POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY
INSTYTUT STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ

PRACA DYPLOMOWA INŻYNIERSKA
na kierunku ELEKTROTECHNIKA
specjalność: AUTOMATYKA I INŻYNIERIA KOMPUTEROWA

Dariusz PŁASZEWSKI
Nr albumu: 230250

Rok akad.: 2012/2013
Warszawa, 10.09.2012 r.

PRZETWARZANIE OBRAZÓW NA URZĄDZENIACH MOBILNYCH Z WYKORZYSTANIEM BIBLIOTEKI OPENCV

Zakres pracy:

1. *Wprowadzenie i sformułowanie celu pracy*
2. *Wstęp do przetwarzania obrazów na urządzeniach mobilnych*
3. *Charakterystyka biblioteki OpenCV*
4. *Analiza porównawcza działania biblioteki OpenCV na urządzeniach mobilnych*
5. *Podsumowanie i wnioski*

Kierujący pracą: dr inż. Witold Czajewski
Konsultant:

KIEROWNIK ZAKŁADU STEROWANIA
Dr hab. Inż. Bartłomiej Beliczyński

Termin złożenia pracy: 28.01.2013 r.
Praca wykonana i obroniona pozostaje własnością Instytutu, Katedry i nie będzie zwrócona wykonawcy.

Przetwarzanie obrazów na urządzeniach mobilnych z wykorzystaniem biblioteki OpenCV

Streszczenie

Niniejsza praca inżynierska miała na celu demonstrację działania oraz możliwości biblioteki OpenCV na urządzeniach mobilnych. Platformą testową wykorzystaną w pracy był system Android. Rozdział pierwszy stanowi wprowadzenie w tematykę pracy, zawarto tutaj podstawowe informacje o współczesnych urządzeniach mobilnych oraz o wykorzystaniu przetwarzania obrazów w celu rozwinięcia możliwości takiego sprzętu.

Wykonanie pracy wymagało zapoznania się z biblioteką OpenCV oraz ze środowiskiem programistycznym Eclipse. Przed realizacją algorytmów ważne było poszerzenie wiedzy o obszernym rynku urządzeń mobilnych oraz o zasadach działania operacji przetwarzania obrazów testowanych w pracy.. Rozdziały drugi i trzeci poświęcono przedstawieniu podstawowych pojęć przetwarzania obrazów, współczesnego rynku urządzeń mobilnych oraz opisowi biblioteki OpenCV.

Napisanie pracy wymagało przygotowania trzech programów mobilnych. Pierwszy z nich testuje wydajność działania dwóch wybranych algorytmów przetwarzania obrazów: filtracji medianowej oraz progowania adaptacyjnego. Aplikacja ta została napisana z wykorzystaniem języka Java. Drugi program bada szybkość działania algorytmu detekcji twarzy a trzeci sprawdza wydajność funkcji detekcji cech. Dwie powyższe aplikacje zostały przygotowane z wykorzystaniem programowania natywnego w języku C++. Głównym źródłem danych na temat wydajności każdej przygotowanej aplikacji jest wbudowany miernik liczby klatek na sekundę.

Wykonane badania pokazały, że w obecnym stadium rozwoju biblioteka OpenCV oferuje bardzo dobre wsparcie dla systemu Android i może być z powodzeniem wykorzystywana przy budowie zaawansowanych aplikacji mobilnych. Wykazano również, że szybkość działania aplikacji opartych o bibliotekę OpenCV jest zależna od parametrów technicznych wykorzystywanego urządzenia mobilnego. Ostatni rozdział zawiera podsumowanie i wnioski z wykonanej pracy inżynierskiej. Dodatkowo przedstawiono tu propozycje dalszej rozbudowy aplikacji testowych.

Image processing on mobile devices using the OpenCV library

Abstract

The aim of this BSc thesis was to demonstrate the activity and the capabilities of the OpenCV library on mobile devices. Android operating system was being used as a test platform for this BSc thesis. The first chapter is an introduction to the topic. It contains the basic information about modern mobile devices and the usage of image processing to extend the capabilities of such equipment.

During the work it was required to become familiar with the OpenCV library and the Eclipse programming environment. Before the implementation of the algorithms, it was necessary to extend the knowledge about the vast mobile devices market and about the principles of image processing operations, which are being tested in this thesis. The second and the third chapter are devoted to introduction of the basics of image processing and the description of the OpenCV library.

The preparation of three mobile applications was required for the purposes of the thesis. First of them tests the performance of two chosen image processing algorithms: a median filter and an adaptive thresholding. This program was written with a Java programming language. The second one examines the speed of the face detection algorithm and the third one checks the performance of the feature detection function. The last two of the aforementioned programs are written with the help of the native programming in C++ language. The main source of the data related to performance is a built – in frames per second meter.

The examinations have shown that the current released version of the OpenCV library offers very good technical support for Android operating system and it can be successfully used in many advanced mobile applications. It was also proved, that the speed of the programs which are based on the OpenCV library depends on the technical specifications of the used mobile device. The last chapter contains a summary and conclusions from the thesis.

Warszawa, dnia 6 lutego 2013 roku.

Politechnika Warszawska
Wydział Elektryczny

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa inżynierska pt. Przetwarzanie obrazów na urządzeniach mobilnych z wykorzystaniem biblioteki OpenCV:

- została napisana przeze mnie/nas samodzielnie
- nie narusza niczyich praw autorskich
- nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam, że przedłożona do obrony praca dyplomowa nie była wcześniej podstawą postępowania związanego z uzyskaniem dyplomu lub tytułu zawodowego w uczelni wyższej.

Jestem świadom, że praca zawiera również rezultaty stanowiące własności intelektualne Politechniki Warszawskiej, które nie mogą być udostępniane innym osobom i instytucjom bez zgody Władz Wydziału Elektrycznego.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Imię i Nazwisko dyplomanta: Dariusz Płaszewski

Podpis dyplomanta: Dariusz Płaszewski

Spis treści

Rozdział 1

Wprowadzenie

W dzisiejszych czasach można zaobserwować znaczny wzrost popularności urządzeń mobilnych. Wyróżniamy wśród nich dwa najczęściej spotykane typy: smartfony oraz tablety. Smartfon jest nazwą urządzenia zawierającego wyświetlacz o rozmiarze maksymalnie 5 cali oraz umożliwiającego użytkownikowi wykonywanie połączeń oraz przesyłanie wiadomości w ramach sieci komórkowej. Tablet jest stosunkowo świeżym rodzajem urządzeń mobilnych. Określenie to dotyczy sprzętu o wymiarach większych od smartfonu, a także posiadającego ekran o rozmiarach od 7 do 10 cali. Takie urządzenia pozwalają wykonywać większość czynności dostępnych wcześniej tylko dla użytkowników klasycznych komputerów typu PC. Ich dużą zaletą jest mobilność oraz łatwość korzystania z nich w miejscach publicznych. Zapewniają swoim właścicielom ciągły dostęp do Internetu, możliwość bezpośredniej komunikacji ze znajomymi, rozrywkę oraz wiele innych udogodnień niespotykanych w prostych telefonach komórkowych.

Rozwój techniki powoduje, że omawiane urządzenia mobilne wyposażane są w coraz bardziej nowoczesne podzespoły. Widoczny jest postęp w jednostkach obliczeniowych, które charakteryzują się dużą wydajnością oraz lepszymi parametrami wbudowanych układów graficznych. W swoich produktach mobilnych producenci umieszczają również wyświetlacze wykonywane w coraz lepszych technologiach, cechujące się doskonałym odwzorowaniem kolorów. Postęp techniczny dokonał się dodatkowo w aparatach fotograficznych montowanych w urządzeniach mobilnych. Dostępne konstrukcje odznaczają się funkcjami dostępnymi dotąd tylko dla zaawansowanych klasycznych aparatów cyfrowych.

Wysoka wydajność współczesnych urządzeń sprawia, że mogą one znaleźć zastosowanie w wielu operacjach, do których wykorzystywane są komputery oraz profesjonalne aparaty cyfrowe. Smartfony oraz tablety umożliwiają edycję zdjęć i filmów bezpośrednio w urządzeniu za pomocą odpowiednich aplikacji. Pozwalają na oglądanie filmów w wysokiej rozdzielczości oraz słuchanie muzyki w wysokiej jakości dźwięku. Wbudowane aparaty fotograficzne mają możliwość robienia dokładnych zdjęć [12], a specjalne algorytmy zajmują się usuwaniem niedoskonałości powstałych w procesie akwizycji zdjęcia.

Oprócz wymienionych zastosowań, urządzenia mobilne można wykorzystywać także do zaawansowanych obliczeń przetwarzania obrazów. Istnieje wiele różnych sposobów aby tego dokonać. Procesy te są użyteczne w zaawansowanych i wydajnych algorytmach uzupełniających i rozbudowujących mobilne aplikacje do obróbki zdjęć i obrazów. Przetwarzanie obrazów jest również przydatne przy aplikacjach umożliwiających rozpoznawanie tekstu. Stosowanie tej metody umożliwiłoby powstanie ulepszonych, dokładnych aplikacji OCR dla urządzeń mobilnych. Aparaty obecne we współczesnych smartfonach i tabletach charakteryzują się dobrymi parametrami technicznymi, co skutkuje powstawaniem wystarczająco szczegółowych zdjęć – dzięki połączeniu ich z technologią OCR, smartfony lub tablety mogą pełnić rolę poręcznych, przenośnych skanerów [4]. Innym zastosowaniem przetwarzania obrazów na urządzeniach mobilnych są różnego rodzaju aplikacje mobilne służące do katalogowania produktów w sklepach oraz porównywania ich cen. Przykładem może tu być aplikacja, którą klient instaluje na swoim urządzeniu mobilnym i używa jej do automatycznego porównania ceny wybranego produktu w odwiedzonym sklepie i ceny w innych sklepach. Interesującym zastosowaniem przetwarzania obrazów na urządzeniach mobilnych mogłoby być wykrywanie uszkodzeń rozmaitych urządzeń, przedmiotów lub produktów spożywczych. Użytkownik smartfonu lub tabletu miałby możliwość łatwego znalezienia uszkodzeń w dowolnym produkcie, np. przed dokonaniem zakupu w sklepie. Najważniejszym składnikiem wykorzystywanym przy programowaniu aplikacji wykonujących przetwarzanie obrazów jest biblioteka OpenCV. Stosowanie tej biblioteki na urządzeniach mobilnych ma na celu umożliwienie tworzenia aplikacji przetwarzania obrazów i wizji komputerowej na takich samych zasadach na jakich są one przygotowywane dla urządzeń stacjonarnych typu komputery PC.

1.1. Cel i układ pracy

Celem pracy jest sprawdzenie działania biblioteki OpenCV na kilku urządzeniach mobilnych różniących się wydajnością oraz należących do różnych segmentów cenowych. Sprawdzona zostanie również praca biblioteki na dwóch różnych wersjach systemu operacyjnego, pracujących na identycznych urządzeniach. Testy działania biblioteki będą przeprowadzane za pomocą aplikacji mobilnych wykonujących wybrane funkcje dostępne w bibliotece OpenCV.

W ramach pracy zostanie dokonana analiza zastosowania biblioteki OpenCV w konkretnych scenariuszach odpowiadających wykorzystanym aplikacjom. Poruszone zostanie również zagadnienie zastosowania biblioteki OpenCV na urządzeniach mobilnych w

codziennym życiu. Realizacja części praktycznej pracy dotyczy przygotowania aplikacji dla posiadanych urządzeń mobilnych, których celem byłoby wykonanie określonych funkcji z biblioteki OpenCV oraz przeprowadzenie pomiarów określających prędkość działania tych aplikacji na danym urządzeniu.

Cała praca wymaga osiągnięcia poniższych celów:

1. Przygotowanie środowiska pracy: pobranie niezbędnego oprogramowania oraz archiwum z biblioteką, dostosowanie składników pakietu Android SDK oraz środowiska Eclipse. Sprawdzenie komunikacji urządzeń mobilnych z komputerem
2. Utworzenie 3 aplikacji mobilnych, wykonujących wybrane, wymagające obliczeniowo funkcje z biblioteki OpenCV. Operacje stosowane w tych aplikacjach mają realne zastosowanie w sprzęcie przenośnym. Wbudowanie w aplikacje 2 różnych algorytmów, których celem jest zbadanie wydajności urządzenia podczas obliczeń związanych z przetwarzaniem obrazów i wizją komputerową
3. Demonstracja działania i możliwości biblioteki OpenCV na urządzeniach mobilnych
4. Zebranie i prezentacja wyników pomiarów wydajności wykorzystywanych urządzeń podczas przetwarzania przygotowanych aplikacji
5. Wyciagnięcie wniosków z otrzymanych wyników oraz próba odpowiedzenia na pytanie: czy przeprowadzanie obliczeń związanych z przetwarzaniem obrazów i widzeniem komputerowym ma zastosowanie na urządzeniach mobilnych, czy lepiej jest wykonywać wszystkie obliczenia na serwerach i wysyłać przetworzone dane na urządzenie?

Nie wliczając wstępu, praca podzielona jest na cztery rozdziały. Rozdział pierwszy zawiera niezbędne informacje wprowadzające w tematykę pracy. Przedstawia podstawowe zagadnienia przetwarzania obrazów oraz prezentuje rynek współczesnych urządzeń mobilnych oraz jego rozwój. Na koniec treść tego rozdziału prezentuje możliwe zastosowania przetwarzania obrazów w urządzeniach mobilnych.

Rozdział drugi dotyczy biblioteki OpenCV, pokazuje czym ona jest oraz prezentuje jej budowę, a także opisuje OpenCV w wersjach przeznaczonych do działania na urządzeniach mobilnych oraz jej wymagania do pracy w systemie Android. Następnie rozdział ten wyjaśnia, czym jest OpenCV Manager oraz demonstruje sposób postępowania przy instalacji i przygotowywaniu biblioteki OpenCV dla systemu Android. Na koniec zawarto informacje o programowaniu natywnym OpenCV dla systemu Android z wykorzystaniem języka C++.

Rozdział trzeci to opis aplikacji napisanych na potrzeby pracy inżynierskiej. Na początku prezentuje urządzenia mobilne, które zostały wykorzystane do pomiarów wydajności działania przygotowanych aplikacji. Następnie rozdział ten przedstawia każdą aplikację oraz prezentuje wyniki pomiarów szybkości ich działania.

Ostatni rozdział zawiera wnioski z przeprowadzonej analizy oraz podsumowuje wykonane pomiary.

Rozdział 2

Przetwarzanie obrazów na urządzeniach mobilnych

2.1. Podstawowe pojęcia z przetwarzania obrazów

Przetwarzanie obrazów jest dziedziną zajmującą się wyodrębnianiem pewnych cech obrazów, które pozwalają w miarę dokładnie określić coś na temat zawartości tych obrazów. Może to być np. rozjaśnienie obrazu, usunięcie szumu, detekcja obiektów na obrazach. Przetwarzanie obrazów znalazło zastosowanie w wielu obszarach techniki. Wykorzystywane jest między innymi w robotyce, gdzie jest narzędziem ważnym przy sterowaniu robotów a także przy nawigacji automatycznej. Innym ważnym zastosowaniem przetwarzania obrazów jest medycyna, gdzie odgrywa ono swoją rolę między innymi w diagnostyce obrazowej wspomagając proces tworzenia obrazów badanych narządów. Pozostałe warte uwagi zastosowania przetwarzania obrazów to przemysł, informatyka, oceanografia i geodezja. Można wymienić następujące obszary przetwarzania obrazów: filtracja, wzbogacanie, rozpoznawanie, kompresja, widzenie robotów i analiza scen wizyjnych.

Każdy system widzenia komputerowego wykonuje obróbkę obrazów w kilku etapach. Wśród nich możemy wyróżnić:

1. **Akwizycja**. Etap ten odpowiada za przetworzenie obrazu widzianego przez kamerę na postać cyfrową w formie reprezentującej obraz cyfrowy – macierzy pikseli obrazu.
2. **Filtracja**. Etap ten ma za zadanie poprawę jakości obrazu pod kątem jego dalszej obróbki. Poprawa jakości obrazu odbywa się między innymi poprzez usunięcie z niego zbędnych elementów, np. szumu. Czynności te wykonywane są za pomocą następujących operacji:
    - Przekształcenia geometryczne
    - Przekształcenia punktowe (bezkontekstowe)
    - Przekształcenia kontekstowe liniowe
    - Przekształcenia kontekstowe nieliniowe
    - Przekształcenia widmowe
    - Przekształcenia morfologiczne

3. **Segmentacja**. Proces ten zajmuje się wyodrębnianiem obiektów znajdujących się na obrazie i zapamiętaniu ich na odrębnym obrazie – wyniku segmentacji. Obiekty znajdujące się na obrazie odpowiadają różnym elementom sceny wizyjnej i zależą od rodzaju badanego obrazu. Przykładowo, w badaniu medycznym rezonansu magnetycznego obiektami podlegającymi segmentacji mogą być różne narządy ludzkiego organizmu. Ważną cechą obrazu będącego wynikiem segmentacji jest wyraźne oznaczenie znalezionych na nim obiektów. Wyróżniamy dwa rodzaje segmentacji: obszarową oraz konturową. W segmentacji obszarowej na obrazie wynikowym znajdują się obiekty charakteryzujące się jednolitą wartością punktu. Natomiast w segmentacji konturowej wynikiem procesu jest obwiednia wykrytych obiektów na obrazie.
4. **Wyznaczanie cech**. Etap ten polega na wyznaczeniu opisu obrazu. W tej fazie każdy z wyodrębnionych obiektów jest opisany za pomocą odpowiednich parametrów – cech. Dokładniejszy opis tego procesu znajduje się w treści poniżej.
5. **Klasyfikacja**. Celem tego procesu jest sklasyfikowanie obiektów na obrazie w odpowiednich klasach w zależności od ich cech. Głównym narzędziem klasyfikacji jest algorytm zwany klasyfikatorem. Zadanie klasyfikacji jest wykorzystywane w procesie uczenia maszynowego i nie jest przedstawione w niniejszej pracy.

W pracy przedstawiona jest aplikacja wykonująca dwie interesujące operacje przetwarzania obrazów zawarte w bibliotece OpenCV. Pierwszą z nich jest progowanie adaptacyjne. Przekształcenie to jest odmianą klasycznej operacji binaryzacji, która polega na zamianie obrazu o wielu poziomach szarości na obraz, którego piksele mają wyłącznie 2 poziomy jasności o wartościach binarnych 0 i 1. Cechą odróżniającą progowanie adaptacyjne od klasycznego jest liczenie progów lokalnych, odrębnych dla każdego piksela. Cała operacja progowania adaptacyjnego obrazu zaczyna się od podziału tego obrazu na obszary. W każdym wydzielonym obszarze wykonuje się obliczenia lokalnych charakterystyk: średniej wartości intensywności oraz minimalną i maksymalną wartość intensywności. Na podstawie tych wyników oblicza się wartość progu lokalnego dla tego wydzielonego obszaru. Aby zapobiec skokowej zmianie wartości progu lokalnego to liczy się m. in wartości progów lokalnych dla każdego piksela obrazu (wykorzystuje się lokalne otoczenie). Progowanie adaptacyjne jest operacją przetwarzania obrazów szczególnie przydatną w biologii, gdzie wykorzystuje się ją przy wykrywaniu krawędzi komórek w obrazie mikroskopowym [13].

Drugą operacją przetwarzania obrazów, która jest przedstawiona w pracy jest filtracja medianowa. Procedura aplikowania takiego filtru jest stosowana w procesie odszumiania obrazów otrzymanych np. w procesie akwizycji zdjęć. Filtry medianowe należą do grupy

filtrów nieliniowych i opierają się na obliczeniach, w których wartość wynikowa punktu jest medianą grupy punktów z sąsiedztwa, które są wykorzystywane do filtracji. Główną zaletą rozmycia medianowego jest eliminacja szumów typu sól i pieprz. Dodatkowo, filtry medianowe w przeciwieństwie do filtracji liniowych i splotowych, nie przyczyniają się do zamazywania krawędzi oraz drobnych detali widocznych na obrazie. Ważną wadą filtrów medianowych jest możliwe wprowadzenie przesunięcia krawędzi obiektów lub też usunięcie istotnych szczegółów obrazu, np. wycinanie rogów obiektów prostokątnych.

