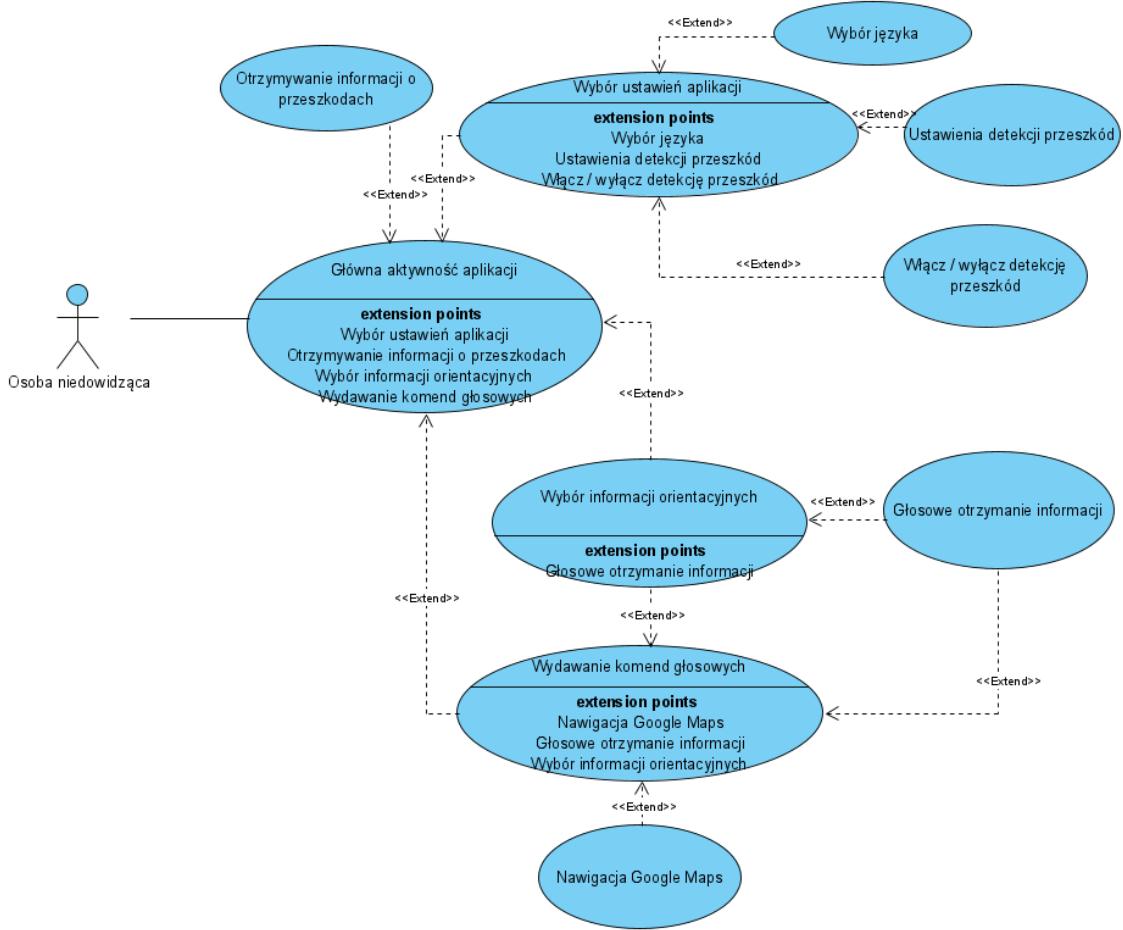
PU-10 Wybór języka

PU-11 Ustawienia detekcji przeskóków

PU-12 Włącz / wyłącz detekcję przeskóków

Z uwagi na zachowanie czytelności diagramu, przypadki użycia PU-02, PU-03, PU-04, PU-05 zawarte są w jednym przypadku użycia, na diagramie widniejącym pod nazwą 'Wybór informacji orientacyjnych'.

Diagram przypadków użycia



Rys. 5.1. Diagram przypadków użycia (źródło: opracowanie własne)

Na diagramie 5.1 przedstawiono aktora, osobę niedowidzącą oraz przypadki użycia. Użytkownik z poziomu głównej aktywności otrzymuje informacje dźwiękowe o przeszkodach i krawędzi chodnika przed nim się znajdujących **PU-01**, o ile detekcja przeszkód nie jest wyłączona **PU-12** w ustawieniach **PU-09**. W ustawieniach użytkownika może także zmienić język komend i syntezatora mowy **PU-10**, wybierając pomiędzy językiem polskim a angielskim. Użytkownik może także ustawić względną szerokość i wysokość pola detekcji przeszkód **PU-11** zależną także od rotacji telefonu. Z poziomu głównej aktywności użytkownik może otrzymać informacje orientacyjne traktujące o jego lokalizacji (w formie adresu) **PU-03**, kierunku geograficznym (kompass) **PU-04**, czasie **PU-05** i dacie **PU-06**. Z poziomu głównej aktywności użytkownik może także wydawać komendy głosowe **PU-07**, uzyskując funkcje zawarte w **PU-03**, **PU-04**, **PU-05**, **PU-06**, **PU-08**, które to zwracane są użytkownikowi w formie głosowych informacji **PU-09** (syntezator mowy).

5.3. Scenariusze przypadków użycia

Na podstawie opracowanego diagramu przypadków użycia oraz wymagań funkcjonalnych, jak i samej problematyki projektowania aplikacji skierowanej do osób niewidomych przedstawiono poniższe scenariusze przypadków użycia.

Przypadek użycia: PU-01 Detekcja przeszkód

Aktor: Użytkownik

Warunki wejściowe: Użytkownik nadał uprawnienia aplikacji na używanie kamery telefonu.

Scenariusz główny:

1. Użytkownik zbliża się do krawędzi chodnika, przeszkody.
2. Aplikacja wykrywa przecięcie krawędzi chodnika, przeszkody z polem detekcji lub wykrywa krawędź zawartą wewnątrz pola detekcji.

Scenariusze alternatywne:

- 2a1. Krawędź chodnika, przeszkody znajduje się po lewej stronie użytkownika.
- 2a2. Aplikacja odtwarza dźwięk w lewej słuchawce, o głośności zależnej od odległości użytkownika od krawędzi.

- 2b1. Krawędź chodnika, przeszkody znajduje się po prawej stronie użytkownika.
- 2b2. Aplikacja odtwarza dźwięk w prawej słuchawce, o głośności zależnej od odległości użytkownika od krawędzi.

- 2c1. Krawędź chodnika, przeszkody znajduje się naprzeciwko użytkownika.
- 2c2. Aplikacja odtwarza dźwięk w prawej i w lewej słuchawce o proporcjach zależnych od położenia najbliższego punktu danej krawędzi, o głośności zależnej od odległości użytkownika od krawędzi.

Przypadek użycia: PU-02 Ustalenie lokalizacji

Aktor: Użytkownik

Warunki wejściowe: Użytkownik nadał uprawnienia aplikacji na używanie lokalizacji telefonu.

Scenariusz główny:

1. Użytkownik kliką przycisk informowania o aktualnej lokalizacji.
2. Aplikacja wykonuje odwrotne geokodowanie na podstawie aktualnych współrzędnych, otrzymując od Geocoder API aktualny adres.
3. Syntezator mowy głosowo informuje użytkownika o lokalizacji w odpowiednim języku.

Scenariusze alternatywne:

- 1a1. Użytkownik klika przycisk wydawania komend, po czym wymawia komendę 'lokalizacja'.
- 2a1. W razie braku Internetu aplikacja zwraca aktualną długość i szerokość geograficzną.
- 2b1. W razie nieistniejącego adresu w danej lokalizacji aplikacja również zwraca dł. i szer. geograficzną.

Przypadek użycia: PU-03 Ustalenie kierunku geograficznego

Aktor: Użytkownik

Warunki wejściowe: brak

Scenariusz główny:

1. Użytkownik klika przycisk informowania o kierunku geograficznym, w jakim użytkownik jest skierowany.
2. Aplikacja odczytuje dane z sensora magnetycznego i akcelerometru i określa aktualny kierunek geograficzny telefonu.
3. Syntezator mowy głosowo informuje użytkownika o kierunku w odpowiednim języku.

Scenariusze alternatywne:

- 1a1. Użytkownik klika przycisk wydawania komend, po czym wymawia komendę 'kompas'.
- 2a1. W razie braku Internetu aplikacja zwraca aktualną długość i szerokość geograficzną.
- 2b1. W razie nieistniejącego adresu w danej lokalizacji aplikacja również zwraca dł. i szer. geograficzną.

Przypadek użycia: PU-04 Informacja o godzinie

Aktor: Użytkownik

Warunki wejściowe: brak

Scenariusz główny:

1. Użytkownik klika przycisk informowania o aktualnej godzinie.
2. Syntezator mowy głosowo informuje użytkownika o aktualnej godzinie i minucie w odpowiednim języku.

Scenariusze alternatywne:

- 1a1. Użytkownik klika przycisk wydawania komend, po czym wymawia komendę 'czas'.

Przypadek użycia: PU-05 Informacja o dacie

Aktor: Użytkownik

Warunki wejściowe: brak

Scenariusz główny:

1. Użytkownik klika przycisk informowania o aktualnej dacie
2. Syntezator mowy głosowo informuje użytkownika o aktualnej dacie w odpowiednim języku.

Scenariusze alternatywne:

- 1a1. Użytkownik klika przycisk wydawania komend, po czym wydaje komendę 'data'

Przypadek użycia: PU-06 Wydawanie komend głosowych

Aktor: Użytkownik

Warunki wejściowe: Dostęp do Internetu (plik audio jest wysyłany do chmury Google za pomocą SpeechRecognition API)

Scenariusz główny:

1. Użytkownik klika przycisk do wydawania komend głosowych.
2. Aplikacja wydaje dźwięk rozpoczęcia nasłuchiwanego.
3. Użytkownik mówi komendę.
4. Aplikacja rozpoznaje komendę i włącza funkcję odpowiadającą komendzie.

Scenariusze alternatywne: Nie rozpoznano komendy

- 3a1. Użytkownik nic nie mówi, powrót do kroku 1.

- 4a1. Aplikacja nie rozpoznaje komendy i za pomocą syntezatora mowy mówi '*<to co powiedział użytkownik> nie jest komenda, powiedz słowo 'pomoc' po więcej informacji*'.

- 4a2. Powrót do kroku 1.

Scenariusze alternatywne: Rozpoznano komende

- 4b1. Aplikacja rozpoznaje komendę 'nawigacja' - dalszy scenariusz w PU-07 Nawigacja Google Maps.

- 4c1. Aplikacja rozpoznaje komendę 'lokalizacja' - dalszy scenariusz w PU-02 Ustalenie lokalizacji.

- 4d1. Aplikacja rozpoznaje komendę 'kompass' - dalszy scenariusz w PU-03 Ustalenie kierunku geograficznego.

- 4e1. Aplikacja rozpoznaje komendę 'czas' - dalszy scenariusz w PU-04 Informacja o godzinie.

- 4f1. Aplikacja rozpoznaje komendę 'data' - dalszy scenariusz w PU-05 Informacja o dacie.

- 4g1. Aplikacja rozpoznaje komendę 'pomoc' i mówi użytkownikowi o liście do-

stępujących komend.

Przypadek użycia: PU-07 Nawigacja Google Maps

Aktor: Użytkownik

Warunki wejściowe: Dostęp do Internetu

Scenariusz główny:

1. Użytkownik klika przycisk do wydawania komend głosowych.
2. Aplikacja wydaje dźwięk rozpoczęcia nasłuchiwanego.
3. Użytkownik mówi komendę '*nawigacja*'.
4. Aplikacja rozpoznaje komendę i pyta użytkownika o miejsce docelowe (np. przystanek tramwajowy świdnicka).
5. Użytkownik mówi, jakie jest miejsce docelowe.
6. Aplikacja pyta się użytkownika czy to na pewno jest ta lokalizacja.
7. Użytkownik potwierdza, mówiąc '*zgadza się*' lub '*tak*', lub '*owszem*'.
8. Aplikacja rozpoznaje komendę.
9. Aplikacja włącza zewnętrzną aplikację mobilną Google Maps w trybie nawigacji.

Scenariusze alternatywne: *Użytkownik nic nie mówi*

- 3a1. Użytkownik nic nie mówi, powrót do kroku 1.
- 5a1. Użytkownik nic nie mówi, powrót do kroku 1.
- 7a1. Użytkownik nic nie mówi, powrót do kroku 1.

Scenariusze alternatywne: *Nie rozpoznano komendy*

- 4a1. Aplikacja nie rozpoznaje komendy, powrót do kroku 1.
- 8a1. Aplikacja nie rozpoznaje komendy, powrót do kroku 1.

Przypadek użycia: PU-08 Głosowe otrzymanie informacji

Aktor: Użytkownik

Warunki wejściowe: brak

Scenariusz główny:

1. Moduł syntezy mowy otrzymuje wiadomość tekstową do wypowiedzenia na głos.
2. Moduł syntezy za pomocą Syntezatora Mowy Google'a mówi zawartość wiadomości w języku wybranym w ustawieniach.

Przypadek użycia: PU-09 Wybór ustawień aplikacji

Aktor: Użytkownik

Warunki wejściowe: brak

Scenariusz główny:

1. Użytkownik klika przycisk ustawień.

2. Aplikacja otwiera aktywność z ustawieniami.
3. Użytkownik po zmianie ustawień kliką strzałkę powrotu do głównego ekranu.

Przypadek użycia: PU-10 Wybór języka

Aktor: Użytkownik

Warunki wejściowe: brak

Scenariusz główny:

1. Użytkownik kliką przycisk 'język'.
2. Na ekranie pojawia się okno z językami do wyboru.
3. Użytkownik wybiera język.

Scenariusze alternatywne:

- 3a1. Użytkownik kliką przycisk 'anuluj' i język nie zostaje zmieniony.

Przypadek użycia: PU-11 Ustawienia detekcji przeszkód

Aktor: Użytkownik

Warunki wejściowe: brak

Scenariusz główny:

1. Użytkownik kliką przycisk 'wysokość pola detekcji'
2. Na ekranie pojawia się okno z wielkościami do wyboru: wysokie, normalne, niskie.
3. Użytkownik wybiera wysokość

Scenariusze alternatywne:

- 1a1. Użytkownik kliką przycisk 'szerokość pola detekcji'
- 1a2. Na ekranie pojawia się okno z wielkościami do wyboru: szerokie, normalne, wąskie.
- 3a1. Użytkownik kliką przycisk 'anuluj' i wysokość pola nie zostaje zmieniona.

