

Imię i nazwisko studenta: Konrad Ossowski

Nr albumu: 155157

Studia pierwszego stopnia

Forma studiów: stacjonarne

Kierunek studiów: Informatyka

Profil: Systemy geoinformatyczne

Imię i nazwisko studenta: Maciej Plewka

Nr albumu: 155170

Studia pierwszego stopnia

Forma studiów: stacjonarne

Kierunek studiów: Informatyka

Profil: Architektura systemów komputerowych

PROJEKT DYPLOMOWY INŻYNIERSKI

Tytuł projektu w języku polskim: Sztuczka karciana z wykorzystaniem NFC i algorytmów rozpoznawania obrazu

Tytuł projektu w języku angielskim: Card trick using NFC and Computer Vision algorithms

Potwierdzenie przyjęcia projektu	
Opiekun projektu	Kierownik Katedry/Zakładu (pozostawić właściwe)
<i>podpis</i>	<i>podpis</i>
dr inż. Grzegorz Fotypa	

Data oddania projektu do dziekanatu:

STRESZCZENIE

Niniejsza praca przedstawia projekt oraz sposób realizacji aplikacji mobilnej. Wykonany program ma być wykorzystywany do wykonania sztuczki karcianej. Aplikacja jest przeznaczona na urządzenia z systemem Android. W pracy przedstawiono podstawy związane z działaniem oraz wykonaniem aplikacji na tej platformie. W celu realizacji projektu konieczne było wykorzystanie wbudowanych w telefon komponentów. Wykorzystanymi podzespołami były, aparat w celu wykonania zdjęcia, moduł NFC by odczytywać i zapisywać informacje w znacznikach umieszczonych na kartach do gry oraz wibrator potrafiący wywibrować binarną reprezentację karty. Dodatkowo należało zaimplementować możliwość detekcji obiektów na wykonanym zdjęciu. Do zrealizowania detekcji wykorzystano kaskadowe klasyfikatory cech Haar'a. Działanie tego rozwiązania zostało opisane w pracy, tak jak proces trenowania własnej kaskady. W następnych rozdziałach opisano dokładniej najważniejsze elementy wchodzące w skład aplikacji, a także sposób ich implementacji. Wyjaśnione zostały także pojęcia potrzebne do zrozumienia działania poszczególnych procesów wykorzystanych w celu wykonania projektu.

ABSTRACT

The following Engineering Thesis presents the project and the method of implementing a mobile application. The prepared program is meant to be used in order to perform card tricks. The application is dedicated for systems running the Android operating system. The paper presents the basics of operating and developing an application on such platform. In order to implement the project, it was necessary to use the phone's built-in components. The components used were: the camera for the purpose of taking a picture, the NFC module in order to read and write information in tags placed on the playing cards, and a vibrator capable of vibrating the binary representation of the card. In addition, it was necessary to implement the ability to detect objects in the taken picture. Haar-like cascade classifiers were used to implement the detection. The operation of this solution has been described in the paper, as well as the process of training one's own cascade. The following chapters describe more specifically the key elements that make up the application, as well as the method of their implementation. The concepts needed to understand the operation of individual processes used to implement the project are also explained.

SPIS TREŚCI

1.	Wprowadzenie (Konrad Ossowski)	7
1.1.	Istota magii	7
1.2.	Idea triku	7
2.	Platforma Android (Konrad Ossowski, Maciej Plewka)	11
2.1.	Programowanie obiektowe (Konrad Ossowski)	11
2.2.	Programowanie na Androida (Konrad Ossowski)	13
2.3.	Intencja (Konrad Ossowski)	14
2.4.	Manifest (Konrad Ossowski)	15
2.5.	Zasoby (Konrad Ossowski)	15
2.6.	Aktywność (Konrad Ossowski)	15
2.7.	Warstwa ograniczeń (Konrad Ossowski)	19
2.7.1.	Elementu graficzne interfejsu (Konrad Ossowski)	20
2.8.	Singleton (Konrad Ossowski)	20
2.9.	System plików (Konrad Ossowski)	21
2.10.	Wibrator (Konrad Ossowski)	21
2.11.	Aparat (Maciej Plewka)	21
3.	Kaskadowe klasyfikatory cech Haar'a (Maciej Plewka)	23
3.1.	Cechy Haar'a	23
3.2.	Obliczanie wartości sumy pikseli	24
3.3.	Proces uczenia mocnych klasyfikatorów	25
3.4.	Kaskada klasyfikatorów mocnych	27
4.	Wykorzystanie biblioteki OpenCV (Maciej Plewka)	29
4.1.	OpenCV w procesie trenowania kaskad	29
4.1.1.	Generowanie zbioru zdjęć pozytywnych	29
4.1.2.	Trening kaskady	31
4.2.	OpenCV na platformie Android	32
4.2.1.	Detekcja obiektów	32
5.	Trening własnej kaskady (Maciej Plewka)	34
5.1.	Zbiór danych uczących	34
5.2.	Proces treningu	36
5.2.1.	Trening detekcji rewersu karty	36
5.2.2.	Trening detekcji karty ze znakiem zapytania	37
6.	Użycie Kaskad w celu realizacji Tricku (Maciej Plewka)	38
6.1.	Detekcja Czoła	38
6.2.	Detekcja Karty	38
6.3.	Dobór parametrów detekcji	39
6.4.	Zmniejszanie zdjęcia	41
6.5.	Wklejanie obrazu karty w miejsce obiektu	42

6.6. Implementacja w aplikacji	42
7. Technologia NFC (Konrad Ossowski).....	45
7.1. Historia NFC	45
7.2. NFC na platformie Android	45
7.2.1. NfcAdapter	46
7.2.2. Znacznik	47
7.2.3. Zapis i odczyt danych z użyciem tagu NFC	47
8. Podsumowanie (Maciej Plewka)	51
Wykaz literatury	52
Wykaz rysunków.....	53
Wykaz tabel.....	54

1. WPROWADZENIE (KONRAD OSSOWSKI)

1.1. Istota magii

Wiele aspektów ludzkiego życia można interpretować jako rodzaj magii. Odprawianie rytuałów, okultyzm, religię, wypowiadanie zaklęć, to można określić negatywną interpretacją magii. Bardziej pozytywnym znaczeniem magii jest określanie jej mianem emocji towarzyszącym nam w różnych sytuacjach. "Magia świąt" gdy spędzamy czas z rodziną lub gdy uczucie zakochania wpływa "magicznie" na nasz nastrój. Nauka nie idzie w parze z magią, mają zupełnie inne fundamenty. Podstawą nauki są badania i dowody, natomiast magia opiera się na wierze i emocjach.

Na szczęście można połączyć naukę z magią, uzyskamy wtedy coś co nazywamy iluzją. Jest to wywołanie u widza wrażenia, iż robi się rzeczy pozornie sprzeczne z prawami fizyki poprzez zastosowanie odpowiednich trików. Każdy trik składa się z dwóch elementów. Elementu naukowego, czyli sekretu oraz elementu magicznego, czyli prezentacji.

Sekret jest bardzo ważnym elementem triku, to on pozwala na wykonanie pozornie niemożliwej czynności. Jednakże prezentacja jest ważniejsza. To ona ukrywa nasz sekret przed publicznością, jednocześnie wprawiając ją w osłupienie.

Prezentację można podzielić na trzy części, chociaż w znacznym stopniu się one zazębiają. Przede wszystkim iluzja ma być rozrywką, ale nie tylko dla publiczności. Magik powinien bawić się tym co robi razem z publicznością, być częścią pokazu, a nie wszystkowiedzącym guru. Kolejnym istotnym elementem prezentacji jest historyjka. Niech ma ona jakieś nawiązanie do kontekstu całej iluzji. Jeśli będzie spójna i logiczna nada większej wiarygodności całemu trikowi. Ostatnią kwestią jest umiejętność dezorientowania publiczności. Są trzy główne sposoby dezorientacji: czasowa, słowna i fizyczna. Czasowa ściśle wiąże się z historyjką. Wykorzystujemy opowiadanie, do odroczenia następnej części triku w czasie, aby publiczność zapomniała o pewnych aspektach, które się już wydarzyły. Dezorientacja słowna polega na kłamaniu. Niekiedy wtmawiamy widzowi, że zrobił coś co nie miało miejsca. Popieramy to sztucznyimi rezultatami jego rzekomych działań, przez co musi nam uwierzyć. Dezorientacja fizyczna jest najprostsza z podanych. Zwracamy się do widza z prośba o wykonanie jakieś nieistotnego działania np. popukanie karty, pstryknięcie palcami. Takie czynności skupią jego uwagę, dzięki czemu sekret pozostaje bezpieczny.

1.2. Idea triku

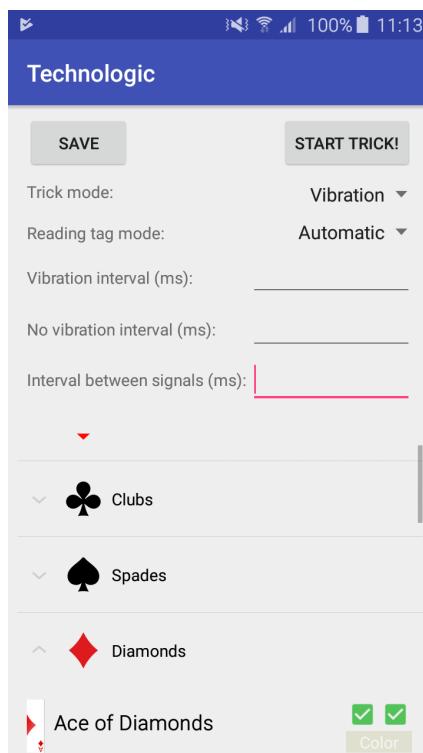
Szybki postęp cywilizacyjny coraz szybciej dostarcza nam coraz lepszych urządzeń. To co kiedyś było na filmach sciencie-fiction, dzisiaj jest rzeczywistością. Ludzie nie obcujący na co dzień z takimi technologiami postrzegają je jako "magiczne pudełka", które "magicznie działają". Łatwo ich zadziwić możliwościami jakie posiada dzisiejszy smartfon. Jednakże dla osób korzystających z możliwości jakie przynoszą obecne czasy potrzeba czegoś więcej. Można skorzystać z technologii, której możliwości nie znajdują. Wpadliśmy na pomysł wykorzystania jednej z dostępnych funkcji systemu operacyjnego Android, NFC. Do triku należy przygotować:

- Pytajnik - kustomowa karta ze wzorem białego znaku zapytania na czerwonym tle . Wzór

pytajnika może zostać nadrukowany na dowolnym materiale. Nam zależało na długiej żywotności, więc zleciliśmy nadrukowanie wzoru na plastikowej karcie.

- Talię - talia kart z wklejonymi naklejkami NFC. Talię można zlecić do wykonania drukarni lub wykonać własnoręcznie naklejając tagi na rewers kart. Nam udało się osiągnąć bardzo dobry efekt, ze względu na dopasowanie fragmentu rewersu do kształtu znacznika.
- Aplikację - poprawnie skonfigurowana aplikacja Technologic na smartfonie z Androidem w wersji conajmniej Lollipop. Wartości zapisane do tagów na konkretnych kartach muszą się zgadzać z tymi w aplikacji. Ponadto należy wybrać, który wariant triku wybieramy (Fotografia/Wibracje).
- Woreczek - ciemny, nieprzeźroczysty woreczek wielkości karty do gry.
- (opcjonalnie) Pokrowiec - pokrowiec na telefon, który nie przekazuje gestów z zewnątrz.
- (opcjonalnie) Schemat wibracji - dla każdej karty ustalić sześciobitowy ciąg. Zaleca się, aby pierwsze dwa bity określały kolor, a pozostałe cztery wartość, jednakże nie jest to obowiązkowe. Jedynka jest sygnalizowana jedną krótką wibracją, natomiast zero dwoma krótkimi wibracjami.

Konfiguracja



Rys. 1.1. Panel konfiguracyjny aplikacji

Przycisk **START TRICK!** uruchamia Aktywność określona w opcji *Trick mode*. Dostępne są trzy możliwości: tryb zapisywania wartości do znaczników oraz dwa rodzaje odczytywania (Fotografia lub Wibracje). Opcja *Reading tag mode* ma dwie możliwości do wyboru, automatyczny lub

manualny. Odpowiada to za operacje związane z NFC, mogą się one wykonywać automatycznie po wykryciu tagu lub na żądanie poprzez wcisnięcie przycisku. *Vibration interval* odpowiada za długość (w milisekundach) trwania pojedynczej wibracji. *No vibration interval* to długość (w milisekundach) odstępu między dwoma wibracjami, gdy sygnalizujemy wartość 0. *Interval between signals* to długość (w milisekundach) trwania przerwy między sygnalizowanymi bitami. Każda z grup koloru kart ma elementy, które zawierają informacje na temat nazwy oraz schematu wibracji kolejnych kart. Nazwę karty podajemy w polu tekstowym znajdującym się po prawej stronie od ikony reprezentującą tę kartę, natomiast schemat wibracji ustalamy za pomocą pól wyboru. Dla ułatwienia mają one podpis, które z nich odpowiadają za kolor, a które za wartość karty. Przycisk **SAVE** służy do zapisania ustawień do pliku, dzięki czemu będą one się automatycznie ładować przy starcie aplikacji.

Po przygotowaniu wszelkich niezbędnych elementów, możemy przystąpić do wykonywania triku. Smartfona z aplikacją chowamy do kieszeni. Aplikacja musi być na pierwszym planie i urządzenie nie może być zablokowane. Zaleca się skorzystanie z pokrowca, aby aplikacja przez przypadek nie przeniosła się na inny plan. Wybieramy widza z publiczności, podajemu mu talię i prosimy go o przetasowanie kart. Następnie prosimy go o wybranie dowolnej karty z talii, zapamiętaniu jej i umieszczeniu w woreczku. Odbieramy od niego resztę talii, a woreczek wkładamy dyskretnie do kieszeni (w tym momencie aplikacja odczytuje zawartość ze znacznika). Dalszy przebieg występu zależy od wybranego wcześniej wariantu. Kluczowym działaniem dla obu wariantów jest dezorientacja czasowa w postaci historyjki.

Fotografia

Możemy odłożyć karty, nie będą potrzebne. Ten wariant można zakończyć na dwa sposoby. W zależności od wybranego zakończenia będzie zastosowana inna historyjka.

Pierwszą opcją jest zrobienie zdjęcia portretowego widzowi za pomocą aplikacji. Na fotografii zostanie umieszczony obrazek ze wzorem karty widza na wysokości jego czoła. Historyjką dla tego zakończenia mogłoby być: "W dzisiejszych czasach technologia coraz bardziej ingeruje w ludzki umysł. Nie orientujemy się kiedy urządzenia czytają nam w myślach upraszczając nam codzienne życie. O ile rzeczach nie musimy pamiętać, ponieważ coś robi to za nas. Mój smartfon również posiada taką możliwość. On umie czytać w myślach! Nie wierzycie? Pozwólcie, że zaprezentuję (...)"

Drugą opcją jest podanie widzowi pytajnika. Następnie robimy dowolne zdjęcie widzowi, w taki sposób, aby pytajnik był widoczny. Na fotografii, obraz pytajnika zostanie zastąpiony obrazem ze wzorem karty widza. W tym przypadku historyjką mogłoby być: "Teraz drodzy Państwo chciałbym Wam przedstawić magiczną kartę. Jest bardzo skryta, a swoją wartość trzyma wewnątrz siebie i nieczęsto ukazuje ją światu. Jednakże jest bardzo podatna na sławę, wtedy ujawnia swoją tajemnicę. Spróbujmy nagrodzić ją brawami, a Pan/Pani niech jej powie, że jest jej największym fanem i zapozuje z nią do zdjęcia. Będę paparazzi! (...)"

W obu sytuacjach efektem triku jest fotografia, na której widnieje karta widza. Można ją wysłać widzowi na pamiątkę, gdyż znajduje się ona w pamięci smartfonu w folderze Technologic.

Wibracje

Ten wariant jest trudniejszy i wymaga pewnych umiejętności od magika. Talię można zostawić przy sobie, gdyż może się przydać do odgadywania karty. Po umieszczeniu woreczka z kartą w kieszeni, smartfon zacznie wibrować przekazując konkretny schemat vibracji ustalony wcześniej w aplikacji. Dzięki znajomości schematu, jesteśmy w stanie wywnioskować jaką kartę wybrał widz. Historyjką do tego zakończenia mogłoby być: "Wielu magików odnajduje wybrane karty w różnych miejscach, które wydają się niemożliwe. Ja natomiast odnajdę kartę w miejscu gdzie jej nie ma! Będę wykładać odkryte karty na stół i na ich podstawię powiem jakiej karty brakuje w talii!". Podczas wykładania kart, nie należy grupować kart w jeden stosik. Niech z kart ułoży się wzór odpowiadający wartości karty znajdującej się w kieszeni. To sprawi wrażenie, że od momentu wykładania kart na stół wiedziałeś jakiej karty brakuje.