Ponadto praca prezentuje działanie funkcji detekcji cech oraz rozpoznawania twarzy. Detekcja cech kluczowych zajmuje się wykrywaniem specyficznych miejsc na obrazie cyfrowym do których należą: krawędzie, linie, zakończenia linii, rogi, skupiska, rozmaite tekstury i grzbiety. Zazwyczaj wykrywa się je za pomocą detektorów punktów kluczowych. Każdy z nich wykrywa określone cechy takie jak rogi lub krawędzie. Do najważniejszych detektorów SURF, SIFT, STAR, FAST, MSER i ORB. Z powodu braku dostępności detektorów SURF oraz SIFT w mobilnej wersji biblioteki OpenCV, w badaniach skorzystano wyłącznie z FAST, MSER oraz ORB. Następnym etapem po wykryciu cech jest ich ekstrakcja, której wynikiem są deskryptory. Są to obiekty zależne od pozycji histogramu lokalnego gradientu sąsiadów znajdujących się naokoło punktu. Do najważniejszych deskryptorów cech można zaliczyć SIFT, SURF, LAZY, BRIEF oraz ORB. Każdy z nich ma swoją specjalizację, np. w zadaniu śledzenia obiektów odpowiednim deskryptorem będzie ORB. Jeśli celem jest dokładna analiza obrazu, dobrym wyborem będzie deskryptor LAZY lub SURF. Odpowiedni dobór deskryptorów pozwoli na uzyskanie najlepszych rezultatów w wykonywanych badaniach.

2.2. Rynek urządzeń mobilnych

Rynek współczesnych urządzeń mobilnych jest bardzo różnorodny. Użytkownicy mają do wyboru szeroką gamę sprzętu mobilnego wielu producentów o rozmaitych parametrach i w różnym przedziale cenowym. Głównym kryterium klasyfikacji urządzeń mobilnych jest podział na smartfony oraz tablety. Rozróżnić je można według rozmiaru ekranu oraz ich zastosowania. Dodatkowo od niedawna można zaobserwować pojawianie się urządzeń nie należących do żadnej z wymienionych grup. Sprzęt taki jest pozycjonowany pomiędzy smartfonami i tabletami: posiada on ekran większy od tego w tradycyjnym smartfonie (rozmiar około 5 cali) a jednocześnie wciąż ma możliwość wykonywania połączeń głosowych i wysyłania wiadomości tekstowych w obrębie sieci komórkowej. Wśród producentów sprzętu mobilnego można wyróżnić nie tylko firmy od lat specjalizujące się w produkcji urządzeń elektronicznych i mobilnych ale także coraz popularniejsi są producenci

urządzeń niskobudżetowych, którzy często ograniczają się do produkcji zaledwie kilku modeli tabletów. Do firm wiodących można zaliczyć Samsunga oraz Apple, którzy od dawna wyznaczają trendy rozwoju rynku urządzeń mobilnych oraz firmy Nokia i HTC, które stale wypuszczają na rynek nowe, ciekawe produkty.

Początki smartfonów sięgają roku 1992, kiedy to IBM zaprezentował prototyp o nazwie Simon. Urządzenie to, oprócz funkcji telefonu komórkowego oferowało również dostęp to poczty elektronicznej, kalendarza, kalkulatora, notatnika i prostych gier. Do sterowania urządzeniem przeznaczone było specjalne pióro świetlne [16]. W następnych latach następowała stopniowa ewolucja tego typu urządzeń. Można było zaobserwować rozdział tego sprzętu na smartfony oraz na komputery PDA. Urządzenia PDA były sprzętem o możliwościach identycznych z ówczesnymi smartfonami. Jedyną różnicą był brak modułu GSM. Obie powyższe grupy urządzeń pracowały pod kontrolą systemu operacyjnego Windows Mobile. Sprzęt mobilny wyposażony w ten system charakteryzował się ekranem opornościowym o słabych parametrach, którego w miarę wygodna kontrola była możliwa wyłącznie za pomocą rysika. Z biegiem lat system Windows Mobile stawał się systemem coraz mniej nowoczesnym. Postęp technologiczny spowodował powstawania coraz doskonalszych podzespołów typu ekran pojemnościowy. Windows Mobile pomimo kilku aktualizacji nie wspierał tego typu wyświetlaczy ani innych nowych technologii. Rok 2007 był początkiem nowej ery w mobilnych systemach operacyjnych [17]. W tym roku powstała pierwsza wersja jednej z najpopularniejszych platform mobilnych – Android firmy Google. W tym roku światło dzienne ujrzał również system operacyjny iOS (wtedy: iPhone OS) na urządzeniu iPhone firmy Apple. Obecnie systemy te uległy znacznemu rozwinięciu i przystosowaniu do współczesnych urządzeń i standardów. Więcej informacji na temat tych platform mobilnych znajduje się w rozdziale 3.

Ostatnie lata spowodowały zwiększenie liczby mobilnych systemów operacyjnych. Często są one ograniczone do urządzeń tego samego producenta, ale można znaleźć też platformy instalowane na urządzeniach różnych firm. Wśród niszowych systemów mobilnych warto wymienić BlackBerry OS, bada, Maemo, MeeGo oraz Symbian, który z powodu powolnego rozwoju zostaje porzucony przez firmę Nokia, która w ostatnich latach zajmowała się rozwijaniem tej platformy oraz instalowaniem jej w swoich urządzeniach. Popularność niektórych mobilnych systemów operacyjnych powoduje, że coraz więcej firm postanawia dołączyć do grupy twórców takich platform. W najbliższym czasie na rynku pojawią się nowe urządzenia firmy BlackBerry z całkowicie przebudowanym systemem BB10 OS. Także firmy dotąd niezwiązane z produkcją mobilnych systemów operacyjnych przygotowują swoje produkty. Wśród tych producentów warto wymienić m. in Canonical, twórców

komputerowego systemu Ubuntu Linux, którzy konstruują system Ubuntu Phone OS, czy połączone firmy Samsung i Intel zajmujące się produkcją systemu Tizen.

W dzisiejszych czasach parametry sprzętowe urządzeń mobilnych są zróżnicowane i zbliżone do tych spotykanych w klasycznych komputerach PC spotykanych kilka lat temu. Natomiast nawet kilka lat temu wydajność urządzeń mobilnych była niska. Układy wbudowane w urządzenia były jednostkami jednordzeniowymi o taktowaniu często nieprzekraczającym 400 MHz. Takie urządzenia nie miały możliwości przeprowadzania skomplikowanych obliczeń. Systemem operacyjnym, który był najczęściej spotykany na takim sprzęcie był Windows Mobile. W niektórych urządzeniach instalowany był system Symbian. Obie platformy pracowały wolno i były słabo przystosowane do obsługi ekranów dotykowych.

Obecnie na rynku można spotkać sprzęt wyposażony w układy obliczeniowe o różnych wartościach taktowania procesora i ilości rdzeni. W segmencie budżetowym dominują tablety z procesorami jednordzeniowymi o wartości taktowania od 1 do 1,2 GHz, jednak coraz częściej można spotkać produkty o układach dwurdzeniowych wyposażonych w dobry układ graficzny i 512 MB pamięci RAM. Wśród urządzeń z segmentu premium spotykać można produkty o zróżnicowanych podzespołach. Dużo sprzętu wyposażone jest w nowoczesne jednostki obliczeniowe charakteryzujące się wieloma rdzeniami oraz wysokim taktowaniem procesora oraz układu graficznego. W bieżącym roku Samsung zaprezentował nowy, wydajny układ obliczeniowy wyposażony w 8 rdzeni, który będzie wykorzystywany w najnowszych urządzeniach mobilnych tej firmy. Również producenci, znani wcześniej tylko z układów graficznych dla komputerów stacjonarnych i laptopów, uczestniczą w rozwoju układów obliczeniowych urządzeń mobilnych. Nvidia, zajmująca się wcześniej tylko produkcją układów graficznych GeForce, od kilku lat istnieje na rynku urządzeń mobilnych ze swoimi układami Tegra. W styczniu 2013 roku podczas targów elektroniki użytkowej w Las Vegas, Nvidia zaprezentowała najnowszą generację swoich układów mobilnych – Tegra 4. Układ ten będzie wyposażony w jednostkę centralną o 4 głównych rdzeniach i 1 dodatkowym) o częstotliwości taktowania oscylującej w granicach 1,8 Ghz [10]. Również firma AMD, znana z produkcji procesorów i układów graficznych dla klasycznych komputerów, pokazała na tegorocznych targach swoją propozycje układów SoC (ang. System on a Chip). Jedną z nich jest wyprodukowana w technologii 28 nm jednostka o nazwie Temash, przeznaczona do tabletów i mająca na celu zapewnienie urządzeniom mobilnym wydajności znanej z komputerów klasycznych.

Systemy operacyjne dla urządzeń mobilnych mają możliwość uruchamiania dużej liczby aplikacji i gier. Smartfony i PDA z pierwszymi wersjami systemu Windows Mobile

miały możliwość instalacji odpowiednich aplikacji z plików wykonywalnych. Aby ułatwić użytkownikom instalację nowych programów, producenci postanowili stworzyć własne sklepy z aplikacjami dla swoich systemów. Pierwsze próby podjął Microsoft w swoim systemie Windows Mobile w wersji 6.5. Sklep z aplikacjami dla tego systemu nie zdobył zbyt dużej popularności, co skutkowało małą liczbą programów w sklepie. Początkiem rewolucji związanej z rozwojem sklepów dla aplikacji mobilnych było powstanie App Store dla systemu iOS w 2007 roku (prezentacja smartfonu iPhone 2G). W następnej kolejności powstał sklep Android Market (dzisiejsza platforma Google Play) dla systemu Android firmy Google. Obecnie powyższe platformy cieszą się dużą popularnością oraz najbogatszym asortymentem aplikacji zarówno darmowych, jak i płatnych. Innymi sklepami z aplikacjami mobilnymi są SamsungApps, który zawiera aplikacje dla systemu mobilnego bada, dla systemu Android oraz dla platformy SmartTV w telewizorach produkcji Samsunga oraz BlackBerry App World, który skupia w sobie aplikacje dla systemów operacyjnych w urządzeniach BlackBerry. Dwie powyższe platformy nie mają dużej popularności i ilość aplikacji dostępnych w tych sklepach jest zdecydowanie mniejsza, niż w sklepach App Store oraz Google Play.

Od kilku lat obserwuje się znaczny wzrost zainteresowania urządzeniami mobilnymi wśród użytkowników w różnym wieku. Dodatkowo widać brak zmian w sprzedaży klasycznych komputerów, zarówno stacjonarnych jak i laptopów. Coraz częściej mówi się o tzw. erze „post-pc", która oznacza właśnie zmniejszenie zainteresowania tradycyjnymi komputerami na rzecz urządzeń mobilnych, które to bardzo dobrze nadają się do zwykłej konsumpcji treści. Dzięki nim można w sposób wygodny i intuicyjny przeglądać strony internetowe, komunikować się ze znajomymi a także oglądać filmy czy grać w gry. Według raportu agencji Nielsen około 40% osób planuje kupić tablet w najbliższym czasie. Natomiast chęć nabycia nowego komputera zgłasza około 18% badanych [9]. Dane te pokazują, jak bardzo popularne stały się urządzenia mobilne. Jednak do zastosowań biznesowych tablety są mało przydatne. W tym sektorze wciąż największe zastosowanie mają tradycyjne komputery, które pozwalają na precyzyjną obróbkę i przygotowywanie rozmaitych danych. Takie czynności nie są w większości przypadków możliwe do wykonania na urządzeniach mobilnych., co powoduje że wśród użytkowników biznesowych, klasyczne komputery stacjonarne i laptopy są najczęściej wybieranymi urządzeniami do zadań niezwiązanych z konsumpcją treści.

Innym rodzajem urządzeń, które należą do grupy mobilnych są tzw. „single board computers", czyli małe komputery, w których wszystkie niezbędne do działania podzespoły zostały umieszczone na jednej płytce drukowanej. Jedyną znaczącą różnicą między takim

sprzętem a klasycznym pełnowymiarowym komputerem jest brak możliwości instalacji kart rozszerzeń. Większość komputerów jednopłytowych oparta jest na architekturze ARM. W tej grupie sprzętu istnieje podział według zastosowanego układu obliczeniowego. Warto wymienić tutaj między innymi komputery VIA APC, Gooseberry lub Arduino Due. Jednak najpopularniejszym przykładem takich mobilnych urządzeń jest Raspberry Pi bazujące na procesorze ARM11. Dużą zaletą takich urządzeń jest ich stosunkowo niska cena oraz duże możliwości wykorzystania, jako zamiennik tradycyjnego komputera. Wadą natomiast są problemy z dostępnością wielu modeli oraz brak możliwości instalacji dowolnego systemu operacyjnego – zazwyczaj muszą to być ściśle określone dystrybucje systemu operacyjnego Linux. W niniejszej pracy pominięto badania biblioteki OpenCV na komputerach jednopłytowych. Mimo szybkiego rozwoju tego typu platform, nie posiadają one wsparcia dla większości zewnętrznych kamer, a także wśród systemów operacyjnych możliwych do uruchomienia jest czasami brak systemu Android lub możliwe problemy z uruchomieniem biblioteki OpenCV na zastosowanej wersji systemu.

Rozdział 3

Charakterystyka biblioteki OpenCV

OpenCV (pełna nazwa „Open Source Computer Vision") jest biblioteką funkcji programistycznych wykorzystywanych w szerokiej dziedzinie przetwarzania obrazów. Biblioteka ta miała swój początek w laboratoriach firmy Intel jako narzędzie mające na celu przyspieszenie wykonywania obliczeń związanych z przetwarzaniem obrazów a także stworzenie infrastruktury wizji komputerowej powszechnie dostępnej. Z biegiem lat biblioteka OpenCV została przekształcona na projekt wolnego oprogramowania (ang. Open Source), którym pozostaje do chwili dzisiejszej. Obecnie za rozwój OpenCV odpowiada Willow Garage, laboratorium badawcze zajmujące się rozwijaniem sprzętu oraz wolnego oprogramowania dla robotyki. Biblioteka OpenCV zawiera ponad pięćset funkcji programistycznych obejmujących sobą dziedziny przetwarzania obrazów oraz widzenia komputerowego. Dzięki niej istnieje między innymi możliwość dokonywania rozmaitych przekształceń obrazów, śledzenia obiektów ruchomych oraz wykrywania obiektów na obrazach. Konstrukcja biblioteki pozwala na wykorzystanie jej do rozmaitych celów, zarówno przez osoby początkujące, pragnące poznać podstawowe tajniki przetwarzania obrazów, jak i przez osoby zaawansowane, wykorzystujące bibliotekę do rozbudowanych projektów badawczych lub komercyjnych. Bogactwo jej funkcji spowodowało, że znalazła ona zastosowanie w wielu narzędziach stosowanych w codziennym życiu. Jako przykład posłużyć mogą usługi map internetowych i satelitarnych, w których biblioteka OpenCV wykorzystywana jest między innymi do prawidłowego połączenia zdjęć satelitarnych oraz zdjęć ulic lub w systemach ochrony służących do monitoringu i wykrywania intruzów [1].

Biblioteka została pierwotnie przygotowana w języku C, lecz jej rosnąca popularność spowodowały powstanie nowych implementacji dla innych języków programowania: C++, Java, Python oraz Matlab. Z biblioteki OpenCV można korzystać na wszystkich popularnych systemach operacyjnych: Windows, Linux oraz MacOS. Istnieją również specjalnie przygotowane wersje biblioteki przeznaczone do uruchamiania jej na urządzeniach mobilnych działających w oparciu o systemy Android firmy Google oraz iOS firmy Apple. Niniejsza praca dotyczy biblioteki OpenCV uruchamianej na urządzeniach mobilnych działających pod kontrolą systemu Android. Z powodu braku dostępu do urządzeń z systemem iOS zostały one pominięte w pracy. Dodatkowym czynnikiem uniemożliwiającym wykorzystanie urządzeń opartych na systemie iOS była konieczność posiadania komputera typu Mac oraz środowiska deweloperskiego Xcode dla systemu MacOS X.

3.1. Modułowość biblioteki

Biblioteka OpenCV została podzielona na moduły, które to zawierają funkcje dotyczące przypisanych im dziedzin. Taka budowa sprawia, że użytkownik może w swoim projekcie wykorzystać tylko te moduły, które są aktualnie potrzebne. Pozwala to na znaczne zmniejszenie rozmiaru programu i zachowanie porządku w kodzie aplikacji. Początkowo, zanim powstała wersja 2.0 biblioteki, OpenCV była podzielone na zaledwie cztery moduły. We wspomnianej wersji liczbę modułów zwiększono do dziewięciu, co umożliwiło wyspecjalizowanie każdego modułu do jednej konkretnej dziedziny oraz doprowadziło do wymienionego wyżej zmniejszenia rozmiaru aplikacji końcowej oraz zwiększenia porządku w kodzie źródłowym programu.
Aktualne najważniejsze moduły zawarte w bibliotece OpenCV to:

- **core** – moduł zawierający podstawowe funkcje biblioteki
- **imgproc** – moduł realizujący podstawowe funkcje przetwarzania obrazów
- **video** – moduł zajmujący się analizą wideo
- **calib3d** – zawiera algorytmy rekonstrukcji 3D oraz zajmuje się kalibracją kamery
- **features2d** – zajmuje się wykrywaniem istotnych cech obrazów 2d
- **objdetect** - algorytmy wykrywania obiektów
- **highgui** – interfejs użytkownika wysokiego poziomu, obsługa interfejsu wejścia/wyjścia
- **gpu** – algorytmy wykorzystujące procesory graficzne do wspomagania obliczeń wizji komputerowej, zgodność z technologią CUDA
- **ml** – moduł odpowiedzialny za uczenie maszynowe

3.2. Biblioteka OpenCV na urządzeniach mobilnych

Duża popularność współczesnych urządzeń mobilnych oraz ich wysoka wydajność spowodowały, że twórcy OpenCV przygotowali specjalne wersje biblioteki gotowe do uruchomienia na najczęściej spotykanych mobilnych systemach operacyjnych. Pierwszą platformą mobilną, na jakiej można było powszechnie używać biblioteki był Windows Mobile firmy Microsoft. Obecnie system ten jest przestarzały i niezgodny ze współczesnymi standardami co spowodowało, że nieopłacalne stało się dalsze wspieranie Windows Mobile i twórcy OpenCV przestali rozwijać bibliotekę na tej platformie.

W dzisiejszych czasach mobilne systemy operacyjne charakteryzują się dużą szybkością działania oraz niezawodnością i dostosowaniem do współczesnych standardów i sprzętu. Wśród nich można wymienić dwa systemy o ugruntowanej od wielu lat pozycji na rynku: Android oraz iOS. Urządzenia w nie wyposażone charakteryzują się wydajnymi układami obliczeniowymi oraz dużą ilością pamięci RAM, której wielkość jest porównywalna do pamięci RAM stosowanej dotychczas w klasycznych komputerach typu PC. Układy obliczeniowe stosowane we współczesnych urządzeniach mobilnych bezproblemowo wykonują wszystkie obliczenia związane z przetwarzaniem obrazów i wizją komputerową.