Przypadek użycia: PU-12 Włącz / wyłącz detekcję przeszkód

Aktor: Użytkownik

Warunki wejściowe: brak

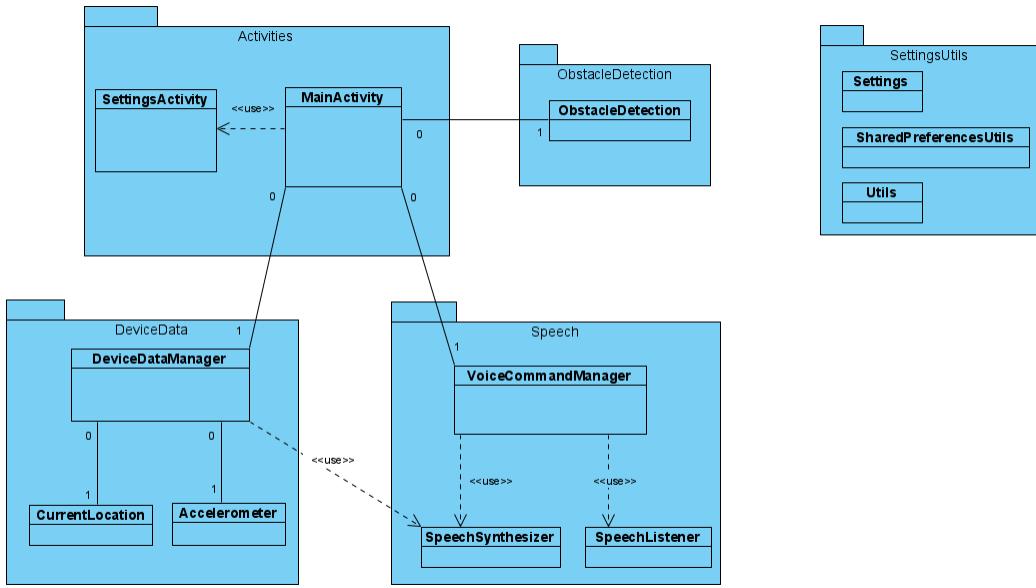
Scenariusz główny:

1. Użytkownik kliką przycisk typu przełącznik (switch) 'Włącz detekcję', włączając detekcję.

Scenariusze alternatywne:

- 1a1. Przełącznik był przed kliknięciem włączony, toteż po kliknięciu detekcja zostaje wyłączona.

5.4. Diagram pakietów



Rys. 5.2. Diagram pakietów (źródło: opracowanie własne)

Na diagramie pakietów (rys. 5.2) zawarto wszystkie 5 pakietów aplikacji:

1. Activities - aktywności aplikacji, odpowiedzialne za widok aplikacji.
2. ObstacleDetection - pakiet dotyczący detekcji przeszkód, krawędzi chodnika.
3. DeviceData - pakiet zajmujący się zbieraniem i przetwarzaniem danych o aktualnej lokalizacji, akcelerometrze, polu magnetycznym (kompas).
4. Speech - pakiet zajmujący się rozpoznawaniem i syntezą mowy, a także zarządzaniem komendami głosowymi
5. SettingsUtils - pakiet zawierający klasy statyczne zorientowane wokół przechowywania i zapisywania, i wczytywania ustawień aplikacji.

Główna aktywność - MainActivity przechowuje referencje do DeviceDataManager, VoiceCommandManager i ObstacleDetection, klas zarządzających głównymi funkcjami aplikacji. DeviceDataManager przechowuje referencje do CurrentLocation oraz Accelerometer, a także używa statycznej klasy SpeechSynthesiser do syntezy mowy i informowania o aktualnej lokalizacji, kierunku geograficznym, dacie czy godzinie. VoiceCommandManager zarządzający komendami głosowymi używa klasy SpeechListener do nasłuchiwanego i zamiany mowy na tekst. Menedżer komend po otrzymaniu odpowiedzi tekstowej używa syntezatora mowy do głosowej reprezentacji informacji. Klasa ObstacleDetection zajmuje się detekcją krawędzi z obrazu kamery. Pakiet SettingsUtils nie jest połączony asocjacjami z innymi pakietami z uwagi na statyczną i wielofunkcyjną naturę klas wewnętrznych zawartych. Klasa statyczna Settings

przechowująca aktualne ustawienia aplikacji używana jest we wszystkich pakietach, np. pakiet Speech - ustawienia języka, pakiet ObstacleDetection - ustawienia detekcji, aktywność Settings - zmiana ustawień, DeviceDataManager - formatowanie daty zależne od języka.

5.5. Wzorce projektowe

W aplikacji zastosowano wstrzykiwanie zależności [15], które umożliwia rozbicie dużych aktywności na mniejsze klasy, gdzie każda pełni pojedynczą, określoną funkcję. W ten sposób otrzymujemy klasy, które są oddzielnymi bytami, co umożliwia prostsze testowanie i rozbudowę aplikacji. Rozproszenie funkcji pozwala na użycie tych klas do innych projektów, np. klasa CurrentLocation, Accelerometer, SpeechSynthesizer, SpeechListener dostarczają uniwersalnych funkcji, które przy niewielkich zmianach i usunięciu zależności, mogą być wykorzystane do innych projektów.

W aplikacji nie zastosowano znanych wzorców projektowych stosowanych powszechnie w aplikacjach mobilnych, takich jak MVC (starszy wzorzec), MVP czy MVVM. Stosowanie tych wzorców znacznie usprawnia rozbudowywanie średnich i większych aplikacji, które przechowują dane i na ich podstawie zmieniają widok. Opierają się one na przepływie View -> ... -> Model, Model -> ... -> View. W aplikacji będącej przedmiotem tej pracy widok - interfejs graficzny aplikacji, pomijając zmianę widoku głównego na widok ustawień, nie jest zmieniany. Przepływ zorganizowany jest na View -> Model, gdzie użytkownik kliką dany przycisk, a aplikacja mówi o aktualnej lokalizacji, kompasie, dacie czy godzinie, mówi, nic na ekranie nie wyświetla. Przepływ Model -> View w tej aplikacji nie występuje, pomijając wyświetlanie obrazu z kamery, gdzie biblioteka OpenCV sama się zajmuje buforowaniem obrazu do odpowiedniego widoku.

5.6. Podsumowanie

Zakres pracy obejmował przegląd literatury oraz istniejących rozwiązań, co zostało wykonane. W zakresie pracy był także sam projekt aplikacji zawierający diagram przypadków użycia, scenariusze i diagram pakietów, co także zostało zrealizowane. W zakresie projektu zawarto projekt takich funkcji jak: detekcja przeszkód i krawędzi chodnika, kompas, aktualna lokalizacja oraz włączenie zewnętrznej aplikacji Google Maps, działającej w tle.

6. Implementacja aplikacji mobilnej

Na podstawie projektu aplikacji dokonano poniższej implementacji. Implementacja uwzględnia wszystkie scenariusze przypadków m.in. detekcję przeszkód, kompas, lokalizację, nawigację, ustawienia detekcji oraz języka. W ramach implementacji umówione zostaną użyte biblioteki, struktura projektu, jak i zasada działania wszystkich funkcji wraz ze zrzutami ekranu z aplikacji.

6.1. Środowisko, biblioteki

Aplikacja mobilna została napisana w języku Kotlin [17] na urządzenia Android za pomocą środowiska Android Studio [6] w wersji 4.1. Do napisania tej aplikacji użyto Kotlina zamiast Javy z uwagi na fakt, że Kotlin zmniejsza liczbę linii kodu w stosunku do Javy, nie tracąc na czytelności kodu, zwiększając skalowalność projektu. Także Kotlin staje się standardem w nowych aplikacjach mobilnych i niezastosowanie go byłoby nieuzasadnione.

Biblioteką użytą do przetwarzania obrazu i detekcji krawędzi przeszkód jest OpenCV [12] w wersji 4.1 na Androida (wersja OpenCV musi być taka sama jak wersja Android Studio). Użytoło właśnie OpenCV z uwagi na fakt, że nie istnieją inne biblioteki na Androida, które posiadałyby wszystkie, a nawet część funkcji potrzebnych do implementacji tej aplikacji. OpenCV jest powszechnie stosowanym narzędziem do przetwarzania obrazów na Androidzie. Co prawda można by pobierać klatki z kamery bez użycia OpenCV i samemu zaimplementować potrzebne funkcje przetwarzania obrazu, lecz znacznie zmniejszyłoby to wydajność aplikacji. Do działania tej funkcji aplikacja potrzebuje zezwolenia na korzystanie z kamery telefonu.

Do rozpoznawania mowy użyto klasy SpeechRecognizer [10] z bibliotek Androida, która łącząc się z serwerami Google, wysyła plik dźwiękowy, a zwracany jest tekst. Ta samo API używane jest w wielu smartfonach w Google Assistant czy przy głosowej opcji wysyłania SMS. Użyto SpeechRecognizer z uwagi na fakt, że nie istnieje inna biblioteka do rozpoznawania mowy w Androidzie. Własnoręczna implementacja modelu sieci konwolucyjnej rozpoznającego mowę jest nieuzasadniona z uwagi na złożoność problemu, jak i fakt, że istnieje już prostsze i wydajniejsze rozwiązanie. Rozpoznawanie mowy wymaga zezwolenia aplikacji na korzystanie z mikrofonu telefonu oraz internetu.

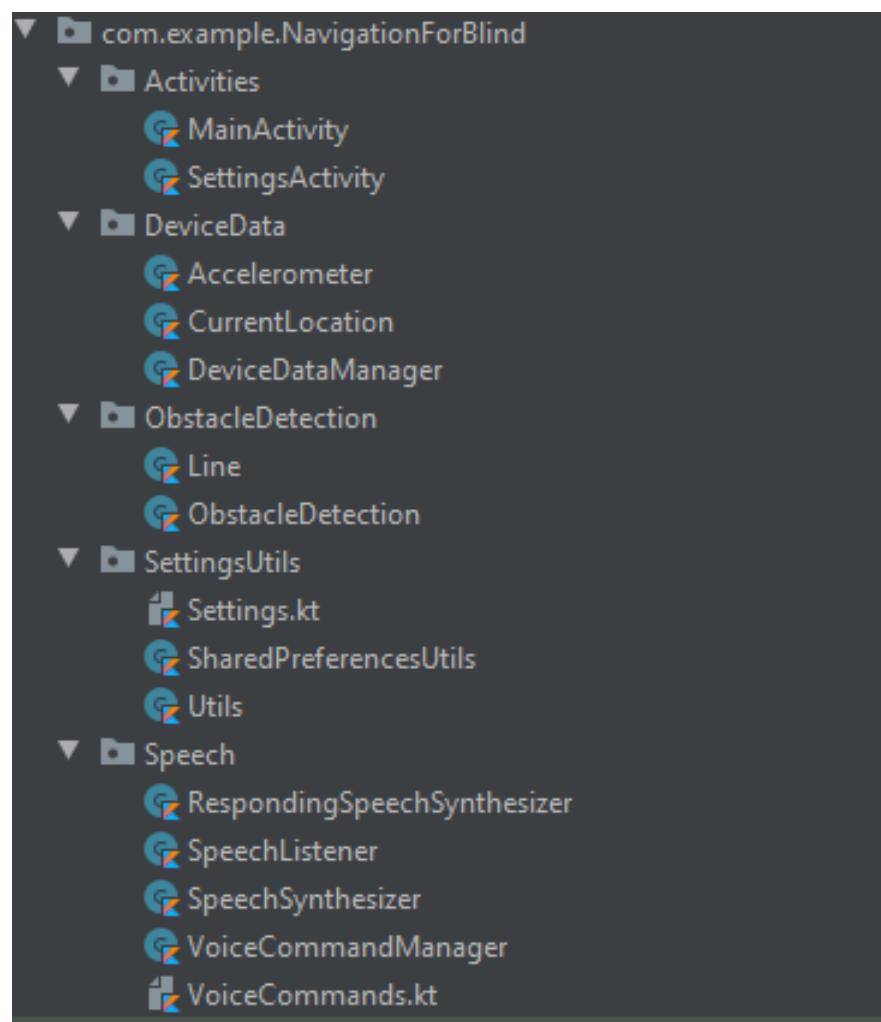
Do transformacji tekstu na mowę użyto klasy TextToSpeech [11], również jest to usługa dostarczana przez Google. Tak jak dla modułu rozpoznawania mowy, TextToSpeech to jedyne rozwiązanie oferujące syntezę mowy na Androidzie.

Do określenia aktualnej lokalizacji użyto FusedLocationProviderClient [7], usługa od Google'a dostarczająca dokładną lokalizację łącząc GPS, akcelerometr, sensor pola magnetycznego ziemi oraz Internet. Istnieje również inna biblioteka do określenia lokalizacji, a mianowicie LocationListener [9]. Jednakże zaleca się używania FusedLocationProviderClient z uwagi na szybsze oraz dokładniejsze określenie aktualnej lokalizacji. Zaznaczyć należy także, że bez włączonego GPS ta funkcja nie działa, stąd potrzebne jest zezwolenie aplikacji na korzystanie z GPS.

Do odwrotnej lokalizacji - uzyskiwania adresu na podstawie szerokości i długości geograficznej użyto Geocoder API [8] wymagającego dostępu do Internetu do poprawnego działania. Istnieje także inne rozwiązanie, a mianowicie bezpośrednie zapytanie do Google Maps Geocoding API [5] i otrzymanie pliku JSON, z którego następnie należy odczytać potrzebne dane. Jest to rozwiązanie bardziej czasochłonne w implementacji w porównaniu do Geocoder API, za pomocą którego otrzymanie adresu z danej lokalizacji wymaga jednej linii kodu.

6.2. Struktura projektu

Na podstawie diagramu pakietów rys. 5.2 zaimplementowano aplikacje o strukturze widocznej na rys. 6.1. Zawarto wszystkie klasy, które przedstawiono na diagramie pakietów oraz dodatkowo klasę RespondingSpeechSynthesizer, VoiceCommands oraz Line. RespondingSpeechSynthesizer pełni taką samą funkcję, jak SpeechSynthesizer, czyli generuje mowę na podstawie tekstu, z tą różnicą, że wywołuje on nasłuchiwanie na komendy głosowe. Jest to funkcja potrzebna do funkcji nawigacji, które wymaga sekwencji głosowych komend użytkownika i odpowiednich odpowiedzi menedżera komend (VoiceCommandManager). VoiceCommands.kt to plik z typami wyliczeniowymi (Enum) wszystkich komend głosowych. Klasa Line to klasa pomocnicza przechowująca dwa punkty (p1, p2).



Rys. 6.1. Struktura projektu (źródło: opracowanie własne)

6.3. Detekcja przeszkode

Opierajac się na pracach naukowych [3] [18] oraz z użyciem OpenCV [4], które udostępnia odpowiednie funkcje do przetwarzania obrazu, zaimplementowano algorytm detekcji przeszkode. Detekcja przeszkode w tej pracy opiera się na założeniu, że większość przeszkode, na które osoba niewidoma nie chciałaby wpaść, ma długie, proste linie. Obiekty takie jak słupki, śmietniki, samochody, krawężniki, lampy, kontenery czy hulajnogi składają się z długich linii, co umożliwia poniższemu algorytmowi wykrycie takich prostych krawędzi, a co za tym idzie wykrycie przeskody. Algorytm ten nie będzie wykrywał przeskód o okrągłych, krzywych krawędziach takich, jak koła samochodu. Zaznaczyć należy również, że algorytm detekcji będzie także błędnie wykrywał ostre, proste cienie, jako przeskody. Istnieją metody usuwania cieni [19], lecz rozwiązanie tego problemu mogłoby być pracą inżynierską samą w sobie, toteż nie jest to przedmiotem tej pracy.

