VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ZPRACOVÁNÍ OBRAZU V ZAŘÍZENÍ ANDROID - DETEKCE A ROZPOZNÁNÍ VIZITKY

IMAGE PROCESSING USING ANDROID DEVICE - AUTOMATIC DETECTION AND RECOGNITION OF BUSINESS CARDS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Martin Krčmář**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Peter Honec, Ph.D.**
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Martin Krčmář ***ID:*** 138559
Ročník: 2 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Zpracování obrazu v zařízení Android - detekce a rozpoznání vizitky

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce bude navrhnout a implementovat aplikaci pro automatickou detekci a rozpoznání obchodní vizitky v obraze s využitím HW a OS Andriod.

1. Vytipujte vhodný HW.
2. Vytvořte GUI umožňující spolupráci s integrovaným fotoaparátem.
3. Navrhněte algoritmy pro rozpoznání údajů na vizitce a jejich další zpracování.
4. Ověřte a zhodnoťte.

DOPORUČENÁ LITERATURA:

Hlaváč, Šonka, Počítačové vidění.

Šonka, Hlávač, Boyle - IMAGE PROCESSING, ANALYSIS, AND MACHINE VISION,

Termín zadání: 8.2.2016 ***Termín odevzdání:*** 16.5.2016

Vedoucí práce: Ing. Peter Honec, Ph.D.
Konzultant diplomové práce:

doc. Ing. Václav Jirsík, CSc., *předseda oborové rady*

Abstrakt

Cílem této diplomové práce je vytvoření mobilní aplikace pro systém Android, která bude sloužit k automatickému rozpoznávání vizitek a importování kontaktních údajů. V první části je popsána historie, architektura a vývojové nástroje operačního systému Android. Ve druhé části je proveden rozbor vybraných metod počítačového vidění, které byly použity při vytváření aplikace. Jsou zde popsány knihovny OpenCV a Tessaract OCR pro zpracování obrazu. V hlavní části je popsán vývoj aplikace spolu s podmínkami a omezeními pro správnou funkci aplikace. V závěrečné části je provedeno vyhodnocení úspěšnosti rozpoznávání a importování kontaktních údajů z vizitek.

Klíčová slova

Android, Java, zpracování obrazu, počítačové vidění, OpenCV, OCR

Abstract

The aim of this Master´s thesis is designing and developing Android application, which will be used for automatic recognition of business cards and import the contact information. The first part describes the history, architecture and development tools of operating system Android. The second part is an analysis of selected computer vision methods that were used during developing application. Libraries OpenCV and Tessaract OCR are described in this part. The main part describes the development of the application with conditions and limitations for the proper function of the application. The final part is an evaluation of the success and recognition of importing contact information from business cards.

Key words

Android, Java, image processing, computer vision, OpenCV, OCR

Bibliografická citace:

KRČMÁŘ, M. *Zpracování obrazu v zařízení Android - detekce a rozpoznání vizitky*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 64s. Vedoucí diplomové práce Ing. Peter Honec, Ph.D.

Prohlášení

„Prohlašuji, že svou diplomovou práci na téma *Zpracování obrazu v zařízení Android - detekce a rozpoznání vizitky* jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **16. května 2016** ……

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Petrovi Honcovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **16. května 2016** ……

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

1 ÚVOD

Systém Android je v dnešní době nejrozšířenějším operačním systémem pro mobilní zařízení. Tyto zařízení již disponují slušným výkonem a praktickými periferiemi, což jim umožňuje provádět výpočetně náročnější úlohy, jakou je například zpracování obrazu. Zařízení s operačním systémem Android se díky své otevřenosti, dostupnosti a komplexním periferiím výborně hodí pro tvorbu různých aplikací počítačového vidění na mobilní platformě.

Cílem diplomové práce je vytvořit mobilní aplikaci pro systém Android, která bude využívat integrovaný fotoaparát pro vyhledávání vizitek v reálném čase a umožňovat automatické importování kontaktních údajů z vizitky do zařízení. Velká většina dnešních chytrých telefonů má integrovaný fotoaparát, proto se pro tuto aplikaci výborně hodí. Práce je rozdělena do několika tematických celků.

V první části je popsán systém Android, jeho historie, architektura a vývojová prostředí.

Druhá část je věnována teoretickému rozboru vybraných metod počítačového vidění, které budou použity v diplomové práci.

Třetí kapitola se zabývá popisem knihoven počítačového vidění, pomocí kterých je aplikace realizována. Je zde popsána knihovna OpenCV pro zpracování obrazu a knihovna Tesseract pro rozpoznávání znaků.

Čtvrtá kapitola je věnována popisu samotná aplikace z uživatelského hlediska. Jsou zde popsány omezení a doporučení pro správnou funkci aplikace.

V páté kapitole jsou popsány jednotlivé kroky vývoje aplikace od získání snímku z kamery, přes vyhledání vizitky, rozpoznání textu až po samotné importování kontaktů do telefonu.

V závěrečné části je provedeno vyhodnocení úspěšnosti rozpoznávání vizitky a importování kontaktních údajů. V příloze se nachází galerie vizitek s ukázkou nalezeného textu a kontaktních údajů.

2 OPERAČNÍ SYSTÉM ANDROID

Android je open source operační systém primárně vyvíjený pro mobilní zařízení firmou Google. Je navržen především pro dotykové mobilní zařízení, ale v současnosti se objevuje i v chytrých hodinkách nebo kapesních minipočítačích. V současnosti je Android nejrozšířenějším a nejrychleji rostoucím OS na světě. [1]

Obrázek 1: Ikona platformy Android
(http://www.dailymobile.net/wpcontent/uploads/2014/12/androidlogo1.jpg)

2.1 Historie

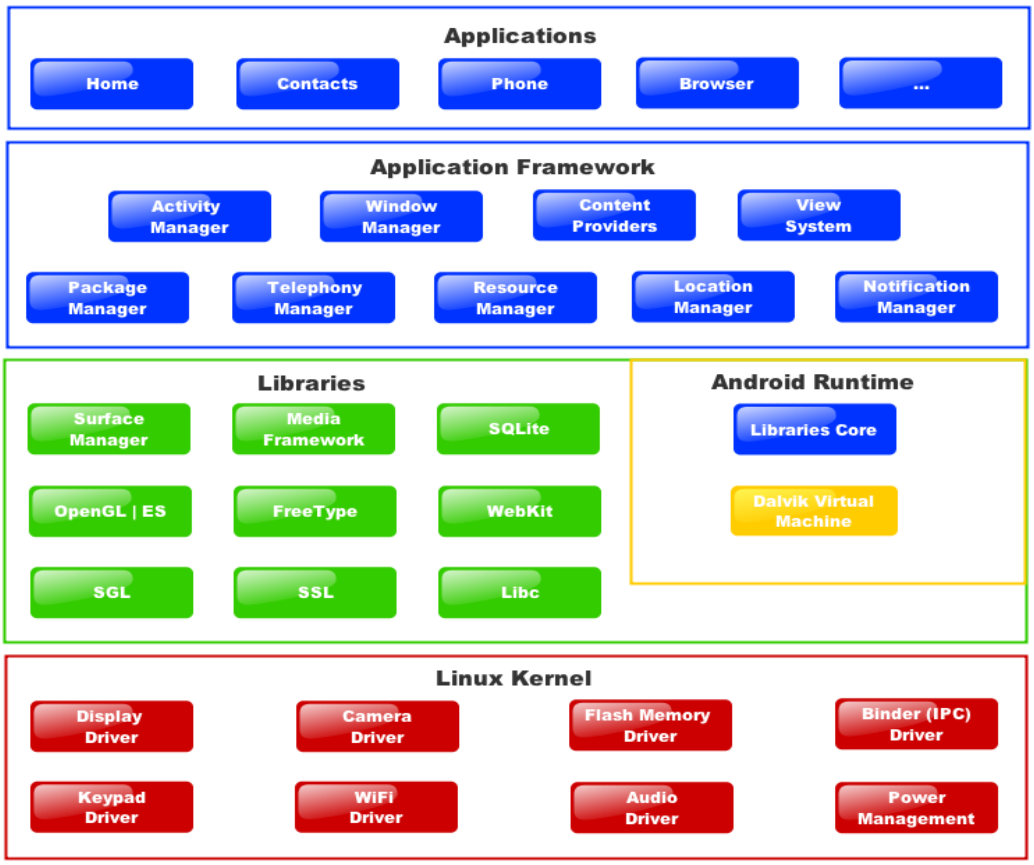
V roce 2003 založili pánové Andy Rubin, Rich Miner, Nick Sears a Chris White společnost Android Inc. Původní myšlenkou bylo vytvořit OS pro digitální kamery, ale tento záměr se brzy změnil na vytvoření OS, který by konkuroval soudobým systémům mobilních zařízení jako Symbian nebo Windows mobile. V srpnu 2005 Android Inc. odkoupila firma Google a Android se stal oficiálně open source. První verze byla zveřejněna v listopadu 2007, ale trvalo téměř celý rok, než byl vydán první telefon s operačním systémem Android. Tím byl HTC Dream s verzí Android 1.6. Od roku 2008 prošel Android řadou vylepšení až po nynější verzi 6.0 Marshmallow. [2][3]

Android 1.0 (API 1)	Android 2.3 – 2.3.7Gingerbread (API 9-10)
Android 1.1 (API 2)	Android 3.0 – 3.2 Honeycomb (API 11-13)
Android 1.5 Cupcake (API 3)	Android 4.0 – 4.0.4 Ice Cream Sandwich (API 14-15)
Android 1.6 Donut (API 4)	Android 4.1 – 4.3 Jelly Bean (API 16-18)
Android 2.0 - Eclair (API 5-6)	Android 4.4 – 4.5 KitKat (API 19-20)
Android 2.1 Eclair (API 7)	Android 5.0 – 5.1 Lollipop (API 21-22)
Android 2.2 – 2.2.3 Froyo (API 8)	Android 6.0 Marshmallow (API 23)

Tabulka 1: *Historický vývoj verzí OS Android*

2.2 Architektura

Architektura operačního systému Android je rozdělena do 5-ti vrstev. Každá vrstva má svůj účel a nemusí být přímo oddělena od ostatních vrstev.



Obrázek 2: Architektura operačního systému Android

[http://www.eazytutz.com/wp-content/uploads/2015/02/Android-Architecure.png]

2.2.1 Linux kernel

Nejnižší vrstvou je jádro operačního systému - Linux Kernel, aktuálně ve verzi 3.6. Představuje úroveň abstrakce mezi hardwarem a vyššími vrstvami Androidu. Důvodem použití jádra Linux je vlastnost poměrně snadného sestavení na různých zařízeních a tím zaručená přenositelnost. [4]

2.2.2 Knihovny

Nad vrstvou linuxového jádra je sada knihoven, které jsou napsány v jazyce C nebo C++. Jsou používány jak systémem, tak aplikacemi. Přístup k nim zajišťuje vrstva Application Framework. Mezi tyto knihovny patří: [4][5]

- OpenGL - pro rendrování 2D a 3D grafiky
- WebKit - pro zobrazování HTML obsahu
- SQLite - pro používání databází
- Surface manager – vytváří uživatelské rozhraní systému
- Media Framework - knihovny k přehrávání a nahrávání medií.

2.2.3 Android RunTime

Tato část poskytuje klíčovou součást nazvanou Dalvik Virtual Machine (DVM). Jedná se o typ Java Virtual Machine (JVM), která slouží ke spouštění aplikací v Androidu. DVM umožňuje každé aplikaci spuštění vlastního procesu. Je optimalizován pro co nejefektivnější využití výpočetního výkonu. Na rozdíl od JVM Dalvik Virtual Machine nespouští *(.class)* soubory, ale *(.dex)* soubory, které mají o 50% nižší nároky na paměť. Soubory třídy *standard Java* lze obvykle převést do formátu *(.dex)* pomocí DX nástroje, který je součástí Android SDK. Výsledný *Dalvik byte kód* je spuštěn na DVM. Každá spuštěná Android aplikace běží ve svém vlastním procesu, s vlastní instancí DVM. [5][6]

2.2.4 Application Framework

Application Framework je pro vývojáře nejdůležitější. Poskytuje sadu služeb, které společně tvoří prostředí, ve kterém Android běží a je spravován. Tyto služby mohou zpřístupňovat data v jiných aplikacích, hardware používaného zařízení, spravovat aplikace v pozadí a mnoho dalšího. [5][6]
Hlavními částmi jsou:
Activity Manager – řídí životní cyklus aplikace

Content providers – umožňuje aplikacím sdílet data s jinými aplikacemi

Resource Manager – umožňuje přístup k různým typům zdrojů, jako jsou řetězce, grafika, atd.

Notifications Manager – umožní aplikacím zobrazovat výstrahy a upozornění

View System – soubor nástrojů sloužící k vytvoření uživatelských rozhraní aplikací

Telephony Manager – poskytuje informace o telefonních službách

Location Manager – umožňuje přístup k lokalizačním službám pomocí GPS.

2.2.5 Aplikace

Nejvyšší vrstvu Androidu tvoří aplikace, které slouží pro běžného uživatele. Umožňuje vykonávat základní funkce, jako je např. telefonování, posílání textových zpráv nebo prohlížení webu. Patří sem aplikace, které jsou součástí předinstalovaného androidu nebo instalované uživatelem. [4]

2.3 Základní části aplikace

Existuje pět základních složek, které se používají k sestavení aplikace.

