VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁNÍ REGISTRAČNÍ ZNAČKY

LICENSE PLATE RECOGNITION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ONDŘEJ MRHAČ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAN NAVRÁTIL, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Mrhač Ondřej**
Obor: Informační technologie
Téma: **Rozpoznání registrační značky**
License Plate Recognition
Kategorie: Počítačová grafika

Pokyny:

1. Seznamte se s knihovnou OpenCV a základními postupy pro zpracování obrazu a videa.
2. Nastudujte problematiku detekce a rozpoznání registračních značek z videa.
3. Popište algoritmy pro detekce a rozpoznání.
4. Implementujte vybrané algoritmy.
5. Navrhněte vzorovou aplikaci, ve které demonstrujete funkčnost vybraných algoritmů.
6. Vyhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu.

Literatura:

- Richard Szeliski: Computer Vision: Algorithms and Applications, http://szeliski.org /Book/, 2010

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

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

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

Vedoucí: **Navrátil Jan, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

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

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zaobírá problematikou detekování, rozpoznávání státních poznávacích značek a následnou implementací pro konkrétní zařízení i.MX 6 Series od společnosti NXP semiconductors s.r.o. S využitím knihovny OpenCV a Tesseractu byl vytvořen vzorový program na detekci a rozpoznání registrační značky, který byl úspěšně zprovozněn na tomto zařízení. Následně byl podroben měření rychlosti běhu na počítači a na daném zařízení. Výsledkem bylo nalezení nejnáročnějších fází programu a dle toho byla navržena další možná vylepšení a rozšíření.

Abstract

This thesis talks about problematics of licence plate detection, licence plate recognition and my implementation for device i.MX 6 Series of NXP semiconductors s.r.o company. Model program for licence plate detection and recognition is written with help of OpenCV library and engine Tesseract and it's successfully put into operation on this device. Afterwards program was measured his runtime on PC and i.MX 6 Series device and those measurements were compared. At the end of this thesis were found the most demanding parts of the program. Future changes and improvements were proposed.

Klíčová slova

detekce objektů, rozpoznávání registrační značky, optické rozpoznávání znaků, zpracování obrazu, počítačové vidění, OpenCV, Tesseract OCR, defisheye

Keywords

object detection, license plate recognition, optical character recognition, image processing, computer vision, OpenCV, Tesseract OCR, defisheye

Citace

MRHAČ, Ondřej. *ROZPOZNÁNÍ REGISTRAČNÍ ZNAČKY*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Navrátil Jan.

ROZPOZNÁNÍ REGISTRAČNÍ ZNAČKY

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Navrátila, Ph.D. Další informace mi poskytla Ing. Irina Trukhina ze společnosti NXP semiconductors s.r.o. Na konci práce jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ondřej Mrhač
16. května 2017

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Janu Navrátilovi, Ph.D. za jeho odborné vedení, cenné rady a přátelský přístup. Poděkování také patří Ing. Irině Trukhině a společnosti NXP semiconductors s.r.o za poskytnuté konzultace a zdroje, a také Ing. Romanu Juránkovi, Ph.D. za jeho odborné rady. V neposlední řadě chci poděkovat své rodině za veškerou podporu nejen při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Detekce registračních značek je téma, které se v dnešní době stává daleko rozšířenější. S nárůstem různých bezpečnostních prvků a zařízení je možné se s kontrolou poznávacích značek setkat při sledování provozu, hlídání poplatků za parkování, zaznamenávání statistik apod. Nejčastěji jde o variantu, kdy nějaká bezpečnostní kamera zaznamenává obraz, který je předáván počítači, a ten se v tomto obraze snaží registrační značku lokalizovat a rozpoznat.

Hlavní motivací při tvorbě této práce bylo ve snaze nahradit klasické výkonné počítače pro zpracování něčím možná méně výkonným, ale za to více přenositelnějším. Pokud by podobné programy nebyly závislé na zařízení, které by potřebovalo velké napájení a výkon, a přitom byly podstatně skladnější, bylo by možné dostat toto téma do míst, ve kterých se s ním ještě nesetkáváme.

Příklad. Velké množství automobilů je dnes vybaveno kamerovým systémem. Ty prozatím slouží především k tomu, aby člověk mohl bezpečně zaparkovat. Jen ve velmi malém počtu případů se využívají např. ke sledování jízdního pruhu. Většinou se tak děje u výrazněji dražších automobilů. Kdyby však bylo možné zprovoznit podobnou aplikaci na daleko levnějším zařízení, už by podobné výhody nemusely být jen výsadou dražších vozů.

Takový konkrétní příklad se však najde i pro detekci registračních značek. Bohužel se stává, že někteří řidiči srazí chodce a z místa nehody ujedou. Následné dohledávání je poté už složitější. Pokud by však bylo poblíž vozidlo, které by bylo vybaveno kamerovým systémem – a umělo by detekovat a rozpoznat registrační značku, následné hledání viníka by bylo podstatně rychlejší.

Cílem této práce je tedy vytvořit vzorovou aplikaci na rozpoznávání registračních značek, která poběží na takovém zařízení nesoucí název i.MX 6 Series od společnosti NXP semiconductors s.r.o. Podstatou je její uzpůsobení pro tuto činnost, zprovoznění a především změření a porovnání dosažených výsledků běhu aplikace na konkrétním počítači a daném zařízení. Práce si dále klade za cíl tyto výsledky vyhodnotit a na jejich základě navrhnout různá vylepšení – ať už z hlediska výkonu či funkčnosti. Výsledná aplikace ověří, zda je vůbec reálné na takové zařízení tento typ aplikace vytvářet. Pokud ano, potom může tento program posloužit jako základní stavební jednotka, na kterou bude s přihlédnutím k výsledkům měření možné nabalovat různá vylepšení a rozšíření.

Práce je rozdělena do několika kapitol. V kapitole 2 se nachází popis této problematiky a stručně sděluje veškeré použité techniky při tvorbě a zprovozňování výsledné aplikace. Kapitola 3 se věnuje použité implementaci. Je rozdělena na takové dvě hlavní části, kdy první část je věnována krátkému náhledu na návrh programu, zatímco ta druhá se zaobírá přímo konkrétní implementací. V kapitole 4 jsou popisovány veškeré experimenty a testy prováděné s tímto programem. Hlavní důraz je zde kladen na výstupy jednotlivých mě-

ření mezi počítačem a zařízením i.MX 6 Series a jejich následné srovnání. Veškeré zjištění je následně shrnuto a zhodnoceno v kapitole 5, kde se kromě samotné evaluace nachází i podněty a nápady na další vylepšení.

Jelikož se tato práce zabývá registračními značkami, tak v neposlední řadě se v příloze A nachází specifikace českých poznávacích značek, se kterými má za úkol výsledná aplikace pracovat. Příloha B poté obsahuje jednoduché vzorové ukázky, jak program spustit a vyzkoušet.

Kapitola 2

Zpracování obrazu

Jelikož se v rámci implementace této práce využívá mnoho různých metod a nástrojů z oblasti počítačového vidění a celkově zpracování obrazu, tak tato kapitola obsahuje základní informace k pochopení jak jednotlivých metod, které byly při návrhu a tvorbě programu použity, tak i samotné problematiky. Tato kapitola tedy slouží jako teoretická příprava k jednotlivým praktickým aspektům popisovaným v následujících kapitolách.

2.1 Počítačové vidění

Lidé využívají ke sledování okolního světa svoje oči a mozek. Počítačové vidění [18] je věda, která si klade za cíl dosáhnout v tomto ohledu stejných – ne-li lepších výsledků. V případě počítačové vidění jsou však oči a mozek nahrazeny konkrétní výpočetní technikou, která zpracovává určité obrazové informace. Pod těmito obrazovými informacemi je možné si představit téměř cokoliv. Může se jednat o lidskou tvář, hustotu dopravního provozu, biologické vzorky, nějaká lidská činnost, a mnoho dalšího. Hlavní charakteristickou vlastností strojového vidění je zpracování obrazových informací pomocí kamerového systému a následným podrobením získaných materiálů specifickým úlohám, jako je např. kontrola kvality vstupu a výstupu, detekce tvarů, počítání objektů, hledání vad apod.

2.1.1 Vývoj

V nedávné historií (datující se od 60. let minulého století) bylo počítačové vidění spojováno se složitými, nespolehlivými a především drahými technologiemi. To veskrze bránilo hromadnému rozšíření. Prvně bylo zapotřebí, aby svého pokroku dosáhly obrazové senzory a samotná výpočetní technika (hlavně paměť a výkon).

Díky tomuto rozvoji se strojové vidění stávalo postupně dostupnější a v dnešní době i (v konkrétních oblastech) výhodnější. Uveďme si příklad. Máme strojové vidění, které nám slouží ke kontrole kvality nějakých výrobků. Zatímco v dřívějších dobách bylo levnější a spolehlivější provádět kontrolu kvality výrobků lidmi, tak v dnešní době je lepší tuto práci svěřit právě těmto technologiím. Proč? Při tom, jak je v dnešní době kladen důraz na kvalitu jednotlivého kusu, tak není možné vyloučit chybu kontroly zapříčiněnou selháním lidského faktoru. Další nespornou výhodou je, že tyto systémy mohou pracovat nepřetržitě 24 hodin denně – 7 dní v týdnu. Právě díky těmto výhodám se počítačové vidění stává v dnešní době stále lepším a dostupnějším řešením.

2.1.2 Použití

Přestože se počítačové vidění řadí mezi relativně nové obory, tak se dnes používá ve spoustě různých profesních odvětvích. Setkáme se s ním například v medicíně, kde slouží ke zpracování obrazových dat (RTG snímků, angiografie, mikroskopu, tomografie, atd.) za účelem stanovení medicínské diagnózy pacienta. Díky tomu je možné získávat informace o velikosti orgánů, toku krve, detekci nádorů, arteriosklerózy apod. Mezi další oblasti použití se řadí průmysl – v tomto odvětví se strojové vidění používá ke kontrole kvality výrobků, dále se užívá při navádění autonomních vozidel, ponorek, bezpilotních letounů, a další.

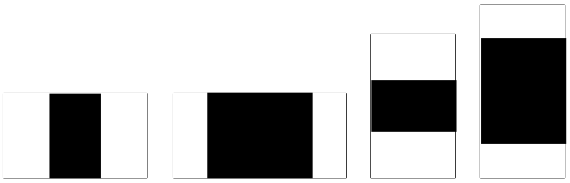
Vyjmenováváním veškerých oborů, ve kterých se dnes počítačové vidění používá, by bylo možné trávit hodně dlouhou dobu. Pokud bych to měl nějak shrnout, tak se strojovým viděním je možné se setkat v oblastech zahrnující ovládání procesů, detekci jevů, organizaci informací, interakcí mezi člověkem a počítačem nebo modelováním objektů a prostředí.

2.2 Haarovy příznaky

V případě trénování kaskádového klasifikátoru jsou na začátku trénovacího procesu vygenerovány Haarovy příznaky [13]. Jedná se o obrazové příznaky, které slouží jako základ slabým klasifikátorům. Poprvé tuto metodu při trénování použila dvojice Paul Viola a Michael Jones ve své práci týkající se detekce obličeje v roce 2001 [19].

Podle počtu obdélníkových oblastí je možné jednotlivé příznaky dělit. Příznak se dvěma obdélníkovými oblastmi se nazývá hranový příznak (Obrázek 2.1), se třemi oblastmi čárový příznak (Obrázek 2.2) a příznak se čtyřmi obdélníkovými oblastmi se nazývá diagonální příznak (Obrázek 2.3).

Obrázek 2.1: Ukázka hranových příznaků.



Obrázek 2.2: Ukázka čárových příznaků.

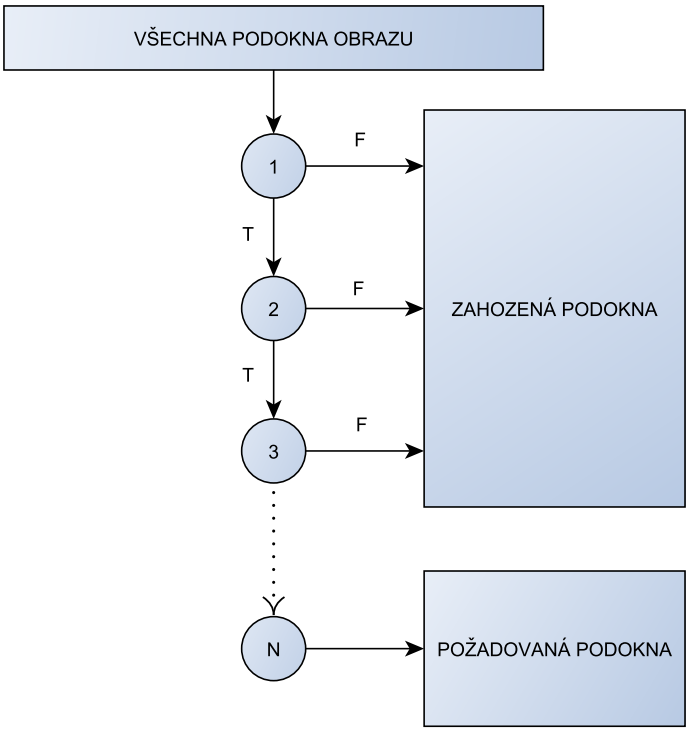
Obrázek 2.3: Ukázka diagonálních příznaků.

Základní princip trénování klasifikátoru s pomocí Haarových příznaků je takový, že po zadaném snímku se posouvá rámec, který se vždy rozdělí na sousedící se obdélníkové oblasti – na Haarovy příznaky. V těchto vybraných oblastech se následně sečítá jas veškerých pixelů. Poté se hodnota samotného příznaku vypočítá jako součet všech pixelů spadající do bílé části, od kterého se odečte součet všech pixelů spadajících do černé části. Tyto příznaky jsou postupně použity na celý obraz. Současně s tím však měníme velikosti jednotlivých příznaků z velikosti 1×1 až na velikost rovnající se samotnému obrazu. Jelikož se touto cestou získá velmi velké množství příznaků, které by ovlivnilo rychlost celého výpočtu, využívá se klasifikační algoritmus Adaboost [20], který vybere důležité příznaky. Teprve tento redukovaný počet příznaků (nijak neovlivňující výslednou kvalitu) je následně použit při detekci.