6.3.1. Algorytm wykrywania krawędzi

Do transformacji klatki RGB z kamery do detekcji krawędzi użyto następującego algorytmu:

1. Transformacja obrazu z RGB do obrazu szarego, z jednym kanałem.
2. Zmniejszenie rozmiaru obrazu, dla większej wydajności. (Rozmiar ok 500x400 pikseli, zależnie od proporcji szerokości do wysokości obrazu z kamery)
3. Zastosowanie filtra medianowego o rozmiarze 5x5.
4. Zastosowanie zwykłego filtra uśredniającego o rozmiarze 5x5.
5. Operacja powiększenia jasnych pikseli (dilation), polegająca na znajdywaniu w sąsiedztwie danego piksela pikseli o maksymalnej jasności. Pełni tu rolę usuwania ciemnych krawędzi pomiędzy kostkami chodnikowymi.
6. Zastosowanie filtra medianowego o rozmiarze 5x5.
7. Detekcja krawędzi Canny [13]
8. Probabilistyczna transformacja Hough'a na liniach, służąca do detekcji prostych linii.

6.3.2. Efekty zastosowania rozszerzania jasnych pikseli (dilation)



Rys. 6.2. Obraz wygładzony, z ciemnymi krawędziami pomiędzy kostkami chodnikowymi
(źródło: opracowanie własne)



Rys. 6.3. Obraz z rozszerzonymi jasnymi pikselami (dilation), bez krawędzi pomiędzy kostkami brukowymi (źródło: opracowanie własne)



Rys. 6.4. Obraz wygładzony, z ciemnymi krawędziami pomiędzy kostkami chodnikowymi
(źródło: opracowanie własne)



Rys. 6.5. Obraz z rozszerzonymi jasnymi pikselami (dilation), bez krawędzi pomiędzy kostkami brukowymi (źródło: opracowanie własne)

Na rys. 6.2 i 6.3 oraz rys. 6.4 i 6.5 przedstawiono wpływ użycia dilation na obraz wejściowy. Co prawda można, by usunąć ciemne krawędzie pomiędzy kostką brukową za pomocą wygładzania dużymi filtrami, ale w ten sposób doszłoby do utraty wielu istotnych krawędzi. Dilation zaś umożliwia usunięcie tychże ciemnych,

nieistotnych krawędzi wewnętrz chodnika bez utraty większych, silnie zaznaczonych krawędzi. Stosowanie dilation opiera się na założeniu, że przerwy pomiędzy kostkami są ciemniejsze od samej kostki chodnikowej. W odwrotnej sytuacji, gdy kostki są ciemniejsze od szpar pomiędzy nimi, to wyrwy te byłyby powiększane aniżeli pomniejszane, jednakże jest to mało prawdopodobny scenariusz.

6.3.3. Wizualizacja algorytmu detekcji krawędzi

W ramach wizualizacji algorytmu ukazano następujące kroki przetwarzania obrazu i detekcji:

1. Transformacja obrazu z kamery do obrazy szarego - rys. 6.6
2. Wygładzanie obrazu filtrem medianowym i uśredniającym - rys. 6.7
3. Rozszerzenie jasnych pikseli (dilation) - rys. 6.8
4. Detekcja krawędzi Canny - rys. 6.9
5. Detekcja prostych lini Hough - rys. 6.10

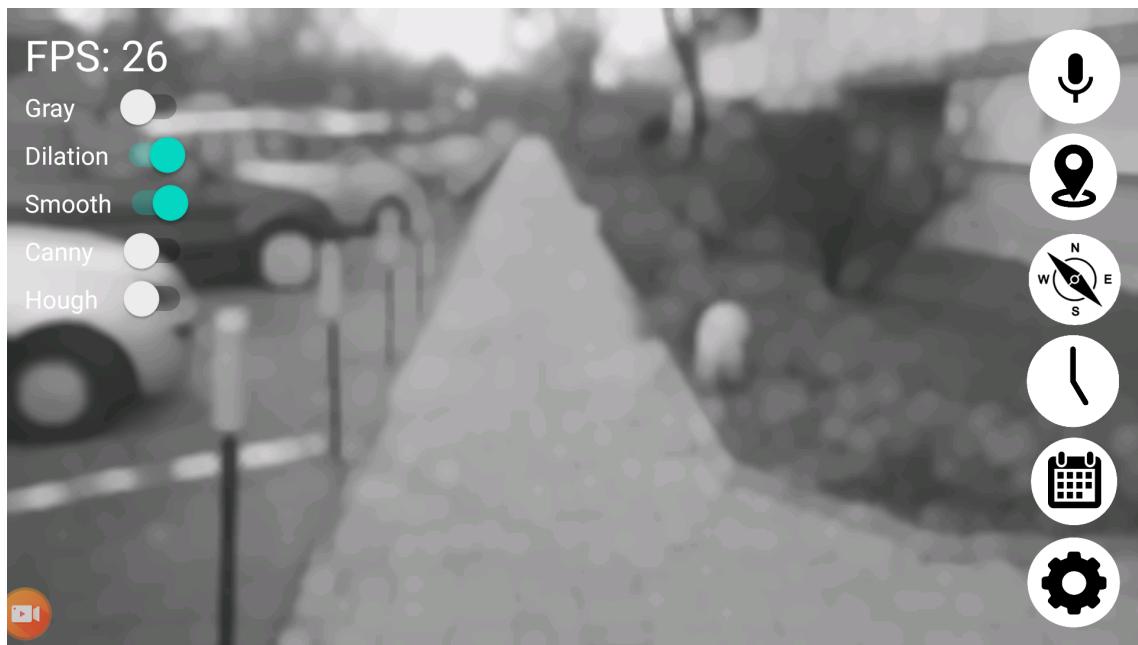


Rys. 6.6. Obraz szary (źródło: opracowanie własne)

Na rys. 6.6 widać, że żółte elementy słupków o wiele bardziej zlewają się z tłem niż ich czarne elementy, wpłynie to na detekcję krawędzi widoczną na rys. 6.9 oraz rys. 6.10. Wraz z utratą kolorów tracimy cenne informacje. W ramach dalszych kierunków prac warto byłoby korzystać także z informacji, jakie niosą ze sobą kolory obiektów. Jednakże do działania algorytmu Canny i Hough konieczna jest zamiana obrazu na szary.



Rys. 6.7. Obraz wygładzony filtrem medianowym i uśredniającym (źródło: opracowanie własne)



Rys. 6.8. Obraz po rozszerzeniu jasnych pikseli (źródło: opracowanie własne)



Rys. 6.9. Obraz po detekcji krawędzi Canny (źródło: opracowanie własne)



Rys. 6.10. Detekcja prostych krawędzi - Hough (źródło: opracowanie własne)

Jak widać na rys. 6.10 żółte elementy słupków nie są wykrywane, bo po transformacji do obrazu szarego zlewają się z chodnikiem za nimi.

6.3.4. Rozmiar pola detekcji



Rys. 6.11. Wysokość pola detekcji, po skierowanie telefonu równolegle do ziemi (źródło: opracowanie własne)



Rys. 6.12. Wysokość pola detekcji po skierowaniu telefonu prostopadle do podłoża (źródło: opracowanie własne)

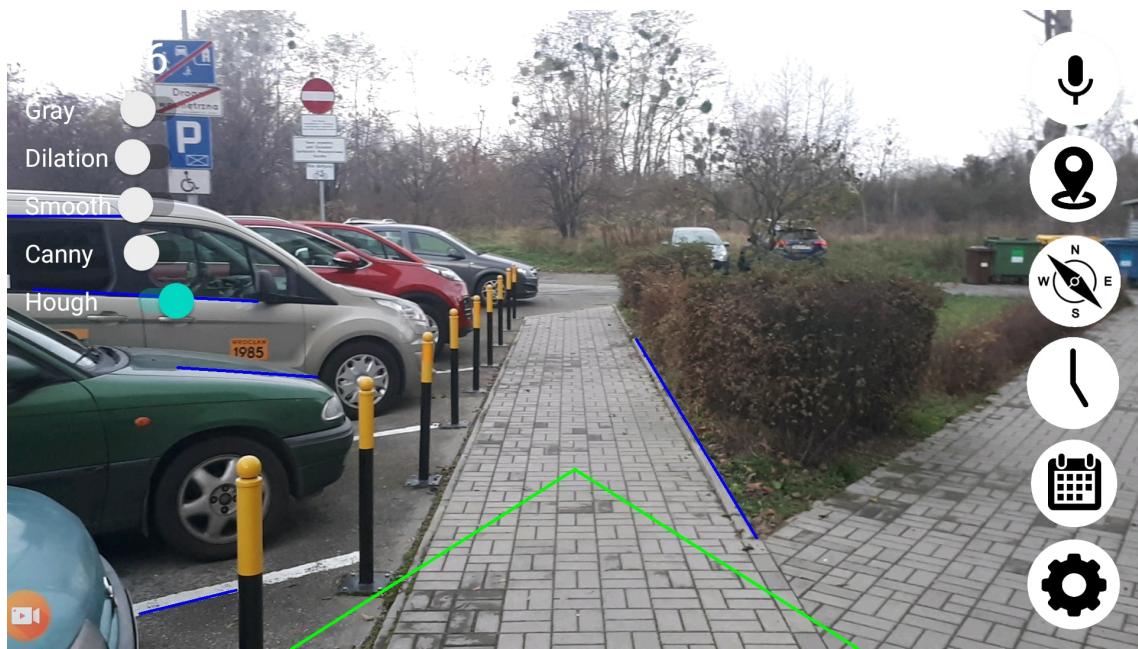
Wysokość pola detekcji zwiększa się, gdy telefon skierowany jest w dół (rys. 6.11) oraz zmniejsza się, gdy skierowany jest wyżej (rys. 6.12). Ta adaptacyjna wysokość

sprawia, że niezależnie od tego, jak użytkownik trzyma telefon nieistotne krawędzie ponad horyzontem, nie będą kolidować z polem detekcji.

Ponadto domyślna wysokość, jak i szerokość pola detekcji może być zmieniana przez użytkownika. Zmiany ustawień pola ukazane są na rys. 6.13, rys. 6.14 i rys. 6.15.



Rys. 6.13. Ustawienia wysokości pola - wysokie, ustawienia szerokości - wąskie (źródło: opracowanie własne)



Rys. 6.14. Ustawienia wysokości pola - normalne, ustawienia szerokości - normalna (źródło: opracowanie własne)



Rys. 6.15. Ustawienia wysokości pola - normalne, ustawienia szerokości - szerokie (źródło: opracowanie własne)

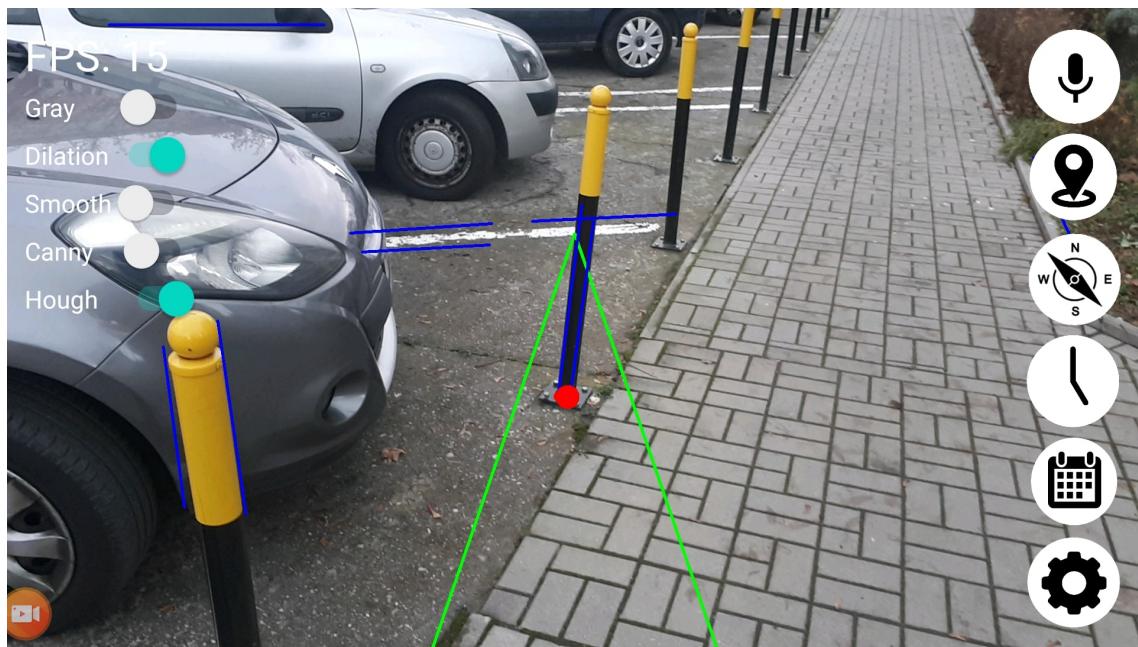
6.3.5. Przykłady detekcji przeszkód

Po wykryciu prostych krawędzi (niebieskie linie), algorytm sprawdza, czy punkty linii znajdują się wewnątrz pola detekcji (zielone linie), jeśli nie to algorytm sprawdza, czy krawędź przecina pole detekcji. Jeśli jedno z dwóch powyższych warunków jest prawdziwe (małe czerwone kropki), to algorytm sprawdza, jak blisko jest dany punkt od użytkownika (założenie, że krawędzie znajdujące się wyżej na obrazie są dalej położone). Do określenia ostatecznego punktu detekcji przeszkody (duża czerwona kropka) algorytm wylicza średnią pomiędzy najbliższym punktem z ostatniej i aktualnej klatki. Jest to punkt estymujący położenie najbliższej przeszkody i względem niego ustawiana jest głośność dźwięku detekcji na lewej i prawej słuchawce. Po upływie 5 klatek bez detekcji krawędzi punkt oznaczający przeszkodę znika i dźwięk detekcji jest wyciszany. Zaimplementowano to w taki sposób, ponieważ ostrość obrazu z kamery jest zmienna, toteż i wykrywanie krawędzi jest niestabilne. Zdarzyć się może, że w 1 klatce krawędź zostanie wykryta, a po lekkim obrocie kamery już nie. Należy też zaznaczyć, że algorytm dobrze wykrywa krawędzie o dużym gradiencie. Przeszkody, które kolorem zlewają się z chodnikiem, nie zostaną wykryte. Widok po lewej stronie interfejsu służy do celów pokazowych i nie jest widoczny w końcowej aplikacji.

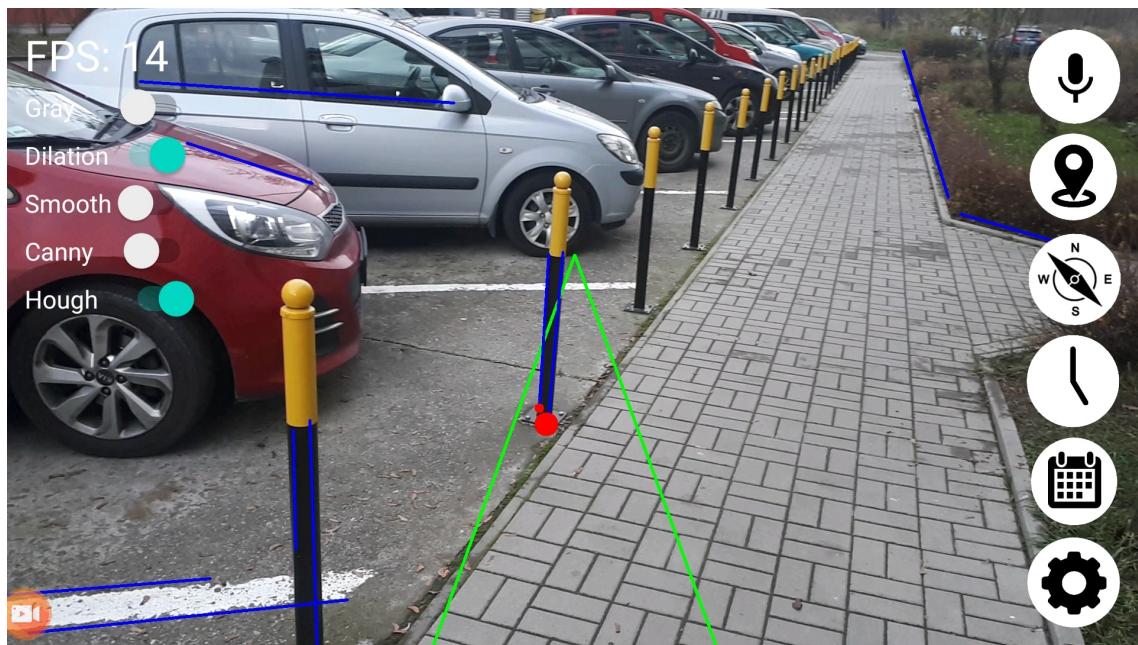


Rys. 6.16. Detekcja samochodów (źródło: opracowanie własne)

Na rys. 6.16 ukazano detekcję krawędzi samochodu. Wykrywane są jedynie proste krawędzie z silnie zaznaczonym gradientem (samochód po prawej), krzywe krawędzie karoserii samochodu po lewej nie są wykrywane.



