VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PŘEKLAD TEXTU V REÁLNÉM ČASE S VYUŽITÍM MOBILNÍHO ZAŘÍZENÍ

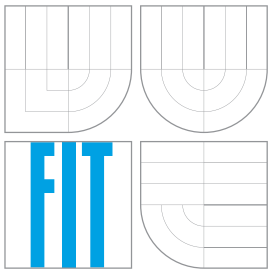
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LUKÁŠ SZTEFEK

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PŘEKLAD TEXTU V REÁLNÉM ČASE S VYUŽITÍM MOBILNÍHO ZAŘÍZENÍ

REAL-TIME TEXT TRANSLATION WITH USING MOBILE DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LUKÁŠ SZTEFEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN SAMEK, Ph.D.

BRNO 2014

Abstrakt

Cílem této práce je návrh a implementace mobilní aplikace pro operační systém Android, která bude sloužit jako real-time překladač textu v obraze z jednoho jazyka do druhého, přičemž musí podporovat několik světových jazyků a češtinu. Text přečtený z obrazu je přeložen a následně nahradí původní takovým způsobem, aby mu byl po vizuální stránce co nejblíže. Čtenář postupně získává povědomí o oblastech získávání textu z obrazových dat, překladu takto získaných dat, jejich zobrazení na mobilním zařízení a implementace v Android OS. V závěru práce se nachází popis experimentů s různými typy obrazových dat a porovnání s konkurenčními aplikacemi.

Abstract

The aim of this thesis is design and implementation of Android application which will serve as real-time translator from one language to another, including several world languages and Czech. The text obtained from image is translated and replaces the original one with keeping its visual look as much as possible. The reader gradually gets familiar with getting text from images, text translation, displaying text on mobile device and Android OS implementation. In conclusion, the experiments with multiple input images and comparison of existings applications have been done.

Klíčová slova

Android OS, OCR, překlad, OpenCV, mobilní zařízení.

Keywords

Android OS, OCR, translation, OpenCV, mobile device.

Citace

Lukáš Sztefek: Překlad textu v reálném čase s využitím mobilního zařízení, diplomová práce, Brno, FIT VUT v Brně, 2014

Překlad textu v reálném čase s využitím mobilního zařízení

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jana Samka Ph. D. Všechny zdroje, ze kterých jsem čerpal, jsem uvedl v seznamu použité literatury.

.....
Lukáš Sztefek
27. května 2014

Poděkování

Děkuji vedoucímu mé práce Ing. Janu Samkovi Ph.D. za ochotu přizpůsobit zadání práce mým přáním, za možnost pravidelných konzultací a za odborné rady a motivaci v průběhu řešení práce.

Obsah

Kapitola 1

Úvod

S postupujícím technologickým vývojem mobilních zařízení rostou i požadavky uživatelů na co nejvyšší komfort a funkcionalitu při užívání. Výkonnější zařízení nám dávají mnohem větší prostor k aplikaci technologií, které bychom mohli ještě nedávno označit, co se týče mobilních telefonů, za nepoužitelné. Oblasti počítačového vidění a rozšířené reality můžeme směle označit za velice výpočetně náročné, avšak v mobilním světě velmi se rozvíjející odvětví. Tato práce popisuje vývoj mobilní aplikace pro operační systém Android, která bude sloužit jako real-time překladač textu snímaného fotoaparátem.

V úvodní části práce se nachází výčet a popis teoretických znalostí potřebných či užitečných pro následný návrh implementované aplikace. Nejprve je popsána oblast rozpoznávání textu v obrazu. Čtenář získá povědomí o pojmu OCR a OCR systémech. Je zde prezentováno několik aplikačních rozhraní, které mohou být pro účely rozpoznávání použity. Následuje kapitola sloužící pro seznámení se systémy pro strojový překlad textu. Jsou zde uvedeny některé on-line i off-line nástroje a zmíněny jejich hlavní přednosti a zápory.

Další kapitola pojednává o možnostech Android OS ve třech oblastech. První z nich popisuje snímání obrazu z fotoaparátu mobilního zařízení a jeho zobrazení na displeji. Následuje specifikace možností zmíněných v předchozích kapitolách, tedy extrakce textu z obrazu a možnostech strojového překladu textu.

Kapitolu zabývající se návrhem ještě předchází popis a porovnání existujících aplikací s podobnou funkčností, jakou by měla disponovat implementovaná aplikace. Návrh samotný spočívá v popisu komponent resp. jednotlivých funkčních bloků a dalších důležitých prvků aplikace. Dále je popsána užitá OCR knihovna včetně možnosti jejího rozšíření a způsob předzpracování obrazu získaného z fotoaparátu mobilního zařízení.

V pořadí sedmá kapitola uvádí popis implementačních detailů dle znalostí získaných v předchozích kapitolách. Postupně je uvedena architektura implementace, životní cyklus aplikace, popis vytváření grafického uživatelského rozhraní, implementace databáze obsahující slovník pro překlad a další dílčí aspekty vývoje.

V závěru práce se nachází popis několika experimentů ověřujících činnost aplikace v různých situacích či s různým nastavením. V samotném závěru práce je prezentováno porovnání s obdobnými aplikacemi.

Kapitola 2

Rozpoznávání textu v obrazu

OCR, neboli Optical Character Recognition, řeší problém rozpoznání znaků v obrazových datech. OCR algoritmy se dělí do různých skupin podle typu zpracovávaných dat. Podle času rozpoznávání je můžeme rozdělit na:

- **on-line** – rozpoznávání právě vytvářených znaků (uživatel zrovna píše),
- **off-line** – rozpoznávání již zaznamenaných dat.

Kvalita výstupu po rozpoznávání do značné míry závisí na vstupních datech. Je proto kladen velký důraz na pre-processing, tedy předzpracování. Metody předzpracování jsou podány v sekci 2.2.2. Následuje popis různých způsobů extrakce rysů a klasifikací znaků. V poslední sekci je zmíněn problém post-processingu, tedy zpracování dat, která jsou výstupem rozpoznávání tvarů písmen (klasifikace).

2.1 Historie

Historii OCR lze rozdělit do 3 generací. Začala se psát v 60. letech 20. století. Před touto dobou existovalo několik pokusů k rozpoznávání psaného písma, vzhledem k nedostatečným technologiím, však nebylo dosaženo vyšších úspěchů. Důležitým bodem bylo zaregistrování prvního patentu na rozpoznávání roku 1929.

V 50. letech začaly vznikat první komerční projekty. OCR tehdejší doby pracovaly na principu využití mechanických masek a fotodetektoru [16]. Pokud světlo procházející přes masku a písmeno nedopadlo na fotodetektor (znak stínil), bylo to známkou odhalení správného znaku.

První generace, datovaná od druhé poloviny 50. do první poloviny 60. let, zvládala pouze jednoduché rozpoznávání znak po znaku [16]. Byla vytvořena speciální písma, pouze pro účely kvalitnějšího OCR, která však nebyla příliš přirozená pro člověka [6]. Stroje začaly podporovat několik druhů písem najednou.

Kolem počátku 70. let, již nebylo rozpoznávání strojového textu problémem. Započaly experimenty s psaným písmem. OCR bylo na takové úrovní, že se začalo využívat např. pro čtení směrovacích čísel na dopisech a tedy k jejich automatickému třídění. Byly vytvořeny vysoce stylizovaná písma OCR-A (USA) a OCR-B (Evropa) pouze pro účely snadného rozpoznávání.

Cenově začalo být OCR dostupné i pro běžného uživatele kolem roku 1986, což je období konce třetí generace. Do tohoto roku naráželo OCR hlavně na nedostatečný výkon tehdejšího hardwaru.

2.2 Komponenty OCR systému

V této sekci si popíšeme jednotlivé komponenty OCR systému v pořadí v jakém vstupují do procesu rozpoznávání. Pomineme-li snímaní samotného obrazu, první částí je lokalizace oblastí s textem a jeho rozdělení na jednotlivé znaky. Následuje podkapitola popisující předzpracování obrazu. Pokračujeme extrakcí rysů, klasifikací konkrétního znaku a post-processingem, tedy konečným zpracováním.

2.2.1 Lokalizace textu

Lokalizace textu v obrazu je prvním netriviálním úkolem. Prohledávání části obrazu, kde ani není nic k nalezení, stojí čas a výkon. Moderní OCR systémy nepracují na principu vyhledávání jednotlivých znaků a jejich rozpoznání, ale na komplexní analýze zadaného dokumentu. To kupříkladu poskytuje informaci o tom, že je dokument rozdělen na dva sloupce. Pokud by tato analýza nebyla provedena, všechny řádky by byly pravděpodobně spojeny, ačkoli na sebe logicky nenavazují.

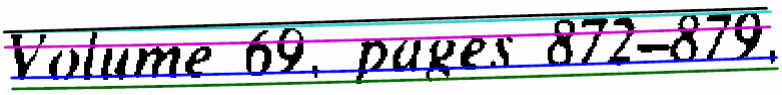
V souvislosti s lokalizací je nutné také zmínit otázku segmentace, někdy též choppingu (rozsekání). Jde o proces rozdělení nalezených textových sekvencí na jednotlivé znaky. V případě, že se znaky nedotýkají nebo nejsou poškozeny, jde o relativně jednoduchý implementační proces. Takto vytvořené obrázky se znaky jsou následně samostatně odeslány k rozpoznání.

Při segmentaci a lokalizaci mohou v zásadě nastat tyto hlavní problémy:

- **Šum v obrazu** – může být chybně označen například za čárku, tečku případně poškodit jiný symbol.
- **Poškozené nebo dotýkající se znaky** – ústí k nemožnosti znak rozpoznat případně označit chybně.
- **Spojení textu s grafikou v obrazu** – výsledkem je nejčastěji nenalezený text, přičemž této vlastnosti je často využíváno u CAPTCHA[1].

Baseline Fitting

Jednou z metod podporující kvalitnější segmentaci a zároveň podporující lepší preprocesing je Baseline Fitting. Na obrázku 2.1 můžeme v textové sekvenci vidět čtveřici čar, určující kupříkladu náklon písma. Znalost této informace nám dává možnost sekvenci narovnat a tím zvýšit šanci na správné rozpoznání [22].



Obrázek 2.1: Baseline Fitting použitý v knihovně Tesseract [22].

[1] test, kdy je uživatel požádán o přepis nějakým způsobem poničeného textu z obrazu, aby dokázal, že je člověk

2.2.2 Pre-processing

Pojmem pre-processing, českým výrazem předzpracování, rozumíme úpravu vstupního obrazu na takovou formu, aby byl text co nejlépe rozeznatelný od svého okolí. Pro tyto účely se používá mnoho technik, zde jsou popsány ty nejčastější.

Binarization

Binarization, neboli binarizace, je převod části obrazu se znakem do černobílé varianty. Jako ideální výsledek této operace můžeme označit obrázek s černým znakem na bílém pozadí (případně inverzní varianta). U každého obrázku specifikujeme hodnotu `threshold` podle které u jednotlivých pixelů určíme, zdali bude jeho nová barva černá či bílá.

Noise elimination

Další technikou, zvanou noise elimination, tedy eliminace šumu, docílíme odstranění šumu typu pepř (náhodné černé pixely) či sůl (náhodné bílé pixely) z obrazu. Tyto náhodné body poté nedělají problémy při rozpoznávání znaku.

Filling, Thining

Filling, též plnění, spočívá v doplňování zlomů, děr, případně jiných chyb ve znaku jeho barvou. Jedná se tedy o opačný postup oproti předchozí metodě. Při rozpoznávání je u celistvého znaku vetší šance k úspěchu [6].

Thining, také ztenčování, označuje proces, kdy se pokoušíme snížit tloušťku čáry znaku.

2.2.3 Extrakce rysů

Pod pojmem rys rozumíme charakteristickou vlastnost daného objektu, v tomto případě symbolu. Extrakce musí produkovat takové rysy, které jednoznačně definují třídu, do které následně symbol zařadíme. V opačném případě není zaručeno správné rozpoznání znaku. Existuje několik metod extrakce, nejčastější jsou:

- **Distribuce bodů** – techniky závislé na rozložení bodů v obrazu. Např. souvislá oblast bílých bodů (pixelů) v centru obrazu může indikovat písmeno „O“.
- **Transformace** – tyto techniky jsou založeny na převodu symbolu v obrazu na jinou reprezentaci, např. dvojici křivek.
- **Strukturální analýza** – spočívá v nalezení struktury, kterou je symbol reprezentován. Může jím být například množina vektorů. Tento přístup funguje velmi dobře např. pro stylizovaná písma.

2.2.4 Klasifikační metody

Klasifikací nazýváme proces, kdy přiřadíme zkoumaný symbol do jemu příslušící třídy. Třída poté reprezentuje jeden znak abecedy. Reprezentace symbolu se různí, většinou jde o `feature vector` neboli seznam rysů. Pro klasifikaci jsou nejčastěji použity tyto principy:

- **Porovnávání** – výpočet diference, často Eukleidovské vzdálenosti, mezi rysy zkoumaného symbolu a prototypy symbolů. V některých případech nedochází ani k extrakci

rysů, ale porovnávány jsou přímo obrazy a diference je určena jako počet rozdílných pixelů. Nižší diference označuje větší pravděpodobnost shody.

- **Statistická klasifikace** – založeno na statistickém přístupu. Symbolu jsou přiřazeny hodnoty určující, jak moc „se hodí“ do jednotlivých tříd.
- **Neuronové sítě** – spočívá v natrénování neuronové sítě na celou abecedu, což odpovídá určení hodnot vah propojení mezi jednotlivými neurony, a následnému vložení rysů na vstup sítě. NS rovnou přiřadí symbol do vybrané třídy. Jejich problémem je malá předpovídatelnost a obecnost.

2.2.5 Post-processing

Post-processing je souhrnné pojmenování pro metody aplikované po klasifikaci znaků na výsledné posloupnosti znaků.

Grouping

Grouping, neboli seskupování, slouží ke spojování řady znaků do řetězců, zpravidla vět, slov nebo čísel. Úlohou je správný odhad toho, co má s čím být spojeno. Čím blíže jednotlivé znaky jsou, tím větší mají šanci k seskupení.

Seskupování slov je relativně jednoduché, u vět však můžou nastat problémy s proměnlivou mezerou mezi slovy.

Spelling verification

Tato metoda skrývá při správném užití veliký potenciál pro úpravu finálního dokumentu. Žádná z metod neposkytuje 100 % spolehlivost rozpoznání všech znaků. Z toho plynou možné příležitosti opravit některé řetězce pomocí syntaktických či sémantických pravidel.

Pomocí syntaxe jsme schopni například první písmeno ve větě určit jako velké. Další možnost spočívá v uchovávání si jistých zákonitostí daného jazyka. Kupříkladu, pravděpodobnost výskytu písmene `k` po písmenu `h` je v anglickém jazyce zcela vyloučena. Pokud na takovou kombinaci narazíme, jedná se zcela určitě o chybu.

Alternativním přístup je vyhledat slovo ve slovníku a ověřit tak jeho existenci. Pokud slovo nenalezneme, vyhledáme náhradní slovo s největší pravděpodobností výskytu právě v tomto dokumentu. Existují však i pokročilejší techniky, kdy OCR systém disponuje několika slovníky (nejpoužívanější slova, nejčastější zkratky, ...) čímž dostáváme další informaci o správném slově. Každé písmeno v předané posloupnosti může navíc disponovat parametrem spolehlivosti, a pokud nám slovník nabídne více alternativ, může i tato informace mít svoji váhu [22].

2.3 Dostupná API

Existuje vícero API, která přímo implementují nebo napomáhají k implementaci OCR. V této kapitole bude rovněž několik nejvýznamnějších zástupců popsáno.

2.3.1 OpenCV

OpenCV, též Open Source Computer Vision Library, je pojmenování pro jednu z největších a také nejvíce používanou knihovnu počítačového vidění a strojového učení. Jak již název napovídá, je šířena pod open source licencí, může do ní tedy přispívat každý. Vznikla v roce 1999 v laboratořích společnosti Intel. Využívají ji globální společnosti jako Google, IBM či Microsoft, ale také spousta výzkumných týmů či startupů. Veškerý následující text se přímo týká OpenCV 2.x API a vyšší [13].

