POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Automatyka i Robotyka (AIR)
SPECJALNOŚĆ: Technologie Informacyjne w Automatyce (ART)

PRACA DYPLOMOWA INŻYNIERSKA

Rozpoznawanie znaków tekstowych drogowych w systemie osób niewidomych

The text road signs recognition in the system of blind people

AUTOR:
Karolina Raczyńska

PROWADZĄCY PRACĘ:
dr inż. Łukasz Jeleń

OCENA PRACY:

WROCŁAW 09.12.2016

Spis treści

Rozdział 1

Wstęp

Postęp technologiczny w dwudziestym pierwszym wieku doprowadził do tego, że wiele rzeczy, które wydawały się kiedyś niemożliwe dziś stoją na porządku dziennym. Problemy, które kiedyś były nie do rozwiązania, przeciwności, które przeszkadzały w normalnym życiu w dzisiejszych czasach powoli zanikają. Postęp ten wpłynął na to, że ludziom z niepełnosprawnością można pomóc żyć w taki sposób, w jaki żyją ludzie pełnosprawni. Rynek jest przepełniony aplikacjami, które pomagają ludziom niedowidzącym, korzystać z telefonów komórkowych. Istnieją aplikację czytające wiadomości tekstowe, nadesłane do posiadacza telefonu, a także takie które odwrotnie, zamieniają tekst mówiony na wiadomość. Pojawiają się różne możliwości powiększania tekstu, dostosowania właściwości telefonu do potrzeb konsumenta.

1.1 Cel pracy

Celem niniejszej pracy było stworzenie aplikacji mobilnej, na tyle intuicyjnej i łatwej w obsłudze, aby korzystać z niej mogły osoby niewidome oraz niedowidzące, a następnie opisanie jej, ze szczególnym zwróceniem uwagi na algorytmy użyte przy jej powstawaniu i przeanalizowanie ich działania, a także naświetlnie środowiska Android. Aplikacja powinna działać na jak najmniejszej liczbie urządzeń, starając się wszystkie swoje funkcje zawrzeć w jednym urządzeniu mobilnym - telefonie z systemem operacyjnym Android, który, na dzień dzisiejszy, jest najbardziej popularnym systemem na rynku. Koniecznym warunkiem ma być jednak to, że ów telefon musi posiadać sprawny aparat. Główną funkcjonalnością aplikacji miało być przetwarzanie zdjęć, na których znajdowały się tekstowe znaki drogowe na komunikat głosowy. Dzięki czemu poruszanie się po ulicy ludziom, mającym trudność z czytaniem drogowych znaków tekstowych, stałoby się łatwiejsze, a przede wszystkim bezpieczniejsze, z uwagi na to, że tego typu znaki przekazują wiadomości istotne dla podróży niezagrażającej ludzkiemu zdrowiu, a także życiu. Praca ta ma być dopełnieniem większego systemu wsparcia dla osób niewidomych, składającego się na rozpoznawanie tekstu mówionego i przetwarzanie elektrowibracji, stanowiącego rozwinięcie projektu zespołowego "Sztuczne Oko". Projekt ów opierał się na połączeniu telefonu z aparatem, który robił zdjęcie i przetwarzał je na krawędzie, kompresował i przesyłał na Raspberry Pi, a tam konwertował na sygnał audio, który prowokował elektrowibracje na panelu dotykowym, ładowanym przez przetwornicę, w miejscu gdzie zarejestrowano położenie palca. Elektrowibracje te miały imitować uczucie szorstkości pod palcami, doprowadzając do możliwości "widzenia" krawędzi odczytanych ze zdjęcia, opuszkami palców.

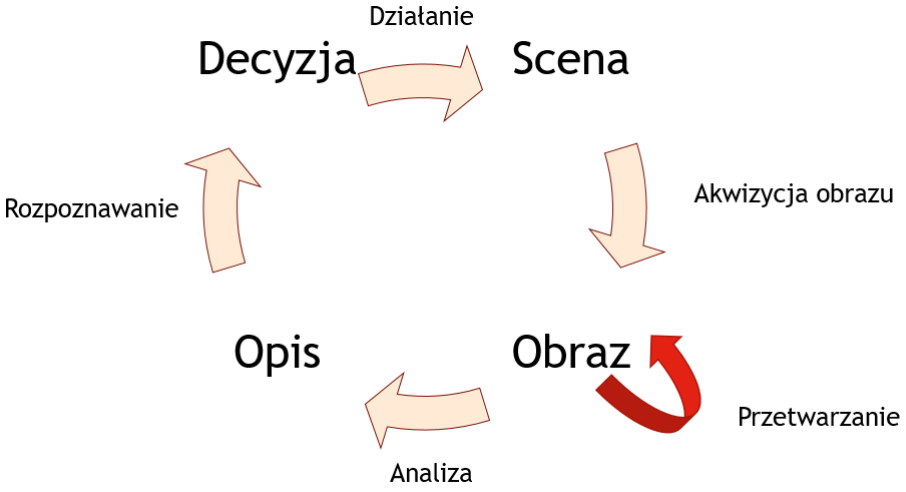
1.2 Zakres pracy

Praca podzielona jest na siedem rozdziałów, pierwszy to Wstęp, wprowadzający czytelnika w temat pracy, określający jej cel i zakres. Ostatni to zakończenie, gdzie podsumowuje się całą pracę i stwierdza czy jej cele zostały spełnione.
Drugi rozdział przedstawia podstawy teoretyczne, które zostały wykorzystane przy tworzeniu aplikacji. Przedstawiono podstawowe zagadnienia z dziedziny przetwarzania obrazu, opisano użyte algorytmy, ich działanie, zalety oraz wady, a także to jak prezentują się przy innych algorytmach rozwiązujących ten sam problem.
W trzecim rozdziale zawarto architekturę stworzonego systemu, zaprezentowano digramy UML: klas oraz przypadków użycia. Opisano strukturę programu oraz działanie poszczególnych klas, a także powiązania między nimi.
W czwartym rozdziale odniesiono się do implementacji programu. Przedstawiono środowisko w którym została napisana i zaprezentowano jej najważniejsze komponenty. Pojawiły się również przykłady kodu, który został napisany na potrzeby pracy wraz z wytłumaczeniem. Opisano również zewnętrzne biblioteki, które zostały użyte.
W piątym rozdziale pokazano działanie aplikacji, dołączony screeny ekranu urządzenia mobilnego, na którym aplikacja była testowana. Dołączono opis wszystkich funkcjonalności oraz możliwych widoków.

Rozdział 2

Przetwarzanie obrazu

Obraz to między innymi źródło informacji - nie tylko wzrokowej. Aby uzyskać dane, które są w nim zapisane, należy go odpowiednio przetworzyć. W dziedzinie nauki, jaką jest przetwarzanie obrazów, wyróżniamy kilka etapów działania. Uproszczony schemat został przedstawiony na rysunku poniżej. Poniżej zostały omówione poszczególne etapy.



Rysunek 2.1: Schemat cyklu przetwarzania obrazu.

1. Akwizycja obrazu – zamiana energii świetlnej od każdej punktu sceny na sygnał elektryczny, czyli zbieranie danych o każdym pikselu. Scena jaką obserwujemy jest funkcją ciągłą, przy przetwarzaniu jej na obraz cyfrowy, pewna część danych zostaje stracona. Typowo korzysta się z dwóch sposobów pozyskiwania sygnały dyskretnego z sygnału ciągłego i jest to: kwantyzacja oraz próbkowanie. Kwantyzacja jest to pobieranie danych w funkcji wartości funkcji – w zależności od niej przyporządkowuje się próbkę do odpowiedniego przedziału, czyli kwantu, natomiast próbkowanie polega na zbieraniu danych w funkcji czasu.

2. Przetwarzanie obrazu – zmniejszenie obrazu, eliminacja zakłóceń, filtracja wstępna, której celem jest wyeksponowanie na obrazie ważnych cech, takich jak krawędzie lub duże jednokolorowe obiekty. Do tego celu stosuje się np. transformację obrazu kolorowego na skalę odcieni szarości, tudzież na bitmapę składającą się jedynie z dwóch wartości.

3. Analiza obrazu – wydobycie wcześniej wyeksponowanych cech, w celu ich późniejszego rozpoznania, już nie jako kształtu ale jako konkretnej informacji. Wynikiem tego etapu nie jest już obraz, a dane w postaci liczbowej lub statystycznej.

