VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ SYSTÉM PRO ROZPOZNÁNÍ TEXTU NA ANDROIDU

MOBILE SYSTEM FOR TEXT RECOGNITION ON ANDROID

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JAN TOMEŠEK**
AUTHOR

VEDOUCÍ PRÁCE **prof. Dr. Ing. PAVEL ZEMČÍK**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/19683/2016/xtomes06

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Tomešek Jan, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Mobilní systém pro rozpoznání textu na Androidu**
Mobile System for Text Recognition on Android
Kategorie: Uživatelská rozhraní

Pokyny:

1. Prostudujte literaturu na téma předzpracování obrazu s textem a tvorbu mobilních aplikací.
2. Navrhněte postup pro implementaci knihovny pro předzpracování obrazu s textem na mobilní platformě Android.
3. Zhodnoťte možnosti a vlastnosti navrženého postupu a dosažitelné výsledky.
4. Implementujte knihovnu pro předzpracování obrazu a demonstrujte funkčnost na vhodném příkladu.
5. Diskutujte dosažené výsledky a možnosti dalšího pokračování práce.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zemčík Pavel, prof. Dr. Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá tvorbou mobilní knihovny pro předzpracování obrazu s textem, která představuje součást systému pro rozpoznávání textu. Knihovna je realizována s důrazem na obecnost použití, efektivitu a přenositelnost. V rámci práce byla vytvořena knihovna, která poskytuje řadu algoritmů především pro hodnocení kvality obrazu a detekci textu, jež umožňují výrazně snížit objem přenášených dat a zrychlit a zpřesnit proces rozpoznávání. Vytvořena byla také příkladová aplikace pro platformu Android, která dokáže analyzovat složení potravin uváděné na jejich obalech. Celkově tak knihovna (systém) zjednodušuje tvorbu mobilních aplikací se zaměřením na extrakci a analýzu textu. Mobilní aplikace pak poskytuje pohodlný způsob ověření škodlivosti potravin. Čtenáři práce nabízí přehled současných řešení i nástrojů dostupných v této oblasti, poskytuje rozbor významných algoritmů předzpracování obrazu a provádí jej budováním knihovny a aplikace pro mobilní zařízení.

Abstract

This thesis deals with creation of a mobile library for preprocessing of images with text which represents a part of a system for text recognition. The library is realized with emphasis on generality of use, efficiency and portability. The library providing a set of algorithms primarily for image quality assessment and text detection was created in this thesis. These algorithms enable a substantial decrease in volume of transmitted data and speed up and refinement of the recognition process. An example application for the Android platform able to analyze composition of foods stated on their wrappings was created as well. Overall, the library (system) simplifies development of mobile applications with focus on text extraction and analysis. The mobile application then provides a comfortable way of food harmfulness verification. The thesis offers a reader an overview of current solutions and tools available in this field, it provides a breakdown of important image preprocessing algorithms and guides him through the construction of the library and the application for mobile devices.

Klíčová slova

Předzpracování obrazu, rozpoznávání textu, mobilní knihovna, mobilní systém, mobilní zařízení, chytrý telefon, Android, mobilní aplikace, složení potravin

Keywords

Image preprocessing, text recognition, mobile library, mobile system, mobile device, smartphone, Android, mobile application, food composition

Citace

TOMEŠEK, Jan. *Mobilní systém pro rozpoznání textu na Androidu.* Brno, 2017. 67 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Zemčík Pavel.

Mobilní systém pro rozpoznání textu na Androidu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana prof. Dr. Ing. Pavla Zemčíka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Tomešek
16. května 2017

Poděkování

Děkuji svému vedoucímu, panu prof. Dr. Ing. Pavlu Zemčíkovi, za velkou vstřícnost, ochotu poradit a schopnost vést při počátečním formování i následné realizaci této práce. Děkuji také svému kolegovi Petru Bobákovi za spolupráci na tvorbě celkového řešení.

Obsah

Kapitola 1

Úvod

V této práci se zabývám tvorbou mobilní knihovny pro předzpracování obrazu s textem, která je významnou součástí vznikajícího systému pro rozpoznávání textu určeného pro chytré telefony. Tento systém si klade za cíl usnadnit vývoj mobilních aplikací využívajících fotoaparát pro nasnímání textu a analyzujících informaci v textu obsaženou. Systém, a tudíž i knihovna, by měl být co nejobecnější, ať už s ohledem na konkrétní uplatnění či z pohledu množství zařízení, která jej dokáží využít.

Motivací ke vzniku tohoto systému je nápad na vytvoření mobilní aplikace pro rychlou analýzu složení uvedeného na obalech potravin, která by uživatele oprostila od čtení titěrného textu či porozumění významu jednotlivých složek. Snadno si však lze představit řadu dalších aplikací využívajících podobnou technologii pro řešení jiných lidských problémů. Právě z tohoto důvodu se práce nejprve zaměřuje na vytvoření obecně využitelné, multiplatformní knihovny, která by mohla být jádrem všech obdobných aplikací. Nad touto knihovnou je následně vybudována zmíněná mobilní aplikace.

Z osobního hlediska mi má práce pomoci proniknout do oblasti zpracování obrazu a zdokonalit se ve vývoji mobilních aplikací.

Jelikož je kompletní proces rozpoznávání textu stále výpočetně náročný a tudíž nepříliš vhodný pro běh na mobilních zařízeních, soustředí se mobilní knihovna na postup předcházející samotnému rozpoznávání textu v obraze. Jedná se předzpracování obrazu ve smyslu vylepšení jeho vlastností, lokalizace textu a zmenšení objemu dat.

Samotné rozpoznání textu pak proběhne na serveru, který bude tvořit druhou neodmyslitelnou součást systému, přičemž tento celek by měl dále sloužit i pro využití ostatními vývojáři. Tvorbou tohoto serveru se zabývá souběžně vznikající práce kolegy Bc. Petra Bobáka, se kterým na tvorbě celého systému úzce spolupracuji.

V následujících kapitolách se budu postupně zanořovat do problematiky zpracování obrazu s textem až k samotným obrazovým algoritmům, odkud se budu následně v opačném směru vynořovat a budovat mobilní knihovnu i příkladovou aplikaci.

Nejdříve provedu průzkum existujících mobilních aplikací se zaměřením na rozpoznávání textu (2.1). Následně prozkoumám systémy alternativní k tomu našemu (2.2), které mohou být v takových aplikacích použity. Dále analyzuji dostupné knihovny počítačového vidění (2.3), které lze pro realizaci podobného systému využít. Popíši také významné techniky a algoritmy (3), které jsou používané pro předzpracování obrazu. Na závěr se budu věnovat vývoji aplikací pro platformu Android (4).

S takto nabytými znalostmi navrhnu architekturu knihovny (6.2) včetně potřebných algoritmů a na nich knihovnu vystavím (6.3). Pro demonstraci využití této knihovny (a celého systému) pak navrhnu (6.5) a realizuji aplikaci pro rozpoznávání složení potravin (6.6) pro operační systém Android. Knihovnu i aplikaci na závěr důkladně otestuji (7).

Diplomová práce plně navazuje na předcházející semestrální projekt. Převzata tedy byla celá kapitola 2 a část podkapitoly 3.1. Z pohledu realizace byl převzat základ mobilní knihovny i kostra mobilní aplikace, jejichž popis je součástí kapitoly 6.

Kapitola 2

Aplikace, systémy a knihovny pro zpracování obrazu a rozpoznávání textu

Pro uvedení do kontextu jsou v této kapitole čtenáři předložena existující alternativní řešení k tomu, které nabízí tato práce, a to v podobě mobilních aplikací a jejich vnitřních systémů pro rozpoznávání textu. S ohledem na pozdější tvorbu vlastní knihovny (části vlastního systému) jsou dále uvedeny knihovny zpracování obrazu, které lze pro tento účel využít.

2.1 Existující mobilní aplikace pro rozpoznávání textu

Průzkum mobilních aplikací vedoucí ke vzniku této podkapitoly je realizován se třemi hlavními záměry. Nutno poznamenat, že mobilní aplikace samozřejmě nejsou ekvivalentem systému pro předzpracování obrazu a rozpoznávání textu, který je hlavní náplní této práce, nicméně jsou s takovými systémy často úzce spojeny a napovídají, jak by odpovídající systém měl pracovat.

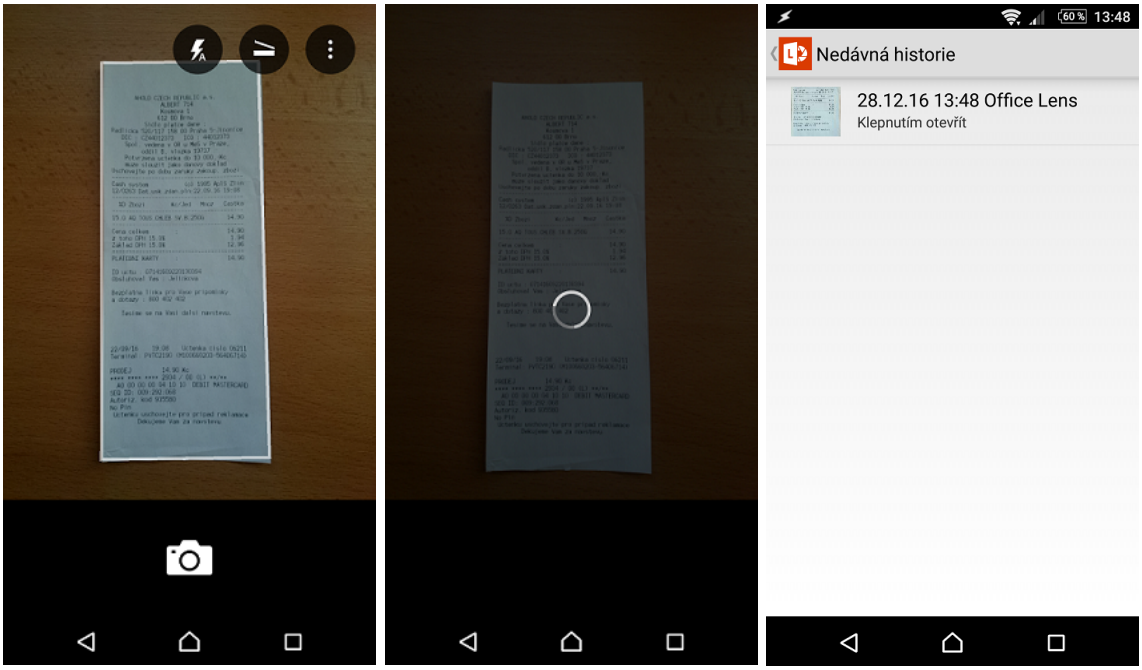
Prvním záměrem je samotné poznání, jaké aplikace pro rozpoznávání již na trhu existují a fungují, a tedy zorientování se v současném stavu. Druhým je analýza funkcí a celkově přístupů, které tyto aplikace v rámci předzpracování a rozpoznávání textu používají, a tedy určení, co všechno by měla vlastní knihovna (část systému) nabízet, aby takovým aplikacím umožnila vše potřebné. Jelikož nezanedbatelnou částí práce je i příkladová aplikace pro rozpoznávání složení potravin, je třetím záměrem nalézt inspiraci i poučení, jak realizovat mobilní aplikaci, jejíž jádro tvoří rozpoznávání textu.

V rámci udržení kompaktnosti textu, dosažení lepší přehlednosti a umožnění jednoduchého srovnání jsou nejdůležitější sledované charakteristiky existujících aplikací zaznamenány do tabulek jednotného tvaru. Tyto charakteristiky by se pomyslně daly rozdělit do tří kategorií. První kategorie souvisí s obecným uživatelským zážitkem z aplikace a patří do ní výchozí obrazovka po spuštění nebo pohodlí práce s fotoaparátem. Druhá kategorie sleduje uživatelský zážitek v průběhu rozpoznávání, jenž může být vylepšen například automatickými inteligentními funkcemi či detailní zpětnou vazbou o aktuálním stavu. Třetí kategorie pak sleduje technické aspekty, mezi které patří doba a přesnost rozpoznávání a architektura celé aplikace.

Na závěr je uveden také příklad aplikace se zaměřením na složení potravin. Všechny popsané aplikace jsou dostupné přinejmenším pro platformu Android.

2.1.1 Office Lens

Aplikace se zaměřuje na skenování dokumentů a poskytuje kvalitní automatickou detekci, zarovnání i ořezání dokumentů. Je vydávána společností Microsoft Corporation. Výstupem rozpoznávání jsou soubory pro programy spadající do balíku Microsoft Office, přičemž je znázorněna i poloha slov v pořízeném snímku.

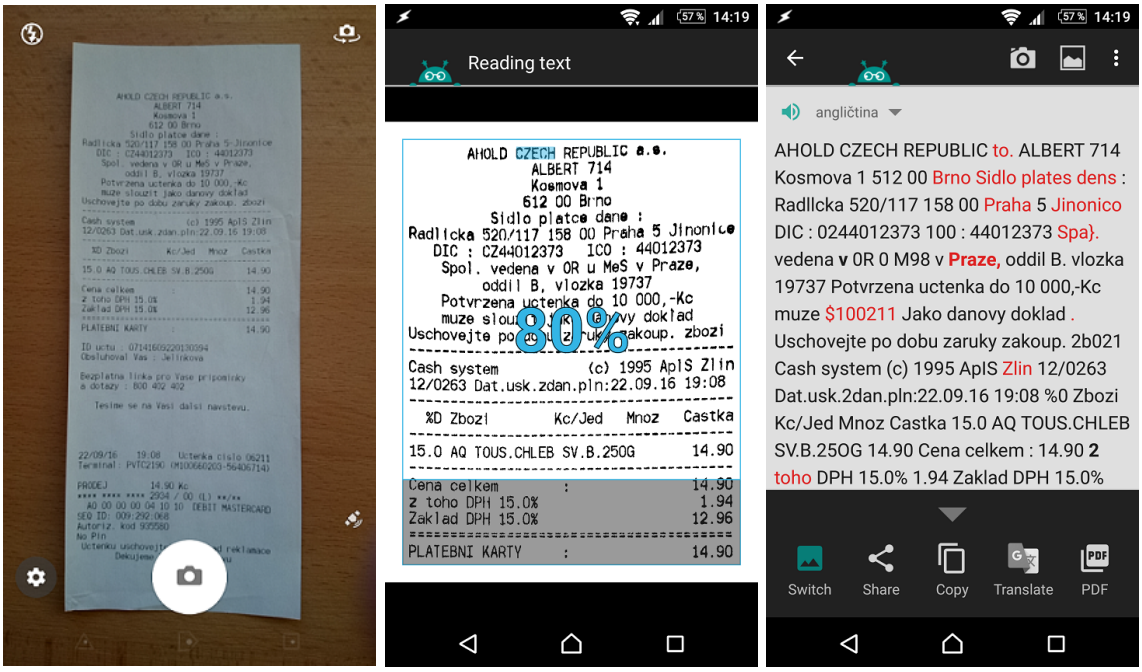


Obrázek 2.1: Aplikace Office Lens

výchozí obrazovka	náhled fotoaparátu
obrazovka fotoaparátu	vlastní (režim na výšku i na šířku)
výběr rozpoznávané oblasti	automatické orámování detekovaného objektu
průběh zpracování	indikátor průběhu nad ztmaveným snímkem
orientační doba rozpoznávání	10 s
orientační přesnost rozpoznávání	mírný až střední počet chyb
architektura aplikace	pouze klient

2.1.2 Text Fairy (OCR Text Scanner)

Aplikace autora Renarda Wellnitze sice přímo deklaruje, že nedisponuje pokročilými funkcemi, jako je překlad či rozpoznávání ručně psaného textu, nicméně běžné rozpoznávání zvládá velmi přesně a rychle, navíc bez nutnosti připojení k internetu. V základu dokáže rozpoznávat pouze ve třech jazycích, lze ovšem doinstalovat další včetně češtiny.



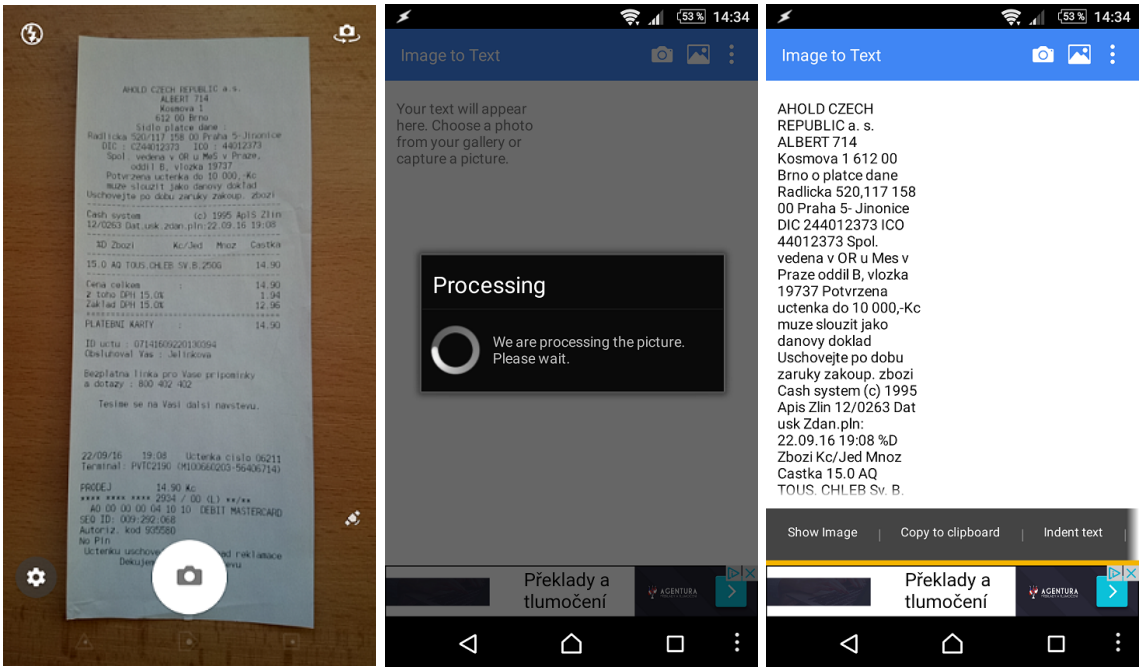
Obrázek 2.2: Aplikace Text Fairy

výchozí obrazovka	historie dokumentů
obrazovka fotoaparátu	systémová aplikace (režimy dle ní)
výběr rozpoznávané oblasti	manuální orámování oblasti
průběh zpracování	vizualizace úprav a stavu zpracování snímku
orientační doba rozpoznávání	10 s
orientační přesnost rozpoznávání	velmi přesné
architektura aplikace	pouze klient

2.1.3 Image to Text (OCR Scanner)

Studio HMA Labs vytvořilo aplikaci s poměrně přímočarým použitím. Aplikace provádí vždy pouze jednorázové rozpoznávání bez možnosti uchování výsledku. Neumožňuje ani vymezení oblasti snímku pro rozpoznávání. Díky zpracování na serveru ovšem nabízí kvalitní výsledky, navíc v poměrně krátkém čase.

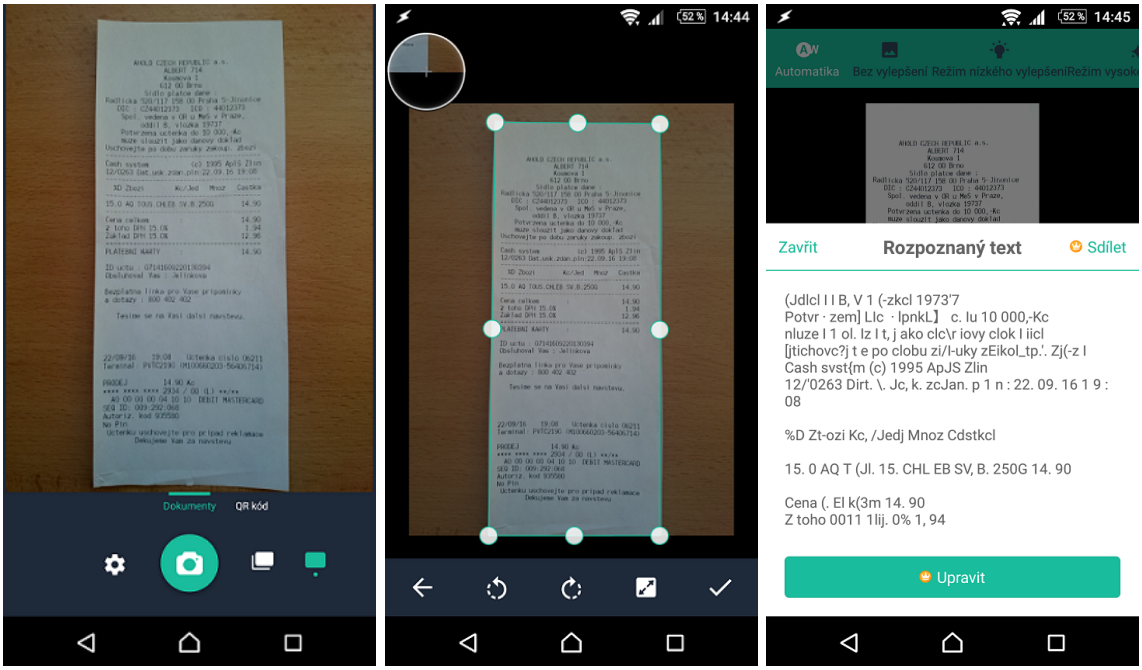
výchozí obrazovka	obrazovka připravená pro rozpoznaný text
obrazovka fotoaparátu	systémová aplikace (režimy dle ní)
výběr rozpoznávané oblasti	nelze
průběh zpracování	indikátor průběhu nad ztmavenou obrazovkou
orientační doba rozpoznávání	10 s
orientační přesnost rozpoznávání	velmi přesné
architektura aplikace	klient-server



Obrázek 2.3: Aplikace Image to Text

2.1.4 CamScanner - Phone PDF Creator

Aplikace sází nejen na bohaté možnosti úprav snímků, ale také na správu a organizaci dokumentů. Společnost INTSIG Information Co., Ltd. se zaměřila i na moderní, esteticky zpracované rozhraní. Rychlému rozpoznání bohužel předchází absolvování mnoha uživatelských kroků a výsledky nejsou příliš přesvědčivé.

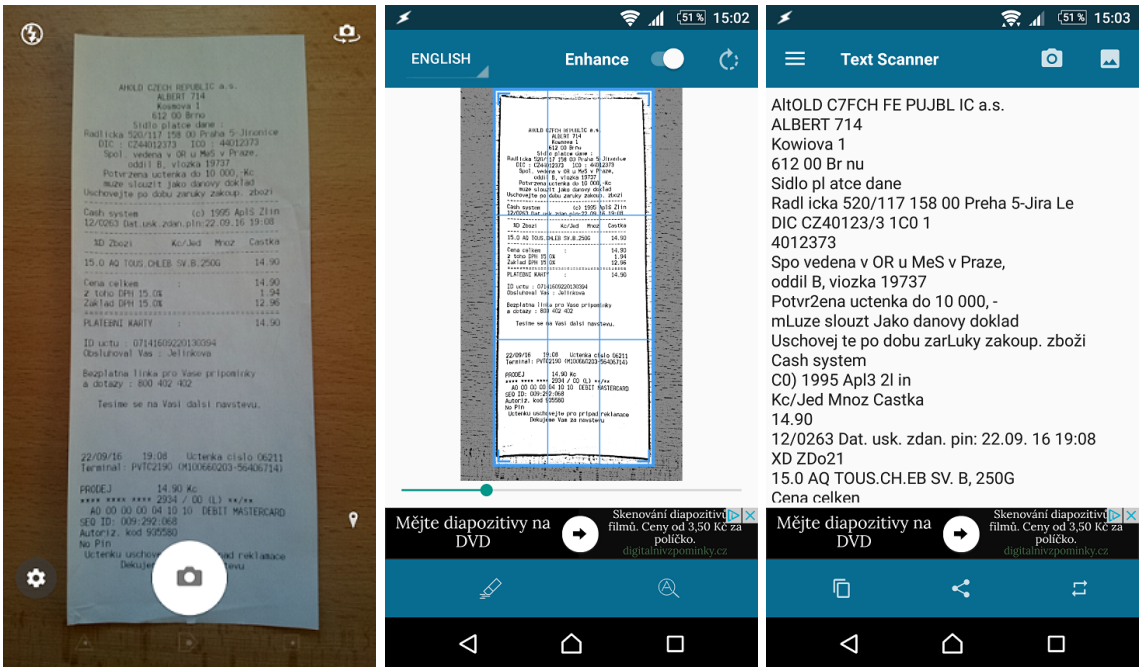


Obrázek 2.4: Aplikace CamScanner

výchozí obrazovka	seznam dokumentů
obrazovka fotoaparátu	vlastní (režim na výšku i na šířku)
výběr rozpoznávané oblasti	manuální orámování (přichytávání k objektu)
průběh zpracování	vizualizace úprav a stavu zpracování snímku
orientační doba rozpoznávání	5 s
orientační přesnost rozpoznávání	mnoho chyb
architektura aplikace	pouze klient

2.1.5 OCR - Text Scanner

Tato aplikace je dalším zástupcem přímočarého přístupu s jednorázovým rozpoznáváním bez možnosti uložení. K rozpoznávání se ve studiu Rishi Apps rozhodli použít systém Tesseract (viz. podkapitola 2.2.1), který v tomto případě běží přímo na mobilním zařízení. Rozpoznávání probíhá velmi rychle a lze jej rozšířit o další jazyky.