2.3 Kaskádový klasifikátor

Kaskádový klasifikátor je klasifikátor, který umožňuje rychle a robustně detekovat objekty v obraze. Tento klasifikátor používá výše uvedené Haarovy příznaky – už však pouze ty, které jsou označené jako relevantní klasifikačním algoritmem Adaboost.

Samotná podoba tohoto klasifikátoru je již zmíněna v jeho názvu. Základem je kaskáda, která funguje na principu rozhodovacího stromu (Obrázek 2.4).



Obrázek 2.4: Princip fungování kaskádového klasifikátoru

Na vstupu této kaskády jsou veškerá podokna obrazu. Na každé úrovni rozhodovacího stromu se dále propouští podokna, která se v danou chvíli tváří jako podstatná, a veškerá negativní okna se odstraňují. Na první úrovni rozhodovacího stromu se vyřadí nejvíce

nevyhovujících podoken, takže na další úroveň se dostanou jen podokna, která vyžadují důkladnější zpracování. Výhodou je v tomto případě razantní redukce negativních podoken, což přispívá k podstatně rychlejšímu zpracování. Na výstupu této kaskády (po projití všech úrovní rozhodovacího stormu) jsou požadovaná podokna.

Kaskádový klasifikátor však nemusí nevyhnutelně pracovat jen s jedním typem příznaků. Kromě Haarových příznaků je možné například využít LBP příznaky [5].

2.4 Technologie OCR

Pod zkratkou OCR [14] je možné si představit metodu, která slouží k optickému rozpoznávání znaků (vychází z anglického Optical Character Recognition). Hlavní úlohou této metody je digitalizovat text tak, aby jakýkoliv text či nápis bylo možné převést do digitální formy a pracovat s ním na výpočetních zařízeních.

Historie optického rozpoznávání se datuje od roku 1914, kdy byl jeho velkým průkopníkem Emanuel Goldberg, který vytvořil první stroj pro čtení znaků a jejich následný převod do telegrafního kódu. V tomto oboru se činil i nadále, kdy následně vytvořil další stroj pro hledání v archivech (s principem založeným na optickém rozpoznávání znaků) [6].

Tato metoda má v praxi široké využití. Nejčastěji se používá pomocí scanerů, kdy se natištěný text převede do digitalizované podoby, přičemž výsledkem není fotka, ale editovatelný soubor obsahující text. Kvalita a správnost rozpoznaného textu závisí především na kvalitě předlohy. S klesající kvalitou vstupního textu je potřeba počítat s větší chybovostí celého rozpoznávání a následnou zdlouhavější korekturou výsledného textu.

Aby bylo vůbec možné pomýšlet na nějaké rozpoznávání textu, je nutné, aby se systém dané znaky nejprve naučil. Těmito znaky jsou myšlena jednotlivá písmena, číslice a veškeré speciální znaky, které se v textu mohou objevit. Principem rozpoznávání potom je takový, že jednotlivé znaky ve vstupním textu porovnává s naučenou databází.

Pro úplné sesumírování je dobré říci, že celý proces rozpoznávání je rozdělen do třech částí:

- Předzpracování – tuhle fázi často nabízí různé OCR softwary za účelem zvýšení šance na úspěšné ropoznání. V této fázi se počítá s tím, že požadovaný text je již naskenován. Kontroluje se zkosení naskenovaného textu, vyhlazují se hrany, z barevného obrazu se převádí na stupně šedi, zkoumá se rozložení textu na stránce, rozlišují se obrázky od textu, izolují se znaky podle spojitých oblastí atd.

- Rozpoznání znaků – při rozpoznávání se většina metod snaží popsat znak z naskenovaného obrázku, zbylé se snaží získat specifické rysy charakterizující daný znak. V případě popisu znaku z naskenovaného obrázku metoda funguje na rozložení bodů v mřížce. Tuto metodu dělíme na:

  - Rozdělení do pásem – jednotlivý znak je mřížkou rozdělen na jednotlivá pásma. V těchto pásmech se následně s pomocí histogramu zkoumají černá místa a tyto histogramy se následně použijí k tomu, aby se porovnaly se znaky z natrénované sady.
  - Průsečíky – u této metody máme předem známý počet konkrétních vektorů. Tyto vektory jsou posléze použity na naskenovaný znak a počet průsečíků se porovná s počtem průsečíků z natrénované databáze, načež je k danému naskenovanému znaku přiřazen příslušný symbol.

- Následné zpracování – v této fázi je vyvinuta snaha o snížení výskytu chyb v souboru s textem. Provádí se to např. tím, že se na výstupu ponechají jen ta slova, která jsou povolena – např. všechna slova patřící do Slovníku spisovného jazyka českého [8] nebo dále s využitím toho, že některá slova jsou častěji viděna spolu, popř. je možné provádět úpravu některou z dalších možných metod.

OCR metodu rozpoznávání textu v dnešní době využívá řada různých softwarů. Rozšířeny jsou na všechny nejpoužívanější operační systémy, některé jsou licencovány jako komerční, jiné jako freeware, GPL či Apache. Jedná se například o Adobe Acrobat, OCRopus, OmniPage, SmartScore, Tesseract a další.

2.5 Tesseract

Pro konkrétní použití v této práci byl z výše uvedených OCR softwarů vybrán Tesseract[1]. Důvodem tohoto výběru je ten, že nabízí nejlepší výsledky kvality rozpoznávání mezi volně dostupnými (bezplatnými) OCR softwary. Výhodou Tesseractu je možnost použití na vícero operačních systémech a také volnost v rozšiřování kódu a jeho užití. Dále v prospěch tohoto softwaru hovoří i to, že na jeho vývoj od roku 2006 dohlíží firma enormních zdrojů a rozměrů – Google, která nejen svým sponzorstvím tento projekt rozhodně podporuje, ale i díky takto velkému jménu v pozadí Tesseract zviditelňuje a protlačuje mezi ostatními OCR aplikaci. V dnešní době navíc Tesseract umožňuje rozeznávat texty v desítkách světových jazycích, mezi které nechybí ani takové (speciální) jazyky, jako jsou arabština, japonština a další. Potěšující okolností je i to, že v této sadě je zahrnutá i čeština. Komu by však tento nemalý balíček předtrénovaných jazyků nestačil, má možnost natrénovat si vlastní sadu, kterou může posléze pro rozpoznání použít.

Určitě je fér, aby kromě výhod zazněla i určitá negativa. Nevýhodou se dá určitě nazvat to, že tento software není schopen zpracovávat jakoukoliv předlohu, nýbrž jen jeho vlastní. Formát této předlohy se nazývá TIFF. V praxi to znamená to, že před tím, než je daná předloha převáděna na text, je nutné tuto předlohu do tohoto formátu převést.

2.5.1 Historie

Historie tohoto softwaru na rozpoznávání znaků se datuje od roku 1985, kdy během následujících devíti letech na něm pracovala firma Hewlett Packard sídlící v anglickém Bristolu. Původní verze, vznikající v těchto letech, neumožňovala migraci softwaru na jiné operační systémy. To se změnilo rokem 1996, kdy Tesseract mohli začít využívat i uživatelé operačního systému Windows.

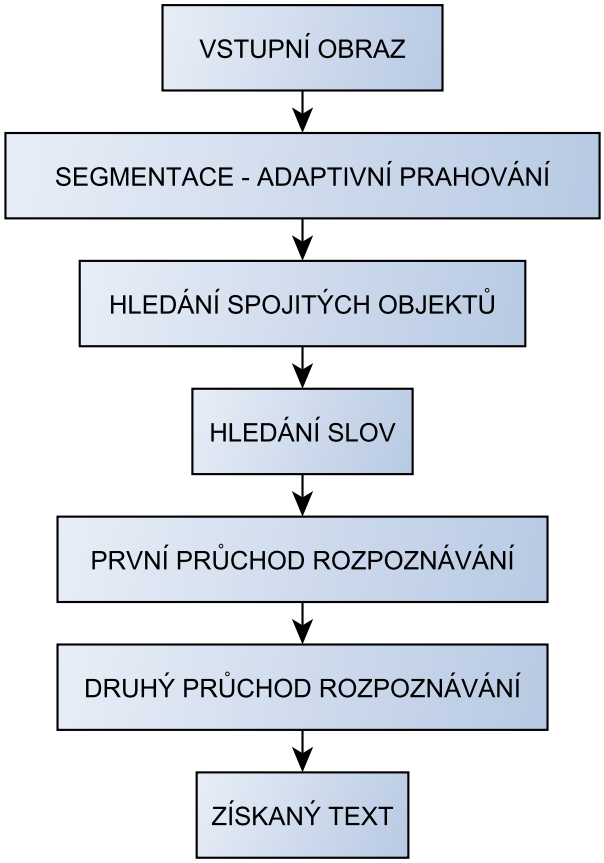
Většina původní verze Tesseractu byla vytvářena v programovacím jazyce C a určitá část v jazyce C++. Z důvodu budoucího rozvoje a další udržitelnosti bylo od původního záměru tvorby v programovacím jazyce C ustoupeno a veškeré zdrojové soubory byly přepsány do jazyka C++. Tato migrace probíhala od roku 1998 až do let 21. století.

Dalším významným milníkem v historií Tesseractu je rok 2005, kdy Hewlett Packard ve spolupráci s nevadskou univerzitou vydali tento software jako open source a o rok později tento projekt zaštítila firma Google.

[1] https://github.com/tesseract-ocr

2.5.2 Princip

Princip fungování algoritmu Tesseractu je na postupném upravování vstupního obrazu a následném víceprůchodovém rozpoznávání textu (Obrázek 2.5).



Obrázek 2.5: Schématický princip fungování Tesseractu.

Na začátku celého procesu je vstupní obraz nejprve zpracováván tak, aby se z něj odstranily veškeré nežádoucí prvky – nazvěme je např. šum. Prakticky se dá mluvit o provádění segmentace nad daným obrazem – konkrétně o využití adaptivního prahování, které je detailněji popsané v kapitole 2.6.2. Dále dostáváme obraz, který monochromatický (jeho jednotlivé pixely jsou buď černé, nebo bílé). V dalším kroku zpracování se na základě jednotlivých pixelů hledají spojité objekty, kterými jsou jednotlivá slova, věty a informace o těchto obrysech jsou následně ukládány pro další zpracování, kterým je seskupování těchto objektů do celků. Z celků se následně pomocí analýzy mezer získávají jednotlivá slova.

Jakmile má software k dispozici tyto potřebné informace, začíná s procesem rozpoznání textu. Celý proces rozpoznávání je rozdělen na dvě části – na dva průchody.

- V prvním průchodu se Tesseract snaží rozpoznat jednotlivá slova. Ty slova, která se zdárně povede v tomto kroku rozpoznat jsou předána adaptivnímu klasifikátoru a použijí se dále jako přídavek k trénovacím datům. Podstatou tohoto procesu je snaha o následné zvětšování šance na správnou klasifikaci konkrétního znaku, slova.

- Při druhém průchodu probíhá rozpoznávání textu jako celku. Díky trénování adaptivního klasifikátoru je v tomto běhu možné rozpoznat slova, která po prvním průchodu zůstala nerozpoznána.

Na konci tohoto celého procesu je výsledkem rozpoznaný text. Kvalita rozpoznání je závislá na kvalitě vstupního obrazu. Přestože Tesseract v rámci samotného rozpoznávání provádí úpravu obrazu na vstupu – výše popsanou část se segmentací, tak touto cestou nikdy není možné upravit obraz se sebehorší kvalitou tak, aby došlo k následnému úspěšnému rozpoznání jednotlivých znaků, slov a vět [17].

Kromě využití předem natrénované sady pro daný konkrétní jazyk je možné v rámci Tesseractu natrénovat sadu vlastní. Jelikož však natrénovaná sada obsahuje češtinu, nebudu se trénováním v rámci této práce dále zabývat.

2.6 Segmentace

Pod pojmem segmentace je možné si představit určitou skupinu metod, fungujících na různých principech zpracování obrazu, které slouží k rozdělení obrazu na části [3]. Těmito částmi jsou myšleny objekty stejných vlastností – objekty zájmu, a nežádoucí pozadí. V rámci této práce bude zapotřebí tyto metody použít k rozeznání znaků na registrační značce vozidla. Mezi nejznámější metody segmentace patří prahování, detekce hran, sledování hranice, apod. Některým z nich se tato práce zabývá detailněji.

2.6.1 Vyrovnání histogramu

Jedná se o metodou, které se používá k úpravě vstupního obrazu. Vstupní obraz může mít v tomto případě špatnou kvalitu vlivem nepříznivých okolních podmínek. Takový obraz má například nízký kontrast. Metoda ekvalizace histogramu transformuje jasovou stupnici, která vyrovnává zastoupení všech jasových hodnot [4]. Toto rozložení umožňuje získat v obraze větší kontrast.

Metoda vyrovnání histogramu je vhodná pro obrazy, které jsou příliš tmavé (viz Obrázek 2.6), nebo naopak velmi světlé, dále je vhodná pro zvýraznění detailů na snímcích, které jsou podexponované nebo přeexponované. Naopak nevýhodou je, že tato metoda nerozlišuje čemu zvýší kontrast, takže se může stát, že po aplikaci této metody může dojít k nežádoucímu zvětšení šumu v pozadí.

Obrázek 2.6: Obraz před a po vyrovnání histogramu.

2.6.2 Prahování

Prahování (z anglického thresholding) funguje na principu úpravy obrazu tak, že samotný obraz převedeme do skupin podle luminace na bílou a černou [2]. Výsledkem je potom zjednodušený obraz, což umožňuje lepší identifikaci – separaci objektů. Nejlepším způsobem použití je, pokud se objekt zájmu odlišuje od přebytečného pozadí.