4. Decyzja – rozpoznanie obrazu, zanalizowanie cech, pozyskanie informacji, a przede wszystkim klasyfikacja[1].

Wszystkie te etapy dzieją się w systemie wizyjnym do którego można zaliczyć np. komputer, telefon, aparat fotograficzny, a do jego podstawowych funkcji należy:

- Przyjęcie obrazu,
- zapisanie obrazu,
- właściwa obróbka obrazu,
- wyświetlenie obrazu[2];

2.1 Przetwarzanie do skali szarości

RGB, jest modelem przestrzeni barw, w którym każdy piksel opisują trzy wartości: R - intensywność koloru czerwonego, G - intensywność koloru zielonego oraz B - intensywność koloru niebieskiego. Obraz w odcieniach szarości to obraz w którym procentowy udział każdej z tych trzech instancji jest równy. Aby uzyskać taki wynik należy przeprowadzić pewne modyfikacje na wartościach pikseli. Przyjmuje się, że p to piksel wynikowy operacji przetwarzania do skali szarości. Najbardziej znane są trzy algorytmy:

- Desaturacja(ang. Desaturation) – metoda ta polega na tym, że z trzech wartości: intensywności koloru czerwonego, zielonego oraz niebieskiego w pikselu, zostaje wybrana wartość największa oraz najmniejsza, a wynikiem transformacji jest średnia arytmetyczna tych dwóch wartości.

$$p = \frac{max(R,G,B) + min(R,G,B)}{3} \tag{2.1}$$

- Średnia (ang. Average) – metoda ta, najprostsza, dzieli sumę wartości intensywności trzech kolorów przez trzy.

$$p = \frac{R+G+B}{3} \tag{2.2}$$

- Jasność (ang. Luminance) – metoda również opierająca się na średniej, ale w tym wypadku jest to średnia ważona. Ludzki wzrok jest najbardziej czuły na kolor zielony, właśnie dlatego intensywność zieleni ma największą wagę. Wyjściowy piksel przyjmuje wartość:

$$p = \frac{0.21R + 0.72G + 0.07B}{3}[3] \tag{2.3}$$

(a) Obraz oryginalny.

(b) Obraz po poddaniu desaturacji.

(c) Obraz po poddaniu uśrednieniu.

(d) Obraz po poddaniu operacji jasność.

Rysunek 2.2: Wynik przetwarzania obrazu na odcienie szarości.

2.2 Binaryzacja

Binaryzacja to operacja punktowa. Wynikiem tej operacji jest obraz binarny, czyli taki, w którym każdy z pikseli przyjmuje tylko jedną z dwóch wartości. Ideą tego zabiegu jest wyodrębnienie obiektu od tła przez nadanie pikselom tych dwóch instancją różnych wartości. Wejściowym obrazem jest zazwyczaj obraz przetworzony do skali szarości, którego piksele przyjmują wartości od 0 do 255. Najczęściej stosowaną metodą klasyfikacji piksela do jednej z dwóch klas jest progowanie – sprawdzenie czy wartość piksela przekracza zadany próg czy też nie i w zależności od wyniku tego zdania logicznego – przypisanie odpowiedniej wartości. Dla progu T, początkowej wartości piksela I(i,j) i wartości piksela po transformacji Y(i,j) mamy:

$$I(i,j) < T \longmapsto Y(i,j) = 0 \tag{2.4}$$

$$I(i,j) \geqslant T \longmapsto Y(i,j) = 1[4] \tag{2.5}$$

Zasadniczą trudnością tego podejścia jest celne dobranie wartości progowej T. Aby wybrać odpowiednią wartość progu można zastosować jeden z poniżej zamieszczonych sposobów:

- Na podstawie histogramu - należy użyć histogramu poziomów szarości analizowanego obrazu. Następnie, w przypadku gdy histogram jest dwumodalny, to wartość progową wybiera się z pomiędzy dwóch maksimów lokalnych. Gdy ma się do czynienia z mniej szczególnym przypadkiem zastosowanie tej metody może spowodować przekłamania w wyborze progu[5].

- Zastosowanie dwóch wartości progowych - rozważa się progowanie z dwoma wartościami $T_1 > T_2 > 0$. Jeżeli wartości piksela znajduje się pomiędzy wartościami T_1 i T_2 przyporządkowuje mu się wartość 0, w przeciwnym przypadku 1. Tego typu podejście ma sens, kiedy tło obrazu jest o różnych poziomach szarości. Można też uśrednić poziom szarości tła. Podobnym sposobem jest zastosowanie histerezy, tutaj analizuje się jednak jeszcze sąsiednie piksele[6].

- Automatyczny dobór progu - próg wybierany jest programowo, korzystając z pewnego algorytmu. Te metody bazują na pewnych założeniach, histogram musi być bimodadalny, obiekty muszą być w kolorach przeciwnych do kolorów tła. Do metod automatycznych zalicza się przedstawioną szerzej metodę Otsu[7].

2.2.1 Metoda Otsu

Metoda Otsu należy do rodziny metod optymalnych względem pewnej funkcji kryterialnej. W tym przypadku ową funkcja kryterialną jest wariancja wewnątrz+klasowa oraz międzyklasowa. Metoda ta jest bardzo prosta, w uogólnieniu używa momentów zerowego oraz pierwszego stopnia histogramu obrazu. Pierwszym etapem poszukiwania wartości progowej metodą Otsu jest normalizacja histogramu obrazu w skali szarośic. Zakłada się, że piksele przyjmują wartości z L poziomów, a do każdego i-tego poziomu jest przydzielana ni liczba pikseli. Przyjmując, że N to liczba wszystkich pikseli obrazu, rozkład prawdopodobieństwa tego histogramu przedstawiony jest następująco:

$$p_i = \frac{n_i}{N} \qquad p_i \geqslant 0, \sum_{i=1}^{L} p_i = 1 \tag{2.6}$$

Następnym krokiem jest dychotomia pikseli, czyli podział wszystkich elementów na dwie klasy względem pewnego progu t. Wtenczas pikseli, które przyjmują wartości mniejsze od zadanej wartości progowej t zostaną sklasyfikowane do klasy K_0, a te większe do klasy K_1. Wtedy prawdopodobieństwo tego, że badany piksel zostanie przypisany do danej klasy jest określone wzorami:

$$\omega_0 = Pr(K_0) = \sum_{i=1}^{t} p_i = \omega(t) \tag{2.7}$$

$$\omega_1 = Pr(K_1) = \sum_{i=t+1}^{L} p_i = 1 - \omega(t) \tag{2.8}$$

Średnia wartość piksela przyjmowana w klasie określona została w poniższych wzorach:

$$\mu_0 = \sum_{i=1}^{t} Pr(i|K_0) = \sum_{i=1}^{t} \frac{ip_i}{\omega_0} = \frac{\mu(t)}{\omega(t)} \tag{2.9}$$

$$\mu_1 = \sum_{i=t+1}^{L} iPr(i|K_1) = \sum_{i=t+1}^{L} \frac{ip_i}{\omega_1} = \frac{\mu_T - \mu(t)}{1 - \omega(t)} \tag{2.10}$$

Gdzie:

$$\omega(t) = \sum_{i=1}^{t} p_i \tag{2.11}$$

$$\mu(t) = \sum_{i=1}^{t} ip_i \tag{2.12}$$

Powyższe wartości są skumulowanymi momentami zerowego oraz pierwszego stopnia obrazu dla wartości progowej równej t. A średnia z wartości pikseli na całym obrazie może zostać zapisane wzorem:

$$\mu_T = \mu(L) = \sum_{i=1}^{L} ip_i \tag{2.13}$$

Niezależnie od wyboru wartości parametru t, prawdziwe są równania:

$$\omega_0\mu_0 + \omega_1\mu_1 = \mu_T, \qquad \omega_0 + \omega_1 = 1. \tag{2.14}$$

Wariancje obu klas można przedstawić następująco:

$$\sigma_0^2 = \sum_{i=1}^{t} (i-\mu_0)^2 Pr(i|K_0) = \sum_{i=1}^{t} (i-\mu_0)^2 \frac{p_i}{\omega_0} \tag{2.15}$$

$$\sigma_0^2 = \sum_{i=t+1}^{t} (i-\mu_1)^2 Pr(i|K_1) = \sum_{i=t+1}^{L} (i-\mu_1)^2 \frac{p_i}{\omega_1} \tag{2.16}$$

W kolejnym etapie należy sprawdzić czy wartość zmiennej k jest wartością optymalną względem funkcji kryterialnej. Funkcją kryterialną, jak wcześniej wspomniano jest:

- Wariancja wewnątrz klasowa, dąży do minimalizacji,

$$\lambda = \frac{\sigma_B^2}{\sigma_W^2}, \quad \sigma_W^2 = \omega_0\sigma_0^2 + \omega_1\sigma_1^2, \quad \sigma_B^2 = \omega_0\omega_1(\mu_1 - \mu_2)^2 \tag{2.17}$$

- Wariancja między klasowa, dąży do maksymalizacji.

