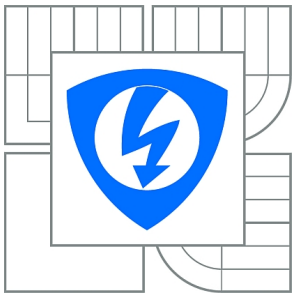

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY


FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION


# ZPRACOVÁNÍ OBRAZU V SYSTÉMU ANDROID - DETEKCE A ROZPOZNÁNÍ SPZ/RZ A VYUŽITÍ EXTERNÍ DATABÁZE ZÁJMOVÝCH VOZIDEL

IMAGE PROCESSING USING ANDROID - LPR WITH EXTERNAL DATABASE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS


AUTOR PRÁCE — Bc. PETER MOLČÁNY

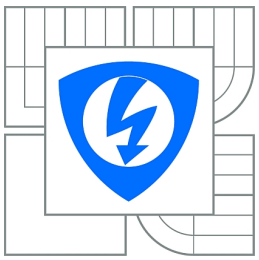
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETER HONEC, Ph.D.

SUPERVISOR



BRNO 2015



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**Fakulta elektrotechniky a komunikačních technologií**

**Ústav automatizace a měřicí techniky**

# Diplomová práce

magisterský navazující studijní obor
**Kybernetika, automatizace a měření**

| | | | |
|---|---|---|---|
| ***Student:*** | Bc. Peter Molčány | ***ID:*** | 134563 |
| ***Ročník:*** | 2 | ***Akademický rok:*** | 2014/2015 |

**NÁZEV TÉMATU:**

## Zpracování obrazu v systému Android - detekce a rozpoznání SPZ/RZ a využití externí databáze zájmových vozidel

**POKYNY PRO VYPRACOVÁNÍ:**

Cílem práce bude vytvoření GUI a algoritmů detekce a rozpoznání RZ automobilů a vyhledávání rozpoznané SPZ/RZ v online/offline databázi zájmových vozidel.
1. Zpracujte rešerši týkající se OS Android a základních metod zpracování obrazu a komunikaci s MySQL.
2. Navrhněte a vytvořte GUI aplikaci, která bude zajišťovat spolupráci s HW prostředky zařízení, pořizovat obraz z kamery zařízení, spouštět algoritmy a získávat data z databáze.
3. Navrhněte a otestujte algoritmy pro detekci a rozpoznání RZ automobilů v reálné scéně.
4. Implementujte do aplikace pro OS Android a otestujte funkcionalitu.

**DOPORUČENÁ LITERATURA:**

Hlaváč, Šonka, Počítačové vidění.
Šonka, Hlávač, Boyle - IMAGE PROCESSING, ANALYSIS, AND MACHINE VISION.

| | | | |
|---|---|---|---|
| ***Termín zadání:*** | 9.2.2015 | ***Termín odevzdání:*** | 18.5.2015 |

***Vedoucí práce:*** Ing. Peter Honec, Ph.D.
***Konzultanti diplomové práce:***

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

**UPOZORNĚNÍ:**

## ABSTRAKT

Cieľom tejto diplomovej práce je vytvorenie aplikácie postavenej na platforme Android, slúžiacej k rozpoznaniu evidenčných čísel vozidiel s vyhľadávaním v externej databáze.
V úvodnej časti rozoberáme možnosti rozpoznania EČV vo všeobecnosti. V druhej kapitole sa zaoberáme popisom platformy Android a využitím potrebných vývojových nástrojov spolu s multiplatformovou knižnicou OpenCV, ktorú aplikácia využíva. V tretej kapitole sú uvedené rôzne typy databáz a synchronizačných postupov. Vo štvrtej kapitole rozoberáme spracovanie obrazu vo všeobecnosti a popis jednotlivých algoritmov potrebných k detekcii. V piatej kapitole definujeme požiadavky na aplikáciu, špeciálne požiadavky na scénu a hardvér. V šiestej kapitole je uvedený popis samotnej realizácie aplikácie a použitých algoritmov. Vyhodnotenie úspešnosti jednotlivých algoritmov je uvedené v siedmej kapitole. V záverečnej časti sú uvedené dosiahnuté výsledky a možnosti pokračovania práce.



## KĽÚČOVÉ SLOVÁ

rozpoznávanie evidenčných čísel vozidiel, android, opencv



## ABSTRACT

The aim of this Master's thesis is designing and developing Android application for automatic number plate recognition with external database lookup.
In the introduction we discuss possibilities of number plate recognition in general. Android platform fundamentals, necessary developer tools and multi-platform image processing library OpenCV are described in the second section. In the third section different database types and synchronization methods are introduced. In the fourth section we describe basics of image processing and different algorithms necessary for recognition. Application requirements, involving scene and hardware requirements are defined in the fifth section. In the sixth section application development and algorithm implementation is described. In the seventh section we evalute the results of the detection. In the last section results are summarized and goals are set for further improvement.



## KEYWORDS

automatic number plate recognition, android, opencv

MOLČÁNY, Peter *Zpracování obrazu v systému Android - detekce a rozpoznání SPZ/RZ a využití externí databáze zájmových vozidel*: diplomová práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2015. 63 s. Vedúci práce bol Ing. Peter Honec

# PREHLÁSENIE

Prehlasujem, že som svoju diplomovoú prácu na tému „Zpracování obrazu v systému Android - detekce a rozpoznání SPZ/RZ a využití externí databáze zájmových vozidel“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., o autorskom práve, o právach súvisiacich s autorským právom a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. dielu 4 Trestného zákoníka č. 40/2009 Sb.

Brno ..... .....

(podpis autora)

## POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce pánovi Ing. Petrovi Honcovi za odborné vedenie, konzultácie a návrhy k vylepšeniu práce.


Brno ..... .....

(podpis autora)

# OBSAH

# ZOZNAM OBRÁZKOV

# ZOZNAM TABULIEK

# 1 ÚVOD

Automatické rozpoznanie evidenčných čísel vozidiel hrá dôležitú úlohu v parkovacích systémoch, monitorovaní spoplatňovaných úsekoch ciest prostredníctvom mýtnych brán, uplatňovaní objektívnej zodpovednosti pri priestupkoch na pozemných komunikáciách a rôznych iných aplikáciách. Základom úspešného rozpoznania evidenčného čísla vozidla je kvalitný obraz, získaný z kamery, prípadne fotoaparátu. Zariadenia môžu zaznamenávať farebný, monochromatický alebo infračervený obraz, prípadne ich kombináciu. Tam kde sa predpokladá použitie s nedostatkom svetla, je potrebné použiť dodatočné osvetlenie, ktoré pracuje spolu so záznamovým zariadením na nastavení vhodných parametrov. Je tiež možné získať viac snímkov s rôznym nastavením a vybrať najvhodnejší.

Evidenčné čísla vozidiel Českej republiky sú navrhnuté tak, aby boli automatické systémy schopné ich rozpoznania. Sú vynechané písmená G, O, Q, V, ktoré by sťažovali ich rozpoznanie. Štandardný rozmer evidenčného čísla vozidla je 520 x 110 mm. Obsahuje spolu sedem alfanumerických znakov.

Problematika automatického rozpoznania evidenčných čísel vozidiel už bola spracovaná mnohokrát. Zvyčajne sa jedná o aplikácie pre PC, prípadne špeciálne zariadenia. Cieľom tejto práce je však vytvoriť systém, ktorý nepotrebuje žiaden špeciálny hardvér ale obyčajný smartfón so systémom Android, ktoré sú dnes v spoločnosti vo veľkej miere rozšírené.

Samotné rozpoznanie prebieha v niekoľkých fázach. Najprv je potrebné na snímke lokalizovať EČV, následne segmentovať jednotlivé znaky a potom použiť metódy rozpoznania znakov. Pre meranie úspešnosti rozpoznania je potrebné stanoviť si hraničné podmienky, v ktorých bude rozpoznanie prebiehať. Po rozpoznaní EČV aplikácia odošle požiadavku s EČV na server, aby sme mohli zistiť, či sa daná EČV nachádza v našej databáze alebo nie. Keďže prístup do databázy môže mať viacero zariadení naraz, rozširuje to možnosti použitia.

Používateľ aplikácie na rozpoznanie EČV v mobilnom zariadení sa môže dostať do lokalít mimo dosahu mobilného signálu, kedy nemá prístup do siete internet. Pre tieto situácie je vhodné uložiť si databázu lokálne a dopyty budú smerované sem, pokým sa pripojenie neobnoví. Užívateľ by mal mať možnosť túto databázu v prípade dostupného pripojenia aktualizovať.

# 2 PLATFORMA ANDROID

V tejto kapitole je uvedený popis platformy Android, vývojové nástroje pre túto platformu, vysvetlený princíp podpory starších verzií systému pomocou podporných knižníc a tiež možnosti spracovania obrazu na tejto platforme.

## 2.1 História

Firma Android, Inc. bola založená v roku 2003 v Palo Alto, Californii. Následne bola firma odkúpená v roku 2005 spoločnosťou Google. Prvou oficiálnou verziou operačného systému Android bol Cupcake (1.5).

| Kódové označenie | Číslo verzie | API |
|---|---|---|
| Cupcake | 1.5 | 3 |
| Donut | 1.6 | 4 |
| Eclair | 2.0 - 2.1 | 5-7 |
| Froyo | 2.2.x | 8 |
| Gingerbread | 2.3 - 2.3.7 | 9-10 |
| Honeycomb | 3.0 - 3.2.x | 11-13 |
| Ice Cream Sandwich | 4.0.1 - 4.0.4 | 15 |
| Jelly Bean | 4.1.x - 4.3.x | 16-18 |
| KitKat | 4.4 - 4.4.4 | 19 |
| Lollipop | 5.0 - 5.1 | 21-22 |

Tab. 2.1: História kódových označení systému Android [2]

V roku 2010 spoločnosť Google vydala prvý Nexus smartfón s Androidom Eclair (2.1), ktorý tiež významným spôsobom prispel k ďalšiemu rastu Androidu. V roku 2011 bolo už 77 miliónov aktívnych užívateľov tohto operačného systému v časovom okne 30 dní. Ďalší výrazný míľnik spoločnosť prezentovala na konferencii I/O 2014, kedy tento počet dosiahol jednu miliardu.

## 2.2 Úvod k platforme

Platforma Android je v súčastnosti najrozšírenejším operačným systémom v oblasti smartfónov a tabletov. Android je viac-užívateľský operačný systém postavený na Linuxovom systéme, kde je každá aplikácia separátnym užívateľom. Systém zabezpečuje oddelenie aplikácii pomocou odlišných identifikátorov užívateľov tak, že

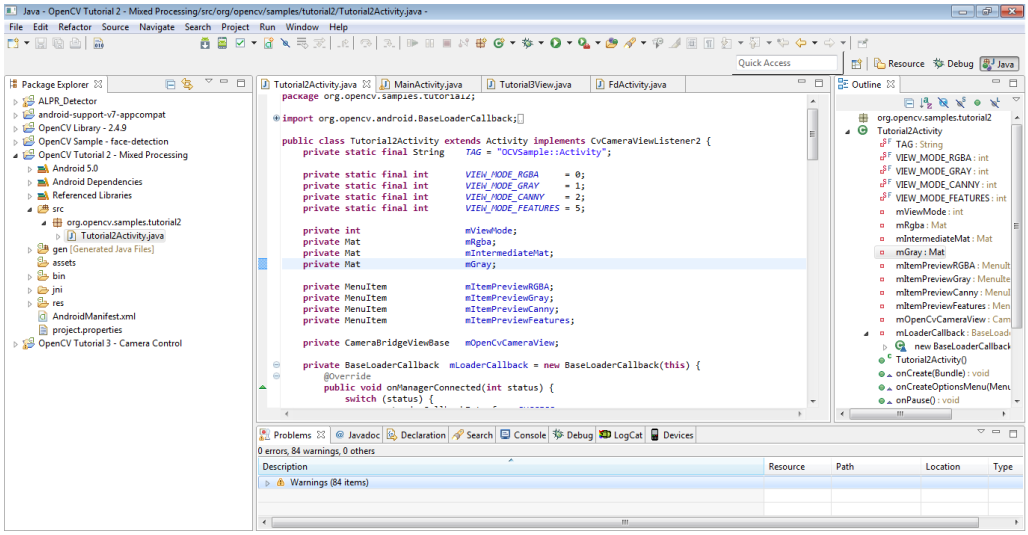
nastavuje práva všetkých súborov osobitne a prístup k nim má iba aplikácia s korešpondujúcim identifikátorom. Každý proces má tiež vlastný Virtual Machine, pomocou ktorého sú aplikácie oddelené. Ak sú však aplikácie podpísané rovnakým certifikátom, môžu vzájomne pristupovať k určeným súborom. Aplikácia tiež môže pristupovať iba ku komponentom, na ktoré má definované oprávnenie. Toto oprávnenie udeľuje užívateľ pri inštalácii aplikácie.

## 2.3 Vývojové nástroje pre platformu Android

Pre vývoj aplikácii pre platformu Android sú odporúčané dve možnosti. Prvou je prostredie Eclipse s pluginom ADT. Druhou možnosťou, ktorá bola oficiálne predstavená v Decembri 2014, je prostredie Android Studio. V ďalšej časti si popíšeme vývoj v oboch prostrediach.

### 2.3.1 Eclipse ADT plugin

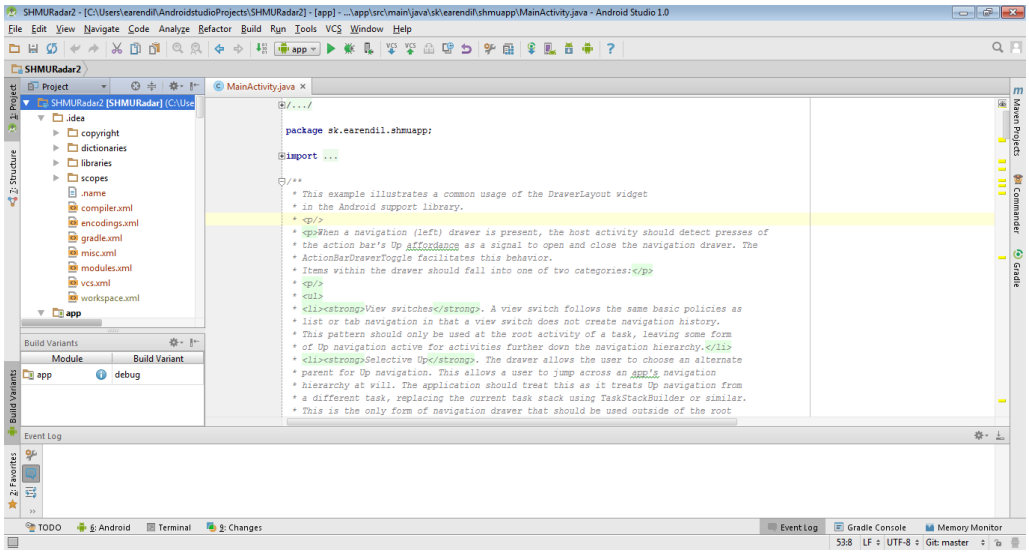
Eclipse bol donedávna oficiálnym vývojovým nástrojom pre platformu Android. Výhodou bola modularita a tiež šetrenie diskového priestoru, keďže bola potrebná iba jedna inštalácia programu Eclipse pre rôzne programovacie jazyky a platformy. Pre programovanie v NDK sa stále odporúča programovanie v programe Eclipse, napriek tomu, že jeho podpora bola ukončená. Na obrázku 4.4 je zobrazneý vývoj aplikácie s natívnym kódom.




Obr. 2.1: Prostredie Eclipse s ADT pluginom

### 2.3.2 Android Studio

Android Studio je momentálne oficiálnym vývojovým nástrojom pre platformu Android. Je postavené na IntelliJ IDEA. V porovnaní s Eclipse zjednodušuje mnohé úkony. Obsahuje flexibilný gradle build systém, potiahni a pusť funkcionalitu pre úpravu tém, podpisovanie aplikácií, podpora Google Cloud platformy a mnohé iné.




