VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

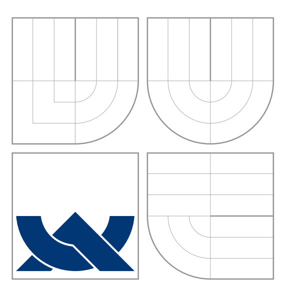
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ČTEČKA BRAILLOVA PÍSMA NA MOBILNÍM ZAŘÍZENÍ

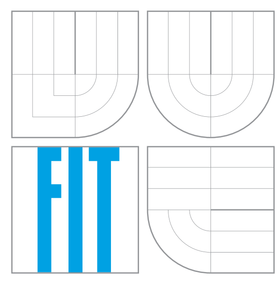
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAN KRUŠINA
AUTHOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ČTEČKA BRAILLOVA PÍSMA NA MOBILNÍM ZAŘÍZENÍ

BRAILLE READER ON MOBILE DEVICE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN KRUŠINA
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAKUB SOCHOR
SUPERVISOR

BRNO 2015

Abstrakt

Tato bakalářská práce se zabývá detekcí a překladem Braillova písma ze snímků pořízených kamerou mobilního telefonu. Práce nejprve obsahuje popis možných řešení této problematiky a detaily ohledně jejich výsledků. Dále bude popsáno zvolené řešení, které bylo použito v této práci, a také bude blíže popsána jeho implementace na vybrané mobilní zařízení. V poslední části bude představen způsob testování aplikace a rozbor naměřených výsledků.

Abstract

This bachelor thesis deals with detection and translation of Braille characters from images taken by a camera on a mobile phone. Firstly, different approaches of solution of this issue are mentioned, and their results are described. Secondly, the chosen method that deals with this problem is introduced, and implementation of this technique on a chosen mobile device is presented. Finally, evaluation of the algorithm is described and results are analyzed.

Klíčová slova

Braillovo písmo, čtečka Braillova písma, mobilní zařízení, Android, OpenCV, detekce, zpracování obrazu

Keywords

Braille Font, Braille Reader, Mobile Device, Android, OpenCV, Detection, Image Processing

Citace

Jan Krušina: Čtečka Braillova písma na mobilním zařízení, bakalářská práce, Brno, FIT VUT v Brně, 2015

Čtečka Braillova písma na mobilním zařízení

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jakuba Sochora.

....
Jan Krušina
16. května 2015

Poděkování

Chtěl bych poděkovat vedoucímu mé práce, panu Ing. Jakubu Sochorovi, za jeho odbornou asistenci a vedení, které mi pomohly při řešení této práce. Dále bych chtěl poděkovat Tyflo-Centru Brno a Centru sociálních služeb pro osoby se zrakovým postižením v Brně-Chrlicích za ochotu a pomoc s poskytnutím testovacích vzorků pro moji práci.

© Jan Krušina, 2015.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **3**

2 Braillovo písmo **4**
2.1 Princip . . . 4
2.2 Historie . . . 5

3 Operační systém Android **6**
3.1 Úvod do Androidu . . . 6

4 Existující řešení detekce Braillova písma **11**
4.1 Detekce prahováním . . . 11
4.2 Detekce pomocí Haarových vlnek a podpůrných vektorů . . . 13
4.3 Detekce prahováním založeným na beta pravděpodobnostním rozdělení . . . 14

5 Využité algoritmy **16**
5.1 Barevný model RGB . . . 16
5.2 Stupně šedi . . . 17
5.3 Integrální obraz . . . 17
5.4 Semínkové vyplňování . . . 18

6 Detekce Braillova písma **20**
6.1 Detekce teček . . . 20
6.2 Seskupení teček . . . 22
6.3 Překlad znaků . . . 22

7 Implementace **26**
7.1 Architektura aplikace . . . 26
7.2 Grafické rozhraní . . . 27
7.3 Optimalizace . . . 27
7.4 Publikování aplikace . . . 29
7.5 Podpůrné nástroje . . . 29

8 Testování **30**
8.1 Chyba detekce teček . . . 31
8.2 Chyba překladu textu . . . 31
8.3 Rychlost aplikace . . . 34
8.4 Zhodnocení výsledků . . . 34

9 Závěr **36**

Seznam obrázků

1.1 Ukázka funkčnosti vytvořené aplikace 3

2.1 Očíslování braillských znaků 4
2.2 Ukázky použití Braillova písma 5

3.1 Architektura systému Android 7
3.2 Poskytovatelé obsahu 8
3.3 Životní cyklus aktivit 9

4.1 Ukázka detekovaných teček 12
4.2 Ukázka výsledných rozpoznaných znaků 12
4.3 Vytvoření binárního obrázku pomocí Haarových příznaků a SVM 13
4.4 Typy Haarových vlnek využitých k detekci vertikálních, horizontálních a diagonálních hran 14
4.5 Ukázka detekce braillských znaků pomocí tří barevných složek 14
4.6 Detekce znaků pomocí mřížky 15
4.7 Výsledná ukázka detekovaných znaků na oboustranném textu 15

5.1 Barevný model RGB 16
5.2 Ukázka obrázku při konverzi do stupňů šedi 17
5.3 Postup vytvoření integrálního obrazu 18
5.4 Ukázka semínkového vyplňování 19

6.1 Srovnání metod prahování 20
6.2 Binární obrázek vytvořený pomocí prahování 21
6.3 Výsledný obrázek po semínkovém vyplňování 22
6.4 Obrázek po seskupení teček do řádků a sloupců 22
6.5 Výsledné schválené a zamítnuté tečky 23
6.6 Přiřazení celkové hodnoty jednotlivým znakům podle umístění teček v daných pozicích 23
6.7 Konečný výstup detekčního algoritmu 23
6.8 Ukázky možných mezer mezi jednotlivými znaky a sloupci teček 25

7.1 Diagram popisující průběh zpracování a detekce snímků 27
7.2 Ukázka grafického rozhraní aplikace na mobilním telefonu 28

8.1 Ukázky z jednotlivých testovacích sad 30
8.2 ROC křivky reprezentující úspěšnost detekce teček 32
8.3 Ukázky z jednotlivých testovacích sad 35

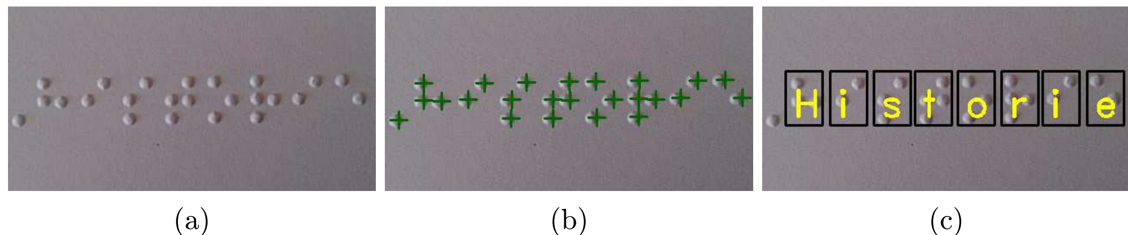
Kapitola 1

Úvod

Braillovo písmo je jeden z hlavních komunikačních prostředků pro nevidomé a zrakově postižené lidi po celém světě. Ve světě je zaznamenáno více než 285 milionů slepých či jinak zrakově postižených lidí. Naučit se číst Braillovo písmo může být složité, nicméně může to být nutnost při komunikaci s lidmi se zrakovou vadou. Právě schopnost číst toto speciální písmo pomáhá bořit komunikační bariéru mezi vidomými a nevidomými lidmi.

Cílem této práce je vytvořit program, který bude schopný převést text psaný v Braillově písmu do latinky. Důležitým aspektem je, aby byl program použitelný prakticky kdekoliv, tudíž musí být přenositelný. To je hlavní důvod, proč je program vytvořen pro mobilní zařízení. Mobilní telefony jsou velmi rozšířené po celém světě a lidé je mají prakticky stále při sobě. Vytvořený program by měl být schopný přečíst a přeložit jak knihy psané v Braillově písmu, tak i jednoduché nápisy a informační značky.

Práce v druhé kapitole nejprve popisuje základní principy Braillova písma, včetně jeho použití, a také se stručně zabývá jeho historií. Ve třetí kapitole je blíže představen operační systém Android využívaný na mobilních telefonech. Výsledná aplikace je dostupná právě pro tuto platformu. Čtvrtá kapitola představuje vybrané způsoby řešení této problematiky, tedy převodu Braillova písma do latinky, použité v předchozích pracích, a to jak na mobilních telefonech, tak i na osobních počítačích. V páté kapitole je podrobně popsán navržený algoritmus pro detekci a překlad Braillova písma z obrázků. Šestá kapitola blíže popisuje implementační detaily na zvolené cílové architektuře. Poslední, sedmá, kapitola pak přináší zhodnocení dosažených výsledků a rozbor naměřených hodnot.



(a) (b) (c)

Obrázek 1.1: Ukázka funkčnosti vytvořené aplikace. Na obrázku a) je možné vidět původní snímek před zpracováním. Obrázek b) znázorňuje tečky detekované navrženým algoritmem a obrázek c) ukazuje výsledný překlad znaků.

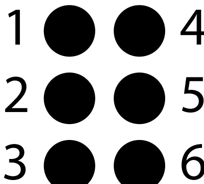
Kapitola 2

Braillovo písmo

Braillovo písmo je speciální druh písma, které je využíváno především zrakově postiženými lidmi. Slouží jak pro zcela nevidomé osoby, tak i pro osoby s částečným zrakovým postižením. Písmo může být vyraženo na papír, případně i na jiný povrch, pomocí speciálních nástrojů. Do povrchu jsou raženy tečky, které čtenář vnímá hmatem. Tečky jsou raženy směrem ke čtenáři a z povrchu vystupují, což usnadňuje jejich hmatové rozeznání. Pro ražbu existují speciální tiskárny, které tisknou na listy papíru Braillovo písmo, i ruční stroje, kterými lze do papíru razit tečky manuálně. Pro tyto účely se používá například speciální psací stroj, nazývaný Pichtův stroj, který obsahuje šest kláves pro ražbu konkrétní tečky v každém znaku písma. Dále existují speciální tabulky s bodátky, pomocí nichž lze tečky také razit do papíru manuálně [14].

2.1 Princip

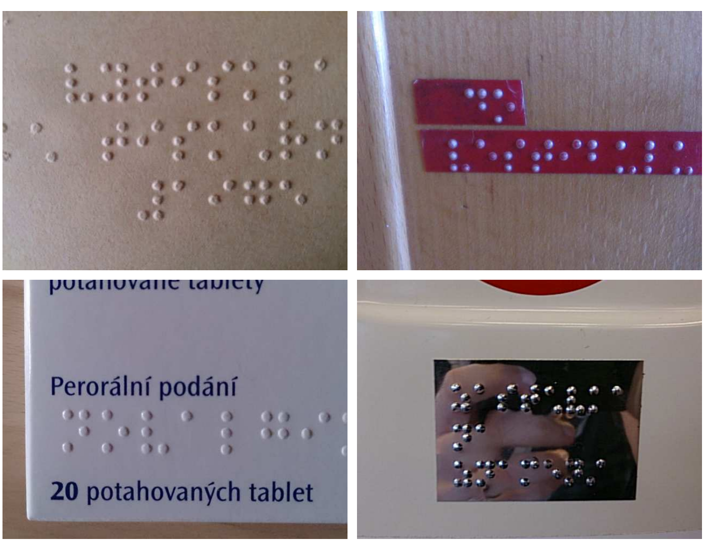
Každý znak Braillova písma je tvořen šesti tečkami, které jsou rozmístěny v obdélníku o rozměrech 2x3 (dva sloupce teček vedle sebe ve třech řádcích pod sebou). Tečky jsou na patřičných místech v obdélníku buď vyraženy směrem ke čtenáři, nebo je místo ponecháno prázdné. Takto lze vytvořit kombinace pro zápis až 63 znaků, jelikož nepovažujeme mezeru za znak. Pomocí Braillova písma lze zapisovat všechna písmena abecedy, v případě češtiny také písmena s diakritikou, číslice, interpunkční znaménka, matematické symboly a další speciální znaky.



Obrázek 2.1: Očíslování braillských znaků.

Jednotlivé tečky braillských znaků se obvykle označují číslicemi od jedné do šesti, což usnadňuje jejich zápis a označení (viz obrázek 2.1). Například zápis písmene *o (o = 135)* značí, že jsou vyraženy tečky jedna, tři a pět, a na zbylých místech tečky nejsou. Pomocí tohoto zápisu lze snadněji zapisovat braillské znaky např. pomocí textových řetězců.

Braillovo písmo není univerzální, prakticky každý jazyk používá svoji formu zápisu, např. anglické Braillovo písmo, francouzské, ale i japonské a další. V případě češtiny je braillské písmo zapisováno po jednotlivých písmenech, tzn., že každá mřížka teček reprezentuje jedno písmeno nebo symbol. Toto se liší například u anglického Braillova písma, kde jsou využity různé druhy zápisu. Text lze zapisovat pomocí jednotlivých písmen nebo po slabikách a dalších zkratkách, což snižuje jeho univerzálnost a rozšiřitelnost. Aktuální sborníky evidují až 133 rozličných jazykových verzí Braillova písma [11]. Na světě je evidováno až 285 miliónů zrakově postižených lidí, z nichž je přibližně 39 miliónů zcela slepých a 246 miliónů má zrak v určitém rozsahu poškozený [16].



Obrázek 2.2: Ukázky použití Braillova písma.

Kromě znaků, které reprezentují každé jednotlivé písmeno, existují znakové prefixy pro zápis čísel a velkých písmen. Standardní znaky braillské abecedy reprezentují pouze malá písmena. Pro zápis jednoho velkého písmene je třeba využít znakový prefix, případně prefix pro zápis celého řetězce velkých písmen. Za prefix se standardně uvedou požadovaná písmena. Stejnou funkci plní prefix pro zápis čísel. Po použití prefixu se využijí znaky pro písmena *a* až *j*, které pak reprezentují čísla od nuly do devítky. Kompletní česká sada znaků je obsažena v příloze C.

2.2 Historie

Braillovo písmo vynalezl francouzský učitel Louis Braill (1809 – 1852). Braill se narodil v městě Coupvray na severu Francie. Ve věku pouhých tří let oslepl na jedno oko, a později kvůli infekci i na druhé. V patnácti letech sestavil unikátní kódovací systém pro čtení a psaní textu slepými lidmi, který je nyní znám jako Braillovo písmo [12]. Písmo bylo odvozeno od vojenského kódování znaků, které vynalezl Charles Barbier de la Serre. Toto kódování využívala armáda především pro čtení zpráv za noci. Braille byl výborný student a později působil jako profesor ve škole pro nevidomé v Paříži. Ve svých 26 letech začal trpět příznaky tuberkulózy, na kterou později i zemřel [7].

Kapitola 3

Operační systém Android

Tato kapitola představuje operační systém Android pro mobilní telefony a stručně popisuje jeho principy a základy fungování systému jako takového. Čtečka Braillova písma, vyvíjená v této práci, je navržena právě pro tuto platformu. Při psaní této kapitoly bylo čerpáno především z knih „Learning Android“ [6], „Android 4“ [2] a dále z oficiální dokumentace Androidu [3].

3.1 Úvod do Androidu

Android je založen na jádře operačního systému Linux a je koncipován jako otevřený software, což umožňuje jeho volné marketingové využití. V současnosti je vyvíjen společností Google. Při jeho zrodu za ním, kromě Googlu, stálo i utvořené seskupení výrobců mobilních telefonů, známé jako Open Handset Alliance (OHA). Toto uskupení tvořily přední světové firmy jako Google, LG, Intel, NVIDIA, Samsung a spousty dalších. První mobilní telefon s tímto systémem se pak začal oficiálně prodávat v roce 2008.