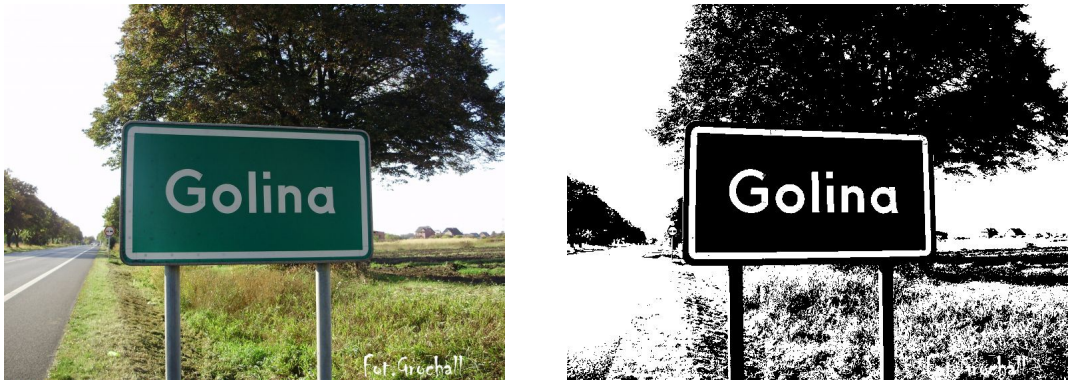
$$\kappa = \frac{\sigma_T^2}{\sigma_W^2}, \quad \sigma_T^2 = \sum_{i=1}^{L} (i-\mu_T)^2 p_i \tag{2.18}$$

Co intuicyjnie zgadza się z pożądanym efektem działania algorytmu. Chce się aby wartości pikseli w klasach były do siebie jak najbardziej zbliżone, a między klasami występowała jak największa różnica. Kryterium jakie stosuje się do optymalizacji wartości k jest łączna wariancja poziomów η. Parametr jest optymalny, jeżeli η przyjmuje wartość jak największą, tym samym wariancja wewnątrz klasowa. Wartość ta będzie zerowa jedynie w przypadku gdy wszystkie piksele obrazu zostaną zakwalifikowane tylko do jednej z klas, jednak jest to sprzeczne z celem działania algorytmu, więc w tym kontekście maksimum zawsze istnieje.

$$\eta = \frac{\sigma_B^2}{\sigma_T^2}, \tag{2.19}$$

Zaletą tej metody jest jej prostota – korzysta się jedynie z momentów zerowego i pierwszego rzędu, a także stabilność – optymalny próg jest wybrany automatycznie, nie bazuje na zróżnicowaniach (takich jak lokalne zbiory pikseli o niskich lub wysokich wartościach),

skupia się na globalnych właściwościach obrazu, histogramie. Ważną cechą jest też uniwersalność metody, dzięki, której można ją wykorzystywać w różnych dziedzinach analizy obrazu. Może służyć do binaryzacji, jak wykorzystano w projekcie, wykorzystywana jest też przy progowaniu[8]. Wadą tej metody jest możliwość błędu poprzez wybranie niewłaściwej wartości początkowej k.



(a) Obraz oryginalny. (b) Obraz po progowaniu.

Rysunek 2.3: Wynik progowania obrazu metodą Otsu.

2.3 Wykrywanie krawędzi

Przetwarzanie obrazów opiera się na wydobywaniu z obrazu takich cech, które są istotne przy późniejszej analizie i identyfikacji obiektów na nim zawartych. Jedną z takich cech jest krawędź, czyli znaczna lokalna zmiana w intensywności obrazu, związana z brakiem ciągłości w intensywności obrazu lub w pierwszej pochodnej intensywności obrazu. Występują dwa rodzaje takiej zmiany ciągłości:

1. Nieciągłość skokowa - intensywność obrazu zmienia się z jednej wartości w drugą i przez jakiś czas tą wartość utrzymuje, przypomina odpowiedź skokową, na obrazie może być to moment styku dwóch obiektów;

2. Nieciągłość liniowa - intensywność obrazu zmienia się z jednej wartości w drugą lecz w krótkim okresie czasu do niej powraca, przypomina odpowiedź impulsową, na obrazie może to być przerwanie w obiekcie;

W praktyce jednak tego typu krawędzie zdarzają się dość rzadko, krawędzie nie są aż tak wyraziste i skok przypomina bardziej rampę, a linia - dach[9].

Wykrywanie krawędzi to przede wszystkim rozpoznawanie znacznych zmian w obrazie, jest to mocno związane z wyznaczaniem maksimum pierwszej pochodnej. Gradient jest miarą zmiany, wektorem, który wskazuje kierunki i szybkość wzrostu wartości, to dwuwymiarowy odpowiednik pierwszej pochodnej. W obrazie zmianą jest różnica w intensywności koloru. To właśnie on jest bazą większości algorytmów wykrywających krawędzie. Gradient obrazu dany jest wzorem:

$$\nabla f = \begin{pmatrix} g_x \\ g_y \end{pmatrix} = \begin{pmatrix} \frac{\delta f}{\delta x} \\ \frac{\delta f}{\delta y} \end{pmatrix} \tag{2.20}$$

g_x- gradient w kierunku x,
g_y- gradient w kierunku y

Kierunek zmian można wyliczyć z poniższego wzoru:

$$\theta = \tan^{-1}\begin{pmatrix} g_x \\ g_y \end{pmatrix} \tag{2.21}$$

Etapy wykrywania krawędzi:

1. Filtracja
   Aby ulepszyć działanie detektora krawędzi należy pozbyć się szumów, które utrudniają wyliczanie gradientów. Należy jednak zachować umiar, zbyt duże wygładzanie obrazu może spowodować osłabienie krawędzi. Często w tym celu stosuje się rozmycie Gaussowskie, które zostało omówione szerzej w podrozdziale 2.3.1.

2. Wzmocnienie
   W polach obrazu gdzie umiejscowione są największe zmiany intensywności koloru podkreśla się te piksele, które mają na tą zmianę największy wpływ. Używa się w tym celu gradientu.

3. Wykrycie
   W związku z licznymi zanieczyszczeniami i szumami obecnymi na obrazie zdarza się, że piksele o niezerowym gradiencie nie należą do krawędzi. Aby pominąć piksele nieistotne, wybiera się próg wedle którego ocenia się czy piksel należy do krawędzi czy też nie.

4. Zlokalizowanie
   Etap ten składa się na większość algorytmów, polega na określeniu rozdzielczości, położenia pikseli, które składają się na krawędź. Jest to zdecydowanie przydatne w momencie potrzeby wyodrębnienia z obrazu konkretnego obiektu[10].

2.3.1 Detektor Canny

Detektor ten jest najczęściej używanym detektorem w świecie przetwarzania obrazów. Należy do rodziny detektorów Gaussowskich, specjalizujących się w wykrywaniu krawędzi skokowych. Jest to algorytm optymalny względem kilku kryteriów:

- Poprawna detekcja - dąży się do minimalizacji klasyfikacji nieistniejących krawędzi jako krawędzi istniejących. Są większe szanse na pominięcie krawędzi niż na wykrycie krawędzi błędnej.

- Poprawna lokalizacja - dąży się do tego, aby wykryte krawędzie znajdowały się jak najbliżej istniejących krawędzi.

- Jednoznaczna odpowiedź - dąży się do minimalizacji lokalnych maksimów wokół wykrytej krawędzi, idea jest taka, aby dla każdego istniejącego punktu krawędzi zwracany był tylko jeden piksel.