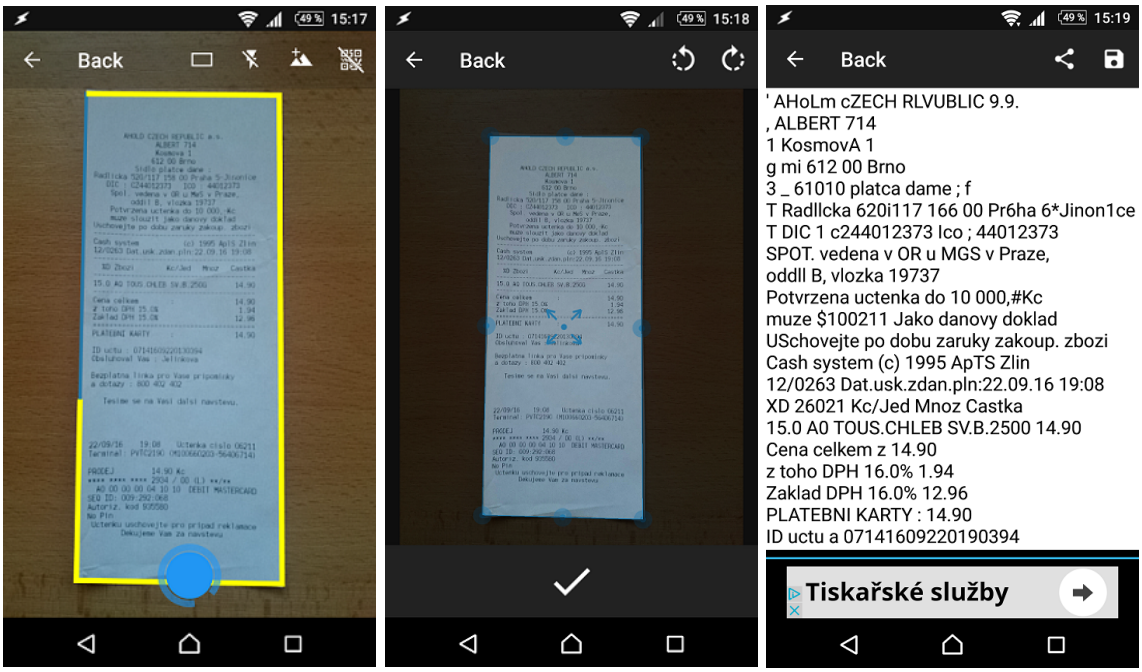


Obrázek 2.5: Aplikace OCR - Text Scanner

výchozí obrazovka	obrazovka připravená pro rozpoznaný text
obrazovka fotoaparátu	systémová aplikace (režimy dle ní)
výběr rozpoznávané oblasti	manuální orámování oblasti (mřížka)
průběh zpracování	indikátor průběhu nad ztmaveným snímkem
orientační doba rozpoznávání	3 s
orientační přesnost rozpoznávání	poměrně přesné
architektura aplikace	pouze klient (Tesseract)

2.1.6 PDF Scanner - OCRDocument Scanner

Aplikace od vývojáře GRYMALA urychluje kroky prováděné před samotným rozpoznáváním díky automatické detekci dokumentu a především automatickému pořízení snímku. Na druhou stranu rozpoznávání již k nejrychlejším nepatří a produkuje mnoho chyb.

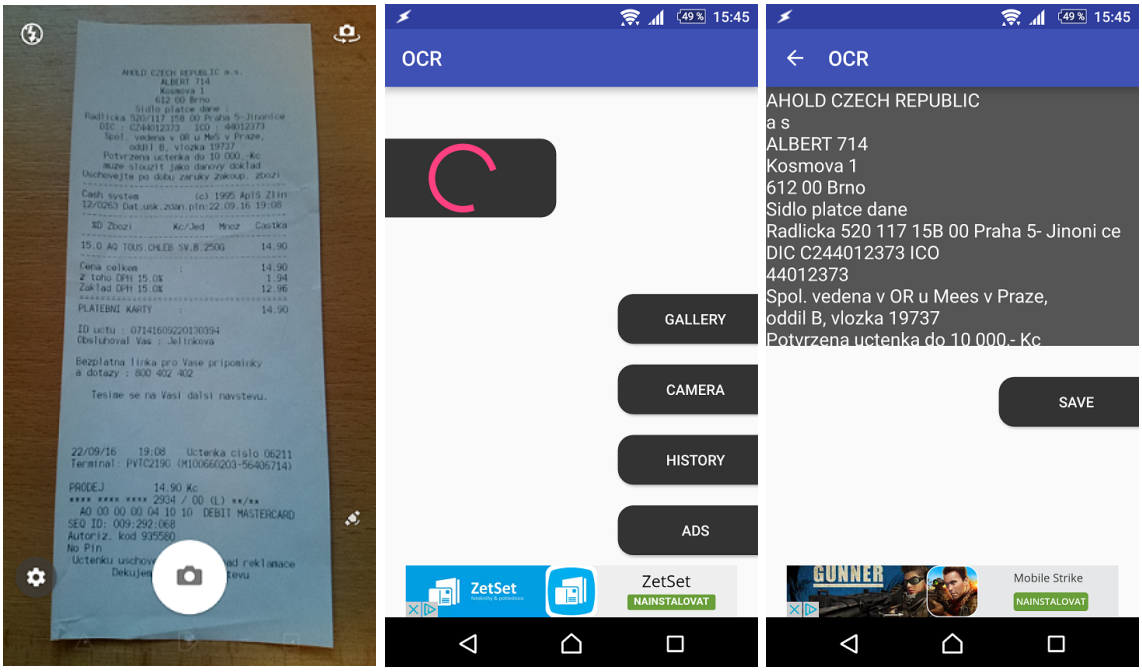


Obrázek 2.6: Aplikace PDF Scanner

výchozí obrazovka	náhled fotoaparátu
obrazovka fotoaparátu	vlastní (pouze režim na výšku)
výběr rozpoznávané oblasti	automatické orámování detekovaného objektu
průběh zpracování	vizualizace úprav a stavu zpracování snímku
orientační doba rozpoznávání	15 s
orientační přesnost rozpoznávání	mnoho chyb
architektura aplikace	pouze klient

2.1.7 OCR - converts characters photo

Vývojář shinozaki gen vytvořil poměrně „surovou“ aplikaci, která slouží spíše jako demonstrace schopností rozpoznávání textu. Využívá systémovou aplikaci fotoaparátu a neumožňuje rozpoznávání jakkoliv ovlivnit. Zajímavá je použitím systému Cloud Vision (viz. podkapitola 2.2.3) pro rozpoznávání na serveru, který produkuje velmi přesné výsledky.



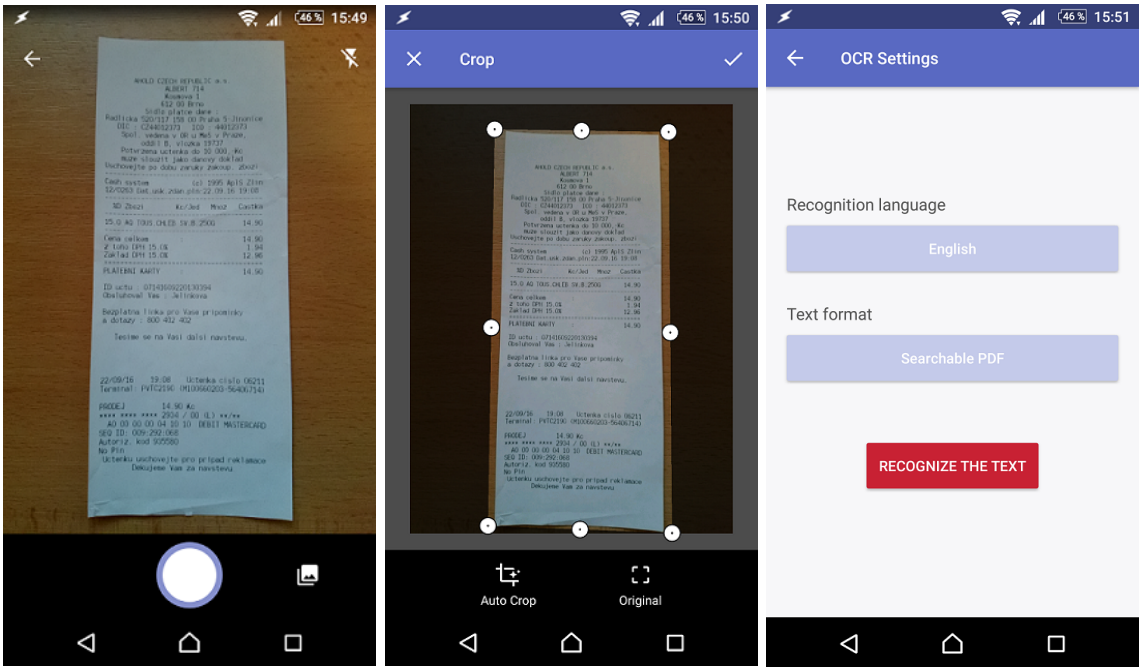
Obrázek 2.7: Aplikace OCR - converts characters photo

výchozí obrazovka	menu
obrazovka fotoaparátu	systémová aplikace (režimy dle ní)
výběr rozpoznávané oblasti	nelze
průběh zpracování	indikátor průběhu
orientační doba rozpoznávání	15 s
orientační přesnost rozpoznávání	velmi přesné
architektura aplikace	klient-server (Cloud Vision)

2.1.8 FineScanner - docs recognition

FineScanner je zařazením profesionální placená aplikace, kterou vydává společnost ABBYY. Rozpoznávání je prováděno na serveru vlastní technologií, kterou ABBYY nabízí i vývojářům třetích stran (viz. podkapitola 2.2.4). Trvá poměrně dlouho, nicméně výsledky tomu odpovídají. Výstup může být exportován do mnoha formátů včetně pdf.

výchozí obrazovka	seznam dokumentů
obrazovka fotoaparátu	vlastní (režim na výšku i na šířku)
výběr rozpoznávané oblasti	manuální/automatické orámování detek. objektu
průběh zpracování	indikátor průběhu
orientační doba rozpoznávání	45 s
orientační přesnost rozpoznávání	velmi přesné
architektura aplikace	klient-server



Obrázek 2.8: Aplikace FineScanner

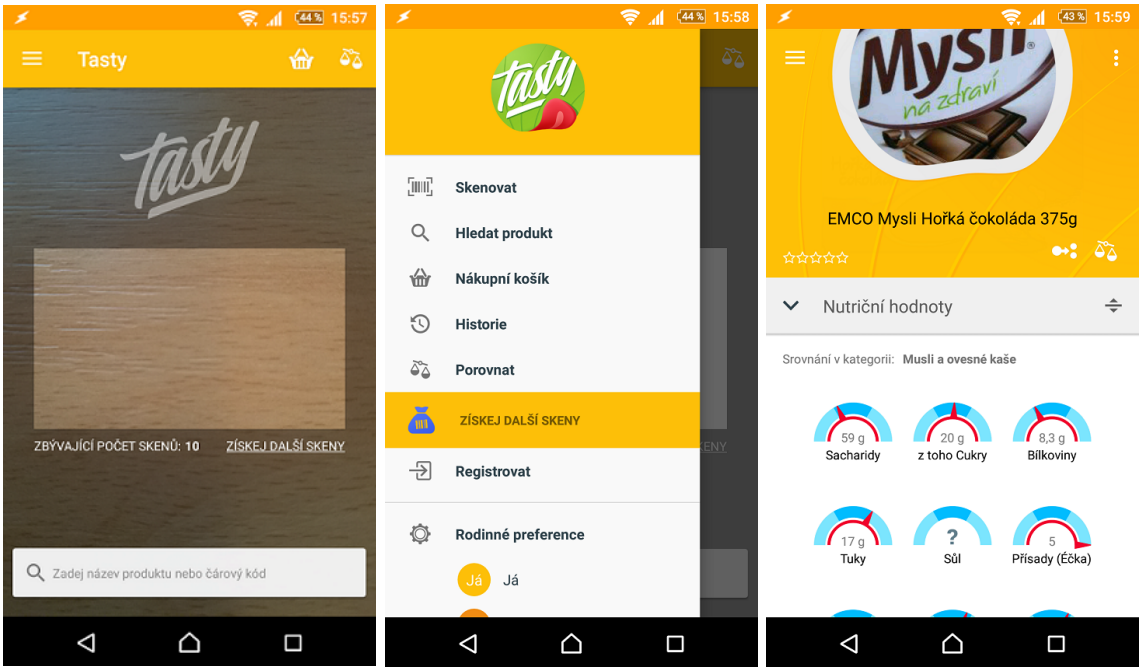
2.1.9 Tasty Food Scanner

Tasty Food Scanner není aplikací pro rozpoznávání textu, jako tomu bylo v předcházejících případech. Jedná se o zástupce aplikací soustředících se na složení potravin. Funguje na stejném principu jako drtivá většina aplikací se stejným zaměřením, je z nich ovšem pravděpodobně nejpropracovanější, a proto představuje vhodný příklad. Za jejím vznikem stojí studio Master App Solutions.

Zjištění složení pomocí rozpoznávání textu sice aplikace nahrazuje skenováním čárových kódů a budováním databáze produktů, některé další části aplikace by ovšem mohly zaujmout místo i v příkladové aplikaci budované v rámci této práce.

Aplikace umožňuje vyhledávání produktů, u kterých lze nalézt fotografii, výrobce či celkové hodnocení. Mezi detailnější informace patří nutriční hodnoty vyjádřené přehlednou infografikou, složení, obsažené alergeny a doba potřebná pro spálení souvisejících kalorií. K produktům lze také přidávat uživatelské komentáře. Dále aplikace nabízí srovnání produktů mezi sebou, tvorbu nákupních seznamů a nastavení osobních preferencí či alergií, které jsou zohledněny při výběru produktů.

Dle komentářů uživatelů ovšem trpí na nedostatek položek v databázi, což je často kritizovaným neduhem i dalších podobných aplikací.



Obrázek 2.9: Aplikace Tasty Food Scanner

2.2 Existující mobilní systémy pro rozpoznávání textu

Pro vytvoření aplikace obdobné těm z předcházející podkapitoly je na místě využít mobilní systém pro rozpoznávání textu. Protože se tvorbou (části) takového systému zabývá právě tato práce, jsou v této podkapitole pro lepší představu o problematice představeny již existující alternativy.

2.2.1 Tesseract

Systém Tesseract[1] je otevřený OCR[2] engine[3] schopný extrahovat tištěný i ručně psaný text z obrázků. Tesseract byl původně vyvinut společnostmi Hewlett-Packard Laboratories a Hewlett-Packard Co v letech 1985 až 1994. Dalšími úpravami prošel v roce 1996, kdy byl naportován na Windows, a v roce 1998, kdy byl zpřístupněn z jazyka C++. V roce 2005 z něj HP udělalo open source[4] projekt a od roku 2006 je vyvíjen společností Google. Aktuálně je pod licencí *Apache License, Version 2.0* a jeho poslední verze 3.04.01 byla vydána v únoru 2016.

Tesseract je dostupný buď přímo pomocí programu `tesseract` pro příkazovou řádku nebo skrze aplikační rozhraní (API) knihovny libtesseract. Vývojáři aplikací pak mohou pro práci s API využít jazyky C a C++. Průběžně však vznikají wrappery[5] pro další jazyky, jako je Java či Python. Je použitelný na všech významných platformách, mezi které patří Windows, Linux či Mac OS X, ale také Android a iOS.

[1] https://github.com/tesseract-ocr
[2] Optical Character Recognition – optické rozpoznávání znaků (textu)
[3] jádro pohánějící počítačový program
[4] otevřený (volně přístupný) zdrojový kód
[5] obálky, obalující soubory

Podporuje kódování unicode (UTF-8) a v základu je schopen rozpoznat text ve více než 100 jazycích včetně češtiny. Může být ale natrénován pro rozpoznávání v dalších jazycích. Pro práci s ním jsou potřeba dvě komponenty. Samotný engine a trénovací data pro vybraný jazyk. Výstupem rozpoznávání může být plain-text[6], html či pdf. Autoři Tesseractu sami upozorňují, že pro dosažení lepších výsledků rozpoznávání je potřeba obrázky předávané do Tesseractu vylepšit předzpracováním. [31]

Na operačním systému Android lze s Tesseractem pracovat pomocí *Tesseract Tools for Android*[7], což je sada API pro samotný rozpoznávač Tesseract a také knihovnu Leptonica pro zpracování obrazu. Alternativou je odvozený projekt *tess-two*[8], který přidává dodatečnou funkcionalitu. Oba projekty jsou pod licencí *Apache License, Version 2.0*.

Vyplývající výhodou Tesseractu je potom možnost spuštění kompletního rozpoznávání textu přímo na mobilním zařízení bez nutnosti připojení k internetu. Příkladem může být demonstrační Android aplikace OCR Demo pro rozpoznávání čísel využívající *tess-two* a pracující offline.

2.2.2 Google Mobile Vision

Mobile Vision API představuje framework[9] pro nalezení objektů ve fotografiích a videu, které probíhá v reálném čase a přímo na zařízení. Využívá tzv. detektory, které lokalizují a popisují objekty v obrázcích či videu, a událostmi řízené API, které ve videu sleduje pohyb objektů.

V současné době toto API obsahuje detektory pro tvář, čárové a QR kódy a text, které lze využívat odděleně nebo dohromady. API jsou ve výsledku dělena na API se společnou funkcionalitou a další tři API pro zmíněné detektory. Mobile Vision lze využít na Androidu i iOS. Na Androidu funguje od verze 4.1. Na iOS je však zatím dostupný pouze detektor tváří. API pro text dokáže rozpoznat jakýkoliv jazyk založený na latince. Rozpoznávač segmentuje text do bloků, řádků a slov a následně je schopen text takovou strukturou i reprezentovat. [7]

Na druhou stranu nepodporuje automatickou detekci jazyka.

Před prvním použitím vyžaduje počáteční inicializaci, kdy se při instalaci aplikace automaticky stáhnou všechna potřebná data. Následně jej lze používat offline. Objekt třídy `TextRecognizer` dokáže zpracovat snímek a určit, jestli a jaký se v něm nachází text. Toto zvládne i v reálném čase z náhledu kamery. Pro náhled je určen objekt třídy `CameraSource`, který se stará o kameru a je přednastaven pro počítačové vidění. Objekt třídy `Processor` umožňuje provádět zpracování kontinuálně tak, jak přicházejí jednotlivé bloky textu z objektu `TextRecognizer`. Velmi zajímavou možností je pak využití objektu třídy `OcrGraphic`, který může být vykreslen do náhledu. V kombinaci s interní reprezentací struktury textu pak lze například rozpoznaným (a libovolně upraveným) textem přímo v náhledu překrýt jeho originální podobu.

[6] prostý (neformátovaný) text
[7] https://github.com/alanv/tesseract-android-tools
[8] https://github.com/rmtheis/tess-two
[9] softwarová knihovna (kostra), na které lze vystavět programy

2.2.3 Google Cloud Vision

Google Cloud Vision API je součástí všestranné Google Cloud Platform, což je platforma poskytující cloudové[10] produkty a služby. Ty přímo využívají infrastrukturu společnosti Google, přičemž důraz je kladen na škálovatelnost a spolehlivost s cílem poskytnout řešení i na podnikové úrovni. Jedná se o desítky služeb zaměřených na náročné výpočty, úložiště a databáze, síťovou infrastrukturu, analýzu velkého množství dat, správu a monitorování, bezpečnost a v neposlední řadě na strojové učení.

Právě do kategorie strojového učení spadá i Cloud Vision API, které umožňuje interpretovat obsah obrázků. Zapouzdřuje různé modely strojového učení a zpřístupňuje je skrze REST API.

Dokáže klasifikovat obrázky do tisíců kategorií, detekovat objekty a tváře a rozpoznávat slova v obrázcích. Pro rozpoznání textu nabízí kromě samotné technologie OCR pro detekci a extrakci i automatickou identifikaci jazyka umožněnou na základě široké podpory světových jazyků.

Požadavky jsou posílány protokolem HTTP metodou POST a požadavky i odpovědi jsou přenášeny ve formátu JSON. Pro práci lze využít programovací jazyky, jako jsou například Java, PHP či Python. Ke Cloud Vision API lze přistupovat také z aplikací pro operační systémy Android a iOS.

Výhodou takového cloudového řešení může být zlepšování s postupem času jak pomocí strojového učení, tak vylepšováním služeb samotným Googlem. Nevýhodou vyplývající z měřítka a profesionality tohoto řešení jsou pak měsíční poplatky rostoucí s každou tisícovkou odeslaných požadavků. [4]

2.2.4 ABBYY Cloud OCR SDK

ABBYY Cloud OCR SDK[11] je systém pro rozpoznávání textu umístěný v cloudu a přístupný přes webové aplikační rozhraní. Jedná se o řešení původně určené pro velké podniky a obrazové specialisty, které deklaruje přesnost až 99,8 %. Systém je škálovatelný, jelikož je založen na platformě Microsoft Azure, a lze s ním pracovat v programovacích jazycích jako třeba Java, C++, Python či PHP.

Díky umístění v cloudu a přístupu přes webové rozhraní je platformě nezávislý a nevytváří žádné systémové požadavky. Stačí pouze internetové připojení. Systém je postaven na principech RESTu a lze k němu přistupovat pomocí HTTP a HTTPS požadavků.

Je natrénován pro rozpoznávání textu ve 198 jazycích a nabízí několik unikátních funkcí, například rozpoznávání zaškrtávacích políček pro zpracování formulářů nebo rozpoznávání pouze vybraných textových polí.

Je vhodný také pro využití z mobilních zařízení, jelikož dokáže pracovat s obrázky s nižší kvalitou (špatné osvětlení, šum), podporuje mnoho mobilních platforem včetně Androidu a iOS a nevyžaduje mnoho výpočetního výkonu.

Umožňuje provádět vylepšení snímků pořízených z mobilních zařízení, jako například odstranění zkreslení, perspektivní korekci či úpravu rozlišení, a nabízí automatickou detekci orientace stránky.

Použití je placené pomocí balíčků či měsíčního předplatného a cena se odvíjí především od počtu rozpoznaných listů A4. [1]

[10] internetová služba dostupná odkudkoliv

[11] Software Development Kit - sada nástrojů pro vývoj software

Výhodou tohoto systému může být fakt, že se jedná rozsáhlé řešení, které se však zaměřuje čistě na rozpoznávání textu, což může vést k velmi kvalitním výsledkům. Nevýhodou je pak opět zpoplatnění, které reflektuje zaměření na podnikové použití.

2.2.5 ABBYY Mobile OCR Engine

Společnost ABBYY nabízí kromě uvedeného cloudového řešení také systém pro rozpoznávání textu pracující přímo na mobilním zařízení. Umožňuje převod obrázků na prohledávatelné a editovatelné dokumenty a podporuje významné mobilní platformy v čele s Androidem a iOS.

Systém pracuje ve čtyřech krocích. Nejprve načte obrázek a předzpracuje jej. Pak provádí analýzu a detekuje písmena, která spojuje do slov a ty do řádků a bloků. Jako třetí probíhá samotné OCR s využitím rozpoznávání vzorů a slovníků. Na závěr produkuje výsledek, nad kterým má vývojář plnou kontrolu.

Systém slibuje vysokou přesnost, jelikož je založen na stejné technologii jako cloudové řešení. Rozpoznávat text pak dokáže v 62 jazycích. Je také optimalizován pro nízkou spotřebu zdrojů včetně paměti.

Nabízí automatickou opravu zkreslení obrazu, detekci orientace dokumentu a spojování slov původně rozdělených na konci řádku. Dále indikátor informující, jak moc si je systém jistý rozpoznaným textem, kontrolu hláskování a zrychlený algoritmus binarizace obrazu.

Zaměřuje se na nízkou spotřebu zdrojů díky správě paměti, která přesně odhaduje potřebné místo pro zpracování obrazu, a díky celkově malé velikosti systému, který zabírá zhruba 8 MB paměti ROM a 10 MB paměti RAM. Navíc využívá paralelismu, kdy rozpoznávání probíhá defaultně ve čtyřech vláknech.

Jeho analytické algoritmy umožňují zachování struktury a formátování původního dokumentu, včetně vícesloupcových dokumentů, a také zachování fontů písma.

Systém může pracovat ve dvou režimech. Rychlý režim se zkráceným časem zpracování a rozpoznávání je vhodný, pokud je k dispozici obrázek v dobré kvalitě. Plnohodnotný režim si pak poradí i s obrázky v kvalitě nízké.

Systém je konkrétně dostupný pro Android od verze 2.2 a iOS od verze 7.1. Stejně jako v předcházejícím případě se potom jedná o placený systém. [2]

2.3 Dostupné knihovny pro zpracování obrazu

Systém pro rozpoznávání textu je smysluplné vybudovat s využitím některé z knihoven pro zpracování obrazu a počítačové vidění. S ohledem na tvorbu vlastního systému tato podkapitola popisuje právě takové knihovny.

2.3.1 OpenCV

OpenCV (plným názvem Open Source Computer Vision) je velmi rozšířená multiplatformní knihovna zaměřená na zpracování obrazu, počítačové vidění i strojové učení. Je vydávána pod *BSD licencí*, a je tedy volně dostupná pro akademické i komerční využití. Obecně s ní lze pracovat v jazycích C++, C, Java a Python a podporuje platformy Windows, Linux i MacOS doplněné o Android a iOS.

Je navržena pro výpočetní efektivitu se silným zaměřením na aplikace pracující v reálném čase. Knihovna dokáže využít vícejádrové procesory a díky spolupráci s OpenCL a CUDA i hardwarovou akceleraci konkrétní platformy. Knihovna je napsána nativně – původně v jazyce C, nové algoritmy jsou již psány v jazyce C++.

Obsahuje přes 2500 optimalizovaných algoritmů s využitím pro detekci a rozpoznávání tváří, identifikaci objektů, sledování pohybu kamery či pohybujících se objektů, vytváření 3D modelů z objektů, spojování obrázků pro dosažení vyššího rozlišení, hledání podobných obrázků, rozpoznání scény, rozmístění značek pro rozšířenou realitu apod. Knihovnu využívají nejen velké společnosti jako Google, Intel či Sony, ale také malé startupy.

Varianta knihovny pro platformu Android se nazývá OpenCV4Android. Důležitou vlastností je, že lze znovupoužít C++ kód z plnohodnotných počítačů. Sami vývojáři dokonce doporučují vyvíjet a ladit algoritmy na stolních počítačích v odpovídajících vývojových prostředích.

Spolu s knihovnou pro Android je dodávána i sada ukázkových příkladů a dokumentace API pro použití z jazyka Java. Využití je však možné i nativně z C++. Použití z C++ je jakožto náročnější varianta doporučeno pro profesionální vývojáře, kterým ale výměnou nabídne více možností a vyšší výkon.