Knihovna obsahuje více než 2500 optimalizovaných algoritmů pro analýzu vizuálního obsahu. Jde o algoritmy na detekci objektů v obrazu, klasifikace lidských činností ve videu, extrahování 3D objektů z obrazu, hledání podobných obrázků v databázi, sledování pohybu očí, spojování obrazů do větších za účelem zisku většího rozlišení a mnoho dalšího. Praktickým příkladem užití je monitorování bazénů za účelem kontroly před tonoucími.

OpenCV má rozhraní pro Javu, C++ (velké možnosti parametrizace užitím šablon), C, Python a MATLAB. Zároveň podporuje většinu dnešních dominantních operačních systémů, jmenovitě Windows, Linux, Android a MacOS. Problémem není ani hardwarová akcelerace pomocí CUDA či OpenCL.

Knihovnu lze rozdělit na tyto hlavní části:

- **core** – modul definující hlavní datové typy a funkce,
- **imgproc** – modul pro zpracovávání obrazu obsahující filtrování, transformace, převody mezi barevnými modely aj.,
- **video** – modul pro analýzu video sekvencí sloužící např. ke sledování objektů v obrazu,
- **calib3d** – nastavení kamery (pohledu), výpočet pozice objektu v obrazu aj.,
- **features2d** – detektory charakteristických prvků v obrazu,
- **objdetect** – detekce objektů v obrazu,
- **highgui** – rozhraní pro záznam videa, video kodeky, prvky UI,
- **gpu** – rozhraní pro HW akceleraci v grafických kartách.

Na platformě Android je knihovna dostupná přímo jako aplikace v Google Play, což má za následek rychlejší instalaci aplikací, ji využívajících, do zařízení. Nedochází totiž k transferu několika megabajtů velké knihovny po každé kompilaci. Implementace v nativním kódu Android OS, tedy v C/C++, dává potenciál pro vysoký výkon. Nad touto implementací se nachází rozhraní v Javě pro pohodlnější vývoj [3].

2.3.2 ABBYY OCR

OCR poskytované společností ABBYY využívá technologie založené na cloudu. Uživatel nahraje dokument/obrázek, u něhož požaduje překlad do strojového textu, na server, kde

dojde ke zpracování. Engine dokáže extrahovat text 198 různých jazyků, včetně češtiny. Rozluštěný text je poté odeslán zpět.

Služba je platformně nezávislá, přistupuje se k ní přes webové rozhraní, nicméně dostupné OCR SDK podporuje většinu operačních systémů, včetně Androidu. Dostupné jsou trial i placené licence. Trial licence umožňuje překlad 50 dokumentů měsíčně, což je pro účely této aplikace nevhodné. Služba může vyhovovat spíše uživatelům, kteří vyžadují zpracování celých stran A4, nikoli pouze několika slov v obrázku [1].

2.3.3 ExperVision OCR SDK

V mnohem podobný předchozímu OCR systému. Jedná se také o komerční záležitost, funkční na většině majoritních operačních systémů. SDK včetně dodatečných utilit je šířeno pod názvem OpenRTK[2]. Podle testu PC Magazine se jedná o nejrychlejší OCR technologii na trhu [7]. Oficiální web slibuje podporu více než 2600 různých fontů, včetně různých formátování typu kurzíva.

Vývojářská licence je zpoplatněna jednorázovou částkou 5 190 $ s povinností platit další roční poplatky. Oproti řešení od ABBYY zde není limit na množství zpracovaných dokumentů. Doporučení pro využití pro zpracování např. celých knih však platí i zde.

2.3.4 OCRopus

Open source projekt pro analýzu a OCR dokumentů. OCRopus obsahuje vlastní grafické uživatelské rozhraní usnadňující práci méně zkušeným uživatelům. Oproti ostatním OCR systémům zvládá rozpoznávání „perchar“ tedy pro každý znak zvlášť.

Implementace provedena v Pythonu. Toto automaticky brání použití na platformě Android, která oficiální podporu pro Python nemá. I kdyby podpora zaručena byla, hardwarové požadavky na zařízení jsou vyšší než prostředky, jakými disponují současné mobilní telefony. Jediným způsobem, jak by bylo možno použít tuto technologii v mé práci, by bylo provozování OCRopusu na straně serveru a posílat mu obraz ke zpracování. Z důvodu vysoké latence mobilního připojení a vysokého objemu dat je však toto řešení prakticky nepoužitelné.

2.3.5 Tesseract

Tesseract je open source OCR engine, který byl původně vyvíjen v laboratořích HP, nyní se nachází pod patronátem společnosti Google. Může být využit přímo jako aplikace (včetně GUI třetích stran) nebo s použitím API. Implementačním jazykem je C/C++ a jsou podporovány všechny majoritní platformy (Linux, Windows i MacOS). Podporuje mnoho jazyků, včetně češtiny, a dokonce obsahuje nástroje, které umožňují natrénování nových jazyků. Spolupracuje s knihovnou Leptonica, která slouží pro zpracování a analýzu obrazu. Ta se stará hlavně o předzpracování obrazu [20].

Tesseract lze v několika ohledech parametrizovat. Lze specifikovat tzv. `whitelist` resp. `blacklist`, tedy seznam znaků, které hledat resp. ignorovat. Dále můžeme upřednostnit rychlost algoritmu před přesností. Při použití knihovny musíme vždy dodat soubor s natrénovaným jazykem [22].

Existuje projekt zvaný tesseract-android-tools[3], který přidává podporu knihovny Tesseract pro Android, včetně implementace základního rozhraní v Javě. Tento framework jde

[2] OpenRTK je ochrannou známkou společnosti ExperVision

[3] http://code.google.com/p/tesseract-android-tools/

libovolně rozšiřovat a v konečném důsledku tedy lze využít všech funkcí enginu v nativní formě. Stejně jako Tesseract knihovna je poskytován pod open source licencí.

2.3.6 Cognitive OpenOCR - CuneiForm

Ruský open source projekt realizovaný v jazycích C/C++, který však již několik let nejeví žádné známky dalšího vývoje. Zvládá rozpoznávání celkem 20 jazyků, čeština však mezi nimi chybí. Pracuje na podobném principu jako Tesseract – ke každému jazyku přísluší slovník slov daného jazyka, což zpřesňuje výsledky rozpoznávání [17].

Bohužel není příliš vhodný pro použití na mobilních zařízeních s Android OS, kvůli neexistujícímu Java rozhraní pro volání funkcí knihovny.

Kapitola 3

Překlad textu

Tato kapitola je zaměřena na možné způsoby překladu textu na mobilním telefonu. Oblast překladu z jednoho jazyka do jiného se nazývá strojový překlad – machine translation. Soustředím se na překlad dostupný jak on-line, tak i přímo v mobilním zařízení bez připojení k internetu. Každý z těchto přístupů má svá pozitiva a negativa, která shrnu v sekcích 3.1 a 3.2. Podle principu funkčnosti se překladače dělí v zásadě na 4 druhy [14]:

- **Založené na pravidlech (Rule-based)** – systém strojového překladu používající velké množiny pravidel pro jednotlivé jazyky. Z velké části spočívá na práci expertů, kteří pravidla specifikují. Jeho vývoj proto bývá velmi drahý a dlouho trvající. Postupným rozšiřováním a aktualizováním pravidel může docházet k degeneraci celého systému.
- **Statistické** – systém založený na statických algoritmech. Výstupem takového algoritmu bývá nejpravděpodobnější, čili nejvíce vyhovující, překlad z množiny všech překladů. Na rozdíl od předchozího přístupu, systém nevyžaduje přílišného zásahu člověka. Pro správnou funkčnost musíme mít k dispozici velké množství dat, na kterých dojde k natrénování vazeb mezi jednotlivými slovy, čímž vzniká statistický model. Větší množství trénovacích dat nám produkuje lepší výsledky překladu. Většina velkých hráčů na trhu používá právě tento přístup. S rostoucím počtem digitálních dat roste „inteligence“ jejich překladačů.
- **Hybridní** – kombinace dvou předešlých metod. Často systém založený na pravidlech jehož jádrem je statistický model. Snaží se přejmout přednosti obou přístupů.
- **Nové přístupy** – používají nové sofistikované techniky např. předzpracování (transformace) jazyka do vlastních tvarů.

Text získaný pomocí OCR může být nepřesný, či může obsahovat některé znaky navíc. Ačkoli to není jeho primární cíl, překladač, který by tyto chyby odhalil a opravil, by byl jistě velkým přínosem.

3.1 On-line

Internet disponuje velkým množstvím on-line překladačů. Jejich použití pro účely této aplikace, ale i dalších mobilních aplikací obecně, nese řadu výhod. Na straně serveru se může nacházet velice komplexní systém pro překlad. Pokud by se nacházel na telefonu,

jistě by se nadále nedalo hovořit o real-time překladu. Jsou zde však další pozitiva ve formě téměř nulové výpočetní náročnosti na straně koncového zařízení a z toho plynoucího nízkého využití baterie.

Základní charakteristiku on-line strojových metod překladu můžeme spatřit v tabulce 3.1.

3.1.1 Google Translate API

Strojový překlad založený na statistickém modelu. Podporuje překlad mezi 80 jazyky včetně češtiny. Nabízí detekci jazyka zadaného textu. Přesnost detekce je závislá na jeho délce. Nabízí uživateli, v případě nekvalitního překladu, navrhnout nový překlad [10]. Google neustále pracuje na přidávání dalších jazyků, protože každý přidaný jazyk může v konečném důsledku zlepšit překlad i mezi jinými jazyky (tranzitivita). Pro většinu jazyků je označován za jeden z nejlepších současných systémů strojového překladu [19].

Google API je nyní bohužel poskytováno formou placené služby zvané jako Google API v2 [11]. Zpoplatněno je však pouze API, nikoliv webová aplikace na adrese `translate.google.com` a překlad lze získat prostřednictvím tohoto rozhraní. Tento přístup však nepovažuji za správný a v aplikaci budou využity pouze standardní postupy, tedy překlad jiným způsobem.

3.1.2 Bing Translate API

Velice podobný předešlému Google Translate API. Ve vlastnictví společnosti Microsoft [15]. Nabízí rovněž rozpoznání jazyka, navrhování nových překladů, podporuje češtinu a jádro tvoří statistický model. K překladu se lze dopracovat přes několikero rozhraní a to:

- **AJAX** – asynchronní požadavek implementován v JavaScriptu,
- **HTTP** – standardní HTTP požadavek,
- **SOAP** – volání metod serveru pomocí XML zpráv.

I v tomto případě je aplikační rozhraní zpoplatněno, vývojáři však mohou využít překladu dvou milionů znaků měsíčně zdarma[1]. Rovněž lze pro překlad využít webovou aplikaci na adrese `bing.com/translator`, která je zdarma.

3.1.3 Yandex Translate API

Překladač ruského vyhledávače `Yandex.ru`. Opět založen na statistickém modelu [24]. Oproti předchozím však nabízí zdarma podstatně větší množství překladů zdarma. Aktuální limit byl stanoven na hranici 10000 požadavků denně nebo jeden milión znaků denně. Systém podporuje zhruba 30 jazyků včetně češtiny. Při každém požadavku na překlad musíme zadat také klíč, který lze však získat zdarma (podmíněno vlastnictvím emailové schránky na doméně @yandex.ru, která je však také zdarma) [25].

Požadavek na překlad je vykonáván jednoduchým HTTP dotazem. Následná odpověď může být ve tvaru XML nebo JSON.

[1] http://datamarket.azure.com/dataset/1899a118-d202-492c-aa16-ba21c33c06cb

3.1.4 BabelFish.com

Zástupce systému založeného na pravidlech. Bohužel neposkytuje aplikační rozhraní a v seznamu podporovaných jazyků schází čeština, což je škoda. Mohlo by být zajímavé sledovat rozdíly v překladu mezi statistickým modelem a modelem založeným na pravidlech, právě pro účely naší aplikace. Lze očekávat, že bude použita hlavně pro překlad několika málo, maximálně desítek, slov. Právě délka vstupu ve velké míře ovlivňuje kvalitu výstupu pro různé typy překladačů [19].

Zajímavostí této služby je bezpochyby lidský překlad zadaného textu. Jde o placenou službu, přičemž jedno přeložené slovo má podle společnosti Yahoo, která BabelFish.com vlastní, hodnotu 0.06 až 0.12 dolarů [2].

3.1.5 Glosbe.com

Jeden z mála překladačů, který lze využívat zcela zdarma. Jediným limitem je nespecifikovaný počet přístupů z jedné IP adresy v rovněž nespecifikovaném časovém úseku. Vytváří jej komunita, jeho kvalita velmi pokulhává za ostatními profesionálními nástroji velkých firem. Pro použití není potřeba žádný API klíč. Výstup po překladu je v porovnání například s `Yandex` naprosto tristní. Dle mého názoru, i po nepřeložení zadaného textu produkuje příliš mnoho netriviálně zpracovatelných dat. Výstupem překladu anglického slova „square“ je 2000 řádkový XML dokument (při maximálním omezení výstupu), zatímco u konkurence získáme řádky tři a výsledek má porovnatelnou kvalitu.

Jak jej prezentují i sami autoři, jedná se spíše o obrovský slovník pro velký počet jazyků, než inteligentní překladač [9].

Tabulka 3.1: Porovnání on-line strojových překladačů

Jméno	Cena	Čeština	Princip	Vyžaduje klíč
Google API	$20 za 1M	ano	statistický	ano
Bing API	2M měsíčně zdarma	ano	statistický	ano
BabelFish.com	zdarma	ne	rule-based	ne
Yandex Translate API	1M denně zdarma	ano	statistický	ano
Glosbe	zdarma	ano	statistický	ne

Poznámky: **xM** x milionů znaků

3.1.6 Ostatní

Do této kategorie jsou zařazeny existující systémy pro překlad, které však příliš nevyhovují požadavkům vytvářené aplikace. Jde o systémy:

- **Gengo API** – Lidský překlad zadaného textu příp. dokumentu, jedná se tedy o dlouhodobější proces. Do procesu překladu se může zapojit každý, přičemž dostává podíl z výdělku. Často se využívá k překladu videí na `Youtube.com`.

3.2 Off-line

Situace s off-line překladači, v tomto případě spíše slovníky, není na tak dobré úrovni, jako s jejich on-line alternativami. Mluvíme zde hlavně o slovnících, protože neexistuje příliš mnoho projektů, které mají dostupné API příp. SDK pro off-line překlad či jsou open source. Základní charakteristiku zde uváděných nástrojů můžeme opět spatřit v tabulce 3.2.

Slovníky jsou nejčastěji vytvářeny komunitou, což bohužel negativně ovlivňuje jejich kvalitu. V hojné míře bývají dostupné jako textové soubory v určitém formátu (např. XML) a jejich použití tedy zahrnuje implementaci další funkcionality vykonávající jejich zpracování, uložení do databáze a rychlé vyhledávání.

Pozitiva využití off-line nástrojů oproti on-line, spatřuji hlavně v rychlosti překladu a nepotřebě aktivního internetového připojení. Jak již bylo zmíněno, nejčastější využití vyvíjené aplikace spatřuji v překladu kupř. informačních cedulí v zahraničí, kde se může datové připojení prodražit.

3.2.1 FreeDict

FreeDict, v nezkráceném tvaru Free Dictionary, je projekt, který si klade za cíl poskytnout komukoli zdarma vícejazyčné slovníky [8]. Tyto slovníky jsou dostupné také jako instalovatelné balíčky do linuxových distribucí.

Všechny slovníky dodržují TEI/XML[2] formát vhodný pro snadné zpracování. Každá položka slovníku obsahuje nejen samotný překlad, ale často také výslovnost aj. Co se týče češtiny, přímo přeložitelná je pouze z/do angličtiny, žádná jiná přímá kombinace bohužel není podporována.

3.2.2 Dicts.info

Projekt `Dicts.info` poskytuje volně ke stažení čtveřici slovníků. Jedná se o:

- **Universal dictionary** – Slovník, který je vytvářen přímo pod záštitou webu `Dicts.info`. Může být však použit pro osobní potřebu a neměl by být dále šířen. Dle prezentovaných statistik obsahuje z uváděné čtveřice největší počet českých frází.
- **Wiktionary** – Partnerský projekt Wikipedie, který si dává za cíl vytvořit slovník obsahující všechny jazyky a definice všech slov. Data samotná podléhají GNU FDL[3] licenci, která se používá pro volně šiřitelné dokumenty apod.
- **Omegawiki** – Ve své podstatě totožný projekt jako předchozí. Opět pod GNU FDL licencí.
- **Wikipedia** – Slovník největší webové encyklopedie. Z těchto čtyř slovníků obsahuje celkově největší počet frází.