Sposób działania algorytmu można opisać w kilku krokach:

Krok 1 Obraz wejściowy należy poddać operacji rozmycia Gaussa, w celu pozbycia się szumów. Polega to na nałożeniu na obraz maski, która jest dyskretną aproksymacją funkcji Gaussa opisaną wzorem poniżej:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.22}$$

Matematycznie, nałożenie maski na obraz to wyliczenie splotu dyskretnych wartości funkcji Gaussa i pikseli obrazu[11].
Rozmycie Gaussa jest zależne od wartości gradientu.

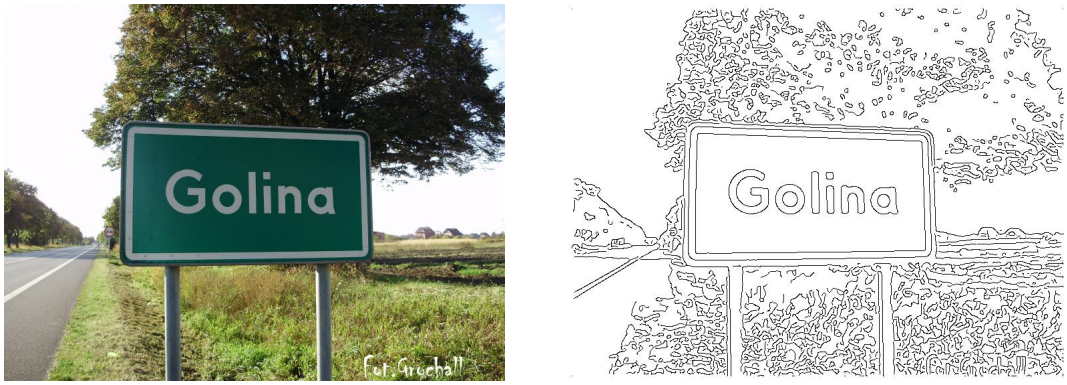
Krok 2 Następnie wyliczona zostaje wartość oraz kierunek gradientu.

$$gradient = \sqrt{G_x^2 + G_y^2} \tag{2.23}$$

$$\theta = \tan^{-1} \frac{G_y}{G_x} \tag{2.24}$$

Krok 3 Wykrywa się punkty należące do krawędzi i tłumi te wartości, które nie są maksimami lokalnymi, aby pozbyć się niewyraźnych krawędzi.

Krok 4 W ostatnim etapie punkty krawędzi zostają połączone i wybiera się histerezę - przedział od najniższej wartości progu do najwyżej wartości progu w jakim ma się znaleźć krawędź. Najwyższy próg k_{max} dotyczy mocnych krawędzi, natomiast najniższy próg k_{min} dotyczy krawędzi słabych.



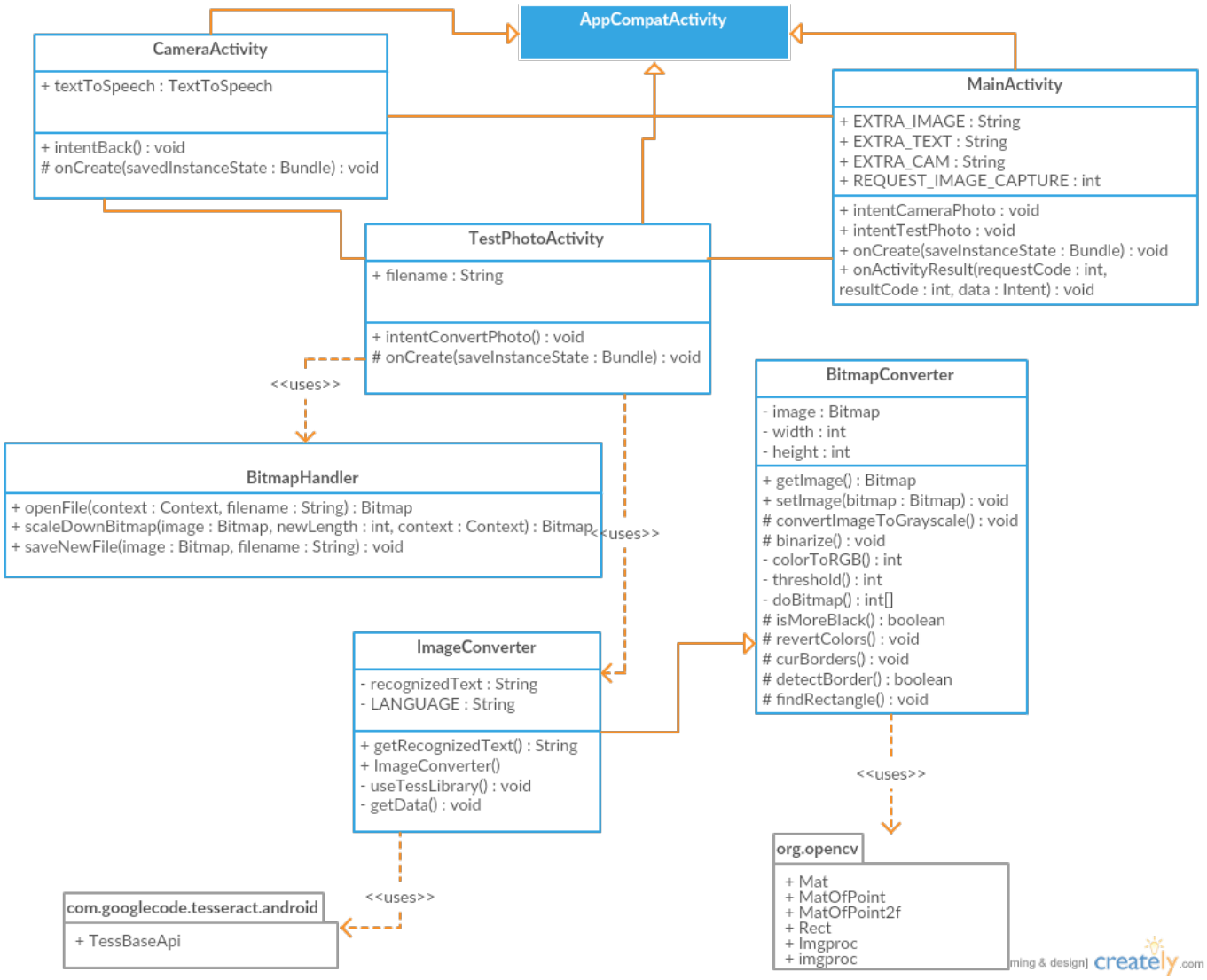
(a) Obraz oryginalny. (b) Obraz po wykryciu krawędzi.

Rysunek 2.4: Wynik wykrywania krawędzi detektorem Canny.

Rozdział 3

Architektura systemu

Program został napisany, korzystając z obiektowości języka Java. Wykorzystano klasy oraz aktywności, które wzajemnie ze sobą powiązano, w celu minimalizacji ilości kodu, a także elastyczności kodu na zmiany. Diagram klas został przedstawiony na Rysunku 3.1.



Rysunek 3.1: Diagram UML klas.

Komunikacja między aktywnościami została zaimplementowana, korzystając z intencji, pojęcia te zostały wyjaśnione w paragrafie 4.1.2. Poniżej zostanie zcharakteryzowana każda z klas i aktywności:

- **AppCompatActivity** to klasa bazowa, używana w przypadku aktywności, które korzystają z dodatkowych bibliotek. Wchodzi w skład podstawowych bibliotek systemu android[12].

- **MainActivity** to aktywność, która dziedziczy po **AppCompatActivity**. Obsługuje widok główny aplikacji, zaprezentowany na rysunku 5.1. Składa się z dwóch obiektów typu *Button*, które przekierowują do dwóch różnych widoków. Jeden jest obsługiwany przez kolejną aktywność - *TestPhotoActivity* i zostanie opisany później, a drugi - kamera urządzenia mobilnego, jest również obsługiwany przez **MainActivity**. W intencji zrobione zdjęcie przesyłane jest do klasy *onActivityResult*, gdzie zostaje odpowiednio przetworzone, tekst zostaje rozpoznany i w kolejnej intencji wyniki tych dwóch operacji zostają przesłane do **CameraActivity**.

