VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

APLIKACE PRO ZPRACOVÁNÍ OBRAZU NA PLATFORMĚ ANDROID
APPLICATIONS FOR IMAGE PROCESSING FOR ANDROID OS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Martin Kiac**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Ondřej Krajsa, Ph.D.**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Martin Kiac ***ID:*** 177265
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Aplikace pro zpracování obrazu na platformě Android

POKYNY PRO VYPRACOVÁNÍ:

V bakalářské práci proveďte návrh a realizaci aplikace využívající knihovnu OpenCV na plattormě Android. Aplikace bude provádět detekci a sledovování předem definovaných objektů z videosignálu kamery daného zařízení.

DOPORUČENÁ LITERATURA:

[1] JOSEPH HOWSE, Steven Puttemans. OpenCV 3 Blueprints. Birmingham: Packt Publishing Ltd, 2015. ISBN 9781784399757.

[2] LACKO, Ľuboslav. Vývoj aplikací pro Android. 1. vyd. Brno: Computer Press, 2015, 472 s. : il., mapy. ISBN 978-80-251-4347-6

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 8.6.2017

Vedoucí práce: Ing. Ondřej Krajsa, Ph.D.
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Táto práca sa zaoberá problematikou spracovania obrazu a detekciou predom definovaných objektov z kamery daného zariadenie na platforme Android. Na spracovanie obrazu sú v práci použité nástroje a metódy z knižnice OpenCV. Práca sa skladá z teoretickej časti, kde je popísaná samotná platforma Android, štruktúra tohto operačného systému a vývojové prostredie Android Studio. Teoretická časť práce ďalej popisuje knižnicu OpenCV a teóriu spracovania obrazu pomocou tejto knižnice. V praktickej časti práce je vysvetlená implementácie knižnice OpenCV do vývojového prostredia Android Studio. Ďalej je v praktickej časti popísaný postup pri spracovaní obrazu pomocou knižnice OpenCV. Následne je táto práca venovaná analýze a vyhľadávaniu kontúr, použitie Houghovej transformácie a segmentácií spracovaného obrazu. V práci je ďalej popísaná realizácia grafického užívateľského rozhrania a následne práca s databázou aplikácie. Záver práce je venovaný testovaniu a vyhodnoteniu dosiahnutých výsledkov.

KĽÚČOVÉ SLOVÁ

Android, platforma, OpenCV, Android Studio, aplikácia, spracovanie obrazu, detekcia objektov

ABSTRACT

This work deals with an issue of image processing and detection of predefined objects from the camera on Android platform. This work uses image processing tools and methods from OpenCV library. The work consists of a theoretical part, in which the Android platform itself is described, the structure of the operating system and Android Studio development environment. The theoretical part also contains a description of OpenCV library and describes the theory of image processing using this library. The practical part of this work describes implementation of OpenCV library to development environment Android Studio. The practical part also describes techniques for image processing by OpenCV library. Subsequently this work contains analysis and searching for contours, the use of Hough transformation and segmentation of the processed image. This work also describes realization of graphical user interface and subsequently work with the application's database. The conclusion is about final evaluation of the work and achieved results.

KEYWORDS

Android, platform, OpenCV, Android Studio, application, image processing, object detection

KIAC, Martin. *Aplikace pro zpracování obrazu na platformě Android*. Brno, Rok, 51 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Ondřej Krajsa, Ph.D.

VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na tému „Aplikace pro zpracování obrazu na platformě Android“ vypracoval(a) samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor(ka) uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil(a) autorské práva tretích osôb, najmä som nezasiahol(-la) nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý(-á) následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

POĎAKOVANIE

Rád by som poďakoval vedúcemu bakalárskej práce pánovi Ing. Ondřejovi Krajsovi, Ph.D. za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci. Ďalej by som chcel poďakovať pánovi doc. Ing. Kamilovi Říhovi, Ph.D. za odborné konzultácie a nasmerovanie pri riešení problematiky práce.

Brno ....

podpis autora(-ky)

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ÚVOD

Ľudské oko patrí k najdôležitejším orgánom, pomocou ktorého dokáže človek poznávať, vnímať a reagovať na svoje okolie. Preto v dnešnej modernej dobe je stále väčšia snaha implementovať túto vlastnosť aj do výpočtovej techniky. Počítačové videnie má v modernej technike široké uplatnenie v rôznych oblastiach ako je robotika, priemyselne systémy, medicínska technika, bezpečnostné systémy až po rozšírenú realitu a mnohé ďalšie oblasti. S príchodom moderných technológií sa počítačové videnie stáva stále dostupnejšie nie len v oblasti priemyslu, ale aj v oblasti bežného života. Tú nachádza využitie v zariadeniach, ktoré sú v dnešnej dobe neoddeliteľnou súčasťou nášho života. Sem patria zariadenia ako sú rôzne smartfóny, kamery, virtuálna realita, inteligentné televízie alebo fotoaparáty. Tu všade nachádza počítačové videnie čoraz väčšie uplatnenie.

Mobilné zariadenia disponujú čoraz väčším výkonom hardvéru a podporou stále kvalitnejších periférií, medzi ktoré patrí kamera. A preto nachádza počítačové videnie uplatnenie aj v oblasti mobilných technológií. Takéto zariadenie je možné následne využiť na sledovanie alebo rozpoznávanie objektov, segmentáciu obrazu alebo aplikovanie rôznych obrazových filtrov.

Bakalárska práca je venovaná spracovaniu obrazu na mobilnej platforme Android. Cieľom práce je navrhnúť a realizovať aplikáciu pre operačný systém Android, ktorá využíva knižnicu OpenCV na spracovanie obrazu. Základnou požiadavkou je, aby aplikácia dokázala sledovať a detekovať vopred definované objekty zo signálovou kamery daného zariadenia. Výsledná aplikácia by mala slúžiť v praxi na detekovanie mikrodoštičky, pipety a následnú analýzu obrazu pri pipetovaní.

Táto práca je rozdelená do dvoch hlavných kapitol, ktoré sa skladajú z menších sekcií. Prvá kapitola je venovaná teoretickým poznatkom tejto práce. Sú tu opísané jednotlivé problematiky, s ktorými je potrebné sa pred riešením práce zoznámiť. Prvá sekcia tejto kapitoly je zameraná na samotnú platformu Android, jeho štruktúru a vnútornú architektúru. Následne sú tu popísané dostupné verzie tohto operačného systému a životný cyklus aplikácií, ktoré bežia na tejto platforme. Ďalšia sekcia kapitoly sa zaoberá vývojovými prostrediami, ktoré sa na vývoj aplikácií pre túto platformu používajú. Následne je táto sekcia venovaná predstaveniu potrebných nástrojov, ktoré je možné pri vývoji aplikácie použiť. Ďalšia sekcia tejto kapitoly sa zaoberá samotnou knižnicou OpenCV, ktorá je v tejto práci použitá na spracovanie obrazu. Ďalej je tu popísaná distribúcia tejto knižnice, ktorá je určená pre platformu Android. Posledná sekcia tejto kapitoly je venovaná teórií spracovania obrazu na tejto platforme. Sú tu predstavené základné triedy a metódy, ktoré knižnica OpenCV obsahuje. Taktiež sú tu popísané dôležité algoritmy, ktoré sa používajú pri spracovaní alebo analýze obrazu.

Druhá kapitola sa zaoberá praktickým riešením bakalárskej práce. Úvodná sekcia tejto kapitoly je venovaná popisu implementácie knižnice OpenCV do vývojového prostredia. Je tu opísaný presný postup implementácie a riešenia prípadných chýb, ktoré môžu nastať pri samotnej implementácií. Ďalšia sekcia kapitoly sa zaoberá spracovaním obrazu, popisuje použitie základných obrazových filtrov. Nasledujúca sekcia tejto kapitoly sa venuje analýze obrazu. Sú tu vysvetlené jednotlivé algoritmy a metódy, ktoré sa pri analýze obrazu používajú. Medzi tieto metódy patrí vyhľadávanie kontúr v obraze, Houghova transformácia alebo dôležitý algoritmus MOG2, použitý na segmentáciu obrazu. Ďalšia sekcia je venovaná popisu algoritmu na vyhodnotenie zhody špičky pipety v jamke mikrodoštičky. Tento algoritmus je dôležitý, pretože jeho modifikácia je v práci často používaná. V ďalšej sekcií sú popísané jednotlivé zobrazovacie módy aplikácie, ktoré aplikácia využíva na zobrazenie určitých výstupných dát užívateľovi. Záver praktickej časti tejto práce sa zaoberá grafickým užívateľským rozhraním, ktoré aplikácia obsahuje, aby bolo možné aplikáciu jednoducho ovládať. Taktiež je tu vysvetlený spôsob, akým aplikácia uchováva všetky užívateľské nastavenia.

Záver tejto práce tvorí komplexné zhrnutie celej bakalárskej práce a dosiahnuté výsledky riešenia problematiky.

1 ÚVOD DO PROBLEMATIKY

V tejto kapitole je vo všeobecnosti predstavená platforma Android, postup a dôležité pojmy pri vývoji aplikácií pre túto platformu. Ďalej sa kapitola zaoberá knižnicou OpenCV a problematikou spracovania obrazu na tejto platforme.

1.1 O platforme Android

Android je rozsiahly otvorený operačný systém založený na linuxovom jadre. V súčastnosti je tento systém vyvíjaný spoločnosťou Google a patrí medzi najrozšírenejšie operačné systémy, ktoré sú využívané v telefónoch, tabletoch, navigáciách, ale aj v ostatných zariadeniach, kde sa stále rozširuje.

V samotných začiatkoch sa táto spoločnosť nazývala Android Inc. Bola založená štvoricou vývojárov v októbri 2003, v tej dobe ešte obyčajných ľudí. Pôvodný zámer spoločnosti bol vyvinúť pokročilý operačný systém pre digitálne fotoaparáty. Spoločnosť si však uvedomila, že v tej dobe trh digitálnych fotoaparátov ešte nebol dostatočne rozšírený, a tak nasmerovali svoje kroky práve k vývoju operačného systému pre mobilné zariadenia. Spoločnosť Android Inc. sa po čase dostala údajne do finančných problémov a tak sa stala hľadáčikom pre väčšie firmy. V auguste 2005 bola firma odkúpená spoločnosťou Google, ktorý podľa údajov mal v tom čase plán preraziť na trh s mobilnými zariadeniami. O dva roky neskôr, Google inicioval vytvorenie skupiny OHA (Open Handset Alliance) s cieľom vytvoriť nový otvorený systém pre mobilné zariadenia. A tak spoločnosť Android, po boku s Google prichádza na trh s plánom vyvinúť flexibilný a ľahko rozšíriteľný systém. Prvá dostupná platforma, ktorú Android pre mobilné telefóny vytvoril, niesla názov Android 1.0 *Alpha*. Tento systém bežal na linuxovom jadre. Obsahoval jednoduché užívateľské rozhranie a aplikácie, ktoré boli napísané v programovacom jazyku *Java*.

Pri vývoji samotnej platformy boli použité také technológie, aby bolo možné výsledný systém integrovať do značného množstva už existujúcich telefónov. Zároveň bol systém stále podporovaný pre najnovšie mobilné zariadenia. S príchodom tohto systému na trh sa otvárali nové možnosti. Prvým komerčne dostupným mobilným zariadením, ktorý bežal na systéme Android bol model HTC Dream, uvedený do predaja v októbri 2008. V roku 2010 Google uviedol vlastnú sériu mobilných zariadení, zvanú Nexus. Od roku 2008 prešiel Android už mnohými zmenami a vylepšeniami, ktoré značne zdokonalili tento operačný systém [11].

1.1.1 Dostupné verzie operačného systému Android

Operačný systém Android je zatiaľ jeden z najmladších operačných systémov, ktorý mobilné telefóny používajú. Populárnym sa stal aj vďaka svojmu intuitívnemu prostrediu či množstvu praktických funkcií, ktoré tento systém ponúka. Operačný systém sa úspešne rozrastá aj vďaka svojmu obchodu s aplikáciami, ktorý dostal názov Google Play (pôvodne Android Market). V súčasnosti totiž ponúka cez 2,4 milióna aplikácií, pričom väčšina z nich je dostupná bezplatne. Od svojho vzniku vyšlo viacero verzií tohto systému, ktoré priniesli radu vylepšení. Nasledujúca tabuľka poskytuje informácie o počte zariadení, ktoré tento systém používajú. Informácie boli aktualizované v Novembri 2016. Z tabuľky môžeme vidieť, že staršie verzie systému Android pomaly zanikajú. Staré verzie sú už zastaralé, strácajú podporu, oproti novým verziám sú pomalšie a menej výkonné. Novšie verzie systému Android prinášajú nové funkcie a technológie [4]. V tabuľke 1.1 je možné vidieť jednotlivé verzie Android a ich rozšírenie na zariadeniach.

Verzia systému	**Využitie**
Froyo	0.1 %
Gingerbread	1.3 %
Ice Cream a Sandwich	1.3 %
Jelly Bean	13.7 %
KitKat	25.2 %
Lollipop	34.2 %
Marshmallow	24.0 %
Nougat	0.3 %

Tab. 1.1: Značenie systému Android

1.1.2 Architektúra operačného systému

Operačný systém Android pozostáva z 5 vrstiev, ktoré sú potrebné pre funkčnosť celého systému na zariadení. Hlavnú časť tohto systému tvorí linuxové jadro **Linux Kernel**. Táto časť zabezpečuje správu a komunikáciu medzi hardvérom a softvérom zariadenia. Obsahuje rôzne hardvérové ovládače pre komunikáciu s perifériami zariadenia, ako je displej, kamera, wifi a ďalšie, ktoré môže zariadenie obsahovať. Ďalšia vrstva **Android Runtime** zabezpečuje priestor pre beh procesov v systéme (Virtual Machine). Každá aplikácia v systéme funguje ako samostatný bežiaci proces. Predchodca tejto vrstvy bol virtuálny stroj Dalvik, ktorý plnil jej funkciu. Ďalšiu

časť systému tvorí vrstva **Libraries**. Obsahuje C alebo C++ knižnice, ktoré systém používa na komunikáciu s rôznymi ďalšími komponentami systému. Tieto knižnice sú dostupné pre vývojára vo vývojovom aplikačnom rozhraní API (Application Programming Interface). Ďalšia podstatná časť, z ktorej Android pozostáva je **Application Framework**. Táto vrstva ponúka vývojárom možnosť pracovať s ďalšími procesmi, službami a manažérmi systému. Práve táto vrstva je najviac podstatná pre vývoj aplikácií. Obsahuje dôležitý Activity Manager, ktorý riadi životný cyklus aplikácie. Slúži na prepínanie medzi jednotlivými aktivitami (napr. spustenie a ukončenie aplikácie). Ďalej táto vrstva obsahuje Package Manager, Resource Manager, Location Manager a ďalšie iné. Posledná vrstva ktorá sa nachádza v systéme je **Aplications**. Všeobecne môžeme rozlišovať dva druhy aplikácií. Aplikácie, ktoré sú súčasťou systému–predinštalované a všetky ostatné užívateľom inštalované aplikácie. Každá distribúcia systému obsahuje základný balíček systémových aplikácií (core aplications). Sú to aplikácie na telefonovanie a prijímanie hovorov, posielanie a prijímanie sms správ, kontakty, kalendár a podobné predinštalované aplikácie [6].

