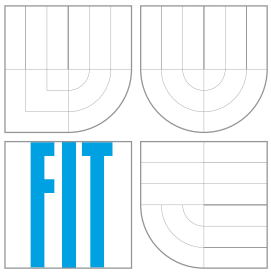
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

CLOUDOVÁ APLIKACE PRO ANALÝZU DOPRAVY

CLOUD APPLICATION FOR TRAFFIC ANALYSIS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VÍT VALCHÁŘ

VEDOUCÍ PRÁCE
SUPERVISOR

prof. Ing. ADAM HEROUT, Ph.D.

BRNO 2016

Abstrakt

Cílem této práce je vytvořit cloudovou aplikaci pro analýzu videa bez znalosti jakýchkoliv parametrů kamery. Jediným vstupem je tak adresa webové kamery. Aplikace je postavena na již existujícím řešení. To je vylepšeno přidáním nového modulu pro odstranění překážek, jako je např. sloup veřejného osvětlení zastiňující část vozovky, a modulu pro rozdělení dvou blízko sebe jedoucích vozidel. Výsledné cloudové řešení je tvořeno soustavou dílčích aplikací, které spolu komunikují pomocí HTTP zpráv a jsou ovládány přes webové rozhraní.

Abstract

The aim of this thesis is to create a cloud application for traffic analysis without knowing anything about the system. The only input is address of the web camera pointing at traffic. This application is build on existing solution which is further enhanced. New modules for removing obstacles (such as lamppost covering part of the road) and splitting overlapping cars were added. The whole cloud solution consists of multiple components which communicates by HTTP messages and are controlled by web interface.

Klíčová slova

Analýza dopravy, Automatická kalibrace kamery, Měření rychlosti, Detekce aut, Klasifikace aut, Cloudová aplikace

Keywords

Traffic analysis, Automatic camera calibration, Speed measurement, Car detection, Car classification, Cloud application

Citace

Vít Valchář: Cloudová aplikace pro analýzu dopravy, diplomová práce, Brno, FIT VUT v Brně, 2016

Cloudová aplikace pro analýzu dopravy

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D.

.....
Vít Valchář
14. května 2016

Poděkování

Chtěl bych poděkovat panu prof. Ing. Adamu Heroutovi, Ph.D. za jeho vedení a věcné připomínky. Zároveň bych rád poděkoval panu Ing. Jakubu Sochorovi za poskytnutí implementace programu a následné konzultace, a v neposlední řadě firmě SDE – Software Solutions za poskytnutí virtuálních serverů.

Obsah

Kapitola 1

Úvod

Analýza dopravy je jedním z užitečných a pro mne nejzajímavějších případů využití počítačového vidění. Stačí jednoduchá kamera snímající provoz a důslednou analýzou lze vypočítat nejen velikost či rychlost vozidla, ale také jeho poznávací značku, nebo typ vozidla.

Existuje mnoho metod, které mají za cíl analyzovat dopravu, nebo alespoň provést základní sledování dopravy. Základní metody potřebují ke své funkci spoustu parametrů zadat ručně. Postupem času se více a více těchto parametrů získává automatickou cestou. Stejně jako v aplikaci, kterou vyvinul Ing. Jakub Sochor jako svou diplomovou práci [15], kterou využívám jako základní implementaci. Jeho aplikace je již v původní podobě velmi rychlá a spolehlivá, přesto je mým cílem ji vylepšit v oblasti detekce a klasifikace automobilů tak, aby fungovala ve všech možných situacích a úhlech ze kterých je snímána, bez nutnosti přetrénovat klasifikátor či jakéhokoliv dalšího nastavování parametrů.

Mým cílem je také aplikaci dále poskytnout jako plnohodnotnou cloudovou aplikaci dostupnou z webu, bez nutnosti jakékoliv interakce uživatele se samotnou aplikací na pozadí a s grafickým výstupem spolu se všemi statistickými daty. Cloudové řešení by mělo být odolné vůči vnějším i vnitřním vlivům jako jsou výpadky sítě, či pády některých součástí a zároveň poskytnout dostatečnou a snadnou škálovatelnost.

Tato práce je rozdělena do několika kapitol, kde existující metody pro analýzu dopravy jsou popsány v kapitole 2, vybrané metody počítačového vidění v kapitole 3, popis aplikace a její rozšíření v kapitole 4 a cloudové řešení v kapitole 5. V kapitole 6 pak naleznete výsledky testování.

Kapitola 2

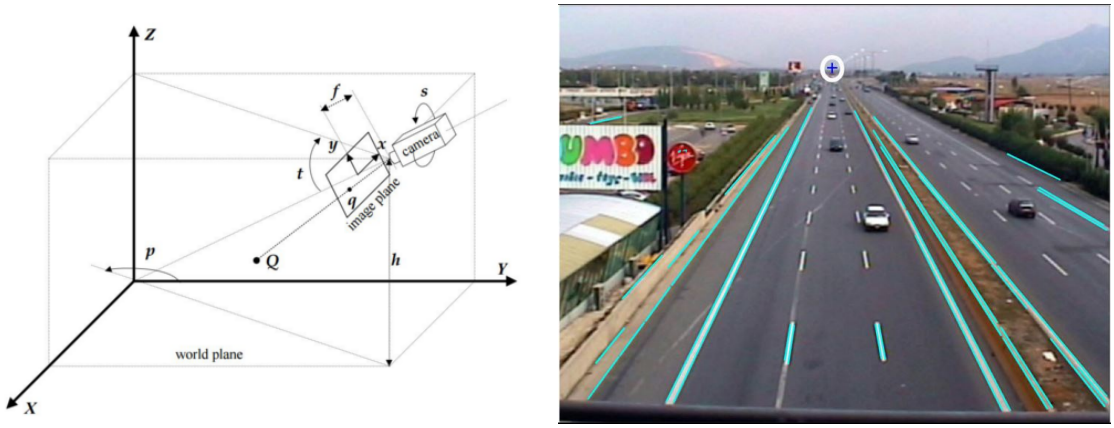
Metody pro analýzu dopravy

V této kapitole bych rád uvedl několik metod pro analýzu dopravy. Nejprve uvedu metody pro kalibraci kamery, na které závisí většina popsaných metod. Dále pak možnosti detekce automobilů v obraze, a nakonec odhad rychlosti pohybu vozidel a klasifikace vozidel.

2.1 Kalibrace kamery

Spousta metod potřebuje nejprve zkalibrovat kameru. Jedna z možností je nalezením základních vlastností kamery (ať ručním zadáním nebo automaticky), kde důležitými vlastnostmi jsou rotace kamery ve 3 osách[1], dále pozice kamery nad zemí, a ohniskovou vzdálenost (obr. 2.1a).

Druhou možností je nalézt tzv. úběžníky (*Vanishing points*). Úběžník vyznačuje bod v prostoru ke kterému se sbíhají všechny rovnoběžné přímky ve snímku (obr. 2.1b). Pomocí těchto bodů pak můžeme vypočítat vlastnosti kamery, jako v předchozím případě.



(a) (b)

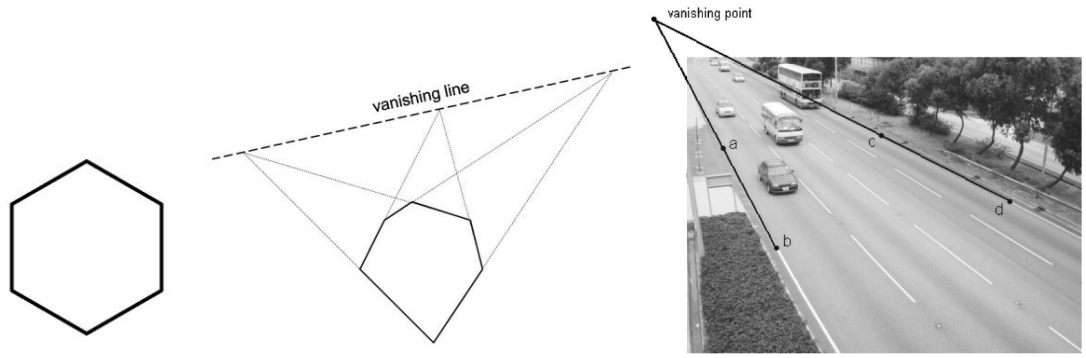
Obrázek 2.1: Základní vlastnosti kamery (a), rotace p, t, s, výška h a ohnisková vzdálenost f, a první úběžník označen kroužkem (b). Převzato z [6] a [11].

[1] U některých metod stačí pouze rotace dvou os, přičemž se předpokládá, že rotace kolem podélné osy je nulová.

2.1.1 Ruční kalibrace

Nejjednodušší metodou z hlediska programu je samozřejmě dostat vlastnosti kamery jako jeden ze vstupů programu. Tento postup je však nevhodný pro reálné použití. Nalezení všech potřebných parametrů ručně může být obtížné např. z důvodu špatného přístupu ke kameře (umístěné na sloupu veřejného osvětlení) nebo při větším množství kamer.

Jako další možnost se proto jeví alespoň poloautomatická kalibrace, kdy se vyžaduje určitý vzor (např. umístění hexagonu na viditelné místo jako v [19]). Díky tomu, že víme přesně, jak původní vzor vypadal, lze spočítat všechny požadované parametry (obr. 2.2a a 2.2b). Nebo můžeme vyžadovat zadání alespoň dvou rovnoběžných hran (neboli 4 bodů) jako v případě [2], kde je důležité označit alespoň dvě podélné čáry, ze kterých se následně vypočte většina parametrů, tedy bez rotace s, pro níž by jsme potřebovali nalézt i 2. úběžník (obr. 2.2c).



(a) (b) (c)

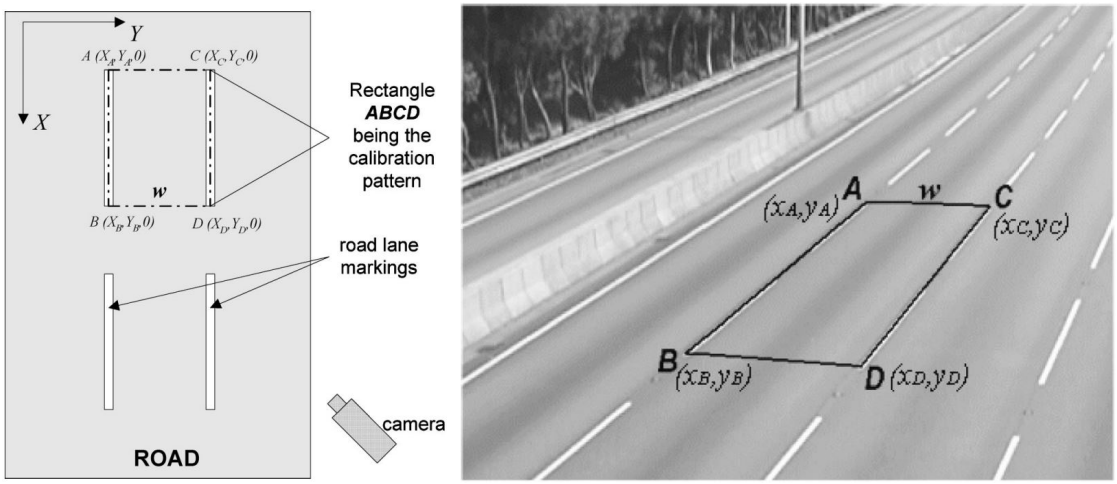
Obrázek 2.2: Hexagon použitý ke kalibraci (a), (b) a ruční zadání 2 podélných čar pro výpočet 1. VP (c). Převzato z [5].

2.1.2 Automatická kalibrace

Jako první z automatických metod bych rád představil metodu George Funga aj. [5] založenou na dvou výše zmíněných. Tato metoda (stejně jako skoro všechny ostatní) předpokládá, že alespoň část cesty je naprosto rovná. Dále předpokládá, že i podélné značení na cestě je rovné a viditelné (obr. 2.3a). Detekcí 4 základních bodů s předpokladem, že šířka pruhu je pevně daná, opět dojdeme k částečnému nalezení vlastností kamery.