Každý ze slovníků podporuje češtinu a nabízí překlad do/z 38 až 60 jazyků (konkrétní hodnota v závislosti na zvoleném slovníku). Stažení kupříkladu česko-vietnamského slovníku není proto žádný problém. Kvalita překladu však bude odpovídat tomu, že jej dobrovolně vytvářela komunita uživatelů. Pro možnosti otestování lze využít webové rozhraní, které vyhledá zadanou frázi ve více slovnících najednou a zobrazí výsledek překladu [5].

[2] http://www.tei-c.org/release/doc/tei-xsl/
[3] http://www.gnu.org/copyleft/fdl.html

3.2.3 ColorDict

Android aplikace pracující jako slovník. Vývojářům jiných Android aplikací nabízí API, přes které lze překládat zadané texty. Aplikační rozhraní je realizováno přes třídu `Intent`. Instance této třídy se používá ke spuštění zvolených aktivit či akcí, přičemž umožňuje volanému subjektu předat nějakou informaci, v tomto případě text k přeložení a některé další parametry. V současné době API umožňuje pouze zobrazení PopUp[4] okna, tedy okna překrývajícího obsah aktuálně běžící aktivity, což není pro účely naší aplikace vhodné.

Přímo v aplikaci lze přidávat vlastní slovníky. Absence českého slovníku tedy není příliš velký problém. Lze se s její pomocí v mnoha případech vyhnout implementaci vlastního off-line překladače.

3.2.4 ABBYY Lingvo Dictionaries

Android aplikaci fungující podobně jako předešlý konkurent. Firma ABBYY se angažuje také na poli překladačů. Aplikace obsahující základní jazykové balíky je oproti předchozí zpoplatněna. Tato sada umožňuje ve spojitosti s češtinou pouze překlad do angličtiny, což není pro naše účely dostačující.

I za předpokladu podpory dalších jazyků, každý uživatel by musel, v případě využití této aplikace pro překlad, aplikaci zakoupit.

3.2.5 QuickDic

Následující Android aplikaci, tentokrát poskytovanou jako open source projekt, můžeme taktéž využít pro překlad. Volitelně lze přidat předzpracované slovníky, které jsou založeny na `Wiktionary`, což dává možnost využít opravdu širokou škálu jazyků. Co se týče češtiny, ihned jsou k dispozici překlady do angličtiny, němčiny a francouzštiny. Autor navíc poskytuje utilitu pro přidání vlastních slovníků [12].

Tabulka 3.2: Porovnání off-line nástrojů

Jméno	Cena	Pouze CZ<>EN	Pouze slovník	Licence
FreeDict	zdarma	ano	ano	GNU GPL
Dicts.info	zdarma	ne	ano	GNU FDL
ColorDict	zdarma	ne	ne	-
ABBYY	$ 6	ne	ne	-
QuickDic	zdarma	ne	ne	Apache

Poznámky: **CZ<>EN** překlad z/do češtiny do/z angličtiny

[4] vyskakovací okno

Kapitola 4

Možnosti Android OS

V této kapitole se zaměřím na možnosti, které poskytuje systém Android v následujících třech, pro naši aplikaci klíčových, oblastech:

- **snímání obrazu** – získání obrazu z fotoaparátu mobilního telefonu,
- **rozpoznávání textu** – přečtení textu, který se vyskytuje v obrazu, získaného z fotoaparátu,
- **překlad textu** – strojový překlad přečteného textu z jednoho jazyka do druhého.

4.1 Snímání obrazu

Pro snímání obrazu z fotoaparátu můžeme využít dva různé způsoby. V prvé řadě se jedná o využití přímo Android frameworku, další možností spočívá ve využití knihovny `OpenCV`. Oba přístupy nabízejí srovnatelné možnosti.

4.1.1 Použití Android SDK

Android SDK poskytuje hned 2 způsoby, jak se dá dopracovat k obrazovým datům z fotoaparátu. Pomocí `Camera API` nebo s využitím `OpenGL API`.

V případě použití prvního aplikačního rozhraní dochází ke zpracovávání obrazových dat na procesoru. Realizace snímání obrazu spočívá v implementaci potomků následujících tříd a rozhraní:

- **SurfaceView** – Potomek třídy `View` sloužící jako plocha na displeji, na kterou dochází k vykreslování dat získaných z fotoaparátu. Pro přístup k samotné ploše, též nazývané `surface`, se využívá `SurfaceHolder`.
- **SurfaceHolder.Callback** – Rozhraní, které implementuje třída `SurfaceView` zajišťující správné chování při změnách týkajících se `surface`. Vyžaduje implementaci metod:
  - *surfaceCreated()* – K volání této metody dochází při vytváření `surface`. Typicky se zde zajišťuje získání výhradního přístupu k fotoaparátu a spuštění vykreslování.
  - *surfaceChanged()* – Reakce na změnu parametrů `surface`, ke které dojde typicky při změně orientace displeje. Očekává se minimálně změna atributů dat z fotoaparátu.

- *surfaceDestroyed()* – Tato metoda slouží k uvolnění zabraných prostředků, kupř. zrušení výlučného přístupu k fotoaparátu, při uvolňování instance `surface` z paměti.

- **Camera.PreviewCallback** – Poslední rozhraní poskytuje callback[1] metodu *onPreviewFrame()*, k jejímuž zavolání dochází při každém obdržení obrazu z fotoaparátu. Veškerý přístup k fotoaparátu se doporučuje implementovat v jiném než UI vlákně[2], aby nedocházelo k zasekávání aplikace [4].

První parametr poslední zmíněné metody nám dává k dispozici obrazová data z fotoaparátu jako pole typu `byte`. Neexistuje však přímočarý způsob, jak tato data modifikovat a zobrazit je uživateli ve změněné podobě (přeložený text). Proto se podíváme na alternativní přístup založený na využití `OpenGL`. Opět si stručně popíšeme hlavní třídy, na kterých tento princip získání obrazu spočívá:

- **GLSurfaceView** – jedná se o přímého potomka třídy `SurfaceView` a vykonává naprosto stejnou činnost, tentokrát s využitím `OpenGL` rozhraní.

- **GLSurfaceView.Renderer** – renderer slouží k vykreslování grafických prvků do příslušného `GLSurfaceView`. Víceméně kopíruje užití rozhraní `SurfaceHolder.Callback` a k němu přidává novou metodu *onDrawFrame()*. Ta je zavolána při každém požadavku o překreslení obsahu `surface`.

Jelikož dochází k veškerým výpočtům na grafickém čipu (GPU), přístup k textuře obsahující data z fotoaparátu je mírně komplikovanější než v předchozím případě. Po zavolání metody *updateTexImage()* instance třídy `SurfaceTexture` dojde ke kopii nejnovějších dat do paměti GPU, načež k nim můžeme přistupovat jako k textuře přes globální konstantu `GL_TEXTURE_EXTERNAL_OES`.

Samotné `OpenGL` nedisponuje metodami pro úpravu textur a grafiky, spíše se soustředí na jejich vytváření. Přistupovat k obrazovým datům přímo např. jako k poli typu `byte` jednoduše nelze.

V obou případech bychom s využitím Android API měli pro účely snímaní obrazu několik problémů. V prvním případě, kvůli absenci jednoduchého rozhraní pro změnu dat a zobrazení uživateli, v případě druhém z důvodu nedostatečné možnosti manipulace s daty.

4.1.2 Použití OpenCV

Alternativa k Android API spočívá v použití knihoven třetích stran, v tomto případě `OpenCV`. Balík `OpenCV SDK` poskytuje několik vzorových příkladů použití knihovny. Jeden z příkladů demonstruje právě zobrazení obrazu z fotoaparátu a to ve dvou variantách – nativní implementace (C/C++) a implementace v Javě. Princip fungování se pokusím vysvětlit v analogii s Android API. Opět zde máme několik tříd a rozhraní:

- **CameraBridgeViewBase** – Třída implementující funkcionalitu `SurfaceView` a `SurfaceHolder.Callback`, usnadňuje tedy programátorovi část práce s inicializací `surface`. Slouží jako prostředník mezi `OpenCV` knihovnou a `Camera API`.

[1] funkce volána jako reakce na nějakou událost
[2] hlavní vlákno Android aplikací starající se mimo jiné o vykreslování uživatelského rozhraní

- **CvCameraViewListener2** – Rozhraní vyžadující implementaci metod *onCameraViewStarted()*, *onCameraViewStopped()* a *onCameraFrame()*. První dvě metody jsou volány, jak již jejich název napovídá, při spuštění resp. ukončení zobrazování preview z fotoaparátu.

  Třetí metoda poskytuje jako parametr matici bodů, pixelů získaného obrazu. Návratovou hodnotou metody, taktéž maticí stejného typu, vracíme obrázek, který bude následně zobrazen uživateli na displeji.

Možnosti knihovny `OpenCV` tedy přesně vystihují požadavky naší aplikace – získání obrázku, jeho zpracování a prezentování upravené verze na displeji.

4.2 Rozpoznávání textu

Pro účely rozpoznávání textu v obrazu není bohužel v `Android API` žádná nativní podpora. Základní metody pro manipulaci s obrazovými daty jsou však samozřejmě k dispozici.

Nabízejí se dva přístupy, jak se s tímto problémem vypořádat. Oba spočívají ve využití knihoven třetích stran. Prvním je implementace vlastního OCR systému s využitím knihoven pro pracování obrazu. Vhodným kandidátem by mohla být knihovna `OpenCV`, která je již zmíněna v sekci 2.3.1. Jako alternativy můžeme zmínit knihovny `BoofCV`[3], `Leptonica`, které obě mohou být použity v Android OS.

Druhou, preferovanější možností je využití již existující knihovny. Pomineme-li online řešení, které vyžadují rychlou síť s vysokou propustností, jediným skutečně použitelným řešením shledávám v knihovně `tesseract`, o které je zmínka v sekci 2.3.5. Existuje balíček s nástroji pro využití na platformě Android a to ve dvou verzích, jako projekt `tesseract-android-tools` a jeho upravená kopie s rozšířenou funkcionalitou `tess-two`.

Kód 4.1 demonstruje, jakým způsobem lze z obrazu získat pomocí knihovny `tess-two` přečtený řetězec. `TessBaseAPI` poskytuje metody pro přístup k přečteným znakům příp. slovům a dalším vlastnostem textu v obrazu. Každý symbol je reprezentován třídou `Pixa` knihovny `Leptonica`.

```
final String TESSBASE_PATH = "/sdcard/tesseract/";
final String DEFAULT_LANGUAGE = "eng";

public String getString(Bitmap bmp){
    // inicializace rozhrani
    final TessBaseAPI baseApi = new TessBaseAPI();
    baseApi.init(TESSBASE_PATH, DEFAULT_LANGUAGE);
    baseApi.setPageSegMode(TessBaseAPI.PageSegMode.PSM_AUTO_OSD);
    baseApi.setImage(bmp); // prirazeni obrazku

    // precteni obrazku
    String text = baseApi.getUTF8String();

    // ukonceni rozhrani
    baseApi.end();
    return text;
}
```

Kód 4.1: Přečtení textu z obrazu pomocí tess-two

[3] http://boofcv.org/

4.3 Překlad textu

Stejně jako v případě OCR, ani zde neumožňuje `Android API` sloužit přímo pro účely strojového překladu mezi jazyky. K překladu musíme využít tvorby třetích stran, přičemž máme několik možností realizace. Jak je uvedeno v kapitole 3, překlad může být realizován buďto pomocí některé z webových služeb či off-line.

Pro první případ platí, že většina překladových slovníků pracuje na bázi HTTP požadavku. Příklad požadavku na překlad serveru `Glosbe.com` můžeme spatřit v kódu 4.2. Specifikuje se zdrojový jazyk, cílový jazyk, text k přeložení a případně další atributy. Výhodou tohoto přístupu můžeme shledávat ve velice snadné implementaci a použitelnosti napříč všemi verzemi systému Android.

```
http://glosbe.com/gapi/translate?from=ces\
                                &dest=eng&format=json\
                                &phrase=kedluben&pretty=true\
```

Kód 4.2: Požadavek překladu slova na serveru Glosbe.com

Pro použití off-line řešení, nutno však podotknout komplikovanější, existuje několik způsobů:

- API již existující aplikace,
- Open Source aplikace typu slovník,
- implementace vlastního slovníku/překladače.

Intent API

Aplikační rozhraní lze v Android OS realizovat pomocí třídy `Intent`. Instance typu `Intent`, které jsme poskytli parametry pro překlad, je spuštěna a my vzápětí získáváme přeložená data. Tento přístup v hojné míře využívají např. čtečky elektronických knih, kdy po kliknutí na zvolené slovo dojde k zobrazení nového okna s překladem. Pro naše využití není zobrazení dalšího okna žádoucí a ne vždy lze toto chování eliminovat.

Open Source aplikace

Tato metoda spočívá ve využití existující implementace slovníku, jehož zdrojový kód je dostupný a může být upraven. V nejlepším případě by měl disponovat nástroji pro vložení vlastních slovníků. Příkladem takové aplikace je `quickdic` zmíněná v sekci 3.2.

Kapitola 5

Přehled dostupných aplikací

Tato kapitola popisuje aplikace, které se používají pro překlad textu z fotoaparátu mobilního telefonu. Nesoustředím se pouze na platformu Android, ale jsou zde zastoupeny i aplikace pro iOS nebo Windows Phone. V případě každé aplikace jsou shrnuty její nejpodstatnější vlastnosti, klady a zápory. Každá z aplikací navíc poskytuje odlišný způsob zobrazení výsledku, nasnadě je jejích porovnání. Vytvořená tabulka 5.1 předkládá základní porovnání.

Tabulka 5.1: Porovnání aplikací pro překlad

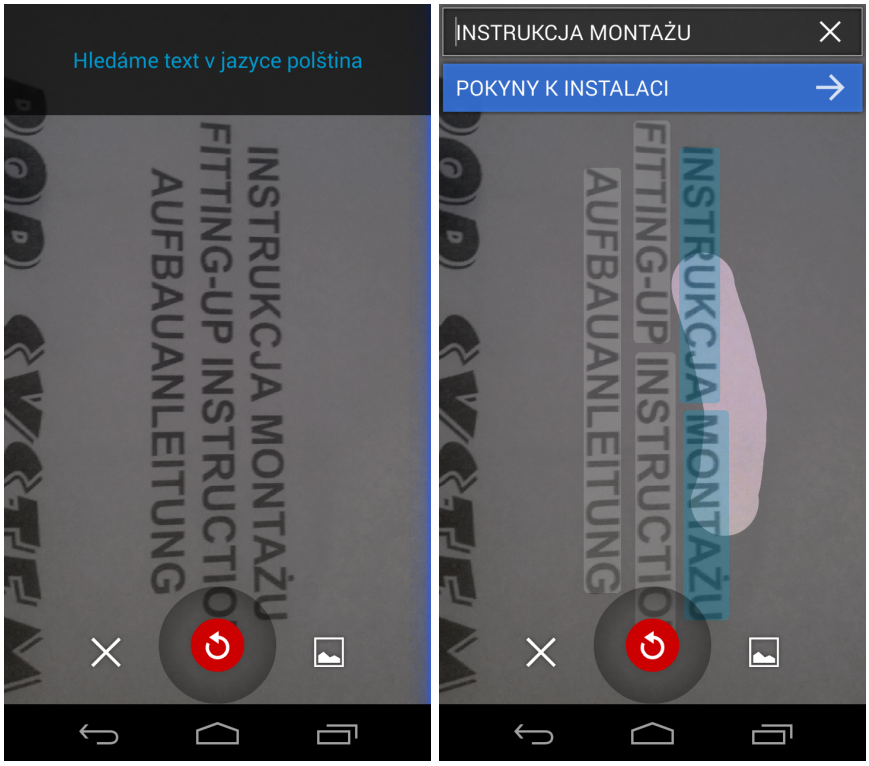
Jméno	Zdarma	Real-time	Off-line	Pozicováno	Čeština
Google Translate	ano	ne	ne	ne	ano
Google Goggles	ano	ne	ne	ne	ano
Bing Translator	ano	ano	ano	ano	ne
Lexicon	ano	ne	ano	ne	ano
Word Lens	ano*	ano	ano	ano	ne

Poznámky:
Real-time přeložený text zobrazen bez delší prodlevy
Off-line použití bez připojení k síti
Pozicováno překlad na displeji umístěn v oblasti původního textu
* aplikace zdarma, slovníky zpoplatněny částkou 5 $/kus

5.1 Google Translate

Nejrozšířenější zdarma dostupný překladač na platformě Android. Překlad textu z fotografie je k dispozici pouze od verze 3.0.4.