Rys. 6.17. Detekcja słupków (źródło: opracowanie własne)

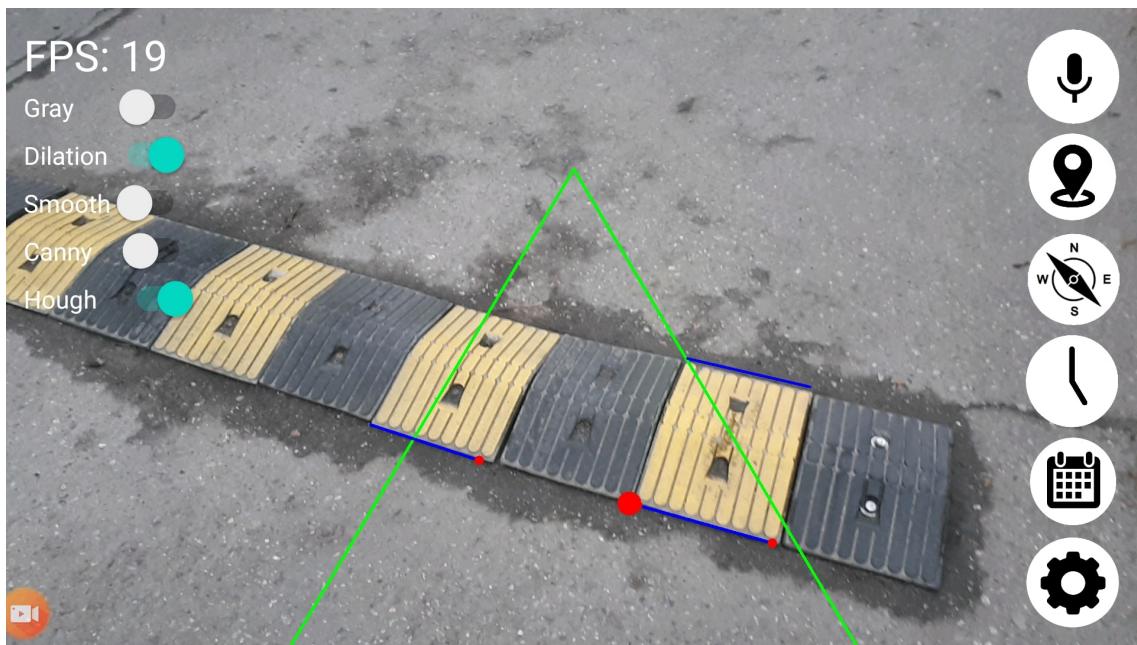


Rys. 6.18. Detekcja słupków - 2 (źródło: opracowanie własne)

Na rys. 6.17 oraz rys. 6.18 ukazano detekcję słupków. Widać, że lepiej wykrywane są słupki bliższe w części zamalowanej na czarno, żółta część jest gorzej wykrywana. Najwyraźniej w obrazie szarym żółty zlewa się z tłem, a czarny nie. Na rys. 6.19 dobrze zaś zostały wykryte krawędzie chodnika. Nie zawsze jednak krawędzie te udaje się wykryć.



Rys. 6.19. Detekcja krawędzi chodnika (źródło: opracowanie własne)

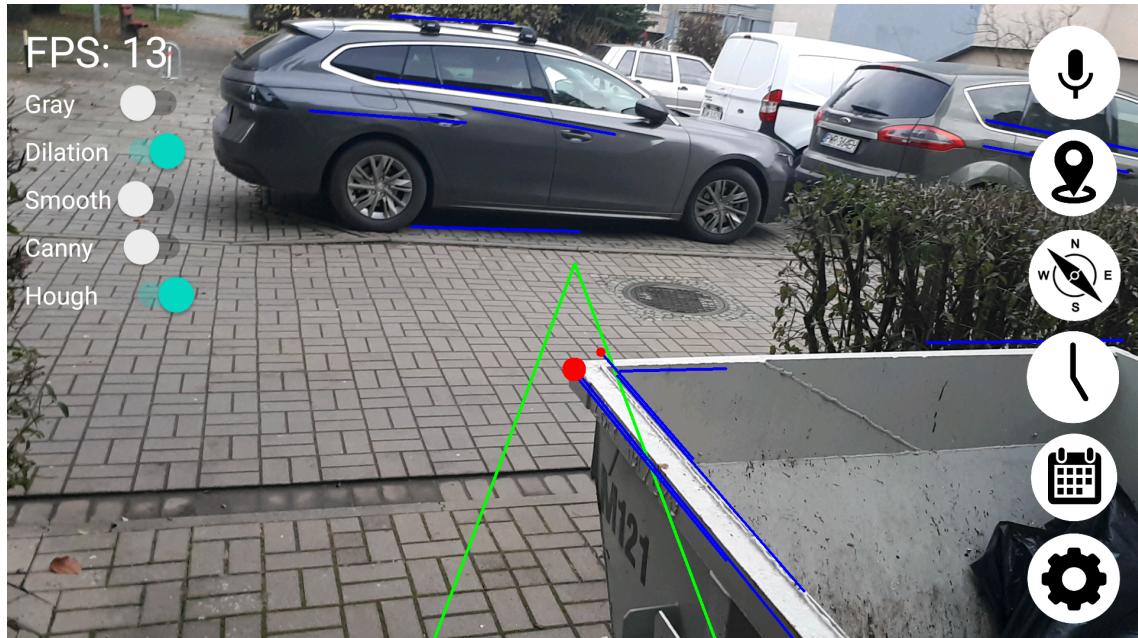


Rys. 6.20. Detekcja progu spowalniającego (źródło: opracowanie własne)

Na rys. 6.20 widać detekcję krawędzi poszczególnych elementów progu spowalniającego. Widać także, że lepiej wykrywane są żółte elementy, aniżeli czarne. Wynikać to może z zawilgocenia betonu wokół progu, które przyjemnia powierzchnie, zlewając ciemne elementy progu z betonem wokół.



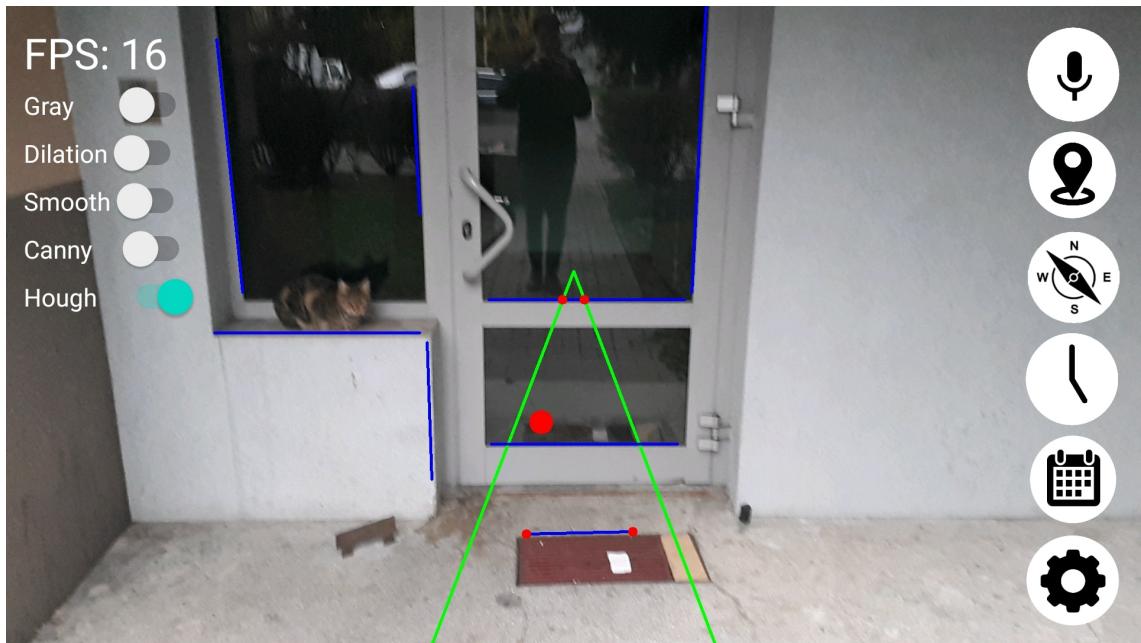
Rys. 6.21. Detekcja kontenera (źródło: opracowanie własne)



Rys. 6.22. Detekcja kontenera - 2 (źródło: opracowanie własne)

Na rys. 6.21 oraz rys. 6.22 przedstawiono detekcję krawędzi kontenera. Pomimo podobieństwa kolorystycznego kontenera z chodnikiem, krawędzie są wykrywane, głównie z uwagi na jego kanciasty kształt, co skutkuje różnym naświetleniem jego powierzchni. Na rys. 6.23 przedstawiono detekcję drzwi wejściowych do budynku.

Widać, że powierzchnie budynku podobnie naświetlone nie będą miały wykrywanych krawędzi, te zaś z cieniami, zmianami w naświetleniu, gradiencie będą wykrywane.



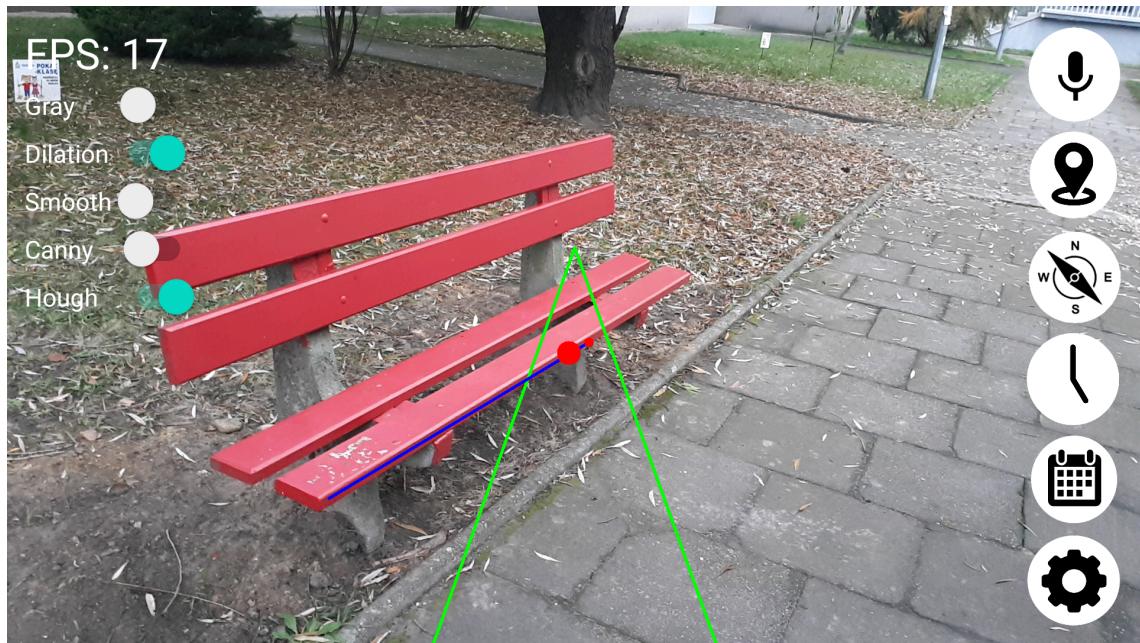
Rys. 6.23. Detekcja drzwi wejściowych (źródło: opracowanie własne)



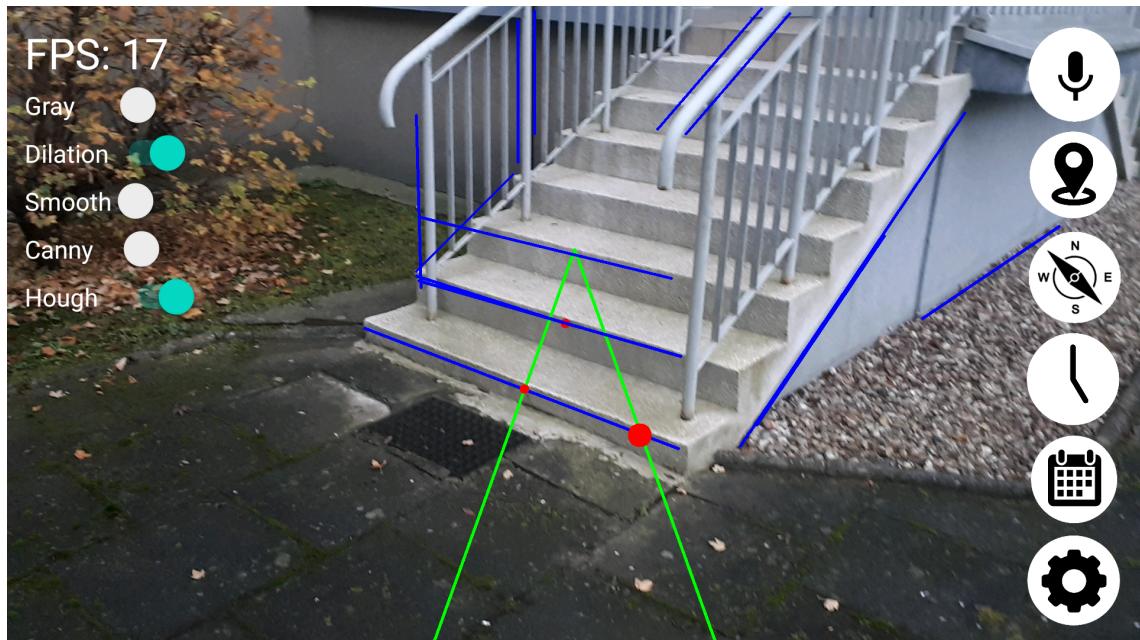
Rys. 6.24. Detekcja blokady samochodowej (źródło: opracowanie własne)

Na rys. 6.24 widać detekcję blokady samochodowej, a właściwie jedynie jej jednego elementu. Obiekt ten zlewa się kolorystycznie z chodnikiem, jest mały i w większości składa się z krzywych krawędzi to i jego wykrycie jest utrudnione.

Na rys. 6.25 wykryto pojedynczą krawędź ławki. Nie została zaś wykryta krawędź chodnika, niezbyt wysoka i zlewająca się z trawnikiem.