Jedna z dalších možností je automatická detekce úběžníků, ze kterých se následně vypočtou vlastnosti kamery. Metod založených na této bázi je více. Jako první bych jmenoval metodu Grammatikopoulose a Karrase [11] založenou na Cannyho detekci hran a následném vytřídění, např. metodou RANSAC (obr. 2.4a), kde pro prodloužení vybraných hran (vytvořením přímek) nalezneme požadovaný úběžník.

Jiné metody (Shih-Hao Yu [22] nebo Markéta Dubská [13], [14]) jsou založeny na prvotním sledování dopravy bez jakékoliv informace. Sbíráním informací o pohybu v obraze určí přímky po kterých se pohybují auta a následně úběžník (obr. 2.4b). Tyto metody mají výhodu toho, že nepředpokládají nic víc než pohyb aut (doba detekce pak ale závisí na velikosti provozu). Na cestě se tedy nemusí nacházet žádné podélné značení nebo svodidla. Tuto metodu popíši podrobněji v sekci 4.1.



(a) (b)

Obrázek 2.3: Znázornění detekce podélných čar (a) a stejný záběr z pohledu kamery, s vyznačenými detekovanými čarami a šířkou jízdního pruhu w (b). Převzato z [5].

(a) (b)

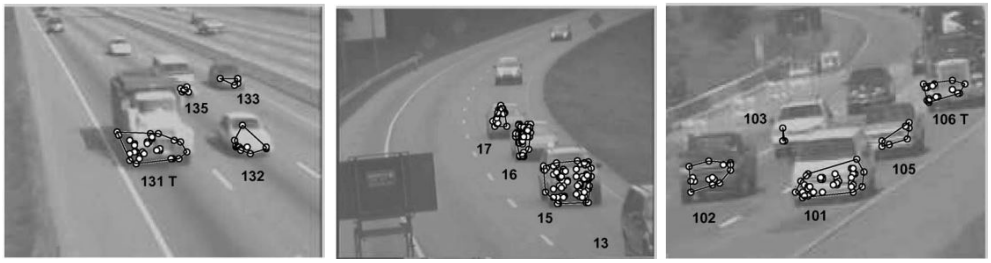
Obrázek 2.4: Detekce hran Cannyho detektorem (a) a detekce pomocí projíždějících aut s detekovanými jízdními pruhy (b). Převzato z [11] a [13].

2.2 Metody pro detekci automobilů

Metod pro detekci automobilů existuje mnoho, já se zde zaměřím pouze na několik metod zajímavých svým přístupem a také metodu využitou ve výsledné aplikaci, která je zároveň nejběžnější za předpokladu, že znají vlastnosti kamery.

2.2.1 Metody založené na extrakci příznaků

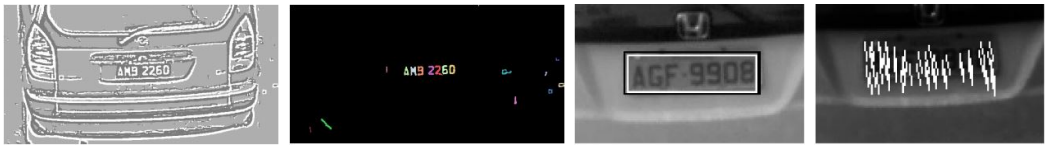
Tyto metody spoléhají na nalezení stabilních, znovu nalezitelných bodů v obraze (*stable features*), na hledaném objektu jako např. v metodě Kanhereho a Birchfielda [9]. Tato metoda spoléhá na nalezení stabilních bodů pomocí metody Lucas-Kanade, kde body nalezené na pozadí jsou zahozeny pomocí metody odečtu pozadí. Tyto body jsou následně sloučeny do skupin (obr. 2.5). Tato metoda nemá problém s částečným překrytím aut.



(a) (b) (c)

Obrázek 2.5: Výsledky metody založené na příznacích pomocí metody Lucas-Kanade. Převzato z [9].

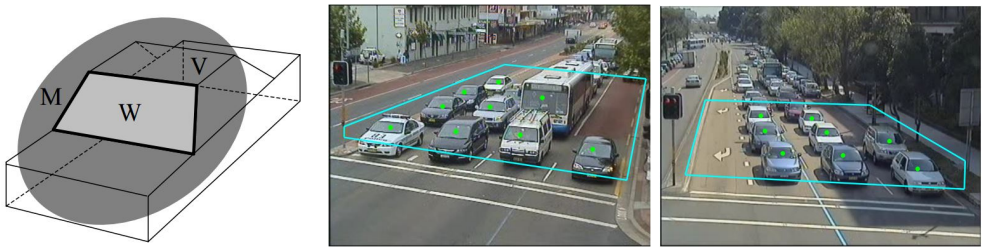
Některé z těchto metod vyhledávají určitou část auta, která je na všech typech aut stejná. Přičemž nemusí být umístěna vždy na stejném místě jako v případě D. Luvizona aj. [12], kde v obraze hledají státní poznávací značku (SPZ). Tu lze nalézt vcelku jednoduše pomocí detekce hran a následné detekce čísel a znaků (obr. 2.6). Nakonec se na značce zaznačí význačné body, díky kterým se poté odhadne rychlost (což popíši v sekci 2.3).



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

Obrázek 2.6: Znázornění postupu detekce poznávací značky podle [12]. Detekce hran (a), detekce znaků (b), určení SPZ (c) a pohybové vektory po přidání dalšího snímku (d).

Dalším příkladem, že pro změření přibližné rychlosti automobilu jej nepotřebujeme detekovat celý ale pouze přesně danou část, je přístup J. Yanga aj. [21], kde detekují čelní sklo na základě tvaru skla, horizontálních hran skla, a průměrné barvy. Výsledky těchto operací se zpracují pomocí mean-shift algoritmu a po následné filtraci slabých, či zdvojených detekcí získáme poměrně přesný detektor aut bez nutnosti předchozích kalibrací. Výsledky této metody jsou na obr. 2.7.

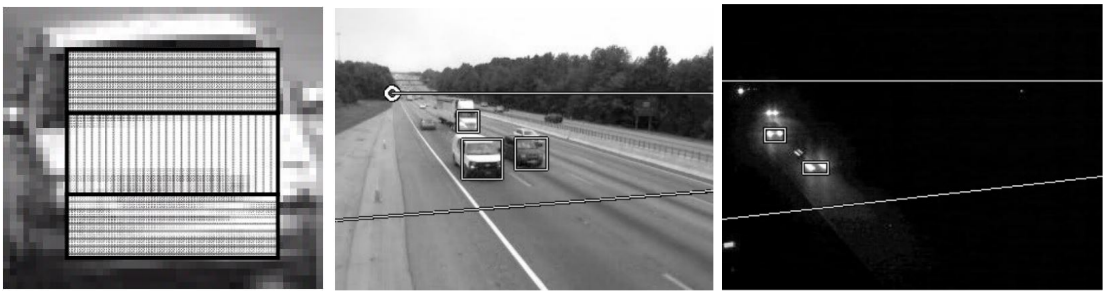


(a) (b) (c)

Obrázek 2.7: Znázornění vyhledávaného tvaru a pozice čelního skla (a), výsledky metody (b) a (c). Převzato z [21].

Poslední metodou, kterou bych rád zmínil v této sekci, je nalezení automobilu dle rozpoznávání vzorů pomocí kaskádového klasifikátoru Viola and Jones, který využil N. Kanhere aj. [10]. Ten vyhledává určité regiony odpovídající naučeným vzorům předních masek

aut. Kamera tedy musí být umístěna v určitém rozmezí úhlu aby detekce byla úspěšná. Výsledky metody jsou zobrazeny na obr. 2.8.



(a) (b) (c)

Obrázek 2.8: Znázornění jednoho jednoduchého klasifikátoru z kaskády (a) a výsledky metody ve dne (b) a v noci (c). Převzato z [10].

2.2.2 Metody založené na odečtu pozadí

Na rozdíl od výše zmíněných metod, tyto metody jsou nezávislé na úhlu kamery k vozidlům a na směru pohybu vozidel. Naopak díky použití metody odečtu pozadí se zde objevuje množství šumu, který musí být filtrován. Navíc zde nastává velký problém při sebemenším překrytí dvou či více aut, kdy nejsme schopni jednoduše rozlišit jednotlivá auta, což ve většině metod založených na příznacích nenastává. Takováto metoda je využita ve více pracích [6, 22, 11, 18]. Celý postup je podrobněji popsán v sekci 4.2.

2.3 Odhad rychlosti vozidel

Zatím jsem se setkal pouze se dvěma základními přístupy, přičemž oba vyžadují alespoň částečnou kalibraci kamery.

Prvním z nich využívá warpování obrazu z perspektivního snímku do pohledu shora [6, 11, 18] (obr. 2.9), ve kterém je nezbytné znát pouze poměr skutečné délky na jeden pixel (l). Poté lze pomocí jednoduchého vzorce

$$v = \frac{PD * l * R}{t}, \tag{2.1}$$

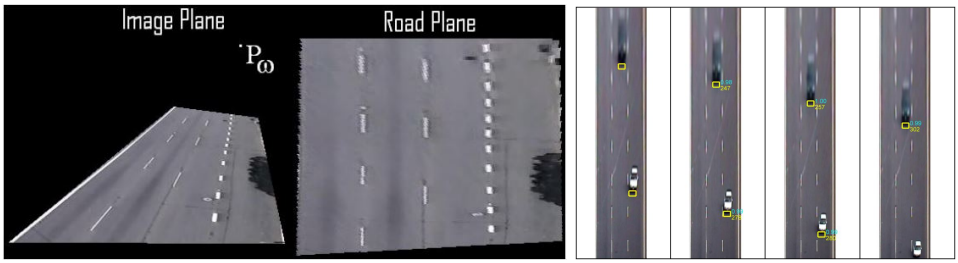
kde PD je délka posunutí auta na 2 po sobě jdoucích snímcích v pixelech, R je rozlišení obrazu a t je čas mezi dvěma snímky, odhadnout přibližnou rychlost vozidla.

Druhou možností [10] je pak znát min. 2 úběžníky (jeden ve směru podélných čar na silnici, a druhý ve směru příčných čar na silnici), nebo rovnou vlastnosti kamery. Následně můžeme přepočítat souřadnice z obrazu na souřadnice reálného světa pomocí rovnic

$$x = \frac{uh}{v\cos\phi + f\sin\phi}, \tag{2.2}$$

$$y = \frac{h(f - v\tan\phi)}{v + f\tan\phi}, \tag{2.3}$$

kde (u, v) jsou souřadnice bodu v obraze, (x, y) jsou reálné souřadnice ve světě a h, f, ϕ jsou parametry kamery jak byly popsány v sekci 2.1 a znázorněny na obr. 2.1a. Díky znalosti reálných souřadnic můžeme vypočíst vzdálenost, kterou auto ujelo za čas mezi jednotlivými snímky jako v předchozím případě.



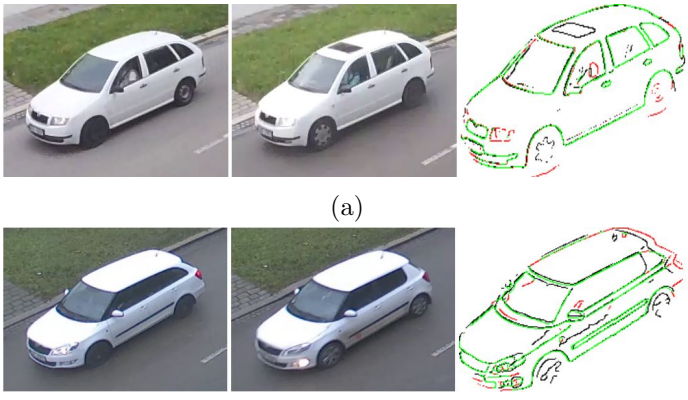
(a) (b)

Obrázek 2.9: Warpování obrazu (a) a měření vzdálenosti na po sobě jdoucích snímcích (b). Převzato z [18] a [11].