Dělení do skupin probíhá tak, že se obraz prochází po jednotlivých pixelech, a pokud je hodnota pixelu menší, než je stanovená hranice prahování, potom je hodnotě pixelu přiřazena hodnota 0. V opačném případě se hodnota nastaví na 1. Výstupem se tak stává binární obraz.

Obrázek 2.7: Obrázek s nízkou úrovní prahování.

Obrázek 2.8: Obrázek s vysokou úrovní prahování.

Nejjednodušším (a často nejméně vhodným) způsobem prahování je globální prahování. Tento způsob funguje na principu popsaném v předchozím odstavci, ale jeho největší nevýhodou je, že ona rozhodovací hodnota je stanovena pro celý obraz. Tato nevýhoda se plně ukáže v případě různého – nerovnoměrného osvětlení. Řešením potom bývá využití např. procentuálního prahování, kde není stanovena pevně daná rozhodovací hodnota, ale práh se nastaví na tolik procent, jaké odpovídá procentnímu zastoupení objektu v obraze.

Daleko zajímavějším řešením je použití adaptivního prahování [7]. Toto prahování se vyznačuje tím, že oproti globálnímu nemá stanovenou rozhodovací hodnotu pro celý snímek, díky čemuž zohledňuje nerovnoměrné rozložení světelné intenzity. U adaptivního prahování je práh proměnná hodnota, která závisí na struktuře obrazu a dalších nastavených parametrech.

2.7 Knihovna OpenCV

V rámci této práce je využíváno knihovny OpenCV, která slouží jako knihovna funkcí pro zpracování obrazového vstupu v reálném čase. Jedná se o volně šiřitelnou knihovnu sloužící k vývoji a komerčnímu použití v rámci licence BSD [9].

Kód této knihovny je napsán v programovacím jazyce C a C++, což byla dobrá volba k plánované jednoduché přenositelnosti mezi platformami. V dnešní době obsahuje přes 500 funkcí, které pomáhají v různých odvětvích počítačového vidění. Nejlepší podporu nabízí knihovna OpenCV pro Windows (konkrétně na 32bitové architektuře od společnosti Intel), dále potom pro Linux (na stejné architektuře). Tento vývoj se posléze přesunul i na 64bitové architektury. Se slabší podporou je nutné počítat na operačních systémech od

firmy Apple – OS X, podpora v rámci tohoto operačního systému není na takové úrovni, jako u dříve uváděných operačních systémů, ale postupem času se tato situace velmi rychle mění.

Mezi aplikace, které je možné v rámci knihovny OpenCV realizovat, je možné zařadit například různé programy pro identifikaci objektů, rozpoznání obličejů, sledování pohybů atd. Veškeré tyto programy je možné využít k inteligentní spolupráci člověka a počítače.

2.7.1 Historie

Historie této knihovny se datuje od roku 1999, kdy podnět ke vzniku a i její následnou realizaci vedla mezinárodní společnost Intel, která ve svých prvopočátcích pracovala s experty z Ruska. Tito experti se pohybovali v oblasti zvyšování výkonu knihoven. Jejich společným cílem bylo, aby pokročili ve vývoji počítačového vidění a poskytli světu otevřený, optimalizovaný kód, který bude jednoduše čitelný a snadno přenositelný. Rozšířením takového kódu by se totiž razantně zvýšila poptávka po výkonnějších a rychlejších procesorech.

Postupem času se Intel pomalu přestal na práci a vývoji knihovny podílet a velkou část práce udělali samotní uživatelé. To samé platí i pro dnešní dobu. Dnes však s malým rozdílem, že se na vývoji podílí několik institucí (zajímavá je například podpora od výzkumného institutu Willow Garage, která se zabývá robotiky) a lze tak predikovat velký pokrok v rozvoji této knihovny (měření vzdálenosti pomocí laseru, vnímání hloubky, multikamerová kalibrace a další).

2.8 Defisheye

Pojem defisheye můžeme svým způsobem brát jako odborné (při dnešním trendu anglického jazyka spíše univerzálnější) označení pro odstraňování zakřiveného obrazu. Jedná se o obrácenou operaci k zakřivení obrazu – fisheye [11].

Samotné rybí oko (jak se tomu v oblasti fotografování říká) je speciální typ širokoúhlého objektivu, ve kterém má čočka výrazně širší úhel záběru než v případě klasického objektivu. Standardně je možné se v dnešní době setkat s objektivy, které mají úhel dosahující 180stupňů. V krajních případech některé objektivy tento úhel navýšily i o dalších 40 stupňů a jsou schopny pojmout obraz v úhlu 220 stupňů.

Díky možnosti zachytit více obrazu do jedné fotografie se tyto objektivy začaly těšit velké popularitě, což mělo za následek většího rozšíření mezi širokou veřejnost. Obraz, který byl pořízen takovýmto objektivem, je na první pohled dobře rozpoznatelný, jelikož se u něj vyskytuje typická vlastnost – takové soudkové zkreslení, které unikátním způsobem zkresluje pořízený obraz. Neexistuje žádná čočka, která by při vlastnosti rybího oka netrpěla žádným sférickým zkreslením.

Obecně se rozlišuje více typů rybího oka:

- Kruhové – tento typ čočky zobrazuje scénu ve formě polokoule, přičemž úhel je z hlediska pohledu vertikálního, horizontálního, tak diagonálního naprosto stejný – 180 stupňů (viz Obrázek 2.9).
- Full frame – při rozšíření mezi širokou veřejnost se původní kruh rozšířil tak, aby pokryl celý snímek na 35mm filmu. Oproti kruhovému typu však zůstává stejný pouze diagonální úhel záběru (180 stupňů), horizontální a vertikální úhel bývá zpravidla menší a liší dle zakřivení daného objektivu (viz Obrázek 2.10).

Obrázek 2.9: Obraz s kruhovým typem rybího oka [12].

Obrázek 2.10: Obraz s rybím okem typu Full frame [15].

S obrazem v podobě rybího oka je možné se dnes setkat v každodenním životě. Tyto zakřivené objektivy (potažmo čočky) jsou instalovány na místech, kde je zapotřebí zachytit co největší úhel záběru (bezpečnostní kamery, akční kamery, kukátko na vstupních dveřích). Dále je možné se s ním setkat v kinematografií, moderních kinech, kde je obraz promítán na polokulové obrazovce, stejně tak v planetáriu pro zobrazení oblohy na vnitřní část polokulové střechy a další.

Spojením vícero záběrů (dvou až tří) pořízených kruhovým objektivem je možné vytvořit panoramatický obraz, který umožňuje zachytit okolí v úhlu 360 stupňů. I tento typ záběru si dnes nachází cestu k veřejnosti a čím dál častěji je možné se s ním setkat – například i v oblasti virtuální reality [10].

Kapitola 3

Implementace

Kapitola Implementace se věnuje detailnějšímu popisu metod a nástrojů, které byly použity při návrhu a tvorbě výsledné aplikace. Jsou v ní uvedeny nejen zajímavé programové konstrukce v daném programovacím jazyce, ale také konkrétní příkazy s vysvětlením podstaty a důvodu jednotlivých jejich parametrů. Těmto výsledným řešením však předcházely různé experimenty, které si kladly za cíl toto nejlepší řešení nalézt. Tyto experimenty jsou však popsány až v kapitole následující.

3.1 Návrh programu

Jelikož je hlavním úkolem vytvořit program pro detekci registračních značek, který bude fungovat na konkrétním zařízení, je v první řadě potřeba zvážit několik aspektů. Výsledná aplikace bude sloužit firmě NXP semiconductors s.r.o a poběží na jejich zařízení i.MX 6 Series. Zde je potřeba počítat s omezeným výkonem, paměť tohoto zařízení je zprostředkována formou paměťové karty, tudíž limitujícím faktorem jsou velikosti SD karet apod. Zároveň by však program měl být uzpůsoben tak, že v budoucnu může být portován na novější – např. třeba právě vyvíjené zařízení, které může výkonově dosahovat úplně jiných parametrů. Současně cílem této bakalářské práce bude program, jehož vývoj neskončí odevzdáním práce, ale dále poslouží jako odrazový můstek k rozšířením – či novým aplikací v oblasti detekce obrazu.

Z tohoto důvodu se jeví vhodné vyvíjet program jako více jednotlivých – samostatných částí, které budou následně překládány a spouštěny prostřednictvím argumentů programu.

Výsledný program je tedy možné předem rozdělit do několika samostatných celků:

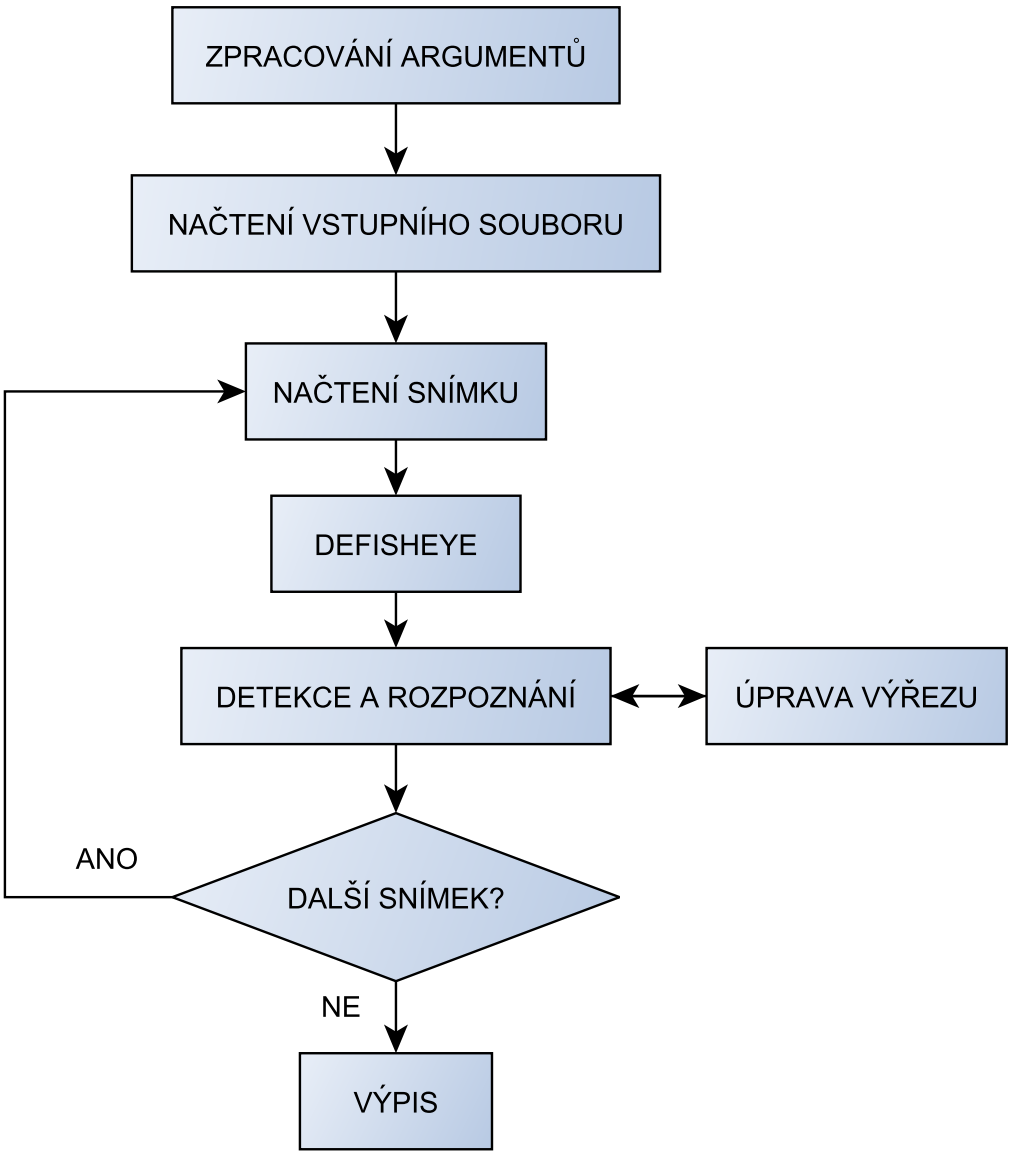
- Odstranění zakřivení obrazu – ať už budou na vstupu fotografie, videa nebo obraz z kamer, veškeré tyto podklady budou pocházet z kamery, která má široký objektiv a tudíž je obraz zkreslen efektem rybího oka. Z hlediska následné detekce je tento jev nežádoucí.

- Detekce registrační značky v obraze – upravený vstupní obraz bude v této fázi zpracován, upraven do nejlepší možné podoby pro co nejvíce úspěšnou detekci. Detekované SPZ poté připraví pro jejich další zpracování.

- Úpravy detekovaných SPZ – jednotlivé registrační značky bude potřeba upravit do podoby, aby v následující části programu (rozpoznávání znaků na SPZ) byly vyhovující. Tím je myšleno odstranění přebytečného pozadí z detekce apod.

- Rozpoznání znaků – detekovaná a upravená SPZ bude podrobena (v tomto případě s největší pravděpodobností) enginu Tesseract, který ze získaného výřezu převede na výstup číslice a písmena uvedená na SPZ. Výstupem může být buď přímo vepsání textu do záběru k dané detekované registrační značce, popř. uložení do souboru.

Následující obrázek 3.1 zachycuje návrh fungování programu jako celku, včetně jeho jednotlivých – výše uvedených částí:



Obrázek 3.1: Diagram zobrazující funkci programu.

3.2 Odstranění zakřivení obrazu kamery

Vstupní obraz, ve kterém je zapotřebí detekovat a následné rozpoznat státní poznávací značku, je kvůli nasazení na přístroji společnosti NXP semiconductors s.r.o "postižen"právě popisovaným kruhovým typem rybího oka. Toto zkreslení je však z hlediska vyhledávání registrační značky (a následné rozpoznávání) naprosto nežádoucí. Pro úspěšnou vyhodnocení je nutné se tohoto efektu zbavit.

