VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

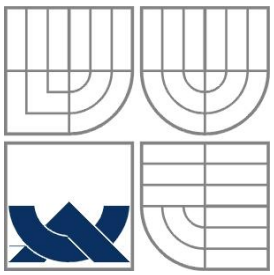
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SVET OKOLO NÁS AKO HYPERLINK

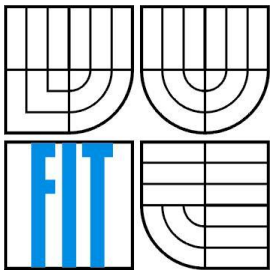
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MAREK MEŠÁR
AUTHOR

BRNO 2013



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

SVĚT KOLEM NÁS JAKO HYPERLINK

LOCAL ENVIRONMENT AS HYPERLINK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK MEŠÁR

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÍTĚZSLAV BERAN, Ph.D.

BRNO 2013

Abstrakt

Tento dokument popisuje vybrané metody a přístupy k problému detekce, extrakce a rozpoznání textu na moderních mobilních zařízeních. Také popisuje jejich vhodnou prezentaci v uživatelském rozhraní a jejich přeměnu na hypertextové odkazy jako zdroj informací o okolním světě. Dokument nastiňuje vyhledávací a rozpoznávánící techniku založenu na detekci MSER oblastí a také popisuje použití obrazových příznaků pro metodu sledování a odhad pohybu textu.

Abstract

This document describes selected techniques and approaches to problem of text detection, extraction and recognition on modern mobile devices. It also describes their proper presentation to the user interface and their conversion to hyperlinks as a source of information about surrounding world. The paper outlines text detection and recognition technique based on MSER detection and also describes the use of image features tracking method for text motion estimation.

Klíčová slova

Android, MSER, detekce textu, rozpoznání textu, OCR, digitalizace textu, tracking

Keywords

Android, MSER, text detection, text recognition, OCR, text digitalization, tracking

Citace

Mešár Marek: Svet okolo nás ako hyperlink, diplomová práce, Brno, FIT VUT v Brně, 2013

Svet okolo nás ako hyperlink

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana
Ing. Vítězslava Berana, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marek Mešár
17. 5. 2013

Poděkování

Chcel by som sa poďakovať svojmu vedúcemu Ing. Vítězslavu Beranovi, Ph.D. za odbornú pomoc, konzultácie, vedenie a ústretovosť poskytnutú pri riešení môjho projektu.

Obsah

1 Úvod

Internet má od svojho vzniku koncom minulého storočia čoraz väčší význam v každodennom živote jednotlivcov. Jeho hlavná úloha sa odvtedy výrazne nezmenila. Bežnému užívateľovi slúži okrem iného aj ako nevyčerpateľný zdroj informácií a faktov o svete naokolo. Podľa štatistík sa počet vyhľadávania informácií pomocou najpoužívanejšieho vyhľadávacieho systému od spoločnosti Google každý rok neustále zvyšuje vysokým tempom.

Ďalším z fenoménov dnešnej doby sú mobilné telefóny. Za posledných 10 rokov nastal obrovský nárast ich predaja na všetkých významných trhoch po celom svete. Stali sa tak neodmysliteľnou súčasťou každodenného života moderného človeka. Takmer všetky tieto zariadenia poskytujú svojim užívateľom plnohodnotný prístup k sieti internet. Väčšina moderných telefónov je tiež vybavená vbudovanou kamerou. Kamera je z pohľadu užívateľa jeden z najprístupnejších spôsobov ako poskytnúť zariadeniu informácie o predmetoch okolitého prostredia. Nevyžaduje si zložité ovládanie, či ich textový popis, stačí ju inicializovať a namieriť do scény.

Práve skombinovaním siete internet ako zdroja informácií so vstupom obrazu prostredia z kamery mobilného telefónu sa naskytá výborná možnosť ako uľahčiť užívateľom prístup k informáciám o svojom bezprostrednom okolí.

Samotné priradenie významu jednotlivým častiam obrazu, resp. identifikovanie zachytených objektov na ňom je úloha pomerne zložitá a bez dodatočnej interakcie s užívateľom veľmi ťažko riešiteľná. V praxi si ľudia nezanedbateľnú časť informácií o okolí a svete vymieňajú tiež v textovej podobe. Sú to napríklad názvy obchodov vystavené na ulici, rôzne reklamné slogany, plagáty, značky a podobne. Pri riešení môjho projektu sa zameriam primárne na túto formu informácií. Hlavnú ideu prezentuje obrázok č.1.

Obrázok 1: Ilustrácia hlavnej idei riešenia.

Predmetom riešenia projektu je teda návrh a implementácia mobilnej aplikácie spracúvajúcej textové informácie z okolia získané pomocou obrazového vstupu zo snímača videa mobilného zariadenia. Spracovávaný obraz je získavaný z reálnych scén, často z exteriéru. Za cieľ si kladie na základe spracovaného textu z obrazu umožniť užívateľom rýchlejšie a efektívnejšie získanie širokého spektra informácií o entitách v ich okolí.

V druhej kapitole uvediem základné informácie ktoré tvoria teoretický základ riešenia projektu. V podkapitole 2.1. rozoberiem doteraz existujúce podobné riešenia a aplikácie zaoberajúce rovnakým, alebo podobným typom problému. Podkapitola 2.2 popisuje špecifiká extrakcie textu z obrazu reálnych scén. Jej súčasťou je aj stručný prehľad súčasných metód zaoberajúcich sa spracovaním obrazu na text. V podkapitole 2.3. uvediem postup výberu a bližšiu špecifikáciu platformy na ktorej je vzniknutá aplikácia vyvíjaná. Uvediem tu tiež stručný všeobecný popis kľúčových častí aplikácie a možnosti vývoja aplikácií na vybraný operačný systém.

Kapitola č. 3 predstavuje návrh a možné postupy riešenia daného problému. Stručne predstavuje používané algoritmy a základný princíp ich fungovania. Podkapitola 3.1 sa venuje procesu spracovania obrazu na text. Postupne predstaví navrhnutú metódu riešenia a postupy z ktorých som pri vytváraní aplikácie vychádzal. Podkapitola je delená na menšie celky podľa jednotlivých krokov metódy spracovania obrazu. V sekcií 3.2 rozoberám problém zobrazenia výsledku spracovania obrazu do užívateľského rozhrania. Špecifikujem tu požiadavky na užívateľské rozhranie aplikácie a rozoberiem problémy vyplývajúce z jeho charakteru. Podkapitola 3.3 predstaví užívateľovi navrhnutý postup sledovania pohybov textu v scéne. Podkapitola je rozdelená podľa jednotlivých krokov navrhnutého postupu.

V kapitole číslo 4 prezentujem postup implementácie riešenej aplikácie. Kapitola predstaví niektoré zaujímavé časti implementácie, stručne vysvetlí a popíše objektový model aplikácie. Nástroje a technológie použité pri implementácií sú stručne predstavené v podkapitole 4.1. Podkapitola nasledujúca po nej popisuje hlavé triedy a ich metódy riadiace celý beh programu. V podkapitole 4.3 sa bližšie venujem implementácií procesu spracovania obrazu na text. Je rozdelená na dve časti, podľa druhu použitého programovacieho jazyka. Podkapitola tiež pojednáva o operáciách potrebných na prepojenie primárnej a native časti programu. Ďalšia podkapitola má číslo 4.4. Rozoberám v nej implementáciu procesu sledovania pohybu textu v scéne a popisujem triedy a metódy tvoriace tento proces. V podkapitole 4.5 krátko predstavujem techniky použité pri implementácií procesu trénovania modelu klasifikácie MSER oblastí.

V kapitole číslo 5 uvádzam popis uskutočnených testov navrhnutého algoritmu, zhodnocujem ich výsledky a načrtám ďalšie možnosti riešenia vzniknutých problémov. Kapitola je členená na podkapitoly podľa zamerania jednotlivých testov.

2 Teória

V úvode kapitoly sa nachádza rozbor existujúcich aplikácií zaoberajúcich sa rovnakým, alebo podobným problémom. Snaží sa predstaviť dobré aj zlé vlastnosti nájdených aplikácií. Nasleduje popis špecifických vlastností extrakcie textu z obrazu reálnych scén a stručný prehľad uverejnených metód autorov zaoberajúcich sa touto témou. Posledná podkapitola bližšie špecifikuje mobilnú platformu na ktorej budem dané riešenie implementovať, uvedie jej základné vlastnosti a špecifiká vývoja aplikácií na zvolenej platforme.

2.1 Existujúce riešenia

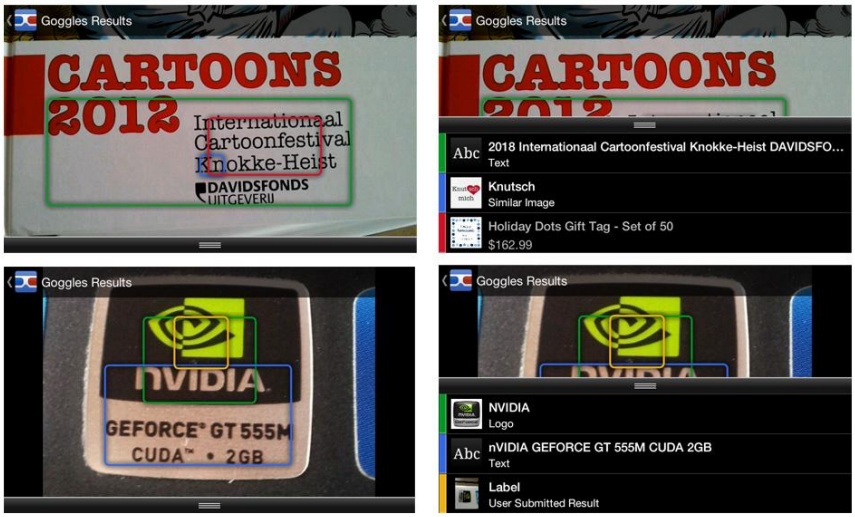
Dôležitým krokom pri riešení zadanej úlohy je zistenie a analýza podobných existujúcich riešení, vyhľadanie aplikácií zaoberajúcich sa extrakciou textu priamo, alebo z časti. Nielen že som sa vďaka nim mohol inšpirovať z ich dobrých vlastností, dopomohli mi aj k identifikácií možných problematických oblastí riešenia. Hlavným zdrojom hľadania podobných aplikácií boli najznámejšie platformy digitálnej distribúcie mobilných aplikácií - služby Google Play[1] a App Store[1].

Aplikáciou, ktorá je najbližšie riešeniu môjho problému je aplikácia *Google Goggles*[2]. Aplikácia je vyvíjaná spoločnosťou Google. Funguje na mobilných zariadeniach s operačným systémom Android a iOS. Pomocou obrázka z videokamery zariadenia dokáže vyhľadávať na internete nasnímané objekty a texty. Je vybavená množstvom praktických funkcií a možností. Zvláda vyhľadávanie známych kultúrnych pamiatok, umeleckých obrazov, kníh, CD alebo DVD obalov, informácií o produktoch pomocou čiarového či QR kódu. Aplikácia tiež dokáže riešiť hlavolamy sudoku, prekladať odfotené texty z rôznych jazykov a digitalizáciu textov pomocou technológie OCR. Užívateľské rozhranie je veľmi moderné a logické. Podľa typu vstupu pracuje aplikácia v troch základných módoch. Užívateľ môže zvoliť na spracovanie obrázok z pamäte zariadenia, alebo priamo obrazový vstup z kamery. Ten môže byť vo forme videa, alebo fotografie. Na fotografii sa tiež dá označiť oblasť záujmu.

Pri testovaní aplikácie som sa primárne zameral na detekciu a rozpoznávanie textových oblastí. Na fotografiách a obrázkoch pracovala aplikácia pomerne spoľahlivo pokiaľ bol text v horizontálnej polohe voči pohľadu kamery. Kvalita spracovania textov bola tiež pomerne citlivá na rozmazanie obrazu, jeho osvetlenie a font textu. Problematické sa tiež ukázalo komplexnejšie pozadie v snímke s textom. Takéto pozadia sa však vyskytujú v nezanedbateľnom počte snímok vytvorených v teréne. Aplikácia oproti tomu dobre zvláda väčšie textové bloky na viacerých riadkoch, ktoré sú kontrastné oproti svojmu pozadiu. Testovanie tiež ukázalo, že detekcia textov vo videu je pre aplikáciu

[1] https://play.google.com/store ; http://www.apple.com/osx/apps/app-store.html
[2] http://play.google.com/store/apps/details?id=com.google.android.apps.unveil

problematická. Samotná detekcia a rozpoznanie textov v snímkach často trvala aj niekoľko sekúnd. Rýchlosť spracovania závisí aj od rýchlosti internetového pripojenia, keďže aplikácia posiela zachytené snímky na server, kde sa deje ďalšie výpočetne náročné spracovanie. Aplikácia čaká na spätné zaslanie výsledkov, ktoré následne prehľadne zobrazí. Bez internetového pripojenia nie je použitie aplikácie možné.



Obrázok 2: Ukážka výstupov a užívateľského rozhrania aplikácie Google Goggles.

Ďalšia aplikácia zaoberajúca sa podobnou problematikou je *Nokia Point & Find*[3]. Umožňuje identifikáciu význačných bodov v mestách, skenovanie čiarových kódov a následné zisťovanie informácií o nájdených produktoch. Užívatelia môžu jednotlivým objektom reálneho sveta priradzovať kľúčové značky, na základe ktorých ich neskôr majú možnosť iní ľudia identifikovať a získať o nich ďalšie informácie. Podobným systémom zadávania kľúčových značiek a oblastí (tzv. *tagovanie*) môžu na seba upozorňovať aj firmy či vydavatelia kníh, filmov a podobne. Aplikácia je vyvíjaná spoločnosťou Nokia a jej použitie je možné iba na pomerne malom množstve nimi vyrobených zariadeniach. Pre jej fungovanie je tiež vyžadovaný prístup k internetu.

Dobrou vlastnosťou aplikácie je real-time rozoznávanie objektov vo video streame. Objekty sú porovnávané na základe centrálnej databázy uloženej na serveroch spoločnosti. Ak sa nájde zhoda, aplikácia identifikuje objekt z obrazu, zobrazí výsledok a umožní ďalšie akcie (zobrazenie informácií, ceny produktu, možné obchody a podobne).

Hlavná nevýhoda aplikácie spočíva najmä v tom, že ak obsah nebol predtým užívateľmi alebo spoločnosťou označený ako interaktívny, nezobrazia sa užívateľom o objekte žiadne informácie. Touto nevýhodou trpí aj podobná aplikácia pre mobilné telefóny *Layar*[4].

Aj keď je primárne určenie množstva dostupných aplikácií od môjho odlišné, vo svojom procese spracovania rieši podobný podproblém akým sa zaoberám pri riešení ja. Tým je detekcia a spracovanie textu z obrazu z kamery. Patria tu napríklad rôzne aplikácie slúžiace ako mobilné

[3] http://betalabs.nokia.com/trials/nokia-point-and-find
[4] http://www.layar.com/

prekladače (*Photo Translator*[5]), či aplikácie slúžiace na digitalizáciu odfotených textových poznámok (*Mobile OCR Pro*[6], ImageReader(OCR)[7], Scanthing OCR[8]). Oba typy aplikácií očakávajú ako vstup odfotené dokumenty, na ktorých majú texty špecifické vlastnosti. Text je oproti pozadiu dostatočne kontrastný a má štandardný font, nesmie byť výrazne natočený a línia textu by mala byť horizontálna. Tieto vlastnosti textu však často pri fotografiách zachytených vo vonkajšom prostredí často nemôžeme zaručiť.

Celkovo sa dá predpokladať, že proces spracovania obrazu na mobilnom zariadení bude náročný na výpočetné zdroje. Trvalým trendom v moderných mobilných aplikáciách je snaha o čo najväčšiu interaktivitu a interakciu užívateľa s jeho bezprostredným prostredím.

2.2 Detekcia a rozpoznávanie textu v obraze

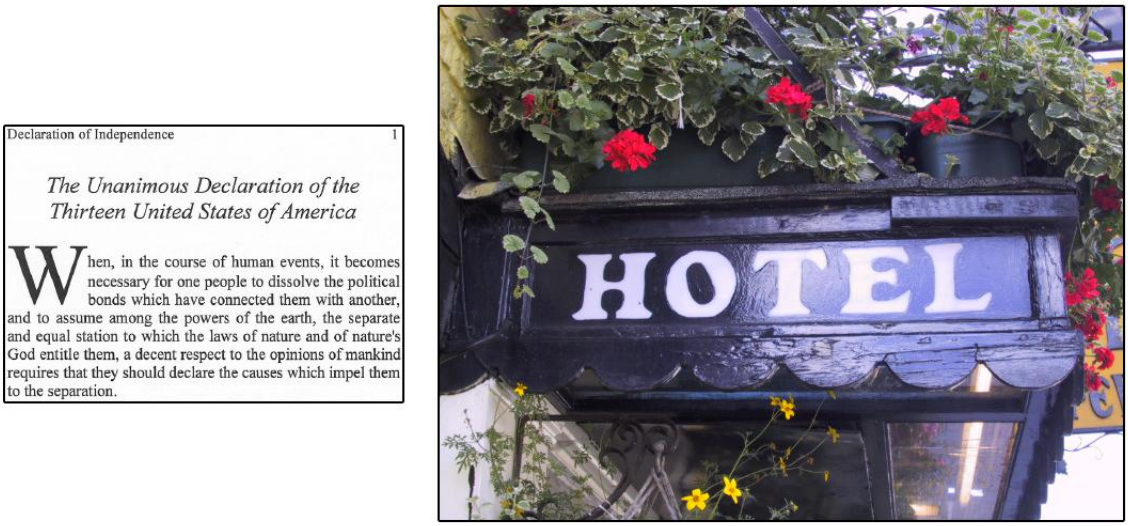
Jadro implementovanej aplikácie je tvorené metódou detekcie a rozpoznávania textu z dodaného obrazu. Voľba vhodného prístupu k riešeniu tohto problému je pri implementácií aplikácie kľúčová. Ovplyvňuje celkový výkon aplikácie z hľadiska presnosti a rýchlosti spracovania textu v obraze. Na základe týchto vlastností posudzuje aplikáciu ako celok jej koncový užívateľ.

Problém detekcie a rozpoznávania textu z obrazu je možné riešiť viacerými prístupmi. Prvým z nich je použitie existujúcich metód OCR (*optical character recognition*) používaných na spracovanie textu z obrazov naskenovaných tlačených dokumentov. Najčastejšie používané prístupy [1, 2] v dnešnej dobe dosahujú dobré výsledky. Existuje množstvo nástrojov[9], ktoré danú funkcionalitu poskytujú širokej verejnosti. Priama extrakcia textu z obrázkov reálnych scén pomocou OCR metód však nie je možná, keďže sú primárne určené na skenované dokumenty.

Spracovanie textov zo skenovaných tlačených dokumentov sa od spracovania obrazov reálnych scén odlišuje viacerými špecifikami s ktorými majú tieto metódy ťažkosti. Umiestnenie a zarovnanie textu v takýchto obrázkoch nemá oproti tlačeným dokumentom žiadne pevne dané pravidlá. Texty sú často písané okrasnými alebo inak neštandardnými druhmi písma, často v rôznych jazykoch. Nezriedka sa stáva, že texty sú perspektívne skreslené a rôzne natočené, teda aj ich pozadie a farba sú premenlivé. Pozadie textu býva na presné rozpoznanie textu pomocou OCR metód často príliš komplexné. Texty môžu byť umiestnené napríklad na sklenených plochách, alebo v tesnej blízkosti rušivých elementov ako tráva, či koruny stromov. Niekedy sa stáva, že texty nie sú voči svojmu pozadiu dostatočne kontrastné. Výnimočné na takýchto obrázkoch nie sú ani rôzne svetelné odlesky, tiene a presahy. V obrazoch z reálnych scén je často ťažké správne určiť už aj pozíciu a rozmery

[5] http://play.google.com/store/apps/details?id=com.smartmobilesoftware.phototranslator
[6] http://play.google.com/store/apps/details?id=com.smartmobilesoftware.mobileocrpro
[7] http://play.google.com/store/apps/details?id=com.nullproduct.imagereader
[8] http://play.google.com/store/apps/details?id=com.scanthing.android
[9] http://ocr-software-review.toptenreviews.com/

všetkých textových oblastí zdrojového obrázku. Pri riešení daného problému je preto potrebné použiť komplexnejší prístup.



Obrázok 3: Porovnanie rozpoznávania textu zo skenovaných dokumentov (*vľavo*)[10] a z obrázkov reálnych scén (*vpravo*)[11].

Práce autorov zaoberajúce sa problematikou získavania textu z fotografií reálnych scén sú zväčša založené na sekvenčnom spracovaní vstupného obrazu. Väčšinou sa skladá z týchto troch základných krokov:

- lokalizácia textových oblastí
- segmentácia a binarizácia textu
- rozpoznanie textu

Sekvenčné spracovanie však prináša nevýhodu v tom, že celková kvalita výsledku závisí na kvalite výsledkov jeho jednotlivých krokov.

Vo všeobecnosti je možné detekciu textu a jeho rozpoznanie v obraze riešiť viacerými prístupmi. Riešenia sa dajú rozdeliť do dvoch hlavných kategórií.

- metódy pracujúce zdola-hore (*bottom-up*)
- metódy pracujúce zhora-dole (*top-down*)

Metódy postupujúce analýzou **zdola-hore** segmentujú vstupný obraz do menších regiónov, ktoré následne klasifikujú na regióny obsahujúce a neobsahujúce znaky textu. Regióny písmen potom spájajú do väčších skupín formujúc oblasti obsahujúce slová, či celé riadky textu. Úspešnosť celého procesu je silne závislá na segmentačnom algoritme a komplexnosti vstupného obrazu.

Druhú skupinu metód tvoria metódy pracujúce prístupom **zhora-dole**. Tieto algoritmy najskôr vo vstupnom obraze hľadajú regióny textu, ktoré následne segmentujú na text a pozadie. Spomínané metódy majú však často problém rozlíšiť textové regióny od komplexnejšieho pozadia (koruna stromov, tráva...).

[10] http://careercenter.tamu.edu/support/filesizes/scanFileSize.cfm?sn=former
[11] Obrázok zo sady ICDAR 2003. Súbor dát je predstavený v ďalších kapitolách.

Množstvo autorov sa tiež pokúsilo vytvoriť metódy spájajúce oba zmieňované postupy a obísť tak niektoré často sa vyskytujúce problémy (komplexné pozadie textu, rôzne skreslenia, skosenia a natočenia textových riadkov).

Kandidátne oblastí textu či znakov sa hľadajú v zdrojovom obraze pomocou rôznych príznakov, či komplexnejších vlastností písmen a textu ako celku. Často používané príznaky sú napríklad pomer medzi veľkými a malými písmenami, kontrast medzi znakmi a pozadím, uniformná farebnosť znakov (pozadia), frekvencia zmien hodnôt v obraze hrán, počet vnútorných dier v písmene, maximálna a minimálna veľkosť detekcie znakov a podobne. Príznaky sa v množstve metód využívajú aj pri kroku segmentácie textu od pozadia, či pri spájaní znakov do slov a riadkov.

Niektorí autori sa pri svojom riešení zameriavajú len na podproblém detekcie a segmentácie textu, iní ponímajú problém všeobecnejšie a snažia sa aj o rozpoznávanie nájdených textových reťazcov. Viacerí používajú pri kroku rozpoznávania komerčné OCR systémy určené na spracovanie binarizovaných skenovaných dokumentov. Nasleduje stručný prehľad niektorých prezentovaných metód detekcie textov v obraze.

Autori Kasar, Kumar a Ramakrishnan [3] získali zo vstupného obrazu mapu hrán objektov kombináciou výstupov Cannyho hranového detektora aplikovaného na každý farebný kanál obrazu. Z nej následne extrahujú všetky spojené komponenty pomocou metódy pracujúcej na princípe 8-susedstva pixelov. Takto získané objekty a ich hraničné obálky (bounding box) analyzujú na základe špeciálne navrhnutého modelu pre reprezentáciu písmen (napríklad počet vnútorných dier nesmie byť viac ako 2, pomer medzi výškou a šírkou atď.). Tým odfiltrujú objekty, ktoré nemôžu byť písmenami. Na odstránenie iných anomálií a šumov používajú predpoklad, že jednotlivé znaky majú približne rovnakú hodnotu intenzity. Problémom metódy je najmä slabá segmentácia textov obklopených komplexným pozadím s veľkou frekvenciou zmien jeho intenzity (napr. lístie).

Autori Tran, Lux, Nguyen a Boucher [4] sa pokúsili vyvinúť metódu detekcie textových oblastí nezávislú od orientácie, veľkosti a farby písmen. Ich metóda je založená na štruktúrnom modeli textu. Príznaky používané týmito autormi sú takzvané hrebene (*ridges*, *ridge points*) vyhľadávané vo viacerých mierkach obrazu. Analýza príznakov v rôznych mierkach obrazu autorom umožňuje získanie nielen lokálnych informácií o oblastiach textu, ale aj celkový globálny pohľad o ich tvare. Riadok textu je uvažovaný ako štruktúrovaný objekt. Pri nižších rozlíšeniach jednotlivé znaky riadku textu akoby splynú v jeden dlhý zhluk. Hrebene pri nižších rozlíšeniach tak reprezentujú stredovú líniu textových reťazcov. Pri vyšších rozlíšeniach reprezentujú kostry (tvar) jednotlivých písmen. Hlavnou výhodou tohto prístupu je, že je nezávislý na abecede, dokáže si poradiť dokonca s ručne písaným textom. Nepravidelné pozadie je tak isto ako pri predošlom prístupe detekcie problémom.