1.1.3 Štruktúra operačného systému

V tejto sekcií je uvedený stručný popis štruktúry operačného systému, a objasnenie niekoľko dôležitých pojmov v tomto systéme. Systém Android je tvorený určitou štruktúrou – komponentami systému. Výhodou takejto stavby systému môže byť odolnosť voči systémovým chybám a väčšia stabilita systému [2]. Operačný systém tvoria 4 základné stavebné komponenty.

- **Aktivity** (Activities)
- **Služby**(Services)
- **Poskytovatelia obsahu**(Content providers)
- **Prijímače vysielania**(Broadcast recievers)

Každá z týchto komponent má určité špecifické vlastnosti, ktoré systém neustále používa pri svojom behu. Aktivity sú primárne stavebné bloky užívateľského rozhrania aplikácie. Aktivitu si môžeme predstaviť ako jednu obrazovku aplikácie s jednoduchým užívateľským rozhraním, ktorú môžeme kedykoľvek ukončiť a znova spustiť. Ak napríklad spustíme aplikáciu, zobrazí sa nám úvodná obrazovka, ktorá je tvorená aktivitou. Každé ďalšie obrazovky aplikácie môžu byť tvorené novými aktivitami. Každá aktivita sa dá charakterizovať svojim aktuálnym stavom, v ktorom sa nachádza, viď obrázok 1.1. Aplikáciu tak môžeme kedykoľvek spustiť, zastaviť, nechať ju bežať na pozadí, prípadne ju úplne zavrieť. Každá aktivita je tak tvorená tzv. životným cyklom. Táto problematika je podrobnejšie vysvetlená v sekcií 1.1.4.

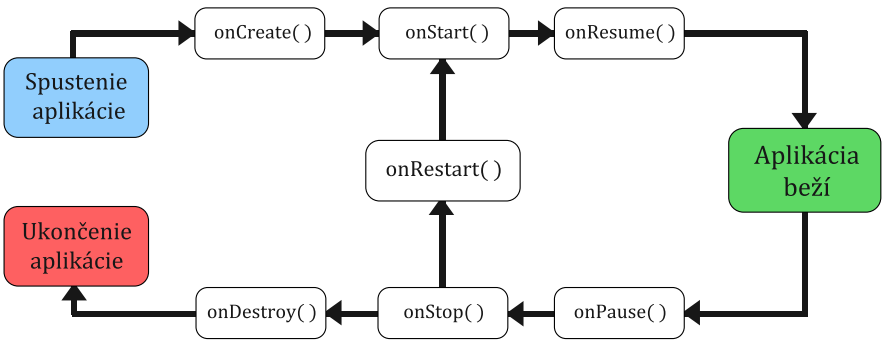
Služby sú navrhnuté k neustálej prevádzke. Používajú sa napríklad ku kontrole dostupných aktualizácií alebo k prehrávaní hudby na pozadí. Taktiež sa používajú na sprístupnenie rozhrania API iným aplikáciám nainštalovaným v zariadení.

Ďalšou dôležitou častou sú sprostredkovatelia obsahu. Tento komponent poskytuje určitú úroveň dát, ktoré sú v zariadení uložené. Vývojový model aplikácií pre Android podporuje užívateľov v tom, aby svoje dáta sprístupňovali aj ostatným aplikáciam. Aby mohli aplikácie pracovať s užívateľskými dátami, potrebujú povolenie od užívateľa. Napríklad aplikácia fotoaparát môže vyžadovať prístup do galérie fotiek užívateľa. Práve k tomuto slúži komponent sprostredkovatelia obsahu, ktorá riadi a pristupuje k užívateľským dátam.

Poslednou dôležitou časťou je komponent prijímače vysielania. Sú to systémové správy, ktoré kolujú v zariadení a upozorňujú aplikáciu na rôzne systémové udalosti. Systém tak môže reagovať napríklad na zmenu stavu hardvéru, vloženie SD alebo SIM karty, prichádzajúce dáta a rôzne ďalšie udalosti aplikácie [2].

1.1.4 Životný cyklus Android aplikácie

Každá aplikácia je tvorená aktivitami, ktoré sú definované určitým stavom, v ktorom sa nachádzajú. Tieto aktivity majú blokovú štruktúru, ktorá tvorí životný cyklus aplikácie. Na zabezpečenie správnej reakcie na každú zmenu stavu aplikácie, nám trieda `Activity.java` poskytuje určite metódy, ktoré sú implicitne pri každej zmene programu zavolané. Aplikácia je tak stabilnejšia, je možné predchádzať chybám a vývojár tak dokáže reagovať na prípadné stavy, ktoré v aplikácií nastávajú.



Obr. 1.1: Životný cyklus aktivity

- **metóda `onCreate()`** – sa volá pri spustení aktivity. V metóde je definované všetko potrebné, aby aktivita mohla spoľahlivo fungovať. Metóda zadefinuje aké grafické rozhranie aktivity sa má zobraziť, aké komponenty daná aktivita bude používať (tlačítka, zobrazovač textu).

- **metóda `onStart()`** – sa volá pokiaľ bola aktivita prvýkrát spustená alebo bola aktivovaná z pozadia po svojom skrytí (systémové dialógy).
- **metóda `onResume()`** – sa volá tesne pred tým, než je aktivita posunutá do popredia (prvé spustenie alebo reštart aplikácie).
- **metóda `onPause()`** – sa volá pred odchodom aktivity na pozadie. Systém tak dostáva práva na uspanie alebo ukončenie aktivity.
- **metóda `OnStop()`** – sa volá pred zastavením aktivity.
- **metóda `OnDestroy()`** – sa volá tesne pred zrušením aktivity. Systém tak dostáva práva na ukončenie aktivity.
- **metóda `onRestart()`** – sa volá ak bola aktivita zastavená metódou `onStop()`. Následne po zavolaní metódy `onRestart()` sa automaticky volá `onStart()` a tak sa aktivita reštartuje.

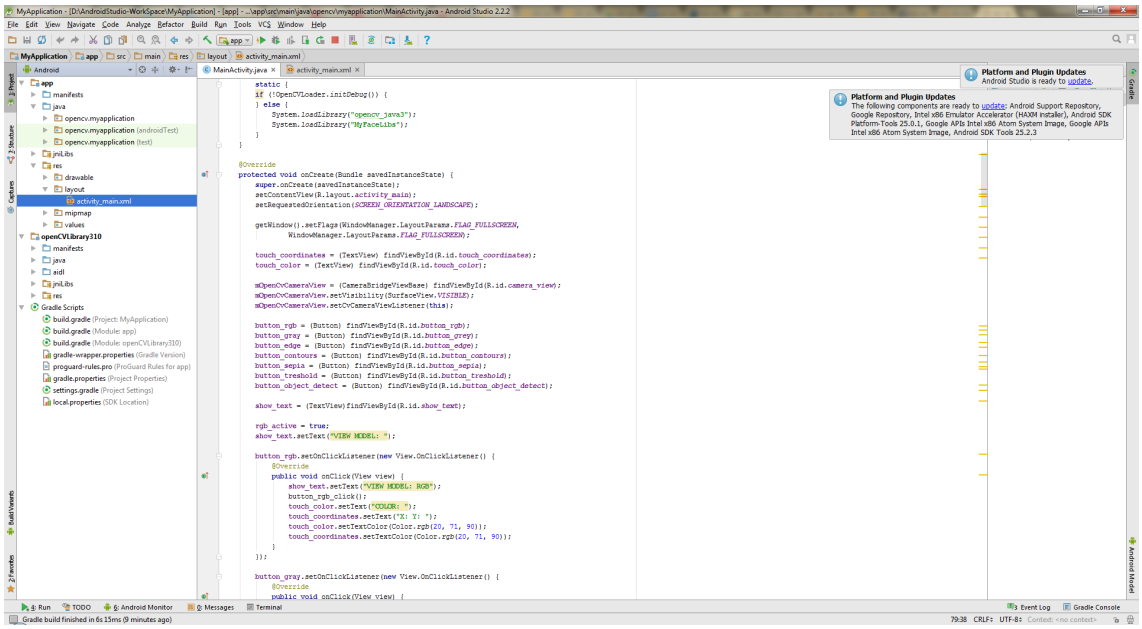
Každá aktivita sa teda vždy nachádza v jednom stave, podľa toho ako s aplikáciou zachádzame. Jednotlivé stavy môžeme následne definovať ako: aktivita je **aktívna**, **pozastavená**, **zastavená** alebo je aktivita **mŕtva** [2]. Podľa toho, v akom stave je aktivita, sa volajú príslušné metódy, viď obrázok 1.1.

1.2 Vývoj aplikácií pre platformu Android

Pred samotným vývojom aplikácie je potrebné, aby si programátor zvolil také vývojové prostredie, ktoré vývoj aplikácií pre daný operačný systém podporuje. Vývojových prostredí na tvorbu aplikácií pre platformu Android existuje niekoľko. Najviac rozšírené sú momentálne dve vývojové prostredia, ktoré ponúkajú širokú škálu rôznych vývojových nástrojov. V nasledujúcej časti sú tieto vývojové prostredia navzájom porovnané.

1.2.1 Vývojové prostredia

Jedným z prvých vývojových prostredí, ktoré bolo donedávna oficiálnym vývojovým nástrojom je **Eclipse**. Toto prostredie podporuje prácu v rôznych programovacích jazykoch a pre rôzne platformy. Aby bolo možné v tomto prostredí aplikáciu vyvíjať, je potrebné si do prostredia stiahnuť ADT (Android Development Tools) plugin. Tento plugin ponúka podporu pre vývoj a prácu s aplikáciami na platforme Android. Vývojové prostredie tak môže ponúknuť priestor a potrebné nástroje na vývoj aplikácií pre túto platformu [2]. Ďalšie vývojové prostredie **Android Studio** je pomerne nové. Prvá verzia tohto prostredia bola dostupná v roku 2013. Toto prostredie je dnes už oficiálnym nástrojom pre vývoj aplikácií na platforme Android. Prostredie je postavené na IntelliJ IDEA technológií a je primárne určené na vývoj aplikácií



Obr. 1.2: Vývojové prostredie Android Studio

pre túto platformu. Android Studio je tvorené flexibilným systémom gradle build, obsahuje rôzne emulátory, systémové nástroje, podporuje Google Cloud platformy a mnohé ďalšie. Do prostredia je možné inštalovať ďalšie moduly, čím sa rozširuje použitie o ďalšie nástroje, ktoré je možné pri vývoji tejto platformy používať. Android Studio je voľné dostupné, súčasťou tohto balíčku je samotné IDE (Integrated Development Environment) Android Studio, Android SDK Tools, kompilátor a balíček tiež obsahuje vývojový emulátor s plnohodnotným systémom Android, v ktorom môže vývojár v reálnom čase simulovať svoju aplikáciu. Na obrázku 1.2 je ukážka vývojového prostredia Android Studio [6].

1.2.2 Prehľad nástrojov potrebných pre vývoj aplikácie

V tejto sekcií je popísaných niekoľko základných nástrojov, ktoré sa na vývoj aplikácií používajú. Použitie niektorých nástrojov je nevyhnutné pre správnu funkčnosť aplikácie.

Android SDK

Sada Android SDK (Software Development Kit) poskytuje nástroje potrebné k vytváraniu a testovaniu aplikácií pre Android. Táto sada sa skladá z dvoch častí. Prvá z nich je tvorená základnými nástrojmi, druhú časť tvoria komponenty pre konkrétne verzie a príbuzné doplnky systému. Nastavenie podpory Android SDK je možné nájsť v súbore `manifest.xml`.

- `android:minSdkVersion` – Implicitne nadobúda hodnotu 1. Deklaruje minimálnu hodnotu verzie API, ktorú aplikácia podporuje.
- `android:maxSdkVersion` – Deklaruje maximálnu hodnotu verzie API, ktorú aplikácia podporuje.
- `android:targetSdkVersion` – Deklaruje verziu API, pre ktorú je aplikácia primárne určená.

Android API

Rozhranie API definuje, aké nástroje je možné pri vývoji aplikácie použiť. Pomocou úrovne API je možné určiť kompatibilitu jednotlivých vývojových nástrojov. Nové verzie API podporujú aktuálne technológie. Pri práci s nižšími úrovňami rozhrania API klesá podpora nových technológií. Staršie úrovne API aktuálne technológie nemusia vôbec podporovať, nie sú na to uspôsobené.

Verzia API je vyjadrená číselnou hodnotou, ktorá zodpovedá jednotlivým úrovniam nástrojovej sady Android SDK. Každá verzia systému Android prislúcha presne jednej úrovni API. Daná verzia API môže byť stále kompatibilná aj so všetkými vyššími prípadne nižšími úrovňami. Napríklad, ak je pri vývoji aplikácie použitá verzia API 21, aplikácia je primárne určená pre operačný systém Android 5.0 *LOLLIPOP*. Vytvorená aplikácia bude kompatibilná aj s vyššími úrovňami API, to znamená 21 a vyššie. Novšie úrovne API by mali mať implementované technológie z predchádzajúcich úrovni API. Obecne platí, ak používame najnovšie úrovne API, máme k dispozícií najširšie možnosti pri vývoji, najmodernejšie nástroje, metódy a technológie. Pri novších úrovniach API klesá podpora pre staršie zariadenia, ktoré podporujú už zastarané úrovne API [10]. V nasledujúcej tabuľke 1.2 sú zobrazené jednotlivé verzie Androidu a ich príslušné úrovne API.

Verzia platformy	**API úroveň**	**Verzia**
Android 7.1	25	Nougat
Android 6.0	23	Marshmallow
Android 5.1	22	Lollipop

Tab. 1.2: Ukážka značenia operačného systému Android

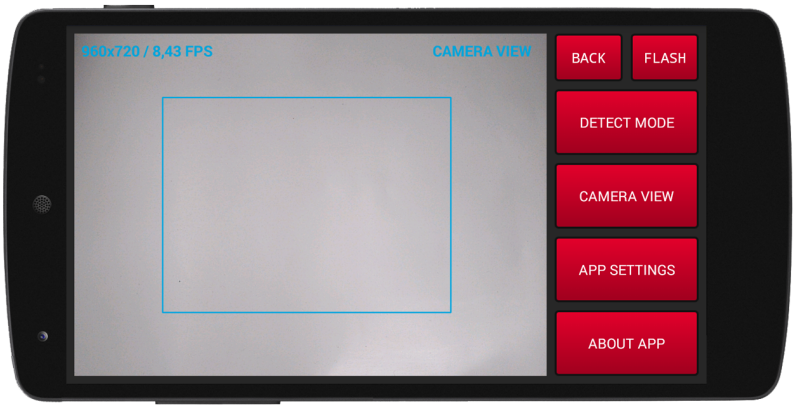
Android NDK

Android NDK (Native Development Kit) je ďalšia sada nástrojov, ktorá umožňuje pri vývoji aplikácie používať programovací jazyk C a C++. Vývojovú sadu NDK má zmysel použiť v prípadoch, kedy je potrebné zo zariadenia získať čo najväčší

výkon alebo sa vyžaduje malé oneskorenie. Ďalšia možnosť je využitie knižníc, ktoré sú napísané v tomto natívnom jazyku. Pomocou NDK je možné priamo pristupovať k fyzickým súčastiam zariadenia, ako sú napríklad rôzne senzory alebo digitizer obrazovky zariadenia.