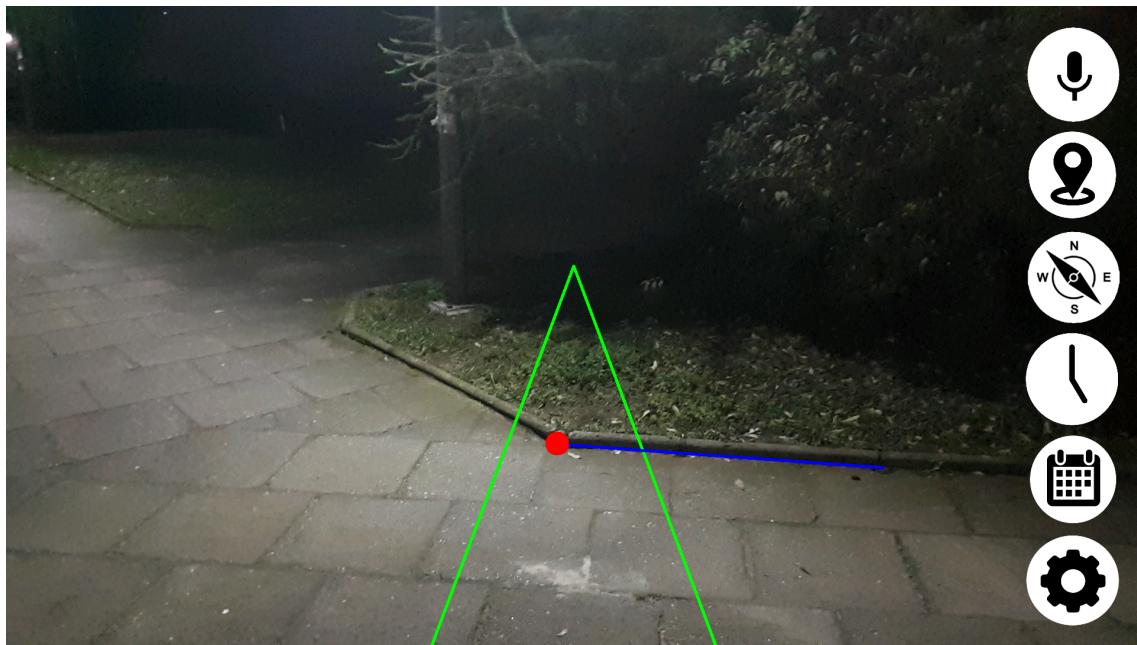
Rys. 6.25. Detekcja ławki (źródło: opracowanie własne)



Rys. 6.26. Detekcja schodów (źródło: opracowanie własne)

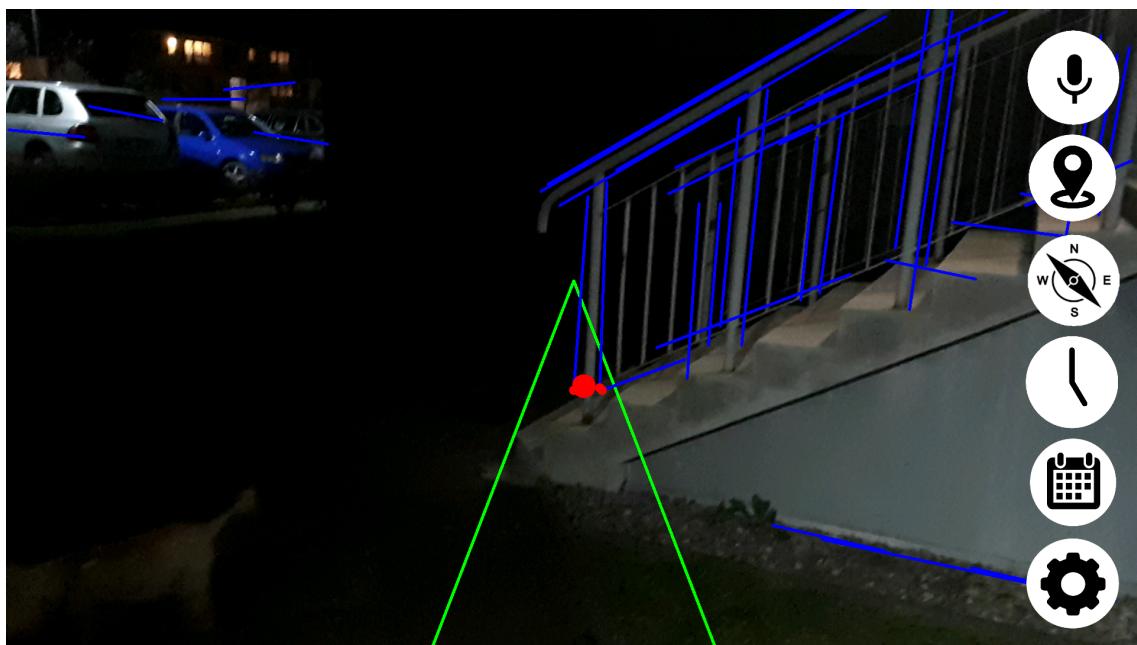
Na rys. 6.26 ukazano detekcję schodów i barierek. Z uwagi na zawartość dużej ilości prostych, dobrze naświetlonych krawędzi są to obiekty stosunkowo łatwe do wykrycia.

6.3.6. Detekcja przeszkód - noc

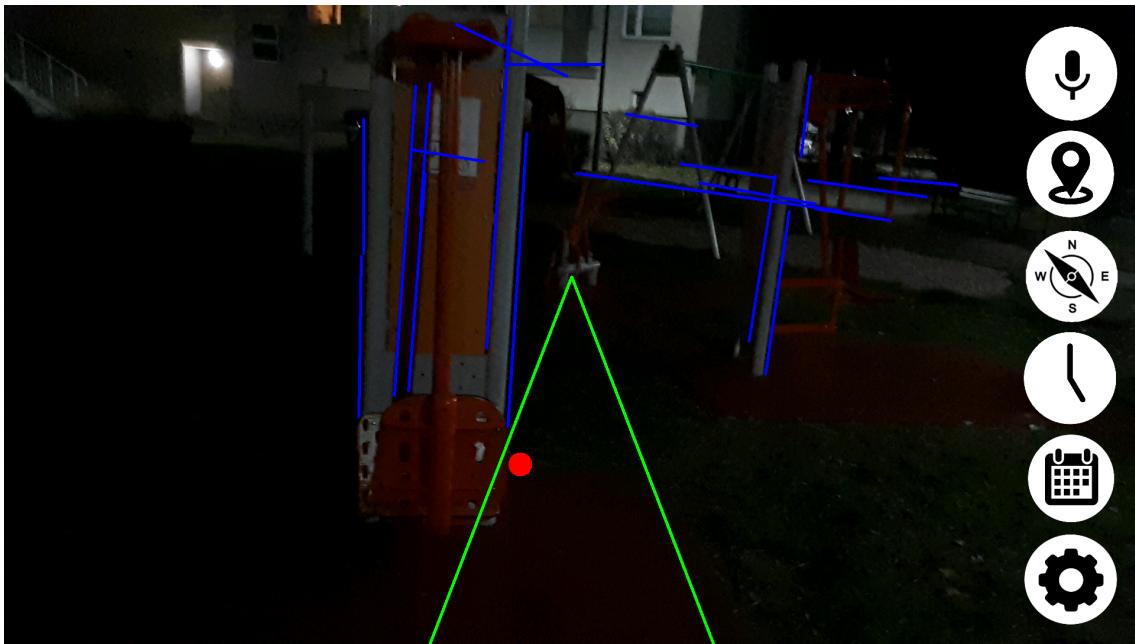


Rys. 6.27. Przykład 1 - detekcja krawędzi nocą (źródło: opracowanie własne)

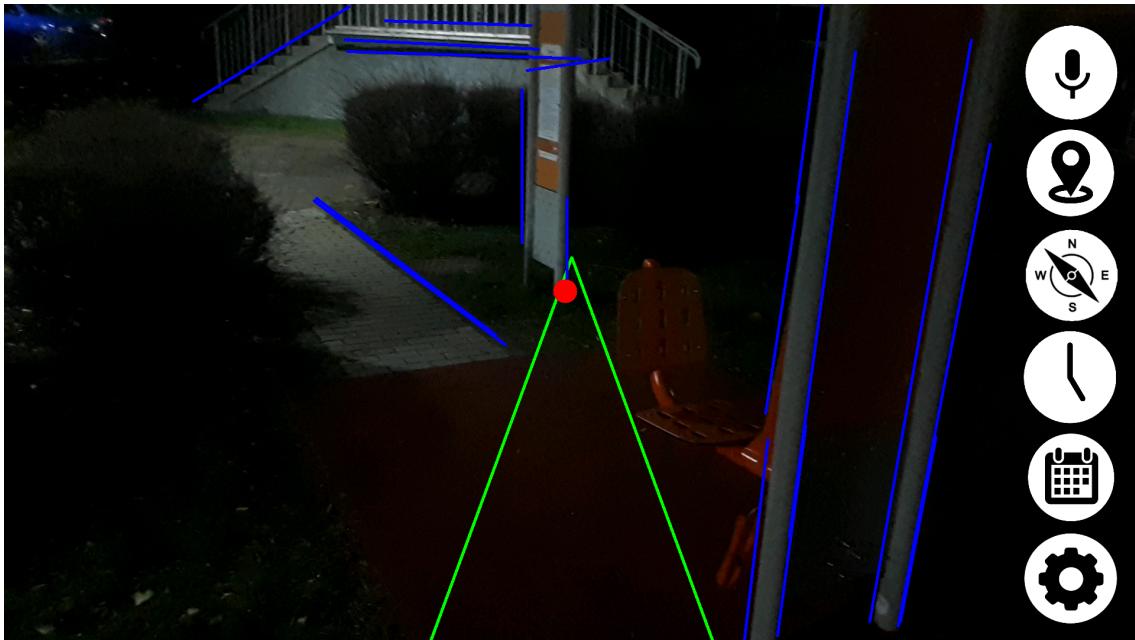
Na rys. 6.27 widać wykrytą krawędź chodnika, inne krawędzie, mniej oświetlone nie są wykrywane. Na rys. 6.28 ukazano detekcję schodów podobnego typu co na rys. 6.26 i pomimo gorszego oświetlenia algorytm nadal dobrze wykrywa schody z barierkami.



Rys. 6.28. Przykład 2 - detekcja krawędzi nocą (źródło: opracowanie własne)



Rys. 6.29. Przykład 3 - detekcja krawędzi nocą (źródło: opracowanie własne)



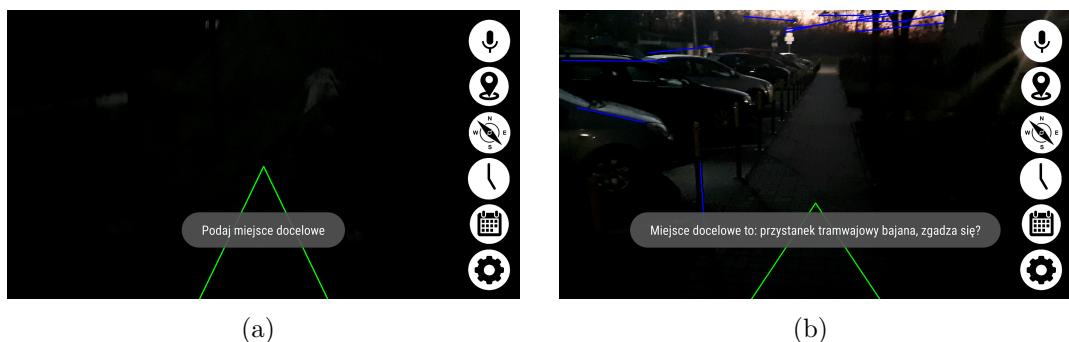
Rys. 6.30. Przykład 4 - detekcja krawędzi nocą (źródło: opracowanie własne)

Na rys. 6.29 oraz 6.30 widać detekcje obiektów na placu zabaw. Zauważać można, że algorytm radzi sobie w różnych warunkach oświetleniowych. Jest to możliwe dzięki adaptacyjnym progom wykrywania krawędzi Canny. Poprzez wyliczanie średniej jasności pikseli w klatce wylicza się dolny i górny próg detekcji Canny'ego, jako

odchylenie standardowe od średniej uzyskując tym samym algorytm działający w różnym oświetleniu.

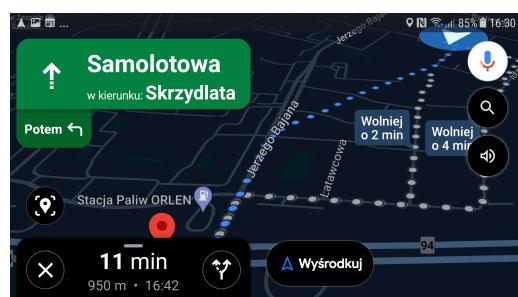
6.4. Przykład użycia nawigacji

Należy zaznaczyć, że aplikacja za pomocą synteza mowy głosowej zwraca odpowiedzi z poniższych funkcji. Z uwagi na fakt, że dźwięku nie da się w pisemnej pracy pokazać, to informacje będą pokazywane w formie powiadomień na dole ekranu co widać na rys. 6.31.



Rys. 6.31. Przykład użycia funkcji nawigacji (źródło: opracowanie własne)

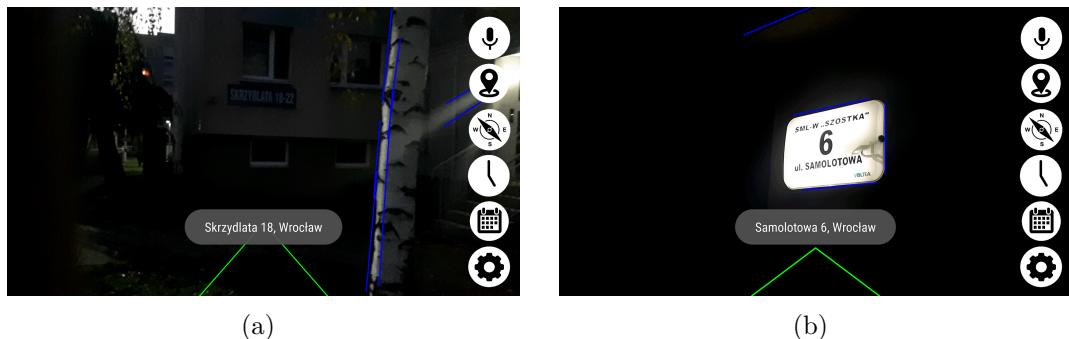
1. Odpowiedź aplikacji po komendzie 'nawigacja' (rys. 6.31)
2. Odpowiedź aplikacji po podaniu miejsca docelowego (rys. 6.31).