System operacyjny Android firmy Google jest platformą rozwijaną od wielu lat i działającą na największej liczbie urządzeń mobilnych na rynku. Łatwa dostępność kodu źródłowego (licencja Open Source) systemu powoduje, że można z nim mieć styczność na szerokiej gamie sprzętu począwszy od smartfonów oraz tabletów po urządzenia typu aparat fotograficzny. Pierwsze komercyjne urządzenia oparte o system operacyjny Android ukazało się na rynku 22 października 2008 roku [17]. Od tego momentu do chwili obecnej system Android jest wciąż rozwijany a na rynku można znaleźć urządzenia pracujące w oparciu o różne jego wersje. Najczęściej spotyka się urządzenia pracujące na systemie Android w wersji 4.0.x (linia Ice Cream Sandwich) oraz w wersji 2.3.x (linia Gingerbread). Przez krótki okres czasu, wśród tabletów można było spotkać system Android 3.x (Honeycomb). Aby zapobiec wielu problemom ze zgodnością aplikacji, twórcy systemu postanowili stworzyć jedną uniwersalną wersję dla wszystkich typów urządzeń mobilnych. Coraz więcej producentów decyduje się udostępniać aktualizacje swoich urządzeń do platformy Android w wersji 4.1.x (Jelly Bean). Obecnie najnowszą wersją jest Android 4.2 o nazwie Jelly Bean. Każde kolejne wersje systemu są dokładnie analizowane przez twórców biblioteki OpenCV pod kątem możliwych błędów w działaniu oraz problemów ze zgodnością biblioteki. Android 4.2 jest systemem w pełni wspieranym przez bibliotekę OpenCV.

System operacyjny iOS, wyprodukowany przez Apple jest platformą mobilną przeznaczoną dla urządzeń tej firmy do których należą smart fony iPhone, tablety iPad oraz urządzenia iPod. iOS charakteryzuje się niedostępnym publicznie kodem źródłowym systemu. Powoduje to brak możliwości zewnętrznej ingerencji w system a także ograniczenie obsługiwanego sprzętu tylko do urządzeń firmy Apple. Wady te rekompensowane są dużą stabilnością systemu, prostotą użytkowania oraz bogatym zasobem aplikacji dostępnych w firmowym sklepie. Pierwsza wersja systemu iOS ukazała się w dniu premiery smartfonu iPhone 2G dnia 29 czerwca 2007 roku [18]. System ten podlega ciągłemu rozwojowi do chwili dzisiejszej. Duża popularność produktów mobilnych marki Apple spowodowała, że postanowiono przygotować wersję biblioteki OpenCV, z której można korzystać na

urządzeniach opartych o system iOS. Na tej platformie biblioteka posiada praktycznie identyczne możliwości, jakie posiada OpenCV w wersji przygotowanej dla systemu Android. Praca nie dotyczy biblioteki OpenCV dla systemu iOS.

3.3. Wymagania biblioteki OpenCV dla systemu Android

Aby móc tworzyć aplikacje dla systemu Android, które korzystają z biblioteki OpenCV, wymagane jest posiadanie odpowiedniego oprogramowania. Pierwszym ważnym składnikiem jest aktualna paczka o nazwie „OpenCV4Android SDK", zawierająca w sobie niezbędne pliki biblioteki, instrukcje, przykładowe aplikacje, oraz program OpenCV Manager wymagany do instalacji na docelowym urządzeniu mobilnym wraz z bibliotekami. Dokładniejszy opis programu OpenCV Manager znajduje się w kolejnym podrozdziale. W pracy wykorzystano bibliotekę OpenCV w wersji 2.4.2. Kolejnymi narzędziami które są konieczne do pisania aplikacji na Androida są Oracle JDK 6 oraz Android SDK. Oracle JDK 6 (Java Development Kit) jest pakietem udostępniającym w systemie środowisko przeznaczone do programowania w języku Java. Zawiera między innymi program kompilujący oraz debugger, które to są niezbędne do tworzenia każdej aplikacji w tym języku. Android SDK (Software Development Kit) jest pakietem dostarczającym programiście interfejs programowania aplikacji (ang. API) oraz narzędzia deweloperskie przydatne przy tworzeniu własnych aplikacji dla systemu Android. Kluczową aplikacją pakietu Android SDK jest SDK Manager. Za jego pomocą wybiera się oraz instaluje niezbędne składniki, które będą pomocne w procesie pisania i testowania aplikacji na systemie Android. Z poziomu tej aplikacji można między innymi pobrać sterowniki USB konieczne do poprawnej współpracy telefonu z komputerem. Aby biblioteka OpenCV prawidłowo współpracowała z posiadanymi urządzeniami, twórcy zalecają pobranie elementów SDK Tools oraz SDK Platform Android 3.0, które można pobrać w aplikacji SDK Manager. Najlepszym środowiskiem programistycznym do pisania aplikacji na system Android jest program Eclipse. Jest to narzędzie, którego współpraca z tym systemem jest zapewniona przez producenta, a także jest polecane przez twórców biblioteki OpenCV. Aby środowisko Eclipse komunikowało się bez przeszkód z systemem Android, wymagana jest instalacja rozszerzenia o nazwie ADT – Android Developer Tools.

Biblioteka OpenCV dla omawianego systemu operacyjnego pozwala na pisanie programów w języku Java i dodatkowo zezwala na wykorzystanie jej natywnych elementów oraz kodu źródłowego napisanego w języku C++. Taka możliwość istnieje po zainstalowaniu dwóch niezbędnych składników: pakietu Android NDK oraz rozszerzenia CDT dla aplikacji

Eclipse. Pakiet NDK – Native Development Kit jest zestawem narzędzi który pozwala twórcy oprogramowania na realizację wspomnianej w bieżącym akapicie implementacji kodu w języku natywnym C lub C++ w aplikacji tworzonej bezpośrednio dla systemu Android. Rozszerzenie CDT (C/C++ Development Tools) umożliwia realizację tego celu za pomocą środowiska Eclipse.

3.4. OpenCV Manager

OpenCV Manager jest programem - usługą instalowaną bezpośrednio na urządzeniu z systemem Android, której głównym zadaniem jest zapewnienie aplikacjom wykorzystującym bibliotekę OpenCV pełnego dostępu do niej a także umożliwienie im bezproblemowego uruchomienia. Do zalet aplikacji OpenCV Manager należą:

1. Bezpośrednie wsparcie dla wszystkich obecnych na rynku platform sprzętowych. Twórcy zapewnili możliwość skorzystania z biblioteki na każdej architekturze spotykanej w procesorach wykorzystywanych w urządzeniach mobilnych. Obsługa ta jest możliwa poprzez zainstalowanie na urządzeniu odpowiedniego rozszerzenia programu OpenCV Manager. W platformie z aplikacjami mobilnymi dla systemu Android można znaleźć rozszerzenia dla urządzeń z procesorami z rodziny ARM, z procesorami z rodziny ARM wykorzystującymi zestaw instrukcji NEON, dla układów opartych na architekturze x86 a także rozszerzenie dla urządzeń pracujących pod kontrolą układów NVIDIA Tegra 3.
2. Niskie zużycie pamięci. Wszystkie aplikacje wykorzystujące OpenCV korzystają z tej samej usługi i nie ma konieczności dołączania biblioteki do każdego programu. Powoduje to, że gotowa aplikacja zajmuje mniej miejsca w pamięci urządzenia i jest efektywniej przetwarzana przez układ obliczeniowy urządzenia mobilnego.
3. Zaufane źródło biblioteki OpenCV. OpenCV Manager jak i jego rozszerzenia pochodzą bezpośrednio od twórców biblioteki i są publikowane w sklepie z aplikacjami Google Play.
4. Regularne aktualizacje i poprawki błędów. Ciągły rozwój biblioteki OpenCV obejmuje między innymi dodawanie nowych funkcji, eliminację błędów oraz poprawianie zabezpieczeń. Wersja dla systemu Android podlega takiemu samemu zakresowi aktualizacji, dzięki czemu jej użytkownicy mają zawsze zapewnioną najnowszą wersję biblioteki.

Każda aplikacja wykorzystująca bibliotekę i będąca zależną od omawianego programu OpenCV Manager podlega następującemu schematowi użycia:

1. Instalacja aplikacji u użytkownika końcowego.
2. Przy pierwszym uruchomieniu, aplikacja sugeruje instalację usługi OpenCV Manager. Jeśli wspomniana aplikacja była wcześniej zainstalowana, to proces jej uruchomienia przebiega bez kontroli usługi OpenCV Manager.
3. Kolejnym krokiem jest pobranie i instalacja usługi OpenCV Manager (w przypadku, gdy nie została wcześniej zainstalowana).
4. Po uruchomieniu OpenCV Manager aplikacja pyta użytkownika o zgodę na instalację dodatkowych bibliotek zoptymalizowanych dla danej architektury urządzenia mobilnego (jeśli są konieczne).
5. Po instalacji dodatkowych bibliotek aplikacja zostaje uruchomiona.

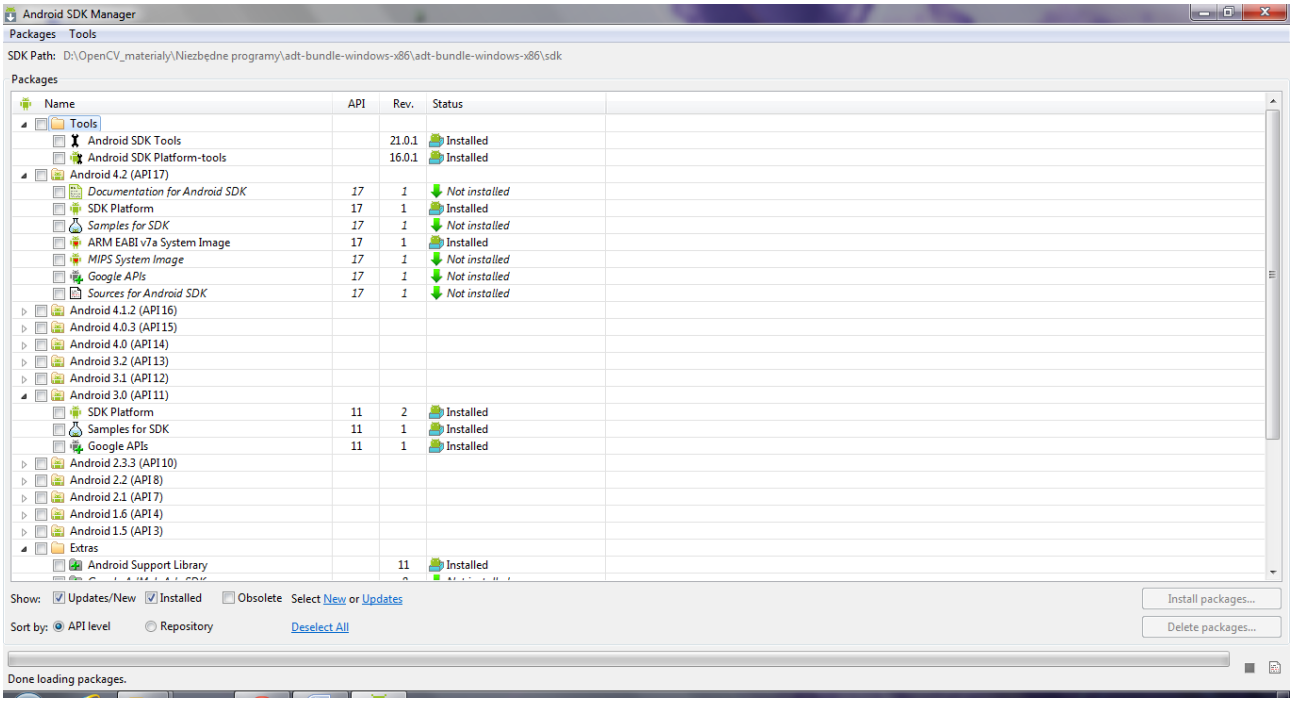
3.5. Instalacja i przygotowywanie biblioteki OpenCV w wersji dla systemu Android

Bezproblemowe korzystanie z biblioteki OpenCV, a także pisanie aplikacji dla systemu Android wymagają zainstalowania w systemie kilku ważnych aplikacji i pakietów wymienionych w podrozdziale 3.3. Ważnym etapem przygotowywania biblioteki OpenCV jest przygotowanie środowiska Eclipse do współpracy z biblioteką oraz zapewnienie sprawnej komunikacji komputera z urządzeniem działającym w oparciu o system Android.
Poniżej zaprezentowano ogólny przebieg instalacji wszystkich składników niezbędnych do pełnego korzystania z możliwości biblioteki OpenCV dla systemu Android. Odkąd firma Google zdecydowała się udostępniać pakiet Android SDK razem z odpowiednio przygotowanym środowiskiem Eclipse, proces przygotowywania komputera do pisania aplikacji dla systemu Android został znacznie ułatwiony.

1. Instalacja pakietu Oracle JDK 6 oraz rozpakowanie paczki ADT-bundle. Najpierw pobrano pakiety Oracle JDK 6 (Java Development Kit 6) oraz ADT-bundle. Ich instalacja sprowadzała się do postępowania zgodnie z poleceniami instalatora pakietu Oracle JDK 6 oraz rozpakowania do wybranej lokalizacji archiwum z oprogramowaniem ADT-bundle.
2. Rozpakowanie archiwum zawierającego bibliotekę OpenCV dla systemu Android. Następną czynnością było pobranie i rozpakowanie paczki zawierającej bibliotekę OpenCV przygotowaną dla systemu Android. W archiwum oprócz folderu z SDK

biblioteki można znaleźć przydatną dokumentację OpenCV oraz zbiór przykładowych aplikacji prezentujących możliwości biblioteki.

3. Dostosowanie elementów pakietu Android SDK. Za pomocą aplikacji SDK Manager, wchodzącej w skład pakietu ADT-bundle, zainstalowano odpowiednie elementy istotne dla procesu programowania aplikacji wykorzystującej bibliotekę OpenCV.



Rysunek 3.1 Widok programu SDK Manager

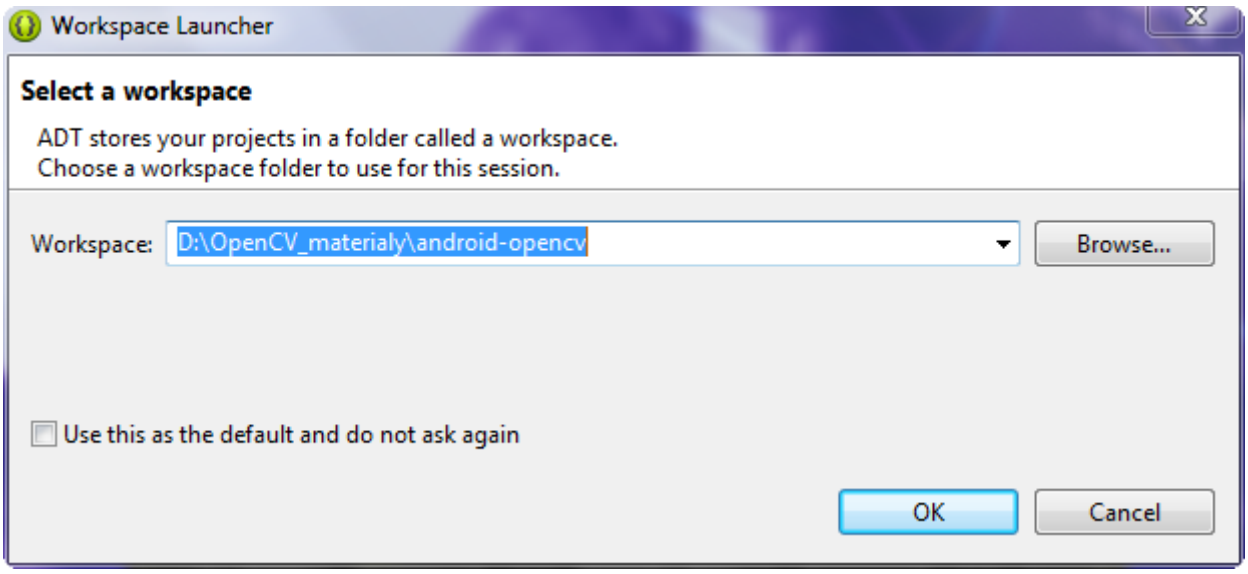
Składnikami kluczowymi dla współpracy biblioteki OpenCV z większością urządzeń mobilnych działających pod kontrolą systemu Android jest posiadanie systemu w wersji 2.2 a także zainstalowanie następujących składników: Android SDK Tools w wersji 20 lub nowszej, sterownika Google USB Driver oraz odpowiedniej paczki SDK Platform Android. Twórcy biblioteki zalecają instalację paczki SDK Platform Android 3.0. Wybór ten zapewnia poprawną kompilację gotowej aplikacji OpenCV i bezproblemowe uruchomienie jej na posiadanych urządzeniach. Okno główne programu z listą pakietów możliwych do instalacji przedstawione jest na rysunku 3.1.

4. Import biblioteki OpenCV razem z przykładami do środowiska Eclipse

Kolejnym krokiem było wczytanie biblioteki OpenCV do środowiska Eclipse.

W tym celu wykonano następujące czynności:

- **Krok 1**: Wybór domyślnej przestrzeni roboczej. Podczas uruchomienia środowiska Eclipse wybrano domyślną przestrzeń roboczą. Zalecany jest wybór przestrzeni roboczej w folderze z rozpakowaną biblioteką OpenCV dla systemu Android. (Rys. 3.2)

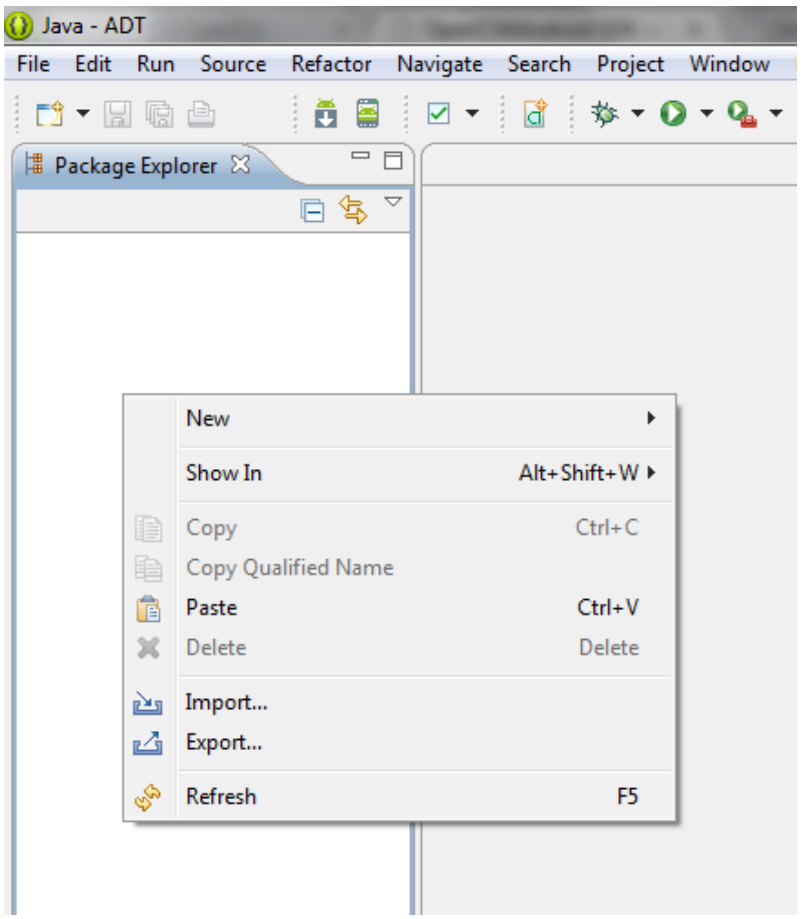


Rysunek 3.2 – Krok1: Wybór przestrzeni roboczej w aplikacji Eclipse

- Import biblioteki OpenCV do przestrzeni roboczej

  W tym kroku należało zaimportować bibliotekę OpenCV do przestrzeni roboczej środowiska Eclipse.