Liu, Jung a Moon [5] vytvorili algoritmus využívajúci vnútorné charakteristiky textu. Najskôr využívajú obrazový filter detekujúci ťahy (*strokes*). Sú to horizontálne, alebo oblúkové segmenty, objavujúce sa v určitej vopred definovanej hustote v spracovávanej oblasti, ktoré sú homogénne

v ponímaní intenzity. Takto získané príznaky vyjadrujú orientáciu textu a jeho veľkosť. Následne je aplikovaná metóda na určenie polarity (tmavosť resp. svetlosť) textu založená získaných štruktúrach z predošlého kroku. V poslednom kroku sú na základe ďalšej analýzy získané oblasti znakov rozširované o svoje okolie tak, aby sa získali celé textové oblasti. Metóda je pomerne rýchla, dokáže si poradiť s rôznym šumom hrán či farby znakov. Avšak nevie detekovať text, ak je textová oblasť menšia ako 10 obrazových bodov, alebo ak textová oblasť obsahuje tmavý aj svetlý text (napr. farebný prechod). Určité problémy pri použití spomínanej metódy môžu tiež nastať pri textoch obklopených komplexným pozadím.

V práci autorov Sobottka, Bunke a Kronenberg [6] je popísaná iná metóda detekcie a segmentácie textových oblastí. Jej princípom je vytvorenie hypotéz o elementoch textu na základe dvoch rôznych analýz. Prvá analýza je založená na rozdelení obrázku striedavo v horizontálnom a vertikálnom smere na oblasti pravouhlého tvaru. Autori predpokladajú, že oblasti obsahujúce text budú takisto obsahovať body pozadia, teda minimálne dve rôzne farby. Farebne homogénne oblasti sú teda pri segmentácií odmietnuté ako netextové. Pomocou druhej analýzy sa autori pokúšajú hľadať farebne homogénne regióny ľubovoľného tvaru pomocou metódy rozširovania regiónov. Deje sa tak aplikáciou techniky na spájanie bodov prislúchajúcich rovnakému segmentu obrazu. Autori ďalej predpokladajú, že riadky textu smerujú v obraze horizontálne. Obe segmentačné analýzy vytvoria na základe tohto predpokladu a získaných oblastí skupiny znakov reprezentujúcich riadky textu (slová). Výsledné kandidátne oblasti sú získané kombináciou hypotéz oboch predošlých analýz. Kandidátne oblasti sú následne binarizované na základe farby textu.

Autori Park, Moon a Oh [7] navrhli na detekciu textu metódu založenú na robustnej morfologickej analýze vstupného obrazu. Metóda pozostávala z niekoľkých krokov. Najskôr je použitá analýza smeru z hraničnej mapy obrazu pomocou *Prewittových operátorov*. Následne sú hrany klasifikované do dvoch skupín, na hrany idúce diagonálne, alebo hrany idúce horizontálne resp. vertikálne. V ďalšom kroku sú odstránené izolované hraničné pixely. Potom je použitá na body oboch skupín hraničných obrazov operácia morfologickej dilatácie. Dilatované binárne obrazy hrán sú skombinované pomocou logického operátoru AND vytvárajúc kandidátne textové oblasti. Metóda je primárne určená na detekciu v naskenovaných textových dokumentoch, preto by jej mohlo robiť problém komplexné pozadie.

V dokumente [8] popisujú autori Wolf, Jolin a Chassaing svoju metódu lokalizácie a segmentácie textových oblastí. Vo svojom prístupe predpokladajú, že znaky vytvárajú pravidelný vzor obsahujúci vertikálne elementy, ktoré sú zoradené horizontálne. Pri detekcii znakov v obraze mierne modifikovali z metódu autora LeBourgeois [9]. Výstup následne podrobili ďalšej matematickej a morfologickej analýze, ktorými sa snažili eliminovať regióny nesprávne klasifikované kvôli rušivým efektom pozadia a podobne. Pomocou analýzy tiež spájali kandidátne oblasti znakov do slov a väčších textov.

V metóde prezentovanej autormi Chucai a Tian [10] je obraz rozdeľovaný na kandidátne oblasti písmen na základe farebnej uniformity znakov a tiež na základe prechodov hrán objektov. Nesprávne identifikované oblasti sú následne filtrované pomocou analýzy založenej na príznakoch spojených komponentov ako veľkosť, počet vnútorných dier a pomer ich výšky a šírky. V ďalšom kroku spracovania sú kandidátne písmena spájané do slov a textov pomocou rôznych štruktúrnych príznakov v dvoch rôznych prístupoch. Aj keď si metóda dokáže poradiť s textami v rôznych pozíciách, veľkostiach, tvaroch a na rôznych povrchoch, jej hlavnou nevýhodou je pomerne vysoká výpočetná náročnosť odzrkadľujúca sa na dlhom čase spracovania snímok.

V súčasnosti je jednou z najúspešnejších metód z hľadiska jej robustnosti a presnosti metóda prezentovaná autormi L. Neumannom a J. Matasom v [11]. Na detekciu a extrakciu textov od pozadia využívajú detekciu *maximálnych stabilných extrémnych oblastí* (MSER). Vyhľadané oblasti sú následne na základe príznakov spájané do hypotéz o slovách a riadkoch. Hypotézy sú v ďalšom spracovaní validované a podrobené analýze ich priestorového skreslenia. Po ich geometrickej normalizácií sú textové znaky rozpoznávané a riadky a slová opäť validované na základe vytvoreného typografického a jazykového modelu. Výstupy kroku vytvárania hypotéz o slovách a kroku validácie na základe jazykového modelu sú spresňované viacerými iteráciami ich predchádzajúcich krokov. Hlavnou výhodou takéhoto spracovania je hlavne výborná presnosť extrakcie textu z obrazu z reálnych scén. Metóda je tiež robustná z hľadiska geometrickej variability a natočenia textu. Problémom je hlavne spracovanie neostrých a rozmazaných obrázkov. Ťažkosti sa môžu objaviť tiež pri spracovaní textu, ktorý nie je dostatočne kontrastný oproti svojmu pozadiu. Pri riešení svojho projektu som sa rozhodol použiť ako základ detekcie a extrakcie textu práve túto metódu.

2.3 Mobilná platforma

Implementácia mobilnej aplikácie extrahujúcej texty z real-time videa reálnych scén sa vyznačuje viacerými špecifikami a obmedzeniami. Detekcia a rozpoznávanie textov musí byť rýchle a výkonné. Operácie nad obrázkami sú vo všeobecnosti pomerne náročné. Vyžadujú si teda dostatočne výkonné zariadenia.

Aj keď je v súčasnosti väčšina telefónov nižšej a strednej triedy vybavených fotografickým snímačom, ich výpočetný výkon sa javí na spracovanie obrazu v reálnom čase ako nedostatočný. Ďalším problémom je veľká variabilita operačných systémov, ktoré na nich bežia. V praxi sa stretávame s tým, že každý výrobca mobilných zariadení používa vlastnú variáciu systému špeciálne upravenú pre jeho zariadenia. Vyvíjaná aplikácia by teda musela byť prispôsobená každému z nich. Navyše by sme museli počítať s veľkou variáciou verzií jednotlivých operačných systémov.

Riešením je zamerať sa na mobilné zariadenia vyššej triedy, takzvané smartfóny. Podľa posledných trendov na trhoch vyspelých krajín a aj globálne sa smartfóny presadzujú medzi zákazníkmi čoraz viac. V budúcnosti predpokladáme pokračovanie tohto trendu a postupné

prenikanie technológií a softvérového vybavenia aj do zariadení strednej a nižšej triedy. Smartfóny vynikajú oproti iným zariadeniam nižších tried najmä použitím najvyspelejších technológií a svojim softvérovým vybavením. V súčasnosti už nie sú výnimkou ani telefóny s viacjadrovými procesormi a gigabajtami operačnej pamäte. Ich najväčšou výhodou je však to, že na nich bežia pokročilé operačné systémy umožňujúce rýchly a pomerne ľahký vývoj nových aplikácií. Medzi najrozšírenejšie dnes patria *OS Android* vyvíjaný organizáciou Open Handset Alliance, *iOS* od spoločnosti Apple, *Windows Phone* od spoločnosti Microsoft. Najvšestrannejším a celkovo v súčasnosti najrozšírenejším z nich je systém Android. Vývoj aplikácií určených pre túto platformu nie je nijak obmedzený poplatkami a špeciálnymi licenciami. Keďže tento systém používajú vo svojich zariadeniach viacerí hlavní výrobcovia telefónov, je medzi nimi zaručená aj presnosť aplikácií. Android sa čoraz viac dostáva aj na iné zariadenia ako sú napríklad tablety a podobne. Vývoj svojej aplikácie budem kvôli zmieňovaným výhodám smerovať na smartfóny so systémom Android.

2.3.1 Android OS

Po zvolení vhodnej platformy bolo potrebné dôkladne preštudovať jej možnosti, výhody a nevýhody a hlavné princípy fungovania jej aplikácií. Nasleduje stručná charakteristika operačného systému Android, jeho princíp fungovania a systém vývoja aplikácií pre tento systém. Ako výborný zdroj informácií či už o jeho fungovaní ale aj o samotnom vývoji aplikácií na tejto platforme sa ukázala najmä jeho on-line dokumentácia a popis rozhrania API [12]. Nasledujúce informácie boli čerpané tiež z [13, 14].

Základné vlastnosti

Operačný systém Android je vyvíjaný organizáciou Open Handset Alliance (Google, Samsung, HTC, Motorola, Intel, NVidia, Qualcomm a ďalšie). Ide o moderný a plnohodnotný operačný systém s užívateľským rozhraním, špecifikáciou ovládačov a podporou externých aplikácií.

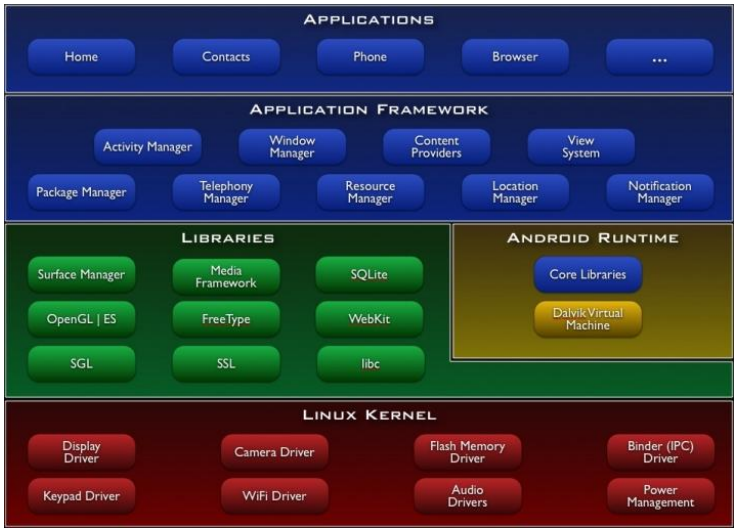
Najväčšou výhodou systému Android je jeho nezávislosť na hardvérovom vybavení zariadenia na ktorom beží. Je nezávislý na veľkosti obrazovky a jej rozlíšenia. Popisovaný operačný systém je otvorené open source riešenie, vrátane modulov linuxového jadra, používaných knižníc až po jeho základné programové rozhranie a vybavenie. Na využitie pre komerčné účely je potrebné zakúpenie licencie. Momentálne je najaktuálnejšia verzia 4.2 označovaná tiež názvom *Jelly Bean*. Ak chceme pri vývoji aplikácie obsiahnuť aspoň 90% všetkých zariadení, mala by aplikácia podporovať prvky systému dostupné vo verzii 2.3 (tabuľka č. 1). Vývojári aplikácií majú tiež k dispozícii prostriedky zaručujúce spätnú kompatibilitu s nižšími verziami operačného systému pri použití niektorých prvkov z vyššej verzie.

Verzia	Meno	API	Zastúpenie
1.6	Donut	4	0,1%
2.1	Eclair	7	1,7%
2.2	Froyo	8	3,7%
2.3 – 2.3.2	Gingerbread	9	0,1%
2.3.3 – 2.3.7		10	38,4%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	27,5%
4.1.x	Jelly Bean	16	26,1%
4.2.x		17	2,3%

Tabuľka 1: Zastúpenie verzií OS Android k 1.5.2013. Čerpané z [15] .

Operačný systém je založený na linuxovom jadre vo verzii 2.6, ktorá sa skladá z viacerých vrstiev (obrázok č. 4). Abstrahovaním hardwaru od softwarovej časti garantuje portabilitu na rôzne platformy. Obsluhuje množstvo služieb ako sú napríklad riadenie procesov, správa využívania operačnej pamäte, ovládače zariadení a periférií, obsluha bezpečnosti a mnoho iných.

Vrstva Android Runtime poskytuje sadu základných knižníc implementujúcich základné funkcionality knižníc programovacieho jazyka Java. Neobsahujú však implementáciu knižníc užívateľského rozhrania AWT a Swing. Tie boli nahradené špeciálnymi knižnicami obsluhujúcimi užívateľské rozhranie systému Android. Kvôli optimalizácii na mobilné zariadenia (výkon a spotreba energie) bol implementovaný špeciálny virtuálny stroj *Dalvik* (DVM). Každá spustená aplikácia spúšťa svoj vlastný proces s vlastnou inštanciou DVM. Virtuálny stroj Dalvik bol implementovaný a optimalizovaný tak, aby mohlo na zariadení efektívne bez problémov bežať niekoľko DVM inštancií naraz. Súbory aplikácií sú v špeciálnom formáte *Dalvik Executable* (.dex). Sú to vlastne triedy kompilované v jazyku Java, ktoré boli optimalizované z hľadiska minimálnej spotreby pamäte a prevedené na *.dex* formát pomocou špeciálneho nástroja.



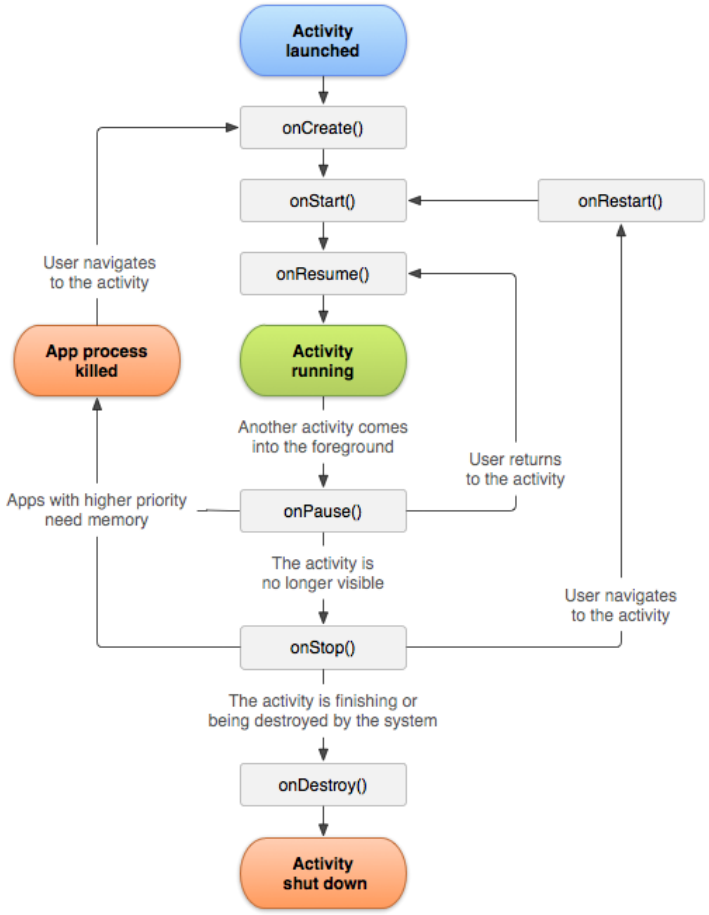
Obrázok 4: Architektúra operačného systému Android. Prevzaté z [13].

Vrstva podporných knižníc operačného systému je implementovaná prevažne v jazyku C/C++. Ich funkcionalitu vyžívajú najrôznejšie komponenty systému. Vývojár sa k ním dostane pomocou vrstvy aplikačného frameworku (*Android application framework*). Zapuzdrujú napríklad štandardné systémové C knižnice pre Linux (*libc*), podporu prehrávania a nahrávania rôznych multimediálnych formátov, zobrazovanie bitmapových a vektorových fontov, obsluhu podsystémov obrazovky a zobrazovania 2D a 3D grafiky a podobne.

Aplikačné rozhranie (Android application framework) poskytuje vývojárom aplikácií prístup k všetkým vlastnostiam a funkcionalitám poskytovanými knižnicami systému. Vývojári majú plný prístup k celému API systému a aj k službám základných aplikácií. Táto vrstvová architektúra je navrhnutá tak, aby zabezpečovala ľahké a transparentné použitie všetkých jej komponentov. Každá aplikácia môže zverejňovať svoje vlastnosti a schopnosti. Tie môžu neskôr využívať ďalšie aplikácie. Všetky aplikácie využívajú základný súbor služieb a systému ako:

- **Pohľady** (*Views*) – môžu byť použité na vytáranie aplikácií, zoznamov, mriežok, textových polí, tlačidiel, alebo dokonca vbudovaného webového prehliadača. Sú plne rozšíriteľné a poskytujú množstvo funkcií.
- **Poskytovatelia obsahu** (*Content Providers*) – dovoľujú aplikáciám prístup k dátam z iných aplikácií a naopak.
- **Manažér zdrojov** (*Resource Manager*) – poskytuje prístup k zdrojom, ktoré nie sú viazané na kód aplikácie (napríklad textové polia, grafika, súbory vzhľadu atď.)
- **Manažér upozornení** (*Notification Manager*) – umožňuje aplikáciám zobrazovať špecifické upozornenia v stavovom paneli systému.
- **Manažér aktivít** (*Activity Manager*) – obsluhuje životný cyklus aplikácií.

Samotná aplikácia sa v systéme Android skladá zo súboru *aktivít*. *Aktivity* sú základné stavebné bloky aplikácie, triedy obsluhujúce ich hlavnú funkcionalitu, vytvárajú okná pre zobrazenie užívateľského rozhrania. Jedna aplikácia je spravidla zostavená z viacerých aktivít. Jedna aktivita reprezentuje jeden pohľad užívateľa v aplikácii. Počas behu aplikácie je vždy aktívna a beží v popredí najviac jedna aktivita. Z nej sa môže užívateľ dostať do inej aktivity (iného pohľadu). Predchádzajúce aktivity môžu bežať na pozadí a sú ukladané na zásobník aktivít. V prípade, že aktivita z popredia bola užívateľom uzavretá, vyberie sa do popredia zo zásobníku predchádzajúca aktivita. Ak nie je na zásobníku žiadna ďalšia aktivita danej aplikácie, dôjde k jej ukončeniu. Detailný životný cyklus prezentuje obrázok č. 5.



Obrázok 5: Životný cyklus aktivity. Prevzaté z [14].

Vo všeobecnosti má každá aktivita priradenú práve jednu obrazovku rozhrania. Tá je popísaná hierarchickou štruktúrou objektov triedy *View*, alebo *ViewGroup*.

Objekty *View* a *ViewGroup* reprezentujú celú štruktúru obrazovky, ktorá obsahuje ďalšie prvky užívateľského rozhrania (tlačidlá, nápisy, textové editačné prvky...). Tieto štruktúry je možné ďalej do seba vkladať, vytvárať komplexnejšie celky. Hierarchická štruktúra danej obrazovky je popísaná pomocou súborov vzhľadu typu XML. Objekty triedy *View* sú v ňom reprezentované ako elementy. Vlastnosti daného objektu špecifikujú atribúty daného elementu. V prípade potreby zmeniť prvky užívateľského rozhrania je možné objekty plne kontrolovať aj programovo.

Každá aplikácia musí obsahovať špeciálny súbor *AndroidManifest.xml*. Súbor musí byť umiestnený v jej koreňovom adresári. Obsahuje dôležité informácie o aplikácií potrebné pre jej beh v systéme. Okrem iných vecí nastavuje unikátne meno aplikácie, opisuje jej časti (aktivity, využívané služby, mená tried atď.), zverejňuje iným aplikáciám akú funkcionalitu im daný program poskytuje a podobne. V súbore sú tiež definované oprávnenia, ktoré musí aplikácia mať aby mohla bežať (napr. oprávnenia ku kamere, k použitiu súborov na pevnom disku...). Špecifikuje aj minimálnu verziu API potrebnú pre jej spustenie a zoznam používaných externých knižníc.

Vývoj aplikácie

Vývojom aplikácií pre operačný systém Android sa zaoberá množstvo nadšencov a profesionálnych programátorov. Na uľahčenie ich práce a je na internete dostupný balík vývojových nástrojov ***Android SDK***[12]. Poskytuje im základný balík API knižníc, nástroje potrebné pre kompiláciu, testovanie a trasovanie kódov aplikácií. Dostupné sú tiež nástroje pre emulovanie zariadení bežiacich na Android OS. Všetky vývojárske nástroje sú dostupné pre platformy Windows, Mac OS X a Linux. Pri použití vývojového prostredia *eclipse*, je tiež k dispozícií podporná sada nástrojov ADT (Android Developer Tools) vo forme pluginu.

Všetky základné aplikácie sú vyvíjane v jazyku Java. Pre niektoré typy aplikácií je výhodné, či potrebné použitie existujúcich kódov v jazyku C/C++. Vďaka sade nástrojov **Android NDK**[13] je možné implementovať časť kódu aplikácie v týchto jazykoch a časť v jazyku Java. Zdrojové súbory naprogramované v C/C++ sú skompilované pomocou rozhrania **JNI** (Java Native Interface) do formy externých knižníc (.so) a následne linkované k danej aplikácii. Použitie NDK prístupu je odporúčané najmä, ak je potrebné implementovať operácie náročné na procesor, ktoré a nealokujú priveľa pamäte. Môže sa jednať napríklad o rôzne spracovania signálov, či fyzikálne simulácie. Vývoj pomocou NDK je pre programátora náročnejší a zvyšuje celkovú komplexnosť výsledného produktu. Prínos v oblasti rýchlosti spracovania nie je pri použití NDK na nevhodnú aplikáciu veľký. Z hľadiska môjho riešenia je však použitie NDK prístupu žiaduce, keďže aplikácia vykonáva množstvo náročných grafických operácií nad obrázkami videa. Pri vývoji si tiež treba uvedomiť, že volania native funkcií (C/C++) z častí písaných v jazyku Java a naopak sú oveľa náročnejšie ako bežné volania metód. Celkovo je potrebné aplikáciu koncipovať tak, aby prechodov Java, C/C++ a späť bolo čo najmenej.

[12] http://developer.android.com/sdk/index.html
[13] http://developer.android.com/tools/sdk/ndk/index.html

3 Návrh riešenia

Zo špecifikácie cieľa projektu som jeho riešenie rozdelil na dve hlavné úlohy. Prvou z nich je **spracovanie obrazu** do formy textu. Druhá predstavuje vhodné **zobrazenie výsledku** užívateľovi tak, aby s nim mohol čo najpríjemnejšie a najinteraktívnejšie pracovať.

V tejto kapitole sa zaoberám popisom rozdelenia oboch problémov na menšie podúlohy. Popíšem tiež svoj návrh a alternatívy ich riešenia. Počas riešenia daných úloh tiež vyvstalo niekoľko ďalších neočakávaných problémov, ktorých riešenie takisto popíšem.

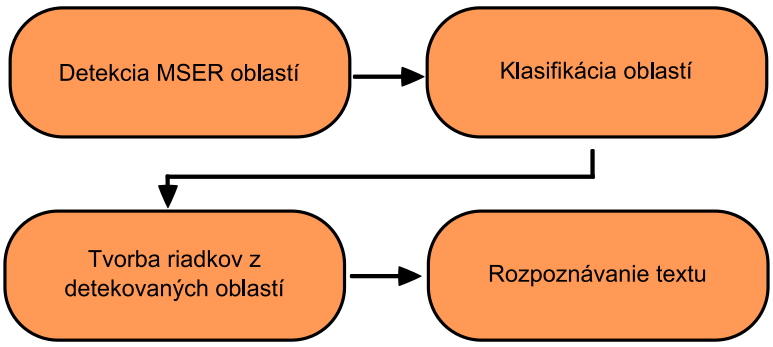
Obsah prvej podkapitoly tvorí rozbor navrhnutej metódy spracovania obrazu na text. Podkapitola je rozdelená na menšie celky podľa jednotlivých krokov popisovaného procesu. V druhej podkapitole uvádzam návrh riešenia interaktívneho zobrazenia výsledkov spracovania obrazu do užívateľského rozhrania aplikácie. Nasleduje podkapitola, ktorá sa venuje problému sledovania pohybov textu v scéne a na obrazovke zariadenia. Podkapitola je delená podľa jednotlivých krokov tohto procesu na menšie celky.