Rys. 6.32. Przykład użycia funkcji nawigacji cd. (źródło: opracowanie własne)

3. Automatyczne włączenie nawigacji Google Maps (rys. 6.32).

6.5. Przykłady użycia funkcji lokalizacji

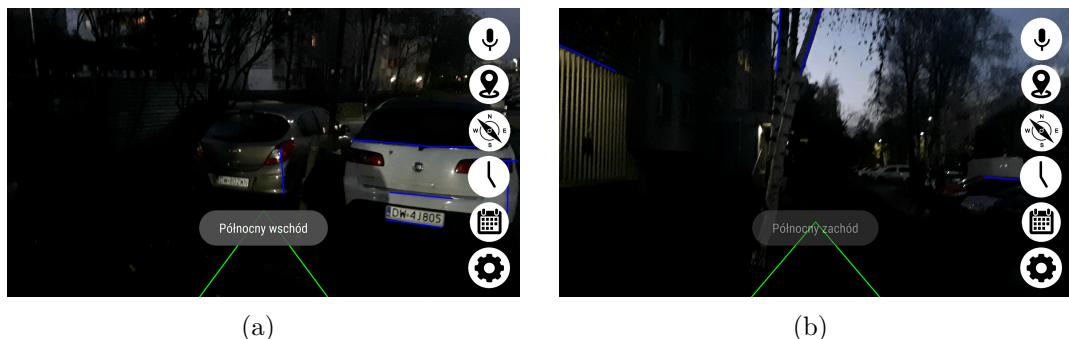


Rys. 6.33. Przykład użycia funkcji lokalizacji (źródło: opracowanie własne)

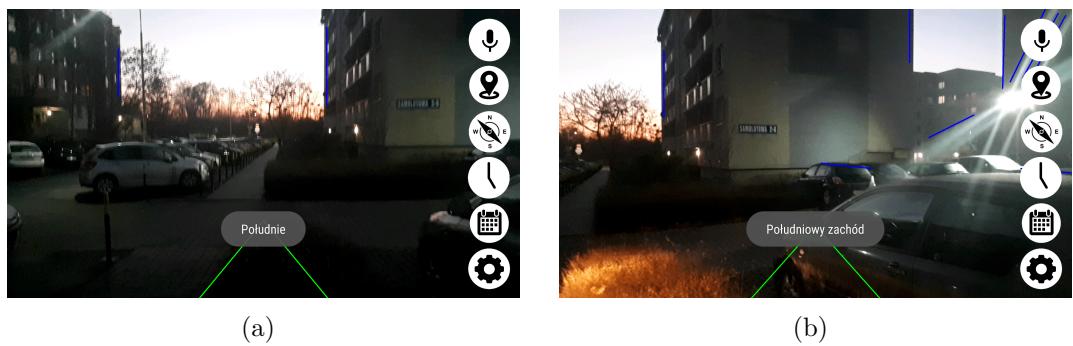
Na rys. 6.33 widać przykładowe użycie funkcji aktualnej lokalizacji. Z funkcji tej można skorzystać za pomocą komendy głosowej 'lokalizacja' lub za pomocą przycisku drugiego od góry po prawej stronie ekranu.

6.6. Przykłady użycia kompasu

Na rys. 6.34 oraz rys. 6.35 widać przykładowe użycia funkcji kompasu wskażającego aktualny kierunek geograficzny. W zależności od testowanego urządzenia poprawny kierunek wskazywany jest do 1-2 sekund (dla starych telefonów) od momentu ustalenia telefonu w danym kierunku. Dla nowszych telefonów poprawny kierunek wskazywany jest w czasie niemalże rzeczywistym (lepsze sensory magnetyczne, akcelerometry). Za kompas odpowiada 3 przycisk od góry lub komenda głosowa 'kompass'.



Rys. 6.34. Przykład 1 użycia funkcji kompasu (źródło: opracowanie własne)



Rys. 6.35. Przykład 2 użycia funkcji kompasu (źródło: opracowanie własne)

6.7. Przykłady użycia funkcji daty i godziny

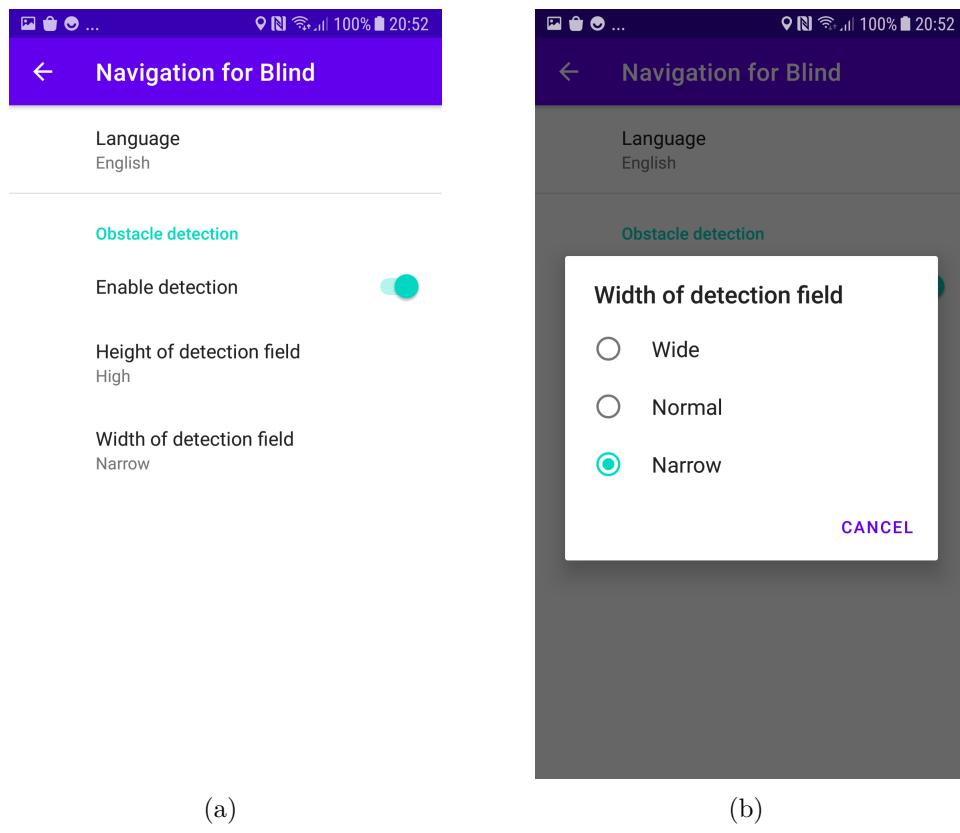
Na rys. 6.36 widać przykładowe użycie funkcji godziny (przycisk 3 od dołu) oraz daty (przycisk 2 od dołu). Funkcje te także są dostępne przy użyciu komend głosowych.



Rys. 6.36. Przykład użycia funkcji daty i godziny (źródło: opracowanie własne)

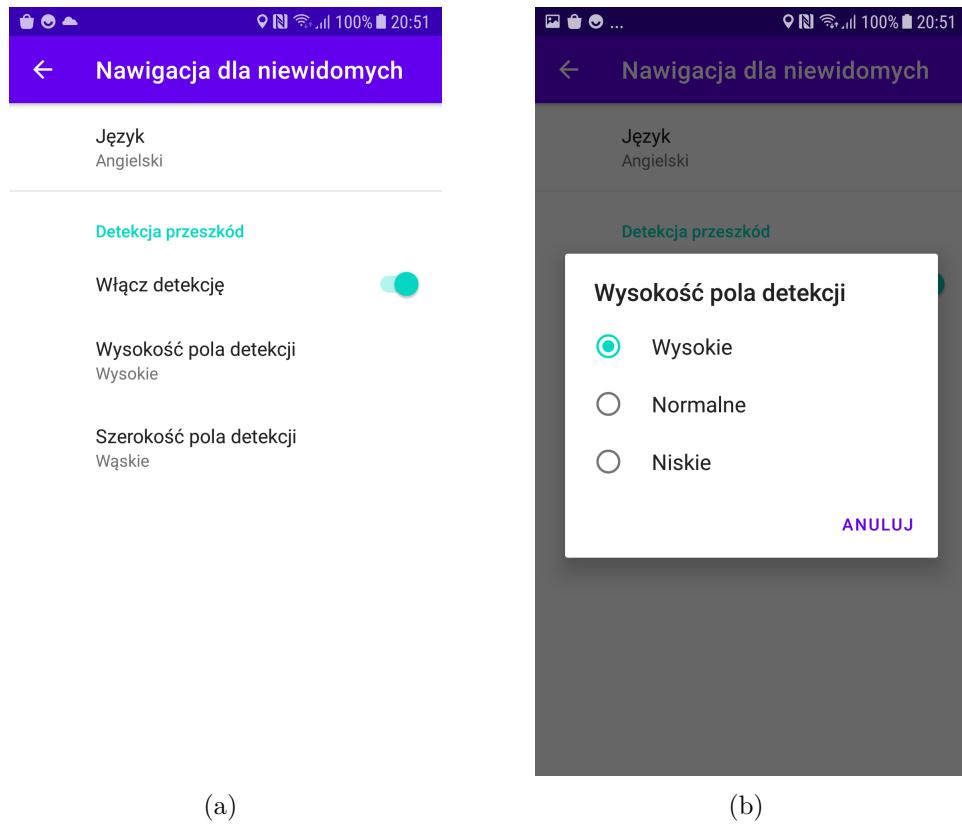
6.8. Interfejs ustawień aplikacji

Na rys. 6.37 przedstawiono widok ustawień w angielskiej wersji językowej. Język tekstów na ekranie jest zależny od języka ustawionego w telefonie. Jeśli język w telefonie jest inny niż angielski czy polski, to domyślnym językiem interfejsu jest język angielski. Język ustawiony w aplikacji wpływa jedynie na język komend głosowych oraz syntezatora mowy.



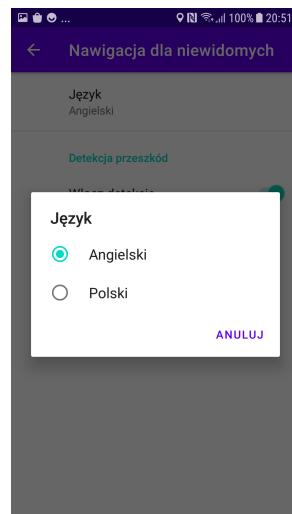
Rys. 6.37. Ustawienia - wersja angielska (źródło: opracowanie własne)

Na rys. 6.38 przedstawiono ten sam widok ustawień co na rys. 6.37, lecz tym razem w wersji polskiej. Zmiany tej dokonano w ustawieniach telefonu, w aplikacji jak widać ustawienia języka komend nadal ustawione są na język angielski. Wiąże się to ze sposobem tworzenia aplikacji za pomocą Android Studio. Tworzy się odpowiednio 2 pliki .xml zawierające teksty wszystkich elementów UI, jeden w wersji angielskiej drugi w wersji polskiej. Aplikacja automatycznie wykrywa język ustawiony w telefonie i analogicznie wybiera odpowiedni plik .xml z tekstami.



Rys. 6.38. Ustawienia - wersja polska (źródło: opracowanie własne)

Na rys. 6.39 przedstawiono widok zmiany języka aplikacji.



Rys. 6.39. Ustawienia języka - wersja polska (źródło: opracowanie własne)

6.9. Podsumowanie

Na podstawie projektu udało się zaimplementować działającą aplikację mobilną dla niewidomych i słabowidzących. Aplikacja dokonuje detekcji przeszkód w różnych warunkach oświetleniowych, choć im gorsze oświetlenie, tym mniej krawędzi zostanie wykryte. Po detekcji przeszkody użytkownik informowany jest dźwiękiem w stereo o głośności zależnej od odległości od obiektu. Estymacja odległości opiera się na założeniu, że bliskie obiekty znajdują się na dole klatki z kamery, te górze obrazu są uznawane za położone dalej. Zaimplementowano moduł odpowiedzialny za rozpoznawanie komend głosowych. Aplikacja umożliwia także uruchomienie nawigacji Google Maps do określonego miejsca za pomocą komend głosowych. Dostępne są także funkcje takie jak kompas, lokalizacja, data i godzina, wszystkie dostępne z poziomu interfejsu graficznego, jak i za pomocą komend głosowych. W ustawieniach można zmienić język aplikacji na angielski lub polski, włączyć i wyłączyć detekcję przeszkód w celu zmniejszenia zużycia baterii oraz zmieniać wielkość pola detekcji. Aby sprawdzić, czy zaimplementowane funkcje działają poprawnie i czy spełniają one założone wymagania, należy przeprowadzić odpowiednie testy, o których więcej w następnym rozdziale.

7. Testowanie aplikacji

Po implementacji aplikacji należy dokonać odpowiednich testów, by sprawdzić, czy wszystkie wymagania funkcjonalne i niefunkcjonalne zostały spełnione. W tym rozdziale omówione zostaną testy wydajnościowe i akceptacyjne.

7.1. Testowanie wydajności

W ramach testowania wydajności przetestowane zostaną następujące parametry:

1. Liczba klatek na sekundę - detekcja przeszkód
2. Zużycie procesora
3. Zużycie pamięci RAM
4. Zużycie baterii

Testowane urządzenia

1. Huawei P8 Lite, Android 6.0, rok produkcji: 2016, procesor: Kirin 620, rozdzielcość video: 1920x1080, bateria: Li-Po 2200 mAh
2. Samsung Galaxy A5, Android 7.0, rok produkcji: 2017, procesor: Exynos 7880, rozdzielcość video: 1920x1080, bateria: Li-Ion 3000 mAh
3. Samsung Galaxy A50, Android 10.0, rok produkcji: 2020, procesor: Exynos 9610, rozdzielcość video: 1920x1080, bateria: Li-Ion 4000 mAh

Tabela 7.1. Liczba klatek na sekundę (źródło: opracowanie własne)

	średnia	minimalna	maksymalna
Huawei P8 Lite	14	8	26
Samsung Galaxy A5	16	12	20
Samsung Galaxy A50	25	11	30

Czas badania: 10 min

Wnioski:

W tabeli 7.1 przedstawiono badania liczby klatek na sekundę dla detekcji przeszkód. Huawei P8 Lite będąc telefonem z niskiej półki cenowej i odstaje wydajnościowo od reszty testowanych urządzeń. Dziwi więc jego stosunkowo dobry wynik, jeśli chodzi o maksymalną liczbę klatek. Bierze się on stąd, że OpenCV wydajnie zmniejsza rozdzielcość obrazu dla Huawei'a, a dla Galaxy A5 tego nie robi. Pomimo że oba telefony nagrywają z taką samą rozdzielcością = 1920x1080, to do OpenCV udostępnia klatkę dla P8 lite o rozmiarze 960x720, w Galaxy A5 i A50 zaś 1920 1080.

Jako że klatka i tak musi być zmniejszona do ok 500x400, by zachować wydajność przy kosztownej operacji Hough (im więcej pikseli tym więcej czasu zajmuje).

Zauważono też zależność, że im więcej wykrytych krawędzi, tym mniejsza liczba klatek/s i tym większe zużycie procesora ~ 15%. Co jest logicznym następstwem tego, że im więcej krawędzi tym należy wykonać więcej iteracji przecięcia z polem detekcji. Jednakże zależność ta występuje jedynie dla P8 Lite i Galaxy A5. Dla Galaxy A50 zaś im jaśniejszy obraz, więcej wykrytych krawędzi, większe zużycie procesora, tym więcej klatek na sekundę: 23-26 kl./s. Im ciemniejszy obraz i mniej wykrytych krawędzi, mniejsze zużycie procesora, tym mniej klatek: 10-15 kl./s. Możliwe, że przy ciemniejszym obrazie Galaxy A50 próbuje rozjaśnić/wyostrzyć obraz co spowalnia buforowanie obrazu do OpenCV i do detekcji przeszkodek, jednakże jest to jedynie hipoteza. Reszta urządzeń przy zasłoniętej kamerze, logicznie ma większą ilość kl./s, tylko nie Galaxy A50.