Aplikace v základu funguje buďto on-line nebo off-line v závislosti na tom, má-li uživatel staženy slovníky požadovaného jazyka. Toto však neplatí pro překlad z fotografie. Ten vyžaduje neustálé připojení k síti z důvodu zpracování obrazu na straně serveru. Musí dojít k vyfocení požadovaného textu, nestačí tedy na něj pouze zamířit objektiv fotoaparátu. Překlad proto neprobíhá okamžitě, nýbrž dochází k analýze trvající několik sekund. Průběh analýzy a zobrazení výsledků při překladu z polštiny do češtiny můžeme spatřit na obrázku 5.1. Slova nalezená v obrazu jsou zobrazena s orámováním, přičemž tahem prstu po displeji



(a) Analýza obrazu (b) Zobrazení výsledku

Obrázek 5.1: Použití aplikace Google Translate

určíme, které sekvence překládat. Přeložený text se objeví v modrém obdélníku v horní části displeje, nedochází k jeho zobrazení přímo v obrazu.

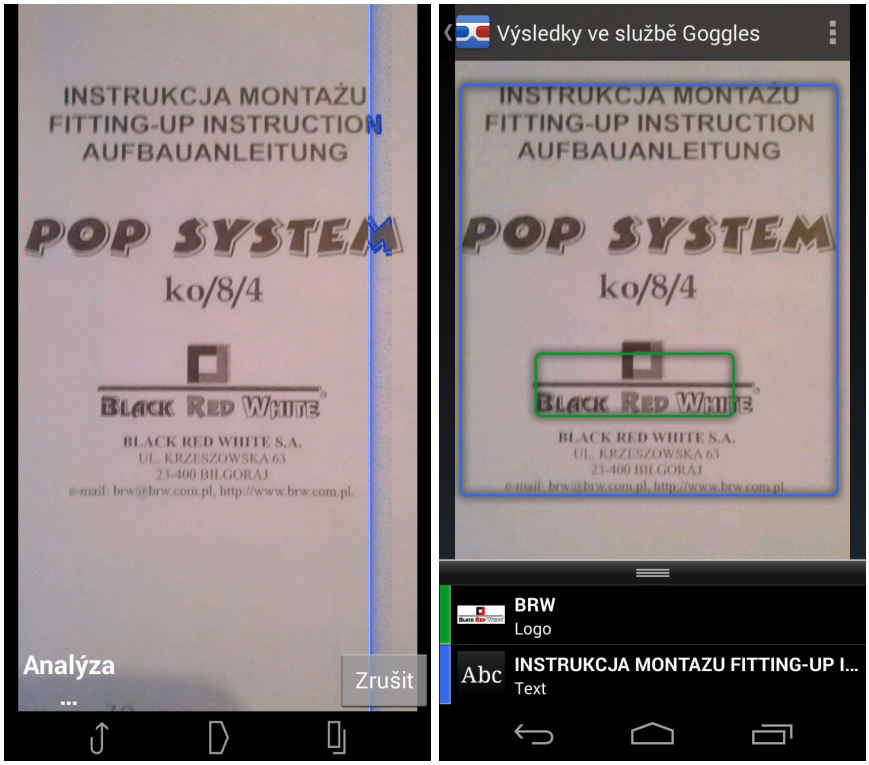
5.2 Google Goggles

Další aplikace z dílny společnosti `Google`. Jejím primárním účelem není překlad, jedná se spíše o vedlejší produkt díky komplexnosti celé aplikace. Aplikace slouží k extrahování co největšího množství dat z obrazu. Dokáže luštit Sudoku, rozpoznávat malby či stavby a v neposlední řadě také text. Po pořízení fotografie s textem probíhá analýza ve stejném duchu jako u předchozího překladače. Pokud dojde k nalezení textu, jedním kliknutím dojde k přesměrování do webového rozhraní a přeložení. Lepší představu o formě výsledku poskytuje obrázek 5.2.

5.3 Bing Translator

První aplikací, která se velmi přibližuje cílovému chování naší aplikace, je následující dílo od společnosti Microsoft. Můžeme si ji stáhnout zcela zdarma na Windows Marketplace[1]. Na obrázku 5.3 můžeme vidět hlavní obrazovku s překladem textu. Aplikace se nepokouší nahrazovat překládaný text přímo v obrazu, zobrazí přes něj poloprůhlednou vrstvu s výsledkem. Rychlost aplikace samotné i překladu hodnotím velice kladně.

[1] www.windowsphone.com/cs-cz/store/app/translator/2cb7cda1-17d8-df11-a844-00237de2db9e



(a) Analýza obrazu (b) Zobrazení výsledku

Obrázek 5.2: Použití aplikace Google Goggles

Obrázek 5.3: Použití aplikace Bing Translator

5.4 Lexicon

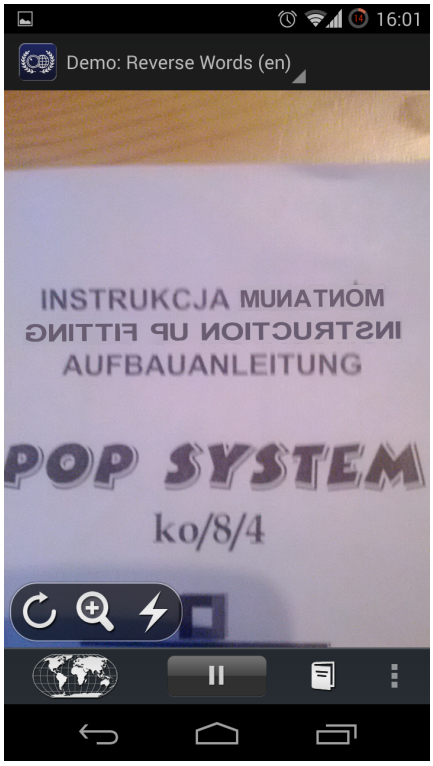
Open source[2] Android aplikace umožňující překlad s využitím uloženého slovníku. Z výčtu vybraných aplikací nejjednodušší. Nezvládá real-time zobrazení, přičemž po vyfocení požadovaného textu orámuje nalezená slova a po kliknutí dojde k jejich překladu. Jako OCR engine využívá knihovnu tess-two, která je zmíněna v sekci 2.3.5.

5.5 Word Lens

Nejzdařilejší aplikace z tohoto výčtu a pravděpodobně nejlépe fungující aplikace tohoto typu na trhu. Původně pro iOS, nyní také pro Android. Aplikaci lze stáhnout zdarma, překladové slovníky jsou však zpoplatněny. V demo režimu lze s textem vykonávat některé operace pouze pro demonstraci činnosti. Jedná se o vymazávání slov příp. jejich reverzaci. Příklad reverzace řetězce „INSTUCTION UP FITTING“ lze pozorovat na obrázku 5.4.

Aplikace má velmi rychlé reakce a již při sebemenším náznaku nalezení textu poskytuje překlad resp. úpravu textu. Dobře působí také překrytí starého textu textem novým, přičemž barva pozadí velmi věrně odpovídá původní předloze.

Jako zápor můžeme označit absenci češtiny. Aplikace sama vybízí k dodržování určitých pravidel při použití kupř. snažit se, aby písmo bylo na displeji vodorovně. Špatně rozpoznatelná jsou stylizovaná písma či vícebarevný text. S těmito problémy však bojují všechny aplikace.



Obrázek 5.4: Použití aplikace Word Lens

[2] https://bitbucket.org/eugenmihailescu/lexicon/overview

Kapitola 6

Návrh aplikace

Návrh před samotnou implementací, je stěžejním bodem celého života každé aplikace. V této kapitole jsou popsány základní komponenty a koncepty vyvíjené aplikace, včetně popisu užitého `Android API`, `OpenCV API` a `Tesseract API`.

6.1 Hlavní komponenty

Pokud činnost aplikace dekomponujeme na posloupnost menších akcí, aplikace musí být schopna vykonat:

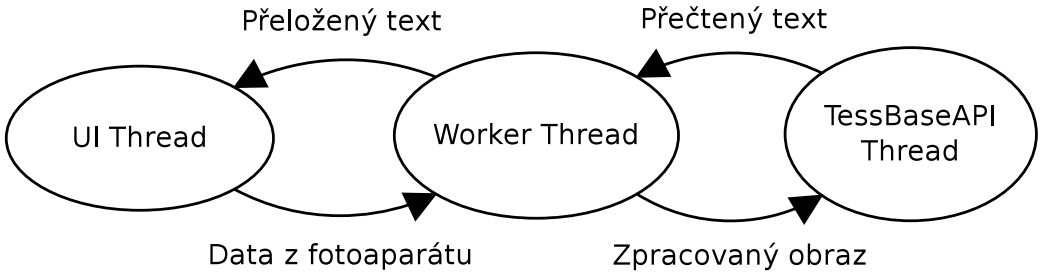
- **Získání dat z fotoaparátu a zobrazení na displeji** – tuto problematiku lze vyřešit pomocí `OpenCV API`. Konkrétně se jedná o třídu `JavaCameraView` pro zobrazení snímané scény na displeji a implementaci rozhraní `CvCameraViewListener2` pro zpřístupnění dat z fotoaparátu. Více o způsobu řešení nabízí kapitoly 7.4.3 a 4.1.2.
- **Zpracování obrazu** – stejně jako minulá úloha, je i tato zcela v režii `OpenCV`. Využitý OCR engine sice disponuje vlastním základním předzpracováním, je ale doporučeno využít některých dalších technik [21]. O použitých technikách předzpracování ve větší míře pojednává kapitola 6.3.
- **OCR** – z dostupných aplikačních rozhraní představených v kapitole 2.3 bylo pro potřeby rozpoznání textu v obrazu zvoleno `Tesseract API`. Pro potřeby implementované aplikace se jeví jako nejvhodnější kvůli vysoké konfigurovatelnosti, možnosti rozšíření aplikačního rozhraní a relativně rychlého zpracování. Popis jeho architektury a další vlastnosti jsou představeny v kapitole 6.2.
- **Překlad mezi jazyky** – lze uskutečnit s nápomocí tříd, které implementují rozhraní `Translater`. Příkladem je například třída `DictionaryManager`, která slouží k přístupu k vestavěnému off-line slovníku implementovaného pomocí `SQLite` databáze. V tomto případě je využito aplikačního rozhraní systému `Android`. Způsob vytváření databáze, popis dotazů a vybrané slovníky popisuji detailněji v kapitole 7.5.
- **Zobrazení boxů s novým textem** – existuje několik variant, jak nahrazovat původní slova v obrazu jejich překlady. Nejjednodušší a zároveň nejefektivnější spočívá v užití základních grafických prvků, tedy `TextView`, kterými disponuje systém Android. Zdůvodnění právě této volby a popis funkce lze nalézt v kapitole 6.4.

Nyní se pokusme výše zmíněné akce zasadit do kontextu běžících vláken. Na obrázku resp. diagramu 6.1 je vyznačen chod hlavních vláken aplikace. Jsou zde pro jednoduchost zanedbána ostatní vlákna systému, která náleží každému procesu.

UI Thread je obecné pojmenování pro hlavní vlákno každé aplikace. Jeho hlavním účelem je aktualizace grafického uživatelského rozhraní v závislosti na akcích vykonaných uživatelem nebo aplikací samotnou. Zároveň jde o jediné vlákno, které tuto možnost má. Pokud by došlo k zablokování tohoto vlákna nějakou dlouho trvající činností, grafické uživatelské rozhraní by přestalo reagovat a operační systém by nabídl uživateli ukončit aplikaci jako neodpovídající. V implementované aplikaci je jeho hlavní činností, jak vyplývá ze zmíněného diagramu, poskytnutí dat získaných z fotoaparátu dalšímu z vláken zvanému *Worker Thread* a zobrazení přeloženého textu.

Již zmíněný *WorkerThread* vykonává vícero činností. Slouží k předzpracování obrazu, překladu slov a pro zprostředkování komunikace mezi posledním z vláken *TessBaseAPI Thread* a hlavním vláknem. *TessBaseAPI Thread*, jak název napovídá, je vláknem OCR systému pro rozpoznávání textu v obrazu. Na vstup ve formě binárního obrazu odpovídá posloupností nalezených slov, jejich pozicí v obrazu a dalšími vlastnostmi.

Přístup užívající více vláken, nepřímo ovlivňuje také výkon, kdy jsou vlákna rozložena mezi více jader procesoru a dochází k paralelnímu zpracování.



Obrázek 6.1: Hlavní vlákna aplikace

6.2 Knihovna Tesseract

Jak již bylo naznačeno v kapitole 2.3.5, Tesseract engine složí k převodu obrazových dat obsahující text do jejich textové podoby. Tato kapitola popisuje architekturu, vnitřní mechanismy a možné způsoby rozšíření pro účely implementované aplikace.

6.2.1 Architektura

Na obrázku 6.3 je možné vidět blokový diagram popisující základní architekturu. Jedná se o klasické zřetězené zpracování. Na vstupu očekáváme binární obraz, nejlépe bez šumu, kde se písmo vyskytuje čistě horizontálně nebo čistě vertikálně. Případný náklon nebo pokřivená perspektiva nemá příliš velký vliv na rozpoznávání znaků samotných, ale problém může nastat při spojování slov. Systém nedokáže rozpoznat, zdali znak patří ke slovu nebo se nachází například o řádek výše.

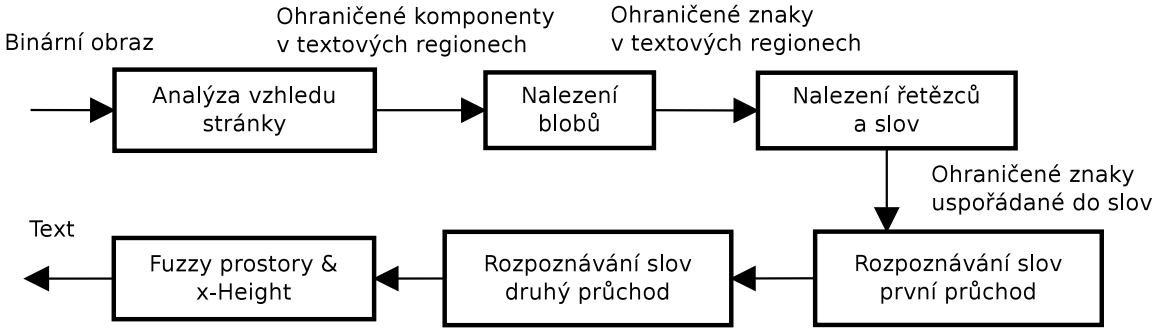
První blok architektury se zabývá komplexní grafickou analýzou stránky. Cílem této značně výpočetně náročné fáze zpracování je specifikace regionů a v nich se nacházejících komponent. Pod pojmem region rozumíme obrázek nebo prostor v obrazu s horizontálním či vertikálním textem. Příklad takových regionů je možno vidět na obrázku 6.2, kde jsou

červeně vyznačeny obrázky, modře horizontální text a žlutě vertikální text. Kromě výše zmíněných činností dochází navíc ke zjišťování průměrné velikosti písma, načež může být této informace v dalším bloku použito k ignorování falešných znaků. Použitá technika je nazývána *connected-component analysis*. Spočívá v hledání a propojování pixelů, mezi kterými existuje určitý vztah, kupříkladu oblastí s textem. Toto mimo jiné umožňuje snadné rozpoznávání nejen černého písma na bílém pozadí, ale také opačnou variantu [23].