2.4 Klasifikace automobilů

Ačkoliv klasifikace automobilů není tak častá jako odhad rychlosti, rád bych zde zmínil dvě zajímavé metody, ze kterých (částečně) vychází metody využité v samotné aplikaci.

Přístup Jakuba Sochora [8] je založen na detekci hran automobilu, který je tak přesný, že určí nejen jestli je porovnávané auto osobní či nákladní, ale dokonce o jaký typ vozidla se jedná (obr. 2.10). Problém tohoto přístupu tkví v nutnosti stejného úhlu kamery pro všechny záznamy a také velkého datasetu.



(a)

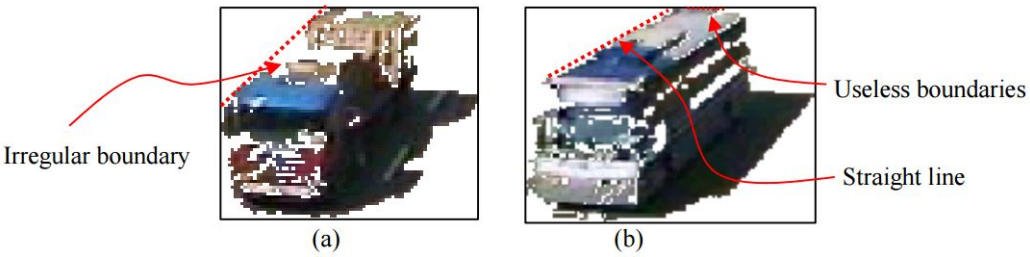
(b)

Obrázek 2.10: Klasifikace stejného modelu auta (a) a odlišného (b). Převzato z [8].

V aplikaci je tato metoda promítnuta využitím HOG detektoru pro rozlišení osobních a nákladních aut.

Jiný přístup zvolil Shih-Hao Yu [22], jeho metoda využívá dvou příznaků a sice velikost auta a „linearitu“ hran. Pro základní rozdělení osobních a nákladních aut postačí velikost auta, která je sbírána po celou dobu jízdy jednoho vozidla a normalizována šířkou jízdního pruhu. Ten je automaticky detekován díky použitému způsobu kalibrace kamery dle projíždějících aut.

Pro podrobnější rozlišení, zda se jedná o nákladní auto či autobus se využívá již zmíněná linearita hrany znázorněná na obr. 2.11.



Obrázek 2.11: Znázornění nelinearity hrany u nákladního automobilu (a) a linearity hrany u autobusu (b). Převzato z [22].

Kapitola 3

Použité algoritmy počítačového vidění

Tato kapitola popisuje některé metody počítačového vidění, které jsou použity ve výsledné aplikaci a to zejména Kalmanův filtr, využitý pro sledování aut a Houghova transformace pro nalezení všech 3 úběžníků.

3.1 Klasifikátor SVM

SVM (*Support Vector Machines*) je metoda strojového učení používaná, mimo jiné, pro klasifikaci objektů, kde má za cíl oddělit 2 třídy objektů nadrovinou tak, aby mezi nimi byl co největší volný prostor. Ten se spočítá jako vzdálenost této nadroviny s nejbližším objektem dané třídy v n-dimenzionálním prostoru. Objekty, které jsou nejblíže specifikované rovině, se pak nazývají *support vectors* a celá nadrovina je definována právě jimi. Díky tomuto mechanizmu nemůže dojít k přetrénování klasifikátoru [16].

Nadrovina je definována tímto výrazem:

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 = 0, \tag{3.1}$$

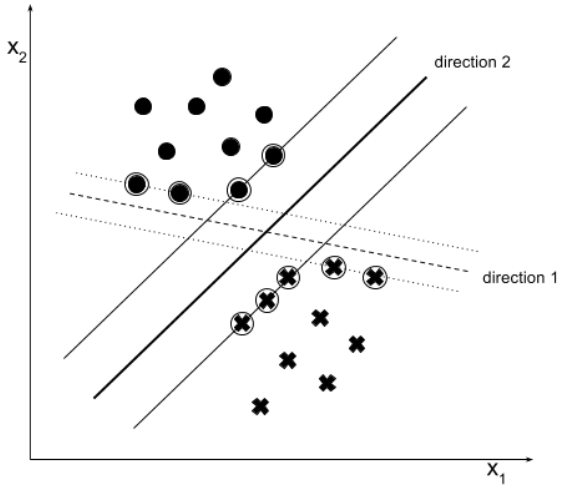
kde $\boldsymbol{w}$ je normálový vektor nadroviny, a w_0 je skalární práh. Takovýchto nadrovin může být samozřejmě nekonečně mnoho. Na obr. 3.1 jsou znázorněny 2 takovéto roviny. Obě jsou legitimní, avšak i pouhým okem lze rozpoznat, že rovina v druhém směru bude daleko lepší pro následné určení třídy s neznámými daty [17].

Jelikož je SVM algoritmem strojového učení, je nutné jej, pro jeho správnou funkci, nejprve natrénovat na trénovacích datech.

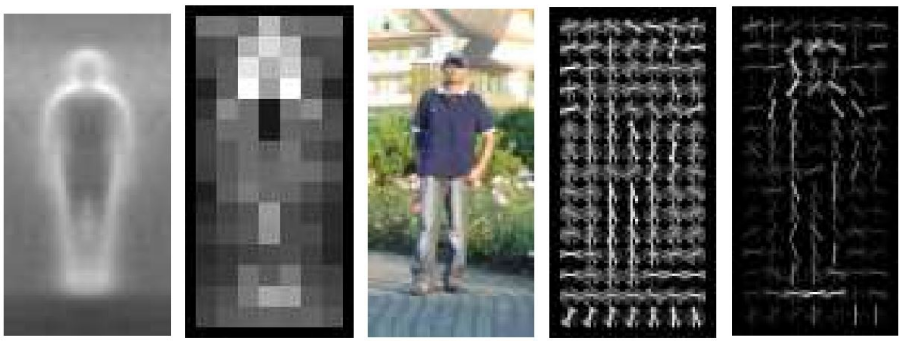
3.2 Histogram of Oriented Gradients

Histogram orientovaných gradientů (HOG) je řetězec postupně aplikovaných metod ústící k lepší extrakci příznaků z obrazu. HOG byl původně vytvořen pro detekci osob N. Dalalem a B. Triggsem [3], ale lze jej použít i pro jiné objekty, mezi které patří i automobily.

První z využitých metod je normalizace barvy. Následně se vypočítají gradienty (tedy se aplikuje detekce hran) ze které jsou následně spojeny po malých blocích (většinou 6×6px) do jednoho 1D histogramu. Tyto bloky mohou být spojeny se svými sousedy a znormalizovány v rámci této skupiny. Nakonec se tyto histogramy sečtou a vytvoří jeden vektor příznaků se kterým pak pracuje klasifikátor, jako je např. lineární SVM. Výsledek je na obr. 3.2.



Obrázek 3.1: Dvě třídy bodů ve 2D s dvěmi možnými nadrovinami. Body v kroužku označují *support vectory*. Překresleno dle [17].



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

Obrázek 3.2: Průměrný gradient pro osobu (a), váhy jednotlivých bloků přidělené SVM klasifikátorem (b), testovací obrázek (c), výsledný vektor metody HOG (d) a výsledný vektor ováhován dle SVM klasifikátoru (e). Převzato z [3].

3.3 Kalmanův Filter

Kalmanův filtr je metoda využívána při sledování objektů, kde na základě dosavadního chování předpovídáme kde se bude sledovaný objekt nacházet v příštím snímku [20]. Tato metoda je typu prediktor-korektor, kdy v první fázi predikujeme novou pozici $\boldsymbol{x}_{k+1}$ dle vzorce

$$\boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + \boldsymbol{w}_k, \tag{3.2}$$

kde A_k je matice změny předchozích stavů (pozic) a $\boldsymbol{w}_k$ jsou parametry Gaussova šumu. Z toho také vyplývají omezení Kalmanova filtru. Předpokládá totiž, že pohyb sledovaného objektu je lineární, a obsahuje bílý Gaussovský šum (a to jak ve sledovaném systému, tak sledovaném objektu). Ten je následně automaticky korigován úpravou kovarianční matice na základě odchylky předpovídaného stavu od později změřeného.

V druhé fázi tedy vypočteme opravdový stav objektu

$$\boldsymbol{z}_{k+1} = H_k \boldsymbol{x}_k + \boldsymbol{v}_k, \tag{3.3}$$

kde H_k je matice skutečné změny stavu a $\boldsymbol{v}_k$ jsou opět parametry Gaussova šumu, a vypočteme chybu (zjednodušeně) jako

$$e_k = x_k - z_k \tag{3.4}$$

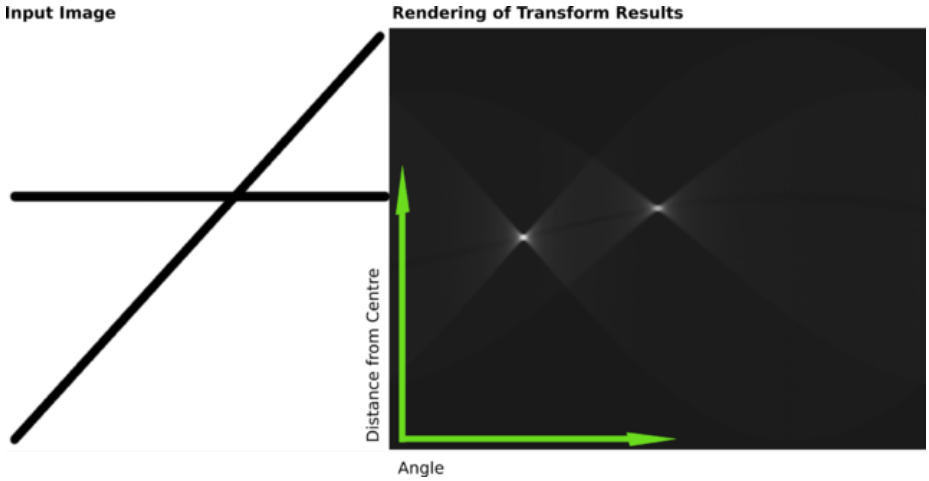
Nakonec upravíme kovarianční matice, tak aby chyba byla co nejmenší (resp. nulová), čímž zpřesníme budoucí odhad stavu objektu.

3.4 Houghova Transformace

Houghova transformace je jednou ze základních technik pro zpracování obrazu. Tato metoda byla původně navržena k detekci rovných čar [7] a následně upravena pro detekci složitějších parametrizovatelných objektů, jako jsou kružnice, elipsy, atd. [1].

Tato transformace používá tzv. Houghův prostor neboli prostor parametrů, který má tolik dimenzí, kolik je potřeba parametrů pro popis hledaného objektu. Pro nalezení přímky nám tedy stačí 2 parametry ($kx+q$), ale pro kružnici již potřebujeme parametry 3 ($a \cdot x^2 + b \cdot y^2 = r^2$). Jeden bod v prostoru parametrů tudíž popisuje celý objekt.

Před použitím transformace se na obraz může aplikovat detektor hran (např. Cannyho detektor). Samotná transformace pak probíhá tak, že pro každý bod obrazu vypočítáme všechny možné objekty, které by tímto bodem procházely, a pro každý z těchto objektů přičteme 1 na odpovídající pozici v prostoru parametrů. Výsledek této transformace pro 2 přímky je na obr. 3.3. Po nalezení maxim v tomto prostoru parametrů získáme parametry objektů které se v původním obraze nacházejí.