Obr. 2.2: Vývoj aplikácií v prostredí Android Studio

## 2.4 Podpora rôznych verzií API, podporné knižnice

Pre spätnú kompatibilitu so staršími verziami Androidu, je potrebné integrovať podporné knižnice. Každá knižnica má definovanú požiadavku na minimálnu verziu, s ktorou môže pracovať. Takto môže vývojár pristupovať k novším API na starších verziách systému. Pre výber správnej podpornej knižnice, je potrebné poznať funkcie, ktoré bude aplikácia potrebovať a tiež funkcie, ktoré poskytujú podporné knižnice. Najpoužívanejšími sú *v4 support library* a *v7 appcompat library*.

### 2.4.1 Podporná knižnica v4 support library

Podporná knižnica v4 zabezpečuje spätnú kompatibilitu so zariadeniami s Androidom 1.6(API 4) a vyšším. Obsahuje podporu rôznych aplikačných komponentov a tiež komponentov používateľského rozhrania. Obsahuje napríklad podporu Fragmentov, ViewPager, DrawerLayout a iných bežne používaných komponentov. Knižnica je umiestnená v *<sdk>/extras/android/support/v4/*.

### 2.4.2 Podporná knižnica v7 appcompat library

Knižnica *v7 appcompat* poskytuje podporu pre ActionBar. Prináša tiež podporu pre *material design*. Preto ak chceme vytvoriť aplikáciu podporujúcu tento nový vizuálny štýl, ktorý bol uvedený v Androide Lollipop, je potrebné použiť túto podpornú knižnicu.

## 2.5 Súbor Manifest

Každá aplikácia musí obsahovať súbor s názvom *AndroidManifest.xml* vo svojom koreňovom priečinku. V tomto súbore je definovaný názov balíka aplikácie, ktorý slúži ako jedinečný identifikátor aplikácie, komponenty aplikácie ako služby, broadcast receivers, content providers, práva na prístup ku chráneným častiam API, požiadavku na minimálnu verziu Androidu a iné.

## 2.6 Aplikačné zdroje

Keďže systém Android nemá úzky výber zariadení, na ktoré by bol navrhnutý, ale je potrebné z hľadiska vývojárov podporovať odlišné typy zariadení od rôznych výrobcov s rozdielnym hardvérom, sú aplikačné zdroje udržiavané externe. Takto je možné vytvoriť štandardný aplikačný zdroj a alternatívne zdroje, ktoré sa použijú podľa typu zariadenia. Navrhne sa napríklad užívateľské rozhranie pre mobilný telefón a tiež pre tablet a počas behu aplikácie sa vyberie ten komponent, ktorý prislúcha zariadeniu, na ktorom bola aplikácia spustená. Podobne sa vytvárajú aj jazykové mutácie aplikácie, pričom sa vyberie ten súbor s popismi komponentov, ktorý sa nachádza v priečinku danej jazykovej mutácií.

## 2.7 NDK

Ak je potrebné do aplikácie integrovať kód v natívnom jazyku(C, C++), sa môže použiť NDK. Táto možnosť je tiež využívaná, ak má byť aplikácia multiplatformová a určité knižnice sú napísané v natívnom jazyku. Pre bežný vývoj aplikácii však NDK nie je určené. Jeho zaradenie sa odporúča pri operáciách, ktoré vyžadujú nadmerné zaťaženie procesora ako napríklad jadrá hier, spracovanie signálov, simulácie fyzikálnych modelov a iné.

## 2.8 OpenCV

OpenCV je multiplatformová knižnica pre manipuláciu s obrazom pod licenciou BSD, dostupná pre akademické aj komerčné účely. Kód môže byť písaný v jazykoch C, C++, Python a Java a podporuje operačné systémy Windows, Linux, Android, Mac OS a iOS. OpenCV bola navrhovaná so zameraním na výpočtovú efektívnosť a výpočty v reálnom čase.

### 2.8.1 OpenCV4Android

Knižnica OpenCV je multiplatformová, ale táto práca sa zameriava len na platformu Android. Ku knižnici je možné pristupovať na platforme Android prostredníctvom natívnych jazykov C a C++, alebo jazyka Java. Výhoda natívneho jazyka je, že kód môže byť navrhnutý a odtestovaný na PC, následne sa len integruje do Android aplikácie. Jazyk Java ponúka rovnakú funkcionalitu, výhodou je, že nie je potrebné NDK a znalosť natívneho jazyka.

#### OpenCV4Android SDK

Pre vývoj aplikácii s OpenCV pre Android je potrebná inštalácia OpenCV4Android SDK. Pre správnu funkcionalitu je potrebné nainštalovať nasledovné komponenty:

- JDK
- Android SDK a NDK
- Prostredie Eclipse alebo Android Studio
- ADT a CDT doplnky v prípade použitia Eclipse

Od verzie 2.3.4 OpenCV4Android SDK používa OpenCV Manager API pre inicializáciu knižnice. OpenCV Manager zabezpečuje prístup k natívnym knižniciam a je potrebné ho mať nainštalovaný v zariadení iba raz, nezávisle od počtu aplikácií, ktoré OpenCV používajú. Ďalším pozitívom je, že užívateľ si môže stiahnuť aplikáciu v Obchode Play a nemusí vedieť akú má hardvérovú platformu. Korektná verzia bude nainštalovaná automaticky. V tabuľke 2.2 je uvedený prehľad rôznych verzií podľa hardvérovej platformy. Nevýhodou je potrebné nainštalovanie ďalšej aplikácie.

| Hardvérová platforma | Názov balíka |
|---|---|
| armeabi-v7a | OpenCV_2.4.9_Manager_2.18_armv7a-neon.apk |
| armeabi | OpenCV_2.4.9_Manager_2.18_armeabi.apk |
| Intel x86 | OpenCV_2.4.9_Manager_2.18_x86.apk |
| MIPS | OpenCV_2.4.9_Manager_2.18_mips.apk |

Tab. 2.2: Názvy balíkov pre rôzne hardvérové platformy(Android 2.3 a vyšší)[4]

# 3 DATABÁZY

Databázy sú organizované zbierky dát, ktoré umožňujú ukladanie a následné získavanie týchto dát. V nasledovnej časti budú popísané databázy MySQL a SQLite a tiež problematika synchronizácie dát.

## 3.1 MySQL

MySQL je najrozšírenejší otvorený databázový softvér s viac ako 100 miliónov inštaláciami. Je súčasťou takzvaného LAMP (Linux, Apache, MySQL, PHP/Perl/Python) podnikového otvoreného softvérového stacku.

Pri práci s MySQL sa používa MySQL server, ku ktorému sa pripája pomocou klienta. Klient vykonáva dopyty na server a zobrazuje výsledky. MySQL môže tiež pracovať v dávkovom režime, kedy je dodaný súbor s danými dopytmi na vykonanie. Pre ukladanie dát do databázy je potrebné najprv jej vytvorenie. V rámci databázy môžu byť rôzne tabuľky, v ktorých sú dáta ukladané. Každá tabuľka tiež obsahuje stĺpce, ktorých prvky môžu byť rôzneho typu. Základné delenie je na číselné typy, časové, dátumové a reťazcové typy. Ich výber závisí od použitej aplikácie a zvoleného formátu.

Pre prácu s databázami je množstvo nástrojov, ktoré sú buď konzolové, alebo grafické. Najznámejším grafickým nástrojom je phpMyAdmin, ktorý je písaný v jazyku PHP a podporuje veľké množstvo operácii s databázami ako zobrazovanie a úprava databáz a tabuliek, vytváranie a kopírovanie databáz, manažovanie užívateľov a ich práv a mnoho ďalších. Výhodou je tiež jednoduchý import a export dát.

## 3.2 SQLite

SQLite je malý, výkonný, relačný databázový systém napísaný v jazyku C. SQLite je voľne dostupný pod licenciou public domain na súkromné alebo komerčné použitie. Hlavný rozdiel oproti systému MySQL je ten, že MySQL server beží ako nezávislý proces od klientskeho procesu. SQLite priamo číta a zapisuje systémové súbory.

Formát databázy je multiplatformový a je možné ju kopírovať medzi rôznymi operačnými systémami a architektúrami. Každý program s prístupom na disk je schopný používania SQLite. Vďaka tomu, že je transakčný, zmeny vrámci jednej transakcie sú vykonané úplne, alebo nie sú vykonané vôbec, aj v prípade, že pri zápise na disk dôjde k pádu aplikácie, pádu operačného systému alebo zlyhaniu napájania.

SQLite bol navrhnutý s ohľadom na kompaktnosť a má minimálne hardvérové nároky. Pri použití SQLite nie je potrebná žiadna inštalácia alebo konfigurácia. Výhodou tiež je, že napriek bezserverovému riešeniu, rôzne aplikácie môžu pristupovať k rovnakej databáze v rovnakom čase.

## 3.3 Synchronizácia databáz

V prípade použitia jedného hlavného databázového servera, môžu klienti odosielať svoje požiadavky nezávisle. Problém nastane, keď klient nemá vo chvíli vytvorenia požiadavky prístup k serveru. Môže to byť z dôvodu nedostupnosti servera, prípadne výpadku spojenia medzi klientom a serverom. Ak je potrebné získavať odpovede na dopyty aj pri nedostupnosti servera, je potrebné navrhnúť alternatívne riešenie.

### 3.3.1 Lokálna databáza len na čítanie

Najjednoduchším prístupom je vytvorenie lokálnej kópie hlavnej databázy. Klient si vytvorí aktuálnu kópiu databázy a v prípade nedostupnosti servera odosiela klient požiadavky do lokálnej databázy. Pri obnovení spojenia sa požiadavky posielajú znovu na hlavný server a vytvorí sa tiež aktuálna kópia hlavnej databázy. V prípade, že by klient do tejto lokálnej databázy zapisoval, zmeny by sa neprejavili v hlavnej databáze. Preto je v režime offline povolené len čítanie databázy.

### 3.3.2 Lokálna databáza so synchronizáciou

V prípade, že je potrebné zaznamenávať zápisy klienta počas nedostupnosti spojenia na hlavný server, je potrebné navrhnúť riešenie synchronizácie databáz. Toto riešenie je výrazne komplikovanejšie, pretože aj hlavná databáza mohla byť v čase nedostupnosti zmenená. V tomto prípade sa zvykne používať pre kľúč na identifikáciu záznamu jedinečný univerzálny identifikátor a nie štandardný autoinkrementálny. Hlavným znakom tohto riešenia je sledovanie zmien, ktoré sa vo forme dávky premietnu do hlavnej databázy.

K dispozícii sú existujúce knižnice, ktoré slúžia na synchronizáciu databáz. Pre platformu Java sú k dispozícii napríklad Daffodil Replicator alebo SymmetricDS.

# 4 SPRACOVANIE OBRAZU

V tejto kapitole sú uvedené základy spracovania obrazu.

## 4.1 Lokálne predspracovanie obrazu

Pri lokálnom predspracovaní obrazu sa využíva okolie bodu vo vstupnom obraze na získanie novej jasovej hodnoty vo výstupnom obraze. Tento postup je tiež možné nazvať filtrácia obrazu. Metódy sú rozdeľované podľa cieľa predspracovania, na metódy vyhladzovania obrazu, ktoré potláčajú šum a gradientné operátory, ktoré slúžia na zvýrazňovanie hrán. Ďalej môžu byť metódy lokálneho predspracovania obrazu rozdelené podľa funkčného vzťahu na lineárne a nelineárne.

Pri lineárnych operáciách sa získa hodnotu jasu výstupného pixla $g(i, j)$ ako lineárna kombinácia jasových úrovní v okolí $O$ pixla $f(i, j)$ vo vstupnom obraze. Príspevky jednotlivých bodov okolia sú váhované koeficientom $h$. Jedná sa o diskrétnu konvolúciu obrazu s konvolučnou maskou $h$, ktorej vzťah je možné vidieť v 4.1.

$$f(i,j) = \sum_{(m,n\,)\in O} \sum h(i-m, j-n)g(m,n) \tag{4.1}$$

### 4.1.1 Vyhladzovanie obrazu

Vyhladzovanie obrazu slúži predovšetkým na potlačenie šumu v obraze. Nová jasová hodnota daného bodu môže byť získaná priemerom jasových hodnôt jeho okolia $O$. Pri vyhladzovaní obrazu týmto spôsobom však dochádza k potláčaniu ostrých hrán v obraze. V prípade potreby je možné vykonať vyhladzovanie, ktoré zachováva hrany v obraze tak, že priemer hodnôt sa počíta iba z tých bodov v okolí, ktoré majú podobnú jasovú úroveň ako vyšetrovaný bod.

Ak je k dispozícii viac snímkov totožnej scény zaťažených šumom, je možné priemerom príslušných bodov získať výsledný obraz, v ktorom bude šum potlačený. Tento postup je možný iba v prípade, ak pre šum v obraze platí, že každý obrazový bod je zaťažený náhodne šumom so strednou chybou priemeru $\sigma_{\overline{X}} = \frac{\sigma}{\sqrt{n}}$, kde $\sigma$ je smerodajná odchýlka výberového súboru a $n$ počet snímkov z ktorého je priemer počítaný. Výsledný obraz bez rozmazaných hrán je možné získať pomocou vzťahu 4.2.

$$f(i,j) = \frac{1}{n} \sum_{k=1}^{n} g_k(i,j) \tag{4.2}$$

Ak nie je k dispozícii viac snímkov a obraz je zaťažený šumom, potlačiť šum je možné priemerovaním jasových hodnôt okolia. Výsledok je prijateľný za predpokladu, ak je šum menší ako najmenší objekt záujmu v obraze. Značnou nevýhodou je rozmazávanie hrán v obraze, keďže podobne ako šum, hrany predstavujú vysoké frekvencie v obraze. Pre okolie 3×3 je konvolučná maska:

$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.3}$$

V konvolučnej maske $h$ nemusí byť všetkým bodom priradená rovnaká hodnota, je možné napríklad zvyšovať hodnotu smerom k stredu masky, ako je znázornené v 4.4. Zvyčajne sa dodržuje zásada, že súčet jednotlivých bodov masky je rovný jednej. Ak by nebol, výsledný obraz by sa javil v prípade hodnoty väčšej ako jedna ako svetlejší, v prípade hodnoty menšej ako jedna ako tmavší.

$$h = \frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad h = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{4.4}$$

**Mediánový filter**

Pokiaľ je konečný rad hodnôt zoradených podľa veľkosti, medián je daný hodnotou stredného prvku v rade. Ak je tento rad zostavovaný, snažíme sa, aby pozostával z nepárneho počtu prvkov, na oboch stranách tak bude rovnaký počet prvkov a medián bude daný jednoznačne. Mediánová filtrácia patrí medzi nelineárne metódy vyhladzovania obrazu. Princíp spočíva v nahradení aktuálnej jasovej hodnoty bodu, hodnotou mediánu z jeho okolia. Tento typ filtrácie dobre filtruje impulzný šum a zároveň dochádza k menšiemu rozmazávaniu hrán, vďaka tomu môže byť aplikovaný aj iteratívne. Jeho nevýhodou je poškodzovanie veľmi tenkých čiar v obraze a orezávanie ostrých rohov. V prípade väčšej masky môže problém predstavovať výpočetná náročnosť, pretože je potrebné zoradiť veľký počet prvkov pri počítaní každého obrazového bodu. Tento problém je možné vyriešiť tak, že prvky sa zoradia v každom riadku raz a následne sa pridá a odoberie jeden stĺpec. Zoradiť preto bude potrebné len novo pridaný stĺpec a nepríde sa tak o už zoradené hodnoty.