Dnes Android neběží pouze na chytrých mobilních telefonech, ale i na dalších moderních zařízeních, např. tabletech, televizích, hodinkách apod. Pro vývojáře přináší široké spektrum nástrojů a frameworků pro vyvíjení mobilních aplikací.

Architektura systému

Díky tomu, že je Android postaven na jádře Linuxu, je snadné zařídit, aby aplikace běžely nezávisle na typu hardwaru mobilních telefonů. Většina nízkoúrovňových obslužných funkcí Linuxu je psána v jazyku C, který je snadno přenositelný a dovoluje používat Android na různých typech zařízení.

Architektura Androidu se skládá z několika hlavních částí. Tyto části jsou *aplikace*, *aplikační framework*, *knihovny* a *linuxové jádro*. Detailnější popis architektury je na obrázku 3.1.

- **Aplikační Framework** – Aplikační framework obsahuje různé speciálně vytvořené knihovny pro Android. Dále obsahuje *služby* a *manažery*, které umožňují využívání různých prvků zařízení, např. manipulace s WiFi, snímání lokace, využívání senzorů telefonu a další.

- **Aplikace** – Aplikace tvoří již výsledné programy, které lze v zařízeních používat. Mobilní telefony obsahují některé implicitní aplikace, které jsou již předinstalované do

Obrázek 3.1: Architektura systému Android. Je tvořena z několika hlavních částí, a to z *aplikací*, *aplikačního frameworku*, *knihoven* a *jádra*[1].

zařízení, a další lze stáhnout například z Google Play. Každá aplikace je zabalena do jednoho spustitelného souboru ve formátu *apk*. *Apk* soubor je tvořen třemi hlavními složkami, a to spustitelným *Dalvik* souborem, *zdroji* a *nativními knihovnami*. Třídy psané v jazyce Java jsou uloženy a zkompilovány právě do *Dalvik* souboru. *Zdroje* jsou tvořeny veškerými podpůrnými materiály, které aplikace využívají, např. obrázky, média, jazykové sady apod. *Nativní knihovny* obsahují zdrojový kód psaný nejčastěji v jazyce C nebo C++.

- **Knihovny** – Tato část je tvořena knihovnami psanými v C nebo C++, které poskytují důležité služby z aplikační vrstvy. Obsahují například knihovny pro podporu databází (SQLite), prohlížení webových stránek (Webkit), vykreslování grafiky (OpenGL) a další.

- **Linuxové jádro** – Jádro tvoří nejnižší vrstvu této architektury a představuje jakousi abstraktní vrstvu mezi hardwarem a softwarem mobilního telefonu. Zajišťuje například korektní správu ovladačů, napájení, zabezpečení apod.

- **Běhové prostředí** – Obsahuje virtuální stroj nazvaný *Dalvik*, který byl vyvinut speci-

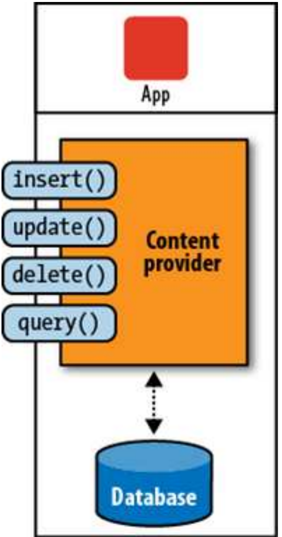
[1] Obrázek byl převzat ze stránky http://fp.edu.gva.es/av/pluginfile.php/745396/mod_imscp/content/2/1_overview_of_the_android_architecture.html.

álně pro potřeby Androidu. *Dalvik* nahrazuje virtuální stroj Javy, který je určený pro všeobecné použití. *Dalvik* se však zaměřuje na potřeby mobilních telefonů, například na lepší úsporu baterie a zlepšení výkonu. Další důvod, proč byl *Dalvik* vyvinut, je licence. Virtuální stroj Javy totiž není koncipovaný jako otevřený software, na rozdíl od knihoven a dalších nástrojů, které jsou v Androidu k dispozici.

Základní prvky Androidu

Jedná se o základní složky, které jsou používané pro vývoj aplikací na Android. Patří sem *aktivity*, *služby*, *poskytovatelé obsahu* a *záměry*. Tyto složky tvoří dohromady celek, kterým je výsledná aplikace.

- **Poskytovalé obsahu** – Jedná se o rozhraní, pomocí nějž jsou aplikace schopny sdílet data mezi sebou. Každá aplikace používá implicitně svůj vlastní datový prostor, který je oddělený od dat ostatních aplikací. Díky *poskytovatelům obsahu* lze jednoduše zpřístupnit data i pro ostatní aplikace. Obsahují čtyři základní metody pro manipulaci s daty – *insert*, *update*, *delete* a *query*.



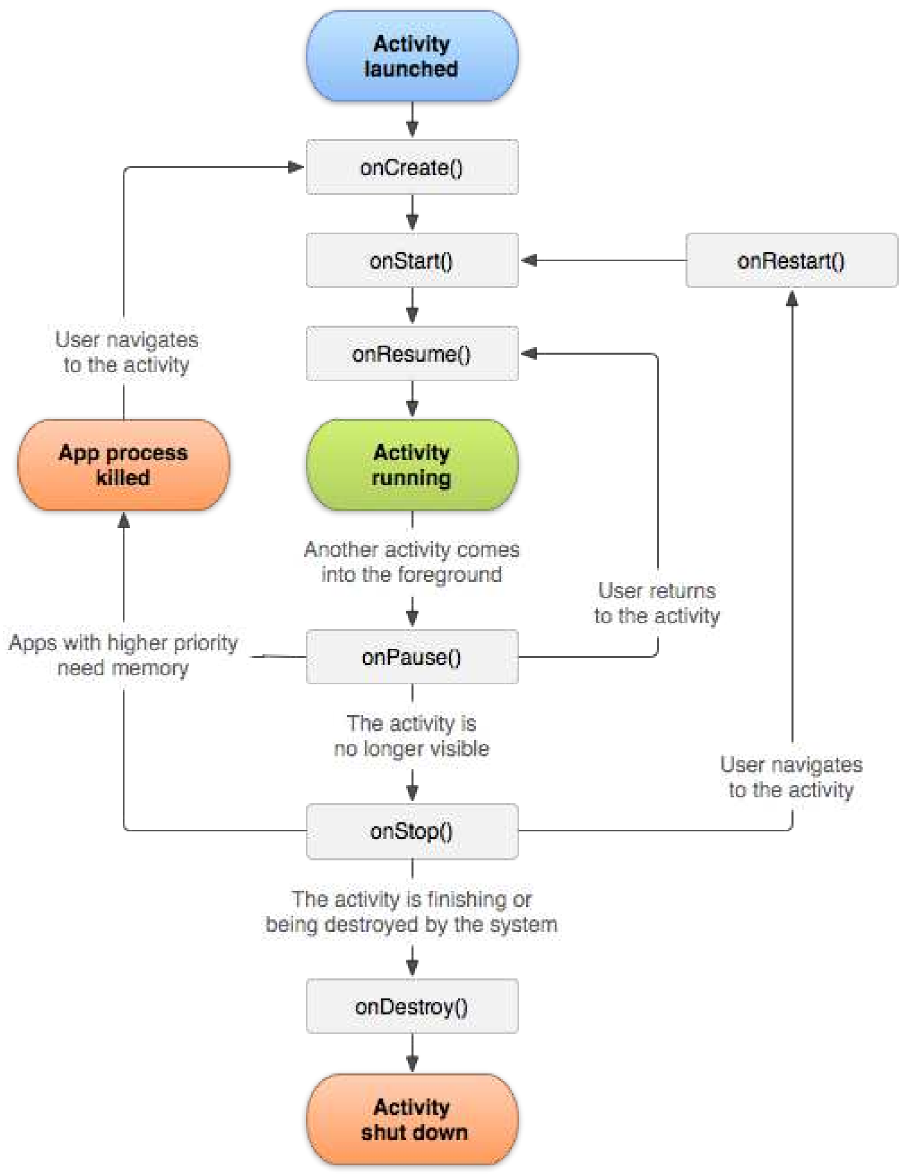
Obrázek 3.2: *Poskytovatelé obsahu*[2].

- **Aktivity** – Jsou to vlastně prvky uživatelského rozhraní. Jedna *aktivita* představuje obvykle jednu obrazovku, kterou vidí uživatel. Aplikace jsou tvořeny několika *aktivitami*, mezi kterými se dá přepínat, jak v rámci stejné aplikace, tak mezi ostatními aplikacemi. Mají prakticky stejnou funkci jako okna ve webovém prohlížeči, nebo okna programů v operačních systémech.

  Každá *aktivita* má svůj životní cyklus. Při spuštění *aktivity* je třeba vytvořit nový proces, alokovat paměť pro prvky uživatelského rozhraní apod. Takovéto operace mohou být náročné, a proto nejsou *aktivity* uvolňovány pokaždé, když uživatel jednu obrazovku opustí. Jejich životní cyklus je spravován *manažerem aktivit*. Ten je zodpovědný za korektní vytváření, uvolňování a spravování *aktivit*.

[2] Obrázek byl převzat ze stránky http://www.edureka.co/blog/beginner-android-tutorials-content-provider.

Po spuštění *aktivity* dojde k její inicializaci a přejde do stavu *running*. Tento přechod tvoří jednu z nejnáročnějších operací a ovlivňuje přímo výdrž baterie. To je hlavní důvod, proč nejsou uvolňovány *aktivity*, i když neběží na popředí. Když je *aktivita* ve stavu *running*, znamená to, že je právě aktivní, a běží na popředí. Přijímá akce od uživatele, jako např. klikání na prvky uživatelského prostředí apod. Pokud je *aktivita* viditelná, ale neinteraguje s uživatelem, je ve stavu *paused*. To se stane například při zobrazení dialogového okna, které původní *aktivitu* překryje. Pokud *aktivita* není vidět, nachází se ve stavu *stopped*. *Aktivity* ve stavu *stopped* mohou být znovu uvedeny do stavu *running* nebo do stavu *destroyed*. Pokud přejde do stavu *destroyed*, dojde k jejímu uvolnění a vymazání z paměti.



Obrázek 3.3: Životní cyklus *aktivit*[3].

[3] Obrázek byl převzat ze stránky http://developer.android.com/reference/android/app/Activity.html

.

- **Záměry** – Jedná se o asynchronní zprávy, které jsou zasílané mezi jednotlivými základními prvky. Mohou například měnit stav *aktivit*. *Záměry* se dělí na dva druhy – implicitní a explicitní. V případě explicitního *záměru* odesílatel specifikuje příjemce dané zprávy. V případě implicitního *záměru* odesílatel specifikuje pouze obecný typ příjemce zprávy.

- **Služby** – *Služby* běží na pozadí bez jakékoliv interakce s uživatelem. Slouží ke stejnému účelu jako *aktivity*, avšak neobsahují žádné uživatelské prvky. Dají se tedy použít pro spuštění akcí, které běží schované na pozadí, např. přehrávání hudby. *Služby* mají odlišný životní cyklus na rozdíl od *aktivit*. Mohou se nacházet pouze ve stavu *running*, do kterého přejdou po spuštění, a ze kterého mohou pouze přejít do stavu *destroyed* a být tak uvolněny.

Kapitola 4

Existující řešení detekce Braillova písma

Tato kapitola má za cíl představit možné způsoby detekování Braillova písma, které byly využity v předchozích publikovaných pracích a studiích. Budou představeny algoritmy pro detekci Braillova písma využité na mobilních telefonech i na osobních počítačích. Jedná se o algoritmy detekující písmo z fotek pořízených mobilním telefonem, případně naskenovaných stránek s braillským textem.

Jelikož je detekce Braillova písma velmi specifická problematika, většina publikovaných článků se zabývá právě detekcí celých stránek slepeckého textu na počítačích. Stránky s písmem jsou většinou naskenovány do počítače, a pak dále zpracovány. Naskenované obrázky jsou tudíž kvalitní a ve vysokém rozlišení, což je však v kontrastu se snímky pořízenými z mobilního telefonu, kde jsou snímky v nízkém rozlišení a v horší kvalitě. Uvedené příklady detekce na osobních počítačích jsou tedy spíše ilustrativní a slouží spíše jako přehled metod, jakými se dá daná problematika řešit. V případě implementace detekce Braillova písma na mobilní zařízení nejsou příliš využitelné, jelikož jsou velmi časově náročné, což je jeden z problémů, kterým se právě tato práce hodlá vyvarovat. Detekce Braillova písma pomocí mobilních telefonů tak není příliš rozšířené téma.

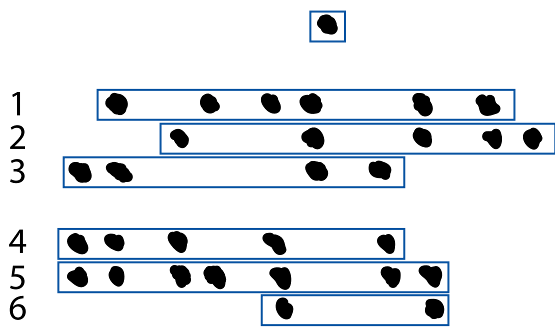
4.1 Detekce prahováním

Článek „A Braille Recognition System by the Mobile Phone with Embedded Camera“ [17] se zabývá detekcí Braillova písma z fotografie pořízené mobilním telefonem. Po pořízení fotografie se převede obrázek z RGB barevného spektra na šedotónový obrázek. Další zpracování obrazu předpokládá, že tečky vyčnívají z povrchu oproti pozadí, mají kruhovitý tvar, a tvoří sady, které jsou si vizuálně velmi podobné.

Jelikož jsou tečky vyvýšené oproti povrchu, na kterém jsou vyraženy, odrážejí odlišně světlo a jsou proto světlejší, než zbylá část povrchu. Pro rozpoznání teček v obrázku je využito dynamické prahování, které odliší tečky od pozadí. Poté jsou jednotlivé detekované pixely prozkoumávány. Pokud se jedná o samostatné pixely, které jsou považovány za šum, jsou odstraněny. Ponechány jsou shluky pixelů, které jsou považovány na potenciální tečky. Stále se však mohou objevit nečistoty v obraze, a proto jsou porovnávány shluky pixelů. Pokud se některé liší od průměru svojí šířkou, výškou nebo velikostí, jsou taktéž odebrány. Ukázka takto detekovaných teček je zobrazena na obrázku 4.1.

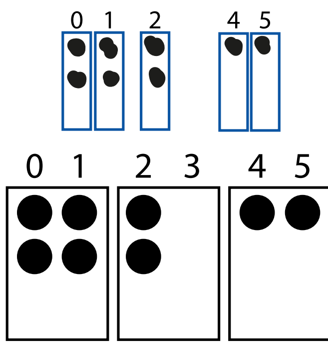
Práce předpokládá, že se tečky vyskytují na pořízeném snímku rovnoměrně, zarovnané

v řádcích a ve sloupcích. To je umožňuje snadno rozpoznat. Článek dále předpokládá, že se každá skupina teček vyskytuje ve třech řádcích, které mají podobné rysy. Pokud se v nich daná tečka nenachází, je taktéž odstraněna. Autoři se ovšem nezmiňují o případné geometrické korekci snímků, která by byla nutná v případě vyfotografování textu z úhlu nebo ze špatné perspektivy.