2.3.1 Activity (aktivita)

Aktivita je grafické uživatelské rozhraní umožňující komfortní ovládání a zobrazování aplikací. Jednotlivé elementy aktivity se nazývají *views (zobrazení)* nebo *widgets*. Widgety mohou být vytvářeny buď čistě pomocí *Java* kódu, nebo pomocí XML kódu definujícím UI. Aplikace může mít více než jednu aktivitu, které buď fungují nezávisle, nebo mohou být propojené. Každá aktivita musí být uvedena v Android manifestu. [8]

2.3.2 Services (služby)

Služba je proces, který běží na pozadí a nemá žádné vizuální uživatelské rozhraní. Mohou provádět stejné úlohy jako aplikace, ale bez interakce s uživatelem. Jsou používány ke zpracování určitých částí aplikace na pozadí. Jsou užitečné v případech, kdy je potřeba zpracovávat něco dlouhého. Zatímco uživatel pracuje na popředí s nějakou aplikací, na pozadí může běžet něco jiného (např. přehrávač hudby – hudba hraje na pozadí nebo stahování dat z internetu atd.). U služeb je mnohem méně pravděpodobné, že budou systémem ukončeny pro uvolnění výpočetního výkonu. [4][5]

2.3.3 Content Providers (poskytovatelé obsahu)

Poskytovatelé obsahu řídí přístup k datům mezi aplikacemi. Prostřednictvím poskytovatele obsahu mohou jiné aplikace požadovat nebo měnit data jiných aplikací (pokud to poskytovatel dovolí). Android v sobě zahrnuje základní poskytovatele obsahu, které spravují data jako je audio, video, obrázky nebo kontakty. Pokud vývojář potřebuje sdílet komplexní data s jinými aplikacemi, musí si vhodného poskytovatele obsahu vytvořit sám. [5]

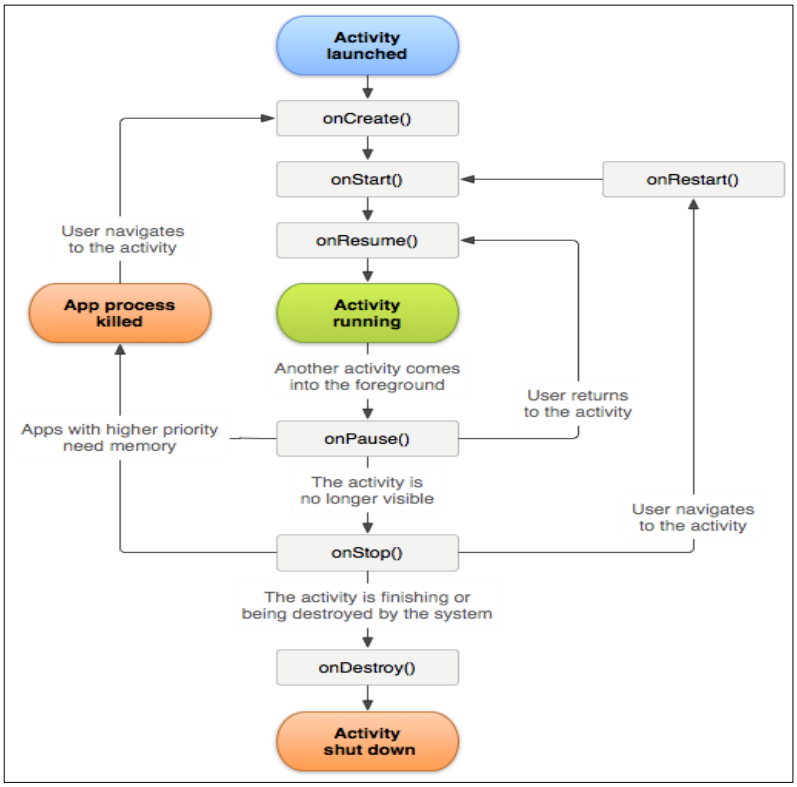
2.3.4 Broadcast Receivers (přijímač vysílání)

Přijímač vysílání je komponenta, která umožňuje reagovat na události (*Intents*) vyvolané systémem nebo aplikací. Události vyvolané systémem mohou být např. vypnutí obrazovky, nízký stav baterie atd. [7]

2.3.5 Intents (záměry)

Intents jsou asynchronní zprávy, které řídí spouštění a ukončování aktivit nebo služeb. Rozlišují se na implicitní a explicitní. Implicitní intent specifikuje pouze typ příjemce, kdežto explicitní ho přesně definuje. Příkladem implicitního intentu je např. žádost o otevření videa. Všechny aplikace, které jsou toho schopny, mohou tuto žádost vyřídit a to tak, že systém nabídne uživateli výběr formou dialogového okna. [4]

2.4 Životní cyklus aplikace



Obrázek 3: Životní cyklus aplikace

http://developer.android.com/reference/android/app/Activity .html#ActivityLifecycle

Oproti desktopovým systémům jako je Windows nebo Mac máme nad životním cyklem aplikace v Androidu mnohem menší kontrolu. Aplikace mají během svého života více stavů než jen vypnuto/zapnuto, jak je vidět na diagramu. O přechody mezi stavy se stará *Activity Manager* (Manažer aktivit). Je možné jednu aplikaci pozastavit a spustit jinou. Když je potřeba se k ní vrátit, jednoduše se obnoví z pozastaveného stavu. Tento proces byl navržen pro zrychlení uživatelského rozhraní. Vývojáři stačí ošetřit několik akcí pomocí připravených metod, aby tuto možnost implementoval. Není třeba psát program

pro jednotlivé stavy aplikace, stačí pouze definovat, co se stane na přechodech mezi nimi. [4][8]

2.4.1 Stavy životního cyklu aplikace

- **Start** (zahájení) **–** během zahájení dojde k inicializaci aplikace, přenesení aktivity do popředí a zpřístupnění aplikace pro uživatele. Spouštění aplikace je, co se týče výpočetní náročnosti, nejnáročnější částí. Proto byla vytvořena možnost pozastavení (*Paused*) aplikace, kdy je možné se k ní znova vrátit, namísto jejího úplného ukončení.
- **Running** (spuštěná aplikace) – je hlavní stav, kdy je aplikace zobrazena na displeji a uživatel ji může využívat. V OS Android platí, že současně nemohou být dvě aplikace ve stavu *running.* Aplikace ve stavu *running* má vyšší prioritu na využívání paměťového prostoru a výpočetního výkonu aby mohla pracovat co nejefektivněji.
- **Paused** (pozastavená) – aplikace je v tomto stavu většinou stále zobrazena, ale není aktivní. K tomuto stavu obvykle dochází např. při vypnutí displeje nebo při aktivaci dialogového okna. Pokud má systém nedostatek paměti, může být aplikace ze stavu *Paused* „natvrdo“ ukončena, což se ve stavu *Running* stát nemůže. Z tohoto důvodu je důležité ukládat aktuální nastavení aplikace vždy při přechodu do tohoto stavu.
- **Stopped** (zastavená) – Tento stav nastane, pokud už aplikace není viditelná, ale stále je uložena v paměti. Z tohoto stavu může být znovu spuštěna rychleji, než kdyby byla zcela vypnuta.
- **Destroyed** (ukončená) – aplikace se už nenachází v paměti, její paměťový prostor je uvolněný.

2.4.2 Metody životního cyklu aplikace

- OnCreate() - tato metoda je zavolána jako první při spouštění aplikace. Dochází zde k vytvoření uživatelského prostředí a incializaci dat.
- OnStart() - metoda je volána před tím, než se stává aplikace viditelná pro uživatele
- onResume() - tato metoda je volána pokaždé, když aplikace přechází do popředí (je zobrazena na displeji). Mohou se zde obnovit změny uložené během metody onPause().
- onPause() - tato metoda se volá, když je do popředí volána jiná aplikace. Používá se pro uložení nedokončených změn v aplikaci (např. rozepsaná sms zpráva, email) nebo pro uvolnění systémových zdrojů jako je GPS, kamera v době, kdy nejsou využívány.

2.5 Struktura projektu v Androidu

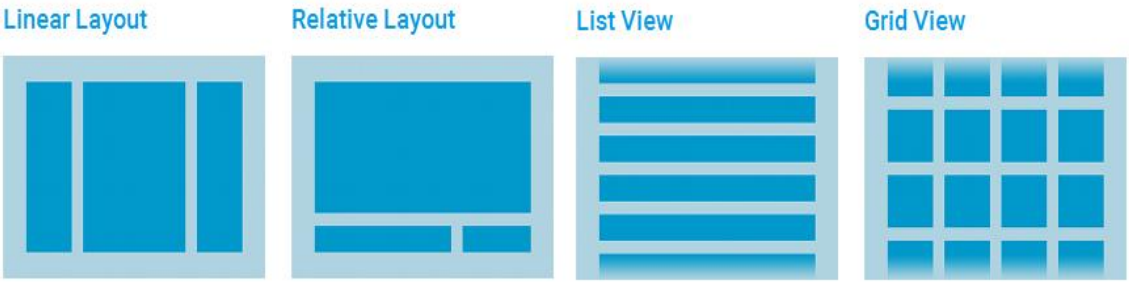
2.5.1 Manifest

Každá aplikace musí mít soubor *AndroidManifest.xml* (právě s tímto názvem) ve svém kořenovém adresáři. V tomto souboru jsou základní informace o aplikaci, jako je:

- název balíku aplikace, který slouží jako jedinečný identifikátor
- komponenty aplikace jako jsou activities, servicies, content providers, broadcast receivers
- práva na přístup k chráněným částem API (např. Kamera)
- požadavky na minimální verzi Androidu

2.5.2 Layouts

Definují rozvržení vizuálního uživatelského rozhraní pro aktivity v jazyce XML. Existují čtyři typy: Linear layout, Relative layout, Grid View a List View.



Obrázek 4: Layouts

[http://developer.android.com/guide/topics/ui/declaring-layout.html]

2.5.3 View (Widgets)

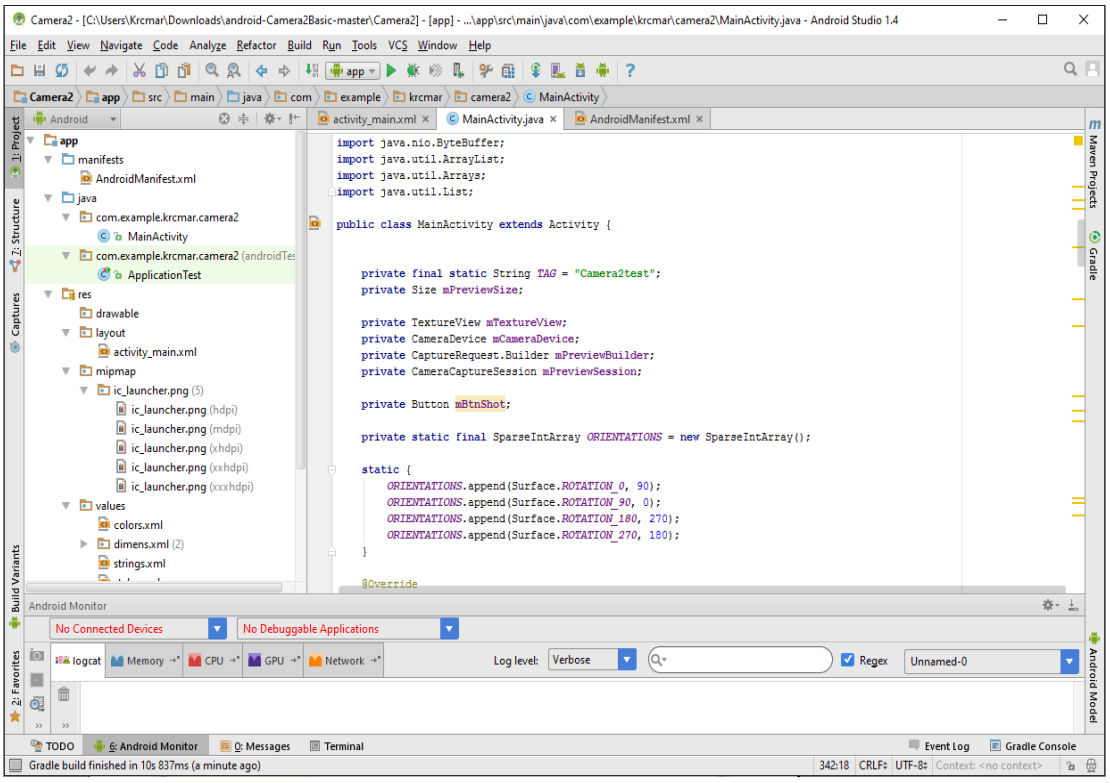
Jsou to objekty, které představují grafické uživatelské rozhraní aplikace. Patří sem např. textová pole *(EditText)*, tlačítka *(Button)* nebo popisky *(labels).*

2.6 Vývojová prostředí

Android je zcela otevřenou platformou, pro kterou se vytvářejí aplikace primárně v jazyce Java. Pokud chceme programovat pro Android, máme k dispozici mnoho vývojových prostředí: Eclipse, NetBeans, Intellij IDEA a Android Studio. Mezi nejrozšířenější vývojové prostředí patří Android Studio a Eclipse.

2.6.1 Android Studio

Android Studio je nový oficiální vývojový nástroj vydaný v prosinci 2014, který je zcela zdarma. Je postaven na prostředí IntelliJ IDEA. Díky tomu získává všechny možnosti práce s kódem, jako je navigace v kódu, našeptávání, refaktoring a analýza kódu. Oproti staršímu vývojovému prostředí Eclipse byl vyvíjen „na míru“ pro Android, tudíž zjednodušuje a zrychluje řadu úkonů během vývoje. [4]



Obrázek 5: Ukázka vývojového prostředí Android Studio

2.6.2 Eclipse

Eclipse je open-source vývojové prostředí, které podporuje vývoj aplikací v různých jazycích, není tedy omezen pouze na Javu. Pomocí plug-inu Android Developer Tools jej lze využít pro vývoj android aplikací. Ještě donedávna byl doporučeným vývojovým prostředím pro Android, ale od vydání Android Studia už jej Google nepodporuje. Donedávna bylo výhodnější pracovat s Eclipse pro lepší podporu NDK, ale v současnosti už ani toto neplatí. [10]

2.7 Knihovny a toolkity

V této části budou popsány knihovny a toolkity, které budou využity v DP.