Android AVD

Medzi ďalšie vývojové nástroje, ktoré je možné použiť pri vývoji aplikácií patrí virtuálne zariadenie AVD (Android Virtual Device). Je to špeciálny software, ktorý simuluje zariadenie Android, na ktorom sa nachádza plnohodnotný operačný systém. Tento nástroj môže byť pri vývoji aplikácie veľmi užitočný, najmä v prípade, kedy nie je dostupné zariadenie s požadovanými parametrami, alebo nie je fyzicky k dispozícii žiadne zariadenie. Výhodou je, že takýto emulátor je výkonný a prenos dát je rýchlejší ako pri použití fyzického zariadenia. Preto môže použitie AVD šetriť čas pri vývoji a testovaní aplikácie [2].



Obr. 1.3: Jednoduché štrukturálne elementy erózie a dilatácie

1.3 Použité nástroje na spracovanie obrazu

Táto kapitola je venovaná spracovaniu obrazu na platforme Android. Sú tu popísané nástroje, ktoré je možné pri spracovaní obrazu použiť. V kapitole je taktiež popísaná knižnica OpenCV, ktorá sa na spracovanie obrazu využíva.

1.3.1 Knižnica OpenCV

V dnešnej dobe existuje pomerne veľa kvalitných, voľne dostupných knižníc určených na prácu s obrazom. Takéto knižnice sú stále dostupnejšie pre širšiu skupinu operačných systémov. Zväčšuje sa aj podpora jednotlivých programovacích jazykov a tým

sú knižnice dostupnejšie pre vývojárov. Takéto knižnice obsahujú hotové nástroje a metódy určené na spracovanie obrazu a jeho analýzu. Použitie takejto knižnice je pre vývojára výhodou v podobe ušetreného času a priestoru. Preto nemá veľký zmysel, aby vývojár pracoval na vývoji vlastnej knižnice na spracovanie obrazu. Takéto algoritmy môžu byť pomalšie, výpočtovo náročné, a tak by mohol klesať potrebný výkon daného zariadenia.

Medzi najpoužívanejšie knižnice na prácu s obrazom patria OpenCV, Vigra, SimpleCV alebo Halcon. V tejto práci sa na spracovanie obrazu používa spomínaná knižnica OpenCV. Je to voľne šíriteľná knižnica pre akademické aj komerčné účely. Je šírená pod licenciou BSD. Knižnica je veľmi robustná a rozsiahla, primárne je určená na prácu s obrazom. Obsahuje viac ako 2500 optimalizovaných algoritmov, ktoré je možné použiť na kompletnú analýzu obrazu, identifikáciu a sledovanie objektov, rozoznávanie tvárí, segmentáciu objektov od scény obrazu a ďalšie iné. Knižnica bola vyvinutá spoločnosťou Intel a ich prvá verzia knižnice bola vydaná v januári roku 1999. Využíva technológie Intel – Integrated Performance Primitives a je napísaná v programovacom jazyku *C* a *C++*. Podporuje prácu na operačných systémoch ako je Linux, Windows, MacOS a v roku 2012 rozšírila svoju podporu o operačný systém Android. S knižnicou je možné pracovať v rôznych jazykoch, primárne v *C*, *C++*, *Python*, *Java* a obsahuje aj rozhranie pre prácu s *MATLAB*. Funkčnosť ostáva zachovaná, rozdiel je v názvoch jednotlivých metód a funkcií [1].

1.3.2 OpenCV4Android SDK

Ako je písané v predchádzajúcej sekcii, knižnica OpenCV je dostupná pre rôzne platformy. Distribúcia pre platformu Android nesie názov *OpenCV4Android*. Tento balíček je potrebné do vývojového prostredia implementovať, následne je možné knižnicu OpenCV používať viď 2.1. Balíček obsahuje samotnú knižnicu OpenCV, potrebné knižnice a nástroje. Vnútorná štruktúra tohto balíčka *OpenCV4Android* môže vyzerať nasledovne.

```
OpenCV-3.1.0-android-sdk
├── apk
├── sample
├── sdk
│   ├── etc
│   ├── java
│   └── native
│       ├── 3rdparty
│       ├── jni
│       └── libs
│           └── architecture
├── LICENSE
└── README.android
```

Pre správnu funkčnosť aplikácie je potrebné mať na zariadení nainštalovaný tvz. *OpenCV Manager*. Tento manažér zabezpečuje prístup aplikácie k natívnym knižniciam. Aplikácie tak dokážu využívať natívne knižnice OpenCV, ktoré komunikujú priamo s hardvérom daného zariadenia. Manažér je možné nainštalovať na zariadenie manuálne a to stiahnutím manažéra pre príslušnú hardvérovú architektúru daného zariadenia. Tento manažér je voľne dostupný na oficiálnej webovej stránke OpenCV alebo priamo na Google Play, kde sa aplikácia nachádza pod názvom **OpenCV Manager**. Manažér je možné používať aj bez nutnosti inštalácie. To je možné v prípade, ak to samotná aplikácia na spracovanie obrazu dovoľuje – má implementované potrebné metódy [7]. V tabuľke 1.3 je prehľad existujúcich verzií manažérov priradených k jednotlivým hardvérovým architektúram.

Architektúra zariadenia	**Verzia manažéra**
`arm64-v8a`	`OpenCV_3.1.0_Manager_3.10_arm64-v8a.apk`
`armeabi-v7a`	`OpenCV_3.1.0_Manager_3.10_armeabi-v7a.apk`
`Intel x86`	`OpenCV_3.1.0_Manager_3.10_x86.apk`

Tab. 1.3: Prehľad CPU architektúry a príslušného manažéra

1.4 Teória spracovania obrazu

Ak je potrebné spracovávať obraz pomocou určitého zariadenia, je predpoklad, že toto zariadenie obsahuje kameru, ktorá dokáže snímať obraz a následne ho transformovať do digitálnej podoby. Spracovanie obrazu zahŕňa veľké množstvo postupov, metód a algoritmov ako s obrazom pracovať. V dnešnej dobe stále modernejšej techniky sú mobilné zariadenia alebo tablety vybavené štandardne aspoň jednou kamerou, no pokročilé spracovanie obrazu je podstate ešte stále v začiatkoch. Postupom času vznikajú nové možnosti ako využívať a uplatniť počítačové videnie nielen v priemysle. Na trh prichádzajú stále nové technológie, a tak spracovanie obrazu poskytuje stále viac možností využitia. Spracovanie obrazu môže byť v určitých prípadoch výpočtovo dosť náročné, ale výkon zariadení sa stále zvyšuje, preto začína byť tento problém minulosťou. Nezvyšuje sa len výkon, ale aj kvalita jednotlivých periférií, ktoré obraz snímajú. Väčšie rozlíšenie kamery zvyšuje kvalitu pri analýze obrazu.

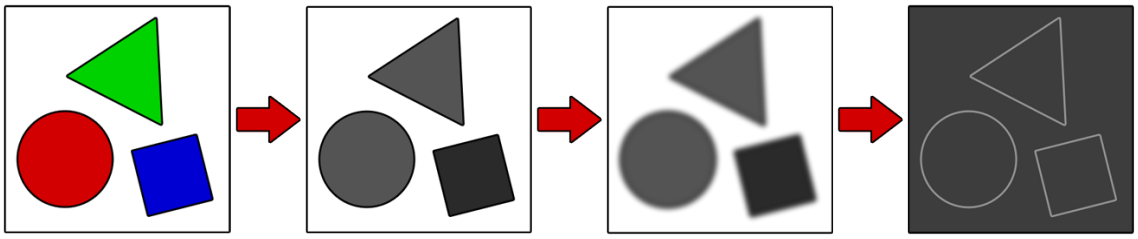
Systém Android podporuje niekoľko možností, ako so snímaním obrazom z kamery pracovať. Samotný systém Android ponúka na spracovanie obrazu API rozhranie so základnými funkciami. Obsahuje základné možnosti na úpravu a vizuálne

efekty obrazu snímaného z kamery zariadenia. Poskytuje rôzne možnosti, ako je napríklad nastavenie kontrastu, jasu a sýtosti, otočenie obrazu alebo efekt rybieho oka. Systém Android vykonáva pri spracovaní obrazu všetky výpočty pomocou GPU (Graphics processing unit) pre získanie maximálneho výkonu. Samotné triedy rozhrania poskytujú teda len obmedzené možnosti a to môže byť pre vývojára nevýhodou. Preto je ďalšou možnosťou pri spracovaní obrazu použitie knižníc, ktoré sú na prácu s obrazom určené. Takéto knižnice obsahujú triedy a metódy, ktoré dokážu využívať rozhranie kamery zariadenia veľmi efektívne. Jednou z týchto knižníc je aj knižnica OpenCV, ktorá obsahuje pokročilé algoritmy, ktoré môže programátor pri vývoji aplikácie na spracovanie obrazu jednoducho používať. Medzi najpoužívanejšie triedy tejto knižnice patria `Imgproc.java`, `Core.java` alebo `Video.java` [9].

- **Imgproc** – Manipulácia s obrazom, aplikovanie rôznych filtrov, detekciu hrán.
- **Core** – Geometrické transformácie, základné operácie s maticami.
- **Video** – Odčítavanie pozadia, detekovanie pohybu v obraze.
- **Highgui** – Poskytuje rozhranie na prácu s videom, čítanie a zápis obrazu.
- **Calib3d** – Kalibrovanie kamery zariadenia, 3D rekonštrukcia obrazu.
- **Features2d** – Detekovanie a popis hrán v obraze.
- **Objdetect** – Sledovanie a detekcia definovaných objektov.
- **GPU** – Nastavenie a získavanie informácií o architektúre a stave hardvéru.

1.4.1 Príprava obrazu na jeho analýzu

Pri zložitejšej práci s obrazom, ako je detekcia objektov, je potrebné snímaný obraz z kamery zariadenia spracovať a pripraviť na detekciu. Na toto slúžia určité metódy a postupy, ktoré je pri detekcií nutné aplikovať. Najpoužívanejšie metódy, ktoré sú používané na úpravu obrazu pri detekcií objektov, sú napríklad Cannyho detektor hrán, Threshold prahovanie alebo Gaussovo rozostrenie.



Obr. 1.4: Ukážka postupu spracovania obrazu pred jeho analýzou

Cannyho detektor hrán

Patrí medzi najpoužívanejší algoritmus pri spracovaní obrazu, ktorý je určený na detekciu hrán v obraze. Detektor hrán je definovaný určitými vlastnosťami, ktoré pre detektor Canny stanovil už v roku 1986 John Canny. Medzi tieto vlastnosti patria nasledovné tri základné požiadavky.

- **Minimálna chyba pri detekcií hrán** - Všetky dôležité hrany v obraze musia byť správne detekované a oblasti, ktoré nie sú hranami, nesmú byť detekované ako hrany.
- **Správna lokalizácia detekcie** - Vzdialenosť medzi skutočnou a detekovanou hranou musí byť čo najmenšia.
- **Správna odozva** - Každá hrana v obraze musí byť detekovaná jedenkrát.

Avšak prvé dve vlastnosti idú proti sebe a preto zlepšením detekčnej schopnosti detektoru zhoršuje jeho schopnosti lokalizácie a opačne. Preto sa odporúča *Canny* použiť v kombinácii *Gaussiánu* a *Sobela*. Tým sa zaručí zníženie obrazového šumu a zvýraznenie hrán.

Threshold prahovanie

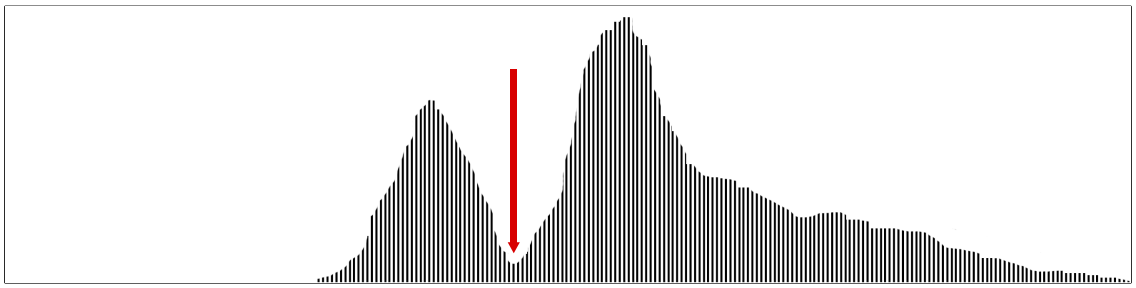
Jedným z najpoužívanejších algoritmov pre segmentáciu obrazu je práve Threshold prahovanie. Je často používaný hlavne pre svoju jednoduchosť implementácie a malú časovú náročnosť pri výpočte. Prahovanie je založené na skutočnosti, že objekty a pozadie majú rozdielnu úroveň intenzity. V algoritme je definovaný prah a každý pixel, ktorý má menšiu hodnotu než tento prah, je určený ako pixel pozadia a všetky ostatné sú považované ako pixely objektu. Výsledkom prahovania je binárny obraz, v ktorom môžu objekty v obraze nadobúdať hodnotu 1, naopak pozadie za objektami hodnotu 0. Prahovanie je teda transformácia obrazu f na binárny obraz g s prahom T (threshold). Pre prahovanie binárneho obrazu $g(x, y)$ potom platí,

$$g(x,y) = \begin{cases} 1 & \text{pre } f(x,y) > T \\ 0 & \text{pre } f(x,y) \leq T \end{cases} . \tag{1.1}$$

Je potrebné zvoliť prahovú hodnotu T správne, inak budú objekty vyhodnotené ako pozadie a zaniknú, alebo naopak budú časti pozadia považované za objekty [8].

Gaussovo rozostrenie

Tento algoritmus sa štandardne používa pre vyhladenie hrán, potlačenie alebo úplné odstránenie šumu v obraze. Rozmazanie obrazu Gaussovým filtrom môžeme realizovať rôznymi spôsobmi, a to napríklad konvolúciou alebo pomocou Furierovej trans-

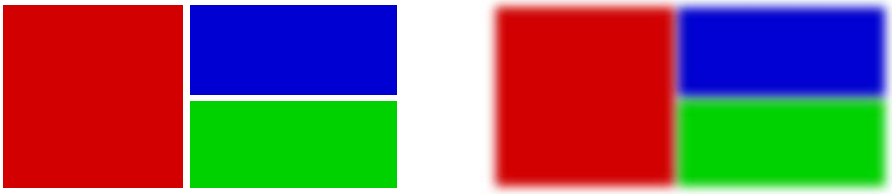


Obr. 1.5: Histogram s viditeľným prahom oddeľujúci tmavé oblasti od svetlých

formácie. Konvolúcia je najjednoduchší spôsob ako použiť Gaussov filter. Základom konvolúcie je konvolučné jadro, ktoré sa posúva po obraze a pre každý pixel sa spočíta výsledná hodnota. Časová náročnosť výpočtu konvolúcie rastie priamoúmerné druhej odmocnine polomeru Gaussového filtru. Konvolučné jadro tohto filtrovania je reprezentované Gaussovou funkciou, ktorá má v 2D nasledujúci tvar,

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{1.2}$$

kde σ je smerodajná odchýlka *Gaussového rozloženia* [3].