Obrázek 6.2: Specifikace regionů

Další blok vyhledává v získaných textových regionech tzv. *blobs*. Pojmenování blob označuje klasifikační prvek, který reprezentuje zpravidla jeden znak, v horším případě část slova např. spojené znaky. Reprezentace spočívá v určení vnějšího ohraničení znaku a případných vnitřních oblastí (díra v písmenu „O“). Vykonává se tedy podobná činnost jako v předchozím bloku, nikoli však na úrovni celého obrazu.

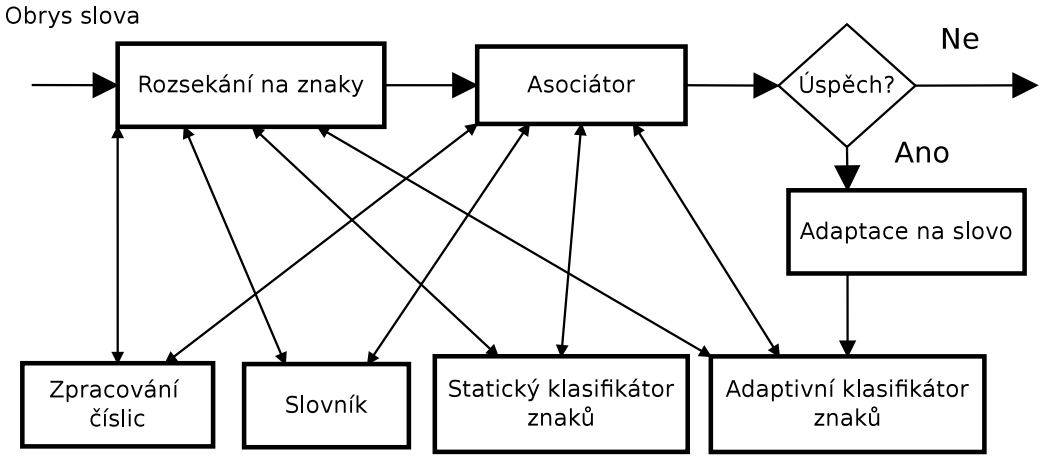
Po vytvoření blobů dochází k jejich spojování do řádků a řetězců, načež probíhá analýza náklonu a výpočet jejich proporcí. Pomocí těchto informací lze zjednodušit odvozování, které posloupnosti vytvářejí slova. Úroveň náklonu zároveň pro každé jednotlivé slovo určuje, jaký se má použít algoritmus pro dělení slov na znaky (segmentaci).



Obrázek 6.3: Architektura knihovny Tesseract [23]

Nalezené obrysy řetězců resp. slov jsou konečně předány bloku rozpoznávání slov, jehož hlavní úkol spočívá ve vyextrahování samotného textu. Blokové schéma tohoto konkrétního bloku specifikující jeho podbloky je možno spatřit na obrázku 6.4. Bloby ze kterých se slova skládají nemusí vždy reprezentovat jeden znak. Nejprve jsou všechny klasifikovány a na základě důvěry ve správnost rozpoznání se rozhoduje, jak budou nadále zpracovány. Rozpoznaná písmena vytvoří slovo a to je konfrontováno se slovníkem. Pokud neexistuje dostatečná důvěra v rozpoznané slovo, dochází k rozsekání potenciálně špatných blobů na místech zvaných *chop points* (body k rozseknutí). Taková místa jsou seřazena v prioritní frontě a po každém rozseknutí a rozpoznání dochází opět ke kontrole ve slovníku. V případě, že není důvěra po úpravě vyšší, změny se zahodí v opačném případě se změny uloží. Je důležité poznamenat, že první „lepší“ řešení neznamená automaticky ukončení zpracování znaku, ale dojde k otestování všech možností.

Pokud postupné rozsekávání stále neprodukuje lepší výsledky, přichází na řadu zcela opačný přístup, kdy dojde k rozsekání všech míst a ta jsou následně spojována. Pozitivním efektem tohoto přístupu je rozeznávání defektních znaků, které byly omylem rozděleny a mohou být tedy takto opět spojeny. Zmiňovaný blok se nazývá asociátor a jeho jádro tvoří algoritmus A* pro hledání optimální cesty v ohodnoceném grafu. Ve skutečnosti ani nedochází k vytváření grafu, ale je tvořena hashovací tabulka s otestovanými případy. Opět dochází k porovnávání se slovníkem slov a následném výběru nejvhodnějšího kandidáta.



Obrázek 6.4: Blokový diagram zpracovávání slov knihovnou Tesseract [23]

Knihovna disponuje dvojicí klasifikátorů, statickým a adaptivním. Statický byl navržen tak, aby se dokázal velmi dobře vypořádat s obecně různou velikostí, tvarem (tučné, kurzíva) a typem písma. Pro účely klasifikaci musíme být schopni nějakým způsobem popsat uložené prototypy a zároveň neznámé prvky, které klasifikujeme. Každý rozpoznávány znak popisuje množina úseček, každá definovaná trojicí `(x pozice, y pozice, úhel)`. Úsečky, kterých se v jednom *feature vektoru*[1] nachází 50 až 100, se snaží co nejvěrněji zachytit tvar znaku. Vektory prototypů obsahují o jednu dimenzi navíc a to délku každé z úseček. Je jich tedy logicky méně a to cca 15.

Statická klasifikace je dvouúrovňová. V prvé dochází k hrubé klasifikaci pomocí třídimenzionální look-up tabulky, jejíž návratovou hodnotu tvoří vektor vah blízkosti k dané třídě. Provádí se z důvodu eliminace naprosto odlišných tříd. V druhé probíhá detailnější

[1] množina nebo n-tice hodnot, která by měla co nejvěrněji popisovat zpracovávaný prvek, aby mohl být spolehlivě klasifikován

porovnávání neznámých znaků s prototypy tříd, které uspěly v prvním kroku, tentokrát však s využitím všech vlastností.

Oproti zpravidla obecnému statickému klasifikátoru, se ten adaptivní, jak název napovídá, snaží adaptovat na písmo v konkrétně zpracovávaném obrazu. Každé slovo s vysokou důvěrou se použije jako trénovací množina tohoto klasifikátoru pro účely zpracování dalších slov respektive znaků.

Aby nedocházelo k využití takto natrénovaných dat pouze ve slovech, která se v obrazu vyskytují níže a více vpravo, dochází k rozpoznávání slov ve dvou bězích. V druhém běhu jsou však ignorována slova, jejichž výše důvěry dosahuje zvolené hodnoty již po prvním zpracování.

Poslední komponenta spojuje či odděluje slova na základě zkoumání velikostí mezer a linií kopírujících proporce písma na daném řádku.

6.2.2 Aplikační rozhraní a jeho možnosti

Aplikační rozhraní této knihovny spočívá hlavně ve dvojici tříd a to `TessBaseAPI` a `ResultIterator`. První z nich zaopatřuje inicializaci, veškerou konfiguraci knihovny, průběh zpracování aj. Druhá hlavní třída slouží výhradně k průchodu, analýze a zpracování získaných výsledků.

6.2.3 Způsob rozšíření API knihovny Tesseract

Vzhledem k faktu, že se jedná o open source knihovnu, lze ji libovolně rozšiřovat nebo upravovat. Již bylo zmíněno dříve, implementačním jazykem je C++ a je přidáno Java rozhraní pro snadné využití na operačním systému Android. Tyto skutečnosti mírně komplikují způsob přidávání nových funkcionalit.

Jedním z mnou implementovaných rozšíření je metoda třídy `ResultIterator` pro získání umístění slov v obrazu. Java rozhraní pro komunikaci s nativní knihovnou demonstruje kód 6.1. Je zapotřebí specifikovat jak veřejnou metodu, tak metodu s klíčovými slovy *private*, *static* a *native*, která slouží jako reference na nativní metodu. Ta je uvedena včetně komentářů, v příloze B. Její název odpovídá kombinaci slova `Java`, plného názvu balíčku, kde je nativní metoda umístěna a názvu metody.

```
public int [] getBounds(int level) {
  int [] values = nativeGetBounds(mNativeResultIterator, level);
  return values;
}

private static native
  int [] nativeGetBounds( int nativeResultIterator, int level);
```

Kód 6.1: Metoda třídy `ResultIterator` zprostředkovávající volání nativní metody

6.3 Předzpracování obrazu

Pro předzpracování obrazu je využita knihovna `OpenCV` a všechny metody zmíněné v této sekci se vztahují právě k této knihovně (pokud není řečeno jinak). Během předzpracování jsou postupně vykonány dva kroky, přičemž první spočívá ve dvojité změně velikosti obrazu a druhý v inteligentním prahování obrazu.

6.3.1 Změna velikosti

Za účelem snížení množství šumu je vhodné zmenšit velikost obrazu a následně jej zvětšit na původní velikost. Pro tyto účely lze použít metodu `Imgproc.resize()`. Mezi procesem zpětného zvětšení je možné obraz ještě vyhladit např. použitím operátoru `closed`. To lze zařídit pomocí metody `Imgproc.morphologyEx()` s vybraným parametrem např. *closing*. Tato operace eliminuje šum a celkově drobnější nepřesnosti v obrazu, protože převádí miniaturní oblasti zcela obklopené oblastmi jinými, právě na hodnoty z okolí.

6.3.2 Prahování

Prahování zajišťuje převod obrazu v odstínech šedi do černobílé varianty. Barevné obrazy musí být nejprve převedeny právě do šedých odstínů. To lze vykonat pomocí statické metody *Imgproc.cvtColor()* s parametrem `Imgproc.COLOR_RGB2GRAY`, který specifikuje kód konverze. Pro prahování samotné je využita metoda *Imgproc.threshold()*. Kromě parametru `Imgproc.THRESH_BINARY_INV`, který zajišťuje převod do binární reprezentace a je přidán také parametr `Imgproc.THRESH_OTSU`, který automaticky vybírá práh na základě předchozí analýzy obrazu.

6.4 Nahrazení původního textu

Nahrazovat původní text v obrazu lze dvojím způsobem. První spočívá přímo v úpravě vstupního obrázku (úprava pixelů). Tento přístup není příliš vhodný, protože využívá výpočetně náročné operace pro práci s obrazovými daty resp. jejich maticovou reprezentací.

Druhý způsob, využitý v aplikaci, přesouvá logiku vykreslování boxů s přeloženým textem na operační systém. Obraz se zdrojovým textem je překryt průhlednou vrstvou, do které jsou postupně již zmíněné boxy umisťovány, přičemž je u nich specifikováno několik parametrů typu pozice (parametr `margin`), barva pozadí, zarovnání na střed atp.

Kapitola 7

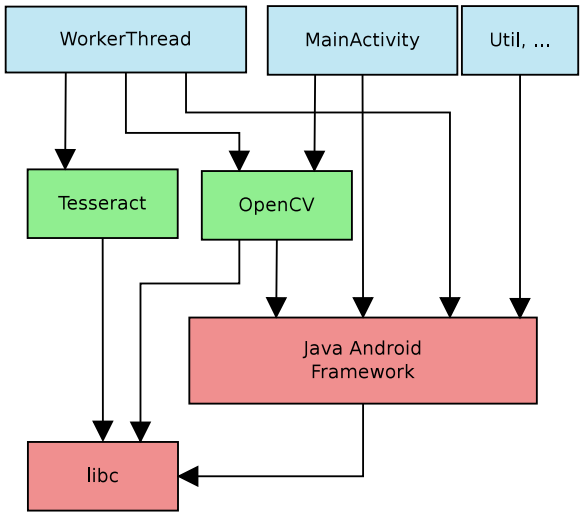
Implementace

Tato kapitola navazuje na předchozí, ve které jsou popsány hlavní komponenty aplikace a popis některých aspektů použitých knihoven. Uvádím zde konkrétní prvky implementace, způsoby řešení důležitých problémů a v neposlední řadě architekturu celé aplikace.

7.1 Architektura implementace

Na obrázku 7.1 lze spatřit vztah mezi implementovanými třídami, užitými knihovnami atp. Šipky znázorňují závislosti komponent na těch, do kterých směřují. Úroveň abstrakce je nejvyšší v horních patrech diagramu, kde jsou s modrým podbarvením vyznačeny implementované třídy `WorkerThread`, `MainActivity` a další.

Zeleně jsou označeny použité externí knihovny. V prvé řadě byla využita knihovna `Tesseract` ve verzi `3.02`. Provedl jsem však v ní několik změn, v aplikaci tedy nelze použít knihovnu referenční. Knihovna je využívána pouze třídou `WorkerThread`, kterou popisuji detailněji v sekci 7.3. Jelikož je víceméně celá její implementace v C/C++, označil jsem ji jako závislou pouze na základní knihovně `libc`, která se nachází v Android OS. Druhou použitou externí knihovnou je `OpenCV` ve verzi `2.4.8`. Ta na rozdíl od první knihovny disponuje jak nativními funkcemi, tak Java rozhraním, což odpovídá požadavkům na knihovnu `libc` přímo, tak i na příslušné třídy rozhraní systému.



Obrázek 7.1: Architektura implementované aplikace a využitých knihoven

Červeně jsou na obrázku vyznačeny podpůrné třídy aplikačního rozhraní systému Android. Jedná se o třídy typu `Camera`, `Activity`, `Thread` aj.

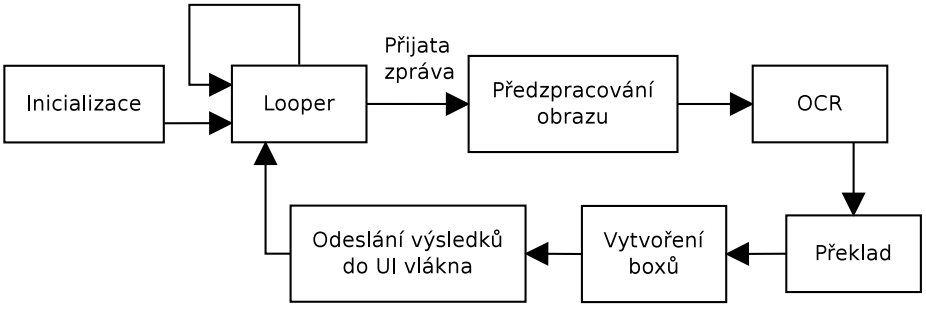
7.2 Android Manifest

Android Manifest je pojmenování pro XML[1] soubor se stromovou strukturou, který se nachází v kořenovém adresáři každého Android projektu. Systému poskytuje základní informace o aplikaci a slouží jako jakýsi rozcestník. Příkladem informací, kterými disponuje, může být pojmenování balíčku s aplikací, výčet použitých aktivit či služeb, specifikace minimální podporované verze systému Android a v neposlední řadě také seznam oprávnění. Při instalaci aplikace do zařízení, je tedy uživateli zřejmé, do jaké míry může zasahovat do systému.

Aplikace vyžaduje pro běh jedno povolení a to `android.permission.CAMERA`, tedy použití fotoaparátu. Pro specifikaci oprávnění se používá tag `<uses-permission>`. Jelikož je fotoaparát nutnou podmínkou pro běh, je žádoucí rovněž specifikovat tag `<uses-feature>` s hodnotami `android.hardware.camera`, `android.hardware.camera.autofocus`, čímž bude v případě uložení do Obchodu Play zamezeno možnosti instalace na zařízení bez fotoaparátu a autofocusu.

7.3 WorkerThread

`WorkerThread` je název třídy, která je potomkem třídy `java.lang.Thread`. Jedná se o implementaci výpočetního vlákna, které doplňuje funkci hlavního (označovaného také jako UI) vlákna aplikace. Existuje několikero způsobů, jak mohou spolu vlákna komunikovat, v aplikaci je využit princip odesílání zpráv.



Obrázek 7.2: Životní cyklus vlákna `WorkerThread`

7.3.1 Životní cyklus

Jak je možné spatřit na obrázku 7.2, životní cyklus samozřejmě začíná inicializací komponent, které využívá. V tomto případě se jedná o inicializaci knihovny `Teseseract`, přes instanci hlavní třídy `TessBaseAPI`. Je zapotřebí definovat jazyky, které specifikují text čtený z obrazu. To lze provést např. pomocí řetězce `"eng+ces"`[2] uvedeného jako parametru

[1] eXtensible Markup Language

[2] označení jazyků musí být uvedena dle formátu ISO 693-3 a oddělena znakem `+`

v metodě *init()* společně s cestou k datům potřebným ke klasifikaci a módem OCR. Zvolené jazyky slouží také k určení slovníků, jejichž slova se použijí pro účely klasifikace. Dále pomocí metody *setVariable()* specifikuji tzv. *char_whitelist*, tedy seznam povolených znaků.