3.1 Spracovanie obrazu na text

Nosnú časť vytvorenej aplikácie tvorí algoritmus spracovania textu z dodaného obrazu videokamery. Jeho cieľom je detekcia faktu, že sa v obraze nachádza textová informácia a jej následná interpretácia v podobe textového reťazca použiteľného na konštrukciu vhodnej internetovej adresy. Z hlavného určenia mobilnej aplikácie vyplýva, že vstupom spracovania bude vo väčšine prípadov obraz z reálnych scén. Rozdiely v extrakcií textov z obrazu naskenovaných tlačených dokumentov a extrakcií textu z obrazu z reálnych scén som popísal v kapitole 2.2. Po dôkladnom preštudovaní existujúcich metód zaoberajúcich sa danou tematikou som sa rozhodol zvoliť ako východiskovú metódu pre moje riešenie metódu L. Neumanna a J. Matasa [11]. Inšpiroval som sa tiež jej rozšírením prezentovaným v [16]. V riešení som z nej využil najmä postupy detekcie, klasifikácie a spájania písmen do slov. Hlavným dôvodom pre zvolenie tejto metódy bol najmä fakt, že sa vyznačuje výbornou robustnosťou voči šumom pozadia a presnosťou lokalizácie a extrakcie textu pri spracovaní obrazov z reálnych scén. V súčasnosti je daná metóda, čo sa týka kvality spracovania obrazov z reálnych scén, jednou z najlepších. Pre použitie v mobilnom zariadení som ju však musel značne modifikovať tak, aby som čo najviac znížil jej nároky na systémové zdroje.

Celý priebeh algoritmu sa skladá zo štyroch hlavných krokov (obrázok č. 6). Kroky sú vykonávané sekvenčne, jeden po druhom. Obmedzil som teda proces návratov k predošlým krokom na korekciu zlých rozhodnutí ako navrhovala pôvodná metóda. Postup tým pádom stratil na svojej presnosti, no ušetril som takto pomerne veľa systémových prostriedkov, či už pamäte potrebnej na uloženie hodnôt v predošlých stavoch, alebo aj samotných zdrojov CPU. Celý proces sa tým pádom

výrazne zrýchlil, čo je potrebné kvôli spracovaniu obrazu z videa. Nevýhoda sekvenčného postupu je najmä to, že presnosť výsledku aktuálneho kroku závisí na presnosti výsledkov z predošlých krokov.



Obrázok 6: Kroky spracovania obrazu na text.

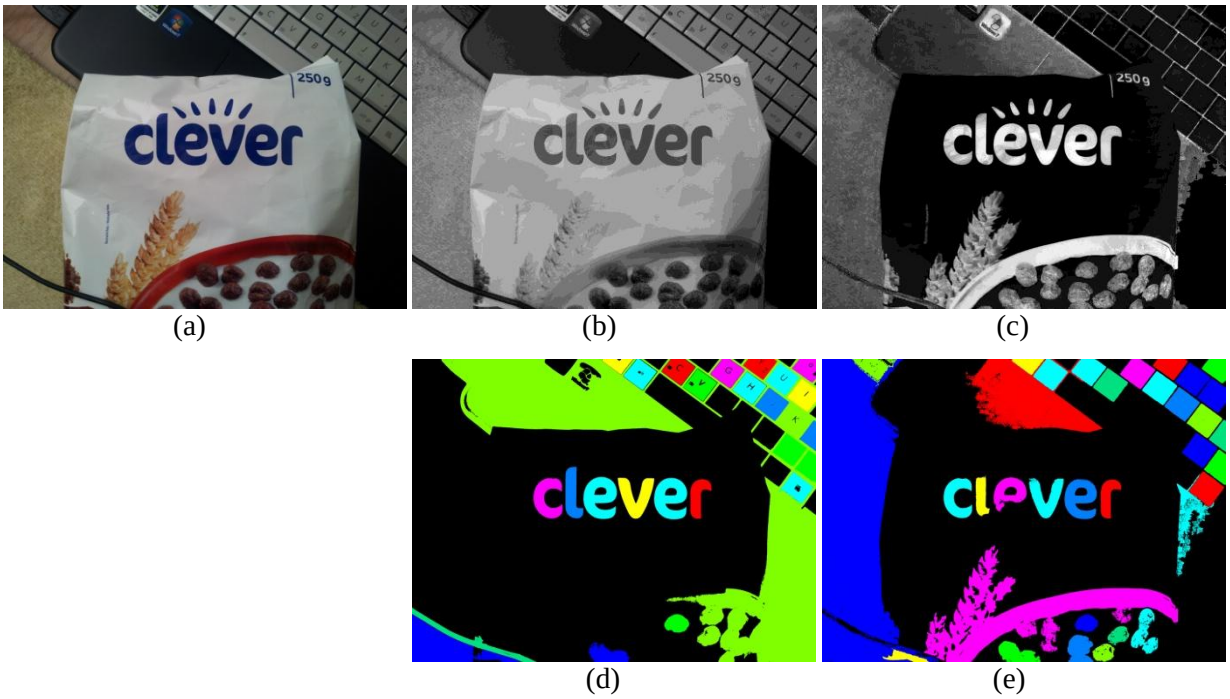
3.1.1 Detekcia textu

Detekcia oblastí obrazu, ktoré potenciálne predstavujú znaky textu je základným krokom spracovania. Metóda je postavená na predpoklade, že znaky textu je možné v obraze vyhľadať vo forme takzvaných *maximálne stabilných extrémnych oblastí* (MSER). Koncept MSER oblastí bol publikovaný J. Matasom v [17]. Detektor MSER je stabilný, invariantný voči afinitnej zmene jasu a súvislým geometrickým transformáciám oblastí. Pôvodne publikovaný algoritmus pracuje v $O(n\ log(log\ (n)))$, kde číslo n predstavuje počet bodov spracovávaného obrazu. Detekcia týmto spôsobom je teda pomerne rýchla. Nedávno sa objavila dokonca implementácia pracujúca v najhoršom prípade v $O(n)$.

Detekcia MSER pracuje primárne nad obrazmi s jedným farebným kanálom. Existujú však aj rozšírenia algoritmu pre spracovanie plne farebných obrázkov, ktoré nahradzujú prahovanie funkcie intenzity zhlukovaním bodov na základe farebných prechodov (gradientov) [18]. Pri implementácii sa však tento variant ukázal ako nevyhovujúci. Spracovanie farebnej fotografie trvalo oproti pôvodnému procesu na obraze s jedným farebným kanálom približne 3,5- krát pomalšie. Okrem toho neprináša výrazné zlepšenie výsledkov detekcie textových znakov oproti spracovaniu pôvodným spôsobom pre každý farebný kanál zvlášť.

V publikácií [11], z ktorej som pri implementácií detekcie znakov čerpal, používajú autori ako vstup pre MSER detektor obraz intenzity a jednotlivé kanály RGB farebného priestoru. V mojom riešení spracovávam oblasti nájdené v kanáloch *saturation* a *value* obrazu reprezentovanom modelom HSV (obrázok č.7). Táto kombinácia kanálov sa pri riešení experimentálne ukázala pre obrazy z reálnych scén ako najvýhodnejšia z hľadiska časovej náročnosti výpočtu (2 detekcie) a kvality detekcie. V kanáli *saturation* sa v priemere našlo približne 70% znakov, v kanále *value* 85% písmen. Kombinovane je algoritmus schopný nájsť v obrazoch v priemere 91% textových znakov. Znak bol považovaný za nájdený, ak ohraničujúci obdĺžnik nájdenej MSER oblasti obsahoval aspoň 90% obsahu ohraničujúceho obdĺžnika reálneho znaku. Nájdená oblasť tiež nesmela obsahovať význačné

fragmenty pozadia, ktoré by zabraňovali jej interpretácií človekom ako textového znaku. Výsledky tiež závisia na charaktere snímok v ktorých vyhľadávame. Pri testovaní som použil verejne dostupné obrázky reálnych scén získané z internetu a tiež vlastné fotografie. Často sa však stáva, že jeden znak je nájdený vo viacerých kanáloch, alebo dokonca v jednom kanále ako viacero MSER oblastí. Tento problém riešim v ďalších krokoch spracovania dodatočnou filtráciou nájdených slov na základe priemernej presnosti rozpoznávania. Z hľadiska presnosti výstupov ďalšieho spracovania je lepšie mať na nazačiatku algoritmu viacero oblastí, ktoré možno ďalej analyzovať, ako ich nemať dostatočný počet.



Obrázok 7: Ukážka detekcie MSER oblastí. (a) Originálny obrázok z kamery. (b) Kanál *saturation* obrázku. (c) Kanál *value* obrázku. (d) Nájdené MSER oblasti v kanáli saturation. (e) Nájdené MSER oblasti v kanáli value.

Dôležitým krokom bolo tiež správne nastaviť parametre vyhľadávania tak, aby bol algoritmus dostatočne citlivý na predpokladaný charakter textu, ktorý budem spracovávať, no aby pri tom vyhľadával čo najmenej oblastí, ktoré nemôžu byť text. Do určitej miery sa mi týmto nastavením podarilo tiež zredukovať počet oblastí reprezentujúcich jeden textový znak na jednu oblasť. Nastavenia som vykonával na základe výsledkov experimentov nad už vyššie spomínanou dátovou sadou. Parameter Δ určujúci minimálny odstup MSER oblasti od okolitých bodov obrazu má hodnotu 10. Parametre určujúce maximálnu resp. minimálnu rozdielnosť medzi rodičom a dieťaťom som nastavil na hodnoty 0,15 resp. 0,5.

Výstupom algoritmu sú množiny bodov obrazu, maska. Jedna oblasť neobsahuje viacero odlišných oddelených komponentov, oblasti sú teda súvislé. Oblasti môžu obsahovať diery. Tento charakter výstupu, nám zaručuje extrakciu potenciálneho textu od jeho pozadia (segmentácia).

3.1.2 Klasifikácia oblastí

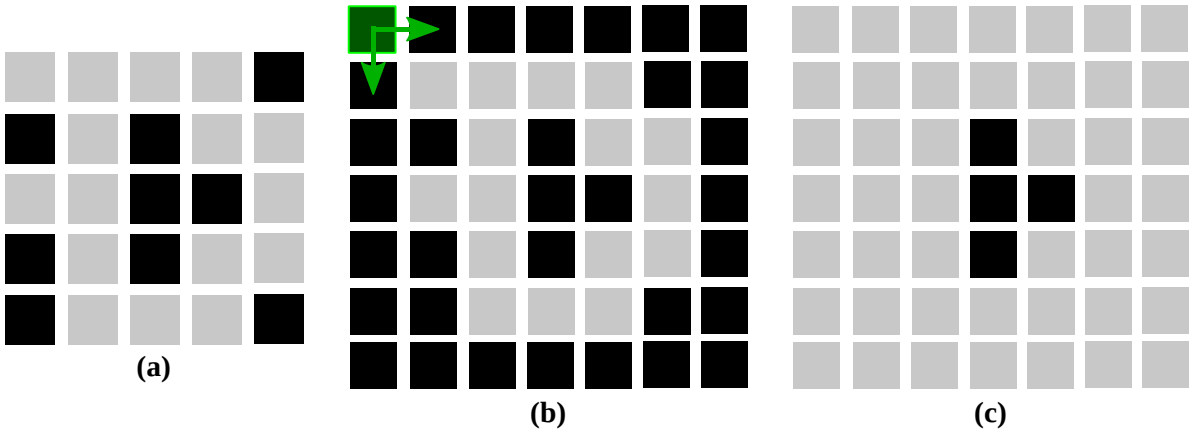
Druhým krokom spracovania nájdených oblastí je ich klasifikácia do dvoch rozdielnych skupín. Výstupom kroku klasifikácie sú dve disjunktné množiny MSER oblastí. Prvá obsahuje oblasti predstavujúce znak textu (písmeno), druhá obsahuje oblasti, ktoré znaky textu nepredstavujú. Zatriedením každej oblasti do jednej z týchto tried vytváram prvotnú aproximáciu pozície textu v spracovávanej snímke.

Každú spracovávanú oblasť zatrieďujem pomocou klasifikátora na základe sady jej špecifických príznakov. Na vypočítanie vektoru príznakov je potrebné pre každú oblasť určiť jej základné vlastnosti:

- plocha oblasti (a)
- výška a šírka oblasti (h, w)
- plocha konvexnej obálky oblasti (a_c)
- plocha dier oblasti (a_h)
- obvod oblasti (o)
- eulerovo číslo (e)

Hodnotu plochy oblasti a predstavuje jednoducho počet obrazových bodov, ktoré oblasť obsahuje. Hodnotu výšky h a šírky w oblasti ľahko odvodíme z rozmerov ohraničujúceho obdĺžnika obopínajúceho danú oblasť. Hodnotu a_c získavam spočítaním bodov, ktoré sú vnútri polygónu predstavujúceho konvexnú obálku spracovávanej oblasti.

V prípade, že spracovávaná MSER oblasť r neobsahuje žiadne diery, je hodnota premennej plochy dier a_h rovná nule. Inak jej priradzujem počet pixlov v dierach danej oblasti. Na začiatku výpočtu tejto hodnoty si vytvorím binárny obraz I_h vo veľkosti $h_r + 2$, $w_r + 2$ naplnený nulami. Do neho prekopírujem obraz spracovávanej MSER oblasti tak, že súradnica každého jej bodu $[x, y]$ je posunutá o jednotku v každom smere $[x + 1, y + 1]$ a jej jednotlivé body majú v I_h hodnotu jedna. Výsledkom je binárny obraz MSER oblasti (hodnota 1) ohraničený prázdnym (hodnota 0) riadkom resp. stĺpcom po každej strane. Následne je potrebné odfiltrovať prázdne okolie MSER oblasti. Okolie filtrujem pomocou algoritmu *floodfill*, ktorý vyplní spojitý komponent obrazu na základe hodnoty jeho jednotlivých bodov. Ako začiatočný bod pre algoritmus používam ľavý horný okraj obrazu I_h, ktorý má určite hodnotu 0. Body okolia vypĺňam na hodnotu 1. Spočítaním zvyšných pixlov s hodnotou 0, ktoré po vyplnení ostali v obraze I_h, získam požadovanú hodnotu a_h. Priebeh algoritmu je znázornený na obrázku č. 8.



Obrázok 8: Zistenie obsahu diery MSER oblasti. Body s hodnotou 0 sú čierne. Body s hodnotou 1 sú šedé. (a) Originálny obraz bodov MSER oblasti. (b) Rozšírený obraz I_h s prekreslenou MSER oblasťou. Zelenou je vstupný bod algoritmu *floodfill*. (c) Obraz po vyplnení okrajov algoritmom *floodfill*.

Obvod o a eulerovo číslo e oblasti počítam pomocou metódy popísanej v [19]. Na začiatku je potrebné opäť vytvoriť binárny ohraničujúci obraz I_e spracovávanej MSER oblasti r s rozmermi $h_r + 2$, $w_r + 2$. Body oblasti r sú reprezentované v I_e hodnotou 1, ostatné hodnotou 0. Po okrajoch spracovávanej MSER oblasti je v I_e prázdny (hodnota 0) riadok resp. stĺpec z každej strany. Ďalej uvažujme nasledovné vzory usporiadania bodov binárneho obrazu

$$Q_1 = \begin{Bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0' & 0 & 0' & 0 & 1' & 0 & 1 \end{Bmatrix}$$

$$Q_2 = \begin{Bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0' & 0 & 1' & 1 & 1' & 1 & 0 \end{Bmatrix}$$

$$Q_3 = \begin{Bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0' & 0 & 1' & 1 & 1' & 1 & 1 \end{Bmatrix}$$

$$Q_D = \begin{Bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1' & 1 & 0 \end{Bmatrix}$$

$$Q_c = \{1 \quad 0, \ 0 \quad 1\}$$

a funkciu $n\{Q_i\}, i \in \{1, 2, 3, D\}$, ktorá určuje počet výskytov jednotlivých vzorov Q v binárnom obraze I_e. Uvažujme tiež funkciu $c(h')$, ktorá určuje počet výskytov vzorov Q_c v riadku h' binárneho obrazu I_e. Eulerovo číslo e oblasti potom vypočítame podľa nasledujúcej rovnice:

$$e = \frac{1}{4}\left[n\{Q_1\} - n\{Q_3\} - 2\, n\{Q_D\}\right]$$

Obvod o oblasti vypočítame podľa nasledovnej rovnice:

$$o = n\{Q_2\} + \frac{1}{\sqrt{2}}\left[n\{Q_1\} + n\{Q_3\} + 2n\{Q_D\}\right]$$

Hodnoty funkcií n a c pre všetky potrebné parametre oblasti r počítam naraz v jednom prechode obrazu I_e.

Z vypočítaných vlastností oblasti následne odvodzujem vektor príznakov. Klasifikátor na základe nameraného vektora príznakov rozhodne do ktorej triedy daná oblasť spadá, či je oblasť v obraze textový znak, alebo nie. Keďže som pre vstup detekcie použil rozdielny formát vstupných

obrazov ako autori pôvodnej metódy, musel som podľa toho upraviť typ a počet zložiek vektora príznakov. V mojom riešení nepoužívam príznaky využívajúce farebnú informáciu oblastí a ich pozadia. Snažil som sa tak ušetriť systémové zdroje potrebné na výpočet týchto komplexných príznakov. Použité zložky vektora príznakov v mojom riešení prezentuje tabuľka č.2. Všetky použité príznaky sú invariantné voči veľkosti nájdených oblastí, vďaka čomu je možné takto vyhľadávať textové znaky rôznych veľkostí. Vybrané príznaky nie sú invariantné k rotácií znakov, čo som sa snažil kompenzovať pri trénovaní klasifikátora zahrnutím vzoriek pootočených písmen.

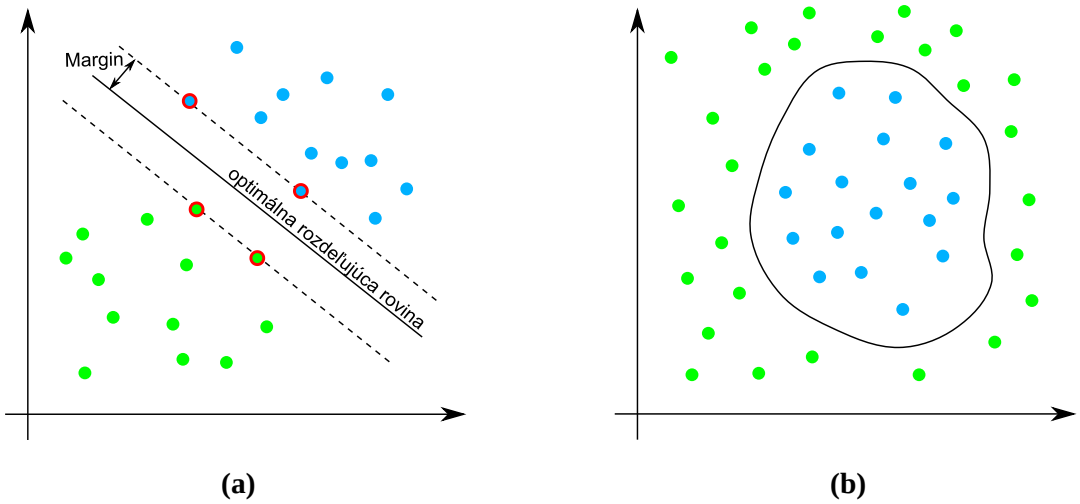
Pomer veľkosti	$\frac{w}{h}$
Počet dier oblasti	$1 - e$
Kompaktnosť	$\frac{o^2}{a}$
Pomer plochy dier k ploche oblasti	$\frac{a_h}{a}$
Pomer konvexnej obálky k ploche oblasti	$\frac{a_c}{a}$
Relatívna výška oblasti	$\frac{a}{h^2}$
Počet horizontálnych prechodov v1/6 výšky	$c(\frac{1}{6}h)$
Počet horizontálnych prechodov v 3/6 výšky	$c(\frac{3}{6}h)$
Počet horizontálnych prechodov v 5/6 výšky	$c(\frac{5}{6}h)$

Tabuľka 2: Príznaky použité pri klasifikácií oblastí.

Podobne ako autori pôvodnej metódy, aj ja som sa na klasifikáciu dát rozhodol použiť klasifikátor typu *Support Vector Machine* (SVM). Klasifikátor je potrebné vopred natrénovať na riešenie danej úlohy pomocou vhodného vzorku trénovacích dát.

SVM je algoritmus strojového učenia hľadajúci nadrovinu, ktorá v priestore príznakov optimálne rozdeľuje trénovacie dáta. Ide teda o hľadanie maximalizácie vzdialenosti rozdeľujúcej nadroviny klasifikátora k bodom z trénovacej množiny. Oddeľujúca nadrovina (oddeľovač) je definovaná sadou takzvaných podporných vektorov (support vectors) . Sú to dátové body, ktoré sú k oddeľujúcej nadrovine najbližšie (obrázok č. 9), teda tie trénovacie príklady, ktoré sú pre oddeľovače tried daného problému podstatné. Ostatné dátové body (vektory) nie sú pre oddeľovač potrebné. Obvyklé algoritmy strojového učenia používajú všetky trénovacie príklady, čo pri ich vysokom počte môže viesť k ich nízkej efektivite. Samotná klasifikácia pomocou SVM je preto oproti iným pomerne rýchla a nenáročná na výpočetný výkon.

Pri trénovaní klasifikátora sa často stáva, že dodané dáta na trénovanie obsahujú rôzne chyby. Môže sa preto stať, že rozdeľujúca nadrovina pre takéto dáta neexistuje, dáta nie sú v pôvodnom priestore príznakov od seba lineárne oddeliteľné. Tento problém je možné riešiť použitím *soft–margin* SVM. V tomto prípade nájde klasifikátor rozdeľujúcu nadrovinu aj v prípade, že niektoré dáta nebudú správne klasifikované. Ďalšie zakomponované rozšírenie klasifikátora predstavuje použitie inej jadrovej funkcie (*kernel function*). Vďaka nej je možné transformovať priestor príznakov do vyššej dimenzie bez toho, aby boli transformované vlastné vstupné príznaky a tým separovať aj inak lineárne neseparibilné trénovacie dáta. V mojom riešení je použitá *Gaussova radiálna jadrová funkcia* (RBF). Tá je jednou z najpoužívanejších SVM jadrových funkcií a aj pri testovaní vykazovala najlepšie výsledky pre daný typ dát.



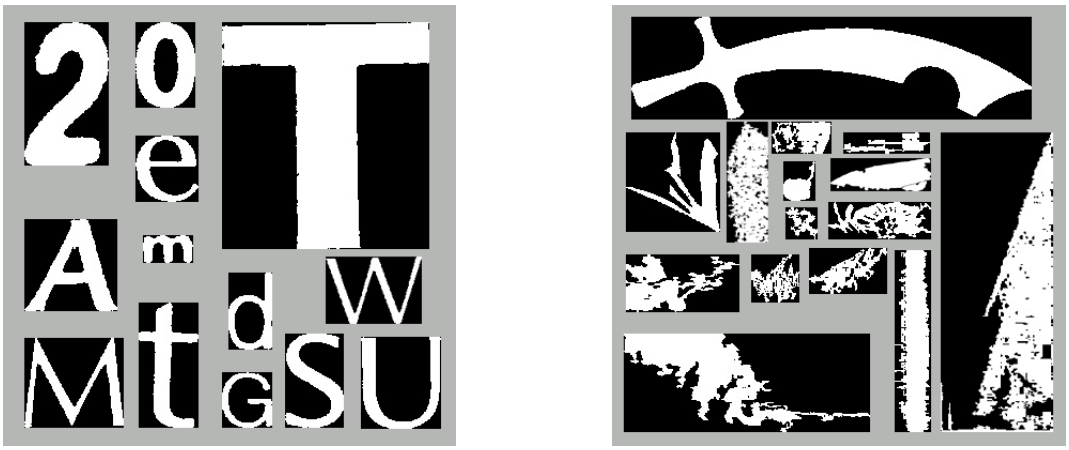
Obrázok 9: Support Vector Machine. (a) Lineárne separabilné dáta. *Support vector* body sú označené červenou. (b) Lineárne neoddeliteľné dáta riešené pomocou soft margin SVM s jadrom RBF.

Na trénovanie a aj následné testovanie natrénovaných modelov SVM klasifikátora som využil sadu fotografií zo súťaže **ICDAR 2003** Robust Reading Competition[14]. Je to v súčasnosti najviac používaná dátová sada pre porovnávanie kvality algoritmov detekcie a rozpoznávania textu v obraze. Dáta sú rozdelené na dve skupiny, trénovacie a testovacie. Trénovacia časť obsahuje 258 a testovacia 251 rôznych obrázkov nápisov, väčšinou v anglickom jazyku. Fotografie na trénovanie obsahujú spolu 6185 písmen latinskej abecedy v 1157 slovách. Testovacie fotografie spolu obsahujú 1111 slov skladajúcich sa dokopy z 5430 písmen. V oboch častiach sa nachádzajú aj arabské číslice.

Z fotografií trénovacej sady som extrahoval pomocou zvoleného algoritmu popísaného v predošlej kapitole všetky MSER oblasti. Citlivosť detekcie som pri generovaní dát o niečo zväčšil, aby som získal čo najlepšiu začiatočnú množinu oblastí. Túto množinu som následne ručne roztriedil do požadovaných tried. Keďže detekcia prebehla na dvoch farebných kanáloch zdrojových dát, výsledná množina MSER oblastí je pomerne veľká. Väčšina písmen bola nájdená v oboch kanáloch,

[14] http://algoval.essex.ac.uk/icdar/Datasets.html

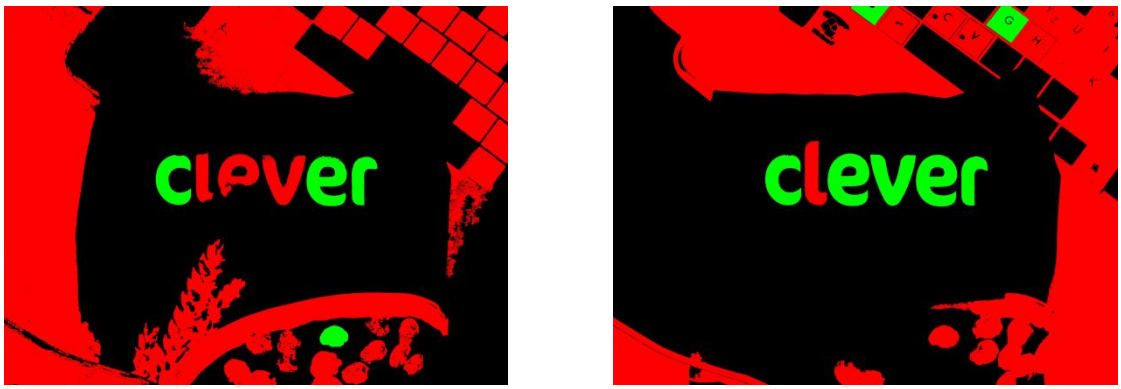
niektoré aj viackrát. Celkovo obsahuje vytriedená trénovacia sada 13 592 MSER oblastí predstavujúcich textové znaky a 53 870 oblastí, ktoré nie sú text. Testovacia sada obsahuje celkovo 10 081 písmen a 64 925 oblastí nepredstavujúcich písmena. Ukážku vytriedených trénovacích dát zobrazuje obrázok č. 10.