Obrázek 4.1: Ukázka detekovaných teček. Detekované tečky se nachází v očíslovaných řádcích. Vrchní tečka se však nachází v odlišné pozici oproti ostatním, a bude tudíž zamítnuta.

V každé skupině teček jsou pak tečky rozčleněny podle své pozice do sloupců. Jednotlivé sloupce jsou od sebe odsazeny pevně danými mezerami, které jsou odlišné pro znaky vedle sebe, a taktéž pro tečky ve dvojici sloupců v jednom znaku. Takto se dá rozlišit, zdali sloupce patří do prvního, nebo do druhého znaku. Pokud je vzdálenost $D(0,1) < D(1,2)$, kde 0, 1 a 2 jsou očíslované sloupce a D je vzdálenost, pak patří sloupec číslo nula do prvního znaku, jinak do druhého. Nicméně komplexnější texty psané v Braillově písmu mohou být různorodé, tzn., že se nelze spoléhat pouze na porovnávání dvou sloupců teček vedle sebe, ale je třeba porovnávat tuto vzdálenost i vzhledem k ostatním sloupcům, aby nedocházelo k mylnému překladu.



Obrázek 4.2: Ukázka výsledných rozpoznaných znaků.

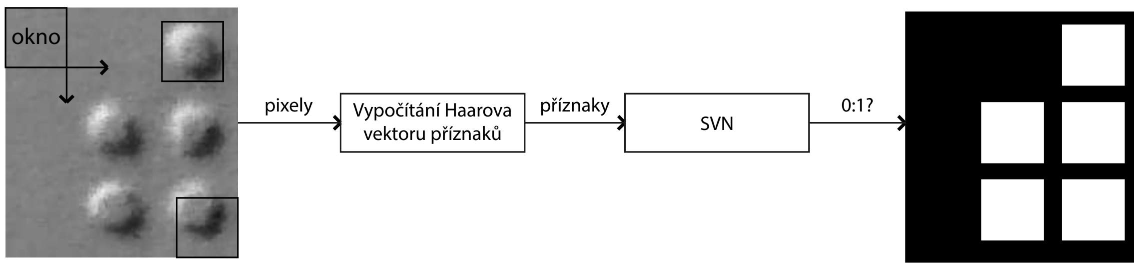
Detekované tečky jsou poté reprezentovány binární posloupností číslic, očíslované zleva doprava, shora dolů. Například tedy písmeno $c = 110000$ značí, že znak obsahuje tečku v levé horní a v levé prostřední buňce. Znak je poté dle hodnoty přeložen (viz obrázek 4.2).

Studie dokládá výsledky pouze na čtyřech testech, z nichž dva byly úspěšné, jeden úspěšný částečně a jeden neúspěšný, jelikož byla fotografie rozmazaná. Průměrná doba zpracování fotografie byla přibližně dvě sekundy. Práce však nedokládá detailnější popis experimentů.

4.2 Detekce pomocí Haarových vlnek a podpůrných vektorů

Čínští vědci, kteří publikovali studii „Optical Braille recognition with Haar wavelet features and Support-Vector Machine“ [9], se zabývali detekováním Braillova písma pomocí metody podpůrných vektorů (SVM) na osobních počítačích.

Obrázek Braillova textu je pořízen standardně pomocí skeneru. Poté je převeden do stupňů šedi, stejně jako v předchozí práci. Následuje případná korekce obrazu, pokud byl při skenování pootočen. Předzpracovaný obraz je dále převeden na binární obraz pomocí posuvného okna, které vyřízne určitou část z obrázku, o velikosti jedné tečky Braillova znaku, a vypočítá Haarův vektor příznaků. Spočítaný vektor příznaků je dále zpracován metodou podpůrných vektorů, která rozhodne, zdali vyříznutá část obrázku obsahuje tečku Braillova znaku, nebo nikoliv. Pomocí této metody je obrázek konvertován do binárního obrázku, který obsahuje nuly v místě pozadí a jedničky v místě, kde se nachází detekovaná tečka. Tento postup je znázorněn obrázkem 4.3. Detekovaný text je pak jednoduše přeložen.



Obrázek 4.3: Vytvoření binárního obrázku pomocí Haarových příznaků a SVM.

Geometrická korekce, kterou je naskenovaný obrázek vyrovnán, je nicméně velmi jednoduchá. Korekce je vyřešena manipulací se samotným zdrojovým textem, na který je nakreslen obdélník, a ten je poté detekován a porovnán, zdali je po naskenování otočený nebo skosený. Tato metoda je velice snadná, avšak vyžaduje přímou manipulaci se zdrojovým textem, tudíž je nevhodná pro implementaci na mobilní telefony. Rotaci obrázku lze vypočítat následujícími rovnicemi:

$$x_2 = cos(\theta) \cdot (x_1 - x_0) - sin(\theta) \cdot (y_1 - y_0) + x_0 \tag{4.1}$$

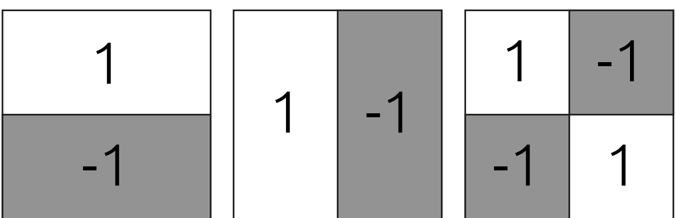
$$y_2 = sin(\theta) \cdot (x_1 - x_0) + cos(\theta) \cdot (y_1 - y_0) + y_0 \tag{4.2}$$

Kde x_0, y_0 jsou souřadnice středu referenčního bodu, x_1, x_1 jsou původní souřadnice bodu, x_2, y_2 jsou nové souřadnice bodu a θ je úhel, o který je obrázek otočen.

Posuvné okno využité pro převod obrazu na binární obraz je o něco větší než předpokládaná fixní velikost jedné tečky ve znaku. Pro detekci tečky jsou využity tři typy Haarových vlnek (viz obrázek 4.4), první slouží pro detekci vertikálních hran, druhá pro detekci horizontálních hran a třetí pro detekci diagonálních hran.

Metoda podpůrných vektorů je podložena 112 kladnými vzorky, které obsahují vystouplé tečky, a 140 zápornými vzorky, které obsahují pozadí, nebo promáčklé tečky z druhé strany. Pomocí těchto vzorků pak metoda rozhoduje o přítomnosti teček v obraze a vytváří postupně binární obraz.

Vzhledem k tomu, že rozmístění teček a jejich odsazení od sebe je dáno standardem, lze tečky jednoduše přerozdělit do znaků. Toto ulehčuje i fakt, že jsou všechny obrázky skenovány při stejném rozlišení skeneru.



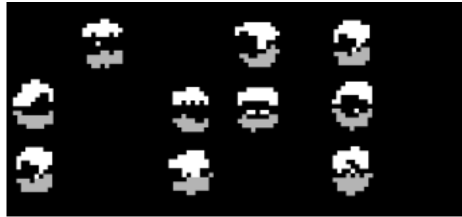
Obrázek 4.4: Typy Haarových vlnek využitých k detekci vertikálních, horizontálních a diagonálních hran.

Autoři uvádějí, že úspěšnost testování byla více než 90 %. Uvádějí však také, že doba detekce jedné stránky byla dlouhých dvacet minut, kterou lze vylepšit jinou implementací algoritmu podpůrných vektorů. Článek však dále neupřesňuje testovací sady nebo další detaily ohledně testování.

I když je správnost detekce velmi vysoká, doba zpracování jednoho snímku je příliš dlouhá na to, aby ji bylo možné použít v mobilní aplikaci.

4.3 Detekce prahováním založeným na beta pravděpodobnostním rozdělení

Článek „An Efficient Braille Cells Recognition" [1] se zabývá metodou detekce Braillova písma prahováním, podobně jako výše uvedené studie, avšak s tím rozdílem, že obrázek není převeden do binární podoby, neskládá se tedy pouze ze dvou barev, ale skládá se ze tří barevných složek, které umožňují přesnější detekci teček Braillova písma (viz obrázek 4.5). Pro navržení hodnoty prahování využívá pravděpodobností rozdělení beta.



Obrázek 4.5: Ukázka detekce braillských znaků pomocí tří barevných složek.

Obrázek je nejprve převeden z barevného spektra do spektra šedotónového. Dále je třeba převést barvy šedi na tříhodnotový obrázek, tzn., že bude obsahovat tři typy barev, tedy černou, bílou a šedou. Šedou barvu pro tmavá místa teček na obrázku a bílou zase pro světlá místa. Obě barvy se týkají vyražených i proražených teček, tedy teček na líci i rubu zdrojového textu v případě oboustranného textu. Černá barva slouží k rozlišení pozadí od teček.

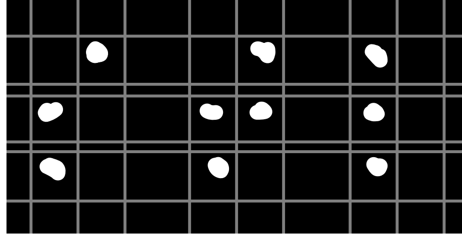
Hlavní problém této metody se týká určení vhodné hodnoty prahování pro dolní i horní mez prahu. Pro výpočet těchto hodnot je využito právě beta pravděpodobnostní rozdělení. Toto rozdělení pracuje s histogramem obrázku, ze kterého získává data, pomocí kterých je schopno určit vhodné dolní a horní hodnoty pro prahování.

Beta pravděpodobnostní rozdělení je spojité pravděpodobnostní rozdělení s hustotou pravděpodobnosti definovanou na intervalu $(0, 1)$. Je definováno následujícím vzorcem:

$$f(x, \alpha, \beta) = \frac{\Gamma \cdot (\alpha + \beta)}{\Gamma(\alpha) \cdot \Gamma(\beta)} \cdot x^{\alpha - 1} \cdot (1 - x)^{\beta - 1} \tag{4.3}$$

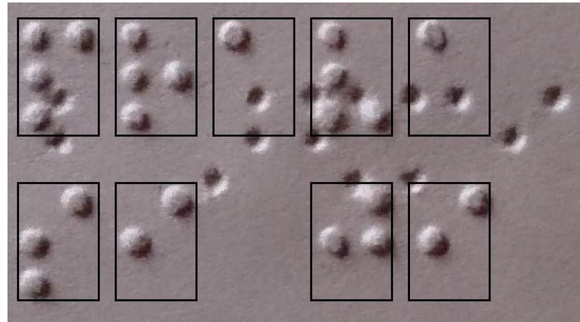
Kde α a β jsou tvarové parametry rozdělení, přičemž $\alpha, \beta > 0$, a kde x je náhodná veličina, přičemž $0 \leq x \leq 1$.

V případě detekce znaků na oboustranném zdrojovém textu je využita mřížka, která ohraničuje všechny znaky na stránce. Algoritmus začíná označením počáteční tečky. Poté je vytvořena mřížka řádků, která ohraničuje jednotlivé znaky. Řádky jsou vykresleny podle fixního odsazení znaků od sebe, které je předem známo. Nicméně nelze takto určit i sloupce, které ohraničují znaky, jelikož nelze přesně dopředu určit rozsazení znaků po stránce. Poté, co je vytvořena mřížka z řádků, se začnou vyhledávat jednotlivé tečky na každém řádku. Po jejich detekci je dokreslena mřížka sloupců, které lze již určit podle detekovaných teček a jejich zjištěných odsazení mezi sebou. Detekce znaků pomocí mřížek je patrná na obrázku 4.6.



Obrázek 4.6: Detekce znaků pomocí mřížky.

Autoři studie tvrdí, že systém je schopný detekovat Braillovo písmo až s 100 % úspěšností, a to jak tečky vyražené zepředu na jednostranném dokumentu, tak i tečky na líci i rubu v případě oboustranného dokumentu. Úspěšnost této metody je výborná, avšak vyžaduje kvalitní obrázek ve vysokém rozlišení pořízený skenerem, což není možné u mobilního telefonu zajistit.



Obrázek 4.7: Výsledná ukázka detekovaných znaků na oboustranném textu.

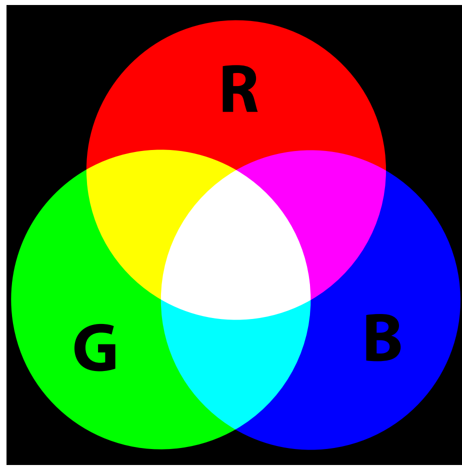
Kapitola 5

Využité algoritmy

Tato kapitola představuje metody, které byly využity při sestavování algoritmu pro detekci Braillova písma. Vysvětluje podstatu jednotlivých metod a odůvodnění jejich použití, případně jejich zamítnutí.

5.1 Barevný model RGB

Barevný model RGB je aditivní model, který zobrazuje barvu každého obrazového bodu pomocí nastavení intenzit primárních barev – červené, zelené a modré. Jednotlivé barvy jsou míchány na černém pozadí, čímž dochází k vytvoření požadované barvy. Černá barva vzniká při nulové intenzitě primárních barev, naopak bílá vzniká při jejich nejvyšší intenzitě.



Obrázek 5.1: Barevný model RGB[4].

Barevné kanály jsou nejčastěji zakódovány na osmi bitech, celkově tedy na 24 bitech pro každý obrazový bod. Hodnota intenzit každé barvy se pohybuje na intervalu $\langle 0, 255 \rangle$. Celkově je možné jejich mícháním vytvořit až 16 777 216 (256^3) barev. Nejčastěji jsou využívány pro zobrazování obrazu na monitorech a ostatních zobrazovacích zařízeních.

Pro detekování teček Braillova písma nenese barevný obrázek prakticky žádnou další hodnotnou informaci, proto je snazší převést obraz do jednoduššího barevného spektra. Konkrétně bude využit obraz ve stupních šedi. Pro účely detekce teček je vhodné zbavit

[4]Obrázek byl převzat ze stránky http://en.wikipedia.org/wiki/RGB_color_model.

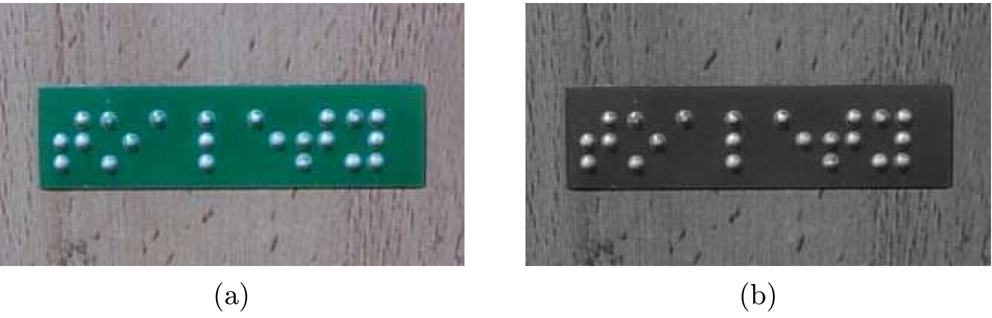
obraz nadbytečných informací ohledně barev, jelikož se zaměřuje především na fakt, že vyražené tečky odrážejí více světla, a tudíž jsou světlejší než ostatní části obrázku.