2.7.1 Android NDK

Native Android Kit je sada nástrojů, které umožňují využití C a C++ kódů v aplikaci pro Android. Používá se pro implementování různých knihoven napsaných C a C++. Také se využívá při tvorbě aplikace, která má být multiplatformní. Pro běžný vývoj se však NDK nepoužívá. [10]

3 POČÍTAČOVÉ VIDĚNÍ

Počítačové vidění je disciplína, která se snaží technickými prostředky napodobit lidské vidění. Jedná se o relativně nový obor, který se mohl začít vyvíjet až s nástupem dostatečně výkonné výpočetní techniky. Počátky počítačového vidění se objevují na začátku 80. let. K řešení reálných aplikací pomocí počítačového vidění je nutná komplexní znalost problému a možností řešení. Zpracování obrazových dat je obvykle založeno na extrakci příznaků, jako je nalezení hran, významných bodů, tvarů z obrazu, které jsou dále zpracovávány. [13][14]

Mezi nejčastější aplikace počítačového vidění patří:

- detekce různých jevů, například sledování kvality výroby
- ovládání a řízení procesů v průmyslu nebo autonomních strojích
- zpracování interakce člověka s počítačem (Kinect)
- řízení provozu
- rekonstrukce obrazu

V následujících kapitolách budou popsány metody počítačového vidění

3.1 Předzpracování obrazu

Tato metoda slouží ke zlepšení kvality obrazu z hlediska dalšího zpracování. Cílem je potlačení šumu vzniklého při digitalizaci, odstranění zkreslení způsobeného typem snímače nebo zlepšení kontrastu. [13]

3.1.1 Jasové korekce

Obraz je vyjádřen jasovými úrovněmi v pravidelné mřížce. Jelikož v praxi snímací zařízení nemívají stejnou citlivost ve všech bodech, vznikají jasové deformace. Ke kompenzaci těchto deformací se používají různé nástroje, například ekvalizace histogramu nebo filtrace šumu. [13]

3.1.2 Ekvalizace histogramu

Histogram je grafické znázornění distribuce dat pomocí sloupcového grafu. Zobrazuje četnosti jednotlivých jasových hodnot obrazu. Integrací (sumací) histogramu vznikne kumulovaný histogram, jehož použitím jako převodní funkce na původní obraz

dosáhneme ekvalizace (vyrovnání) histogramu. Ekvalizace histogramu zajistí rovnoměrné rozložení jednotlivých jasových složek v obrazu, čímž se zvýší kontrast. [13] Ekvalizace histogramu je vhodné provádět na barevných modelech, které mají oddělenou barevnou a jasovou složku jako je YUV nebo HSV/HSL. U těchto barevných modelů stačí provést ekvalizaci jasové složky, aniž by se měnily hodnoty barev. [14]

3.1.3 Filtrace obrazu

Filtrace se používá k potlačení šumu v obrazu. Tento proces může být proveden pomocí konvoluce daného obrazu a konvoluční masky. Nejjednodušší metodou je filtrace průměrováním. Tato filtrace počítá výslednou hodnotu jasu jako průměr okolních bodů. Tato metoda potlačuje všechny vyšší frekvence bez rozdílu, takže kromě šumu potlačuje i hrany a ostatní ostré přechody, čímž dochází k rozmazání obrazu. Aby se tomu jevu zabránilo, používají se metody, které průměrují jen tu část okolí, ke které bod pravděpodobně patří. [13][14]

Jednou z nich je průměrování s prahem. Tato metoda brání rozmazávání hran tím, že povoluje pouze určitou, předem stanovenou míru diference mezi původní hodnotou a výsledkem průměrování.

Další metodou je například filtrace pomocí mediánu, kdy je výsledná hodnota jasu rovna mediánu zvoleného okolí.

Nebo může být použita metoda rotující masky, která se snaží podle homogenity jasu najít k filtrovanému bodu tu část jeho okolí, ke které pravděpodobně patří. Pro výpočet výsledné hodnoty je použita tato homogenní část. [13]

3.2 Detekce hran

Hrana představuje jasovou nespojitost, která je v obraze reprezentována vysokou prostorovou frekvencí. K určení velikosti a směru změny hodnoty jasové funce se používá gradient, který lze matematicky vyjádřit jako derivaci obrazové funkce. V počítačovém vidění se používají konvoluční masky, které aproximují tuto derivaci. Natočením masky se dosáhne citlivosti na hranu v daném směru. Mezi nejznámější konvoluční operátory patří: [14]

Robertsův

$$h=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix},\quad h=\begin{bmatrix}0 & 1\\ -1 & 0\end{bmatrix}$$

Prewitt

$$h=\begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ -1 & -1 & -1\end{bmatrix},\quad h=\begin{bmatrix}0 & 1 & 1\\ -1 & 0 & 1\\ -1 & -1 & 0\end{bmatrix}$$

Sobelův

$$h=\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix},\quad h=\begin{bmatrix}0 & 1 & 2\\ -1 & 0 & 1\\ -2 & -1 & 0\end{bmatrix}$$

Robinsonův

$$h=\begin{bmatrix}1 & 1 & 1\\ 1 & -2 & 1\\ -1 & -1 & -1\end{bmatrix},\quad h=\begin{bmatrix}1 & 1 & 1\\ -1 & -2 & 1\\ -1 & -1 & 1\end{bmatrix}$$

Kirschův

$$h=\begin{bmatrix}3 & 3 & 3\\ 3 & 0 & 3\\ -5 & -5 & -5\end{bmatrix},\quad h=\begin{bmatrix}3 & 3 & 3\\ -5 & 0 & 3\\ -5 & -5 & 3\end{bmatrix}$$

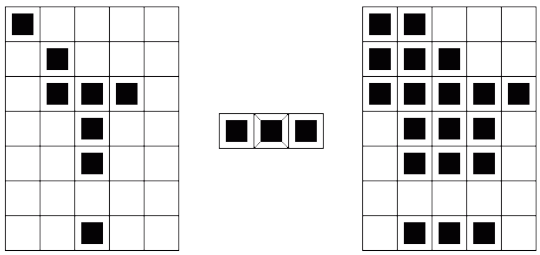
Cannyho detektor – je jedním z nejpoužívanějších algoritmů pro detekci hran. Tento algoritmus se skládá ze čtyř kroků: [13]

1. eliminace šumu Gaussovským filtrem (nejčastěji konvoluční maskou)
2. určení gradientu aplikací Sobbelova operátoru
3. výběr lokálních maxim ze zjištěných gradientů
4. eliminace nevýznamných hran pomocí prahování

3.3 Matematická morfologie

Matematická morfologie je metoda zpracování obrazu založená na teorii množin, integrální algebře a algebře svazků. Původně byla určená pro použití s binárními obrazy, ale později byla zevšeobecněna i na šedo-tónové obrazy.
Hlavní myšlenkou morfologické analýzy je získávání znalostí z relace obrazu a malé sondy (nazývané strukturní element), která má předdefinovaný tvar. V každém pixelu se ověřuje, jak tato sonda odpovídá nebo neodpovídá lokálním tvarům v obraze. [14]
Morfologické operace se používají pro předzpracování (odstranění šumu, zjednodušení tvaru objektů), zdůraznění struktury objektů (kostra, ztenčování, zesilování, konvexní obal, označování objektů) a pro popis objektů číselnými charakteristikami (plocha, obvod, projekce, atd.). [13]
Mezi dvě základní morfologické operace patří dilatace a eroze. Dilatace skládá body dvou množin vektorového součtu. Objekty jsou po aplikaci dilatace zvětšeny o jednu "slupku" na úkor pozadí. Tato operace se používá na zaplnění děr a zálivů v obraze. Morfologická dilatace je definována tímto vztahem:[13]

$$X \oplus B = \{d \in E^2 : d = x + b, x \in X, b \in B\} \qquad X \oplus B = \bigcup_{b \in B} X_b$$



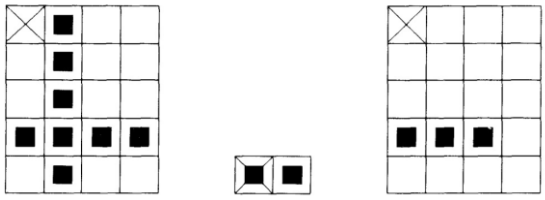
Obrázek 6: Ukázka aplikace dilatace pomocí jádra

[http://midas.uamt.feec.vutbr.cz/ZVS/Exercise10/content_cz.php]

Druhou základní morfologickou operací je eroze. Eroze skládá dvě bodové množiny s využitím vektorového rozdílu. Jedná se o duální operaci k dilataci, ale není inverzní. Používá se pro zjednodušení struktury objektů. Objekty jednotkové tloušťky zmizí a

složité objekty se rozloží na jednodušší. Pomocí erodování obrazu lze nalézt obrysy objektů a může tak sloužit jako hranový detektor. Morfologická eroze je definována tímto vztahem: [13]

$$X \ominus B = \{d \in E^2 : d + b \in X \ pro \ \forall b \in B\} \qquad X \ominus B = \bigcap_{b \in B} X_{-b}$$



Obrázek 7: Ukázka aplikace eroze pomocí jádra

převzato z [13]

Další morfologické operace jsou otevření a uzavření, které vznikly vzájemnou kombinací operací dilatace a eroze. Výsledkem obou je zjednodušený obraz, který obsahuje méně detailů (odstraní detaily menší, než strukturní element, celkový tvar objektu se ale neporuší). Eroze následovaná dilatací se nazývá morfologické otevření. Oddělí objekty spojené úzkou šíjí a tak zjednoduší strukturu objektů. Dilatace následovaná erozí se naopak nazývá morfologické uzavření. Spojí objekty, které jsou blízko u sebe, zaplní díry a vyhladí obrys. [14]

3.4 Geometrické transformace

Geometrické transformace umožňují kompenzaci geometrického zkreslení obrazu. Tyto funkce mapují vstupní pixel z pozice (x, y) do nové pozice (x‘, y‘). Mezi základní geometrické transformace patří posunutí, změna měřítka, otočení okolo počátku a zkosení. Složitější operace jsou realizovány skládáním základních operací. Existuje dvojí vyjádření geometrické transformace: [16]

- **Dopředná transformace**

$$(x', y') = \mathbf{T}(x, y)$$

  U dopředné aproximace se mapují body vstupního obrazu do výstupního podle transformační matice. Nevýhodou této transformace je, že souřadnice ze vstupního obrazu mohou být namapovány mimo rast některým pixelům nemusí být přiřazena žádná hodnota - vznikají díry. [16]

- **Zpětná transformace**

$$(x, y) = \mathbf{T}^{-1}(x', y')$$

U zpětné aproximace se pro každý pixel výstupního obrazu hledá poloha ve vstupním obraze. Její hodnota se aproximuje z okolních bodů vstupního obrazu. Díky tomu nevznikají díry v obraze. [16]

3.4.1 Transformace souřadnic bodů

$$x' = \sum_{r=0}^{m} \sum_{k=0}^{m-r} a_{rk}\, x^r\, y^k\,, \qquad y' = \sum_{r=0}^{m} \sum_{k=0}^{m-r} b_{rk}\, x^r\, y^k$$

Geometrická transformace se obvykle aproximuje polynomem *m-tého* stupně. Neznámé koeficienty se získají řešením soustavy lineárních rovnic, kde slouží jako známé body dvojice sobě odpovídajících bodů (x,y) a (x‘,y‘). V závislosti na míře zkreslení se volí stupeň polynomu *m*.[17]

3.4.2 Bilineární transformace

$$\begin{aligned} x' &= a_0 + a_1 x + a_2 y + a_3 xy\,, \\ y' &= b_0 + b_1 x + b_2 y + b_3 xy\,. \end{aligned}$$

Při použití bilineární transformace stačí použití 4 párů korespondujících bodů. Tato transformace se používá pro korekci perspektivního zkreslení [17]

3.4.3 Afinní transformace

$$\begin{aligned} x' &= a_0 + a_1 x + a_2 y\,, \\ y' &= b_0 + b_1 x + b_2 y\,. \end{aligned}$$

Při použití afinní transformace stačí použití 3 párů korespondujících bodů. Tato transformace se používá pro korekci natočení obrazu. [16]

3.5 Houghova transformace

Houghova transformace je metoda, která se používá pro nalezení parametrů geometrických tvarů v obraze, které lze analyticky popsat. Nejčastěji je tato metoda používaná pro vyhledávání přímek, kružnic a elipsy. Hlavní výhodou této metody je schopnost úspěšného vyhledávání tvarů s přerušovanou hranicí. Metoda je založena na

vytváření prostoru příznaků o n proměnných, do kterého se mapuje vstupní obraz. Nalezením lokálních maxim v prostoru příznaku docílíme nalezení hledaného tvaru. [14] Pro vyhledávání přímek se používá zápis v polárních souřadnicích.

$$r = x \cdot cos\theta + y \cdot sin\theta$$

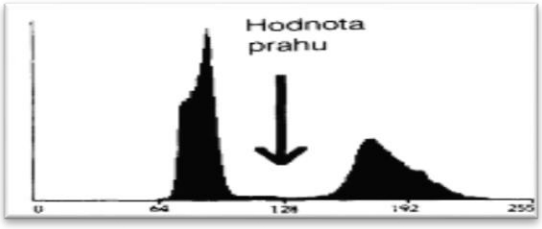
Kde r – představuje délku normály od přímky a θ – úhel mezi osou x a normálou. Příznakový prostor pro vyhledávání přímky bude tedy dvojrozměrný. Postupným dosazováním bodů ze vstupního obrazu dojde k vytvoření spojité křivky v Houghově obrazu. Po promítnutí všech bodů dané přímky dojde k protnutí těchto přímek v jednom bodě (r_{max}, θ_{max}) čímž zjistíme požadované parametry hledané přímky.