(a) **(b)**

Obrázok 10: Ukážka dát použitých na trénovanie modelu klasifikátora. (a) Oblasti reprezentujúce písmená. (b) Oblasti, ktoré nereprezentujú písmená.

Po ručnom vytriedení oblastí som klasifikátor trénoval na celej trénovacej sade. Skúšal som trénovať aj na menšej podmnožine trénovacích dát. Po každej trénovacej iterácií som výsledný model testoval na vytriedenej testovacej sade. Najlepší vytrénovaný model dosahoval presnosť okolo 80%. Ukážku výsledkov klasifikácie oblastí z jednotlivých kanálov obrázku č. 7 (a) prezentuje obrázok č.11. Problém pomerne nízkej celkovej úspešnosti klasifikácie môže tkvieť najmä v povahe príznakov (vypustenie farebnej informácie oblastí) a povahe trénovacích a testovacích dát. V globálnom hľadisku však pomerne nízka úspešnosť klasifikácie z pohľadu celkovej úspešnosti algoritmu až tak nevadí, keďže v tomto kroku sa vytvárajú len prvotné hypotézy o pozícii textu. V ďalšom kroku sú tieto hypotézy korigované a výsledky klasifikácie spresňované.



(a) **(b)**

Obrázok 11: Výsledky klasifikácie MSER oblastí. Oblasti klasifikované ako písmená sú zelené, nepísmená sú červené. (a) Klasifikácia oblastí z kanálu *value*. (b) Klasifikácia oblastí z kanálu *saturation*.

3.1.3 Skladanie slov a riadkov textu

V tomto kroku spracovania nájdené oblasti spájam do väčších celkov, do slov a riadkov. S ohľadom na primárny účel vytváranej aplikácie a efektívnosti jej spracovania som formuloval niekoľko základných predpokladov o forme spracovávaného textu.

Prvým predpokladom je, že spracovávame texty písané latinkou, či iným jazykom čítaným horizontálne. Jednotlivé znaky textu sú teda v spracovávanom obraze umiestnené v rovných, či mierne zakrivených riadkoch. Spodným okrajom jednotlivých písmen je teda možné viesť myslenú priamku, na ktorej tieto znaky budú ležať. Výnimku tvoria znaky v bežnom texte umiestnené pod mysleným riadkom, napríklad znaky q, p, j, y. V praxi však väčšina slov obsahuje viac znakov umiestnených na riadku, je teda možné priamku mysleného riadku s dostatočnou presnosťou aproximovať. Texty v reálnych scénach bývajú tiež často umiestnené na krivkách. Pre jednoduchosť implementácie som však s takýmito textami zatiaľ nepočítal. Vytvorený prototyp aplikácie je však možné ľahko na spracovanie takýchto textov prispôsobiť zavedením komplexnejšej funkcie na aproximácie riadku. V momentálnej implementácií predpokladám, že text je na spracovávanej snímke umiestnený približne horizontálne s toleranciou ±10° od línie rovnobežnej s jej spodným okrajom.

V jednom slove spracovávaného textu by mali mať všetky jeho písmená rovnaký font. Písmená sa vyskytujú v slove v dvoch rozličných výškach (napr. ako v slove „Začiatočné“), alebo majú všetky v slove rovnakú výšku (všetky písmená sú malé, alebo sú všetky veľké). Počítam tiež s faktom, že medzi jednotlivými písmenami sú medzery s približne rovnakou veľkosťou.

Prototyp aplikácie tiež predpokladá, že jedno slovo sa skladá aspoň z troch textových znakov. Toto obmedzenie je takisto možné v nasledujúcich verziách algoritmu zmeniť, alebo úplne zrušiť.

Výstupom z predošlého kroku spracovania je množina MSER oblastí klasifikovaných do dvoch tried. Klasifikáciou oblasti do triedy predstavujúcej písmena je teda učinená prvotná aproximácia pozície textu v obraze.

Pre každú MSER oblasť klasifikovanú ako písmeno vytváram na začiatku spracovania prvotnú hypotézu o riadku textu. Pojem hypotéza riadku znamená skupinu MSER oblastí, ktoré spolu reprezentujú v obraze nejaký potenciálny text. Na začiatku teda každá prvotná hypotéza obsahuje jednu jedinečnú MSER oblasť, v predošlom kroku klasifikovanú ako písmeno. V ďalších krokoch spracovania sa postupne pokúšam každú z týchto hypotéz rozširovať o ďalšie MSER oblasti. Rozširovanie hypotéz o ďalšie oblasti prebieha za pomoci heuristickej funkcie určujúcej, či možno danú oblasť do hypotézy pripojiť, alebo nie.

Vyberiem teda z množiny prvú hypotézu a postupne sa do nej snažím pripojiť ďalšie MSER oblasti. To opakujem, pokým sa ku hypotéze ešte nejaká nová oblasť dá pripojiť. V prípade, že už taká oblasť neexistuje, označím túto hypotézu ako uzavretú a podobne pokračujem ďalšou, ešte neuzavretou hypotézou. Postup opakujem kým už neexistuje žiadna neuzavretá hypotéza.

Pri zvažovaní pripájania ďalších oblastí k hypotézam už nijak nezohľadňujem výsledok ich predošlej klasifikácie (okrem výnimky popísanej v odseku nižšie). Niekedy teda pripájam k hypotézam aj oblasti, ktoré vplyvom chyby neboli klasifikované ako písmena a naopak, môže sa stať, že nepripojím oblasť chybne klasifikovanú ako písmeno. Metóda sa teda takto pomocou heuristickej funkcie snaží opraviť chyby kroku klasifikácie.

Niekedy sa môže stať, že do hypotézy môže byť pripojených viac, ako len jedna MSER oblasť. Autori pôvodnej metódy v takomto prípade vytvoria pre všetky varianty spojenia nové neuzatvorené hypotézy. Experimenty však ukázali, že daný postup je pre riešenie na testovacom mobilnom zariadení príliš náročný na systémové zdroje. Rozhodol som sa pre to v tomto prípade už množinu hypotéz nerozširovať, ale určiť ktorú z daných MSER oblastí k hypotéze prednostne pripojiť. Ak sa vyskytne viacero pripojiteľných oblastí, prednostne pripájam prvú nájdenú oblasť klasifikovanú ako písmeno. V prípade, že z pripojiteľných oblastí nie je ani jedna písmeno, pripojím prvú podľa poradia ich detekcie. Po pripojení takto vybranej oblasti sa v drvivej väčšine prípadov stáva, že ostatné pripojiteľné MSER oblasti sú pripojené k hypotéze v ďalších iteráciách.

Pre vyhodnotenie heuristiky je potrebné pre každú hypotézu t sledovať tieto vlastnosti:

- šírka spojených znakov hypotézy (w_t)
- priemerná šírka znakov hypotézy $(\overline{w_t})$
- priemerná výška znakov hypotézy $(\overline{h_t})$
- priemerná plocha znakov hypotézy $(\overline{a_t})$
- maximálna šírka znakov hypotézy $(\widehat{w_t})$
- maximálna výška znakov hypotézy $(\widehat{h_t})$

Okrem nich je ešte potrebné vypočítavať odhad maximálnej horizontálnej vzdialenosti pre ďalšiu oblasť (d_t). Túto hodnotu získam pomocou rovnice

$$d_t = \left[\left(\frac{c_t}{2} + \frac{1}{2}\right) \cdot \widehat{w_t}\right] + \left(\frac{c_t}{2} \cdot \overline{w_t} \cdot \frac{2}{10}\right)$$

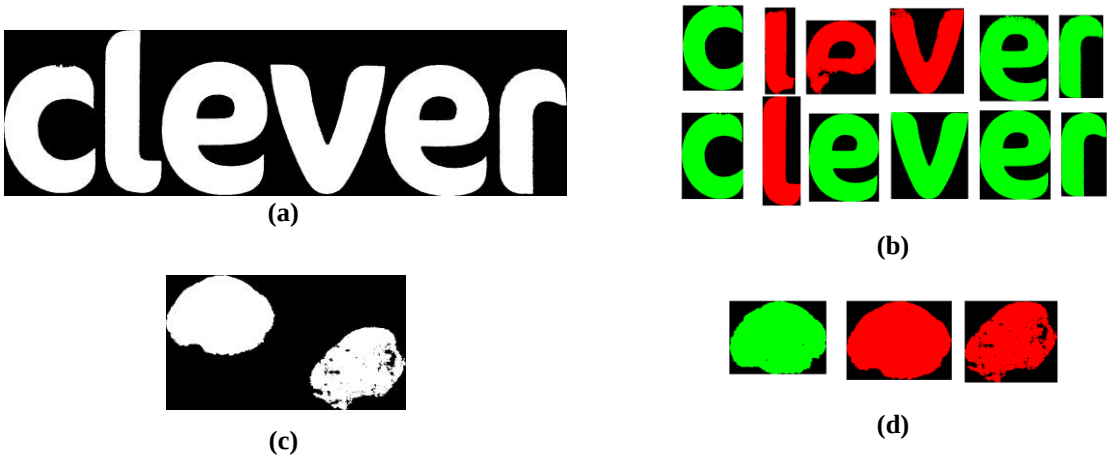
Premenná c_t vyjadruje počet doteraz zahrnutých MSER oblasti v hypotéze t. Vždy po pridaní novej MSER oblasti do hypotézy je potrebné spomínané vlastnosti aktualizovať.

Heuristická funkcia pripájania novej oblasti r k hypotéze t obsahuje podmienky uvedené v tabuľke č. 3. Pri výpočte podmienky pre vertikálnu vzdialenosť oblasti od hypotézy je použitá aproximácia vertikálnej súradnice riadku hypotézy $l_t(b)$, po pridaní oblasti s jej spodným okrajom na y-súradnici b. V pôvodnej metóde autori na tento výpočet používajú robustnú metódu *Least Median of Squares* a neskôr aproximáciu pomocou paraboly. V rámci zjednodušenia implementácie a zníženia nárokov algoritmu na systémove zdroje počítam v mojej aplikácií len priemer spodných okrajov oblastí do hypotézy zahrnutých. Riadok teda aproximujem priamkou rovnobežnou so spodným okrajom snímky. Symboly x a y označujú súradnice ľavého horného rohu ohraničujúceho obdĺžnika.

Šírka spracovávanej oblasti	$\frac{1}{10} < \frac{\overline{w_t}}{w_r} < 5$
Výška spracovávanej oblasti	$\frac{1}{2} < \frac{\overline{h_t}}{h_r} < 2$
Plocha spracovávanej oblasti	$\frac{1}{8} < \frac{\overline{a_t}}{a_r} < 2$
Horizontálna vzdialenosť oblasti od hypotézy	$\left\| \left(x_r + \frac{w_r}{2} \right) - \left(x_t + \frac{w_t}{2} \right) \right\| < d_t$
Vertikálna vzdialenosť oblasti od hypotézy	$\| l_t(y_r + h_r) - (y_r + h_r) \| < \frac{1}{2}\widehat{h_t}$
Nachádza sa znak na priamke riadku?	$l_t(y_r + h_r) > y_r$

Tabuľka 3: Podmienky heuristickej funkcie.

Oproti pôvodnému algoritmu sa moja metóda tiež líši podmienkou pre horizontálnu vzdialenosť oblasti od hypotézy. Experimenty ukázali, že moja podmienka je pri spájaní oblastí presnejšia a navyše som sa vďaka nej vyhol radeniu oblastí pred ich zhlukovaním. Hypotézu obohatím o novú oblasť len ak sú všetky podmienky spomínané v tabuľke č. 3 splnené.



(a) **(b)** **(c)** **(d)**

Obrázok 12: Poskladané hypotézy slov. (a) Spojený obrázok oblastí validnej hypotézy. (b) Jednotlivé oblasti tvoriace validnú hypotézu. Zelené sú klasifikované ako písmena, červené ako nepísmená. (c) Spojený obrázok oblastí vylúčenej nevalidnej hypotézy. (d) Oblasti tvoriace vylúčenú hypotézu.

Výstupom zhlukovania je množina hypotéz obsahujúcich spojené MSER oblasti. Výsledné hypotézy sú následne podrobované validácií. Do ďalšieho spracovania postupujú len validné hypotézy, ostatné sú vylúčené. Cieľom validácie je odstránenie hypotéz, ktoré nemajú želané vlastnosti riadkov textu. Množinu MSER oblastí r, ktoré obsahuje hypotéza t označíme symbolom $\mathcal{R}_t$. Množinu MSER oblastí klasifikovaných ako potenciálne znaky označme symbolom $\mathcal{C}$. Symbol $\bar{\mathcal{C}}$ označuje množinu oblastí klasifikovaných ako nepísmená. Symbol l'_t označuje hodnotu vertikálnej súradnice odhadu línie riadku hypotézy. Aby hypotéza prešla krokom validácie, musí splniť všetky podmienky uvedené v tabuľke č. 4. Ukážku validnej a vylúčenej hypotézy prezentuje obrázok č. 12.

V hypotéze je viac písmen ako nepísmen	$\lvert \{r \in \mathcal{R}_t : r \in \mathcal{C}\} \rvert > \lvert \{r \in \mathcal{R}_t : r \in \bar{\mathcal{C}}\} \rvert$
Hypotéza obsahuje požadovaný počet oblastí	$\lvert \mathcal{R}_t \rvert > 3$
Zaplnenie oblastí hypotézy	$\underset{r \in \mathcal{R}_t}{\text{median}} \left(\frac{a_r}{w_r \cdot h_r} \right) < \frac{9}{10}$
Väčšina znakov hypotézy sa nachádza na riadku	$\frac{\left\lvert \left\{ r \in \mathcal{R}_t : \lvert l'_t - y_r \rvert < \frac{3}{10} h_r \right\} \right\rvert}{\lvert \mathcal{R}_t \rvert} \geq \frac{3}{4}$

Tabuľka 4: Podmienky validácie hypotéz.

Keďže pri zhlukovaní označujem za uzavreté len jednotlivé hypotézy riadkov, pri spracovaní žiadnym spôsobom neoznačujem, ktoré MSER oblasti už boli do nejakej z hypotéz zahrnuté. Často preto vzniká viacero validných hypotéz popisujúcich ten istý riadok reálneho textu. V prípade, že sú hypotézy tvorené rovnakými množinami MSER oblastí, beriem ďalej do spracovania len jednu z hypotéz. V prípade, že dve hypotézy popisujú rovnaký riadok textu pomocou iných sád MSER oblastí (napríklad oblasťami z rôznych farebných kanálov), beriem do ďalšieho spracovania obe. Pomocou porovnávania v ďalšom kroku považujem za konečný výsledok slova tú, ktorej rozpoznanie sa vyznačovalo lepšou presnosťou. Týmto spôsobom sa snažím zvýšiť potenciál kvality celkového výstupu spracovania textu.

3.1.4 Rozpoznávanie textu

Autori pôvodnej metódy sa pokúšajú ešte pred krokom rozpoznávania odstrániť prípadne perspektívne skreslenie nájdených riadkov textu. Zlepšujú takto presnosť výstupu kroku rozpoznávania. Autori hľadajú zložky transformačnej matice homografie pomocou postupného otáčania a skosenia textu v daných intervaloch. Obrazy získane rotáciami a skoseniami sú následne analyzované a najlepšie z nich predstavujú normalizované textové hypotézy. Navrhnutá metóda sa však ukázala pre moje riešenie príliš náročná na výpočetný výkon. Pre ušetrenie systémových zdrojov som pri implementácií svojho riešenia tento krok vynechal.

Spracovanie pokračuje krokom rozpoznávania textu. Autori vyvinuli vlastný systém rozpoznávania znakov textu fungujúci na základe postupnej klasifikácie detekovaných oblastí do jednej z 62 tried (10 číslic, 26 veľkých a 26 malých písmen). Ako klasifikátor použili opäť zovšeobecnený algoritmus SVM s vektorom príznakov skladajúcim sa z 233 príznakov. Výsledky klasifikácie ďalej spracovávajú analýzou pomocou typografického modelu textu riadka. Všetky hypotézy taktiež podrobujú analýze na základe jazykového modelu daného jazyka, založenom na modeli Markovovho reťazca 2. radu.

Pri svojej implementácií som sa však rozhodol použiť na rozpoznávanie lokalizovaných textových častí externý OCR systém *Tesseract*. Ušetril som si tak mnoho práce s ďalším trénovaním a ladením modelov. Tento systém je vyvíjaný už dlhšiu dobu pomerne veľkou komunitou nadšencov

a tiež niektorými komerčnými spoločnosťami, čo zaručuje jeho dobrú optimalizáciu a požiadavky na presnosť riešenia daného problému. Viac je o systéme *Tesseract* napísané v kapitole 4.1.

Z predošlého kroku máme teda množinu hypotéz, to znamená zhluky jednotlivých MSER oblastí predstavujúce slová a riadky textu ako obraz. Každú hypotézu treba najskôr previesť na jeden, jej prislúchajúci, spojený obraz. Tento obraz riadku bude slúžiť ako vstup pre modul rozpoznávania. Na základe maximálnych a minimálnych hodnôt súradníc ohraničujúcich obdĺžnikov zahrnutých MSER oblastí v originálnej snímke, dokážeme vypočítať rozmer ohraničujúceho obdĺžnika celej hypotézy. Pre každú hypotézu vytvoríme binárny obraz naplnený nulami s rozmermi jej ohraničujúceho obdĺžnika. Do tohto obrazu pre každú hypotézu prekreslím všetky jej prislúchajúce oblasti. Vstupom pre rozpoznanie hypotézy je teda jednokanálový binárny obraz. Ukážku obrazov a výstupu ich rozpoznania je možné vidieť v tabuľke č. 5.

Experimenty ukázali, že použitím obrazu, v ktorom bola zahrnutá aj farebná informácia oblastí a ich okolia kvalitu výstupu rozpoznávania výrazne nezlepšilo. Práve naopak, keďže je použitý OCR rozpoznávací modul navrhnutý prevažne na spracovanie skenovaných dokumentov, kvalitu jeho výstupu komplexné farby na pozadí často znížili. Zlepšenie výstupu rozpoznávania sa mi tiež podarilo rozšírením vstupného binárneho obrazu riadku o prázdny okraj na všetkých jeho stranách. Okraj nemusí byť veľký, stačí, aby sa textové znaky nedotýkali okraja spracovávaného obrázka riadku.

Pri svojom postupe museli autori tiež riešiť problém, keď sa dva rôzne textové znaky zobrazia do jednej MSER oblasti. Problém vyriešili hľadaním miesta rozrezania oblastí, ktoré sú v kontexte riadku o polovicu širšie ako ostatné oblasti. Miesto oddelenia hľadajú pomocou vyhľadávania lokálnych extrémov funkcie vertikálnej vzdialenosti najvyššieho a najnižšieho bodu každého stĺpca danej oblasti. Keďže v mojom postupe predstavuje vstup modulu rozpoznávania obraz celého slova, tomuto problému som sa vyhol.

Po rozpoznávaní je potrebné hypotézy opäť filtrovať tak, aby sa užívateľovi nezobrazovali výsledky, ktoré pre neho nie sú z hľadiska použitia aplikácie relevantné. Najskôr sa vypúšťajú hypotézy, ktorých rozpoznaný text je kratší ako tri znaky. Systém *Tesseract* pri každom rozpoznanom slove vráti tiež priemernú hodnotu presnosti tohto rozpoznávania. Na základe tejto hodnoty filtrujem všetky hypotézy, ktorých presnosť rozpoznania je menšia ako 60%.

V sade zvyšných hypotéz ďalej hľadám tie, ktoré predstavujú v obraze ten istý rozpoznaný text. Tento fakt sa dá zistiť tým, že prienik obsahov ich ohraničujúcich obdĺžnikov vypĺňa aspoň 90% obsahu oboch hypotéz. V prípade, že takéto dve hypotézy nájdem, tak ako výsledok ponechám tú, ktorej rozpoznanie dosiahlo väčšiu priemernú presnosť. Podmienky filtrovania je možné v ďalších verziách ešte pozmeniť a zlepšovať.

Vstupný obraz hypotézy	Výstupný rozpoznaný text. (presnosť vrátená knižnicou *Tesseract*)
	`clever` (86%)
	`SALE` (82%)
	`centre` (79%)
	`w f iffy` (42 %)

Tabuľka 5: Rozpoznanie textu z obrázkov hypotéz.

Týmto sa proces spracovania textu z obrazu končí. Každá hypotéza o riadku má priradený svoj odpovedajúci text, pozíciu v spracovávanej snímke a svoju veľkosť vo forme ohraničujúceho polygónu.

3.2 Zobrazenie a interakcia s výsledkom

Druhou hlavnou úlohou pri implementácií projektu predstavuje vhodné **zobrazenie výsledku** spracovania textu užívateľovi tak, aby s nim mohol čo najpríjemnejšie a najinteraktívnejšie pracovať. Poslaním vyvíjanej aplikácie je priblížiť užívateľom informácie z internetu o ich okolitom prostredí. Nejde preto len výhradne o návrh užívateľského rozhrania aplikácie. Riešenie daného problému spočíva v návrhu vhodného komunikačného rozhrania medzi užívateľom, ovládacími prvkami aplikácie a samotným prostredím okolitého sveta. Pri návrhu tohto rozhrania som vychádzal aj zo skúseností získaných počas testovania skúmaných aplikácií zaoberajúcich sa rovnakou, alebo podobnou tematikou popísaných v kapitole 2.1.

Základné vlastnosti, ktoré by malo dané komunikačné rozhranie mať sú:

- **informatívnosť** – Jednoduchosť pochopenia celého významu prvkov, ktorými sú zobrazené informácie o daných objektoch záujmu.

- **jednoduchosť** ovládania – Aj keď aplikácia bude poskytovať široké spektrum možností, jej ovládanie by malo byť čo najjednoduchšie, s použitím čo najmenšieho počtu ovládacích prvkov rozhrania.
- **intuitívnosť** – Užívateľ by mal byť schopný riešiť komplexnejšie scenáre použitia aplikácie bez nutnosti meniť zaužívané spôsoby ovládania a pochopenia informácií.
- **interaktívnosť** – V prípade viacerých očakávaných výsledkov, môže aplikácia ponúknuť užívateľovi návrhy na riešenie danej úlohy, alebo jej podúloh.
- **atraktívnosť** – Ovládacie prvky aplikácie musia mať pre užívateľa atraktívny dizajn a funkcionalitu.

Najlepším a najpoužívanejším nosičom informácie medzi užívateľom a jeho okolitým prostredím je zrak, resp. obraz. Ideálnym premostením prostredia s užívateľom je teda obraz z kamery mobilného zariadenia. Základným prvkom užívateľského rozhrania aplikácie by teda mal byť náhľad obrazového vstupu aplikácie. Pomocou neho si bude môcť užívateľ pohodlnejšie vybrať danú oblasť záujmu prostredia tým, že priamo pred sebou uvidí vstup celého algoritmu extrakcie textu. Tento spôsob komunikácie s prostredím je medzi užívateľmi dobre známy z mnohých ďalších aplikácií využívajúcich obraz z kamery. Hlavným prvkom celého užívateľského rozhrania aplikácie je teda prvok zaberajúci celú plochu obrazovky, zobrazujúci obraz z kamery v reálnom čase.

Text extrahovaný zo vstupného obrazu je potrebné užívateľovi prezentovať v podobe výrazov do internetového vyhľadávania. Výrazy pre vyhľadávanie sa môžu skladať z viacerých rôznych častí obrazu, prípadne môžu pochádzať z úplne rozdielnych častí scény. Nemôžeme teda text interpretovať vo forme URL adresy vyhľadávania ihneď po jeho extrakcii z obrazu. Je potrebné ponechať užívateľovi schopnosť si daný výraz poskladať podľa jeho želania. Užívateľ by mal mať možnosť s týmito výrazmi pohodlne pracovať, v prípade potreby ich upravovať či meniť, ešte pred ich interpretáciou ako URL. Užívateľské rozhranie aplikácie pre to obsahuje štandardný text box, prvok na úpravu textu. V tomto poli bude užívateľ konštruovať želaný výraz do vyhľadávania.

Vyhľadávaný výraz vo väčšine prípadov pozostáva z jedného, alebo viacerých slov oddelených medzerou. Na zlepšenie praktickosti som do užívateľského rozhrania zakomponoval tlačidlo na rýchle mazanie textu v editačnom poli. Krátke stlačenie tlačidla vymaže z konca jedno celé slovo. Dlhé stlačenie tohto tlačidla vymaže z editačného poľa celý textový reťazec.