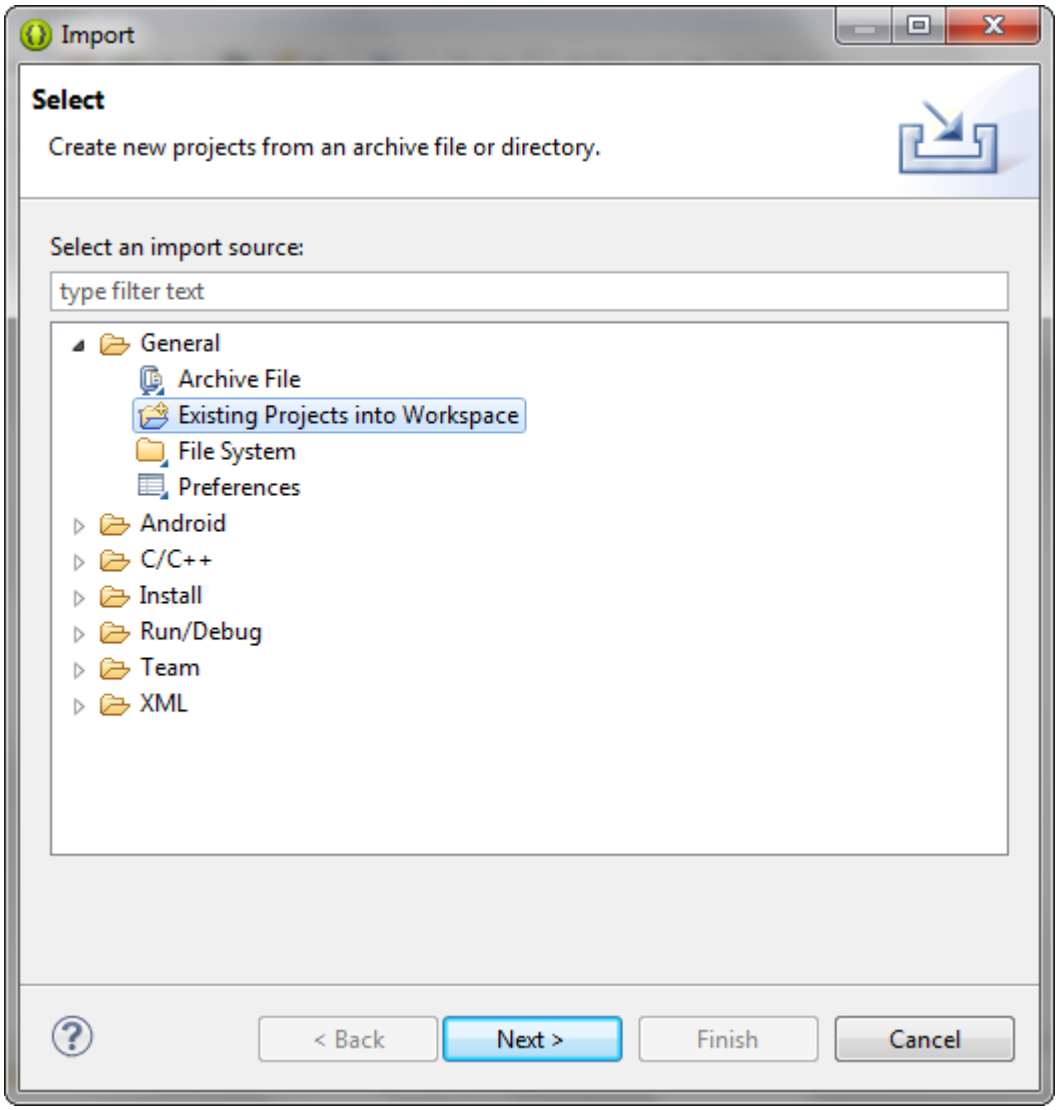
- **Krok 2**: W oknie Package Explorer należy wybrać z menu podręcznego opcję „Import” (Rys. 3.3)



Rysunek 3.3– Krok 2: Położenie opcji „Import” w menu podręcznym okna Package Explorer

- **Krok 3**: Wybór opcji wczytania gotowych projektów do środowiska roboczego środowiska Eclipse.

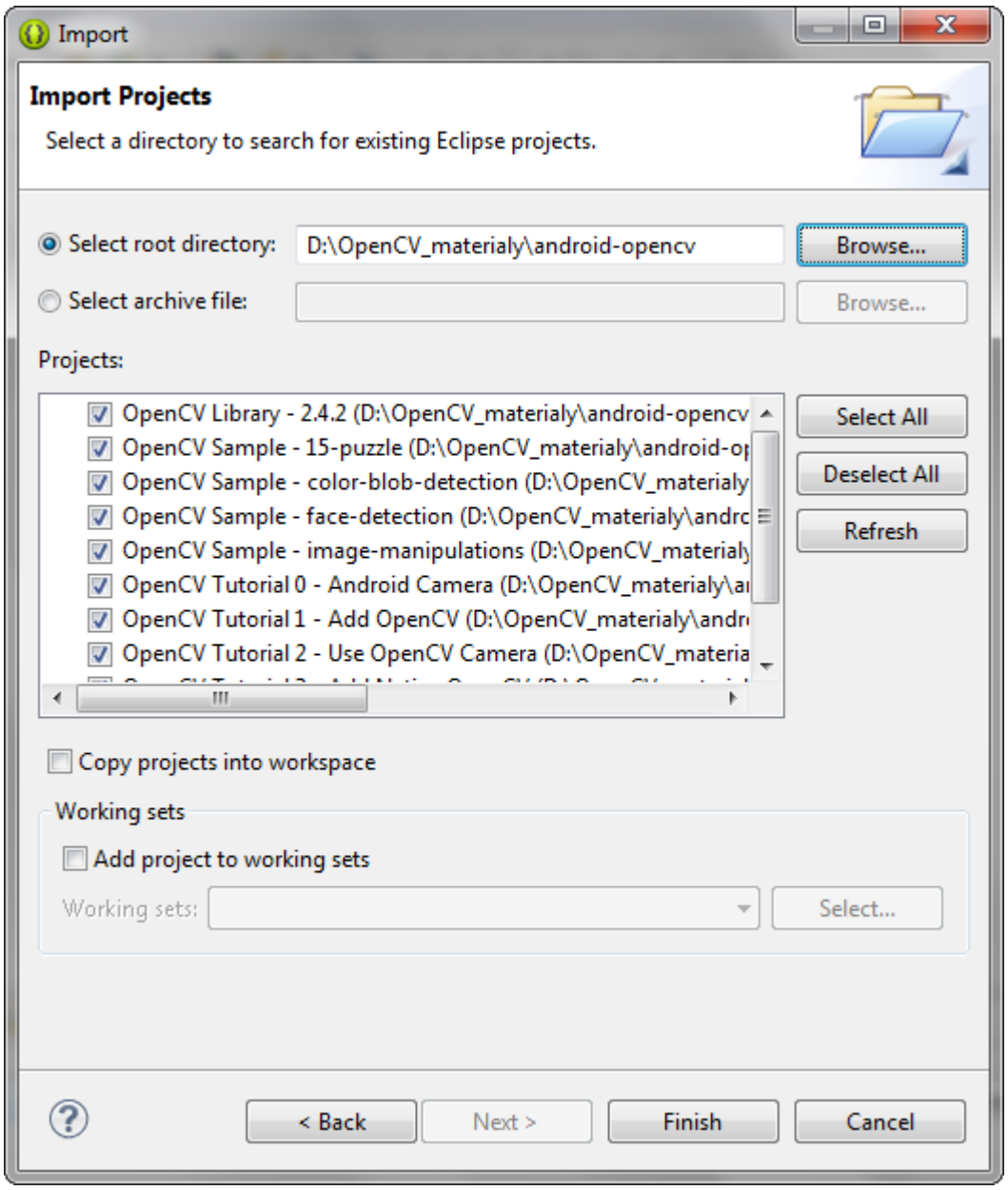
  W nowym oknie zaznaczono opcję „Existing projects into workspace”, z grupy „General” oraz wciśnięto przycisk „Next”. Opcja ta pozwala na import biblioteki OpenCV oraz przygotowanych aplikacji przykładowych. (Rys. 3.4)



Rysunek 3.4 – Krok 3: Wybór źródła importu

- **Krok 4**: Wskazanie ścieżki dostępu do biblioteki OpenCV dla systemu Android.

  W kolejnym oknie należy wcisnąć przycisk „Browse” oraz w nowopowstałym okienku wskazać lokalizację folderu z rozpakowaną paczką zawierającą bibliotekę OpenCV dla systemu Android. Jeśli postąpiono zgodnie z zaleceniem przedstawionym w Kroku 1, wskazywanie lokalizacji biblioteki nie jest konieczne. Wystarczy wówczas wcisnąć przycisk „OK” i poczekać, aż środowisko Eclipse rozpozna projekty znajdujące się w wybranym folderze. Na koniec tego etapu należało wcisnąć przycisk „Finish”. (Rys. 3.5)

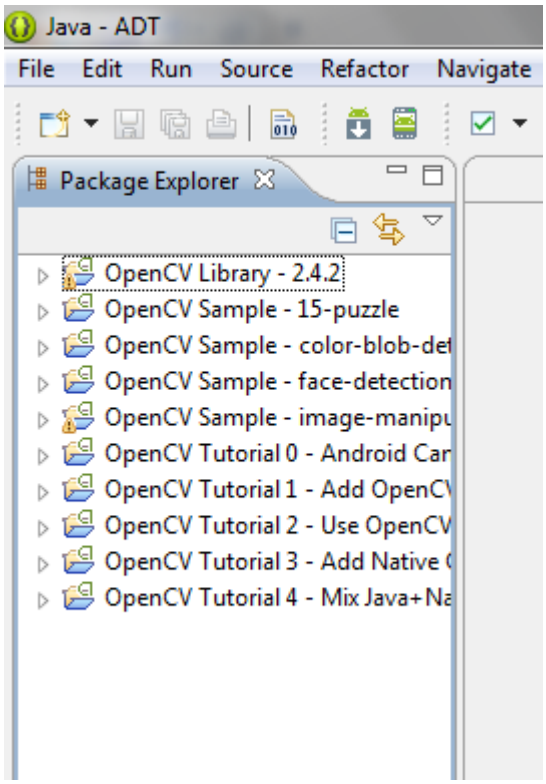


Rysunek 3.5 – Krok 4: Okno wyboru projektów oraz biblioteki OpenCV do importu

- **Krok 5**: Całkowite wczytanie biblioteki do środowiska Eclipse oraz naprawa błędu uniemożliwiającego całkowity import.

  W tym etapie Eclipse zaimportował bibliotekę do przestrzeni roboczej widocznej w oknie Package Explorer. Zanim biblioteka i przykładowe aplikacje były gotowe do użycia, środowisko Eclipse przeprowadziło szereg procesów kompilacji i budowy biblioteki i przykładowych aplikacji w środowisku roboczym. Po ukończeniu tego procesu wystąpił szereg błędów budowy w bibliotece OpenCV. Jest to znany problem spowodowany nieprawidłowościami we współpracy obecnej wersji biblioteki OpenCV dla systemu Android oraz dostarczanego przez Google środowiska Eclipse ADT. Aby zlikwidować ten problem, należy przed importem biblioteki do środowiska Eclipse usunąć folder „.settings” znajdujący się w lokalizacji: **OpenCV-2.4.2-**

android-sdk/sdk/java w folderze do którego rozpakowano archiwum z biblioteką. (Rys. 3.6)



Rysunek 3.6 – Krok 5: Widok zaimportowanej biblioteki w oknie Package Explorer środowiska Eclipse

- W tym momencie komputer został przygotowany do współpracy z posiadanymi urządzeniami opartymi na systemie Android oraz z biblioteką OpenCV na tę platformę.

3.6. Programowanie natywne biblioteki OpenCV w języku C++

Podczas programowania dla systemu Android istnieje możliwość wykorzystania natywnych elementów napisanych w języku C++. Jest to wykonywalne przy pomocy dwóch wymienionych wcześniej pakietów: Android NDK oraz rozszerzenia CDT.

W związku z dużym uproszczeniem procesu przygotowywania komputera do programowania dla systemu Android, rozszerzenie CDT jest już zawarte w programie Eclipse należącym do pakietu ADT-bundle. Programista powinien w tym przypadku jedynie pobrać i rozpakować paczkę z pakietem narzędzi Android NDK.

Dla programowania natywnego w systemie Android ważna jest struktura kodu źródłowego każdej aplikacji.

Struktura ta wygląda następująco:

Folder główny programu:

- *src/*
- *jni/*
- *libs/*
- *res/*
- *Plik AndroidManifest.xml*
- *Plik project.properties*

W przypadku programowania wykorzystującego język C++ ważnym folderem w powyższej strukturze jest *jni*, który to zawiera kod źródłowy aplikacji w języku C lub C++ (plik z rozszerzeniem cpp), a także skrypty *Android.mk* oraz *Application.mk*.

Plik *AndroidManifest.xml* zawiera m. in. podstawowe informacje o tworzonej aplikacji takie jak nazwa, wymagane pozwolenia, składniki aplikacji. Plik *project.properties* przechowuje dane o docelowej wersji systemu Android oraz inne informacje tworzone w czasie kompilacji aplikacji. Plik *Android.mk* jest najważniejszy w procesie programowania z wykorzystaniem elementów natywnych. Jego zadaniem jest przeprowadzanie kompilacji kodu C++ tworzonej aplikacji. Natomiast drugim ważnym plikiem jest *Application.mk*, który jest konieczny do kompilacji aplikacji z natywnym kodem wykorzystującym bibliotekę OpenCV, z powodu stosowania wyjątków oraz standardowej biblioteki szablonów w kodzie C++.

Plik źródłowy aplikacji w języku C++ zawiera w sobie kod odwołujący się do biblioteki OpenCV w sposób natywny.

Jego struktura wygląda następująco:

- Dyrektywy #include załączające pliki nagłówkowe modułów biblioteki wykorzystywane w danej aplikacji, np. *#include <opencv2/core/core.hpp>*
- Określenie przestrzeni nazw wykorzystywanych w pliku cpp programu. Standardowo dołącza się przestrzeń nazw *std* za pomocą polecenia *using namespace std;*, a także dołącza się przestrzeń nazw *cv* odpowiedzialną za nazwy stosowane w bibliotece OpenCV – w tym przypadku zadanie to realizuje polecenie *using namespace cv;*
- Odwołanie JNIEXPORT łączące odpowiednią funkcję realizowaną w pliku cpp z aplikacją w języku Java. Cała linia odwołania wygląda w sposób następujący: JNIEXPORT *typ danych* JNICALL Java*_nazwa paczki aplikacji Android_odniesienie do konkretnej funkcji*

- Funkcje biblioteki realizowane za pomocą kodu C++

Zawartość plików cpp aplikacji wykorzystywanych w pracy zostanie omówiona przy szczegółowej prezentacji poszczególnych programów w kolejnym rozdziale.

Rozdział 4

Analiza porównawcza działania wybranych elementów biblioteki OpenCV na urządzeniach mobilnych

Utworzone aplikacje mobilne miały za zadanie zademonstrować możliwości biblioteki OpenCV w wersji skompilowanej dla systemu operacyjnego Android. Drugim celem przygotowanych aplikacji był pomiar wydajności działania biblioteki OpenCV uruchomionej na kilku dostępnych urządzeniach mobilnych działających pod kontrolą systemu operacyjnego Android. Wykorzystane urządzenia należą do grupy smartfonów oraz tabletów, a także różnią się parametrami technicznymi zastosowanych podzespołów oraz zainstalowanymi różnymi wersjami systemu Android. Do testów wydajności wybrano algorytmy odpowiadające za następujące funkcje: nakładanie filtru medianowego na obraz przekazywany z kamery urządzenia, wykonywanie progowania adaptacyjnego na obrazie z kamery urządzenia, wykrywanie twarzy obecnych na obrazie z kamery urządzenia oraz mechanizm detekcji cech na obrazie z kamery. Powyższe algorytmy zostały wybrane z powodu ich dużych możliwości zastosowania w urządzeniach mobilnych w wielu codziennych czynnościach. Sprawdzenie wydajności działania powyższych algorytmów jest bardzo dobrą demonstracją sposobu funkcjonowania biblioteki OpenCV na platformie Android. Pomiary wykonywano poprzez umieszczanie obiektywu kamery urządzenia na przybliżonej ustalonej wysokości nad wydrukowanym obrazem testowym. Powyższa metoda zbierania pomiarów została wybrana, gdyż pozwalała na proste przygotowanie aplikacji testowych. Nie było konieczne tworzenie interfejsów użytkownika ani dodawanie zbędnych z punktu widzenia pracy opcji. Każdy program stanowi proces wyświetlający obraz bezpośrednio z kamery urządzenia i na nim wykonywane są wszystkie operacje.

4.1. Wykorzystany sprzęt mobilny

Do testów biblioteki OpenCV wybrano zestaw 4 urządzeń działających pod kontrolą systemu Android i należących zarówno do grupy smartfonów jak i tabletów. Urządzenia te dobrano w taki sposób aby różniły się parametrami technicznymi zastosowanych podzespołów wewnętrznych oraz wersją zainstalowanego systemu Android.

Urządzenia, które wybrano do testów OpenCV:

1. Dwa egzemplarze smartfonu Samsung Galaxy S2

Źródło: strona internetowa producenta

Najważniejsze elementy specyfikacji tego urządzenia [12]:

- Procesor dwurdzeniowy Dual-Core Cortex-A9 1.2GHz
- Aparat fotograficzny 8 Mpix, posiadający doświetlającą diodę LED oraz nagrywający filmy w rozdzielczości 1920x1080 pikseli (Full HD)
- Wyświetlacz pojemnościowy o rozmiarze 4,3", rozdzielczości 480 x 800 pikseli oraz wyświetlający 16 milionów kolorów
- System operacyjny Android w wersji 2.3.3 oraz 4.0.4

Testy biblioteki OpenCV wykonano w systemie Android w wersji 2.3.3 oraz 4.0.4 w celu sprawdzenia czy różne wersje systemu operacyjnego mają wpływ na wydajność działania biblioteki na tym urządzeniu. Rozdzielczość obrazu podlegającego obróbce w aplikacjach testowych jest zgodna z rozdzielczością wyświetlacza i wynosi 480 x 800 pikseli.

2. Tablet Asus Eee Pad Transformer TF101

Źródło: http://www.chip.pl/testy/sprzet-mobilny/tablety/asus-eee-pad-transformer-tf101

Najważniejsze elementy specyfikacji tego urządzenia [13]:

- Procesor dwurdzeniowy NVIDIA Tegra 2 1,0 GHz
- 5 Megapikselowa kamera nagrywająca filmy w rozdzielczości Full HD.
- 10.1" ekran pojemnościowy wyświetlający obraz o rozdzielczości 1280x800 pikseli, wykonany w technologii IPS
- System operacyjny Android w wersji 3.0/ Android 4.0.3 po aktualizacji

Testy biblioteki OpenCV wykonano w systemie Android 3.0 oraz Android 4.0.3 w celu sprawdzenia, czy inna wersja systemu operacyjnego może mieć wpływ na wydajność biblioteki OpenCV. Rozdzielczość obrazu podlegającego obróbce w aplikacjach testowych jest zgodna z rozdzielczością wyświetlacza i wynosi 1280 x 800 pikseli.

3. Tablet GoClever TAB A73

Źródło: strona internetowa producenta

Najważniejsze parametry techniczne tego urządzenia [14]:

- Procesor jednordzeniowy Allwinner A10 1GHz oparty na architekturze Cortex A8 z zintegrowanym układem graficznym GPU Mali400.
- Kamera przednia VGA CMOS 0.3 Mpix
- Wyświetlacz pojemnościowy 7" LCD TFT o rozdzielczości 800x480 pikseli
- System operacyjny Android w wersji 4.0.3

Z powodu braku dostępności nowszej wersji systemu Android na posiadany egzemplarz urządzenia, testy biblioteki OpenCV wykonano wyłącznie w systemie Android w wersji 4.0.3. Rozdzielczość obrazu podlegającego obróbce w aplikacjach testowych jest zgodna z rozdzielczością wyświetlacza i wynosi 800 x 480 pikseli.