Po zakończeniu triku w pierwszym wariantie nie należy chować telefonu do kieszeni z woreczkiem. Najlepiej odłożyć go gdzieś lub ewentualnie schować do innej kieszeni. W drugim wariantie natomiast nie należy wyciągać telefonu z kieszeni, aby uniknąć podejrzeń.

2. PLATFORMA ANDROID (KONRAD OSSOWSKI, MACIEJ PLEWKA)

Android to system operacyjny przeznaczony dla urządzeń mobilnych z jądrem Linuxa. Jego kolejne wersje zaczynają się na kolejne litery alfabetu i nawiązują do jakieś słodkiej przekąski lub deseru. Trzema najpopularniejszymi wersjami na rynku są Lollipop, Marshmallow i Nougat. Ostatnią wypuszczoną wersją jest Oreo (sytuacja z grudnia 2017). Każda z wersji udostępnia własny interfejs programowania aplikacji (ang. application programming interface) zwany **API**. Im nowsza wersja, tym wyższy jest poziom API, który daje nam szersze możliwości programistyczne.

2.1. Programowanie obiektowe (Konrad Ossowski)

Podejście do sposobu programowania zmieniało się w czasie. Tradycyjnym podejściem było programowanie proceduralne. Polegało ono na rozdzieleniu danych od operujących na nich funkcji. Niestety takie podejście odizolowania danych od kodu może spowodować przypadkową ich modyfikację przez procedury, które nie były logicznie z nimi związane. Do takiego programu ciężko również było wprowadzić wszelkie modyfikacje.

Na początku lat 60. XX w. pojawiła się pierwotna koncepcja programowania obiektowego. Miała ona być bardziej naturalna dla ludzkiego mózgu, a samo programowanie stać się bardziej zgodne z rzeczywistością. Popularność zdobyła dopiero w okolicach lat 80. XX w., gdy pojawił się język C++ rozszerzający język C. Wyniknęło to ze wzrostu popularności graficznych interfejsów użytkownika, do których koncepcja programowania obiektowego sprawdza się wzorowo.

Tworzenie programu zgodnego z koncepcją programowania obiektowego sprawdza się do rozdzielenia logiki na osobne klasy (typy danych), istniejące w tych klasach procedury oraz trzymaniu się konkretnych zasad. Te zasady nazywamy **paradygmatami programowania obiektowego**.

Abstrakcja (ang. abstraction)

Sprowadzenie danego problemu do prostszej postaci. Polega na wyznaczeniu dla obiektów cech, niezbędnych dla algorytmów, ale jednocześnie niezależnych od implementacji.

Enkapsulacja (ang. encapsulation)

Ukrycie cech obiektu. Zewnętrzne otoczenie nie może zmienić danych znajdujących się wewnątrz obiektu w nieoczekiwany sposób. Każda klasa udostępnia abstrakcyjną reprezentację siebie zwaną interfejsem. Za jego pomocą zewnętrzne otoczenie może się komunikować z danym obiektem.

Polimorfizm (ang. polymorphism)

Używanie wartości zmiennych i podprogramów na kilka różnych sposobów. Wykazywanie przez metodę różnych form działania w zależności od tego jaki typ obiektu jest wskazywany przez wskaźnik lub referencję.

Dziedziczenie (ang. inheritance)

Porządkuje polimorfizm i enkapsulację przez definiowanie i tworzenie specjalizowanych obiektów na podstawie bardziej ogólnych.

Obiekt a klasa (Konrad Ossowski)

Podstawową cegiełką każdego programu zgodnego z koncepcją programowania obiektowego jest obiekt. Obiekt to nie to samo co klasa. Klasa opisuje pola (zmienne) i metody (procedury) dla wszystkich reprezentantów tej klasy, czyli obiektów.

Zarówno pola jak i metody muszą mieć określony poziom dostępu. Wyróżniamy trzy poziomy dostępu: publiczny, chroniony oraz prywatny. Dostęp publiczny oznacza, że dany atrybut może zostać użyty zarówno w obiekcie tej klasy jak i poza nim. Dostęp chroniony pozwala na używanie atrybutów tylko w obiekcie tej klasy, ale również w obiektach dziedziczących z tej klasy. Dostęp prywatny nie pozwala na używanie atrybutów klasy żadnym obiektem nie będącym reprezentacją tej klasy. Jeśli nie podamy jawnie poziomu dostępu do danego atrybutu, zostanie on potraktowany jako prywatny, jest to domyślny poziom dostępu.

Wszystkie pola klasy powinny mieć dostęp prywatny, a modyfikację tych pól powinny być ewentualnie możliwe przez publiczne metody wykonujące operacje na tych polach. Metodę publiczną pobierającą wartość danego pola nazywamy getter'em, natomiast metodę ustawiającą wartość danemu polu setter'em.

Każdy z atrybutów może być dodatkowo statyczne dla danej klasy. Oznacza to, że możemy odwołać się do tego atrybutu bez tworzenia reprezentanta tej klasy. Najczęściej takie statyczne atrybuty mają dostęp publiczny. Jeśli metoda nie jest zadeklarowana jako statyczna, może zostać zadeklarowana jako wirtualna. Oznacza to, że nie można wywołać tej metody dopóki nie zostanie ona nadpisana w klasie dziedziczącej.

Klasa implementuje tylko abstrakcyjny typ danych więc pamięć przydzielana jest dopiero do obiektów, a nie do samej klasy. Natomiast obiekt jest już strukturą zawierającą dane oraz funkcje pracujące na tych danych. Każdy obiekt będący reprezentantem danej klasy nazywamy instancją tej klasy. Każda instancja ma taki sam zestaw pól oraz metod, jednakże mogą się różnić wartościami pól wewnętrz. Tworzenie instancji następuje przez wywołanie konstruktora. Konstruktor to specjalna metoda w klasie, musi mieć ona identyczną nazwę co klasa. Po wywołaniu konstruktora następuje alokacja pamięci dla obiektu, połączenie tego obszaru z metadanymi z klasą, wykonaniu kodu klasy bazowej oraz konstruktora. Dopiero wtedy nasz obiekt zaczyna fizycznie istnieć. Jednakże zmienna obiektowa nie przechowuje wewnątrz siebie danych obiektu, ona przechowuje referencję do tego obiektu. Referencja to wartość, która informuje o lokalizacji danych obiektu w pamięci.

Istnieją klasy, których instancji nie można utworzyć. Nazywamy je klasami abstrakcyjnymi. Klasa abstrakcyjna musi posiadać conajmniej jedną metodę wirtualną w sobie. Jeśli posiada wyłącznie metody wirtualne, jest wówczas nazywana interfejsem klasy.

2.2. Programowanie na Androida (Konrad Ossowski)

Android nie ma ścisłe określonej technologii, za pomocą której można tworzyć dla niego aplikacje. Wybór zależy od efektów jakie chcemy osiągnąć. Technologii za pomocą których chcielibyśmy tworzyć aplikację jest bardzo dużo, jednakże można je podzielić na dwa rodzaje. Programowanie natywne (np. Android oraz JNI) zwiększy wydajność aplikacji. Programowanie multiplatformowe (np. Apache Cordova oraz Xamarin) zwiększy przenośność.

Android-Java

Jest najpopularniejszą technologią do tworzenia aplikacji na Androida. Jak nazwa wskazuje, bazuje ona na języku Java, chociaż często nazywa się tę technologię po prostu Android. Programista ma dostęp do obszernej dokumentacji całego języka.

Java Native Interface (JNI)

Java Native Interface jest to framework, który umożliwia aplikacji Java uruchomionej na wirtualnej maszynie Javy (ang. Java Virtual Machine) wołać funkcje natywne. To znaczy takie, które wykonują się bezpośrednio na komponentach komputera, napisane w innych językach programowania takich jak C czy C++. Możliwe jest także wołanie metod kodu Java przez natywne aplikacje. Możliwe jest użycie tego interfejsu na platformie Android. Kod natyny wykonuje się znacznie szybciej w porównaniu do Javy. Kolejną powodem do stosowania JNI jest fakt, że istnieje wiele bibliotek i narzędzi w językach C i C++, które nie mają swoich odpowiedników dla języka Java. Wywołanie metody natywnej z poziomu kodu Javy jest kosztowne. Pojedyncze wywołanie takiej metody trwa znacznie dłużej niż zwołanie takiej metody w obrebie jednej platformy. Dodatkowo tablice często są kopiowane dla funkcji natywnej, a po jej wykonaniu kopiowane są z powrotem dla kodu Java. Przykładowy kod z wykorzystaniem JNI w środowisku Android, zwracający łańcuch znaków:

```
1 #include <jni.h>
2 #include <string>
3
4 extern "C"
5 JNIEXPORT jstring JNICALL
6 Java_com_example_maciej_myapplication_MainActivity_stringFromJNI(
7     JNIEnv *env,
8     jobject /* this */) {
9     std::string hello = "Hello from C++";
10    return env->NewStringUTF(hello.c_str());
11 }
```

Dodatkowo w kodzie Java potrzebna jest deklaracja funkcji

```
1 public native String stringFromJNI();
```

Apache Cordova

Cordova to framework, który opakowuje aplikację HTML/JavaScript do macierzystego kontenera, który ma dostęp do funkcji urządzenia na kilku platformach. Funkcje te są udostępniane za pośrednictwem zunifikowanego interfejsu JavaScript API, dzięki czemu można łatwo napisać jeden zestaw kodu.

Xamarin

Xamarin jest platformą, na której w język C# jesteśmy w stanie stworzyć aplikacje na Androida, WindowsPhona i iOS jednocześnie. Posiada specjalne zestawy narzędzi dla programistów (ang. software development kit) zwane SDK: Xamarin.iOS i Xamarin.Android. Tworzenie na WindowsPhona nie wymaga dodatkowego SDK.

2.3. *Intencja (Konrad Ossowski)*

Intencja (ang. Intent) to abstrakcyjny opis akcji jaką chcemy wykonać. Możliwości wykonania akcji przy pomocy Intencji jest bardzo dużo. Do podstawowych zalicza się przejście do kolejnego widoku w aplikacji czy wykonanie zdjęcia. Pełni ona rolę spoiny między kodem różnych aplikacji. Opisuje ona czynność do wykonania, dzięki czemu osobna aplikacja realizuje swoją pracę i odsyła ewentualne wyniki. Pierwszorzędnymi atrybutami struktury Intencji jest opis akcji do wykonania oraz ewentualne dane niezbędne do wykonania tej akcji. Do drugorzędnych atrybutów należą:

- Kategoria (ang. category) - zapewnia dodatkową informację na temat akcji do wykonania.
- Typ (ang. type) - precyzuje typ danych jaki jest przekazywany.
- Komponent (ang. component) - precyzuje nazwę komponentu (klasy), która będzie używała Intencji. Zazwyczaj jest to określone z pozostałych atrybutów i na podstawie tego dopasowywany jest komponent, który obsłuży daną akcję. Ustalenie tego atrybutu powoduje, że pozostałe są opcjonalne.
- Dodatki (ang. extras) - przechowuje wszelkie dodatkowe informacje jakie chcielibyśmy przekazać do komponentu.

Ponadto każdej Intencji możemy ustawić flagi. Lista dostępnych flag jest opisana w dokumentacji Androida. Zapewniają one dodatkowe informacje konfiguracyjne np. ustawienie flagi *FLAG_FROM_BACKGROUND* informuje, że dana Intencja pochodzi od procesu w tle, a nie od interakcji użytkownika. Istnieją dwie podstawowe formy Intencji:

- Jawne (ang. explicit) Intencje mają określony atrybut komponent. Można tego dokonać poprzez metodę *setComponent()* lub *setClass()*. Dzięki temu, jednoznacznie jest ustalona klasa, która ma zająć się wykonaniem akcji określonej w Intencji.
- Niejawne (ang. implicit) Intencje nie określają atrybutu komponent. Muszą natomiast, poprzez pozostałe atrybuty, zawierać wystarczająco dużo informacji, aby system określił, który z dostępnych komponentów będzie najlepszy do przeprowadzenia danej akcji.

Rozstrzygnięcie niejawnej Intencji następuje na podstawie trzech atrybutów: akcji, typu i kategorii. Korzystając z tych informacji wykonywane jest zapytanie w menadżerze pakietów dla komponentu, który może obsłużyć Intencję. Decyzję o tym czy dany komponent zostanie użyty określa się w następujący sposób. Jeśli został ustalony atrybut akcja i/lub typ w Intencji, to musi on być również wymieniony przez komponent jako obsługiwany. Natomiast określenie kategorii odnosi się do Aktywności, musi ona obsługiwać daną kategorię, aby zrealizować Intencję.[7]

2.4. Manifest (Konrad Ossowski)

Każda aplikacja w technlogii Android musi posiadać plik AndroidManifest.xml w swoim głównym folderze. Zapewnia nazwę pakietu Java dla aplikacji, która pełni rolę unikalnego identyfikatora. Manifest dostarcza podstawowych informacji o aplikacji (etykieta, ikona) do systemu Android, które musi posiadać, zanim będzie mógł uruchomić kod aplikacji. Zawiera listę bibliotek, z którymi aplikacja musi być połączona. Opisuje komponenty aplikacji oraz podaje nazwy klas ich implementujących. Deklaruje minimalny poziom interfejsu API Androida oraz uprawnienia, które muszą być spełnione do prawidłowego funkcjonowania. Zgoda użytkownika co do uprawnień jest niezbędna, aby aplikacja mogła korzystać z chronionych części API. Niezbędnym elementem do ustawienia jest ustalenie widoku, który się wyświetli przy pierwszym uruchomieniu. Dokonuje się tego poprzez Filtr Intencji (ang. Intent Filter). Wystarczy w elemencie opisującym Aktywność wstawić Filtr Intencji z odpowiednimi wartościami atrybutów akcji oraz kategorii (odpowiednio *android.intent.action.MAIN* oraz *android.intent.category.LAUNCHER*).[2]

2.5. Zasoby (Konrad Ossowski)

W dzisiejszych czasach często wprowadza się zmiany w istniejących aplikacjach. Może to być spowodowane zmianą właściciela aplikacji, decyzją ulepszenia aplikacji lub po prostu ludzkim kaprysem. Dlatego wysoki nacisk kładzie się na to, aby aplikacja była łatwo modyfikowalna. Android zapewnia mechanizm określonym jako **Zasoby (ang. Resources)**.

Wszelkie Zasoby powinny zostać podzielone i umieszczone w specjalnych podfolderach katalogu z zasobami o nazwie */res*. Mamy kilka takich specjalnych podfolderów do wyboru, między innymi folder */drawable* służy do przechowywania w nim obrazków wyświetlanych w aplikacji. Folder */color* zawiera zdefiniowane kolory używane w aplikacji. Folder */layout* zawiera pliki xml opisujące widoki używane w komponentach graficznych aplikacji. Folder */values* przechowuje kilka plików xml zawierających proste wartości takie jak łańcuchy znaków (*/strings.xml*) lub wartości liczbowe np. */dimens.xml*.[4]

Kiedy aplikacja zostaje skompilowana, zostaje wygenerowana klasa R. Zawiera ona identyfikatory wszystkich zasobów umieszczonych w podfolderach */res*. Dla każdego podfolderu katalogu */res* istnieje podklasa w klasie R. Przykładowo dla obrazka *hello.png* znajdującego się w podfolderze */drawable* mamy podkласę w postaci R.drawable, w której znajduje się statyczne pole hello, możemy się do niego odwołać poprzez R.drawable.hello. Dzięki temu uzyskujemy identyfikator danego zasobu, który jest liczbą całkowitą. Wiele obiektów w Androidzie ma metody, które przyjmują jako parametr identyfikator zasobu.[3]

2.6. Aktywność (Konrad Ossowski)

Aktywność (ang. Activity) służy głównie do interakcji z użytkownikiem. Z tego względu klasa ta odpowiedzialna jest za utworzenia okna gdzie umieszczane są komponenty interfejsu użytkownika. Najczęściej jej zawartość przedstawia się na pełnym ekranie. Jednakże może również zostać użyta jako ruchome okno lub jako wbudowany element innej Aktywności tworząc wię-

sy komponent. Nowa klasa Aktywności musi dziedziczyć po klasie bazowej AppCompatActivity oraz w projekcie musi się również znajdować przygotowany dla niej zasób zawierający jej widok. Możemy wówczas nadpisać każdą z dostępnych metod z klasy bazowej, dodawać własne pola oraz pisać własne metody. Najważniejszą metodą do nadpisania jest onCreate(Bundle). Należy w niej wywołać analogiczną metodę rodzica oraz ustawić dla tej aktywności odpowiedni widok za pomocą metody setContentView(int). Metoda setContentView(int) przyjmuje jako parametr identyfikator zasobu zawierającego informacje o widoku danej aktywności.[5] Przykład:

```
1 public class MainActivity extends AppCompatActivity {
2     @Override
3     protected void onCreate(Bundle savedInstanceState) {
4         super.onCreate(savedInstanceState);
5         setContentView(R.layout.activity_main);
6     }
7 }
```

W tym przykładzie nasza aktywność jest absolutnie pusta. Użytkownik widzi jedynie biały ekran, z którym nie ma żadnej interakcji. Bez dodatkowych komponentów graficznych niewiele uda się tu osiągnąć. Jednakże są pewne elementy, które mogą zapewnić interakcję z użytkownikiem poprzez samą pustą Aktywność. Jednym z nich jest obiekt **Toast (ang. Toast)**.