Ak ma užívateľ pripravený celý výraz, indikuje to stlačením tlačidla vyhľadávania. Po zvolení tlačidla bude text v editovacom poli prevedený do formy internetovej adresy a spustená externá aplikácia, internetový prehliadač s danou stránkou. Použitím externej aplikácie zaručím, že užívateľ bude vedieť s výsledkami vyhľadávania ďalej pracovať podľa svojich potrieb. Ak je na zariadení nainštalovaných viac aplikácií internetového prehliadača, dostane užívateľ možnosť vybrať ktorý z nich pri presmerovaní použiť. V ďalších verziách aplikácie je možné daný výsledok interpretovať rôznym iným spôsobom, napríklad vyhľadávaním v rozmanitých databázach výrobkov, alebo v iných internetových službách.

Tlačidlá užívateľského rozhrania sú označené obrazovými ikonami. Použité ikony sú štandardne používané v operačnom systéme mobilného zariadenia na tlačidlá s obdobnou funkciou. Užívateľ by nemal mať problém vďaka ich grafickej reprezentácií rýchlo priradiť jednotlivým tlačidlám ich funkcionalitu. Pri návrhu rozhrania som tiež zohľadnil takzvaný princíp „*hero button*“, teda grafické zvýraznenie dôležitých prvkov na úkor ostatných, menej dôležitých podporných prvkov.

Najdôležitejšou časťou užívateľského rozhrania aplikácie je spôsob, akým užívateľ vyberie z množstva nájdených textových reťazcov tie, ktoré ho ďalej zaujímajú. Jednou z možností je zobraziť užívateľovi jednotlivé výsledky vo forme zoznamu tak, ako to robí napríklad aplikácia Google Goggles. V zozname sa tiež stráca informácia z ktorého miesta obrazu bol daný text rozpoznaný. Daná aplikácia sa to snaží riešiť náhľadom výrezov danej oblasti v samotnom zozname výsledkov. Okrem toho približnú pozíciu jednotlivých výrezov znázorni aj v obraze. Užívateľ však nemá možnosť jednotlivé výsledky zo zoznamu kombinovať a spájať.

Pri riešení svojej aplikácie som sa rozhodol interaktívne zobrazovať výsledky priamo do vstupného obrazu. Jednotlivé texty sú v obraze zvýraznené pomocou farebných polygónov. Užívateľ si zvolí nájdené texty, ktoré ho zaujímajú, jednoduchým kliknutím na ním prislúchajúcu zvýraznenú časť obrazu. Snažím sa tak zvýšiť mieru interaktivity užívateľa s jeho okolitým prostredím. Zvýraznenie výsledkov detekcie vo videu z kamery v reálnom čase predstavuje tým pádom určitú formu rozšírenej reality (*augumented reality*). Užívateľom označené textové reťazce sa automaticky vkladajú do editačného poľa. Jednotlivé slová sú v poli vkladané na koniec už existujúceho reťazca a sú od seba oddelené medzerou. Komponent editačného poľa sa teda pre užívateľa stáva akýmsi zásobníkom slov. Môže teda pomocou neho ľahko kombinovať výsledky získané v rôznych častiach scény a prostredia. Prvky užívateľského rozhrania vytvoreného prototypu aplikácie prezentuje obrázok číslo 13.

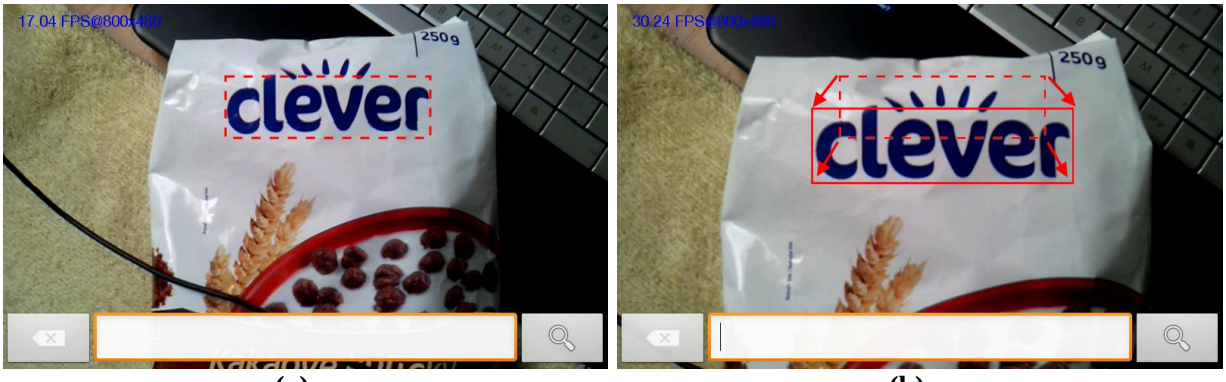
Obrázok 13: Užívateľské rozhranie prototypu aplikácie.

Pri implementácii popísanej metódy spracovania textu, popísanej v kap. 3.1, však vyplynulo niekoľko ďalších problémov. Hlavnou nevýhodou použitia pôvodnej metódy extrakcie textu je jej dlhá doba behu. Keďže autori spracovávajú vstup vo forme statických fotografií v neobmedzenom

čase, metódu nemuseli z tohto ohľadu nijak výrazne optimalizovať. Aj keď som sa snažil postup navrhnúť a zrealizovať tak, aby bol čo najrýchlejší, nepodarilo sa mi ho optimalizovať tak, aby bežal na mobilnom zariadení vo videu v reálnom čase. Výsledkom bola veľmi dlhá doba medzi spracovávanými snímkami, užívateľovi obraz na obrazovke sekal (nízke FPS = *frame rate*). Rýchle nastavenie kamery do vhodnej pozície k textu reálneho sveta bolo veľmi zložité.

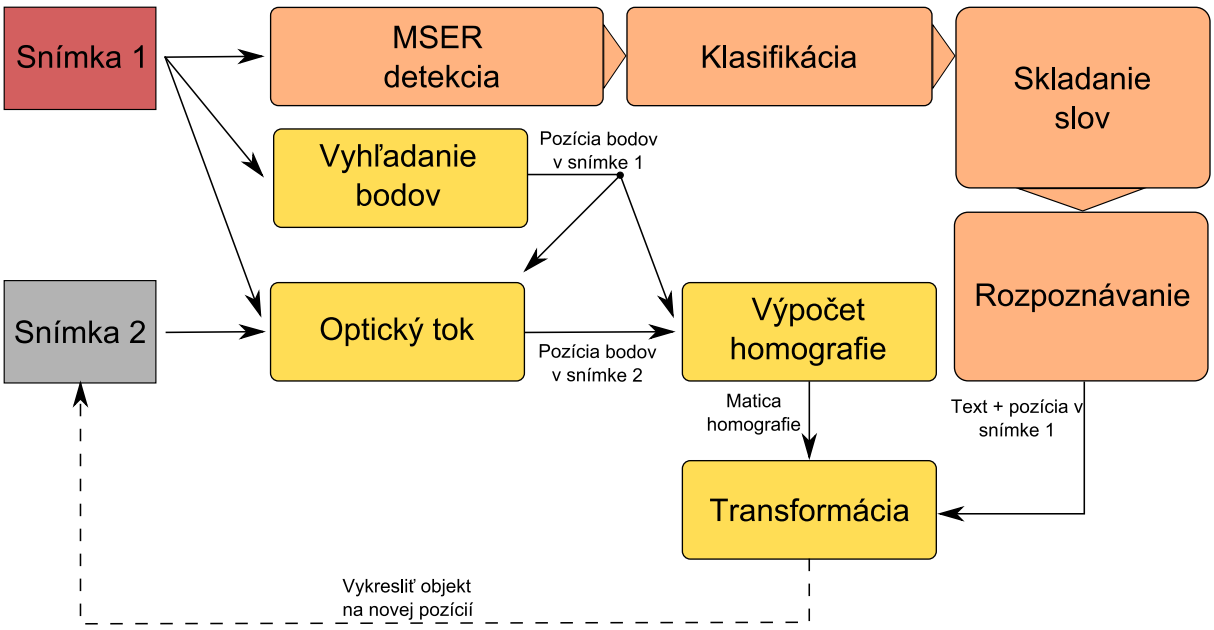
Ako riešenie daného problému sa ponúka implementácia aplikácie vo viacerých paralelných vláknach. V jednom vlákne sa spracováva a extrahuje daný text zo vstupnej snímky, druhé vlákno zatiaľ obsluhuje zobrazenie snímok na obrazovke. Do spracovania teda vstupuje vždy jedna snímka videa (tzv. *inicializačná*). Snímky, ktoré prídu počas procesu extrakcie textu, nie sú týmto procesom spracované. Na obrazovke mobilného zariadenia sa však zobrazia. Rozpoznanie textu teda prebieha v samostatných behoch, iteráciách.

Keďže spracovávam obraz z videa, vyvstáva pri tomto spôsobe riešenia ďalší problém. Užívateľ mohol počas rozpoznávania textu zmeniť nasmerovanie kamery v priestore. Pozícia extrahovaného textu môže byť teda po jeho spracovaní v aktuálnom snímku videa iná, ako bola vo vstupnej snímke (obrázok 14). V prípade, že užívateľ počas spracovania zmenil nielen smer pohľadu kamery, ale aj jej pozíciu v priestore, bude v obraze zmenená nielen pozícia výsledku, ale aj veľkosť jeho ohraničujúceho polygónu, prípadne aj jeho tvar (perspektívne skosenie atď.). Napríklad, ak užívateľ kameru k reálnemu spracovávanému textu priblížil resp. oddialil, bude veľkosť výsledného ohraničujúceho polygónu výsledku v obraze zväčšená resp. zmenšená. Možným riešením je zakomponovanie algoritmu pre sledovanie takýchto zmien medzi jednotlivými snímkami videa, *tracking textu*. Celú schému spracovania ukazuje obrázok číslo 15.



(a) **(b)**

Obrázok 14: Zmena polohy textu pri priblížení kamery. (a) Poloha textu v *inicializačnej* snímke. (b) Poloha textu v snímke po skončení procesu extrakcie textu zo snímky (a).



Obrázok 15: Schéma spracovania a zobrazenia výsledkov extrakcie textu. Snímka 1 je *inicializačná*. Oranžovou farbou je zvýraznená časť spracovania textu. Časť sledovania pohybov v obraze je žltá.

3.3 Sledovanie textu

Cieľom procesu je odhadnúť pohyb kamery v scéne z dvoch za sebou idúcich snímok videa, či v dlhšej videosekvencii. Na základe tohto pohybu je možné neskôr vypočítať pozíciu a zmeny tvaru textov v čase, pri ich premietaní na obrazovku mobilného zariadenia. Výstupom je teda takzvaná matica *homografie*, ktorá odzrkadľuje perspektívnu zmenu objektov scény v čase, kým beží spracovanie textu.

Hodnoty musíme sledovať od prvej snímky, ktorá bola vstupom pre spracovanie obrazu, postupne cez všetky snímky až po poslednú snímku iterácie spracovania obrazu, kedy sa dozvieme pozíciu a veľkosť ohraničujúceho polygónu výsledku. Táto pozícia je však platná len v prvej vstupnej snímke spracovania textu. Pozíciu a veľkosti ohraničujúcich polygónov platné v aktuálnej snímke potom ľahko vypočítame aplikovaním vypozorovanej perspektívnej transformácie medzi snímkami.

Jednotlivé polygóny menším pomocou transformácie ich kľúčových bodov. Každý bod majúci v obraze súradnice $[x, y]$ je mapovaný na novú pozíciu $[x', y']$. Transformáciu súradníc je možné vyjadriť rovnicou $P' = P \cdot A$, kde P je matica popisujúca bod v *homogénnych súradniciach* a A je transformačná matica. Pojem homogénne súradnice je v 2D priestore usporiadaná trojica $[x, y, w]$, kde x a y sú súradnice bodu a w je jeho váha. V prípade lineárnych transformácií je w rovné 1.

Transformované objekty už len zobrazíme v aktuálnej snímke videa. Originál aktuálnej snímky slúži ako vstup pre ďalšiu iteráciu spracovania textu. Kým ešte nevieme jej výsledky, je potrebné v obraze priebežne zobrazovať a prepočítavať transformácie výsledkov z predchádzajúcej iterácie.

Získanie transformačnej matice nie je triviálna záležitosť. Celkovo je možné celý proces rozdeliť do troch krokov:

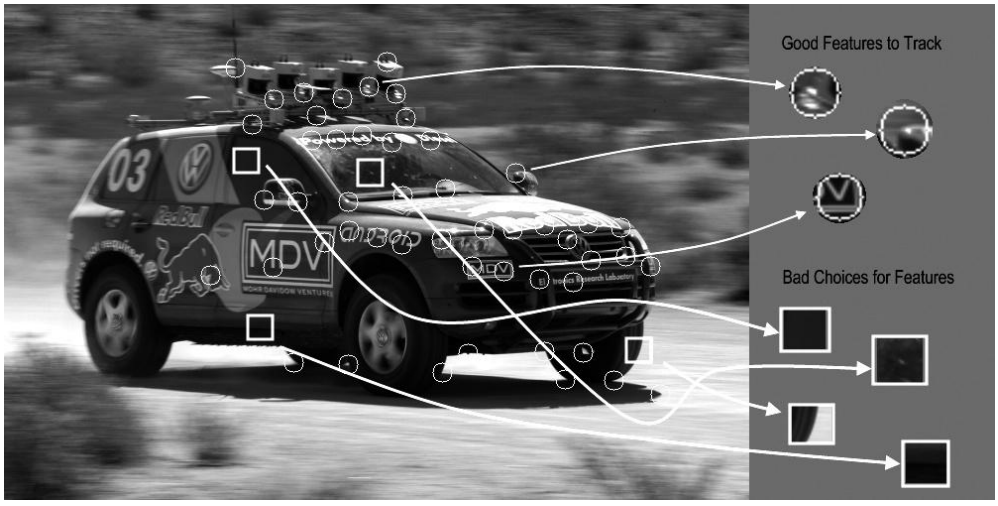
- Extrakcia význačných bodov v obraze
- Výpočet optického toku
- Aproximácia matice homografie

Informácie prezentované v nasledujúcich podkapitolách som čerpal z publikácie [20].

3.3.1 Detekcia význačných bodov

Vstupný obraz algoritmu je potrebné popísať množinou kľúčových bodov, ktoré budem v ďalších snímkach sledovať. Aby bolo možné dané body medzi snímkami spájať a čo najpresnejšie sledovať, je potrebné zvoliť metódu popisu tak, aby tieto body mali vhodné vlastnosti. Obvykle sú kľúčové body popísané pomocou viacdimenzionálnych vektorov vychádzajúcich zo zmien v ich okolí.

Väčšina techník popisu objektov v obraze pracuje s bodmi, v ktorých sa vyskytuje silná derivácia intenzity obrazu v jednom smere. Takéto body väčšinou ležia na hranách objektov (*edges*). Na rovnakej hrane však často leží množstvo bodov s podobnou vlastnosťou, preto je ich sledovanie vo videu často nevhodné (tzv. *aperture problem*). Ak je však pozorovaná silná derivácia v dvoch navzájom kolmých smeroch, je šanca pre unikátnosť bodu oveľa väčšia. Takéto body väčšinou predstavujú rohy objektov, preto sa aj v literatúre často označujú termínom *corners*. Pre sledovanie sú takéto body veľmi vhodné. Ukážku vhodných a nevhodných bodov na sledovanie pohybu prezentuje obrázok 16.



Obrázok 16: Body príznakov pre sledovanie pohybu v obraze. Prevzaté z [20].

V súčasnosti najpoužívanejšiu definíciu *corner* bodov sformuloval autor Harris [21]. Zakladá sa na matici derivácií intenzity obrázku druhého stupňa. Harrisové body sú miesta v obraze, kde má autokorelačná matica druhých derivácií intenzít dve veľké vlastné hodnoty. V podstate to znamená, že okolo bodu existuje textúra, alebo dve hrany, idúce v aspoň dvoch rozdielnych smeroch

stretávajúce sa práve v tomto bode. Autor použil druhú deriváciu intenzity, lebo sa neprejavuje ako rovnomerný gradient. Výhodou Harrisového popisu rohov je tiež jeho invariantnosť voči rotácii objektu. Použitím harrisových kľúčových bodov teda môžeme sledovať aj rotujúce objekty a nielen tie, ktoré sa v obraze hýbu. Základnú definíciu ešte viac zdokonalili a zrýchlili vo svojej práci autori Shi a Tomasi [22].

V praxi existuje ešte množstvo iných definícií význačných bodov obrazu vhodných na tracking. Často používané sú napríklad body získane postupom SIFT (*scale-invariant feature transform*) prezentované v práci [23]. Tento spôsob popisu je nezávislý na rotácii a malých afinných transformáciách.

Z hľadiska spôsobu implementácie aplikácie na mobilnom zariadení musí byť extrakcia týchto príznakov dostatočne rýchla a nenáročná na systémové zdroje. Rozhodol som sa pre to použiť metódu `cvGoodFeaturesToTrack()`, ktorá je implementovaná v používanej pomocnej knižnici na spracovanie obrazu (*openCV*). Táto implementácia je v knižnici už dostatočne optimalizovaná nielen z hľadiska jej výkonu, ale aj na presnosť trakingu jej výsledkov. Samotná implementácia detekcie význačných bodov je založená na metóde Shi a Tomasi [22]. Daný typ význačných bodov je možné v ďalších verziách aplikácie ľahko zmeniť, nahradiť iným.

3.3.2 Optický tok

Aby sme mohli získať informáciu kam sa kamera v scéne pohla, je potrebné vypočítať takzvaný *optický tok obrazu*, teda kam sa body z predchádzajúcej snímky videa posunuli v aktuálnom snímku. Vo všeobecnosti sa známe techniky delia na dve hlavné skupiny. Buď počítajú takzvaný *hustý optický tok* (*dense optical flow*), alebo *riedky optický tok* (*sparse optical flow*).

Hustý optický tok znamená, že každému bodu v obraze priradíme hodnotu a smer v ktorom sa posunul oproti predošlému obrázku. Môžeme ho počítať pomocou *Horn-Schnuck* metódy [24], alebo metódou využívajúcou postupné porovnávanie pixlov v obrázkoch pomocou pohyblivého okna. Počítanie týmito metódami je vo väčšine prípadov pomerne zložité, pomalé a náročné na výpočetný výkon.

Riedky optický tok počítajú metódy zaradené v druhej skupine. Vektor posunutia priradzujú len určitým význačným bodom obrazu, rohom (*corners* – pozri kapitolu 3.3.1). Ak majú tieto body určité želané vlastnosti, je ich sledovanie pomerne robustné a presné. Výpočetná náročnosť týchto algoritmov je pritom omnoho nižšia ako pri použití predošlých metód, sú oveľa rýchlejšie. Typickým reprezentantom je napríklad metóda *Lucas-Kanade* (LK) [25]. Pri implementácii aplikácie som sa rozhodol použiť práve túto metódu.

Lucas-Kanade metóda

Originálne bola táto metóda navrhnutá na výpočet hustého optického toku, no vzhľadom na spôsob jej práce ju môžeme aplikovať len na určitú podmnožinu bodov obrazu. Vstupom do spracovania sú príznaky získané v predchádzajúcom kroku, popísané v predošlej kapitole. Výsledkom bude teda riedky optický tok.

Metóda je postavená na troch základných predpokladoch:

1. *Konštantnosť jasu* – Vzhľad bodu na objekte v scéne sa počas pohybu medzi jednotlivými snímkami mení čo najmenej. Keďže spracovávame jednokanálové obrázky v stupňoch šedej, predpokladáme nemennosť veličiny jasu sledovaných bodov.
2. *Časová konzistencia pohybu* – Pohyb sledovaného bodu a jeho okolia sa v čase mení dostatočne pomaly. Časové vzorkovanie jednotlivých snímok je oproti veľkosti pohybu dostatočne veľké.
3. *Priestorová súvislosť* – Okolie bodu v scéne prislúcha rovnakému objektu, má spoločný pohyb, zobrazí sa v obraze do okolia daného bodu.

Uvažujme problém pre v jeden rozmer obrazu I. Podľa predpokladu č.1 je jas bodu v obraze $f(x,t)$, približne rovnaký v čase.

$$f(x,t) \equiv I(x(t),t) = I(x(t+dt), t+td)$$

Sledovaný bod teda nevykazuje v čase zmenu jasu:

$$\frac{\partial f(x)}{\partial t} = 0$$

Druhá podmienka vraví, že pohyby medzi snímkami sú dostatočne malé. Môžeme teda tieto zmeny aproximovať ako deriváciu intenzity podľa času (zmena medzi predošlou a nasledujúcou snímkou je diferenciálne malá).

$$\frac{\partial I}{\partial x}\Big|_t \left(\frac{\partial x}{\partial t}\right) + \frac{\partial I}{\partial t}\Big|_{x(t)} = 0$$

$$I_x \quad u \quad + \quad I_t \quad = 0$$

kde I_x je parciálna derivácia cez prvý obrázok podľa času, I_t je derivácia medzi jednotlivými obrázkami v čase a symbol u označuje hľadanú veľkosť vektoru v danej dimenzii x. Zovšeobecníme problém pre bod v dvojrozmernom priestore. Symbolom v označíme y-ovú zložku hľadaného vektora, pre x-vú zložku ostane symbol u.

$$I_x u + I_y v + I_t = 0$$

Pre jeden bod má však táto jedna rovnica dve neznáme. Môžeme však využiť predpoklad číslo 3. Ak sa lokálne okolie bodu pohybuje medzi snímkami koherentne, môžeme riešiť problém pohybu stredného bodu s použitím okolitých pixlov tak, aby sme vytvorili systém rovníc. Napríklad pre okno s veľkosťou 5x5 bodov zostavíme sústavu 25 rovníc, pre každý bod jednu.

$$I_x(q_1)u + I_y(q_1)v = -I_t(q_1)$$
$$I_x(q_2)u + I_y(q_2)v = -I_t(q_2)$$
$$\vdots$$
$$I_x(q_n)u + I_y(q_n)v_y = -I_t(q_n)$$

kde symboly $q_1, q_1, \dots, q_n$ označujú jednotlivé body vnútri okna. Funkcie $I_x(q_i)$, $I_y(q_i)$, $I_t(q_i)$ sú parciálne derivácie intenzity obrázku Ipodľa x, y a t vyčíslené v bode q_i v aktuálnom čase. Sústavu môžeme napísať v maticovom tvare $Ad = b$, takto:

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \qquad d = \begin{bmatrix} u \\ v \end{bmatrix}, \qquad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_t) \end{bmatrix}$$

V systéme je viac rovníc ako neznámych, takže je riešiteľná pomocou metódy minimalizácie najmenších štvorcov (*least-squares minimization*):

$$(A^T A)\, d = A^T b$$

Riešením je teda:

$$d = (A^T A)^{-1}\, A^T b$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)I_x(q_i) & \sum_i I_x(q_i)I_y(q_i) \\ \sum_i I_y(q_i)I_x(q_i) & \sum_i I_y(q_i)I_y(q_i) \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i)I_t(q_i) \\ -\sum_i I_y(q_i)I_t(q_i) \end{bmatrix}, i = 1..n$$

Daná rovnica je riešiteľná, keď je jej člen $(A^T A)$ inverzibilná. Takým sa tento člen stáva, keď má plný stupeň (2), teda keď má dva veľké vlastné vektory (textúra okolia bodu beží v aspoň dvoch rôznych smeroch). Najlepšie vlastnosti trackingu teda získame s použitím oblasti bodu definovaného ako roh (*corner* – pozri kapitolu 3.3.1).

Jeden z hlavných problémov spracovania pomocou prezentovanej metódy je, že v praxi sa v obraze vo veľkej miere vyskytujú veľké pohyby. Aby sme tieto veľké pohyby čo najlepšie zachytili, môžeme použiť na tracking väčšie okno spracovávajúce okolie bodu. Ľahko sa ale potom stane, že toto okno bude porušovať druhý predpoklad, teda priestorovú súvislosť (koherentnosť pohybu). Aby sme obišli tento problém, môžeme najskôr sledovať body v obraze vo väčšej mierke (nižšie rozlíšenie) a neskôr vylepšiť tento začiatočný odhad pohybu postupným spracovávaním obrazu vo vyššom rozlíšení až kým neprídeme k pôvodnému obrazu. Pri procese teda používame takzvané *obrazové pyramídy*.

Výpočet optického toku som implementoval pomocou knižnice OpenCV. Konkrétnu implementáciu rozoberám v kapitole 4.4. Optický tok vypočítavam nad každou snímkou danej iterácie detekcie a extrakcie textu, okrem inicializačnej snímky. Pomocou výpočtu optického toku som získal novú pozíciu kľúčových bodov v aktuálnej snímke videa. Ukážku prezentuje obrázok 17.

(a) (b)

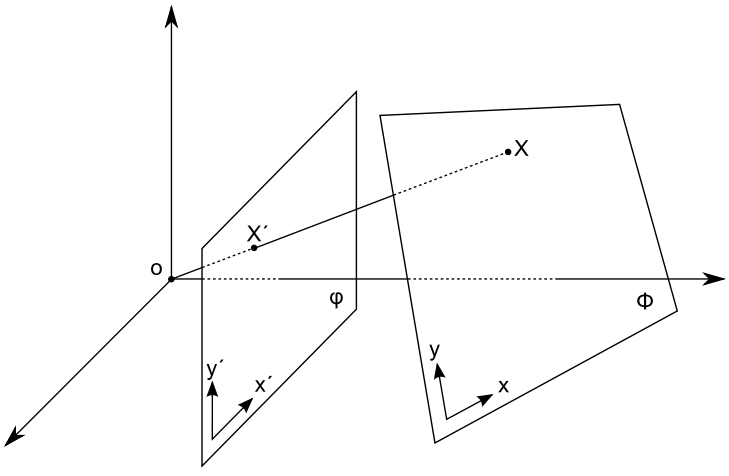
Obrázok 17: Optický tok videa. (a) Inicializačná snímka. Sledované kľúčové body sú zvýraznené zeleným krúžkom. (b) Snímka z videa z iného pohľadu. Fialové čiary indikujú pôvodnú polohu kľúčových bodov.