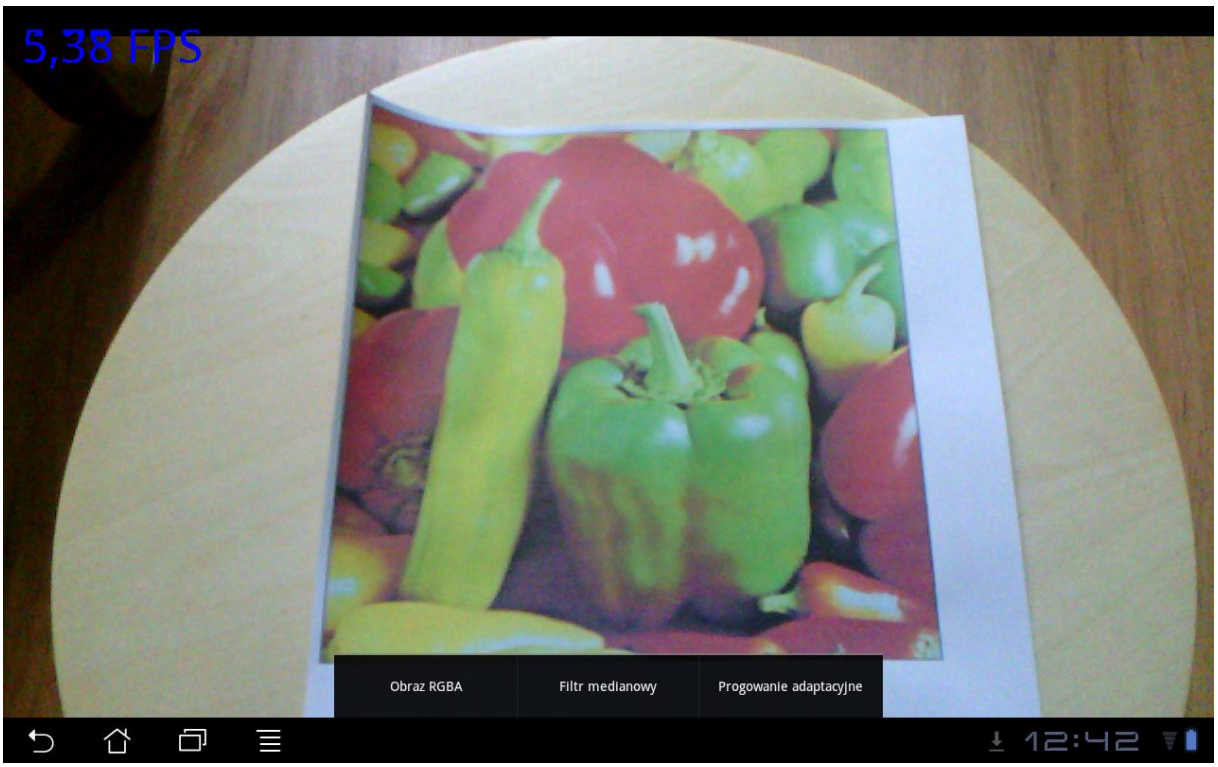
4.2. Badanie wydajności działania aplikacji pierwszej - Opis programu

Pierwsza aplikacja mobilna miała za zadanie przetestować wydajność biblioteki OpenCV podczas nakładania filtru medianowego na obraz pobierany na bieżąco z kamery urządzenia oraz podczas wykonywania progowania adaptacyjnego tego obrazu. Testy można wykonać niezależnie poprzez wybór pożądanej operacji przetwarzania obrazów w menu podręcznym aplikacji. Dzięki swojej prostej strukturze program nie wymaga długotrwałego wczytywania żadnych składników zaraz po uruchomieniu. Cała aplikacja jest zintegrowana z usługą OpenCV Manager, co spowodowało, że aplikacja nie zawiera dołączonej biblioteki OpenCV oraz waży zaledwie kilkaset kilobajtów.

Istnieje możliwość dowolnego ustawienia parametrów wykonywanych operacji przetwarzania obrazów poprzez pożądanej wartości i wstawienie jej w odpowiednie miejsce w kodzie źródłowym aplikacji. Pozwala to na dostosowanie efektów działania zaimplementowanych operacji do własnych potrzeb.

Nieskomplikowana budowa aplikacji umożliwia bezproblemowe rozszerzenie jej o nowe funkcjonalności dostępne w bibliotece OpenCV w wersji dla systemu Android. W ten sposób do programu można dołączyć inne rodzaje filtrów, maski wykrywające krawędzie, a także funkcjonalność wykrywania twarzy lub detekcji cech.

Testy wydajności działania aplikacji są przeprowadzane za pomocą wbudowanego miernika liczby klatek na sekundę. Pozwala on uzyskać aktualizowany na bieżąco wynik, który w sposób prosty obrazuje wydajność urządzenia, na którym został uruchomiony.



Rysunek 4.1. Widok aplikacji numer 1 wraz z menu wyboru dostępnych opcji

4.2.1. Moduły aplikacji

Cała aplikacja składa się z czterech plików Java zawierających kod źródłowy. Zastosowany podział pozwala na łatwe rozpoznanie, które funkcjonalności są zawarte w każdym pliku. Funkcje biblioteki OpenCV realizujące operacje przetwarzania obrazów są zlokalizowane w jednym pliku, a dodanie nowych sprowadza się do dołączenia nowej opcji menu i dopisania kolejnego warunku *case* odnoszącego się do tej opcji.

Omawiana aplikacja składa się z następujących modułów:

- Moduł odpowiedzialny za aktywność aplikacji. Plik z kodem źródłowym tego modułu nosi nazwę ImageManipulationsActivity.java. Zawarte w nim polecenia zajmują się tworzeniem menu wyboru opcji, wczytywaniem biblioteki OpenCV (poprzez powiązanie z usługą OpenCV Manager) oraz tworzeniem okien dialogowych informujących o błędach.
- Moduł odpowiedzialny za przeprowadzanie operacji przetwarzania obrazów. Plik z kodem źródłowym dla tego modułu nazwano ImageManipulationsView.java. Zawarte są tu polecenia tworzące widoczny na ekranie urządzenia obraz po dokonaniu operacji przetwarzania obrazów. Najważniejszym elementem tego modułu jest instrukcja

switch, która za pomocą bloków *case* pozwala na wybór konkretnej metody przetwarzania obrazów. Każdy blok *case* zawiera instrukcje odpowiedzialne za wybrany sposób prezentacji obrazu.

- Moduł odpowiedzialny za działanie miernika liczby klatek na sekundę. Plik z kodem źródłowym dla tego modułu nazwano FpsMeter.java. Zadeklarowano tutaj zmienne wykorzystywane do działania miernika oraz metody inicjujące działanie licznika oraz algorytm zajmujący się dokonywaniem pomiarów. Na koniec pliku zaimplementowano kod odpowiedzialny za wyświetlanie miernika klatek na sekundę na ekranie urządzenia.
- Moduł odpowiedzialny za obsługę kamery urządzenia mobilnego, zarządzanie interfejsem aplikacji oraz powiązaniem całości z miernikiem liczby klatek na sekundę. Plik z kodem źródłowym dla tego modułu nazwano SampleCvViewBase.java. Moduł ten rozpoczęto deklaracją zmiennych odpowiedzialnych za obsługę kamery, interfejsu aplikacji oraz miernika liczby klatek na sekundę. Następnie kod źródłowy tego modułu zawiera metody, których zadaniem jest obsługa kamery. Czynność ta składa się na 3 etapy: otworzenie kamery, uwolnienie jej oraz ustawienie parametrów pracy. W tym module następuje wybór optymalnej rozdzielczości obrazu z kamery urządzenia. Na koniec umieszczono metody zarządzające interfejsem aplikacji oraz inicjującą pobieranie kolejnych klatek obrazu z kamery.

4.2.2. Główny algorytm aplikacji

Na główny algorytm aplikacji składają się poniższe instrukcje:

```
@Override
    protected Bitmap processFrame(VideoCapture capture) {
        switch (ImageManipulationsActivity.viewMode) {

    case ImageManipulationsActivity.VIEW_MODE_RGBA:
            capture.retrieve(mRgba, Highgui.CV_CAP_ANDROID_COLOR_FRAME_RGBA);
            break;

    case ImageManipulationsActivity.VIEW_MODE_MEDIAN:
            capture.retrieve(mRgba, Highgui.CV_CAP_ANDROID_COLOR_FRAME_RGBA);
                    if (mRgbaInnerWindow == null|| mGrayInnerWindow == null)
                CreateAuxiliaryMats();
                    long start = System.currentTimeMillis();
                    Imgproc.medianBlur(mRgbaInnerWindow, mIntermediateMat, 19);
             Core.convertScaleAbs(mIntermediateMat, mIntermediateMat, 1./10, 0);
             Core.convertScaleAbs(mIntermediateMat, mRgbaInnerWindow, 10, 0);
                    break;

        case ImageManipulationsActivity.VIEW_MODE_ATHRESHOLD:
             capture.retrieve(mRgba, Highgui.CV_CAP_ANDROID_COLOR_FRAME_RGBA);
             capture.retrieve(mGray, Highgui.CV_CAP_ANDROID_GREY_FRAME);
```

```
                    if (mRgbaInnerWindow == null|| mGrayInnerWindow == null)
                CreateAuxiliaryMats();
Imgproc.adaptiveThreshold(mGrayInnerWindow, mIntermediateMat, 255,
Imgproc.ADAPTIVE_THRESH_MEAN_C, Imgproc.THRESH_BINARY, 15, 4 );

Core.convertScaleAbs(mIntermediateMat, mIntermediateMat, 10, 0);
Imgproc.cvtColor(mIntermediateMat, mRgbaInnerWindow, Imgproc.COLOR_GRAY2BGRA, 4);
                    break;
    }

        Bitmap bmp = Bitmap.createBitmap(mRgba.cols(), mRgba.rows(),
Bitmap.Config.ARGB_8888);

        try {
             Utils.matToBitmap(mRgba, bmp);
            return bmp;
        } catch(Exception e) {
             Log.e("org.opencv.samples.puzzle15", "Utils.matToBitmap() throws an
exception: " + e.getMessage());
            bmp.recycle();
            return null;
        }
    }
```

Kod wykonujący operacje przetwarzania obrazów ma prostą budowę. Jego całość zawarta jest w module ImageManipulationsView. Zamieszczono tutaj deklaracje zmiennych określających macierze wykorzystywane do pracy biblioteki OpenCV. Następna w kolejności jest inicjalizacja metody z grupy interfejsu użytkownika, w ramach której następuje stworzenie nowych macierzy przechowujących przetwarzany obraz. Później zadeklarowano metodę *CreateAuxiliaryMats()*, w ramach której następuje ustawienie rozmiarów domyślnej macierzy *mRgba*, w której składowany jest obraz bezpośrednio pobrany z kamery urządzenia. Po powyższej czynności następuje zadeklarowanie macierzy pomocniczych, których zadaniem jest przechowywanie obrazu po przetworzeniu jedną z operacji i o rozmiarze mniejszym od rozmiaru ekranu urządzenia mobilnego.

W tym momencie w kodzie źródłowym zaimplementowano wybrane funkcje przetwarzania obrazów, których algorytm należy do metody process*Frame*. Kod źródłowy tej metody przedstawiono powyżej. Konstrukcja tego algorytmu powiązana jest z modułem aktywności, gdzie zaimplementowano kod odpowiedzialny za stworzenie prostego menu pozwalającego na wybór konkretnej opcji, które jest dostępne pod systemowym przyciskiem menu opcji.

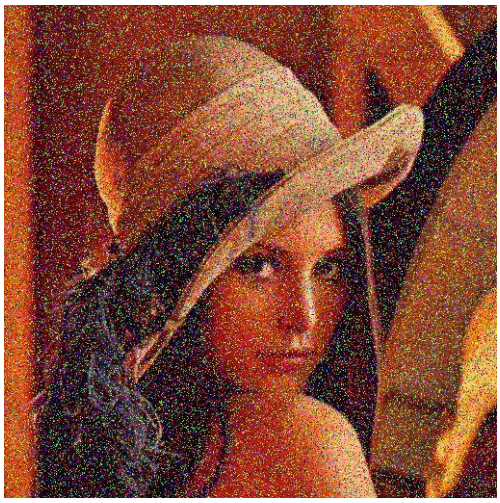
Do nałożenia filtru medianowego wykorzystano polecenie *Imgproc.medianBlur*. Aby wykonać progowanie adaptacyjne na obrazie pobieranym z kamery urządzenia, zastosowano instrukcję *Imgproc.adaptiveThreshold*. Wybór pożądanej opcji z menu aplikacji zakańczany jest instrukcją *break*, która blokuje dalsze wykonywanie bloku *case*. Dalsza część algorytmu

sprowadza się do kodu realizującego przekształcenie obrazu z macierzy do postaci gotowej do wyświetlenia na ekranie. Na koniec następuje zwolnienie wszystkich macierzy.

4.2.3. Wyniki pomiarów wydajności działania

Testy wydajności aplikacji numer jeden przeprowadzono osobno dla operacji nakładania filtru medianowego i dla operacji progowania adaptacyjnego. Każdy wynik pomiaru był odczytywany trzykrotnie a na koniec wyliczono jego średnią. Czynności te były powtórzone dla każdego urządzenia.

Wyniki testów działania operacji nakładania filtru medianowego.



Rysunek 4.2 Przykładowy obraz testowy

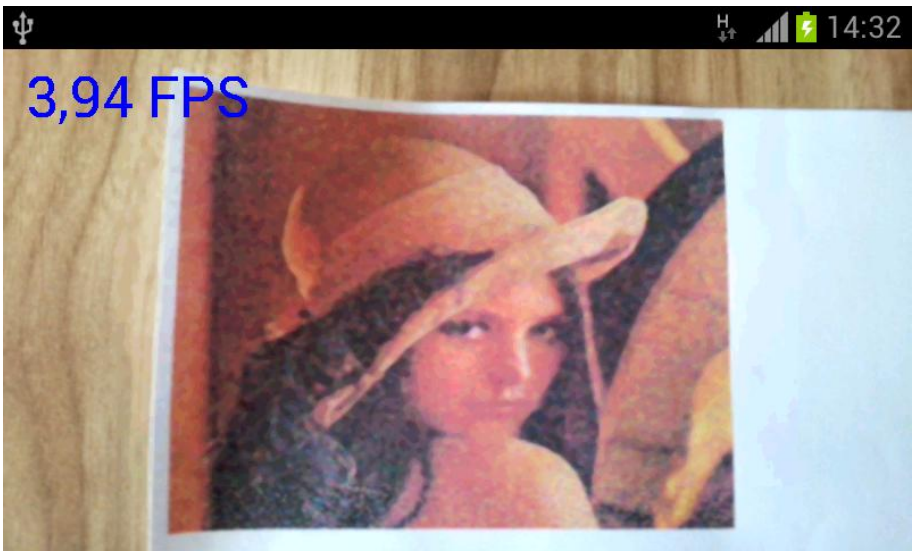
Badanie filtru medianowego wykonano na obrazie „smallena.jpg” (rys. 4.2) zaszumionym za pomocą szumu typu „sól i pieprz”. Zmianom podlegała wielkość elementu strukturującego – wynosiła ona odpowiednio 3x3, 5x5 oraz 21x21. Następnym krokiem było nakierowanie obiektywu kamery aktualnie wykorzystywanego urządzenia na wydrukowany obraz testowy. Wydajność aplikacji sprawdzono na bazie wbudowanego miernika liczby klatek na sekundę dla konfiguracji urządzeń: Samsung Galaxy S2 z systemem Android 4.0.4, Asus Transformer TF101 z systemem Android 3.0 oraz Goclever Tab A73. Po zakończeniu wszystkich pomiarów sprawdzono, czy zmiana systemu z wersji 4.0.4 na starszą 2.3.3 w przypadku smartfonu Galaxy S2 oraz z systemu w wersji 3.0 na 4.0.3 na tablecie Transformer miała wpływ na wydajność pracy aplikacji. W tabeli 4.1 przedstawiono wyniki pomiarów wydajności działania algorytmu filtru medianowego na urządzeniach wykorzystywanych do badań. Otrzymane rezultaty dotyczą pomiarów wydajności aplikacji pracującej na każdym urządzeniu w jego natywnej rozdzielczości. Zawartość wiersza tabeli oznaczonego „Obraz

bez modyfikacji” określa wydajność aplikacji przy wyłącznym pobieraniu obrazu z kamery i jego bezpośrednim wyświetlaniu na ekranie urządzenia.

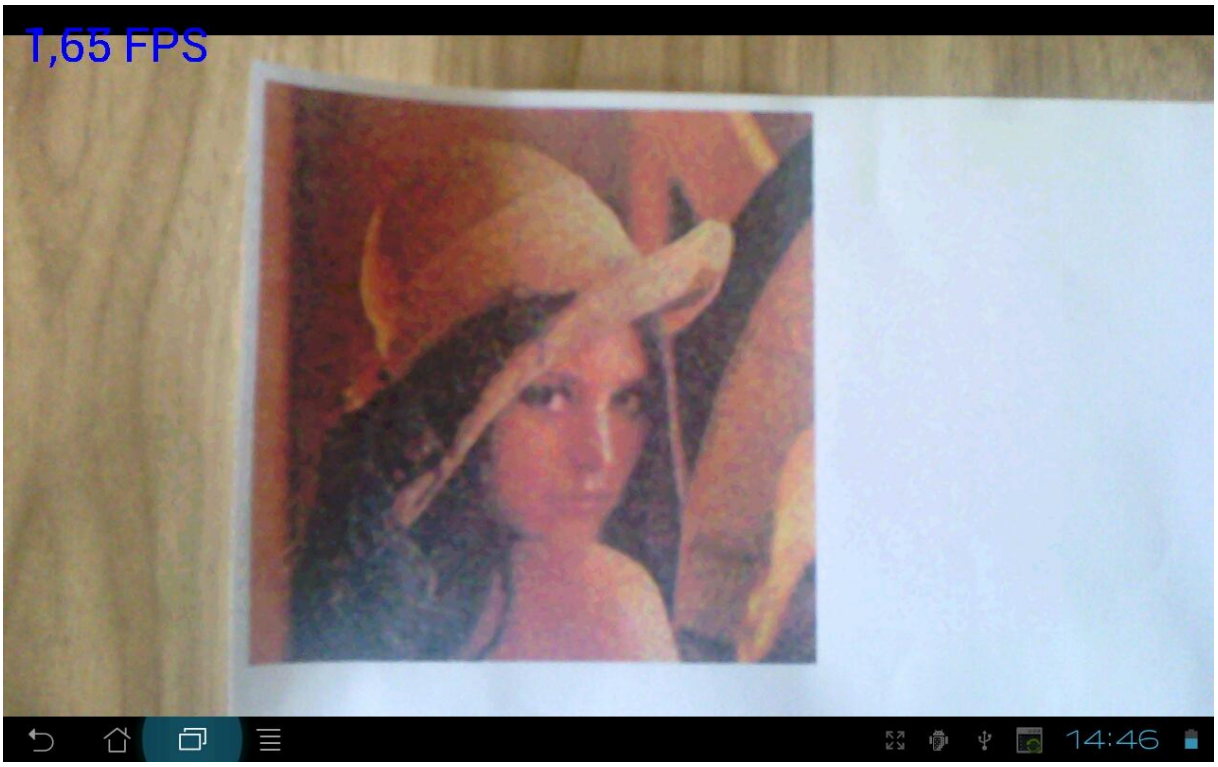
	Samsung Galaxy S2	Asus Transformer TF101	Goclever Tab A73
Obraz bez modyfikacji	11	5	7,2
Element strukt. 3x3	5,3	2,9	1,5
Element strukt. 5x5	3,9	1,7	1,3
Element strukt. 21x21	2,1	1	1,2

Tabela 4.1: Wyniki testów filtru medianowego, podane w kl/s

Jak widać, najwydajniejszym urządzeniem jest Samsung Galaxy S2. Rezultaty działania testowanej operacji przedstawiono na rys. 4.3a oraz 4.3b. Filtr medianowy jest operacją zużywającą dużo zasobów obliczeniowych procesorów co powoduje znaczy spadek liczby wyświetlanych klatek w trakcie jednej sekundy. Dodatkowo rozmiar elementu strukturującego miał wpływ na wydajność działania aplikacji. Największy spadek wydajności był zauważalny w przypadku elementu strukturującego o rozmiarze 9x9. W każdym wykorzystanym urządzeniu różnica w wydajności była znaczna. Niewielka różnica w wydajności działania aplikacji na urządzeniach firm Asus oraz Goclever może być spowodowana niską jakością zamontowanych układów optycznych w kamerach a także słabą optymalizacją biblioteki OpenCV pod układ obliczeniowy Nvidia Tegra 2. Układ ten nie jest bezpośrednio rozpoznawany przez usługę OpenCV Manager. Identyfikowany jest jako układ bazujący na architekturze ARMv7. Zastosowany filtr medianowy bezproblemowo usunął szum typu „sól i pieprz” z obrazu testowego, na którym gęstość tego zakłócenia wynosiła 0.2. Wadą filtru medianowego stosowanego w aplikacji jest (w przypadku zastosowania dużego elementu strukturującego) znaczne rozmycie konturów obiektu widocznego na obrazie.