Jest to widok, który zawiera krótką wiadomość dla użytkownika. Jest ona wyświetlana na ekranie przez krótką chwilę, po czym automatycznie znika. Pozycję można dowolnie ustalić, domyślnie wyświetla się u dołu ekranu. Ten element jest nieresponsywny, aby uniknąć wprowadzenia użytkownika w błąd. Ideą tego mechanizmu jest, aby powiadomić dyskretnie użytkownika o jakieś sytuacji np. jego ustawienia zostały zapisane. Najprościej wykorzystać tę klasę poprzez wywołanie jednej ze statycznych metod, która ustawi wszystko czego potrzebujemy i wróci nam obiekt klasy Toast. Jedną z tych metod jest *makeText(Context context, CharSequence text, int duration)*. Parametr *context* to kontekst aplikacji. Jest globalną informacją o stanie środowiska. Innymi słowy, gdy tworzymy nowy obiekt nierzadko musi on wiedzieć co się w danym momencie dzieje w aplikacji, aby funkcjonował prawidłowo. Najczęściej podajemy tam po prostu kontekst Aktywności, w której aktualnie się znajdujemy. Parametr *text* jest niczym innym jak łańcuchem znaków, który zostanie wyświetlony w Toaście. Alternatywnie możemy podać w to miejsce zasób, który zawiera łańcuch znaków. Ostatni parametr *duration* jest określeniem jak długo ma się wyświetlać ta informacja. Należy tutaj podać jedną z dwóch stałych znajdujących się w klasie Toast, LENGTH_SHORT lub LENGTH_LONG. Po wywołaniu tej metody zostanie nam zwrócony obiekt typu Toast, aby wyświetlić jego zawartość na ekranie należy wywołać na końcu metodę *show()*.[19]

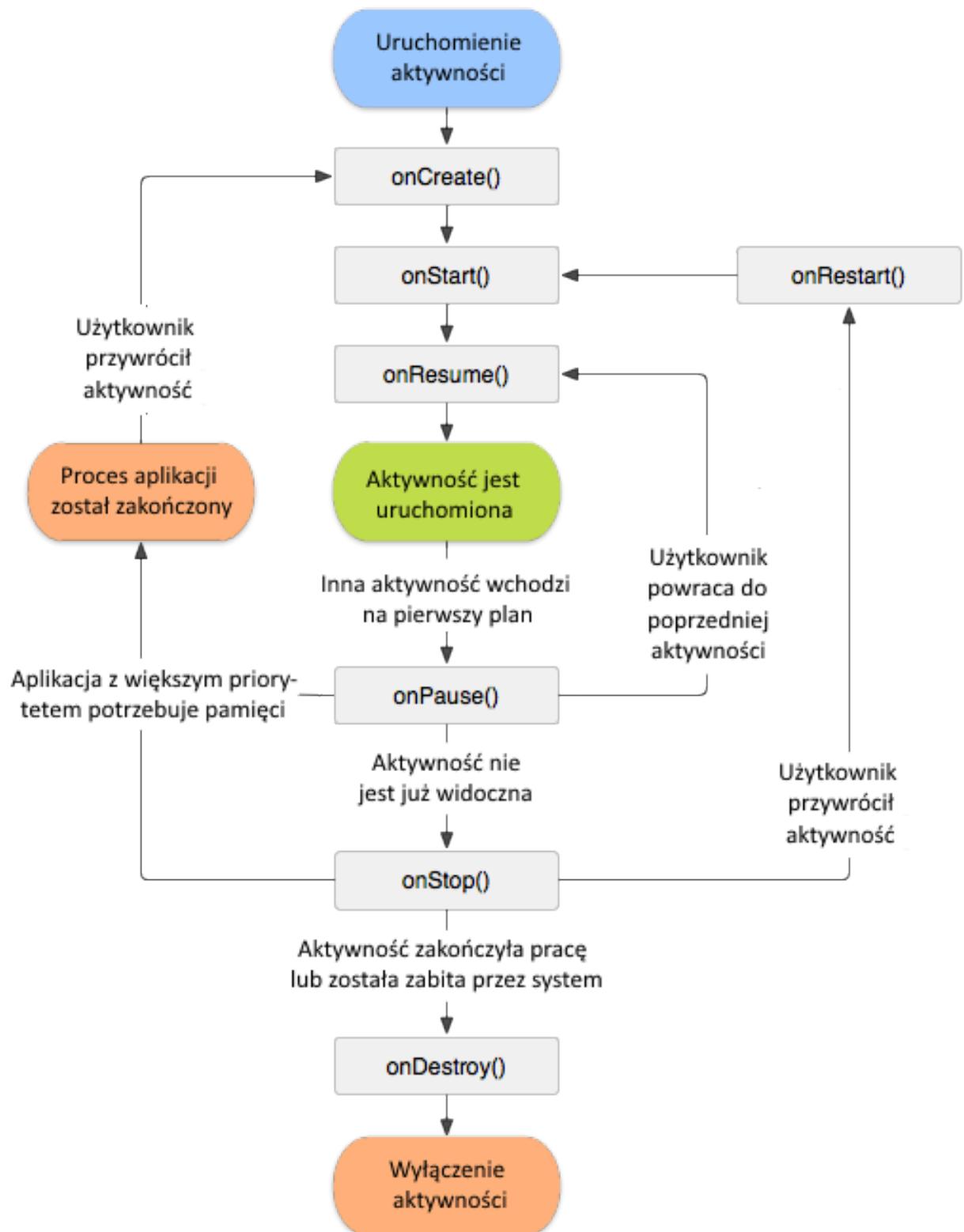
Przykład:

```
1 Toast.makeText(this, "Hello World!", Toast.LENGTH_SHORT).show();
```

Odwoływanie się do poszczególnych elementów z pliku zawierającego widok Aktywności następuje poprzez metodę *findViewById(int id)*. Parametr *id* to identyfikator elementu interfejsu, który znajduje się na widoku. Nie musimy pamiętać lub zapisywać tych identyfikatorów. Mamy do nich dostęp poprzez klasę *R.id*, która zawiera w sobie wszystkie identyfikatory. Obiekt zwracany przez metodę *findViewById(int id)* należy zrzutować na odpowiedni typ.

```
1 private void initializeSpinners(){  
2     Spinner trickModeSpinner = (Spinner) findViewById(R.id.trickModeSpinner);  
3 }
```

Podczas pracy na Androidzie możemy minimalizować aplikację, wyłączać ją, blokować ekran, przechodzić między kolejnymi widokami. Każda z tych sytuacji wpływa na Aktywności, które były uruchomione. Poniższy schemat przedstawia cykl życia Aktywności, przy niektórych działańach ze strony użytkownika lub systemu. Na końcu każdej strzałki znajduje się nazwa metody jaka jest wywoływana w aktywności na rzecz, której wydarzyła się dana akcja.



Rys. 2.1. Cykl życia aktywności[5]

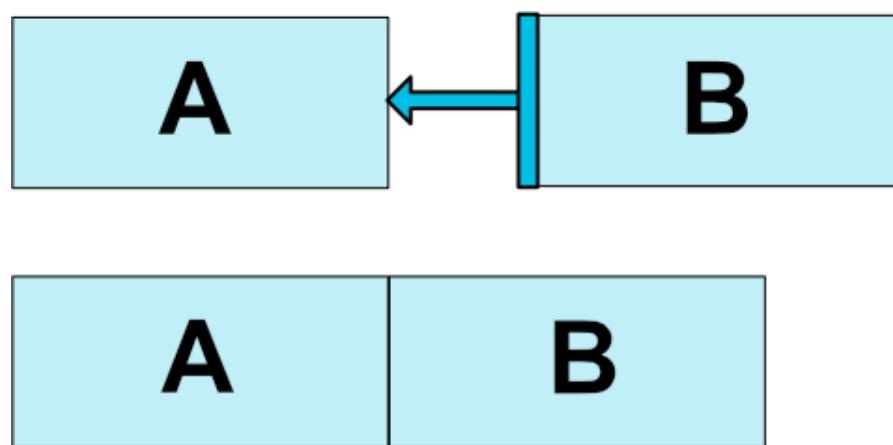
2.7. Warstwa ograniczeń (Konrad Ossowski)

O Aktywności można myśleć jak o sztaludze. Jest to dopiero podstawa dla dalszej pracy. Aby namalować obraz, będzie nam potrzebne płótno. Analogicznie, aby wyświetlić coś na ekranie, będzie nam do tego potrzebna warstwa. Jest to podstawowy element interfejsu użytkownika, na którym możemy budować kolejne komponenty lub bardziej skomplikowane struktury.

Warstwa ograniczeń (ang. constraint layout) to specjalna warstwa, która pozwala na pozycjonowanie komponentów oraz określenie ich wymiarów w bardzo elastyczny sposób. Dokonuje się tego za pomocą **Ograniczeń (ang. constraints)**.

Są one relacją między komponentami. Informują o tym gdzie i w jakiej odległości leżą od siebie elementy widoku. Dzięki nim można budować interfejs na kilka sposobów. Rozmiar możemy ustawić poprzez rozciąganie lub zwężanie w dowolnych kierunkach komponentu. Wówczas mówi się, że ten element ma ustalony (ang. fixed) rozmiar. Alternatywą dla określenia wymiarów komponentu jest ustawienie, aby ograniczył się on do rozmiaru jaki zajmuje jego zawartość (ang. wrap-content). Jeśli chcemy ustalić szerokość lub wysokość, można tego dokonać za pomocą jednej z opcji: dopasuj do rodzica (ang. match-parent) lub dopasuj do ograniczeń (ang. match-constraint). Dopasowanie do rodzica spowoduje wpasowanie komponentu wertykalnie lub horyzontalnie do zewnętrznych krawędzi rodzica. Dopasowanie do ograniczeń ustali rozmiar elementu na podstawie określonych dla niego ograniczeń horyzontalnych lub wertykalnych. Jednakże należy mieć na uwadze, że bazowa klasa komponentu może nie być przystosowana do zmiany rozmiarów (np. nie posiada algorytmu skalującego domyślnej grafiki). Pozycję kolejnych elementów możemy uzależniać od pozycji już umieszczonych widoków.

Pozycjonowanie relatywne (ang. relative positioning) jest najprostszym sposobem pozycjonowania. Kolejny element interfejsu użytkownika ustawiany jest wybraną krawędzią do wybranej krawędzi rodzica.



Rys. 2.2. Przykład relatywnego pozycjonowania[9]

Można rozszerzyć ten mechanizm o dodatkowe marginesy. Wówczas możemy wprowadzić atrybut z wartością liczbową mówiącą o tym o ile od siebie mają być oddalone ustalone wcześniej krawędzie. [9]

2.7.1. Elementu graficzne interfejsu (Konrad Ossowski)

Na utworzonej wcześniej warstwie ograniczeń możemy umieszczać wybrane przez nas komponenty interfejsu graficznego. Dla każdego z komponentów ustalamy ich rozmiar oraz położenie na warstwie ograniczeń.

Etykieta (ang. Text View)

Wyświetla na danym obszarze zwykły tekst. Służy głównie do przekazywania informacji. Brak interakcji z użytkownikiem.[18]

Przycisk (ang. Button)

Najprostszy interaktywny element, użytkownik może w niego kliknąć co spowoduje wywołanie metody `onClick()` w obiekcie Przycisku. [13]

Przycisk wyboru (ang. Check Box)

Ma bardzo podobne działanie do przycisku, jednakże różni się od niego tym, że posiada stan *prawdy* lub *falszu*. Akcję po zmianie stanu przycisku wyboru można przeprowadzić poprzez nadpisanie metody `onCheckChanged()`.

Pole tekstowe (ang. Edit Text)

Kolejny prosty interaktywny element, użytkownik po kliknięciu w to pole może wprowadzić jakąś wartość. Może to być wartość liczbową, słowną lub inną ustaloną przez programistę. Służy głównie do uzyskiwania danych od użytkownika. Istnieje opcja ustawienia reagenta na zmiany wartości tego pola, jest to obiekt klasy Obserwatora Tekstu (ang. Text Watcher).[15]

Lista rozwijana (ang. Spinner) i Widok listy rozszerzanej (ang. Expandable List View)

Wyświetla opcje w postaci listy. Dla tych kontrolek niezbędne jest stworzenie dwóch dodatkowych obiektów. Adaptera, czyli obiektu zawierającego dane do wyświetlenia w odpowiednim formacie. Reagenta na wybranie pozycji z listy (ang. On Item Click Listener) i odpowiednim nadpisaniu mu metody `onItemClick(AdapterView parent, View view, int position, long id)`.[12]

2.8. Singleton (Konrad Ossowski)

Zdarza się, że w ramach naszej aplikacji chcielibyśmy posiadać globalny obiekt zawierający zestaw pól i metod, z których chcielibyśmy korzystać. Problem w tym, że każdy nowo utworzony obiekt, ma bazowe parametry, a my chcielibyśmy obiekt, który raz zmieniony zapamiętuje swoją zmianę przy kolejnych dostępach. W takiej sytuacji tworzymy właśnie Singleton, niestety nie ma dobrego polskiego tłumaczenia. Realizuje się to poprzez utworzenie w tej klasie statycznego pola, które jest obiektem tej klasy oraz statycznej metody zwracającej właśnie to pole. Oto nasza realizacja:

```
1 public class SettingsSingleton {  
2     private static SettingsSingleton ourInstance = new SettingsSingleton();  
3 }
```

```
4     private SettingsSingleton(){};  
5  
6     public static SettingsSingleton getInstance(){  
7         return ourInstance;  
8     }  
9 }
```

Dzięki takiemu rozwiązaniu w dowolnym miejscu w kodzie możemy dostać się do obiektu *SettingsSingleton* poprzez *SettingsSingleton.getInstance()*. Wszelkie modyfikacje wprowadzone w tym obiekcie będą aktualne przy każdym dostępie do tego obiektu.

2.9. System plików (Konrad Ossowski)

Jako użytkownik, na urządzeniu mobilnym z systemem operacyjnym Android, mamy swobodny dostęp do systemu plików poprzez interfejs graficzny. Możemy dostać się do plików dowolnej aplikacji. Jako programista również mamy dostęp do systemu plików. Możemy utworzyć folder aplikacji w pamięci urządzenia i przechowywać w nim wszelkie niezbędne dane do jej prawidłowego funkcjonowania. Dostęp do głównego katalogu pamięci urządzenia realizuje się przy pomocy klasy *File*, która reprezentuje zarówno pliki jak i katalogi (zgodnie z filozofią systemu Linux). Podajemy jej w konstruktorze ścieżkę do tego folderu, można ją uzyskać poprzez statyczną metodę w klasie Środowiska (ang. Environment) - *Environment.getExternalStorageDirectory()*. Gdy mamy uchwyty do tego katalogu możemy w nim dowolnie tworzyć nowe katalogi metodą *makedirs()* lub pliki metodą *createNewFile()*.

2.10. Vibrator (Konrad Ossowski)

Wibrowanie urządzenia możliwe jest przy użyciu klasy Vibrator (ang. Vibrator). Obiekt tej klasy uzyskuje się poprzez wywołanie metody klasy Kontekst (ang. Context) *getSystemService(Context context)*. Jako parametr *context* należy podać stałą *Context.VIBRATOR_SERVICE*. Należy pamiętać, aby wynik tej metody zrzutować na klasę Vibratora.