3.3.3 Aproximácia matice homografie

Z predchádzajúcich krokov spracovania máme množinu bodov na pozíciách z predošlej snímky a vďaka optickému toku poznáme ich pozíciu v aktuálnej snímke videa. Ostáva už len zistiť, ako sa zmenilo snímané prostredie na základe zmeny ich pozície. Zmenu snímaného prostredia vyjadríme pomocou matice *homografie*.

Homografia

Transformáciu medzi dvoma projektívnymi perspektívami tak, že priamky v nich budú opäť mapované na priamky, nazývame homografia. V literatúre sa často používa aj pojem lineárna projektívna transformácia (*linear projective transformation*). Nasledujúce informácie som čerpal z publikácií [20, 26].



Obrázok 18: Homografia stredového premietania so stredom v bode *o*. Body z roviny Φ sú mapované na body v rovine φ . Obrázok bol prevzatý a upravený z [26].

Homografickú transformáciu môžeme vyjadriť pomocou matice. Matica homografie H určuje vzťah medzi bodmi zdrojovej p_{src} a bodmi na výslednej ploche p_{dst} pomocou nasledujúceho vzťahu:

$$p_{dst} = Hp_{src}$$

$$p_{src} = H^{-1}p_{dst}$$

$$p_{dst} = \begin{bmatrix} x_{dst} \\ y_{dst} \\ 1 \end{bmatrix},\ p_{src} = \begin{bmatrix} x_{src} \\ y_{src} \\ 1 \end{bmatrix}, H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Súradnice bodov sú vo forme homogených súradníc. Jednotlivé členy matice H vyjadrujú danú transformáciu obdobne, ako bežné transformačné matice v počítačovej grafike. Keďže sú v matici homografie dôležité len pomery jej prvkov, je matica H homogenná (obdobne ako pri reprezentácií bodu pomocou homogenných súradníc).

Výpočet homografie je závislý na vyhľadaní korešpondencie jednotlivých dvojíc sledovaných bodov obrazu. Pre nájdenie správnej projektívnej transformácie dvoch rovín je potrebné nájsť minimálne štyri navzájom si odpovedajúce dvojice bodov. Tieto body pritom nesmú ležať na jednej priamke. Výpočet je založený na algoritmu RANSAC.

RANSAC

Algoritmus RANSAC (*Random Sample Consensus*) je iteratívna metóda pre odhad parametrov určitého matematického modelu. Metóda bola publikovaná autormi Fischler a Bolles v práci [27]. Nasledujúce informácie som čerpal z [28].

Cieľom algoritmu je vyhľadať v sade vstupných dát všetky také vzorky, ktoré odpovedajú danému modelu. Tieto vzorky dát sa nazývajú pojmom *inliner*. Pojmom *outliner* sa označujú body, ktoré danému modelu nezodpovedajú. Počet inliner vzoriek nie je vopred známy.

Zo vstupných dát je opakovane vyberaná náhodná podmnožina vstupných bodov tak, aby počet jej prvkov bol najmenší možný pre získanie parametrov modelu. Pomocou tohto vzorku sú vypočítané parametre modelu. V ďalšom kroku je platnosť takto vypočítaného modelu overovaná na všetkých vstupných dátach. Overovanie prebieha pomocou rôznych ohodnocovacích funkcií. Najčastejšie sa používa množstvo inliner bodov. Koniec algoritmu nastáva ak pravdepodobnosť nájdenia lepšieho modelu klesne pod vopred definovanú hodnotu prahu. Pravdepodobnosť nájdenia lepšieho modelu η je rovná:

$$\eta = (1 - P_I)^k,$$

$$P_I = \frac{\binom{I}{m}}{\binom{N}{m}} = \prod_{j=0}^{m-1} \frac{I - j}{N - j} \approx \varepsilon^m,$$

kde symbol P_I označuje pravdepodobnosť, že vzorka s veľkosťou m je náhodne vybraná z N dátových bodov. Symbol Ije počet inliner bodov v k vzorkách. Symbolom ε označujeme pomer

počtu inliner prvkov k počtu prvkov vo vstupnej množine, teda $\varepsilon = \frac{I}{N}$. Počet vzoriek potrebných na zaistenie danej pravdepodobnosti η je daný rovnicou

$$k = \frac{\log(n)}{\log(1 - P_I)}.$$

Pri výpočte matice homografie je zo sledovaných bodov v každej iterácií náhodne vybraná vzorka štyroch párov navzájom si korešpondujúcich bodov. Z nich je vypočítaná matica homografie. Pomocou tejto matice sú následne všetky ostatné body transformované. Jednotlivé body sú označené ako inlinery resp. outlinery podľa toho, či odpovedajú vypočítanej homografii. Po ukončení všetkých iterácií je vybraná homografia s najväčším počtom inlinerov. Výsledná matica homografie je vypočítaná z daných inliner bodov.

Získanú maticu homografie ďalej používam ako maticu perspektívnej transformácie bodov ohraničujúcich polygónov nájdeného textu. Transformáciu vykonávam v každej snímke videa, pokiaľ existujú nájdené texty.

Problémom popísaného postupu je akt, že na výpočet transformačnej matice používam kľúčové body z celej scény. V reálnych scénach sa môže ľahko stať, že sa bude počas spracovania časť obrazu hýbať nezávisle na pohybe kamery (autá, chodci…). Ak sa pri výpočte homografie použijú body z povrchu takéhoto hýbajúceho sa objektu, bude určenie novej polohy textu chybné. Riešením by bolo na výpočet homografie používať iba body, ktoré sú vnútri ohraničujúceho obdĺžnika textu, alebo v jeho blízkom okolí. Vypočítavala by sa tým pádom pre každe slovo vlastná transformačná matica. Pri extrakcií bodov však ešte neviem povedať kde sa bude v obraze text nachádzať. Nemožno preto zaručiť, že sa v každej nájdenej oblasti budú nejaké body vhodné na sledovanie nachádzať.

4 Implementácia

V nasledujúcom texte v skratke popíšem beh a štruktúru výsledného programu. Popíšem tiež niektoré zaujímavé časti jeho implementácie.

Kapitola sa začína časťou prezentujúcou nástroje a technológie použité pri implementácii výslednej aplikácie. V ďalšej podkapitole je stručne popísaná hlavná štruktúra programu, jeho trieda vstupnej aktivity a metódy riadiace jej beh. V podkapitole 4.3 je detailnejšie popísaná implementácia procesu spracovania obrazu na text. Podkapitola je členená na dva menšie celky, podľa programovacieho jazyka v ktorom boli kroky procesu naprogramované. Podkapitola tiež rozoberie spôsob a triedy potrebné pre komunikáciu medzi týmito dvoma programovými časťami procesu. V nasledujúcej podkapitole číslo 4.4 je stručne rozobratá implementácia sledovania pohybu textu v obraze. Posledná podkapitola predstavuje spôsob implementácie trénovania modelu klasifikácie MSER oblastí.

4.1 Použité nástroje

Väčšina kódu výslednej aplikácie je implementovaná pomocou jazyka Java. Výpočetne náročné časti a časti zaoberajúce sa spracovaním obrazu sú implementované pomocou jazyka C++. Pri vývoji som využíval možnosti nástrojov Android SDK v kombinácií s Android NDK. Všetky balíky nástrojov som navzájom prepojil vo vývojovom prostredí *eclipse*.

Navrhnuté riešenie úlohy si vyžaduje implementáciu množstva komplexných algoritmov spracovania obrazu. Z pohľadu efektívnosti a praktickosti by bola ich ručná implementácia neúmerne zdĺhavá a náročná. Druhou možnosťou je využitie niektorej existujúcej knižnice, ktorá dané operácie už implementuje. Tento variant poskytuje tiež niekoľko značných výhod. Algoritmy sú v týchto knižniciach už plne odladené pre danú platformu a optimalizované na čo najvyšší výkon. Väčšinou sú vo forme komplexného balíka knižníc, implementujúcich väčšinu potrebných operácií.

V nasledujúcom texte sa pokúsim stručne zhrnúť výhody a nevýhody balíkov knižníc, ktoré som pri svojom riešení použil.

Asi najznámejšou a najrozšírenejšou knižnicou nástrojov na spracovanie obrazových dát je ***OpenCV***. Obsahuje cez 2500 optimalizovaných algoritmov implementovaných pomocou jazyka C a C++. Knižnica je prenosná na mnoho operačných systémov, medzi nimi napríklad MS Windows, Linux, Mac a aj na systém Android. Knižnicu je možné použiť zdarma pre komerčné aj nekomerčné účely. V súčasnosti sa nachádza vo verzií 2.4 a je neustále aktualizovaná a rozširovaná. Poskytuje výbornú on-line dostupnú dokumentáciu a množstvo príkladov. Keďže knižnicu pri svojej práci používa veľké množstvo vedeckých pracovníkov a nadšencov, riešenia problémov a postupov sú

často dostupné na množstve diskusných fór. *OpenCV* sa zameriava na riešenie širokej palety problémov počítačového videnia a spracovania obrazu, a má preto mnoho rôznych využití. Okrem iných v sebe obsahuje funkcie na získavanie príznakov z obrazu, rozoznávanie tvárí, rozoznávanie giest, segmentáciu, stereo projekciu a stereo vnímanie, nástroje pre tracking a ďalšie. Ako podporné elementy k zmieňovaným funkcionalitám sú v knižnici implementované metódy štatistického strojového učenia, ako napríklad neurónové siete, alebo SVM. Pomocou nástrojov OpenCV je tiež možné pristupovať k videu z kamery mobilu za pomoci integrovaného rozhrania. Pre použitie na platforme Android je možné použiť dva rôzne prístupy: OpenCV s Android NDK, alebo *JavaCV* s Android SDK. JavaCV je framework zaobaľujúci väčšinu funkcií OpenCV do jazyka Java. Pri implementácií aplikácie som sa rozhodol využiť riešenie OpenCV + Android NDK. V mojom riešení je použitá pri väčšine operácií s obrazom, od detekcie MSER cez podporu pri získavaní príznakov až po metódy pre tracking. Veľkou pomocou bolo tiež použitie vbudovaných algoritmov strojového učenia, SVM.

Jednotlivé časti knižnice OpenCV je možné do vyvíjanej aplikácie integrovať už v čase kompilácie. Niektoré jej komponenty a funkcie však môžu byť aplikácií poskytované aj z iného zdroja, konkrétne z aplikácie *OpenCV Manager*[15]. Tá obsahuje vždy najaktuálnejšie zdrojové súbory knižnice pre dané mobilné zariadenie. Obsahuje tiež najnovšie aktualizácie ohľadom stability a výkonu na danom zariadení. V mojom riešení takto používam rozhranie pre prístup k snímkam z kamery. V prípade, že užívateľ spustí moju aplikáciu a aplikácia *OpenCV Manager* nie je na danom zariadení ešte nainštalovaná, bude mu táto možnosť automaticky ponúknutá.

Posledným krokom spracovania textu v navrhnutej metóde je rozpoznávanie obrazu (kap. 3.1.4). Autori v pôvodnej prezentovanej metóde navrhli vlastný spôsob rozpoznávania, pracujúci na základe klasifikácie základných syntetických fontov písma. Použili pri nej tiež špeciálny typografický a jazykový model. V tomto kroku som sa od prezentovanej metódy pri svojom riešení odchýlil. Na rozpoznávanie som využil OCR nástroj na rozpoznávanie textu *Tesseract*.

Knižnica ***Tesseract*** je open source OCR systém vyvíjaný od roku 2006 spoločnosťou Google. Je primárne navrhnutý na spracovanie skenovaných dokumentov a celých stránok textu. V kombinácii s knižnicou na spracovanie obrázkov od firmy Leptonica, dosahuje v týchto dokumentoch veľmi dobré výsledky. Výsledky jej priameho využitia na rozpoznávanie textov zo snímok z reálneho sveta však nie sú presvedčivé, najmä kvôli často sa vyskytujúcemu komplexnému pozadiu a množstvu ďalších obrazových šumov. Ako vstup pre systém *Tesseract* však používam binárny obraz nájdeného slova vo forme spojených MSER oblastí s odfiltrovaným pozadím. Presnosť rozpoznania takýchto výsekov by preto mala byť dostačujúca.

Systém existuje pre systémy MS Windows, Linux a Mac OS X. Existujú nástroje umožňujúce využitie tejto knižnice aj na mobilných zariadeniach so systémom Android. Na tejto platforme ale nie

[15] https://play.google.com/store/apps/details?id=org.opencv.engine

je ešte jej bezchybné fungovanie príliš otestované. V mojom riešení som použil framework ***tess-two***[16], zaobaľujúci funkcie do jazyka Java. Rozpoznanie bolo preto potrebné implementovať na Java strane kódu, čím sa koncept implementácie spracovania textu skomplikoval. Ostatné časti sú totiž implementované v native časti programu, písanej v jazyku C++.

Súčasťou riešenia *Tesseract* je tiež slovník, slúžiaci na zlepšenie výsledného rozpoznania. Okrem najznámejších svetových jazykov ako sú angličtina, nemčina, španielčina či francúzština, podporuje okolo 60 ďalších menej používaných jazykov (vrátane slovenčiny a češtiny). Pri rozpoznávaní je možné použiť aj viac jazykov naraz. Celý proces sa tým však znateľne predĺži.

4.2 Riadenie spracovania

Pri implementácii aplikácie som použil techniky objektového programovania. Minimálna verzia potrebného API pre spustenie vytvorenej aplikácie je verzia 8, teda minimálne systém vo verzii 2.2 (Fryo). Podľa oficiálneho prieskumu[17] som tak obsiahol som v súčasnosti približne 95% všetkých zariadení s OS Android. Aplikácia potrebuje pri inštalácií povolenie prístupu k vbudovanej kamere zariadenia.

Vstupným bodom programu je Java trieda `MainActivity` rozširujúca základnú systémovú triedu `Activity`. Štandardným spôsobom obsluhuje odozvy na stavy životného cyklu aktivity v triedach `onCreate(…)`, `onPause()`, `onResume()` a `onDestroy()`.

Najdôležitejšou z nich je metóda `onCreate(…)`, ktorá je volaná pri prvom spustení aktivity. Inicializuje sa tu hierarchia hlavných ovládacích prvkov definujúca vzhľad hlavného okna. Prebieha tu tiež registrácia systémových event listenerov zachytávajúcich udalosti užívateľa spojené s dotykovým ovládaním (dlhšie stláčanie tlačidiel, stláčanie do priestoru určeného na vykreslenie obrazu z kamery…), zmenou zdieľaných nastavení aplikácie a linkovanie listenera reagujúceho na udalosti kamery.

Počas svojho behu aplikácia potrebuje načítavať informácie z niekoľkých externých súborov. Prvý súbor definuje natrénovaný model SVM. Uložený je ako súbor typu XML. Ďalšími sú súbory slovníkov v rôznych jazykoch pre rozpoznávací systém Tesseract (typ .traineddata). Do vytvoreného prototypu aplikácie som integroval slovník anglicky a český. Pri zostavovaní aplikácie sú tieto súbory integrované do výsledného balíčku *.apk* ako assety. Počas svojho prvého spustenia aplikácia v metóde `onCreate()`kontroluje prítomnosť týchto zdrojov v internom úložisku, ktoré jej bolo pridelené systémom. V prípade, že tam tieto súbory nenájde, extrahuje ich na správne miesto pomocou objektu systémovej triedy `AssetManager` v metóde `copyAsset()`. Súbory v internom úložisku po vypnutí

[16] https://github.com/rmtheis/tess-two
[17] http://developer.android.com/about/dashboards/index.html

aplikácie ostávajú, takže každé ďalšie spustenie je už oproti prvému rýchlejšie. Po odinštalovaní aplikácie sú súbory vymazané.

Pri prvom spustení a pri opätovnej aktivácií hlavnej aktivity do popredia je volaná metóda `onResume()`. V nej sa volá inicializácia komponentov knižnice OpenCV. Tiež tu nastáva natiahnutie a linkovanie natívnej časti kódu v podobe knižnice. Systémové volanie linkovania natívnej knižnice je možné vykonať až po správnej inicializácii všetkých častí OpenCV knižnice.

Aby sa do triedy `MainActivity` dostal obraz z kamery zariadenia, implementuje táto trieda listener OpenCV knižnice `CvCameraViewListener`. Pomocou implementácií jeho metód reaguje na udalosti spojené s obrazom z kamery.

- `onCameraViewStarted(…)` – Zavolaná po inicializácii kamery. Hodnoty predaných argumentov udávajú výšku a šírku obrazu z kamery.
- `onCameraViewStopped()` – Zavolaná po zrušení práce s kamerou, napríklad ak užívateľ zvolí do popredia inú aktivitu.
- `onCameraFrame(…)` – Volaná pri príchode novej snímky z videa.

Riadenie spracovania obrazu prebieha v metóde `onCameraFrame(…)`. Hodnota jej vstupného argumentu reprezentuje aktuálnu snímku videa. Jeho typ je `Mat` (matica). Matica je štandardným vyjadrením obrázku v knižnici OpenCV. Prvky matice reprezentujú hodnoty jednotlivých bodov obrazu. Metóda na výstupe očakáva opäť maticu obrázku, ktorý sa zobrazí na obrazovke zariadenia, pomocou objektu triedy `CameraView`.

Schému spracovania znázorňuje obrázok 15, v kapitole 3.2. Navrhnuté riešenie beží vo viacerých vláknach. Po príchode snímky je potrebné v prvom rade zistiť, či vlákno detekcie a extrakcie textu stále beží, alebo nie.

Ak vlákno momentálne nebeží, je prichádzajúca snímka takzvane *inicializačná*. Znamená to, že je vstupom pre ďalší beh spracovania obrazu na text. Spustím teda nové vlákno s rozhraním `m_runnable` implementujúcim tento proces nad danou snímkou. Tiež je potrebné nanovo inicializovať proces trackingu, zistiť body ktorých pohyb budem v obraze sledovať. Novou inicializáciou však stratím hodnoty transformačnej matice vypočítanej v predošlom behu. Pred inicializáciou je teda potrebné prípadné výsledky z predošlého behu transformovať.

Ak vlákno detekcie a extrakcie textu pri príchode novej snímky ešte stále beží, snímka nie je inicializačná, ale treba v nej pozorovať pohyb obrazu. Spustím preto proces výpočtu matice homografie s touto a uloženou predošlou snímkou.

V oboch prípadoch nasleduje proces vykreslenia prípadných výsledkov do výstupného obrazu metódy. Pred vykreslením sú výsledky transformované pomocou aktuálnej matice homografie. Ak je táto snímka *inicializačná*, výsledky už transformované boli, a preto sa transformačný krok v tomto prípade preskakuje.

4.3 Spracovanie obrazu na text

Keďže je proces spracovania obrazu náročný a využíva množstvo operácií nad jednotlivými bodmi obrazu, chcel som ho celý implementovať v *native* časti programu. Native časť je programovaná v jazyku C++ a k výslednej aplikácií sa linkuje ako vopred skompilovaná knižnica funkcií a tried. Prepojenie Java a C++ častí programu zaručuje framework JNI (*java native interface*). Na rozpoznávanie slov som použil implementáciu OCR systému *Tesseract* vo forme frameworku *tess-two*, ktorý funguje z Java časti programu. Musel som teda celý proces spracovania obrazu rozdeliť na dve časti naprogramované v iných programovacích jazykoch (Java, C++). Výsledok procesov detekcie, extrakcie a zhlukovania oblastí do slov je potrebné poskytnúť z native časti do rozpoznávania implementovaného v Java časti kódu.

Celý proces spracovania obrazu na text prebieha v samostatnom vlákne. Je obalený v privátnej statickej Java metóde `detectAndRecognizeWords(…)` triedy `MainActivity`. Vstupom metódy je matica farebného obrázku z videa. Výstupom je zoznam objektov triedy `DetectedWord`.

Trieda `DetectedWord`

Objekt triedy `DetectedWord` reprezentuje v systéme jeden v obraze rozpoznaný riadok textu, jedno slovo. Atribútmi triedy sú rozpoznaný text (reťazec), pravdepodobnosť jeho rozpoznávania (celé číslo 0 až 100), zoznam štyroch kľúčových bodov jeho ohraničujúceho obdĺžnika a obraz (matica) vytvorený spojením MSER oblastí, ktoré dané slovo reprezentujú. Tento obraz je však potrebný len pri rozpoznávaní a pre ušetrenie pamäti ho po tomto kroku uvoľním.

Trieda implementuje okrem štandardných get/set metód aj verejnú metódu porovnania `compareMe(…)` dvoch objektov tohto typu. Používa sa pri filtrovaní hypotéz reprezentujúcich rovnaký riadok textu v obraze. Využíva pri tom privátnu metódu `intersectRect(…)` vracajúcu priesečník dvoch obdĺžnikových polygónov v 2D priestore.

Pomocná metóda `isPointInside(…)` určí, či sa bod dodaný ako argument nachádza vo vnútri ohraničujúceho obdĺžnika slova. Program pomocou nej zisťuje, na ktoré ohraničujúce obálky slov užívateľ na obrazovke klikol.

Spracovanie obrazu na text sa v Java metóde `detectAndRecognizeWords(…)` začína inicializáciou systému *Tesseract*. V prípade úspechu spracovanie pokračuje volaním native metódy `HandleFrame(…)`. Tá mi po svojom ukončení poskytne naplnený zoznam nájdených hypotéz riadkov textu reprezentovaných vo forme binárnych obrázkov. Tieto obrázky sú následne rozpoznávané na text systémom Tesseract. Výsledný vektor rozpoznaných slov (`ArrayList<DetectedWord>`) je následne ďalej spracovávaný v hlavnej aktivite aplikácie.

4.3.1 Native časť spracovania obrazu

Native časť spracovania obrazu na text je riadená metódou `HandleFrame()`. V tejto metóde sa vykonávajú kroky detekcie, klasifikácie a skladania oblastí do slov. Vstupom metódy sú adresa matice obrázku reprezentujúceho snímku videa, cesta k súboru definujúcemu natrénovaný SVM model a objekt triedy `DetWordWrapper`. Výsledkom je naplnený zoznam slov (`m_words`) predaného objektu `DetWordWrapper`.

Detekcia MSER oblastí

Prvým krokom v metóde `HandleFrame(…)` je detekcia MSER oblastí volaním funkcie `detect_MSER_areas(…)`. Jej jediným parametrom je daná matica vstupného obrazu. Pred detekciou je vstupný obraz prevedený z RGB do HSV farebnej reprezentácie a rozdelený do odpovedajúcich matíc jednotlivých kanálov. Samotné vyhľadávanie MSER oblastí v každom kanále obrazu prebieha pomocou volania OpenCV MSER detektoru. Jeho výsledkom je vektor oblastí. Jedna oblasť je reprezentovaná opäť ako vektor bodov (`vector<Point>`). Jednotlivé vektory oblasti z každého spracovávaného kanálu postupne pridávam do objektu triedy `ImageAreaSet`. Tento objekt je predaný ako výsledok funkcie `detect_MSER_areas(…)`. Trieda `ImageAreaSet` reprezentuje súbor oblastí celého spracovávaného snímku. Pri pridávaní nového vektora oblastí do objektu tejto triedy je každá nájdená oblasť reprezentovaná inštanciou triedy `DetectedMSERArea`. Táto trieda obsahuje okrem zoznamu jednotlivých súradníc bodov v obraze patriacich danej MSER oblasti aj ďalšie vlastnosti. V konštruktore je napríklad k oblasti vypočítaný odpovedajúci ohraničujúci obdĺžnik pomocou OpenCV metódy `boundingRect(…)`. Tiež sú vypočítané všetky príznaky potrebné pre krok klasifikácie pomocou metódy `computeFeatures()`.

Klasifikácia oblastí

Vytvorený objekt triedy `ImageAreaSet` teda po detekcií obsahuje vektor naplnený nájdenými MSER oblasťami (`DetectedMSERArea`) z celého obrazu s názvom `m_areaSet`. V ďalšom kroku je potrebné každú oblasť klasifikovať pomocou natrénovaného SVM modelu. Klasifikácia je v metóde `HandleFrame` vykonávaná pomocou OpenCV objektu triedy `CvSVM`. Predchádza jej inicializácia modelu z XML súboru. Oblasti sú klasifikované pomocou volania metódy `CvSVM.predict(Mat)`. Príznaky jednotlivých oblastí je pred tým potrebné previesť do jednej matice typu `CV_32FC1` predanej ako argument. Každej oblasti objektu `ImageAreaSet`, ktorá bola klasifikovaná ako písmeno, je nastavený atribút `label` na hodnotu 1. Oblasti opačnej triedy sú reprezentované hodnotou -1.