- **TestPhotoActivity** to aktywność, która również dziedziczy po **AppCompatActivity**. Zajmuje się obsługą widoku testowego, wyświetla obiektu typu *ImageButton*, która po kliknięciu przesyłają intencję z odpowiednim obrazem, który ma zostać poddany przetwarzaniu. Widok ten jest przedstawiony na rysunku 5.2. Przetworzony obraz wraz z rozpoznanym tekstem zostają przesłane do **CameraActivity**.

- **CameraActivity** to aktywność, która także dziedziczy po **AppCompatActivity**. Zajmuje się obsługą widoku, który reprezentuje końcowy wynik działania aplikacji. Widok ten można znaleźć na rysunkach 5.3b, 5.4b oraz 5.5b. Sam widok zawiera obiekty: *ImageView*, służący do wyświetlenia przetworzonego obrazu, *TextView*, służący do wyświetlenia odczytanego tekstu, oraz dwa obiektu typu *Button*, gdzie jeden przekierowuje użytkownika do poprzedniego widoku, a drugi pozwala na powtórzenie komunikatu głosowego, który jest odczytywany z przekazanego do aktywności tekstu. W klasie odczytywana jest intencja przesłana z aktywności **MainActivity**, pobierany jest tekst, który rozpoznano na obrazie, typu String, następnie tekst ten zostaje odczytany, korzystając z klasy *TextToSpeech*, która została szerzej opisana w paragrafie 4.1.1.

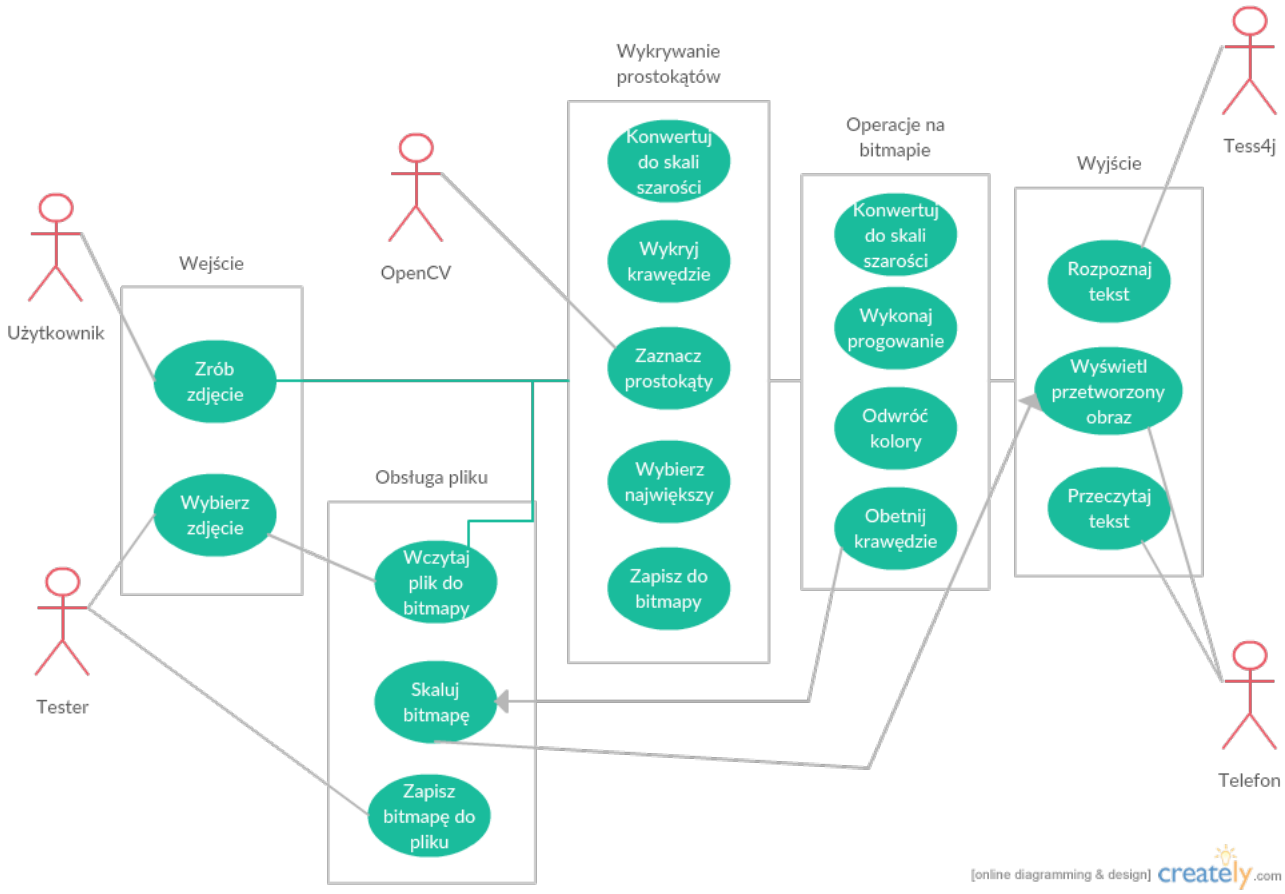
- **BitmapHandler** to klasa, która zajmuje się obsługą plików i bitmap. Konwertuje obraz w pliku do obiektu typu bitmap w funkcji *openFile*. Zapisuje bitmapę do pliku w funkcji *saveNewFile*. Daje też możliwość przeskalowania bitmapy do mniejszego rozmiaru w funkcji *scaleDownBitmap*, co jest przydatną funkcją, o tyle, o ile Android jest systemem, który jest w stanie sobie poradzić z obróbką większych obrazów, to już aby je wyświetlić, rozmiarowo należy dopasować je do ekranu urządzenia mobilnego na którym działa aplikacja.

- **BitmapConverter** to klasa, która zajmuje się zasadniczą trudnością niniejszego projektu - przetwarza bitmapę w taki sposób, aby biblioteka Tess4j poradziła sobie z przetworzeniem jej na tekst. Wykorzystuje w tym celu bibliotekę OpenCV, a także własne, zaimplementowane funkcje. Na początek wycina ze zdjęcia prostokąt, przyjmując, że jest on znakiem drogowym, korzystając z funkcji $findRectangle$, której działanie zostało opisane w paragrafie 4.3.1. Kolejno prostokąt zostaje poddany operacjom:

1. Przetworzenia do skali szarości w funkcji *convertImageToGrayscale*, korzystając z metody jasność, opisanej w paragrafie 2.1.
2. Progowania metodą Otsu w funkcji *binarizeImage*, metoda ta została opisana w paragrafie 2.2.1.
3. Sprawdzenia czy jest więcej koloru czarnego na obrazie w funkcji *isMoreBlack* i odwrócenia kolorów w funkcji *revertColors*, gdy czarny kolor występuje częściej, korzystając z założenia, że jeżeli chce się otrzymać czarny tekst na białym tle, to procentowo białe tło składa się z większej liczby pikseli.
4. Sprawdzenia czy obraz jest obramowany w funkcji *detectBorders*. I jeżeli jest to pozbycie się tych obramowań w funkcji *cutBorders*.

- **ImageConverter** to klasa, która dziedziczy po **BitmapConverter** i korzysta z biblioteki Tess4j. Konstruktor tej klasy wykonuje całe przetwarzanie obrazu i rozpoznanie tekstu. Przyjmuje zdjęcie w formacie Bitmap, wykonuje wszystkie operacje opisane we wcześniejszym punkcie, następnie przetwarza wynikową bitmapę na tekst i ten tekst jest wartością zwracaną konstruktora. Klasa ta powstała w celu rozdzielenia metod przetwarzających bitmapę i tych związanych z rozpoznawaniem tekstu. Zmniejszyła także liczbę komend wywoływanych w intencjach, przesyłanych do widoku z wynikiem działania aplikacji. W intencjach inicjalizuje się jedynie obiekt klasy, korzystając z konstruktora. Obiekt ten składa się z przetworzonego obrazu i rozpoznanego tekstu.