Najprostszym użyciem tej klasy jest wywołanie metody *vibrate(long milliseconds)*, która spowoduje, że urządzenie będzie wibrowało przez podany okres. Istnieje możliwość stworzenia bardziej skomplikowanego schematu vibracji dzięki metodzie *vibrate (long[] pattern, int repeat)*. Parametr *pattern* to tablica, której elementy stojące na nieparzystych pozycjach są okresami wibrowania, a parzyste elementy to okresy przerwy między vibracjami. Parametr *repeat* informuje o tym ile razy podana sekwencja ma zostać powtórzona. Jeśli chcemy podany schemat wykonać tylko raz, należy podać w tym parametrze wartość -1. Vibrator udostępnia możliwość odgrywania efektów dźwiękowych podczas vibracji. Jeśli wyjdziemy z aktywności, która odpowiada za vibracje, nie będą one kontynuowane.[8]

2.11. Aparat (Maciej Plewka)

Możliwe jest użycie wbudowanego aparatu poprzez wbudowaną systemową aplikację. Wywołuje się ją przy użyciu intencji. Atrybutem określającym rodzaj akcji jest tutaj *MediaStore.ACTION_IMAGE_CAPTURE* podany jako parametr konstruktora intencji. Po utworzeniu inten-

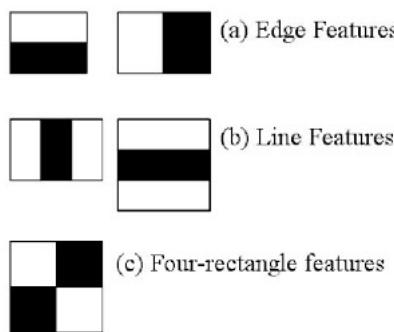
cji w jej dodatkach należy umieścić ścieżkę w jakiej ma być zapisane wykonane zdjęcie. W celu uruchomienia aparatu w bieżącym kontekście wołamy metodę startActivityForResult. W parametrach tej metody umieszczamy intencję oraz kod identyfikujący te wywołanie przy powrocie. Po zakończeniu działania aparatu w funkcji onActivityResult sprawdza się kod identyfikujący uruchomioną aktywność. Jeśli kod zgadza się z tym który umieściliśmy w wywołaniu aparatu, możemy wywołać akcje które powinny się wykonać zaraz po wykonaniu zdjęcie np zaktualizowanie widoku.

3. KASKADOWE KLASYFIKATORY CECH HAAR'A (MACIEJ PLEWKA)

Wybraną metodą zaimplementowaną w celu zrealizowania wariantu sztuczki z fotografią, w którym potrzebne jest wskazanie obiektu na zdjęciu, są kaskady Haar'a. Jest to bardzo skuteczny i szybki sposób detekcji oparty na kaskadowych klasyfikatorach cech. Metoda ta działa sprawnie przetwarzając wideo w czasie rzeczywistym, więc w doskonale sprawdza się w naszym przypadku, ze względu na szybki czas działania. Takie rozwiązanie detekcji obiektów zostało zaprezentowane przez Paula Viola i Michael Jones w jednej z ich prac naukowych [29].

3.1. Cechy Haar'a

Proces detekcji obiektów oparty jest o tzw. Cechy Haara. Są to prostokąty takie jak przedstawiono na rysunku 3.1. Zawierają one informacje na temat różnicy wartości pikseli na zaznaczonych obszarach, dodatkowo wartości te są przeskalowane odwrotnie proporcjonalnie do ich wielkości.



Rys. 3.1. Cechy Haara Źródło: [20]

Zdefiniowane zostały 3 rodzaje cech. Pierwsza (3.1a)) jest przedstawiana jako różnica sum pikseli znajdujących się na zaznaczonych prostokątach. Druga (3.1b)) Opisywany jest jako różnica sumy skrajnych prostokątów i prostokątu wewnętrznego. Trzecia cecha reprezentowana jako cztery prostokąty jest różnicą sum prostokątów leżących na jednej przekątnej. Każda z takich cech może zawierać informacje między innymi na temat krawędzi i różnych zmian w tekstuze na podstawie przejść z jasnych do ciemnych fragmentów i odwrotnie. Dla obrazu o rozmiarze 24x24 można wyznaczyć łącznie 162336 cech wszystkich typów przedstawionych na rysunku 3.1. Jednak do detekcji obiektu potrzebne jest tylko kilkadziesiąt lub kilkaset z nich. Algorytm 1

pokazuje obliczanie wszystkich cech Haar'a typu 3.1 a.

Algorithm 1: Wyznaczanie możliwych cech Haar'a typu 3.1a

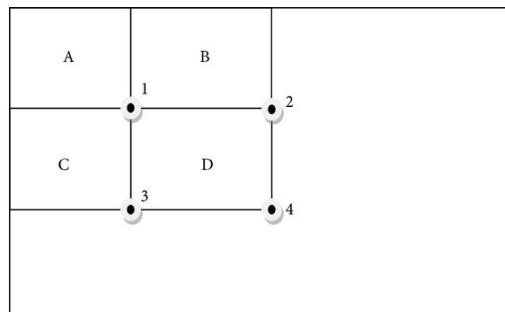
```
x, y; // wymiary zdjęcia, szerokosc i wysokosc
for i = 1 to x do
    for j = 1 to y do
        for w = 1 to i + h - 1 do
            for h = 1 to j + 2w - 1 do
                Oblicz sumę pikseli S1 w [i, i + h - 1] x [j, j + w - 1]
                Oblicz sumę pikseli S2 w [i, i + h - 1] x [j+w, j + 2w - 1]
                zapisz cechę o parametrach (i, j, w, h): S1 - S2
            end for
        end for
    end for
end for
```

3.2. Obliczanie wartości sumy pikseli

Każdorazowe obliczanie wartości sumy pikseli dla każdego prostokąta wewnętrz zdjecia byłoby bardzo kosztowne i algorytm detekcji wykonywałby się w długim czasie. W celu skrócenia czasu zastosowano tak zwany Obraz scałkowany (ang. Integral Image). Obraz ten zawiera sumy pikseli. W pierwszej kolejności obliczane są sumy pikseli dla każdego punktu (x,y) zdjecia w odniesieniu do lewego górnego rogu. Wartości te obliczane są analogicznie do algorytmu Summed-area table używanego w grafice [21]. Wzór 1 przedstawia w jaki sposób obliczana jest wartość obrazu scałkowanego dla punktu (x, y)

$$ii(x, y) = \sum_{x' < x, y' < y} i(x', y') \quad (1)$$

Wyznaczanie sumy zdjecia scałkowanego to dwie rekurencyjne operacje wektorowe, na kolumnach i wierszach. Posiadając obliczone wartości sumy pikseli w każdym punkcie obrazu w którym szukamy obiektu możemy łatwo obliczyć wartość pikseli wewnątrz każdego prostokąta w stałym czasie.

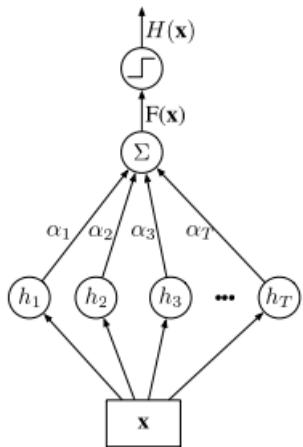


Rys. 3.2. Obliczanie sumy wartości pikseli Źródło: [26]

Rysunek 3.2 przedstawia schemat obrazu, po transformacji do obrazu scałkowanego dla którego próbujemy wyznaczyć sumę pikseli w wewnętrznym prostokącie. W celu obliczenia sumy pikseli dla zaznaczonego prostokąta D musimy uwzględnić wartości sum 4 prostokątów. Suma pikseli dla prostokąta A jest równa wartości zdjęcia scałkowanego w punkcie 1, Analogicznie w punkcie 2 opisuje wartość sumy dla prostokąta A + B, w punkcie 3 dla prostokąta A + C, natomiast w punkcie 4 dla prostokąta A + B + C + D. Do wyznaczenia sumy pikseli w prostokącie D potrzebujemy zatem wartości zdjęcia scałkowanego w punktach opisującym ten prostokąt. I tak dla przedstawionego schematu suma pikseli w prostokącie D jest równa $4 - 2 - 3 + 1$. Wyznaczenie tej wartości dla każdego prostokąta zatem zawsze jest wykonane w stałym czasie, są to zawsze te same operacje dodawania i odejmowania.

3.3. Proces uczenia mocnych klasyfikatorów

Każdy klasyfikator włączony do kaskady jest złożony z wielu tzw. Słabych klasyfikatorów. Każdy z nich jest pojedynczą cechą Haar'a która została włączona do klasyfikatora w procesie uczenia. Pojedynczy klasyfikator zwykle nie jest w stanie jednoznacznie sklasyfikować obiektu. W celu usprawnienia skuteczności detekcji za pomocą cech stosuje się tak zwany Boosting. Celem tego zabiegu jest stworzenie klasyfikatora, którego ocena będzie składała się z ocen wielu klasyfikatorów odpowiednio przeskalowane w stosunku do ich skuteczności określonym w etapie uczenia.



Rys. 3.3. Działanie mocnego klasyfikatora Źródło: [25]

Rysunek 3.3 przedstawia działanie mocnego klasyfikatora. Na wejściu podane jest zdjęcie X które jest analizowane przez każdą cechę h . Następnie wynik każdej cechy (możliwe $\{1, -1\}$) jest przemnażany przez jej współczynnik, na koniec wyniki są sumowane. Obliczona suma większa od zera oznacza, że klasyfikator odnalazł obiekt, w przeciwnym razie analizowany fragment nie został sklasyfikowany jako szukany element.

Mocne Klasyfikatory otrzymywane są w procesie uczenia maszynowego. Trening klasyfikatorów odbywa się w oparciu o zbiór danych uczących. W skład takiej bazy będą wchodziły zdjęcia, na których znajduje się szukany obiekt, tak zwane zdjęcia pozytywne, oraz zdjęcia na których nie ma obiektu, tak zwane zdjęcia negatywne. Dodatkowo dane muszą posiadać etykiety które informują czy na zdjęciu znajduje się obiekt czy też nie. W tym przypadku kaskady Haar'a są otrzymywane poprzez implementacje algorytmu AdaBoost jest to jeden ze sposobów uczenia z nadzorem. To iteracyjny algorytm, w każdej iteracji wybierany jest jeden słaby klasyfikator w tym przypadku cecha Haar'a, która najskuteczniej klasyfikuje obiekt. Dene, które zostały źle sklasyfikowane przez wybraną cechę zostają obciążone większymi wagami, by w następnych iteracjach kolejne klasyfikatory musiały radzić sobie szczególnie dobrze ze zdjęciami błędnie oznaczonymi przez poprzednio wybrane cechy. Tak dobrane słabe klasyfikatory łącznie tworzą silny klasyfikator, który jest włączany do kaskady. Na wejście algorytmu podany jest zestaw danych uczących wraz z etykietami $\{x_i, y_i\}$ x_i jest kolejnym zdjęciem a y_i jest etykietą $\{1, -1\}$. Zbiór danych liczy n

elementów.

Algorithm 2: AdaBoost

```

 $D_k(i)$ : // przykładowa waga elementu  $i$  klasyfikatora  $k$ 
 $\alpha_k$ : // waga klasyfikatora  $k$ 
 $\forall_i: D_1(i) \leftarrow \frac{1}{n}$  // początkowe wagi danych
for  $k = 1$  to  $K$  do
     $\forall h \epsilon_k \leftarrow \sum_{i:y \neq h(x_i)} D_k(i)$  // obliczanie błędu dla każdego klasyfikatora
     $h_k \leftarrow \operatorname{argmin}(\epsilon_k(h))$  // wybór klasyfikatora z najmniejszym błędem
    if  $\epsilon_k \geq \frac{1}{2}$  then
        stop // jeśli klasyfikator ma 50% błędu to stop
    else
         $\alpha_k \leftarrow \frac{1}{2} \frac{1-\epsilon_k}{\epsilon_k}$ 
         $D_{k+1}(i) \leftarrow \frac{D_k(i) e^{-y_i \alpha_k h_k(x_i)}}{Z_k}$ 
    end if
end for
 $H_{final}(x) = \operatorname{sgn}(\sum_{k=1}^K \alpha_k h_k(x))$ 

```

Ocena skuteczności klasyfikatora opiera się na tak zwanej macierzy pomyłek (z ang. Confusion Matrix), która definiuje współczynniki określające:

- Liczba dobrych klasyfikacji jako obiekt
- Liczba złych klasyfikacji jako obiekt
- Liczba dobrych klasyfikacji jako nie obiekt
- Liczba złych klasyfikacji jako nie obiekt

Na podstawie tych danych obliczany jest współczynnik dobrze sklasyfikowanych obiektów w stosunku do liczby zdjęć pozytywnych, dalej określany jako TPR (z ang. true positive rate). Obliczany jest także współczynnik błędnych decyzji jako stosunek liczby błędnie sklasyfikowanych jako obiekt do liczby zdjęć negatywnych, dalej określany jako FPR (z ang. false positive rate)

3.4. Kaskada klasyfikatorów mocnych

W celu dodatkowego poprawienia skuteczności zaproponowano zastosowanie kaskady klasyfikatorów, która jest czymś na wzór drzewa decyzyjnego. Każdy z mocnych klasyfikatorów otrzymanych na podstawie działania algorytmu AdaBoost jest jednym z elementów kaskady. Aktualnie poszukiwany obszar jest sprawdzany przez kolejne klasyfikatory. Na wejście kaskady podawany jest analizowany obszar. Pierwszy klasyfikator analizuje fragment wszystkimi cechami które wchodzą w jego skład. Jeśli klasyfikator stwierdzi że we wskazanym obszarze znajduje się obiekt fragment zostaje poddany analizie przez następny klasyfikator w kaskadzie. Jeśli fragment zostanie niesklasyfikowany przez któryś element kaskady zostaje podjęta jednoznaczna decyzja dla całej kaskady o braku obiektu. O obecności szukanego wzorca w kaskadzie, muszą wskazać wszystkie elementy kaskady. Podczas uczenia kolejne etapy wybierają do klasyfikatora mocnego cechy z coraz mniejszej puli jest ich zatem coraz więcej w kolejnych elementach kaskady.

Sama detekcja jest oparta na sprawdzaniu kolejnych fragmentów zdjęcia przesuwając miejsce badanego fragmentu po całym obrazie. Analiza fragmentów, które są tłem obiektu trwa stosunkowo krótko ponieważ o braku obiektu powinny stwierdzić pierwsze elementy kaskady, które składają się z mniejszej ilości cech niż końcowe etapy. Analiza przy użyciu końcowych etapów kaskady jest wykorzystana prawie tylko w przypadku badania fragmentu na którym znajduje się obiekt. Kosztowna analiza za pomocą wszystkich cech wchodzących w skład kaskady zazwyczaj jest wykonywana tylko dla mniejszej części zdjęcia, co sprawia, że algorytm jest szybki.

4. WYKORZYSTANIE BIBLIOTEKI OPENCV (MACIEJ PLEWKA)

Biblioteka OpenCV to otwarte oprogramowanie, które jest wykorzystywane do przetwarzania zdjęć. Jest to narzędzie wspierane przez Windows, GNU/Linux, macOS oraz Android. W naszej pracy biblioteka ta została wykorzystana na systemie Ubuntu (jednej z wielu dystrybucji systemu Linux) w celu wytrenowania własnej kaskady oraz testowania a także na platformie Android by wykorzystać wytrenowane modele zaraz po wykonaniu zdjęcia aparatem. Źródła biblioteki są publicznie dostępne w zdalnym repozytorium [22]. Narzędzia OpenCV umożliwiają wykorzystanie standardu OpenCL który pozwala na równoległe obliczenia z użyciem procesorów centralnych oraz graficznych.

4.1. *OpenCV w procesie trenowania kaskad*

Biblioteka OpenCV oprócz zestawu funkcji służących do wykonywania operacji na zdjęciach, dostarcza także aplikacje umożliwiające przeprowadzenie procesu treningowego własnych kaskadowych klasyfikatorów cech Haar'a. Pierwszą aplikacją jest opencv_createsamples, która pozwala na wygenerowanie dużego zbioru zdjęć pozytywnych. Program opencv_traincascade służy do właściwego treningu kaskady. Dodatkowo wykorzystać można jeszcze opencv_visualisation jest to aplikacja, dzięki której możemy w trakcie treningu zobrazować dotychczas dobrane cechy na tle wzoru.