Z toho důvodu se každý snímek obrazu nejprve nechá upravit a navrátí se mu jeho pravá podoba. O tuto záležitost se ve zdrojových souborech stará soubor `defisheye.cpp`.

Před nějakým pomýšlením na odstraňovaní rybího oka z obrazu je nutné nejprve zjistit údaje o kameře, kterou byl obraz zachycen. Těmito údaji jsou myšleny hodnoty, které mají vliv na to, jak moc (či málo) je daný obraz zakřiven. Standardní metodou je napsání programu, který provede kalibraci dané kamery. Před tuto kameru se položí šachovnice, kde za její pomocí je možné zjistit, jaké je zakřivení dané čočky, která má vliv na zkreslení v obraze. Tuto metodu – část nebudu vůbec v této práci rozebírat, protože vzhledem k zaměření výsledné aplikace jde o nepodstatnou část.

Jelikož výsledný program poběží na konkrétním zařízení konkrétní firmy, má tato firma veškeré údaje o zakřivení k dispozici. Vnitřní parametry kamery jsou uvedeny v následující matici:

$$A = \begin{bmatrix} 429.099 & 0,0 & 650.150 \\ 0,0 & 429.099 & 405.687 \\ 0,0 & 0,0 & 1,0 \end{bmatrix}$$

a jednotlivé koeficienty k1 – k3 označují radiální zkreslení kamery a koeficienty p1 a p2 označují zakřivení tangenciální.

$$k_1 = -0,26335$$

$$k_2 = 0,06298$$

$$p_1 = 0,00013$$

$$p_2 = -0,00029$$

$$k_3 = -0,00626$$

S vědomím, jaká kamera byla použita (jaké jsou její vlastnosti) můžu přistoupit k úpravě jednotlivých obrazů. Nejjednodušší cestou se nabízí využití funkcí knihovny OpenCV. Výsledné zpracování obrazu bude jednoduché, rychlé a efektivní.

V první fázi zadávám matici zkreslení do kódu manuálně. Jelikož bude program uzpůsoben právě na jedno konkrétní zařízení, neočekává se, že by se v budoucnu jakkoliv musely tyto hodnoty měnit, a proto jsou ručně zadány do funkce `SetMat()`. V rámci této funkce se na začátku běhu programu vytvoří výše zmiňovaná matice o rozměrech 3×3 a jsou do ní zapsány hodnoty zakřivení obrazu kamery. Stejná věc je provedena s koeficienty zkreslení, které jsou taktéž pevně zadány ve funkci `SetCoeff()`, kde jsou umístěny v matici o rozměru 1×5.

Stěžejní částí kódu je funkce `defish()`. Sem se předávají veškeré získané hodnoty a taktéž vstupní obraz. Obraz na vstupu je brán z videa, ale do této funkce už vstupují pouze jednotlivé framy – obrazy. Nejdůležitější částí je funkce z knihovny OpenCV – `undistort()`. Jedná se o funkci, která má sama o sobě za úkol transformovat vstupní zkreslený obraz na výstupní, který bude zbaven efektu rybího oka. Tato funkce v jazyce C++ kombinuje dvě základní funkce knihovny OpenCV – `initUndistortRectifyMap()` a následnou funkci `remap()`. Díky tomu je v rámci volání jedné funkce možné nalézt správné umístění pixelů

ve zkresleném obraze a zároveň je do nově vytvořeného obrazu přímo umístit. Argumenty této funkce jsou:

- vstupní obraz
- výstupní obraz
- vstupní matice kamery
- vstupní vektory koeficinetů zkreslení

Obraze se poté navrací do hlavního programu. Jelikož je nyní bez jakéhokoliv zkreslení, je možné jej předat dalším částem programu, které se věnují samotné detekci a rozpoznávání.

Obrázek 3.2: Obraz z kamery firemního zařízení před úpravou.

Obrázek 3.3: Obraz z kamery firemního zařízení po úpravě.

3.3 Trénování vlastního klasifikátoru

Aby bylo možné vůbec pomýšlet na nějaké detekování objektu v obraze, je ze všeho nejdříve nutné vzít klasifikátor a říci mu, co se má v obraze snažit najít. On sám to neví a je potřeba ho to naučit – natrénovat ho. Obecně platí, stejně jako ve skutečném světě, že čím lepší je trénink, tím jsou poté lepší výsledky. Trénování je zde myšleno poskytnutí klasifikátoru obrázky věcí, které má detekovat, a také obrázky věcí, které detekovat nemá. Vychází tedy z toho, že čím více bude obrázků v obou kategoriích, tím bude klasifikátor robustnější a přesnější. Této obrázkové sadě se říká dataset.

Z hlediska trénování klasifikátoru je vhodné využít funkcí knihovny OpenCV. V případě tréninku kaskádového klasifikátoru se jedná o funkce:

- `opencv_createsamples` – zajišťuje přípravu souborů pro trénink – vytváří z nich datovou množinu pozitivních vzorků ve formátu, který podporuje následující funkce `opencv_traincascade`. Výstupem je poté soubor s příponou *.vec, což je binární soubor, který obsahuje obrázky.
- `opencv_traincascade` – vezme připravené obrázky z předcházející funkce a natrénuje s nimi kaskádový klasifikátor. Výstupem je *.xml soubor, který se poté volá z programu a s jeho pomocí probíhá vyhledávání požadovaného objektu v novém obraze.

Aplikace těchto funkcí při tvorbě kaskádového klasifikátoru vypadala následovně. Prvně byla volána funkce `opencv_createsamples`, která vytvořila pozitivní vzorky z jednoho obrázku (případně kolekce obrázků), a byla spuštěna s následujícími parametry:

```
./opencv_createsamples
   -info lp.info
   -num 500
   -w 220
   -h 44
   -vec lp.vec
```

Parametry této funkce jsou zdrojový soubor obsahující seznam (a umístění) pozitivních vzorků, dále jejich počet, výšku, šířku a nakonec výsledný *.vec soubor. Po vytvoření `lp.vec` souboru je možné překročit k samotné fázi trénování. V rámci této práce bylo trénování spuštěno s následujícími parametry:

```
./opencv_traincascade
   -data data
   -vec lp.vec
   -bg bg.txt
   -numPos 500
   -numNeg 282
   -numStages 25
   -w 50
   -h 10
   -featureType LBP
```

Parametry této funkce jsou: složka, která bude obsahovat výsledky trénování, `lp.vec` soubor z předchozí funkce, soubor s umístěním negativních obrázků, počty pozitivních a negativních obrázků, počet úrovní trénování, minimální výška a šířka registrační značky a příznaky, které byly použity při trénování.

V rámci této práce byly pro trénink klasifikátoru použity příznaky LBP. Důvodem použití těchto příznaků (na rozdíl od příznaků HOG) bylo to, že poměr rychlosti trénování a výsledné přesnosti natrénovaného klasifikátoru je lepší právě ve prospěch příznaků LBP.

3.3.1 Dataset

Jak již bylo psáno výše, pod pojmem dataset si může člověk představit sadu obrázků, které jsou v tomto konkrétním případě používány na trénování klasifikátoru. Veškeré obrázky v tomto datasetu obsahují záběr, kde je jasně viditelná jedna – či více registračních značek vozidel.

Tento dataset byl pro potřeby této bakalářské práce poskytnut jejím vedoucím. Samotný dataset obsahoval celkem 672 obrázků SPZ vozidel, které byly následně manuálně protříděny. Důvodem tohoto vybírání bylo to, že některé obrázky obsahovaly registrační značky vozidel, které nebyly vodorovně se spodní hranou obrázku (případně s mírným náklonem) apod. Výsledný použitý dataset tedy obsahoval 508 snímků, ve kterých se nacházely SPZ vozidel.

Pro zlepšení výsledného kaskádového klasifikátoru byly obrázky v datasetu uloženy s různou expozicí.

Obrázek 3.4: Dataset obsahující různé obrázky s SPZ a s různou expozicí

3.3.2 Pozitivní vzorky

Pro výše uváděné funkce (`opencv_createsamples` a `opencv_traincascade`) je potřeba dodat z datasetu pozitivní vzorky.

Pozitivními vzorky jsou v tomto případě myšleny obrázky, které obsahují registrační značku vozidla. Žádoucí v tomto případě je, aby pozitivní vzorek neobsahoval nic jiného, než pouze danou konkrétní SPZ. Z tohoto důvodu bylo zapotřebí vytvořit skript, který vezme všech 508 snímků z datasetu a vyřízne z nich jednotlivé registrační značky. Nejtěžším problémem by asi bylo v každém obrázku datasetu lokalizovat SPZ před jejím výřezem. Výhodou bylo, že v balíčku s datasetem od vedoucího byl i soubor, který již obsahoval souřadnice konkrétních registračních značek.

Výsledný skript tedy poté vypadal tak, že postupně načítal jednotlivé obrázky z datasetu a odpovídající soubor se souřadnicemi SPZ. Kolem dané registrační značky byl následně vytvořen ROI, který byl po menší velikostní úpravě vyříznut a uložen. Nakonec byl výsledný pozitivní vzorek znovu velikostně upraven na požadovaných 220×44 pixelů. Tento rozměr je poté jednotný pro všechny pozitivní vzorky, což je předpoklad ke správnému natrénování kaskádového klasifikátoru.

Obrázek 3.5: Pozitivní vzorek.

3.3.3 Negativní vzorky

Stejně jako potřebuje funkce na trénink klasifikátoru pozitivní vzorky, tak stejně potřebuje i ty negativní. Negativním vzorkem je tedy vzorek, který neobsahuje jakoukoliv registrační značku vozidla.

Jelikož v rámci této práce nebyla k dispozici žádná rozumná sada negativních obrázků, která by splňovala podmínky pro správný negativní vzorek, bylo zapotřebí si takovou sadu vyrobit. Výroba spočívala ve využití skriptu na získání pozitivních vzorků.

Stejně jako v předchozím případě program načítal obrázky z datasetu a zároveň získával souřadnice SPZ v přiloženém souboru. Znovu se vytvořil ROI, ale na místo jeho vyříznutí se daný ROI zvětšil ve všech směrech o 100 pixelů a celý ROI se vyplnil černou barvou. Tím došlo k tomu, že celý dataset se najednou proměnil v obrázky, ve kterých nebyla žádná registrační značka a takové obrázky mohly být použity k trénování klasifikátoru.

Obrázek 3.6: Negativní vzorek

3.4 Detekce registrační značky v záběru

Společně s natrénovaným kaskádovým klasifikátorem je možné přikročit k detekci registračních značek v obraze. Cílem této části aplikace je vzít vstupní obraz (ať už je to video či pouhá jedna fotka) a nalézt v něm veškeré SPZ. Z implementačního hlediska dává tuto možnost uživateli prostřednictvím argumentů programu.

V hlavní části programu se vyhodnotí, jaká možnost na vstupu byla zadána a podle toho proběhne zpracování obrazu. Pokud je na vstupu zadána fotka, uloží se do matice a je předána funkci na rozpoznávání. Pokud je zadáno video (či živý vstup z kamery), je potřeba brát vstup po jednotlivých snímcích.

Samozřejmou částí je převod obrazu na jeho černobílou variantu a následné upravování tak, aby se ze sebehoršího vstupního obrazu zvýšila šance na správnou detekci státní poznávací značky (např. vyrovnáním histogramu).

Takto upravený obraz je již předán funkci `detectLp()`, jejíž hlavní složkou metoda knihovny OpenCV – `detectMultiScale()`:

```
lp_cascade.detectMultiScale(
    frame_gray,
    lps,
    1.1,
    3,
    0|CASCADE_SCALE_IMAGE,
    Size(30, 30)
);
```

Parametry této metody jsou: vstupní obraz (černobílý), vektor obdélníků, kde každý obdélník obsahuje detekovaný objekt, dále zmenšení obraz v měřítku, minimální počet sousedů, nastavené příznaky a minimální a maximální velikost objektu.

V proměnné `lps` jsou tedy nyní uloženy veškeré detekované registrační značky, které byly v daném obraze nalezeny. Nyní je potřeba zobrazit na výstup a uložit pro další zpracování. Postupně se projdou jednotlivé SPZ uložené v proměnné `lps` a s pomocí následující funkce z knihovny OpenCV se zobrazí na výstup:

```
rectangle(frame, pt1, pt2, Scalar(255),2,8,0);
```

Na vstupu je původní získaný obraz, souřadnice, odkud – kam je potřeba vykreslit obdélník zobrazující detekovanou registrační značku, dále barva, síla a typ čáry.

Současně se pozice vypočítaná pro vykreslení obdélníku kolem ROI využije i pro pozici vyříznutí dané SPZ a ta následně pokračuje k dalšímu zpracování – rozpoznávání znaků.

3.5 Rozpoznání znaků s využitím Tesseract OCR

Než je možné provést rozpoznání s pomocí Tesseractu, je zapotřebí ho inicializovat. V rámci této práce však před samotnou inicializací probíhá nastavení povolených a zakázaných znaků. Toto je provedeno následujícím příkazem:

```
tess.SetVariable(
    "tessedit_char_blacklist",
    "abcdefghijklmnopqrstuvwxyzGOQW"
);
```

První argument zajišťuje, že se jedná o zakázané znaky, které se v daném textu nemají vyhledávat, druhý argument čítá výčet jednotlivých zakázaných znaků. Jsou to znaky, které není možné na žádné standardní české poznávací značce nalézt. Více o specifikaci těchto značek je možné nalézt v příloze A.

Jelikož tato metoda vyloučení nevhodných znaků není ideální, je možné vytvořit sadu znaků povolených. Toto nastavení je provedeno následovně:

```
tess.SetVariable(
    "tessedit_char_whitelist",
    "ABCDEFHIJKLMNPRSTUVXYZ0123456789"
);
```

Oproti předchozímu případu se parametry akorát liší v tom, že v tomto případě jsou vepsány znaky, které má Tesseract vyhledávat.