S knihovnou OpenCV4Android je tedy možné pracovat na dvou úrovních. Základní úroveň odpovídá využití pouze Java API. Skrze Javu je možné přistupovat k většině funkcionality OpenCV, včetně kamery, takže není vůbec potřeba přecházet na nativní úroveň. Naopak všechny výpočty na nativní úrovni prováděny jsou, a overhead[12] tedy odpovídá ceně souvisejících volání skrze rozhraní JNI (Java Native Interface). Jedná se o nejrychlejší a nejjednodušší způsob vývoje pro Android s využitím OpenCV, navíc s automatickým uvolňováním paměti. Složitější aplikace s mnoha OpenCV voláními však budou pomalejší kvůli ceně dodatečných JNI volání. Java API navíc neobsahuje vše, co nabízí C++ API.

Pokročilá úroveň pak odpovídá využití nativního rozhraní. To se většinou používá pro profesionální vývoj. Android umožňuje volání nativních funkcí, což znamená, že lze využít původní C++ rozhraní. Pokud je využíváno mnoho algoritmů zpracování, začne výše zmíněné využití Java API konzumovat příliš mnoho času. V takovém případě je lepší zapouzdřit potřebnou funkcionalitu do jediné C++ třídy a tu volat pouze jedenkrát pro každý snímek. Využití nativního rozhraní navíc umožňuje vývoj a ladění na plnohodnotném počítači. Tento přístup přináší maximální možný výkon, 100% dostupnost OpenCV funkcí, možnost přenášet kód na ostatní (i mobilní) platformy a možnost kombinování Java a nativního kódu. Nevýhodou je nutnost seznámení se s nativním vývojem pro Android a celkově složitější vývoj. [8]

2.3.2 JavaCV

JavaCV je knihovna spadající pod skupinu Bytedeco. Tato skupina se snaží zpřístupnit nativní knihovny platformě Java. Pomocí technologie JavaCPP, která je vyvíjena souběžně s JavaCV, jsou generovány potřebné vazby (*bindings*).

JavaCPP slouží ke generování JNI kódu a k sestavování nativních obalujících souborů pro knihovny na základě rozhraní napsaných v Javě, přičemž tato rozhraní dokáže automaticky vytvářet z hlavičkových C a C++ souborů.

Skupina Bytedeco se tak snaží postavit pomyslný chybějící most mezi Javou a C++. Bindingy zpřístupňují mnoho významných API a vytváří z nich přenositelný kód schopný běhu v libovolném Java virtuálním stroji včetně Androidu, jako by se jednalo o běžné Java knihovny. Mezi ně patří C/C++ knihovny, jako jsou OpenCV, FFmpeg, LLVM, Tesseract či TensorFlow. [3]

[12] režijní náklady

JavaCV[13] je potom knihovna, která obaluje široce rozšířené nativní knihovny z oblasti počítačového vidění – OpenCV, FFmpeg, libdc1394, PGR FlyCapture, OpenKinect, librealsense, CL PS3 Eye Driver, videoInput, ARToolKitPlus a flandmark. Nabízí také užitečné třídy, které zjednodušují použití těchto knihoven na platformě Java, opět včetně Androidu. Poskytuje rozhraní pro získávání snímků z kamery, jejich zpracování a uložení na disk či poslání po síti.

JavaCV také poskytuje hardwarově akcelerované zobrazování full-screen[14] obrázků, metody pro paralelní spouštění kódu na více jádrech, geometrickou a barevnou kalibraci kamer a projektorů, detekci a porovnání význačných bodů a třídy pro zarovnání obrazu u systémů s projektorem a kamerou. Některé třídy poskytující zmíněnou funkcionalitu mají také své alternativní verze určené pro OpenCL a OpenGL.

Dokumentace této knihovny bohužel není příliš propracovaná, a tak je potřeba nastudovat příklady použití, které jsou ovšem dostupné i pro Android.

Pro použití JavaCV je potřeba nejprve nainstalovat vybranou implementaci Java SE 7 nebo vyšší. Některá funkcionalita JavaCV vyžaduje také další nástroje, u Androidu například SDK API 14 či vyšší. U těchto nástrojů je pak potřeba se vyhnout kombinování 32bitových a 64bitových verzí. [6]

2.3.3 FastCV

FastCV (FastCV Computer Vision SDK) je knihovna pro počítačové vidění vydávaná společností Qualcomm Technologies, Inc. Aplikacím využívajícím fotoaparát nabízí funkčnost, jako je rozpoznávání gest, detekce, rozpoznávání a sledování tváří, rozpoznávání a sledování textu a rozšířená realita. Knihovna cílí na vývojáře sofistikovaných aplikací počítačového vidění pro mobilní zařízení.

Knihovna je pro mobilní zařízení optimalizována a zahrnuje nejčastěji používané funkce pro počítačové vidění pro využití na rozmanitých mobilních zařízeních včetně běžných chytrých telefonů. Knihovnu FastCV lze využít pro tvorbu frameworků, které využijí další vývojáři při tvorbě aplikací s počítačovým viděním, případně ji lze v aplikacích využít napřímo.

Je navržena pro efektivitu na všech procesorech postavených na architektuře ARM, ale je vyladěna pro plné využití procesorů Qualcomm Snapdragon. Díky hardwarové akceleraci lze na mobilních zařízeních pracovat s výpočetně náročnými API pro počítačové vidění a dosahovat dobrého výkonu.

FastCV byla specificky navržena pro efektivní běh na mobilních zařízeních s Androidem, později přibyla podpora pro Windows Mobile a do budoucna je plánována podpora i pro zařízení s iOS. Systém Android podporuje konkrétně od verze 2.1 a vyžaduje Android SDK i NDK (Native Development Kit).

Knihovna je vydávána v unifikované binární podobě. Tento binární soubor má jediné API, ale obsahuje dvě implementace. První implementace je navržena pro efektivní práci na libovolném ARM procesoru a nese název *FastCV for ARM*. Druhá implementace pracuje pouze na systémech na čipu (SoC) od Qualcommu a je nazývána *FastCV for Snapdragon*. Má totožné API, ale poskytuje hardwarově akcelerované implementace funkcí pro počítačové vidění. [5]

[13] https://github.com/bytedeco/javacv
[14] přes celou obrazovku

2.3.4 RenderScript

RenderScript představuje framework pro platformu Android pro vykonávání výpočetně náročných úkonů při dosažení vysokého výkonu. Primárně se orientuje na výpočty prováděné paralelně nad více daty, přestože dokáže přinést zlepšení i do sériových výpočtů. Runtime[15] RenderScriptu paralelizuje úlohy mezi jednotky dostupné na zařízení, jako jsou vícejádrové CPU a GPU. Cílem je umožnit vývojářům soustředit se na algoritmy samotné namísto na plánování úloh. RenderScript je obzvláště vhodný pro aplikace vykonávající zpracování obrazu a počítačové vidění.

RenderScript je založen na dvou hlavních konceptech. Prvním je jeho jazyk pro psaní vysoce výkonného kódu v kernelech[16], který je odvozen od normy C99. Druhým je řídící API pro správu životního cyklu zdrojů RenderScriptu a řízení spouštění kernelů. API je dostupné v jazycích Java, C++ v rámci Android NDK a samotném jazyku pro kernely. RenderScript je dostupný pro zařízení s Androidem od verze 3.0. Upravená verze se zpětnou kompatibilitou pak pro zařízení s Androidem verze 2.3 a vyšším. [9]

Poskytuje základní funkce pro úpravu obrázků a manipulaci pixelů, pro které tedy není potřeba psát vlastní kernely.

Na RenderScript se však dá nahlížet i jiným způsobem než jako na knihovnu pro zpracování obrazu. Jako obecný framework pro psaní efektivního nativního kódu představuje alternativu k využívání Android NDK (viz. podkapitola 4.4). Rozhodnutí pro jeho použití tedy může mít významné důsledky.

Android NDK umožňuje vývojářům psát kód v jazycích C a C++ a komunikovat s Android aplikací pomocí rozhraní JNI. K dispozici jsou standardní knihovny a existující C/C++ kód lze často využít po pár úpravách. RenderScript naopak využívá vlastní jazyk a nové API.

Z pohledu přenositelnosti se jako výhodnější jeví NDK, které dokáže pracovat s existujícím C/C++ kódem, který může být současně využit i na jiných platformách. RenderScript s jinými C aplikacemi pracovat nedokáže. Na druhou stranu je dostupnější na více Android zařízeních než NDK, včetně Android televizí.

Zatímco kód psaný v rámci NDK musí být předem zkompilován pro každou cílovou platformu, RenderScript provádí první část kompilace ještě na vývojovém stroji a druhou část až na cílovém zařízení, což vede na efektivnější nativní kód. To navíc znamená, že libovolné zařízení s podporou RenderScriptu zvládne příslušný kód spustit bez ohledu na architekturu.

Aplikace využívající NDK mohou být debugovány za běhu, což u RenderScriptu neplatí. Lze ale využít logování.

Pro účely výpočtů jako takových může lépe posloužit RenderScript, který dokáže překonat obdobné implementace v NDK, navíc s využitím méně kódu. NDK se ovšem více hodí pro náročné aplikace, které potřebují přístup k více funkcím grafického SDK nebo knihovnám třetích stran. [10]

[15] program za běhu
[16] části kódu (funkce) určené pro paralelní běh na více výpočetních jednotkách

Kapitola 3

Významné techniky používané pro předzpracování obrazu

Mobilní aplikace, systémy a knihovny z předcházející kapitoly jsou pouze prostředky a nástroji umožňujícími realizovat požadovanou funkcionalitu – rozpoznávání textu. Jádrem samotného řešení jsou ovšem techniky a algoritmy zpracování obrazu.

Tato kapitola uvádí elementární techniky předzpracování obrazu, které lze využít na cestě k vylepšení vlastností obrazu, detekci textu a v konečném důsledku zmenšení datového objemu a zlepšení vlastností rozpoznávání textu. Kapitola celou oblast detekce textu také detailněji přibližuje a popisuje jednu z pokročilých metod v současnosti používaných v této oblasti.

3.1 Základní techniky předzpracování obrazu

3.1.1 Šedotónový obraz

Prvním krokem v rámci předzpracování obrazu může být v mnoha případech převod do šedotónové podoby. Tato transformace se může zdát bezvýznamná a teoreticky by často opravdu mohla být vynechána, jelikož její provedení v zásadě nepřináší žádné nové poznatky. Nicméně má hned několik vedlejších dopadů, které mohou být pro další zpracování přínosem, a zároveň nepůsobí tak velkou ztrátu informace, jako by se na první pohled mohlo zdát.

Nejvýznamnějším důvodem, proč si lze dovolit převod obrázku do šedotónové podoby, a zahodit tak informaci o barvě, je poznatek, že jasová (šedotónová) složka obrazu nese zdaleka nejvíce užitečné informace, především při rozlišování jednotlivých objektů v obraze.

Zároveň má tento převod několik pozitivních důsledků. Barva obsažená v obraze v mnoha aplikacích k identifikaci objektů nepomáhá a z pohledu poměru užitečného signálu a šumu ji za takových okolností lze považovat za neužitečný šum, který transformace odstraní. Nezanedbatelné je i zjednodušení implementace obrazových algoritmů, pokud se využívá pouze jasová složka.

Zohlednit lze i lidské hledisko, kdy je pro učení technik zpracování obrazu vhodnější nejdříve pochopit práci se šedotónovým obrazem a tyto poznatky teprve později aplikovat na obraz vícesložkový. Zatímco šedotónový obraz si člověk dokáže představit ve 3D prostoru jako kombinaci dvou prostorových dimenzí představujících základnu a jedné jasové dimenze představující pomyslné vyvýšení, u vícesložkového obrazu to již tak dobře nezvládne.

Konečně nelze opomenout ani zrychlení výpočtů. I přes výkon současných výpočetních zařízení může vykonání obrazových algoritmů stále trvat delší dobu, jestliže se jedná o náročnější metody, je prováděno nad celou databází obrázků, případně běží na mobilním zařízení. Urychlení výpočtů bude vždy přínosem a zpracování šedotónového obrázku může být teoreticky až třikrát rychlejší.

Obrázek 3.1: Převod do šedotónové podoby má minimální vliv na rozlišitelnost objektů [27]

Konkrétních realizací převodu barevného obrazu na šedotónový existuje celá řada. Standard *Recommendation ITU-R BT.709-6*[1] například zohledňuje vyšší citlivost lidského oka na zelenou barvu a definuje formuli

$$Y = 0,2126 * R + 0,7152 * G + 0,0722 * B \tag{3.1}$$

pro získání výsledné jasové (šedotónové) složky Y z původní červené složky R, zelené složky G a modré složky B. [21]

3.1.2 Prahování

Další používanou úpravu obrazu představuje prahování, které bývá často využíváno pro oddělení objektu zájmu od pozadí. Prahování se obvykle aplikuje na obraz převedený do šedotónové podoby.

Jestliže obraz například obsahuje světlý objekt a tmavé pozadí a jim odpovídající úrovně šedé barvy vytvářejí v histogramu obrazu dvě výrazné skupiny (obr. 3.2 vlevo), lze objekt z pozadí extrahovat s využitím prahu T, který obě skupiny odděluje. Pokud má pak daný bod hodnotu větší než práh T, jedná se o bod objektu, jinak se jedná o bod pozadí.

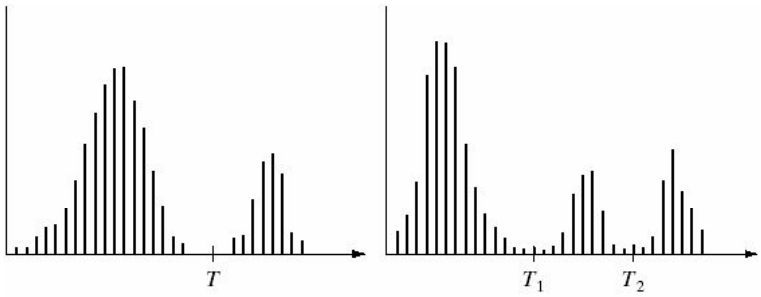
Je-li původní obraz reprezentován funkcí $f(x, y)$, potom je výsledný vyprahovaný obraz $g(x, y)$ definován jako

$$g(x,y) = \begin{cases} 1 & \text{pokud f(x, y)} > \text{T} \\ 0 & \text{pokud f(x, y)} \leq \text{T} \end{cases} \tag{3.2}$$

Body hodnoty 1 potom odpovídají objektu, zatímco body s hodnotou 0 odpovídají pozadí.

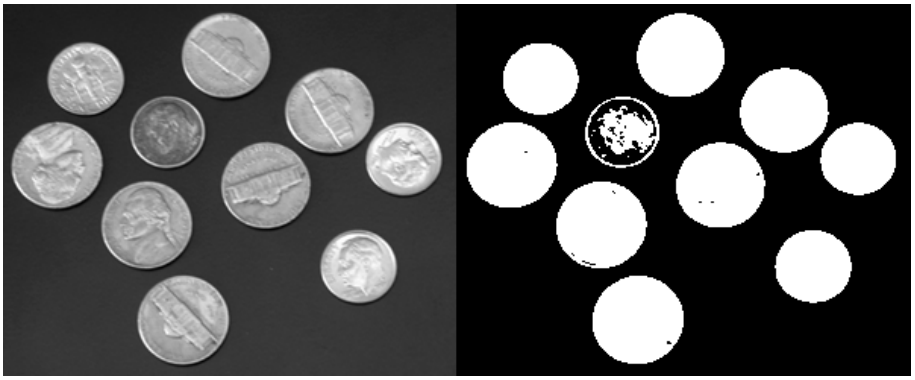
[1] https://www.itu.int/rec/R-REC-BT.709-6-201506-I/en

Pokud je obraz různorodější a obsahuje například více objektů a pozadí (obr. 3.2 vpravo), je potřeba využít víceúrovňové prahování, které lze realizovat například *region growing*[2] metodami.



Obrázek 3.2: Histogramy obrázků s 1 objektem (vlevo) a 2 objekty (vpravo) [24]

Je-li hodnota prahu T totožná pro celý obraz $f(x, y)$, jedná se o globální práh, a tedy globální prahování. Hodnota prahu T může být závislá také na určité lokální vlastnosti $p(x, y)$ bodu (x, y) (například na průměrné úrovni šedi v rámci okolí bodu), tehdy jde o prahování lokální. Pokud práh T navíc závisí na prostorových souřadnicích x a y, prahování se nazývá adaptivní či dynamické.



Obrázek 3.3: Oddělení objektů zájmu od pozadí s využitím prahování. [12]

Negativní vliv na přesnost prahování má nerovnoměrné osvětlení, které z původních jasně oddělených skupin úrovní šedi v histogramu dokáže vytvořit skupinu jedinou bez zřejmého oddělení objektu a pozadí. V této situaci je vhodné využít adaptivní prahování, v rámci kterého je obraz rozdělen do podobrázků s téměř uniformním osvětlením, z nichž v každém je použit jiný práh. Jemnější dělení obrazu potom může napomoci k lepším výsledkům. [24]

3.1.3 Morfologické operace

Morfologické operace (morfologické filtrování) byly vyvinuty v 60. letech minulého století pro analýzu a zpracování diskrétních obrázků. Jedná se o sérii operátorů, které transformují obraz pomocí elementu určitého tvaru. Způsob, jakým element protíná okolní body, rozhoduje o výsledku operace. [25]

[2] metody šíření oblastí

Morfologické zpracování obrazu lze použít jako nástroj pro extrakci komponent obrazu, které jsou užitečné pro reprezentaci a popis tvaru oblastí, jako jsou hranice, kostry či konvexní obálky. Morfologické techniky se hodí pro předzpracování stejně jako pro zpracování následné, například filtraci či ztenčování. Nástroje jako morfologie jsou základním stavebním kamenem pro extrakci „významu“ z obrazu.

Matematický základ morfologie se nalézá v teorii množin. Díky tomu nabízí morfologie jednotný a mocný přístup k řešení mnohých problémů ve zpracování obrazu. Množiny v matematické morfologii reprezentují objekty v obraze (například množina černých pixelů v binárním obraze). Pro binární obrázky spadají tyto množiny do 2D prostoru Z^2, kde každý prvek množiny je dvojice souřadnic (x, y) černého (příp. bílého) pixelu v obraze. Šedotónové obrázky mohou být reprezentovány jako množiny s prvky v prostoru Z^3.

Morfologie je kromě základních konceptů teorie množin, jako jsou sjednocení, průnik či komplement, založena také na tzv. odrazu a posunu. Odraz množiny B je definován jako

$$\hat{B} = \{w | w = -b, \text{ pro } b \in B\}$$

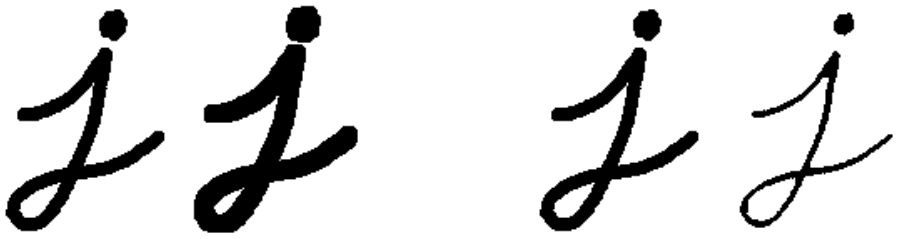
a posun množiny A o $z = (z_1, z_2)$ je definován jako

$$(A)_z = \{c | c = a \ + z \ , \text{ pro } a \ \in A\}$$

Jelikož mnoho morfologických technik pracuje s binárními obrázky, lze pro implementaci zpracování založeného na morfologii použít logické operace. Logické operace AND, OR a NOT totiž přímo odpovídají množinovým operacím. Jediným rozdílem je, že logické operace jsou omezeny na binární proměnné. [24]

Dilatace a eroze

Tyto operace jsou základem pro morfologické zpracování a mnoho morfologických algoritmů je založeno na těchto dvou primitivních operacích.



Obrázek 3.4: Dilatace a eroze aplikovaná na písmeno (černé pixely jsou „aktivní“). [13]

Jsou-li A a B množiny v Z^2, je **dilatace** množiny A množinou B definována jako

$$A \oplus B = \{z \ | (\hat{B})_z \cap A \ \neq \varnothing\}$$

Jde tedy o získání odrazu B a jeho posun o z. Dilatace je pak množina všech *posunutí* z takových, že $\hat{B}$ a A se překrývají alespoň v jednom prvku. Množina B bývá označována jako *strukturní element*. Jednou z nejjednodušších aplikací dilatace je vyplňování mezer.

Jsou-li A a B množiny v Z^2, je **eroze** množiny A množinou B definována jako

$$A \ominus B = \{z | (B)_z \subseteq A\}$$

Jedná se tedy o množinu bodů z takových, že B posunuté o z je obsaženo v A. Nejjednodušším využitím eroze je odstranění z pohledu velikosti irelevantních detailů z binárního obrazu.

Dilatace tedy množinu (objekt) rozšiřuje, zatímco eroze ji zmenšuje. [24]

Otevření a uzavření

Otevření a uzavření představují další dvě důležité morfologické operace.

Otevření množiny A strukturním elementem B je definováno jako

$$A \circ B = (A \ominus B) \oplus B$$

Jde tedy o erozi následovanou dilatací. Otevření obecně vyhlazuje kontury objektu, přerušuje tenké části a eliminuje úzké výčnělky.

Uzavření množiny A strukturním elementem B je definováno jako

$$A \bullet B = (A \oplus B) \ominus B$$

Tedy dilatace následovaná erozí. Uzavření sice také vyhlazuje části kontur, obecně ale spojuje úzké mezery, eliminuje malé díry a vyplňuje mezery v konturách. [24]

Obrázek 3.5: Otevření a uzavření aplikované na písmeno (černé pixely jsou „aktivní“). [15]

Přestože je konečný výsledek použití otevření/uzavření podobný použití eroze/dilatace, otevření a uzavření přesněji zachovávají původní plochu zaujímanou objekty. [21]

Další morfologické operace

Existuje mnoho dalších morfologických operací se specifickým využitím. Například morfologická transformace *hit-or-miss* je základním nástrojem pro detekci tvarů.

Rozšíření morfologických operací pro šedotónový obraz pak umožňuje zavedení dalších operací. Například *morfologický gradient* dokáže zvýraznit ostré šedotónové přechody v obraze při nižší závislosti na orientaci hran. [24]

3.2 Hodnocení kvality obrazu

Techniky vyhodnocení kvality obrazu spadají do trochu jiné kategorie než ostatní techniky uvedené v této práci. Jejich cílem totiž není obraz upravit (předzpracovat), nýbrž pomoci vybrat, na který obraz předzpracování aplikovat. Na „kvalitu“ obrazu lze nahlížet různě. V rámci této práce je „kvalita“ obrazu chápána jako míra ostrosti. Pro určení této míry lze využít tzv. metriky ostrosti (*focus measures*) a související obrazové operátory.

3.2.1 Operátory měření ostrosti

Operátory měření ostrosti by na základě svého principu mohly být rozděleny do zhruba 6 skupin:

1. **Operátory založené na gradientech** - Tyto operátory využívají gradientů či první derivace obrazu. Jsou založeny na předpokladu, že zaostřený obraz obsahuje více ostrých hran než obraz rozmazaný. Gradienty jsou tedy využity ke změření stupně ostrosti.

2. **Operátory založené na Laplaciánu** - Tyto operátory také měří množství hran v obrazu, nicméně tak činí prostřednictvím druhé derivace či Laplaciánu.

3. **Operátory založené na vlnkové transformaci** - Tato skupina operátorů využívá schopnost koeficientů diskrétní vlnkové transformace (DWT) popisovat frekvenční a prostorový obsah obrazu. Koeficienty tedy mohou být využity pro měření úrovně ostrosti.

4. **Operátory založené na statistikách** - Operátory této skupiny využívají mnohé statistiky obrazu pro popis textury k výpočtu úrovně ostrosti.

5. **Operátory založené na kosinové transformaci** - Obdobně jako v případě DWT tato skupina využívá koeficienty diskrétní kosinové transformace (DCT) pro výpočet úrovně ostrosti obrazu na základě jeho frekvenčního obsahu.

6. **Ostatní operátory** - Tato skupina sdružuje operátory založené na jiných principech než operátory předcházejících skupin.

[30]

3.2.2 Vybrané operátory

Práce autora *S. Pertuze* [30] vybírá z celkového počtu 36 operátorů měření ostrosti 11 operátorů s konzistentně dobrými výsledky. Jedná se především o operátory ze skupin operátorů založených na gradientech, Laplaciánu, statistikách a vlnkové transformaci.

S ohledem na tento předvýběr se tato práce zaměřuje na zástupce výše zmíněných skupin.

Tenengrad variance

Zástupcem *operátorů založených na gradientech* je *Tenengrad variance*. U dobře zaostřeného obrázku jsou očekávány ostřejší hrany. Obrazové gradienty jsou tedy velmi dobrým nástrojem k určení spolehlivé metriky ostrosti.

Tato metoda se zakládá na odhadu magnitudy (velikosti) gradientů v každém bodu obrazu a tyto magnitudy sčítá.

Pro konvoluci je využit Sobelův operátor daný maskami

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Magnituda gradientů je vypočítána jako

$$S(m,n) = \sqrt{[G_x(m,n)]^2 + [G_y(m,n)]^2}$$

kde $G_x(m,n)$ a $G_y(m,n)$ jsou po řadě konvoluce vstupního obrázku $I(m,n)$ s maskami S_x a S_y.

Operátor *Tenengrad variance* provádí výpočet rozptylu magnitud gradientů. Tedy pro obraz velikosti $M \times N$ je tento operátor definován

$$TEN\_VAR(I) = \sum_{m}^{M} \sum_{n}^{N} \left[S(m,n) - \overline{S}\right]^2$$

kde $\overline{S}$ je průměrná hodnota magnitud, tedy

$$\overline{S} = \frac{1}{NM} \sum_{m}^{M} \sum_{n}^{N} S(m,n)$$

[29]

Variance of Laplacian

Tento operátor je zástupcem *operátorů založených na Laplaciánu*. Stejně jako ostatní operátory z této skupiny používá druhou derivaci pro analýzu vysokých prostorových frekvencí, které jsou spojené s ostrými hranami. Jako operátor druhé derivace lze využít operátor Laplacián.