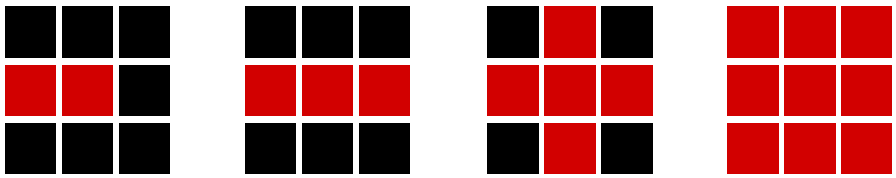


Obr. 1.6: Aplikácia Gaussovova rozostrenia na vstupný obraz

Erózia

Erózia sa používa na zjemnenie hrán detekovaných hrán obrazu. Aplikovaním tohto algoritmu je možné tiež odstrániť šum v obraze. Erózia sa dá popísať ako vektorový rozdiel množín X a B 1.3, kde X je vstupný binárny obraz a B je štruktúrny element. Tento obrazový element je zobrazený na obrázku 1.7.

$$X \ominus B = p \in \varepsilon^2 : p + b \in X \, \forall \, b \in B \tag{1.3}$$

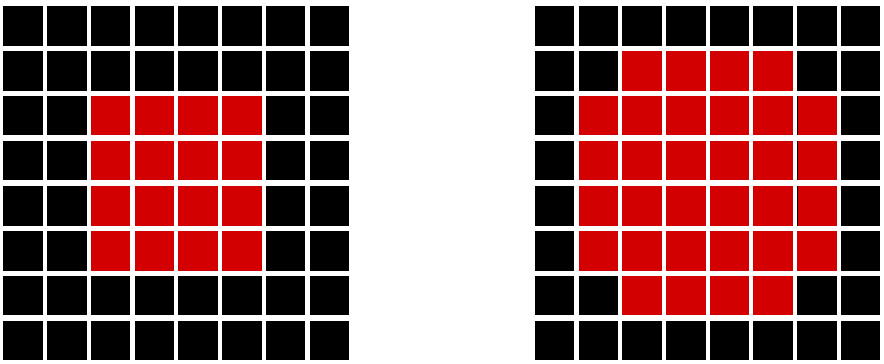


Obr. 1.7: Jednoduché štrukturálne elementy erózie a dilatácie

Dilatácia

Úlohou aplikovania dilatácie na vstupný obraz je zvýraznenie hrán objektu, ale nastáva aj zvýraznenie šumu v obraze. Erózia a dilatácia nie sú navzájom inverzné operácie. Dilatáciu môžeme vyjadriť ako vektorový súčet množín X a B 1.4, kde X je opäť vstupný binárny obraz a B je štruktúrny element.

$$X \oplus B = p \in \varepsilon^2 : p = x + b, x \in X\, b \in B \tag{1.4}$$



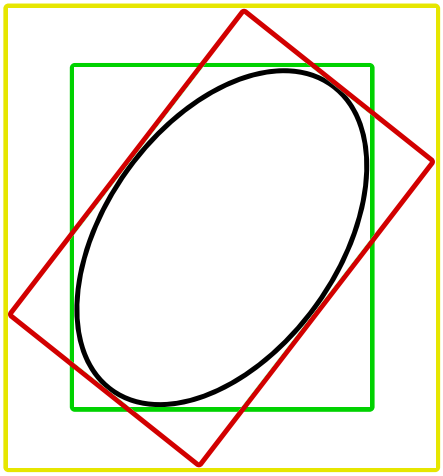
Obr. 1.8: Výsledok aplikácie dilatácie na vstupný obraz

1.4.2 Kontúry v obraze

Kontúry sú množina bodov, ktoré reprezentujú uzavretú krivku ohraničujúcu časť alebo celý objekt v obraze. Často sú využívané v rôznych algoritmoch pre analýzu alebo rozoznávanie objektov. Platí teda tvrdenie, že kontúrami je možné jednoducho popísať tvar predmetu v obraze. Kontúry sú štandardne vyhľadávané v binárnom obraze, ktorý by mal byť na detekciu pripravený. Knižnica OpenCV poskytuje funkcie na detekciu kontúr v obraze. Nájdené kontúry sú reprezentované ako zoznam absolútnych súradníc bodov.

Ohraničujúci obdĺžnik

Je to špeciálny druh kontúry v knižnici OpenCV nazývaný ako *bounding boxes*, ktorý sa používa v prípadoch, kedy je potrebné vyznačiť detekovaný objekt ako celok a nie je tak potrebné vyznačovať jeho presné hranice. Každý vyznačený objekt v obraze pomocou *bounding boxes* je obdĺžnik, ktorý ohraničuje všetky kontúry daného detekovaného objektu [9].



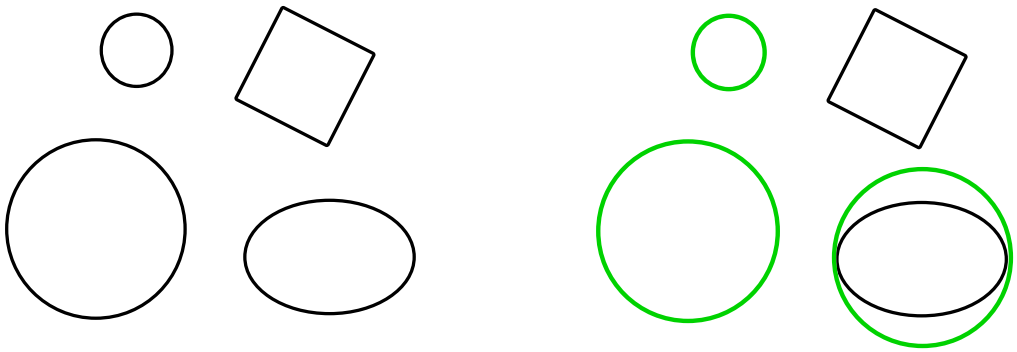
Obr. 1.9: Možná aplikácia *bounding boxes* na detekované kontúry objektu

1.4.3 Houghova transformácia

K detekcií jednoduchých geometrických tvarov, ako sú priamky alebo kružnice sa často využíva Houghova transformácia. Z počiatku bola táto transformácia určená len pre detekciu priamok, ale neskôr bola transformácia rozšírená aj o identifikáciu ľubovoľných tvarov, ako sú kružnice a elipsy. Táto metóda je založená na princípe znalosti matematického popisu hľadaného geometrického tvaru. Pre Houghovú transformáciu platí nasledujúci vzťah 1.5

$$(x - x_{center})^2 + (y - y_{center})^2 = r^2, \tag{1.5}$$

kde $(x - x_{center})$ je stred a r je polomer hľadaného kruhu. Zo vzťahu je možné vidieť, že na Houghovú transformáciu sú potrebnú iba 3 parametre, čo môže byť dosť neefektívne a nepresné. Preto knižnica OpenCV využíva aj zložitejšiu metódu *Hough Gradient Method*, ktorá naviac pracuje s informáciami o hranách detekovaných objektoch v obraze. [3]



Obr. 1.10: Houghová transformácia na hľadanie kruhových objektov v obraze

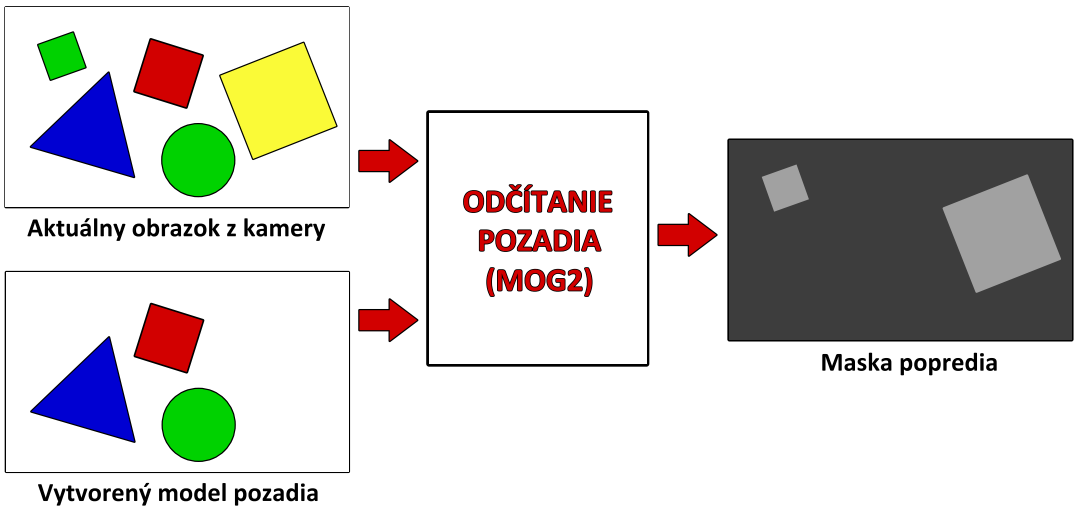
1.4.4 Segmentácia obrazu

Pod pojmom segmentácia sa v oblasti spracovania obrazu rozumie oddelenie objektov od pozadia obrazu, prípadne rozlišovať viacero objektov od seba. Rozlišujú sa dve hlavné metódy segmentácie pri analýze obrazu. Je to segmentácia v **statickom** a **dynamickom** obraze. Segmentácia v statickom obraze je jednoduchšia, nie je vyžadovaná vysoká rýchlosť segmentácie. Na detekciu objektov sa v tomto obraze používajú často rôzne detektory hrán, ktoré sú opísané v sekcii 1.4.1. Na segmentáciu v dynamickom obraze sa používa metóda odčítania pozadia. Pri tomto type obrazu je kladený dôraz na výpočtovú rýchlosť odčítania pozadia od objektov, keďže sa jedná o sériu obrázkov. Metóda odčítania pozadia je vysvetlená v nasledujúcej sekcii 1.4.4.

Odčítanie pozadia

Segmentácia obrazu metódou odčítania pozadia je technika, určená pre analýzu dynamického obrazu. Princípom tejto metódy je identifikovať každý pixel v obraze a vyhodnotiť jeho príslušnosť k objektom záujmu alebo pozadiu scény obrazu. Odčítaním pozadia zo scény obrazu získame objekty záujmu. Výsledkom tejto operácie je binárny obraz, takzvaná maska popredia (*foreground mask*). Pokročilejšie techniky odčítania pozadia scény obrazu dokážu identifikovať výskyt tieňa, ktorý detekovaný objekt spôsobuje, prípadne tento tieň úplne odfiltrovať.

K odčítaniu pozadia dochádza medzi aktuálnom snímkou videa a modelom pozadia, ktorý je statický alebo dynamický, teda adaptívny. Statický model pozadia je veľmi citlivý na každú zmenu, nedokáže sa adaptovať na zmeny v scéne obrazu, ako sú napríklad zmeny osvetlenia. Odčítaním pozadia so statickým modelom znamená, že každá zmena scény v obraze bude vyhodnotená ako nový objekt. Naopak, adaptívne modely sú výhodnejšie v tom, že dokážu pracovať s rušivými vplyvmi,



Obr. 1.11: Princíp segmentácie objektov od scény obrazu

ako je napríklad zmena osvetlenia alebo nečakaný príchod nového objektu do scény obrazu. Pri objavení neočakávaného objektu v scéne obrazu bude tento objekt detekovaný ako objekt záujmu, ale za určitý čas, ktorý je potrebný na adaptáciu, tento objekt zanikne a stane sa súčasťou pozadia scény obrazu. Na odstránenie pozadia sa štandardne používajú rôzne techniky segmentácie obrazu, z nich najjednoduchšia je odčítavanie dvoch po sebe nasledujúcich snímok. Ako pokročilé metódy na segmentáciu obrazu sú používané takzvané štatistické modelovače pozadia, medzi ktoré patrí *MOG* (Mixture of Gaussians), *GMG* (Gaussian Mixture Model) alebo *KNN* (K-Nearest Neighbors) [5].

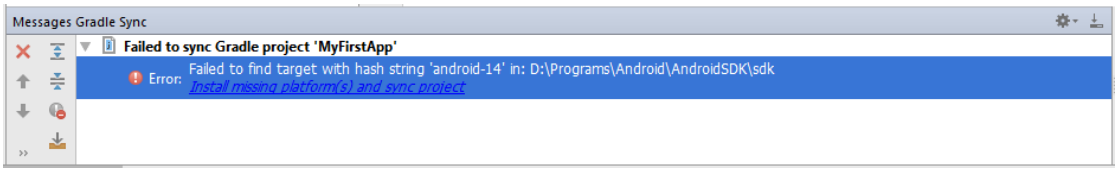
2 NÁVRH A REALIZÁCIA RIEŠENIA PRÁCE

Táto kapitola je venovaná praktickej časti bakalárskej práce. Je tu opísaný konkrétny postup riešenia problematiky tejto práce, implementácia knižnice OpenCV do vývojového prostredia, práca s knižnicou pri spracovaní obrazu z kamery mobilného zariadenia a následná implementácia postupov a metód na riešenie konkrétnych problémov pri detekcii špecifických objektov v spracovanom obraze.

2.1 Implementácia knižnice OpenCV do vývojového prostredia

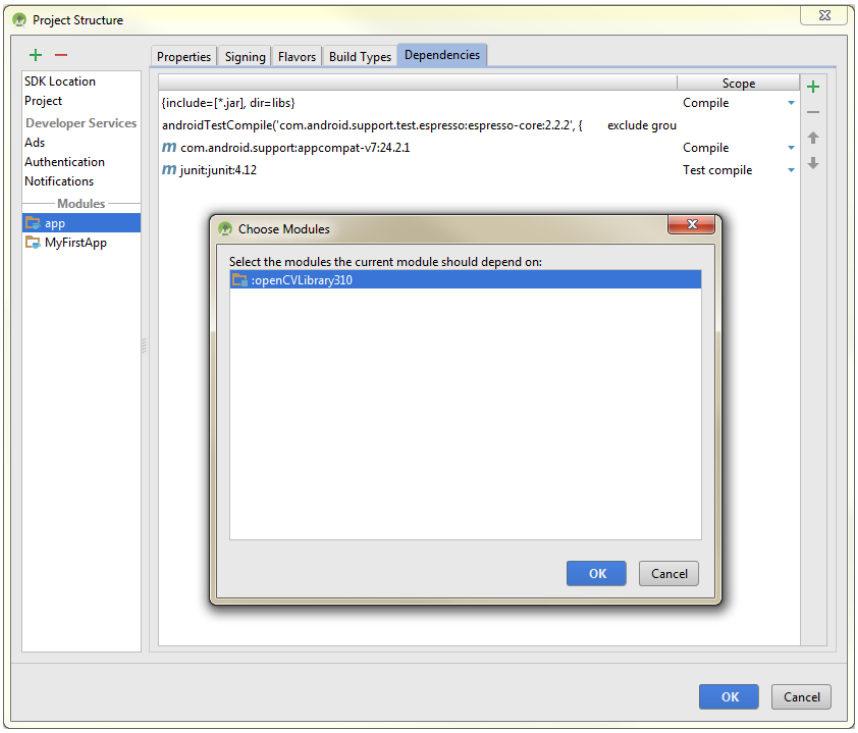
Táto sekcia je sprievodcom implementácie knižnice OpenCV do vývojového prostredia Android Studio. Existuje viacero možností, ako túto knižnicu do prostredia implementovať. Každá z možností má svoje výhody, a naopak nevýhody v podobe časovej náročnosti. V tejto sekcii je spomenutý jeden z možných spôsobov. Taktiež sa tu nachádzajú postupy na riešenie problémov, ktoré môžu pri implementácií nastať.