4.1.1. *Generowanie zbioru zdjęć pozytywnych*

Przed rozpoczęciem trenowania naszej kaskady klasyfikatorów potrzebne są zdjęcia na których umieszczony jest obiekt który ma być rozpoznawany oraz kolekcje zdjęć na których nie ma naszego wzorca. By kaskada została skutecznie wytrenowana potrzebne jest jak najwięcej zdjęć. Do wytrenowania kaskady, która rozpoznaje twarze, dołączonej wraz z biblioteką OpenCV wykorzystano 3000 zdjęć pozytywnych oraz 1500 zdjęć negatywnych. Jasnym problemem staje się wykonanie takiej liczby zdjęć na których znajduje się wybrany obiekt a następnie oznać dokładnie jego położenie. Do stworzenia zbioru zdjęć pozytywnych wykorzystano aplikację opencv_createsamples. Program ten pozwala na wygenerowania zbioru zdjęć pozytywnych na podstawie jednego zdjęcia na którym tylko i wyłącznie znajduje się przedmiot który potem będzie szukany. Zdjęcie to zostaje umieszczone na każdym zdjęciu w losowych miejscach o losowym rozmiarze i o losowym przesunięciu w każdej z trzech płaszczyzn. Dobór zdjęć nie jest dowolny, ważne aby na zdjęciach na które zostanie nałożony nasz obiekt. Ostatnim elementem niezbędnym do skorzystania z aplikacji opencv_createsamples jest przygotowanie pliku tekstowego w którym wylistowane będą ścieżki do plików na które mają zostać nанiesione wzory naszego obiektu. Dla tak przygotowanych danych możemy uruchomić aplikację opencv_createsamples wraz ze wskazanymi parametrami:

- -img <ścieżka do wzorcowego zdjęcia >
- -bt <ścieżka do pliku z opisem zdjęć tła>

- -info <ściezka do pliku w którym mają być zapisane informacje o utworzonych zdjęciach>
- -pngout <ściezka do katalogu w jakim mają być zapisywane pozytywne zdjęcia>
- -bgcolor <ośmiobitowa wartość koloru tła>
- -bgtrash <zakres wartości koloru>
- -maxxangle <maksymalny kąt w płaszczyźnie x wyrażony w radianach>
- -maxyangle <maksymalny kąt w płaszczyźnie y wyrażony w radianach>
- -maxzangle <maksymalny kąt w płaszczyźnie z wyrażony w radianach>
- -num <liczba zdjęć która ma zostać wygenerowana>

Po uruchomieniu aplikacji we wskazanym folderze zostaną wygenerowane pozytywne zdjęcia.



Rys. 4.1. Wygenerowane pozytywne zdjęcie

Rysunek 4.1 przedstawia przykładowo wygenerowane pozytywne zdjęcie. Informacje na temat zdjęcia zapisany we wskazanym pliku info dla tego zdjęcia wygląda następująco:

0185_0196_0219_0293_0454.jpg 1 196 219 293 454 Elementy kolejno oznaczają:

- Nazwa pliku
- Liczba obiektów na zdjęciu
- Współrzędna x lewego górnego rogu obiektu na zdjęciu
- Współrzędna y lewego górnego rogu obiektu na zdjęciu
- Szerokość obiektu
- Wysokość obiektu

Posiadając plik info z informacjami na temat położenia zdjęć możemy uzyskać plik z rozszerzeniem vec, który następnie będzie używany przez aplikację trenującą kaskadę. Posłuży nam do tego także aplikacja opencv_createsamples. Tym razem należy wskazać następujące parametry:

- -info <ściezka do pliku info w którym znajdują się informacje o obiektach >
- -vec <ściezka do pliku w jakim ma być zapisany plik vec>
- -num <ilość zdjęć jaka ma być zapisana w pliku vec>
- -w <zerokość obiektu>
- -h <wysokość obiektu>

4.1.2. *Trening kaskady*

Do treningu kaskady wykorzystujemy aplikację opencv_traincascade, która implementuje algorytm 2 opisany w punkcie 3.3. Aplikacja zostanie uruchomiona z następującymi parametrami:

- -vec <ścieżka do pliku z wektorem opisującym pozytywne zdjęcia >
- -bg <sciezka do pliku z zapisanymi ścieżkami plików tworzących zbiór zdjęć negatywnych >
- -numPos <liczba pozytywnych zdjęć użyta do treningu >
- -numNeg <liczba zdjęć negatywnych użytych do treningu >
- -w <szerokość obiektu, taka sama jak w podanym pliku vec >
- -h <wysokość obiektu, taka sama jak w podanym pliku vec >
- -numStages <liczba elementów z których ma się składać kaskada >
- -maxyangle <maksymalny kąt w płaszczyźnie y wyrażony w radianach>
- -maxzangle <maksymalny kąt w płaszczyźnie z wyrażony w radianach>
- -num <liczba zdjęć która ma zostać wygenerowana>

Te parametry są niezbędne do uruchomienia treningu za pomocą tej aplikacji. Możliwe do podania są także inne parametry, które deklarują inną dokładność słabych i silnych klasyfikatorów niż zdefiniowanych domyślnie przez aplikację. W przypadku podania parametrów które stwierdzają większość dokładność algorytm prawdopodobnie będzie bardziej skuteczny lecz trening będzie trwał dłużej. Natomiast przy zmniejszeniu czas treningu ulegnie skróceniu. Te parametry to:

- -minHitRate <minimalna wartość współczynnika oznaczającego pozytywne klasyfikacje >
- -maxfalsealarm <maksymalna dopuszczalna wartość błędnych decyzji >

Podczas trenowania kaskady przydatna jest także aplikacja opencv_visualisation, która dla dotychczasowych etapów generuje podgląd dobranych cech na tle obiektu. Program ten przyjmuje następujące parametry:

- -data <ścieżka do wygenerowanych obrazów z naniesionymi cechami >
- -image <zdjęcie obiektu o rozmiarze takim jaki podany do uczenia kaskady >
- -model <ścieżka do pliku xml z wytrenowanym dotychczas modelem kaskady >

Po uruchomieniu aplikacji zostaną wygenerowane pliki zdjęciowe, osobno dla każdego etapu kaskady. Na każdym z utworzonych obrazów znajduje się wizualizacja osobno wszystkich cech wchodzących w skład danego etapu. Obok zdjęć generowany jest także plik wideo w formacie avi na którym przedstawione są po kolejno wszystkie cechy wchodzące w skład kaskady na tle obiektu. Rysunek 4.2 przedstawia przykładową wizualizację jednego z etapów kaskady.



Rys. 4.2. Cechy mocnego asyfikatora

4.2. OpenCV na platformie Android

Biblioteka OpenCV dla Androida dostarczona jest w zestawie narzędzi deweloperskich OpenCV. Daje ona api Javy. Posiada on prawie wszystkie funkcjonalności Javy. Wszystkie operacje wykonywane są przy użyciu natywnych metod zaimplementowanych w języku C++. Istnieją dwa podejścia w wykorzystaniu tej biblioteki. Pierwsze zakłada wołanie funkcji poprzez dostarczone api Javy pozwalające bezpośrednio i w łatwy sposób wykorzystanie funkcji w celu przetwarzania obrazu. Wadą takiego podejścia jest wykorzystanie metod JNI, które trwają stosunkowo długo, do tego dochodzi kwestia kopiowania i mapowania obiektów co także mocno wykorzystuje zasoby. Drugie podejście zakłada użycie metod biblioteki z poziomu funkcji natywnych. Zastosowanie takiego rozwiązania jest korzystne jeśli w konkretnym bloku wykonujemy szereg funkcji które z poziomu kodu java wołyby metody jni. W takim przypadku, kosztowne zwołanie metody przy użyciu jni zostało by wykonane raz, natomiast reszta natywnych funkcji została by zwołana bezpośrednio z niskiego poziomu. Takie rozwiązanie jest szybsze oraz wykorzystuje mniej zasobów. Dodatkowo w przypadku wywoływanego metod z niskiego poziomu mamy dostęp do wszystkich funkcji dostarczanych przez bibliotekę. Jest to rozwiązanie bardziej skomplikowane, trzeba także pamiętać o odpowiednim zarządzaniu pamięcią, gdyż garbage collector działa tylko w ramach maszyny wirtualnej java.

4.2.1. Detekcja obiektów

Cały proces detekcji odbywa się przy użyciu kilku funkcji z biblioteki openCV. Na początku należy wczytać zdjęcie. W zależności od źródła zdjęcia może być ono odczytane np. jako pojedyncza klatka z kamery lub tak jak w naszym przypadku pojedyncze zdjęcie z pamięci urządzenia. Służy do tego metoda imread wywoływana na obiekcie Imgcodecs. Parametrami tej funkcji są: ścieżka do pliku zdjęciowego oraz parametr oznaczający format w jakim ma być odczytane zdjęcie. Takimi formatami są:

- CV_LOAD_IMAGE_UNCHANGED (<0) ładuje zdjęcie w takim formacie, w jakim oryginalnie jest zapisane.
- CV_LOAD_IMAGE_GRAYSCALE (0) ładuje zdjęcie w skali szarości
- CV_LOAD_IMAGE_COLOR (>0) ładuje zdjęcie w formacie BGR

Do późniejszej detekcji potrzebujemy zdjęcia wczytanego w skali szarości dlatego wczytujemy je z

parametrem cv2.IMREAD_GRAYSCALE. Załadowane zdjęcie znajduje się w obiekcie klasy Mat. Jest to klasa przechowująca macierze oraz obrazy. Posiada wiele właściwości opisujące zdjęcie takie jak np. rozmiar. Do detekcji potrzebujemy załadować także nasz plik z kaskadą w formacie xml. Tworzymy obiekt klasy CascadeClassifier, w konstruktorze obiektu podajemy ścieżkę do naszego wytrenowanego modelu. Kiedy mamy przygotowany obraz oraz kaskadę, możemy na obiekcie z załadowanym modelem zwołać metodę detectMultiScale która przyjmuje następujące parametry:

- *image* to obiekt klasy Mat opisujący załadowane zdjęcie na którym szukamy obiektu.
- *objects* to referencja do kolekcji MatOfRect, w której zostaną zapisane fragmenty zdjęcia z szukanym obiektem.
- *scaleFactor* określa stopień zmiany wielkości analizowanego fragmentu w kolejnych iteracjach algorytmu.
- *minNeighbors* określa minimalną liczbę sąsiednich fragmentów uznanych jako wykryty obiekt, żeby uznać go za pozytywny
- *minSize* jest to obiekt typu Size, określający minimalną wielkość analizowanego fragmentu zdjęcia.
- *maxSize* jest to obiekt typu Size, określający maksymalną wielkość analizowanego fragmentu zdjęcia.

Po wykonaniu metody detectMultiScale w kolekcji MatOfRect, której referencja była jednym z argumentów wywołanej metody, zapisane są informacje o współrzędnych odnalezionych obiektów. W tych miejscach możemy wkleić prostokąt lub w jakiś inny sposób modyfikować.

5. TRENING WŁASNEJ KASKADY (MACIEJ PLEWKA)

W celu zrealizowania detekcji naszej karty, potrzebne było wytrenowanie własnej kaskady klasyfikatorów Haar'a. W tym celu skorzystano z aplikacji dostarczonych przez openCV szerszej opisanych w rozdziale 4. Cały proces uczenia został wykonany na komputerze osobistym z procesorem Intel Core i3 4000M o częstotliwości taktowania zegara 2,4 GHz i 8GB pamięci operacyjnej. Biblioteka OpenCV wraz z aplikacjami zostały skompilowane ze źródeł znajdujących się w publicznym repozytorium [22]. Dodatkowo podczas budowania biblioteki zaznaczono flagę `by` zostało włączone wspieranie standardu OpenCL, który umożliwia zrównoleglenie obliczeń, co znacząco przyspiesza czas działania zbudowanych aplikacji.

5.1. Zbiór danych uczących

Zdobycie przynajmniej 2000 zdjęć wykorzystanych do uczenia jest dużym problemem. Dlatego dane uczące zostały wygenerowane przy użyciu aplikacji `opencv_createsamples`, opisanej w rozdziale 4.1.1. Do wygenerowania zdjęć pozytywnych wykorzystano 5 zdjęć na których znajduje się tylko i wyłącznie nasz wzór. W pierwotnych założeniach rozpoznawany miał być rewers wybranej karty. Z powodu skomplikowanego wzoru rewersu karty, gdzie kolor biały i czarny gesto się przeplata, postanowiono proces treningu przeprowadzić także na własnej, przygotowanej karcie, której wzór jest stosunkowo prosty. Potrzebny był także zbiór zdjęć na którym nie znajduje się nasza karta. Wykorzystano do tego serwis internetowy [27], który posiada bogaty zbiór sklasyfikowanych zdjęć, udostępniony z intencją wykorzystania do uczenia maszynowego. Z tej bazy zdjęć pobrano niecałe 3000 zdjęć sklasyfikowanych jako niedźwiedzie, robaki, gitary, pociągi i książki. Zbiór tych zdjęć pogrupowano na 5 podgrup po 500 zdjęć. W celu skrócenia czasu treningu zdjęcia te zostały zmniejszone za pomocą funkcji `resize` biblioteki `openCv`, która wykorzystuje interpolację. Jest to proces który dzięki któremu wyznaczamy wartość nowego piksela na podstawie innych sąsiadujących z nim pikseli. Zdjęcia tła zostały zmniejszone do rozdzielczości 120x186 pikseli. Jest to wymiar którego stosunek wysokości i szerokości jest taki sam jak w przypadku karty. Obrazy wzorów kart zostały również pomniejszone, jednak do rozdzielczości 40x62. Z początku rozważano użycie mniejszego rozmiaru zdjęć, co spowodowałoby zmniejszenie liczby możliwych cech Haar'a dla obiektu. Dość skomplikowany wzór awersu karty powodował nieczytelność zdjęcia, a co za tym idzie cechy słabo generalizowały by odpowiednie fragmenty prawdziwego obiektu. Wszystkie wykonane operacje dla obu wzorów dalej będą wykonywane analogiczne, z tymi samymi parametrami.



Rys. 5.1. Rewersy kart użyte do treningu



Rys. 5.2. Wzory własnej karty użyte do treningu

Zdjęcia zostały podzielone na 5 katalogów, by każdy z nich odpowiadał jednemu obrazowi wzoru. By utworzyć zbiór zdjęć pozytywnych potrzebujemy jeszcze pliku tekstowego w którym wylistowane będą ścieżki do zdjęć będących tłem wybranego wzoru. Posiadając wszystkie wymagane pliki tj. zdjęcie z pozytywnym wzorem oraz plik tekstowy ze ścieżkami do obrazów tła, możemy uruchomić aplikację opencv_createsamples. Dodatkowo ustawiono losowy kąty nachylenia na, do 0,2 rad w płaszczyźnie x oraz y natomiast w płaszczyźnie z do 0,1 rad. Koniecznym też było ustawienie odpowiedniego koloru tła naszego pozytywnego zdjęcia. Nasze pozytywne zdjęcia zawierają tylko i wyłącznie przycięty wzór, bez żadnego tła. Domyślnie jako kolor tła jest brana wartość 0 tj. kolor czarny, co w przypadku obu kart powoduje ignorowanie niektórych fragmentów, powodując że są one jakby przezroczyste. Kolor tła w tej aplikacji określany jest jako zakres obliczany z wartości parametrów bgcolor oraz bgthresh. Jako tło brany jest każdy kolor z przedziału od bgcolor-bgthresh do bgcolor+bgthresh. W naszych wzorach tło nie występowało, więc jako parametr -bgcolor podano 200 tj. kolor nie znajdujący się na naszym wzorze a -bgthresh przyjęto wartość 1. Po uruchomieniu aplikacji posiadamy 2500 sklasyfikowanych zdjęć pozytywnych na których znajdują się wzory, których detekcje chcemy osiągnąć. Teraz dla każdego pliku info z informacjami na temat wygenerowanych obrazów możemy wygenerować plik z rozszerzeniem vec, który jest potrzebny do poprawnego uruchomienia aplikacji trenującej kaskadę klasyfikatorów. Polecenie które uruchomiło wygenerowanie obrazów dla jednego pozytywnego zdjęcia i 500 zdjęć będących tłem wygląda następująco:

```
1 opencv\_createsamples -img pos?/pos5.jpg -bg bg5.txt -info info/info5.list -pngoutput
   info/ -bgcolor 200 -bgthresh 1 -maxxangle 0.2 -maxyangle 0.2 -maxzangle 0.1 -num 500
   -w 40 -h 62
```

Aplikacja została uruchomiona dla pięciu różnych zdjęć karty. Kolejne polecenia uruchamiające różniły się wskazaniem na inne zdjęcie wzorca, inny plik ze ścieżkami zdjęć będących

tłem oraz innym plikiem info do którego zapisywane są informacje o wygenerowanych zdjęciach. Ostatnim krokiem kończącym proces przygotowania danych jest połączenie pięciu plików vec w jeden, który będzie jednym z argumentów wejścia programu opencv_traincascade. Wykorzystujemy do tego skrypt mergevec.py [23]. Narzędzie to przyjmuje dwa argumenty:

- -v <Scieżka do katalogu z plikami vec które należy scalić >
- -o <nazwa scalonego wynikowego pliku >

Pliki vec skopiowano do katalogu vec, skrypt mergevec.py znajduje się w katalogu mergevec. Polecenie które wygenerowało scalony plik vec wygląda następująco:

```
1 python mergevec/mergevec.py -v vec/ -o posss.vec
```

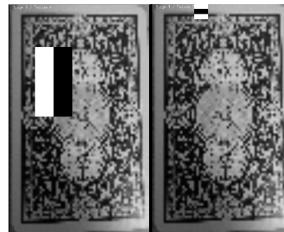
5.2. Proces treningu

Proces treningu kaskady przeprowadzono dla dwóch wzorów. Jednym z nich jest rewers karty, z talii kart do gier karcianych, drugi to przygotowany przez nas wzór przedstawiający znak zapytania na czerwonym tle. Dla każdego wzorca proces uczenia odbywał się na podstawie zdjęć wygenerowanych i przeskalowanych do tej samej wielkości. Początkowo zakładano by rozmiar wzorców wynosił 20x31 pikseli, by obniżyć do minimum liczbę możliwych cech Haar'a. Po przeskalowaniu zdjęć do określonej wielkości okazało się, że rewers kart jest bardzo niewyraźny, co ogranicza zdolność cech do generalizowania. Dlatego rozmiar wzorców ustalono na 40x62 pikseli. Wymiar ten zachowuje proporcje naszych kart. Liczba unikalnych cech Haar'a dla takiego rozmiaru wynosi 2978000. Rysunki 5.2 i 5.1 przedstawiają wzory użyte do treningu kaskad po przeskalowaniu. Przeprowadzony trening odbywał się na zbiorze tysiąca ośmiuset zdjęć pozytywnych oraz dziewięciuset zdjęciach negatywnych. Obrazy te posiadały jednakowy rozmiar 120x186. Zastosowano taki rozmiar by skrócić czas treningu, ponieważ dla większego zdjęcia podczas treningu trzeba przeanalizować większą ilość fragmentów. By zapobiec przeuczeniu tj. sytuacji w której kaskada będzie wytrenowana tylko i wyłącznie do rozpoznawania zdjęć treningowych, postanowiono ograniczyć się do wygenerowania dwunastu etapów kaskady. Do uruchomienia aplikacji potrzebne są: wygenerowany plik vec oraz plik ze ścieżkami zdjęć negatywnych. Współczynniki pozytywnych decyzji i negatywnych zostały zastosowane takie jak domyślne wartości aplikacji TPR = 0.995 a FPR = 0.5. Komenda uruchamiająca trening dla obu wzorów wygląda następująco:

```
1 opencv\traincascade -data data -vec posss.vec -bg bgall.txt -numPos 1800 -numNeg 900 -numStages 11 -w 40 -h 62
```

5.2.1. Trening detekcji rewersu karty

Ze względu na pierwotne założenia pierwszym wzorem na którym był przeprowadzony proces detekcji był rewers karty. Proces uczenia trwał trzydzieści osiem godzin. Wytrenowana kaskada składa się z dwunastu etapów i posiada dziewięćdziesiąt osiem cech opisujących wzór karty.

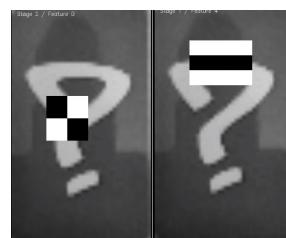


Rys. 5.3. Cechy opisujące rewers karty

Rysunek 5.3 przedstawia przykładowe cechy włączone do kaskady celnie opisujące wzór. Wytrenowana kaskada słabo radzi sobie z jednoznaczną detekcją obiektów. Na dziesięć zdjęć na których znajduje się tylko i wyłącznie karta na różnych tłaach, tylko na dwóch z nich celnie odnaleziono obiekt. Dodatkowo w obrębie karty odnalazł także kilka obiektów, więc na żadnym ze zdjęć nie był w stanie jednoznacznie wskazać obiektu. Powodem takiego wyniku jest najprawdopodobniej skomplikowany wzór, który gęsto przeplata biały i czarny kolor. Spowodowało to, że cechy pasują do różnych miejsc w obrębie karty, przez co nie jest w stanie jednoznacznie odnaleźć obiektu. Kolejnym elementem wpływającym niekorzystnie na detekcję jest pomniejszony obraz wzoru. Spowodowało to, że cechy odzwierciedlają wyraźnie zniekształconą wersję obiektu.

5.2.2. *Trening detekcji karty ze znakiem zapytania*

Z powodu słabej skuteczności kaskady przeznaczonej do detekcji rewersu karty postanowiono użyć innego wzoru. Słaba skuteczność spowodowana była najprawdopodobniej skomplikowanym wzorem. Trening kaskady został uruchomiony z takimi samymi parametrami jak w przypadku rewersu karty. Cały proces uczenia zajął jedenaście godzin. Kaskada składa się z jedenastu etapów, ponieważ po jedenastym etapie, współczynnik FPR osiągnął już zadowalający poziom. Kaskada składa się z pięćdziesięciu czterech cech Haar'a.



Rys. 5.4. Cechy opisujące naszą kartę

Rysunek 5.4 przedstawia przykładowe cechy celnie opisujące naszą kartę. Na dziesięć zdjęć zawierających jeden obiekt na jednolitym tle, obiekt został poprawnie oznaczony w każdym z nich. Trzeba zaznaczyć, że na trzech zdjęciach zaznaczone zostały także fragmenty nie związane z szukaną kartą. Otrzymano zatem siedem jednoznacznych decyzji o odnalezieniu obiektu, więc można powiedzieć, że osiągnięto sukces.

6. UŻYCIE KASKAD W CELU REALIZACJI TRICKU (MACIEJ PLEWKA)

W tym rozdziale opisany zostanie sposób użycia kaskady do zrealizowania dwóch wariantów sztuczki. Pierwszy to ten w którym wklejana jest karta w miejsce odnalezionej czoła, w drugim przypadku zamieszczamy ją w miejsce wyznaczonego obiektu. Opisane będą także sposoby dzięki którym osiągnięto optymalną skuteczność.

6.1. Detekcja Czoła

W celu detekcji czoła wykorzystano kaskady zapisane w plikach w plikach z rozszerzeniem xml, które są dołączone razem z biblioteką openCV. Wykorzystano plik kaskadowy pliki lbpcascade_frontalface.xml oraz haarcascade_lefteye_2splits.xml. Pierwszy z nich opisuje klasyfikatory służąca do rozpoznawania przodu twarzy, drugi teoretycznie służy do detekcji lewego oka które jest otwarte. Kaskada służąca do rozpoznawania lewego oka w praktyce jest w stanie skutecznie rozpoznać także prawe oko. Dzięki temu przy użyciu jednego modelu jesteśmy w stanie wskazać parę oczu, wykonując proces detekcji tylko raz. W naszej aplikacji miejsce czoła zostaje wyznaczone na podstawie informacji na temat położenia twarzy oraz oczu. Wykrywanie oczu zostaje uruchomione w momencie rozpoznania twarzy, gdyż wiadomym jest, że do poprawnego wykrycia oczu muszą się one znajdować w twarzy. Następnie, jeśli udało się nam odnaleźć w zdjęciu twarz, a w niej parę oczu uznajemy że detekcja przebiegła poprawnie. Zwrócone obiekty są typu Rect. Jest to typ z biblioteki openCV. W tym momencie wyznaczamy matematycznie domniemanie miejsce czoła na podstawie położenia na zdjęciu twarzy oraz oczu. Lecz zanim określmy dokładne miejsce czoła należy określić które oko znajduje się po lewej. Za lewy dolny punkt czoła przyjmiemy lewy górny punkt oka znajdującego się po lewej stronie, natomiast prawy dolny punkt jest równy co do wartości prawemu górnemu rogowi oka prawego. Różnica tych wartości jest zatem szerokością czoła. Wysokość czoła jest to szerokość czoła podzielona przez 2 albo jeśli wyznaczona w ten sposób wartość nie znajduje się w obrębie twarzy. Za wysokość przyjmujemy różnice między wyznaczonymi dolnymi punktami czoła a współrzędną y prostokąta znajdującego się w twarzy.

6.2. Detekcja Karty

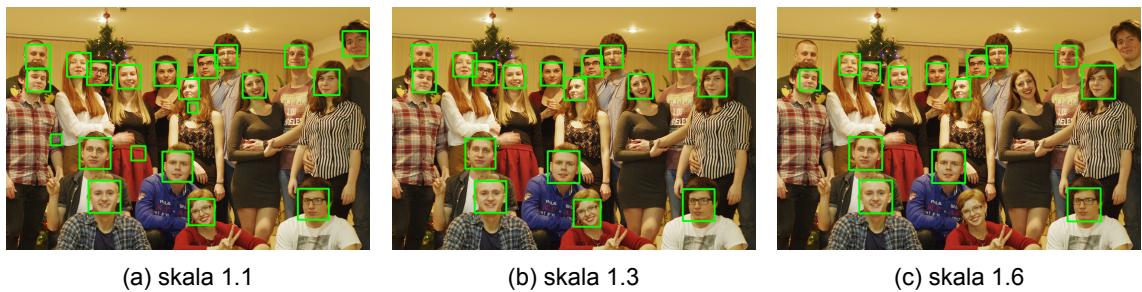
W odróżnieniu od detekcji czoła znalezienie karty ogranicza się do zwołania metody detectmultiscale na wcześniej załadowanej kaskadzie. Po otrzymaniu informacji na temat odnalezionych obiektów analizuje się ich liczbę. Przyjęto, że detekcja zostanie oceniana jako poprawna, gdy odnaleziony zostanie dokładnie jeden element, ponieważ sztuczka będzie wykonywana jednej osobie na raz, a osoba ta będzie trzymać jedną kartę. W przypadku, w którym rozpoznany będzie więcej niż jeden obiekt lub nie odnaleziony będzie ani jeden, zostanie zwrócony komunikat o potrzebie ponownego wykonania zdjęcia.

6.3. Dobór parametrów detekcji

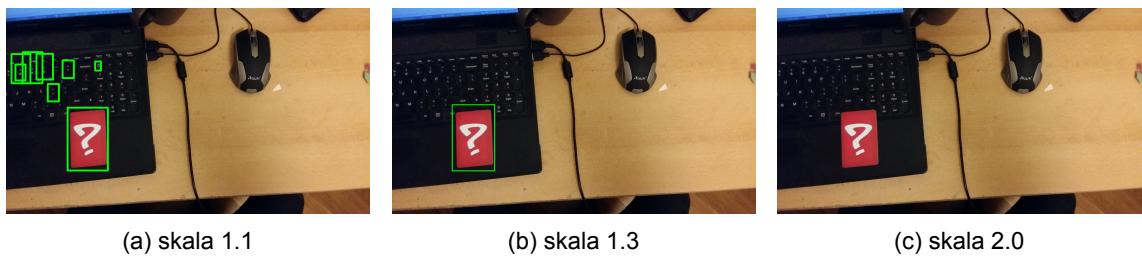
Parametry które mają wpływ na skuteczność klasyfikatora podczas detekcji oraz sprawiają, że cały proces trwa dłużej są:

- -minNeighbors
- -scaleFactor

Parametr scaleFactor determinuje różnice między wielkościami ramki, w której szukany jest obiekt, podczas kolejnych iteracji algorytmu. Mała wartość tego parametru ogranicza przypadek pominięcia szukanego wzorca. Jednak wraz ze wzrostem liczby analizowanych fragmentów, rośnie liczba obszarów dla których zostaje podjęta błędna decyzja o odnalezieniu obiektu. Dodatkowo duża liczba miejsc do przeanalizowania znacząco wydłuża czas działania detekcji. Duża wartość tego współczynnika może spowodować przeoczenie obiektu.



Rys. 6.1. Detekcja twarzy dla różnych wartości scaleFactor



Rys. 6.2. Detekcja karty dla różnych wartości scaleFactor

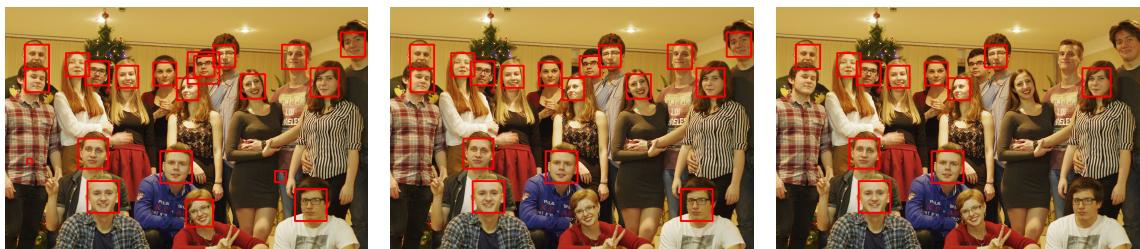
Tabela 6.1. Wyniki detekcji w zależności od parametru scaleFactor

Rysunek	6.1a	6.1b	6.1c	6.2a	6.2b	6.2c
Skala	1.1	1.3	1.6	1.1	1.3	2.0
Czas	379,397	161,874	105,139	475,516	215,779	108,829
F_{pos}	3	0	0	7	0	0
F_{neg}	0	0	3	0	0	1

Rysunki 6.1 i 6.2 przedstawiają wyniki detekcji twarzy oraz karty dla różnych parametrów scaleFactor. W tabeli 6.1 przedstawiono czas detekcji, liczbę błędnych klasyfikacji jako obiekt oznaczonych jako F_{pos} oraz liczbę błędnych klasyfikacji jako nie obiekt F_{neg} . Jak widać im mniejszy parametr tym czas krótszy, F_{pos} także się zmniejsza natomiast rośnie wartość F_{neg} .

Parametr minNeighbors określa w ilu sąsiednich ramkach został odnaleziony obiekt, dopiero przy odpowiedniej liczbie tych sąsiadów podejmowana jest decyzja o znalezieniu wzoru.

Zwiększenie tego współczynnika zmniejsza prawdopodobieństwo podjęcia błędnej decyzji o poprawnej detekcji, natomiast zwiększa ryzyko pominięcia elementu.

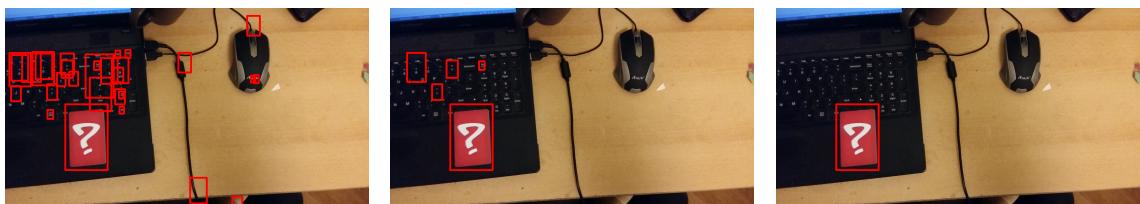


(a) minNeighbors równe 1

(b) minNeighbors równe 3

(c) minNeighbors równe 8

Rys. 6.3. Detekcja twarzy dla różnych wartości minNeighbors



(a) minNeighbors równe 1

(b) minNeighbors równe 6

(c) minNeighbors równe 16

Rys. 6.4. Detekcja karty dla różnych wartości minNeighbors

Tabela 6.2. Wyniki detekcji w zależności od parametru minNeighbors

Rysunek	6.3a	6.3b	6.3c	6.4a	6.4b	6.4c
Liczba sąsiadów	1	3	8	1	6	16
Czas	162,046	163,162	162,370	185,266	193,285	193,086
F_{pos}	2	0	0	28	4	0
F_{neg}	0	1	6	0	0	0

Rysunki 6.3 i 6.4 przedstawiają wyniki detekcji twarzy oraz karty dla różnych parametrów minNeighbors. W tabeli 6.2 przedstawiono rezultaty detekcji dla tych rysunków. Wynika z tego, że zmiana minimalnej liczby sąsiadów nie wpływa na czas detekcji. Przy zmianie parametru można jednak zauważać dużą zmianę skuteczności. Dla wykrywania twarzy widać mniejszą liczbę F_{pos} , stąd też dla mniejszej liczby minimalnych sąsiadów osiągnięto idealny rezultat. W przypadku detekcji twarzy oraz oczu zdecydowano by zastosowano sugerowane wartości dla tych kaskad, są to:

- minNeighbors = 3
- -scaleFactor = 1.3

Dla detekcji karty postanowiono przetestować dla jakich parametrów osiągnięto najlepszą skuteczność. W tym celu zebrano 10 zdjęć testowych na których znajduje się jedna karta. Na początku postanowiono ustawić parametr minNeighbors. Zdjęcia przeszły proces detekcji, gdzie parametr minNeighbors przyjmował co drugą wartość z zakresu <1,40> oraz scaleFactor równemu 1.3. Sprawdzano liczbę odnalezionych obiektów. W przypadku wskazania jednego obiektu nie będącego szukaną kartą, nie brano takiego pomiaru pod uwagę.