5.2 Stupně šedi

Šedotónové obrázky neobsahují informace o různých barevných kanálech, jako je tomu například u RGB. Namísto toho obsahují pouze informaci o hodnotě intenzity světla každého obrazového bodu. Kódování barev zabírá tudíž pouze osm bitů, na rozdíl od 24 u RGB. Převod barevného RGB obrázku na obrázek ve stupních šedi lze učinit pomocí následujícího vzorce:

$$I = 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B \tag{5.1}$$

Proměnné R, G a B představují hodnoty primárních barevných složek v obrazu, tedy červené, zelené a modré barvy. Výsledná proměnná I pak obsahuje hodnoty obrazového bodu ve stupních šedi. Jednotlivé barvy jsou různě vyvážené podle citlivosti lidského oka na barvu. Lidský zrak je nejcitlivější na zelenou barvu, poté na červenou a nejméně na modrou barvu. Převod barevného obrázku do šedotónového je ukázán na obrázku 5.2.



(a) (b)

Obrázek 5.2: Ukázka obrázku při konverzi do stupňů šedi.

5.3 Integrální obraz

Integrální obraz je nejvíce využíván pro rychlý výpočet součtu hodnot pixelů v obrázku nebo pro výpočet jejich průměrné intenzity [8]. Integrální obraz má stejnou velikost, tedy stejný počet bodů, jako původní obrázek, přičemž každý bod integrálního obrazu je tvořen součtem hodnot z původního obrázku směrem doleva a nahoru od současného bodu. Což se dá popsat následující rovnicí:

$$I_N(x,y) = f(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1) \tag{5.2}$$

Kde I_N je nová hodnota v integrálním obraze, f je hodnota v originálním obrázku a I jsou již vypočítané hodnoty v integrálním obraze uložené v předchozích krocích.

Integrální obraz je v práci využíván především proto, aby se dosáhlo lepší a stabilnější separace obrazu při prahování snímku. Výsledný obrázek je pak méně náchylný vůči změnám osvětlení v prostředí, které se mohou vyskytovat často a nahodile při snímání obrazu

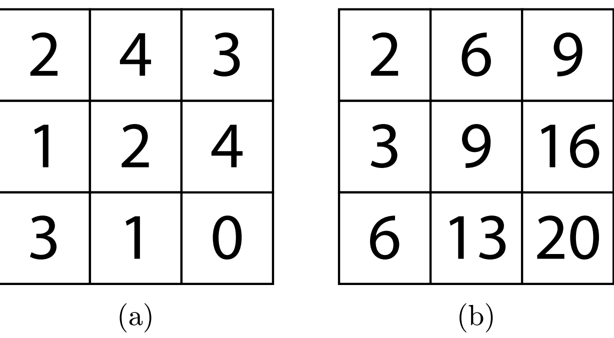
Algoritmus 1: VÝPOČET INTEGRÁLNÍHO OBRAZU

vstup : Původní obrázek $in[w][h]$.
výstup: Integrální obraz $Int[w][h]$.

```
1  for x = 0 to w − 1 do
2      for y = 0 to h − 1 do
3          out = in[x][y];
4          if (x − 1 ≥ 0) then
5              out = out + Int[x − 1][y];
6          end
7          if (y − 1 ≥ 0) then
8              out = out + Int[x][y − 1];
9          end
10         if (x − 1 ≥ 0 and y − 1 ≥ 0) then
11             out = out − Int[x − 1][y − 1];
12         end
13         Int[x][y] = out;
14     end
15 end
```

v reálném čase. Algoritmus 1 pak popisuje vytvoření integrálního obrazu. Tento algoritmus je také použit v této práci.

Proměnná w v algoritmu je šířka vstupního obrázku v pixelech, h je výška vstupního obrázku v pixelech, out je vypočítaná hodnota, in je vstupní – původní obrázek a Int je výstupní integrální obraz. Integrální obrázek je tedy vytvořen po prvním průchodu snímkem.

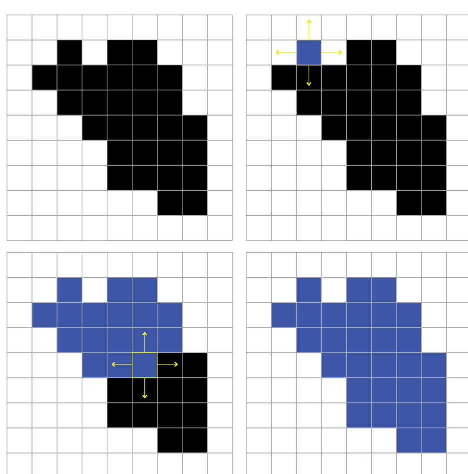


Obrázek 5.3: Postup vytvoření integrálního obrazu. Obrázek a) ukazuje výřez z obrázku s určitými hodnotami. Obrázek b) ukazuje vytvořený integrální obraz, jehož body jsou vytvořeny součtem hodnot původního obrázku doleva a nahoru od každého bodu.

5.4 Semínkové vyplňování

Semínkové vyplňování je označení pro algoritmus, který prohledává obrázek a hledá hranice skupiny stejné barvy v obrázku. Aplikace využívá jednodušší typ vyplňování, a to čtyřsměrové. Tedy, pro každý prohledaný bod je uložen jeho sousední levý, pravý, horní

a spodní bod, který je dále porovnán, jestli obsahuje stejnou barvu. Příklad semínkového vyplňování je na obrázku 5.4.

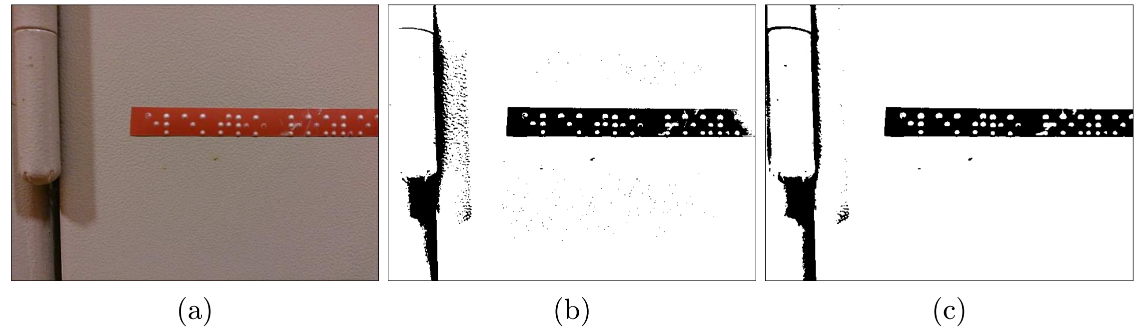


Obrázek 5.4: Ukázka semínkového vyplňování.

Kapitola 6

Detekce Braillova písma

V této kapitole je konkrétně popsán navržený algoritmus detekce Braillova písma. Algoritmus je založen na článku „Adaptive Thresholding Using the Integral Image“ [4], který staví na původním Wellnerovu algoritmu adaptivního prahování [15]. Tento článek přináší spolehlivý, a zároveň rychlý algoritmus adaptivního prahování, který je použitelný i pro zpracování obrazu v reálném čase. Metoda adaptivního prahování je využita především proto, že se lépe přizpůsobuje změnám osvětlení v obraze, které může nastat snadno při zpracování snímků v reálném čase. Bradleyho metoda se od původního algoritmu liší tím, že k detekci využívá integrální obraz, s jehož pomocí je algoritmus odolnější proti náhlým změnám osvětlení v obraze, a tím pomáhá udržovat stabilnější segmentaci snímku. Další výhodou je, že metoda je snadná na implementaci. Nicméně, největší nevýhodou tohoto algoritmu je, že je třeba každý snímek zpracovat dvakrát. Nejprve je nutné vypočítat jeho integrální obraz, a poté snímek zpracovat. Je tak pomalejší, než metoda, ze které čerpá, nicméně tato nevýhoda není tak významná. Obrázek 6.1 ukazuje srovnání obou metod.



(a) (b) (c)

Obrázek 6.1: Srovnání metod prahování. Obrázek a) ukazuje originální snímek, b) ukazuje snímek zpracovaný pomocí Wellnerovy metody a c) pomocí Bradleyho metody. Z prostředního obrázku je patrné, že na obrázku zůstává stále šum a nečistoty. Obrázek napravo je lépe separovaný a s méně nečistotami v obraze.

6.1 Detekce teček

Snímek pořízený kamerou mobilního telefonu je nejprve převeden z barevného spektra do stupňů šedi (viz obrázek 5.2). Jelikož je tato metoda založena na detekování vržených stínů od teček, je nepotřebné uchovávat i informace o barvě obrázku. Stíny teček jsou lépe patrné

na obrázku ve stupních šedi. Poté je snímek ještě upraven pomocí filtru, který rozostří obrázek tak, aby se obrázek částečně zbavil šumu, který je přítomen.

V dalším kroku je třeba vypočítat integrální obraz pořízeného snímku. Konkrétní postup výpočtu integrální obrazu byl popsán v předcházející kapitole. Výpočet využívá algoritmus 1 taktéž popsaný v minulé kapitole.

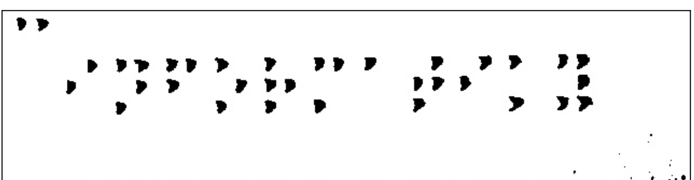
Poté, co je spočítán integrální obraz, je vytvořen binární obrázek, který obsahuje černou barvu v místě výskytu teček a bílou barvu v pozadí obrázku. Je vytvořeno speciální okno, které odpovídá asi osmině velikosti snímku, a je iterováno obrázkem zleva doprava, shora dolů. Hodnota prahu pro vytvoření obrázku se odvíjí od vypočítání průměrné hodnoty pixelů v daném okně. To je možné spočítat pomocí následující rovnice:

$$\sum_{x=x_1}^{x_2} \sum_{y=y_1}^{y_2} f(x,y) = I(x_2,y_2) - I(x_2,y_1-1) - I(x_1-1,y_2) + I(x_1-1,y_1-1) \quad (6.1)$$

Kde f je hodnota původního obrázku a I jsou hodnoty integrálního obrazu. Souřadnice x_1, y_1 pak znamenají levý horní roh okna a souřadnice x_2, y_2 pak představují pozici pravého dolního rohu vytvořeného okna. Součet pixelů je vypočítán pro každé okolí jednoho pixelu v obrázku. Po vypočítání celkového součtu hodnot pixelů v obrázku je vytvořen binární obrázek pomocí prahu vypočítaného z následující rovnice:

$$g(x,y) = \begin{cases} black & f(x,y) \leq \frac{s\ \cdot(1.0-p)}{c} \\ white & \text{jinak} \end{cases} \quad (6.2)$$

Kde *black* a *white* je výsledná barva v binárním obrázku, tedy černá, nebo bílá. $f(x,y)$ je hodnota současného pixelu, c je počet pixelů v okně, s je celkový součet hodnot pixelů v okně a p je tolerance prahu pro sumu hodnot. Aplikace využívá hodnoty $p = 0,15$. Výsledek této techniky je zobrazen na obrázku 6.2.



Obrázek 6.2: Binární obrázek vytvořený pomocí prahování. Šum je stále přítomný v určitých částech obrázku.

Vytvořený binární obrázek ještě není finální. Stále obsahuje pouze detekované skupiny teček, které mohou být potenciálně zvoleny za platné, a také může obsahovat šum v obrázku, který je třeba odstranit.

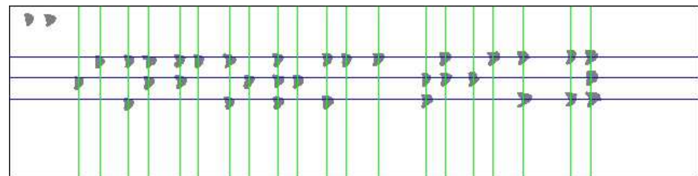
Obrázek je prohledáván a za pomoci semínkového vyplňování (viz obrázek 5.4 jsou detekovány výsledné tečky, které budou považovány za finální. Prohledané skupiny jsou přebarveny z černé na šedou, aby nedošlo k jejich opětovnému prohledávání. Obrázek je prohledán zleva doprava, shora dolů, a pokud se narazí na pixel, který má černou barvu, je spuštěn algoritmus semínkového vyplňování, který prohledá danou skupinu černých bodů a najde jejich hranice. Pokud je velikost dané skupiny příliš malá nebo velká, je skupina zahozena a považována na šum v obraze. Pokud je tečka zachována, je nalezen střední bod

této tečky, a tečka je uložena včetně pozice. Po tomto kroku je uložen seznam detekovaných teček. Výsledná podoba snímku je zobrazena v obrázku 6.3. V dalším kroku jsou pak nalezené tečky seskupovány do řádků a sloupců.

Obrázek 6.3: Výsledný obrázek po semínkovém vyplňování. Obrázek obsahuje pouze skupiny, které jsou považovány za tečky, a zbylý šum je ze snímku odstraněn.

6.2 Seskupení teček

Tečky jsou seskupeny do znaků ve dvou krocích. V prvním kroku jsou prohledány všechny tečky a jsou porovnávány jejich x-ové souřadnice. Všechny tečky, které patří na stejný řádek, tzn. mají podobné souřadnice, jsou uloženy. Takto jsou uloženy všechny linky s tečkami, které jsou detekovány. Dále jsou porovnávány vertikální mezery mezi linkami. Pokud je nalezena podobná velikost mezi dvěma porovnávanými linkami, jsou tyto linky uloženy. Ostatní linky již nejsou uvažovány a jsou zahozeny. Algoritmus se tedy snaží najít vždy horní tři linky s tečkami. Předpokládá, že každé slovo psané v Braillově písmě je tvořeno ze tří linek (řádků). Pro každou tečku je uložena pozice linky, na které se nachází. V dalším kroku jsou prohledávány tečky, které mají přiřazenou pozici vybrané linky, na které se nachází. Jsou porovnávány y-ové souřadnice teček a tečky jsou seskupeny do sloupců.

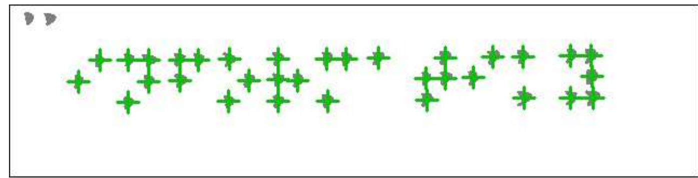


Obrázek 6.4: Obrázek po seskupení teček do řádků a sloupců. Modré horizontální linky představují tečky, které se nacházejí na stejném řádku. Zelené vertikální linky zase představují tečky, které se nacházejí ve stejných sloupcích. Tečky v levém horním rohu jsou ignorovány, jelikož se nacházejí mimo řádek textu.

Po těchto krocích jsou přiřazeny sloupce a řádky pro každou vybranou tečku. Pokud některá tečky nemá přiřazený sloupec a zároveň řádek, na kterém se nachází, je zahozena. Algoritmus je totiž zaměřen pouze na detekci jednoho řádku Braillova písma, tedy tři linky teček pod sebou. Výslednou detekci znaků, které byly vybrány zobrazuje obrázek 6.5.

6.3 Překlad znaků

V poslední fázi jsou porovnávány mezery okolo jednotlivých teček. Podle vzdáleností teček mezi sebou jsou vytvořeny znaky Braillova písma. Rozmístění sloupců ve znacích a jejich možné kombinace jsou znázorněny na obrázku 6.8. Každému znaku je přiřazen levý, pravý nebo oba sloupce znaků a vrchní, prostřední, spodní řádek či jejich kombinace, podle jejich



Obrázek 6.5: Výsledné schválené a zamítnuté tečky. Schválené tečky jsou označeny zeleným křížem, ostatní tečky byly algoritmem zamítnuty.