Po takto definovaném nastavení Tesseractu je přistoupeno k jeho inicializaci. V rámci této inicializace se načte natrénovaná jazyková sada. V tomto případě je využita anglická datová sada, jelikož není možné mít na registračních značkách znaky s čárky a háčky. Toto nastavení je provedeno následujícím příkazem:

```
tess.Init(NULL, "eng", tesseract::OEM_DEFAULT);
```

Úspěšným inicializováním je možné započít kroky vedoucí k rozpoznání SPZ. V další fázi potřeba vzít vyříznutý obrázek (se všemi jeho údaji – výška, šířka apod.) a předat ho následujícím dvěma funkcím, které se starají o rozpoznání:

```
tess.Recognize(0);
string text = tess.GetUTF8Text();
```

Výstupem tohoto rozpoznávání je potom proměnná typu string (v tomto případě nesoucí název `text`), která obsahuje rozpoznané znaky ze vstupní šablony. Často se stává, že kromě správně detekovaných znaků z SPZ obsahuje tento řetězec i další "domyšlené" znaky, jejichž původem je nedokonale oříznutá SPZ. Z tohoto důvodu je zavedena kontrola, která má na starost průchod tohoto stringu a veškeré nežádoucí znaky z něj odstraní. Ponechá je ty, které je možné nalézt na českých registračních značkách.

Kapitola 4

Experimenty a testování

Následující odstavce popisují v první části kapitoly veškeré myšlenkové a programové postupy, které předcházely výsledné realizaci popsané v předchozí kapitole. Některé byly ověřeny testováním, u některých však zjišťování probíhalo experimentálně.

Druhá část kapitoly se věnuje čistě vyhodnocení, zpracování výsledků konečné verze programu a celkové interpretaci těchto dat.

4.1 Defisheye a akční kamera GoPro

Při tvorbě této aplikace bylo od firmy sděleno, že kamera, která bude na daném zařízení snímat obraz, tento obraz zkreslí. Jak bylo již zmíněno výše – z hlediska správného detekování a rozpoznání je nutné nejprve tento obraz vrátit do jeho "původní podoby" a teprve poté je možné s ním nějak dále pracovat. V době této informace však firma nemohla zařízení poskytnout.

V rámci příprav a seznámení se s touto (v té době novou) věcí, začaly přípravy a studium ohledně tohoto tématu. Co to rybí oka jsou, jak vypadají, jaké mohou mít podoby atd. Vzápětí přišlo na řadu studium, jak je možné tento efekt odstranit.

Jelikož se v rámci této práce jedná pouze o přípravu k hlavní části programu, jevilo se jako nejlepší cesta využít (stejně do budoucna plánovanou) knihovnu OpenCV, která má velmi mnoho kvalitních funkcí pro tuto práci. Po všech možných hledání a doporučení byl plánován programovací jazyk Python. Důvodem tohoto rozhodnutí byly především informace, že Python (a společně v kombinaci s knihovnou OpenCV) mají dobré funkce, které mohou vést k efektivnímu a relativně rychlému řešení tohoto problému.

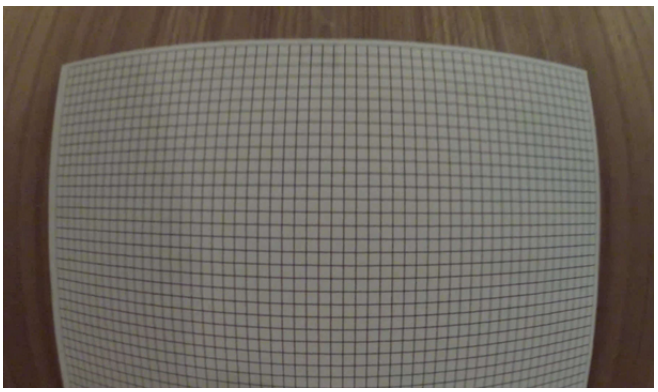
Poslední částí již bylo zvolit vzorovou kameru, pro kterou cvičně program vznikne a v budoucnu se poté uzpůsobí pro potřeby konkrétní firemní kamery. Nejdostupnější kamery s rybím okem jsou v dnešní době nejspíše akční outdoorové kamery, které jsou mezi veřejností velmi rozšířeny. Pro tyto účely byla vybrána akční kamera od společnosti GoPro – konkrétně verze Hero2. Z pohledu rychlosti vývoje těchto kamer patří ke kamerám již služebně starším, ale pro konkrétní účely jsou její parametry naprosto dostačující.

V první fázi programování bylo potřebné zjistit, jaké zakřivení (zkreslení obrazu) má daná čočka této kamery. Postupně začala vznikat funkce `CalibrateCamera()`, která měla za pomoci nasnímané šachovnice získat parametry zakřivení, které by byly posléze použity na zrušení zkreslení ve vstupním obraze. V době vzniku této funkce však společnost GoPro poskytla tyto konkrétní údaje – a jelikož stejné údaje měla ke své vlastní kameře i firma,

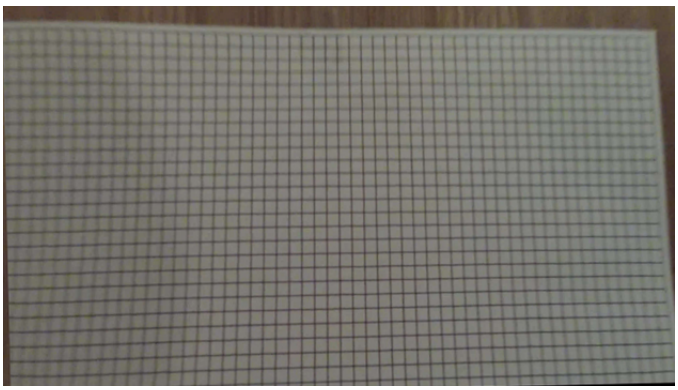
pro jejíž přístroj tato práce vzniká, bylo již redundantní pokračovat v dané chvíli na funkci `CalibrateCamera()`, když v konečném výsledku nebyla (a ani nebude) zapotřebí.

Se získanými parametry bylo nutné vytvořit matici, kdy byly použity funkce knihovny OpenCV – `CreateMat()`, pomocí které se vytvořila matice o velikosti 3×3. Následně byla volána funkce `SetReal2D`, s jejíž pomocí se tato matice naplnila hodnotama zkreslení. Stejným principem byla vytvořena i matice obsahující koeficienty zkreslení o velikosti 1×5. V Pythonu však při rušení rybího oka byly stěžejní dvě funkce knihovny OpenCV. Konkrétně funkce `InitUndistortMap()`, která zpracuje zkreslení s využitím matice a koeficientů pro remapovací funkci `remap()`, která každému pixelu vypočítá jeho správnou pozici ve výsledném obraze. Tato remapovací funkce je samozřejmě volána pro každý snímek ze vstupu kamery a postupně je výsledný snímek rovnou zobrazován na výstup.

Tento jednoduchý program pro úpravu vstupního obrazu fungoval v dané chvíli naprosto bezchybně. Na výstupním obraze nebylo znát, že by jakkoliv vycházel z obrazu, který byl nějakým způsobem zkreslený. Patrná byla jen logická ztráta některých krajních pixelů, protože z velmi širokého obrazu najednou vznikl obraz odpovídající standardním objektivům bez efektu rybího oka. Výsledek je patrný i na následujících dvou obrázcích. Obrázek 4.1 zobrazuje zkreslený obraz na vstupu a obrázek 4.2 následný obraz po průchodu tímto jednoduchým programem.



Obrázek 4.1: Obraz z kamery GoPro před úpravou [16].



Obrázek 4.2: Obraz z kamery GoPro po úpravě [16].

S postupem času se však ukázalo, že pro celý projekt je lepší variantou zvolit jiný programovací jazyk než je Python (konkrétně C++), a proto bylo v budoucnu vhodné přeprogramovat tuto část výsledné aplikace do tohoto jazyka. Ve výsledku se i ukázalo, že kombinace knihovny OpenCV a programovacího jazyka C++ byla v tomto případě ještě více úspornější, než předchozí kombinace OpenCV a Pythonu.

4.2 Závislost trénování klasifikátoru na kvalitě výstupu

V začátcích práce, kdy docházelo k seznamování se s tématem detekce objektů v obraze jako takovým, nebyl ještě připraven potřebný dataset obrázků. V tu chvíli docházelo k vyhledávání různých datasetů a trénování si zkušebních klasifikátorů, které postupně detekovaly různé věci. Po zkompletování datasetu s registračními značkami, bylo zapotřebí při tréninku klasifikátoru použít i negativní sadu obrázků – obrázků, kde se nenachází žádná SPZ vozidla. V první fázi byla použita volně dostupná sada negativních obrázků ze sady na trénink klasifikátoru pro detekci automobilů ze státní univerzity University of Illinois Urbana-Champaign[1] v americkém Illinois.

Obrázek 4.3: Negativní vzorek sady z UIUC Illinois.

Tato volně dostupná sada (nejen negativních vzorků) čítala 500 obrázků v malém rozlišení 210×115 pixelů. Prvotní klasifikátor na detekci registračních značek tak vznikl s touto negativní sadou a byl vytvořen s těmito parametry:

```
./opencv_traincascade
   -data data
   -vec lp.vec
   -bg bg.txt
   -numPos 500
   -numNeg 500
   -numStages 10
   -w 50
   -h 10
   -featureType LBP
```

Nejvíce zajímavou hodnotu měl v tomto případě parametr `-numStages`. Počet úrovní byl při tomto zkušebním pokusu schválně nastaven na nižší hodnotu, aby byl výsledný klasifikátor rychle hotový a bylo možné jej hned vyzkoušet. Reálná zkouška však byla horší než původní (již tak slabé) očekávání. Klasifikátor byl použit v programu, který z kamery

[1] http://www.cogcomp.cs.illinois.edu/Data/car

načítal obraz, a měl v něm detekovat registrační značku. Situace však byla taková, že registrační značku v obraze jen těžko nalezl, ale za zároveň detekoval SPZ ve spoustě míst, kde žádná registrační značka vozidla nebyla.

Dalším testováním bylo zvýšení hodnoty parametru `-numStages` z původních 10 na 30. Čas strávený tréninkem takového klasifikátoru vzrostl na více jak trojnásobek, avšak očekávání v lepší výsledek se nenaplnilo. Falešných detekcí SPZ neubylo – jediným rozdílem bylo, že nyní se úspěšnost detekce z původních 3 % zvedla na 12 %.

Po konzultaci s vedoucím práce byl shledán problém v oné negativní sadě obrázků. Obrázky z této sady byly více jak 4× menší a ani jejich kvalita nebyla zrovna dle očekávání. Pro ověření této teorie byla tato sada negativních vzorků nahrazena vzorky zobrazující pouze bílé stěny pokoje. Těchto obrázků však nebylo příliš – pouhých 35. Opakovaná tvorba klasifikátoru byla spuštěna s těmito parametry:

```
./opencv_traincascade
   -data data
   -vec lp.vec
   -bg bg.txt
   -numPos 500
   -numNeg 35
   -numStages 30
   -w 1280
   -h 720
   -featureType LBP
```

V tuto chvíli byly obrázky negativní sady (oproti té předchozí) mnohonásobně větší a parametr `-numStages` byl ponechán na vyšší hodnotě. Očekáváním bylo, že při spuštění programu před bílou stěnou pokoje, na které se bude nacházet registrační značka vozidla, nebude mít tento program problém tuto SPZ detekovat. Původní očekávání se však nenaplnila v takové míře, protože registrační značku to sice dovedlo správně detekovat, ale kromě ní program zobrazoval i spoustu falešně detekovaných SPZ v obraze.

Začalo se tedy uvažovat o setu negativních obrázků, který bude mít velké rozlišení, bude obsahovat objekty, které by se mohly v případě výsledné aplikace v obraze objevit jako nežádoucí, a aby jejich počet byl ideálně co největší. Na snaze bylo řešení, že se negativní obrázky vyrobí z původních obrázků tak, že se celý obrázek ponechá až na dané registrační značky, které se začerní. Program byl následně spuštěn s parametry, které kombinovaly výhody předchozích tréninků. Nová sada obsahovala (po vhodném protřídění) 282 negativních obrázků. Výsledkem bylo, že program neměl problém detekovat SPZ v obraze a počet falešných detekcí registračních značek se rapidně snížil.

Problém s nevhodnou sadou negativních vzorků byl vyřešen, avšak prostor ke zlepšení klasifikátoru zde pořád byl. Další úvahou bylo s těmito sety vzorků zvýšit rapidně parametr `-numStages` na hodnotu 40 – 50. První (a ve výsledku i poslední) testování této myšlenky byl s hodnotou parametru `-numStages 40`, kdy trénink klasifikátoru na průměrné pracovní stanici zabral téměř 8 hodin trénování. Očekávání byla taková, že vymizí veškeré falešné detekce a program bude s velkou přesností detekovat jednotlivé registrační značky. Po spuštění programu s tímto natrénovaným klasifikátorem se však tato hypotéza nepotvrdila. Rozdíl oproti předchozímu klasifikátoru nebyl vůbec patrný – neustále docházelo ke stejnému počtu falešných detekcí SPZ v obraze a přesnost správné detekce značek se také nezvýšila. S ohledem na toto zjištění bylo experimentálně ověřeno, že nejvhodnější hodnotou parame-

tru `-numStages` je hodnota od 20 do 30 a žádné další navyšování této hodnoty nepovede k lepšímu výsledku.

Závěrem těchto experimentů s tréninkem kaskádového klasifikátoru je to, že pro případné další zvyšování přesnosti – spolehlivosti detekce je nutné dívat se směrem k datasetu, který bude mít nejspíše největší vliv na další případné zvyšování přesnosti detekce. Aktuální počet pozitivních vzorků z tohoto datasetu (508 vzorků) se jeví jako malý a jeho do budoucna případné zvětšení na několikanásobek vypadá jako správná cesta.