Pre výpočet novej hodnoty vyšetrovaného bodu z obrázku 4.1, sa zoradia jednotlivé body z okolia vrátane vyšetrovaného bodu podľa hodnoty jasovej úrovne. Nová hodnota bude rovná hodnote prvku v strede radu: 2 2 3 3 **3** 3 4 4 7

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 3 | 5 |
| 1 | 2 | 3 | 3 | 5 |
| 1 | 2 | 7 | 3 | 5 |
| 2 | 3 | 4 | 4 | 5 |
| 2 | 3 | 5 | 4 | 6 |

Obr. 4.1: Príklad filtrácie pomocou mediánového filtra

## 4.2 Detekcia hrán

Detektory hrán sú algoritmy, pomocou ktorých je možné nájsť veľké zmeny intenzity obrazovej funkcie, ktoré predstavujú hrany. Na tento účel je možné použiť operátor gradient $\nabla$ 4.5. Výsledkom operácie je vektorová veličina, ktorú je možné vyjadriť pomocou absolútnej hodnoty 4.6 a smeru 4.7 [8]:

$$\nabla f(x,y) = \frac{\partial f(x,y)}{\partial x}\bar{i} + \frac{\partial f(x,y)}{\partial y}\bar{j} \tag{4.5}$$

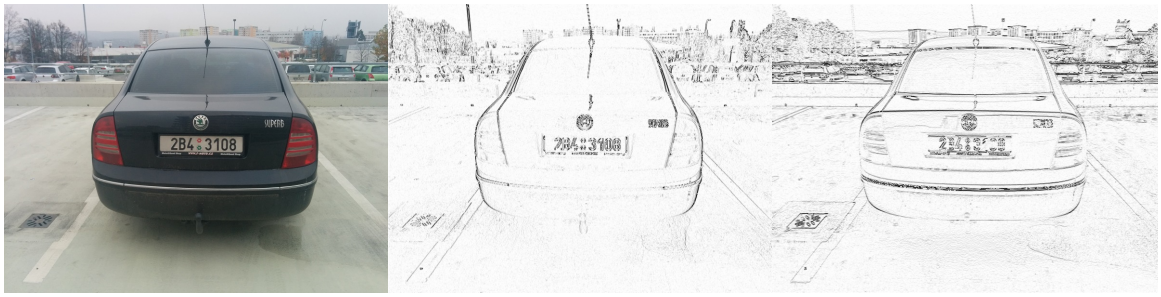
$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2} \tag{4.6}$$

$$\varphi = arg\left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right) \tag{4.7}$$

### 4.2.1 Sobelov operátor

Sobelov operátor slúži na detekciu hrán pomocou konvolúcie obrazu a jadra. Tieto jadrá môžu byť napríklad vertikálne alebo horizontálne ako je uvedené v 4.8

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{4.8}$$



(a) (b) (c)

Obr. 4.2: Porovnanie (a) originálneho obrazu a obrazov s použitím (b) vertikálneho a (c) horizontálneho jadra sobelovho operátora

### 4.2.2 Cannyho detektor

Cannyho detektor hrán je populárny algoritmus na detekciu hrán vyvinutý Johnom Cannym v roku 1986. Cannyho detektor bol navrhnutý tak, aby spĺňal nasledovné požiadavky:

- Minimálny počet chýb - dôležité hrany by nemali byť vynechané a oblasti, ktoré nie sú hranami by nemali byť falošne detekované ako hrany
- Minimálna chyba vzdialenosti - vzdialenosť medzi skutočnou a nájdenou hranou by mala byť minimálna
- Jednoznačnosť - na jednu hranu by mala byť len jedna odozva, rieši problém šumu v okolí hrany

Algoritmus Cannyho detektora [1] je možné vyjadriť pomocou nasledovného postupu:

1. Eliminácia šumu konvolúciou obrazu $f$ s Gaussovým filtrom so smerodajnou odchýlkou $\sigma$
2. Nájdenie gradientu hrán získaného pomocou horizontálneho a vertikálneho detektora hrán
3. Stenčovanie potlačením nemaximálnych bodov na detekciu hrany v mieste najväčšieho gradientu
4. Prahovanie s hysterézou

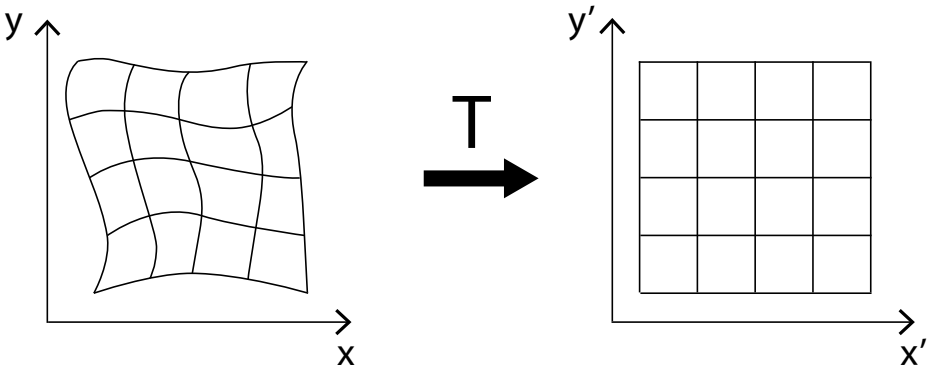
Výsledok algoritmu je zobrazený na obrázku 4.3

(a) (b)

Obr. 4.3: Porovnanie (a) originálu a (b) obrazu po aplikácii Cannyho detektora hrán

## 4.3 Geometrické transformácie

Geometrické transformácie umožňujú elimináciu geometrického skreslenia obrazu. Sú to funkcie $T$, ktoré mapujú pixel z pozície $(x, y)$ vo vstupnom obraze, na novú pozíciu $(x', y')$ vo výstupnom obraze.




Obr. 4.4: Geometrická transformácia roviny [1]

Vektorovú transformáciu **T**, je možné zapísať pomocou jej zložiek.

$$x' = T_x(x, y), \qquad y' = T_y(x, y) \tag{4.9}$$

### 4.3.1 Transformácie súradníc

Súradnice bodu vo výstupnom obraze po geometrickej transformácii súradníc je možné vypočítať podľa vzťahu 4.10 [1]. Táto transformácia je lineárna s ohľadom na

koeficienty $a_{rk}, b_{rk}$. Ak sú k dispozícii súradnice korešpondujúcich bodov $(x, y), (x', y')$, je možné riešením sústavy lineárnych rovníc získať koeficienty $a_{rk}, b_{rk}$.

$$x' = \sum_{r=0}^{m} \sum_{k=0}^{m-r} a_{rk} x^r y^k, \qquad y' = \sum_{r=0}^{m} \sum_{k=0}^{m-r} b_{rk} x^r y^k \tag{4.10}$$

V prípade, že sa transformácia nemení výrazným spôsobom v závislosti na pozícii v obraze, je možné transformáciu aproximovať polynómami nižších rádov a stačí tak tiež menší počet korešpondujúcich bodov. Tieto body by však mali byť v obraze rovnomerne rozmiestnené.

Pri použití bilineárnej transformácie stačia na získanie transformačných koeficientov 4 páry korešpondujúcich bodov [1]:

$$\begin{aligned} x' &= a_0 + a_1 x + a_2 y + a_3 xy \\ y' &= b_0 + b_1 x + b_2 y + b_3 xy \end{aligned} \tag{4.11}$$

Pokiaľ použijeme afinnú transformáciu, na výpočet transformačných koeficientov nám postačia 3 páry korešpondujúcich bodov [1]:

$$\begin{aligned} x' &= a_0 + a_1 x + a_2 y \\ y' &= b_0 + b_1 x + b_2 y \end{aligned} \tag{4.12}$$

## 4.4 Segmentácia

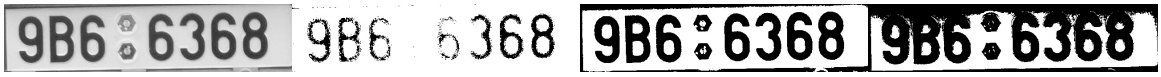
Správna segmentácia je dôležitým krokom v spracovaní obrazu. Cieľom je získanie oblastí, ktoré súvisia s predmetmi alebo oblasťami v reálnom svete. Oblasti môžu byť získané na základe určitých vlastností v obraze ako napríklad jas, farba alebo textúra, prípadne určovaním hraníc medzi oblasťami, využitím detektorov hrán, alebo postupným vytváraním oblastí. V prípade kompletnej segmentácie oblasti priamo zodpovedajú objektom v obraze, v prípade čiastočnej segmentácie oblasti priamo nezodpovedajú objektom v obraze. Na dosiahnutie kompletnej segmentácie je potrebné mať k dispozícii znalosti z danej problematiky. Na dosiahnutie čiastočnej segmentácie je obraz rozdelený do homogénnych regiónov s ohľadom na jas, farbu, textúru, či iné.

### 4.4.1 Segmentácia prahovaním

Prahovanie je jednoduchý spôsob segmentácie, ktorým je možné rozdeliť obraz na objekty a pozadie. Je to transformácia obrazu $f$ na binárny obraz $g$ s prahom $T$. Výsledkom prahovania je binárny obraz, kde obrazové elementy prislúchajúce objektu majú hodnotu 1 a obrazové elementy pozadia hodnotu 0.

$$g(i,j) = \begin{cases} 1 & \text{pre } f(i,j) \geq T. \\ 0 & \text{pre } f(i,j) < T. \end{cases} \tag{4.13}$$

Zvolenie hodnoty prahu T je pre segmentáciu prahovaním kľúčové. Ak nie je prah vhodne zvolený, môžu objekty zaniknúť v pozadí, prípadne môžu byť časti pozadia považované za objekty.



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

Obr. 4.5: Porovnanie (a) šedotónového originálu a obrazu po prahovaní s prahom (b) T=30, (c) T=100, (d) T=150

Prah môže byť určený rôzne, napríklad experimentálne, z histogramu, percentuálne, či štatisticky. Okrem zvolenia jedného prahu, môže byť prahov viac, prípadne prahovanie môže byť len čiastočné. V prípade nerovnomerného osvetlenia je potrebné prah postupne meniť, využíva sa na to adaptívne prahovanie.

## 4.5 Klasifikácia

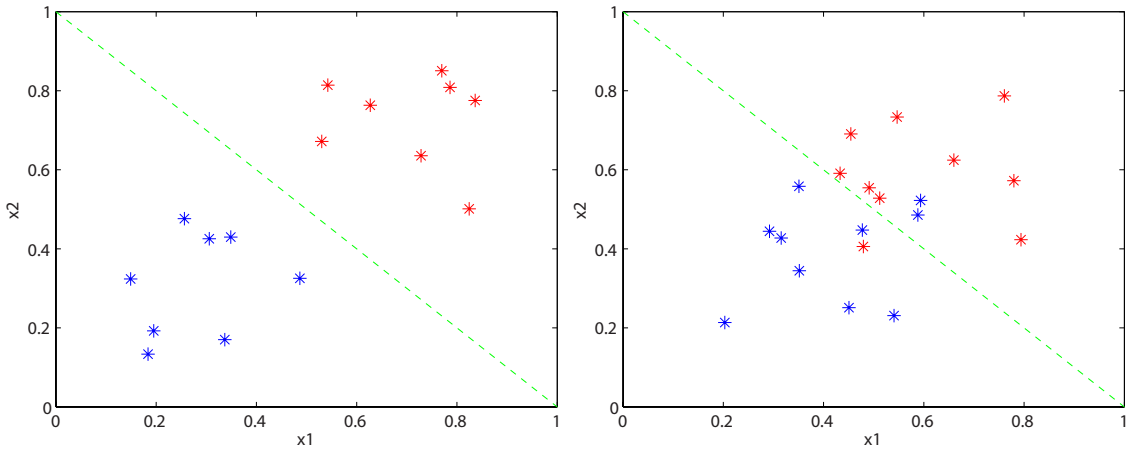
Rozpoznávanie objektov je založené na priraďovaní tried k objektom a tento proces sa nazýva klasifikácia. Počet tried je väčšinou dopredu známy a vyplýva z podstaty riešeného problému. Klasifikátor je algoritmus, ktorý vykonáva klasifikáciu. Klasifikátor neklasifikuje priamo podľa objektov, ale podľa ich príznakov. Príznaky sú merateľné vlastnosti objektov. Vzor, ktorý popisuje objekt, je tvorený príznakovým vektorom, ktorý pozostáva z $n$ príznakov. Všetky možné kombinácie vektorov príznakov tvoria priestor príznakov. Ak sú príznaky vhodne zvolené, objekty zaradené do rovnakých tried sa vyznačujú svojou blízkosťou príznakov v priestore príznakov. Presnosť klasifikácie (accuracy) môžeme určiť podľa vťahu 4.14, kde $N_{ok}$ je počet správne klasifikovaných objektov a $N_{tot}$ je počet všetkých klasifikovaných objektov.

$$\delta_{acc} = 100 \cdot \frac{N_{ok}}{N_{tot}} \qquad [\%] \tag{4.14}$$

### 4.5.1 Lineárna separabilita

Ak existuje nadplocha, ktorá rozdeľuje príznakový priestor tak, že len objekty z určenej triedy sú v danej oblasti, jedná sa o úlohu so separabilnými triedami. Porovnanie

lineárne separabilnej a lineárne neseparabilnej úlohy v 2D priestore môžeme vidieť na obr. 4.6. Väčšina praktických úloh nemá lineárne separabilné triedy, a preto budú niektoré objekty pri použití lineárneho klasifikátora vždy nesprávne klasifikované.




(a) (b)

Obr. 4.6: (a) lineárne separabilná a (b) lineárne neseparabilná úloha v 2D priestore

### 4.5.2 Lineárny klasifikátor

Oddeľujúce nadplochy sú definované pomocou $R$ skalárnych funkcií $g_1(x), g_2(x), ..., g_R(x)$, ktoré sa nzývajú diskriminačné funkcie [1]. Musia spĺňať nasledovnú podmienku, pre každé x$\in K_r$ a pre každé $s \in \{1, ..., R\}, s \neq r$:

$$g_r(\mathrm{x}) \geq g_s(\mathrm{x}) \tag{4.15}$$

Diskriminačná nadplocha medzi oblasťami tried $K_r$ a $K_s$ je definovaná ako

$$g_r(\mathrm{x}) - g_s(\mathrm{x}) = 0 \tag{4.16}$$

Objekt so vzorom x bude klasifikovaný do triedy, ktorej diskriminačná funkcia dosahuje maximum vzhľadom ku všetkým ostatným diskriminačným funkciám.

$$d(\mathrm{x}) = \omega_r \iff q_r(\mathrm{x}) = \max_{s=1,...,R} q_s(\mathrm{x}) \tag{4.17}$$

Lineárne diskriminačné funkcie sú najpoužívanejšie a zároveň najjednoduchšie. Ich všeobecný zápis je v 4.18.

$$g_r(\mathrm{x}) = q_{r0} + q_{r1}x_1 + ... + q_{rn}x_n \tag{4.18}$$

Pokiaľ sú všetky diskriminačné funkcie lineárne, jedná sa o lineárny klasifikátor.

## 4.6 Matematická morfológia

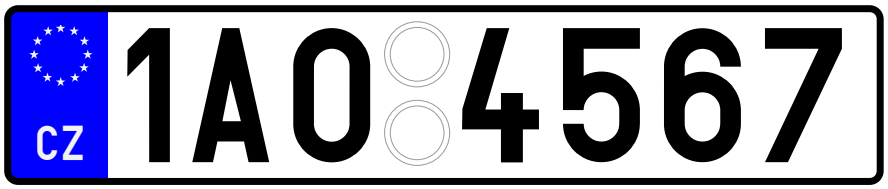
Je technika spracovania geometrických štruktúr, pôvodne založená na teórii množín. Pôvodne bola určená na použitie s binárnymi obrazmi, ale bola zovšeobecnená aj na šedotónové obrazy. Medzi základné operátory matematickej morfológie patrí:

- Dilatácia $I \oplus B$
- Erózia $I \ominus B$
- Otvorenie $I \circ B$
- Uzavretie $I \bullet B$
- Traf či miň $I \otimes B$
- Stenčovanie $I \oslash B$
- Zosilňovanie $I \odot B$
- Skelet

Morfologické operácie sa používajú predovšetkým na predspracovanie obrazu, pri stenčovaní a zhrubnutí obrazu , vytvorení kostry, vytvorení konvexného obalu a segmentácii.

# 5 POŽIADAVKY NA APLIKÁCIU

Aplikácia by mala byť schopná detekcie tabuliek s evidenčným číslom vozidla pre Českú republiku štandardných rozmerov 520 x 110 mm, kde je sedem čiernych alfanumerických znakov na bielom podklade v jednom riadku [16], ako je znázornené na obrázku 5.1. Aplikácia deteguje pozíciu EČV v obraze, následne sú z tohto výseku segmentované jednotlivé alfanumerické znaky a pomocou optického rozpoznávania znakov (OCR), získa číslo EČV. Získaný reťazec je možné v rámci aplikácie použiť na vyhľadávanie v databáze záujmových vozidiel. Výstupom teda bude informácia, či sa dané vozidlo v databáze nachádza alebo nie. Táto databáza bude uložená na vzdialenom serveri a tiež lokálne. V prípade, že nie je dostupné pripojenie na internet, bude možné vyhľadať číslo EČV v offline databáze. V prípade zlyhania detekcie EČV bude aplikácia obsahovať manuálne textové pole, aby bolo možné vyhľadávať v databáze aj priamo. V rámci aplikácie bude možné lokálnu databázu aktualizovať.