Laplacián obrazu je diskrétní aproximací druhé derivace obrazu, kde dochází ke zvýraznění oblastí s rapidními změnami v intenzitě. Z tohoto důvodu se hodí pro detekci změn v ostrosti. [30]

Samotný operátor Laplacián může být aproximován maskou

$$L = \frac{1}{6} \times \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Operátor *variance of Laplacian* provádí výpočet rozptylu absolutních hodnot Laplaciánu. Tedy máme-li obraz velikosti $M \times N$ a $L(m,n)$ představuje konvoluci vstupního obrazu $I(m,n)$ s maskou L, je tento operátor měření ostrosti definován

$$LAP\_VAR(I) = \sum_{m}^{M} \sum_{n}^{N} \left[|L(m,n)| - \overline{L}\right]^2$$

kde $\overline{L}$ představuje průměrnou hodnotu z absolutních hodnot, tedy

$$\overline{L} = \frac{1}{NM} \sum_{m}^{M} \sum_{n}^{N} |L(m,n)|$$

[29]

Gray-level variance

Rozptyl úrovní šedi, který představuje zástupce *operátorů založených na statistikách*, je jednou z nejpopulárnějších metod pro výpočet míry ostrosti obrazu.

Operátor *gray-level variance* provádí pro obraz f o rozměrech $M \times N$ výpočet

$$GRAY\_VAR = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (f(x,y) - \mu)^2$$

kde μ je střední hodnota luminance (jasu) obrazu. [28]

Histogram range

Přestože je předcházející operátor nejoblíbenějším zástupcem *operátorů založených na statistikách*, operátor *histogram range* o něco více zapadá do konceptů statistiky.

Operátor *histogram range* provádí výpočet

$$HIST\_RNG = max\{h(k)\} - min\{h(k)\}$$

kde $h(k)$ je počet pixelů s luminancí (jasem) hodnoty k. [28]

3.3 Detekce textu

Detekce textu představuje velmi důležitou techniku zpracování obrazu, potažmo počítačového vidění. V některých případech, jako je například spojování obrázků (*stitching*), může figurovat sama o sobě, jelikož stačí zajistit, aby spoje nešly skrze text. Mnohem častěji ale detekce textu funguje jako důležitý krok předzpracování pro rozpoznávání textu (OCR).

Zatímco tradiční využití OCR pro skenování kancelářských dokumentů nepřináší zásadní komplikace a detekce je snadná, rozpoznávání textu v přirozených obrázcích naopak komplikací nese mnoho. Je potřeba se vypořádat s velkou variabilitou fontů, krátkými texty, proměnlivým osvětlením, šumem, rozmazáním, nejasným rozložením textu v obraze či zakrytím části textu. Za takových okolností je detekce textu klíčová.

Mezi významné aplikace vyžadující detekci textu v přirozených obrázcích patří například automatická navigace po městech, pomoc vizuálně postiženým, překlad tabulí a značek či geokódování[3]. Někdy je také nalezení a identifikování textu jediným způsobem, jak obrázky rozlišit (například dvě knihy stejného vzhledu).

Existují dvě základní kategorie přístupů k detekci textu – detekce založená na textuře a detekce založená na oblastech. [23, 22]

Detekce založená na textuře

Detekce založená na textuře (*texture-based*) pracuje se skenovacím oknem, se kterým prochází obraz se snahou najít oblasti, které mohou být klasifikovány jako text. Jelikož text v přirozených obrázcích bývá hodně variabilní z pohledu měřítek, používá se několik skenovacích oken. Potřeba je také klasifikátor pro rozdělení oblastí na textové a netextové.

Tato detekce s sebou nese několik komplikací. U skenovacích oken může snadno dojít k vynechání vhodného měřítka. Pro klasifikaci oblastí se využívají příznaky specifické pro text a mnoho z nich je určeno pouze pro horizontální text. Výsledky detekce také bývá obtížné sjednotit. [23, 22]

[3] odvození reálné zeměpisné polohy na základě adresy

Detekce založená na oblastech

Detekce založená na oblastech (*region-based*) provádí agregaci pixelů na základě lokálních příznaků (například barvy). Dochází k extrakci spojených komponent (*connected components*), jež jsou považovány za kandidáty na písmena. Tito kandidáti na písmena jsou spojováni do kandidátů na řádky textu a jsou aplikovány geometrické kontroly.

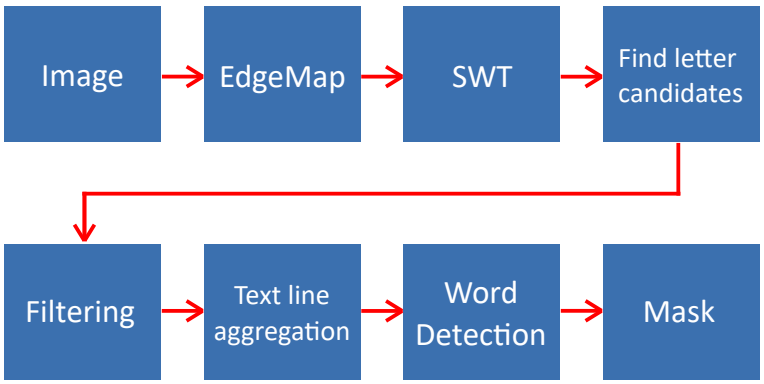
Problematická je lokality příznaků, která je nespolehlivá například při výskytu barevného šumu, který znemožní správné seskupení pixelů na základě barvy. Výhodou práce se spojenými komponentami je naopak vytvoření počáteční segmentace, která je důležitá pro OCR. Zároveň je tento přístup invariantní vůči změnám měřítka. Není ani omezen na pouze horizontální řádky. [23, 22]

3.3.1 Detekce s využitím Stroke Width Transform

Detekce využívající Stroke Width Transform (SWT) je zástupcem detekce založené na oblastech. Napravuje ale nedostatky související s lokalitou příznaků zavedením nového příznaku v podobě šířky tahu (*stroke width*). O SWT lze tedy zároveň mluvit jako o obrazovém operátoru hledajícím hodnotu šířky tahu pro každý pixel. Tento přístup se snaží využít zjištění, že v mnoha abecedách a fontech je šířka tahu konstantní, respektive se mění pouze pomalu. Detekce je celkově nezávislá na měřítku, orientaci, fontu i jazyku.

Postup detekce se nese v duchu všech metod detekce založených na oblastech:

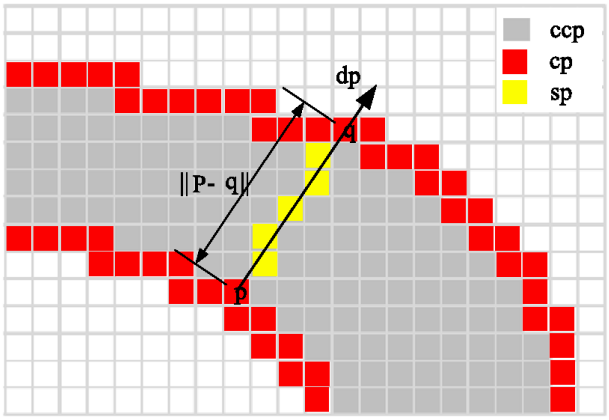
1. aplikace Stroke Width Transform
2. extrakce spojených komponent (kandidátů na písmena)
3. filtrování kandidátů na písmena (uplatnění geometrických omezení)
4. agregace do řádků textu
5. filtrování řádků textu



Obrázek 3.6: Schéma detekce SWT [23]

Výsledkem detekce jsou mimo jiné obalující obdélníky (*bounding boxes*) jednotlivých oblastí s textem.

Prvním krokem algoritmu je detekce hran, konkrétně pomocí Cannyho detektoru, i když by mohly být použity i jiné detektory. Následně je z každého pixelu hrany vystřelen paprsek ve směru gradientu až k dalšímu pixelu hrany se zhruba opačným směrem gradientu. Pokud takový pixel není nalezen, je paprsek zahozen. Dojde ke spočítání počtu pixelů tvořících paprsek a zapsání této hodnoty do každého pixelu tohoto paprsku. Jelikož každým pixelem prochází více paprsků, použije se vždy nejkratší délka paprsku (nejmenší šířka tahu). Problematické jsou rohy písmen, v rámci kterých bývají paprsky chybně dlouhé. Proto se provádí druhý průchod a tyto vysoké hodnoty se nahrazují mediánem délek paprsku (šířek tahu) z ostatních pixelů paprsku. Výsledkem této části je mapa, která každému pixelu přiřazuje šířku tahu.



Obrázek 3.7: Určení šířky tahu v rámci SWT transformace. [33]

Dále se utvářejí spojené komponenty na základě šířky tahu, kdy se spojují pixely s podobnou šířkou tahu.

Následně se provádějí geometrické kontroly. Kromě tradičních, založených na velikosti, změně barvy, poměru stran či průměru komponenty, je možno využít také poměr výšky ku šířce tahu nebo rozptyl (varianci) šířky tahu a podobně.

Poté dochází k seskupování do řádků. Algoritmus nepředpokládá horizontálnost, a tak je seskupování nezávislé na směru. Dojde k vytvoření spojů mezi dvojicemi „kompatibilních“ spojených komponent, přičemž se zahazují ty s příliš rozdílnou šířkou tahu, velikostí, barvou či s velkou vzdáleností od sebe. Spojováním spojů k sobě vznikají lineární řetězce.

Na závěr jsou provedeny další geometrické kontroly řetězců, pro které mohou být využity i klasifikátory z metod detekce založených na textuře.

Na testovací sadě ICDAR překonal tento algoritmus (v roce 2010) všechny ostatní používané algoritmy. Kód algoritmu ovšem společností Microsoft zveřejněn nebyl. Šířka tahu (měřena pro každý pixel) je tedy robustní, nelokální příznak, který si navíc dokáže poradit i s nepřesnostmi hranového detektoru. Detekce je nezávislá na abecedě, fontu či jazyku a důležitým vedlejším produktem algoritmu je segmentace (mapa/maska). [23, 22]

Kapitola 4

Vývoj pro platformu Android

Android představuje mobilní operační systém, s využitím především v chytrých telefonech, za jehož vznikem stojí společnost Google, dnes spadající pod společnost Alphabet Inc. Jedná se o open source platformu, tedy počítačový software s otevřeným zdrojovým kódem, což značí snadnou technickou i licenční dostupnost. Systém lze tedy při splnění jistých podmínek využívat zdarma, lze přistupovat ke zdrojovým kódům a také je upravovat. Operační systém Android je založen na Linuxovém jádře. To zajišťuje zabezpečení systému, správu paměti, správu procesů, přístup k síti a ovladačům senzorů a dalších komponent. K funkcím jádra se ovšem nepřistupuje přímo, ale prostřednictvím Android API. Android také využívá vlastní virtuální stroj navržený s ohledem na optimalizaci paměti a hardwarových prostředků. [32]

Pro přiblížení platformy Android představují následující podkapitoly nástroje potřebné pro vývoj a architekturu platformy samotné. S ohledem na vývoj příkladové aplikace jsou představeny základní komponenty, ze kterých lze aplikaci pro Android sestavit. Jelikož aplikace zaměřená na zpracování obrazu vyžaduje vysoký výkon, věnují se závěrečné podkapitoly možnostem vývoje nativních a paralelních aplikací pro Android.

4.1 Nástroje potřebné pro vývoj

V současné době existuje jediný oficiálně podporovaný způsob vývoje aplikací pro platformu Android. Je jím použití integrovaného vývojového prostředí **Android Studio**, které nahradilo v minulosti používanou kombinaci prostředí Eclipse a doplňku Android Development Tools (ADT). Android Studio je založeno na vývojovém prostředí IntelliJ IDEA, které nabízí kvalitní editor kódu a vývojové nástroje pro Javu. Význačnou součástí Android Studia je potom sestavovací systém Gradle.

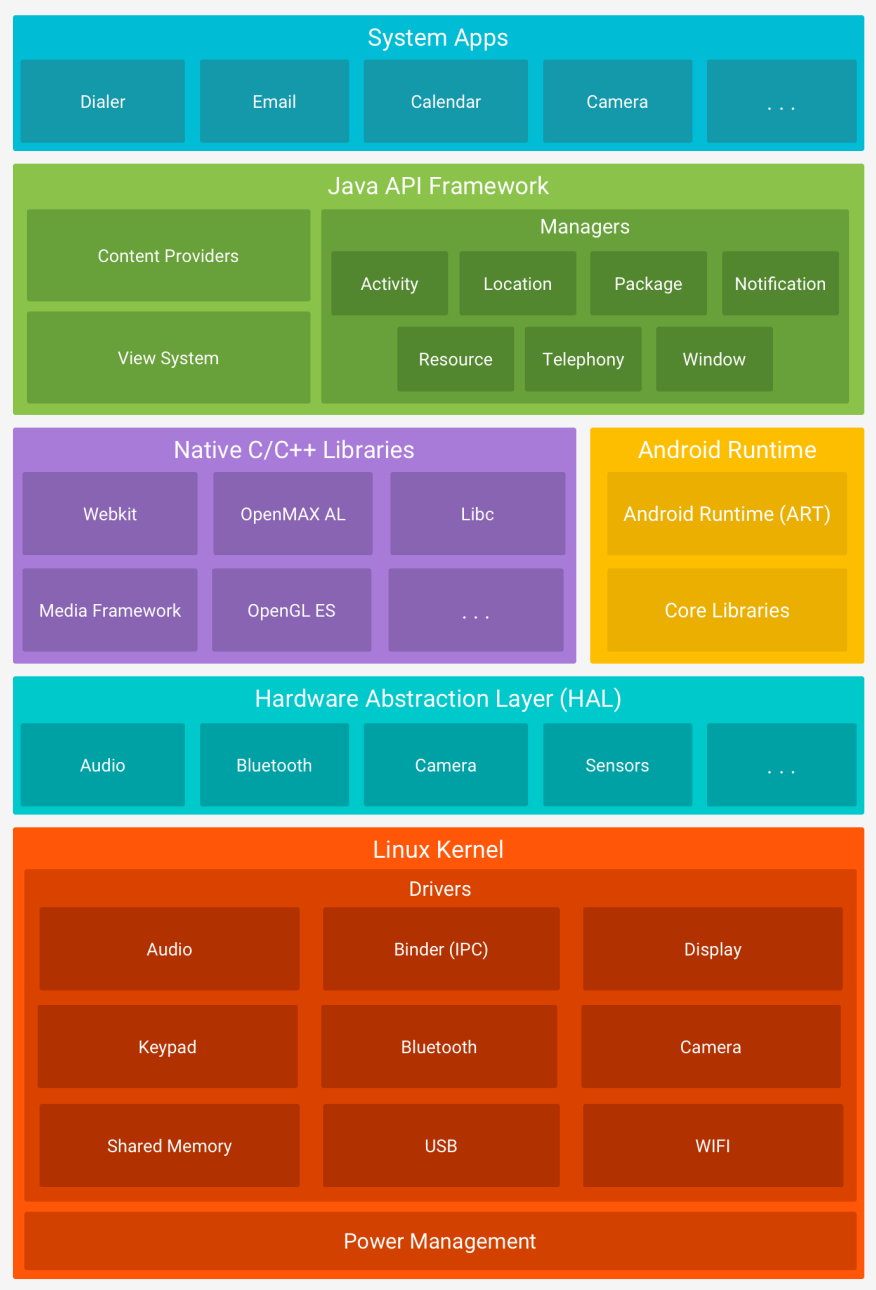
Pro vývoj aplikací je dále potřeba doplnit Android Studio o dvě sady nástrojů. První sadou je **Java Development Kit** (JDK), což je soubor základních nástrojů a knihoven pro vývoj aplikací (a apletů) pro platformu Java. Součástí JDK je Java Runtime Environment (JRE), které umožňuje běh aplikací i vývojových nástrojů, dále překladač, debugger[1] a další.

Druhou sadou vývojových nástrojů je **Android Software Development Kit** (Android SDK). Obecně každé SDK obsahuje knihovny API, dokumentaci, příklady použití v podobě zdrojových kódů apod. Android SDK pak přidává speciální knihovny Javy pro tvorbu Android aplikací, další nástroje pro vývoj a ladění (Android Debug Bridge, Dalvik Debug Monitor Server) a také emulátor. [32]

[1] nástroj pro odladění chyb v programu

4.2 Architektura platformy Android

Architektura operačního systému Android je složena z pěti (respektive šesti) významných vrstev. Z teoretického pohledu lze tyto vrstvy považovat za samostatné, v praxi však vrstvy spolupracují a hranice ustupují do pozadí.



Obrázek 4.1: Architektura platformy Android [16]

Nejnižší vrstvu architektury představuje jádro operačního systému, konkrétně se jedná o **Linuxové jádro** (*Linux Kernel*) verze 2.6. Při startu zařízení dochází k zavedení jádra do operační paměti a předání řízení právě jádru. To má pak pod kontrolou celý systém, koordinuje běžící procesy, má na starosti správu paměti, správu síťové komunikace apod.

Nad jádrem stojí **vrstva abstrakce hardware** (*Hardware Abstraction Layer*, HAL). Jejím primárním účelem je vytvářet abstrakci mezi hardwarem zařízením a softwarem vyšších vrstev architektury. Poskytuje standardní rozhraní, která zpřístupňují hardware zařízení vrstvě *Java API Framework*. Skládá se z několika modulů knihoven, z nichž každý implementuje rozhraní pro konkrétní typ hardwarové komponenty, jako je kamera či bluetooth modul. [16]

Následuje vrstva **nativních C/C++ knihoven** (*Native C/C++ Libraries*). Na těchto nativních knihovnách je postaveno mnoho základních systémových komponent, v čele s ART a HAL. Jedná se například o OpenGL (3D grafika), SQLite (relační databáze) a SSL (šifrovaná internetová komunikace). Tyto knihovny jsou aplikacím zpřístupněny skrze *Java API Framework*. S využitím Android NDK lze k těmto nativním knihovnám přistupovat přímo z nativního kódu. [16]

Další vrstvu představuje **Android Runtime**. Tato vrstva obsahuje virtuální stroj a základní Java knihovny. Dlouhou dobu byl tímto virtuálním strojem DVM (Dalvik Virtual Machine), jenž byl vyvíjen speciálně pro Android. Později byl ale nahrazen virtuálním strojem ART (Android Runtime) s dopřednou kompilací. Virtuální stroj využívá základní funkce Linuxového jádra, jako je koordinace běžících procesů, správa paměti či správa vláken. Hlavním důvodem vzniku vlastního virtuálního stroje byla nemožnost volného šíření standardního virtuálního stroje JVM. Snahou bylo také optimalizovat virtuální stroj s ohledem na použití v mobilních zařízeních.

Zmíněné základní Java knihovny jsou obsahově srovnatelné s platformou Java Standard Edition. Pouze knihovny pro vytváření desktopových aplikací byly nahrazeny knihovnami pro tvorbu uživatelského rozhraní Android aplikací a byly přidány knihovny Apache pro síťovou komunikaci.

Předposlední, pro vývojáře nejdůležitější, vrstvou je **Java API Framework**. Tento aplikační rámec umožňuje přístup ke všem službám Androidu přímo z aplikací skrze API napsané v Javě. Mezi tyto služby patří například tvorba grafického uživatelského rozhraní, zobrazování notifikací, práce s obsahem jiných aplikací, řízení životního cyklu aplikace či přístup k „nekódovým“ zdrojům (textové řetězce, grafika, ...).

Nejvyšší vrstva je zastoupena samotnými **aplikacemi**. Ty nejen slouží přímo uživatelům, ale také nabízejí funkcionalitu, kterou mohou vývojáři využít ze svých aplikací, aniž by ji museli sami implementovat. [32]

4.3 Základní komponenty aplikace pro Android

Aplikace pro operační systém Android se skládají z volně vázaných komponent. Zřejmě nejdůležitější komponentu představuje **aktivita** (*Activity*), která reprezentuje prezentační i řídící vrstvu aplikace. Tato základní znovupoužitelná vizuální komponenta představuje právě jednu obrazovku aplikace. Aplikace je obvykle složena z několika aktivit, které jsou mezi sebou volně vázány. Aktivita, která je určena jako hlavní, je uživateli po spuštění aplikace zobrazena jako první. Každá aktivita může spouštět aktivity další pro realizaci dalších úkonů. Lze spouštět i aktivity jiných aplikací, což umožňuje sdílet funkcionalitu.

Komponenty aplikace, které provádějí dlouhotrvající operace na pozadí, ale nedisponují uživatelským rozhraním, se nazývají **služby** (*Services*). Činnosti služeb tedy nejsou vázány na grafické rozhraní. Služba může být spuštěna z jiné komponenty aplikace a zůstává aktivní na pozadí, i když uživatel přejde do jiné aplikace. Komponentu lze ke službě navázat a komunikovat s ní. Mezi pokročilejší využití služeb patří také zpřístupnění API dalším aplikacím v zařízení. [19]

Poskytovatelé obsahu (*Content Providers*) umožňují ukládání a načítání dat i jejich zpřístupnění ostatním aplikacím. Představují jediný způsob, jak uložená data sdílet mezi aplikacemi. Součástí Androidu jsou poskytovatelé obsahu pro běžné typy dat, jako je audio, obrázky, video či kontakty.

Pokud je potřeba vykonat určitou operaci, **záměr** (*Intent*) slouží k její abstrakci. Záměry fungují jako zprávy, které se posílají mezi komponentami aplikace. Obecně se záměr skládá z činnosti, která se má vykonat, parametrů poskytujících doplňující informace pro činnost a aplikace, která má činnost vykonat. Záměry se dělí na explicitní a implicitní. Explicitní záměr obsahuje komponentu, která se má spustit. Implicitní záměr potom obsahuje činnost, nikoliv však již konkrétní komponentu, která ji má vykonat.

Přijímač (*Broadcast Receiver*) slouží k naslouchání zprávám zevnitř i z vnějšku aplikace. Na tyto zprávy může následně libovolně reagovat, například spuštěním jiné komponenty. Podobně jako služba nemá ani přijímač uživatelské rozhraní.

Poslední, spíše vedlejší, komponentu představují **oznámení**. Existují tři přístupy, jak uživatele informovat v případě výskytu odpovídající situace. Prvním je použití *toast* oznámení. Jedná se o vyskakovací oznámení, které zabírá jen nezbytný prostor pro text, nevyžaduje (ani neumožňuje) interakci a automaticky zmizí. Jako doplněk k *toastu* později vznikl *snackbar*, který umožňuje i jednoduchou interakci. Druhým přístupem je využití notifikace (*Notification*). Ve stavovém řádku se pak zobrazí ikona aplikace a do notifikačního centra se přidá zpráva. Součástí notifikace může být i zvuková, vibrační či světelná signalizace. Třetím přístupem je zobrazení dialogového okna. Dialogové okno se hodí pro situace, kdy je vyžadována interakce uživatele.

Prostředkem, jak výše zmíněné komponenty svázat a jak operačnímu systému sdělit, které komponenty jsou k dispozici, je **Android Manifest**. Jde o XML soubor uložený v kořenovém adresáři aplikace. Specifikuje především všechny komponenty aplikace a požadovaná systémová oprávnění. [32]

Přestože se nejedná přímo o komponenty, jsou významnou součástí aplikace také **zdroje** (*resources*). Mezi zdroje se řadí vykreslitelná grafika (*drawable*), uživatelské rozhraní (*layout*), textové řetězce či hodnoty barev (*values*) a jiná data (*raw*). [26]

4.4 Nativní vývoj na platformě Android

Synonymem pro nativní vývoj je na Androidu sada **Native Development Kit**, zkráceně NDK. Jedná se o sadu nástrojů, která umožňuje implementovat části aplikace s využitím jazyků pro tvorbu nativního kódu, kterými jsou třeba C a C++. U některých typů aplikací může tento přístup usnadnit znovupoužitelnost kódu či knihoven napsaných v těchto jazycích. NDK navíc nabízí platformní knihovny, které lze použít pro tvorbu nativních aktivit nebo pro přístup k fyzickým komponentám zařízení, jako jsou senzory nebo dotykový vstup.

Použití NDK není na místě vždy, zvláště pak pokud se jedná o jednodušší aplikace, které si téměř vždy vystačí s použitím Java kódu a základních API. Hodí se však pro případy, kdy je žádoucí:

- dostat ze zařízení co nejvyšší výkon s cílem snížit latenci nebo spouštět výpočetně náročné aplikace (hry, fyzikální simulace)
- znovupoužít knihovny napsané v C či C++ (ať už vlastní nebo vývojářů třetích stran)

S pomocí NDK je C/C++ kód zkompilován do nativní knihovny, která je následně zabalena do souboru s aplikací (přípona `.apk`). Výchozím nástrojem pro kompilaci nativních knihoven je u Androidu (ve vývojovém prostředí Android Studio) nástroj CMake, který nahradil starší ndk-build. [11]

Typickými kandidáty na implementaci v C/C++ s pomocí NDK jsou samostatné operace náročné na CPU, které nepotřebují mnoho paměti, jako jsou zpracování signálů nebo fyzikální simulace. Prosté přepsání běžné metody do C/C++ běžně nevede na znatelné zvýšení výkonu. Přestože je v současnosti dominantní architekturou ARM, je také dobré si uvědomit, že kód z NDK není přímo přenositelný na jinou architekturu.

Android NDK je tedy rozšíření k Android SDK, které umožňuje použití C/C++ kódu v Android aplikacích. **Java Native Interface** (JNI) je potom rozhraní, které Android používá pro přístup k tomuto kódu.

JNI je částí Java standardu, která umožňuje volat nativní metody napsané v jiných jazycích (např. C a C++) z kódu v Javě. JNI dovoluje nativním metodám používat Java objekty stejným způsobem, jako to dělá Java kód. Nativní metody tedy mohou využívat Java objekty předané z Java (a tedy i Android) aplikace.

Mezi stabilní nativní knihovny (respektive jejich hlavičky) poskytované přímo Androidem (NDK) patří například libc, libm, JNI rozhraní, OpenGL ES, podpora C++ či API pro nativní Android aplikace. [26]

4.5 Vícevláknové aplikace

Při spuštění komponenty aplikace, která nemá spuštěnu žádnou jinou ze svých komponent, spustí Android nový Linuxový proces, který k aplikaci přiřadí, a v rámci něj jedno vlákno. Ve výchozím stavu běží všechny komponenty aplikace ve stejném procesu i vlákně. Toto vlákno se pak označuje jako hlavní (*main thread*).