4.3 Tesseract a rozpozávání znaků

První zkouška s tesseractem proběhla tak, že se výřez detekované registrační značky uložil jako obrázek (v požadovaném formátu pro tesseract) znacka.tiff. Z prostředí terminálu byl nad tímto obrázkem následně zavolán samotný tesseract – a to příkazem:

```
$ tesseract znacka.tiff output.txt
```

Parametry tohoto příkazu byl jak samotný obrázek, tak soubor, kam se mají detekované znaky z SPZ uložit. Následně pro větší zautomatizování byl vytvořen jednoduchý skript, který byl z hlavního programu pravidelně volán. Jeho hlavní podstatou bylo, že jakmile se uloží nový obrázek, tak má přes terminál zavolat tesseract výše popisovaným příkazem, provést rozpoznání a následně vrátit chod zpět do hlavního programu. Ten si poté otevřel soubor a přečetl z něj, co tesseract rozpoznal. Ačkoliv to řešení bylo funkční, tak z praktického hlediska se jevilo jako nepoužitelné. Náročnost takového programu, kdy by při každé detekované registrační značce v obraze musel řešit tvorbu souboru, zápis a čtení atd., by byla opravdu veliká a to rozhodně nesplňovalo aspekty, které byly stanoveny při návrhu projektu. Řešením se tak muselo stát zahrnutí tesseractu do zdrojového kódu tak, aby nebylo nutné absolvovat celý tento náročný a zdlouhavý proces.

4.4 Úprava výřezů SPZ před rozpoznáváním znaků

Často viděným problém u detekcí je jejich určitá nepřesnost. Z pohledu registračních značek by bylo žádoucí, aby detekovaná oblast (a její výřez) obsahovala pouze jednotlivé znaky, které se následně předají tesseractu na rozpoznání, ale skutečnost bývá často jiná. SPZ bývá detekována nejen s okolními rámečky registrační značky (které tesseract může vyhodnotit jako nějaké konkrétní znaky), ale také dalším přebytečným okolím – v tomto případě i nárazníkem, kapotou či maskou automobilu. Do výsledku se s takovým obrazem dostávají nepřesnosti a výsledná rozpoznaná registrační značka poté obsahuje přebytečné znaky navíc.

Pro eliminaci tohoto jevu je nutné, aby byl výřez detekované registrační značky upraven. Ideálním případem by po této úpravě byla skutečně pouze registrační značka (její bílá část bez modrého proužku), která by obsahovala pouze znaky určené k rozpoznání. Tím by se docílilo nejlepších možných výsledků.

Z teoretického hlediska je tedy zapotřebí najít ve výřezu registrační značku dle nějakých konkrétních specifikací a podle nich SPZ oříznout. Hlavní vlastností českých standardních registračních značek je, že mají bílé pozadí s černým textem. Této znalosti se využilo a vznikl nápad na lokalizaci SPZ tak, že program postupně projde jednotlivé řádky výřezu a provede v nich součet bílých pixelů. Jelikož znaky na registrační značce jsou ve všech směrech umístěny od kraje značky v určité vzdálenosti, nabízí se tedy varianta, že hledáním dvou

maxim (a následném oříznutí podle nich), by bylo možné zbavit se v horní a spodní části výřezu přebytečných pixelů. Stejným postupem (avšak aplikováním součtu pixelů na sloupce) by bylo možné zbavit se i levé a pravé části a výsledný ořez by byl ideální pro předání tesseractu.

Aby bylo možné počítat bílé pixely v jednotlivých řádcích a sloupcích, v prvním kroku přišla na řadu technika prahování. Cílem této techniky je z černobílého osmibitového obrázku získat obraz, jehož pixely budou mít hodnotu buď 0, nebo 255. Toho bylo docíleno voláním následující OpenCV funkce:

```
threshold(
    image,
    image_gray,
    0,
    255,
    CV_THRESH_BINARY | CV_THRESH_OTSU
);
```

Argumenty jsou vstupní obraz, výstupní obraz, hodnota prahu, maximální hodnota prahu pro prahovací typy a nakonec typ se speciální hodnotou `THRESH_OTSU`, která v praxi znamená, že optimální hodnota prahování je vypočítána dle algoritmu Otsu.

Výsledkem prahování je tedy obraz, který má pixely pouze o hodnotách 0, nebo 255. Nyní je tedy možné v něm začít hledat po řádcích a sloupcích největší součty. O tuto část se starají funkce `pruchodCol()` a `pruchodRow()`. Princip obou funkcí je stejný, liší se pouze v procházení řádků a sloupců pixelů obrazu. V první řadě volána funkce `pruchodRow()`, kde ve for cyklu (který běží pouze nad spodní půlkou řádků obrazu) se postupně dělají součty bílých pixelů na konkrétním řádku a průběžně se i sleduje maximální hodnota. Na konci procházení se vytvoří nový ROI, který bude ve spodní části oříznut na pozici, kde se nacházel největší součet. Následně je volána funkce knihovny OpenCV – `flip()`, která obraz otočí vzhůru nohama a znovu se zavolá funkce `pruchodRow()`. Po dokončení těchto dvou volání `pruchodRow()` je obraz z vrchní a spodní části oříznut. Na totožném principu funguje i funkce `pruchodCol()`, na jejímž skončení je obraz, který je ze všech stran oříznut dle nalezených maximálních hodnot součtu bílých pixelů. Ideálně by tak výřez SPZ měl obsahovat pouze text, který se má před dále na rozpoznání.

V praxi se tak bohužel stalo na pouhých 8 % případů (konkrétně na osmi obrázcích ze sta). Zbylé obrázky buď nebyly dostatečně oříznuty, nebo byly ořezávány až příliš. Nejčastějším problémem bylo, že v obrázku se po prahování nacházel na jednom řádku větší počet bílých pixelů mimo registrační značku, tudíž došlo k malému – případně nulovému ořezu. Dalším problémem bylo, že tento princip ořezávání se nesetkal s úspěchem na výřezech, ve kterých byla registrační značka vozidla pootočena oproti její standardní vodorovné pozici. Problémy působila už taková SPZ, jejíž natočení bylo i v jednotkách stupňů. Při malém natočení docházelo k situaci, že počáteční či koncové znaky nebyly kompletní, což pro následné OCR byl problém a docházelo tak ke ztrátám těchto neúplných znaků (potažmo jejich nesprávnému vyhodnocení). Při velkém natočení (v desítkách stupňů) poté docházelo k naprosto nevyhovujícím a nežádoucím ořezání.

Možným vylepšením se zdála varianta, kdy princip sčítání bílých pixelů zůstane stejný, ale prahování bude nahrazenou jinou metodou předzpracování obrazu. Důvodem snahy o vylepšení byla evidentně nízká úspěšnost předchozí metody. Novou variantou se nabízela detekce hran v obraze, kdy jednotlivé hrany budou označeny bíle a zbytek obrazu bude

černý. Sčítáním pixelů by se měla jako největší hodnota jevit hrana státní poznávací značky vozidla, tím by mělo dojít k lepšímu nalezení SPZ v obraze a k lepšímu následnému ořezu.

Jako vhodným řešením se nabízelo využít Cannyho detektor hran. Podobně jako předcházející funkce `threshold()` je i tato funkce implementována v knihovně OpenCV. Cannyho hranový detektor v rámci tohoto experimentu použit následovně:

```
Canny(
    image_gray,
    image_gray,
    200,
    200*3,
    3
);
```

Velkou výhodou bylo, že různé rušivé elementy se po aplikaci tohoto hranového detektoru ztratily a podstatně více vyzněla samotná registrační značka. Na první pohled na tom byla tato varianta předzpracování obrazu podstatě lépe než v předchozím případě. Pokud byla značka ve výřezu přesně vodorovně se spodní hranou výřezu, potom úspěšnost (při stejné velikosti testovací sady) dosahovala 97 %. Avšak tato varianta předzpracování obrazu trpěla stejnou vadou jako ta předchozí. Stačilo, aby obraz byl o jednotky stupňů pootočený a výsledná hrana poznávací značky se rozdělila do více řádků, takže ve výsledku při hledání největšího počtu bílých pixelů byla pro ořez detekována hrana některého nežádoucího objektu na místo hrany státní poznávací značky.

4.5 Portace zdrojových souborů na zařízení i.MX 6 Series

Prvotní nahrání zdrojových souborů na zařízení i.MX 6 Series proběhlo v sídle společnosti NXP semicondutors s.r.o pod odborným dohledem Ing. Iriny Trukhiny. Z její strany proběhla veškerá instruktáž, aby později mohl být stejný typ zařízení zapůjčen pro testovací účely.

Ke startu i.MX 6 stačí napájení o velikosti 12 V a k další práci s tímto zařízením je zapotřebí monitor a kabel HDMI na propojení. Propojení desky a notebooku, na kterém byl program vyvíjen, proběhl pomocí ethernetového kabelu. Jak na zařízení, tak na notebooku se nastavila IP adresa pomocí příkazu ifconfig, obě zařízení se propojily ethernetovým kabelem a i.MX 6 byla následně připravena na přenos souborů.

Samotný přenos souborů byl realizován pomocí bezpečného protokolu SCP. Přenos byl proveden v následující podobě:

```
$ sudo scp -r lp root@192.168.0.1:/home/root
```

Jelikož `lp` je označení pro složku, ve které se nachází veškeré zdrojové soubory, bylo zapotřebí provést přenos rekurzivně. Celá složka se tak zkopírovala na zařízení s IP adresou 192.168.0.1 do složky `/home/root` pod uživatelským jménem root.

V době testování nemělo v sobě zařízení nainstalováno tesseract (Linux je na něm zkompilován pomocí Yocto project[2], proto nebylo možné v danou chvíli balíček tesseractu jednodušše stáhnout), byly na zařízení postupně nahrány nejjednodušší části programu.

První částí programu bylo, jestli je možné na daném zařízení správně zkompilovat část programu, která má za úkol detekovat SPZ v obraze. Překlad a nalinkování veškerých funkcí

[2]https://www.yoctoproject.org/

knihovny OpenCV proběhl bez jakýchkoliv problémů. Program byl prvně spuštěn s takovými parametry, aby otestoval detekci z pořízené fotografie. Spuštění mělo sice několikasekundovou prodlevu, ale jinak bylo testování úspěšné. Ve druhé fázi byl program spuštěn s takovými parametry, aby postupně načítal jednotlivé snímky z videa a na nich provedl detekci registračních značek. V tu chvíli přestala výpočetní síla zařízení stíhat a přehrávané video bylo oproti původní verzi více než 3× zpomalené.

Dále proběhlo testování dalších částí programu – podobně byl testována funkce defisheye, která ze zkresleného videa neměla problém udělat video, které bylo tohoto efektu zbaveno a stejně tak byla zkušebně otestována i funkce ořezávání, při které mělo i.MX 6 výkonu dostatek.

Výsledky portace zdrojových souborů na zařízení proběhly oproti očekávání nadmíru dobře. Přestože nebyly otestovány veškeré zdrojové kódy, účel seznámení se ovládáním zařízení byl splněn.

4.6 Vyhodnocení

4.6.1 Specifikace jednotlivých zařízení

Pro účely vyhodnocení běhu aplikace bylo provedeno testování na těchto dvou konkrétních zařízení: počítač a i.MX 6 Series. Pro jasnější představu výsledného měření, obsahuje následující tabulka 4.1 parametry těchto zařízení:

Zařízení	Specifikace
i.MX 6 Series	i.MX 6Quad procesor running up to 1 GHz 2 GB DRAM running up to 533 MHz (DDR3-1066), 64-bit bus 32 MB 16-bit parallel NOR flash NAND flash socket LVDS LCD output HDMI display jack MIPI CSI connector SD card slot High-Speed USB (OTG) interface, pin configurable as OTG 1.5 Gb/s SATA interface JTAG and UART interfaces
Počítač	Intel Core i5 4200M Haswell 3,1 GHz 8 GM RAM DDR3 NVIDIA GeForce GT 645M 2 GB SSHD 5 400 ot/min RJ-45 (LAN), WiFi HDMI, VGA

Tabulka 4.1: Porovnání délky zpracování fotografie a jednoho snímku z videa.

4.6.2 Měření rychlosti běhu na počítači a i.MX 6 Series

Po vytvoření vzorové aplikace bylo nutné ji otestovat. Hlavní důraz se zde kladl na testování výkonu na počítači, na kterém vznikala aplikace, a na zařízení i.MX 6 Series. Tyto výkony jsou v rámci této podkapitoly srovnávány.

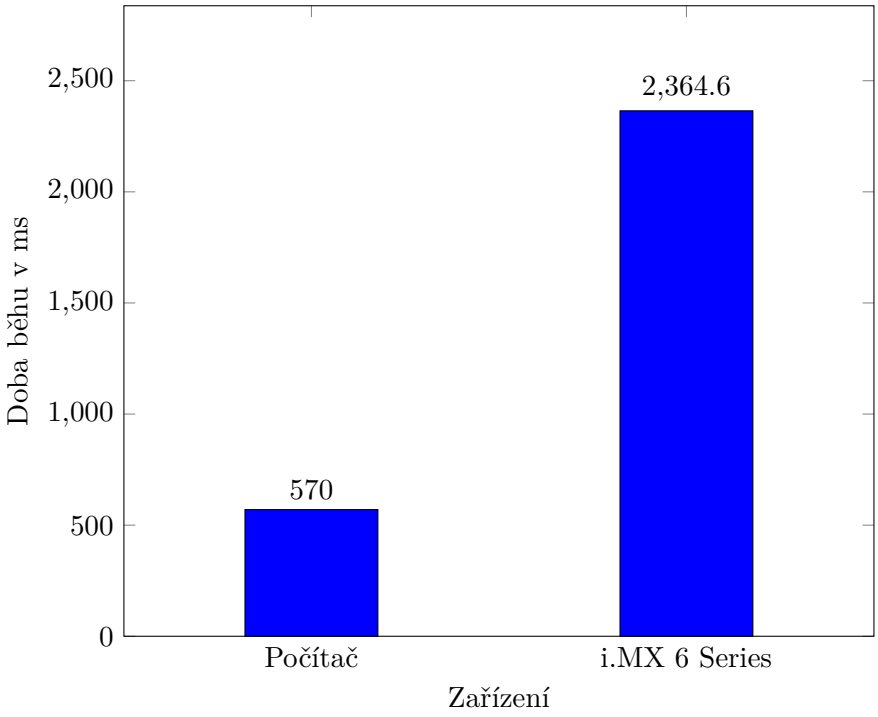
Podobně jako byl program vyvíjen (postupně po jednotlivých modulech), tak byl i takto testován.