Obr. 5.1: Tabuľka s evidenčným číslom vozidla pre Českú republiku

## 5.1 Požiadavky na scénu

Snímanie EČV je predpokladané zo vzdialenosti v rozsahu 1 - 2 m. Snímacie zariadenie by malo byť umiestnené vo výške v rozsahu 0,3 - 1,6 m od vodorovnej roviny vozidla, čo zodpovedá výške bežnej pre fotografovanie smartfónom. Snímanie by malo byť uskutočnené priamo pred vozidlom buď z prednej alebo zadnej strany. Zariadenie by malo byť počas snímania stacionárne a musí byť poskytnutý dostatočný čas pre zaostrenie fotoaparátu. EČV by nemala byť nadmerne znečistená, zhrdzavená alebo deformovaná. Snímanie by malo byť uskutočnené za denného svetla.

## 5.2 Požiadavky na hardvér

Aplikácia je určená pre zariadenia s operačným systémom Android, s verziou 4.0 a vyššou. Predovšetkým sa jedná o smartfóny a tablety. Zariadenie musí disponovať zadnou kamerou s minimálnym rozlíšením 1024 × 768 pixlov v režime *live preview*.

Pri nižšom rozlíšení nie je garantovaná správnosť rozpoznania EČV. Zariadenie musí byť podporované knižnicou OpenCV.

## 5.3 Testovacia množina

Pre účely testovania bolo zhotovených spolu 270 testovacích snímkov. Boli zosnímané za rozdielnych svetelných podmienok, zo vzdialenosti približne 1,5 m pred vozidlom, vo výške 0,5 - 1,5 m, smartfónom LG Nexus 5 v rozlíšení $1024 \times 768$ a automatickým nastavením jasu a zaostrenia. Tieto snímky boli vybrané z náhľadu počas režimu *live preview*, preto nebola dosiahnutá maximálna ostrosť snímok.

# 6 REALIZÁCIA

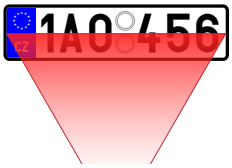
V nasledujúcej kapitole bude uvedený postup návrhu a finálne riešenie vrátane použitých algoritmov. Aplikácia bola písaná v jazyku Java v prostredí Android Studio vo verzii 1.2, s použitím OpenCV knižnice vo verzii 2.4.11.

## 6.1 Nastavenie prostredia

Pred samotnou realizáciou bolo potrebné pripraviť a nainštalovať prostredie, aby bolo možné aplikáciu programovať. Aplikácia bola spočiatku vyvíjaná v prostredí Eclipse s doplnkom Android Development Tools, no keďže sa počas vývoja aplikácie stal oficiálnym vývojovým nástrojom Android Studio, rozhodol som sa pre migráciu projektu do tohto programu. Android studio výrazne zjednodušuje vývoj aplikácie, preto bola migrácia výhodou. Ďalším krokom bola integrácia knižnice OpenCV. Podobne aj v tomto prípade je k dispozícii SDK pre Android. Pokiaľ chceme programovať aplikáciu s použitím OpenCV, je potrebné pridať medzi projekty knižnicu OpenCV. Následne je tiež potrebné importovať projekt podporných knižníc v7. V poslednom kroku prípravy bol vytvorený hlavný projekt bez aktivity.

## 6.2 Aplikačné zdroje

Pre správnu funkčnosť aplikácie je tiež potrebné pridať všetky aplikačné zdroje, s ktorými bude aplikácia pracovať. Na začiatku bol vytvorený vektorový model ikony, ktorý bol konvertovaný do rastrového súboru *ic_launcher.png*. Môžeme ho vidieť na obrázku 6.1. Vygenerovanie ikon pre rôzne rozlíšenia zariadení bolo uskutočnené v Android Asset Studio a tieto ikony boli umiestnené do príslušných *drawable* priečinkov. Ďalej bol do priečinka *values* pridaný súbor *strings.xml*, ktorý obsahuje všetky texty zobrazované v aplikácii. Týmto spôsobom sa aplikácia stáva modulárnou a je možné vytvárať jednoducho jazykové mutácie, napríklad anglický preklad sa umiestni do *values-en*, alebo slovenský do *values-sk*.



Obr. 6.1: Vektorový model ikony aplikácie

Samotné používateľské rozhranie je definované v priečinku *layout* súborom *main_view.xml*.

Bolo tiež potrebné upraviť a doplniť *AndroidManifest.xml* súbor. Definíciou *android:screenOrientation="landscape"* je určené, že aplikácia bude používaná v režime na šírku. Názov balíka bol zadefinovaný ako **eu.istrocode.alprdetector**. Tento názov slúži ako jedinečný identifikátor aplikácie. V prípade publikovania aplikácie do Obchodu Play, nie je možné tento identifikátor zmeniť. Ak by sa tak stalo, išlo by už o novú aplikáciu. *minSdkVersion* bolo zvolené na číslo 8. Z tabuľky 2.1 vyplýva, že podporované sú všetky zariadenia s Androidom Froyo a novším. Aby mohla aplikácia pristupovať k fotoaparátu bolo pridané povolenie

- *android.permission.CAMERA*

Keďže aplikácia odosiela dopyty na externú online databázu, bolo potrebné pridať povolenie pre prístup na internet:

- *android.permission.INTERNET*

V prípade, že pripojenie na internet nie je dostupné, vyhľadávanie sa uskutoční v offline databáze uloženej v zariadení. Pre kontrolu pripojenia je potrebné pridať povolenie:

- *android.permission.ACCESS_NETWORK_STATE*

Do aplikácie bola pridaná možnosť vibrácii pri nájdení hľadanej EČV v databáze, aby bolo možné ovládať vibrácie zariadenia bolo pridané povolenie:

- *android.permission.VIBRATE*

Ďalej bol aplikácii umožnený zápis na externé úložisko, ktoré je využívané, keď aplikácia ukladá snímky do zariadenia a tiež vtedy, keď sa vykonáva rozpoznanie EČV zo statických snímkov:

- *android.permission.WRITE_EXTERNAL_STORAGE*

Pre spoľahlivé sťahovanie databázy do zariadenia je potrebné získať wake_lock, ak by totiž užívateľovi zhasol displej na zariadení, alebo by užívateľ zámerne zamkol displej stlačením zapínacieho tlačidla, došlo by k prerušeniu sťahovania. Aby bolo možné wake_lock aktivovať, bolo pridané povolenie

- *android.permission.WAKE_LOCK*

V súbore *AndroidManifest.xml* sú tiež definované dve aktivity. Prvá, *MainActivity*, je hlavná aktivita aplikácie. Je v nej zobrazený obraz z kamery zariadenia a všetky potrebné komponenty potrebné pri používaní aplikácie.

Druhou aktivitou je *SettingsActivity*, ktorá obsahuje všetky potrebné nastavenia aplikácie.

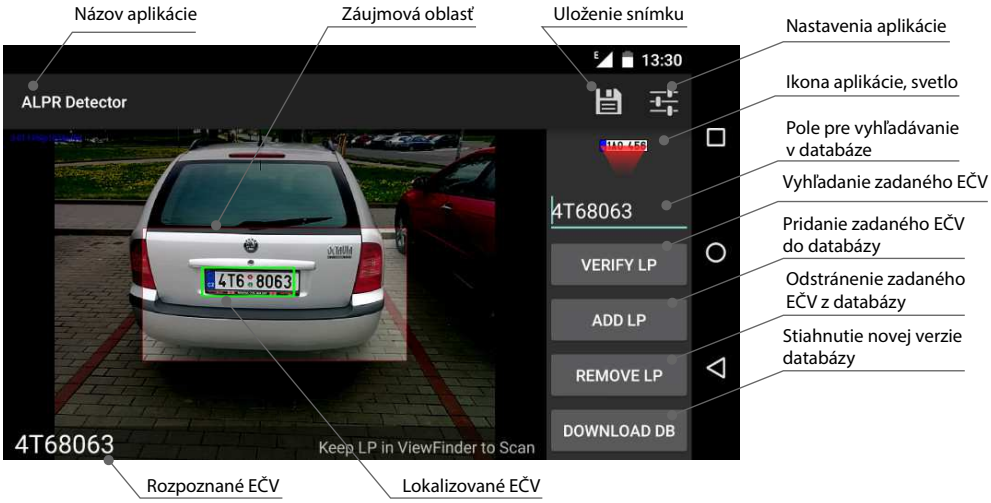
## 6.3 Používateľské prostredie aplikácie

V nasledujúcej časti bude popísané používateľské prostredie aplikácie, jednotlivé aktivity, tlačidlá a voliteľné nastavenia.

### 6.3.1 MainActivity

Základ hlavnej aktivity aplikácie *MainActivity* tvorí *SurfaceView* s identifikátorom *main_activity_java_surface_view*. Do tohto *SurfaceView* sa vykresľuje obraz z kamery zariadenia. Primárne je zvolená zadná kamera, pre jednoduchšie používanie. Toto zobrazenie je ďalej delené na dve časti, stredná časť zobrazuje aktívny snímací región lemovaný červeným obdĺžnikom, ktorý sa používa pri analýze obrazu a neaktívne okolie, ktoré slúži k jednoduchšej orientácii pri smerovaní záberu na vozidlo. Po nasmerovaní aktívneho regiónu na EČV, sa zobrazí okolo danej EČV zelený lem, ktorý indikuje správnu detekciu regiónu EČV. Užívateľ tak získa prehľad, z ktorej oblasti sa bude EČV rozpoznávať. V ľavom dolnom rohu sa po rozpoznaní nachádza text EČV.

Výhodou z pohľadu užívateľa je real-time analýza obrazu, pričom nie je potrebné stláčať spúšť fotoaparátu, aplikácia vykoná rozpoznanie znakov ihneď ako deteguje región EČV, preto sa ani v tomto zobrazení tlačidlo spúšte nenachádza. Na pravej strane sa nachádzajú ovládacie komponenty. V hornej časti sú to nastavenia aplikácie, ktoré sú bližšie popísané v kapitole 6.3.2. Pod nastaveniam je zobrazená ikona aplikácie, ktorá slúži zároveň na spúšťanie osvetlenia a pod ňou vstupné textové pole pre vyhľadávanie EČV v databáze. Keďže sa po rozpoznaní EČV automaticky skopíruje do tohoto poľa, z pohľadu užívateľa nie je potrebné text prepisovať. Toto pole je však k dispozícii pre prípady, keď je potrebné urobiť korektúru rozpoznaného textu, prípadne ak zlyhá rozpoznanie EČV. Ďalej sú tu tlačidlá na prácu s databázou. Toto zobrazenie predpokladá užívateľa s plnými právami na čítanie a zápis do databázy. Súčasťou aplikácie by mohlo byť prihlásenie, ktoré by jednotlivé funkcie podľa oprávnení užívateľa limitovalo. Tlačidlom *VERIFY LP* sa vykoná vyhľadanie EČV v databáze. Ďalej sú tu tlačidlá *ADD LP* na pridávanie nových EČV do databázy a *REMOVE LP* na odstraňovanie. V spodnej časti sa nachádza tlačidlo *DOWNLOAD DB* na stiahnutie databázy do zariadenia.

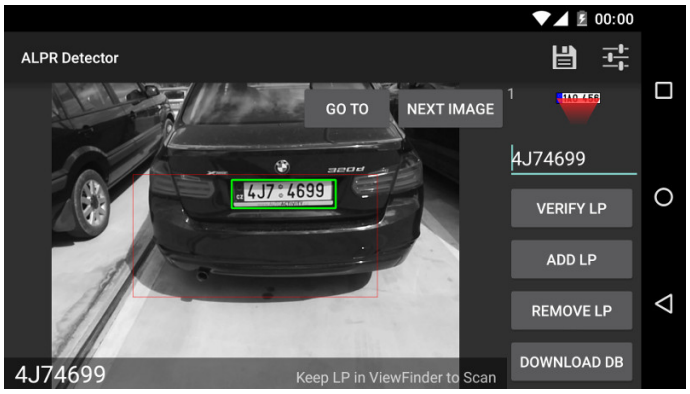



Obr. 6.2: Používateľské prostredie aplikácie na smartfóne Nexus 5

### Analýza statických snímkov

V prípade, že užívateľ zvolí v nastaveniach voľbu statických snímkov, zobrazenie *live preview* sa deaktivuje a namiesto neho sa zobrazí komponent *ImageView* obsahujúci statický snímok. Tieto snímky sú načítavané z externého úložiska zariadenia. Snímky určené na detekciu, sa vložia do priečinka *alprdetector/samples* na externom úložisku. Názvy týchto súborov musia byť čísla od 1, s príponou *jpg*. Aby rozpoznanie prebiehalo rovnakým spôsobom ako pri režime *live preview*, je vhodné tieto snímky získať práve z tohto režimu pomocou tlačidla na ukladanie snímkov.

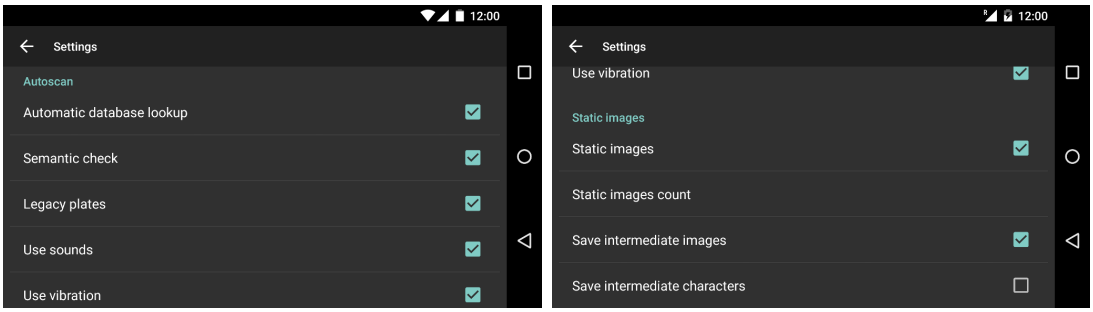
Pri zvolení voľby statických snímkov, sa na hlavnej obrazovke zobrazia doplňujúce tlačidlá na ovládanie režimu statických snímkov. Po stlačení tlačidla *GOTO*, sa preskočí na snímok s číslom, ktorý sa zadá do textového poľa. Vedľa ikony aplikácie sa zobrazí index práve zobrazovaného snímku. Po pridržaní tohto tlačidla sa počítadlo obnoví a detekcia môže začať odznovu. K dispozícii je tiež tlačidlo *NEXT IMAGE*, ktorým sa môže priamo nastaviť ďalší snímok a nemusí sa zadávať jeho index. Ak sa v režime statických snímkov vykoná uloženie snímku, uloží sa anotovaný snímok po vykonaní detekcie.




Obr. 6.3: Analýza statických snímkov

### 6.3.2 SettingsActivity

Z hlavnej aktivity aplikácie *MainActivity*, sa po kliknutí na nastavenia zobrazí druhá aktivita *SettingsActivity*, kde môžu byť menené nastavenia aplikácie. Tieto nastavenia sú rozdelené do dvoch sekcií. Prvá sa týka automatického snímania a vyhľadávania, druhá rozpoznávania statických snímkov.




(a) (b)

Obr. 6.4: Nastavenia aplikácie (a) automatické snímanie a vyhľadávanie (b) načítavanie statických snímkov

#### Nastavenia aplikácie, sekcia Autoscan

V prvej časti nastavení označenej ako *Autoscan*, sú k dispozícii voľby týkajúce sa automatického načítavania EČV a vyhľadávania v databáze. V prípade, že užívateľ zvolí možnosť automatického vyhľadávania, ihneď po načítaní EČV sa odošle dopyt s číslom EČV na vzdialenú databázu. V prípade, že pripojenie do siete internet nie je dostupné, táto požiadavka sa presmeruje do lokálnej offline databázy. Ak užívateľ túto možnosť vypne, EČV sa po načítaní len skopíruje do vyhľadávacieho

poľa. Týmto spôsobom môže užívateľ pred samotným odoslaním požiadavky spraviť korektúru, ale odoslanie požiadavky musí potvrdiť.