Pred samotnou implementáciou je potrebné knižnicu OpenCV stiahnuť z dostupného webového portálu `www.opencv.org/platforms/android`. Pracovné súbory knižnice je možné nájsť v sekcii *releases*, kde sa nachádza prehľad všetkých dostupných verzií knižnice OpenCV spolu s príslušnou dokumentáciou. Je lepšie stiahnuť si najnovšiu dostupnú verziu, ktorá môže obsahovať opravené chyby, ktoré sa vyskytovali v starších predchádzajúcich verziách tejto knižnice. Ďalšou výhodou je skutočnosť, že najnovšie verzie tejto knižnice prinášajú častokrát nové funkcie, prípadne optimalizácie v podobe rýchlejších a efektívnejších algoritmov pre prácu s obrazom. Po stiahnutí je potrebné knižnicu rozbaliť do ľubovoľnej zložky na pracovné miesto (*Workspace*) vývojového prostredia, kde sa nachádza vytvorený projekt (`/WorkSpace/MyApp/OpenCV-android-sdk`). Po rozbalení sa vo vývojovom prostredí Android Studio klikne do menu na `File > New > Import Module`. Následne sa otvorí okno *New Module*, kde sa v položke *Source directory* zvolí cesta ku knižnici `/OpenCV-android-sdk/sdk/java`. Je možné si všimnúť, že prostredie automaticky registruje prítomnosť knižnice OpenCV a v kolonke *Module name* sa automaticky do-



Obr. 2.1: Chybové hlásenie vývojového prostredia

plnil názov tohto modulu ako *openCVLibrary310*. Následne, po kliknutí na tlačítko `Next` sa otvorí nové okno, kde sa nachádzajú čekboxy (*jars*, *libraries* a *import options*), ktoré musia byť zaškrtnuté. Kliknutím na tlačítko `Finish` sa dokončí nastavenie importovaného modulu OpenCV. Následne vývojové prostredie automaticky importuje knižnicu do aktuálneho projektu, všetky súbory sú synchronizované. Po dokončení tohto procesu sa zobrazí výstupné hlásenie v súbore `import-summary.txt`. Toto hlásenie informuje o stave importovania modulu do vývojového prostredia, zobrazuje ktoré súbory boli do prostredia importované. Súbor je možné zavrieť, knižnica je úspešne implementovaná. Následný postup bude venovaný konfigurácií knižnice, pre správnu funkčnosť. Je možné si všimnúť, že zlyhala konfigurácia projektu, podľa chybovej hlášky v okne *Messages Gradle Sync*, v dolnej časti vývojového prostredia, viď obrázok 2.1. Gradle vyhlásil chybu, pretože `build.gradle`



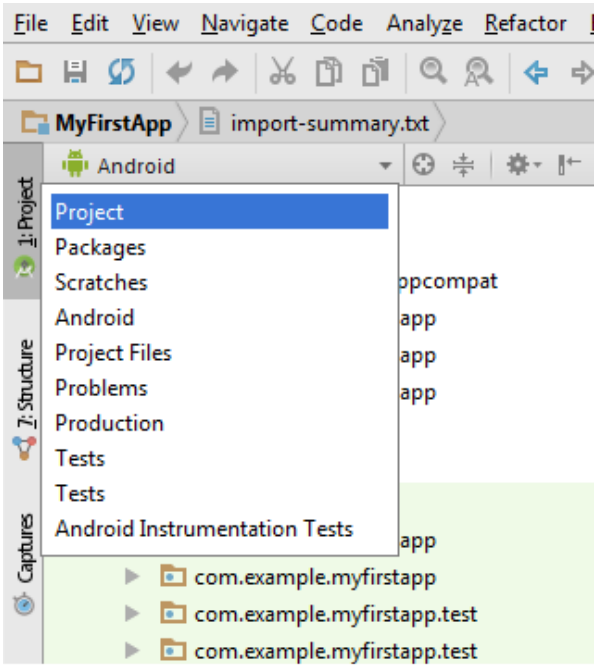
Obr. 2.2: Inicializácia knižnice OpenCV ako modul vo vývojovom prostredí

súbor je potrebné synchronizovať, následne prekladať s potrebným podporovaným API rozhraním, v tomto prípade s verziou API 14. Vývojové prostredie toto rozhranie implicitne neobsahuje, preto je potrebné ho stiahnuť a inštalovať kliknutím na `Install missing platform(s) and sync project`.

Knižnica OpenCV je v projekte vložená, napriek tomu s ňou zatiaľ nie je možné pracovať. Je potrebné ju v prostredí inicializovať ako nový modul. Kliknutím na menu v `File > Project Structure` sa otvorí nové okno *Project Structure*. V tomto

okne sa klikne v kolónke *Modules* na `app`, následne kliknúť na záložku `Dependencies`. V tejto záložke okna *Project Structure* kliknutím vpravo hore na ikonku zelené plus (*Add*) je potrebné vybrať možnosť *Module Dependency*. Otvorí sa nové okno *Choose Modules*, viď obrázok 2.2, ktoré ponúka na výber modul s názvom *:openCVLibrary310*. Je potrebné vybrať tento modul, potvrdiť a následne je možné projekt znova synchronizovať.

Teraz bude potrebné nastaviť súbory typu `build.gradle`. Projekt obsahuje viac súborov `build.gradle`. Ten ktorý je potrebný upraviť zahŕňa základné nastavenia o verzií prekladača a SDK, ktoré projekt používa. Prehľad všetkých súborov, z ktorých sa projekt skladá, je možné vidieť v stromovej štruktúre v ľavej časti prostredia. Aby bolo možné s týmto typom súborov zaobchádzať, je potrebné prepnúť pohľad zobrazenia v stromovej štruktúre projektu. Kliknutím na položku *Android* v stromovej štruktúre sa zobrazí ponuka, z ktorej sa vyberie *Project*, viď obrázok 2.3. V tomto náhľade projektu vyhľadáme v stromovej štruktúre `MyFirstApp/app`, na-



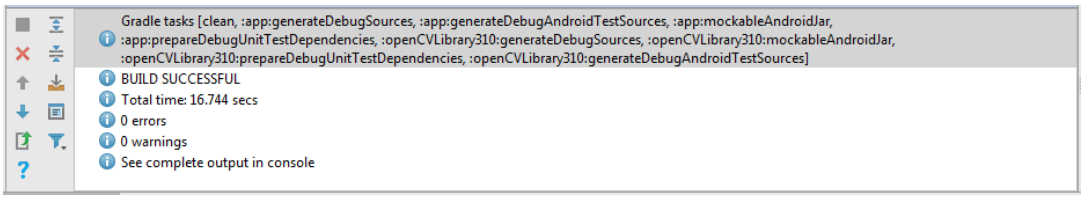
Obr. 2.3: Zmena náhľadu štruktúry projektu

chádza sa tu spomínaný súbor `build.gradle`, ktorý sa dvojklikom otvorí. Teraz je potrebné nájsť druhý súbor, ktorý definuje nastavenia knižnice OpenCV pridaného modulu *openCVLibrary310*. Súbor je možné nájsť v stromovej štruktúre projektu ako `MyFirstApp/MyFirstApp/openCVLibrary310/src/main`, preto je nutné zmeniť náhľad z *Project* na *Project Files*. Opäť je potrebné nájdený súbor `build.gradle` dvojklikom otvoriť. Súbory sú otvorené a pripravené na konfiguráciu ich nastavení. Je potrebné dodržať vzájomnú zhodu nastavení jednotlivých hodnôt v súboroch,

inak opätovná synchronizácia `build.gradle` súborov neprebehne úspešne. Preto je potrebné nasledujúce hodnoty nastaviť v oboch súboroch rovnako.

- compileSdkVersion 23
- buildToolsVersion 23.0.2
- minSdkVersion 8
- targetSdkVersion 23

Po nastavení hodnôt je potrebné súbory uložiť a následne kliknúť v pravo hore na ponuku `Sync Now`. Po úspešnej synchronizácií projektu je možné celý projekt preložiť a to kliknutím na `Build > Clean Project`. Ak je všetko nastavené správne, projekt by sa mal úspešne preložiť bez chybových hlásení a s nulovými upozorneniami. Tento výstup po preložení projektu je možné vidieť v spodnej časti vývojového prostredia, viď obrázok 2.4. Teraz je potrebné presunúť sa v stromovej štruktúre z náhľadu

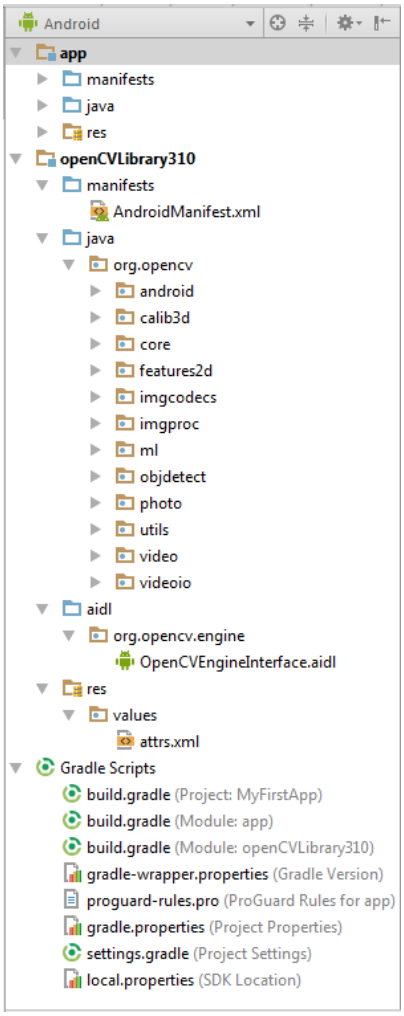


Obr. 2.4: Výstup programu po preložení

Project Files späť na pôvodný náhľad *Android*. Kliknutím v štruktúre projektu na adresár `openCVLibrary310` sa zobrazí jeho vnútorný obsah. Je dôležité skontrolovať, či tento adresár obsahuje všetky potrebné súbory a adresáre (viď obr 2.5), ktoré zaručujú správnu funkčnosť implementovanej knižnice OpenCV. Ak je všetko v poriadku, nasleduje kontrola ďalšieho `build.gradle` súboru, ktorý nesie názov `build.gradle (Module: openCVLibrary310)`. Tento súbor je možné nájsť v stromovej štruktúre pod adresárom `Gradle Scripts`. Ak sa tento súbor v štruktúre nenachádza, chyba je vo vývojovom prostredí. Niektoré súbory nie sú stále korektne synchronizované. Riešením tohto problému je opätovná synchronizácia projektu a následný reštart vývojového prostredia. Tento postup sa opakuje aj v prípade neprítomnosti ďalších súborov v projekte.

Ako posledná vec, ktorú je potrebné pri implementácii knižnice vykonať je kopírovanie menších knižníc, ktoré samotná knižnica OpenCV obsahuje. Knižnice sa líšia podporou rôznych hardvérových architektúr, no jadro všetkých funkcií ostáva zachované. Je potrebné opäť zmeniť pohľad v stromovej štruktúre na *Project Files*, následne sa presunúť do `MyFirstApp/OpenCV-android-sdk/sdk/native`, kde sa nachádza priečinok `libs`. Tento priečinok sa následne prekopíruje do adresára

v `MyFirstApp/OpenCVLibrary310/src/main`. Prekopírovaný priečinok `libs` je potrebné premenovať na `jniLibs`. Projekt je možné znova synchronizovať, aby sa prejavili nové zmeny v projekte kliknutím na `Tools > Android > Sync Project with Gradle Files`. Po úspešnej synchronizácii projektu je knižnica korektne implementovaná a tak pripravená na použitie. Nasledujúca kapitola je venovaná použitiu knižnice OpenCV na spracovanie obrazu.



Obr. 2.5: Náhľad stromovej štruktúry

2.2 Spracovanie obrazu pred jeho analýzou

Spracovanie obrazu je veľmi dôležitý proces, ak je potrebné snímaný obraz ďalej analyzovať. To znamená, že zosnímaný obraz je nutné pripraviť tak, aby bolo možné s obrazom čo najlepšie ďalej pracovať.

2.2.1 Použitie obrazových filtrov

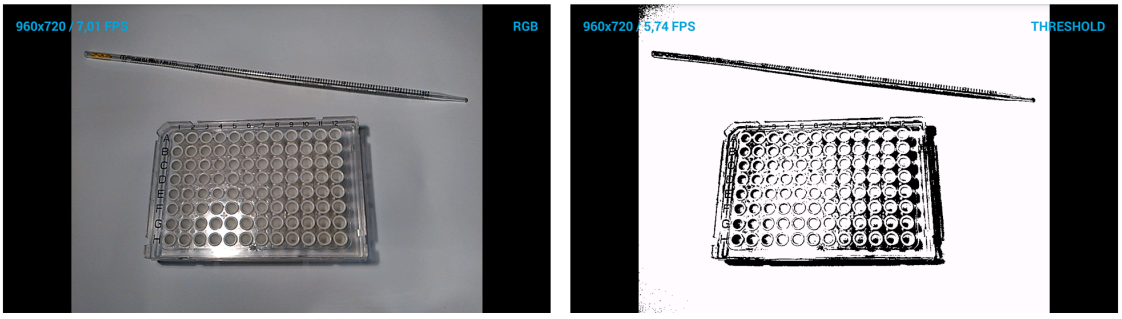
Dôležitou časťou pri analýze obrazu je použitie rôznych obrazových filtrov. Najčastejšie je to konvertovanie farieb, vyhladenie šumu v obraze, alebo prahovanie na zvýraznenie hrán, ktoré sú v tejto práci často používané.

Konvertovanie farieb

Jedna zo základných operácií, ktorá je v práci používaná, je metóda na konvertovanie farieb v obraze. Táto metóda sa používa volaním `cvtColor()` z triedy `Imgproc.java`. Ako vstupné parametre metóda očakáva vstupný a výstupný obraz v tvare matice, číslo kódovania farieb a počet kanálov výstupného obrazu. Použitie tejto metódy môže vyzerať nasledovne `cvtColor(rgba, gray, COLOR_RGB2GRAY);`

Prahovanie obrazu

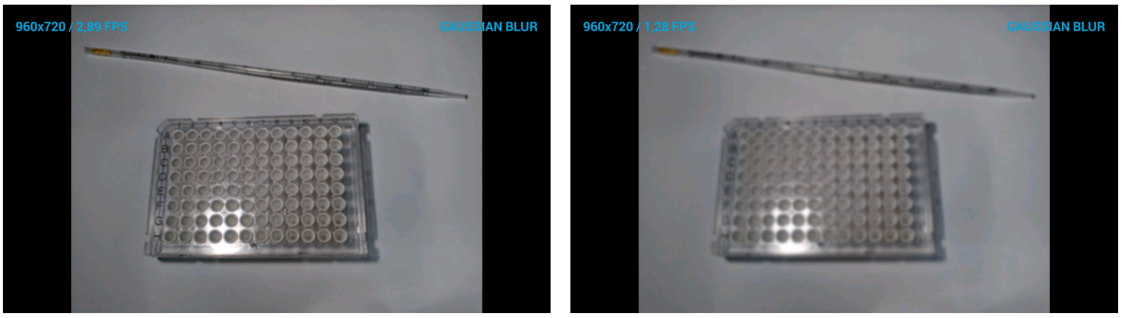
Ak je potrebné rozlišovať hrany v obraze, je možné použiť Threshold prahovanie. Táto metóda je v práci často používaná pri vyhľadávaní kontúr v obraze. Základný threshold ale v práci nie je použitý, pretože nevie reagovať na zmenu svetelných podmienok. Preto je v práci použitý adaptivný threshold, ktorý sa vie prispôsobiť na zmenu intenzity svetla v obraze. To je veľmi dôležité, pretože pohybom ruky s pipetou pri zariadení sa prirodzene mení intenzita svetla. Zavolaním metódy `adaptiveThreshold()`, ktorá patrí do triedy `Imgproc.java`, je vstupný obraz prahovaný adaptivným threshold algoritmom. Pred zavolaním samotnej metódy `adaptiveThreshold()`, je potrebné v obraze konvertovať farby do sivej, inak by prahovanie v tomto prípade nedávalo zmysel, viď sekcia 1.4.1. Metóda na prahovanie obrazu je v práci používaná na rozoznanie hrany pipety, tým je možné presnejšie detekovať jej kontúry. Prahovú hodnotu threshold je možné prispôsobiť v nastavení aplikácie v sekcií *Pipette Settings*.