V rámci první testování byl program spuštěn na obou zařízení s těmito parametry:

```
$ ./lp -f foto.png -s
```

Kombinací těchto parametrů měl program za úkol načíst fotografii, odstranit zakřivení obrazu, provést detekci, ořezání, následné rozpoznání a výsledný text zobrazit v obrázku nad registrační značkou vozidla.

Obrázek 4.4 zobrazuje srovnání délky běhu v milisekundách na obou zařízeních:



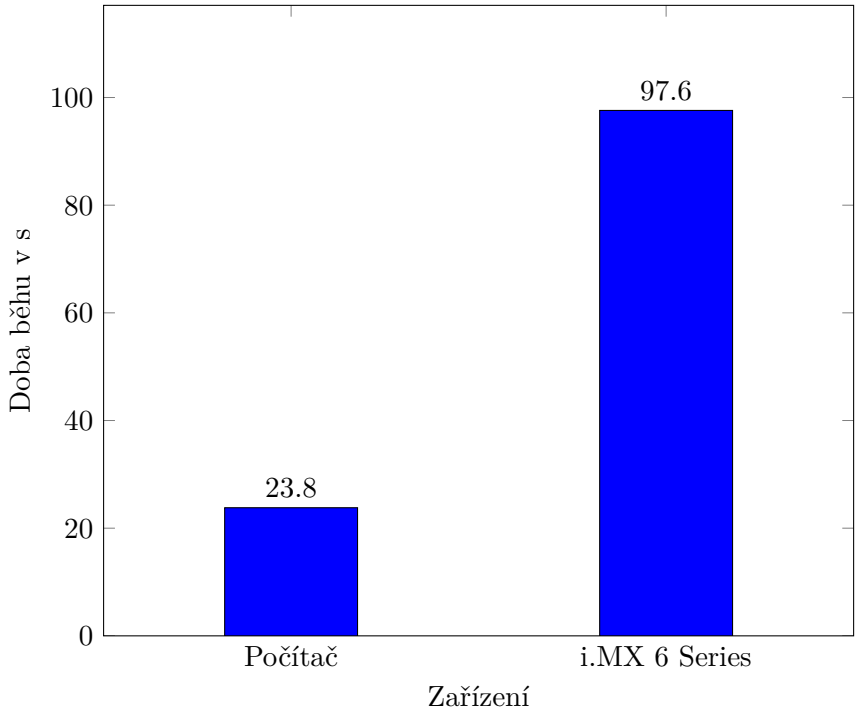
Obrázek 4.4: Srovnání délky běhu zpracování fotografie.

Dle původních předpokladů vyšel v rámci rychlosti lépe počítač, kde rozpoznanou registrační značku z fotografie zvládl program dostat na výstup za cca půl sekundy. U zařízení i.MX 6 Series bylo potřeba si na stejný výsledek počkat necelé 2,5 sekundy.

Dalším testem bylo spuštěním programu tak, aby místo fotografie vzal jako vstupní soubor video. Program byl spuštěn s následujícím parametry:

```
$ ./lp -v video.mp4 -s
```

Spuštěním programu s těmito parametry měl program stejný úkol, jako v předchozím případě s fotografií, nyní však byl na vstupu soubor s videem. Délka tohoto videa byla 6 sekund a video se skládalo z celkem 42 snímků. Následující obrázek 4.5 zobrazuje získané časy (v sekundách) měření na obou zařízení:



Obrázek 4.5: Srovnání délky běhu zpracování videa.

Zařízení trvalo zpracovat video zhruba 4× déle než konkrétnímu počítači. Jelikož je video zpracováváno po jednotlivých snímcích, nabízí se zde srovnání délky zpracování jednoho snímku z videa s délkou zpracování snímku v předchozím testu. Podělením celkových časů z tohoto testu počtem snímků, vychází skutečně velice podobné hodnoty jako v předchozím testu – viz následující tabulka 4.2:

Délka zpracování vstupu	**Počítač**	**i.MX 6 Series**
Fotografie	570 ms	2364,6 ms
Spočítaný jeden snímek z videa	566,6 ms	2323,8 ms

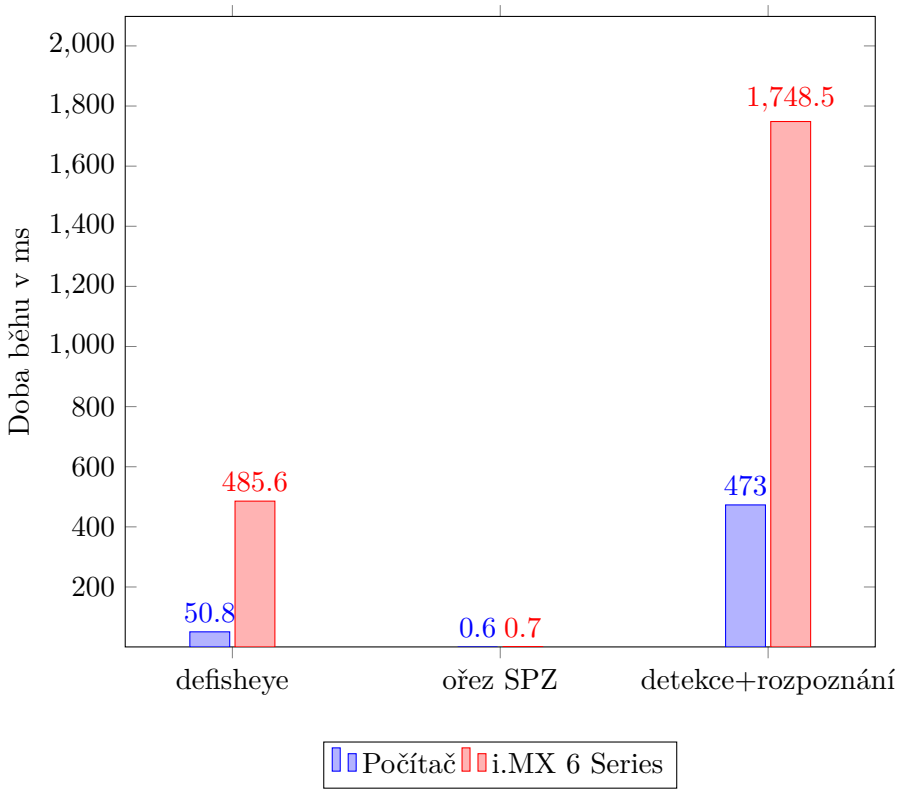
Tabulka 4.2: Porovnání délky zpracování fotografie a jednoho snímku z videa.

V další části byly testovány jednotlivé moduly dané aplikace zvlášť. Měření bylo provedeno tak, že na vstupu byl obrázek, ve kterém se nacházela jedna registrační značka vozidla. Program byl spuštěn s těmito parametry:

```
$ ./lp -f foto.png -s
```

Následující obrázek 4.6 zobrazuje výsledky doby běhu jednotlivých modulů. Nejlépe z něj vychází modul, který se po detekci snaží o co nejlepší ořez registrační značky, aby bylo docíleno úspěšného rozpoznání. Naopak nejhůře z něj vychází modul starající se o detekci a rozpoznání. V případě zařízení i.MX 6 Series dosahuje téměř hodnoty 2 sekund. Jelikož je tato hodnota značně vysoká, bude tomu věnován další test, kde tento modul bude proměřen po jednotlivých částech.

Dobrým znamením je, že při pohledu na modul defisheye a detekce a rozpoznání, vychází podíly jednotlivých částí na obou zařízení velmi podobně.



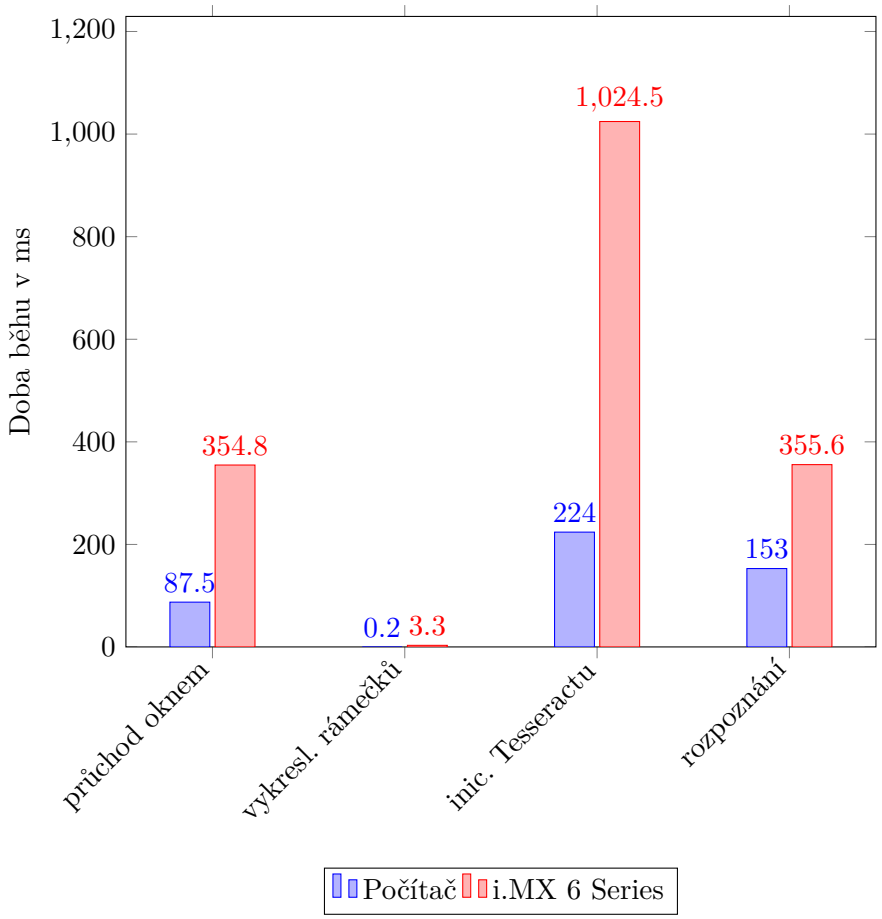
Obrázek 4.6: Doba běhu jednotlivých modulů programu.

Protože je hodnota necelých dvou sekund na jedno zpracování značně vysoká, věnuje se následující test jejím jednotlivým podstatným částem. Logicky lze tento modul rozdělit na několik částí:

- Průchod oknem – v principu se jedná o vyhledávání registrační značky v obraze. V rámci aplikace je na to využita funkce knihovny OpenCV – `detectMultiscale()`.
- Vykreslení rámečků – kolem nalezených registračních značek se vykreslují modré rámečky, které jsou vepsány do výstupního obrazu. Současně se u nich počítají jednotlivé souřadnice, které dále slouží k ořezávání těchto detekovaných oblastí pro další zpracování.
- Inicializace Tesseractu – příprava Tesseractu na rozpoznávání.
- Rozpoznání – část, která se vezme výřez detekované registrační značky a provede nad ním proces rozpoznání textu, který převede do řetězce na výstup.

Při pohledu na jednotlivé sloupečky zařízení i.MX 6 Series v obrázku 4.7, je patrné, že vyhledávání registrační značky a rozpoznání textu nepředstavuje pro toto zařízení nějaký velký problém. Velmi výrazně však vyčnívá jedna konkrétní hodnota – inicializace Tesseractu. Na první pohled je to část, od které se neočekávalo zatížení v takové míře, ale opak je pravdou. Tato inicializace je v programu tvořena voláním jediné metody:

```
tess.Init(NULL, "eng", tesseract::OEM_DEFAULT);
```



Obrázek 4.7: Doba běhu jednotlivých částí detekce a rozpoznání.

Metoda se sestává ze tří parametrů:

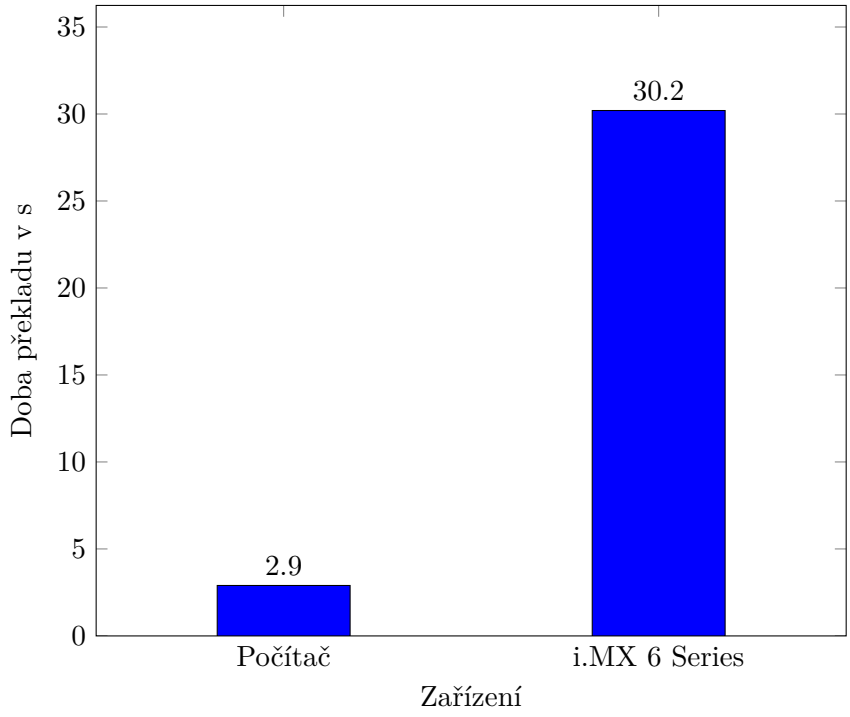
- cesta k datům – parametr obsahující nadřazený adresář tessdata končící lomítkem
- jazyk rozpoznávání
- mód – nastavení režimu rozpoznávání znaků Tesseractu

První parametr je v tomto případě nastaven na hodnotu NULL, jelikož není potřeba nastavovat žádnou cestu k nadřazenému adresáři, druhý parametr obsahuje třípísmenné označení jazyka, ve kterém bude rozpoznání probíhat a poslední parametr nastavuje režim Tesseractu na základní.