Užívateľ tiež môže zapnúť kontrolu sémantiky EČV, ktorá je bližšie popísaná v kapitole 6.6, pričom môže zapnúť podporu aj pre staršie typy EČV.

Ďalej má užívateľ možnosť zapnúť dodatočné signalizácie pri načítaní EČV. K dispozícii je zvuková signalizácia, ktorá je aktivovaná po každom načítaní EČV, nezávisle na automatickom vyhľadávaní v databáze. V prípade, že sa daná EČV v databáze našla, je táto udalosť signalizovaná dvojitým pípnutím. V istých prípadoch je zvuková signalizácia nežiadúca a užívateľ má preto možnosť aktivovať nezávisle na nej vibračnú signalizáciu. Vibračný mechanizmus sa aktivuje v rovnakých prípadoch ako zvuková signalizácia.

#### Nastavenia aplikácie, sekcia statické snímky

Táto sekcia bola využívaná hlavne pri vývoji, ale slúži aj pre demonštračné účely funkcie jednotlivých algoritmov. Prvou položkou je aktivácia statických snímkov. Súčasťou nastavení je voľba, koľko snímkov bude postupne analyzovaných. Po dosiahnutí tohto počtu sa počítadlo nuluje a detekcia môže začať odznova. K dispozícii je tiež ukladanie obrázkov v priebehu detekcie. Získajú sa tak obrázky z jednotlivých krokov detekcie a môžu byť tak ľahšie skúmané príčiny zlyhania algoritmov. Okrem toho je možné zvoliť ukladanie jednotlivých znakov, pre ladenie algoritmu OCR.

### 6.3.3 Prisvetlenie scény

Z požiadaviek na aplikáciu, ako je uvedené v kapitole 5 vyplýva, že aplikácia má byť použitá za denného svetla. V prípade použitia za šera, prípadne zhoršených svetelných podmienok bola do aplikácie pridaná možnosť zapnúť svetlo na zariadení pokiaľ ním disponuje. Svetlo je možné zapnúť stlačením ikony aplikácie. Pri nastavení osvetlenia je najprv potrebné získať aktuálne parametre objektu *mCamera*:

```
Camera.Parameters p = mCamera.getParameters();
```

následne sa nastaví požadovaný mód pre zapnutie

```
p.setFlashMode(Camera.Parameters.FLASH_MODE_TORCH);
```

a pre vypnutie

```
p.setFlashMode(Camera.Parameters.FLASH_MODE_OFF);
```

Tieto nastavenia sa zapíšu nasledovne

```
mCamera.setParameters(p);
```

## 6.4 Detekcia evidenčných čísel vozidiel

Aby bolo možné vyhľadávať v databáze záujmových vozidiel je potrebné najprv evidenčné číslo vozidla v obraze správne lokalizovať, následne segmentovať jednotlivé znaky, a potom metódou rozpoznania znakov získať text z EČV. Výber algoritmov bol robený na základe porovnania výsledkov implementácie v programe Matlab. Následne som sa rozhodol pre implementáciu priamo pomocou knižníc OpenCV v jazyku C++. Tento prístup má výhodu, že je písaný v natívnom jazyku, no v skorých štádiách sa zdal ako málo efektívny. Preto som uprednostnil využitie knižníc OpenCV v jazyku Python. Použité funkcie existujú aj v Jazyku Java, preto prepísať kód vo finálnom riešení do tohto jazyka nie je veľmi problematické.

### 6.4.1 Lokalizácia EČV

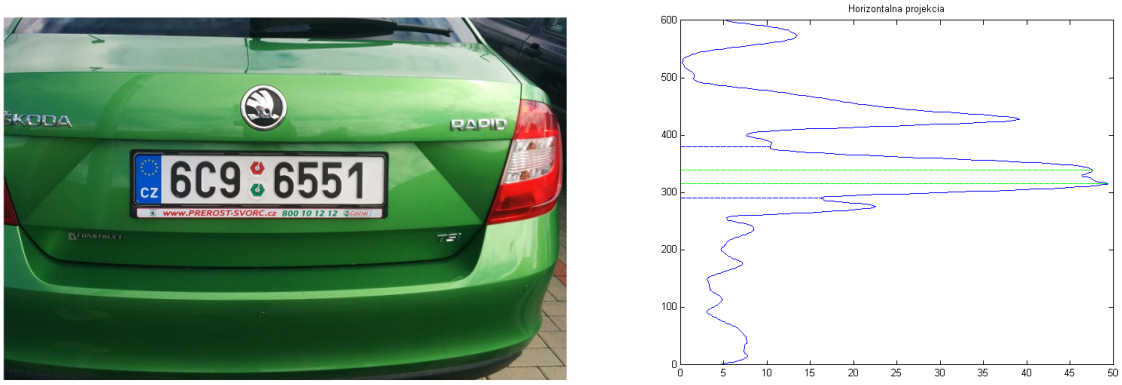
Úlohou lokalizácie EČV je vyhľadať EČV v obraze, pokiaľ sa tam nachádza. Rozhodol som sa vyskúšať viacero postupov.

#### Detekcia pomocou vertikálnych hrán

Táto metóda bola testovaná ako prvá. Vychádza z predpokladu, že EČV pozostáva z vyššieho množstva vertikálnych hrán v obraze. Na začiatok sa vykoná konvolúcia vstupného obrazu s konvolučným filtrom pozostávajúcim z vertikálneho Sobelovho jadra 6.1.

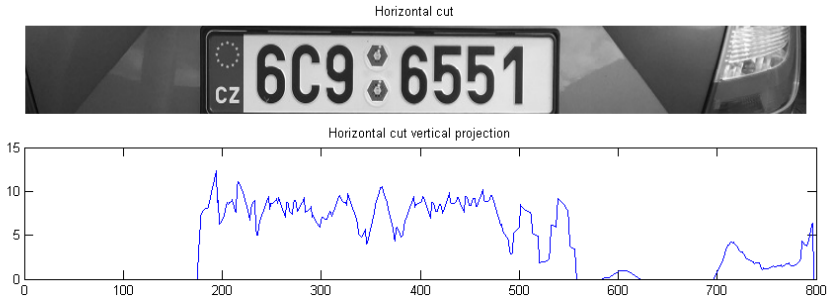
$$G_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{6.1}$$

Po nájdení maxima bol urobený výrez v horizontálnom smere, ktorý obsahoval aj EČV vozidla. Podobne bola urobená vertikálna projekcia výrezu. Priebeh projekcie bol vyhladený a pomocou podmienky na minimálnu hodnotu orezaný. Tento konečný obraz obsahoval aj EČV.




(a) (b)

Obr. 6.5: Obraz a jeho horizontálna projekcia po aplikovaní Sobelovho operátora




Obr. 6.6: Vertikálna projekcia vertikálnych hrán v obraze horizontálneho výrezu

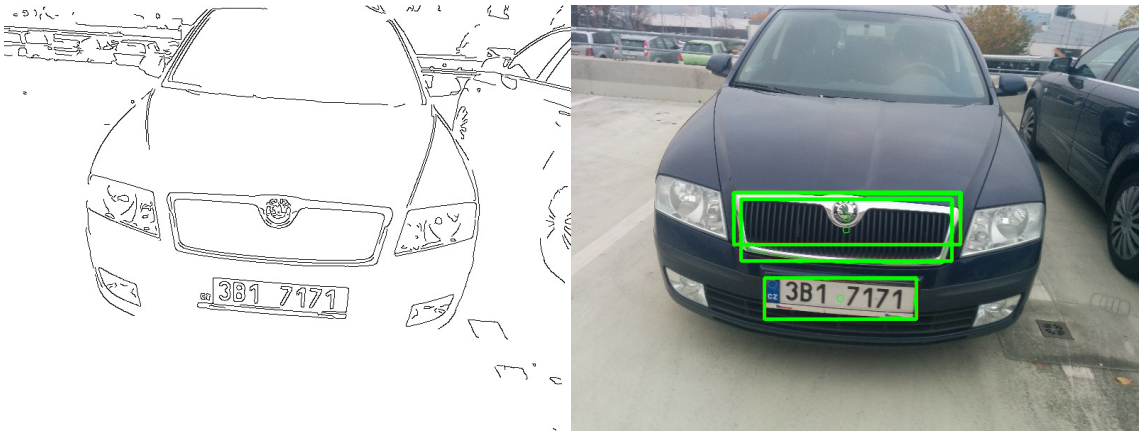

Obr. 6.7: Finálny výrez EČV získaný pomocou detekcie vertikálnych hrán v obraze

Táto metóda bola pomerne spoľahlivá, pokiaľ bolo na snímke iba vozidlo bez okolitého prostredia. Pokiaľ sa v okolí nachádzali iné objekty, často dosahovali vyššiu hustotu vertikálnych hrán. Taktiež spôsobovala problémy predná maska vozidla, ktorá tiež niekedy dosahovala vyššiu hustotu vertikálnych hrán ako EČV. Algoritmus bol implementovaný v Matlabe a tiež v OpenCV pre Android.

### Detekcia pomocou kontúr

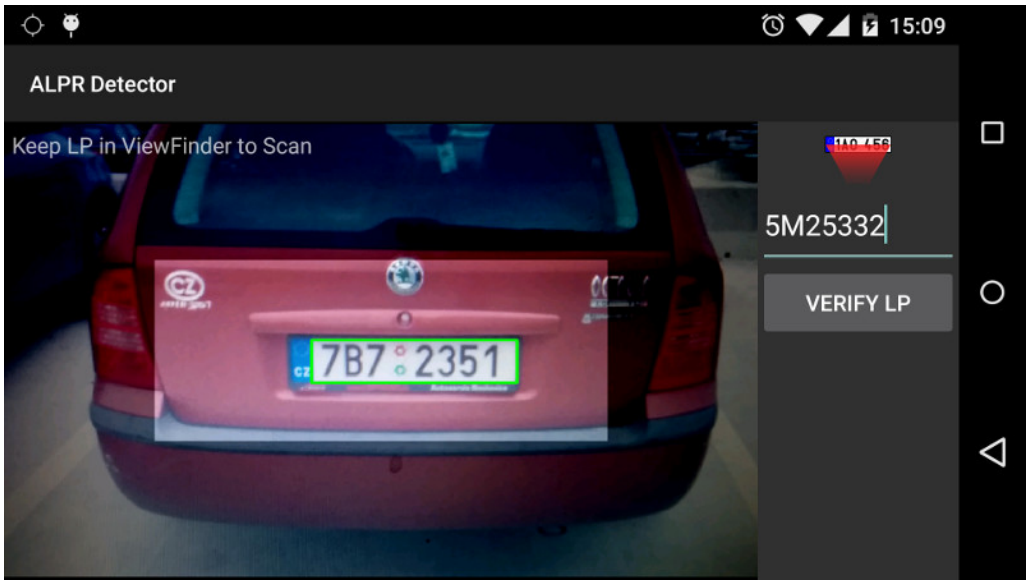
Tento typ detekcie využíva Cannyho detektor hrán. Po aplikácii Cannyho detektora sa použije detekcia kontúr. Detekované sú kontúry rôznych tvarov, pričom na základe

rozmerového kritéria sa vyberú kandidátne regióny, ktoré zodpovedajú určeným mediam. Tieto oblasti sú v obrázku 6.8 vyznačené zelenou farbou. Ako vyplýva z tohto obrázku, rozmerové kritérium nie je postačujúce, lebo kandidátnych regiónov môže byť nájdených viac. Ukázalo sa, že pre úspešnú detekciu je dôležité správne nastaviť parametre Cannyho detektora a výsledok tiež ovplyvnilo rozlíšenie obrázku. Rozlíšenie sme nastavili na 800 x 600 pixlov, spodný prah pre Cannyho detektor 100, horný 150.




(a) (b)

Obr. 6.8: Obraz z (a) Cannyho detektora a (b) kandidátne regióny




Obr. 6.9: Lokalizácia EČV pomocou Cannyho detektora s použitím popisu kontúr

Problém nastal pri EČV, ktoré neboli dostatočne kontrastné voči farbe vozidla,

prípadne boli čiastočne znečistené. Preto sme sa rozhodli ju vo finálnom riešení nepoužiť.

### Lokalizácia pomocou detekcie hrán a masky

Ďalšou testovanou metódou bola detekcia hrán s využitím masky. Pri použití tejto metódy bolo predpokladané, že EČV je oblasť s vysokou hustotou vertikálnych hrán, v ktorej okolí je oblasť s nízkou hustotou vertikálnych hrán. Bola vytvorená konvolučná maska, ktorá pozostávala z troch častí. V jadre boli kladné hodnoty, v jej okolí bola neutrálna zóna tvorená nulovými hodnotami a okolo tejto zóny bola tretia, ktorá mala hodnoty záporné. Celkový súčet všetkých bodov masky bol nulový. Výsledok konvolúcie je uvedený na obr. 6.10.


(a) (b)

Obr. 6.10: (a) pôvodný obraz (b) výsledok konvolúcie

Problematické bolo presné určenie hraníc EČV a tiež toto riešenie bolo výrazným spôsobom závislé na veľkosti detekovanej EČV.

### Detekcia regiónov s vysokou hustotou vertikálnych hrán

Predošlé metódy neboli dostatočne spoľahlivé a pri zhoršených okolitých podmienkach zlyhávali. Preto bolo potrebné nájsť inú, vhodnejšiu metódu lokalizácie EČV. Znovu bol využitý predpoklad, že EČV obsahuje vyššiu koncentráciu vertikálnych hrán a navrhnutý algoritmus lokalizácie túto skutočnosť využíval. Najprv bol obraz filtrovaný, gaussovým filtrom a následne bol získaný obraz vertikálnych hrán pomocou sobelovho operátora 4.8. V ďalšom kroku bol získaný prahovaním binárny obraz hrán. Pre určenie tohoto prahu sa ukázalo najvhodnejším riešením nastaviť prah napevno, keďže obraz hrán nie je veľmi závislý na osvetlení. Hrany sa najčastejšie degradujú pri snímku, ktorý je rozmazaný, v tomto prípade to však nevadí,

lebo snímky sú získavané hneď za sebou a zo špecifikácie požiadaviek uvedených v kapitole 5 vyplýva, že je potrebné nechať dostatočný čas zariadeniu na zaostrenie. Následne na tomto binárnom obraze bola vykonaná konvolúciá s obdĺžnikovou maskou. Boli tak získané zvýraznené oblasti s vysokou hustotou vertikálnych hrán. Následne bola použitá detekciu kontúr pre získanie obdĺžnikových regiónov. Pomocou rozmerových kritérií, môžeme kandidátov vyfiltrovať. Táto metóda sa ukázala ako pomerne spoľahlivá a výrazne lepšia ako detekcia pomocou Cannyho detektora a kontúr, no mala tiež svoje slabé stránky. Hlavný problém nastal v prípade, že sa nachádzali v okolí EČV vertikálne hrany. Dochádzalo tak k detekcii príliš veľkých regiónov, ktoré nezodpovedali rozmerom EČV. Preto bolo potrebné vyvinúť spoľahlivejší algoritmus lokalizácie.

### Viackrokové riešenie detekcie regiónov EČV

Pri tomto riešení boli využité poznatky z predošlých uvedených metód. Podstatou zostal predpoklad, že oblasť obsahujúca EČV obsahovala vysokú hustotu vertikálnych hrán. Najprv sa vybrala oblasť záujmu tak, ako je definovaná v užívateľskom rozhraní, zmenšilo sa tak rozlíšenie obrazu s ktorým bolo manipulované. Pri rozlíšení $1024 \times 768$ bodov mal orezaný obraz $614 \times 307$ bodov.

Tento obraz sa ďalej previedol do odtieňov šedej. Následne bol vyhladený gausovým filtrom s veľkosťou jadra $3 \times 3$. Potom bola vykonaná konvolúcia s vertikálnym jadrom ako je uvedené v 4.8. V OpenCV knižnici bol výsledok konvolúcie v rozsahu double $< 0, 1 >$. Aby mohol byť obraz ďalej spracovávaný, bol prevedený ho do rozsahu $< 0, 255 >$. Na odstránenie šumu bol použitý mediánový filter s veľkosťou jadra $3 \times 3$. Po filtrácii boli zvolené iba významné hrany v obraze pomocou prahovania. Aby boli významné hrany viac zvýraznené a vytvorili sa zhluky oblastí v vyskou koncentráciou hrán, bola vykonaná konvolúcia s jadrom o veľkosti $10 \times 10$ bodov. V predošlej metóde bola veľkosť konvolučného jadra $20 \times 10$ bodov, čím vznikli súvislé regióny. Hlavnou nevýhodou však bolo, že vznikali veľké oblasti, ktoré častokrát s EČV nesúviseli. Týmto spôsobom došlo k fragmentácii EČV, no boli vynechané regióny, ktoré do EČV nepatria.