Obr. 2.6: Aplikácia treshold prahovania na farebný RGB obraz

Rozostrenie obrazu

Pri zhoršených svetelných podmienkach môže vznikať obrazový šum, prípadne iné chyby obrazu. Na potlačenie tohto javu sa v práci používa Gaussovo rozostrenie, ktoré sa aplikuje zavolaním metódy `GaussianBlur()`. Táto metóda patrí opäť do triedy `Imgproc.java`. V tejto metóde je dôležitá hodnota *ksize*, ktorá je dátového typu *Point*, a preto môže nadobúdať hodnoty *ksize.width* a *ksize.height*, ktoré musia byť kladné a nepárne. Hodnoty udávajú veľkosť jadra Gaussového rozostrenia, viď obrázok 2.7. Metóda pre rozostrenie obrazu je používaná ešte pred jeho analýzou v algoritme na detekciu jamiek v mikrodoštičke. Hodnota rozostrenia *ksize* sa dá nastaviť v nastavení aplikácie v sekcii wells settings.



Obr. 2.7: Porovnanie metódy *GaussianBlur()* s rôznymi hodnotami rozostrenia

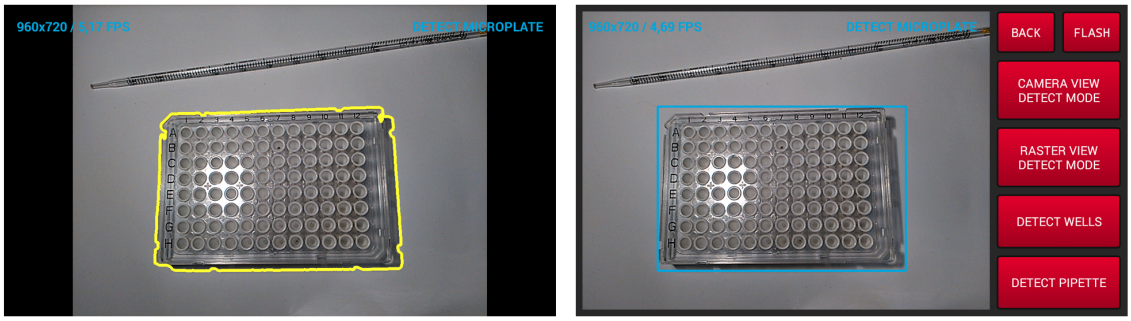
2.3 Detekcia špecifických objektov v obraze

Táto sekcia je venovaná algoritmom, ktoré sú používané na analýzu obrazu. V práci sú použité 3 hlavné výpočtové algoritmy. Prvý algoritmus slúži na detekciu jamiek v mikrodoštičke, druhý sleduje pozíciu pipety v obraze a vypočítava jej špičku. Posledný z algoritmov vyhodnocuje pozíciu špičky pipety a jej výskyt v jamke mikrodoštičky. V nasledujúcich sekciách sú jednotlivé algoritmy podrobnejšie vysvetlené.

2.3.1 Detekcia mikrodoštičky v snímanom obraze

Jednou z dôležitých metód, ktoré sa v práci používajú, je práve metóda `findMicroplateInFrame()`. Úlohou tejto metódy je analyzovať snímaný obraz a následne určiť oblasť záujmu tzv. ROI (Region Of Interest), v ktorej budú kruhy vyhľadávané. To znamená, že metóda dokáže v obraze rozlíšiť mikrodoštičku, zistiť jej pozíciu a túto oblasť vyznačiť. Následne pri detekcii kruhov už nie je potrebné kontrolovať celú oblasť scény obrazu. Detekcia prebieha len v zistenej oblasti ROI. Tým sa znižuje

riziko detekcie nežiaducich kruhov mimo mikrodoštičky, pretože detekcia je obmedzená len na túto oblasť. To prináša ďalšiu výhodu. Zmenšením oblasti vyhľadávania vedie k šetreniu výpočtového výkonu zariadenia. Princíp činnosti tejto metódy spočíva vo vyhľadaní kontúr v obraze a ich následnej analýze. Po aplikovaní tejto metódy sú v obraze vyhľadané všetky kontúry. Tie sú postupne analyzované. Algoritmus vyhľadá kontúru s najväčšou plochou, ktorá patrí mikrodoštičke. Takúto kontúru je možné vyhľadať pomocou metódy `contourArea()`. Ak je najväčšia kontúra vyhľadaná, je potrebné zistiť jej pozíciu v obraze. Pre získanie tejto pozície slúži metóda `boundingRect()`, ktorá vytvorí obdĺžnik obopínajúci hranice získanej kontúry. Následne je už možné vypočítať pozíciu a rozmery detekovanej doštičky. Oblasť je tak možné vyznačiť a detekcia jamiek prebieha len v tejto oblasti, viď obrázok 2.8.

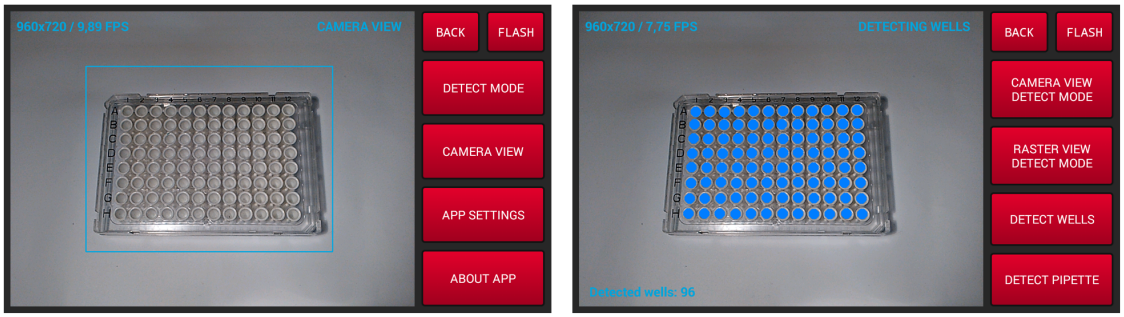


Obr. 2.8: Detekcia mikrodoštičky v snímanom obraze z kamery zariadenia

2.3.2 Houghova transformácia a detekcia jamiek v mikrodoštičke

Ďalšou dôležitou metódou, ktorá sa v práci nachádza je `detectCirclesInFrames()`. Úlohou tejto metódy je analyzovať kruhy v spracovanom obraze. Tento algoritmus musí byť spoľahlivý, aby boli pri detekovaní nájdené všetky jamky v snímanej mikrodoštičke. Ďalšou požiadavkou pri tomto algoritme je, aby neboli detekované nežiadúce kruhové objekty mimo doštičky. Toto zabezpečuje metóda `findMicroplateInFrame()`, ktorá detekuje okraje mikrodoštičky a následne určí jej polohu v snímanom obraze viď 2.3.1. Tým je možné detekovať kruhy len v určitej oblasti záujmu ROI, v tomto prípade v oblasti mikrodoštičky. Samotná detekcia kruhov nie je náročná. Na detekovanie kruhov v obraze slúži metóda `HoughCircles()`. Pri implementácii tejto metódy sú potrebné parametre ako je vstupný binárny obraz, matica pre ukladanie nájdených kruhov a ďalším dôležitým parametrom tejto funkcie môže byť typ metódy vyhľadávania kruhov *Hough Gradient Method*, viď 1.4.3. Tento parameter

je celočíselná konštanta, ktorá patrí do triedy `Imgproc.java`. Hodnotu je možné získať zavolaním *Imgproc.CV_HOUGH_GRADIENT*. Nasledujúce parametre už len matematicky bližšie špecifikujú hľadaný objekt v obraze, napríklad minimálny a maximálny polomer hľadaného kruhu, alebo vzdialenosť medzi kruhmi. Tieto hodnoty môže užívateľ v aplikácií jednotlivo nastaviť, alebo je možné zvoliť očakávaný počet jamiek v mikrodoštičke a systém sám vyberie optimálne hodnoty nastavenia pre dané jamky. Keďže algoritmus detekuje jamky na viacerých obrázkoch, môžu tak vznikať duplicitné jamky. To znamená, že na jednej pozícii v obraze existuje niekoľko jamiek. Preto je potrebné implementovať metódu, ktorá by riešila tento problém. V práci sa táto metóda nachádza pod názvom `sortingDetectedCirclesCenter()`. Metóda optimalizuje všetky detekované kruhy. To znamená, že všetky kruhy, ktoré sa nachádzajú blízko seba v určitej medzi sú navzájom spriemerované a stáva sa z nich jeden kruh.



Obr. 2.9: Detekcia jamiek v mikrodoštičke aplikovaním Houghovej transformácie

2.3.3 Segmentácia obrazu a detekcia pozície pipety

V tejto sekcii je opísaný postup riešenia segmentácie obrazu, potrebný na detekovanie pipety v spracovanom obraze. Tento algoritmus je zložitejší ako detekcia jamiek v mikrodoštičke. Obraz je potrebné spracovať a pripraviť na analýzu. Následne je možné obraz segmentovať. To znamená, že je potrebné v reálnom čase oddeliť pohybujúcu sa pipetu od pozadia scény obrazu.

V práci je na segmentáciu obrazu použitý algoritmus *MOG2*, ktorý je implementovaný v triede `BackgroundSubtractorMOG2.java`. Princíp tohto algoritmu je vysvetlený v teoretickej časti viď 1.4.4. Algoritmus vytvára určitý model pozadia z posledných N prichádzajúcich snímok z kamery zariadenia. Počet snímok z ktorých sa model pozadia vytvára je možné nastaviť hodnotou *history*. Čím je väčší počet snímkov, tým je väčšia výpočtová náročnosť na zostavenie modelu pozadia. Táto hodnota je v práci nastavená na 256 snímkov, čo je optimálna hodnota, ktorá

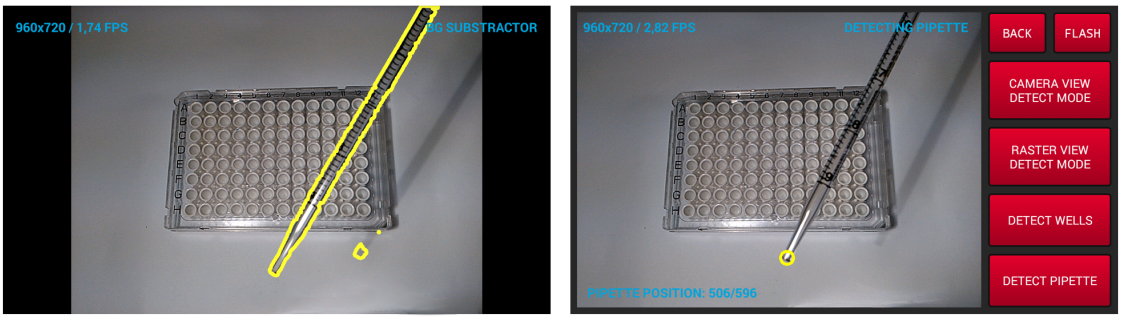
dostačuje pri detekcii pipety. Väčšia hodnota počtu snímkov nedáva veľký zmysel, pretože pipeta je väčšinu času v pohybe, a tak nie je možné, aby sa stihla adaptovať a stala sa súčasťou pozadia. Pipeta čaká na jednej pozícii pri pipetovaní maximálne v jednotkách sekúnd, preto je zvolená hodnota vyhovujúca. Ďalším parametrom tejto metódy je *varThreshold*, ktorou sa nastavuje prahovanie obrazu pri jeho odčítavaní. Posledný parameter metódy je dátového typu *boolean*, ktorým sa rozhoduje o potlačení tieňa, ktorý generujú objekty v obraze. Na obrazku 2.10 je názorná ukážka ako *MOG2* algoritmus oddeľuje pipetu od pozadia scény obrazu. Po segmentácií obrazu

Obr. 2.10: Segmentácia spracovaného obrazu aplikovaním *MOG2* algoritmom

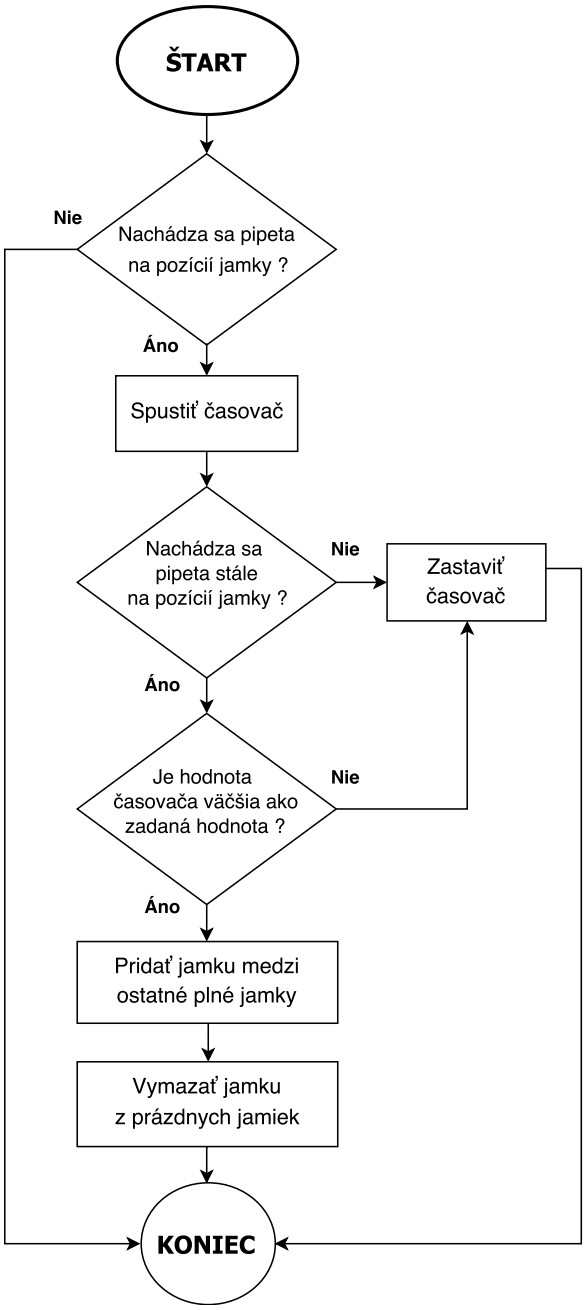
je získaný tvar pipety, ktorý je oddelený od pozadia, ale obsahuje malé chyby, ako je obrazový šum. Tieto chyby je potrebné odstrániť, pretože môžu ovplyvňovať následný výpočet špičky pipety. Na získaný tvar pipety je potrebné aplikovať metódu `GaussianBlur()`, ktorá potlačí obrazový šum. Následne je možné aplikovať metódu na prahovanie obrazu `threshold()`. Výsledný obraz po aplikácii thresholdu, je potrebné ďalej spracovať. Môže sa stať, že Gaussové rozostrenie nerozmazalo všetok šum v obraze, preto je potrebné aplikovať metódy `erode()` a `dilate()`. Poradie musí byť dodržané. Ako prvé musí byť obraz erodovaný (*erode*), následne až potom dilatovaný (*dilate*). Dôvod tejto aplikácie je vysvetlený v teoretickej časti, viď 1.4.1. Výsledkom je tvar pipety, ktorý obsahuje malé množstvo šumu. Hrany pipety sú viditeľne zosilnené, následne je možné nájsť obrazové kontúry pipety. Pre hľadanie kontúr v obraze je v práci použitá metóda `findContours()`, viď 1.4.2. Posledná dôležitá metóda, ktorá je v práci použitá pri detekcii pipety je metóda `findBottommostPointROIFrame()`, ktorá pracuje s vyhľadanými kontúrami v obraze. Táto metóda spracuje všetky nájdené kontúry a vracia absolútnu pozíciu bodu najnižšej kontúry v obraze. Tento bod je teda špička pipety, viď obrázok 2.11.