Tabela 7.2. Zużycie procesora, pamięci RAM, baterii (źródło: opracowanie własne)

	zużycie procesora [%]	zużycie pamięci RAM [MB]	pojemność baterii [mAh]	zużycie baterii [%/h]*	estymowana liczba godzin do rozładowania
Huawei P8 Lite	13 %	98 MB	2200 mAh	60%	1.66 h
Samsung Galaxy A5	15 %	110 MB	3000 mAh	36%	2.77 h
Samsung Galaxy A50	13 %	166 MB	4000 mAh	30%	3.33 h
Samsung Galaxy A50 (wyłączona detekcja)	1 %	92 MB	4000 mAh	12%	8.33 h

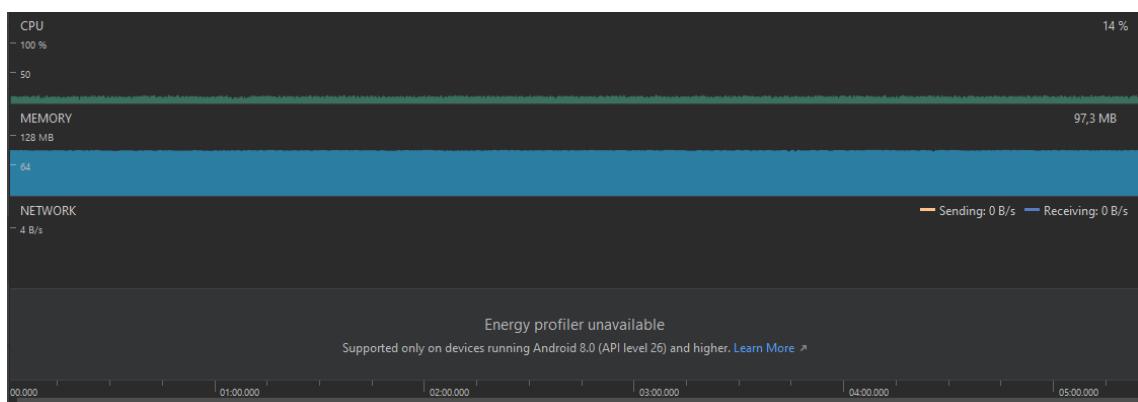
*Czas badania: 10 min

Wnioski:

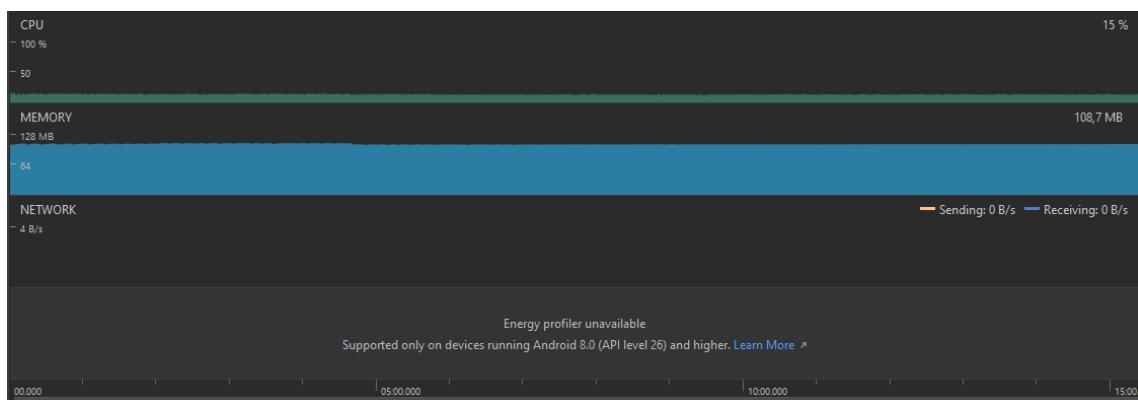
W tabeli 7.2 przedstawiono badania wydajności pod kątem zużycia procesora, pamięci, baterii dla różnych urządzeń. Zauważać można podobne zużycie procesora i pamięci RAM niezależnie od półki cenowej i wieku telefonu. Różnice widać przy zużyciu baterii i choć niewielka pojemność baterii Huawei P8 Lite niewątpliwie wpływa na jej zużycie, to Samsung Galaxy A5 mając baterię 1.35 razy bardziej pojemną,

wytrzymuje 1.65 razy dłużej. Niewielka różnica w zużyciu jest pomiędzy Galaxy A5 i Galaxy A50 mając na uwadze baterię o 1000 mAh mniejszą w przypadku Galaxy A5 oraz 3 lata dodatkowej eksploatacji! Należy też dodać, że obie baterie są tego samego typu - Litowo-Ionowe, uważane za bardziej żywotne w porównaniu do Litowo-Polimerowych. Z badania tego można wywnioskować, że bateria baterii nierówna, nawet jeśli są tego samego typu. Widać także, jak duży wpływ na zużycie baterii ma detekcja przeszkód.

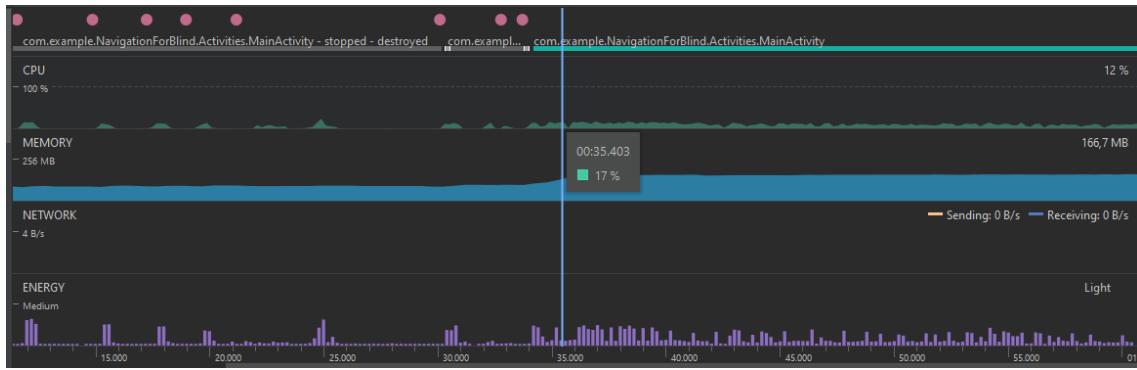
Wykresy wydajności



Rys. 7.1. Wykres wydajności dla Huawei P8 Lite (źródło: opracowanie własne)



Rys. 7.2. Wykres wydajności dla Samsunga Galaxy A5 (źródło: opracowanie własne)



Rys. 7.3. Wykres wydajności dla Samsunga Galaxy A50 (źródło: opracowanie własne)

Na rys. 7.1 i rys. 7.2 przedstawiono wykresy wydajności z włączoną detekcją przeszkoł. Na przestrzeni kilku minut widać stabilne działanie aplikacji z zużyciem procesora na poziomie 15% i i zużyciem pamięci RAM na poziomie 100-110MB, bez wycieków pamięci. Na rys. 7.3 przedstawiono wykres wydajności dla Galaxy A50, gdzie detekcja była w pierwszej połowie testu wyłączona, a pojedyncze wzrosty na wykresie zużycia procesora odpowiadają działaniom użytkownika, klikającego przyciski lokalizacji, kompasu etc. Przy wyłączonej detekcji przeszkoł widać także znacznie zmniejszone zużycie baterii na najniższym wykresie. Zauważać można korelację pomiędzy wykresem zużycia procesora, a zużyciem baterii.

7.2. Testy akceptacyjne

Wstęp

W celu sprawdzenia, czy wymagania funkcjonalne i historyjki użytkownika zostały spełnione, należy przeprowadzić testy akceptacyjne. Testy alfa z racji jednoosobowego charakteru pracy wykonywane były przez tę samą osobę, która dokonała implementacji. Testy beta przeprowadzono z udziałem trzech niezależnych, w pełni widzących ochotników. Przed pierwszą fazą testu ochotnicy zostali poinstruowani o charakterze aplikacji i funkcjach, jakie ona oferuje. Do sprawdzenia jakości zaimplementowanych funkcji skorzystano z poniższych scenariuszy testowych skonstruowanych na podstawie bloga tsjs.pl [23]. Scenariusze testowe nie objęły wszystkich możliwych scenariuszy alternatywnych z uwagi na ograniczony czas uczestników testów beta. W ramach testów alfa sprawdzono wszystkie możliwe scenariusze alternatywne, także te niezawarte w scenariuszach testowych.

Scenariusze testowe

Scenariusz testowy: Wydawanie komend głosowych

Aktor: Użytkownik

Warunki wstępne: Połączenie internetowe, zezwolenie aplikacji na używanie mikrofonu

Kroki do wykonania:

1. Użytkownik klika przycisk (*ikona mikrofonu*) do wydawania komend głosowych, a aplikacja wydaje dźwięk rozpoczęcia nasłuchiwanego.
2. Użytkownik mówi odpowiednią komendę.

Rezultat: Aplikacja rozpoznaje komendę i włącza funkcję odpowiadającą komendzie.

Scenariusz testowy: Detekcja przeszkód

Aktor: Użytkownik

Warunki wstępne: Zezwolenie aplikacji na używanie kamery

Kroki do wykonania:

1. Użytkownik zbliża się do krawędzi chodnika lub przeszkody.

Rezultat: Aplikacja wykrywa przecięcie krawędzi chodnika lub przeszkody z polem detekcji lub wykrywa krawędź zawartą wewnętrz pola detekcji i wydaje dźwięk w stereo zależnie od umiejscowienia przeszkody na obrazie.

Scenariusz testowy: Ustalenie lokalizacji

Aktor: Użytkownik

Warunki wstępne: Zezwolenie aplikacji na używanie danych o lokalizacji

Kroki do wykonania:

1. Użytkownik klika przycisk informowania o aktualnej lokalizacji (*ikona markera*).

Rezultat: Aplikacja wykonuje odwrotne geokodowanie na podstawie aktualnych współrzędnych, otrzymując od Geocoder API aktualny adres. Syntezator mowy głosowo informuje użytkownika o lokalizacji w odpowiednim języku.

Scenariusz testowy: Ustalenie kierunku geograficznego

Aktor: Użytkownik

Warunki wstępne: Brak

Kroki do wykonania:

1. Użytkownik klika przycisk informowania o kierunku geograficznym (*ikona kompasu*), w jakim użytkownik jest skierowany.

Rezultat: Aplikacja odczytuje dane z sensora magnetycznego i akcelerometru i określa aktualny kierunek geograficzny telefonu. Syntezator mowy głosowo informuje

użytkownika o kierunku w odpowiednim języku.

Scenariusz testowy: Informacja o godzinie

Aktor: Użytkownik

Warunki wstępne: Brak

Kroki do wykonania:

1. Użytkownik klika przycisk informowania o aktualnej godzinie (*ikona zegara*).

Rezultat: Syntezator mowy głosowo informuje użytkownika o aktualnej godzinie i minucie w odpowiednim języku.

Scenariusz testowy: Informacja o dacie

Aktor: Użytkownik

Warunki wstępne: Brak

Kroki do wykonania:

1. Użytkownik klika przycisk informowania o aktualnej dacie (*ikona kalendarza*).

Rezultat: Syntezator mowy głosowo informuje użytkownika o aktualnej dacie w odpowiednim języku.

Scenariusz testowy: Nawigacja Google Maps

Aktor: Użytkownik

Warunki wstępne: Połączenie internetowe

Kroki do wykonania:

1. Użytkownik klika przycisk do wydawania komend głosowych, po czym aplikacja wydaje dźwięk rozpoczęcia nasłuchiwanego.
2. Użytkownik mówi komendę "nawigacja". Aplikacja po rozpoznaniu komendy pyta użytkownika o miejsce docelowe (np. przystanek tramwajowy świdnicka).
3. Użytkownik mówi, jakie jest miejsce docelowe. Po czym aplikacja pyta się użytkownika czy to na pewno jest ta lokalizacja.
4. Użytkownik potwierdza, mówiąc 'zgadza się' lub 'tak', lub 'owszem'.

Rezultat: Aplikacja włącza zewnętrzną aplikację mobilną Google Maps w trybie nawigacji.

Scenariusz testowy: Wybór języka

Aktor: Użytkownik

Warunki wstępne:

Kroki do wykonania:

1. Użytkownik klika przycisk '*Język*'. Po czym na ekranie pojawia się okno z językami do wyboru.
2. Użytkownik wybiera język.

Rezultat: Język komend głosowych i synteza mowy został zmieniony.

Scenariusz testowy: Włącz / wyłącz detekcję przeszkodek

Aktor: Użytkownik

Warunki wstępne:

Kroki do wykonania:

1. Użytkownik klika przycisk typu przełącznik (switch) '*Włącz detekcję*'.

Rezultat: W zależności od tego czy przełącznik był wcześniej włączony czy wyłączony, detekcja przeszkodek zastała odpowiednio wyłączoną lub włączoną.

Scenariusz testowy: Zmiana wysokości pola detekcji przeszkodek

Aktor: Użytkownik

Warunki wstępne:

Kroki do wykonania:

1. Użytkownik klika przycisk '*Wysokość pola detekcji*'. Po czym na ekranie pojawia się okno z wielkościami do wyboru: wysokie, normalne, niskie.
3. Użytkownik wybiera odpowiednią wysokość.

Rezultat: Wybrana wysokość zostaje zapisana na stałe i po przejściu do głównego widoku wysokość pola została odpowiednio zmieniona. Po wyłączeniu i włączeniu aplikacji wysokość pola detekcji została zachowana.

Scenariusz testowy: Zmiana wysokości pola detekcji przeszkodek

Aktor: Użytkownik

Warunki wstępne:

Kroki do wykonania:

1. Użytkownik klika przycisk '*Szerokość pola detekcji*'. Po czym na ekranie pojawia się okno z wielkościami do wyboru: szerokie, normalne, wąskie.
3. Użytkownik wybiera odpowiednią szerokość.

Rezultat: Wybrana szerokość zostaje zapisana na stałe i po przejściu do głównego widoku szerokość pola została odpowiednio zmieniona. Po wyłączeniu i włączeniu aplikacji szerokość pola detekcji została zachowana.

7.3. Podsumowanie:

W ramach wielokrotnych testów alfa wykonywanych na podstawie części scenariuszy testowych od początku do końca implementacji, wykazano błędy, których większość udało się naprawić. Jednego jednak nie udało się naprawić np. określenie aktualnej lokalizacji na telefonach Huawei z pomocą FusedLocationProviderClient [7]. Wynika to z faktu, że Huawei od dłuższego czasu nie wspiera Google Play Services, tym samym niedostępne jest API do lokalizacji dostarczane przez Google.