Vytváranie a validácia hypotéz o riadkoch

Po dokončení klasifikácie všetkých oblastí nasleduje proces ich spájania do slov. Ako som popísal v kapitole 3.1.3, spájanie je založené na vytváraní hypotéz, ktoré sú neskôr validované. Vytvorenie a validácia hypotéz prebehne v `HandleFrame(…)` pomocou volania verejnej metódy

`CreateHypotheses()` objektu triedy `ImageAreaSet`. Jej výsledkom je vektor objektov triedy `WordHypothesis`. Táto trieda reprezentuje hypotézu riadku, teda množinu MSER oblastí obrazu, ktoré spolu tvoria riadok textu. Jednotlivé oblasti z ktorých sa hypotéza skladá, sú definované pomocou ich jednoznačného indexu vo vektore oblastí objektu `ImageAreaSet`. Z tohto vektora sú na začiatku práce metódy `CreateHypotheses()` vybrané oblasti klasifikované ako písmená. Pre každú z nich je vytvorený jeden objekt triedy `WordHypothesis`.

Následne postupne prechádzam každú z takto vytvorených hypotéz. Pri spracovaní danej hypotézy si vytváram dočasný vektor indexov oblastí, ktoré možno k hypotéze pripojiť v aktuálnej iterácii. Rozhodnutie možnosti o pripojenia, alebo nepripojenia oblasti sa deje v metóde `canBeAdded(…)` danej hypotézy, s argumentom reprezentujúcim skúmanú oblasť ako ukazovateľ na objekt `DetectedMSERArea` z vektoru `m_areaSet`. Táto metóda implementuje podmienky heuristickej funkcie definovanej v kapitole 3.1.3. V prípade, že sa našlo viac ako jedna MSER oblasť s možnosťou pripojenia, vyberie sa prvá z nich ktorá bola klasifikovaná ako písmeno. Ak nie je takto klasifikovaná ani jedná z kandidátnych oblastí, vyberie sa prvá nájdená (najnižší index). Pripojenie oblasti k hypotéze sa deje pomocou volania metódy `addArea(…)`. Musia sa v nej okrem pridania daného indexu tiež prepočítať všetky vlastnosti a atribúty danej hypotézy. V prípade, že sa už nenájde žiadna nová oblasť na pripojenie, je spracovávaná hypotéza takzvane uzavretá. Vtedy je potrebné skontrolovať duplicitu vzniknutej hypotézy s ostatnými hypotézami vo vektore uzavretých hypotéz. Duplicita sa prejavuje tým, že dve hypotézy reprezentujú rovnakú postupnosť MSER oblastí. V prípade, že sa nájde vo vektore uzavretých hypotéz duplicita s aktuálne uzavretou hypotézou, nová hypotéza sa doň nevloží. Spracovanie pokračuje, až kým nie sú všetky začiatočné hypotézy uzavreté.

Následne je potrebné nájdené hypotézy validovať pomocou podmienok popísaných v tabuľke 4, v kapitole 3.1.3. Validácia prebieha v metóde `validateHypothesis()` danej hypotézy. Do výsledného vektoru metódy `CreateHypotheses()` sú pridané len prvky spĺňajúce všetky podmienky validácie. Pri vyčísľovaní podmienok sa ešte využíva viacero ďalších metód triedy `WordHypothesis`. Detaily je možné získať štúdiom zdrojových súborov aplikácie.

Zaslanie výsledkov do Java kódu

Výsledkom metódy `CreateHypotheses()` je vektor hypotéz zložených z MSER oblastí `vector<WordHypothesis>`, ktoré je potrebné ďalej spracovať v kroku rozpoznávania. Dostali sme sa opäť na úroveň metódy `HandleFrame(…)`. Ďalší krok teda spočíva v zaslaní potrebných informácií do Java časti aplikácie.

Toto prepojenie dvoch častí programu nie je jednoduché, keďže framework JNI poskytuje priame mapovanie pamäte vstupov a výstupov volaných native metód len pre základné typy parametrov. Ja som však potreboval z native časti preniesť tam vytvorené matice obrázkov reprezentujúce nájdené a poskladané slová. Pre zobrazenie slov v užívateľskom rozhraní som tiež potreboval preniesť pozíciu týchto slov v súradniciach spracovaného obrazu. Po skončení native

metódy sa na Java strane nedali štandardne použiť vytvorené native objekty, lebo daná pamäť už nebola platná. Tento problém som vyriešil pomocou implementácie premosťovacej triedy `DetWordWrapper`.

Pomocná Java trieda `DetWordWrapper` zaobaľuje vo svojich metódach všetky operácie, ktoré je potrebné vykonať na Java strane kódu. Do native funkcie pomocou nich poskytujem potrebné odkazy na vytvorené Java objekty. Trieda obsahuje ako svoj jediný verejný atribút (`m_words`) zoznam objektov typu `DetectedWord`. Volaním metódy `addNewWord(…)` pridám do tohto zoznamu na Java strane nový prvok (inštanciu objektu `DetectedWord`) . Parametrami tejto metódy sú súradnice ľavého horného bodu ohraničujúceho obdĺžnika pridávaného slova v spracovanom obraze a jeho šírka a výška. Konštruktor objektu `DetectedWord` vyžaduje rovnako len špecifikáciu daných súradníc, výšky a šírky. Zavolaním konštruktoru objektu `DetectedWord` sa však pre nový prvok zoznamu `m_words` vytvorí aj prázdna inštancia matice obrázku (je to jeden z jeho atribútov). Druhou metódou implementovanou v triede `DetWordWrapper` je verejná metóda `getWordMatAddr()`. Táto metóda vráti číslo (`long`) reprezentujúce adresu matice obrázku posledného prvku v zozname `m_words`, alebo -1 ak sa v zozname nenachádzajú žiadne prvky. Hodnotu adresy pre native kód získa pomocou metódy `getNativeObjAddr()`, implementovanej v Java reprezentácií OpenCV triedy `Mat`. V native metóde spracovávajúcej obraz pomocou získanej adresy naplním objekt matice vytvorený na strane Java kódu. Matica obrázku je preto následne platná aj v Java kóde.

Pomocou volania `FindClass(…)` je možné získať vo funkcii `HandleFrame()` z ukazovateľa na Java prostredie JNI frameworku odkaz na Java triedu s daným menom. Následne je možné podobne získať komunikačné ID číslo metódy implementovanej takto získanou triedou, pomocou volania `GetMethodID(…)`. V mojom riešení teda najskôr získam odkaz na triedu typu `DetWordWrapper` a následne jej metódy `addNewWord(…)` a `getWordMatAddr()`. Potom pre každý prvok z vektoru hypotéz postupne volám tieto metódy objektu `DetWordWrapper`. Volanie metód je opäť sprostredkované ukazovateľom na Java prostredie z JNI frameworku. Najskôr spustím metódu `addNewWord(…)`, ktorá vytvorí prvok v zozname hypotéz na Java strane. Pre tento prvok tiež alokuje maticu obrázku. Jej adresu si následne zistím zavolaním Java metódy `getWordMatAddr()`. V ďalšom kroku adresu použijem pre ukazovateľ a danú maticu naplním binárnym obrázkom riadku. Ten vytváram spojením jednotlivých obrázkov MSER oblastí tvoriacich danú hypotézu. Spojený obraz má tvar obdĺžnika. Jeho dimenzie určím s pomocou vyhľadania minimálnych x a y súradníc ľavých horných resp. maximálnych x a y súradníc pravých spodných bodov ohraničujúcich obdĺžnikov oblastí tvoriacich danú hypotézu.

4.3.2 Rozpoznávanie

Proces spracovania obrazu pokračuje krokom rozpoznávania vytvorených obrázkov slov na text, v metóde `detectAndRecognizeWords(…)` triedy `MainActivity`. Po úspešnom skončení metódy `HandleFrame(…)` sú tieto obrázky riadkov uložené ako atribúty typu `Mat`, prvkov zoznamu typu `DetectedWord`. Objekt rozhrania systému Tesseract však ako typ vstupného obrázku rozpoznania akceptuje len bitmapu, súbor typu `File`, alebo typ firmy *Leptonica* – `Pix`. Pre každý riadok teda vytvorím prázdnu bitmapu typu `ARGB_8888` s rozmermi daného obrázku. Následne do prázdneho bitového obrázka prekopírujem hodnoty matice riadku pomocou OpenCV metódy `Utils.matToBitmap(…)`.

Pred rozpoznávaním je potrebné rozhranie Tesseract inicializovať pomocou volania metódy `init(…)`. Pomocou hodnôt jej parametrov sa dá nastaviť metóda rozpoznávania a tiež ktoré jazyky slovníku budú použité. Pri testovaní na mobile (Samsung Galaxy S II) sa žiaľ ukázalo, že pri použití inej ako štandardnej prednastavenej metódy (`OEM_DEFAULT`), bolo rozpoznávanie veľmi nestabilné a celý systém často padal. Iné metódy by mali vykazovať kvalitnejšie výsledky, no celý proces rozpoznania by sa s ich použitím aj značne spomalil. Proces by sa spomalil aj použitím viacerých jazykových mutácií slovníkov.

Po inicializácii rozhrania nastavím obrázok na rozpoznávanie pomocou volania metódy `setImage(Bitmap)`. Rozpoznanie spustím zavolaním Tesseract metódy `getUTF8Text()`. Po úspešnom behu metóda vráti hodnotu rozpoznaného reťazca. Z neho sú následne odfiltrované znaky iné ako veľké a malé písmena a čísla. Ak je dĺžka takto upraveného výsledného rozpoznaného reťazca menšia ako 3 znaky, alebo je priemerná hodnota presnosti rozpoznania riadku nižšia ako 60 percent, bude tento riadok z výsledku odfiltrovaný. Presnosť rozpoznania riadku získavam z Tesseract rozhrania pomocou metódy `meanConfidence()`.

Vstupom detekcie MSER oblasti je viac farebných kanálov toho istého obrazu. Môže sa preto ľahko stať, že to isté slovo v obraze bude reprezentované viacerými rôznymi hypotézami. Výsledný zoznam riadkov je preto v ďalšom kroku podrobený vyhľadávaniu a filtrácií takýchto duplicitných slov. Pri vzájomnom porovnávaní prvkov zoznamu hypotéz využívam ich metódu `compareMe(…)`.

Metóda vráti -1, ak považuje hypotézy sa rozdielne. Ak ale na základe porovnania usúdi, že hypotézy reprezentujú v obraze ten istý text, vráti buď 1, alebo 2. Hodnotu 1 vráti, ak je pre ďalšie spracovanie lepšia prvá z nich, 2 v opačnom prípade. Porovnanie funguje nasledovne:

Ak sa nájde priesečník ohraničujúcich obdĺžnikov porovnávaných riadkov, zisťuje sa jeho plocha. Ak je plocha priesečníku minimálne 90% plochy aspoň jedného z ohraničujúcich obdĺžnikov porovnávaných riadkov, reprezentujú hypotézy rovnaký riadok. Na určenie lepšej z nich porovnávam ich rozpoznané reťazce. Ak sú rovnaké, vrátim hodnotu 1. Inak zisťujem, ktorá hypotéza má vyššiu hodnotu presnosti rozpoznania textu riadku. V prípade že je to prvá z nich, vrátim 1. Inak vrátim 2.

Vo výsledku filtrovania nechávam vždy len jednu najlepšiu variantu duplicitných hypotéz. Algoritmus filtru v jazyku Java funguje takto:

Vstup: Zoznam (`ArrayList`) objektov typu `DetectedWord` s menom `retArrList`

Výstup: Zoznam `retArrList` odfiltrovaný od objektov reprezentujúcich rovnaký text

Metóda:

```
int dwCount = retArrList.size();
boolean mask [] = new boolean[dwCount];
boolean somethingToDelete = false;
for(int i=0; i<dwCount; i++){
	DetectedWord dw1 = retArrList.get(i);

	for(int j=i+1; j<dwCount; j++){
		if(!mask[j]){
			int compare = dw1.compareMe(retArrList.get(j));
			if( compare != -1){
				somethingToDelete = true;
				if(compare == 2){
					mask[i] = true;
					dw1 = retArrList.get(j);
				}else{
					mask[j] = true;
				}
			}
		}else
			continue;
	}

}

if(somethingToDelete){
	ArrayList<DetectedWord> swap = new ArrayList<DetectedWord>();
	for(int i=0; i<dwCount;i++){
		if(!mask[i]){
			swap.add(retArrList.get(i));
		}
	}
	retArrList = swap;
}
return retArrList;
```

Výsledný zoznam objektov typu `DetectedWord` vrátim z vlákna spracovania obrazu do zdieľaného statického atribútu `m_detectedWords` hlavnej aktivity `MainActivity` riadiacej celé spracovanie.

4.4 Sledovanie pohybu

Proces sledovania pohybov v scéne je riadený hlavnou triedou `MainActivity` v metóde `onCameraFrame(…)`. Metóda je volaná ako odozva pri príchode novej snímky videa. Proces využíva niekoľko privátnych statických atribútov tejto triedy, ktoré zachovávajú potrebné informácie

o predchádzajúcej snímke videa. Pre sledovanie pohybu je potrebné uchovávať si maticu obrazu predchádzajúcej snímky (`prev_Img`) a dve matice reprezentujúce polohy sledovaných bodov v predchádzajúcej (`prev_points`) a aktuálnej (`next_points`) snímke. Výsledkom spracovania je hodnota matice homografie (`mHomogrMat`), ktorú využívam pri transformácii bodov ohraničujúcich polygónov textu v obraze.

Výpočty spojené s procesom sledovania pohybu snímok a odvodenia matice homografie sú implementované v native časti kódu, vo funkcii nazvanej `OpticalFlow(…)`. Parametrami volania sú adresy matice obrázku predošlého a aktuálneho snímku, adresa matice pre novú pozíciu sledovaných bodov, adresa matice homografie a príznak boolean indikujúci, či je snímka *inicializačná* (zdrojová snímka pre proces spracovania obrazu na text). V prípade, že matica predošlého snímku nie je platná (pri volaní funkcie na inicializačný snímok), zavolá sa funkcia s novou prázdnou maticou na tejto pozícii.

Práca v metóde `OpticalFlow(…)` sa vždy začína prevedením matice aktuálnej snímky do reprezentácie obrazu v stupňoch šedej. Prevod vykonávam pomocou OpenCV prevodovej funkcie `cvtColor(…)` s parametrom `CV_BGRA2GRAY`.

Následne je potrebné zistiť pomocou hodnoty posledného argumentu, či je táto snímka inicializačná. Ak áno, stačí v nej vyhľadať body s dobrými vlastnosťami na sledovanie pohybu. Výpočet optického toku a matice homografie z tejto snímky nevykonávam, keďže polohu bodov nemám s čím porovnať. Extrakciu vhodných bodov som implementoval pomocou OpenCV funkcie `goodFeaturesToTrack(…)`[18]. Jej prvý argument je matica spracovávaného snímku, druhým je matica do ktorej sú po behu funkcie vložené výsledné body. Ďalšími parametrami špecifikujem želaný maximálny počet nájdených bodov (`500`), želanú kvalitu nájdených bodov (`0,01`), ich minimálnu vzdialenosť (`10`), želanú obrazovú masku (nevyužívam) a priemernú veľkosť bloku (`3`). Tiež je možné špecifikovať, či sa pri spracovaní použije Harrisov detektor bodov (`true`) a jeho presnosť (`0,04`). V prípade potreby je možné ľahko zmeniť charakter sledovaných bodov zmenou alebo výmenou tejto funkcie.

Ak spracovávaná snímka nie je *inicializačná*, sú pri volaní funkcie `OpticalFlow(…)` definované hodnoty maticových argumentov obrázka predošlej snímky a pozície v nej sledovaných bodov. Pomocou volania OpenCV funkcie `calcOpticalFlowPyrLK(…)` najskôr vypočítavam novú pozíciu sledovaných bodov v aktuálnom snímku. Parametrami funkcie sú obrázky predošlej a aktuálnej snímky, matica polôh sledovaných bodov v predošlej snímke, matica pre vypočítané polohy bodov v aktuálnej snímke a matica označujúca chybné korešpondencie bodov. Ďalšími parametrami sa nastavujú vlastnosti výpočtu ako napríklad veľkosť vyhľadávacieho okna pre každý bod (`Size(10,10)`), maximálna hĺbka použitých obrazových pyramíd (3), ukončovacie kritéria

[18] http://docs.opencv.org/modules/imgproc/doc/feature_detection.html#goodfeaturestotrack

algoritmu. Presnejší popis metódy je možné získať zo stránok dokumentácie knižnice OpenCV[19]. Po výpočte optického toku je potrebné odhadnúť výslednú maticu homografie. Tento výpočet som implementoval pomocou volania OpenCV funkcie `findHomography(…)`[20]. Vstupnými parametrami algoritmu sú matice pozícií sledovaných bodov v predošlom a aktuálnom snímku. Argumentom je možné tiež definovať spôsob vyhľadávania ich korešpondencie (`CV_RANSAC`). Výsledkom volania je matica homografie dvoch snímok s rozmermi 3×3.

Posledným krokom spracovania native funkcie `OpticalFlow(…)` je nastavenie vypočítaných hodnôt matíc na adresy im korešpondujúcich matíc definovaných vstupnými argumentmi tejto funkcie.

Ak je naplnený atribút triedy `MainActivity` určujúci zoznam rozpoznaných slov z predchádzajúcej inicializačnej snímky, v jej metóde `onCameraFrame(…)` sú pre každý snímok volané procesy transformácie ich ohraničujúcich polygónov. Operácie prebiehajú v metóde `transformWordPositions(…)`. Vstupnými parametrami sú zoznam nájdených slov a transformačná matica homografie. Samotnú transformáciu kontrolných bodov obdĺžnikov spúšťam pomocou volania OpenCV metódy `perspectiveTransform(…)`. Vykreslenie takto pozmenených obdĺžnikov do matice aktuálnej snímky sa deje pomocou volania native funkcie `DrawPolygon(…)`, využívajúcej OpenCV volania vykresľovania základných geometrických primitív.

4.5 Trénovanie modelu klasifikácie

Na trénovanie a testovanie natrénovaného SVM modelu pre klasifikáciu nájdených MSER oblastí som si vytvoril samostatnú aktivitu `SVMTrainDataTool`. Tá však slúži len ako užívateľské rozhranie pre volania korešpondujúcich native metód. Metódy obsluhujú proces trénovania, testovania a vytvárania dát zo zdrojových obrázkov. Na túto aktivitu sa užívateľ vo finálnej verzii aplikácie zo základného pohľadu nedostane.

Dátové sady použité pri trénovaní a testovaní bližšie predstavuje kapitola 3.1.2. V tej istej kapitole je uvedený aj popis príznakov MSER oblastí použitých pri ich klasifikácií a proces ich extrakcie. Pri trénovaní a klasifikácií som využil OpenCV triedu `CvSVM` reprezentujúcu model SVM klasifikátora. Na samotné trénovanie modelu som využil volanie jej metódy `CvSVM.train_auto(…)`.

Pri volaní metódy je potrebné nastaviť viacero povinných trénovacích parametrov. Prvým parametrom je typ trénovaného klasifikátoru. Jeho hodnota bližšie špecifikuje charakter trénovacích dát a spôsob ich rozdelenia klasifikátorom. V mojom riešení som použil model typu `NU_SVC`. Ten obecne dokáže rozdeliť dáta do n tried a počíta s možnosťou ich nepresného rozdelenia (chyba trénovacieho vzorku). Pre tento typ klasifikátora je potrebné ešte určiť hodnotu parametra

[19] http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html
[20] http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

nu s možným rozsahom od nula po jedna. Parameter udáva hornú hranicu chyby a spodnú hranicu podielov podporných vektorov vzhľadom k celkovému počtu trénovacích vzoriek. Čím je jeho hodnota vyššia, tým bude rozhodovacia hranica hladšia. Experimenty ukázali, že pre dané dáta je najvhodnejšie použiť hodnotu 0,23.

Druhým nastavovaným parametrom trénovania je typ jadrovej funkcie klasifikátoru. Na výber je lineárne jadro, polynomiálne jadro, RBF (*radial basis function*), alebo sigmoidné jadro. V mojom riešení som zvolil jadrovú funkciu RBF, ktorá je odporúčaná pre väčšinu riešených problémov.

Metóda `train_auto(…)` počas svojho behu automaticky vykonáva proces cross validácie natrénovaného modelu. Trénovaciu sadu rozdelí do menších skupín, ktorých počet určuje parameter `k_fold`. Pomocou jednej podmnožiny model testuje a ostatné slúžia na jeho trénovanie a vylepšenie jeho parametrov. V prípade, že je vykonávaná klasifikácia len do dvoch tried, je možné pomocou parametru `balanced` nataviť lepšie vyváženie dát vo vytvorených podmnožinách. Znamená to, že veľkosti zastúpenia tried v jednotlivých podmnožinách sú čo najbližšie ich zastúpeniu v celej trénovacej množine.

5 Testy

Vytvorenú aplikáciu som testoval z viacerých hľadísk. Väčšina testov bola zameraná na zistenie postupov, ktoré by mohli zlepšiť výsledky spracovávania textu, keďže táto oblasť sa ukázala pre moje riešenie kritickou. Popri implementácii som tiež vykonával experimenty s rôznymi časťami postupu a nastavením ich kľúčových parametrov, zamerané na získanie čo najpresnejších výsledkov v čo najkratšom čase.

Nasledujúci text stručne popíše vykonávané testy, ich metódy a výsledky. Prvou podkapitolou je test vhodného vstupu spracovania obrazu. Druhá podkapitola má číslo 5.2 a rozoberá testy modelu klasifikácie MSER oblastí. Podkapitola s číslom 5.3 informuje o teste detekcie textu v obraze. Nasleduje podkapitola s číslom 5.4 rozoberajúca rozpoznávanie textu zo získaných obrazov hypotéz.

5.1 Vstup spracovania obrazu

Prvá séria testov mala za cieľ odhaliť najvhodnejšiu formu vstupu pre spracovanie obrazu na text. Experiment som popísal aj v kapitole 3.1.1 . Sadu 20 testovacích obrázkov z internetu a vlastných fotografií reálnych scén som rozložil na jednotlivé farebné kanály, nad ktorými som následne spúšťal detekciu MSER oblastí s rôznymi parametrami. Výsledné oblasti som ručne triedil. Určoval som počet písmen reprezentovaných v danom kanáli MSER oblasťami. Výsledky prezentuje tabuľka č. 6. Ako sa ukázalo neskôr, výsledky značne záležali na povahe testovanej snímky a povahe zašumenia textu. Nakoniec som sa rozhodol použiť pri implementácií kombináciu kanálov *saturation* a *value* z HSV farebnej reprezentácie obrázku. Táto kombinácia ponúka dobrý kompromis medzi počtom potrebných behov detekcie (2) a percentom nájdených znakov. Zaujímavosťou bolo, že oblasti písmen nájdené v kanáli *hue* obsahovali oproti tým istým písmenám v iných kanáloch toho istého obrázku veľké množstvo nepresností, šumu a často aj rozostrené okraje. Ilustruje to obrázok č. 19.

(a)

(b)

(c) 

Obrázok 19: MSER oblasti z rovnakého detektoru z rôznych farebných kanálov obrázku reprezentujúce tie isté písmená. (a) Oblasti z kanálu *hue*. (b) Oblasti z kanálu *saturation*. (c) Oblasti z kanálu *value*.

Farebné kanály	Percento nájdeného textu
R	81,5 %
G	83,7 %
B	82,8 %
H	67,3 %
S	70,1 %
V	85,4 %
R + G + B	92,6 %
H + S + V	93,1 %
G + B	89,4 %
S + V	91,2 %

Tabuľka 6: Percento nájdeného textu vo forme MSER oblastí podľa farebných kanálov.

5.2 Klasifikácia MSER oblastí

Druhou skupinou testov boli testy natrénovaných modelov klasifikátora MSER oblastí. Z testovacej skupiny fotografií ICDAR 2003 datasetu[21] (251 obrázkov) som extrahoval všetky MSER oblasti v každom používanom farebnom kanáli (saturation, value). Tie som ručne triedil na písmená a oblasti nepredstavujúce písmená. Výsledná testovacia sada obsahovala spolu 75 006 oblastí, z toho bolo 10 081 písmen. Vzorky testovacej sady vyzerali obdobne ako vzorky trénovacej sady, ktorej ukážku prezentuje obrázok č.10 v kapitole 3.1.2. Presnosť rozpoznávania natrénovaného modelu a som určil nasledovne:

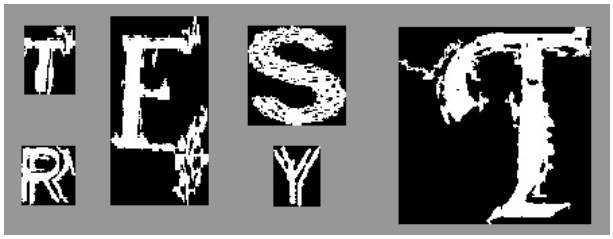
$$a = \frac{TP + TN}{S} \cdot 100$$

- TP (true positives) = počet správne identifikovaných písmen.
- TN (true negatives) = počet správne identifikovaných nepísmen.
- FP (false positives) = počet písmen nesprávne určených ako nepísmená.
- FN (false negatives) = počet nepísmen nesprávne určených ako písmená.

[21] http://algoval.essex.ac.uk/icdar/Datasets.html

- S = počet vzorkov celkovo (všetkých písmen aj nepísmen spolu).

Pri riešení projektu som skúšal rôzne nastavenia trénovania modelov, ako aj trénovanie modelov s použitím rôznych podmnožín sady trénovacích dát. Vyskúšal som modely trénovať na rôzne veľkých množinách oblastí získaných z rôzneho počtu snímok. Pri ručnom triedení niektorých oblastí bolo občas ťažké rozhodnúť do ktorej triedy by ich klasifikátor mal zaradiť. Boli to napríklad MSER oblasti písmen s nepresnými okrajmi. Podobnou kategóriou boli oblasti obsahujúce veľké množstvo malých dier, ktoré však v globálnom hľadisku neznižovali čitateľnosť písmena ako takého. Ukážku nejednoznačných oblastí prezentuje obrázok č. 20. Vyskúšal som preto tiež trénovať model na špeciálnej množine neobsahujúcej takéto sporné oblasti. Táto množina obsahovala spolu 4000 jednoznačných písmen a 4000 jednoznačných nepísmen. Výsledky prezentuje tabuľka č. 7. V riešení som napokon použil model s najväčšou presnosťou.