Działanie programu można podzielić na kilka etapów, jak na rysunku 3.2, przedstawiającym diagram przypadków użycia. Mamy: wejście, obsługę pliku, wykrywanie prostokątów, operacje na bitmapie oraz wyjście. Zaczynając od wejścia, pojawiają się dwie opcje w zależności od typu aktora, użytkownik będzie chciał zrobić zdjęcie obiektowi znajdującym się przed nim, więc wybierze opcję "Zrób zdjęcie", natomiast tester będzie chciał przetestować działanie aplikacji na znanych mu danych testowych, dlatego wybierze opcję "Wybierz zdjęcie". Kolejno wybrane zdjęcie będzie musiało zostać pobrane z pliku, następnie program zajmie się wykrywaniem prostokątów, zdjęcie z kamery od razu trafia do tego bloczka. Tutaj uczestniczy aktor, którym jest biblioteka OpenCV. Następnie prostokąty są odpowiednio przetwarzane wcześniej opisanymi operacjami na bitmapie. Dalej rozpoznaje się tekst, w czym uczestniczy aktor: Tess4j, i na urządzeniu mobilnym wyświetla się przetworzony obraz, a tekst zostaje przeczytany.



Rysunek 3.2: Diagram przypadków użycia.

Rozdział 4

Implementacja

Projekt został napisany pod system operacyjny Android, a stworzony na platformie Android Studio, w związku z czym językiem implementacji była Java.W niniejszej pracy wykorzystano dwie opensourcowe biblioteki dla platformy Java: Tess4j oraz OpenCV. Tess4j - bibliteka OCR, przetwarzała obraz na mowę, a z biblioteki OpenCV, która oferuje bardzo dużo funkcji do przetwarzania obrazów wykorzystano funkcję wykrywającą na obrazie prostokąty.

4.1 System operacyjny Android

Aplikacja została napisana pod system operacyjny Android, jako, że ma on największy procentowy skład na rynku telefonów - 85[20]. Natywnym jeżykiem w jakim pisze się aplikacje na system operacyjny Android jest Java. Jednak istnieją platformy, które pozwalają pisać również w innych językach np. C#, korzystając z Xamarin. Xamarin był wcześniej platformą niezależną, ale po wykupieniu przez firmę Microsoft został dodany jako nakładka do Visual Studio 2015 w Aktualizacji 2. Języki C# i Java są do siebie bardzo podobne, działają nawet na ekwiwalentnych wirtualnych maszynach - Java na JVM(Java Virtual Machine), a C# na .NET. Jeszcze kilka lat temu największym atutem Javy był fakt, że jest rozwiązaniem międzysystemowym, działającym zarówno na systemie Windows jak i Linux[21], jednak aktualne aktualizację .NETowe sprawiły, że różnica ta się rozmyła. Jednakże w tej pracy zdecydowano się na język natywny aplikacji androidowych, czyli Javę, z uwagi na o wiele bogatszą dokumentację, zarówno w języku polskim i angielskim, a także znaczną liczbę kursów i artykułów pomocnych przy pisaniu kodu. Java jest językiem, który radzi sobie z tymi problemami, z którymi nie radzą sobie języki skryptowe i jest jednym z najpopularniejszych takich języków. Radzi sobie z takimi problemami jak dostęp do baz danych, przetwarzanie rozproszone, programowanie wielowątkowe, a także sieciowe[22]. Jako IDE wykorzystano oficjalne dla platformy Android Studio. Pierwsze implementacje były pisane na platformie Intellij IDEA, aby ułatwić debuggowanie oraz testowanie kodu, które trwało tutaj zdecydowanie krócej niż na podłączonym urządzeniu tudzież wirtualnej maszynie, a Android Studio jest produktem stworzonym przez tą samą firmę, w związku z czym migracja między tymi dwiem platformami była najrozsądniejszym rozwiązaniem.

4.1.1 TextToSpeech - zamiana tekstu na mowę

Do przeczytania tekstu - konwersji obiektu typu String na mowę - użyto klasy TextToSpeech, znajdującej się w bibliotece platformy Android. Instancja tej klasy, po odpowiedniej

inicjalizacji, od razu odtwarza dźwięk, który czyta podany tekst. Odpowiednia inicjalizacja polega na zaimplementowaniu TextToSpeech.OnInitListener[25]. Inicjalizacja instancji następuje wtedy kiedy zostaje ona wywołana. Ustawia się tam język w jakim użytkownik chce aby tekst został przeczytany. Wartość języka przechowuje się w zmiennej locale. W przypadku ustawienia wartości domyślnej, tekst powininen zostać przeczytany w języku jaki ustawiony jest na urządzeniu mobilnym na którym działa aplikacja. Do zmiennej toSpeak zostaje przesłana wartość z intencji - wynik działania Tess4j.

Kod 4.1: Kod programu, który odpowiada za przeczytanie tekstu z wykorzystaniem klasy TextToSpeech

```
TextToSpeech textToSpeech=new TextToSpeech(
   getApplicationContext(),
new TextToSpeech.OnInitListener() {
    @Override
    public void onInit(int status) {
        if(status != TextToSpeech.ERROR) {
            Locale locale = new Locale("pl","PL");
            textToSpeech.setLanguage(locale);
        }
     }
});;

final String toSpeak = getIntent().getExtras().getString(
   MainActivity.EXTRA_TEXT);
textToSpeech.speak(toSpeak, TextToSpeech.QUEUE_FLUSH, null);
```

4.1.2 Intencje i aktywności - komponenty platformy Android

Aktywność(ang. *Activity*) to pojedyncza, skondensowana funkcjonalność, jaką może wykonać użytkownik. Skoro praktycznie każda aktywność jest związana z interakcją z użytkownikiem, więc klasa ta zajmuje się tworzeniem **widoków**(ang. *View*) z interfejsem użytkownika, które ustawiane są za pomocą funkcji *setContenView*(*View*), gdzie zmienna *View* to widok, który programista chce połączyć z daną aktywnością[13]. Widok to klasa reprezentująca GUI, czyli graficzny interfejs użytkownika. Korzysta z interaktywnych komponentów jak *Button*, czyli obiekt, reagujący w momencie kiedy się go naciśnie, wykonując akcję opisaną w aktywności połączonej z tym widokiem[14]. Widok może być określony w pliku XML lub zaprogramowany metodą *draganddrop*(z ang. *weiupu*) - pojedyncze komponenty są pobierane z biblioteki oferowanej przez środowisko Android Studio, i umiejscawiane w odpowiednich regionach na prostokątnym polu, imitującym ekran telefonu. Cechy owych komponentów, takie jak nazwa, położenie, zależności między pozostałymi komponentami i inne są również ustawiane, poprzez wybieranie wartości cech z list lub wpisywanie ich do odpowiednich pól.
Intencja(ang. *Intent*) to kolejny komponent androidowej aplikacji, abstrakcyjny opis operacji, która ma zostać wykonana[15]. Umożliwia komunikację między aplikacjami, a także wewnątrz pojedynczej aplikacji - pomiędzy Aktywnościami. Intencje, przez pryzmat ich drugiej własności - komunikacji między Aktywnościami, wykorzystanej w tej pracy, można podzielić na dwa rodzaje[**?**]:

- **Jawne** - w konstruktorze intencji, obiekt, który ma wykonać zadanie jest ściśle określony - nazwą jego klasy.

- **Niejawne** - zadeklarowana jest konkretna akcja, ale nie jest powiedziane jaki komponent ma ją wywołać[17].

Poniżej zaprezentowano działanie intencji, wykorzystane w niniejszym projekcie. W Kodzie 4.2 nastąpiło jawne zadeklarowanie intencji w Aktywności - **MainActivity** i będzie ona wykonana w Aktywności **CameraActivity**. W Kodzie 4.3 do intencji zostają dodane obiekty różnego typu - przetworzony obraz oraz rozpoznany tekst. W Kodzie 4.4 intencja zostaje wywołana. Następnie w Kodzie 4.4 pokazano jak obsłużono intencje w klasie **CameraActivity**. Funkcją *getIntent* pobrano intencję, a następnie wszystkie dodatkowe obiekty, które zostały do niej dołączone, odwołując się do klasy z której pochodzą zmienną typu String.