Další komponentou je `Looper`, který opětovně kontroluje zda-li se nenachází ve frontě vlákna nějaká zpráva pro zpracování. Pojmenování může být mírně zavádějící, protože navozuje dojem nekonečného cyklu, ale ve skutečnosti vlákno nedělá nic až do přijetí zprávy. `Looper` pracuje ve společnosti instance třídy `Handler`, která umožňuje interakci jiných vláken s vláknem samotným pomocí přijímání zpráv.

Následuje předzpracování obrazu, kterému se ve větší míře věnuji v sekci 6.3 a konečně získání textu z obrazu. Formátů, v jakých lze text samotný získat, existuje větší množství. Toto řešení využívá třídy `ResultIterator`, která iteruje přes veškerý získaný text slovo po slovu. To platí pouze v případě, používáme-li volání metody *getUTF8Text()* s parametrem `RIL_WORD`. Lze totiž iterovat po jednotlivých symbolech, řádcích, odstavcích či blocích.

Takto získaná data jsou přeložena do cílového jazyka a zaznamenána jako instance třídy `TextViewBox`. Každý takto vytvořený box obsahuje:

- originální přečtené slovo,
- překlad slova (pokud existuje),
- souřadnice specifikující umístění slova v obrazu,
- barva pozadí, ve kterém je text umístěn, pro pozdější obarvení pozadí `TextView`.

V další fázi dochází k předání pole vytvořených boxů do hlavní aktivity aplikace, kde dojde k jejich vykreslení přes analyzovaný obraz.

7.3.2 OnBoxesChangeListener

Pro účely snadného informování hlavní aktivity o nově získaných datech z obrazu se velmi hodí implementované rozhraní `OnBoxesChangeListener`. Aplikace se snaží dodržovat zažité implementační principy z Android frameworku, kterými callback[3] funkce jistě jsou.

Celý princip spočívá v tom, že hlavní aktivita rozhraní implementuje a její referenci předá výpočetnímu vláknu. Až to dokončí svou práci, zavolá metodu *OnBoxesChange()* a jako parametr předá seznam boxů. Je důležité si uvědomit, že ke grafickému uživatelskému rozhraní může přistupovat pouze hlavní vlákno aplikace a je tedy zapotřebí vynutit proces vykonávání právě v něm. Kód 7.1 demonstruje metodu, pomocí které lze tohoto chování docílit.

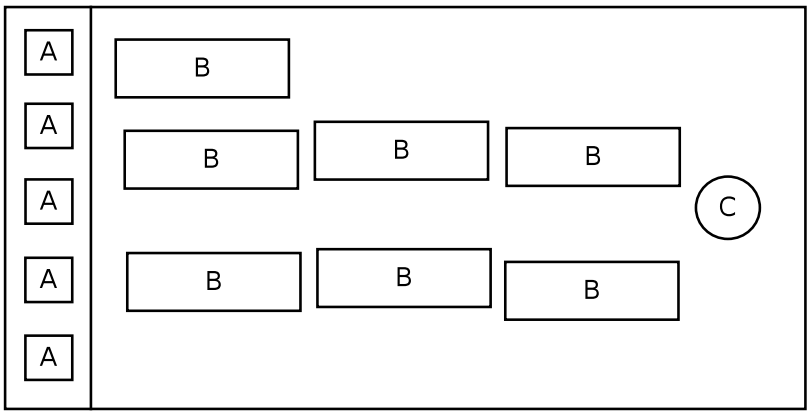
```
runOnUiThread(new Runnable() {

  @Override
    public void run() {
     // vlozeni boxu s~prelozenym textem do UI
    }
  });
```

Kód 7.1: Způsob zajištění běhu nějaké části implementace v hlavním vláknu aplikace

[3] funkce, volaná jako reakce na událost, která v systému nastala

7.4 Grafické uživatelské rozhraní

V této sekci se zabývám implementací grafického uživatelského rozhraní. Na obrázku 7.3 můžeme spatřit základní koncept, jak by měla vypadat hlavní obrazovka. Očekává se použití v `landscape` režimu, tedy režimu na šířku. Komponenty se symbolem `A` specifikují prvky menu, konkrétně mód fungování aplikace, zdrojový jazyk, cílový jazyk, zobrazení/skrytí boxů s přeloženým textem apod. Pod symbolem `B` se ukrývají boxy obsahující překlad slova, které překrývají a konečně symbol `C` označuje tlačítko pro pozastavení zobrazování dat z fotoaparátu na displeji a započetí rozpoznávání okamžitě po zaostření.



Obrázek 7.3: Struktura hlavní obrazovky aplikace

Veškeré komponenty GUI v systému Android jsou potomky třídy `View`. Některé z nich fungují jako kontejnery, určující jejich vnořeným potomkům nějaké uspořádání a jiné se vyznačují konkrétním chováním. `View` komponenty mají vždy stromovou strukturu a lze ji plnohodnotně specifikovat jak pomocí xml souboru, tak pomocí Java kódu.

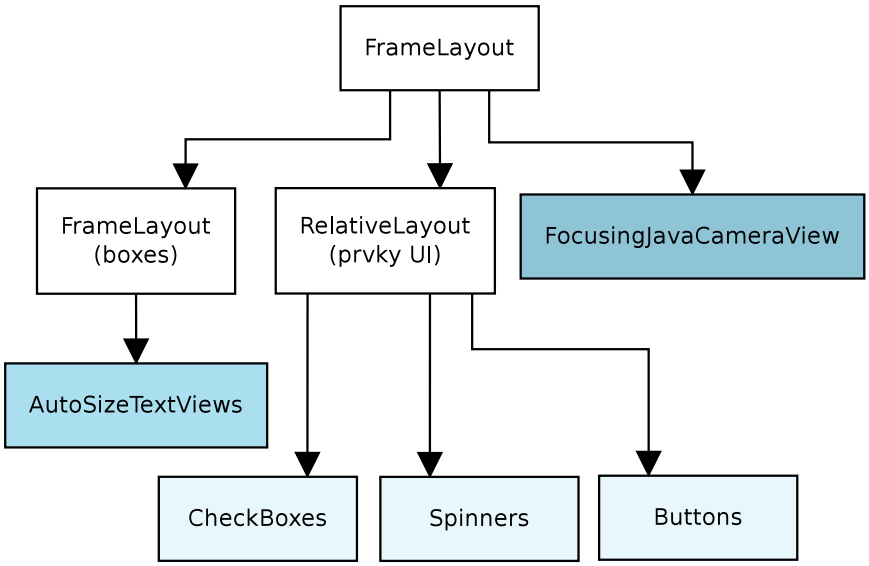
Na obrázku 7.4 lze vidět, za účelem zjednodušení mírně pozměněnou strukturu, která specifikuje hlavní obrazovku aplikace. Komponenty, které jsou vyznačeny modrou barvou, jsou přímo viditelné na obrazovce, ostatní, s bílým pozadím slouží jako kontejnery. Zároveň platí, že čím tmavší odstín modré barvy blok má, a čím výše v diagramu se nachází, tím dále je posunut do pozadí. Na pomyslné `z`-ose tedy nikdy nedojde k překrytí uživatelského rozhraní boxem s textem apod.

Pokud se zaměříme na popis jednotlivých komponent, pozici kořenové komponenty zaujímá `FrameLayout`. Jeho specifikem je, že všechny potomky, pokud explicitně neurčíme jinak, umisťuje do středu obrazovky. Atributy specifikující výšku a šířku jsou u všech tří potomků nastaveny na hodnotu `fill_parent`, což nám říká, že jsou roztaženy přes celou obrazovku. V praxi tedy fungují jako tři vrstvy.

První z nich slouží k zobrazení aktuálního náhledu z fotoaparátu a je instancí třídy `FocusingJavaCameraView` (více v sekci 7.4.3). Následuje opět kontejner typu `FrameLayout`, tentokrát na umisťování boxů s textem. Každý box reprezentuje `AutoSizeTextView` jehož parametry jsou nastaveny tak, aby v co největší míře podobaly překrývané oblasti. Tím je myšlena například barva pozadí či velikost textu.

Poslední z trojice vrstev sumarizuje samotné ovládací prvky uživatelského rozhraní. Jedná se o různá tlačítka, komponenty typu `Spinner` (výsuvná nabídka) či `CheckBox`. Kontejner, jehož jsou tyto komponenty potomky, je typu `RelativeLayout`. Ten disponuje možností komponenty uspořádat tak, že určíme, kde se má nacházet vzhledem ke komponentám

ostatním. Příkladem nám může být atribut `android:layout_below="@id/spinner"`, který komponentě říká, že se má nacházet pod komponentou s identifikátorem `spinner`. Takto můžeme vytvořit velice robustní, na rozlišení či velikosti displeje nezávislé, uživatelské rozhraní.



Obrázek 7.4: Stromová struktura znázorňující hierarchii GUI

7.4.1 Výpočet barvy pozadí boxů

Pro kvalitní „umístění“ boxu do obrazu je žádoucí, aby se jeho barva co nejvíce podobala původní předloze. Pro účely takového chování dochází k výpočtu nové barvy pozadí dle barev vrcholů nejmenšího obdélníku obepínajícího vybraný text. Výpočet probíhá postupným průměrováním všech barevných složek pixelů, které se naházejí v rozích nejmenšího obdélníku. Podle vypočtené barvy pozadí je následně vybrána jedna ze dvou barev písma (tmavá/světlá) pro snadnou čitelnost na vypočítaném pozadí.

7.4.2 AutoSizeTextView

Třída `AutoSizeTextView` jakožto přímý potomek třídy `TextView` slouží k zobrazování přeložených slov. Rozdíl spočívá v automatické úpravě velikosti písma v závislosti na délce zobrazovaného textu. Ve všech případech, kdy dochází ke změně textu resp. velikosti `view` dochází k přepočítání dispozic textu. Přepočet se provádí pomocí binárního prohledávání. Vždy, když zvětšíme resp. zmenšíme velikost písma na dvojnásobnou resp. poloviční velikost oproti průměru aktuálních hraničních hodnot, testujeme, zdali slovo sedí do boxu dostatečně přesně. Test analyzované velikosti probíhá s využitím třídy `Paint` a příslušné metody *getTextBounds()*. Ta dává k dispozici informace o výšce a šířce textu o zvolené velikosti písma.

7.4.3 FocusingJavaCameraView

`FocusingJavaCameraView` poskytuje plochu pro náhled z fotoaparátu. Nepřímým předkem je třída `SurfaceView`, která poskytuje metody pro zjednodušené vykreslování bitmap

přímo na plátno s využitím `SurfaceHolder`. V hierarchii dědících tříd se nacházejí dva mezičlánky a to ve formě `CameraBridgeViewBase`, které slouží k provázání Android rozhraní fotoaparátu a OpenCV knihovny a dále `JavaCameraView` implementující rozhraní pro získání/uvolnění přístupu k fotoaparátu.

Samotný účel vytvoření třídy `FocusingJavaCameraView` spočívá v nemožnosti získat informaci o tom, zda je aktuálně zobrazovaný obraz zaostřen. V základním nastavení kamera pracuje (pokud to umožňuje) v režimu kontinuálního zaostřování. Nově lze požádat o informaci o aktuálním stavu zaostření a také zadat konkrétní požadavek na zaostření.

7.5 Překlad textu

Pod posledním velkým celkem, který je součástí aplikace, si můžeme představit implementaci překladu. Ten funguje v off-line režimu, přičemž data pro překlad jsou uložena v `SQLite` databázi. Tento typ databáze má podporu přímo v Android frameworku, což značně usnadňuje její správu a manipulaci s obsahem.

Pro přehlednost, třídy, které slouží k překladu, implementují rozhraní `Translater`. To vyžaduje implementaci metody *translate(String from, String to, String [] words)* a její přetížené varianty pouze s jedním slovem a nikoli polem slov. Parametry *from* a *to* nesou zkratku zdrojového a cílového jazyka.

7.5.1 Užité slovníky

V aplikaci jsou užity slovníky ze dvou zdrojů. Pro anglicko-český a česko-anglický překlad posloužil slovník dostupný on-line na `slovnik.zcu.cz`, který je dílem Milana Svobody[4]. Pro neanglické překlady jsou použity slovníky projektu `www.dicts.info`. Konkrétně se jedná o data z databáze `Wiktionary`.

Všechny zde zmíněné slovníky jsou šířeny pod GNU FDL[5] licencí a to umožňuje s nimi nakládat, s povšimnutím na potřeby aplikace, téměř neomezeně.

7.5.2 SQLite databáze

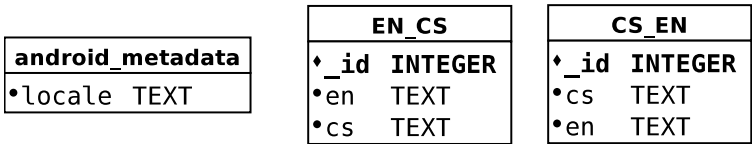
Jak bylo zmíněno výše, pro uložení slovníkových dat je využita `SQLite` databáze. Pro účely vyhledávání překladů poskytuje dostatečnou rychlost. Schéma databáze pro kombinaci jazyků čeština-angličtina můžeme spatřit na obrázku 7.5. Pro další jazyky existují analogické tabulky. Povinná tabulka `android_metadata` obsahující jeden sloupec `locale` slouží výhradně pro účely systému. SQL dotazy jsou vzhledem ke schématu velice jednoduché, zpravidla v následujícím tvaru:

```
SELECT <cilovy jazyk> from <ZDROJOVY JAZYK>_<CILOVY JAZYK>
       WHERE <zdrojovy jazyk> = <prekladane slovo>;
```

7.5.3 DictionaryManager

`DictionaryManager` vykonává práci jakéhosi prostředníka ke komunikaci s `SQLite` databází reprezentující slovník s překlady. Jeho rodičovská třída `SQLiteHelper` vyžaduje implementaci metod *onCreate()* a *onUpgrade()*, avšak pro naše využití, kdy je databáze dopl-

[4] http://fek.zcu.cz/osobni.php?IDWorker=62
[5] http://www.gnu.org/copyleft/fdl.html



Obrázek 7.5: Schéma databáze slovníku pro kombinaci jazyků čeština-angličtina

něna v APK balíčku ztrácí implementace těchto metod smysl. Běžně se v nich nachází SQL příkazy pro vytváření, mazání, či jinou úpravu jednotlivých tabulek.

Databáze s daty pro překlad se v našem případě nachází v adresáři `assets/`, který shromažďuje dodatečné soubory aplikace. Přístup k těmto souborům zajišťuje `AssetManager`, který umožňuje otevření souboru pro čtení a v druhém kroku i kopírování do adresáře s databázemi, tedy do `/data/data/<nazev_balicku>/databases/`. Kopie se provádí pomocí standardních Java vstupně/výstupních tříd `InputStream` a `OutputStream`.

Přístup k databázi

Za účelem snadného přístupu ke slovníkům z celé aplikace lze referenci na instanci slovníku získat přes statickou metodu *getDicManager()* základní třídy `App`. Její metoda *onCreate()* poskytuje prostor pro inicializaci příp. vytvoření slovníku. Ukončení přístupu ke slovníku je vhodné v metodě `onTerminate()`, kdy dochází ke kompletnímu vymazání aplikace z operační paměti. Aplikace v základu využívá standardní třídu `Application` a je tedy zapotřebí ji v Android Manifestu pomocí atributu `android:name` umístěného v tagu `<application>`, nahradit celou cestou k nově vytvořené třídě, v našem případě `cz.skywall.dp.App`.

Kapitola 8

Testování a experimenty

Tato kapitola je zaměřena na testování aplikace a experimentování s rozličnými parametry vstupního obrazu. Nejdříve jsou popsány možnosti aplikace, v souvislosti s grafickým uživatelským rozhraním a následně experimenty samotnými. Většina z experimentů bude zkoumat vlastnosti, které jsou uvedeny v tabulce 8.1. Pro účely testování bylo využito zařízení `Samsung Galaxy Nexus`, které disponuje dvoujádrovým procesorem TI OMAP4460[1] o taktu jádra 1,2 GHz. Telefon využíval operační systém Android verze `4.4.2` v komunitní úpravě od Cyanogenmod Inc[2].