Hlavní vlákno je velmi důležité, jelikož je zodpovědné za rozesílání událostí jednotlivým elementům uživatelského rozhraní, včetně překreslování. Zároveň v tomto vlákně aplikace s komponentami uživatelského rozhraní interaguje. Z tohoto důvodu se hlavní vlákno někdy označuje také jako vlákno uživatelského rozhraní (*UI thread*).

Všechny komponenty běžící ve stejném procesu jsou instanciovány v hlavním vlákně a systémová volání komponent se rozesílají z tohoto vlákna. Metody, které reagují na systémová volání, proto vždy běží v hlavním vlákně.

Když aplikace vykonává intenzivní práci v reakci na uživatelský vstup, výpočetní model s jediným vláknem vede k nízkému výkonu. Konkrétněji, pokud se vše děje v hlavním vlákně, provádění dlouhotrvajících operací, jako jsou síťová komunikace nebo databázové dotazy, zablokuje celé uživatelské rozhraní.

Ke komponentám uživatelského rozhraní navíc není bezpečné přistupovat z jiných vláken. Již před mnoha lety bylo zjištěno, že jelikož rozhraní musí reagovat na asynchronní události z více zdrojů, je téměř nemožné vyvarovat se deadlocku[2], když je uživatelské rozhraní řízeno více vlákny. [26] Veškerou manipulaci s rozhraním je tedy potřeba provádět z hlavního vlákna.

Pro responzivitu uživatelského rozhraní aplikace je důležité neblokovat hlavní vlákno a déletrvající operace vykonávat v samostatných vláknech na pozadí (*background thread*).

[2] situace, kdy je dokončení první akce podmíněno dokončením druhé akce, přičemž druhá akce může být dokončena až po dokončení první akce

Jak již ale bylo zmíněno, uživatelské rozhraní nelze aktualizovat z jiného než hlavního vlákna. Pro vyřešení této překážky Android nabízí několik způsobů, jak využít hlavní vlákno z vláken ostatních. Jedná se o tři metody (`runOnUiThread`, `post` a `postDelayed`), kterým lze předat kód k provedení (`Runnable`), jenž je následně vykonán v hlavním vlákně.

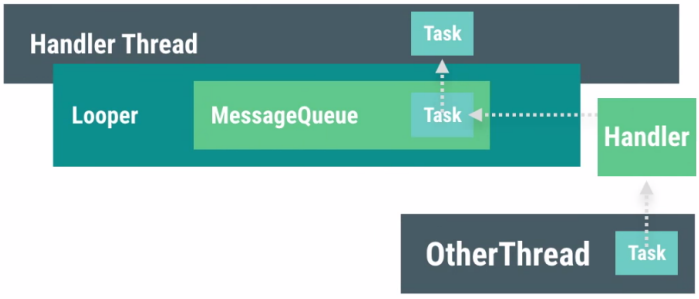
Jak se však komplexnost operací rozrůstá, začne být tento přístup komplikovaný a obtížný pro udržování. Pro zvládnutí složitějších interakcí lze ve vláknech využít `Handlery`, které zpracovávají zprávy, jež si mezi sebou vlákna posílají. [17]

Pro komplikovanější operace však Android nabízí přímo třídy, představující jakási „vláknová primitiva“, která za vývojáře řeší mnoho běžných problémů. Každé z nich se však hodí pro jiný účel.

Prvním primitivem je `AsyncTask`. Ten ve vláknech na pozadí zjednodušuje provádění úkolů, které potřebují interagovat s uživatelským rozhraním. `AsyncTask` provádí asynchronní (blokující) operace na pozadí a zveřejňuje výsledky v hlavním vlákně, bez nutnosti starat se o vlákna či `Handlery`. [17]

Ideálně by však měl být používán pouze pro krátké operace, nejdéle pár sekund.

Druhým, poněkud všestrannějším, primitivem je `HandlerThread`. V základu se jedná o standardní vlákno `Thread` používané v Javě. Na rozdíl od běžného vlákna, které po vykonání práce skončí, však disponuje objektem `Looper`, který vlákno udržuje naživu a vytahuje úkoly k provedení z fronty zpráv `MessageQueue`. Zároveň má k dispozici `Handler` pro přidávání zpráv do fronty a jejich zpracování.



Obrázek 4.2: Schéma principu vláknového primitiva `HandlerThread` [17]

`IntentService` je třetím primitivem, které na vyžádání zpracovává asynchronní požadavky vyjádřené pomocí `Intentů`. Při obdržení požadavků je `IntentService` nastartován, `Intenty` zpracovány ve vlákně na pozadí a `IntentService` je zastaven, jakmile zpracuje všechny požadavky. Všechny požadavky jsou zpracovány v jediném vlákně. V každém okamžiku je tedy zpracováván jediný požadavek. [14]

Poslední primitivum představuje `ThreadPoolExecutor`, který cílí na výrazný paralelismus. Každý dodaný úkol provádí s využitím jednoho z mnoha vláken, která jsou pro `ThreadPoolExecutor` vyhrazena. Vyhrazení vláken řeší dva problémy. Jednak poskytují vyšší výkon při provádění většího množství asynchronních úkolů díky snížené režii zakládání vláken. Za druhé poskytují prostředky pro omezení a správu zdrojů, včetně vláken. [18]

Kapitola 5

Zhodnocení současného stavu a specifikace práce

Tato kapitola nabízí závěry vyvozené z analýzy současného stavu existujících řešení a dostupných technologií v oblasti předzpracování obrazu a rozpoznávání textu. Následně také blíže specifikuje náplň celé práce.

5.1 Zhodnocení existujících mobilních aplikací

Na první pohled bohatý a rozmanitý trh aplikací pro rozpoznávání textu se při druhém pohledu mění na sadu navzájem si velmi podobných aplikací se základní očekávatelnou funkcionalitou. Prakticky všechny aplikace jsou zaměřeny na obecné využití, kterým je skenování dokumentů či jiných záznamových ploch.

Přestože jsem se zatím o konceptu příkladové aplikace pro rozpoznávání potravin nezmiňoval, očekávám u něj – alespoň do jisté míry – snahu o kontinuální zpracování v reálném čase s vykreslováním do náhledu v duchu rozšířené reality. Žádná z uvedených aplikací se o nic podobného ani nepokouší a všechny striktně oddělují fázi jednorázového pořízení snímku od následující fáze postprocessingu[1], s dobou zpracování často v řádu desítek sekund. Současně se většinou nesnaží ani o obohacení rozhraní fotoaparátu – tím spíše, když mnoho z nich využívá systémovou aplikaci namísto vlastní implementace. Tyto aplikace navíc sázejí na ochotu uživatelů podstoupit několik kroků v rozhraní, než dojde k rozpoznávání. Něco takového by u implementace pracující (do jisté míry) v reálném čase bylo nutné eliminovat.

S ohledem na mobilní knihovnu tvořenou v této práci napovídají existující aplikace několik potřebných funkcí, které by knihovna mohla nabízet. Aplikace například velmi často v úvodních fázích zpracování umožňují vymezení rozpoznávané oblasti a provádějí její zarovnání.

Mnoho aplikací má formu samostatného klienta, což má negativní dopad především na přesnost rozpoznávání.

[1] (až) následné zpracování

5.2 Zhodnocení dostupných knihoven pro zpracování obrazu

Pro obecné využití vychází z analýzy dostupných knihoven nejlépe knihovna OpenCV díky své univerzalitě a rozšířenosti.

Nativní použití knihovny OpenCV (na Androidu umožněné pomocí OpenCV4Android v kombinaci s NDK) se hodí pro případy, kdy je potřeba mít k dispozici více možností či větší kontrolu. Z knihoven pro počítačové vidění je OpenCV asi nejdospělejší, z pohledu funkcionality nejrozsáhlejší a má také nejbohatší dokumentaci. Velkou výhodou je možnost implementace a testování C/C++ kódu na plnohodnotném počítači a následné přímé přenesení na mobilní zařízení.

Knihovna JavaCV nabízí možnost, jak pracovat s OpenCV v Javě. Obaluje hlavně OpenCV API pro jazyk C. Implementace je proto o něco náročnější jak z pohledu psaní kódu, tak z pohledu ladění. V JavaCV je k dispozici většina funkcionality z OpenCV, chybí ovšem podrobná dokumentace, takže je potřeba si dopomoci dokumentací OpenCV, v níž mají metody obdobné názvy.

Pokud je funkcionalita obsažená v JavaCV dostatečná a není potřeba žádné vlastní zpracování obrazu, je na Androidu použití JavaCV jednodušším přístupem. Pokud je ovšem použito značné množství vlastního zpracování, začne Java kód zpomalovat celou aplikaci a je vhodnější přejít na nativní úroveň s využitím OpenCV z C/C++.

Ani v případě práce v Javě ovšem není JavaCV na Androidu jasnou volbou. Knihovna OpenCV4Android totiž na Androidu umožňuje pracovat s OpenCV i v Javě. Vzhledem k tomu, že obaluje OpenCV API pro jazyk C++ a je pro Android přímo určená, stává se neoficiální JavaCV vlastně zbytečnou. Jedinou výhodou může být fakt, že JavaCV obaluje i další knihovny z oblasti počítačového vidění.

Knihovna FastCV poskytuje špičkovou optimalizaci pro SoC od Qualcommu, která jsou na chytrých telefonech s Androidem velmi rozšířená. Nabízí ovšem méně funkcí než OpenCV a náskok v optimalizaci může být časem smazán.

Pro běžné úpravy obrazu je na Androidu asi nejlepší volbou RenderScript, který je ze všech knihoven pravděpodobně nejrychlejší.

5.3 Specifikace práce

Cíl této práce se nezrodil přímo a okamžitě, nicméně vykrystalizoval postupně během uvažování nad řešením jednoho lidského problému. Na počátku totiž stála myšlenka na vytvoření mobilní aplikace, jež by dokázala analyzovat (rozpoznávat) složení potravin uváděné na obalech, a oprostit tak zákazníky od čtení drobného textu či nutnosti znát význam jednotlivých složek. V potravinách se totiž (mimo jiné) mohou nacházet jak škodlivá aditiva, tak alergeny, které pro zákazníky mohou představovat reálné nebezpečí. Některé aplikace pro dosažení tohoto cíle využívají skenování čárových kódů a budování vlastní databáze složení produktů. I když je tento přístup jednodušší a přesnější z technického pohledu, v praxi může být problém jak s neaktuálními informacemi kvůli změnám receptur, tak se samotným plněním databáze jednotlivými položkami.

Zároveň si lze představit mnoho dalších aplikací, které by řešily lidské problémy obdobného charakteru – jmenuji například analýzu a porozumění příbalovým letákům u léků. Takové aplikace pak spojuje jejich podkladová technologie – rozpoznávání textu. Jako rozumnější se tak jevilo nejdříve vytvořit více flexibilní systém pro rozpoznávání textu a nad ním následně vybudovat danou mobilní aplikaci.

V kontextu této práce je systém pro rozpoznávání textu chápán jako celek, jejž lze rozdělit na klientskou knihovnu pro předzpracování obrazu s textem a na server provádějící rozpoznávání textu. Náplň této práce ve výsledku tvoří:

- mobilní knihovna pro předzpracování obrazu s textem
- mobilní aplikace pro rozpoznávání složení potravin

Serveru pro rozpoznávání textu se věnuje práce kolegy Bc. Petra Bobáka [20].

Cílem knihovny (stejně jako celého systému) je usnadnit a zrychlit vývoj mobilních aplikací využívajících rozpoznávání textu. Knihovna by měla poskytnout funkcionalitu předzpracování obrazu s textem, která je nejčastěji potřebná pro rozpoznávání textu. Kromě základních technik pro úpravu obrazu se primárně jedná o **detekci textu**. Výběr textových oblastí umožní zejména snížit datový objem přenášených dat, zrychlit rozpoznávání textu díky menší velikosti zpracovávaných oblastí a v neposlední řadě zvýšit přesnost rozpoznávání díky eliminaci případů, kdy je v netextové oblasti chybně „rozpoznán“ text. Dále se jedná o možnosti **vyhodnocení kvality pořízených snímků** pro výběr toho nejvhodnější s ohledem na zvýšení přesnosti rozpoznávání. Přestože bývá obecné předzpracování integrováno i v samotných systémech rozpoznávání textu, může vést předzpracování v mobilním zařízení na lepší výsledky díky možnosti přizpůsobení pro účely konkrétní aplikace. S ohledem na činnost kolegy Bobáka tato práce také předpokládá primární použití rozpoznávacího software **Tesseract**, u něhož je vlastní předzpracování přímo doporučováno samotnými vývojáři.

Mobilní aplikace si pak klade za cíl usnadnit uživatelům výběr potravin nebo uživatele dokonce varovat před potenciálním nebezpečím. Měla by umožnit pořízení snímků, jejich předzpracování, odeslání na rozpoznávací server, zpracování odpovědí a interpretaci a vhodnou prezentaci výsledků (složení) uživatelům. Mobilní aplikace bude realizována na platformě **Android** v jazyce Java.

Konečným záměrem této a kolegovy práce není překonání již existujících řešení, ale vytvoření otevřeného systému postaveného na veřejně dostupných technologiích. Velkou osobní motivací je také proniknutí do oblasti zpracování obrazu, které by pouhé využití existujících alternativ neumožnilo.

Kapitola 6

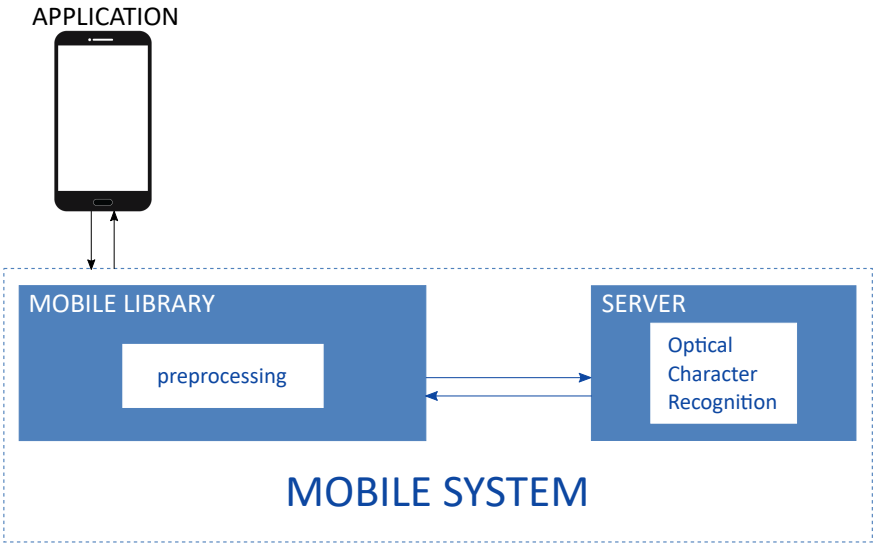
Realizace mobilní knihovny a mobilní aplikace

6.1 Architektura systému

Celková architektura řešení, která byla naznačena již v předcházející kapitole a která bude v následujících podkapitolách detailně rozebrána, se bude skládat ze tří významných částí.

Proces začne u **mobilní aplikace**, která obecně nebude omezena platformou, a tedy by se mohlo jednat nejen o aplikaci pro Android, ale například i pro iOS. Obecně bude aplikace zodpovědná za pořizování snímků, síťovou komunikaci a především za interpretaci a vhodnou prezentaci informací uživateli. Veškeré technologie spojené s rozpoznáváním textu pak pro aplikaci zajistí **mobilní systém**.

V rámci systému bude mobilní aplikace pro svou činnost moci **mobilní knihovnu** využívat na dvou úrovních. Jednak bude moci využít platformě specifické komponenty nabízené knihovnou (např. rozhraní fotoaparátu pro Android). Především ale pořízené snímky předá mobilní knihovně, kde dojde k předzpracování ve smyslu základních úprav obrazu v čele s detekcí textu. Takto předzpracované snímky budou následně odeslány na **rozpoznávací server**, k čemuž bude opět možné využít platformě specifickou komponentu knihovny.

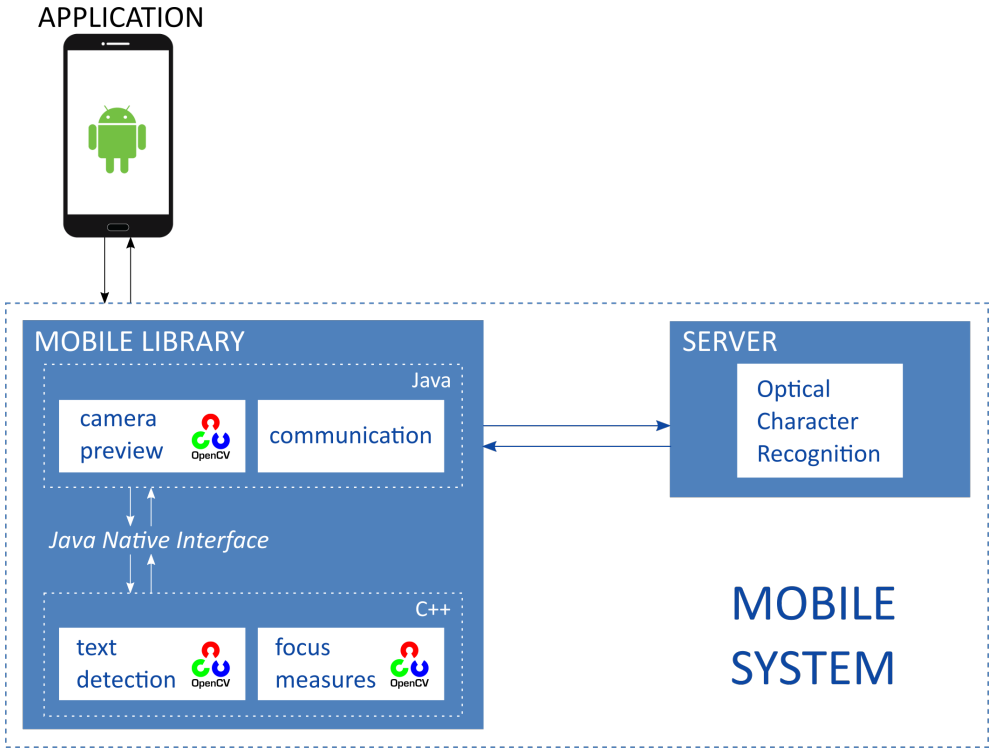


Obrázek 6.1: Architektura celého řešení

Na serveru dojde k rozpoznání textu s využitím jedné ze tří technologií – Tesseract (viz. 2.2.1), Google Cloud Vision (viz. 2.2.3) nebo Microsoft Computer Vision. Výsledky rozpoznávání budou odeslány zpět a předány aplikaci, která s nimi naloží dle svého zaměření.

6.2 Architektura knihovny pro předzpracování obrazu

Primárním účelem knihovny pro předzpracování obrazu s textem je usnadnit a urychlit vývoj mobilních aplikací zahrnujících rozpoznávání textu, a to poskytnutím často potřebné funkcionality. Z tohoto důvodu se bude knihovna skládat ze dvou hlavních částí – platformě specifických komponent pro vybudování aplikace a algoritmů zpracování obrazu.



Obrázek 6.2: Architektura mobilní knihovny pro předzpracování obrazu s textem

6.2.1 Platformě specifické komponenty

První část knihovny bude obsahovat platformě specifické komponenty, které bude možné volitelně použít v rámci aplikace, resp. nad kterými bude možné aplikaci vybudovat. V této práci se bude jednat o komponentu **rozhraní fotoaparátu** umožňující snadné aplikování algoritmů zpracování obrazu na pořízené snímky a tvorbu vlastního uživatelského rozhraní (nadstavby) a komponentu usnadňující **síťovou komunikaci** při odesílání obrazových dat na server. Jelikož je mobilní aplikace tvořená v této práci určena pro platformu Android, budou obě komponenty napsány v jazyce **Java**.

Pro samotné pořizování snímků fotoaparátem mobilního zařízení nabízí Android několik možností. Tou nejzákladnější je využití systémové aplikace fotoaparátu, což spadá do filozofie „sdílení funkcionality“, které se Android drží. Pro tento účel stačí v rámci aplikace vyjádřit záměr pořízení fotografie, načež je spuštěna systémová aplikace, uživatel pořídí snímek a tento snímek je předán zpět do aplikace, která s ním naloží dle vlastního uvážení. Hlavní výhodou tohoto přístupu je extrémní jednoduchost a vyhnutí se práci s hardwarem zařízení. Aplikace má navíc k dispozici rozhraní fotoaparátu s kompletními možnostmi nastavení. Nevýhodou je však nemožnost začlenění rozhraní přímo do aplikace, a především nulová možnost úprav rozhraní jak po vizuální, tak po funkční stránce.

Pro větší flexibilitu, kterou vyžaduje nejen aplikace tvořená v této práci, je nutné vybudovat rozhraní vlastní. Pro tyto účely nabízí Android starší API `android.hardware.Camera` nebo novější verzi `android.hardware.camera2`. V této práci však bude použita třetí alternativa. S ohledem na následné zpracování obrazu prováděné v rámci knihovny bude komponenta rozhraní fotoaparátu vybudována s využitím knihovny **OpenCV4Android**.

Komponenta pro síťovou komunikaci se serverem pro rozpoznávání textu bude vybudována s využitím knihovny **Volley**, což je oficiální HTTP knihovna pro platformu Android zajišťující jednoduchou a rychlou komunikaci po síti. Detailnější popis této komponenty je součástí podkapitoly 6.4.

6.2.2 Algoritmy zpracování obrazu

Do druhé části knihovny budou spadat algoritmy pro zpracování obrazu. Půjde o algoritmy pro **vyhodnocení kvality pořízených snímků** a algoritmy realizující **detekci textu**, jejichž součástí budou i základní úpravy obrazu. Přestože se nejedná o vyloženě náročné algoritmy, je potřeba uvážit jejich nasazení na mobilních zařízeních, která disponují nižším výkonem. Navíc jejich použití v kontextu mobilních aplikací zvyšuje požadavky na rychlost těchto algoritmů, aby uživatel nemusel strávit příliš mnoho času se zařízením v ruce. V neposlední řadě pak nelze opomenout snahu o nezávislost těchto algoritmů na platformě. Pro splnění všech těchto požadavků bude pro implementaci algoritmů zpracování obrazu použita knihovna **OpenCV4Android** a její nativní použití v jazyce **C++**, které zajistí jak dosažení vysokého výkonu, tak přenositelnost algoritmů. Přidanými bonusy budou bohatá dokumentace a možnost rychlého vývoje a testování na plnohodnotném počítači.

Algoritmy pro vyhodnocení kvality obrazu budou zaměřeny na analýzu ostrosti obrazu, a budou tedy využívat operátory měření ostrosti (viz. podkapitola 3.2). Konkrétně se bude jednat o 4 operátory zastupující kategorie operátorů založených na gradientech, Laplaciánu a statistikách. Primární využití by měly nalézt při pořízení více snímků a výběru toho nejlepšího pro zvýšení přesnosti rozpoznávání.

Pro detekci textu nabídne knihovna 2 algoritmy s rozdílnou filozofií. V prvním případě půjde o „základní detekci“, jejíž jádro budou tvořit morfologické operace. Tento přístup pravděpodobně nebude příliš robustní a bude vhodné jej upravit pro účely konkrétní aplikace, na druhou stranu by však měl nabídnout relativně rychlou detekci, což je v prostředí mobilních aplikací velmi žádoucí. Druhým algoritmem bude detekce postavená na Stroke Width Transform. Tato moderní a úspěšná technika by mohla nabídnout mnohem přesnější výsledky, cenou však bude také výrazně delší doba zpracování, která nemusí být vždy přípustná. Využití by měla najít v aplikacích, které budou vyžadovat detekci textu v přirozeném prostředí.

6.2.3 Spolupráce částí mobilní knihovny

Jak již bylo v minulých podkapitolách vysvětleno, knihovna bude složena ze dvou částí, z nichž každá bude využívat jiný programovací jazyk. Zatímco platformě specifické komponenty pro Android budou vytvořeny v jazyce Java, ostatní algoritmy zpracování obrazu budou napsány v jazyce C++. Pro komunikaci mezi oběma částmi a zajištění této méně tradiční spolupráce bude potřeba vložit meziprvek v podobě rozhraní **Java Native Interface** (JNI). Jedná se o nativní programové rozhraní, které umožňuje Java kódu běžícímu v Java virtuální stroji interagovat s kódem napsaným v jiných programovacích jazycích, jako je C, C++ či Assembler. S pomocí JNI lze v nativním kódu také pracovat s Java objekty či volat Java funkce.

6.3 Implementace knihovny pro předzpracování obrazu

Klíčové implementační detaily mobilní knihovny a dostupné algoritmy zpracování obrazu jsou přiblíženy v této podkapitole.

6.3.1 Komponenta rozhraní fotoaparátu

Použitá knihovna OpenCV4Android sama nabízí rozhraní, které je však doprovázeno dvěma nevýhodami. Tou první, méně kritickou, je využití staršího API `android.hardware.Camera`. Druhou, poměrně zásadní, nevýhodou je práce rozhraní pouze v režimu na šířku. Při snaze o použití v režimu na výšku je náhled pootočen a výrazně deformován. Režim na šířku se hodí pro klasické fotografování s použitím obou rukou, nicméně pro účely rozpoznávání textu – ať už se jedná o skenování dokumentů nebo rozpoznávání složení potravin – je vhodnější režim na výšku. Technicky vzato není problémem ani tak samotné rozhraní, v rámci kterého by šlo získaný snímek před zpracováním transformovat a prvky rozhraní by šlo pootočit. Problémem je však zbytek systému, který do režimu na šířku přizpůsobí stavové i navigační panely, a pak je držení zařízení na výšku zcela neintuitivní. Vlastní rozhraní fotoaparátu `PortraitJavaCameraView` tak toto rozhraní z OpenCV4Android využívá, avšak přidává transformační krok ihned po pořízení snímku, díky čemuž může náhled fotoaparátu i zbytek systému pracovat v režimu na výšku.

Komponenta umožňuje jednoduchou aplikaci obrazových algoritmů na pořízené snímky ve funkci `onCameraFrame`, do které jsou všechny snímky posílány před jejich vykreslením do náhledu fotoaparátu.