Rysunek 4.3a: Wynik działania filtracji medianowej na obrazie testowym – smartfon Samsung Galaxy S2



Rysunek 4.3b: Wynik działania filtracji medianowej na obrazie testowym – tablet Asus Transformer

Wykonanie testów działania aplikacji po zmianie systemu operacyjnego na wersję 2.3.3 w przypadku smartfonu Galaxy S2 nie wykazało znacznej różnicy w szybkości działania algorytmu filtru medianowego, natomiast obraz wyświetlany bez żadnych modyfikacji cechował się liczbą ok 30 klatek na sekundę. Testy działania przeprowadzone na tablecie firmy Asus po zmianie systemu na wersję 4.0.3 nie wykazały zmiany wydajności działania algorytmu.

Wyniki testów wydajności działania algorytmu progowania adaptacyjnego

Rysunek 4.4: Obraz testowy

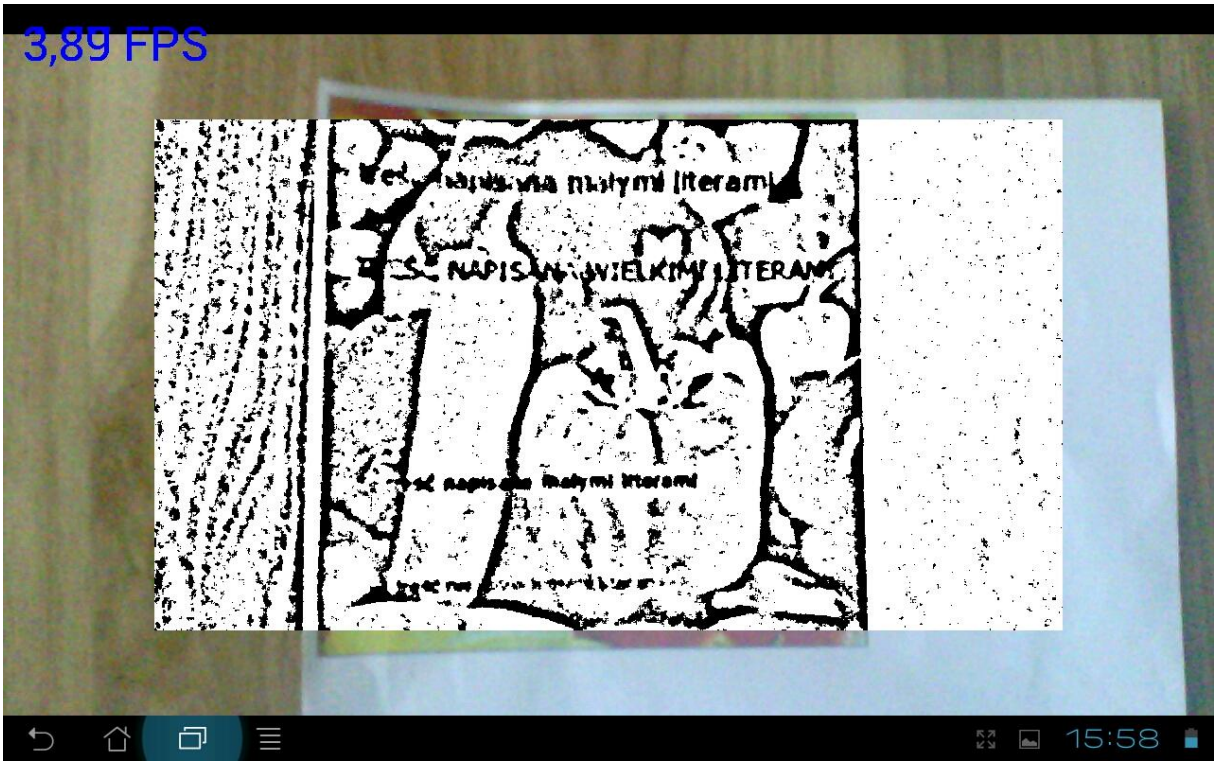
Badanie algorytmu progowania adaptacyjnego przeprowadzono na testowym obrazie (rys. 4.4) z tekstem napisanym literami o różnych wielkościach. Parametrem, który podlegał zmianie jest rozmiar rozważanego przez filtr otoczenia. Do badań zdecydowano się wykorzystać rozmiary 15x15, 19x19 oraz 23x23. Pierwszym krokiem było nakierowanie kamery na obraz testowy. Pomiary wydajności przeprowadzono, podobnie jak przy testach filtru medianowego przy pomocy wbudowanego miernika liczby klatek na sekundę. Konfiguracja urządzeń testowych była identyczna z tą w poprzednim teście. W tabeli 4.2 przedstawiono wyniki pomiarów wydajności algorytmu progowania adaptacyjnego dla wykorzystanych urządzeń. Otrzymane rezultaty dotyczą aplikacji działającej na każdym urządzeniu w jego natywnej rozdzielczości. Badana metoda progowania adaptacyjnego to *Adaptive_Thresh_Mean_C* a rodzajem progowania jest progowanie binarne. Zawartość wiersza tabeli oznaczonego „Obraz bez modyfikacji” określa wydajność aplikacji przy wyłącznym pobieraniu obrazu z kamery i jego bezpośrednim wyświetlaniu na ekranie urządzenia.

	Samsung Galaxy S2	Asus Transformer TF101	Goclever Tab A73
Obraz bez modyfikacji	11	5,5	7,2
Otoczenie 15x15	9	4,2	4,7
Otoczenie 19x19	7	4,4	4,5
Otoczenie 23x23	9,3	4,2	4,7

Tabela 4.2: Wyniki testów operacji progowania adaptacyjnego, podane w kl/s

Obserwując powyższe wyniki można łatwo wywnioskować, że algorytm progowania adaptacyjnego nie obciąża w znacznym stopniu układu obliczeniowego. Dla każdego wykorzystanego urządzenia różnica wydajności była niewielka. Dwukrotne zwiększenie rozmiaru rozważanego przez filtr otoczenia również nie spowodowało zmian w szybkości działania algorytmu. W przypadku tabletu Asus maksymalna różnica wyniosła w przybliżeniu jedną klatkę na sekundę. Zaobserwowano, że na efektywność działania operacji progowania adaptacyjnego ma wpływ rodzaj tła, z którego odczytywana jest treść. Dodatkowo duży wpływ na wyrazistość tekstu na progowanym adaptacyjnie obrazie ma wpływ jakość kamery zamontowanej w urządzeniu. Najlepsze rezultaty otrzymano podczas badań za pomocą smartfonu Samsung Galaxy S2. Urządzenie to cechuje najlepsza jakość obrazu oraz wideo w wysokiej rozdzielczości możliwa dzięki połączeniu matrycy o dobrych parametrach oraz wydajnego układu obliczeniowego.

Rysunek 4.5a: Wynik działania operacji progowania adaptacyjnego na obrazie testowym – smartfon Samsung Galaxy S2



Rysunek 4.5b: Wynik działania operacji progowania adaptacyjnego na obrazie testowym – tablet Asus Transformer

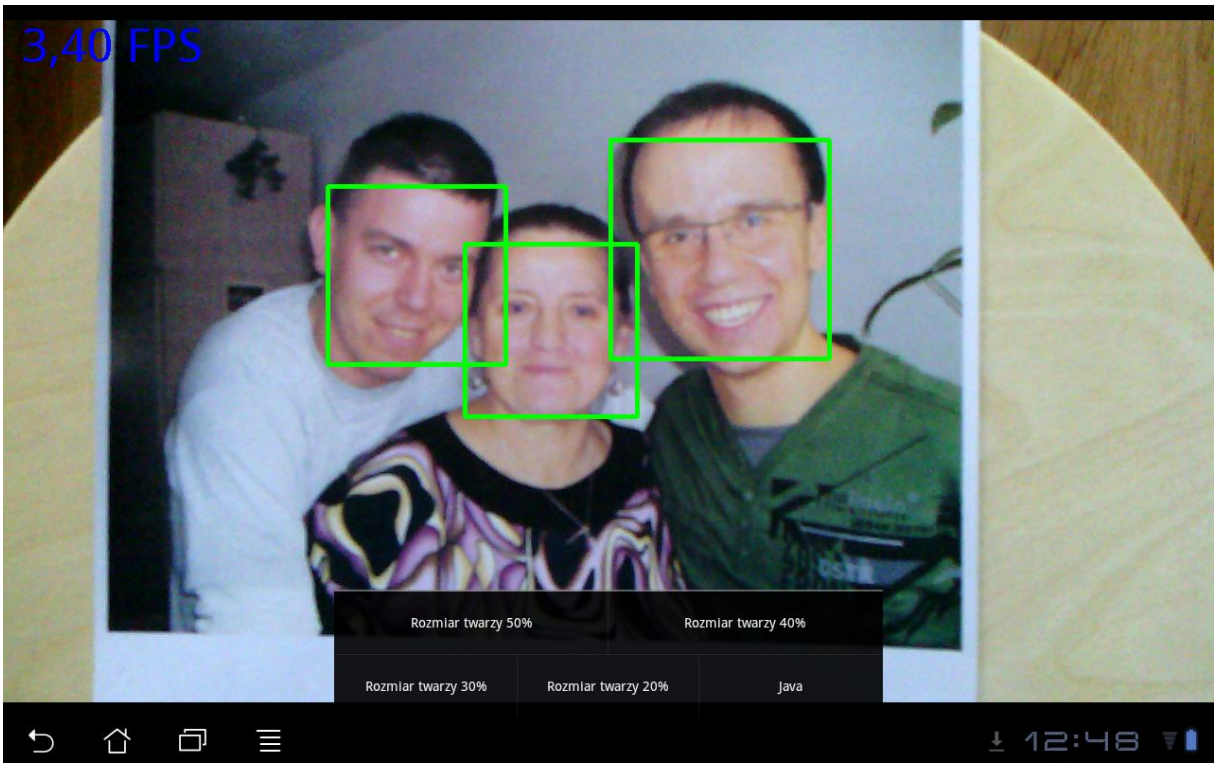
Wykonanie testów wydajności aplikacji po zmianie systemu operacyjnego smartfonu Samsung Galaxy S2 na wersję 2.3.3 przyniosło nieoczekiwane rezultaty. Liczba klatek na sekundę wzrosła przy badaniu progowania adaptacyjnego do wartości 14. Może być to spowodowane wewnętrzną budową starszej wersji systemu operacyjnego Android, która w mniejszym stopniu wykorzystuje zasoby urządzenia. Zmiana systemu na Android 4.0.3 w tablecie Asus Transformer nie przyniosła znaczących zmian w wydajności działania algorytmu progowania adaptacyjnego.

4.3. Badanie wydajności działania aplikacji drugiej - Opis programu

Druga aplikacja ma na celu demonstrację możliwości algorytmu wykrywania twarzy oraz przetestowanie wydajności jego działania na dostępnych urządzeniach z systemem Android. Podstawowe testy wykonuje się poprzez odczyt wyników miernika aktualizowanych na bieżąco na ekranie urządzenia. W aplikacji do detekcji twarzy zastosowano metodę programowania natywnego. Oznacza to, że główny szkielet aplikacji jest napisany w języku Java, natomiast niektóre elementy programu zostały przygotowane w języku C++. Omawiany program pozwala na skorzystanie z algorytmu detekcji twarzy zarówno za pomocą funkcji

wykrywania napisanej w języku Java, jak i za pomocą kodu natywnego w języku C++. Cała aplikacja, podobnie jak program pierwszy, jest zintegrowana z usługą OpenCV Manager, co pozwoliło na ograniczenie rozmiaru programu w pamięci wewnętrznej urządzenia.

W prezentowanej aplikacji użytkownik może w łatwy sposób dostosowywać parametry pracy algorytmu detekcji. Do jego dyspozycji oddano możliwość regulacji czułości wykrywania dużej liczby twarzy dostępnych na obrazie pobieranym z kamery urządzenia. Ustawienia te są dokonywane poprzez wybór wielkości twarzy obecnej na obrazie, która jest wyrażona w procentach. Dodatkowo, bezpośrednio w programie, użytkownik może zdecydować, czy chce korzystać z mechanizmu detekcji twarzy przygotowanego w języku Java, czy z algorytmu wykorzystującego programowanie natywne. Wszystkie wymienione powyżej funkcje są dostępne w menu opcji aplikacji.



Rysunek 4. 6: Widok aplikacji numer 2 wraz z menu wyboru dostępnych opcji

4.3.1. Moduły aplikacji

Omawiany program składa się z 5 modułów, będących kodem źródłowym aplikacji napisanym w języku Java oraz z 1 modułu stworzonego w języku C++, który stanowi element natywny oprogramowania. Dodatkowo do kodu natywnego dołączony jest niezbędny plik nagłówkowy, którego znaczenie wyjaśniono w dalszej części pracy.

Wymienione powyżej składniki to:

- Moduł odpowiedzialny za działanie miernika liczby klatek na sekundę - *FpsMeter.java*. Jego działanie jest identyczne do funkcjonowania tego miernika w aplikacji pierwszej. Jego opis znajduje się w podrozdziale „Moduły aplikacji" traktującym o programie numer jeden.
- Moduł odpowiedzialny za obsługę kamery urządzenia, interfejs aplikacji oraz za powiązanie całego kodu z miernikiem liczby klatek na sekundę - *SampleCvViewBase.java*. Jego działanie jest takie samo jak w przypadku programu pierwszego. Jego opis także został przedstawiony w podrozdziale poświęconym modułom aplikacji numer jeden.
- Moduł odpowiedzialny za zarządzanie aktywnością aplikacji. Plik z kodem źródłowym nazwano *FdActivity.java*. W pierwszej kolejności w pliku zadeklarowano zmienne określające możliwe do wyboru opcje detekcji twarzy na obrazie z kamery. Następnie w kodzie zaimplementowano metody odpowiadające za powiązanie z usługą OpenCV i za podstawową obsługę błędów. Na koniec modułu umieszczono instrukcje, których celem jest stworzenie menu opcji oraz powiązanie ich z konkretną funkcjonalnością programu.
- Moduł odpowiedzialny za przeprowadzanie detekcji twarzy napisany w języku Java. Plik z kodem źródłowym nazwano *FdView.java*. Całość rozpoczęto od deklaracji zmiennych określających macierze przechowujące obraz z kamery oraz dwa typy detekcji wykorzystywane w aplikacji. Zaimplementowano tutaj możliwość selekcji metody wykrywania twarzy. Dalsza część modułu to instrukcje programujące działanie mechanizmu wykrywania twarzy za pomocą algorytmu przygotowanego w języku Java. Omawiany moduł został szerzej opisany w podrozdziale „Główny algorytm aplikacji".
- Moduł odpowiedzialny za połączenie kodu przygotowanego w języku Java z algorytmem detekcji twarzy wykonanym w technice programowania natywnego. Plik z kodem źródłowym nazwano *DetectionBasedTracker.java*. Zawarto tu metody odwołujące się bezpośrednio do instrukcji zawartych w algorytmie C++ mechanizmu detekcji twarzy.
- Moduł przygotowany w języku C++ odpowiedzialny za natywną metodę detekcji twarzy. Zamieszczony tutaj algorytm programuje wszystkie natywne czynności wykonywane przy mechanizmie wykrywania. Niniejszy moduł opisano dokładniej w podrozdziale „Główny algorytm aplikacji".

4.3.2. Główny algorytm aplikacji

Na główny algorytm aplikacji składają się poniższe instrukcje:

```
@Override
    protected Bitmap processFrame(VideoCapture capture) {
        capture.retrieve(mRgba, Highgui.CV_CAP_ANDROID_COLOR_FRAME_RGBA);
        capture.retrieve(mGray, Highgui.CV_CAP_ANDROID_GREY_FRAME);

        if (mAbsoluteFaceSize == 0)
        {
             int height = mGray.rows();
             if (Math.round(height * mRelativeFaceSize) > 0);
             {
                    mAbsoluteFaceSize = Math.round(height * mRelativeFaceSize);
             }
             mNativeDetector.setMinFaceSize(mAbsoluteFaceSize);
        }

        MatOfRect faces = new MatOfRect();

        if (mDetectorType == JAVA_DETECTOR)
        {
             if (mJavaDetector != null)
                mJavaDetector.detectMultiScale(mGray, faces, 1.1, 2, 2
                        , new Size(mAbsoluteFaceSize, mAbsoluteFaceSize), new
Size());
        }
        else if (mDetectorType == NATIVE_DETECTOR)
        {
             if (mNativeDetector != null)
                mNativeDetector.detect(mGray, faces);
        }
        else
        {
             Log.e(TAG, "Detection method is not selected!");
        }

        Rect[] facesArray = faces.toArray();
        for (int i = 0; i < facesArray.length; i++)
            Core.rectangle(mRgba, facesArray[i].tl(), facesArray[i].br(),
FACE_RECT_COLOR, 3);

        Bitmap bmp = Bitmap.createBitmap(mRgba.cols(), mRgba.rows(),
Bitmap.Config.ARGB_8888);

        try {
             Utils.matToBitmap(mRgba, bmp);
        } catch(Exception e) {
             Log.e(TAG, "Utils.matToBitmap() throws an exception: " +
e.getMessage());
            bmp.recycle();
            bmp = null;
        }

        return bmp;
    }
```

Główny algorytm aplikacji podzielony jest na dwie części. Powyżej zaprezentowano metodę odpowiedzialną za wczytanie obrazu z kamery urządzenia oraz za mechanizm

detekcji twarzy. Cała czynność sprowadza się do zastosowania pętli typu „if…else…”. Pierwszy warunek zajmuje się sprawdzeniem rzeczywistego rozmiaru twarzy widzianej na ekranie urządzenia. Następny natomiast zawiera instrukcje odpowiedzialne za przeprowadzenie wykrywania twarzy jedną z dwóch wymienionych wcześniej metod. W przypadku ciągu instrukcji detekcji przygotowanych w języku Java, za właściwą pracę odpowiada polecenie *detectMultiScale()*, gdzie pierwszy parametr w nawiasie oznacza obraz, na którym dokonywana jest omawiana tutaj operacja, następnie wpisane są obiekty podlegające detekcji, po czym umiejscowiono wartości współczynnika skali, liczbę sąsiadów, którą każdy prostokąt wyznaczający wykrytą twarz powinien mieć oraz parametr będący pozostałością po funkcji *cvHaarDetectObjects*. Ostatnimi wartościami obecnymi w poleceniu *detectMultiScale()* są *minSize* oraz *maxSize*, które określają minimalną i maksymalną wielkość wykrywanych twarzy. Kolejne instrukcje obecne w tym module służą do przekształcenia wyników otrzymanych z wcześniejszego polecenia do postaci szyku zrozumiałego dla komendy *Core.rectangle()*, która wizualizuje je na wyświetlaczu urządzenia za pomocą wyraźnych prostokątów otaczających wykrytą twarz.

```
JNIEXPORT void JNICALL
Java_org_opencv_samples_fd_DetectionBasedTracker_nativeSetFaceSize
(JNIEnv * jenv, jclass, jlong thiz, jint faceSize)
{
    try
    {
    if (faceSize > 0)
    {
        DetectionBasedTracker::Parameters DetectorParams = \
        ((DetectionBasedTracker*)thiz)->getParameters();
        DetectorParams.minObjectSize = faceSize;
        ((DetectionBasedTracker*)thiz)->setParameters(DetectorParams);
    }

    }
    catch(cv::Exception e)
    {
    LOGD("nativeStop catched cv::Exception: %s", e.what());
    jclass je = jenv->FindClass("org/opencv/core/CvException");
    if(!je)
        je = jenv->FindClass("java/lang/Exception");
    jenv->ThrowNew(je, e.what());
    }
    catch (...)
    {
    LOGD("nativeSetFaceSize catched unknown exception");
    jclass je = jenv->FindClass("java/lang/Exception");
    jenv->ThrowNew(je, "Unknown exception in JNI code
{highgui::VideoCapture_n_1VideoCapture__()}");
    }
}


JNIEXPORT void JNICALL Java_org_opencv_samples_fd_DetectionBasedTracker_nativeDetect
(JNIEnv * jenv, jclass, jlong thiz, jlong imageGray, jlong faces)
```

```
{
    try
    {
    vector<Rect> RectFaces;
    ((DetectionBasedTracker*)thiz)->process(*((Mat*)imageGray));
    ((DetectionBasedTracker*)thiz)->getObjects(RectFaces);
    vector_Rect_to_Mat(RectFaces, *((Mat*)faces));
    }
    catch(cv::Exception e)
    {
    LOGD("nativeCreateObject catched cv::Exception: %s", e.what());
    jclass je = jenv->FindClass("org/opencv/core/CvException");
    if(!je)
        je = jenv->FindClass("java/lang/Exception");
    jenv->ThrowNew(je, e.what());
    }
    catch (...)
    {
    LOGD("nativeDetect catched unknown exception");
    jclass je = jenv->FindClass("java/lang/Exception");
    jenv->ThrowNew(je, "Unknown exception in JNI code
{highgui::VideoCapture_n_1VideoCapture__()}");
    }
}
```