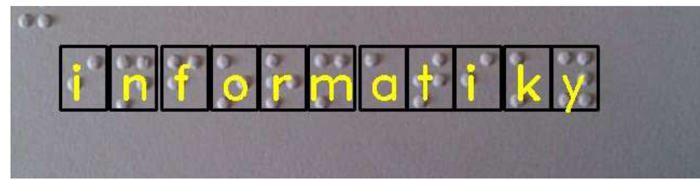
příslušných pozic. Po sestavení znaků jsou jednotlivým znakům přiřazeny hodnoty podle pozice teček. Obrázek 6.6 znázorňuje přiřazování hodnot znakům.

Algoritmus 2 ve zjednodušené verzi popisuje proces přiřazení sloupců znakům. Algoritmus přebírá na vstupu seznam y-ových pozic detekovaných sloupců *columns* v obrázku. Seznam je poté iterován a jsou porovnávány dva záznamy mezi sebou. Proměnná *curr* značí hodnotu na pozici aktuálního elementu v seznamu, *prev* značí hodnotu na pozici předcházejícího elementu. Rozdíl těchto hodnot je pak uložen v proměnné *dist*. Po dokončení jedné iterace je pak aktuální vzdálenost uložena do proměnné *prevDist*, která v příští iteraci bude obsahovat vzdálenost mezi předchozími sloupci. Proměnné a, b, c, d, e a f odpovídají velikosti mezer mezi sloupci, které jsou znázorněny na obrázku 6.8. Hodnoty těchto vzdáleností byly experimentálně naměřeny. Poté, co je detekována příslušná mezera, je uložen nový záznam do seznamu znaků L. Do záznamu se ukládají dvě pozice, které značí souřadnice sloupců, ze kterých je znak složen. Pokud je do znaku v některém ze sloupců vložena nula, pak znak daný sloupec neobsahuje.

1	8
2	16
4	32

Obrázek 6.6: Přiřazení celkové hodnoty jednotlivým znakům podle umístění teček v daných pozicích.

Podle výsledné hodnoty znaku je daný znak přeložen do latinky a vykreslen zpět na pozici původního znaku Braillova písma na obrazovce, jak ukazuje obrázek 6.7. Před vykreslením jsou ještě ošetřeny výskyty speciálních znaků, a to znaků, které fungují jako prefixy. Pokud se tedy jedná o prefix, daný znak není vykreslen, ale jsou upraveny následující znaky podle typu prefixu, například konvertování písmena na číslo nebo na velké písmeno.

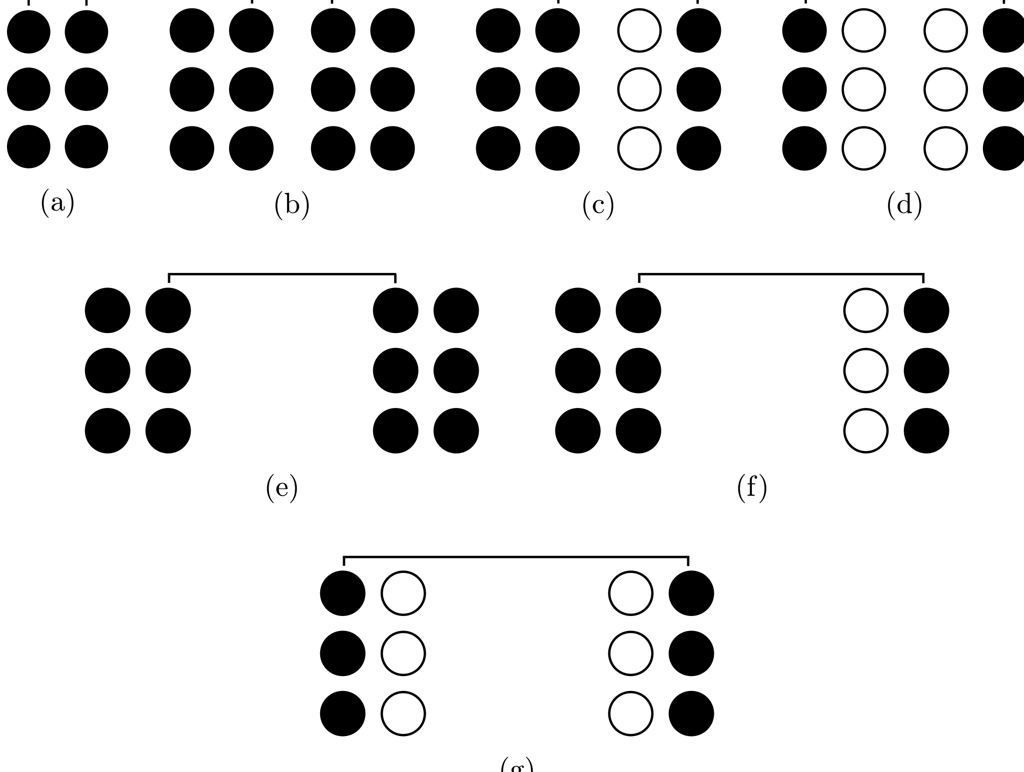


Obrázek 6.7: Konečný výstup detekčního algoritmu. Vytvořené znaky jsou přeloženy do latinky a vykresleny zpět na displej mobilního telefonu.

Algoritmus 2: Rozdělení sloupců do znaků podle mezer

vstup : Seznam y-ových pozic detekovaných sloupců *columns*.
výstup: Seznam znaků Braillova písma *L*.

```
1  for i = 0 to columns.size() do
2      curr = columns[i];
3      dist = curr − prev;
4      if (dist ≤ a) then
5          L.add(prev, curr)
6      else if (dist ≤ b) then
7          if (i = columns.last() and prevDist ≤ a and prevDist ≠ 0) then
8              L.add(curr, 0);
9          else
10             L.add(0, prev);
11         end if
12     else if (dist ≤ c) then
13         L.add(prev, 0);
14     else if (dist ≤ d) then
15         L.add(prev, 0);
16     else if (dist ≤ e) then
17         if (prevDist ≤ b) then
18             L.add(prev, 0);
19         else if (prevDist ≤ d) then
20             L.add(0, prev);
21         else if (prevDist ≤ e) then
22             L.add(prev, 0);
23         else if (prevDist ≤ f) then
24             L.add(0, prev);
25         end if
26     else if (dist ≤ f) then
27         L.add(prev, 0);
28     else
29         if (i = columns.last()) then
30             L.add(curr, 0);
31         else
32             L.add(prev, 0);
33         end if
34     end if
35     prevDist = dist;
36     prev = curr;
37 end
38 return L;
```



Obrázek 6.8: Ukázky možných mezer mezi jednotlivými znaky a sloupci teček. Obrázek a) ukazuje odsazení mezi sloupci v jednom znaku, b) ukazuje odsazení mezi dvěma znaky, c) ukazuje mezeru mezi dvěma znaky, přičemž u jednoho z nich chybí vnitřní sloupec, d) ukazuje odsazení mezi dvěma znaky, přičemž oběma chybí vnitřní sloupce teček, e) ukazuje odsazení mezi znaky s jednou mezerou, f) ukazuje odsazení mezi znaky s mezerou a jedním chybějícím vnitřním sloupcem a g) ukazuje odsazení s mezerou a chybějícími oběma vnitřními sloupci u znaků.

Kapitola 7

Implementace

Tato kapitola se zabývá popisem implementace aplikace na mobilní telefony. Konkrétně pro mobilní telefony s operačním systémem Android.

Aplikace byla vyvíjena ve vývojářském prostředí *Eclipse* s Android Development Tools (ADT) pluginem, který rozšiřuje dané prostředí a přináší lepší podporu programování na platformě Android.

Pořizování a zobrazování snímků z kamery mobilního telefonu je naprogramováno pomocí jazyku Java. Převážná část aplikace, včetně samotného zpracování snímků, je však psána v jazyce C++. Tato metoda byla zvolena pro zajištění rychlejšího chodu aplikace a zmenšení jejich nároků, jelikož zpracování obrázků psané v jazyku Java se ukázalo jako příliš pomalé. Aplikace ke svému běhu využívá multiplatformní open source knihovnu OpenCV [10], která poskytuje funkce pro zpracování obrazu.

Kvůli využití knihovny OpenCV je třeba pro spuštění aplikace doinstalovat i podpůrnou aplikaci OpenCV Manager, která automaticky vyzve k jejímu doinstalování při spuštění aplikace.

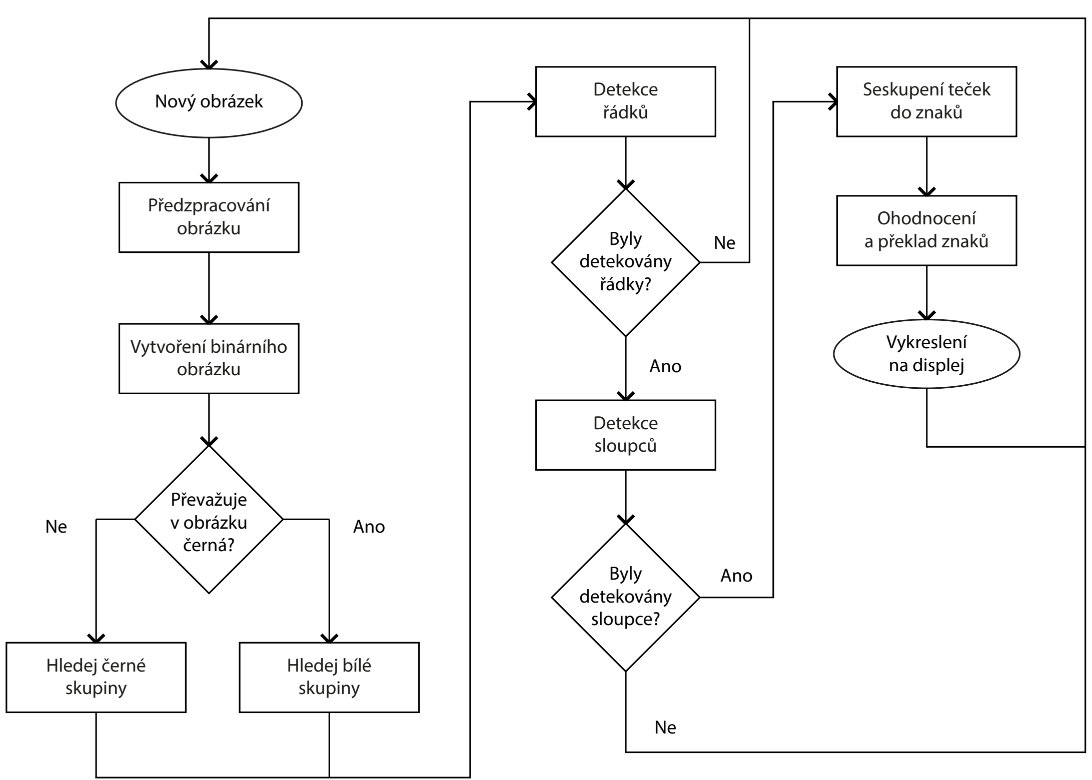
7.1 Architektura aplikace

Aplikace je naprogramována ve dvou jazycích, a to v jazyku Java a v jazyku C++.

Část aplikace psaná v Javě je tvořena hlavní třídou *BrailleReaderActivity*, která se stará o pořizování snímků z kamery mobilního telefonu a o převod snímku do RGB formátu z nativního formátu NV21. Jelikož všechna Android zařízení podporují tento formát, a pouze některá podporují RGB formát, je třeba každý snímek převést do RGB pro jeho další zpracování. Nelze se tedy spoléhat na podporu RGB formátu v jednotlivých zařízeních. Třída dále zobrazuje zpracovaný snímek na displej mobilního telefonu a stará se o další podpůrné funkce jako načítání knihoven, nastavení parametrů kamery, korektní zapínání a vypínání aplikace, uvolňování kamery apod.

Druhá část aplikace, která je psaná v C++, tvoří většinu zdrojového kódu programu. Hlavní třída, která obstarává zpracování snímků, je třída *BrailleReader*. Při volání nativního kódu je z Javy parametrem předán odkaz na pořízený snímek. Snímek je uložen ve struktuře *Mat*, která je dostupná v OpenCV. Tento snímek je ve třídě zpracován a pozměněn a vrácen zase zpět do Javy, kde je vykreslen na displej. Princip fungování programu, resp. zmíněné třídy, je zobrazen na obrázku 7.1. Kód je tvořen dalšími dvěma podpůrnými třídami, a to třídou *BrailleDot*, která reprezentuje jednu detekovanou tečku Braillova znaku a uchovává o ní informace, a druhou třídou *BrailleCharacter*, která uchovává informace o celém znaku,

včetně informace z jakých teček se skládá. Nastavení parametrů aplikace je pak uvedeno v hlavičkovém souboru hlavní třídy, který obsahuje veškeré stěžejní hodnoty a nastavení, které jsou nezbytné pro chod aplikace.



Obrázek 7.1: Diagram popisující průběh zpracování a detekce snímků.

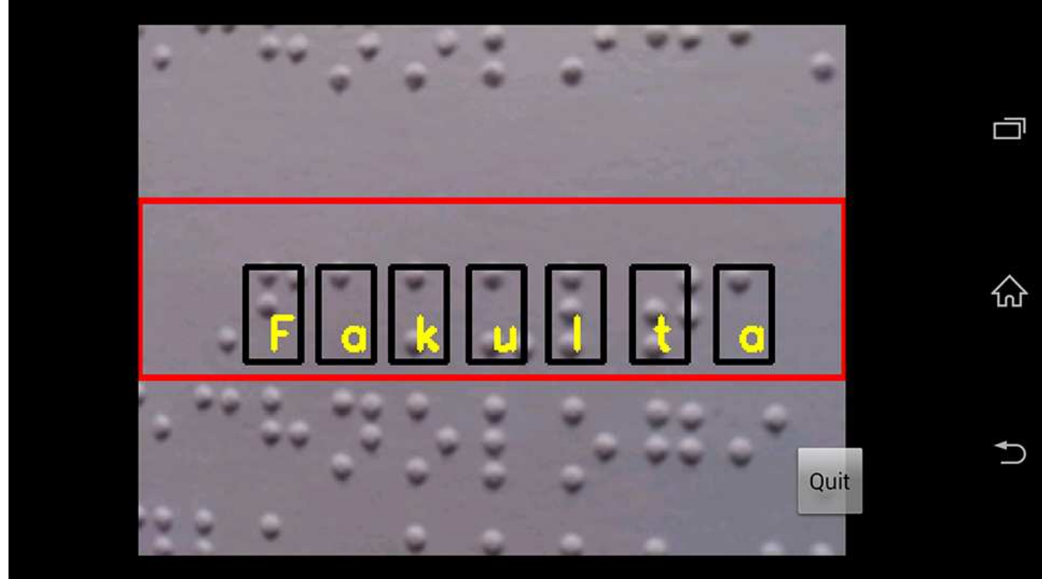
7.2 Grafické rozhraní

Vytvořené uživatelské rozhraní aplikace je velmi jednoduché, jak ukazuje obrázek 7.2. Cílem bylo vytvořit aplikaci velmi snadnou na použití, což se podařilo. Proto není potřeba ani téměř žádná interakce od uživatele, což se mimo jiné odráží i na uživatelském rozhraní.