Tabela 6.3. Poszukiwanie najlepszej wartości minNeighbors cz. 1

minNeighbors	1	3	5	7	9	11	13	15	17	19
Liczba odnalezionych	11,62	6,89	4,89	3,68	3,03	2,86	2,51	1,89	1,68	1,82

Tabela 6.4. Poszukiwanie najlepszej wartości minNeighbors cz. 2

minNeighbors	21	23	25	27	29	31	33	35	37	39
Liczba odnalezionych	1,34	1,20	1,06	0,91	0,75	0,65	0,48	0,44	0,41	0,34

W tabelach 6.3 i 6.4 przedstawiono średnie wartości liczby odnalezionych obiektów w zależności od parametru minNeighbors. Wynika z nich, że najlepsze decyzje były podejmowane dla wartości 25. Na podstawie tego wyniku szukano najlepszej wartości parametru scaleFactor. Podobnie jak w poprzednim przypadku użyto tych samych dziesięciu zdjęć. Testowano wartości współczynnika scaleFactor od 1,1 do 1,7 co 0,05.

Tabela 6.5. Poszukiwanie najlepszej wartości scaleFactor

minNeighbors	1.1	1.15	1.2	1.25	1.3	1.35	1.4	1.45	1.5	1.55	1.6	1.65	1.7
Liczba odnalezionych	3,9	2,6	2,1	1,4	1,5	1,3	1,2	1,2	1,3	0,8	0,8	0,9	0,8

W tabeli 6.5 przedstawiono średni wynik ilości odnalezionych obiektów w zależności od parametru scaleFactor. Dla detekcji karty w naszej aplikacji parametr ten będzie wynosił 1,4, ponieważ dla takiego parametru uzyskano najlepszą wartość będącą większą od 1, jednocześnie z pośród innych parametrów dla których uzyskano podobny wynik, ten gwarantuje mniejszą szansę przeoczenia obiektu.

6.4. Zmniejszanie zdjęcia

W przypadku naszej aplikacji wymaganie związane ze skutecznością ma większy priorytet niż czas działania. Dlatego założono, że w przypadku kiedy nie zostanie podjęta jednoznaczna decyzja o odnalezieniu dokładnie jednego przedmiotu należy pomniejszyć zdjęcie i ponownie spróbować odnaleźć obiekt. Działanie takie podjęto ponieważ kaskada została wytrenowana na podstawie pomniejszonych zdjęć, więc powinna lepiej generalizować dla zdjęć pomniejszonych. W celu sprawdzenia tej teorii zebrano 10 zdjęć, dla których nie udało się wskazać dokładnie jednego obiektu, tak jak powinno zostać określone. Algorytm zakłada, że jeśli nie odnaleziono dokładnie jednego elementu, wysokość i szerokość zdjęcia dzielona jest przez 1,5. Następnie ponownie przeprowadzana jest detekcja obiektu. Proces ten zakończy się, jeśli na którymś z etapów odnaleziono obiekt, lub szerokość osiągnie wartość minimalną, ustaloną na 120 pikseli. W rezultacie testu okazało się, że dla sześciu z dziesięciu zdjęć, w których przy oryginalnym rozmiarze wynik detekcji był niepoprawny, dla pomniejszonego zdjęcia udało się poprawnie odnaleźć obiekt. Operacja zmiany wielkości zdjęcia jest kosztowna, jednak trwa znacznie krócej niż wykonanie ponownego zdjęcia i przeprowadzenie dla niego detekcji. Algorytm służący do testowania przeniesiono do kodu aplikacji, co znacząco poprawiło skuteczność

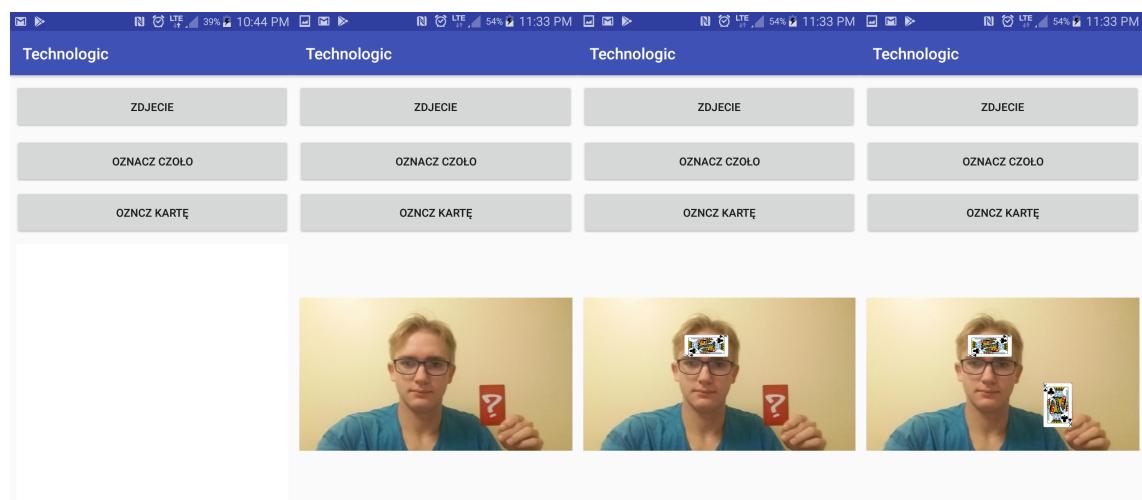
6.5. Wklejanie obrazu karty w miejsce obiektu

Obrazy całej talii kart znajdują się zasobach aplikacji. Każdy z obrazów ma przypisane unikalne ID. Podczas wykonywania sztuczki, w momencie zbliżenia znacznika NFC z wiadomości zawartej w tagu odczytujemy ID przypisane do karty i zapisujemy w pamięci, tak by w trakcie wklejania wiadomym było która karta powinna zostać wklejona. W momencie kiedy detekcja obrazu jest zakończona, zostaje wywołana procedura wklejania obrazu w miejsce wskazanego obiektu. W wyniku detekcji otrzymujemy współrzędne obiektu oraz jego szerokość i wysokość. Pierwszym etapem jest wczytanie obrazu karty z zasobów aplikacji. Następnie w zależności od wariantu, jeśli wklejamy obraz na czoło występuje konieczność obrotu obrazu karty, ponieważ w zasobach jest zapisane w orientacji pionowej. Tak wczytany obraz karty następnie jest pomniejszany do rozmiarów odnalezionego obiektu. Kolejnym krokiem jest wyznaczenie w wykonanym zdjęciu obszaru obiektu, po czym w to miejsce jest kopiowana nasza przygotowana karta. Cały ten opis realizuje następująca funkcja.

```
1 public void insertCardIntoImage(Mat img, Point lt, Point br) throws IOException {
2     Mat card = Utils.loadResource(mContext, resourceId, Imgcodecs.CV_LOAD_IMAGE_COLOR);
3     if(br.x-lt.x> br.y-lt.y) {
4         org.opencv.core.Core.flip(card.t(), card, 0);
5     }
6     Imgproc.resize(card, card, new Size((int)br.x - (int)lt.x, (int)br.y - (int)lt.y
7         ));
8     Mat selectedArea = img.submat((int)lt.y, (int)br.y, (int)lt.x, (int)br.x);
9     card.copyTo(selectedArea);
}
```

6.6. Implementacja w aplikacji

W naszej aplikacji działania związane z wykonaniem zdjęcia oraz jego przetwarzania wykonują się z poziomu osobnej Aktywności.



(a) po uruchomieniu (b) po wykonaniu zdjęcia (c) po detekcji czoła (d) po detekcji karty

Rys. 6.5. Widoki aktywności obsługującej zdjęcie

Rysunek 6.5 przedstawia widok Aktywności w której realizowane są operacje związane

ze zdjęciem. Widok aktywności składa się z trzech przycisków oraz okienka z obrazkiem. Po wcisnięciu przycisku z napisem ZDJĘCIE zostaje uruchomiona systemowa aplikacja aparatu, która po wykonaniu zdjęcia prosi o zatwierdzenie wykonanego zdjęcia. Jeśli zaakceptowano zdjęcie następuje automatyczny powrót do naszej aplikacji. Po stronie naszej aplikacji kod wygląda następująco:

```

1 public void takePic(){
2     pictureFile = getPictureFile();
3     Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
4     intent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(getPictureFile()));
5     //pathPic = dest.getPath();           ((Activity)mainContext).startActivityForResult(
6         intent, REQUEST_IMAGE);
7 }
8 @Override
9 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
10     if( requestCode == pictureService.REQUEST_IMAGE && resultCode == Activity.RESULT_OK
11     ){
12         pictureService.updateImage();
13     }
14 }
15 }
```

Po wykonaniu zdjęcia w okienku pojawia się wykonane zdjęcie tak jak na rysunku 6.5b. Następnie w zależności od wariantu sztuczki można wybrać przycisk OZNACZ CZOŁO albo OZNACZ KARTĘ. W przypadku pierwszego uruchamiamy następującą funkcję, która naniesie obraz karty w miejsce odnalezienia czoła:

```

1 public Mat picToMark(Mat aInputFrame) throws IOException {
2     Imgproc.cvtColor(aInputFrame, grayscaleImage, Imgproc.COLOR_RGBA2RGB);
3     ObjectReconizer fR=new ObjectReconizer(mainContext, R.raw.lbpcascade_frontalface,
4         mainContext.getString(R.string.cascadeFrontalFaceXML), grayscaleImage, 1.3, 3);
5     MatOfRect faces=fR.getObjects();
6     org.opencv.core.Rect[] facesArray = faces.toArray();
7     if(facesArray.length>0) {
8         ObjectReconizer eR=new ObjectReconizer(mainContext, R.raw.
9             haarcascade_lefteye_2splits, "haarcascade_lefteye_2splits.xml",
10            grayscaleImage, 1.3, 3);
11         for (int i = 0; i < facesArray.length; i++) {
12             org.opencv.core.Rect face=facesArray[i];
13             org.opencv.core.Rect abcd =new org.opencv.core.Rect(face.x, face.y, face.
14                 width, face.height);
15             Mat roi_color= new Mat(aInputFrame, abcd);
16             eR.setGrayScaleImage(roi_color);
17             MatOfRect eyes=eR.getObjects();
18             org.opencv.core.Rect[] eyesArray=eyes.toArray();
19             if(eyesArray.length==2){
20                 int x=(eyesArray[0].x<eyesArray[1].x) ? eyesArray[0].x:eyesArray[1].x;
21                 int x2=(eyesArray[0].x>eyesArray[1].x) ? eyesArray[0].x+eyesArray[0].
22                     width:eyesArray[1].x+eyesArray[1].width;
23                 int y = eyesArray[0].y;
24                 int w=x2-x;
25                 int h=w/2;
26                 insertCardIntoImage(roi_color, new Point(0, x), new Point(h, w+x));
27             }
28         }
29     }
30     return aInputFrame;
31 }
```

Po wykonaniu tej operacji w miejscu odnalezienia czoła zostanie wklejony obraz karty, tak jak na rysunku 6.5c. Jeśli wybrany jest wariant by karta została wklejona w miejsce naszego znaku zapytania wybieramy przycisk OZNACZ KARTĘ w tym wypadku uruchomi się następująca funkcja:

```

1 public void markCard() throws IOException {
2     if(grayscaleImage==null)grayscaleImage=new Mat();
3     Mat pictureToMark= Imgcodecs.imread(pictureFile.getPath());
4     double width=pictureToMark.size().width;
```

```

5   double height=pictureToMark.size().height;
6   MatOfRect cards = new MatOfRect();
7   while(width>120.0){
8       Imgproc.cvtColor(pictureToMark, grayscaleImage, Imgproc.COLOR_RGBA2RGB);
9       ObjectReconizer cR=new ObjectReconizer(mainContext, R.raw.newcard, "newcard.xml"
10          , grayscaleImage, 1.4, 25);
11      cards=cR.getObjects();
12      if(cards.toArray().length!=1{
13          width/=1.5;
14          height/=1.5;
15          Imgproc.resize(pictureToMark, pictureToMark, new Size(width,height ));
16      }
17      else break;
18  }
19  org.opencv.core.Rect[] cardsArray = cards.toArray();
20  if(cardsArray.length==1){
21      Mat picToMarkAfterResize = Imgcodecs.imread(pictureFile.getPath());
22      double ratio= picToMarkAfterResize.size().width/width;
23      Point lt=new Point(cardsArray[0].tl().x*ratio, cardsArray[0].tl().y*ratio);
24      Point br=new Point(cardsArray[0].br().x*ratio, cardsArray[0].br().y*ratio);
25      insertCardIntoImage(picToMarkAfterResize, lt, br);
26      pictureToMark=picToMarkAfterResize;
27  }
28  pictureFile = getPictureFile();
29  Imgcodecs.imwrite(pictureFile.getPath(), pictureToMark);
30  updateImage();
}

```

Po wykonaniu tej operacji karta została wklejona w miejsce odnalezionej obiektu. Rysunek 6.5d przedstawia widok po wykonaniu detekcji karty po wcześniejszej detekcji czoła.

7. TECHNOLOGIA NFC (KONRAD OSSOWSKI)

NFC (ang. Near Field Communication) to w dosłownym tłumaczeniu komunikacja bliskiego zasięgu. Polega ona na bezprzewodowej technologii krótkiego zasięgu. Tag NFC to pasywne urządzenie NFC, zasilane polem NFC urządzenia, gdy znajduje się ono w zasięgu. Wyróżnia się cztery rodzaje tagów, różnią się od siebie pojemnością i/lub formatem. Zazwyczaj wymagają odległości 4cm lub mniejszej, aby zainicjować połączenie. Tagi mogą mieć różną złożoność. Proste znaczniki oferują tylko możliwość odczytu i zapisu, czasem z programowalnym obszarem informującym, że zawartość jest tylko do odczytu. Bardziej skomplikowane tagi oferują operacje matematyczne i mają sprzęt kryptograficzny do uwierzytelniania. Najbardziej wyrafinowane tagi zawierają środowisko operacyjne, pozwalające na złożone interakcje z kodem wykonującym na tagu.

Zasada działania

NFC opiera się na efekcie indukcji elektromagnetycznej. Gdy znacznik znajdzie się obrębie pola magnetycznego, generowana jest energia elektryczna. Sygnał przepływa między dwiema antenami – w smartfonie i w sprzęcie docelowym (również może to być smartfon). W ten sposób zestabilizowane zostaje połączenie zapewniające komunikację między urządzeniami za pomocą fal radiowych krótkiego zasięgu.

7.1. Historia NFC

Pierwsze próby z tą technologią rozpoczęto jeszcze na początku lat 80. W 1983 r. złożono wówczas pierwszy patent związanego z technologią RFID (ang. Radio-frequency identification), która jest inspiracją dla NFC. Dopiero w 2004 roku została utworzona organizacja mająca na celu rozwój technologii NFC - NFC Forum.

Trzeba było kolejnych lat, by dopiero w 2009 r. zostały opracowane kolejne elementy układanki, która pozwoliła na wygodne korzystanie z NFC. Dopiero siedem lat temu zaczęły działać pierwsze standardy przesyłania zarówno kontaktów czy adresów URL. W tym samym czasie stworzono również sposób na inicjalizację nadajnika Bluetooth. Dzięki temu nie trzeba korzystać z niezbyt wygodnego parowania urządzeń. Zamiast tego wystarczy zbliżyć do siebie czytnik (smartfon) z kartą NFC (głośnik bezprzewodowy) by nawiązać między nimi łączność Bluetooth.

7.2. NFC na platformie Android

Funkcja NFC umożliwia udostępnianie niewielkich ładunków danych między tagiem NFC, a urządzeniem z systemem Android lub między dwoma urządzeniami. Dane przechowywane w tagu mogą być również zapisane w różnych formatach, ale wiele interfejsów API systemu Android jest opartych na standardzie NFC Forum o nazwie Ndef. Urządzenia z systemem Android z obsługą NFC realizują jednocześnie trzy główne tryby działania:

- Tryb czytnika/zapisywacza - umożliwia urządzeniu NFC odczytywanie i/lub zapisywanie pasywnych tagów NFC i naklejek.