Część druga głównego algorytmu to kod aplikacji napisany w języku C++. Podstawowe działanie natywnej części głównego algorytmu aplikacji opiera się na wykonywaniu czynności w ramach odwołań z głównego kodu źródłowego napisanego w języku Java. Całość instrukcji jest bezpośrednio powiązana z opisanym wcześniej plikiem *DetectionBasedTracker.java*. Każde pojedyncze działanie rozpoczyna się od linii kodu informującej o odwołaniu z głównego algorytmu źródłowego. Szczególnie istotne są odwołania *nativeSetFaceSize* oraz *nativeDetect*, które to odpowiadają za kontrolę i ustawienie wielkości twarzy widzianej w obrazie kamery oraz za główny proces detekcji. Instrukcje te przedstawiono w kodzie powyżej. W trakcie pierwszej z wymienionych czynności algorytm kontroluje wartość zmiennej *faceSize*, co w rzeczywistości oznacza sprawdzenie, czy na obrazie z kamery widoczna jest twarz. Jeśli wartość zmiennej jest dodatnia, warunek jest spełniony i następuje realizacja wewnętrznej części pętli, w której zostają pobrane i ustawione parametry detektora dostępne w dołączonym pliku źródłowym *detection_based_tracker.hpp*. Właściwy proces detekcji odbywa się w ramach odwołania *nativeDetect*. W ramach tej czynności wykrywanie odbywa się z wykorzystaniem zawartości wymienionego wyżej pliku źródłowego. W pierwszej kolejności odbywa się przetworzenie otrzymanego obrazu szarościowego, po czym następuje procedura *getObjects(RectFaces)*. Jej zadaniem jest wydobycie z obrazu twarzy według algorytmów zapisanych w dołączonym źródle. Wykryte obiekty są przechowywane w zmiennej *RectFaces*. Kolejna instrukcja odpowiada że przekazanie otrzymanych wyników ze zmiennej *RectFaces* do macierzy *Mat*, będącej głównym nośnikiem danych w opisywanym module.

4.3.3. Wyniki pomiarów wydajności działania

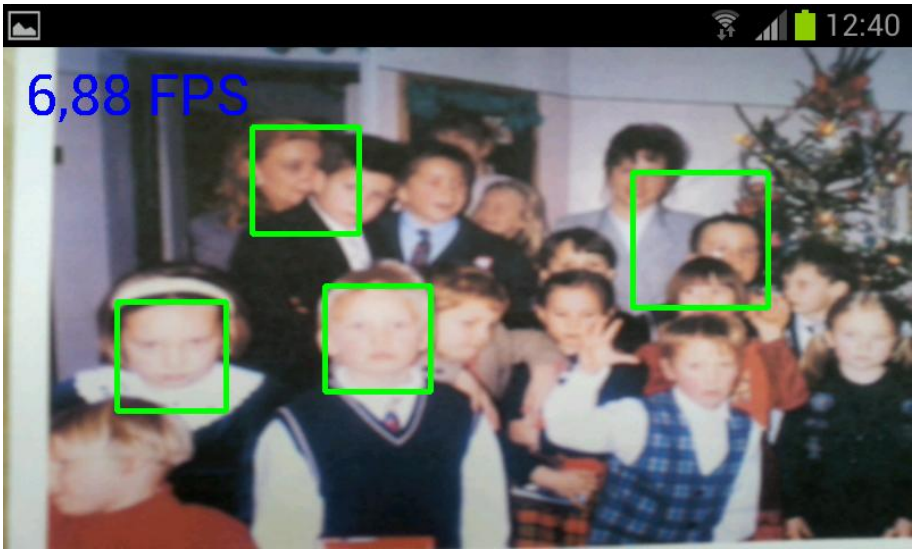
Testy algorytmu detekcji twarzy przeprowadzono na trzech zdjęciach przedstawiających różną liczbę postaci: jedną osobę, wiele osób oraz trójkę osób o twarzach skierowanych bezpośrednio w stronę obiektywu kamery. Zadaniem algorytmu było wyszukanie możliwie największej liczby twarzy na tych zdjęciach. W badaniach wykorzystano oba algorytmy detekcji zaimplementowane w aplikacji. Pomiary wydajności przeprowadzono za pomocą wbudowanego miernika liczby klatek na sekundę. Wyniki zapisywano osobno dla wykrywania metodą natywną oraz algorytmem napisanym w języku Java. Tabela 4.3 przedstawia wyniki pomiarów wydajności algorytmu detekcji twarzy. Otrzymane rezultaty dotyczą aplikacji działającej na każdym urządzeniu w jego natywnej rozdzielczości.

Rysunek 4.7: Przykładowe zdjęcie testowe

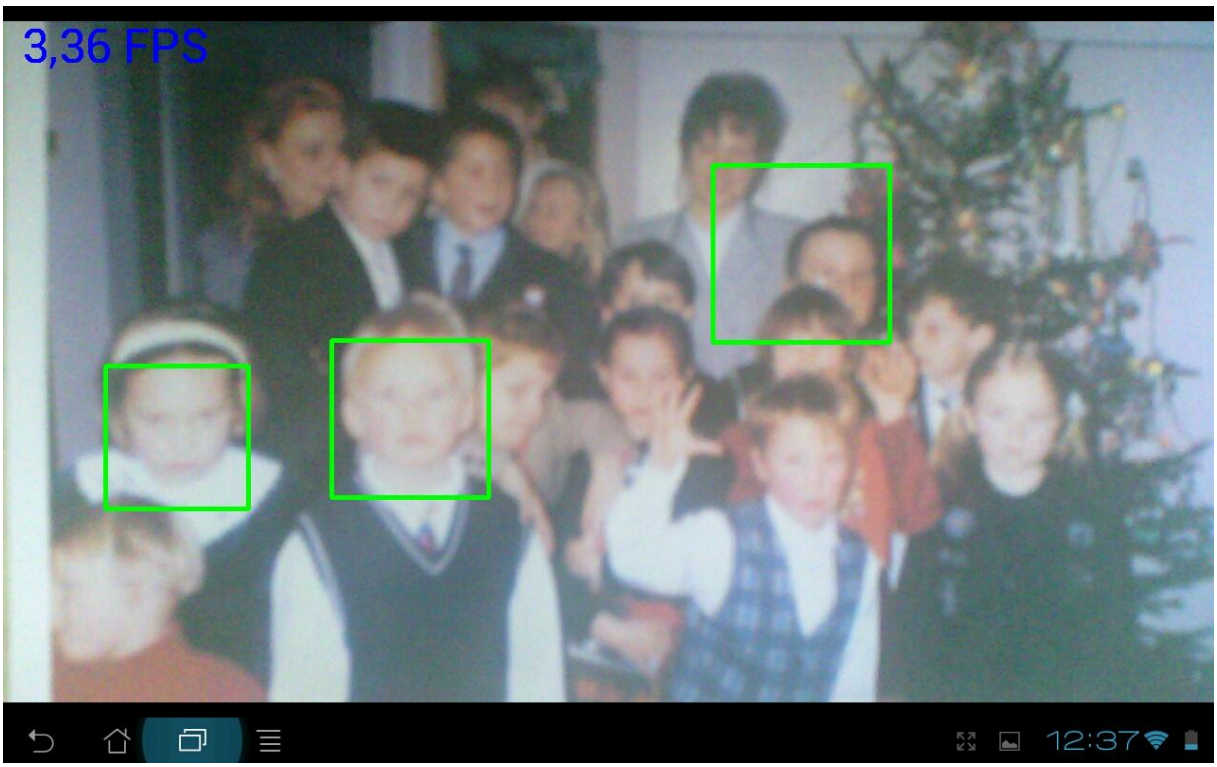
	Samsung Galaxy S2		Asus Transformer TF101		Goclever Tab A73	
	Natywnie	Java	Natywnie	Java	Natywnie	Java
Jedna twarz	5,9	11,6	4	4,9	4,8	4,8
Wiele twarzy	5,8	9,4	3,9	4,8	4,3	5
Trzy twarze	6,6	8,8	3	3,6	4,2	4,8

Tabela 4.3: Wyniki pomiarów wydajności algorytmu detekcji twarzy, podane w kl/s

Na podstawie powyższych wyników można stwierdzić, że algorytm detekcji twarzy napisany w języku Java jest wydajniejszy od tej samej procedury przygotowanej w języku C++. W przypadku smartfonu Samsung Galaxy S2 oraz tabletu Asus Transformer TF101 różnica w szybkości działania wynosi od 1 do prawie 4 klatek na sekundę. W urządzeniu Goclever Tab A73 przyrost wydajności jest mniejszy. Prawdopodobnie spowodowane jest to niską ogólną wydajnością tego tabletu. Podczas badań algorytmu wykrywania twarzy zaobserwowano że szybkość i dokładność wyszukiwania twarzy zależy od jakości kamery zamontowanej w urządzeniu. Ważnym czynnikiem wpływającym na detekcję jest sposób ustawienia i ułożenia twarzy na obrazie. Twarze skierowane wprost na obiektyw zostają wykryte szybciej i precyzyjniej. Wykonanie testów aplikacji po zmianie systemu operacyjnego nie wykazało znaczących zmian w wydajności działania algorytmu.



Rysunek 4.8a: Wynik działania operacji detekcji twarzy na zdjęciu testowym – smartfon Samsung Galaxy S2



Rysunek 4.8b: Wynik działania operacji detekcji twarzy na zdjęciu testowym – tablet Asus Transformer

4.4. Badanie wydajności działania aplikacji trzeciej - Opis programu

Trzecia aplikacja testowa zajmuje się prezentacją działania oraz badaniem wydajności algorytmów detekcji cech na urządzeniach działających pod kontrolą systemu Android. Testy te wykonuje się poprzez odczyt wskazań dwóch różnych mierników wyświetlanych na ekranie. W aplikacji wykorzystano technikę programowania natywnego. Działa ona na podobnej zasadzie jak w aplikacji zajmującej się wykrywaniem twarzy na obrazie. Podstawowy algorytm programu, którego zadaniem jest detekcja i ekstrakcja cech znajduje się w części natywnej aplikacji przygotowanej w języku C++. Podobnie jak poprzednie oprogramowanie testowe, również i niniejsza aplikacja jest zintegrowana z usługą OpenCV Manager, co pozwoliło na zachowanie możliwie najmniejszego rozmiaru aplikacji.

Omawiany program pozwala na szybką modyfikację czułości pracy mechanizmu detekcji oraz zmianę detektorów wykorzystywanych w algorytmie. Wszelkie zmiany wykonuje się bezpośrednio w kodzie źródłowym aplikacji. Czułość mechanizmu wyrażona jest liczbowo. Im wstawiona liczba jest mniejsza, tym więcej cech zostaje znalezionych na obrazie pobieranym z kamery urządzenia.

Rysunek 4.9: Widok aplikacji numer 3

4.4.1. Moduły aplikacji

Opisywana aplikacja składa się z czterech modułów będących plikami z kodem źródłowym w języku Java oraz z jednego modułu, który został przygotowany zgodnie z techniką programowania natywnego i zawiera główny algorytm przeprowadzający operację detekcji oraz ekstrakcji cech.

Do wymienionych powyżej składników należą:

- Moduł odpowiedzialny za integrację aplikacji z biblioteką OpenCV w usłudze OpenCV Manager i z biblioteką w wersji natywnej. Jego drugim zadaniem jest zarządzanie komunikatami o błędach związanych z działaniem aplikacji. Plik z kodem źródłowym nazwano *Sample3Native.java*. Moduł rozpoczęto od załadowania biblioteki OpenCV w obu wymienionych powyżej wersjach. Pozostałe instrukcje zawarte w tym pliku są zbliżone do tych w plikach zawierających aktywności w dwóch poprzednich aplikacjach omówionych we wcześniejszych podrozdziałach.
- Moduł odpowiedzialny za podstawowe operacje zajmujące się przetworzeniem obrazu na potrzeby algorytmu detekcji cech. Plik z kodem źródłowym nazwano *Sample3View.java*. Całość rozpoczęto od deklaracji zmiennych określających podstawowe macierze przechowujące dane przetwarzane przez aplikację. Wykorzystano tu zmienną *mFrameSize*, której zadaniem jest określenie rozmiaru klatki obrazu pobieranego z kamery urządzenia i wyświetlanego na ekranie.

Pozostałe dwie zmienne określają macierze zajmujące się przechowywaniem obrazu pobieranego oraz wyświetlanego na ekranie. Pierwsza metoda, *onPreviewStarted()*, definiuje jego rozmiar. Kolejnymi ważnymi metodami jest *processFrame()*, w której zawarto instrukcję FindFeatures(getFrameWidth(), getFrameHeight(), data, rgba), która wykonuje podstawową detekcję cech i stanowi połączenie z głównym algorytmem przygotowanym w języku C++.

- Moduł odpowiedzialny za zarządzanie kamerą urządzenia, obsługę interfejsu oraz powiązanie aplikacji z miernikiem liczby klatek na sekundę. Plik z kodem źródłowym nazwano *SampleViewBase.java*. Zaimplementowano tu metody zajmujące się uruchomieniem kamery, jej ustawieniem i rozpoczęciem procesu pobierania obrazu. Swoim działaniem jest on zbliżony do odpowiadających mu modułów w programie numer jeden i numer dwa.
- Moduł odpowiedzialny za działanie miernika liczby klatek na sekundę. Plik z kodem źródłowym nazwano *FpsMeter.java*. Jego działanie jest identyczne ze sposobem funkcjonowania miernika liczby klatek na sekundę w opisanych powyżej aplikacjach.
- Moduł przygotowany w języku C++, zawierający algorytm wykonujący mechanizm detekcji cech na obrazie pobieranym z kamery urządzenia. Plik z kodem źródłowym nazwano *jni_part.cpp*. Nazwa ta jest charakterystyczna dla tego typu budowy aplikacji i opiera się na dodawaniu tylko fragmentu aplikacji jako kodu natywnego. Pozostała część aplikacji, w tym wszystkie instrukcje pobierające obraz i zarządzające kamerą są napisane w języku Java. Więcej informacji na temat tego modułu przedstawiono w podrozdziale „Główny algorytm aplikacji".

4.4.2. Główny algorytm aplikacji

Na główny algorytm aplikacji składają się poniższe instrukcje:

```
JNIEXPORT void JNICALL
Java_org_opencv_samples_tutorial3_Sample3View_FindFeatures(JNIEnv* env, jobject, jint
width, jint height, jbyteArray yuv, jintArray bgra)
{
    jbyte* _yuv  = env->GetByteArrayElements(yuv, 0);
    jint*  _bgra = env->GetIntArrayElements(bgra, 0);

    Mat myuv(height + height/2, width, CV_8UC1, (unsigned char *)_yuv);
    Mat mbgra(height, width, CV_8UC4, (unsigned char *)_bgra);
    Mat mgray(height, width, CV_8UC1, (unsigned char *)_yuv);


    cvtColor(myuv, mbgra, CV_YUV420sp2BGR, 4);

    vector<KeyPoint> v;
```

```
   MserFeatureDetector  detector(12);
       double t = (double)getTickCount();
       detector.detect(mgray, v);
    for( size_t i = 0; i < v.size(); i++ )
        circle(mbgra, Point(v[i].pt.x, v[i].pt.y), 10, Scalar(0,0,255,255));
       Mat descriptors;
// Tworzenie ekstraktora
OrbDescriptorExtractor OrbDesc;
// Ekstrakcja deskryptorów
OrbDesc.compute(mgray,v,descriptors);
t = ((double)getTickCount() - t)/getTickFrequency();
char str[200];
sprintf(str, "%f s – Laczny czas operacji",t);
putText(mbgra, str, Point2f(100,100), FONT_HERSHEY_DUPLEX, 1,  Scalar(0,255,255,255));
//sprawdzanie wielkości wektora v
double s = v.size();
char str1 [300];
sprintf(str1, "%f - rozmiar wektora",s);
putText(mbgra, str1, Point2f(100,200), FONT_HERSHEY_DUPLEX, 1, Scalar(0,255,255,255));
    env->ReleaseIntArrayElements(bgra, _bgra, 0);
    env->ReleaseByteArrayElements(yuv, _yuv, 0);
}

}
```

Główny algorytm aplikacji został napisany w języku C++. Jego zadaniem jest realizacja operacji detekcji i ekstrakcji cech na obrazie wczytywanym z kamery urządzenia. Cały kod składa się z jednego odwołania pochodzącego z głównych modułów aplikacji. Ważnym elementem jest deklaracja zmiennych przechowujących przetwarzany obraz na potrzeby algorytmu. W celu ułatwienia nawigacji po kodzie, zastosowano nazwy bezpośrednio odnoszące się do przestrzeni kolorów przechowywanego obrazu. Następnym istotnym elementem jest wprowadzenie wektora cech kluczowych. Tuż po tym napisane są instrukcje wykonujące operację detekcji. Pierwsza linia tych poleceń służy do zadeklarowania pożądanego detektora cech oraz do ustawienia jego czułości. W następnej natomiast umieszczono instrukcje inicjującą funkcję detekcji na obrazie w zmiennej *mgray*, z wykorzystaniem wprowadzonego wcześniej wektora punktów kluczowych. W celu dokładniejszego zbadania wydajności tej instrukcji umieszczono ją wewnątrz funkcji timera, którego zadaniem jest pomiar czasu jej wykonywania. Wynik działania timera jest wyświetlany na bieżąco na ekranie urządzenia za pomocą komend *sprintf* oraz *putText*. Do wizualizacji wykrytych cech służy pętla, w ramach której dla ich określonego minimalnego rozmiaru rysowane są okręgi je oznaczające. Parametry okręgów można dowolnie zmieniać. Następne instrukcje zaimplementowane w algorytmie dotyczą mechanizmu ekstrakcji cech. W pierwszej kolejności zadeklarowano macierz deskryptorów. W omawianym procesie

ważną czynnością jest stworzenie ekstraktora cech, który pozwala na przeprowadzenie finalnej ekstrakcji zawartości macierzy deskryptorów.