6.3.2 Operátory měření ostrosti

Vybrané operátory (viz. podkapitola 3.2.2) jsou implementovány výhradně s využitím knihovny OpenCV, a to v souladu s teoretickým popisem uvedeným v této práci. Proto jsou v této části pouze zdůrazněny užitečné funkce, jež OpenCV nabízí.

Tenengrad variance

Základem tohoto operátoru je filtrace s využitím Sobelova operátoru. Pro tyto účely nabízí OpenCV funkci `Sobel`, která počítá první derivaci obrazu. Filtrace je provedena pro horizontální i vertikální směr, a to s jádrem velikosti 3×3. Výsledky filtrace jsou dále sloučeny dle rovnice uvedené v teoretické části. Pro výpočet rozptylu (variance) OpenCV žádnou funkci nenabízí. Lze však spočítat směrodatnou odchylku funkcí `meanStdDev` a její výsledek umocnit na druhou právě pro získání rozptylu.

Variance of Laplacian

Operátor Laplacián je aplikován s využitím funkce `Laplacian` při velikosti jádra 3×3. Pro výpočet rozptylu je opět využita funkce `meanStdDev` s následným umocněním výsledku pro získání rozptylu.

Gray-level variance

Při aplikaci tohoto operátoru je směrodatná odchylka vypočítána nad zdrojovým obrazem převedeným do šedotónové podoby. Její umocnění na druhou poskytuje výslednou hodnotu. Pro kompenzaci rozdílů v průměrném jase napříč různými obrázky může být rozptyl normalizován střední hodnotou μ.

Histogram range

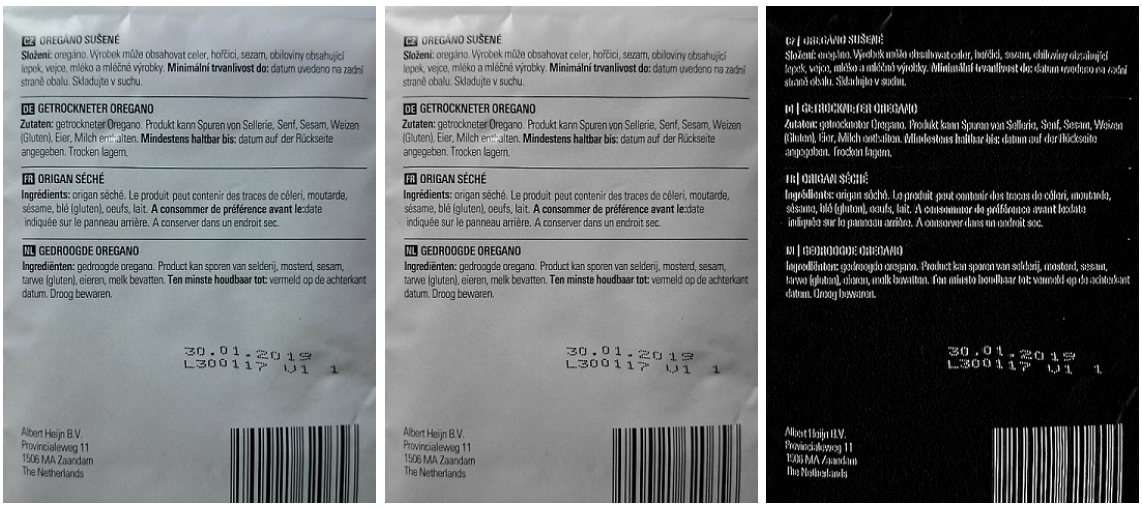
Pro výpočet histogramu je využita funkce `calcHist`. V rámci tohoto histogramu je potřeba nalézt nejmenší a největší hodnotu, což lze realizovat s pomocí funkce `minMaxLoc`, která dokáže v případě potřeby vrátit i polohu minima a maxima. Výsledná míra ostrosti je získána rozdílem těchto hodnot.

6.3.3 Základní detekce textu

Tato základní detekce je založena na použití morfologických operací. Hlavní výhodou je její relativně rychlé provedení.

Prvním krokem je převedení pořízeného snímku z barevného modelu RGBA, který je výchozím barevným modelem snímků pořízených kamerou při použití OpenCV, do šedotónové podoby. Pro tento převod lze využít funkci `cvtColor` s parametrem udávajícím patřičný typ převodu.

V získaném šedotónovém obraze jsou detekovány hrany. Detekci hran lze provést celou řadou detektorů, v tomto konkrétním případě je použit Sobelův operátor. Jelikož se text vyznačuje vysokou frekvencí změn v horizontální ose, stačí Sobelův operátor aplikovat pouze jedenkrát pro detekci vertikálních hran. Detekce hran je provedena funkcí `Sobel`. Krok detekce hran je velmi kritický, jelikož umožňuje detekovat tmavý text na světlém pozadí stejně jako světlý text na tmavém pozadí.



Obrázek 6.3: Originální obrázek, šedotónová podoba a detekované hrany

Obraz s detekovanými hranami je následně vyprahován pro získání binárního obrazu, ve kterém je text (jeho hrany) reprezentován bílou barvou a pozadí barvou černou. Pro zvýšení přesnosti prahování je použita metoda Otsu, která zajišťuje automatickou volbu vhodného prahu. Prahování je realizováno funkcí `threshold`.

Nejvýznamnější částí algoritmu je potom aplikace morfologického uzavření na binární obraz. Morfologické uzavření se skládá z dilatace následované erozí. Dilatace zajistí spojení blízkých písmen i řádků do jednoho celku, eroze pak zamezí zbytečnému nárůstu oblastí reprezentujících text. Strukturní element je také upraven do obdélníkového tvaru, aby primárně spojoval písmena v řádcích a aby řádky navzájem spojoval pouze, pokud jsou velmi blízko sebe. Obecným problémem morfologických operací je jejich závislost na velikosti strukturního elementu. V tomto konkrétním případě to znamená, že příliš malý element nezajistí spojení písmen a příliš velký element by naopak mohl spojit i zcela nesouvisející oblasti. Jinak řečeno pro každou velikost písma je vhodná jiná velikost elementu a s jednou velikostí elementu nelze spolehlivě detekovat všechny velikosti písma. Pro konkrétní aplikaci, jako je např. rozpoznávání složení potravin, si však lze dovolit zvolit fixní velikost elementu, jelikož lze očekávat, že složení bude vždy uvedeno se zhruba podobnou velikostí písma. Morfologické uzavření je realizováno funkcí `mophologyEx` s nastaveným typem operace `MORPH_CLOSE`.

Pro vyhlazení tvaru oblastí a odstranění příliš tenkých spojů je dále aplikováno morfologické otevření, tedy eroze následovaná dilatací. Eroze odstraní zbytečné výčnělky či úzké spoje a dilatace zachová celkovou plochu oblastí. Stejně jako v předcházejícím případě je morfologické otevření provedeno funkcí `morphologyEx`, tentokrát s typem operace `MORPH_OPEN`.

Dále je potřeba určit kontury jednotlivých oblastí. Pro tento účel je použita funkce `findContours`, která je nastavena na nalezení pouze vnějších kontur.

Pro všechny nalezené kontury jsou dále určeny obalující obdélníky funkcí `boundingRect`. Na tyto obdélníky jsou následně aplikovány geometrické kontroly ověřující například poměr stran, absolutní velikost či procentuální poměr mezi plochou kontury a plochou obalujícího obdélníku.

Obdélníky, které projdou kontrolami, jsou na závěr ještě mírně zvětšeny, aby kolem textu zůstal dostatek místa pro snazší rozpoznávání textu na serveru.

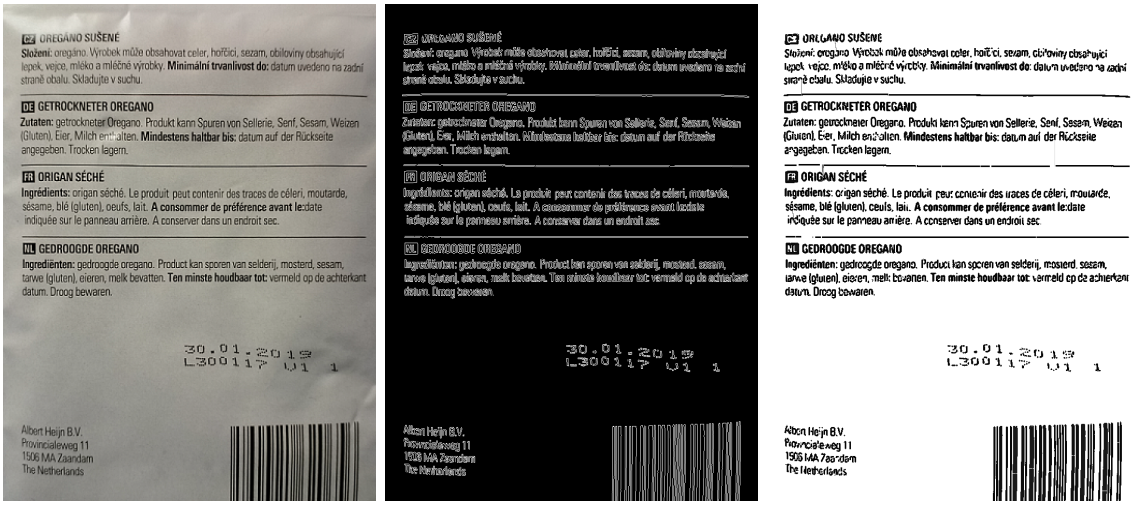
Obrázek 6.4: Prahování, morfologické operace a výsledné obalující obdélníky

6.3.4 Detekce textu založená na Stroke Width Transform

Tato detekce je určena pro použití v přirozeném prostředí, kde se kromě textu vyskytuje celá řada jiných objektů. Je výrazně robustnější než základní detekce založená na morfologických operacích, na druhou stranu její provedení trvá mnohem déle.

Jelikož je tato detekce výrazně komplexnější a její kompletní realizace by byla nad rámec této práce, byla zvolena cesta použití existující implementace a její úpravy pro účely i možnosti mobilního zařízení. I přes velmi kvalitní výsledky se zatím tato metoda nedočkala mnoha implementací. Ani OpenCV tuto detekci nenabízí. Jednu z profesionálnějších implementací lze nalézt uvnitř knihovny počítačového vidění CCV[1]. Jelikož je však zbytek algoritmů v této práci napsán s využitím knihovny OpenCV, byla nakonec vybrána implementace[2], která tuto knihovnu také využívá. Stojí za ní autoři Saurav Kumar a Andrew Perrault.

Kromě knihovny OpenCV však tato implementace vyžaduje ještě knihovnu Boost. Bohužel pro platformu Android neexistuje žádná předpřipravená (sestavená) verze, a tak je potřeba knihovnu zkompilovat přímo ze zdrojových kódů, a to samostatně pro jednotlivé architektury procesorů. K dispozici není ani oficiální a spolehlivý návod, jak kompilaci provést. Jelikož kompilace nativní knihovny v rámci Android aplikace není rozhodně jednoduchým úkolem, nabízí se využít neoficiální předkompilované verze od jiných vývojářů, jako např. *Boost-for-Android-Prebuilt*[3]. Protože je však potřeba získat správnou verzi pro danou kombinaci verze NDK a architektury zařízení, je vysoká pravděpodobnost, že ani tento přístup nepomůže. V rámci této práce se nakonec podařilo implementaci detekce zprovoznit s využitím pouze hlavičkových souborů oficiální verze knihovny Boost[4].



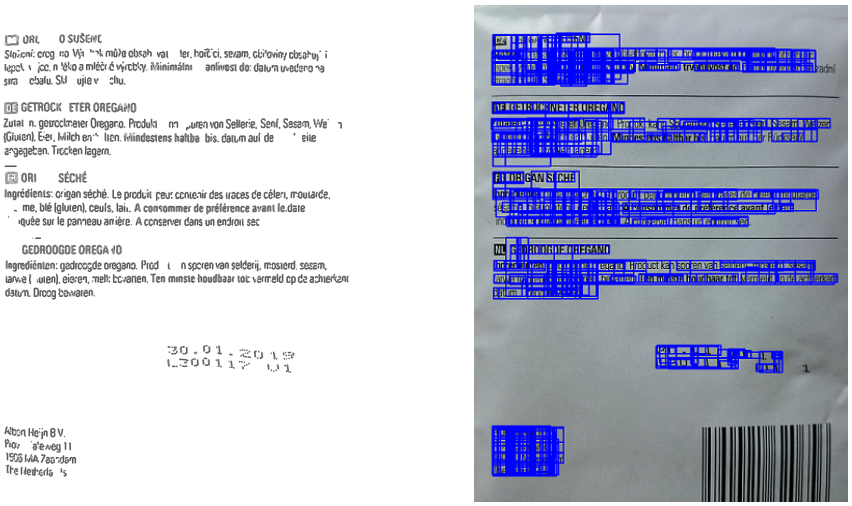
Obrázek 6.5: Originální obrázek, detekované hrany a operátor SWT

Kód byl dále zjednodušen a byly odebrány zbytečné mezistupně, které přímo nerealizovaly detekci textu, jako např. příprava nalezených entit (mezivýsledků) na renderování. Původním výstupem detekce navíc nebyly obalující obdélníky detekovaného textu, nýbrž obrázek s uměle vyrenderovanými komponentami představujícími textu. Ty však pro následné rozpoznávání textu nejsou dostatečně kvalitní. Z tohoto důvodu je implementace

[1] http://libccv.org/doc/doc-swt/
[2] https://github.com/aperrau/DetectText
[3] https://github.com/r4sas/Boost-for-Android-Prebuilt
[4] http://www.boost.org/

upravena pro nalezení obalujících obdélníků, s pomocí kterých může být následně text extrahován z původního obrázku. Výstup je tak navíc jednotný s algoritmem základní detekce, a oba algoritmy jsou tedy jednoduše zaměnitelné.

Na obrázku 6.6 si lze všimnout vyšší přesnosti detekce založené na SWT, která dokázala odlišit čárový kód od zbylého textu.



Obrázek 6.6: Uměle vyrenderované komponenty a výsledné obalující obdélníky

6.3.5 Pomocné algoritmy

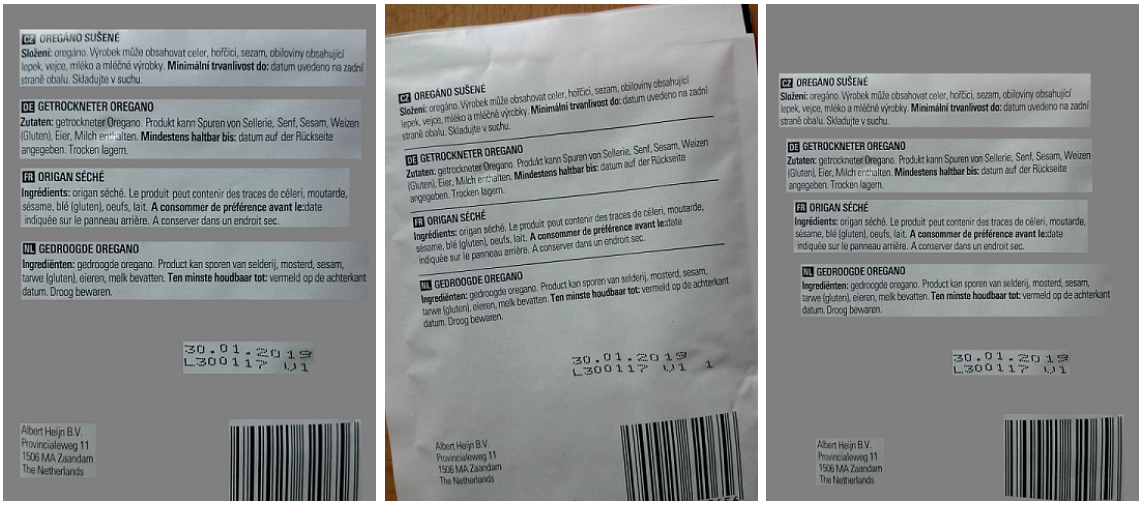
Algoritmy uvedené v předcházejících podkapitolách představují ty nejklíčovější části předzpracování. Tyto algoritmy samotné však ještě neumožňují dosáhnout požadované funkcionality. Z tohoto důvodu obsahuje knihovna několik dalších pomocných funkcí.

Hned tři funkce slouží ke zužitkování obalujících obdélníků nalezených v rámci detekce textu. Funkce `retainTextRegions` umožňuje z originálního snímku odstranit všechny netextové části a nahradit je souvislým pozadím. Je vhodná pro případy, kdy je potřeba odeslat na server všechny textové části v rámci jednoho souboru.

Funkce `extractTextRegions` slouží k efektivnímu přenosu popisu nalezených textových oblastí z C++ do Java kódu, kde mohou být dle těchto popisů extrahovány pouze textové části, které mohou být na server odeslány odděleně.

Třetí a zároveň nejdůležitější funkcí je `rotateAndRetainTextRegions`. Jak název napovídá, účel této funkce je velmi podobný jako u funkce první, klíčová je však schopnost horizontálně srovnat všechny detekované textové části. To je opravdu důležité, jelikož rozpoznávací software Tesseract si neporadí ani s mírně nakloněným textem.

V podkapitole o detekci textu využívající SWT nebylo zmíněno, že použitá implementace nepodporuje současnou detekci tmavého textu na světlém pozadí a obráceně. Informaci o druhu detekce je potřeba explicitně dodat. Z tohoto důvodu nabízí knihovna funkci `darkOnLight`, která pomocí analýzy průměrné barvy snímku odhaduje, zda se jedná právě o tmavý text nebo opačně.



Obrázek 6.7: Výsledek funkce `retainTextRegions` aplikované po základní detekci, snímek nakloněného obalu a jeho srovnání pomocí funkce `rotateAndRetainTextRegions`

6.4 Spolupráce se serverem pro rozpoznávání textu

Jak již bylo zmíněno, komponenta pro spolupráci se serverem využívá pro síťovou komunikaci knihovnu Volley. Jedná se o HTTP knihovnu pro jednoduchou a rychlou síťovou komunikaci v Android aplikacích. Volley nabízí mnoho užitečné funkcionality v čele s automatickým plánováním požadavků, udržováním více současných spojení, využitím cache pamětí a jednoduchou upravitelností.

Server pro rozpoznávání textu přijímá HTTP požadavky realizované metodou POST a očekává dotazy s typem obsahu `multipart/form-data`. Tento typ je vhodný pro předávání několika souborů (o různých typech) současně. Knihovna Volley však nenabízí možnost dotaz s obsahem `multipart/form-data` vytvořit. Z tohoto důvodu byl základní dotaz třídy `Request` rozšířen pro vytvoření vlastního dotazu (komponenty) `VolleyMultipartRequest`. S pomocí této komponenty lze jednoduše vytvořit dotaz obsahující i několik obrázků. Dle požadavků serveru tak lze v jednom dotazu posílat nejen jediný obrázek se všemi detekovanými částmi textu, ale i několik obrázků reprezentujících jednotlivé části odděleně.

Dotaz má také prodloužený časový limit a upravenou politiku opakování neúspěšných dotazů. Výchozí časový limit používaný při běžné síťové komunikaci totiž nemusí být pro provedení rozpoznávání textu dostatečný. Vícenásobné opakování dotazu by navíc zbytečně zatížilo server.

Pro využití funkcionality serveru je nejprve potřeba provést autorizaci, konkrétně prostřednictvím protokolu OAuth 2.0. Na webovém rozhraní serveru je nutno jednorázově zaregistrovat mobilní aplikaci. Na oplátku jsou vygenerovány údaje *client id* a *client secret*. Před samotným rozpoznáváním je potřeba tyto údaje odeslat na server, který mobilní aplikaci dočasně poskytne tzv. token. Při všech dalších požadavcích musí aplikace tento token do svých dotazů vložit prostřednictvím hlavičky `Authorization`.

Pro rozpoznávání textu server využívá jednu ze tří technologií – Tesseract (viz. 2.2.1), Google Cloud Vision (viz. 2.2.3) nebo Microsoft Computer Vision. Použitou technologii lze určit s pomocí parametru `method` předávaného v dotazu. Tento parametr tedy může nabývat hodnot `tesseract`, `google` či `microsoft`, případně `auto` pro automatickou volbu nejlepší metody. Tato práce primárně předpokládá využití systému Tesseract, který je jako jediný bezplatný a jakožto open source projekt nabízí flexibilitu při tvorbě vlastního řešení.

Server vrací odpovědi ve formátu JSON. Pro získání rozpoznaného textu je z odpovědi potřeba vyextrahovat obsah položky `text`.

6.5 Architektura aplikace pro rozpoznávání složení potravin

Cílem mobilní aplikace pro rozpoznávání složení potravin je pomoci uživatelům ve výběru potravin díky odhalování škodlivých látek. Mezi kroky, které bude muset aplikace pro dosažení tohoto jednoduše formulovaného cíle realizovat, patří pořízení snímků, jejich předzpracování, odeslání na rozpoznávací server, zpracování odpovědí a interpretace a vhodná prezentace složení potravin.

Velkou přidanou hodnotou a odlišností od existujících aplikací bude možnost **kontinuálního rozpoznávání** a vykreslování výsledků do náhledu fotoaparátu. Aplikace tedy nebude striktně rozdělena na obrazovku pro jednorázové pořízení snímku a obrazovku pro následné čekání na výsledky.

Jelikož se nejedná o triviální ani nenáročné kroky, je potřeba vhodně navrhnout architekturu aplikace jak z pohledu logického členění, tak z pohledu prostředků pro dosažení dostatečného výkonu.

Jedním z prostředků pro dosažení vysokého výkonu je nativní vývoj. Volba použití knihovny OpenCV v jazyce C++ v rámci knihovny pro předzpracování je přesně tímto případem. Mobilní aplikaci je tak potřeba na takové předzpracování připravit. Toho bude dosaženo použitím sady **Android Native Development Kit** (NDK), která umožňuje začlenit C++ kód do Android aplikace. Pro kooperaci s tímto kódem pak poslouží rozhraní **Java Native Interface** (JNI) umožňující především volání nativních metod z Android aplikace.

Vhodné logické členění – které se promítne i do oblasti výkonu – lze realizovat na úrovni paralelismu a vícevláknového zpracování. Hlavní vlákno aplikace totiž nelze zatěžovat náročnějšími úkony, aniž by došlo ke snížení responzivity nejen uživatelského rozhraní, ale i aplikace jako celku.

6.5.1 Vícevláknové zpracování

V případě aplikace pro rozpoznávání složení budou vyčleněny speciální prostředky pro:

- správu uživatelského rozhraní a základní obsluhu
- správu náhledu fotoaparátu
- předzpracování obrazu
- síťovou komunikaci

Primární zodpovědností **hlavního vlákna** (*main thread*) je vykonávání systémových funkcí (volání), které jsou součástí životního cyklu aplikace, případně se jedná o zpětná volání umožňující reagovat na systémové či uživatelské události. Dále má hlavní vlákno na starost vykreslování obrazovky, a to ideálně rychlostí 60 snímků za sekundu, tedy zhruba každých 16 ms. Právě z tohoto důvodu je vhodné jakoukoliv práci (nejen) delší než 16 ms realizovat ve vlákně na pozadí.

V případě aktivního náhledu fotoaparátu bude souběžně s hlavním vláknem pracovat i speciálně vyhrazené **vlákno fotoaparátu** (*camera thread*). Jeho zodpovědností bude pořizování snímků a jejich předávání k vykreslení na obrazovku. Pořízené snímky budou nejdříve předávány do systémové funkce, kde nad nimi bude možné provádět libovolné úpravy předtím, než dojde k vykreslení na obrazovku. Doba zpracování však bude přímo určovat zpoždění mezi pořízením snímku a jeho zobrazením v náhledu. Proto bude vhodné do této funkce umístit pouze rychlé zpracování.

Pro déle trvající předzpracování obrazu a detekci textu bude vyhrazeno **vlákno předzpracování** (*prepro thread*). To bude realizováno třídou `HandlerThread`. Půjde tedy o dlouhodobě běžící vlákno, které bude s pomocí objektu třídy `Handler` zpracovávat zprávy (požadavky na práci), které budou vláknu doručeny. Toto vlákno bude skrze rozhraní JNI komunikovat s knihovnou pro předzpracování.

Speciálně bude také vyhrazeno **vlákno síťové komunikace** (*network thread*). To bude zpracovávat požadavky vytvořené v hlavním vlákně a realizovat (blokující) síťovou komunikaci se serverem pro rozpoznávání textu. Odpovědi serveru bude následně předávat zpět hlavnímu vláknu.

6.5.2 Návrh uživatelského rozhraní

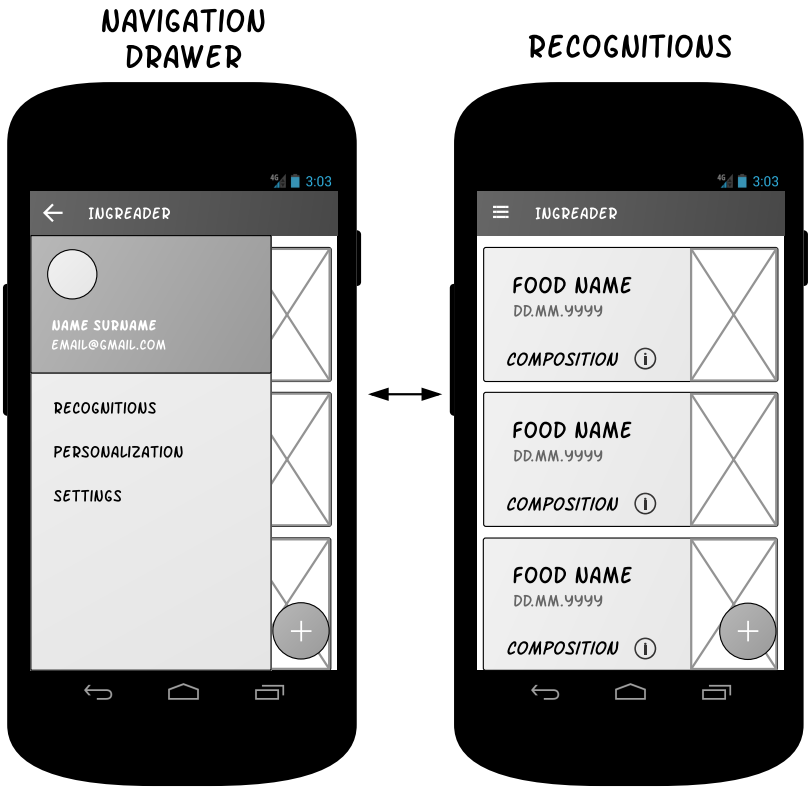
Pro zpřístupnění základní funkcionality aplikace pro rozpoznávání složení potravin není nutné vytvářet spletitou síť obrazovek. Uživatel by si naopak pro ty nejklíčovější činnosti mohl vystačit pouze se dvěma obrazovkami. A sice obrazovkou s přehledem posledních rozpoznávání a obrazovkou s náhledem fotoaparátu umožňující rozpoznávání realizovat.