Z tohoto výčtu parametru se jeví jako nejnáročnější načítání konkrétní jazykové sady. V praxi to vypadá tak, že soubor s danou natrénovanou jazykovou sadou je stažen přímo z uložiště Tesseractu[3]. Tato natrénovaná datová sada (v případě tohoto projektu nesoucí název `eng.traineddata`) se nachází v `/usr/share/tesseract-ocr/tessdata`. Kdykoliv je poté zapotřebí rozeznat nějaký text v programu, je tato jazyková sada načtena a s její pomocí je text poté rozpoznán. Toto načítání je tedy v programu prováděno pokaždé, když je volána funkce na detekování registrační značky v obraze.

[3]https://github.com/tesseract-ocr/tessdata

Posledním měřením v této práci bylo měření rychlosti překladu programu na obou testovaných zařízeních. Překlad probíhal příkazem `make` a čas jednotlivých překladu byl měřen. Výsledky zobrazuje následující obrázek 4.8:



Obrázek 4.8: Délka překladu programu na jednotlivých zařízeních.

Jednoznačně v tomto případě rychlejší počítač, na kterém proběhl překlad až 10× rychleji než v případě překladu na zařízení i.MX 6 Series.

4.6.3 Kvalita klasifikátoru

Pro zajímavost byla kromě veškerého měření vyhodnocena i kvalita natrénovaného klasifikátoru. Jelikož má na vliv tohoto vyhodnocení i více různých parametrů metod, pro vyhodnocení byla vybrána jedna z těch, které mají na výsledek velký vliv. Konkrétně se jedná o parametr `minNeighbors`, který udává minimální počet sousedních oblastí nutný označení oblasti jako kandidátní. Tento parametr byl v průběhu vyhodnocování měněn v rozsahu od 1 do 5 a pro každou tuto hodnotu byl provedno testování. Celkový počet obrázků testovací sady byl 189 fotografií. Jednotlivé hodnoty jsou uvedeny v tabulce 4.3.

minNeighbors	**TP**	**FP**	**úspěšnost**	**přesnost**
1	166	735	87,8 %	18,4 %
2	147	221	77,7 %	40 %
3	135	184	71,4 %	42,3 %
4	128	82	67,7 %	61 %
5	126	73	66,6 %	63,3 %

Tabulka 4.3: Kvalita natrénovaného klasifikátoru.

Zkratky TP v tabulce označují hodnotu `True Positve` (správně detekované registrační značky) a FP, které označují hodnotu `False Positive` (uvádějící počet falešných detekcí). Jak vyplývá z tabulky 4.3, tak nejlepším možným poměrem úspěšnosti a přesnosti byla zvolená hodnota 4. Za cenu menší šance na detekování registrační značky vyrostla šance na její správnou detekci. V tomto konkrétním případě velmi rapidně. Dalším důvodem vybrání této hodnoty bylo, že čím menší počet falešně detekovaných registračních značek bude předáno Tesseractu na rozpoznání, tím rychleji zvládne vstupní soubory zpracovat.

Kapitola 5

Závěr

V rámci této práce byla vytvořena jednoduchá vzorová aplikace, která byla v návrhu rozdělena na konkrétní moduly, které byly vyvíjeny jako větší samostatné celky. Tyto jednotlivé části je možné ovládat pomocí argumentů tohoto programu. Dále bylo úspěšně připraveno zařízení i.MX 6 Series pro tuto aplikaci, na které po potřebném zprovoznění proběhla portace zdrojových souborů. Veškerý překlad a spuštění proběhl úspěšně. Důležitou částí bylo vyhodnocení běhu programu na počítači, kde byl program vyvíjen, a daném zařízení. Z tohoto hodnocení jasně vyplývá, že výkonově se i.MX 6 Series nemůže rovnat výsledkům počítače, ale zároveň ukazuje, že je schopná si se zpracováním obrazu v takovémto měřítku poradit. Cenou je za to však snížení počtu snímků za sekundu či několikavteřinová prodleva při rozpoznávání z fotky. Jako největším zatížením se z jednotlivých měření ukázal Tesseract sloužící k rozpoznávání znaků – konkrétně jeho velmi náročná příprava na rozpoznávání. Z tohoto hlediska by do budoucna stálo za zvážení vytvořit vlastní implementaci na rozpoznávání jednotlivých znaků na registrační značce. Toto řešení by rozhodně pomohlo i na následné zvyšování přesnosti rozpoznávání programu, jelikož u Tesseractu dochází při geometrickém zakřivení registrační značky k velmi citelným ztrátám spolehlivosti rozpoznání.

Přestože nebylo hlavním cílem vytvořit perfektní program na detekci registračních značek, byla zde i snaha o tento záměr. Aktuální dataset obrázků však v této práci činí pouze 508 vzorků použitelných poznávacích značek pro trénování, což se ukázalo jako velmi malé číslo. Zlepšením výstupu by do budoucna určitě bylo natrénování klasifikátoru na větším množství obrázků (např. 2000 vzorků).

Ve výsledku tato aplikace ukázala, že má smysl se pro tyto konkrétní zařízení i do budoucna věnovat programům na detekci např. právě popisovaných registračních značek. Rozhodně tato aplikace poslouží jako odrazový můstek pro další vylepšení či programy. Snaha snížit náročnost a naopak zvýšit přesnost detekce, půjde jistě ruku v ruce rostoucím výkonem dalších nástupců tohoto zařízení i.MX 6 Series, kde tato aplikace může dosahovat vskutku úctyhodných výsledků.

Literatura

[1] Vyhláška č. 343/2014 Sb. apr 2017.
URL https://www.zakonyprolidi.cz/cs/2014-343

[2] Baggio, D. L.: *Mastering OpenCV with practical computer vision projects*. Packt Publishing Ltd, 2012.

[3] Bradski, G.; Kaehler, A.: *Learning OpenCV: Computer vision with the OpenCV library*. "O'Reilly Media, Inc.", 2008.

[4] Brahmbhatt, S.: *Practical OpenCV*. Apress, 2013.

[5] Brahnam, S.; Jain, L. C.; Nanni, L.; aj.: *Local binary patterns: new variants and applications*. Springer, 2014.

[6] Buckland, M. K.: *Emanuel Goldberg and his knowledge machine*. Greenwood Publishing Group, 2006.

[7] Dawson-Howe, K.: *A practical introduction to computer vision with opencv*. John Wiley & Sons, 2014.

[8] Havránek, B.; aj.: *Slovník spisovného jazyka českého*. Nakl. Československé akademie věd, 1960.

[9] Kaehler, A.; Bradski, G.: *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. "O'Reilly Media, Inc.", 2016.

[10] Kalawsky, R.: The science of virtual reality and virtual environments. 1993.

[11] Kelby, S.: *The digital photography book*. Peachpit Press, 2012.

[12] Kroupa, M.: Recenze objektivu MADOKA 180 7,3mm f4. *Fotorádce*, 2012.

[13] Monteiro, G.; Peixoto, P.; Nunes, U.: Vision-based pedestrian detection using Haar-like features. *Robotica*, ročník 24, 2006.

[14] Mori, S.; Nishida, H.; Yamada, H.: *Optical character recognition*. John Wiley & Sons, Inc., 1999.

[15] Rengadurai, R.: Photographer of the week – Anthony Gelot, Cityscape Photographer. *cambyte*, 2015.

[16] SKISOO: How to remove the fisheye effect from GoPro Hero 2 videos with free software ? sep 2012.
URL http://skisoo.com/blog/en/2012/how-to-remove-the-fisheye-effect-from-gopro-hero-2-videos-with-free-software/

[17] Smith, R.: An overview of the Tesseract OCR engine. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, ročník 2, IEEE, 2007.

[18] Szeliski, R.: *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[19] Viola, P.; Jones, M.: Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, ročník 1, IEEE, 2001.

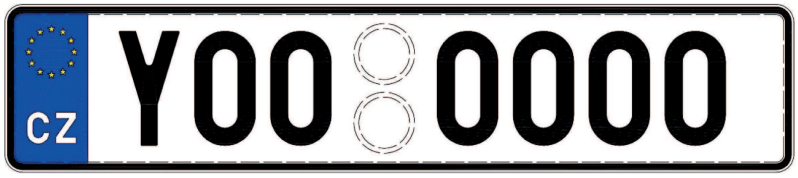
[20] Viola, P.; Jones, M.: Fast and robust classification using asymmetric adaboost and a detector cascade. *Advances in neural information processing systems*, ročník 2, 2002.

Přílohy

Příloha A

Specifikace registračních značek

Registrační značka vozidel se využívá k jednoznačné identifikaci veškerých motorových a přípojných vozidel a musí být na takovém vozidle povinně umístěna (konkrétně dvakrát – jednou z přední části a jednou ze zadní části vozidla). Jedná se obdélníkovou destičku ze slitin lehkých kovů, nejčastěji v bílé barvě, na které jsou nanesena písmena a číslic v černé barvě. Od 1. května 2004 se na českých registračních značkách vyskytuje navíc modrý pruh s dvanácti žlutými hvězdami a zkratkou země (CZ). Samozřejmostí jsou prolisy určené pro nálepky technické kontroly.
Obecně se pro označení těchto destiček používá název Státní poznávací značka (zkráceně SPZ), ale používají se dále i termíny jako poznávací značka či registrační značka (RZ).
Podoba současných registračních značek je předepsaná vyhláškou č. 343/2014 Sb. [1] a v současném provedení jsou vydávány dle platného zákona č. 56/2001 Sb.



Obrázek A.1: Ukázka české SPZ v nejčastějším rozměru 520 × 110 mm [1].

Mezi číslice, která mohou být použita na registračních značkách, patří veškeré arabské číslice (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Naproti tomu mezi písmeny, které jsou povoleny, nejsou všechna písmena české abecedy. Povolené jsou pouze: A, B, C, D, E, F, H, I, J, K, L, M, N, P, R, S, T, U, V, X, Y, Z. Důvodem vynechání zbylých písmen je ten, že jsou oproti jiným hůře rozeznatelná a mohlo by docházet k jejich nechtěnému zaměňování. Pořadí písmen a číslic není nijak limitován a z výše uvedeného výčtu může nastat jejich jakákoliv kombinace. Limitujícím je pouze počet znaků (písmen a číslic) na registrační značce. Ten sice není nijak pevně stanoven, ale pohybuje se (v závislosti na typu) v rozmezí 5 – 8 znaků na registrační značku.
Obecně se dá říci, že celkově rozeznáváme 3 typy státních poznávacích značek:

- Standardní registrační značky
- Registrační značky na přání
- Zvláštní registrační značky

Mezi standardní registrační značky patří takové značky, které jsou povinné pro všechna motorová a přípojná vozidla a jsou to ty značky, které je možné vidět na silnicích nejčastěji. Obrázek takové standardní značky zobrazuje i obrázek A.1. Od roku 2016 je možné si zažádat i o registrační značku na přání. Žadatel má možnost vybrat si z osmi alfanumerických znaků, přičemž alespoň jeden musí být číslo. Samozřejmostí je, že text na SPZ nesmí tvořit žádný vulgární a hanlivý výraz, a také nesmí tvořit název nějakého úřadu. Volba alfanumerických znaků podléhá výše uvedeným povoleným písmenům a číslicím. Dalším typem jsou zvláštní registrační značky – jedná se o speciální SPZ, která jsou vydávány např. historickým vozidlům (nesou písmeno V), sportovním vozidlům (nesoucí písmeno R), vozidlům pro zkušební účely (nesoucí písmeno F) apod.
Mezi další dělení je možné zařadit dělení standardních registračních značek dle písmen na krajskou příslušnost. Jednotlivé rozdělení kódových písmen a krajů je uvedeno v následující tabulce:

Kraj	**Písmeno**
Praha	A
Jihomoravský kraj	B
Jihočeský kraj	C
Pardubický kraj	E
Královéhradecký kraj	H
Kraj Vysočina	J
Karlovarský kraj	K
Liberecký kraj	L
Olomoucký kraj	M
Plzeňský kraj	P
Středočeský kraj	S
Moravskoslezský kraj	T
Ústecký kraj	U
Zlínský kraj	Z

Tabulka A.1: Tabulka krajů a jejich kódových označení

Příloha B

Příklady spuštění

Program je možné ovládat několika parametry. Jejich výčet a popis je možné nalézt v souboru `Readme.txt`, popřípadě zavoláním programu v terminálu s argumentem `-h`:

```
$ ./lp -h
```

Pro vyzkoušení aplikace je možné využít následující příklady:

B.1 Zpracování fotografie

Pro zpracování fotografie se zobrazením výstupu na monitor lze použít následující příkaz ke spuštění:

```
$ ./lp -f foto.png
```

Na výstupu se objeví vstupní fotografie, ve které se odstranil efekt rybího oka, nalezla se registrační značka, rozpoznala se text na ní a vypsal na výstup.

B.2 Zpracování videa

Na místo použití fotografie jako vstupu, je možné nahrát na vstup videosoubor. Spuštění takového programu je možné v následujícím tvaru:

```
$ ./lp -v video.mp4
```

Na výstupu se postupně přehraje videosoubor, který již nebude obsahovat efekt rybího oka z kamery a orámují se detekované SPZ, nad kterými se zobrazí rozpoznaný text.

B.3 Zpracování do souboru

Pro předchozí příklady B.1 a B.2 je možné místo detekce a rozpoznání na výstup uložit veškeré rozpoznané znaky do souboru. Toto lze ovlivnit přidáním parametru `-s`. Spuštění tedy může vypadat například takto:

```
$ ./lp -f foto.png -s
```