Následne boli pomocou detekcie kontúr zaznamenané všetky regióny a boli získané ich hraničné obdĺžniky, medzi ktorými bolo aj fragmentované EČV. Následne bola vykonaná filtrácia oblastí. Podľa vytýčených podmienok pre minimálnu a maximálnu plochu oblasti a maximálnu výšku oblasti. Minimálnou plochou boli vyfiltrované nevýznamné regióny, maximálnou plochou boli vyradené regióny, ktoré boli väčšie ako je maximálny detekovaný rozmer EČV a maximálnou výškou tie regióny, ktoré sa mohli nachádzať v blízkosti EČV, no určite neboli súčasťou EČV. Po tejto filtrácii boli získané buď EČV v kompletnom stave, čo bolo veľmi zriedkavé,

ale častejšie bolo EČV fragmentované. V tomto obraze sa stále nachádzali nevýznamné regióny, preto nebolo možné ich jednoducho spojiť. Z tohto dôvodu bolo zvolené selektívne spájanie regiónov. Regióny, ktoré boli geometricky blízko seba, boli navzájom zlúčené.

Tieto zlúčené regióny, boli ďalej podrobované viacerým kritériám. Medzi tieto kritériá patrili minimálna a maximálna plocha, maximálna šírka a výška EČV a tiež pomer strán regiónu musel byť v určenom rozsahu. Bolo zistené, že toleranciu rozmerov EČV je potrebné rozšíriť, pretože pri snímaní EČV pod uhlom dochádza k deformácii EČV a tiež zmene pomeru strán. Uvedené parametre boli nastavovali experimentálne tak, aby bola dosiahnutá čo najvyššia úspešnosť lokalizácie v testovacej množine. Oblasti, ktoré spĺňali tieto podmienky, boli prehlásené za kandidátne regióny a boli ďalej zaradené k ďalšiemu spracovaniu.

### 6.4.2 Korekcia natočenia EČV

Po úspešnom získaní regiónu EČV je vhodné na danom obraze vykonať korekciu natočenia. Zvýši sa tak predovšetkým úspešnosť rozpoznania znakov. Pred vykonaním samotnej korekcie natočenia, je potrebné najprv zistiť uhol, ktorým bude natočenie kompenzované. Používaným, ale výpočetne náročným je použitie Houghovej transformácie.

V tejto práci bolo zvolené jednoduchšie riešenie, pomocou hľadania maxima v horizontálnom priemete vertikálnych hrán. Najprv boli vo vstupnom obraze detekované vertikálne hrany, podobne ako pri lokalizácii EČV. Následne bol tento obraz hrán rotovaný v rozsahu $\pm 30°$. V každom priebehu boli ukladané dosiahnuté maximá horizontálneho priemetu. Uhol pri ktorom bolo dosiahnuté absolútne maximum, bol použitý pri finálnej korekcii. Spočiatku bola korekcia vykonávaná s krokom 1°, ale ako sa ukázalo, pre spracovanie v reálnom čase, to bol príliš jemný krok a výsledný počet spracovaných snímkov klesol bez rozpoznania znakov na len 1-2 fps. Preto bolo potrebné zvoliť kompromis medzi presnosťou korekcie natočenia a výpočetnou náročnosťou. Nakoniec bol zvolený krok 3°, ktorý bol pre účely korekcie dostačujúci a zároveň nedošlo k výraznému zníženiu počtu spracovaných snímkov za sekundu.

### 6.4.3 Segmentácia znakov

Po úspešnom získaní regiónu obsahujúceho EČV a prípadnej korekcii natočenia, bolo možné pristúpiť ku segmentácii znakov. Obraz daného regiónu bol najprv prahovaný. Prahovať je možné rôznymi spôsobmi, avšak adaptívny prah sa v tomto prípade neosvedčil, pretože vo výslednom binárnom obraze bolo príliš veľa šumu,

ktorý komplikoval detekciu znakov. Bola zvolená preto radšej metóda Otsu. Výsledok prahovania je možné vidieť na obr. 6.11.


Obr. 6.11: Prahovanie regiónu EČV metódou Otsu

Následne bol získaný binárny obraz, v ktorom boli vyhľadané kontúry. Tie boli ďalej filtrované, podľa určených podmienok, ktorými boli minimálna a maximálna plocha znaku a tiež maximálna chyba pomeru strán. Pokiaľ oblasť spĺňala uvedené podmienky, bola klasifikovaná ako znak. Jednotlivé oblasti znakov sú vykreslené v pôvodnom obraze na obr. 6.12


Obr. 6.12: Zobrazenie oblastí s detekovanými znakmi

Pri získaní regiónov obsahujúcich znaky sa však stratila informácia o ich postupnosti, keďže algoritmus vyhľadávania kontúr túto informáciu nezaznamenával. Bola však získaná jednoduchým spôsobom tak, že boli vybrané obdĺžniky ohraničujúce jednotlivé znaky, v ktorých boli následne zoradené pozície ľavých horných rohov, podľa osi x.

Znaky boli normalizované na rozmer $32 \times 58$ pixlov s použitím interpolácie najbližšieho suseda, aby bol obraz zachovaný ako binárny. Danému znaku boli invertované farby a doplnená oblasť okolo znaku 4 pixle v horizontálnom a 1 pixel vo vertikálnom smere po celom obvode. Porovnanie originálov segmentovaných znakov a obrazov s dodatočným spracovaním je na obr. 6.13.


Obr. 6.13: Detekované jednotlivé alfanumerické znaky EČV

### 6.4.4 Klasifikácia znakov

Pri klasifikácii znakov boli použité dve rôzne metódy. V nasledovnej časti budú popísané.

#### Priame porovnávanie vzorov

Prvou metódou na klasifikáciu znakov bolo priame porovnávanie vzorov. Najskôr bolo potrebné pripraviť jednotlivé vzory, s ktorými sa segmentovaný znak porovnával. Tieto vzory boli vytvorené zo vzoru písma SPZ ČR TrueType Font s drobnými korektúrami. Problém nastal hlavne pri rozpoznávaní čísla 1, keďže je užšie ako zvyšné znaky. Bol preto umelo rozšírený, aby korešpondoval s ostatnými predlohami. Každý znak bol uložený ako samostatný png obrázok. Na samotné porovnanie so vzorom bola použitá metóda z OpenCV *Core.absdiff*, ktorou bol získaný binárny obraz. V tomto obraze boli zobrazené iba body, ktoré nekorešpondovali so vzorom. Následne bol pomocou metódy *Core.countNonZero* spočítaný počet nekorešpondujúcich bodov. Kritériom bolo nájdenie minima tejto funkcie. Vzor, s ktorým sa daný obraz najlepšie zhodoval, bol určený ako výsledný znak. Algoritmus bol nespoľahlivý v prípade, že znak nevypĺňal celé preddefinované okno, napríklad vplyvom šumu pri nesprávnej segmentácii. Taktiež bol nespoľahlivý v prípade, že znaky boli mierne naklonené.

#### Porovnávanie vzorov s posúvaním okna

Ďalšou metódou klasifikácie znakov bol tzv. template matching (porovnávanie vzorov). Vytvorili sme základný binárny obraz, v ktorom boli všetky porovnávané znaky zoradené do riadku a farby invertované. Na tento účel bolo podobne ako v predošlom porovnávaní vzorov využité mierne modifikované písmo SPZ ČR TrueType Font. Následne bola použitá metóda z OpenCV *Imgproc.matchTemplate*. Funguje tak, že vyšetrovaný obraz je prekladaný neznámym znakom, ktorý sa následne po obraze posúva. Získaný je tak výstupný obraz, ktorý dosahuje maximum v bode, v ktorom došlo k najväčšej zhode daného regiónu a neznámeho znaku. Táto metóda vykazovala lepšie výsledky ako priame porovnávanie vzorov. Vzor slúžiaci pre porovnávanie môžeme vidieť na obr. 6.14.


Obr. 6.14: Vzor použitý pri metóde hľadania zhody, template matching

## 6.5 Výber algoritmov pre finálne riešenie

Na základe rozboru výsledkov úspešnosti z jednotlivých algoritmov bolo uskutočnené rozhodnutie pre výber algoritmov do finálnej aplikácie. Tieto algoritmy boli ďalej vylepšované, aby bola dosiahnutá čo najlepšia úspešnosť každého z nich.

### Lokalizácia regiónu EČV

Na lokalizáciu som sa rozhodol použiť viackrokové riešenie detekcie regiónov popísané v kapitole 6.4.1.

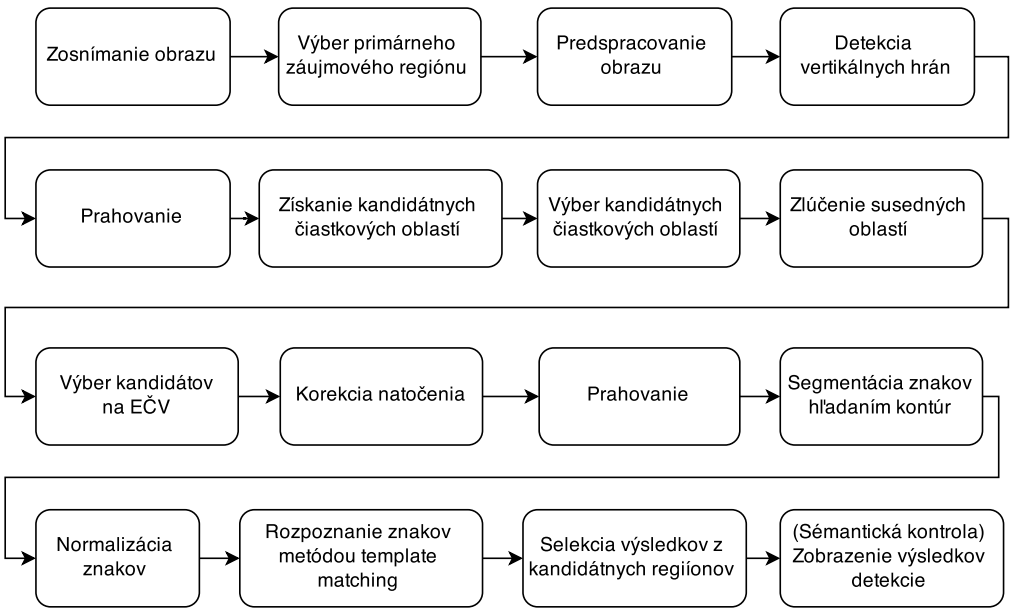
### Segmentácia znakov

Na segmentáciu znakov som vybral algoritmus zostavený z prahovania a následného vyhľadávania kontúr v obraze, ako je uvedený v kapitole 6.4.3. Znaky boli následne normalizovalé na zvolený rozmer.

### Rozpoznávanie znakov

Posledným krokom v reťazci rozpoznania EČV je rozpoznanie jednotlivých znakov. Na tento účel som využili metódu *template matching* popísanú v kapitole 6.4.4

### Postupnosť krokov rozpoznania EČV




Obr. 6.15: Výsledný vývojový diagram rozpoznania EČV

## 6.6 Sémantická kontrola

Úspešnosť rozpoznania znakov je možné zvýšiť tak, že bude vykonaná kontrola sémantiky daného reťazca. Štandardné EČV pre Českú republiku obsahuje sedem alfanumerických znakov, z ktorých je na prvom mieste číslica, nasleduje písmeno, na treťom mieste môže byť číslica alebo písmeno a na zvyšných miestach len číslice [16]. Prípadne je možné vykonať kontrolu na starší typ EČV, ktorý obsahuje na prvých troch miestach písmená a na ďalších štyroch číslice [16]. Kontrolou je možné vylúčiť nesprávne určené reťazce, napríklad číslica 2 mohla byť v určitých prípadoch nesprávne klasifikovaná ako písmeno Z, podobne číslica 8 mohla byť nesprávne klasifikovaná ako písmeno B. V prípade špeciálnych značiek nie je možné kontrolu sémantiky použiť.

## 6.7 Vyhľadávanie v databáze záujmových vozidiel

Pre uloženie záujmových EČV bola zvolená databáza SQL. Testovacia databáza bola vytvorená na serveri nástrojom phpMyAdmin. Názov tabuľky bol zvolený ako *LPtable*. Každý záznam obsahoval jedinečný identifikátor *id* a číslo EČV *plateNo*. API na prístup k databáze bolo navrhnuté tak, aby bolo možné v tejto databáze vyhľadávať na základe EČV. V neskorších štádiách bola databáza MySQL vymenená databázou SQLite. Toto riešenie prinieslo jednoduchšie spravovanie databázy a vysokú portabilitu. Z pohľadu aplikácie nebolo potrebné nič meniť vďaka použitému API, ktoré tieto systémy oddelilo. Bol tiež odstránený identifikátor *id*, keďže ako jedinečný identifikátor môžeme použiť EČV, ktoré je v rámci Českej republiky jedinečné.

### 6.7.1 PHP API na vyhľadávanie v databáze

Pre zjednodušenie prístupu k databáze bolo vytvorené API na báze PHP, ktoré umožňovalo prístup do databázy nie len Android aplikácii, ale potencionálne aj iným obslužným webovým nástrojom. Na strane servera sa nachádzal súbor *index.php*, ktorý predstavoval vstupnú bránu pre všetky požiadavky typu POST. Na začiatok sa skontrolovalo aký typ požiadavky bol zvolený. Boli implementované tri typy požiadaviek: *check*, ktorá slúžila na overenie prítomnosti konkrétneho EČV v databáze, *add* na pridávanie nových EČV do databázy a *remove* na odstránenie EČV z databázy. Pomocou direktívy include bol pripojený súbor *include/Database.php*, ktorý obsahoval triedu zdedenú po *SQLite3* v ktorej sa nachádzali metódy na vyhľadanie EČV v databáze, ich pridávanie a odoberanie.

### 6.7.2 Príklad vyhľadania EČV v databáze pomocou API

Vývoj a prvé testovanie bolo uskutočnené v doplnku POSTMAN pre prehliadač Chrome, ktorý umožňuje odosielať požiadavky rôzneho typu, zahŕňajúc GET aj POST. Najprv bolo potrebné zadať adresu k súboru *index.php* a vybrať metódu POST. Pre metódu typu POST bol vybraný formát *x-www-form-urlencoded* s pármi kľúč-hodnota. Prvý kľúč predstavoval typ požiadavky a jeho hodnota bola *check*, pre vyhľadanie EČV v databáze. Druhý kľúč predstavoval číslo EČV a jeho hodnota bola nastavená na *5M21234*. Metóda POST:

```
POST /android/index.php HTTP/1.1
Host: www.example.com
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
tag=check&lp=5M21234
```

Kladná odpoveď znamená, že EČV bolo v databáze nájdené.

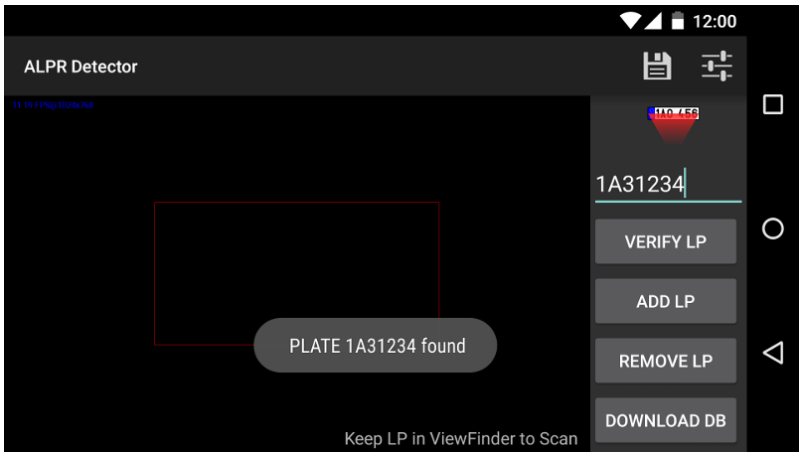
```
{
    "tag": "check",
    "success": 1,
    "error": 0
}
```

Chybové hlásenie uvádza, že dané EČV sa v databáze nenašlo.