Uživatelské rozhraní obsahuje pouze tlačítko na ukončení aplikace, jelikož je program jako takový plně automatizován. Hlavní obrazovka aplikace je rozdělena na třetiny. Horní a spodní třetina obrazovky je nevyužitá a vykresluje na displej pouze originální část snímku. Prostřední třetina, ohraničená červeným obdélníkem, představuje hlavní část aplikace, kde se také vykreslují detekované znaky. Jelikož cílem práce bylo zaměřit se na plynulé čtení textu Braillova písma, byla ponechána pouze střední část obrazovky, která odráží čtení jednoho řádku textu Braillova písma. Využití této metody přispělo i ke zrychlení běhu samotné aplikace, protože je zpracovávána pouze třetina obrazu.

7.3 Optimalizace

Cílem aplikace je, aby ji bylo možné používat v reálném čase, a tudíž aby byla schopná poskytovat uživateli okamžitý výstup. Proto je nutné optimalizovat aplikace na mobilní



Obrázek 7.2: Ukázka grafického rozhraní aplikace na mobilním telefonu.

telefony, aby běžela co nejrychleji. Bohužel hardware mobilních telefonů není tak výkonný, jako hardware stolních počítačů, a proto je třeba zavést určitá opatření a kroky, které zrychlí běh aplikace. Jelikož při použití knihovny OpenCV dosahuje aplikace rychlosti pouhých 15 snímků za sekundu bez jakéhokoliv zpracování obrázků, pouze při vykreslování náhledů snímků z kamery mobilního telefonu v rozlišení 320x240 pixelů, je nutné aplikaci co nejlépe optimalizovat.

Rozlišení snímků, které kamera pořizuje a zobrazuje na displej, je omezeno na 320x240 pixelů, což je velmi nízké rozlišení, avšak významně redukuje počet operací, které je potřeba provést při zpracovávání snímků. Jelikož je aplikace zaměřena na plynulé čtení jednoho řádku, lze vstupní snímek navíc ještě více ořezat. A to tak, že se zpracuje pouze jeho prostřední třetina z jeho výšky. Ve výsledku se tedy ušetří další dvě třetiny doby, která by byla potřeba pro zpracování celého snímku. Za použití výše popsaného algoritmu a optimalizace se dá určit počet nutných operací pro zpracování snímku pomocí následující rovnice:

$$(w \cdot h) + (\frac{s}{2} \cdot w) + (\frac{s}{2} \cdot h) - (\frac{s^2}{4}) \tag{7.1}$$

Kde w je šířka obrázku, h je výška obrázku a s je velikost posuvného okna.

Aplikace dále předpokládá, že uživatel bude mít mobil při čtení přiložen relativně blízko k textu, jelikož se zobrazuje na displeji pouze jeden řádek Braillova písma. Proto je při vyhledávání teček omezena hranice jejich velikosti na určitou experimentálně naměřenou hodnotu, která zaručuje, že tečky budou pokaždé podobně veliké, a donutí uživatele manipulovat s mobilním telefonem tak, aby ho držel pokaždé ve zhruba stejné vzdálenosti od textu.

Další hlavní metoda optimalizace je využití programovacího jazyka C++ pro zpracování obrázku. Ukázalo se, že C++ je mnohonásobně rychlejší, než zpracování snímků v jazyku Java. Aplikace využívá Java Native Interface (JNI) pro práci s nativním kódem, zde právě psaným v C++, který je do Javy importován, a tento kód je volán při obdržení každého

nového snímku. V jazyce Java je implementována pouze práce s kamerou, tedy pořizování a posléze vykreslování zpracovaných snímků na displej.

Dále je kód navržený tak, aby využíval co nejmenší datové typy, které jsou potřeba. Aplikace používá pouze celočíselné datové typy, a to především *short* a *char*. Datový typ *integer* není v aplikaci téměř využíván, jelikož hodnoty v aplikace nedosahují takových čísel. Dále je pro kolekce dat využíván datový typ *vector* pro rychlejší přístup k datům. Dále jsou průběžně dealokovány všechny datové struktury, které jsou v průběhu zpracování snímku alokovány.

7.4 Publikování aplikace

Aplikace byla publikována na internetu a je dostupná zdarma ke stažení na Google Play[5]. Jméno aplikace je *BrailleReader* a ke stažení je dostupná verze s podporou českého jazyka. Při vyhledávání jména aplikace se pohybuje na prvních místech po zadání *BrailleReader* nebo *Čtečka Braillova písma*.

Žádné podobné aplikace, které by předkládaly Braillovo písmo v reálném čase nebyly na Google Play nalezeny. Většina dostupných aplikací zabývajících se nějakým způsobem Braillovým písmem slouží spíše jako slovníky pro jeho výuku. Aplikace, které řeší vizuální detekci braillského písma a jeho překlad, se zaměřují pouze na detekci z fotografie pořízené kamerou mobilního telefonu.

Pro instalaci aplikace je vyžadováno mobilní zařízení se zabudovanou kamerou a zároveň zařízení s operačním systémem Android ve verzi minimálně 2.0 s API verzí 5. Dále je také třeba, aby byla v mobilu přítomna další aplikace, a to OpenCV Manager, který je využíván knihovnou OpenCV.

7.5 Podpůrné nástroje

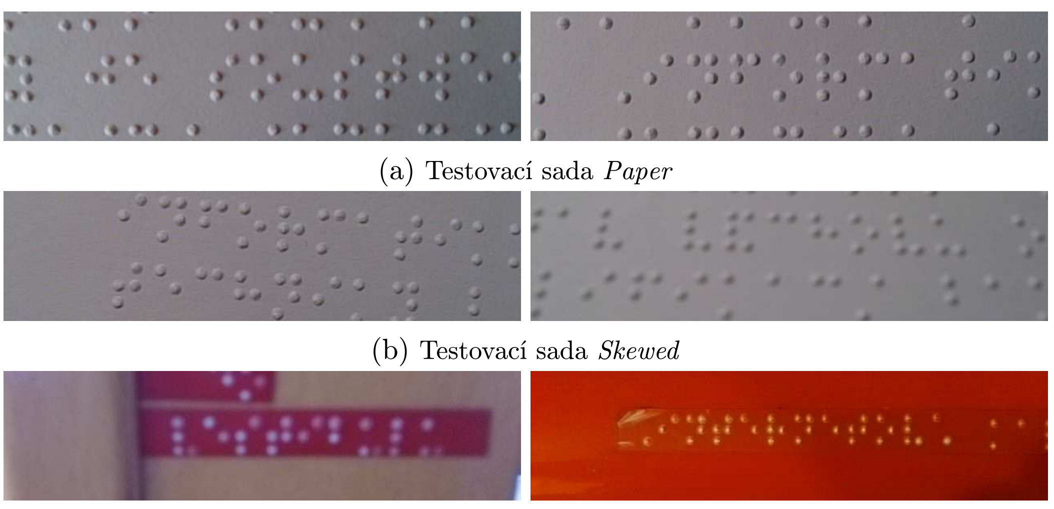
Pro testovací účely bylo vytvořeno několik pomocných aplikací, které sloužily pro vyhodnocení testování navrženého algoritmu. Jedná se o tři pomocné aplikace psané v jazyce C# a jednu psanou v jazyce Java. První aplikace, nazvaná *ImageAnnotator*, slouží k anotování obrázků v testovacích sadách. Pomocí této aplikace jsou ručně označeny pozice teček v obrázcích a tyto pozice jsou uloženy do *csv* souboru, který je vytvořen po spuštění aplikace. Druhý vytvořený program, nazvaný *LevenshteinDistance*, slouží k porovnání chybovosti překladu textu z obrázků. Program porovnává záznam dvou *csv* souborů s textem. První soubor je ručně anotovaný soubor a druhý je vytvořen výstupem z programu. Pomocný program tyto soubory porovná a vypíše výslednou chybovost na výstup. Třetí program, *ResultsComparer*, porovnává pozice teček mezi dvěma *csv* soubory. Srovnává pozice anotovaných teček z fotek s výstupem pozic teček z testovacího programu a měří tak úspěšnost detekce algoritmu. Poslední program psaný v Javě, nazvaný *Evaluator*, funguje stejně jako navržená aplikace pro mobilní telefony, avšak slouží pro vyhodnocení testování na osobním počítači. Program po spuštění zpracuje všechny snímky v testovacích sadách detekčním algoritmem a vytvoří kopie snímků s vykreslenými přeloženými znaky Braillova písma. Dále také vytvoří soubory s přeloženým textem.

[5] Odkaz na stažení aplikace: `https://play.google.com/store/apps/details?id=fit.vut.braillereader`

Kapitola 8

Testování

Tato kapitola popisuje formu testování aplikace a vyhodnocení naměřených hodnot. Testování se skládalo ze dvou hlavních částí, a to z měření úspěšnosti detekce teček a z měření chybovosti překladu detekovaných znaků do latinky. Vyhodnocení probíhalo na třech sadách fotek, které byly označeny jako *Paper*, *Skewed* a *Metal*. Sada *Paper* obsahovala pouze kvalitní fotky s Braillovým písmem vyraženým do papíru. Fotky byly dobře osvětlené a byly minimálně zkosené a rozostřené. Druhá sada, *Skewed*, obsahovala pouze fotky, které byly špatně osvětlené, rozmazané, hodně pootočené či jinak defektní. Braillovo písmo bylo vyraženo také do papírového povrchu. Třetí sada fotek, *Metal*, obsahovala taktéž nekvalitní fotky jako předchozí sada, tedy špatně osvětlené, rozmazané a zkosené, přičemž Braillovo písmo bylo vyraženo do kovového materiálu. Tedy testovací sady *Skewed* a *Metal* byly především experimentální. Ukázky testovacích sad jsou uvedeny na obrázku 8.1.



(a) Testovací sada *Paper*

(b) Testovací sada *Skewed*

(c) Testovací sada *Metal*

Obrázek 8.1: Ukázky z jednotlivých testovacích sad. Zde jsou zobrazeny pro ukázku originální snímky v testovacích sadách.

Testovací sady obsahovaly celkem 100 obrázků, které byly před testováním anotovány. V případě vyhodnocení detekce teček byly na snímcích označeny všechny tečky, které by měl algoritmus zachytit. Byly zaznamenány jejich pozice a tyto pozice byly posléze porovnávány

s pozicemi detekovaných teček získaných pomocí navrženého algoritmu. V případě měření chybovosti překladu probíhalo testování na stejných sadách fotek. Tyto snímky byly taktéž anotovány a výstup algoritmu byl porovnáván s označenými znaky.

8.1 Chyba detekce teček

V první části testování se měřila úspěšnost detekce teček v obraze. Výsledné hodnoty jsou pak reprezentovány pomocí Receiver Operating Characteristic (ROC) křivek [5].

ROC křivky

ROC křivky slouží jako grafické vyjádření závislosti úspěšnosti zásahu, často označováno jako *recall*, a falešných poplachů, tedy detekcí, které jsou mylně považovány za správné. Hlavní typy detekcí, které se tedy zaznamenávají, jsou čtyři. Jedná se o skutečně pozitivní (TP) detekce, skutečně negativní (TN), falešně pozitivní (FP) a falešně negativní (FN).

V této práci byly měřeny hodnoty pro *recall* a pro *precision*, které vypovídají o přesnosti algoritmu detekce teček.

Pro výpočet úspěšnosti zásahu je použita následující rovnice:

$$Recall = \frac{TP}{TP + FN} \tag{8.1}$$

Kde TP je počet skutečně pozitivních detekcí a FN je počet falešně negativních detekcí.

Pro výpočet prediktivní hodnoty pozitivního testu, často označováno jako *precision*, neboli výpočet pravděpodobnosti, která určuje, že výskyt je skutečně pozitivní, v případě, že byl jako pozitivní detekován, je použitá následující rovnice:

$$Precision = \frac{TP}{TP + FP} \tag{8.2}$$

Kde TP je počet skutečně pozitivní výskytů a FP je počet falešně pozitivní výskytů.

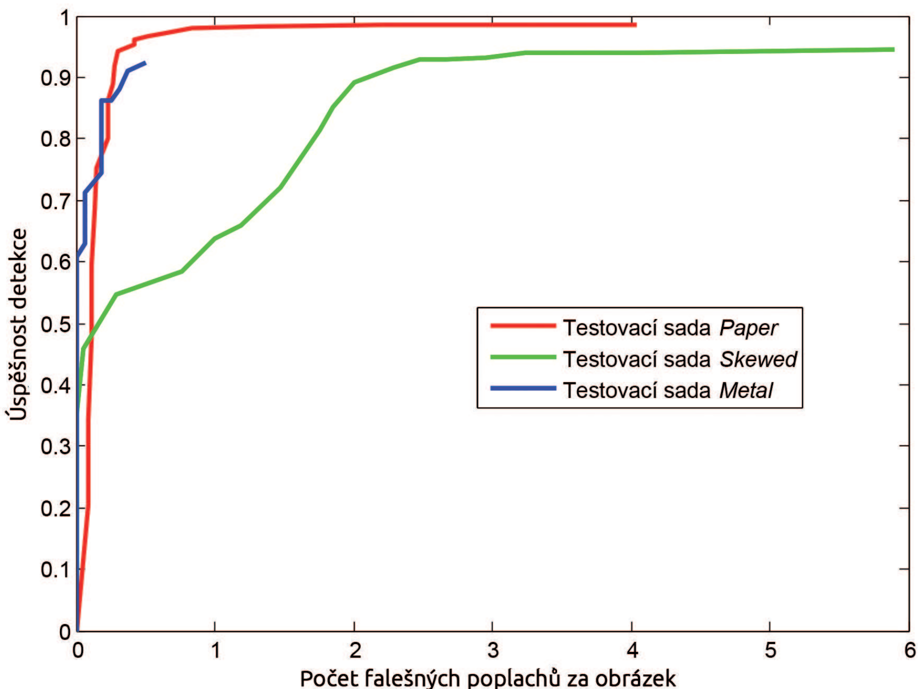
Výsledky detekce

Testování algoritmu probíhalo na uvedených sadách. Pro všechny sady byly naměřeny hodnoty pro *recall* a *presicion*. Tabulka 8.1 pak ukazuje výsledné hodnoty. Hodnoty v tabulkách jsou uvedeny pro hodnotu prahu mezi 5 až 205, která značí velikost detekovaných teček v obrázku. Algoritmus detekce byl velmi úspěšný pro všechny tři testovací sady. Nejlepší výsledky dosáhl v první testovací sadě *Paper*, kde byl *recall* více než 98 % a *precision* přibližně 91 %. Druhé nejlepší výsledky měla sada *Skewed*, ve které byl naměřen *recall* 94 % a *precision* necelých 87 %. Testovací sada *Metal* byla taktéž úspěšná. Naměřené hodnoty pro *recall* byly více než 93 % a pro *precision* více než 96 %.

Graf 8.2 pak představuje naměřené hodnoty jako ROC křivku, která vznikla zobrazením úspěšnosti detekce oproti počtu falešných poplachů za obrázek (FPPI).

8.2 Chyba překladu textu

Druhá část testování byla soustředěna na měření správnosti výstupu aplikace, tedy překladu Braillova písma do latinky. Toto měření probíhalo pomocí výpočtu Levenshteinovy vzdálenosti [13], která se používá pro výpočet míry odlišnosti dvou řetězců od sebe.



Obrázek 8.2: ROC křivky reprezentující úspěšnost detekce teček pro každou testovací sadu. Křivka vznikla vykreslením úspěšnosti detekce oproti počtu falešných poplachů za obrázek (FPPI). Čím vyšší je úspěšnost zásahu, neboli míra skutečně pozitivních výskytů, tím lepší – vyšší je úspěšnost algoritmu. FPPI představuje průměrný počet falešných výskytů za obrázek, tedy počet mylně detekovaných teček v obraze.