Obrázek 3.3: Ukázka vstupního obrázku (vlevo) a prostoru parametrů (b), ve kterém se nacházejí 2 maxima pro 2 přímky.[1]

[1] Převzato z https://en.wikipedia.org/wiki/Hough_transform#/media/File:Hough-example-result-en.png

Kapitola 4

Aplikace pro analýzu dopravy

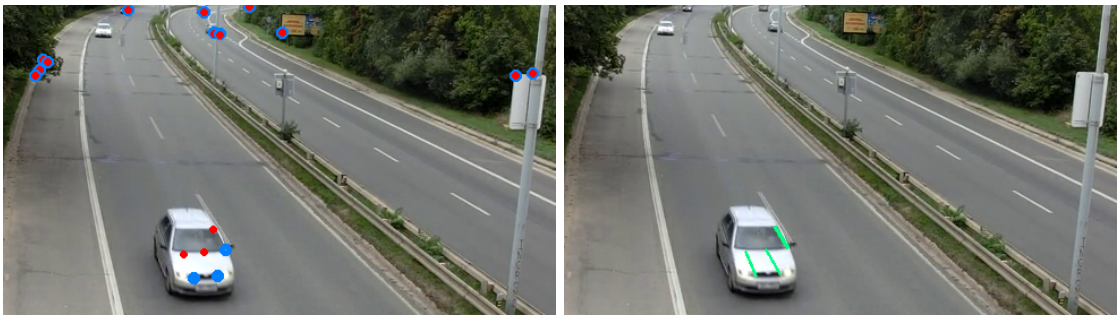
Tato práce navazuje na aplikaci Ing. Jakuba Sochora, kterou implementoval jako svou diplomovou práci [15]. Mým cílem je vylepšit tuto aplikaci a proto bych ji zde nejdříve rád popsal v následujících dvou sekcích. Při psaní těchto sekcí jsem čerpal ze zmíněné diplomové práce a z kódu samotného. V následujících sekcích pak popíši mnou nově přidané moduly více podrobně.

4.1 Nalezení úběžníků

Tato aplikace využívá 3 úběžníků pro kalibraci kamery, které získá metodou postupného sledování dopravy. V této sekci podrobněji popíšu nalezení všech tří.

4.1.1 Nalezení prvního VP

První z úběžníků určuje směr rovnoběžný s podélným značením na cestě, a tedy i směr ze kterého/ke kterému míří všechny auta. K jeho nalezení se využívá nalezení toku dle stabilních příznaků (obr. 4.1). V potaz se berou vždy body z aktuálního a minulého snímku, přičemž k akumulaci dochází pouze u dvojic, které se posunuly o určitou minimální vzdálenost.



(a) (b)

Obrázek 4.1: Zobrazení příznakových bodů, červeně minulý snímek, modře aktuální snímek (a) a vybrané dvojice pro akumulaci (b).

K akumulaci směrů a následnému nalezení úběžníku se využívá Houghova transformace a tzv. diamantový prostor [4]. Ten je využit pro mapování přímek z kartézských souřadnic

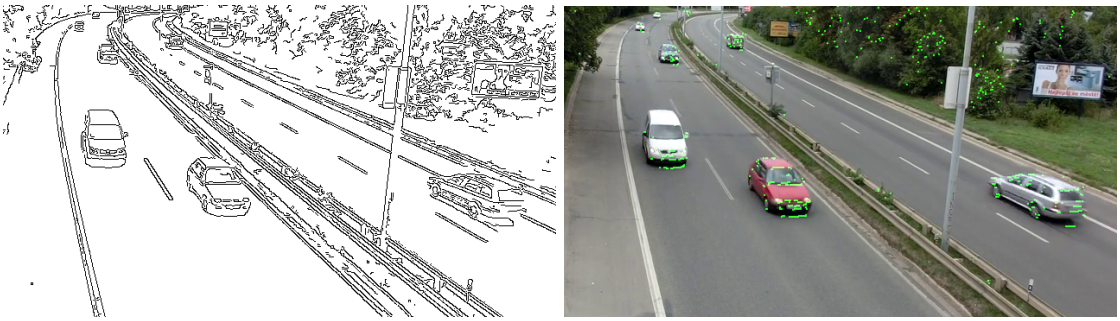
do paralelního prostoru[1]. Každá přímka je pak zobrazena jako lomená čára v diamantovém prostoru. Nalezením globálního maxima v diamantovém prostoru tedy nalezneme bod, který patří nejvíce přímkám v původním prostoru a tedy náš první úběžník. Znázornění diamantového prostoru je na obr. 4.2.

(a) (b) (c)

Obrázek 4.2: Vyobrazení diamantového prostoru pro první úběžník (a) a (b) a pro druhý úběžník (c). Globální maximum je označeno červeným kroužkem.

4.1.2 Nalezení druhého VP

Druhý úběžník určuje bod ke kterému se sbíhají všechna příčná značení a je tedy kolmý na směr prvního úběžníku. K jeho nalezení se využívá Cannyho detektor hran a následně Sobelův filtr o velikosti 5×5 v obou směrech pro nalezení úhlu gradientu. Dále se využívá model pozadí pro odstranění detekovaných hran v pozadí snímku a nakonec jsou vyřazeny hrany, které jsou vertikální, nevýrazné, nebo směřují k prvnímu úběžníku (obr. 4.3). Zbylé hrany (po prodloužení na přímky) jsou opět akumulovány v diamantovém prostoru, díky kterému je nalezen i druhý úběžník.



(a) (b)

Obrázek 4.3: Výsledek detekce hran pomocí Cannyho detektoru (a) a vyznačené hrany použité k detekci 2. VP (b).

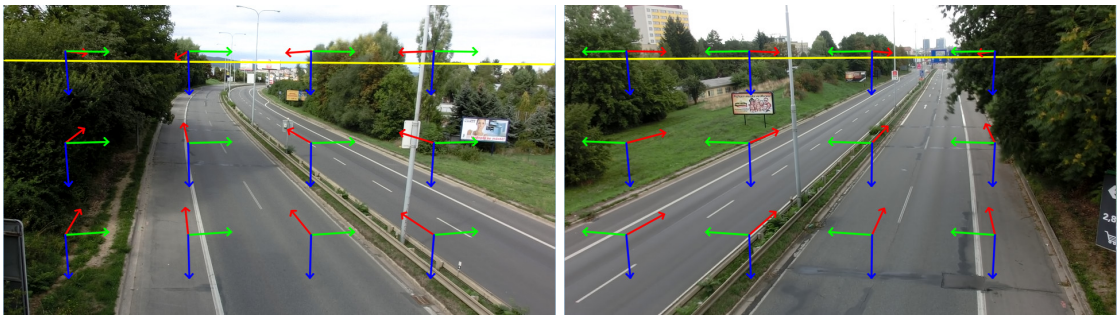
[1] V paralelním prostoru jsou osy vzájemně rovnoběžné a jakýkoliv bod v původním prostoru je zobrazen jako lomená čára.

4.1.3 Nalezení třetího VP a ohniskové vzdálenosti

Pro další výpočty budeme předpokládat, že známe první dva úběžníky (U a V), a že optický střed čočky (*principal point*) P je ve středu obrazu. Pak můžeme vypočítat ohniskovou vzdálenost f a reálné souřadnice všech tří úběžníků U', V', W' rovnicemi:

$$f = \sqrt{-(U - P) * (V - P)}, \tag{4.1}$$
$$U' = (U_x, U_y, f), \tag{4.2}$$
$$V' = (V_x, V_y, f), \tag{4.3}$$
$$P' = (P_x, P_y, 0), \tag{4.4}$$
$$W' = (U' - P') \times (V' - P'), \tag{4.5}$$

Po nalezení ohniskové vzdálenosti a všech tří úběžníků je kamera zkalibrována a můžeme začít se samotnou detekcí. Výsledek kalibrace je znázorněn na obr. 4.4.



(a) (b)

Obrázek 4.4: Znázornění detekovaných úběžníků na dvou různých záběrech.

4.2 Detekce automobilů

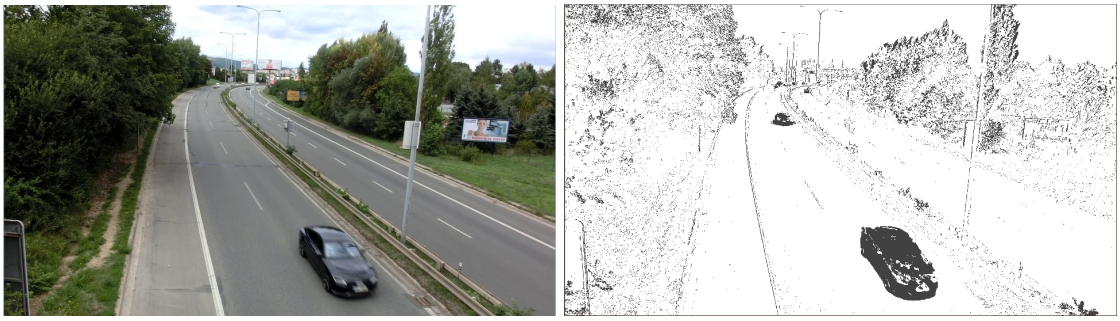
Nyní již známe úběžníky a dokážeme určit reálný souřadný systém pro každý bod v obraze. Na řadě je tedy využití této znalosti pro detekci a následnou klasifikaci automobilů.

Snímky jsou načítány z videa (offline) nebo streamu (online). V obou případech video podvzorkujeme na rychlost přibližně 12-14 snímků za sekundu. Taková rychlost je zvolena proto, aby se snímané objekty dostatečně posunuly mezi jednotlivými snímky a metoda odečtu pozadí nám dávala (z našeho pohledu) lepší výsledky.

4.2.1 Prvotní detekce oblastí s auty

První na řadě (po načtení a případném zmenšení snímku) je tedy využití metody odečtu pozadí. Tato metoda je velice jednoduchá a rychlá. Jde pouze o rozdíl dvou snímků v jase každého pixelu. Pro tuto metodu se většinou používá šedotónový obrázek. Výsledkem této metody je pak obrázek stejné velikosti s vyznačenými místy, které se mezi jednotlivými snímky liší (obr. 4.5b).

Jelikož výsledný obrázek obsahuje příliš mnoho šumu, provádí se dále morfologické operace otevření a uzavření (skládající se z eroze a dilatace v určeném pořadí). Tyto operace odstraní osamocené body (tedy šum) a přitom vyplní případné malé díry ve větších celcích (obr. 4.6a).



(a) (b)

Obrázek 4.5: Jeden z původních obrázků (a) a výsledný obrázek metody odečtu pozadí (b).

V tomto kroku se také výsledné detekované oblasti vyfiltrují na základě předpřipravené černobílé masky (pokud byla nastavena). Ta se aplikuje pomocí bitového násobení pro každý pixel obrazu.

Nakonec se naleznou kontury detekovaných oblastí a kolem nich nejmenší obalující obdélník rovnoběžný se souřadným systémem (obr. 4.6b), se kterým budeme pracovat v následujících krocích.

(a) (b)

Obrázek 4.6: Obrázek po aplikování operací otevření a uzavření (a) a původní barevný obrázek se zakreslenými konturami a obalujícím obdélníkem (b).

4.2.2 Sledování detekovaných oblastí

Na základě obalového obdélníku detekované oblasti z předchozího kroku nyní vytvoříme záznam o nalezeném objektu. Tento záznam obsahuje již zmíněnou konturu a obalový obdélník, dále pozici středu a model skládající se z histogramu jednotlivých složek barev formátu HSV, vypočítaný z výřezu dle obalového obdélníku. Model je znázorněn na obr. 4.7 pro 2 vozidla.

V tuto chvíli je na řadě nalezení odpovídající, již detekované oblasti z dřívějších snímků. Pro názornost napřed popíši případ, že pro aktuální oblast neexistuje předchozí detekce. A následně samotné prohledávání.

V takovém případě se vytvoří nový, globální záznam obsahující navíc jednoznačný identifikátor a vyřezaný snímek. V tuto chvíli se také nastaví parametry pro sledování pomocí Kalmanova filtru, popsaného v kapitole 3.3, a vypočítá se 3D obalující kvádr (pomocí znalosti světa z kapitoly 4.1).

(a) (b)

Obrázek 4.7: Histogram kanálů HS, kanálu V a výsledný spojený a normalizovaný histogram pro tmavé vozidlo (a) a pro světlé vozidlo (b).