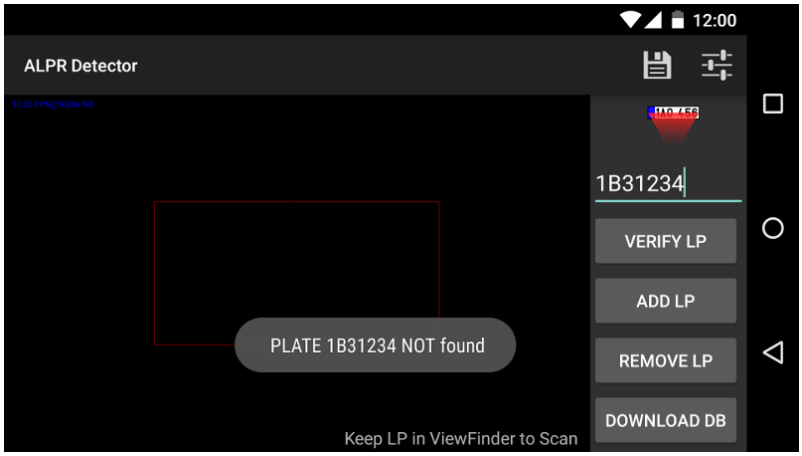
```
{
    "tag": "check",
    "success": 0,
    "error": 1,
    "error_msg": "LP_not_found!"
}
```

#### Implementácia v Android aplikácii

Vyhľadávanie je iniciované stlačením tlačidla *VERIFY LP*, s ktorým je vygenerovaný *AsyncTask*, ktorý beží nezávisle ako ďalšie vlákno. Týmto spôsobom je vyriešené blokovanie užívateľského prostredia pri čakaní na odoslanie požiadavky a prijatie odpovede. Požiadavka je typu HTTP POST s párovými parametrami typu kľúč-hodnota. Po úspešnom odoslaní požiadavky aplikácia čaká na odpoveď vo formáte JSON, ktorá je následne parsovaná. Kontroluje sa parameter *success*, ktorý je v prípade nájdenia EČV v databáze nastavený na hodnotu 1 v opačnom prípade na hodnotu 0. Po spracovaní odpovede, je výsledok odovzdaný UI vláknu a zobrazený pomocou *Toast* správy.




Obr. 6.16: Vyhľadávanie EČV v databáze, EČV nájdená




Obr. 6.17: Vyhľadávanie EČV v databáze, EČV nenájdená

Podobným spôsobom sú riešené požiadavky na pridávanie a odoberanie EČV z databázy. Každá požiadavka má teda vlastný *AsyncTask*.

### 6.7.3 Online/offline použitie

Z požiadaviek na aplikáciu uvedených v kapitole 5 vyplýva, že aplikácia musí byť schopná vyhľadávania EČV v online aj offline databáze. Vyhľadávanie v režime online zabezpečuje aktuálnosť získavaných údajov. Aby bolo možné takéto vyhľadávanie uskutočniť, je potrebné mať k dispozícii pripojenie do siete internet. Nie je dôležité, či sa zariadenie pripája pomocou wifi, alebo mobilného dátového pripojenia. V prípade, že pripojenie do siete internet nie je k dispozícii, nie je možné dané požiadavky odosielať na server. V tomto prípade je potrebné mať k dispozícii offline databázu uloženú v zariadení. Offline databáza je uložená v internej pamäti zariadenia a nie je do nej povolený zápis.

#### Aktualizácia databázy

Pred použitím offline databázy, je potrebné stiahnutie aktuálnej verzie zo servera. Pokiaľ sa offline databáza v zariadení nenachádza, užívateľ je na túto skutočnosť upozornený *Toast* správou. Ak sa databáza už v zariadení nachádza, ale je neaktuálna, je možná jej aktualizácia. Novšia verzia v tomto prípade nahradí databázu, ktorá sa nachádza v zariadení. Aktualizovať databázu je možné stlačením tlačidla *DOWNLOAD DB*. Po jeho stlačení sa spustí *AsyncTask* s názvom *DownloadTask*. Aby sa zabezpečilo úspešné stiahnutie databázy, ktoré by bolo je nezávislé na akciách užívateľa, je potrebné požiadať o *WAKE LOCK*.

```
PowerManager pm =
(PowerManager) context.getSystemService(Context.POWER_SERVICE);
mWakeLock = pm.newWakeLock(
  PowerManager.PARTIAL_WAKE_LOCK,getClass().getName());
mWakeLock.acquire();
```

Následne je možné pristúpiť k samotnému stiahnutiu databázy. Databázový súbor je získaný odoslaním požiadavky na */download.php*. Do tejto požiadavky by bolo možné doplniť ďalšie parametre, ako napríklad zabezpečovací token. Bezpečnosť API je bližšie rozoberaná v časti 6.7.5. Sťahovanie databázy prebieha v metóde *doInBackground*, čím je zabezpečené, že nebude blokované vlákno používateľského prostredia. Aby mohol byť užívateľ oboznámený s výsledkom sťahovania a tiež aby mohol byť inicializovaný modul na prístup k databáze, bolo potrebné vytvoriť callback *DbDownloadCallback*, v ktorom sa volá metóda:

```
onDbDownloadCompleted(Boolean success)
```

Pokiaľ bolo sťahovanie úspešné, inicializuje sa modul na prístup k databáze.

### 6.7.4 Testovanie databázy

Pri vývoji bola najprv použitá databáza MySQL, ktorá bola dostatočne rýchla aj pri veľkom množstve záznamov. Bolo však potrebné otestovať, či je možné nahradiť ju databázou SQLite3 na strane servera a tiež v zariadení ako lokálnu databázu. Na tento účel bola vytvorená testovacia databáza, ktorá obsahovala 10 000 náhodne generovaných reťazcov v dĺžke sedem znakov, ktorá prislúcha dĺžke reťazca EČV. Následne bolo z aplikácie odoslaných 10 nezávislých požiadaviek s dostatočným časovým odstupom na server. Podobne bolo odoslaných 10 požiadaviek do lokálnej databázy uloženej v zariadení a bol sledovaný čas úspešného spracovania požiadavky. Výsledky sú uvedené v tabuľke 6.1.

Aby bolo možné vyhľadávanie v databáze aj bez pripojenia na internet, je možne vrámci aplikácie stiahnuť databázu lokálne. Veľkosť databázy bola 312 kB vďaka

čomu bolo objemovo nenáročné jej sťahovanie do zariadenia. Čas stiahnutia databázy závisí predovšetkým na rýchlosti pripojenia. Pri pripojení prostredníctvom wifi bola databáza stiahnutá vždy do 1 s. Pri použití mobilného pripojenia trvalo stiahnutie databázy bežne 2 s, no pri slabšom signále aj viac ako 10 s. Z týchto zistení vyplýva, že aj keby bola databáza rozsiahlejšia, prípadne by sa pridali ďalšie sledované parametre o vozidle, nie len EČV, stále by bola databáza SQLite3 vhodným riešením.

| | Online databáza | Offline databáza |
|---|---|---|
| Najkratší čas spracovania [ms] | 44 | 6 |
| Najdlhší čas spracovania [ms] | 920 | 51 |
| Priemerný čas spracovania [ms] | 346 | 26 |

Tab. 6.1: Časy úspešného spracovania požiadavky z celkového počtu 10 požiadaviek

### 6.7.5 Bezpečnosť API

Do uvedeného funkčného riešenia vyhľadávania, pridávania, odoberania EČV a sťahovania databázy by bolo možné zakomponovať bezpečnostné prvky, ktoré by bránili neoprávnenému prístupu do databázy. Lokálna databáza je uložená v internom úložisku zariadenia, priestore vyhradenom pre aplikáciu a nie je možné do neho pristupovať z iných aplikácii. Spojenie na server by mohlo byť zabezpečené protokolom *HTTPS*, pričom na serveri by sa nachádzal privátny a v aplikácii verejný kľúč. API môžeme zabezpečiť tak, že v rámci aplikácie by bolo potrebné prihlásenie a údaje by boli dostupné iba autorizovaným užívateľom. Mohol by tak vzniknúť komplexný systém s viacerými užívateľmi, ktorí by mali rôzne oprávnenia. Bežní užívatelia by mohli EČV v databáze iba overovať a privilegovaní by mohli do databázy zapisovať, teda pridávať a odoberať EČV. Tieto navrhované riešenia sú však mimo rozsahu tejto práce.

## 6.8 Používanie aplikácie

Aplikácia je schopná lokalizácie a rozpoznania EČV vozidiel pre Českú republiku, ale v mnohých prípadoch je ju možné použiť aj pre EČV Slovenskej republiky. Používateľ aplikácie sa postaví vo vzdialenosti 1-2 m pred, alebo za vozidlom, pričom zariadenie drží v horizontálnom smere stabilne, bez nadmerného trasenia. Do hľadáčika nasmeruje EČV vozidla a počká na zaostrenie kamery. V prípade, že má zapnutý *Autoscan* režim, EČV bude po rozpoznaní automaticky skontrolovaná na serveri dopytom na databázu. V prípade, že pripojenie na internet nie je dostupné,

aplikácia vykoná dopyt na lokálnu databázu. Z tohto dôvodu, je vhodné databázu priamo z aplikácie pravidelne aktualizovať. Ako vyplýva z testovania databázy 6.7.4, aj pri väčšom množstve záznamov je databázový súbor dostatočne malý, aby mohol byť stiahnutý aj prostredníctvom mobilného dátového pripojenia.

V prípade, že rozpoznané EČV neobsahuje štandardných 7 alfanumerických znakov, aplikácia aj v režime *Autoscan* nevykoná dopyt na databázu automaticky. Vtedy užívateľ stlačí náhľad EČV v ľavom dolnom rohu a číslo EČV sa prekopíruje do poľa pre vyhľadávanie. Následne je možné vykonávať všetky operácie, ako pri štandardnom EČV.

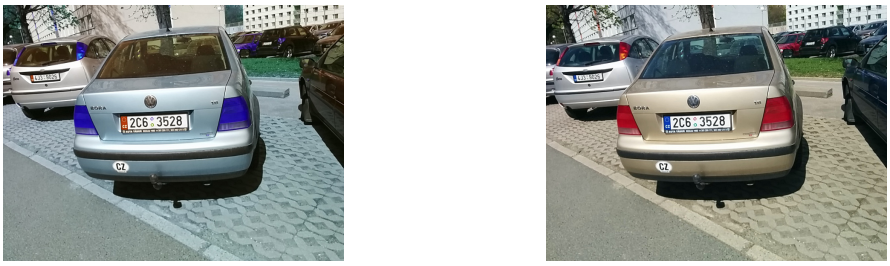
Pokiaľ je zapnuté kontrolovanie sémantiky EČV, zvyšuje sa tým úspešnosť rozpoznania. Užívateľ si môže zvoliť, či povolí aj kontrolu pre starší typ EČV. V prípade, že sa na určenom mieste nenachádza požadovaná číslica, alebo písmeno podľa predpisu, EČV bude ignorované a rozpoznané znovu.

V rámci aplikácie je možné vyhotovenie záznamu aktuálneho pohľadu. Užívateľ si môže po kliknutí na tlačidlo uloženia snímok uložiť do externého úložiska zariadenia. Pri zvolení režimu statického rozpoznávania snímkov je možné tieto snímky dodatočne rozpoznať. Táto voľba bola využívaná predovšetkým pri vývoji aplikácie. Okrem toho je možné v nastaveniach zapnúť ukladanie snímkov počas spracovania. Môžeme tak vidieť vertikálne hrany, lokalizované znaky, segmentované a prahované znaky. Na základe týchto výstupov je možné analyzovať, prečo dané EČV nebolo lokalizované, znaky neboli správne segmentované, prípadne zistiť, prečo bol konkrétny znak tak klasifikovaný.

# 7 VYHODNOTENIE

Na účely testovania boli vytvorené dve testovacie množiny. V prvom prípade boli zvolené vhodné svetelné podmienky, v druhom prípade bolo snímanie uskutočnené za zmiešaných podmienok, pri priamom slnečnom svite aj za šera. Prvá testovacia množina obsahovala 150 snímkov, druhá 120.

Záznam bol v oboch prípadoch uskutočnený smartfónom Nexus 5, priamo v aplikácii. Snímky mali rozlíšenie $1024 \times 768$ pixlov a boli ukladané vo farebne schéme RGBA. Snímky ktoré získame pomocou knižnice OpenCV, majú štandardnú farebnú schému BGRA, teda kanály B a R sú prehodené. Z tohto dôvodu bola potrebná konverzia. Rozdielnosť snímkov môžeme vidieť na obr. 7.1.



(a) (b)

Obr. 7.1: Porovnanie (a) pôvodného snímku v priestore BGRA (b) snímku po konverzii do priestoru RGBA

Snímky boli ukladané priamo zo živého zobrazenia *live preview*. Rozdiel od klasického fotografovania je v tom, že pri fotografovaní je k dispozícii dostatočný čas na ostrenie a rozlíšenie výsledného snímku je výrazne vyššie. V prípade použitého smartfónu Nexus 5, $3200 \times 2368$ pixlov. V počiatkoch vývoja aplikácie bola testovacia množina vytvorená pomocou klasického fotografovania. Tieto snímky boli veľmi kvalitné a nezodpovedali kvalite ktorú máme k dispozícii pri režime *live preview*.

## 7.1 Úspešnosť rozpoznania EČV

Pri testovaní úspešnosti boli snímky z testovacích množín nahrané do externého úložiska zariadenia a následne boli spustené detekčné algoritmy. Výstupom boli anotované snímky s vyznačenými kandidátnymi regiónmi a rozpoznaným EČV v spodnej časti.

Najprv bola vyhodnotená úspešnosť lokalizácie regiónu EČV. Regióny boli vo výstupných snímkoch vyznačené zelenou farbou. V prípade, že bolo kandidátnych regiónov vybratých viac a región EČV sa medzi nimi nachádzal, bola braná táto

detekcia ako úspešná, pretože v ďalšom kroku boli nevhodní kandidáti zahodení. Ak bol región EČV rozdelený na časti, táto detekcia bola vyhodnotená ako neúspešná, pretože v ďalšom kroku by bola získaná iba časť textu EČV.

Segmentácia znakov bola vyhodnotená ako úspešná, ak boli všetky znaky v EČV správne segmentované a v kroku rozpoznávania bolo k dispozícii sedem regiónov obsahujúcich znaky. Úspešnosť segmentácie bola počítaná iba zo snímkov, v ktorých boli úspešne lokalizované kandidátne regióny.

Úspešnosť rozpoznávania znakov bola určená zo všetkých správne lokalizovaných regiónov EČV. V prípade, že zlyhala segmentácia znaku, bolo považované aj rozpoznanie znaku za neúspešné. Ak boli na EČV rozpoznané správne len niektoré znaky, táto skutočnosť sa prejavila aj v úspešnosti, keďže úspešnosť bola počítaná zo všetkých znakov ktoré sa nachádzali v kandidátnych regiónoch.