Tabulka 8.1: Statistiky úspěšnosti detekce teček

Testovací sada	**Recall**	**Precision**
Testovací sada *Paper*	98,73 %	91,33 %
Testovací sada *Skewed*	94,00 %	86,92 %
Testovací sada *Metal*	93,31 %	96,06 %

Levenshteinova vzdálenost

Jedná se o metriku, pomocí které se dá změřit podobnost dvou řetězců mezi sebou. Princip této metody spočívá v určení počtu operací, které je třeba provést, aby se jeden řetězec modifikoval na druhý. Operace, které tato metoda využívá, jsou tři. Jedná se o *vložení*, *smazání* a *nahrazení* znaku. Cílem je tedy najít nejmenší možný počet operací, které je nutné provést pro kompletní modifikaci řetězce na druhý. Následující algoritmus popisuje, jak se vzdálenost mezi dvěma řetězci pomocí této metody vypočítá:

Výše popsaný algoritmus produkuje hodnoty podrobně popsané v tabulce 8.2. Konečný výpočet chybovosti překladu se pak vypočítá podle následující rovnice:

$$E = \frac{Levenshtein(A, B)}{max(A, B)} \cdot 100 \tag{8.3}$$

Kde E je chybovost v procentech, $Levenshtein(A, B)$ je hodnota Levenshteinovy vzdá-

Algoritmus 3: Výpočet Levenshteinovy vzdálenosti

```
vstup  : Řetězce A, B.
výstup: Levenshteinova vzdálenost D.
1  for i = 0 to A.length do
2      D[i, 0] = i;
3  end
4  for j = 0 to B.length do
5      D[0, j] = j;
6  end
7  for i = 1 to A.length do
8      for j = 1 to B.length do
9          cost = 0;
10         if (A[i] ≠ B[j]) then
11             cost = 1;
12         end
13         D[i, j] = min(D[i − 1, j] + 1, D[i, j − 1] + 1, D[i − 1, j − 1] + cost);
14     end
15 end
16 return D[A.length, B.length];
```

lenosti vypočítána podle algoritmu 3 a $max(A, B)$ je největší možná vzdálenost mezi dvěma řetězci vypočítaná také podle Levenshteinova algoritmu.

Tabulka 8.2: Průběh výstupu algoritmu 3 pro výpočet Levenshteinovy vzdálenosti mezi dvěma řetězci A a B. Výsledná hodnota vzdálenosti D se nachází v pravém dolním rohu tabulky.

D		A				
			a	**b**	**c**	**d**
B		0	1	2	3	4
	a	1	0	1	2	3
	c	2	1	1	1	2
	b	3	2	1	2	2
	d	4	3	2	2	**2**

Naměřená chybovost

Chybovost překladu se měřila taktéž pro všechny tři testovací sady. Každý obrázek v sadě byl vyhodnocen samostatně, porovnáním textu z výstupu aplikace a textu z předem anotovaných fotografií. Hodnoty pak byly zprůměrovány v rámci každé sady. Tabulka 8.3 ukazuje konkrétní naměřené hodnoty. Nejlepší výsledek, a tedy nízkou chybovost, měla první testovací sada, tedy sada *Paper*. Necelých 8 %. Zbylé dvě sady již nebyly tak úspěšné, především kvůli špatné kvalitě obrázků a jejich přílišnému zkosení, špatnému osvětlení, rozmazání či

jinému defektu. Druhá testovací sada vykazovala chybovost přibližně 28 % a třetí testovací sada necelých 70 %.

Tabulka 8.3: Chybovost překladu textu

Testovací sada	**Chybovost**
Testovací sada *Paper*	7,21 %
Testovací sada *Skewed*	28,26 %
Testovací sada *Metal*	69,29 %

8.3 Rychlost aplikace

Vzhledem k tomu, že cílem práce je vytvořit aplikaci, která bude schopná detekovat Braillovo písmo ze snímků pořizovaných kamerou mobilního telefonu, je důležité zaměřit se i na rychlost aplikace, aby byla zajištěna dostatečná plynulost běhu aplikace.

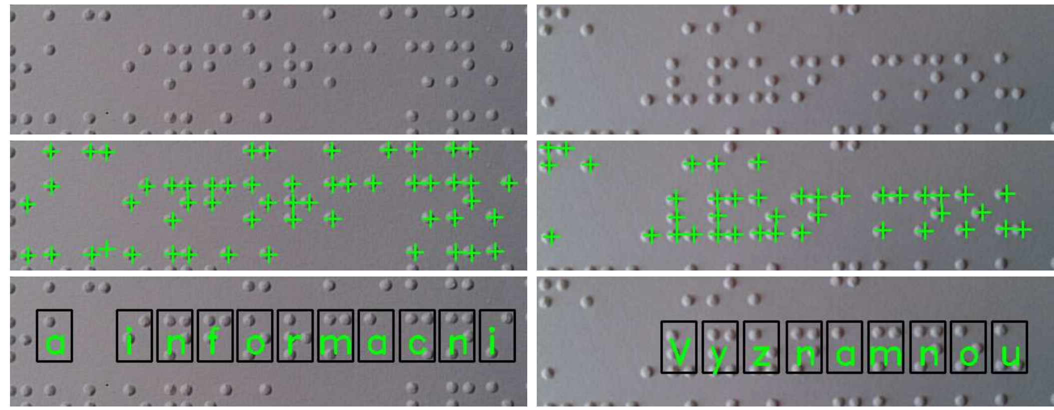
Rychlost aplikace byla stanovována podle měření počtu snímků za sekundu (FPS). Pro testovací účely bylo zvoleno mobilní zařízení Sony Xperia Z s dvěma gigabyty operační paměti a čtyřjádrovým procesorem o frekvenci 1,5 GHz. Měření bylo prováděno na několika testovacích objektech po různě dlouhou dobu. Aplikace byla schopna běžet průměrně v 13,5 snímcích za sekundu v rozlišení 320x240 pixelů s využitím OpenCV knihovny. Naměřené hodnoty jsou dostatečné pro zajištění relativně plynulého chodu aplikace. Nicméně i při vypnutém zpracovávání obrazu, s využitím OpenCV pouze pro zobrazování náhledu kamery, běžela aplikace na pouhých 15 snímcích za sekundu ve stejném rozlišení. Tudíž lze navržený algoritmus považovat za dobře optimalizovaný.

8.4 Zhodnocení výsledků

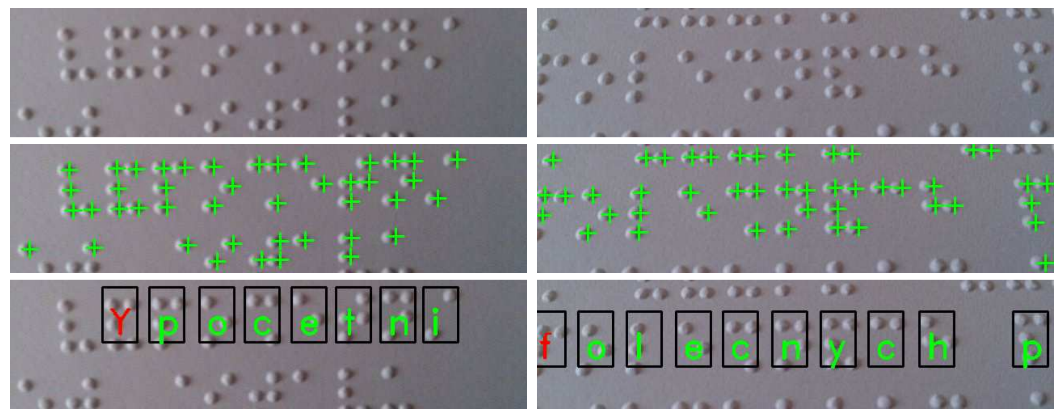
Naměřené hodnoty prokázaly, že aplikace je schopná nejlépe detekovat tečky vyražené do papírového povrchu. Zde se pohybovala úspěšnost zásahu okolo 98 %. Přičemž je nezbytné používat aplikaci v dostatečně osvětleném prostředí, ve kterém mohou tečky vrhat patřičný stín, na kterém je detekce založena. Snímky mohou být i rozmazané, pootočené a jinak nepřesné.

Chybovost překladu však více závisí na kvalitních fotkách, na rozdíl od pouhé detekce teček. Nejmenší chybovost, a tudíž nejlepší výsledky, dosáhla aplikace u první testovací sady, necelých 8 %. U ostatních sad nebyla již tak úspěšná především kvůli nekvalitním fotkám a jejich deformacím.

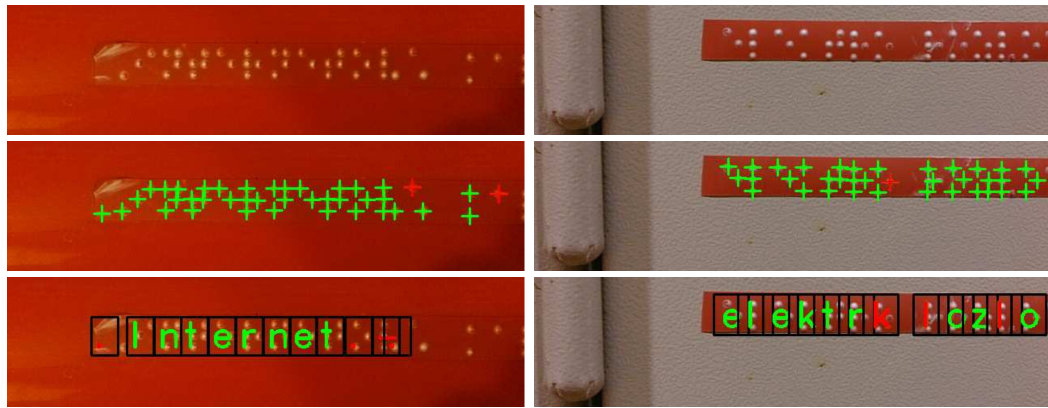
Aplikace běží průměrně v 13,5 snímcích za sekundu, což zajišťuje relativně plynulý chod programu. V této fázi bylo třeba dosáhnout určitého kompromisu mezi přesností detekce a rychlosti běhu aplikace. Jelikož je aplikace zaměřena na zpracování snímků pořizovaných kamerou, je nutné, aby byla zároveň úspěšná v detekci znaků, ale i dostatečně rychlá.



(a) Testovací sada *Paper*



(b) Testovací sada *Skewed*



(c) Testovací sada *Metal*

Obrázek 8.3: Ukázky z jednotlivých testovacích sad. Obrázek ukazuje příklady detekce teček a znaků na vybraných fotkách z testovacích sad.

Kapitola 9

Závěr

Tato bakalářská práce se zaměřuje na vytvoření přenosné čtečky Braillova písma. Práce představuje novou metodu čtení Braillova písma pomocí mobilního telefonu, a to čtení písma v reálném čase. Tento cíl se podařilo částečně splnit, jelikož je aplikace schopná běžet v 13,5 snímcích za sekundu, což zajišťuje relativně dostatečně plynulý běh programu. Jelikož je aplikace vyvinuta pro mobilní telefony s platformou Android, umožňuje široké spektrum využití a vysokou dostupnost, což je jeden z hlavních přínosů této aplikace. Použití nevyžaduje prakticky žádnou interakci uživatele, stačí pouze podržet telefon nad textem psaným v Braillově písmu, a přeložený text do latinky se vykreslí na displej.

Čtečka je zaměřena na detekci a překlad Braillova písma pouze z jednostranného textu. Úspěšnost použitého algoritmu pro detekci teček se jeví jako relativně vysoká. Celková průměrná úspěšnost detekce se pohybuje okolo 95 %. Nejlepší výsledky dosahuje aplikace při čtení textu z papírového povrchu. Úspěšnost detekce teček se zde pohybuje přes 98 % a chybovost překladu textu je necelých 8 %.

Práce byla přihlášena do prvního ročníku soutěže *Excel@FIT*, do které byla posléze také přijata. Pro účely prezentace aplikace byl vytvořen článek, plakát a demonstrační video, které představuje fungování programu. Plakát je přiložen v příloze D a video je dostupné na přiloženém DVD a také na YouTube[6]. Práce se na soutěži umístila v kategorii „Výborný nápad“ na 5. místě, a v kategorii „Společenský přínos“ na 1. místě.

Práce může být v budoucnu rozšířena několika způsoby. Jednak je do aplikace možné přidat podporu pro čtení Braillova písma i v jiných jazycích kromě češtiny. To umožňuje fakt, že Braillovo písmo je pouze odlišně reprezentováno v každém jazyce, nicméně kombinace teček zůstávají zachovány. Další způsob, jakým lze aplikace vylepšit, je upravit detekční algoritmus tak, aby fungoval lépe v hůře osvětleném prostředí a dokázal lépe zpracovat i snímky v horší kvalitě a vypořádal se s jejich geometrickými deformacemi a dalšími defekty. Další potenciální rozšíření aplikace lze dosáhnout přidáním podpory pro hlasový výstup. Aplikace by mohla být schopna číst detekovaný text a umožnit tak využití aplikace i nevidomým lidem.

[6] Odkaz na video na YouTube: https://youtu.be/EfODEhcaGVQ.

Literatura

[1] Al-Salman, A. S.; El-Zaart, A.; Al-Suhaibani, Y.; aj.: An Efficient Braille Cells Recognition. *Wireless Communications Networking and Mobile Computing (WiCOM)*, ročník 6, 2010: s. 1–4.

[2] Allen, G.: *Android 4: průvodce programováním mobilních aplikací*. Computer Press, 2013, iSBN 978-80-251-3782-6.

[3] Android: Getting Started Android Developers [online]. `https://developer.android.com/training/index.html`, [cit. 2015-05-09].

[4] Bradley, D.; Roth, G.: Adaptive thresholding using the integral image. *Journal of graphics, gpu, and game tools*, ročník 12, č. 2, 2007: s. 13–21.

[5] Fawcett, T.: Roc graphs: Notes and practical considerations for researchers. *Machine learning*, ročník 31, 2004: s. 1–38.

[6] Gargenta, M.: *Learning Android: Building Applications for the Android Market*. O'Reilly Media, 2011, iSBN 978-1-449-39050-1.

[7] Jiménez, J.; Olea, J.; Torres, J.; aj.: Biography of Louis Braille and Invention of the Braille Alphabet. *Survey of Ophthalmology*, ročník 54, č. 1, 2013: s. 142–149.

[8] Kelly, M.: Computer Vision The Integral Image [online]. `https://computersciencesource.wordpress.com/2010/09/03/computer-vision-the-integral-image/`, 2010 [cit. 2015-05-09].

[9] Li, J.; Yan, X.; Zhang, D.: Optical Braille recognition with Haar wavelet features and Support-Vector Machine. *Computer, Mechatronics, Control and Electronic Engineering (CMCE)*, 2010: s. 64–67.

[10] OpenCV: ABOUT OpenCV [online]. `http://opencv.org/`, [cit. 2015-05-09].

[11] Perkins: World braille usage [online]. `http://www.perkins.org/assets/downloads/worldbrailleusage/world-braille-usage-third-edition.pdf`, 2013 [cit. 2014-12-23].

[12] Roth, G. A.; E.Fee: The Invention of Braille [online]. `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3036681/`, 2011 [cit. 2014-12-23].

[13] Soukoreff, R. W.; MacKenzie, I. S.: Measuring errors in text entry tasks: an application of the Levenshtein string distance statistic. *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, 2001: s. 319–320.