Nyní se můžeme podívat na tu zajímavější část, kterou je nalezení odpovídající nové detekce k již stávajícímu seznamu detekcí. Tato se provádí s využitím Kalmanova filtru. Pro každý objekt z globálního seznamu tedy vypočítáme předpokládanou pozici, kde by se měl nacházet na aktuálním snímku a tu porovnáme s pozicí nového objektu. Pokud jsou dostatečně blízko, a jejich barevné modely se shodují, můžeme prohlásit, že nový objekt patří k tomuto globálnímu objektu.

Při každém spárování se aktualizuje aktuální poloha globálního objektu a korekční matice Kalmanova filtru. Také je vytvořen nový obalový kvádr s výřezem obrázku. Většina těchto údajů je uložena k objektu a proto máme později k dispozici celou historii, která je následně využita ke klasifikaci a dalším výpočtům.

Pokud pro daný globální objekt nelze nalézt nový objekt po dobu několika snímků je velmi pravděpodobné, že již opustil zaznamenávaný prostor a není tudíž potřeba tento objekt nadále vyhledávat. V ideálním světě by byl nyní takto detekovaný globální objekt s celou historií označen za auto. To ovšem nemusí být pravda. Proto je nutná další analýza a klasifikace.

4.2.3 Analýza a klasifikace detekovaného objektu

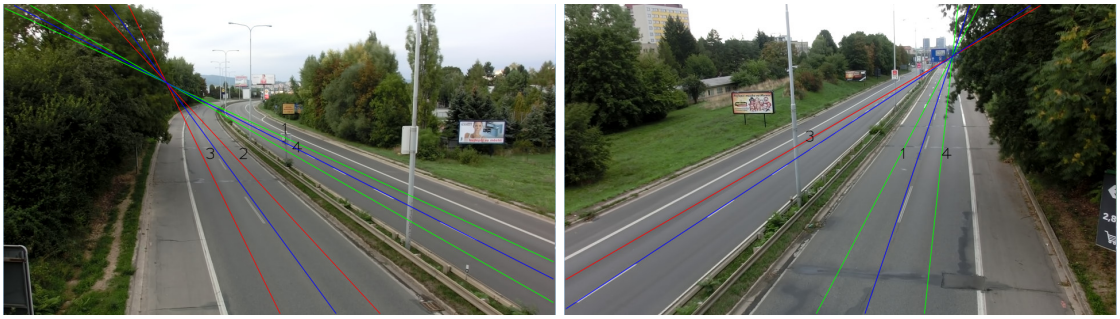
Nejprve tedy musíme zjistit, zda se opravdu jedná o automobil, či pouze náhodnou detekci. Tou může být např. šum v obraze nebo pohyb okolí (pohyb listů stromů ve větru). Nyní je využito historie pozic detekovaného objektu po celou dobu jeho výskytu. Pomocí první a poslední pozice vytvoříme pomyslnou přímku, po které se objekt pohyboval. Pokud jsou všechny pozice mezi těmito body poblíž vytvořené přímky a pokud tato přímka směřuje k prvnímu úběžníku a navíc se objekt posunul o určenou minimální vzdálenost, pak je tento objekt označen za auto.

Námi vytvořená přímka pohybu auta je ještě využita pro detekci směru jízdy, tedy zda jede k úběžníku nebo od něj, a také pro následnou detekci jízdních pruhů. Ta je realizována pomocí histogramu přímek vytvořených z bodů v těžišti auta, a druhého histogramu přímek vytvořených z krajních bodů. Ten se využívá pro detekci čar dělících jízdní pruhy. V těchto dvou histogramech nalezneme lokální maxima, které překračují určitou hodnotu (obr. 4.8). K jednotlivým pruhům se nakonec přidá informace o směru jízdy aut v něm (k/od VP) a přiřadí se jednoznačné identifikátory.

Jízdní pruhy jsou znovu přepočteny vždy po 200 projetých vozidlech a původní značení je přepočteno (a to i v případě, že některý pruh nebyl detekován) tak, že se přidělené identifikátory nezmění a dají se tak použít pro statistické účely.

Následně se provádí výpočet rychlosti vozidla. Ten je závislý na reálné vzdálenosti, a přesném času za který jej vozidlo ujelo. Čas lze určit poměrně přesně pomocí počtu snímků a hodnotě fps[2]. Ujetá vzdálenost je pak vzdálenost 2 bodů, ve kterých se vozidlo

[2]Počet snímků za sekundu (*Frames per second*).



(a) (b)

Obrázek 4.8: Znázorněny detekované jízdní pruhy a dělící čáry, směr od VP červeně, k VP zeleně a dělící čáry modře. Úspěšná detekce všech 4 pruhů (a) a momentální detekce bez pruhu č. 2 (b).

nacházelo, přepočtených na reálné souřadnice. Jelikož je však kamera zkalibrována pomocí úběžníků nejsou tyto hodnoty v metrech, ale v určitém, zatím neznámém měřítku.

Měřítko λ se vypočítá díky předpokladu, že většina aut má přibližně stejné rozměry. V průběhu detekce se tedy sbírají údaje o výšce, šířce a délce všech aut a jejich medián se využije při výpočtu měřítka spolu s mediánem velikosti běžných aut (4,27m x 1,74m x 1,51m). Jednotlivé poměry by měly být stejné, ale díky perspektivnímu zobrazení se většinou mírně liší a tak je vybrán ten nejmenší, a je využit pro přepočet ujeté vzdálenosti na skutečné jednotky.

Nakonec použijeme HOG (popsanou v kapitole 3.2) na jednom či několika výřezech, na kterých se automobil nachází. Těmto výřezům je nejprve změněna velikost tak, aby všechny byly stejně velké. HOG se poté provede na celém takovémto snímku pouze jednou (nemusíme obrázky nadále škálovat) a je tak rychlejší.

Výsledkem je pole deskriptorů popisujících daný automobil. Tyto využijeme při klasifikaci pomocí SVM (kapitola 3.1). Tento klasifikátor musí být natrénován předem na trénovacích datech, které můžeme z aplikace také získat.

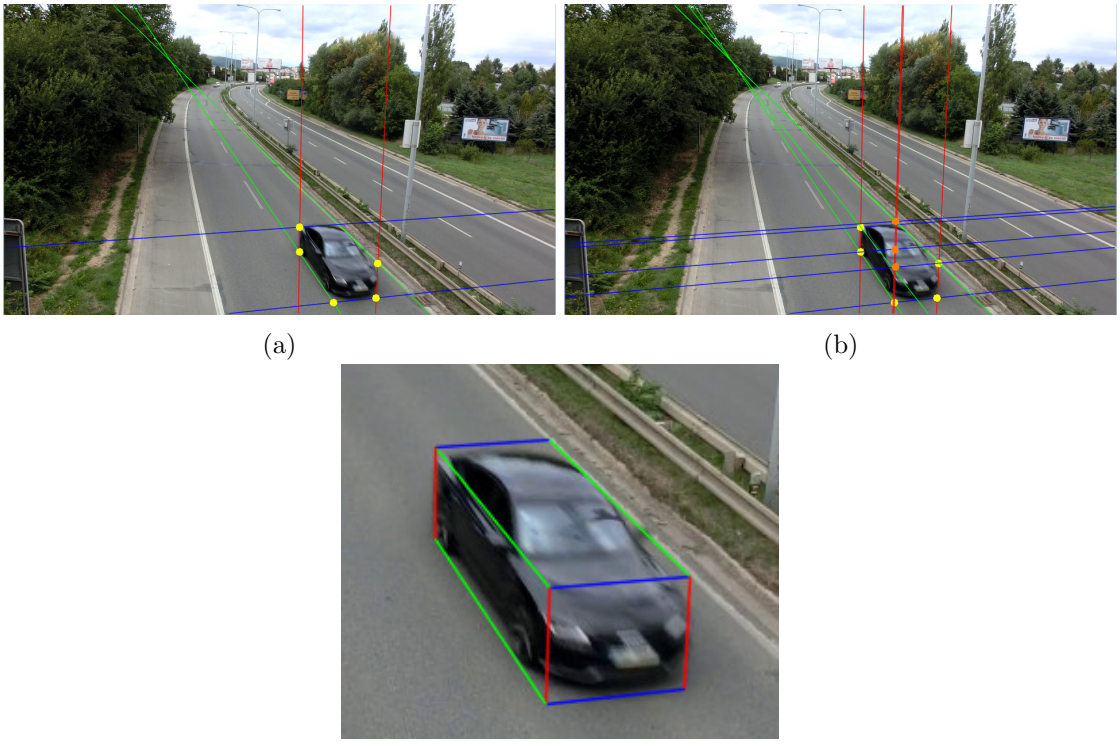
Po těchto krocích známe všechna statistická data o průjezdu jednoho automobilu. Známe typ (osobní, nákladní, a další, které klasifikátor naučíme), průměrnou rychlost i velikost auta. Právě tyto údaje mohou být při ukončení aplikace vyexportovány do souboru CSV.

4.2.4 Sestavení obalového kvádru

Pro sestaveni 3D obalového kvádru se využívá již nalezené kontury auta a všech tří úběžníků. V prvním kroku se nalezne nejlevější a nejpravější bod kontury, tyto dva body následně vymezují prostor pro sestavení obalového kvádru. Poté je potřeba nalézt 6 základních hran stavěného kvádru tak, že pro každý bod kontury je vypočtena přímka směřující k jednomu z úběžníků, přičemž hledáme přímky s nejmenším a největším úhlem. Tyto základní hrany jsou zobrazeny na obrázku 4.9a.

Nalezené hrany musíme nejprve setřídit a rozlišit která z dvojic (pro každý úběžník) je vlevo, příp. nahoře. Poté již můžeme přistoupit k samotnému sestavení kvádru. Ten se skládá z 8 bodů. My ovšem momentálně známe pouze 6 přímek. Nejprve tedy nalezneme 5 bodů z průsečíků těchto přímek (obr. 4.9a). Následně sestavíme odvozené přímky z nově nalezených bodů a úběžníků (obr. 4.9b), kde výpočet zbylých 3 bodů je opět záležitostí výpočtu správných průsečíků. Po jejich nalezení již můžeme sestavit celý kvádr (obr. 4.9c).

Správné sestavení obalového kvádru je velmi závislé na předchozí detekci úběžníků a díky přepočtu délky hrany do reálných jednotek lze následně využít pro zlepšení klasifikace aut, což ukážu v příští sekci 4.3.



(a) (b)

(c)

Obrázek 4.9: Postup vytváření obalového kvádru. (a) určení hlavních přímek a bodů (žlutě), (b) určení vedlejších přímek a bodů (oranžově) a (c) výsledný kvádr.

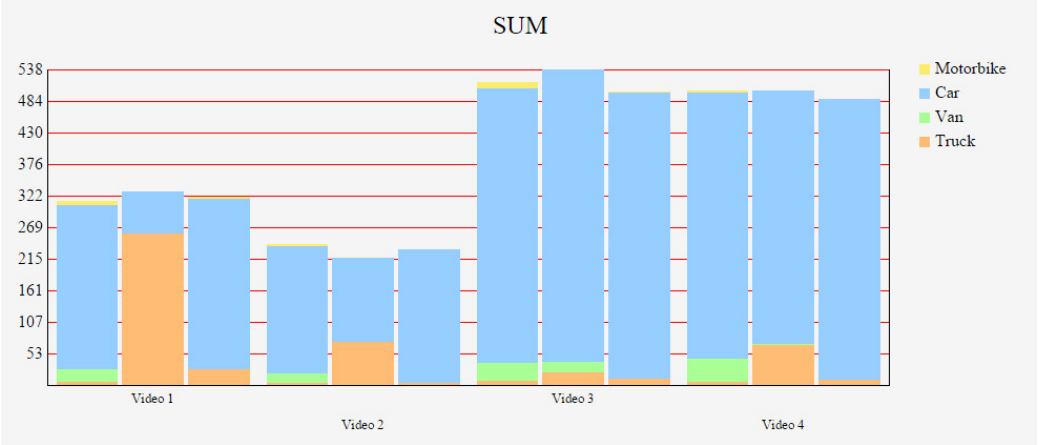
4.3 Klasifikace aut pomocí rozměrů auta

Tento způsob klasifikace je výsledkem řešení problému klasifikace na základě HOG detekce. Na tento problém jsem narazil při spuštění aplikace s kamerou, která sleduje dopravu pod takovým úhlem, na který původní klasifikátor nebyl natrénován a proto dával velmi špatné výsledky (méně než 50% viz graf 4.10 Video 1).