2.3.4 Algoritmus pre detekciu pozície pipety v jamke mikrodoštičky

Táto sekcia je venovaná algoritmu, ktorý tvorí jadro každej metódy, kde je potrebné detekovať, či sa pozícia pipety zhoduje s pozíciou niektorej z jamiek mikrodoštičky. Algoritmus teda dokáže v reálnom čase vyhodnocovať pozíciu pipety a kontrolovať jej pozíciu s pozíciou jamky. Skladá sa z niekoľkých dôležitých častí, ktoré medzi sebou tvoria logickú závislosť viď, 2.12. Patrí sem kontrolná časť, časovač a časť pre stanovanie výsledku výpočtu. Kontrolná časť je permanentne aktívna, tým je dosiahnutá stála kontrola pozície pipety v obraze, ktorá je následne porovnávaná s pozíciou jamiek v mikrodoštičke. Ak sa pipeta dostane do blízkosti jamky, nastane zhoda ich pozície, ktorá aktivuje časovač. Ak sa pozícia pipety zmenila a jej nová pozícia nie je v rámci tolerancie, algoritmus tento stav vyhodnotí ako úplnú zmenu pozície pipety. Následne sa resetuje časovač a algoritmus sa vracia na začiatok ku kontrole zhody pozície pipety s jamkou mikrodoštičky. Ak je pozícia pipety konštantná, prípadne sa pozícia pipety zmení len v jednotkách pixelov v určitej tolerancií, algoritmus tento stav vyhodnotí a pokračuje ďalej ku kontrole časovača. V tejto časti je kontrolovaný čas časovača s danou hodnotou času, ktorá je nastavená užívateľom v nastavení aplikácie. To znamená, že pipeta musí zotrvať na mieste určitý čas, ktorý je daný užívateľom, aby algoritmus mohol vyhodnotiť stav, ktorý simuluje naplnenie konkrétnej jamky v mikrodoštičke. Následne je daná jamka vymazaná zo zoznamu prázdnych a pridaná do zoznamu plných jamiek. To už záleží na tom, v ktorom zobrazovacom móde je algoritmus implementovaný. Princíp algoritmu je jednoduchý. Výhodou je možnosť jeho modifikácie, preto je algoritmus v aplikácií využívaný pri každej forme detekcie. Podrobnejší popis správania algoritmu pri zhode pipety s jamkou je opísaný v ďalšej sekcií, viď 2.4.



Obr. 2.11: Metóda na získanie pozície najnižšieho bodu kontúr pipety



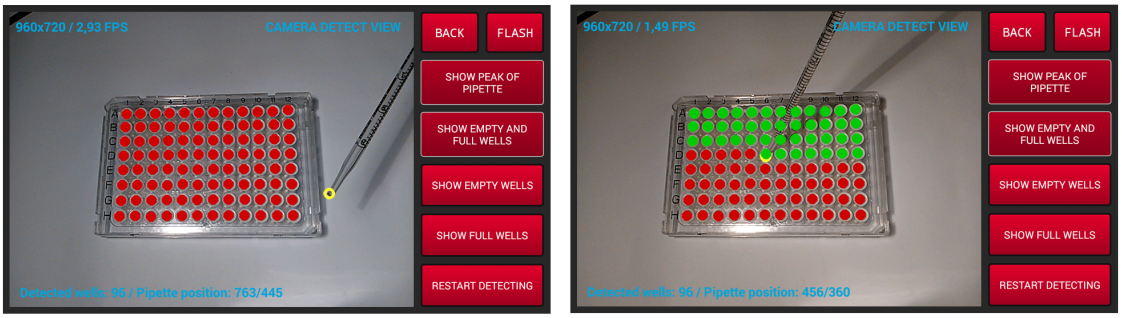
Obr. 2.12: Vývojový diagram algoritmu pre vyhodnotenie zhody pipety s jamkou

2.4 Módy zobrazenia detekcie

Aplikácia dokáže úspešne detekovať pozíciu pipety v okolí jamky v mikrodoštičke. Výsledné dáta je potrebné zobraziť užívateľovi v definovanej forme. Na to v aplikácii slúžia dva zobrazovacie módy pri detekcii. Tieto módy využívajú spomínaný algoritmus, viď 2.3.4 na vyhodnotenie pozície pipety a jamky. Jeho implementácia je ale v každom móde rozličná podľa potreby a následného zobrazenia dát.

2.4.1 Priamy mód zobrazenia detekcie

Tento mód ponúka jednoduchú deteckiu pipety v jamke mikrodoštičky, na ktorú využíva spomínaný algoritmus, viď 2.3.4. V aplikácii sa tento mód nazýva `Camera view detect mode`. Jadro tohto módu je tvorené dvoma polami *emptyWellsInCameraDetectView* a *fullWellsInCameraDetectView*. Pri aktivácii tohto módu v užívateľskom paneli aplikácia automaticky prechádza do módu detekovania jamiek v obraze, viď 2.3.2. Detekcia jamiek prebieha na 5 snímkach za sebou. Po detekcii sú jamky optimalizované a následne pridané do lineárneho poľa *emptyWellsInCameraDetectView*. Toto pole reprezentuje prázdne, nenaplnené jamky v mikrodoštičke. Jamky sú v poli uložené ako obrazové súradnice x a y dátového typu Point. Po detekcii jamiek aplikácia následne automaticky prechádza do módu detekcie pozície pipety a porovnávania jej zhody s jamkou. Algoritmus permanentne kontroluje pozíciu pipety a porovnáva ju s hodnotami pozície jamiek v poli *emptyWellsInCameraDetectView*. Pri vyhodnotení zhody sa súradnice danej jamky uložia do pola *fullWellsInCameraDetectView* a z poľa *emptyWellsInCameraDetectView* bude jamka vymazaná. Detekcia je aktívna a neustále kontroluje každý jeden obrázok, ktorý je snímaný kamerou. Ďalšou dôležitou časťou tohto módu je zobrazenie jamiek. Užívateľ aplikácie musí mať prehľad aký je momentálny stav jamiek, teda ktoré sú plné a ktoré prázdne. Toto zabezpečujú funkcie, ktoré v náhľade kamery vykresľujú príslušné jamky. Užívateľ má v aplikácii možnosť počas detekcie nastavovať, aké jamky budú zobrazené. Je možné zobraziť prázdne a plné osobitne, alebo je možné zobraziť prázdne aj plné zároveň. Pre lepšiu orientáciu je možné nastaviť aj zobrazovanie špičky pipety, viď obrázok 2.13.

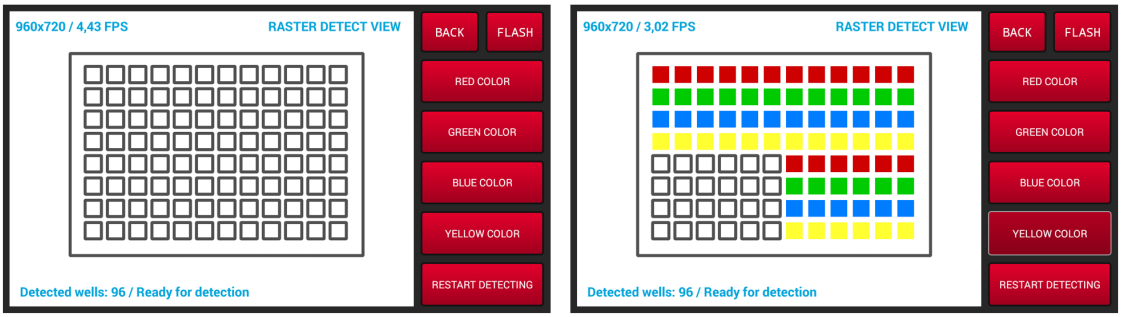


Obr. 2.13: Ukážka aplikácie v móde Camera view detect mode

2.4.2 Rasterový mód zobrazenia detekcie

Ďalší mód zobrazenia, ktorý aplikácia ponúka, je rasterový mód zobrazenia. Tento mód sa líši od priameho zobrazenia tým, že všetky detekované jamky sú zobrazo-

vané v tzv. rastri. To znamená, že každá jedna bunka v rastri reprezentuje príslušnú jamku. Detekcia sa vykonáva v pozadí aplikácie a detekované jamky v rastri sa zobrazujú nezávisle na ich reálnej pozícii v obraze. Rasterový mód zobrazenia sa aktivuje kliknutím na tlačítko `Raster view detect mode` v paneli aplikácie. Následne aplikácia detekuje všetky jamky v mikrodoštičke. Duplicitné jamky sú optimalizované. Pri tomto type zobrazenia je potrebné všetky jamky zoradiť do riadkov a stĺpcov, aby sa dali ľahko a presne identifikovať. Je to dôležité z dôvodu nezávislého zobrazenia jamiek v rastri od reálnej pozície v obraze. Zoradenie a následné naplnenie do pola obstaráva metóda `rasterizationOfCircles()`. Táto metóda hľadá postupne všetky detekované jamky a podľa ich pozície ich zoraďuje do stĺpcov, a následne do riadkov. Všetky tieto zoradené jamky sú uložené v dvojrozmernom poli *centersOfCirclesSorted*. Po aplikovaní tejto metódy je možné zistiť pre každú jamku jej pozíciu v rastri, ale aj jej reálnu pozíciu x a y v obraze. Podľa počtu riadkov a stĺpcov v poli *centersOfCirclesSorted* sa vytvorí ďalšie dvojrozmerné pole *allDetectedWellsInRaster*, ktoré slúži iba na uchovávanie hodnoty a následné vykreslenie jamiek v rastri. Toto pole už neobsahuje jednotlivé pozície jamiek, ale farbu vykreslenia jamky v ras-



Obr. 2.14: Ukážka aplikácie v móde Raster view detect móde

tri. Každá bunka v tomto poli nadobúda číselnú hodnotu od 0 do 5. Hodnotou 0 sú reprezentované všetky prázdne jamky a hodnoty od 1 do 5 sú naplnené jamky v mikrodoštičke, ktoré sa líšia farbou podľa príslušného čísla. Všetky jamky sú vykresľované z pola *allDetectedWellsInRaster* a na základe číselnej hodnoty sa určí farba každej bunky v rastri. Druhé pole *centersOfCirclesSorted* slúži ako referenčné pri detekcii, kedy sa pozícia pipety porovnáva s hodnotami v tomto zoradenom poli. Ak pri detekcii nastane zhoda pozície pipety s jamkou v mikrodoštičke, toto zoradené pole na základe reálnej pozície jamky vráti jej polohu v rastri. Následne je možné podľa získaných hodnôt vyhľadať jamku v poli *allDetectedWellsInRaster* a označiť ju príslušnou farbou ako naplnenú. Na vykreslenie jamiek v rastri slúži metóda `showAllWellsInRasterView()`, ktorá vykresľuje raster s určitými rozmermi podľa počtu detekovaných jamiek v mikrodoštičke, viď obrázok 2.14. V náhľade aplikácie

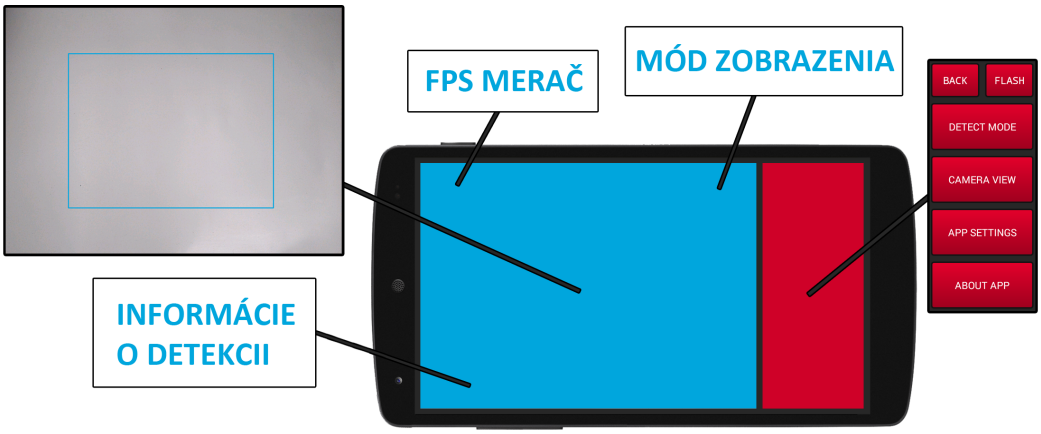
je možné vidieť, že užívateľ má možnosť pri detekcii zmeniť si farbu detekovanej bunky v rastri.

2.5 Grafické užívateľské rozhranie aplikácie

Aby mohol užívateľ jednoducho komunikovať s aplikáciou, je potrebné, aby mala aplikácia implementované základné užívateľské rozhranie. Toto rozhranie spája užívateľa s aplikáciou a je tak možné aplikáciu ovládať. Pre užívateľa musí byť rozhranie prehľadné a intuitívne, aby bolo možné s aplikáciou jednoducho zaobchádzať. Užívateľské rozhranie plní nielen funkciu vstupného terminálu. Jeho ďalšou úlohou je zobrazovať dôležité informácie pre užívateľa tak, aby boli prehľadné a ľahko zrozumiteľné.

2.5.1 Realizácia užívateľského rozhrania

Pri návrhu užívateľského rozhrania bolo dôležité držať sa skutočnosti, že je potrebné zobrazovať snímaný obraz z kamery zariadenia a zároveň musí aplikácia obsahovať základné ovládacie prvky pre komunikáciu užívateľa s aplikáciou. Jednotlivé prvky



Obr. 2.15: Rozloženie jednotlivých prvkov grafického rozhrania aplikácie

musia byť preto vhodne umiestnené. Časť obrazovky aplikácie, ktorá zobrazuje užívateľovi snímaný obraz, musí byť dostatočne veľká, aby bolo možné prehľadne pozorovať snímanú scénu kamerou daného zariadenia. Taktiež samotné umiestnenie zobrazovacích prvkov pri detekcii a informácie o stave aplikácie je dôležité umiestniť tak, aby boli ľahko čitateľné a neprekážali užívateľovi pri pozorovaní scény obrazu. Ovládacie prvky aplikácie musia byť dostatočné veľké pre užívateľa, ľahko zrozumiteľné a intuitívne. V aplikácii sa ovládacie prvky vyskytujú vo forme tlačidiel a horizontálnych posuvníkov. Tlačítka sa nachádzajú v ovládacom paneli, ktorý je

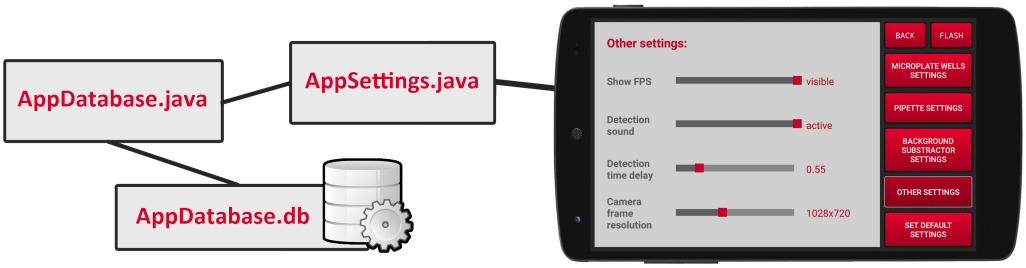
vhodne umiestnený po pravej strane aplikácie. Pomocou tlačidiel je možné obsluhovať celú aplikáciu. Posuvníky sa nachádzajú v nastaveniach aplikácie a slúžia na úpravu užívateľských nastavení. Rozhranie je realizované v značkovacom jazyku *XML*, ktoré vývojové prostredie Android Studio ponúka na tvorbu grafického rozhrania aplikácie.