[14] Tyflokabinet: Braillovo bodové písmo [online]. http://www.tyflokabinet-cb.cz/braill.htm, 2009 [cit. 2015-05-09].

[15] Wellner, P. D.: Adaptive thresholding for the digitaldesk. *Xerox, EPC1993-110*, 1993.

[16] WHO: Visual impairment and blindness [online]. http://www.who.int/mediacentre/factsheets/fs282/en/, 2014 [cit. 2014-12-23].

[17] Zhang, S.; Yoshino, K.: A Braille Recognition System by the Mobile Phone with Embedded Camera. *Innovative Computing, Information and Control, ICICIC '07*, ročník 2, 2007: s. 223–223.

Seznam příloh

A Obsah DVD ... 40
B Spuštění přiložených programů ... 41
C Česká sada braillských znaků ... 42
D Plakát ... 43
E Ukázky detekce snímků ... 44
F Ukázky detekce z aplikace ... 45

Příloha A

Obsah DVD

Přiložené DVD obsahuje všechny materiály k bakalářské práci.

- **Apk** – Složka obsahuje výslednou aplikaci ve formátu *apk* na mobilní telefony s operačním systémem Android, a také odkaz na stažení aplikace přímo z Google Play.
- **Evaluation** – Všechny testovací aplikace, v jejich spustitelné podobě, jsou umístěny v této složce, včetně návodu, jak dané programy spustit.
- **EvaluationSource** – Zde jsou umístěny veškeré zdrojové kódy pro testovací programy, které byly napsány za účelem vyhodnocení aplikace.
- **Photos** – Tato složka obsahuje všechny obrázky z testovacích sad, které byly použity při testování aplikace.
- **Poster** – Obsahuje plakát ve vysokém rozlišení ve formátu *pdf*, který je také přiložen v příloze D.
- **SourceCodes** – Zde se nacházejí zdrojové kódy pro výslednou aplikaci na Android.
- **ThesisSource** – V této složce se nachází zdrojové soubory textu této práce pro LaTeX. Dále je zde obsažena finální verze práce ve formátu *pdf*.
- **Video** – K práci je také přiloženo demonstrační video, které ukazuje funkčnost aplikace. Video je ve formátu *mp4* v rozlišení 1280x720 pixelů.

Příloha B

Spuštění přiložených programů

V této příloze je stručně popsán postup, jak lze spustit přiložené testovací aplikace na DVD. Vytvořené testovací programy sloužily pro vyhodnocení úspěšnosti navrženého algoritmu. Spuštění programů vyžaduje operační systém Windows, jelikož většina z programů je psaná v jazyce C#.

V kořenovém adresáři přiloženého DVD se ve složce Evaluation nacházejí již zkompilované a spustitelné aplikace testovacích programů. Každou aplikaci je možné spustit samostatně, ale je nutné zachovat pořadí spouštění programů. Nejprve je třeba spustit aplikaci *Evaluator*, která zpracuje všechny snímky ze složky Photos, nacházející se na stejné adresářové úrovni jako složka Evaluation. Po spuštění programu dojde k vytvoření složky Results, která obsahuje výsledné fotografie s detekovanými znaky Braillova písma. Pro spuštění samotné aplikace je vyžadovaná knihovna OpenCV, která je přiložena v adresářích pro 32bitovou i 64bitovou verzi.

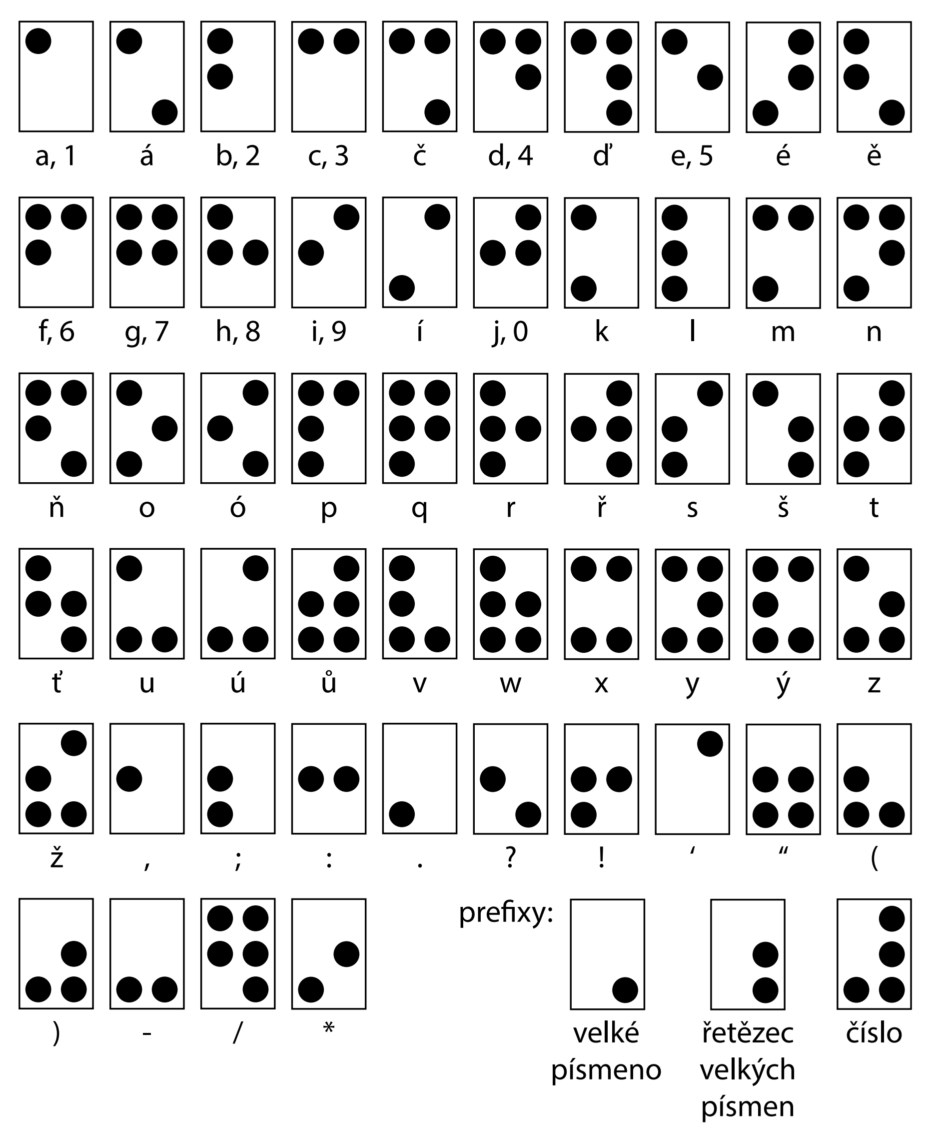
Poté, co jsou vytvořeny zpracované fotografie, lze spustit jednotlivě oba vyhodnocovací programy. *ResultsComparer* vyhodnocuje úspěšnost detekce navrženého algoritmu porovnáváním pozic teček mezi anotovaným souborem a výstupem z programu. *LevenshteinDistance* měří chybovost překladu taktéž porovnáváním anotovaných fotografií a výstupu z testovací aplikace.

Celý tento postup je detailně popsán v přiloženém souboru *readme* v adresáři. Nejsnadnější cesta pro spuštění testování je spustit přiložený skript *run.bat*, který provede automaticky výše uveděného kroky, a vytvoří patřičné soubory.

Je nezbytné, aby se zachovala stejná adresářová struktura, jako na přiloženém DVD. Pokud by došlo k jejímu porušení, nebudou vyhodnocovací programy funkční. Bylo by nutné upravit cestu k potřebným souborům přímo ve zdrojových kódech programů, které se nacházejí ve složce EvaluationSource v kořenovém adresáři DVD.

Příloha C

Česká sada braillských znaků



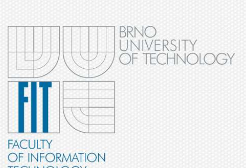
Příloha D

Plakát

Čtečka Braillova písma na mobilním zařízení

Jan Krušina
xkrusi00@stud.fit.vutbr.cz

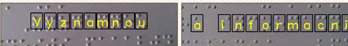
Vedoucí: Ing. Jakub Sochor



Abstrakt

Aplikace slouží jako čtečka Braillova písma. Braillovo písmo je rozpoznáváno ze snímků pořízených kamerou mobilního telefonu. Detekované znaky jsou poté přeloženy do latinky a vykresleny zpět na displej telefonu.

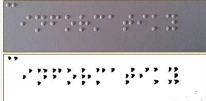
Aplikace je zaměřena na čtení jednostranného textu psaného Braillovým písmem.



Detekce teček

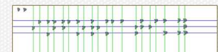
Algoritmus rozpoznávání teček je založen na detekování stínu, který tečky vrhají na povrch, proto je nezbytné, aby byla aplikace používaná v patřičně osvětleném prostředí.

Obraz je nejprve převeden z barevného spektra do odstínů šedi a poté je spočítán jeho integrální obraz. Za pomoci adaptivního prahování s využitím integrálního obrazu je vytvořen binární obraz.



Detekované skupiny jsou pak prohledávány pomocí semínkového vyplňování. Šum je odstraněn a jsou zachovány skupiny, které velikostně odpovídají tečkám.

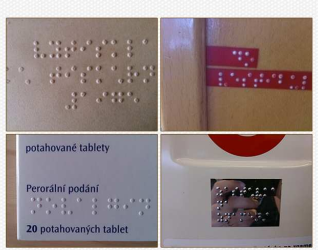
Poté jsou vyhledány tečky, které patří do stejných řádků a sloupců. Zachovány jsou pouze tečky, které patří do jednoho řádku Braillova písma, tedy tři řádky teček.



Následně jsou změřeny vzdálenosti mezi jednotlivými tečkami. Tečky jsou seskupeny do znaků právě podle jejich rozmístění v obraze. Každý znak je poté přeložen a vykreslen zpět na obrazovku mobilního telefonu.

Braillovo písmo

Braillovo písmo je speciální druh písma, které využívají zrakově postižení a nevidomí lidé. Každý znak se skládá z jedné až šesti teček, které jsou vyraženy do určitého povrchu (například papíru nebo kovu). Tečky jsou tedy vystouplé, narozdíl od pozadí, což umožňuje jejich detekci.



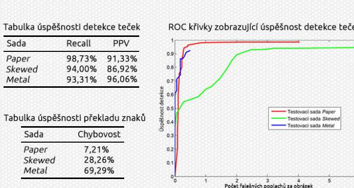
• *ukázky běžného použití Braillova písma*

Vyhodnocení

Aplikace byla testována na třech různých sadách fotek. Tyto sady byly označeny *Paper*, *Skewed* a *Metal*. Pro všechny byla měřena úspěšnost detekce teček a úspěšnost překladu znaků. Sada *Paper* obsahovala kvalitní fotky s mimálním zkosením obrazu. *Skewed* obsahovala pouze rozmazané, nekvalitní fotky s velkým zkosením či jinak defektní. *Metal* sada obsahovala taktéž nekvalitní fotky s Braillovým písmem vyraženým do kovového povrchu.

Aplikace byla implementována na mobilní telefony s platformou Android s využitím multiplatformní open source knihovny OpenCV pro zpracování obrazu.

Na testovacím zařízení byla schopna běžet průměrně v 13,5 snímcích za sekundu v rozlišení 320x240 pixelů.



Tabulka úspěšnosti detekce teček

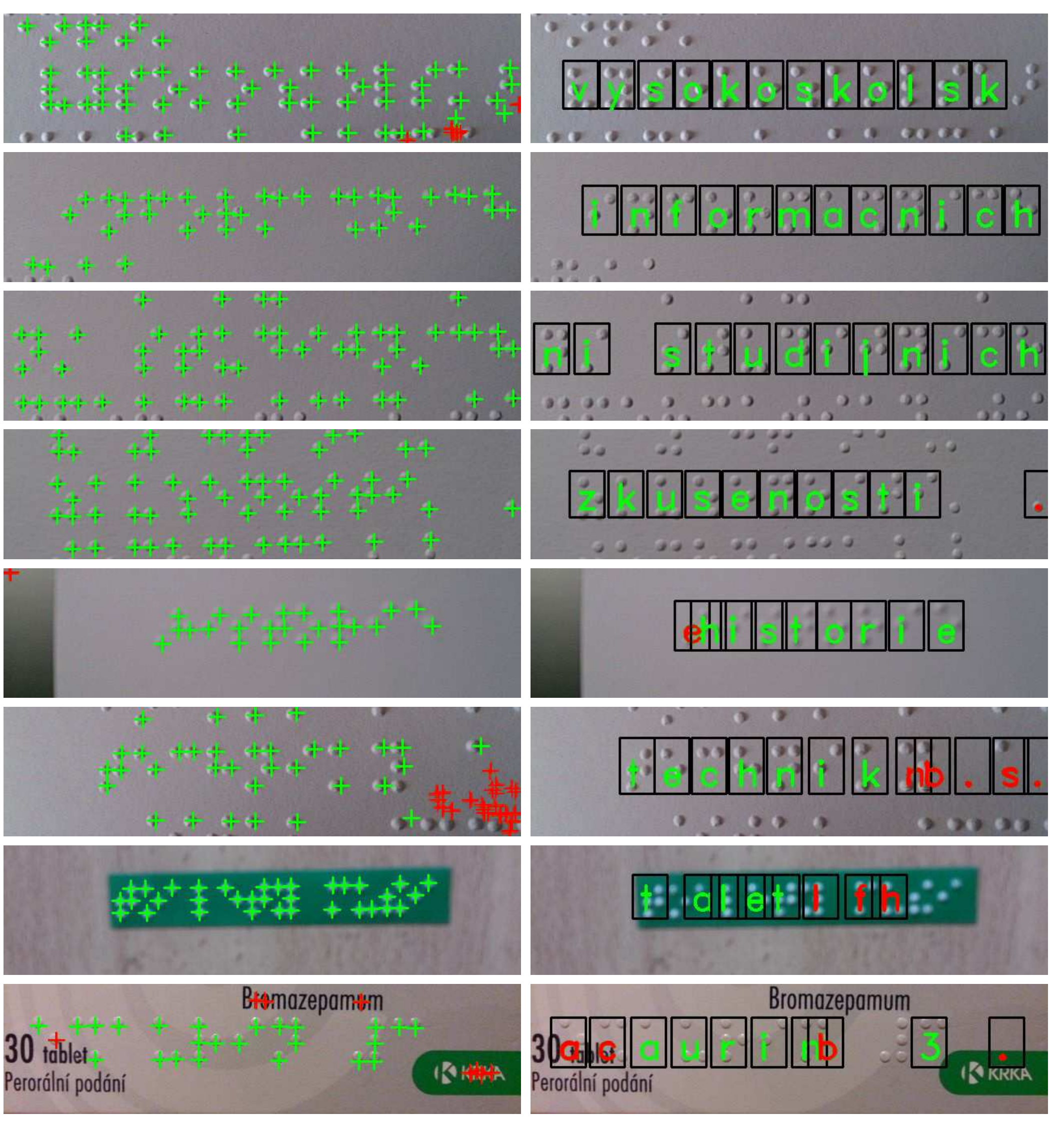
Sada	Recall	PPV
Paper	98,73%	91,33%
Skewed	94,00%	86,92%
Metal	93,31%	96,06%

Tabulka úspěšnosti překladu znaků

Sada	Chybovost
Paper	7,21%
Skewed	28,26%
Metal	69,29%

Příloha E

Ukázky detekce snímků



Příloha F

Ukázky detekce z aplikace