4.3.1 Implementace

Samotná myšlenka je velmi jednoduchá. Jestliže známe rozměry auta, můžeme je využít ke zpřesnění klasifikace. Tyto rozměry, při správné detekci, budou vždy podobné pro určitý typ vozidla nezávisle na parametrech kamery.

Vypočtené rozměry jsou však nepřesné (a to jak při získání základních rozměrů, tak i následný přepočet do reálných jednotek) a proto jsem se rozhodl využívat tuto informaci nepřímo (tedy netrénovat klasifikátor na `[výška, šířka, délka]` ale např. využít poměru těchto hodnot). Při zkreslení hodnot budou zkresleny všechny přibližně stejně a proto poměr mezi nimi by mel být daleko stabilnějším ukazatelem než hodnoty samotné.



(a)

Obrázek 4.10: Graf úspěšnosti klasifikace pro 4 testovaná videa, vlevo vždy reálné hodnoty, uprostřed klasifikace pomocí HOG a vpravo klasifikace pomocí rozměrů.

4.3.2 Testování deskriptorů

Postupně jsem vyzkoušel několik možností. U jejich popisu budu využívat následující zkratky `w`, `h`, `l` pro šířku, výšku a délku v relativních jednotkách a `wr`, `hr`, `lr` pro šířku, výšku a délku přepočteny v reálných jednotkách.

Jako první jsem zvolil deskriptor s co největším počtem údajů (zatím bez využití reálných parametrů), tedy `[w, h, l, w / l, w / h, l / h, (w*w) / l, (w*w) / h, (l*l) / h, w / (l*l), w / (h*h), l / (h*h), (w*w) / (l*l), (w*w) / (h*h), (l*l) / (h*h), w*h / l, w*l / h, l*h / w, w * h * l]` což, jak se ukázalo hned při prvním testu, výsledky pouze zhoršilo.

Po prvním neúspěchu jsem pochopil, že řešení musí být daleko jednodušší a nesmí záviset na samotných hodnotách, či jejich násobcích. Dalšími pokusy tedy vznikly deskriptory:

1. `[w, h, l, w / l, w / h, l / h]`
2. `[l, w / l, w / h, l / h]`
3. `[sqrt(l), w / l, w / h, l / h]`
4. `[w / l, w / h, l / h]`

Třetí zmíněný již zaznamenal poměrně slušnou úspěšnost 80 – 90% (graf 4.10 v pravých sloupcích). Kromě posledního deskriptoru jsou ale všechny stále závislé na alespoň jedné přímé hodnotě a proto nefungovaly dobře pro všechny kamery. Naopak poslední deskriptor měl problémy s rozeznáním motocyklu a nákladního automobilu (k mému podivu mají oba typy vozidel velmi podobný poměr rozměrů – tedy na to jak jsou široké jsou vysoké a dlouhé). Tento objev znamenal nutnost využití alespoň jedné reálné hodnoty a to nejlépe hodnoty, která se pro jednotlivé typy vozidel liší nejvíce.

Konečným řešením je tedy deskriptor `[lr, w / l, w / h, l / h]`, kde jednotlivé hodnoty pro `w`, `h`, `l` nejsou jen průměrem všech snímků na kterém se vozidlo nachází, ale nejprve jsou odstraněny hodnoty ze začátku a konce sledování (které jsou zkresleny díky pouze částečné viditelnosti na okraji kamery) a také jsou odstraněny hodnoty odlišující se

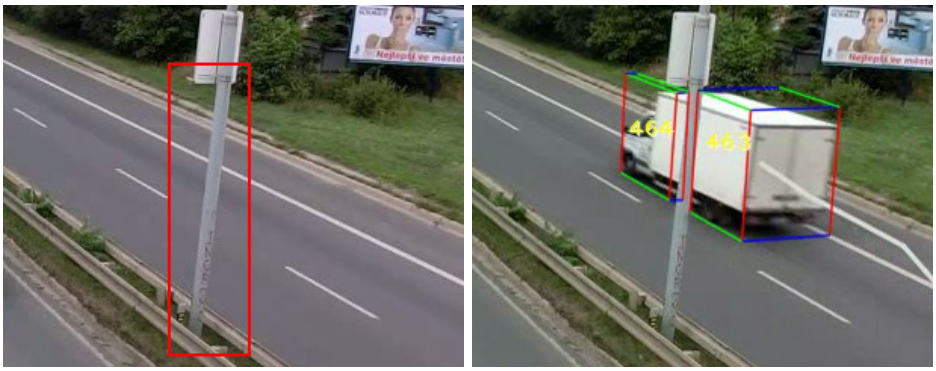
od průměru (tedy hodnoty, které jsou očividně špatné např. z důvodu chvilkového překrytí vozidla jiným vozidlem). Toto řešení dosáhlo na testovacích datech největší úspěšnosti, která se blíží k 95% (podrobněji v kapitole 6 a tabulce 6.2).

Nevýhodou tohoto řešení je pak především nutnost znát reálnou délku vozidla, která se vypočítá pomocí hodnoty λ popsané v sekci 4.2.3. Ta se bohužel počítá v průběhu detekce a proto i tento klasifikátor začne pracovat až ve chvíli, kdy je tato hodnota známá. V mnou testovaných videích (i v reálném streamu) tato fáze trvala dle hustoty provozu 5 – 10 minut.

4.4 Detekce a odstranění překážek

V rámci této diplomové práce jsem dále implementoval dvě rozšíření. První z nich je detekce a odstranění překážek, a druhým pak rozdělení dvou překrývajících se vozidel, které popíši v příští sekci.

Překážkou je zde myšlena taková překážka, která samozřejmě nevadí běžnému provozu, ale zastiňuje část vozovky z pohledu kamery (obr. 4.11a). To mohou být např. sloupy veřejného osvětlení, dopravní značky, či malé reklamní poutače. Jejich detekce a následné odstranění pomůže při sledování zejména velkých vozidel, které jsou touto překážkou detekována jako dvě nezávislá vozidla (obr. 4.11b).



(a) (b)

Obrázek 4.11: (a) překážka stínící provoz, v tomto případě sloup veřejného osvětlení a (b) projíždějící vozidlo rozděleno na dvě části.

4.4.1 Detekce překážek

Pro samotnou detekci překážek se využívá již existujících informací a to výstup z detekce pohybu v obraze a detekované jízdní pruhy.

Samotná detekce pak probíhá v několika krocích, přičemž prvním je sběr výstupu z detekce pohybu (pouze část pro dané vozidlo, obr. 4.6a). Pokud daná detekce byla označena jako vozidlo pak jsou tyto data přidána do integrálního obrazu (obr. 4.12b).

Ve chvíli kdy jsou dostupné údaje o jízdních pruzích, se tyto použijí pro ořezání dat z integrálního obrazu (obr. 4.13a). V tuto chvíli si můžeme všimnout šedé plochy v místech jízdního pruhu, na který kamera vidí, a bílé v místě překážky.

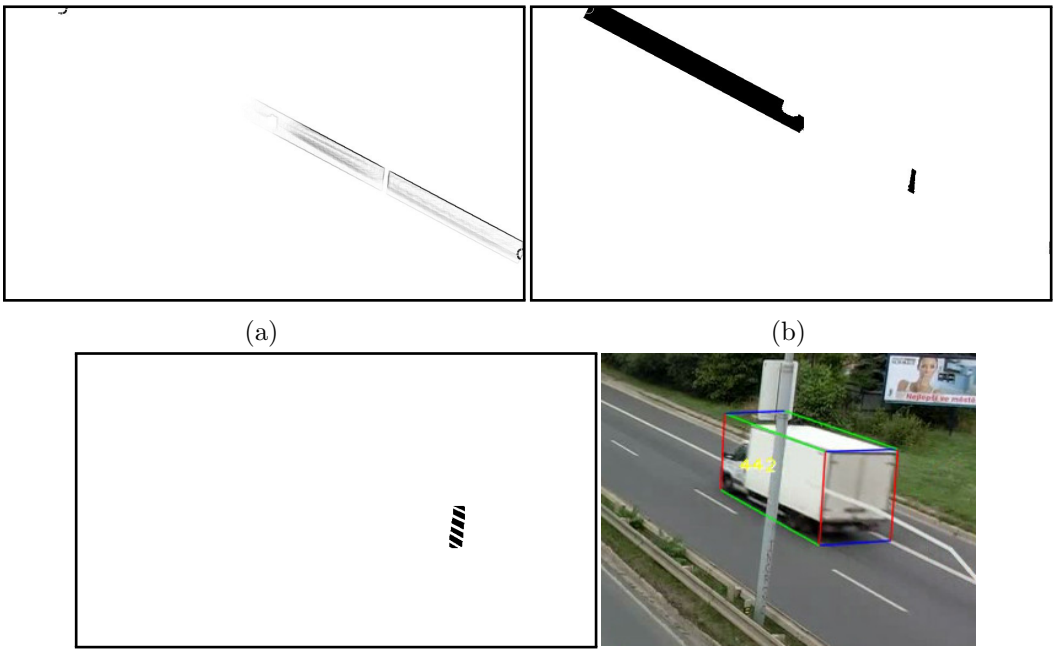
Pro následující využití je tento obraz dále zpracován invertováním barev (překážka je označena černou barvou na bílém pozadí obr. 4.13b), a následným filtrováním a rozsekáním na menší (nepravidelné) tvary ve směru k prvnímu úběžníku (obr. 4.13c). Výsledkem je

(a) (b)

Obrázek 4.12: Výsledný integrální obraz. (a) před filtrováním a (b) po filtrování pouze blobů označených jako vozidlo.

tedy soubor relativně malých ploch rovnoběžných s jízdními pruhy. Těchto vlastností se využívá při následném odstranění překážky.

Na obr. 4.14 je pak znázorněno, proč se překážka dělí na více částí. U jednotlivých vozidel by první případ nebyl tak hrozný, ale je neslučitelný s modulem pro rozdělení dvou překrývajících se vozidel, který popíši v příští sekci 4.5.



(a) (b)

(c) (d)

Obrázek 4.13: Postup detekce překážky. Osekání integrálního obrazu dle jednoho jízdního pruhu (a), invertování barev (b), filtrace a rozdělení na několik částí (c) a výsledný obraz s využitím znalosti překážky (d).

V tuto chvíli se také výsledný obraz ukládá (pokud je aplikace nastavena pro detekci a následné uchování detekovaných překážek). Aplikace je také schopna tento soubor načíst při startu a celý proces detekce tak přeskočit.

(a) (b)

Obrázek 4.14: Ukázka použití detekované překážky bez rozdělení (a) a s rozdělením na části (b).

4.4.2 Odstranění překážek

Odstraněním překážek je pak myšleno odstranění (nebo alespoň zmírnění) negativního efektu, který tato překážka vytváří. Tím je většinou nežádoucí zdvojená detekce jednoho vozidla.

Samotné odstranění překážky pak probíhá ihned po detekci pohybu v původním programu. V tu chvíli se obraz s nově detekovaným pohybem spojí s mapou překážek, kde případné přebývající části překážky jsou odstraněny. Výsledný spojený obraz je na obr. 4.13d.

4.4.3 Výsledky

Vyrušení výše zmíněného negativního efektu je velmi závislé na detekci jízdních pruhů, pokud by tedy nastala chyba v této části, pak ani odstranění překážek nemusí fungovat na 100%. Při testech na testovacích datech tato situace nastala pouze jednou. V ostatních případech spolehlivě spojila překážkou rozdělená vozidla, čímž výrazně zvýšila úspěšnost detekce a následné klasifikace dodávek a nákladních automobilů.