Kod 4.2: Zadeklarowanie intencji.

```
Intent intent = new Intent(this, CameraActivity.class);
```

Kod 4.3: Przekazanie obiektów różnego typu w intencji.

```
extra.putParcelable(MainActivity.EXTRA_IMAGE, imageConverter.
   getImage());
extra.putString(MainActivity.EXTRA_TEXT, imageConverter.
   getRecognizedText());
intent.putExtras(extra);
```

Kod 4.4: Wywołanie intencji.

```
startActivity(intent);
```

Kod 4.5: Obsługa intencji w klasie w którym ma zostać wykonana.

```
imageView.setImageBitmap((Bitmap) getIntent().getExtras().
   getParcelable(MainActivity.EXTRA_IMAGE));
final String toSpeak = getIntent().getExtras().getString(
   MainActivity.EXTRA_TEXT);
```

4.2 OCR - optyczne rozpoznawanie znaków

OCR (ang. Optical Character Recognition) to system przetwarzania dokumentów w formie papierowej. na tekst, w taki sposób, aby otrzymanym wynikiem były cyfrowe informacje gotowe do dalszego przetwarzania. Ten zbiór technik jest często wykorzystywany w biznesie, przechowywanie wszystkich dokumentów zajmuje wiele miejsca, bo zamianie ich na postać cyfrową - miejsce jest zaoszczędzane, a dane gotowe do ewentualnej dalszej obróbki[18]. OCR wykorzystywany jest także przy odczytywaniu tablic rejestracyjnych samochodów, w technikach stosowanych przez policję. [19] W niniejszej pracy techniki OCR zostały zastosowane do odczytania z odpowiednio przetworzonego obrazu - tekstu, i zapisanie go w obiekcie typu String.

4.2.1 Biblioteka Tess4j

Tess4j to opensource'owa biblioteka. Jest to nakładka na Tesseract OCR API dla platformy Java. Język w jakim napisana jest biblioteka Tesseract to C++. Biblioteka ta umożliwia optyczne rozpoznawanie znaków (OCR) umieszczonych w plikach w formatach

- TIFF, JPEG, GIR, PNG oraz BMP, wielostronicowe obrazy TIFF, a także dokumenty w formacie PDF [23]. Poniżej została przedstawiona część kodu programu. Zostaje stworzony obiekt klasy TessBaseAPI, określa się miejsce w urządzeniu mobilnym, gdzie znajduje się folder tessdata - z danymi uczącymi w języku polskim, jeżeli na urządzeniu ów folder się nie znajduje, zostaje on ściągnięty z repozytorium umieszczonym w sieci na platformie Github. Następnie rozpoczyna się działanie aplikacji funkcją init, gdzie podaje się ścieżkę do pliku z danymi uczącymi oraz język w jakim mają być rozpoznawane słowa. Kolejno do funkcji setImage podaje się obraz w jednym ze wcześniej przytoczonych formatów. Obraz ten musi być odpowiednio przetworzono, gdyż biblioteka nie wykrywa miejsca położenia tekstu na zdjęciu:

- Tekst nie może być pisany ręcznie.
- Tekst musi być w kolorze czarnym.
- Tekst musi być umiejscowiony na białym tle.
- Na obrazie może znajdować się tylko tekst.

Kod 4.6: Kod programu, odpowiadający za odwołanie do zewnętrznego API Tess4j

```
TessBaseAPI baseApi = new TessBaseAPI();
baseApi.setDebug(true);
File tessdataFolder = new File(
Environment.getExternalStorageDirectory().
   getAbsoluteFile()
+ "/tessdata");
if (!tessdataFolder.exists()) {
    getData();
}
String path = String.valueOf(
Environment.getExternalStorageDirectory()) + "/";
baseApi.init(path, LANGUAGE);
baseApi.setImage(bitmap);
String recognizedText = baseApi.getUTF8Text();
baseApi.end();
```

4.3 Biblioteka OpenCV

Biblioteka OpenCV to zbióR funkcji wykorzystywanych przy przetwarzaniu obrazów, jest opensource'owa oraz można z niej korzystać na różnych systemach, jak Mac OS X, Windows i Linux. Biblioteka ta została napisana w jeżyku C, ale korzystając z odpowiednich nakładek można ją stosować również w języku takim jak Java. Obraz w bibliotece OpenCV jest traktowany jako macierz, przecięcia kolumn i wierszy opisują wartość piksela, albo w postaci jednej wartości gdy obraz jest w skali szarości, albo w postaci trzech liczb, gdy obraz jest kolorowy. Wystpuje kilka typów zapisu danych Arr, Scalar i Mat. Ten ostatni został wykorzystany w niniejszej pracy[24].

4.3.1 Wykrywanie prostokątów

W celu użycia metody boundingRect, która wykrywa prostokąty na obrazie należało odpowiednio przygotować obraz wejściowy. Należało przetworzyć go do odcieni szarości, a następnie wyrysować jego krawędzie, metodą Canny. W poniższym kodzie tak przetworzony obraz znajduje się w zmiennej typu Bitmap o nazwie *converted*. Ze wszystkich wykrytych krawędzi, które tworzą obiekty przeanalizowano wszystkie te, które mają kształt prostokąta. Za każdym razem liczono pole wykrytej figury i sprawdzano czy jest większa od poprzedniego prostokąta. Korzystano z założenia, że znaki drogowe tekstowe mają kształt prostokątów i na zdjęciu zcentrowanym na taki znak, założono, że będzie on największym prostokątem. Następnie tworzy się nowy obiekt typu Bitmap o wymiarach znalezionego prostokąta i w wyniku działania poniższej funkcji otrzymuje się obiekt zawierający jedynie prostokąt, czyli w domniemaniu znak drogowy, gotowy do dalszej obróbki. Prostokąt ten jest wycięty z oryginalnego zdjęcia, z uwagi na to, że skorzystanie z detektora Canny o dużej rozpiętości progów zarysowało najważniejsze krawędzie, ale mogło zignorować napisy.

Kod 4.7: Kod programu, odpowiadający za wykrycie prostokątów przy użyciu biblioteki Open CV

```
List<MatOfPoint> contours = new ArrayList<MatOfPoint>();
Imgproc.findContours( converted, contours, new Mat(), Imgproc.
   RETR_LIST,Imgproc.CHAIN_APPROX_SIMPLE);
MatOfPoint2f approxCurve = new MatOfPoint2f();
Bitmap bmp = null;
Mat copy = new Mat();
   for (int i=0; i<contours.size(); i++) {
       MatOfPoint2f contour2f = new MatOfPoint2f(
                           contours.get(i).toArray() );
       double approxDistance = Imgproc.arcLength(contour2f,
          true)*0.02;
       Imgproc.approxPolyDP(contour2f, approxCurve,
          approxDistance, true);
       MatOfPoint points = new MatOfPoint( approxCurve.toArray
          () );
       Rect rect = Imgproc.boundingRect(points);

                if (rect.area() > max) {
                        Rect roi = new Rect(rect.x, rect.y,
                        rect.width, rect.height);
                bmp = Bitmap.createBitmap(rect.width, rect.
                   height,
                Bitmap.Config.ARGB_8888);
                copy = new Mat(mat, roi);
                        max = rect.area();
        }
    }
Utils.matToBitmap(copy, bmp);
image = bmp;
}
```

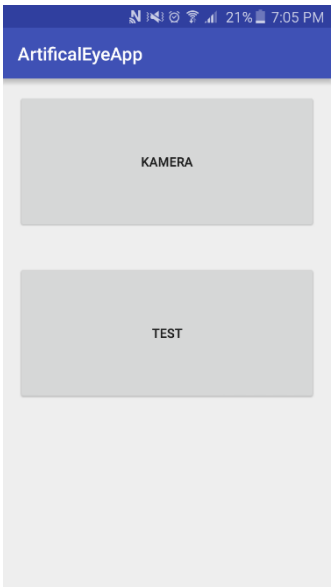
Rozdział 5

Działanie aplikacji i testy

5.1 Wymagania wstępne

Aplikacja działa na smartfonach z oprogramowaniem Android o wersji SDK nie starszej niż 17. Wersją najbardziej oczekiwaną jest wersja 23. Smartfon musi posiadać aparat oraz ponad 13Mb wolnej pamięci – zostanie na niej zapisany folder – tessdata, z plikiem zawierającym dane potrzebne bibliotece Tess4j do przetwarzania obrazu w języku polskim. Język telefonu powinien być ustawiony na ten w jakim oczekiwane jest czytanie komunikatów przez aplikację.

5.2 Widok główny



Rysunek 5.1: Główny widok aplikacji.