3.6 Segmentace

Segmentace obrazu je jedním z nejdůležitějších kroků vedoucích k analýze obsahu obrazu. Cílem je rozčlenit obraz do částí, které souvisí s předměty či oblastmi reálného světa. Požadovaným výsledkem je soubor vzájemně se nepřekrývajících oblastí, které zahrnují jednotlivé objekty v obraze. [13]

3.6.1 Segmentace prahováním

Jedná se o nejjednodušší a nejstarší způsob segmentace, který je stále hojně využíván kvůli své nízké výpočetní náročnosti. Vychází z předpokladu, že různé objekty mají různou odrazivost. Díky tomu je možné, za využití vhodného jasového prahu, tyto objekty oddělit. Správná volba prahu je pro úspěšnou segmentaci zásadní. Často používanou metodou pro zvolení prahu je analýza tvaru histogramu, kde hledáme minima mezi maximy. [14]



Obrázek 8: Ukázka určení prahu pomocí histogramu [13]

3.6.2 Segmentace na základě detekce hran

Tato metoda vychází ze skutečnosti, že jednotlivé objekty jsou ohraničeny hranami. Tyto hranice jsou nalezeny aplikací některého hranového operátoru. Takto nalezené hranice

jsou přímo pro segmentaci nepoužitelné a musejí se dále zpracovat tak, aby odpovídaly hranám objektu. K tomu se využívají různé algoritmy pro sledování hranice. [14]

3.6.3 Segmentace narůstáním oblastí

Základní myšlenkou segmentace narůstáním je rozčlenění obrazu do maximálně souvislých oblastí tak, aby byly z hlediska zvoleného způsobu popisu homogenní. Kritérium homogenity může být různé (např. jasové vlastnosti, textura nebo barva).
Jednou ze základních metod je spojování oblastí. Nejprve je vstupní obraz rozdělen na velké množství podoblastí - nejlépe jeden bod. Poté se začnou jednotlivé oblasti spojovat podle definovaného kritéria. Jakmile už nejdou žádné dvě oblasti spojit, je segmentace dokončena. [13]

3.7 Optické rozpoznávání znaků (OCR)

Optické rozpoznávání znaků je podskupinou počítačového vidění, která se používá pro získání textu z obrazu nebo naskenovaného dokumentu. Často slouží pro získání dat z tištěných dokumentů, faktur, bankovních výpisů, konverzi ručně psaného textu nebo pro rozpoznávání SPZ. [11]
Správné rozpoznání tištěného textu závisí na několika faktorech. Předně musí být snímek kvalitní – čím kvalitnější předloha, tím větší úspěšnost při převodu a hlavně nižší chybovost u hůře rozpoznatelných znaků (m *vs*. n, č *vs*. ě a další). Pro správné rozpoznávání je důležité, aby byl obraz v dostatečném rozlišení (jako minimum se uvádí 150 DPI), kontrastní a nebyl rozostřený.

Mezi nejpoužívanější OCR knihovny patří:
Tesseract – free
ABBYY FineReader – placená
OmniPage – placená

4 KNIHOVNY POTŘEBNÉ PRO PRÁCI

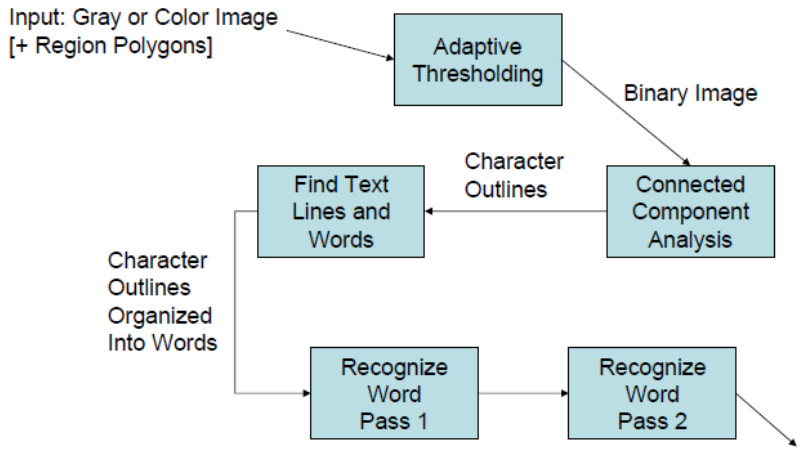
V této kapitole budou popsány knihovny, které jsem využil při tvorbě aplikace.

4.1 Tesseract OCR

Obrázek 9: Logo Tesseract [github.com/tesseract-ocr/]

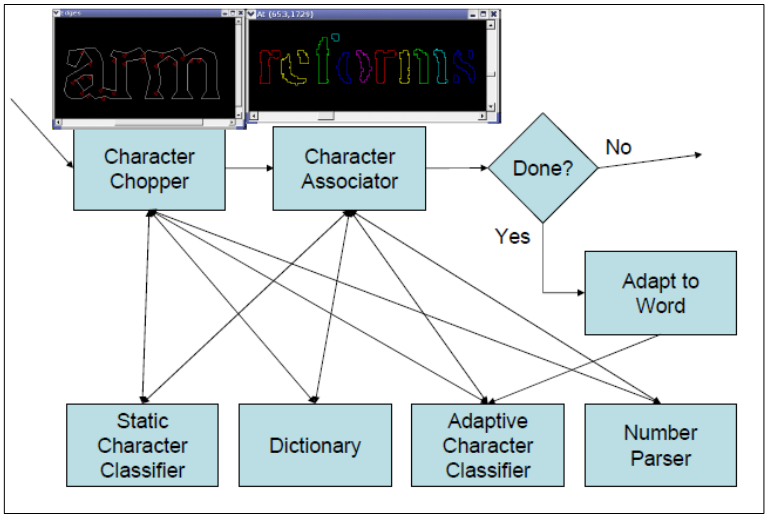
Tesseract je OCR nástroj, jehož vývoj začal jako doktorandská práce už v roce 1985. Od roku 2006 je vyvíjen firmou Google jako open source. Často bývá označován za nejlepší open-source OCR, který je k dispozici. V kombinaci s Leptonica Image Processing knihovnou dokáže přečíst z obrazu celou řadu formátů a převést je na text ve více než 60 světových jazycích. Navíc je plně trénovatelný na nové jazyky a formáty textu. Tesseract podporuje operační systémy Linux, Windows a Mac OSX, takže k jeho využití pro Android aplikaci je nutné použít NDK popsané v kapitole 2.7. [12]

4.1.1 Architektura



Obrázek 10: Ukázka fází rozpoznávání textu [12]

Jako vstupní obraz muže být použit buď originální barevný, nebo již předzpracovaný binární obraz. Tesseract umožňuje definovat oblasti v obraze, ve kterých má vyhledávat. Pokud nebyl vstupní obraz předzpracován uživatelem, Tesseract provede binarizaci pomocí adaptativního prahování.
Dalším krokem je analýza binárního obrazu, kde jsou nalezeny obrysy potenciálních písmen a ty jsou spojeny do shluků. Tyto shluky jsou uspořádány do textových řádků a rozděleny do slov. [12]
Rozpoznávání slov pokračuje jako dvouprůchodový proces pomocí statického a adaptativního klasifikátoru. V prvním průchodu se Tesseract snaží rozpoznat jednotlivá slova pomocí statického klasifikátoru. Každé slovo, které bylo dostatečně rozpoznáno, slouží adaptativnímu klasifikátoru jako trénovací data k doučení. To by mělo mít za následek lepší rozpoznání zbylých slov. V druhém průchodu jsou znovu rozpoznávána slova, která nebyla úspěšně rozpoznána v prvním průchodu s doučeným adaptovaným klasifikátorem.[12]
Rozpoznávání slova je v Tesseract OCR rozděleno do několika částí, jak je vidět na obrázku 11.



Obrázek 11: Rozpoznávání slova

Pokud byla klasifikace slova neúspěšná, přichází na řadu dva nástroje:

- Character Chopper (rozdělovač znaků)
  Tesseract se pomocí něj snaží vylepšit rozdělením shluků s nejhorší důvěryhodností klasifikace do menších částí. Kandidáti na rozdělení jsou nalezeni z konkávních vrcholů obrysů. Takto získaní kandidáti jsou testováni, jestli po jejich rozdělení dojde ke zlepšení klasifikace. Pokud ano, jsou zachovány, jinak jsou zamítnuty.[12]

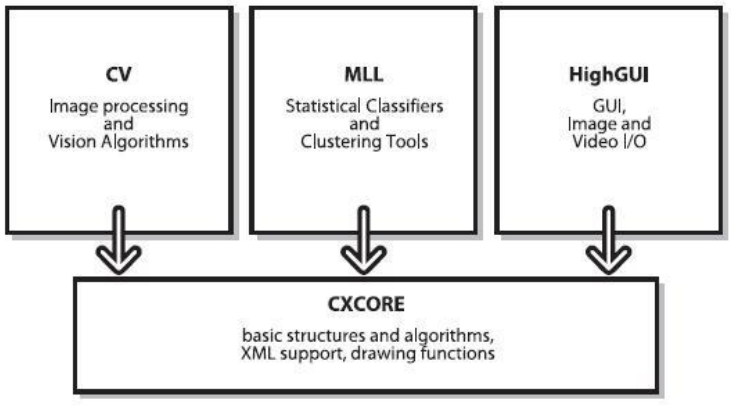
- Associator
  Pokud ani po aplikaci rozdělovače znaků nedošlo k dostatečnému zlepšení klasifikace, přichází na řadu *Associator*. Ten má opačnou úlohu – rozpoznávat neúplné poškozené znaky.[12]

4.2 OpenCV

Obrázek 12: Logo OpenCV

[https://github.com/Itseez/opencv/wiki/OpenCVLogo]

OpenCV *(Open Source Computer Vision)* je otevřená multiplatformní knihovna zaměřená především na zpracování obrazu v reálném čase, ale zvládá i jiné úkoly. Tato knihovna je dostupná pod licencí BSD, tudíž je zdarma pro akademické i komerční využití. Má vytvořený interface pro jazyky C++, Python, Java (Android) a IOS.



Obrázek 13: Struktura knihovny OpenCV

Knihovna OpenCV se skládá ze čtyř hlavních celků, jak je vidět na obrázku. Nejdůležitějším blokem je CV (Computer Vision), který obsahuje funkce pro zpracování obrazu a různé algoritmy počítačového vidění. Dalším blokem je MLL (Machine Learning), který obsahuje algoritmy strojového učení, jako jsou různé statistické klasifikátory a shlukové analýzy. Blok HighGUI se stará o grafické rozhraní knihovny

pro obraz a video. Posledním blokem je CXCORE, který obsahuje potřebné datové struktury k propojení jednotlivých komponent.

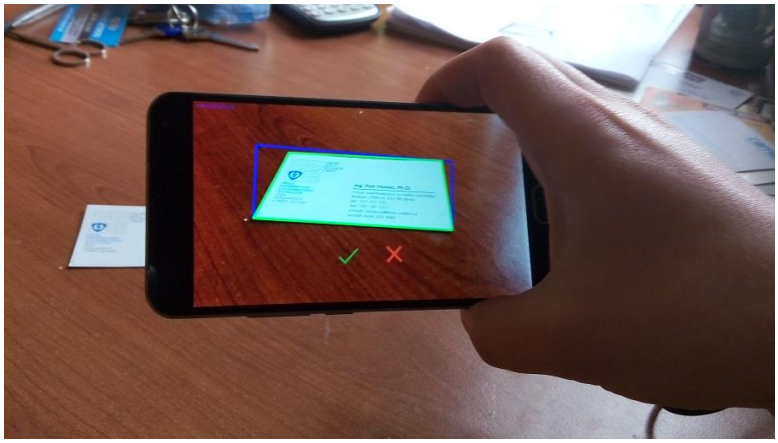
4.3 SQLite

Obrázek 14: Logo SQLite
[https://www.sqlite.org/]

Protože budu v aplikaci potřebovat zpracovávat velkou databázi křestních jmen, která bude sloužit pro rozpoznávání jména kontaktu, rozhodl jsem se použít SQLite.

SQLite je relační databázový systém napsaný v jazyce C. SQLite je volně dostupné pod licencí *public domain* jak pro soukromé, tak i komerční použití.
Na rozdíl od databází založených na principu klient-server, kde je databázový server spuštěn jako samostatný proces, je SQLite pouze malá knihovna, která se přilinkuje k aplikaci a pomocí jednoduchého rozhraní ji lze začít využívat. SQLite byl navrhnut s cílem minimálních hardwarových nároku. Formát databáze je multiplatformní a je tedy možné ji používat mezi různými operačními systémy a architekturami, včetně Androidu. [15]

5 POPIS APLIKACE



Obrázek 15: Ukázka aplikace

V rámci Diplomové práce jsem vytvořil aplikaci pro importování kontaktních údajů z vizitek pro platformu Android. Aplikace vyhledává vizitku v reálném čase, kterou po nalezení analyzuje a vyhledává kontaktní údaje: jméno, telefon a e-mail. Z nalezených položek umožňuje vytvořit nový kontakt v telefonu. Aplikace se skládá z jednoduchého graficky-uživatelského rozhraní, které je rozděleno do dvou Aktivit. Rozhraní jsem se snažil vytvořit co nejjednodušší, bez zbytečných ovládacích prvků.

5.1 MainActivity

MainActivity je první obrazovkou, která se spustí při startu aplikace. Po startu aplikace je ihned zahájeno vyhledávání vizitky v reálném čase. Protože je většina vizitek orientována na šířku, zvolil jsem orientaci *MainActivity* na šířku displeje.
Tato Aktivita je typu Relative Layout kvůli relativnímu umístění tlačítek na střed displeje. Hlavní částí je komponenta *JavaCameraView*, do které se vykresluje náhled z kamery zařízení a průběh vyhledávání kontur vizitky. Tato komponenta je nastavena na celou plochu displeje, aby umožňovala co největší plochu pro vyhledávání. Aplikace upřednostňuje pro vyhledávání zadní kameru zařízení, ale pokud není k dispozici, umožňuje použití i přední kamery. Vyhledávání vizitky je ovládáno pomocí dvou tlačítek, jak je vidět na obrázku 10. Pro jejich vytvoření jsem použil místo obyčejného tlačítka komponentu *ImageButton*, která umožňuje vytvářet grafická tlačítka. Zelená tlačítko "fajfka" slouží pro potvrzení nalezené vizitky a červený tlačítko "křížek" pro zamítnutí a

restartování vyhledávání. Aby bylo zřetelné, že došlo ke stisku, doplnil jsem stisk tlačítek o jednoduchou animaci, kdy se při stisku tlačítka orámují.

Jakmile je vizitka nalezena, vyhledávání se zastaví a je na uživateli, jestli nalezený objekt potvrdí jako vizitku nebo odmítne, jak je ukázáno na obrázku 11. Než se tak stane, je přijímání snímků z kamery pozastaveno a je zobrazena nalezená vizitka, orámovaná červeným obdélníkem.