V tabuľke 7.1 sú uvedené úspešnosti jednotlivých krokov rozpoznania EČV vozidla. Z uvedených výsledkov vyplýva, že, pri vhodných svetelných podmienkach je úspešnosť pomerne vysoká. Celková úspešnosť v tomto prípade dosiahla 94,0%.

| | Úspešnosť [%] |
|---|---|
| Lokalizácia EČV | 99,3 |
| Segmentácia znakov | 94,6 |
| Rozpoznanie znakov | 96,9 |
| Celkovo | 94,0 |

Tab. 7.1: Úspešnosť jednotlivých algoritmov rozpoznávania EČV pre prvú množinu

V druhej testovacej množine boli sťažené svetelné podmienky, čo sa prejavilo aj na úspešnosti jednotlivých krokov rozpoznania EČV. Výsledky je možné vidieť v tabuľke 7.2. Celková úspešnosť v tomto prípade dosiahla 60,8%.

| | Úspešnosť [%] |
|---|---|
| Lokalizácia EČV | 90,0 |
| Segmentácia znakov | 67,6 |
| Rozpoznanie znakov | 77,9 |
| Celkovo | 60,8 |

Tab. 7.2: Úspešnosť jednotlivých algoritmov rozpoznávania EČV pre druhú množinu

Tieto dve množiny boli zlúčené aby mohla byť vyhodnotená úspešnosť detekcie za rôznych podmienok. Výsledky je možné vidieť v tabuľke 7.3. Celková úspešnosť

rozpoznania EČV zlúčenej množiny dosiahla 79,3%. Ako z uvedených výsledkov vyplýva, najväčším problémom rozpoznania EČV je v tomto prípade chybná segmentácia znakov pri zhoršených podmienkach. Jedná sa predovšetkým o tieň vrhajúci pri snímaní zo zadnej strany vozidla, ktorý spôsobí, že sa znaky pri segmentácii spoja s obrysom EČV. Problémy tiež spôsobuje snímanie obrazu za šera, kedy je problematické zaostrenie. V tomto prípade by bolo možné použiť namiesto *live preview*, režim klasického fotografovania, prípadne režim HDR+.

| | Úspešnosť [%] |
|---|---|
| Lokalizácia EČV | 95,2 |
| Segmentácia znakov | 83,2 |
| Rozpoznanie znakov | 88,9 |
| Celkovo | 79,3 |

Tab. 7.3: Úspešnosť jednotlivých algoritmov rozpoznávania EČV pre obe množiny

### 7.1.1 Zhrnutie

Táto úloha bola náročnejšia v tom, že som sa rozhodol pre analýzu snímkov v reálnom čase. Použité algoritmy museli byť dostatočne efektívne a rýchle, aby bol dosiahnutý čo najvyšší počet analyzovaných snímkov za sekundu a zobrazenie bolo plynulé. Pri štandardnom vyhľadávaní EČV sa mi podarilo dosiahnuť takmer 10 fps, pri analýze EČV a spustení algoritmov OCR kleslo toto číslo na približne 2 fps. Bez spustených akýchkoľvek algoritmov bola obnovovacia frekvencia snímkov 15 fps.

Pri zapnutom režime *Autoscan*, predstavuje pre užívateľa výhodu možnosť vyhľadávania v databáze bez klikania na obrazovku. Stačí, že užívateľ nasmeruje do hľadáčika EČV a po načítaní sa uskutoční dopyt na server, pripadne v nedostupnosti pripojenia do siete internet do lokálnej databázy.

Z výsledkov uvedených v tabuľke 7.3 vyplýva, že celková úspešnosť rozpoznania EČV je 79,3%. Tieto výsledky sú získané zo statických snímkov a v praxi je úspešnosť vyššia. Je to z dôvodu, že pri snímaní rozpoznávame EČV kontinuálne a aj keď jeden cyklus detekcie zlyhá, v ďalšom behu už môže byť EČV rozpoznaná úspešne. Užívateľ tiež môže počas používania aplikácie korigovať získavanie snímku svojou polohou. K nesprávnej detekcii môže prísť pri režime *Autoscan* s automatickým vyhľadávaním v databáze, kedy sa pri pohybe zariadenia, keď snímok ešte nie je dostatočne zaostrený, snímok vyhodnotí s nesprávne rozpoznanými znakmi. Ak sa zapne sémantická kontrola, úspešnosť sa ešte zvýši. Pri rozpoznávaní statických snímkov táto kontrola nebola použitá z dôvodu, že z každej scény bol k dispozícii len jeden snímok.

# 8 ZÁVER

Cieľom tejto práce bolo vytvorenie aplikácie na automatické rozpoznanie EČV s použitím zariadenia s operačným systémom Android. Pre zjednodušenie práce so spracovaním obrazu som sa rozhodol použiť existujúcu multiplatformovú knižnicu OpenCV, kde som implementoval detekčné algoritmy. Preto je potrebné po inštalácii aplikácie nainštalovať tiež OpenCV Manager, ktorý obsahuje všetky potrebné komponenty. Aplikácia je navrhnutá tak, že obsahuje podporné knižnice pre staršie verzie systému Android.

Na začiatku som si definoval požiadavky na scénu a hardvér, uvedené sú v kapitole 5. Následne som pristúpil k samotnej realizácii. Algoritmy boli najprv vyvíjané a testované v programe Matlab. Rozhodol som sa pre použitie knižníc OpenCV, kde algoritmy boli spočiatku implementované v jazyku C++, no pre zrýchlenie vývoja bol neskôr zvolený jazyk Python. Funkčnosť knižníc bola pritom v oboch prípadoch rovnaká. Po odladení bol detekčný algoritmus portovaný pre Android v jazyku Java.

Pre aplikáciu som navrhol užívateľské prostredie, ktoré obsahuje v ľavej časti obraz z kamery s hľadáčikom a v pravej ovládacie komponenty, ktorými je možné EČV v databáze overovať, pridávať a odoberať. Tiež je tu možnosť sťahovania novej verzie offline databázy, keďže táto funkcionalita bola tiež súčasťou požiadavkov na aplikáciu.

Algoritmus lokalizácie a rozpoznania EČV je viackrokový. Lokalizácia je uskutočnená pomocou hľadania regiónov s vysokou hustotou vertikálnych hrán a ich spájania. Znaky sú segmentované prahovaním s následným vyhľadávaním kontúr. Po získaní znaku je na jeho rozpoznanie použitá metóda template matching, porovnávanie vzorov. Hľadá sa tak miesto najlepšej zhody vo vzore na obr. 6.14, pomocou hľadania maxima. Po nájdení maxima využijeme informáciu o jeho polohe na výber znaku z poľa, ktoré korešponduje so sledom znakov vo vzore. OCR som tým pádom implementoval vlastné, bez samostatných knižníc na rozpoznávanie znakov. Celý postup rozpoznávania EČV je zobrazený na obr. 6.15.

V aplikácii sa nachádza vstupné textové pole na overovanie EČV na serveri. Bolo tiež vytvorené PHP API pre prístup k databáze. V počiatkoch bola zvolená databáza typu MySQL, ale neskôr bola nahradená jej odľahčenou verziou SQLite. Toto riešenie zjednodušuje manažment databázy, pretože rovnaká databáza sa stiahne do internej pamäti zariadenia pre offline použitie. Databáza bola naplnená náhodne vygenerovanými EČV s počtom záznamov 10 000. Z testovania rýchlosti databázy 6.7.4 vyplýva, že SQLite je vhodným riešením a aj pri väčšom počte záznamov je vyhľadávanie v databáze rýchle a jej veľkosť umožňuje bezproblémové sťahovanie do zariadenia. Databáza v tejto práci obsahovala len EČV, no každý záznam by mohol obsahovať doplňujúce informácie plynúce z použitia, napríklad časovú plat-

nosť parkovania, platnosť povolenia na vstup, informácie o vozidle, či iné. Aplikácia pristupuje do databázy pomocou PHP API generovaním požiadavky typu *POST*. Implementoval som tri základné základné požiadavky, pridávanie, odoberanie a overovanie EČV. Aplikácia prijme odpoveď, v ktorej je prípade overovania informácia, či sa dané EČV v databáze nachádza, v prípade pridávania a odoberania informácia o úspešnosti vykonania akcie.

Úspešnosť detekčných algoritmov som overoval na testovacej množine, ktorú tvorilo celkovo 270 snímkov. Túto množinu som rozdelil na dve podmnožiny. V prvej boli podmienky pre snímanie EČV vhodné, v druhej boli tieto podmienky sťažené nevhodným nasvetlením scény. V prvom prípade som dosiahol úspešnosť lokalizácie 99,3%, segmentácie znakov 94,6% a rozpoznania znakov 96,9%. Celková úspešnosť rozpoznania EČV bola 94,0%. V druhej testovacej množine sa prejavil vplyv zhoršených podmienok, kedy úspešnosť lokalizácie bola 90,0%, segmentácie znakov 67,6% a rozpoznania znakov 77,9%. Celková úspešnosť detekcie dosiahla 60,8%. Pri zahrnutí všetkých prvkov z oboch množín som dospel k nasledovným výsledkom. Úspešnosť lokalizácie EČV bola 95,2%, segmentácie znakov 83,2% a rozpoznania znakov 88,9%. Celková úspešnosť dosiahla 79,3%. Z týchto výsledkov vyplýva, že úspešnosť sa rapídne zhoršuje komplikovanými svetelnými podmienkami. Problém nastáva za šera, kedy kamera zariadenia nie je schopná správneho ostrenia. Problémy tiež spôsobuje v istých prípadoch priamy slnečný svit, kedy sú vrchné časti znakov tieňom opticky spojené s EČV. Keďže je použité spracovanie obrazu v reálnom čase, úspešnosť sa môže zvýšiť opakovaným behom algoritmov, prípadne sémantickou kontrolou 6.6.

Prácu s aplikáciou uľahčuje režim *Autoscan*, kedy je okrem kontinuálneho snímania pri korektnom formáte načítaného EČV aj automaticky vykonaný dopyt na databázu, v prípade dostupnosti internetového pripojenia na online databázu, v opačnom prípade na offline databázu uloženú v internom úložisku zariadenia. Užívateľ tak nemusí stláčať žiadne tlačidlá, iba smeruje hľadáčik na EČV.

V prípade nasadenia aplikácie by bolo potrebné doplniť zabezpečenie API, ako je uvedené v časti 6.7.5. Jedným z riešení je prihlásenie do aplikácie, čím by mohol vzniknúť systém užívateľov s rôznymi stupňami oprávnení. Bežní užívatelia by mohli iba vykonávať dopyty na overenie EČV a sťahovať databázu lokálne a privilegovaní užívatelia by mohli do databázy záznamy pridávať aj ich odoberať. Databáza je v tejto práci zabezpečená uložením do priestoru vyhradeného pre aplikáciu, ostatné aplikácie k nej preto nemajú prístup.

Vylepšenie úspešnosti rozpoznania EČV by bolo možné vylepšením použitých algoritmov, predovšetkým segmentáciu znakov, ktorá najviac ovplyvnila celkovú úspešnosť rozpoznania.

# LITERATÚRA


[1] ŠONKA, M. , V. HLAVÁČ, R. BOYLE: *Image Processing, Analysis, and Machine Vision, 3rd Edition, Thomson* 2007, ISBN 049508252X

[2] Android Open Source Project: *Codenames, Tags, and Build Numbers* [online]. 2015-05-09 [cit. 2015-05-09]. Dostupné z URL: *https://source.android.com/source/build-numbers.html*

[3] Android Developer Portal: Application Fundamentals [online]. 2015-05-09 [cit. 2015-05-09]. Dostupné z URL: *http://developer.android.com/guide/components/fundamentals.html*

[4] OpenCV 2.4.11.0 documentation [online]. 2015-05-09 [cit. 2015-05-09]. Dostupné z URL: *http://docs.opencv.org/platforms/android/service/doc/UseCases.html*

[5] Bloomberg Businessweek: Google Buys Android for Its Mobile Arsenal [online]. 2005-08-16 [cit. 2015-05-09]. Dostupné z URL: *http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal*

[6] AndroidHive: Android Login and Registration with PHP, MySQL and SQLite [online]. 2012-01-31 [cit. 2015-05-09]. Dostupné z URL: *http://www.androidhive.info/2012/01/android-login-and-registration-with-php-mysql-and-sqlite/*

[7] Oracle Corporation: MySQL 5.7 Reference Manual [online]. 2015-05-09 [cit. 2015-05-09]. Dostupné z URL: *http://dev.mysql.com/doc/refman/5.7/en/*

[8] PETYOVSKÝ, P.: *MPOV prednášky – Reprezentace a vlastnosti obrazových dat.* 2014-09 [cit. 2015-05-09].

[9] ŠIKUDOVÁ, E. , Z. ČERNEKOVÁ , V. BENEŠOVÁ, Z. HALADOVÁ, J. KUČEROVÁ: *Počítačové videnie. Detekcia a rozpoznávanie objektov*, Wikina 2013, ISBN: 978-80-87925-06-5

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

[11] TARABEK, P.: *A Real-Time License Plate Localization Method Based on Vertical Edge Analysis* 09/2012, ISBN: 978-83-60810-51-4

[12] MENDES, P. et al.: *Towards an Automatic Vehicle Access Control System: License Plate Location* 10/2011, ISBN: 978-1-4577-0652-3

[13] IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY *Automatic License Plate Recognition (ALPR): A State-of-the-Art Review.* VOL 32, NO. 2, 02/2013, ISSN: 1051-8215

[14] KIM. J., HAN. Y., HAHN. H., *Character Segmentation Method for a License Plate with Topological Transform* [online], World Academy of Science, Engineering and Technology. VOL 3, 2009-08-25 [cit. 2015-05-07]. Dostupné z URL: *http://waset.org/publications/1727/character-segmentation-method-for-a-license-plate-with-topological-transform*

[15] SUZUKI, S., KEIICHI, A., *Topological Structural Analysis of Digitized Binary Images by Border Following* [online], Computer vision, graphics, and image processing, 30, 23-46, 1985 [cit. 2015-05-09]. Dostupné z URL: *http://tpf-robotica.googlecode.com/svn-history/r397/trunk/Vision/papers/SA-CVGIP.PDF*

[16] Ministerstvo dopravy České republiky: *Vyhláška Ministerstva dopravy a spojů ze dne 29. června 2001 o registraci vozidel* [online]. 2001-06-29 [cit. 2015-05-14]. Dostupné z URL: *http://www.mdcr.cz/NR/rdonlyres/99FBC861-81DB-4BFA-8925-F4C58C9FEDB3/0/v24301.rtf*

# ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

SDK Softvérový vývojový kit - Software development kit

NDK Natívny vývojový kit - Native development kit

ADT Vývojové nástroje pre Android - Android Development Tools

RZ Registrační značka

SPZ Státní poznávací značka

EČV Evidenčné číslo vozidla

OCR Optické rozpoznanie znakov - optical character recognition

SQL Štruktútovaný vyhľadávací jazyk - Structured Query Language

UI Používateľské rozhranie - User Interface

API Rozhranie pre programovanie aplikácií - Application programming interface

# ZOZNAM PRÍLOH

# A PRILOHA A

## A.1 Príklady úspešne rozpoznaných EČV


5C00762


2L18018

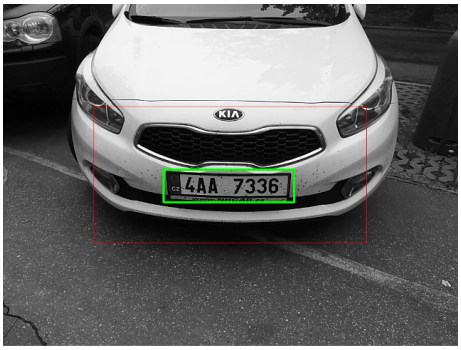

MT310BY


2Z50953


8B04045


PK056DE

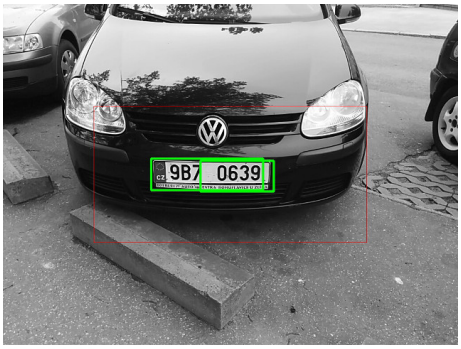

DS820EM




4AA7336


TN900BZ


4Z82558



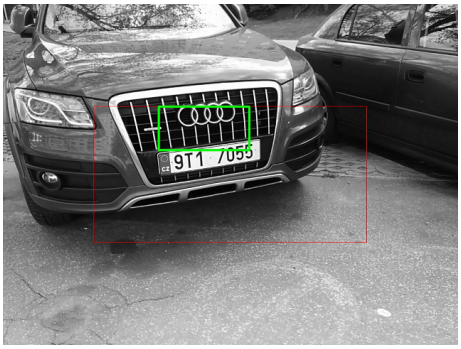

9B70639


6B66251


ZA615F0


9B11810

## A.2 Príklady neúspešne rozpoznaných EČV




X


7U3


5U9529




8

# B DVD S ANOTOVANÝMI SNÍMKAMI, INŠTALAČNÝM SÚBOROM APLIKÁCIE, ZDROJOVÝMI SÚBORMI APLIKÁCIE, SÚBORMI OSAHUJÚCIMI API PRE PRÍSTUP K DATABÁZE A VIDEOM ZOBRAZUJÚCIM DETEKCIU EČV A VYHĽADÁVANIE V DATABÁZE V REÁLNOM ČASE