Volba hlavní, tedy vstupní, obrazovky je pro každou aplikaci rozhodující. Velmi užitečnou filozofií při návrhu rozhraní bývá snaha zpřístupnit uživateli klíčovou funkcionalitu ihned po spuštění aplikace, bez zbytečných kroků navíc. Někdy se jedná o velké tlačítko na vstupní obrazovce přímo aktivující danou funkcionalitu, v odvážnějších případech realizuje užitečnou činnost samotná vstupní obrazovka.

V tomto duchu by se za vstupní obrazovku dala zvolit obrazovka s náhledem fotoaparátu. Nicméně kvůli kontinuálnímu pořizování snímků a jejich odesílání na server by se mohlo jednat o poměrně nepříjemné a nečekané narušení soukromí. Proto je hlavní obrazovka zvolena poněkud tradičněji a bude obsahovat **přehled naposledy provedených rozpoznávání** (*recognitions*). U každého záznamu o rozpoznávání by měl uživatel nalézt pojmenování (např. název potraviny), které si vyplní dle svého uvážení. K jednoduché identifikaci záznamu poslouží také pořízená fotografie potraviny. Dalším užitečným údajem může být datum provedení rozpoznávání. Přestože je jistě vhodné uživateli umožnit výběr konkrétního záznamu a přechod na jeho detail pro bližší informace, může uživateli dobře posloužit také stručná statistika složení (např. nalezených škodlivin) uvedená přímo v přehledu rozpoznávání. S ohledem na rychlý přístup ke klíčové funkcionalitě aplikace se pak na této obrazovce bude nacházet tlačítko vedoucí přímo do náhledu fotoaparátu.

Dalším důležitým aspektem návrhu uživatelského rozhraní je volba navigace v rámci aplikace. Pro nízký počet obrazovek (maximálně tři) jsou na Androidu doporučovaným způsobem navigace tzv. *tabs*, tedy jakési záložky pro rychlé přepínání obrazovek. Tyto záložky bývají napevno umístěny v horní části všech obrazovek. Dvě výše uvedené klíčové obrazovky však nejsou zdaleka vším, co bude aplikaci tvořit. Uživateli by jistě bylo dobré umožnit personalizaci aplikace, tedy specifikaci látek, které má aplikace u potravin hledat. U rozsáhlejší aplikace dává také smysl přihlašovací a profilová obrazovka. Téměř každá aplikace pak najde využití pro obrazovku nastavení, zákaznické podpory či informací

o aplikaci. Pro zpřístupnění tohoto množství obrazovek bude použita navigace pomocí tzv. ***navigation draweru***, tedy „navigačního šuplíku“, který lze gestem vytáhnout ze strany obrazovky a který obsahuje seznam většiny obrazovek aplikace. V *navigation draweru* lze navíc snadno oddělit klíčové obrazovky od obrazovek vedlejších.



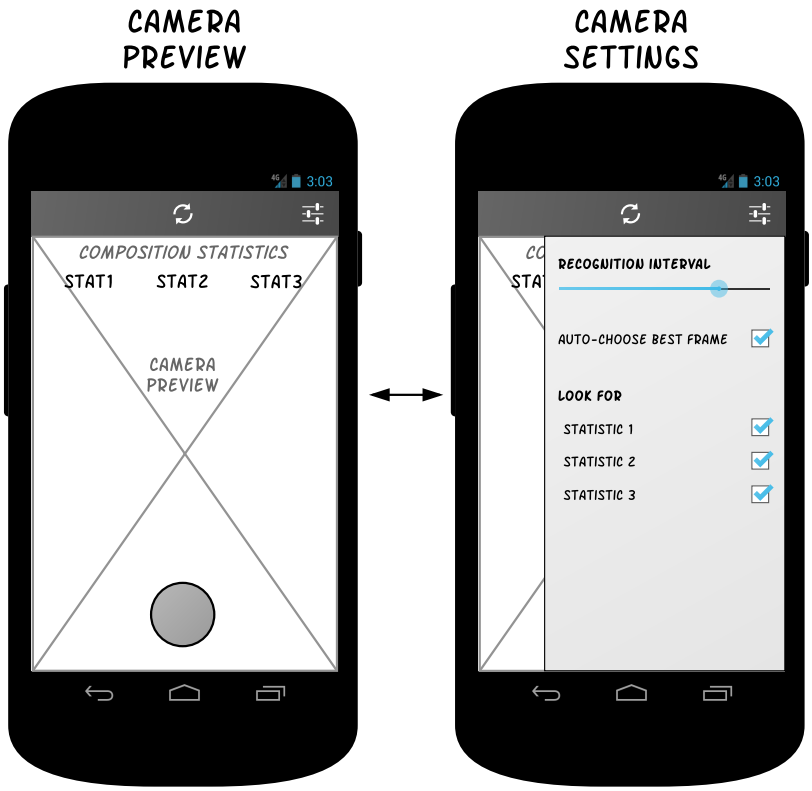
Obrázek 6.8: Navigace a obrazovka s přehledem rozpoznávání

Pro zahájení rozpoznávání stiskne uživatel tlačítko, které bude na hlavní obrazovce vždy k dispozici a které jej přenese přímo do **náhledu fotoaparátu** (*camera preview*), jenž bude připraven pro okamžité rozpoznávání. Obrazovka náhledu fotoaparátu je pro aplikaci zásadní a snaha o kontinuální rozpoznávání dále navyšuje nároky na její rozhraní. Pro rychlý přístup k nejdůležitějším nastavením bude uživateli sloužit panel nástrojů v horní části obrazovky. Zde se může nacházet například ovládání LED diody pro přisvícení ve zhoršených světelných podmínkách. Primárně zde však bude k dispozici přepínač režimu rozpoznávání. Přestože je režim kontinuálního rozpoznávání poměrně netradičním vylepšením, nehodí se do všech situací. Někdy může uživatel chtít šetřit datový limit, jindy kontinuální rozpoznávání jednoduše nemusí být potřeba. Proto panel nástrojů umožní aktivaci režimu pro jednorázové pořízení snímku a jednorázové rozpoznávání. Jedná se navíc o elegantní způsob, jak bude uživatel moci deaktivovat odesílání obrazu na server, a tedy chránit své soukromí. Při jednorázovém rozpoznávání pak přijde vhod tlačítko spouště fotoaparátu v dolní části obrazovky.

U rozpoznávání však může uživatel chtít nastavit celou řadu dalších parametrů a větší počet ovládacích prvků zakrývajících náhled může degradovat nejen viditelnost, ale i celkový uživatelský zážitek. Vedlejší ovládací prvky proto budou umístěny v pomocném **panelu nastavení** (*camera settings*), který půjde gestem vysunout z pravé strany obrazovky. Tyto

prvky tak za normálních okolností nebudou překážet, ale v případě potřeby budou k dispozici v přehledné formě. Mezi tato nastavení může patřit interval rozpoznávání či specifikace hledaných látek.

To, co uživatel očekává nejvíce, jsou ovšem výsledky rozpoznávání. Ty budou v podobě kompaktní statistiky zobrazeny přímo v horní části obrazovky a budou průběžně aktualizovány, např. když bude uživatel muset pro nasnímání celého obalu pořídit více snímků. Dotykem této statistiky si pak uživatel zobrazí podrobnější **dialog specifikující nalezené látky** doplněné o další vysvětlující informace.



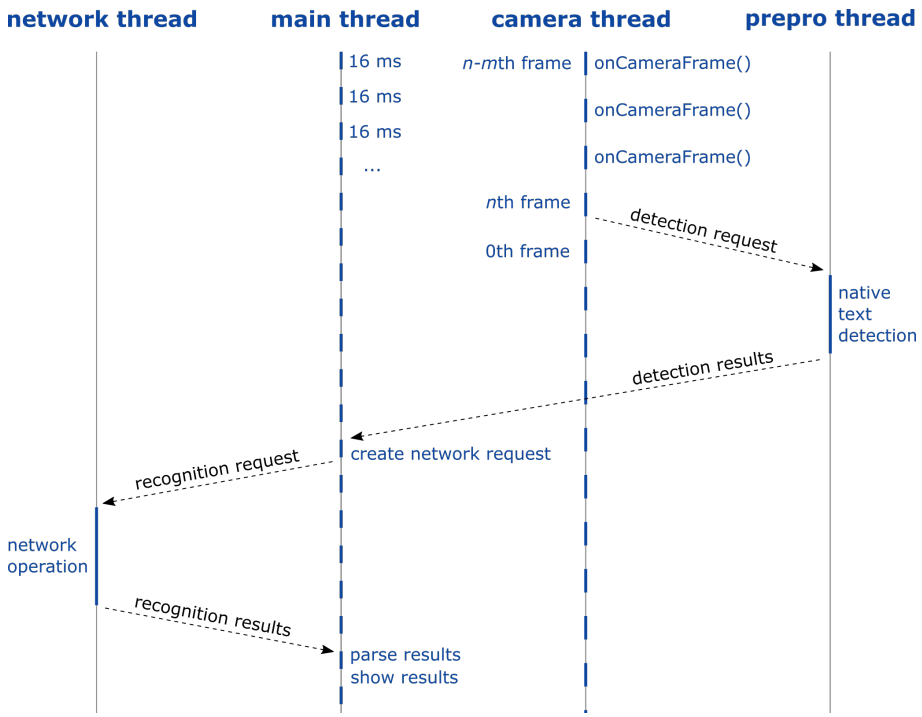
Obrázek 6.9: Náhled fotoaparátu a panel nastavení

6.6 Výsledná aplikace pro rozpoznávání složení potravin

Tato podkapitola poskytuje popis funkčního principu a přehled významných charakteristik výsledné aplikace pro rozpoznávání složení potravin.

6.6.1 Tok událostí uvnitř aplikace

Po spuštění aplikace je aktivní pouze hlavní vlákno. To běží kontinuálně a nezávisle na případných ostatních vláknech aplikace. Při přechodu na obrazovku s náhledem fotoaparátu je dále spuštěno vlákno fotoaparátu a vlákno předzpracování. Zatímco vlákno fotoaparátu pravidelně pořizuje snímky, vlákno předzpracování pouze čeká na požadavek na práci.



Obrázek 6.10: Tok událostí uvnitř aplikace

Cílem vlákna fotoaparátu je každý n-tý (např. 60.) snímek spustit předzpracování. S předstihem m (např. 5) snímků začne vlákno ve funkci `onCameraFrame` na snímky aplikovat vybraný operátor měření ostrosti se snahou najít mezi těmito snímky ten nejostřejší. Skrze rozhraní JNI je tedy využívána knihovna pro předzpracování. Při dosažení každého n-tého snímku je vytvořena kopie nejostřejšího snímku a ta je v požadavku na detekci textu odeslána do vlákna předzpracování. Další předzpracování snímků je dočasně pozastaveno.

Vlákno předzpracování obdrží požadavek a skrze rozhraní JNI aktivuje nativní detekci textu v knihovně pro předzpracování. Po aplikaci pomocných algoritmů (viz. podkapitola 6.3.5) je konečným výsledkem detekce textu buď jediný obrázek obsahující pouze textové části, nebo popis textových oblastí. Ve druhém případě dojde k vytvoření podobrázků s jednotlivými textovými částmi, které jsou odeslány do hlavního vlákna.

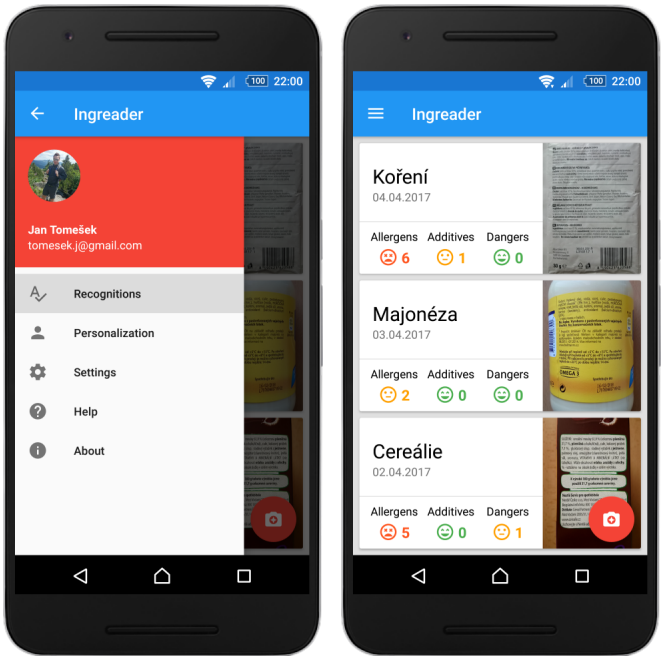
`Handler` hlavního vlákna obdrží výsledky z vlákna předzpracování. Následně sestaví dotaz pro rozpoznávací server a předá jej vláknu síťové komunikace, které komunikaci se serverem uskuteční. Po obdržení odpovědi vlákno provede prvotní zpracování a rozpoznaný text předá hlavnímu vláknu.

Hlavní vlákno následně rozpoznaný text analyzuje a vyhledá v něm škodlivé látky. Jelikož se počet běžně se vyskytujících škodlivin pohybuje pouze v řádu desítek, jsou hledané látky uloženy jako zdroje (*resources*) aplikace a dynamicky načteny při startu aplikace. Tento přístup umožňuje velmi rychlé prohledávání. Větší počet hledaných látek by ale pravděpodobně bylo vhodnější uchovávat v databázi. Hlavní vlákno vytvoří seznam(y) nalezených škodlivin a zajistí vykreslení výsledků do uživatelského rozhraní. Na závěr je znovu povoleno další předzpracování snímků.

6.6.2 Výsledná aplikace v kontextu platformy Android

Mobilní aplikace je vytvořena nejen s ohledem na své zaměření, ale také v souladu s mnoha návrhovými vzory a doporučenými postupy pro Android aplikace definovanými mateřskou společností Google.

Aplikace je implementována s důrazem na znovupoužitelnost nejen z pohledu kódu, ale i z pohledu použití komponent uvnitř aplikace. V současné době např. neexistuje snadný způsob, jak automaticky integrovat *navigation drawer* do všech obrazovek. Pro tyto účely byla vytvořena nová bázová třída, kterou mohou rozšířit všechny obrazovky (aktivity) aplikace, následkem čehož mají tyto obrazovky okamžitě k dispozici tuto navigaci. Obrazovky navíc nejsou realizovány přímo aktivitami, ale **fragmenty**, které jsou do aktivit zabaleny. Zatímco jedna obrazovka vždy odpovídá jedné aktivitě, u fragmentů toto neplatí a jedna obrazovka (jedna aktivita) může obsahovat více fragmentů. Tímto způsobem lze lépe zužitkovat prostor větších displejů (např. na tabletech), kde lze na jednu obrazovku umístit více fragmentů, které by v mobilním telefonu představovaly oddělené obrazovky. V konečném důsledku tak fragmenty zajišťují silnou znovupoužitelnost a umožňují přizpůsobení rozhraní napříč zařízeními, obzvláště při uvážení možnosti jejich přidávání za běhu aplikace.



Obrázek 6.11: Výsledná navigace a obrazovka s přehledem rozpoznávání

Aktuálně používaným přístupem pro ochranu soukromí uživatele jsou na Androidu tzv. *runtime permissions*, tedy **oprávnění udělovaná za běhu** aplikace. Tento přístup nabízí pro uživatele mnoho výhod. Použití aplikace již není dáno přístupem „všechno nebo nic“ jako dříve, kdy uživatel musel pro instalaci aplikace povolit všechna oprávnění, jinak k instalaci vůbec nedošlo. Nyní jsou oprávnění vyžadována jednotlivě za běhu, ideálně až při snaze o přístup k související funkcionalitě (např. přístup k fotoaparátu), což uživateli přímo objasňuje, nač je oprávnění potřeba, a vzbuzuje v něm větší důvěru. Uživatel má navíc granulární kontrolu nad oprávněními aplikace a jednotlivá oprávnění může rušit dle libosti. Pro vývojáře je tento přístup naopak poměrně náročný. Nejzásadnější změnou je, že musí předpokládat nepřidělení oprávnění a v rámci možností zajistit běh aplikace i bez

nich. Navíc musí vhodně naplánovat vyžádání jednotlivých oprávnění a implementovat reakce na jejich povolení i zamítnutí. Mobilní aplikace pro rozpoznávání složení potravin tato dynamická oprávnění beze zbytku implementuje.

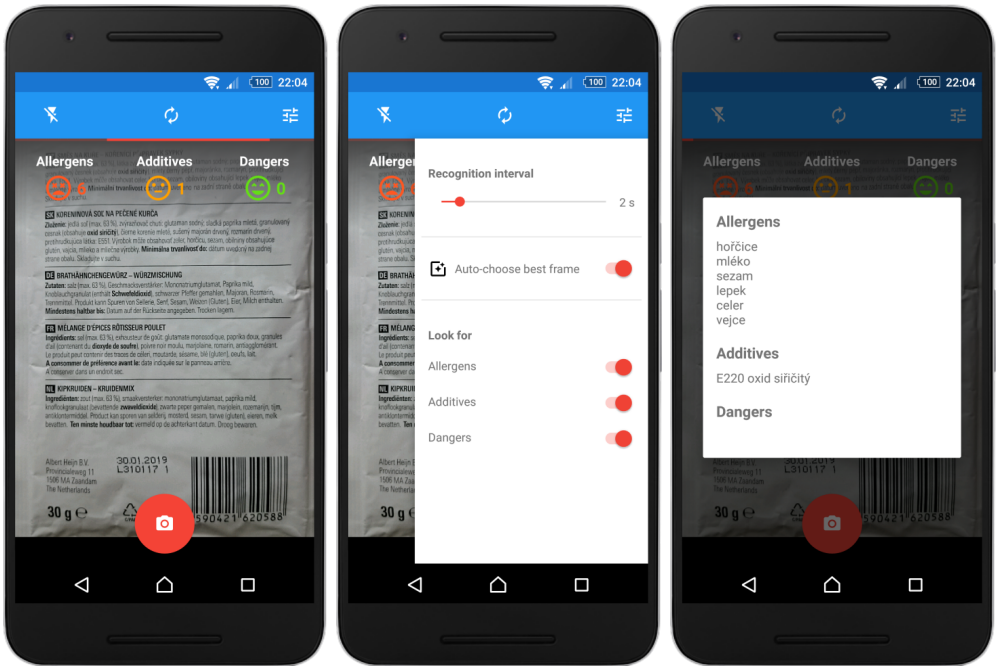
V oblasti designu se pak aplikace řídí zásadami **Material Designu**, které jsou pro Android výchozí. Rozvržení rozhraní, jeho komponenty (plovoucí tlačítka, karty, jejich vyvýšení a stíny), barevná paleta i ikonografie jsou zvoleny v souladu právě s těmito zásadami.

Vlastní obrazovka náhledu fotoaparátu je dle návrhu implementována na míru aplikaci a je založena na komponentě z knihovny pro předzpracování, která umožňuje použití rozhraní v orientaci na výšku.

Jako alternativa k „softwarovým“ metrikám ostrosti jsou v aplikaci využity také **pohybové senzory** zařízení, které umožňují detekovat otřesy zařízení a odhadnout rozmazání pořízeného snímku.

Aplikace cílí na Android zařízení s verzí platformy 15 a vyšší. Tedy na všechna zařízení s Androidem 4.0.3 IceCreamSandwich a novějším, což v okamžiku psaní této práce představuje **97,4 % zařízení** aktivních na Google Play Store.

Z pohledu přínosu uživateli pak výsledná mobilní aplikace dokáže v potravinách vyhledávat 3 kategorie škodlivin. První, asi nejdůležitější, skupinou jsou **alergeny**. Kromě základních 14 alergenů hledá aplikace i plodiny, ve kterých se tyto alergeny nacházejí. Druhou kategorií jsou tzv. „éčka“ či **aditiva**. Jedná se o barviva, konzervanty, emulgátory a další přídatné látky o různé míře škodlivosti. Aplikace pak upozorňuje na ty, které jsou definovány jako nebezpečné či dokonce nepovolené. Aditiva navíc dokáže najít jak podle jejich kódového označení, tak podle standardního názvu. Poslední kategorie obsahuje látky, které mohou být zdraví **nebezpečné** a nespadají do žádné z předchozích kategorií. Typickým příkladem je palmový olej.



Obrázek 6.12: Výsledný náhled fotoaparátu, panel nastavení a dialog s nalezenými látkami

V souladu s návrhem uživatelského rozhraní jsou tyto 3 kategorie uváděny ve statistikách, a to jak v přehledu rozpoznávání, tak v samotném náhledu fotoaparátu. U každé kategorie je vždy uveden počet nalezených látek. Pro rychlou orientaci je v závislosti na míře nebezpečí použito také vhodné barevné odlišení a doplňující grafika.

Kapitola 7

Testování mobilní knihovny a mobilní aplikace

Cílem této kapitoly je podat základní přehled o vlastnostech vytvořeného řešení a navrhnout postup pro případné důkladnější testování.

Testování je rozděleno na oblast hodnocení kvality obrazu (operátorů měření ostrosti), oblast detekce textu a celkové vlastnosti mobilní aplikace. Primární cílem je prozkoumat rychlost provedení a přínos jednotlivých částí zpracování.

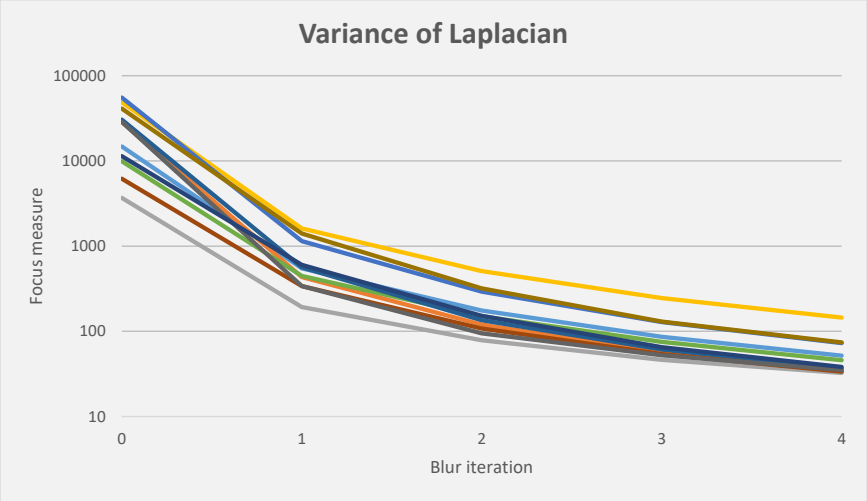
Pro testování byla speciálně vytvořena menší datová sada 10 snímků obsahujících obaly různorodých potravin. Rozlišení snímků činí 720×960 pixelů. Jedná se o stejné rozlišení, v jakém snímky pořizuje příkladová aplikace. Sada obsahuje snímky s velkým množstvím textu i snímky, ve kterých text tvoří menší část celkové plochy. Jelikož však snímky obsahují obaly celé a uživatelé v praxi spíše nasnímají pouze vybranou část obalu, lze naměřené hodnoty považovat za hraniční a při reálném použití často očekávat výsledky lepší.

Aby byly výsledky více vypovídající, byla některá měření prováděna na dvou různých zařízeních. Pro referenci a jednodušší srovnání byla měření realizována na počítači s plnohodnotným procesorem Intel Core i7-5500U s taktem 2,40 GHz. Naopak pro lepší ilustraci reálného použití byla měření prováděna na mobilním zařízení s procesorem Qualcomm Snapdragon 801 s taktem 2,3 GHz, a to přímo v rámci příkladové aplikace, kde výkon ovlivňují i jiné faktory jako další vlákna aplikace (fotoaparát, síťová komunikace) či ostatní aplikace na pozadí systému. Je však potřeba poznamenat, že se jednalo o starší zařízení (procesor) z roku 2014 a současná zařízení by velmi pravděpodobně dosáhla ještě lepších výsledků.

7.1 Testování hodnocení kvality obrazu

V této části byla nejprve vyhodnocena spolehlivost jednotlivých operátorů měření ostrosti a následně měřena doba jejich provedení.

Speciálně pro testování **spolehlivosti operátorů**, které mohou být použity i pro snímky s jiným obsahem, než kterým je text, byla vytvořena samostatná sada 11 obrázků, z nichž 6 obsahuje různé druhy scén (příroda, město, voda, ...) a dalších 5 s ohledem na zamýšlené využití operátorů představuje snímky obalů potravin s textem. Rozlišení obrázků se pohybuje okolo HD rozlišení. Na každý obrázek byly aplikovány všechny 4 implementované operátory, a to celkem 5krát. Nejdříve na originální podobu obrázku, následně na jeho čím dál více rozmazané verze po aplikaci Gaussova rozmazání s velikostí jádra 7×7.



Obrázek 7.1: Příklad měření spolehlivosti operátoru *variance of Laplacian*

Grafy A.1 až A.4, které jsou součástí příloh, ukazují hodnoty vrácené jednotlivými operátory při aplikaci na všech 11 obrázků. Aby šlo (v mezích této práce) označit jednotlivé metriky (operátory) za spolehlivé, musí všechny křivky s přibývajícím rozmazáním vykazovat sestupnou tendenci. Lze pozorovat, že operátory *Tenengrad variance*, *variance of Laplacian* a *gray-level variance* toto kritérium zcela splňují. Je však potřeba poznamenat, že operátor *gray-level variance* má rozdíl mezi jednotlivými hodnotami poměrně nízký. Na druhou stranu operátor *histogram range* za spolehlivý označit nelze. Zatímco v některých případech pracuje správně, v jiných vykazuje dokonce naprosto opačnou (rostoucí) tendenci při zvyšujícím se rozmazání.