Tabulka 8.1: Šablona tabulky experimentů

Vzorek	KRozpS	CHRozpS	KPřelS	CHPřelS	Čas [ms]	X
1	-	-	-	-	-	-

Poznámky:
`KRozpS` – Korektně rozpoznaná slova.
`CHRozpS` – Chybně rozpoznaná slova.
`KPřelS` – Korektně přeložená slova (vstupní hodnotou překladu je počet úspěšně přečtených slov, nikoli celkový počet slov).
`CHPřelS` – Chybně přeložená slova.
`Čas [ms]` – Doba běhu překladu v milisekundách.
`X` – Další parametr/y specifické pro konkrétní experiment.

8.1 Užití aplikace

Na obrázku 8.1 lze možné vidět snímek obrazovky s přeloženým textem. V tomto případě se jedná o text „Klub Českých turistů Třinec“. Tlačítka a přepínače v levé straně obrazovky, reprezentující formu menu, jsou specifikována následujícím chováním:

- `BL` – Přepínač mezi zobrazením průběhu (scény) z fotoaparátu v odstínech šedi či barevným zobrazením. Šedá verze může uživateli nápomoci vybrat ideální podmínky pro pořízení snímku, který bude zpracován. Velký vliv má zejména konstantní osvětlení celé scény.

[1] http://www.ti.com/product/omap4460
[2] http://download.cyanogenmod.org/?device=maguro

Obrázek 8.1: Snímek obrazovky demonstrující možnosti aplikace.

- `BX` – Přepínač pro zobrazení resp. skrytí boxů s přeloženým textem.
- `MO` – Přepínač mezi dvěma módy, ve kterých může aplikace fungovat. Prvním z nich je vyfocení scény a její následné zpracování. Druhý mód umožňuje kontinuální zobrazování scény a postupný překlad.
- `CS` – Tlačítko pro výběr zdrojového jazyka, při pořizování snímku nastavený na češtinu.
- `EN` – Tlačítko pro výběr cílového jazyka.

8.2 Počet slov v obrazu

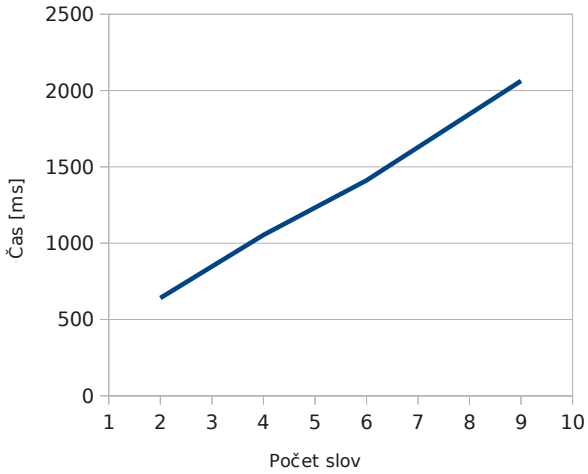
Tento experiment zkoumá vztah mezi počtem slov ve snímaném obrazu vzhledem k rychlosti zpracování využitou knihovnou. Pro měření doby zpracování je užito metody *nanoTime()* třídy `System`, která udává počet nanosekund od určité události v systému. V případě mobilních telefonu se nejčastěji jedná o dobu od zapnutí zařízení. Časový interval mezi dvěma voláními této metody lze získat prostým rozdílem získaných dvou časů.

V tomto experimentu byl postupně analyzován obraz C.1 (příloha C) s rozličným počtem slov, přičemž hodnoty v tabulce udávají průměrné hodnoty tří za sebou jdoucích nezávislých měření. Jednalo se o text v češtině obsahující diakritiku. Hlavním zkoumaným faktorem je zde rychlost zpracování a následného zobrazení překladu na displeji. Výsledky můžeme spatřit v tabulce 8.2.

Z grafu 8.2 lze vypozorovat, že rychlost zpracování dosahuje téměř lineární závislosti na počtu zpracovávaných slov resp. znaků. Pro vyšší počty slov (více než 10) trvá vyhodnocení několik sekund a uživatel může nabýt dojmu, že aplikace neodpovídá. Nejvýhodnější se tedy jeví využití, jak bylo zpočátku zamýšleno, pro překlady kratších textů, například informačních tabulí.

Tabulka 8.2: Zkoumání rychlosti aplikace dle počtu slov v obrazu

Vzorek	Počet slov (znaků)	KRozpS	CHRozpS	KPřelS	CHPřelS	Čas [ms]
1	2 (11)	2	0	2	0	641
2	4 (24)	3,66	0,33	3	0,66	1054
3	6 (41)	5	1	4	2	1412
4	9 (60)	7,66	1,33	7	1,33	2063



Obrázek 8.2: Závislost rychlosti běhu na počtu slov ve vstupním obrazu

8.3 Zdrojový jazyk

Následující experiment zkoumá výsledky překladu v závislosti na jazyku zdrojového textu resp. textu v obrazu. Byly vykonány celkem čtyři měření, opět každé měření třikrát. Nejdříve byl ověřován překlad jednoduchých slov z češtiny do angličtiny, následovaný překladem slov méně častých. Poté dochází k překladu textu s velice podobnými výrazy z angličtiny do češtiny.

Výsledky znázorněné v tabulce 8.3 udávají, že mírně lepší výsledky generuje překlad z anglického jazyka do češtiny. Toto chování má několik příčin. První z nich je dostupnost slovníku slov přímo u dat pro knihovnu Tesseract. Jak je popsáno v předešlých kapitolách, slovník je použit k ověřování existence slov. Pro češtinu takový slovník není a jeho existence by navíc zvětšila velikost výsledného `apk` balíčku. Druhou příčinu shledávám ve velikosti anglické abecedy, která neobsahuje diakritiku a poskytuje proto méně prostoru pro případné chyby, kupř. vynecháváním háčků či čárek.

Tabulka 8.3: Zkoumání výstupu aplikace dle zdrojového jazyka

Vzorek	Počet slov	Zdroj. jazyk	KRozpS	CHRozpS	KPřelS	CHPřelS
1	11	EN	10	1	10	1
2	11	CS	9,66	0,33	9,33	0,66
3	7	EN	7	0	5	2
4	7	CS	5,66	1,33	3.33	2

Vzhledem k vysoké univerzalitě vzhledem k natočení písmen a velkého množství písem celkově, v některých případech docházelo k zaměňování některých dvojic písmen. Příkladem mohou být písmena „n“ a „u“. Bylo možné se tak setkat s výrazy jako „settiugs“ místo „settings“ apod. K potlačení tohoto chování je slovo, jehož odpovídající překlad nebyl v databázi nalezen, zkontrolováno na výskyt právě této dvojice písmen a znovu dojde k hledání v databázi s novou variantou.

8.4 Rozlišení snímaného obrazu

Tento experiment řeší otázku vlivu rozlišení vstupního obrazu na počet korektně přečtených slov a na dobu zpracování. Se vzrůstající velikostí roste výpočetní náročnost rozpoznávání, ale zároveň se zvyšuje pravděpodobnost správného přečtení slova.

V tabulce 8.4 vidíme čtení textu pro tři podporovaná rozlišení a to `800x480`, `640x480` a `352x288`. Jako vstup opět slouží obraz C.1. Aplikace ve všech případech produkovala velice dobré výsledky, ale za cenu pomalejší odezvy, ačkoli by se dalo po více než dvojnásobném snížení rozlišení čekat mnohem vyšší zrychlení. Poměr mezi velikostí písma a rozlišením obrazu zůstává přibližně na stejné úrovni a právě toto hraje důležitou roli při rozpoznávání samotném (ani velký ani malý text).

Tabulka 8.4: Zkoumání rychlosti aplikace a kvality překladu dle rozlišení vstupního obrazu

Vzorek	Počet slov	Rozlišení obrazu	KRozpS	CHRozpS	Čas [ms]
1	9	800x480	9	0	2120
2	9	640x480	8	1	1983
3	9	352x288	8,33	0,66	1899

Aplikace je standardně nastavená využívat rozlišení `800x480` jelikož se, vzhledem k naměřeným hodnotám, jeví jako nejlepší. Produkuje nejzdařilejší výsledky s pouze malým zpožděním oproti ostatním.

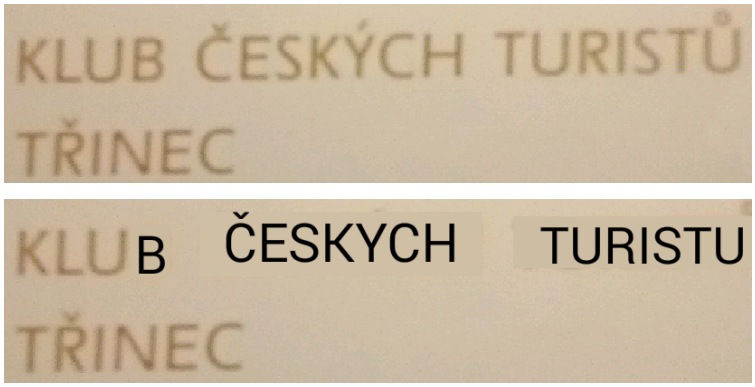
8.5 Vliv zaostřenosti obrazu

Pomineme-li nekvalitní nerovnoměrné osvětlení scény s textem, rozostření obrazu zabírá pomyslnou druhou příčku v negativních vlivech, které mohou překladu škodit. Na obrázku 8.3 lze vidět, jaký vliv má byť minimální rozostření. V tomto konkrétním případě nebylo správně rozpoznáno žádné slovo. Buďto nebylo nalezeno vůbec, nebylo nalezeno celé či chyběla diakritika. Letmým pohledem však obraz působí rozumným dojmem.

Z důvodu takto silných vlivů se aplikace snaží zpracovat obraz hlavně v momentech, kdy je obraz zaostřen. Dopomáhá ji k tomu např. neustálá snaha fotoaparátu ostřit scénu, pozdržování analýzy do zaostření aj.

8.6 Kontinuální běh

Od aplikace je vyžadováno, aby byla schopna analyzovat vstupní obraz také kontinuálně, tedy zobrazovat boxy s přeloženým textem v reálném průběhu z fotoaparátu. Tento experiment si dává za cíl ověřit možnosti běhu právě s důrazem na rychlost mezi zobrazeními.



Obrázek 8.3: Výsledek zpracování vstupního obrazu (dole) pod vlivem byť sebemenšího rozostření (nahoře)

Vstupní obraz C.2 (příloha C) obsahuje čtveřici slov. Každý jednotlivý vzorek ve výsledné tabulce 8.5 znázorňuje jedno zpracování obrazu kontinuálního běhu, nikoli tedy trojici, jako v předchozích experimentech. Z výsledků je patrné, že k překreslování boxů docházelo přibližně každých 1,5 sekundy. Mírné zrychlení v analyzování dalších a dalších obrazů lze přisoudit práci adaptivního klasifikátoru, který se za dobu běhu dokázal lépe adaptovat na písmo v obrazu a tím zrychlit celkovou dobu vyhodnocování.

Tabulka 8.5: Zkoumání rychlosti aplikace v kontinuálním běhu

Vzorek	KRozpS	CHRozpS	KPřelS	CHPřelS	Čas [ms]
1	1	3	1	0	1616
2	4	0	4	0	1523
3	4	0	4	0	1496
4	3	1	3	0	1102
5	4	0	4	0	1302

8.7 Barevné písmo či pozadí

Pro účel překladu informačních tabulí, které zpravidla nebývají černobílé, je žádoucí se zaměřit na barevně podané texty. Vstupem pro tento experiment byly obrázky různých výstražných tabulí či značek. Výsledky překladů a také vstupní obrazy se nachází v příloze D.

Co se týče barevné věrnosti lze výsledky označit za uspokojivé. Aplikace analyzuje pouze barvu pozadí, nikoli barvu písma. Ta je zvolena tak, aby bylo písmo vždy čitelné. U slovníkového překladu nastává problém s nemožností zachytit sémantiku analyzovaného textu, ale překlad v některých případech může působit značně „kostrbatým“ dojmem. Jako konkrétní příklad může posloužit sousloví „in front of“ přeložené jako „v přední na“ namísto „před“. Další problém, který se během experimentování objevil, spočívá ve špatné rozeznatelnosti textu ohraničeného určitou formou rámečku, který je rovněž vnímán jako velké písmeno (zpravidla „I“ či „L“).

8.8 Porovnání s aplikacemi se stejným zaměřením

V této sekci se zabývám porovnáním implementované aplikace s vybranými aplikacemi z kapitoly 5. Do výběru byly zařazeny ty, které se svým použitím a chováním nejvíce podobají vytvořené aplikaci.

8.8.1 Google Translate

První z podobných aplikací je `Google Translate`. Ve prospěch této aplikace hovoří velice kvalitní statistický překladač, který je schopný pracovat jak on-line, tak off-line. Off-line verze pro dvojici jazyků angličtina - čeština však zabere přibližně 160MB dat ve vnitřním úložišti. Stejný slovník, nutno uznat s menší kvalitou překladu, v implementované aplikaci zabírá přibližně 5MB. `Google Translate` po vyfocení obrazu provádí několik sekund trvající analýzu, načež uživatele vyzve k označení textu. Tento proces je však dostupný pouze při aktivním internetovém připojení. V implementované aplikaci dochází rovnou k překladu veškerého textu z obrazu, přičemž celková doba od vyfocení do získání výsledku je pro zvolený počet slov přibližně stejná.

Verze překladače pro mobilní telefony, na rozdíl od té pro `Google Glass`[3], nepracuje na principu zobrazování překladů zpátky ve vstupním obrazu. Aplikace také nedisponuje možností kontinuálního zpracování. Velkou výhodou je naopak již zmíněný překladač, který umožňuje přeložit i text s drobnými chybami či překlepy, což v implementované aplikaci automaticky značí chybný překlad.

8.8.2 Bing Translator

Další aplikací s podobnou funkčností pochází z dílen společnosti `Microsoft`. Ta umožňuje překlad pouze s připojením k internetu, nicméně rychlost zobrazení překladu je s ohledem na tento fakt na velice dobré úrovni. Přeložený text je sice vsazen zpět do obrazu, ale nesnaží se napodobovat vzhled textu původního. Jde o prosté překrytí původního obrázku novou vrstvou, kde pozice textu zhruba odpovídá pozici textu překládaného. Velkým záporem je absence překladu z a do češtiny. Jelikož je překlad prováděn online přes překladač vyhledávače `Bing`, který češtinu podporuje, nemožnost jejího využití je překvapující. Stejně jako v případě předchozí aplikace, překladač obsahuje spellchecker[4]. Rozdíl oproti předchozí můžeme v tomto případě spatřit v možnosti využít kontinuální překlad.

8.8.3 WordLens

S vysokou pravděpodobností nejlepší aplikace ze všech zmíněných. Co do funkčnosti a koncepce se velice podobá aplikaci implementované. Umí pracovat pouze v kontinuálním módu, ale zato s velice rychlou odezvou. Verze zdarma neumožňuje využívat překlad samotný, slovníky lze však dokoupit. Mezi podporovanými jazyky však schází čeština.

Kromě velice rychlého přečtení textu nelze nezmínit širokou škálu parametrů, jakými aplikace formátuje výsledné boxy s přeloženým textem kupř. kurzíva. V důsledku proto velice věrně napodobuje původní text. Zajímavostí je, že společnost `QuestVisual`, která za touto aplikací stojí, byla v květnu 2014 odkoupena společností `Google` [18].

[3] http://www.google.com/glass/start/
[4] kontrola gramatické správnosti řetězce

Kapitola 9

Závěr a další vývoj

Cílem této práce bylo popsat návrh a vývoj Android aplikace zabývající se překladem textu, který se nachází v obrazových datech. Jako zdroj těchto dat měl posloužit vestavěný fotoaparát mobilního zařízení a přeložený text měl být umístěn na displeji tak, aby přímo nahrazoval text původní. Motivací pro vytvoření této aplikace je poskytnout lidem nacházejícím se v cizích zemích možnost přeložení některých textů, kupříkladu cizojazyčných výstražných tabulí, co nejpřirozenější cestou.