4.4.3. Wyniki pomiarów wydajności działania

Badanie algorytmu detekcji cech przeprowadzono na dwóch obrazach wykorzystanych w testach wcześniejszych aplikacji. Celem eksperymentu było sprawdzenie wydajności działania wybranych detektorów cech na posiadanych urządzeniach. Aplikacja nie uruchomiła się na tablecie Goclever TAB A73. Wyniki pomiarów zapisywano dla dwóch wersji algorytmu. Pierwsza seria pomiarów wykonywana była na podstawie algorytmu zawierającego każdy z wybranych detektorów z wyłączoną funkcją ekstrakcji cech, natomiast w drugiej turze funkcja ta była włączona. Do badań wybrano detektory FAST, MSER i ORB oraz deskryptor ORB. Pomiary wykonano na obrazach testowych (rys. 4.10) za pomocą wbudowanego miernika liczby klatek na sekundę oraz za pomocą timera liczącego łączny czas wykonywania operacji detekcji i ekstrakcji. Aby zachować jednakową liczbę wykrytych cech kluczowych, dla każdego wariantu badań dobierano indywidualną wartość progu. Testy postanowiono wykonywać dla jedenastu cech kluczowych. Tabele 4.4 i 4.5 przedstawiają wyniki pomiarów wydajności działania algorytmu detekcji cech. Otrzymane rezultaty dotyczą aplikacji działającej na smartfonie Samsung Galaxy S2 oraz tablecie Asus Transformer TF101 w ich natywnych rozdzielczościach.

Rysunek 4.10: Wykorzystane zdjęcia testowe

		Samsung Galaxy S2					
		Fast		Mser		Orb	
		kl/s	s	kl/s	s	kl/s	s
Obraz „Smallena”	Ekstr. wył.	16	0,007	3,4	0,212	8,1	0,056
	Ekstr. wł.	10,4	0,05	3,8	0,269	4,6	0,152
Obraz „Warzywa”	Ekstr wył.	18	0,007	3,8	0,2	11,6	0,071
	Ekstr wł.	9,5	0,049	4	0,205	4,9	0,138

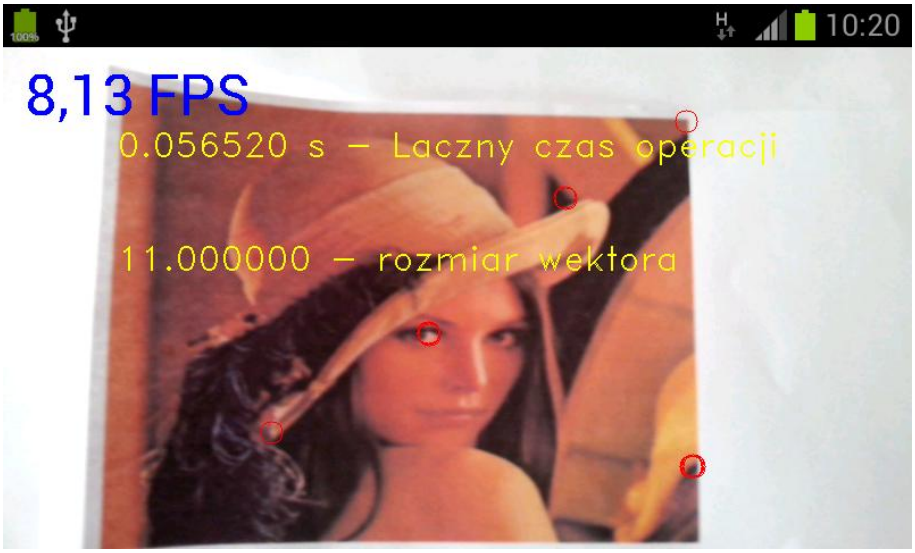
Tabela 4.4a: Wyniki pomiarów wydajności algorytmu detekcji cech na smartfonie Samsung Galaxy S2

		Asus Transformer TF101					
		Fast		Mser		Orb	
		kl/s	s	kl/s	s	kl/s	s
Obraz „Smallena”	Ekstr. wył.	5,6	0,033	1,5	0,671	3,9	0,132
	Ekstr. wł.	4	0,109	1,6	0,882	1,8	0,468
Obraz „Warzywa”	Ekstr. wył.	6	0,028	1,2	0,697	3,4	0,132
	Ekstr. wł.	3,6	0,123	1,1	0,744	1,7	0,476

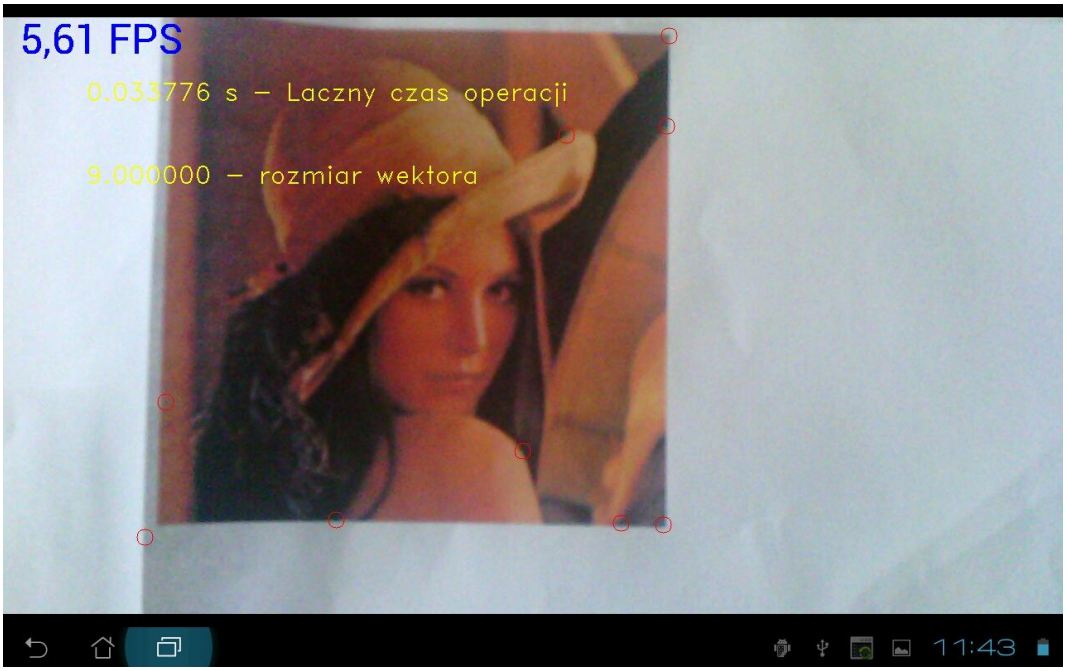
Tabela 4.4b: Wyniki pomiarów wydajności algorytmu detekcji cech na tablecie Asus Transformer TF101

Na podstawie otrzymanych wyników zaobserwowano, że dołączenie algorytmu ekstrakcji cech spowodowało wyraźny spadek wydajności działania całej aplikacji. Zjawisko to jest szczególnie widoczne w przypadku badań detektora ORB. Mając na uwadze fakt, iż ekstrakcja cech była objęta pomiarami timera, można zauważyć wyraźną zmianę czasu wykonywania całej operacji. Tak jak w poprzednich testach, również i w tym smartfon Samsung Galaxy S2 wykazał się największą szybkością działania. Zgodnie z teorią najwolniejszym detektorem z badanych okazał się MSER, uzyskując średnio 4 klatki na sekundę podczas uruchomienia na smartfonie oraz 1 klatki na sekundę na tablecie. Ponadto, zaobserwowano dużą stałość detektora Orb. Niezależnie od ustawienia kamer obu urządzeń, ilość wykrytych cech kluczowych pozostawała przy pożądanej liczbie jedenastu. MSER wykrył najmniej cech na obrazach testowych. Obserwacje działania detektorów pozwoliły stwierdzić, że najwięcej cech zostało znalezionych przez detektor ORB. Zmiana systemu operacyjnego zarówno w przypadku smartfonu Galaxy S2, jak i w przypadku tabletu

Transformer nie wykazała wyraźnej różnicy w wydajności działania algorytmu. Do badań wykorzystano też obraz przedstawiający jednolicie, równo rozłożone panele podłogowe. Gładka struktura powierzchni paneli spowodowała, że żaden z testowanych detektorów nie znalazł jakichkolwiek cech.



Rysunek 4.11a: Wynik działania operacji detekcji cech na obrazie testowym – smartfon Samsung Galaxy S2



Rysunek 4.11b: Wynik działania operacji detekcji cech na obrazie testowym – tablet Asus Transformer

Rozdział 5

Podsumowanie i wnioski

Przeprowadzona analiza pokazała wybrane możliwości biblioteki OpenCV na urządzeniach mobilnych. Przygotowane w środowisku Eclipse aplikacje pozwalały na przeprowadzenie odpowiednich testów wydajności wybranych algorytmów przetwarzania i detekcji na obrazach. Aplikacje napisano zarówno za pomocą samego języka Java jak i z wykorzystaniem elementów natywnych opracowanych w języku C++. Pierwszy program został w całości przygotowany w języku właściwym dla platformy Android i realizuje dwie wybrane operacje przetwarzania obrazów. Pozostałe dwie aplikacje napisano w języku Java, ale część ich algorytmów jest stworzonych natywnie. Na każdym programie testowym źródłem danych na temat wydajności urządzenia był wbudowany miernik liczby klatek na sekundę pokazywanego na ekranie obrazu z kamery. W przypadku aplikacji wykrywającej cechy, do pomiarów wydajności posłużyła dodatkowo funkcja timera mierząca czas jaki wymagany jest na wykonanie detekcji cech.

Wykonane testy polegały na uruchomieniu przygotowanych aplikacji na każdym urządzeniu mobilnym. Dokonano odczytu wyników pomiaru wydajności dla kilku różnych obrazów testowych odpowiednich dla danego programu. Testy pokazały, w jaki sposób można wykorzystać bibliotekę OpenCV na sprzęcie mobilnym. Przeprowadzone doświadczenia pozwoliły zauważyć, w jakim stopniu badane funkcje biblioteki obciążają układy obliczeniowe takich urządzeń. Najbardziej wymagającą operacją okazała się filtracja medianowa, której proces obniżał odczyt liczby klatek na sekundę o kilka ich wielkości. Warto zaznaczyć, że tablet Asus dysponuje ekranem o znacznie większej rozdzielczości od dwóch pozostałych urządzeń, więc otrzymane w trakcie badań wyniki nie świadczą o jego niskiej wydajności. Układ obliczeniowy tego tabletu ma w przybliżeniu trzy razy więcej pikseli obrazu do przetworzenia, co powoduje otrzymanie słabszych wyników.

5.1. Wnioski z przeprowadzonej analizy

Analiza przeprowadzonych badań oraz ich wyników ukazała, że w obecnym stadium rozwoju biblioteka OpenCV dobrze radzi sobie z działaniem na platformie Android. W trakcie pomiarów wszystkie aplikacje uruchomiły się na dwóch z trzech urządzeń wykorzystywanych na potrzeby pracy. Jedynie program zajmujący się detekcją cech nie

uruchomił się na tablecie Goclever TAB A73. Może to być spowodowane niską rozdzielczością kamery zamontowanej w tym sprzęcie. Podczas testów przygotowanych aplikacji nie zauważono żadnych problemów z funkcjonowaniem któregokolwiek z wybranych algorytmów. Regularne aktualizacje biblioteki OpenCV w wersji dla systemu Android powodują, że ewentualne trudności z pracą innych funkcji biblioteki na urządzeniach mobilnych zostają szybko usunięte.

Przy przeprowadzeniu testów operacji filtracji medianowej zauważono duży spadek wydajności przetwarzania obrazu na każdym urządzeniu. Na smartfonie Galaxy S2 spowolnienie w działaniu aplikacji było najmniej odczuwalne. Tablety Asus Transformer oraz Goclever TAB A73 zaprezentowały znacznie większy spadek szybkości funkcjonowania aplikacji. Prawdopodobnie spowodowane to było mniejszym wsparciem biblioteki OpenCV dla architektur zastosowanych w tych urządzeniach. Testy algorytmu progowania adaptacyjnego wykazały, że operacja ta nie obciąża mocno układów obliczeniowych. Sprawia to, że może być z powodzeniem implementowana na urządzeniach mobilnych z każdej półki cenowej. Efekt jej działania został sprawdzony na spreparowanych obrazach testowych zawierających tekst o czcionkach różnych wielkości i kolorów. Również w tym przypadku badany algorytm spełniał swoje zadanie i potrafił wydobywać tekst z tła o niejednorodnej barwie. Jakość obrazu przetworzonego za pomocą wymienionych operacji zależała od kamery zamontowanej w urządzeniu. W testach najlepszą szczegółowością wykazał się smartfon Samsung Galaxy S2 z powodu zastosowania wysokiej jakości optyki w urządzeniu.

W badaniach algorytmu detekcji twarzy zauważono większą wydajność działania mechanizmu napisanego w języku Java. Szybkość detekcji twarzy zależała od ułożenia twarzy widocznych na obrazie. Powoduje to, że można uznać ten mechanizm za nie w pełni szczegółowy. Aby twarze były poprawnie wykryte, na obrazie powinny być skierowane w miarę możliwości w stronę obiektywu a także w powinna być zauważalna granica między nimi. Wykorzystane zdjęcia testowe wykazały, że algorytm potrafi wykryć na obrazie cztery osoby widoczne na zdjęciu grupy ludzi. Prawdopodobnie po spełnieniu warunku wymienionego wyżej liczba ta byłaby większa. Wadą algorytmu detekcji twarzy jest jego mała precyzja.

Pomiary wydajności działania aplikacji do detekcji cech ukazały widoczną różnicę pomiędzy ilością cech wykrytych przez każdy z zastosowanych detektorów. Badania wykonywano za pomocą wbudowanych funkcji miernika liczby klatek na sekundę oraz timera. Najwolniejszym detektorem okazał się MSER, który znacznie spowolnił działanie aplikacji oraz wyszukał najmniejszą ilość cech, natomiast najszybszym był FAST, którego działanie było najmniej wymagające oraz znalazł najwięcej cech.

Podczas realizacji pracy największą trudność sprawiało wyświetlenie wyników timera zamieszczonego w aplikacji zajmującej się detekcją cech. Należało znaleźć odpowiedni ciąg instrukcji, który współpracowałby z komendą *putText*, obrazującą dowolny tekst na ekranie w aplikacjach OpenCV napisanych w języku C++.

Na podstawie otrzymanych wyników można stwierdzić, że prowadzenie obliczeń przetwarzania obrazów ma zastosowanie na urządzeniach mobilnych i w większości przypadków nie ma potrzeby realizacji takich kalkulacji na zewnętrznych serwerach, które wysyłałyby potem wyniki na smartfony oraz tablety.

5.2. Możliwości rozbudowy

Napisana praca inżynierska nie wyczerpała wszystkich możliwości wykorzystania biblioteki OpenCV. Skupiono się na jej najbardziej reprezentatywnych funkcjach. Zbadane zostały operacje, dla których bezproblemowo można znaleźć zastosowanie w urządzeniach mobilnych. Aplikacje testowe mogłyby zostać rozbudowane o zastosowanie innych filtrów lub o algorytmy służące do detekcji różnych obiektów.

W przyszłości istnieje możliwość rozszerzenia pracy o dodatkowe porównanie testowanych algorytmów z ich odpowiednikami na komputerze stacjonarnym. Stworzyłoby to możliwość sprawdzenia, jak szybko działa biblioteka na urządzeniach mobilnych w odniesieniu do jej funkcjonowania na komputerach stacjonarnych. Inną próbą rozwinięcia pracy mogłoby być całkowite przebudowanie aplikacji testowych tak aby była możliwość wczytania dowolnych plików graficznych bezpośrednio z pamięci urządzenia. Pomiary wydajności działania algorytmów przeprowadzono by wtedy z wykorzystaniem innych narzędzi. Na potrzeby testów rozbudowano by moduł timera, którego zadaniem byłoby zbieranie czasu działania wielu funkcji zawartych w algorytmie. Dodatkowym sposobem pomiaru wydajności można również uczynić funkcję profilowania metod dostępną w środowisku programistycznym Eclipse.

Biblioteka OpenCV w wersji dla komputerów stacjonarnych posiada zaimplementowane wsparcie dla procesorów wielordzeniowych. Dzięki jej współpracy z biblioteką Intel TBB programiści mają możliwość tworzenia własnych aplikacji OpenCV wykorzystujących wielowątkowość procesorów. Dzisiejsze urządzenia mobilne są coraz częściej wyposażane w układy wielordzeniowe. Powoduje to że OpenCV w wersji dla systemu Android w niedługim czasie powinno otrzymać pełną współpracę z wieloma rdzeniami co stwarzałoby możliwość wykorzystania jej wielowątkowości na smartfonach oraz tabletach. Spowodowanie, że biblioteka mogłaby obsługiwać wiele rdzeni przyniosłoby oczekiwane przyspieszenie w jej działaniu.

Bibliografia

[1] Gary Bradski, Adrian Kaehler, „Learning OpenCV: Computer Vision with the OpenCV Library”, O’Reilly Media, 2008

[2] Ewaryst Rafajłowicz, Wojciech Rafajłowicz, Andrzej Rusiecki, „Algorytmy przetwarzania obrazów i wstęp do pracy z biblioteką OpenCV”, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2009

[3] Robert Laganière, „OpenCV 2 Computer Vision Application Programming Cookbook”, Packt Publishing, 2011

[4] Bartłomiej Mrożewski „Smartfon zamiast ksero”, artykuł zamieszczony w magazynie komputerowym PCFormat, nr. 2/2013

[5] Dokumentacja biblioteki OpenCV, dostępna w Internecie: http://docs.opencv.org/ [dostęp 5 lutego 2013]

[6] Dokumentacja biblioteki OpenCV w wersji dla systemu Android, dostępna w Internecie: http://docs.opencv.org/java/ [dostęp 5 lutego 2013]

[7] Poradnik instalacji i przygotowywania środowiska programistycznego Eclipse oraz biblioteki OpenCV dla systemu Android, dostępny w Internecie: http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/android_dev_intro.html [dostęp 5 lutego 2013]

[8] Strona encyklopedyczna zawierająca zbiór pomocniczych informacji na temat biblioteki OpenCV, dostępna w Internecie: http://opencv.willowgarage.com/wiki/FullOpenCVWiki [dostęp 5 lutego 2013]

[9] Artykuł komentujący wyniki badań na temat zwiększenia zainteresowania urządzeniami mobilnymi wśród osób w różnym wieku, dostępny w Internecie: http://technologie.gazeta.pl/internet/1,104530,13180721,Koniec_ery_PC__Raczej_koniec_komputerow_domowych.html [dostęp 5 lutego 2013]

[10] Wpisy traktujące o urządzeniach mobilnych umieszczone na blogu technologicznym Spidersweb.pl: http://www.spidersweb.pl/ [dostęp 5 lutego 2013]

[11] Wpis na forum pomocy biblioteki OpenCV, dostępny w Internecie: http://answers.opencv.org/question/445/242-android-missing-nonfree-package/ [dostęp 5 lutego 2013]

[12] Strona internetowa smartfonu Samsung Galaxy S2, dostępna w Internecie: http://www.samsung.com/pl/consumer/mobile-phone/mobile-phones/smartphone/GT-I9100LKAERA-spec [dostęp 5 lutego 2013]

[13] Strona internetowa tabletu Asus Transformer TF101, dostępna w Internecie: http://pl.asus.com/Eee/Eee_Pad/Eee_Pad_Transformer_TF101/#specifications [dostęp 5 lutego 2013]

[14] Strona internetowa tabletu Goclever TAB A73, dostępna w Internecie: http://goclever.com/pl/pl/goclever_tab_a73#specification [dostęp 5 lutego 2013]

[15] Informacje o progowaniu adaptacyjnym, dostępne w Internecie: http://aragorn.pb.bialystok.pl/~boldak/DIP/CPO-W07-v01-50pr.pdf [dostęp 5 lutego 2013]

[16] Artykuł „Smartfon" w encyklopedii Wikipedia, dostępny w Internecie: http://pl.wikipedia.org/wiki/Smartfon [dostęp 5 lutego 2013]

[17] Artykuł „Android" w encyklopedii Wikipedia, dostępny w Internecie: http://pl.wikipedia.org/wiki/Android_(system_operacyjny) [dostęp 5 lutego 2013]

[18] Artykuł „iPhone" w encyklopedii Wikipedia, dostępny w Internecie: http://pl.wikipedia.org/wiki/IPhone [dostęp 5 lutego 2013]