2.6 Databáza aplikácie

Aplikácia pracuje s väčším počtom dát, ktoré je potrebné organizovať. Všetky dáta je potrebné uchovávať tak, aby boli kedykoľvek k dispozícii na čítanie alebo zápis. Preto aplikácia komunikuje s jednoduchou databázou, ktorá slúži pre uchovávanie týchto dát.

2.6.1 Databáza užívateľského nastavenia aplikácie

Hlavným dôvodom realizácie databázy je úschova užívateľských nastavení, nezávisle od aplikácie. To prináša podstatnú výhodu v tom, že dáta sú dostupné vždy aj po reštartovaní aplikácie, pretože všetky užívateľské nastavenia nie sú uložené fyzicky v pamäti aplikácie, ale v externej databáze, ktorá s aplikáciou komunikuje. Táto



Obr. 2.16: Princíp priebehu komunikácie medzi aplikáciou a databázou

komunikácia prebieha medzi nasledovnými uzlami, viď obrázok 2.16. Prvý hlavný uzol tvorí samotná databáza `AppDatabase.db`, kde sú uložené všetky dáta v jednej organizovanej štruktúre. Táto štruktúra sa volá tabuľka `settings`. Je zložená so stĺpcov a riadkov. Priamu komunikáciu s touto databázou zabezpečuje trieda `AppDatabase.java`. V tejto triede sa generujú všetky požiadavky, ktoré sa posielajú do databázy. Takáto požiadavka vzniká v prípade výberu alebo uloženia dát do databázy. Ďalší uzol komunikácie tvorí trieda `AppSettings.java`. Je to akýsi medzistupeň medzi databázou a samotnou aplikáciou. Vlastnosti tejto triedy slúžia pre dočasné uchovávanie užívateľský nastavení aplikácie a metódy triedy slúžia na manipuláciu s jednotlivými dátami. Posledný uzol komunikácie tvorí samotná aplikácia, ktorá so získanými dátami pracuje.

3 ZÁVER

Úlohou tejto práce bolo navrhnúť a realizovať aplikáciu pre operačný systém Android, ktorá využíva knižnicu OpenCV na spracovanie obrazu. Bolo požadované, aby aplikácia dokázala sledovať a detekovať vopred definované objekty z videosignálov kamery daného zariadenia. Aplikácia by mala slúžiť na detekovanie mikrodostičky, pipety a následnú analýzu obrazu pri pipetovaní.

Pred riešením práce bolo potrebné zoznámiť sa s problematikou, potrebnými technológiami a nástrojmi, ktoré bolo možné pri riešení práce použiť a v neposlednej rade určiť postup, ako pri riešení danej problematiky postupovať.

V úvode riešenia práce bolo potrebné zoznámiť sa s vývojovým prostredím Android Studio, následne bola vytvorená jednoduchá aplikácia, ktorá bola v začiatkoch testovaná vo virtuálnom zariadení, ktoré vývojové prostredie ponúka. Následne bolo nutné zoznámiť sa s knižnicou OpenCV, ktorú bolo potrebné do vývojového prostredia korektne implementovať. Aplikácia bola rozšírená o nové metódy, ktoré knižnica OpenCV na spracovanie obrazu ponúka. Následne bolo možné pomocou aplikácie snímaný obraz z kamery spracovávať a vykonávať základné úpravy obrazu pomocou grafických filtrov.

Ďalší vývoj vyžadoval odbornejší prístup, preto bolo nevyhnutné naštudovať potrebné materiály a premyslieť si ďalší postup. Knižnica OpenCV obsahuje veľké množstvo algoritmov pre rôzne praktické použitie, preto bolo potrebné knižnicu dôsledne preštudovať a stanoviť presné ciele a postup, ako pri riešení zadania práce ďalej postupovať. Na začiatok boli implementované jednoduchšie algoritmy, na úpravu obrazu a prácu s kontúrami pomocou ktorých aplikácia dokázala detekovať 2D a jednoduché 3D objekty. Postupne boli jednotlivé algoritmy rozširované a tým bola detekcia presnejšia. Následne bolo možné začať implementovať algoritmus pre detekciu samotnej mikrodoštičky a pomocou Houghovej transformácie boli v doštičke detekované jednotlivé jamky. Ďalej bolo potrebné naučiť sa pracovať s algoritmami pre segmentáciu obrazu. V tejto časti bolo nutné implementovať algoritmus na odčítavanie pozadia. Po zoznámení sa s problematikou bol v práci na segmentáciu obrazu implementovaný algoritmus *MOG2*, ktorý taktiež dokáže potlačiť tiene, ktoré pohybujúci sa objekt – pipeta môže generovať. Segmentáciou obrazu bolo možné rozlíšiť pohybujúcu sa pipetu od pozadia scény obrazu. Bolo tak možné presne detekovať pohyb a špičku pipety.

V práci boli jednotlivé funkcionality postupne rozširované, a tým prinášala aplikácia širšie možnosti pre budúceho užívateľa. Aplikácia dokáže v obraze vyhľadať mikrodoštičku, v tejto doštičke detekovať jamky a následne sledovať pohyb pipety v obraze. Ponúka taktiež viacero inteligentných módov detekcie, ako je priamy zobrazovací mód a rasterový zobrazovací mód. Obidva tieto módy ponúkajú možnosť

detekcie pipety v jamke mikrodoštičky, respektíve naplnenie danej jamky. Líšia sa manipuláciou s dátami a následným zobrazením týchto dát užívateľovi.

Po zvládnutí tejto problematiky bolo potrebné, aby mohol užívateľ a aplikácia medzi sebou komunikovať. Preto bolo navrhnuté následne realizované grafické užívateľské rozhranie tak, aby bolo pre užívateľa prehľadné a intuitívne. Ďalším problémom bolo, že aplikácia obsahuje dáta, ktoré je potrebné uchovávať aj po jej vypnutí. Preto bola realizovaná databáza `AppDatabase.db`, kde sú tieto dáta uložené a kedykoľvek k dispozícií.

Zadané ciele tejto práce boli úspešné splnené, bola vytvorená aplikácia pre operačný systém Android, ktorá umožňuje detekciu predom definovaných predmetov v obraze z kamery daného Android zariadenia. Užívateľ má možnosť vybrať si z niekoľkých detekčných režimov, ako bude detekcia prebiehať a následne ako budú spracované dáta užívateľovi zobrazované. Aplikácia obsahuje prehľadné grafické užívateľské rozhranie pre jednoduchú obsluhu aplikácie, taktiež umožňuje všetky užívateľské nastavenia uchovávať v databáze.

LITERATÚRA

[1] About *OpenCV library* [online]. [cit. 27. 11. 2016]. Dostupné z: <http://opencv.org/about.html>.

[2] ALLEN, G. *Android 4: průvodce programováním mobilních aplikací*. Brno: Computer Press, 2013. 656 s. ISBN 978-80-251-3782-6.

[3] BRADSKI, G., KAEHLER, A. *Learning OpenCV*. O'Reilly & Associates, Inc., 2008. 571 s. ISBN 978-0-596-51613-0.

[4] Dashboards *Android Developers* [online]. [cit. 24. 11. 2016]. Dostupné z: <https://developer.android.com/about/dashboards/index.html>.

[5] HOWSE, J., PUTTEMANS, S. *OpenCV 3 Blueprints*. Birmingham: Packt Publishing Ltd, 2015. 382 s. ISBN 978-1-78439-975-7.

[6] LACKO, L. *Vývoj aplikací pro Android*. Brno: Computer Press, 2015. 472 s. ISBN 978-80-251-4347-6.

[7] OpenCV4Android SDK *OpenCV documentation* [online]. [cit. 14. 4. 2017]. Dostupné z: <http://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/O4A_SDK.html>.

[8] ŘÍHA, K. *Pokročilé techniky zpracování obrazu*. 1. vydání. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav komunikací, 2012. 143 s. ISBN 978-80-214-4894-0.

[9] *The OpenCV Tutorials* [online]. [cit. 14. 4. 2017]. Dostupné z: <http://docs.opencv.org/2.4/opencv_tutorials.pdf>.

[10] Uses Sdk Element *Android Developers* [online]. [cit. 19. 4. 2017]. Dostupné z: <https://developer.android.com/guide/topics/manifest/uses-sdk-element.html>.

[11] Zdrojak *Vyvíjíme pro Android: Dialogy a activity* [online]. [cit. 27. 11. 2016]. Dostupné z: <https://www.zdrojak.cz/clanky/vyvijime-pro-android-dialogy-a-activity>.

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

OpenCV	Open Source Computer Vision
OHA	Open Handset Alliance
IDE	Integrované vývojové rozhranie – Integrated Development Environment
API	Aplikačné programovacie rozhranie – Application Programming Interface
ADT	Android vývojové nástroje – Android Development Tools
SDK	Softvérový vývojový kit – Software Development Kit
NDK	Pôvodný vývojový kit – Native Development Kit
AVD	Android virtuálne zariadenie – Android Virtual Device
GPU	Grafická výpočtová jednotka – Graphics processing unit
MOG	Mixture of Gaussians
GMG	Gaussian Mixture Model
KNN	K-Nearest Neighbors
BSD	Berkeley Software Distribution
ROI	Region of interest – Oblasť záujmu

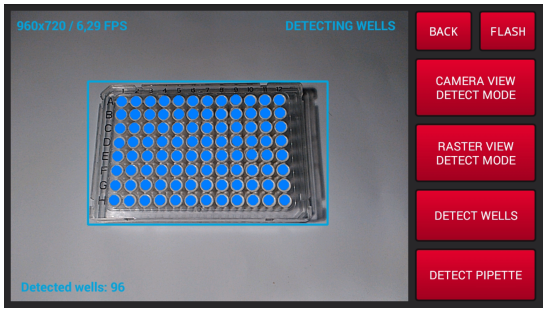
ZOZNAM PRÍLOH

A OBSAH PRILOŽENÉHO DVD

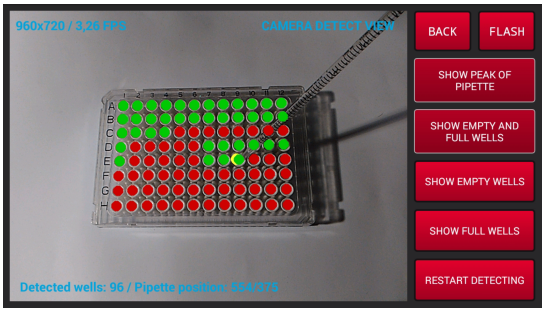
Priložené DVD obsahuje elektronickú verziu bakalárskej práce. Hlavný dokument „Bakalárska práca - Martin Kiac“ sa nachádza v zložke „Dokumentácia“. Inštalačný súbor „app-release“ pre operačný systém Android sa nachádza v zložke „Aplikácia“. V ďalšej zložke „Zdrojové súbory“ sa nachádzajú zdrojové súbory praktickej časti bakalárskej práce. Tieto zdrojové súbory je možné otvoriť ako pracovný projekt vo vývojovom prostredí Android Studio. V poslednej zložke „Video ukážky“ sa nachádzajú video ukážky z aplikácie.

```
/
├── Dokumentácia
│   └── Bakalárska práca - Martin Kiac.pdf
├── Aplikácia
│   └── app-release.apk
├── Zdrojové súbory
│   └── Pipetting Assistant
└── Video ukážky
    ├── detectWells01.mp4
    ├── cameraDetectView01.mp4
    ├── cameraDetectView02.mp4
    ├── cameraDetectView03.mp4
    ├── rasterDetectView01.mp4
    ├── rasterDetectView02.mp4
    └── appSettings.mp4
```

B UKÁŽKY APLIKÁCIE

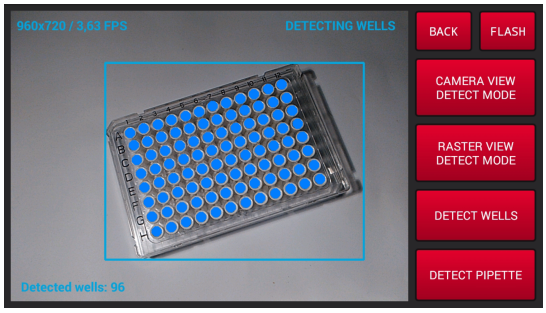


(a) Detekcia jamiek

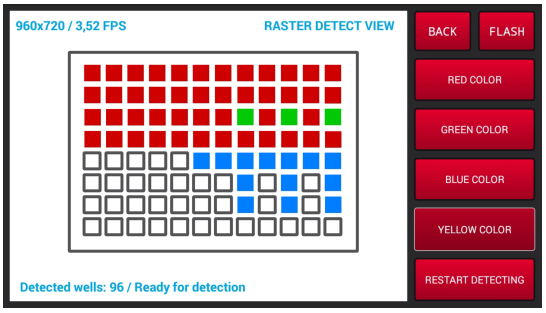


(b) Algoritmus pre vyhodnotenie zhody

Obr. B.1: Priamy mód zobrazenia detekcie

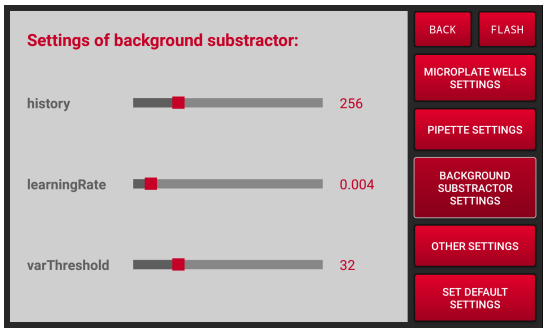


(a) Detekcia jamiek

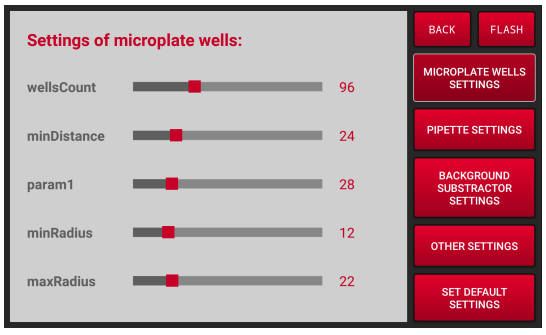


(b) Detekcia v rasterovom móde zobrazenia

Obr. B.2: Rasterový mód zobrazenia detekcie



(a) Nastavenie odčítania pozadia



(b) Nastavenie detekcie jamiek v mikrodoštičke

Obr. B.3: Užívateľské nastavenia aplikácie