4.5 Rozdělení překrývajících se vozidel

Překrývání jedoucích vozidel je jedním z největších problémů detekce vozidel na základě detekce pohybu v obraze. Nezřídka se tak stává, že dvě (nebo dokonce více) vozidel jedou blízko za sebou nebo vedle sebe a z pohledu kamery je tedy téměř nemožné je rozlišit a jsou tudíž detekována jako jedno vozidlo. Tento efekt je velice nepříjemný z hlediska následné klasifikace protože většinu zaznamenaného času jsme detekovali pouze jedno neklasifikovatelné vozidlo a po jejich případném pozdějším rozdělení (např. z důvodu zvětšení mezery mezi nimi) již nemáme dostatek údajů pro správnou klasifikaci.

Já jsem se rozhodl vyřešit pouze menší část tohoto problému, a to rozdělení dvou překrývajících se vozidel, jedoucích vedle sebe. Pro toto rozdělení je využito zejména barevného obrazu, výsledku detekce pohybu detekovaného dvojvozidla a detekovaným jízdním pruhům, přesněji detekovaným půlicím čarám jízdních pruhů.

Celý subsystém je rozdělen do 3 částí, a je využit při sledování detekovaných blobů popsaných v sekci 4.2.2. První částí je rozlišení, zda se jedná o jedno vozidlo, nebo o více

vozidel[3]. Druhou je rozdělení vozidel na dvě samostatná vozida a třetí pak vyčištění a vyfiltrování.

4.5.1 Detekce překrývajících se vozidel

První částí je tedy detekce takto se překrývajících vozidel. Ta je spuštěna na každý nalezený blob v aktuálním snímku a to těsně před tím, než jsou tyto bloby spojeny s již existujícími detekovanými auty. Tato část (a tím i celý subsystém) je možné použít pouze ve chvíli kdy již známe pozici půlících čar, tedy začíná pracovat až nějakou dobu po zapnutí aplikace.

Jako první se využije vstupní maska detekovaného blobu a do ní se vyznačí půlící čáry (obr. 4.15b), čímž (v případě dvou vozidel vedle sebe) dostaneme detekovaný blob rozdělený na dvě části. Následně nalezneme kontury těchto dvou (či více) nových blobů, které postupně prochází dvěma testy.

1. Spočítá se poměr velikosti původního blobu a nových blobů $\frac{subArea}{area}$, kde $area$ je plocha původního blobu v pixelech a $subArea$ je plocha nového (právě testovaného) blobu. Pokud existují alespoň dva bloby s poměrem nejméně 0.4 pak je původní blob označen jako dvě auta.

2. Vypočítá se 3D obalový kvádr a získají se souřadnice jeho dvou spodních rohů v reálném světě. Stejně tak se vypočítají souřadnice v reálném světě pro průsečík těchto dvou bodů a každé půlící čáry (obr. 4.15c). Pokud poslední jmenovaný průsečík leží mezi oběma zbývajícími body a je od nich vzdálen určitou minimální vzdálenost, je původní blob označen jako dvě auta.

Pokud byl blob označen jako dvě auta, pak subsystém pokračuje jejich rozdělením. Pokud nebyl označen, pak celý subsystém končí a začne zpracovávat další blob v pořadí.

4.5.2 Rozdělení vozidel

První část subsystému rozhodla, že se pravděpodobně jedná o dvě či více vozidel. Nyní se je druhá část pokusí rozdělit. První na řadě je využití detekce hran z původního barevného snímku (obr. 4.16c). Ten se promítne do snímku s maskou blobu (obr. 4.16d).

Tímto jsme získali obraz vozidel rozdělený na spoustu malých částí. Pro každou takovouto část je vypočítán střed[4] (obr. 4.16e) a nakonec nad těmito částmi spustíme algoritmus K-means s $k = 2$ a automaticky určenými středy dle hmoty. Tento algoritmus rozdělí všechny části do dvou shluků (obr. 4.16f), které jsou spojeny do jednoho celku morfologickými operacemi[5] (obr. 4.17b).

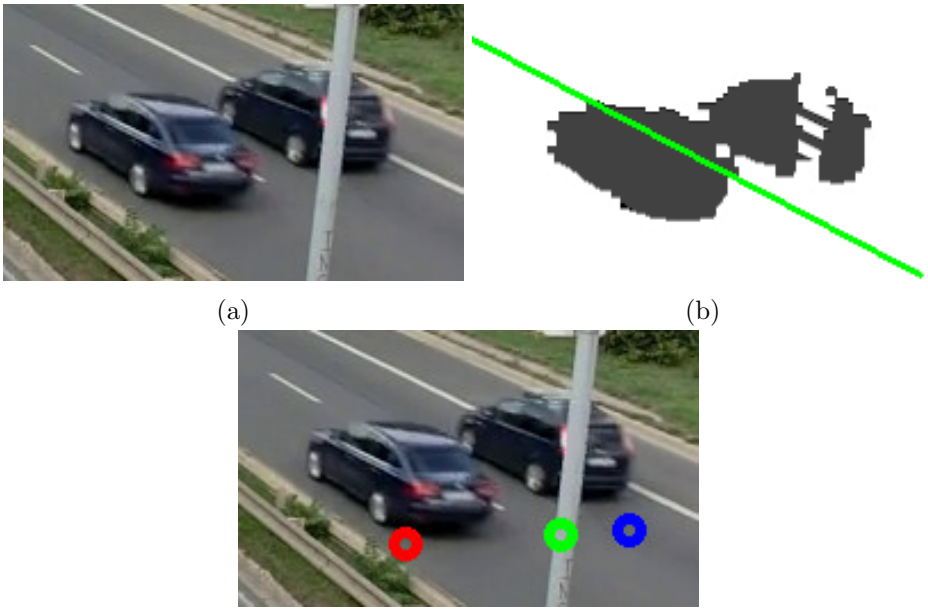
4.5.3 Filtrování a využití rozdělených blobů

Z předchozího kroku tedy máme dva bloby, které by měly odpovídat dvěma vozidlům. Jelikož jsme si nebyli jistí zda jde o dvě vozidla již v prvním kroku, nemůžeme si být jisti ani teď. Proto přichází na řadu několik po sobě jdoucích testů, které by tuto pravděpodobnost měly razantně zvýšit.

[3] Přesněji by se dalo říct, že jde o jakousi pravděpodobnost že se jedná o dvě vozidla.

[4] Kvůli rychlosti a jednoduchosti se nepočítá opravdové těžiště. To by možná trochu vylepšilo výsledek ale díky následným filtrům je střed dostačující.

[5] Díky tvaru aut se jako nejlepší jeví elipsovitý/kruhový tvar jádra.



(a) (b)

(c)

Obrázek 4.15: Ukázka testů pro rozlišení, zda se jedná o dvě vozidla či nikoliv. (a) původní snímek, (b) nalezený blob s vyznačenou půlící čarou, (c) původní snímek s vyznačenými rohy blobu (červeně a modře) a jejich průsečík s půlící čarou (zeleně).

Jako první je na řadě test zda, jde o dva bloby. Z předchozího kroku jich může vzejít více i méně. K-means sice rozdělí všechny části do dvou shluků, ovšem to neznamená, že po spojení částí vzniknou právě dvě. Např. osamocená část bude přiřazena k jednomu ze shluků, po spojení částí ale zůstane stále mimo ostatní.

Další z testů je již více sofistikovaný. Naleznou se středy obou blobů a vypočte se úhel mezi přímkou vytvořenou těmito středy a půlící čarou. Tento úhel by se měl co nejvíce blížit 90°[6] a zároveň by se půlící čára měla nacházet mezi těmito dvěma středy. Porovnejte obrázky 4.16f a 4.17b, že v situaci s jedním autem jsou oba středy napravo od půlící čáry a jsou téměř rovnoběžné s půlící čarou, zatímco v situaci s dvěma auty jsou na obou stranách a navíc svírají skoro pravý úhel.

Následuje test velikosti obou blobů a příprava dat na finální výsledek. Ještě ale proběhne jeden poslední test. Kolem obou blobů vytvoříme 2D obalový obdélník. Pokud se tyto obdélníky překrývají z více než 60%, pak se jedná o jedno vozidlo. Tento test lze opět ilustrovat na nám již známé dvojici obrázků 4.16f a 4.17b a dvojici 4.18a a 4.18b s vyznačenými obalovými obdélníky.

4.5.4 Výsledky

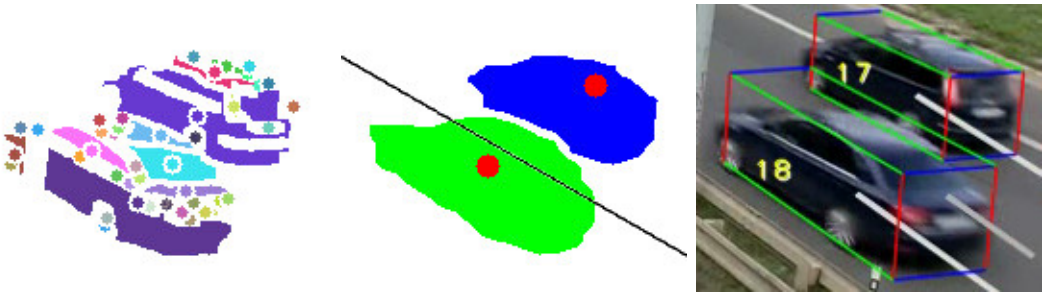
Samotné rozdělení dvou překrývajících se aut funguje vcelku spolehlivě (asi 70 – 80% případů je rozděleno tak, že vzniknou dvě vozidla přibližně správných rozměrů). Největší problém ale způsobuje detekce těchto případů, kde přibližně 50% jich není zachyceno. Toto bohužel není způsobeno pouze výše zmíněným algoritmem, ale souhrou okolností. Zde se např. často stává, že jedno (nebo obě) vozidla nemají zcela přesnou detekci (třeba o půl auta, obr. 4.19), čímž dojde ke špatnému rozhodnutí.

[6] Ve většině případů bude tento úhel spíše kolem 45°. Málokdy totiž obě vozidla jedou přesně vedle sebe.

(a) (b) (c)

(d) (e) (f)

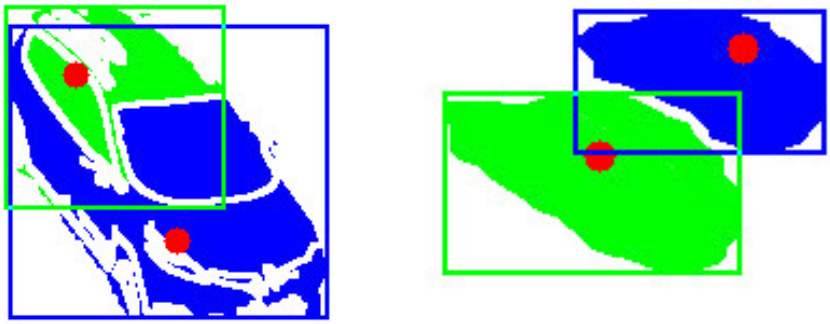
Obrázek 4.16: Postup rozdělení auta na části a následné spojení do dvou shluků, znázorněno na jednom vozidle. (a) původní barevný snímek, (b) původní blob, (c) detekce hran, (d) invertování detekce hran a spojení s maskou, (e) rozdělení na jednotlivé části (označené barevně) s jejich středy a (f) obarvení dle výsledných shluků (černá čára označuje půlící čáru).



(a) (b) (c)

Obrázek 4.17: Postup rozdělení dvou vozidel. (a) rozdělení na jednotlivé části (označené barevně), (b) spojení částí dle shluků do dvou nových blobů (černá čára označuje půlící čáru) a (c) výsledný obraz s dvěma vozidly.

Celkový přínos tohoto modulu k přesnosti celého systému je daleko menší než detekce a odstranění překážek. To je hlavně způsobeno malým výskytem těchto situací v mých testovacích videích (podrobnosti jsou v sekci 6.2), zatímco za překážkou projede každé vozidlo jedoucí daným směrem.



(a) (b)

Obrázek 4.18: Rozdělené bloby a jejich obalové obdélníky. (a) jedno vozidlo (překrytí menšího obdélníku je přibližně 90%), (b) dvě vozidla (překrytí menšího obdélníku je přibližně 25%).