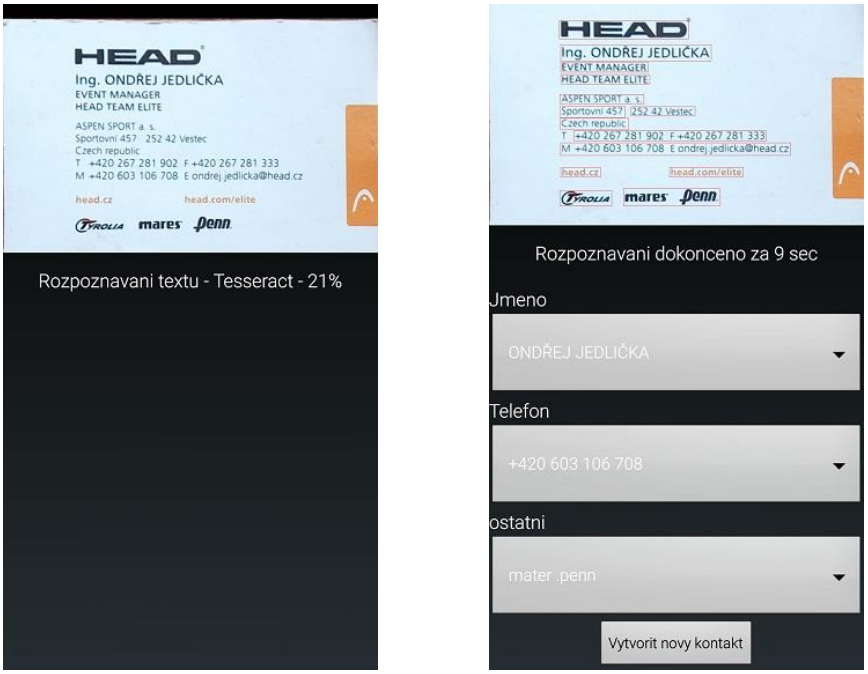
Poslední částí hlavní aktivity je stav vyhledávání v levém dolním rohu. Stav se vykresluje přímo do náhledu z kamery, pomocí vykreslovacích funkcí knihovny OpenCV. Pokud je vizitka potvrzena, je spuštěna další aktivita – *AnalyzeActivity*.

Obrázek 16: Ukázka MainActivity ve stavu vyhledávání

Obrázek 17: Ukázka MainActivity ve stavu Nalezeno, aplikace čeká na zamítnutí nebo potvrzení aktivity

5.2 Analyze Activity

Tato aktivita slouží pro zpracování vizitky. Má rozložení typu Linear Layout, kde jsou jednotlivé komponenty řazeny přímo pod sebou. Orientace aktivity je pevně nastavená na výšku displeje, kvůli přehlednějšímu řazení jednotlivých objektů. Zpracování vizitky v této aktivitě běží asynchronně se zobrazovacími funkcemi, díky čemuž je možné sledovat průběh zpracování. V horní části Aktivity je zobrazena nalezená vizitka po provedení korekce zkreslení. Pod ní se nachází stavový řádek, který zobrazuje aktuální stav rozpoznávání vizitky. Jakmile je rozpoznávání dokončeno, zobrazí se rozpoznané údaje. Pro jejich zobrazení jsem použil komponentu Spinner, který představuje rolovací seznam, umožňující výběr správného údaje. Pro každý vyhledávaný údaj (jméno, telefon, email) jsem vytvořil vlastní Spinner. Poslední Spinner zobrazuje veškerý rozpoznaný text na vizitce. Pokud není některý kontaktní údaj na vizitce nalezen, daný Spinner se vůbec nezobrazí.



Obrázek 18: Ukázka Activity Analyze

Posledním komponentou je tlačítko "*Vytvoř nový kontakt*", které slouží pro vytvoření nového kontaktu v telefonu.

5.3 Omezení a doporučení pro použití aplikace

Rozpoznávací schopnost aplikace je omezena rozlišením displeje zařízení. Snímky, které jsou zpracovávány z kamery v reálném čase, jsou vždy maximálně v takovém rozlišení, jako je rozlišení displeje, jak je zmíněno v kapitole 8.2. Toto omezení je aplikované přímo v knihovně OpenCV ve třídě *CameraBridgeViewBase* kvůli zohlednění výkonu různých zařízení. Vývojář používající tuto třídu nemá možnost získat snímek v plném rozlišení snímače zařízení, což je škoda. Tento nedostatek jsem se snažil vyřešit vlastní modifikací této třídy, ale bohužel neúspěšně.
Aby mohla aplikace správně nalézt a rozpoznat vizitku, je tedy důležité použít telefon s dostatečným rozlišením displeje. V praxi se ukázalo jako dostatečné rozlišení HD (720p). Aplikace může teoreticky pracovat na libovolném rozlišení, ale s klesající hodnotou rozlišení displeje klesá i míra úspěšnosti správného rozpoznání.
Jako minimální verzi SDK jsem zvolil 14, což odpovídá verzi 4.0 Ice Cream Sandwich. Toto omezení pokrývá přibližně 95% zařízení na světě. Jelikož nevyužívám žádnou funkci Andoridu, která byla implementována ve vyšší verzi, nebyl důvod volit vyšší verzi SDK.

Pro zajištění správného lokalizování vizitky by měly být splněny tyto podmínky:

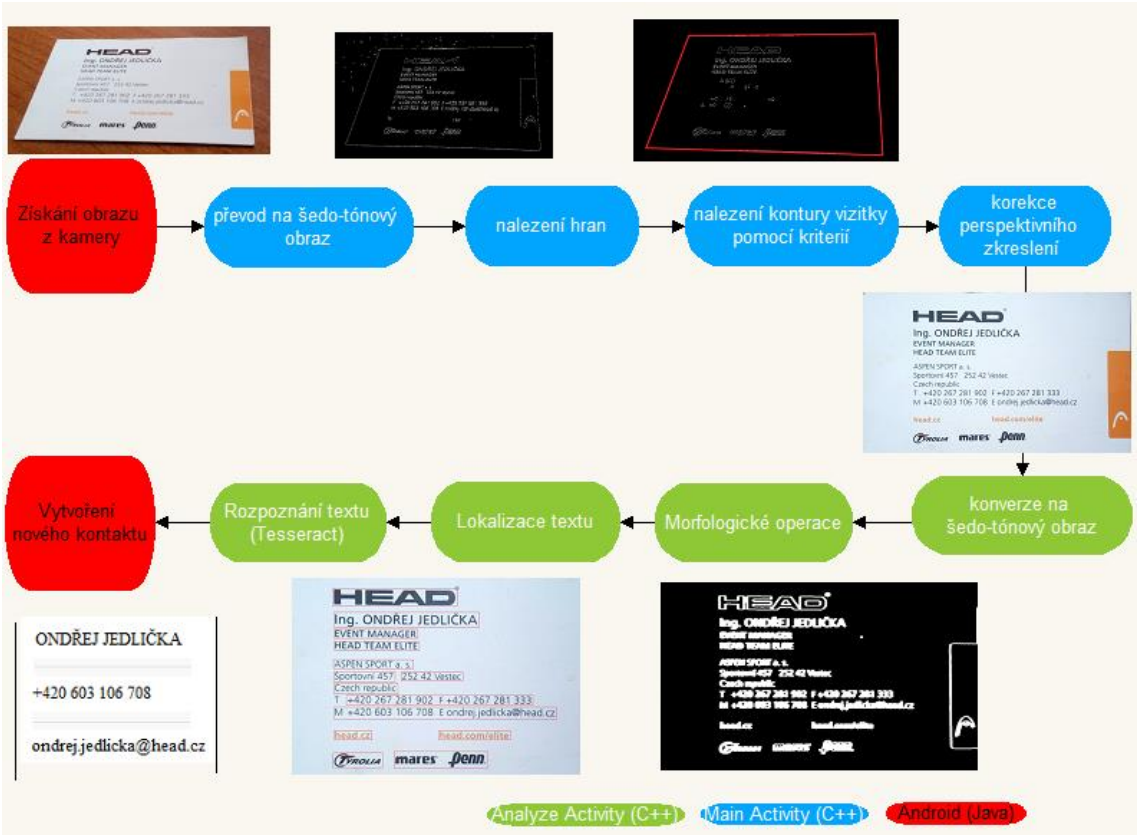
- Snímání za dobrých světelných podmínek. Šero a stíny ztěžují nalezení vizitky a následné rozpoznávání.
- Vizitka by měla být vyhledávána na kontrastním pozadí. Pokud je vizitka snímána na odstínově podobném pozadí, nemusí být správně lokalizována.

Pro zajištění správného rozpoznání kontaktních údajů z vizitky by měly být splněny tyto podmínky:

- Snímat vizitku tak, aby zabírala co největší část displeje (alespoň polovinu). Čím větší bude oblast vizitky, tím více dat je k dispozici pro vyhledávání kontaktních údajů.
- Snímat vizitku ve směru rovnoběžném s textem. Pokud bude vizitka natočená o více než 45°, aplikace ji během korekce nesprávně otočí.
- Snímat vizitku pokud možno kolmo. I když je aplikována perspektivní transformace, ztrácí se při ní rozlišení nutné pro správnou funkci OCR.

6 VÝVOJ APLIKACE

V této kapitole bude postupně popsán průběh aplikace tak, jak je znázorněný na obrázku 19.

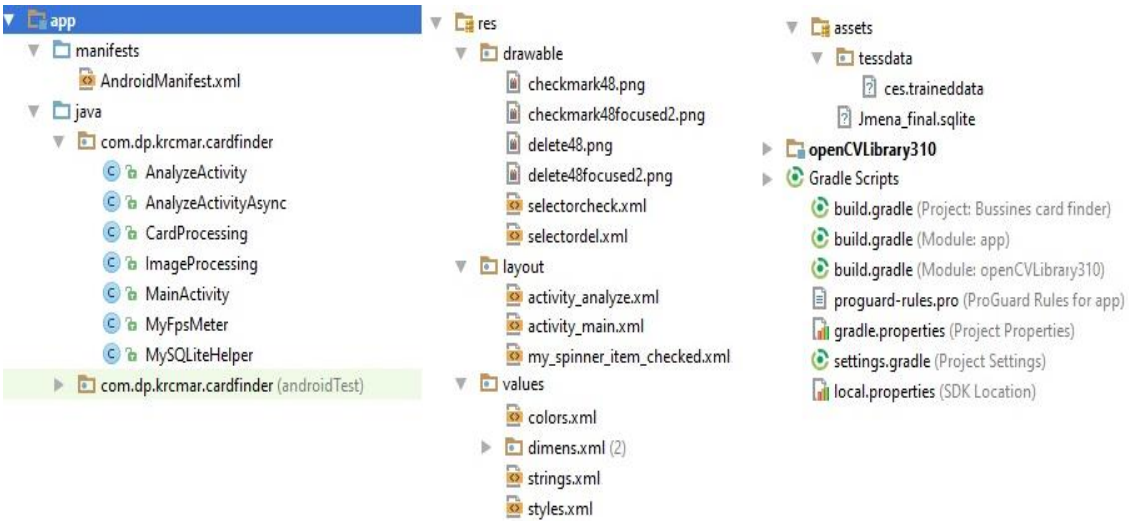


Obrázek 19: Průběh zpracování vizitky

Vizitka je jednoduchý informační prostředek, který nese informaci o obchodní společnosti nebo individuální osobě. Nejčastěji obsahuje kontaktní údaje, jako je vlastníkovo jméno, telefon, email a adresa. Problémem při lokalizaci vizitky je, že existuje velké množství různých vizitek. Mohou mít různou barvu, různou orientaci (na výšku nebo na šířku) a různý poměr stran. Jediným společným znakem vizitek je obdélníkový tvar. Proto jsem vyhledávání vizitky v obraze založil na vyhledávání největšího obdélníkového tvaru.

Aplikaci jsem vyvíjel v prostředí Android Studio, které je vhodné pro vytváření mobilních aplikací, ale není moc vhodné pro vytváření a ladění funkcí počítačového vidění. A to hlavně z důvodu, že existuje pouze jedno zobrazovací okno – displej telefonu. V platformě Android si nemůžu zobrazit průběh zpracování obrazu pomocí více oken jako na PC, což je nepraktické. Další nevýhodou je poměrně dlouhé sestavování aplikace

a její následné nahrávání do telefonu po každé úpravě kódu, které je několika násobně delší než na stolním počítači. Tento proces trvá na mém, nepříliš výkonném počítači, mezi dvěma až čtyřmi minutami, což je dost pomalé. Z toho důvodu jsem se rozhodl nejprve vytvořit a otestovat algoritmy pro vyhledávání a rozpoznání vizitky v jazyce C++ ve Visual Studiu 2013, což sice zabralo další čas pro implementaci OpenCV a Tesseract do dalšího vývojového prostředí, ale výrazně to zjednodušilo práci. Téměř všechny funkce OpenCV knihovny existují jak v C++ verzi tak i v Java verzi, proto nebyl problém přepsat již otestované řešení z C++ do Javy.
Funkce, pro vyhledávání vizitky jsou implementovány ve třídě *ImageProcessing* a funkce pro analýzu vizitky ve třídě *CardProcessing*. Struktura aplikace je zobrazena na obrázku 20.
V následujících kapitolách bude popsán vývoj aplikace, kde kapitoly jsou řazeny stejně jako operace prováděné při rozpoznávání vizitky viz obrázek 19.