Popis využitelných podpůrných knihoven se nachází v první části práce, přičemž výběr konkrétních užitých jsem odůvodnil v kapitole zabývající se návrhem. Navržená a implementovaná aplikace se skládá ze tří hlavních komponent. Pro účely rozpoznání textu v obrazu jsem využil knihovny `Tesseract`, pro překlad textu slovníky dostupné na `dicts.info` resp. `slovnik.zcu.cz` a nakonec pro práci se vstupními obrazy a zobrazením překladu `Android API` a knihovnu `OpenCV`.

Z vykonaných experimentů je patrné, že pro zamýšlené použití, tedy překlad informačních cedulí, aplikace disponuje dostatečně rychlým zpracováním. Barevné podání boxů obsahující přeložený text, které jsou zobrazeny přes analyzovaný obraz, je rovněž na velmi dobré úrovni. Příklady přeložených textů na barevném pozadí je možno vidět v příloze D. Aplikace umožňuje analýzy scény ve dvou režimech, kontinuálním a okamžitém. V prvním případě dochází ke zpracování neustále a boxy s textem jsou postupně překreslovány přes náhled z fotoaparátu. Druhý režim spočívá v pořízení pouze jednoho snímku a jeho okamžitému zpracování.

Aplikace dává mnoho prostoru k dalšímu vývoji. Potenciál k největšímu množství možných úprav zřejmě poskytuje slovník. K výraznému zkvalitnění překladu by jistě pomohlo vyhledávání slovních spojení, nikoli slov osamocených. Příkladem budiž spojení „zlepšit se“, kde sloveso „se“ nemůže být s využitím slovníku nikdy přeloženo ve správném kontextu a znehodnocuje výstup. Obdobná situace nastává s anglickým zájmenem „you“. Další úprava překladu spočívá v možnosti dovolit uživateli kliknout na box s přeloženým textem a navrhnout alternativní překlad. Kromě slovníku, lze několika způsoby urychlit běh jednoho zpracování obrazu. Ve společné kooperaci akcelerometru a paměti již nalezených slov by aplikace mohla měřit, jakým směrem se telefon pohybuje a na základě této informace posouvat s boxy. Ušetří se tím zbytečné zpracování slov, která byla v minulosti ohodnocena jako správně přečtená.

Literatura

[1] ABBYY, Inc. OCR for Android, iPhone and any other mobile device. *ABBYY* [online]. 2013 [cit. 30. 11. 2013].
Dostupné z: http://ocrsdk.com/producttour/mobile-devices/

[2] BabelFish. BabelFish - Free Online Translator. *BabelFish.com* [online]. 2013 [cit. 26. 12. 2013]. Dostupné z: http://www.babelfish.com

[3] BRADSKI, Gary. a KAEHLER, Adrian. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008. ISBN 9780596554040.

[4] Controlling the Camera. *Android Developers* [online]. 2014 [cit. 19. 4. 2014].
Dostupné z:
http://developer.android.com/training/camera/cameradirect.html

[5] Dicts. Download Bilingual Dictionaries. *Dicts.info* [online]. 2013 [cit. 27. 12. 2013].
Dostupné z: http://dicts.info/uddl.php

[6] EIKVIL, Line. OCR - Optical Character Recognition. *nr.no* [online]. 1993 [cit. 26. 12. 2013]. Dostupné z: http://www.nr.no/~eikvil/OCR.pdf

[7] Exper-OCR, Inc. TypeReader Desktop 7.0. *Exper-OCR* [online]. 2013 [cit. 30. 11. 2013]. Dostupné z:
http://www.expervision.com/ocr-software/desktop-ocr-typereader-7

[8] EYERMANN, Horst a BUNK, Michael a kolektiv. FreeDict - free (multi/bi)lingual dictionaries. *FreeDict.com* [online]. 2013 [cit. 30. 11. 2013].
Dostupné z: http://www.freedict.org/en/

[9] Glosbe. Glosbe API. *Glosbe.com* [online]. 2013 [cit. 26. 12. 2013].
Dostupné z: http://glosbe.com/a-api

[10] Google Developers. Google Překladač. *translate.google.com* [online]. 2013 [cit. 26. 12. 2013].
Dostupné z: http://translate.google.com/about/intl/cs_ALL

[11] Google Developers. Google Translate API FAQ. *developers.google.com* [online]. 7. 11. 2013 [cit. 26. 12. 2013].
Dostupné z: https://developers.google.com/translate/v2/faq

[12] HUDGES, Thad. Offline multi-lingual dictionary for Android. *code.google.com* [online]. 2013 [cit. 27. 12. 2013].
Dostupné z: http://code.google.com/p/quickdic-dictionary/

[13] LAGANIÈRE, Robert. *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Pub Limited, 2011. ISBN 1849513244.

[14] Safaba Translation Solutions, LLC. WHAT ARE THE MAIN TYPES OF MACHINE TRANSLATION? *MachineTranslation.net* [online]. 2013 [cit. 26. 12. 2013]. Dostupné z: http://goo.gl/q9wMKB

[15] Microsoft. Microsoft Translator. *Microsoft.com* [online]. 2013 [cit. 26. 12. 2013]. Dostupné z: http://www.microsoft.com/en-us/translator

[16] MORI, Shunji. a SUEN, Ching Y. a YAMAMOTO, Kazuhiko. Historical review of OCR research and development. *Proceedings of the IEEE*. 1992, vol. 80, no. 7, 1029-1058. ISSN 0018-9219.

[17] OpenOCR. Download. *OpenOCR.org* [online]. 2008 [cit. 26. 12. 2013]. Dostupné z: http://en.openocr.org/download/

[18] ROSENBLANTT, Seth. Google buys Word Lens maker to boost Translate. *Cnet.com* [online]. 16. 5. 2014 [cit. 20. 5. 2014]. Dostupné z: http://www.cnet.com/news/google-buys-word-lens-maker-to-boost-translate

[19] SHEN, Ethan. Comparison of online machine translation tools. *TCWorld.info* [online]. Červen 2010 [cit. 26. 12. 2013]. Dostupné z: http://bit.ly/1ijdtgY

[20] SMITH, Ray. Tesseract-OCR. *code.google.com* [online]. 2013 [cit. 30. 11. 2013]. Dostupné z: http://code.google.com/p/tesseract-ocr/

[21] SMITH, Ray. Improving the quality of the output. *code.google.com* [online]. 2013 [cit. 16. 4. 2014]. Dostupné z: http://code.google.com/p/tesseract-ocr/wiki/ImproveQuality

[22] SMITH, Ray. An Overview of the Tesseract OCR Engine. *Document Analysis and Recognition, ICDAR 2007, Ninth International Conference*. 2007, vol. 2, 629-633. ISSN 1520-5363.

[23] SMITH, Ray. a ANTONOVA, Daria. a LEE, Dar-Shyang. Adapting the Tesseract Open Source OCR Engine for Multilingual OCR. *MOCR '09: Proceedings of the International Workshop on Multilingual OCR*. 2009. Dostupné z: http://doi.acm.org/10/1145/1577802.1577804

[24] Yandex. Yandex's Machine Translation Technology. *Yandex.com* [online]. 2013 [cit. 26. 12. 2013]. Dostupné z: http://company.yandex.com/technologies/translation.xml

[25] Yandex. Translate API. *Yandex.com* [online]. 2013 [cit. 26. 12. 2013]. Dostupné z: http://api.yandex.com/translate/

Seznam příloh

Příloha A

Obsah CD

Na přiloženém CD se nacházejí následující adresáře:

- `/src` – zdrojové kódy implementovaných aplikací
- `/apk` – zkompilované aplikace ze složky `/src`
- `/dp_pdf` – diplomová práce ve formátu PDF
- `/dp_latex` – zdrojové kódy diplomové práce v LaTeXu

Příloha B

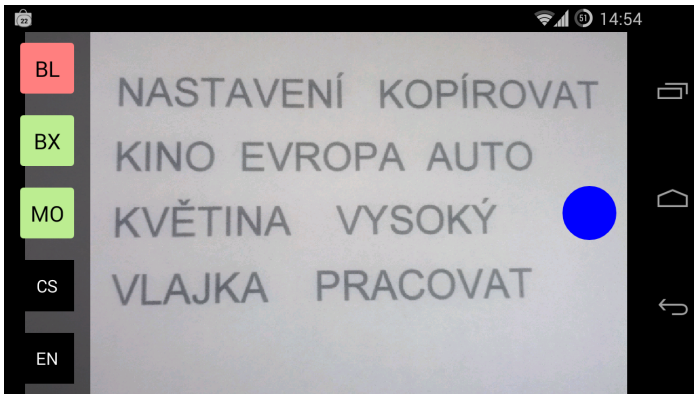
Nativní funkce pro získání umístění slova v obrazu

```
jintArray
  Java_com_googlecode_tesseract_android_ResultIterator_nativeGetBounds(
    JNIEnv *env, jclass clazz, jint nativeResultIterator, jint level) {

  // reference na nativni result iterator
  ResultIterator *resultIterator = (ResultIterator *) nativeResultIterator;
  PageIteratorLevel enumLevel = (PageIteratorLevel) level;

  int left, top, right, bottom;
  int * vals = new int[4];

  // ziskani nejmensiho ctverce obalujiciho slovo
  if(resultIterator->BoundingBox(enumLevel, &left, &top, &right, &bottom)){
      vals[0] = left; vals[1] = top;
      vals[2] = right; vals[3] = bottom;
  }

  // vytvoreni a naplneni pole, ktere je navraceno do Java prostredi
  jintArray arr = env->NewIntArray(4);
  env->SetIntArrayRegion(arr, 0, 4, vals);
  return arr;
}
```

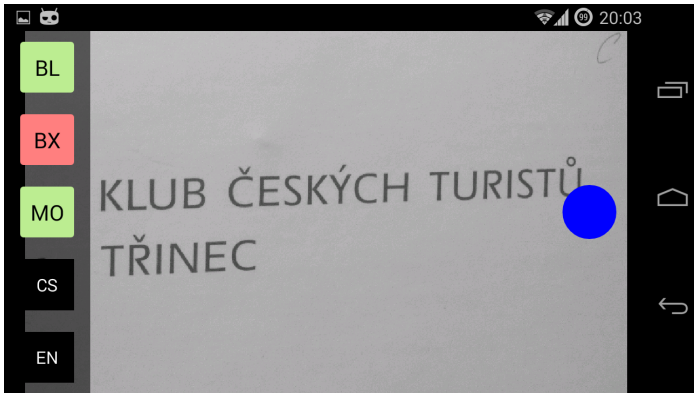
Kód B.1: Funkce pro získání umístění slova v obrazu

Příloha C

Vstupní obrazy užité v experimentech



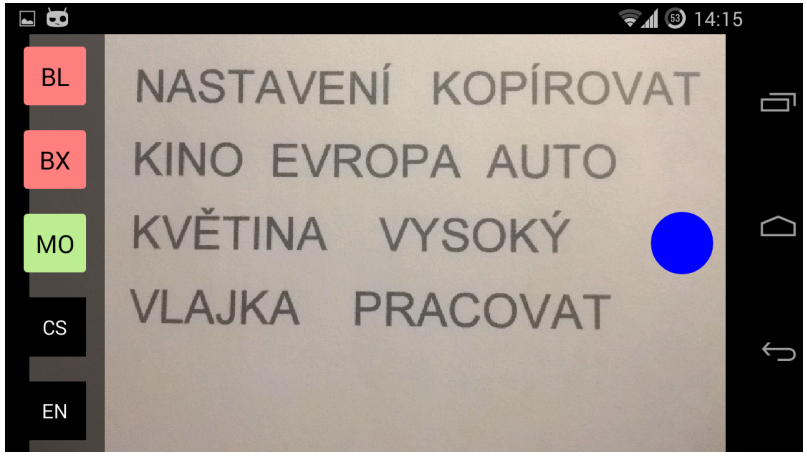
Obrázek C.1: Vstupní obraz pro experiment zkoumající závislost počtu slov v obraze na dobu běhu. V případě meření menšího počtu slov, byly některé řádky vynechány.



Obrázek C.2: Vstupní obraz pro experiment zkoumající kontinuální běh

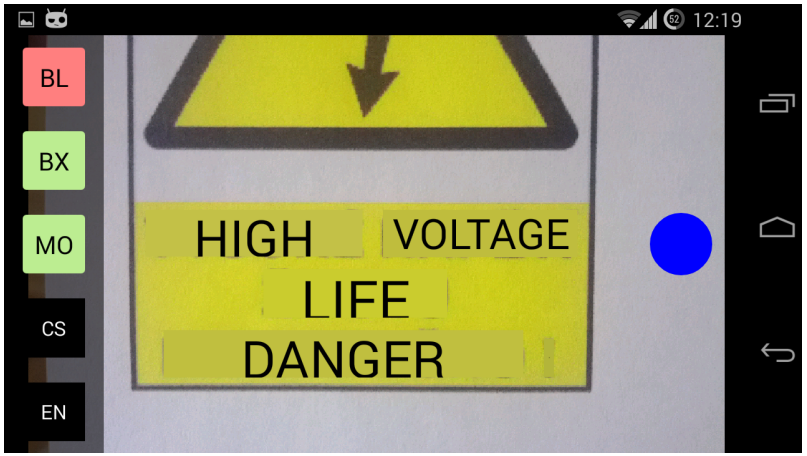
Příloha D

Výsledky překladu


14:15
BL
BX
MO
CS
EN
NASTAVENÍ KOPÍROVAT
KINO EVROPA AUTO
KVĚTINA VYSOKÝ
VLAJKA PRACOVAT


12:20
BL
BX
MO
CS
EN
VYSOKÉ NAPĚTÍ
ŽIVOTU
NEBEZPEČNO !


14:14
BL
BX
MO
CS
EN
SETTINGS
COPY
CINEMA
EUROPE
AUTOMOBILE
FLOWER
WSOKÝ
FLAG
WORK


12:19
BL
BX
MO
CS
EN
HIGH
VOLTAGE
LIFE
DANGER

Příloha E

Instalace a kompilace aplikace

Aplikace se ve zkompilované podobě nachází v adresáři `/apk` na CD a lze ji tedy přímo nainstalovat do telefonu a to buďto:

a. Zkopírováním na externí kartu s využitím instalátoru systému Android.

b. Pomocí nástroje `adb`, který se nachází v adresáři `platform-tools/` v `Android SDK`. Konkrétně jde o příkaz `adb install /apk/DP_xsztef01.apk`.

Postup pro kompilaci zdrojových souborů:

1. Aplikace byla vytvořena pomocí IDE Eclipse[1] s využitím ADT pluginu[2], Android SDK[3] a Android NDK[4]. Všechny tyto čtyři nástroje jsou pro překlad potřeba.

2. V prvé řadě je nutné v Eclipse nastavit správné cesty k Android SDK a Android NDK. To lze učinit v nabídce `Window` → `Preferences` → `Android` pro SDK a `Window` → `Preferences` → `Android` → `NDK` pro NDK.

3. Další krok spočívá v importu projektů pomocí posloupnosti `File` → `Import` → `Android` → `Existing project into Workspace`, postupně pro všechny 3 projekty ze složky `src/` z CD, tedy `DP_preview`, `tess-two`[5] (verze 3.02 s několika úpravami) a `OpenCV Library`[6] (verze 2.4.8). Poslední 2 projekty je zapotřebí nejprve extrahovat (soubor `tess_opencv.zip`) pomocí nástroje `unzip`. Knihovny jsou předkompilované a tedy i velice objemné, což je důvodem pro archivaci.

4. Všechny projekty musí být otevřeny.

5. Nyní je zapotřebí nastavit závislosti projektu s aplikací na ostaních dvou knihovnách. Toto se dá vykonat posloupností pravý klik na projekt `DP_preview` → `Properties` → `záložka Android` → `Add (spodní část okna)` → postupné vybrání obou knihoven.

6. Pokud nedošlo k automatickému nastavení API systému Android, je zapotřebí jej nastavit na hodnotu 16 či vyšší na stejném místě jako v předešlém bodu (`záložka Android`).

7. Nyní by měla být aplikace zkompilovatelná pomocí zkratky `Ctrl + B`.

[1] https://www.eclipse.org/
[2] http://developer.android.com/tools/sdk/eclipse-adt.html
[3] http://developer.android.com/tools/sdk/tools-notes.html
[4] https://developer.android.com/tools/sdk/ndk/index.html
[5] šířeno pod licencí `Apache License, Version 2.0`
[6] šířeno pod licencí `BSD License`