Obrázok 20: Nejednoznačné MSER oblasti.

Trénovacia sada modelu	**True positives (TP)**	**True negatives (TN)**	**False positives (FP)**	**False negatives (FN)**	**Presnosť (a)**
120 snímok	5 860	52 015	4 221	12 910	77,16 %
180 snímok	5 166	52 932	4 915	11 993	77,46 %
258 snímok	**5 390**	**54 569**	**4 691**	**10 356**	**79,93 %**
jedn. MSERy	7 777	48 125	2 304	16 800	74,53 %

Tabuľka 7: Testy natrénovaných modelov. Použitý model je zvýraznený hrubo.

Z výsledkov vyplynulo, že veľkosť a charakter trénovacej sady nemajú na kvalitu výsledného modelu zásadný rozhodujúci vplyv. Oveľa viac záleží na charaktere zvolených príznakov oblastí. Použitý model je schopný správne zaradiť niečo viac ako 3/4 všetkých MSER oblastí. Tento výsledok je pre algoritmus použiteľný, keďže v ďalších krokoch spracovania sa pokúšam výsledok klasifikácie vylepšiť. Nesprávne klasifikované oblasti sa snažím preradiť do správnej množiny v kroku skladania slov.

5.3 Detekcia textu

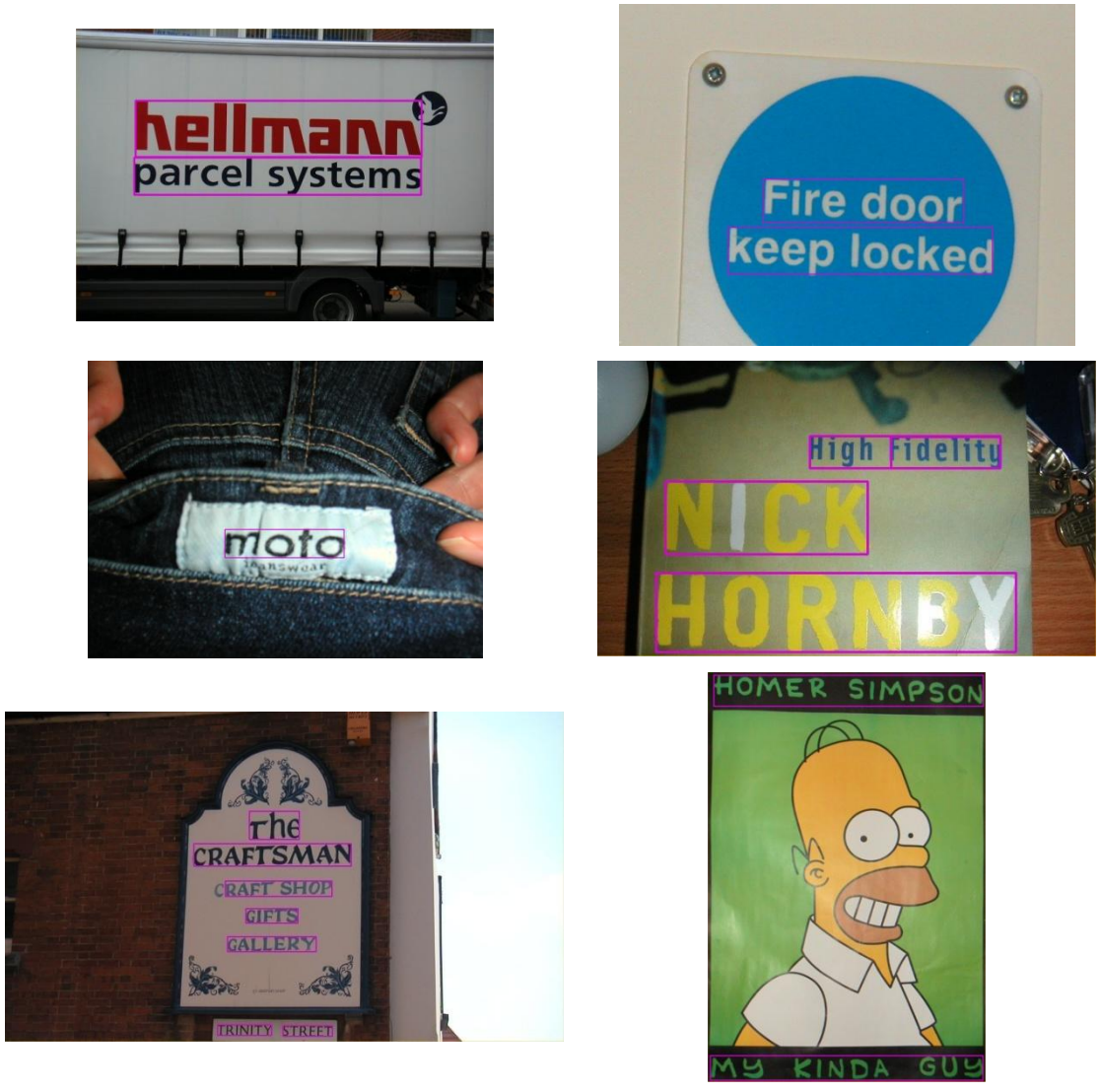
Na otestovanie navrhovaného algoritmu pre detekciu textu som opäť použil fotky zo sady ICDAR 2003. Do fotiek z testovacej množiny dátovej sady som nechal algoritmus vykresľovať ohraničujúce obdĺžniky hypotéz reprezentujúce nájdené slová. Následne som v takto upravených fotkách ručne počítal správne a nesprávne nájdené resp. nenájdené slová. Program by mal v sade

spracovávaných fotografií (251) v ideálnom prípade vyhľadať 849 slov. Výsledky testu prezentuje tabuľka číslo 8. Za správne nájdené som označil slovo, ktoré bolo celé vnútri ohraničujúceho obdĺžnika tohto slova, alebo jeho riadku. Čiastočne nájdenému mohlo chýbať zopár písmen, no väčšina slova musela byť v danom obdĺžniku.

	Počet	Percento z celku
Správne nájdených celých slov	620	73,03 %
Čiastočne nájdených slov	79	9,30 %
Nenájdených slov	150	17,68 %
Nájdené niečo iné ako slovo	186	

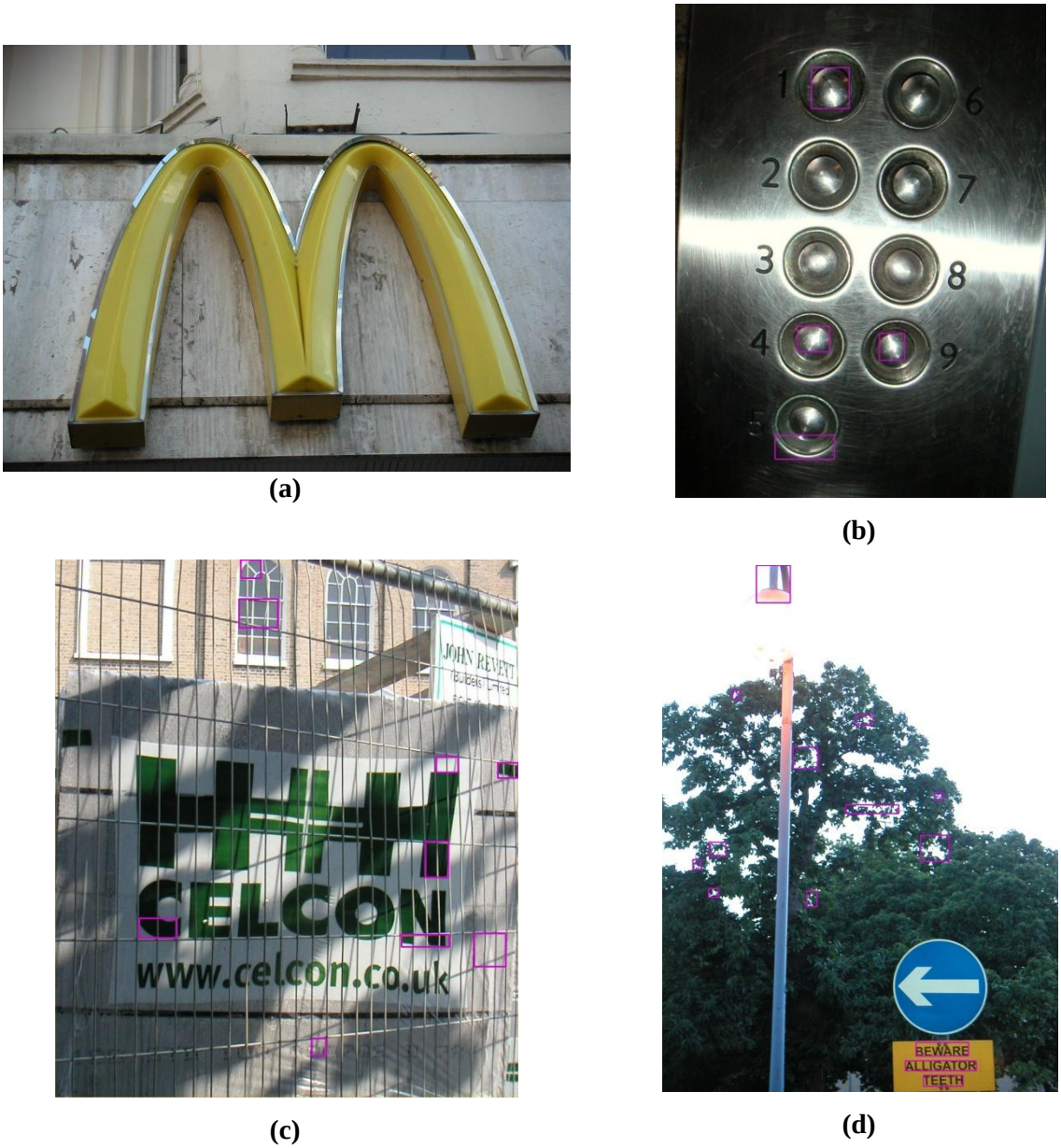
Tabuľka 8: Výsledky testu vyhľadania textu v obraze.

Detekcia dosiahla v celku uspokojivé výsledky. V obrázkoch je väčšina textu nájdená celkovo, alebo čiastočne. Ak je text hlavným elementom obrázku, je jeho detekcia takmer vždy úspešná. Ukážku správnych výstupov testovania prezentuje obrázok č. 21.



Obrázok 21: Správne nájdené slová a riadky textu. Texty sú zvýraznené fialovým rámčekom.

Občas sa v testovacej sade však objavili fotky, na ktorých má text nepodporovanú formu (obsahuje málo písmen, text je príliš malý a pod.). Na niektorých fotografiách sa dokonca žiaden text nenachádza, obsahujú len objekty pripomínajúce textové tvary. Vo väčšine takýchto prípadov proces správne nenašiel nič. Problémom však boli najmä fotky obsahujúce koruny stromov a iné veľmi komplexné pozadia. V nich algoritmus väčšinou chybne pospájal do slov nezmyselné MSER oblasti. Tesseract by ale takéto poskladané slovo nevedel správne interpretovať, preto by bola hypotéza riadku s veľkou pravdepodobnosťou vyradená filtráciou po kroku rozpoznávania. Do istej miery boli problémom aj texty, ktoré nie sú od svojho pozadia dostatočne kontrastné a neostré texty s rozmazaným okrajom. Ukážku problémových obrázkov prezentuje obrázok č. 22.



(a) **(b)** **(c)** **(d)**

Obrázok 22: Ukážka problémových vstupov. Hypotézy nájdené ako text sú zvýraznené fialovým obdĺžnikom. (a)(b) Neplaté texty. (c)(d) Komplexné pozadie textu.

Testovanie tiež ukázalo, že presnosť procesu zhlukovania MSER oblastí je do veľkej miery závislý na vhodnej podmienke horizontálnej vzdialenosti pripájanej oblasti od hypotézy riadku. Najskôr som vyskúšal implementovať podmienku horizontálnej vzdialenosti prezentovanú autormi východiskovej prace. Tí najprv prechádzali všetky MSER oblasti v obrázku zľava doprava a na rozhodnutie používali vzdialenosť ľavého okraja pripájanej oblasti od pravého okraja skladaného riadku. Pri testovaní tento spôsob vykazoval viacero problémov. Okrem nutnosti radiť oblasti zľava doprava sa tiež často stávalo, že úzke písmená na okrajoch a vnútri slova boli z hypotézy vynechané. Neúplná hypotéza predstavuje problém pre presnosť kroku rozpoznávania. Implementoval som preto vlastnú horizontálnu podmienku popísanú v tabuľke č. 3 v kapitole 3.1.3. Tá pracuje výrazne lepšie, no nie je takisto ideálna. Porovnávam v nej vzdialenosť stredov riadku a pripájanej oblasti. Počas pripájania počítam počet medzier medzi pripojenými slovami riadku a pritom odhadujem ich veľkosť. Problémom sa stávajú dlhé slová, kedy sa chyba výpočtu naakumuluje do tej miery, že sú často pripájané aj oblasti, ktoré sú horizontálne od riadku ďaleko. Rovnaký problém nastáva, aj keď je k riadku pripojená oblasť vnútri inej (napr. v niektorých prípadoch je MSER oblasťou aj vnútro písmen O, P, R). S mojou podmienkou sú vytvorené hypotézy vo väčšine prípadov správne a kompletné. Skladanie slov tiež záleží od nastavenia zvoleného detektoru. V prípade, že je príliš citlivý, môžu výsledné hypotézy obsahovať množstvo zbytočných MSER oblastí, ktoré ju buď diskvalifikujú pri podmienke počtu písmen verzus nepísmen, alebo spôsobujú problémy neskôr pri rozpoznávaní textu.

5.4 Rozpoznávanie textu

Proces rozpoznávania textu som síce netestoval systematicky, no pri celkovom testovaní aplikácie sa ukázalo viacero problémov v tejto oblasti. Použité rozhranie implementujúce systém Tesseract bolo na mojom testovacom zariadení za určitých okolností nestabilné. Aplikáciu som testoval na mobile *Samsung Galaxy S II* (Dual-core 1.2 GHz, 1GB RAM, Android v4.0.4). Nepodarilo sa mi však izolovať a zistiť čo konkrétne spôsobuje problémy. Knižnica pri rozpoznávaní niektorých obrázkov hypotéz jednoducho padla bez akéhokoľvek špecifického hlásenia chyby. Určite to nebolo nedostatkom operačnej pamäte, skôr to po trasovaní vyzeralo na skrytú chybu vnútri frameworku pri komunikácií s jeho native objektmi. Skúšal bez väčších výsledkov taktiež rôzne typy vstupných obrázkov, meniť ich veľkosti, rozlíšenie, farebnú škálu. V prípade, že kvôli tejto chybe vlákno spracovania textu spadne, pokúsim sa ho opäť spustiť skôr ako bude aplikácia zastavená pre čakanie na výsledok zombie procesu.

Druhým problémom je samotná presnosť rozpoznania extrahovaných slov. Často nie je zaručené stopercentné rozpoznanie ani v prípade, ak sú od seba znaky slova oddelené dostatočnými medzerami a ich jednotlivé časti nesplývajú. Skúšal som použitie opravy rozpoznania pomocou viacerých jazykových mutácií slovníku systému Tesseract, no výsledky neboli vo väčšine prípadov

omnoho lepšie. Rozpoznanie sa mi podarilo trochu vylepšiť pridávaním okraja do obrázku hypotézy. Ukážky vstupov a ich korešpondujúcich výstupov rozpoznania prezentuje tabuľka č. 9. Presnosť by bolo ešte možné zlepšiť použitím inej ako prednastavenej metódy rozpoznania. Experimenty ukázali, že algoritmus v takomto prípade padá v približne 90% prípadoch, čo pri spracovaní videa znamená zhruba 2 sekundy. Použil som teda prednastavenú metódu, ktorá by mala byť oproti ostatným rýchla no menej presná. Často stojí za zlým rozpoznaním nesprávne zložené slovo, alebo ak daná hypotéza obsahuje zbytočné oblasti nereprezentujúce textové znaky. To však záleží od MSER detektora a teda charakteru vstupných obrázkov. Ak sú snímky videa neostré, je veľká pravdepodobnosť, že aj rozpoznanie nebude fungovať správne. Implementoval som pre to možnosť meniť formu zaostrovania kamery a možnosť použiť vbudovaný blesk ako reflektor osvetľujúci scénu. Toto nastavenie je však veľmi náročné na výdrž batérie zariadenia. Užívateľ môže meniť nastavenia pomocou volieb menu z aktivity Options.

Obrázok hypotézy, vstup rozpoznávania.	Výstupný rozpoznaný text. (presnosť vrátená knižnicou Tesseract)
	`Kakaove kulzéky` (76%)
	`Kakaove kulncky` (78%)
	`ClVr` (79%)
	`SUMMER SCHQQL` (75 %)
	`8AEWC GLASq` (48 %)

Tabuľka 9: Ukážka vstupov poznačených rôznou chybou a korešpondujúcich výstupov rozpoznávania pomocou systému Tesseract.

6 Záver

Cieľom projektu bolo zrealizovať mobilnú aplikáciu spracovávajúcu obraz z kamery zariadenia. V obraze bolo potrebné lokalizovať a rozpoznať texty do digitálnej podoby. Následne bolo potrebné výsledky rozpoznania vhodne zobraziť v užívateľskom rozhraní a zvolený textový reťazec interpretovať ako hyperlink internetového vyhľadávania.

Vytvorený prototyp aplikácie implementuje detekciu a rozpoznanie textu z videa reálnych scén. Detekciu slov a písmen som testoval na sade fotografií z reálnych scén ICDAR 2003. Pri práci s grafickými dátami a obrazovými operáciami som využíval metódy knižnice OpenCV. Hlavné riadenie aplikácie je implementované v jazyku Java, výpočetne náročné úlohy v jazyku C++.

Metóda spracovania obrazu na text koncepčne vychádza z metódy publikovanej L. Neumannom. Pri detekcii písmen využíva maximálne stabilné extrémne oblasti obrazu. Jednotlivé kroky metódy boli pozmenené s ohľadom na použitie na mobilnom zariadení. Autor uvádza priemerne trvanie pôvodnej metódy v rade desiatok sekúnd, pri použití na počítači. Jeden beh spracovania textu mnou modifikovanej metódy beží na testovacom mobilnom zariadení, na snímkach s rozlíšením 800x480 bodov, približne 3 až 5 sekúnd. Oproti publikovanej metóde však nedosahuje také presné a kvalitné výsledky nájdenia a rozpoznania textu. Presnosť pôvodnej metódy sa blížila ku 100%. Domnievam sa, že vhodnou zmenou príznakov používaných pri klasifikácií oblastí by sa dalo výsledky algoritmu výrazne zlepšiť. Zlepšenie algoritmu predpokladám tiež pri zmene heuristiky rozhodujúcej o zhlukovaní oblasti do slov a ich validačných podmienok. Ďalším problémom bolo rozpoznávanie slov pomocou OCR systému Tesseract. To bolo často nepresné a na mobilnom zariadení nestabilné. V budúcnosti by som zvážil použitie inej rozpoznávacej metódy.

V prototype aplikácie som tiež implementoval postup na sledovanie a zobrazovanie nájdených textov priamo vo videu z kamery zariadenia. Chcel som tým zlepšiť interaktivitu a atraktivitu výslednej aplikácie. Umožnilo mi to tiež vyriešiť problémy spojené s rozdelením procesu spracovania textu a zobrazovania výstupu z kamery do samostatne bežiacich vlákien.

Okrem vyššie spomínaných zlepšení vidím priestor na vylepšenie praktickosti vytvoreného prototypu aplikácie implementáciou spracovania statických snímok, kedy by mohla extrakcia textu z obrazu trvať aj dlhšie. Mohli by byť použité náročnejšie techniky lokalizácie a rozpoznania slov, ktoré by vykazovali lepšiu kvalitu výstupu. Okrem nastavení fotoaparátu zariadenia by bolo tiež možné umožniť užívateľovi špecifikovať vlastný formát vyhľadávacieho linku, mohol by teda využiť veľmi široké spektrum rôznych internetových služieb. Aplikácia by mohla byť použitá napríklad pre rýchle a ľahké vyhľadávanie informácií o výživových hodnotách potravín v obchode alebo na jedálnom lístku reštaurácie. Pri použití vyhľadávania v internetových prekladových službách sa z aplikácie ľahko stane mobilný tlmočník.

Riešením projektu som si veľmi rozšíril obzory v poli OCR a v téme detekcie textu v obraze.

Literatúra

[1] Vincent, L.: Google book search: Document understanding on a massive scale. In *ICDAR '07: Proceedings of the 9. International Conference on Document Analysis and Recognition*, Washington, D.C., 2007, s. 819 – 823.

[2] Lin, X.: Reliable OCR solution for digital content re-mastering. In *Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, 2001, s. 223 – 231.

[3] Kasar, T.; Kumar, J.; Ramakrishnan, A. G.: Font and background color independent text binarization. In *Proceedings 2nd International Workshop Camera-Based Document Anal. Recognition*, 2007, s. 3 – 9.

[4] Tran, H.; Lux, A.; Nguyen, H. L.; Boucher, A.: A novel approach for text detection in images sing structural features. In *Proceedings of 3rd Interational Conference Adv. Pattern Recognition*, 2005, s. 627 – 635.

[5] Liu, Q.; Jung, C.; Moon, Y.: Text segmentation based on stroke filter. In *Proceedings of the International Conference Multimedia*, 2006, s. 129 – 132.

[6] Sobottka, K.; Kronenberg, H.; Perroud, T.; Bunke, H.: Text extraction from colored book and journal covers. In *Proc. 10th Int. Conf. Document Anal. Recognit.*, 1999, s. 163 – 176.

[7] Park, C. J.; Moon, K. A.; Oh, W. G.; Choi, H. M.: An efficient extraction of character string positions using morphological operator. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, 2000, s. 1616 – 1620.

[8] Wolf, C.; Jolin, J. M.; Chassaing, F.: Text localization, enhancement and binarization in multimedia documents. In *Proc. Int. Conf. Pattern Recognit.*, 2002, s. 1037 – 1040.

[9] LeBourgeois, E.: Robust Multifont OCR System from Gray Level Images. In *Proceedings of the 4th Int. Conference on Document Analysis and Recognition*, 1997, s. 1 – 5.

[10] Chucai , Y.; Tian, Y.: Text String Detection From Natural Scenes by Structure-Based Partition and Grouping. In *IEEE Trans. Image Processing*, 2011.

[11] Neumann, L.: *Vyhledání a rozpoznání textu v obrazech reálných scén.* [diplomová práca], České vysoké učení technické v Praze, Fakulta elektrotechnická, 2010.

[12] Android API [online]. Dostupné z WWW: < http://developer.android.com/guide/components/index.html >, [cit. 2013-04-03].

[13] Android Architecture [online]. Dostupné z WWW:
< http://developer.android.com/about/versions/index.html >, [cit. 2013-04-03].

[14] Android Activity Lifecycle [online]. Dostupné z WWW:
< http://developer.android.com/reference/android/app/Activity.html >, [cit. 2013-04-03].

[15] Android Version Statistics [online]. Dostupné z WWW:
< http://developer.android.com/about/dashboards/index.html >, [cit. 2013-05-10].

[16] Neumann, L.; Matas, J.: Real-Time Scene Text Localiation and Recognition. *Conference on Computer Vision and Pattern Recognition*, 2012.

[17] Matas, J.; Chum, O.; Martin, U.; ai.: Robust wide baseline stereo from maximally stable extremal regions, [online]. Dostupné z WWW:
< http://cmp.felk.cvut.cz/matas/papers/matas-bmvc02.pdf >, 2002, [cit. 2013-05-11].

[18] Forssén, P-E.: Maximally Stable Colour Regions for Recognition and Matching. *Conference on Computer Vision and Pattern Recognition*, 2007.

[19] Pratt, W. K.: *Digital image processing*. Willey-interscience publication, John Wiley & Sons, INC., 2001, iSBN 0-471-22132-5.

[20] Bradski, G.; Kaehler, A.: *Learning OpenCV*. O'Reilly, 2008, iSBN 978-0-596-51613-0.

[21] Harris, C.; Stephens, M.: A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, 1988, s. 147 – 151.

[22] Shi, J.; Tomasi, C.: Good features to track. *IEEE Conference on Computer Vision and Pattern Recognition*, Jún 1994.

[23] Lowe , D. G.: Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004, s. 91 – 110.

[24] Horn, B. K. P.; Schunck, B. G.: Determining optical flow. *Artificial Intelligence*, 1981, s. 185–203.

[25] Lucas, B. D.; Kanade, T.: An iterative image registration technique with an application to stereo vision. In *Proceedings of the 1981 DARPA Imaging Understanding Workshop*, 1981, s. 121 – 130.

[26] Hartley, R.; Zisserman, A.: *Multiple View Geometry in Computer Vision*. Cambridge University Press, vydanie druhé, 2003, iSBN 0-521-54051-8.

[27] Fischler, M. A.; Bolles, R. C.: *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*. Communications of the ACM, ISSN 0001-0782.

[28] Chum, O.; Matas, J.; Kittler, J.: Locally Optimized RANSAC. In *Pattern Recognition, Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2003, ISBN 978-3-540-40861-1.