Obrázek 20: Struktura aplikace v prostředí Android Studio

6.1 AndroidManifest

Manifest je základním souborem každé aplikace platformy Android. Jsou zde definovány základní náležitosti aplikace, jak bylo popsáno v kapitole 2.5. Výňatek z manifestu je zobrazen níže, celý manifest je k dispozici v příloze.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.dp.krcmar.cardfinder" >

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@android:style/Theme.NoTitleBar.Fullscreen">

        <activity
            android:name="com.dp.krcmar.cardfinder.MainActivity"
            android:configChanges="keyboardHidden|orientation"
            android:screenOrientation="landscape" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="com.dp.krcmar.cardfinder.AnalyzeActivityAsync"
            android:screenOrientation ="portrait" >
        </activity>
    </application>
</manifest>
```

- uses-permission – popisuje jaké systémové zdroje aplikace využívá. Moje aplikace potřebuje přístup ke kameře a přístup k externí paměti, na kterou se ukládají data pro Tesseract a snímek vizitky během zpracování. Proto je potřeba tyto zdroje definovat.
- application – základní element manifestu, který popisuje základní náležitosti aplikace. Je zde definována ikona, název a všechny aktivity, které do aplikace patří.
- Activity - deklaruje jednotlivé aktivity aplikace. Slouží jako hlavička aktivit. Je zde definována konfigurace aktivity, hlavně její orientace. Pomocí komponenty *intent-filter* se definuje aktivita, která bude zobrazena při spuštění aplikace.

6.2 Získání snímku z kamery

Pro zpracování snímku z kamery v reálném čase jsem použil předdefinovanou třídu *CameraBridgeViewBase,* která byla přímo vytvořena pro zpracování snímků v reálném čase. Je to základní třída, která provádí interakci s kamerou zařízení a knihovnou *OpenCV*. Hlavní částí této třídy je objekt *CvCameraViewListener*, který má na starosti zpracování snímku z kamery. Má tři stavy, při kterých jsou volány tyto metody:

6.2.1 onCameraFrame

Jedná se o nejdůležitější metodu celé třídy. Umožňuje editovat snímek z kamery před tím, než je zobrazen na displeji. Editovaný snímek vrací jako výstupní parametr. V této metodě probíhá vyhledávání vizitky v reálném čase.
OpenCV automaticky vybere podle parametrů kamery nejvyšší možné rozlišení, které je zároveň současně možné zobrazit na displeji (v závislosti na jeho rozlišení). Podporovány jsou snímky o velikosti:
1920×1080, 1440×1088, 1280×720, 1088×1088, 960×720, 960×544, 800×480,768×464, 768×432, 720×480, 640×480, 640×384, 640×368, 576×432, 480×320, 384×288,352×288, 320×240, 240×160 a 176×144 obrazových bodů.

6.2.2 onCameraViewStarted

Tato metoda je volána před tím, než začnou přicházet snímky z kamery. Slouží k inicializaci proměnných, ve kterých budou zpracovávány snímky. Inicializace proměnných je důležitá pro správnou práci s proměnnými.

6.2.3 onCameraViewStopped

Tato metoda je volána po zastavení přijímání snímků z kamery. Slouží pro úklid proměnných pro přijímání snímků, které byly vytvořeny v metodě *onCameraViewStarted.*

6.3 Lokalizace vizitky

Aby bylo možné importovat kontakty z vizitky, je nutné ji nejprve správně lokalizovat. Jak jsem psal výše, hlavním znakem vizitky je její obdélníkový tvar. Proto lokalizace vizitky probíhá jako lokalizace největšího obdélníkového tvaru v obraze, který splňuje určitá kritéria. Dále je na uživateli, aby nalezený objekt potvrdil jako vizitku nebo odmítl. Jelikož vyhledávání vizitky probíhá v reálném čase, je nutné vyhledávání realizovat s co možná nejnižší výpočetní náročností, aby byla zachována plynulost obrazu.

6.3.1 Předzpracování obrazu

Předzpracování obrazu zahrnuje převod vstupního barevného obrazu na šedo-tónový, odstranění šumu a nalezení hran v obraze. Jasová korekce vstupního obrazu už není nutná, jelikož kamera zařízení ji provádí automaticky během pořizování snímku.

Obrázek 21: Ukázka šedo tónového a výsledného hranového obrazu

6.3.2 Odstranění šumu

Kvůli vysoké výpočetní náročnosti jsem nemohl použít klasické metody odstranění šumu s konvolučním jádrem, které byly popsány v kapitole 3.1.3. Hlavní nevýhodou těchto metod je vysoká výpočetní náročnost, která roste s velikostí konvolučního jádra. Proto jsem se snažil nalézt méně výpočetně náročnou metodu a nakonec jsem použil pyramidovou metodu. Tato metoda nejprve provede snížení rozlišení podvzorkováním vstupního obrazu a následně zvýší rozlišení na původní hodnotu. Chybějící hodnoty interpoluje podle nejbližších sousedů z podvzorkovaného obrazu. Jelikož se při podvzorkování a zpětnému navzorkování ztratí část informace, dochází k mírnému rozmazání obrazu a odstranění šumu. Tato metoda sice využívá konvolučního jádra pro interpolaci chybějících hodnot, ale používá velikost jen 3x3. Díky tomu se ukázala jako výrazně rychlejší, než metody s výpočtem okolí, což se projevilo dvojnásobnou hodnotou FPS.

Obrázek 22: Ukázka hran bez filtrace šumu

Obrázek 23: Ukázka hran s filtrací šumu mediánovým filtrem s velikostí jádra 5x5

Obrázek 24: Ukázka hran s filtrací šumu Pyramidovým filtrem

6.3.3 Nalezení hran

Pro nalezení hran jsem použil Cannyho hranový detektor, který byl popsán v kapitole 3.2. Hodnoty parametrů detektoru jsem zvolil:

- 50 – dolní práh
- 150 – horní práh

Tyto hodnoty jsem volil experimentálně. Pokoušel jsem se je stanovit z tvaru histogramu, na jehož analýzu jsem vytvořil funkci *FindPeaks*, ale takto získané hodnoty prahů nedosahovaly lepších výsledků než pevně zvolené prahy. Proto jsem zvolil jednodušší variantu s pevně nastavenými prahy.

6.4 Vyhledávání kontur vizitky

Pro vyhledávání kontury vizitky jsem vytvořil funkci *FindRectangles*, která je implementována ve třídě *ImageProcessing*. Tato funkce nalezne všechny kontury v obraze a vybere z nich ty, které splňují tato kritéria:

- nezasahují mimo displej
- kontura je tvořena nejméně 3 - mi body
- obsah kontury je větší než 1% displeje
- poměr stran obdélníku obepínající konturu leží v rozsahu 1.0 až 3. Teoretický poměr stran vizitek je v rozsahu (1.4 až 1.8), který jsem rozšířil o případné zkreslení vizitky.

Jako výsledný kandidát na vizitku je vybrána kontura s největším obsahem. Obsah kontury jsem počítal jako obsah minimálního obdélníku, který obepíná konturu. Proces vyhledávání signalizuje stavový popis v levém dolním rohu.

Obrázek 25: Vyhledávání vizitky

Nejprve jsem se snažil vyhledávat vizitku pomocí Houghovy transformace modifikované pro vyhledávání přímek, která byla popsána v kapitole 3.5. Pomocí této metody jsem vyhledal přímky v obraze a pomocí průsečíků těchto přímek určil oblast vizitky. Tato metoda měla lepší úspěšnost vyhledávání než analyzování kontur. Především výrazně lépe detekovala rohy vizitky, což je důležité pro korekci zkreslení. Bohužel se tato metoda ukázala jako příliš výpočetně náročná, než aby šla použít při zpracování obrazu v reálném čase na mobilním zařízení.

6.5 Nalezení vizitky

Obrázek 26: Ukázka nalezené vizitky

Problém jak rozpoznat, jestli je aktuálně nalezená kontura výsledným objektem jsem vyřešil předpokladem, že se daná kontura musí vyskytovat v obraze po určitou dobu. Tuto dobu jsem stanovil na 1s, což mi připadá jako optimální doba pro vyhledávání. Proto jsem si vytvořil třídu *MyFPS,* která měří čas mezi vykreslováním jednotlivých snímků a převádí ho na počet snímků za sekundu (FPS).
Nalezení vizitky je realizováno pomocí zásobníku, do kterého se ukládají kontury kandidátů na vizitku z jednotlivých snímků. Zásobník je plněn zepředu, kde se na první pozici uloží kontura z aktuálního snímku, ostatní se posunou o jednu pozici dozadu a poslední se smaže. Zásobník má proměnnou délku, která se upravuje podle průměrné hodnoty měřeného FPS. Díky tomu běží vyhledávání stejně rychle na zařízeních s různým výkonem a tedy různou hodnotou FPS. Po zpracování každého snímku proběhne analýza zásobníku, kde se zjišťuje shoda mezi jednotlivými konturami. Jako kritérium shody jsem zvolil polohu středu, obsah a poměr stran kontury s tolerancí 10%. Pokud se kontura z aktuálně analyzovaného snímku shoduje alespoň s 50% kontur uložených v zásobníku, je prohlášena za výslednou a vyhledávání je ukončeno. Pokud ji uživatel odsouhlasí, vyhledávání vizitky je dokončeno. Pokud ji uživatel zamítne, vyhledávání se spustí od začátku. Touto kapitolou končí část, která běží v reálném čase.

6.6 Korekce zkreslení

Vzhledem ke skutečnosti, že vizitka nemusí být snímána z ideální pozice, a může být zkreslena nepřímím snímáním kamery, je vhodné před zpracováním vizitky provést transformaci, která bude toto zkreslení kompenzovat. Moje aplikace je schopná provést dvě různé transformace pro korekci zkreslení v závislosti na tom, jak kvalitně je vizitka nasnímána.

6.6.1 Perspektivní transformace

Obrázek 27: Výsledná vizitka po perspektivní transformaci

Pro perspektivní transformaci je potřeba nalézt čtyři vzájemně si korespondující body vstupního a výstupního obrazu, které jsou použity k vytvoření transformační matice. Jedná se o bilineární typ transformace, který byl popsán v kapitole 3.4.2. Jako vzájemně si odpovídající body jsem zvolil rohy vizitky nalezené z kontury (obr.28 - zelené body) a rohy minimálního obdélníku, který obepíná vizitku (obr. 28 - červené body). Tato transformace muže být provedena, pokud je nalezena celá kontura vizitky nebo alespoň část, která obsahuje všechny čtyři rohy vizitky.

Obrázek 28: Ukázka čtyř vzájemně korespondujících bodů

Pro vytvoření transformační matice existuje v knihovně OpenCV funkce *getPerspectiveTransform*, jejímž vstupem jsou dvě čtveřice vzájemně si odpovídajících bodů. Pro nalezení a provedení perspektivní transformace jsem vytvořil vlastní funkci *PerspectiveTransform.*

Jelikož takto zvolený minimální obdélník je také zatížen zkreslením poměru stran, je potřeba výsledný poměr stran dopočítat. Jako šířku a výšku výsledného obdélníku použiji delší z dvojice stran nalezené vizitky. Poměr stran běžných vizitek je v intervalu od 1.3 do 1.8. Pokud poměr stran výsledného obdélníku leží mimo tento interval, je upraven na hodnotu 1.6 zvětšením nebo zmenšením výšky vizitky.

Obrázek 29: Nalezená vizitka před korekcí a po korekci

6.6.2 Afinní transformace

Druhou variantou je afinní transformace. Tato transformace slouží pouze jako záloha, pokud selže vyhledávání rohů vizitky a nemůže být provedena perspektivní transformace jak je vidět na obrázku 30. Tato transformace alespoň provede korekci natočení vizitky pomocí afinní transformace, která byla popsána v kapitole 3.4.3 Úhel natočení je získán pomocí minimálního natočeného obdélníku, který je zobrazen na obrázku červenou barvou. Pro výpočet tohoto obdélníku pro danou konturu existuje v OpenCV funkce *minAreaRect*.

Obrázek 30: Ukázka Vizitky před a po afinní transformaci

Nejprve je pomocí funkce *GetRotationMatrix2D* vypočtena rotační matice, která je následně aplikována na vstupní obraz pomocí funkce *warpAffine*. Z takto získaného obrazu už jen stačí vybrat část vizitky pomocí souřadnic minimálního obdélníku.

6.7 Převod dat mezi Aktivitami

Rozpoznávání vizitky je rozděleno do dvou aktivit. První aktivita se stará o vyhledání vizitky v reálném čase a druhá o její zpracování, tudíž je potřeba mezi aktivitami přenést data. Jednou z metod je použití komponenty *Bundle*, která umožňuje pomocí funkce *PutExtras()* přenášet data mezi aktivitami. Nejčastěji se používá pro přenos méně objemných dat, ale lze ji použít i pro přenos bitmapy. Během testování jsem zjistil, že maximální velikost takovéto přílohy je 1Mb, což může být pro přenos snímku vizitky nedostačující. Proto jsem se rozhodl použít variantu, kdy je snímek vizitky nejprve uložen do paměti telefonu, pomocí *Bundle je* odeslána pouze jeho adresa a v nové aktivitě je z této adresy znovu vytvořena bitmapa. Tento postup je časově náročnější, ale zajišťuje přenos libovolně velkého snímku.

6.8 Lokalizace textu

V této kapitole bude popsáno nalezení textu na vizitce. Lokalizace textu probíhá v aktivitě *AnalyzeActivity,* kam byla odeslána nalezená vizitka po korekci. Během lokalizace předpokládám, že text na vizitce je orientován rovnoběžně s vizitkou. Jinak natočený text nebude rozpoznán.

6.8.1 Předzpracování vizitky

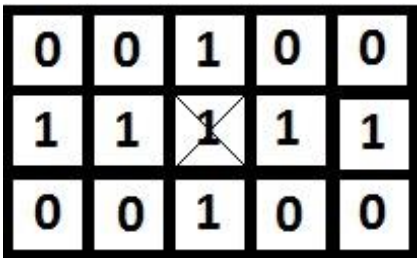
Předzpracování vizitky je provedeno ve třech krocích:

- převod na šedo-tónový obraz
- nalezení hran vizitky
- morfologická dilatace

Obrázek 31: Ukázka jednotlivých kroků předzpracování vizitky (originální obraz, hranový a morfologicky dilatovaný)

Během testování algoritmů ve Visual Studiu jsem použil pro segmentaci textu algoritmus *MSER(Maximal stable external regions),* který se jevil jako ideální pro detekci textu. Nebyl citlivý na šum, jako obyčejný hranový detektor a detekoval textová pole s vysokou úspěšností. Když jsem jej chtěl použít v Java implementaci OpenCV, zjistil jsem, že pro tuto implementaci ještě není k dispozici. To byl jediný rozdíl, který jsem mezi Java a C++ verzí našel.

Z toho důvodu jsem pro segmentaci textu z vizitky znovu použil Cannyho hranový detektor a analýzu kontur. Dále jsem použil morfologickou dilataci, která zajistí splynutí mezer mezi písmeny a dosáhne spojení slov. Jako strukturní element dilatace jsem zvolil element velikosti 3x5, který je na obrázku 32. Tento tvar jsem zvolil z toho důvodu, aby došlo ke splynutí mezer mezi písmeny ve vodorovném směru, ale aby současně nebyly poškozeny mezery mezi řádky. Velikost elementu jsem zvolil experimentálně, z čehož tato varianta dosahovala nejlepších výsledků. Pokud jsem použil větší element, docházelo u některých vizitek k řádkovému slévání textových oblastí, které už k sobě nepatřily, pro jsem se rozhodl použít raději menší element za cenu nespojení některých slov.

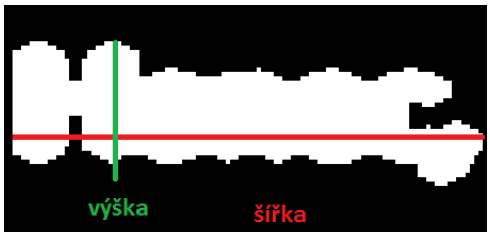


Obrázek 32: Ukázka použitého morfologického jádra

6.8.2 Segmentace textových oblastí

Objekty nalezené v morfologicky dilatovaném obrazu jsou postupně filtrovány na textové oblasti podle tohoto algoritmu:

1. Filtrace horizontálních objektů porovnáním výšky a šířky objektů
   - Pokud je objekt více než 2krát vyšší než širší je zamítnut. Tato filtrace odstraní svislé objekty v obraze.



Obrázek 33: Dilatovaný objekt vizitky

2. Filtrace nesouměrných objektů porovnáním skutečného obsahu a obsahu minimálního obdélníku objektu

Obrázek 34: Porovnání obsahu minimálního obdélníku a skutečného obsahu objektu

Skutečný obsah je vypočten pomocí matematických momentů pro každý objekt. Jak je vidět na obrázku, tyto obsahy by se neměly pro text příliš lišit. Pokud je skutečný obsah nižší než 30% obsahu obdélníku, je objekt zamítnut. Tato filtrace odstraňuje zbytky různých log nebo hran vizitky.

3. Filtrace pomocí výškového histogramu

Obrázek 35: Filtrace nízkých objektů

Z výšek jednotlivých objektů je sestaven výškový histogram, ze kterého je zjištěna nejčastější výška textových objektů. Takto je získána nejčastější velikost písma,

pomocí které jsou odfiltrovány malá objekty. Bohužel toho nejde využít i pro filtraci velikých objektů, protože důležité údaje na vizitce jako je jméno nebo název firmy bývají napsány větším písmem než zbytek textu.

4. Spojovaní oblastí

Obrázek 36: Spojování nalezených oblastí

Pro spojování textových oblastí jsem vytvořil funkci *MergeCloseBoundingBoxes*, která rekurzivně prochází jednotlivé oblasti a spojuje je. Kritériem pro sjednocení je, aby ležely na stejném řádku a nebyly od sebe vzdáleny více než jeden znak. Jakmile jsou všechny možné oblasti spojeny, přijde na řadu funkce *RemoveOrphans*, která odstraní oblasti s šířkou menší než jeden znak.

5. Adaptativní prahování

Obrázek 37: Adaptativní prahování

- Posledním krokem segmentace oblastí je adaptativní prahování. Pro každou nalezenou oblast je extrahován výřez z původního obrazu, ze kterého je pomocí adaptativního prahování získaný binární obraz. Adaptativní prahování používá pro výpočet hodnoty prahu aktuální okolí pixelu, proto je vhodné pro prahování textových oblastí. Takto vzniklý binární obraz je připraven pro Tesseract.

6.9 Rozpoznávání textu (Tesseract)

Obrázek 38: Ukázka nalezených textových oblastí a rozpoznaného textu

Moje aplikace používá pro rozpoznání textu knihovnu *tess-two* ve verzi 3.04, která je odnoží Tesseract Tools pro Android. Implementování této knihovny do Android aplikace bylo obtížné. Bylo potřeba tuto knihovnu správně zkompilovat pomocí NDK-build a správně ji implementovat do systému *Gradle,* který slouží pro sestavení aplikace. Hlavním překážkou byla velice špatná dokumentace pro použití této knihovny v systémem Android. Většinu informací jsem musel čerpat z různých uživatelských webů, které nebyly moc spolehlivé.

Tesseract umožňuje rozpoznávání více než 60 světových jazyků, ale aby toho byl schopen, musí mít k dispozici natrénovaná data pro konkrétní jazyk. Tato data jsou dostupná na oficiálních stránkách Tesseract OCR. Tyto datové soubory jsou relativně veliké, česká verze má 12 Mb. Proto je výsledná aplikace relativně velká.

Moje aplikace slouží pro zpracování českých vizitek, proto jsem použil natrénovaná data pro češtinu. Pro předávání dat aplikaci existuje v každém projektu v Android Studiu složka “assetes”, do které jsem data nahrál. Z této složky si moje aplikace zkopíruje trénovací data na SD kartu, aby mohly být použity při inicializaci Tesseractu. Teď už jen stačí předat lokalizované oblasti ke zpracování.

Rozpoznání textu pomocí Tesseract OCR zabere zhruba 90% celkového času rozpoznávání vizitky a může trvat od 1s až do 10s v závislosti na počtu nalezených textových oblastí. Z toho důvodu provádím rozpoznávání asynchronně, v novém vlákně. Díky tomu můžu zobrazovat průběh aplikace i rozpoznávanou vizitku, což by zpracování v jednom vlákně neumožňovalo.

Jako vstup Tesseractu předávám binární obrazy jednotlivých textových oblastí, které byly získaný během segmentace. Výstupem je nalezený text v kódování UTF-8. V aplikaci je tato operace znázorněna ve stavovém řádku (viz obrázek):

Obrázek 39: Ukázka grafického zobrazení průběhu rozpoznávání

6.10 Nalezení kontaktních údajů

Jakmile je k dispozici rozpoznaný text z Tesseract OCR, přichází na řadu další úkol, kterým je nalezení údajů pro vytvoření nového kontaktu v telefonu. K tomu jsem využil filtrování rozpoznaného textu pomocí regulárních výrazů. V jazyce Java je použita syntaxe regulárních výrazů z *Pearl5*. Pro použití regulárních výrazů jsou v Jave vytvořeny objekty *Parser* a *Matcher*. *Parser* slouží pro definici regulárního výrazu a *Matcher* aplikuje výraz na text.

6.10.1 Jméno

Vyhledávání jména je z těchto třech kontaktních údajů nejsložitější. Jméno nemá oproti emailu a telefonnímu číslu žádné výrazné syntaktické znaky, které by ho jasně definovaly. Proto jsem se, kromě regulárních výrazů, rozhodl ještě využít databázi všech křestních jmen v České republice. Tato databáze má zhruba 360 000 položek a je volně dostupná na oficiálních stránkách Ministerstva vnitra. K jejímu zpracování jsem použil databázový systém *SQLite*, jelikož takto rozsáhlý soubor dat by nešel zpracovávat pomocí standardních zásobníků v jazyce Java. Tuto databázi načítám přes složku „*assets*“, stejně jako trénovací data pro Tesseract. Pro práci s databází jsem vytvořil třídu *MySQLiteHelper,* ve které jsou implementovány metody pro načtení, inicializaci a prohledávání databáze.

Vyhledávání jména tedy probíhá ve dvou fázích:

- **Filtrace pomocí regulárních výrazů**

  Pro vyhledávání jména jsem vytvořil tři regulární výrazy, které pokrývají různé kombinace velkých a malých písmen: (*Jméno Příjmení, JMENO PŘIJMENÍ, Jméno PŘIJMENÍ*).
  Regulární výraz pro detekci varianty (*Jméno Příjmení*) vypadá následovně:

```
\\b[A-ZŠČŘŽŇŤĎ][a-zěščřžťýáíéúůďň]{2,15}([- ]{1,3}->
    ->[A-ZŠČŘŽŇŤĎ][a-zěščřťžýáíéúůďň]{2,15}){1,2}
```

  Vysvětlení významu částí regulárního výrazu je rozlišeno barvami:
  Musí začínat velkým písmenem, následuje 2 až 15 opakování malých písmen, následuje 1 až 3 opakování mezery nebo pomlčky, následuje velké písmeno a 2 až 15 opakování malých písmen. Část příjmení se může opakovat jednou nebo dvakrát proto, aby filtr reagoval i na jména tvořené třemi slovy.
  Varianty regulárních výrazů pokrývající zbylé dvě kombinace velkých a malých písmen fungují stejně.

- **Filtrace pomocí databáze křestních jmen**
  Všechny textové řetězce, které projdou filtrací regulárními výrazy, jsou rozděleny na jednotlivá slova. Pro každé slovo je zjišťováno, jestli se nachází v databázi křestních jmen. Pokud ano, je celý textový řetězec prohlášen za jméno. Jinak je zamítnut.

6.10.2 Telefon

Vyhledávání telefonního čísla je oproti vyhledávání jména relativně snadné a stačí k tomu jeden regulární výraz. Tento výraz hledá posloupnost čísel v určitém tvaru:

```
\\+420|\\+421)?(-| |\\\\|/)?[0-9]{3}(-| |\\\\|/)?
            [0-9]{3}(-| |\\\\|/)?[0-9]{3}
```

Vysvětlení významu částí regulárního výrazu je rozlišeno barvami:
Může začínat předvolbou +420 nebo +421, může následovat jeden dělící znak "-| |\\\\|/", následuje posloupnost tří čísel. Zelená a červená část se třikrát opakuje
Tento výraz rozpozná čísla ve tvaru: +420 123456789, 123 456789, 123-456-789, 123/456/789, 123456//789, ale nerozpozná řetězce, který obsahuje méně jak 9 číslic nebo obsahuje falešný znakem.

6.10.3 Email

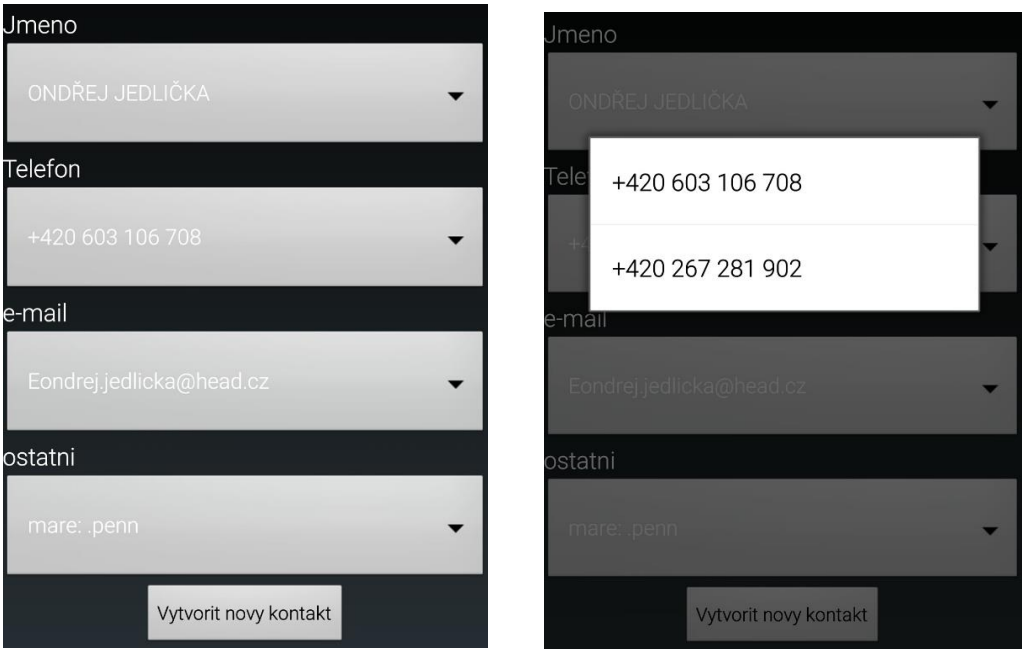
Vyhledávání emailu je založeno na vyhledávání znaku "@" a doménové přípony (např. *.com*), který je obklopen textem v určitém formátu. Detekce je realizována jedním regulárním výrazem:

```
[a-zA-Z0-9\\. \\-|]+[@][a-zA-Z0-9\\.\\-|]+\\.+[a-zA-Z]{2,4}
```

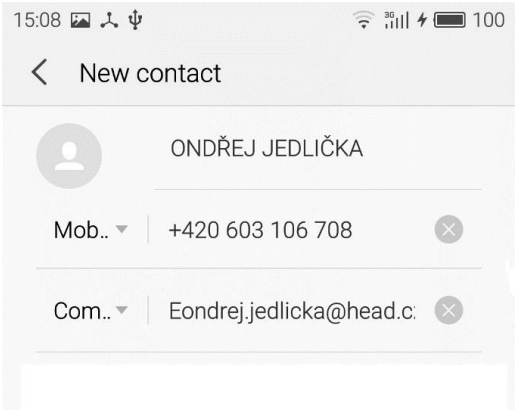
Vysvětlení významu částí regulárního výrazu je rozlišeno barvami:
Libovolný počet znaků z množiny "[a-zA-Z0-9\\._\\-|]", následuje jeden znak @, následuje libovolný počet znaků z množiny "[a-zA-Z0-9\\.\\-|]+", následuje tečka a 2 až 4 opakování písmen.
Tento výraz rozpozná všechny standardní formáty emailové adresy, ale neporadí si se špatně rozpoznanými znaky v adrese.

6.11 Vytvoření nového kontaktu

Nalezené kontaktní údaje jsou zobrazeny v komponentě *Spinner*, která představuje rolovací seznam. Umožňuje zvolit si správný údaj, pokud jich bylo nalezeno více. Po zvolení správných údajů stačí kliknout na tlačítko "Vytvořit nový kontakt", které zavolá nový systémový *Intent* pro vytvoření nového kontaktu v adresáři telefonu. Kontaktní údaje jsou předány pomocí objektu *Bundle*, jak bylo popsáno v kapitole 6.5.



Obrázek 40: Ukázka nalezených kontaktních údajů



Obrázek 30: Ukázka vytvoření nového kontaktu

7 ÚSPĚŠNOST ROZPOZNÁVÁNÍ VIZITKY

Rozpoznávací úspěšnost aplikace jsem testoval na skupině 40 vizitek, jejíž část i s rozpoznaným textem je ukázána v příloze. Tato testovací skupina je tvořena běžnými vizitkami, které jsem nashromáždil ve svém okolí. Pro ověřování úspěšnosti byl použitý telefon Meizu M2 s FullHD displejem. Měření úspěšnosti rozpoznávání jsem rozdělil do dvou částí:

7.1.1 Úspěšnost lokalizace vizitky

Nejprve byla zjišťována úspěšnost lokalizování vizitky v obraze, kdy ze 40 vizitek bylo správně lokalizováno 35, což představuje úspěšnost 87,5%. Lokalizaci jsem prohlásil za úspěšnou, pokud byla nalezena alespoň taková část vizitky, aby mohla být provedena perspektivní transformace. Vyhledávání je silně závislé na kvalitě vizitky a na scéně, ve které je vyhledáváno. Běžná bílá vizitka bude správně lokalizována téměř se 100% úspěšností, kdežto různobarevná vizitka s různými obrázky bude správně lokalizována s výrazně menší pravděpodobností. Proto je složité stanovit konkrétní úspěšnost tak, aby nebyla zavádějící. Obecně se jako nejproblematičtější vizitky pro lokalizaci jevili ty, které měly jako pozadí fotku nebo různobarevný motiv.

7.1.2 Úspěšnost rozpoznání kontaktních údajů

Úspěšnost rozpoznávání kontaktních údajů byla měřena na vizitkách, které byly správně lokalizovány, tedy na vzorku 35 vizitek. Ne všechny vizitky obsahovaly všechny vyhledávané údaje. Všechny vizitky obsahovaly alespoň jedno telefonní číslo, ale jméno obsahovalo 31 vizitek a email 28 vizitek. Výsledky ověřované úspěšnosti jsou uvedeny v tabulce 2.

	Počet vizitek s kontaktním údajem	Počet úspěšně rozpoznaných vizitek	Úspěšnost [%]
Jméno	31	28	90,3%
Telefon	35	32	81,4%
Email	28	21	75%

Tabulka 2: Úspěšnost rozpoznávání kontaktních údajů

Nalezené jméno jsem bral jako úspěšné, pokud bylo správně rozpoznáno celé jméno i příjmení bez jakýchkoliv chyb. Pro tento kontaktní údaj bylo dosaženo nevyšší úspěšnosti, protože jméno je většinou hlavním a nejzřetelnějším údajem na vizitce, tudíž je na vizitce nejkvalitněji zobrazeno.

Při stanovování úspěšnosti rozpoznání telefonního čísla bylo potřeba zohlednit skutečnost, že většina vizitek obsahuje více telefonních čísel. Tuto úspěšnost jsem vyjádřil jako procentuální část rozpoznaných čísel ke všem telefonním číslům na vizitce. Pokud byly správně rozpoznány pouze dvě telefonní čísla ze čtyř, stanovil jsem výslednou úspěšnost jako 50% pro danou vizitku.
Rozpoznání emailu jsem bral jako úspěšné, pokud byla správně rozpoznána celé adresa bez jakýchkoliv chyb. Vyhledávání emailu dopadlo nejhůře, protože je nejvíce náchylné na kvalitu snímku vizitky. Také bývá napsaný menším písmem než jméno a telefon a má ze všech vyhledávaných údajů nejsložitější strukturu.

8 ZÁVĚR

Cílem této diplomové práce bylo vytvoření aplikace pro lokalizaci vizitek a importování kontaktů pro mobilní platformu Android. Po provedení průzkumu aplikací na Google Play jsem zjistil, že už existuje několik aplikací s podobnou funkcí, ale žádná neumožňuje vyhledávání vizitky v reálném čase. Všechny potřebují vyfotit statický snímek vizitky, což mi připadá zdlouhavé. Z tohoto důvodu jsem chtěl vytvořit aplikaci, která bude schopná lokalizovat vizitku v reálném čase přímo v náhledu z kamery, což se mi podařilo. Pro vytvoření této aplikace jsem zvolil prostředí Android Studio, které je jediným oficiálně podporovaným nástrojem pro vývoj Android aplikací.

První část diplomové práce je věnována popisu operačního systému Android. Je zde popsána historie, architektura a vývojové nástroje tohoto systému.

Druhá část práce je věnována popisu základních nástrojů počítačového vidění, které jsem využil při tvorbě aplikace. Také jsou zde popsány některé metody, které jsem se snažil využít, ale během tvorby aplikace jsem je musel zamítnout. Jednou z těchto metod byla metoda vyhledávání přímek v obraze pomocí Houghovy transformace, která se jevila jako vhodná při vývoji v C++, ale při převodu do Javy jsem ji musel zamítnout z důvodu vysoké výpočetní náročnosti.

V třetí části diplomové práce byly prozkoumány open-source knihovny pro zpracování obrazu. Pro nalezení vizitky a celkového zpracování obrazu jsem použil knihovnu OpenCV a pro rozpoznávání kontaktních údajů OCR knihovnu Tesseract.

Čtvrtá část diplomové práce se věnuje popisu samotné aplikace z uživatelského hlediska. Je zde popsána funkce jednotlivých ovládacích prvků spolu s ukázkou funkce aplikace. Také jsou zde popsány omezení a doporučení pro správné nalezení vizitky a rozpoznání kontaktních údajů.

V páté části je popsán vývoj aplikace. Během vývoje aplikace jsem zjistil, že Android Studio není moc vhodné pro vytváření a ladění algoritmů počítačového vidění. Proto jsem se rozhodl algoritmy pro vyhledávání a zpracování vizitky nejprve vytvořit v C++ verzi OpenCV ve Visual Studiu a tyto hotové algoritmy převést do Javy. Tento postup se ukázal jako správný, protože vývoj ve Visual Studiu postupoval výrazně rychleji než v Android Studiu.

Aplikace je rozdělena na dvě logické části. První část se stará o vyhledání vizitky v reálném čase a druhá o rozpoznání kontaktních údajů. První část je tedy náchylná na výpočetní výkon zařízení, což mě omezovalo při použití náročnějších algoritmů pro vyhledávání vizitky, které by vylepšily přesnost vyhledávání. Druhá část již neběží v reálném čase, a tudíž jsem si mohl dovolit aplikovat i výpočetně náročnější algoritmy. Celková rychlost rozpoznávání vizitky se v průměru pohybuje od 2s do 10s. Z toho přibližně 90% času zabere rozpoznávání znaků Tesseractem. Tato doba je poměrně dlouhá, ale bohužel se mi ji nijak nepodařilo snížit. Pro nalezení kontaktních údajů z textu

na vizitce jsem použil regulární výrazy, které se na tuto činnost osvědčily. Pro zvýšení úspěšnosti detekce jména na vizitce jsem doplnil regulární výrazy o ověření nalezeného řetězce pomocí databáze křestních jmen.
V závěrečné části jsem se věnoval testování úspěšnosti aplikace. Úspěšnost vyhledávání je silně závislá na kvalitě vizitky a na scéně, ve které je vyhledáváno. Proto bylo náročné stanovit úspěšnost tak, aby nebyla zavádějící. Nejprve jsem měřil úspěšnost lokalizování vizitky v obraze. Ze skupiny 40 testovacích vizitek bylo správně lokalizováno 35, což představuje úspěšnost 87,5%. Při stanovování úspěšnosti rozpoznání kontaktních údajů jsem dosáhl nejlepších výsledků pro rozpoznání jména, která dosahovala 90,3%. Pro vyhledávání telefonního čísla jsem dosáhl hodnoty 81,4% a pro vyhledávání emailu 75%.

LITERATURA

[1] Android (Operating system) In: Android (operating system) [online] 2014 [cit. 2015-12-22] Dostupné z: [https://en.wikipedia.org/wiki/Android_%28operating_system%29]

[2] MYSLIVEČEK, D., Krátké ohlédnutí za historií Androidu [online] 2013 [cit. 2015-12-22] Dostupné z: [http://www.svetandroida.cz/kratke-ohlednuti-za-historii-androidu-201305]

[3] KILIÁN, K., Historie Androidu v kostce aneb Od verze 1.0 až po Android M [online] 2015 [cit. 2015-12-22] Dostupné z: http://www.svetandroida.cz/historie-androidu-201506

[4] SHIJU P, John, Android Development Tutorial [online] 2015 [cit. 2015- 12-22] Dostupné z: http://www.eazytutz.com/android/

[5] Application Fundamentals, In: Android developer [online]. [cit. 2015-12- 22] Dostupné z: http://developer.android.com/guide/components/fundamentals.html

[6] Android - Architecture In: Tutorialspoint [online] [cit. 2015 12- 26] Dostupné z http://www.tutorialspoint.com/android/android_architecture.html

[7] Android - Broadcast Receivers In: Tutorialspoint [online] [cit. 2015 12-26] Dostupné z http://www.tutorialspoint.com/android/android_broadcast_receivers.htm

[8] Activity, In: Android developer [online]. 2007 [cit. 2015-12-28] Dostupné z: http://developer.android.com/ndk/guides/index.html

[9] OpenCV Reference Manual,[online]. 2004 [cit. 2015-12-29] Dostupné z: http://docs.opencv.org/2.4/modules/core/doc/intro.html

[10] Android NDK, In: Android developer [online]. [cit. 2016-01-03] Dostupné z: http://developer.android.com/ndk/guides/index.html

[11] Abby - ABBYY Mobile OCR Engine [online]. 2014. [cit. 2015-12-29] dostupné z: http://www.abbyy.com/mobileocr/

[12] SMITH, R., An Overview of the Tesseract OCR Engine [online] [cit. 2015-12-29] dostupné z: https://github.com/tesseractocr/docs/blob/master/tesseracticdar2007.pdf

[13] ŠONKA, M., V. HLAVÁČ, R. BOYLE: Image Processing, Analysis and Machine Vision, Thomson 2007

[14] HORÁK K., I. KALOVÁ, P. PETYOVSKÝ, M. RICHTER: Počítačové vidění,

Brno 2008

[15] About SQLite [online]. 2015-03-15 [cit. 2015-03-15]. dostupné z: http://www.sqlite.org/about.html

[16] HLAVÁČ V.: Jasové a geometrické transformace, [online], ČVUT Praha 2006. dostupné z: http://cmp.felk.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/18BrighGeomTxCz.pdf

[17] MUDROVÁ M.: Geometrické transformace obrazu a související témata, [online], 2004. dostupné z: http://uprt.vscht.cz/ucebnice/zob/prednasky/09-TRANSFORMACE/ transformace-tisk.pdf

SEZNAM ZKRATEK

SDK	Software Development Kit
NDK	Native Development Kit
ADT	Android Development Tools
OCR	Optical Character Recognition
API	Application Programming Interface
SQL	Structured Query Language
FPS	Frame per second
IDE	Integrated Development Environment
GUI	Graphical User Interface
C++	programovací jazyk
OpenCV	Open Source Computer Vision

PŘÍLOHY

Příloha 1 – ukázka rozpoznaných vizitek

Tesseract OCR výstup

PODLAHÁŘSTVÍ TRUHLÁŘSTVÍ
Návrh realizace
ZDENĚK MIKŠÍK
Svat. Čecha
731 556159
JOSEF MIKŠÍK ml.
688 01 Uh. Brod
731 556163
E-mail: Miksik.Truhlarstvi@seznam.cz

Nalezené kontaktní údaje

Jméno: ZDENĚK MIKŠÍK
JOSEF MIKŠÍK
Telefon:731 556159
731 556163
E-mail: Miksik.Truhlarstvi@seznam.cz

Tesseract OCR výstup

BESTPROJEKT Miroslav Vypušťák
jednatel společnosti
+420 606 048586
www.bestprolekt.cz
Tel ;+420 57! 898 085 e-malt: m
vypustak@beslpro;ekt.cz

Nalezené kontaktní údaje

Jméno: Miroslav Vypušťák
Telefon: +420 606 048586
E-mail:

Tesseract OCR výstup

Jiří Kryštof ml.
Autoservis EGP, s.r.o.
U Olšavy 2463
+420 725 873 692
Autoservisegp@seznamcz

Nalezené kontaktní údaje

Jméno: Jiří Kryštof
Telefon: +420 725 873 692
E-mail:

Tesseract OCR výstup

www.ski-spon.cz
E-mail: holasek@ski-spon.cz
688 01 Uh. Brod. U Nádraží 1765
Mobil: 00420 – 602 731 900
Fax: 00420 – 572 631 060
Tel.: 00420 – 572 631 160
HOPE SPORT, s.r.o.
Ing. Petr HOLÁSEK
SKI - SPORT

Nalezené kontaktní údaje

Jméno: Petr HOLÁSEK
HOPE SPORT
Telefon: 602 731 900
572 631 060
572 631 160
Email: holasek@ski-spon.cz

Tesseract OCR výstup	Nalezené kontaktní údaje
Roman Šašinka	Jméno: Roman Šašinka
realitní makléř	Telefon: +420 773 756 700
mobil: +420 773 756 700	E-mail: sasinka@rkcoloseum.cz
email: sasinka@rkcoloseum.cz	

Tesseract OCR výstup	Nalezené kontaktní údaje
(BADMINTON;; LÍŠEŘCZ	Jméno:
www.badminton\isen.cz	Telefon: +420 777 076 620
info@badmintonhsen.cz	E-mail: info@badmintonhsen.cz
+420 777 076 620	
areá\ Zetoru	
Úlehloíľavš-OŠČ-|-iig.	
6.670316	

Příloha 2 – ukázka nerozpoznaných vizitek