- Tryb peer-to-peer - umożliwia urządzeniu NFC wymianę danych z innym urządzeniem NFC. Korzysta z niego funkcja Android Beam.
- Tryb emulacji kart - umożliwia urządzeniu NFC działać jak karta NFC. Dostęp do emulowanej karty NFC można uzyskać za pomocą zewnętrznego czytnika np. terminalu płatniczego NFC.

Funkcja Android Beam umożliwia urządzeniu przesyłanie wiadomości NDEF na inne urządzenie poprzez fizyczne dotykanie urządzeń. Ta interakcja zapewnia łatwiejszy sposób wysyłania danych niż inne technologie bezprzewodowe, takie jak Bluetooth, ponieważ w przypadku NFC nie jest wymagane ręczne wykrywanie urządzeń ani parowanie. Połączenie zostanie automatycznie uruchomione, gdy dwa urządzenia znajdą się w zasięgu.

7.2.1. *NfcAdapter*

Reprezentuje lokalny moduł NFC. Posiada trzy stałe określające akcję dla Intencji:

- *ACTION_TAG_DISCOVERED* - dowolny tag NFC został wykryty
- *ACTION_NDEF_DISCOVERED* - został wykryty tag z danymi NDEF
- *ACTION_TECH_DISCOVERED* - został wykryty tag, którego technologia została zarejestrowana w jakieś Aktywności

Tworzenie nowego obiektu zawierającego strukturę *NfcAdapter* sprowadza się do wywołania statycznej metody tej klasy *getOrDefaultAdapter(context)*. Poniżej przedstawiony jest konstruktor zaimplementowanej przez nas klasy *NfcService*.

```
1 public NfcService(Context context){
2     this.context = context;
3     this.nfcAdapter = NfcAdapter.getDefaultAdapter(context);
4 }
```

W parametrze *context* przekazywany jest obiekt Aktywności, która korzysta z NFC. Dodatkowo, aby nasza Aktywność miała pierwszeństwo do obsłużenia wykrytego tagu, gdy jest na pierwszym planie, musimy nadpisać jej metodę *onResume()*. W ciele tej metody musi występować wywołanie funkcji klasy *NfcAdapter enableForegroundDispatch (Activity activity, PendingIntent intent, IntentFilter[] filters, String[][] techLists)*. Jeśli został podany parametr *filters*, jest ona używana do dopasowania Intencji wysyłanych zarówno dla *ACTION_NDEF_DISCOVERED*, jak i *ACTION_TAG_DISCOVERED*. Ponieważ *ACTION_TECH_DISCOVERED* opiera się na metadanych zawartych w tagu, jego dopasowanie jest obsługiwane przez osobną listę zarejestrowanych technologii w parametrze *techLists*. Jeśli parametry *filters* i *techList* będą *null* to wszystkie wykryte znaczniki zostaną przekazane do Aktywności na pierwszym planie. W naszej klasie *NfcService*, zostało to zrealizowane w postaci publicznej metody, która może zostać wywołana z poziomu Aktywności, gdzie został utworzony obiekt klasy *NfcService*.

```
1 public void enableForegroundDispatchSystem(){
2     Intent intent = new Intent(context, context.getClass());
3     intent.addFlags(Intent.FLAG_RECEIVER_REPLACE_PENDING);
4     Intent[] intentArray = new Intent[]{intent};
5     PendingIntent pending = PendingIntent.getActivity(context, 0, intentArray, 0);
6     IntentFilter[] filters = new IntentFilter[] {};
7     nfcAdapter.enableForegroundDispatch((Activity)context, pending, filters, null);
8 }
```

Gdy Aktywność obługująca wykryte tagi NFC zniknie z pierwszego planu, należy niezwłocznie wyłączyć jej tę możliwość. Najlepiej zrobić to w nadpisanej metodzie `onPause()` Aktywności korzystającej z NFC.

```
1 public void disableForegroundDispatchSystem(){
2     nfcAdapter.disableForegroundDispatch((Activity)context);
3 }
```

7.2.2. Znacznik

Znacznik (ang. Tag) jest to niezmienny obiekt reprezentujący stan znacznika NFC w momencie wykrycia. Może być używany jako uchwyt, aby wykonywać bardziej zaawansowane operacje. Nowy obiekt Znacznika jest tworzony za każdym razem, gdy moduł NFC go wykryje. Nie ma znaczenia czy jest to fizycznie ten sam znacznik. Uzyskanie obiektu Znacznika jest możliwe poprzez wyciągnięcie go z obiektu Intencji przekazanego do Aktywności pierwszego planu. Dlatego tak ważne jest uruchamianie metody `enableForegroundDispatch()`, w metodzie `onResume()` Aktywności.

```
1 public boolean setNfcTag(Intent intent) {
2     if(intent.hasExtra(NfcAdapter.EXTRA_TAG)){
3         tag = getTag(intent);
4         return true;
5     }
6     return false;
7 }
8 private Tag getTag(Intent intent) {
9     return intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
10 }
```

Metoda `setNfcTag()` jest wywoływana w nadpisanej metodzie `onNewIntent()` Aktywności, która obsługuje żądanie NFC.

7.2.3. Zapis i odczyt danych z użyciem tagu NFC

Ndef (ang. NFC Data Exchange Format) to lekki format binarny, używany do enkapsulacji wpisanych danych. Jest określony przez NFC Forum do transmisji i przechowywania danych za pomocą NFC. Obiekt tej klasy uzyskuje się za pomocą statycznej metody klasy Ndef `get(Tag tag)`. Zawiera w sobie Wiadomości (ang. NdefMessages) i Rekordy (ang. NdefRecords). Rekordy zawierają w sobie dane, może to być nośnik MIME, URI lub niestandardowe dane aplikacji. Natomiast Wiadomości są zbiorem Rekordów.

Zapis

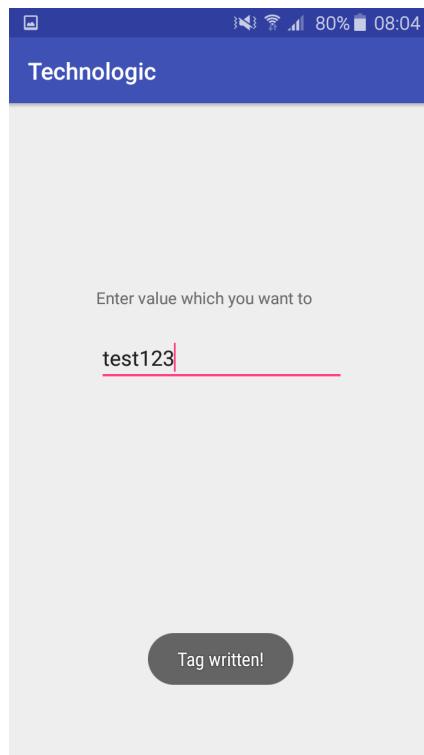
Zapis danych do tagu polega na poprawnym przygotowaniu Rekordu Ndef. Stworzeniu na jego podstawie Wiadomości Ndef, a następnie nadpisaniu poprzedniej Wiadomości Ndef w znaczniku (o ile znacznik nie jest tylko do odczytu).

```
1 public boolean writeToNfcTag(String content){
2     return writeNdefMessage(content);
3 }
4
5 private NdefMessage createNdefMessage(String content){
6     NdefRecord record = NdefRecord.createTextRecord(null, content);
7     return new NdefMessage(record);
8 }
```

```

9 private boolean writeNdefMessage( String content){
10    try{
11        if(tag == null){
12            return false;
13        }
14        else{
15            Ndef ndef = Ndef.get(tag);
16            NdefMessage mess = createNdefMessage(content);
17            if(ndef == null){
18                return false;
19            }
20            else{
21                ndef.connect();
22                if(!ndef.isWritable()){
23                    ndef.close();
24                    return false;
25                }
26                ndef.writeNdefMessage(mess);
27                ndef.close();
28                return true;
29            }
30        }
31    }
32    catch(Exception e){
33        return false;
34    }
35 }
36 }
```

Efektem graficznym tego fragmentu jest następująca Aktywność. Wyświetlony Toast został wywołany po zbliżeniu znacznika do smartfonu i poprawnym zapisie danych do niego.



Rys. 7.1. Aktywność do zapisywania danych do tagu NFC

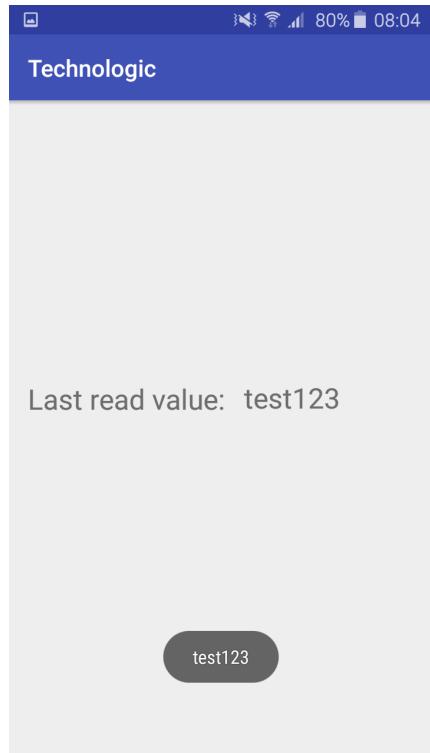
Odczyt

Odczyt danych Ndef z tagu NFC jest obsługiwany za pomocą systemu rozsyłania tagów, który analizuje wykryte znaczniki NFC, odpowiednio kategoryzuje dane i uruchamia aplikację, która

jest zainteresowana skategoryzowanymi danymi. Aplikacja, która chce obsłużyć zeskanowany tag NFC, może zadeklarować filtr intencji i żądanie obsługi danych.

```
1 public String readFromNfcTag(){
2     return readTextFromTag();
3 }
4
5 private String readTextFromTag() {
6     Ndef ndef = Ndef.get(tag);
7     if(ndef == null) return null;
8     NdefMessage ndefMessage;
9     try {
10         ndef.connect();
11         ndefMessage = ndef.getNdefMessage();
12     }
13     catch(Exception ex){
14         return null;
15     }
16     NdefRecord[] records = ndefMessage.getRecords();
17     if(records != null && records.length > 0){
18         NdefRecord record = records[0];
19         return getTextFromNdefRecord(record);
20     }
21     return null;
22 }
23
24 private String getTextFromNdefRecord(NdefRecord record) {
25     String tagContent = null;
26     try{
27         byte[] payload = record.getPayload();
28         String textEncoding = "UTF-8";
29         int languageSize = payload[0] & 0063;
30         tagContent = new String(payload, languageSize + 1,
31             payload.length - languageSize - 1, textEncoding);
32
33     } catch (UnsupportedEncodingException e) {
34         e.printStackTrace();
35     }
36     return tagContent;
37 }
```

Efekt graficzny tego fragmentu przedstawia się następująco. Jest to sytuacja, gdy zbliżony tag został poprawnie odczytany. Jednocześnie, jeśli był dla danego tagu (karty) określony schemat wibracji, zostaje on wywibrowany.



Rys. 7.2. Aktywność do odczytywania danych do tagu NFC

8. PODSUMOWANIE (MACIEJ PLEWKA)

Zrealizowany projekt miał na celu opracowania aplikacji mobilnej, umożliwiającej wykonanie sztuczki karcianej. Wynikiem pracy jest działający program możliwy do uruchomienia na telefonach z systemem Android w wersji 5.0+. W realizacji celu wykorzystano wbudowane komponenty telefonu takie jak aparat, wibrator oraz moduł NFC. Dodatkowo wytworzono własne kaskady klasyfikatorów zdolnych do rozpoznawania obiektów. Zrealizowano obsługę aparatu z poziomu aplikacji, możliwość odczytu i zapisu wiadomości do znaczników NFC, wykorzystanie wibratora do wibrowania w zdefiniowanych przez nas odstępach czasu, detekcję obiektów na wykonanym zdjęciu oraz wytrenowaną własną kaskadę Haar'a. Dodatkowo można wskazać kilka funkcji, które usprawniłyby naszą aplikację lub rozszerzyły by jej funkcjonalność:

- Zastosowanie innych sposobów detekcji, np. przy użyciu splotowych sieci neuronowych.
- Umożliwienie wysyłania wykonanego zdjęcia przy użyciu poczty elektronicznej
- Zastosowanie wstępnej detekcji w czasie rzeczywistym podczas wykonywania zdjęcia.
- Dodanie samouczka, który wyjaśniał by w jaki sposób używać aplikacji
- Umieszczenie aplikacji w Sklepie Play

WYKAZ LITERATURY

1. Autor Artykułu 2 Autor Artykułu 1. "Fajny albo i nie artykuł". W: *WETI PG* (2014).
2. <https://developer.android.com/guide/topics/manifest/manifest-intro.html>.
3. <https://developer.android.com/guide/topics/resources/accessing-resources.html>.
4. <https://developer.android.com/guide/topics/resources/providing-resources.html>.
5. <https://developer.android.com/reference/android/app/Activity.html>.
6. <https://developer.android.com/reference/android/app/Service.html>.
7. <https://developer.android.com/reference/android/content/Intent.html>.
8. <https://developer.android.com/reference/android/os/Vibrator.html>.
9. <https://developer.android.com/reference/android/support/constraint/ConstraintLayout.html>.
10. <https://developer.android.com/reference/android/view/View.html>.
11. <https://developer.android.com/reference/android/widget/Adapter.html>.
12. <https://developer.android.com/reference/android/widget/AdapterView.html>.
13. <https://developer.android.com/reference/android/widget/Button.html>.
14. <https://developer.android.com/reference/android/widget/CheckBox.html>.
15. <https://developer.android.com/reference/android/widget/EditText.html>.
16. <https://developer.android.com/reference/android/widget/ExpandableListView.html>.
17. <https://developer.android.com/reference/android/widget/Spinner.html>.
18. <https://developer.android.com/reference/android/widget/TextView.html>.
19. <https://developer.android.com/reference/android/widget/Toast.html>.
20. https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html.
21. https://en.wikipedia.org/wiki/Summed-area_table.
22. <https://github.com/opencv/opencv>.
23. <https://github.com/wulfebw/mergevec>.
24. <https://nfc-forum.org/our-work/specifications-and-application-documents/>.
25. https://www.cse.buffalo.edu/~jcorso/t/CSE555/files/lecture_boosting.pdf.
26. https://www.researchgate.net/publication/262936238_Obscenity_Detection_Using_Haar-Like_Features_and_Generalization.
27. <http://www.image-net.org/>.
28. Jakis ziomek w internetach. *Przykładowy artykuł online*. Dostęp 8.12.2016. 2016. URL: <https://pl.wikipedia.org/wiki/BibTeX>.
29. Michael Jones Paul Viola. "Rapid Object Detection using a Boosted Cascade of Simple Features". W: *Cambridge* (2001).

WYKAZ RYSUNKÓW

1.1.	Panel konfiguracyjny aplikacji	8
2.1.	Cykl życia aktywności[5]	18
2.2.	Przykład relatywnego pozycjonowania[9]	19
3.1.	Cechy Haara Źródło: [20].....	23
3.2.	Obliczanie sumy wartości pikseli Źródło: [26]	25
3.3.	Działanie mocnego klasyfikatora Źródło: [25]	26
4.1.	Wygenerowane pozytywne zdjęcie.....	30
4.2.	Cechy mocnego klasyfikatora	32
5.1.	Rewersy kart użyte do treningu	35
5.2.	Wzory własnej karty użyte do treningu	35
5.3.	Cechy opisujące rewers karty	37
5.4.	Cechy opisujące naszą kartę.....	37
6.1.	Detekcja twarzy dla różnych wartości scaleFactor	39
6.2.	Detekcja karty dla różnych wartości scaleFactor.....	39
6.3.	Detekcja twarzy dla różnych wartości minNeighbors.....	40
6.4.	Detekcja karty dla różnych wartości minNeighbors	40
6.5.	Widoki aktywności obsługującej zdjęcie	42
7.1.	Aktywność do zapisywania danych do tagu NFC	48
7.2.	Aktywność do odczytywania danych do tagu NFC	50

WYKAZ TABEL

6.1.	Wyniki detekcji w zależności od parametru scaleFactor.....	39
6.2.	Wyniki detekcji w zależności od parametru minNeighbors	40
6.3.	Poszukiwanie najlepszej wartości minNeighbors cz. 1	41
6.4.	Poszukiwanie najlepszej wartości minNeighbors cz. 2	41
6.5.	Poszukiwanie najlepszej wartości scaleFactor	41