Jest to intuicyjny widok z dwoma dużymi przyciskami. Przy uruchomieniu aplikacji, informuje ona użytkownika komunikatem głosowym o położeniu przycisków i ich przeznaczeniu: „Kliknij na górze ekranu aby przejść do kamery – Kliknij na dole ekranu aby przejść do testów". Aplikacja, poza widokiem głównym, posiada dwa widoki dodatkowe: użytkownika oraz testowy. Można się do nich dostać po wciśnięciu w odpowiedni przycisk.

Przycisk „Kamera” przekierowuje użytkownika do widoku z kamery smartfona, natomiast przycisk „Test” przekierowuje do widoku testowego.

5.3 Widok docelowy

Widok docelowy przenosi użytkownika do kamery zainstalowanej na urządzeniu na którym działa aplikacja. W tym momencie użytkownik ma możliwość kliknięcia w ekran i zrobienia fotografii. W zależności od rodzaju telefonu, kamera może poprosić o potwierdzenie przesłania zdjęcia do dalszej przeróbki, o czym aplikacja poinformuje komunikatem głosowym, z dokładnym umiejscowieniem odpowiedniego przycisku na ekranie, zdjęcie też może zostać od razu przesłane.

5.4 Widok testowy

Rysunek 5.2: Testowy widok aplikacji.

Osoba testująca, klikając na wybrane zdjęcie sprawia, że zostaje ono poddane przetwarzaniu. W efekcie końcowym otrzymuje się rozpoznany znak, wycięty ze zdjęcia, jako obraz binarny – zaprezentowany przez dwa odcienie – czarny oraz biały, kolory odwrócone, odpowiednio tak aby tekst był czarny, a tło białe, czyli obraz jaki na wejście otrzymuje algorytm rozpoznający tekst. Pod obrazem zostaje wyświetlony tekst, którzy otrzymano w wyniku działania biblioteki Tess4j. Zostaje on przeczytany przez aplikację, użytkownik może odtwarzać go dowolną liczbę razy, klikając na przycisk „Powiedz”. Poniżej, na zdjęciach numer 5.3, 5.4 oraz 5.5, zostały przedstawione wyniki działania aplikacji w trybie testowym.

(a) Obraz przetwarzany.

(b) Obraz przetworzony.

Rysunek 5.3: Wynik działania aplikacji.

(a) Obraz przetwarzany.

(b) Obraz przetworzony.

Rysunek 5.4: Wynik działania aplikacji.

(a) Obraz przetwarzany.

(b) Obraz przetworzony.

Rysunek 5.5: Wynik działania aplikacji.

Rozdział 6

Zakończenie

6.1 Podsumowanie

Stworzona aplikacja i jej zaimplementowane funkcjonalności pokryły się z zamierzonym celem niniejszej pracy inżynierskiej. Środowisko Android Studio, w którym została napisana umożliwiło spełnienie wszystkich założeń projektowych, a także przeprowadzanie testów nie tylko na maszynie wirtualnej, ale również na urządzeniu mobilnym, czyli docelowym urządzeniu, na którym aplikacja ma pracować. Powstała prosta w obsłudze aplikacja, z interfejsem adekwatnym dla obranego użytkownika, jakim jest osoba niewidoma lub niedowidząca. Aplikacja ta korzysta z popularnych, prostych, ale skutecznych algorytmów przetwarzania obrazów, które zostały w powyższej pracy skrupulatnie opisane i przedstawione na tle innych możliwych rozwiązań, poza własną implementacją algorytmów, skorzystano także z dostępnych w sieci otwartych bibliotek - OpenCV oraz Tess4j. One również zostały omówione, przedstawiono ich zastosowania oraz przykładowe użycie.
Aplikacja jest napisana obiektowo, z możliwością rozszerzenia o kolejne funkcjonalności. Po prostu dodając kolejne Aktywności i Widoki. Napisano ją w języku Java, korzystając również z bibliotek natywnie napisanych w języku C++. Przy minimalnych zmianach, z uwagi na to, że biblioteki używane w Androidzie a te typowo związane z platformą Java, przy przetwarzaniu obrazów nieznacznie się różnią, aplikacja ta może pełnić funkcję aplikacji desktopowe do przetwarzania wczytanych obrazów na informację tekstową. Migracja programu na inne platformy, takie jak WindowsPhone, nie powinna być problematyczna ze względu na niebotyczne podobieństwo języków C# oraz Java.

6.2 Możliwe rozszerzenie

Praca, jak wspomniano we wstępie, w małym stopniu opierała się na projekcie zespołowym, w celu rozszerzenia działania aplikacji o inne urządzenia, można by oprzeć ją o "Sztuczne oko" w większym stopniu, np. dodając przetwarzanie znaków nie tylko na tekst, a później mowę, ale także na elektrowibrację, które mogłyby zwracać uwagę użytkownika na znaki tekstowe drogowe, które jednoznacznie informują o możliwości zdarzenia niebezpiecznego.

Bibliografia

[1] R. Tadeusiewicz, P. Korohoda: *Komputerowa analiza i przetwarzanie obrazów*, wyd. Fundacji Postępu Telekomunikacji, ('3)

[2] jw. ('5)

[3] J.D. Cook: *Converting color to grayscale* (dostęp: 3.12.2016)
URL: http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/

[4] Rafajłowicz E., Rafajłowicz W.: *Wstęp do przetwarzania obrazów przemysłowych*, Wrocław : Oficyna Wydawnicza Politechniki Wrocławskiej, [2010]. ('82)

[5] jw. ('83-'84)

[6] jw. ('85)

[7] jw. ('85)

[8] *IEEE Transactions on systems, man, and cybernetics*, vol. smc-9, no. 1, 1979 (dostęp: 3.12.2016)
URL: http://web-ext.u-aizu.ac.jp/course/bmclass/documents/otsu1979.pdf

[9] R. Jain, R. Kasturi, B. G. Schunck, *Machine Vision* McGraw-Hill, Inc., 1995 ('140)

[10] jw. ('145-146)

[11] *Gaussian Smoothing* (dostęp: 3.12.2016)
URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

[12] Android Developers: *AppCompatActivity* (dostęp: 3.12.2016)
URL: https://developer.android.com/reference/android/support/v7/app/AppCompatActivity.htm

[13] Android Developers: *Activity* (dostęp: 3.12.2016)
URL: https://developer.android.com/reference/android/app/Activity.html

[14] Android Developers: *View* (dostęp: 3.12.2016)
URL: https://developer.android.com/reference/android/view/View.html

[15] Android Developers: *Intent* (dostęp: 3.12.2016)
URL: https://developer.android.com/reference/android/content/Intent.html

[16] android4devs: *Wstęp do Intencji (Intents)* (dostęp: 3.12.2016)
URL: http://www.android4devs.pl/2011/07/wstep-do-intencji-intents/

[17] Damian Chodorek: *Kurs Android (5)* (dostęp: 3.12.2016)
URL: http://damianchodorek.com/2015/02/12/kurs-android-intent-intencje-komunikacja-aktywnosci-5/

[18] *The role of OCR in invoice processing* (dostęp: 3.12.2016)
URL: http://smart-soft.net/support/articles/automated-invoice-processing/the-role-of-ocr-in-invoice-processing.htm

[19] Wikipedia: *Automatic number plate tecognition* (dostęp: 3.12.2016)
URL: http://en.wikipedia.org/wiki/Automatic_number_plate_recognition

[20] *Porównanie mobilnych systemów Android, iOS i Windows Phone 8* (dostęp: 3.12.2016) URL: http://www.benchmark.pl/testy_i_recenzje/android-ios-windows.html

[21] B. Eckelo: *Thinking in Java edycja polska*, Gliwice: Helion, wydanie 4, [2006] . ('63-'64)

[22] jw. ('62)

[23] *Tess4J - JNA wrapper for Tesseract* (dostęp: 3.12.2016)
URL: http://tess4j.sourceforge.net/

[24] E. Rafajłowicz, W. Rafajłowicz, A. Rusiecki: *Algorytmy przetwarzania obrazów i wstęp do prazy z biblioteką OpenCV*, Wrocław: Oficyna Wydawnicza Politechniki Wrocławskiej, [2010], ('23-'27)

[25] Android Developers: *TextToSpeech* (dostęp: 3.12.2016)
URL: https://developer.android.com/reference/android/speech/tts/TextToSpeech.html