Dále byla měřena **doba provedení** jednotlivých operátorů. Pro toto měření, stejně jako všechna následující, již byla použita hlavní sada snímků s obaly potravin zmíněná v úvodu kapitoly. Pro referenci byla doba provedení nejdříve měřena na počítači s plnohodnotným procesorem.

Tenengrad variance	123,77 ms
variance of Laplacian	33,19 ms
gray-level variance	0,21 ms
histogram range	0,64 ms

Tabulka 7.1: Doba provedení operátorů měření ostrosti na referenčním počítači

Naopak pro představu o reálné použitelnosti bylo stejné měření provedeno i na mobilním zařízení v rámci příkladové aplikace.

Tenengrad variance	151,99 ms
variance of Laplacian	25,42 ms
gray-level variance	1,49 ms
histogram range	3,06 ms

Tabulka 7.2: Doba provedení operátorů měření ostrosti v rámci mobilní aplikace

Informace získané z testování spolehlivosti i doby provedení naznačují trend, kdy s klesajícím časem provedení klesá také spolehlivost operátorů.

Pro příkladovou aplikaci (a další měření) byl nakonec vybrán operátor ***variance of Laplacian***, který nabízí dostatečnou spolehlivost a přijatelnou dobu provedení i pro kontinuální zpracování snímků z náhledu fotoaparátu.

7.2 Testování detekce textu

V této části byl nejprve zkoumán **procentuální poměr plochy** detekovaných textových oblastí ku celkové ploše snímku. Tento poměr je samozřejmě dán především množstvím textu v originálním obrázku, cílem ovšem bylo demonstrovat, jak velká část snímku nemusí být v průměru díky detekci přenášena na server a zbytečně rozpoznávána. Nemluvě o vylepšení přesnosti rozpoznávání díky vyloučení netextových částí, ve kterých by mohl být chybně rozpoznán „text“.

základní detekce	50,2 %
SWT detekce	33,4 %

Tabulka 7.3: Poměr plochy detekovaných textových oblastí ku celkové ploše snímku

Informace uvedené v tabulce lze také interpretovat jako průměrnou redukci zpracovávané plochy o 49,8 % v případě základní detekce a redukci o 66,6 % v případě detekce založené na SWT. Lepší výsledky detekce založené na SWT jsou dány její operací na úrovni písmen namísto celých bloků textu, díky čemuž detekované oblasti lépe přiléhají k textu a neobsahují téměř žádné prázdné plochy.

Zároveň byla měřena **doba provedení** obou algoritmů detekce. Toto měření bylo provedeno na referenčním počítači.

základní detekce	119,68 ms
SWT detekce	1 295,44 ms

Tabulka 7.4: Doba provedení detekce textu na referenčním počítači

Jelikož pro účely rozpoznávání složení potravin nabízí obě detekce dostatečnou přesnost i redukci plochy, byla pro příkladovou aplikaci (a zbylá měření) vybrána **základní detekce**, jejíž rychlé provedení je pro kontinuální zpracování velkou výhodou.

Přínosné by také jistě bylo prozkoumání přesnosti rozpoznávání. Takové testování by ovšem vyžadovalo mnohem komplexnější přístup a zapadá spíše do práce kolegy, který se serveru pro rozpoznávání věnuje. Možný postupem by bylo vytvoření ručních přepisů textu ze snímků v testovací sadě a následné srovnání s výstupy rozpoznávacího software.

7.3 Celkové vlastnosti mobilní aplikace

Tato část se zabývala vlastnostmi aplikace jako celku. Nejdříve byla analyzována **celková doba provedení** jednoho cyklu rozpoznávání složení od pořízení snímku přes rozpoznávání na serveru až po vykreslení výsledků.

Následující tabulka uvádí průměrnou dobu cyklu rozpoznávání včetně doby detekce při běhu na mobilním zařízení v rámci příkladové aplikace.

základní detekce	290,93 ms
celkový čas	8 683,83 ms

Tabulka 7.5: Průměrná doba jednoho cyklu rozpoznávání v rámci mobilní aplikace

Z doby 8,6 sekund zabere drtivou většinu času rozpoznávání textu na serveru mého kolegy. Veškeré zbývající činnosti související s aplikací, včetně síťové komunikace, zabírají zhruba 1 sekundu.

Na tuto dobu rozpoznávání lze nahlížet různě. Ve srovnání s existujícími aplikacemi se jedná o poměrně slušný výsledek, z pohledu uživatelského zážitku to však stále není ideální. Je však potřeba připomenout, že testovací sada obsahuje snímky celých obalů potravin, zatímco v praxi uživatel pravděpodobně zaměří pozornost pouze na konkrétní část obalu, kdy lze v kombinaci s předzpracováním snížit dobu rozpoznávání na zhruba 2 sekundy.

Současně byl zkoumán **pokles v počtu snímků za sekundu** (FPS), ke kterému došlo v náhledu fotoaparátu kvůli celému procesu předzpracování. V průměru došlo k **poklesu o 5,3 snímku za sekundu**, což lze považovat za přijatelné.

Finálně bylo zkoumáno **zmenšení objemu dat** přenášených na server díky detekci textu, do kterého se promítají i vlastnosti použité komprese. Následující tabulka uvádí průměrnou velikost snímku bez předzpracování a průměrnou velikost textových částí po aplikaci základní detekce. Vzhledem k tomu, že je v příkladové aplikaci před odesláním dat použita také komprese JPEG s nastavenou kvalitou 80 %, je uvedena také tato hodnota.

originální velikost	1 024 455,5 B
textové části (základní detekce)	641 823,1 B
textové části po JPEG kompresi	97 054,4 B

Tabulka 7.6: Zmenšení datového objemu díky předzpracování

Následující tabulka vyjadřuje stejné údaje procentuálně.

originální velikost	100 %
textové části (základní detekce)	62,13 %
textové části po JPEG kompresi	9,38 %

Tabulka 7.7: Zmenšení datového objemu díky předzpracování

Jinými slovy zajišťuje základní detekce textu redukci množství dat o 37,87 %, celkově s využitím komprese se pak v průměru jedná o **redukci 90,62 % datového objemu**.

Kapitola 8

Závěr

Cílem této práce bylo vytvořit mobilní knihovnu pro předzpracování obrazu s textem jakožto významnou součást systému pro rozpoznávání textu určeného pro chytré telefony. Nad tímto týmově budovaným systémem měla být dále vystavěna mobilní aplikace umožňující analýzu složení uváděného na obalech potravin. Oba tyto cíle byly v práci naplněny.

Vytvořená mobilní knihovna obsahuje rozmanité algoritmy zpracování obrazu. Kromě základních úprav obrazu nabízí možnosti hodnocení kvality pořízených snímků a především algoritmy detekce textu, které významně snižují datový objem a zrychlují a zpřesňují rozpoznávání textu. Uvedené algoritmy jsou k dispozici ve více variantách pro možnost výběru nejvhodnější varianty pro zvolený účel. Tyto hlavní algoritmy jsou rozšířeny o algoritmy pomocné, které dokáží odhadnout barvu textu, různorodě zužitkovat detekované textové oblasti či snímek vyrovnat. Algoritmy jsou navíc doplněny o platformě specifické komponenty, které mobilním aplikacím velmi usnadňují celý proces rozpoznávání textu.

Aplikace pro rozpoznávání složení potravin je vybudována s důrazem na efektivitu i užitek – je technicky vyspělá, zahrnuje vícevláknové zpracování a využívá možností nativního vývoje. Její velkou přidanou hodnotou je vlastní obrazovka rozhraní fotoaparátu, která umožňuje kontinuální rozpoznávání složení včetně vykreslování výsledků do náhledu a nabízí řadu nastavení. Uživatele dokáže upozornit na přítomnost alergenů, aditiv či jiných škodlivých látek. Navíc aplikace dodržuje implementační i designové zvyklosti platformy Android.

Práce byla realizována v plném souladu se zadáním. V rámci studia byla provedena analýza současného stavu existujících řešení a dostupných technologií (2) v oblasti předzpracování obrazu a rozpoznávání textu, a to na mnoha úrovních. Velký prostor byl věnován také algoritmům předzpracování obrazu (3) v čele s detekcí textu a vývoji aplikací pro platformu Android (4).

Byl proveden návrh knihovny pro předzpracování obrazu s textem (6.2) z pohledu výběru technologií, algoritmů i dalších komponent. Součástí návrhu bylo také zhodnocení možností a dosažitelných výsledků, které byly zohledněny především na úrovni obrazových algoritmů, kde bylo zvoleno více algoritmů o různých vlastnostech pro uspokojení rozmanitých potřeb. Knihovna byla následně implementována (6.3) dle vytvořeného návrhu. Velké úsilí bylo také vynaloženo na návrh (6.5) a tvorbu (6.6) příkladové aplikace pro rozpoznávání složení potravin. Mobilní knihovna i příkladová aplikace byly na závěr testovány (7) v oblasti spolehlivosti, rychlosti i přínosu a získané výsledky byly diskutovány.

Možnosti dalšího pokračování práce vidím ve dvou hlavních směrech. Mobilní knihovnu by jistě bylo možné obohatit o další užitečné algoritmy, například z oblasti vylepšení kvality obrazu, jako jsou zaostření či odstranění šumu. Aplikaci pro rozpoznávání složení potravin by zase bylo možné rozšířit o detailní personalizaci a sociální prvky a následně vydat jako plnohodnotnou aplikaci, potažmo službu.

V důsledku všech těchto skutečností jsem přesvědčen, že výsledky této práce mohou představovat reálný přínos jak pro vývojáře mobilních aplikací, tak pro uživatele se zájmem o své zdraví.

Literatura

[1] *ABBYY Cloud OCR SDK*. ABBYY Cloud OCR SDK, 2016, [Online; navštíveno 21.12.2016].
URL http://ocrsdk.com/

[2] *ABBYY Mobile OCR Engine*. ABBYY, 2016, [Online; navštíveno 21.12.2016].
URL https://www.abbyy.com/en-ee/mobile-ocr/

[3] *Bytedeco*. Bytedeco, 2016, [Online; navštíveno 22.12.2016].
URL http://bytedeco.org/

[4] *Cloud Vision API*. Google Cloud Platform, 2016, [Online; navštíveno 21.12.2016].
URL https://cloud.google.com/vision/

[5] *FastCV Computer Vision SDK*. Qualcomm Developer Network, 2016, [Online; navštíveno 23.12.2016].
URL https://developer.qualcomm.com/software/fastcv-sdk

[6] *Java interface to OpenCV and more*. GitHub, 2016, [Online; navštíveno 23.12.2016].
URL https://github.com/bytedeco/javacv

[7] *Mobile Vision*. Google Developers, 2016, [Online; navštíveno 20.12.2016].
URL https://developers.google.com/vision/

[8] *OpenCV*. OpenCV, 2016, [Online; navštíveno 22.12.2016].
URL http://opencv.org/

[9] *RenderScript*. Android Developers, 2016, [Online; navštíveno 23.12.2016].
URL https://developer.android.com/guide/topics/renderscript/compute.html

[10] *Writing Native Android Code: NDK vs. RenderScript*. Developer.com, 2016, [Online; navštíveno 23.12.2016].
URL http://www.developer.com/ws/android/development-tools/writing-native-android-code-ndk-vs.-renderscript.html

[11] *Android NDK*. Android Developers, 2017, [Online; navštíveno 16.03.2017].
URL https://developer.android.com/ndk/index.html

[12] *Binarize image by thresholding*. MathWorks, 2017, [Online; navštíveno 16.03.2017].
URL https://www.mathworks.com/help/images/ref/imbinarize.html

[13] *Eroding and Dilating*. OpenCV, Březen 2017, [Online; navštíveno 16.03.2017].
URL http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

[14] *IntentService*. Android Developers, 2017, [Online; navštíveno 16.03.2017].
URL
https://developer.android.com/reference/android/app/IntentService.html

[15] *More Morphology Transformations*. OpenCV, Březen 2017, [Online; navštíveno 16.03.2017].
URL http://docs.opencv.org/2.4/doc/tutorials/imgproc/opening_closing_hats/opening_closing_hats.html

[16] *Platform Architecture*. Android Developers, 2017, [Online; navštíveno 16.03.2017].
URL https://developer.android.com/guide/platform/index.html

[17] *Processes and Threads*. Android Developers, 2017, [Online; navštíveno 16.03.2017].
URL https://developer.android.com/guide/components/processes-and-threads.html

[18] *ThreadPoolExecutor*. Android Developers, 2017, [Online; navštíveno 16.03.2017].
URL https://developer.android.com/reference/java/util/concurrent/ThreadPoolExecutor.html

[19] Allen, G.: *Android 4*. Brno: Computer Press, 2013, ISBN 978-80-251-3782-6.

[20] Bobák, P.: *Mobilní systém pro rozpoznání textu na iOS*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, 2017, vedoucí práce Zemčík Pavel.

[21] Bradski, G.; Kaehler, A.: *Learning OpenCV*. Sebastopol: O'Reilly Media, Inc., 2008, ISBN 978-0-596-51613-0.

[22] Epshtein, B.: *Detecting Text in Natural Scenes with Stroke Width Transform*. 23rd IEEE Conference on Computer Vision and Pattern Recognition 2010, San Francisco, Červenec 2010, [Online; navštíveno 04.03.2017].
URL http://videolectures.net/cvpr2010_epshtein_dtns/

[23] Epshtein, B.; Ofek, E.; Wexler, Y.: Detecting Text in Natural Scenes with Stroke Width Transform. *CVPR 2010, IEEE International Conference on Computer Vision and Pattern Recognition 2010*, San Francisco, Červen 2010, doi:10.1109/CVPR.2010.5540041, [Online; navštíveno 04.03.2017].
URL https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/1509.pdf

[24] Gonzalez, R. C.; Woods, R. E.: *Digital Image Processing*. Upper Saddle River: Prentice Hall, druhé vydání, 2002, ISBN 0-201-18075-8.

[25] Laganière, R.: *OpenCV 2 Computer Vision Application Programming Cookbook*. Birmingham: Packt Publishing Ltd., 2011, ISBN 978-1-849513-24-1.

[26] Mednieks, Z.; Dornin, L.; Meike, G. B.; aj.: *Programming Android*. Sebastopol: O'Reilly Media, Inc., 2011, ISBN 978-1-449-38969-7.

[27] Meerwald, P.: *Digital Watermarking*. Salzburg: Universität Salzburg, Prosinec 2005, [Online; navštíveno 16.03.2017].
URL https://www.cosy.sbg.ac.at/~pmeerw/Watermarking/

[28] Mir, H.; Xu, P.; van Beek, P.: *An extensive empirical evaluation of focus measures for digital photography*. Waterloo: University of Waterloo, 2014, doi:10.1117/12.2042350, [Online; navštíveno 12.03.2017].
URL https://cs.uwaterloo.ca/~vanbeek/Publications/spie2014.pdf

[29] Pech-Pacheco; Cristóbal; Chamorro-Martínez; aj.: *Diatom autofocusing in brightfield microscopy: a comparative study*. Madrid: Instituto de Optica, 2000, doi:10.1109/ICPR.2000.903548, [Online; navštíveno 12.03.2017].
URL http://optica.csic.es/papers/icpr2k.pdf

[30] Pertuz, S.; Puig, D.; Garcia, M. A.: Analysis of focus measure operators for shape-from-focus. *Pattern Recognition*, 46(5), 2013: s. 1415–1432, ISSN 0031-3203, doi:10.1016/j.patcog.2012.11.011, [Online; navštíveno 12.03.2017].
URL https://www.researchgate.net/publication/234073157_Analysis_of_focus_measure_operators_in_shape-from-focus

[31] Podobny, Z.: *Tesseract Open Source OCR Engine*. GitHub, 2016, [Online; navštíveno 20.12.2016].
URL https://github.com/tesseract-ocr/tesseract

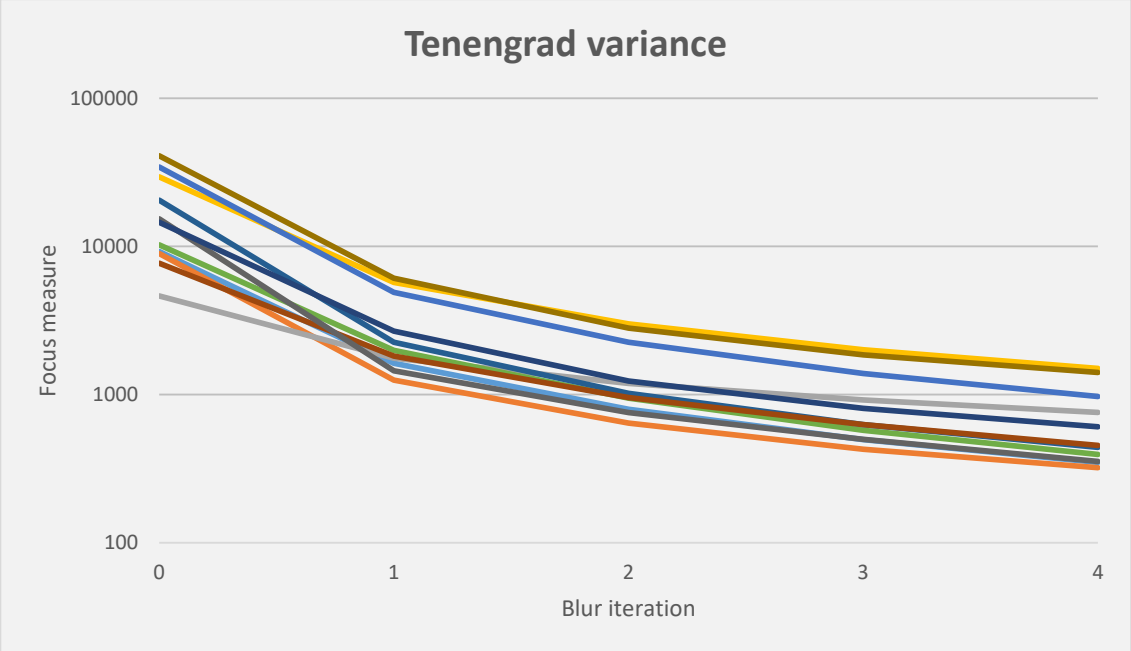
[32] Ujbányai, M.: *Programujeme pro Android*. Praha: Grada Publishing, a.s., 2012, ISBN 978-80-247-3995-3.

[33] Zhou, G.; Liu, Y.: Scene text detection based on probability map and hierarchical model. *Optical Engineering*, 51(6), Červen 2012, doi:10.1117/1.OE.51.6.067204, [Online; navštíveno 16.03.2017].
URL http://opticalengineering.spiedigitallibrary.org/article.aspx?articleid=1306651

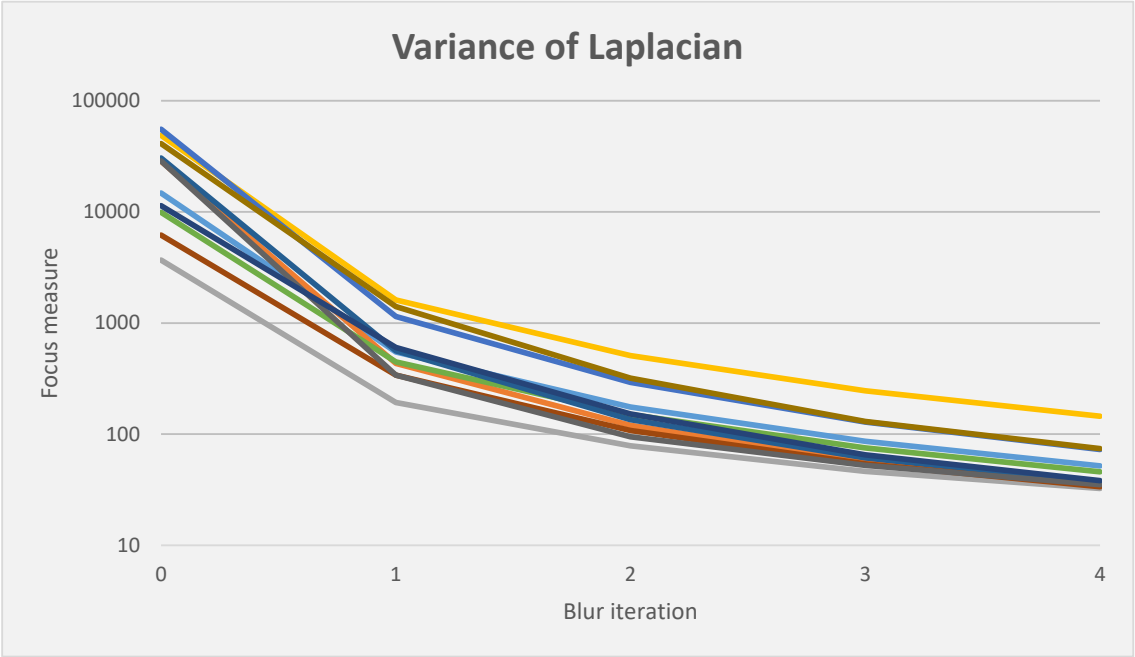
Přílohy

Příloha A

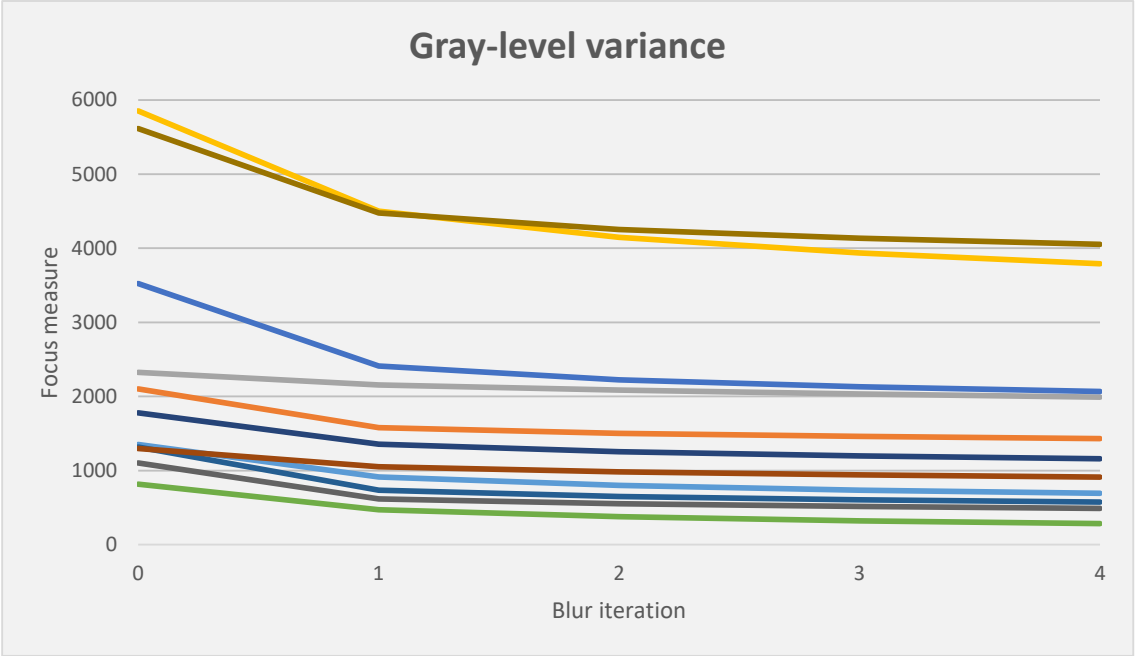
Spolehlivost operátorů měření ostrosti



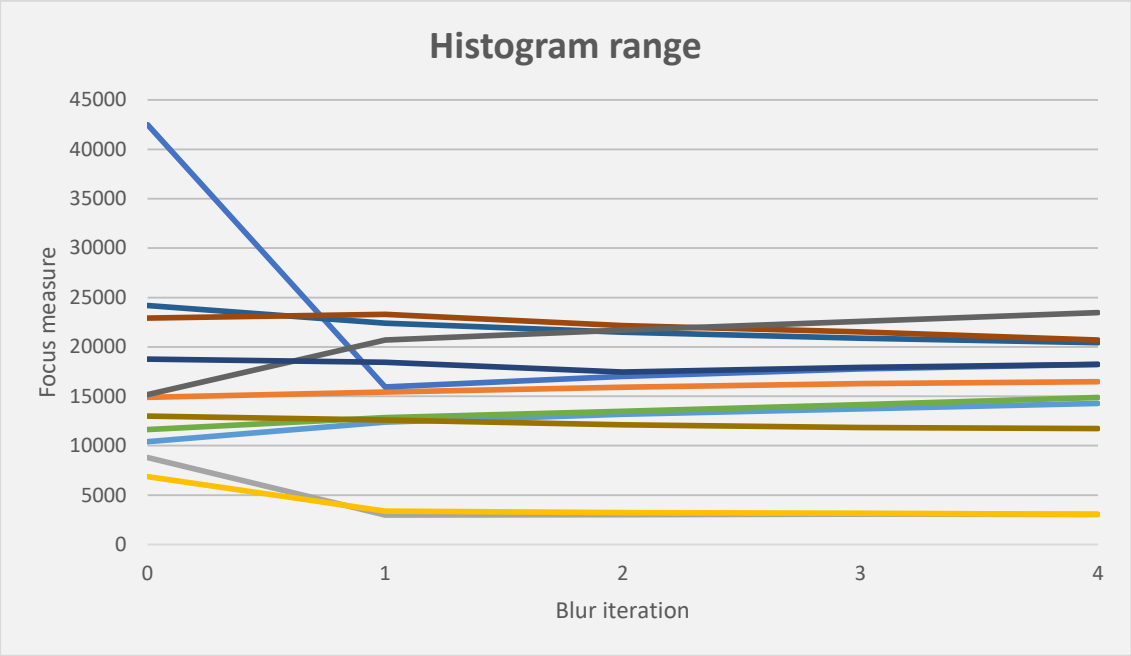
Obrázek A.1: Spolehlivost operátoru *Tenengrad variance*



Obrázek A.2: Spolehlivost operátoru *variance of Laplacian*



Obrázek A.3: Spolehlivost operátoru *gray-level variance*



Obrázek A.4: Spolehlivost operátoru *histogram range*