W ramach jednokrotnych testów beta na podstawie scenariuszy testowych wszystkie funkcje zostały przetestowane z wynikiem pozytywnym (pomijając urządzenia Huawei). Testerzy nie mieli problemów z rozpoznaniem funkcji, jakie dane przyciski pełnią pomimo braku opisów pod ikonami. Choć jest to mało istotne z uwagi na fakt, że osoby niewidome nie widzą ekranu i używając czytnika ekranu, poruszają się po przyciskach, tekstach, jedno po drugim. Z tego powodu wykonano testy interfejsu aplikacji z użyciem czytnika ekranu, gdzie uczestnicy mieli zawiązane oczy. Dzięki niewielkiej ilości przycisków i tekstów interfejs okazał się prosty w nawigowaniu nawet dla osób na co dzień niekorzystających z czytnika ekranu. Choć należy zauważać, że nie szli na ślepo po mieście z uwagi na ich bezpieczeństwo. W wyniku testów stwierdzono, że ustalenie aktualnej lokalizacji działa z błędem 15-20 metrów. W budynkach dokładność ta znacznie spada, nawet do 50-100 metrów. Testerzy zauważali też, że krawędź chodnika nie zawsze jest wykrywana. To samo tyczy się samych przeszkód, jeśli kolorem czy naświetleniem nie różnią się od otoczenia, to często nie zostawały wykryte. Sam system informowania o przeszkodach w postaci cichszego i głośniejszego dźwięku był oceniany pozytywnie z zastrzeżeniami, że zmiana częstotliwości dźwięku byłaby bardziej intuicyjna (z czym się zgodziłem, ale były problemy z implementacją tej funkcji). W kolejnym rozdziale przedstawiono podsumowanie pracy.

8. Zakończenie

Zrealizowane prace

W tej pracy przedstawiono istniejące rozwiązania, które pomagają lub mogłyby pomóc niewidomym w poruszaniu się po mieście. Przedstawiono ich wady oraz zalety. Przedstawiono projekt w postaci historyjek użytkownika, diagramu przypadków użycia, scenariuszy przypadków użycia. Następnie na podstawie projektu zaimplementowano aplikację mobilną, spełniając określone wymagania funkcjonalne i niefunkcjonalne. Celem pracy był projekt i implementacja aplikacji mobilnej dla niewidomych służącej do nawigacji po mieście z uwzględnieniem nawigacji, jak i detekcji przeszkód czy krawędzi chodnika.

Cel ten został spełniony, choć należy zaznaczyć, że nawigacja w formie własnej implementacji nawigacji Google Maps została zastąpiona włączeniem zewnętrznej aplikacji Google Maps, która również dobrze działa w tle.

W samej pracy skupiono się na detekcji prostych krawędzi z założeniem, że przeszkody i krawędzie chodnika zawsze są proste. Niestety to podejście jest naiwne i nie wykrywa wszystkich przeszkód, tych o krzywych, kolistych krawędziach. Nie wykrywa przeszkód, gdy te naświetlaniem i kolorem podobne są do otoczenia. Błędnie wykrywa ostre cienie jako przeszkody. Także, gdy krawędź chodnika nie jest wyraźnie zaznaczona, np. za pomocą krawężnika to nie zawsze jest wykrywana. Patrząc jednak na całość pracy, otrzymany produkt prezentuje wartość i mógłby być gotowy do wdrożenia po wprowadzeniu odpowiednich poprawek do implementacji aplikacji.

Trudności napotkane podczas realizacji pracy

W trakcie pracy napotkano problem wycieków pamięci przy detekcji przeszkód. Okazało się, że alokacja nowych obrazów i tablic w pętli iterującej 30 razy na sekundę źle wpływa na pamięć. Wymuszając tym samym częstsze czyszczenia jej przez odśmiecacz pamięci (Garbage Collector), co mogło mieć negatywny wpływ także na liczbę klatek na sekundę. Rozwiązano ten problem poprzez alokacje obrazów i tablic poza pętlami przy inicjalizacji klasy ObstacleDetection.

Kierunki dalszych prac

W ramach dalszych pracy przede wszystkim należy zatrzymać usuwanie cieni z obrazu [19]. Cienie znacznie zmniejszają dokładność detekcji w słoneczne dni, których w większości krajów jest niemało. Kolejną funkcją, jaką można by zaimplementować, jest wykrywanie obiektów za pomocą Tensorflow Lite. Przez wykrywanie obiektów

mam na myśli rozpoznanie dokładnej nazwy obiektu np. człowiek, samochód oraz jej dokładne położenie na obrazie. To pomogłoby z detekcją ludzi, którzy często nie są wykrywani przez algorytm detekcji z uwagi na ubrania, które nie przylegają idealnie do ciała i nie tworzą idealnie prostych krawędzi.

Także przy pomocy Tensorflow Lite można spróbować zaimplementować model do estymacji głębokości obrazu za pomocą konwolucyjnej sieci neuronowej [20]. Kolejną opcją jest skorzystanie z Depth API z biblioteki ARCore [1], która oferuje detekcję głębokości za pomocą jednej linijki kodu, jednakże tu barierą jest wysoka cena telefonów wspieranych przez to API.

Kolejnym rozwinięciem tej aplikacji byłaby implementacja nawigacji za pomocą API od Google Maps czy Mapboxa. Należy też zaznaczyć, że Directions API od Google Maps do znajdywania trasy jest płatne. Przed napisaniem tej pracy dołączylem do pewnej grupy dla niewidomych na portalu Facebook, by dowiedzieć się, co sprawia im największy problem, gdy poruszają się po mieście. I poza wykrywaniem samochodów, słupków, schodów, hulajnóg, krawędzi ulicy to nawigacja pojawiła się również często. Korzystając API Google można mieć dostęp do dokładnego położenia następnego zakrętu. Znając swoje położenie oraz aktualny kierunek geograficzny można by nawigować do kolejnego zakrętu na bieżąco.

Podsumowując, rozwiązanie przeze mnie przedstawione, nie jest idealne i ma szereg wad. Nie działa najlepiej w słabym oświetleniu, nie wykrywa krawędzi przeszkód, krawędzi chodnika, gdy te zlewają się z otoczeniem. To, że obiekt wyróżnia się kolorystycznie np. żółty słupek, nie musi oznaczać, że zostanie on wykryty. Dzieje się tak, ponieważ obraz RGB zamieniany jest na szary i w takiej sytuacji żółty słupek staje się szary, więc może zlewać się z szarym chodnikiem. Liczy się jedynie jasność koloru danego obiektu, ile światła on odbija w stosunku do otoczenia. Wykrywanie przeszkód ograniczone jest jedynie do przeszkód zawierających proste, długie krawędzie. Przeszkody z łagodnymi, okrągłymi krawędziami nie będą wykrywane. Jednakże większość przeszkód, które można spotkać na chodniku, zazwyczaj te proste krawędzie ma np. krawędź chodnika, samochód, słupek, lampa, drzewo, rower, hulajnoga, śmiertnik, kontener etc. Niewątpliwie zaletą tego rozwiązania są niskie wymagania sprzętowe, wspierając tym samym tanie telefony, dając w zamian detekcję przeszkód, która w większości przypadków działa poprawnie, a także kompas i aktualny adres, datę i godzinę. Rozwiązania używające bardziej zaawansowanych technik detekcji np. przy pomocy sieci konwolucyjnych do detekcji ludzi czy samochodów lub do estymacji głębokości obrazu wymagają znacznie droższych telefonów do optymalnego działania.

Bibliografia

- [1] ARCore, Google. Depth api overview for android. <https://developers.google.com/ar/develop/java/depth/overview>.
- [2] D. Sundararajan. *Digital Image Processing: A Signal Processing and Algorithmic Approach*. Springer, 2017.
- [3] D. Castells, J.M.F. Rodrigues, and J.M.H. du Buf. Obstacle detection and avoidance on sidewalks. https://www.researchgate.net/publication/216435190_Obstacle_detection_and_avoidance_on_sidewalks, 2010.
- [4] Dr. M. Gevorgyan, A. Mamikonyan, M. Beyeler. *OpenCV 4 with Python Blueprints*. Packt Publishing, 2020.
- [5] Google. Geocoding API. <https://developers.google.com/maps/documentation/geocoding/overview>.
- [6] Google, JetBrains. Android Studio. <https://developer.android.com/studio>.
- [7] Google, JetBrains. FusedLocationProviderClient. <https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient>.
- [8] Google, JetBrains. Geocoder. <https://developer.android.com/reference/android/location/Geocoder>.
- [9] Google, JetBrains. LocationListener. <https://developer.android.com/reference/android/location/LocationListener.html>.
- [10] Google, JetBrains. SpeechRecognizer. <https://developer.android.com/reference/android/speech/SpeechRecognizer>.
- [11] Google, JetBrains. TextToSpeech. <https://developer.android.com/reference/android/speech/tts/TextToSpeech>.
- [12] Intel Corporation, Willow Garage, Itseez. OpenCV. <https://opencv.org/>.
- [13] J. F. Canny. Canny, detekcja krawędzi. https://docs.opencv.org/master/dad22/tutorial_py_canny.html, 1986.
- [14] J. F. Canny. A computational approach to edge detection. <https://ieeexplore.ieee.org/document/4767851>, 1986.
- [15] J. Howard. Common design patterns for android with kotlin, 2017.
- [16] J. Howse, J. Minichino. *Learning OpenCV 4 Computer Vision with Python 3*. Packt Publishing, 2020.
- [17] JetBrains. Kotlin Programming Language. <https://kotlinlang.org/>.
- [18] J. José, M. Farrajota, J.M.F. Rodrigues, and J.M.H. du Buf. A vision system for detecting paths and moving obstacles for the blind. https://www.researchgate.net/publication/216435222_A_vision_system_for_detecting_paths_and_moving_obstacles_for_the_blind, 2010.

- [19] K. Irie, A. E. McKinnon, K. Unsworth, I. M. Woodhead. Shadow removal for object tracking in complex outdoor scenes. <https://researcharchive.lincoln.ac.nz/handle/10182/1853>, 2014.
- [20] M. Poggi, F. Aleotti, F. Tosi, S. Mattoccia. Towards real-time unsupervised monocular depth estimation on cpu. <https://arxiv.org/abs/1806.11430>, 2018.
- [21] P. E. Hart. How the hough transform was invented. https://www.researchgate.net/publication/224586582_How_the_Hough_Transform_Was_Invented, 2009.
- [22] T. Panek. How project guideline gave me the freedom to run solo. <https://blog.google/outreach-initiatives/accessibility/project-guideline/>.
- [23] T. Statkowski. Dobry scenariusz przyjacielem testera. <https://tsjs.pl/scenariusz-testowy/>, 2019.
- [24] W. Sugiyama. Voice guidance in maps, built for people with impaired vision. <https://blog.google/products/maps/better-maps-for-people-with-vision-impairments>, 2019.

Spis rysunków

3.1. Interfejs graficzny Google Maps (źródło: https://support.google.com/maps/thread/14375376?hl=en)	10
3.2. Interfejs graficzny BlindSquare (źródło: https://www.blindsquare.com/)	11
3.3. Interfejs graficzny Be My Eyes (źródło: https://www.bemyeyes.com/)	11
3.4. Interfejs graficzny Lazarillo GPS for Blind (źródło: https://lazarillo.app/)	12
3.5. Laska z sensorem ultradźwiękowym (źródło: https://wewalk.io/en/)	13
3.6. Interfejs graficzny aplikacji WeWalk (źródło: https://wewalk.io/en/)	13
5.1. Diagram przypadków użycia (źródło: opracowanie własne)	20
5.2. Diagram pakietów (źródło: opracowanie własne)	26
6.1. Struktura projektu (źródło: opracowanie własne)	30
6.2. Obraz wygładzony, z ciemnymi krawędziami pomiędzy kostkami chodnikowymi (źródło: opracowanie własne)	32
6.3. Obraz z rozszerzonymi jasnymi pikselami (dilation), bez krawędzi pomiędzy kostkami brukowymi (źródło: opracowanie własne)	32
6.4. Obraz wygładzony, z ciemnymi krawędziami pomiędzy kostkami chodnikowymi (źródło: opracowanie własne)	33
6.5. Obraz z rozszerzonymi jasnymi pikselami (dilation), bez krawędzi pomiędzy kostkami brukowymi (źródło: opracowanie własne)	33
6.6. Obraz szary (źródło: opracowanie własne)	34
6.7. Obraz wygładzony filtrem medianowym i uśredniającym (źródło: opracowanie własne)	35
6.8. Obraz po rozszerzeniu jasnych pikseli (źródło: opracowanie własne)	35
6.9. Obraz po detekcji krawędzi Canny (źródło: opracowanie własne)	36
6.10. Detekcja prostych krawędzi - Hough (źródło: opracowanie własne)	36
6.11. Wysokość pola detekcji, po skierowanie telefonu równolegle do ziemi (źródło: opracowanie własne)	37
6.12. Wysokość pola detekcji po skierowaniu telefonu prostopadle do podłoża (źródło: opracowanie własne)	37
6.13. Ustawienia wysokości pola - wysokie, ustawienia szerokości - wąskie (źródło: opracowanie własne)	38
6.14. Ustawienia wysokości pola - normalne, ustawienia szerokości - normalna (źródło: opracowanie własne)	39
6.15. Ustawienia wysokości pola - normalne, ustawienia szerokości - szerokie (źródło: opracowanie własne)	39
6.16. Detekcja samochodów (źródło: opracowanie własne)	40

6.17.	Detekcja słupków (źródło: opracowanie własne)	41
6.18.	Detekcja słupków - 2 (źródło: opracowanie własne)	41
6.19.	Detekcja krawędzi chodnika (źródło: opracowanie własne)	42
6.20.	Detekcja progu spowalniającego (źródło: opracowanie własne)	42
6.21.	Detekcja kontenera (źródło: opracowanie własne)	43
6.22.	Detekcja kontenera - 2 (źródło: opracowanie własne)	43
6.23.	Detekcja drzwi wejściowych (źródło: opracowanie własne)	44
6.24.	Detekcja blokady samochodowej (źródło: opracowanie własne)	44
6.25.	Detekcja ławki (źródło: opracowanie własne)	45
6.26.	Detekcja schodów (źródło: opracowanie własne)	45
6.27.	Przykład 1 - detekcja krawędzi nocą (źródło: opracowanie własne)	46
6.28.	Przykład 2 - detekcja krawędzi nocą (źródło: opracowanie własne)	46
6.29.	Przykład 3 - detekcja krawędzi nocą (źródło: opracowanie własne)	47
6.30.	Przykład 4 - detekcja krawędzi nocą (źródło: opracowanie własne)	47
6.31.	Przykład użycia funkcji nawigacji (źródło: opracowanie własne)	48
6.32.	Przykład użycia funkcji nawigacji cd. (źródło: opracowanie własne)	48
6.33.	Przykład użycia funkcji lokalizacji (źródło: opracowanie własne)	49
6.34.	Przykład 1 użycia funkcji kompasu (źródło: opracowanie własne)	49
6.35.	Przykład 2 użycia funkcji kompasu (źródło: opracowanie własne)	50
6.36.	Przykład użycia funkcji daty i godziny (źródło: opracowanie własne)	50
6.37.	Ustawienia - wersja angielska (źródło: opracowanie własne)	51
6.38.	Ustawienia - wersja polska (źródło: opracowanie własne)	52
6.39.	Ustawienia języka - wersja polska (źródło: opracowanie własne)	52
7.1.	Wykres wydajności dla Huawei P8 Lite (źródło: opracowanie własne)	56
7.2.	Wykres wydajności dla Samsunga Galaxy A5 (źródło: opracowanie własne)	56
7.3.	Wykres wydajności dla Samsunga Galaxy A50 (źródło: opracowanie własne)	57

Spis tabel

1.1.	Słownik pojęć (źródło: opracowanie własne)	5
7.1.	Liczba klatek na sekundę (źródło: opracowanie własne)	54
7.2.	Zużycie procesora, pamięci RAM, baterii (źródło: opracowanie własne)	55