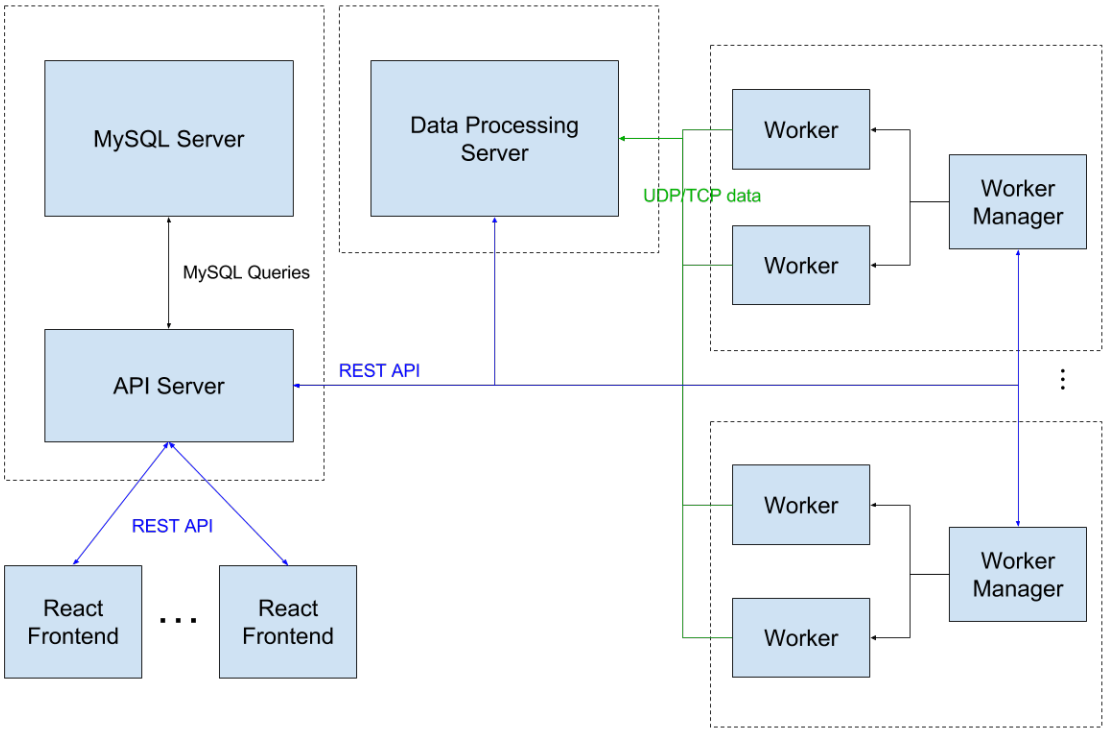
(a) (b) (c)

Obrázek 4.19: Ukázka nepřesných detekcí vozidel a tudíž i nerozdělení těchto dvou vozidel. Všechny tři obrázky jsou z jednoho průjezdu, na kterých lze vidět různě špatné detekce šedého vozidla.

Kapitola 5

Cloudové řešení

Cloudové řešení je rozděleno na několik samostatných části z důvodů jednoduššího návrhu i provedení a také kvůli možnému rozšíření pracovních serverů. Celý návrh architektury je na obr. 5.1, přičemž v následujících sekcích popíši funkcionalitu každé z nich.



Obrázek 5.1: Schéma modulů a propojení modulů cloudového řešení. Čárkovaně jsou obaleny moduly běžící na stejném fyzickém serveru s tím, že Data Processing Server může běžet prakticky kdekoliv ale pouze v jediné instanci.

5.1 Worker Process

Jedná se o základní aplikaci jak byla popsána v předchozí kapitole. Z důvodu spolupráce s dalšími servery je však rozšířena o UDP a TCP komunikaci (možno nastavit v konfi-

guračním souboru). Tato komunikace se využívá k odesílání jednotlivých záznamů o každém autě zvlášť. Veškerá komunikace je tedy jednostranná. Tato aplikace běží (může běžet) na každém stroji (clusteru) vícekrát.

Samotná komunikace probíhá tak, že Worker proces při spuštění vyšle paket se svým označením (jako jeho označení je využito url adresy videa, které by, v ideálním případě, mělo být zpracováváno pouze jedním Worker procesem v danou chvíli). Následující pakety již obsahují samotný záznam o projíždějícím vozidle, tedy přidělené ID, odhadovaný typ vozidla, jeho velikost a rychlost.

Další změnou bylo přidání modulu pro RTSP/RTMP protokoly pro online přenos obrazu. Tento modul čte stream dat v druhém vlákně, čímž nedochází k vypadávání snímků ke kterému by jinak docházelo z důvodu pomalejšího zpracovávání jednoho snímku (většinou se čte každý druhý snímek při 25fps kamery).

5.2 Worker Manager

Tato aplikace běží vždy pouze jednou pro každý dostupný cluster. Jejím hlavním cílem je spravovat Worker procesy, tedy kontrolovat jejich běh, spouštět či zastavit je. Kromě této funkce, Worker Manager komunikuje pouze s API serverem, který rozhoduje která videa by měla být zpracovávána.

Při spuštění vyšle Worker Manager HTTP požadavek na API server s uvítací zprávou která obsahuje dva údaje:

- název clusteru (je určen uživatelem při spuštění procesu) ,
- maximální zatížení daného clusteru (automaticky zjištěno dle počtu procesorů – zde předpokládám, že každý procesor je dostatečně silný na zvládnutí plynulého běhu jednoho Worker procesu).

Na tuto zprávu server odpoví seznamem Worker procesů, které jsou přiřazeny tomuto clusteru (tento seznam bude v tuto chvíli s velkou pravděpodobností prázdný, naplněn by byl pouze ve chvíli, kdy počítač neočekávaně ukončí činnost a obnoví se dříve, než server přerozdělí jeho práci na jiné, běžící clustery[1]). Každý záznam obsahuje url adresu kamery a hodnotu, zda má daný Worker proces běžet, či nikoliv.

Worker Manager proces následně zasílá aktualizační zprávy se seznamem právě běžících Worker procesů a aktuální zátěž clusteru. Server odpoví se stejným seznamem jako v předchozím případě.

Po obdržení tohoto seznamu je pro každý záznam zvlášť vyhodnocena akce:

- nedělej nic,
- spusť nový Worker proces,
- zastav běžící Worker proces,
- obnov Worker proces, který neočekávaně ukončil činnost.

Pokud byla vykonána jakákoliv z vyjmenovaných akcí, pak je ihned zaslána aktualizační zpráva, jinak je vlákno uspáno na 20 sekund[2].

[1] V případě, že mají volnou kapacitu.
[2] Lze změnit parametrem při spuštění procesu.

5.3 Data Processing Server

Tato aplikace má za cíl shromažďovat data ze všech Worker procesů a ukládat je do databáze. Běží proto vždy pouze jedna instance pro všechny clustery.

Celkově je aplikace rozdělena na 2 části (vlákna). V první běží TCP a UDP asynchronní server naslouchající na stejném portu pro oba protokoly (ten lze specifikovat při startu aplikace). A druhé vlákno, které každou minutu sečte všechna přijaté data za předposlední minutu[3] pro každý Worker proces zvlášť (vypočítá počet vozidel každého typu a v případě rychlosti vozidla vypočítá průměrnou rychlost). Vypočítané hodnoty následně pošle pomocí HTTP požadavku na API server, který jej uloží do databáze.

5.4 API Server

PHP script obsahující REST API pro aplikace Worker Manager i Data Processing Server a zprostředkovává tak komunikaci mezi těmito aplikacemi a databází. Dále poskytuje REST API pro webovou aplikaci. Seznam poskytovaných volání je v tabulce 5.1.

5.5 MySQL Databáze

K ukládání dat byla zvolena MySQL databáze obsahující 4 tabulky znázorněné na obr. 5.2.

- Tabulka **data** – uchovává záznamy zpracované Data Processing serverem. Obsahuje tedy jeden záznam pro každou minutu, kdy byl spuštěn daný Worker proces. Většina položek tohoto záznamu je inkrementální, tedy obsahuje sumu od začátku zpracovávání dané kamery. Tento způsob byl zvolen, protože je zde předpoklad, že jednu kameru můžeme zpracovávat i několik měsíců, což vyústí v tisíce záznamů a je snazší a hlavně výpočetně méně náročné odečíst dva záznamy a získat tak údaje pro dané období, než všechny záznamy spadající do daného období sčítat.
- Tabulka **processes** – obsahuje záznam o každém Worker procesu.
- Tabulka **servers** – obsahuje záznam o každém Worker Manager procesu.
- Tabulka **serverloads** – obsahuje záznamy o každém přihlášení Worker Manager procesu a jeho zátěži v danou chvíli.

5.6 Webová aplikace

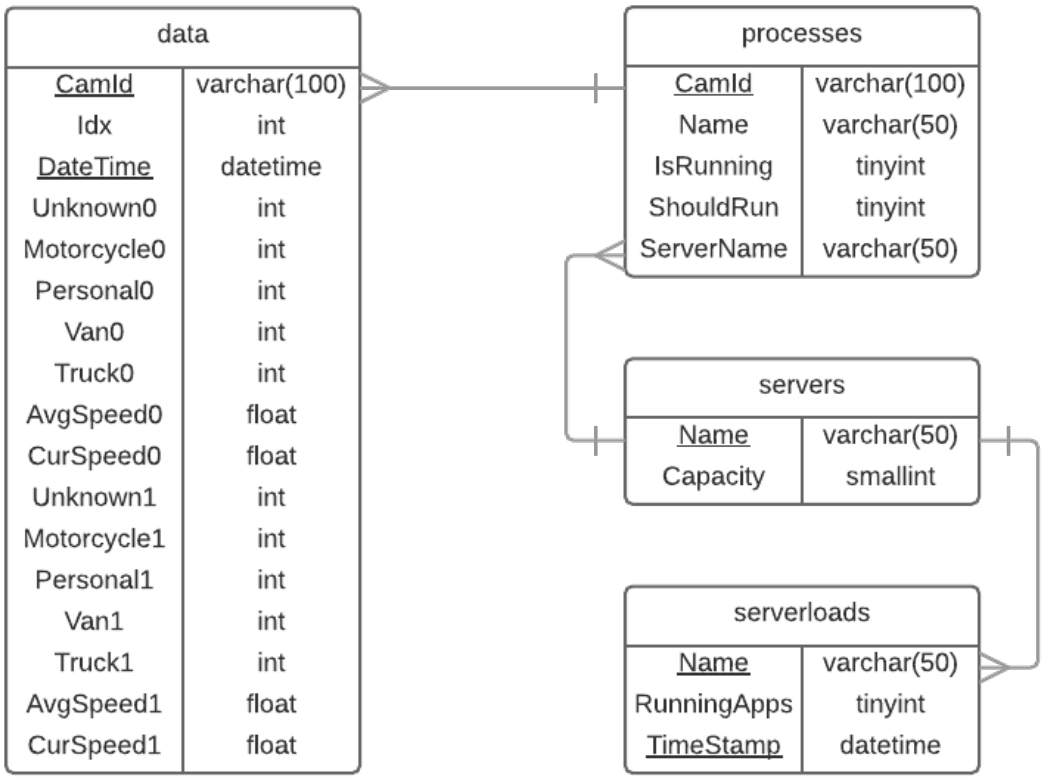
Single page aplikace využívá framework Facebook React a jQuery. Komunikuje s API serverem pomocí asynchronních dotazů a následného zpracování surových dat ve formátu JSON. Toto je jediná část se kterou má uživatel možnost se setkat, a ovládat s ní celý systém.

Aplikace přehledně zobrazuje seznam všech procesů a clusterů (s označením zda běží či ne), přičemž tyto seznamy jsou automaticky aktualizovány každých 30 sekund[4] (67 sekund[5]

[3] Např. Pokud je právě 15:48:05 pak jsou zpracovávány minuty do 15:46. Tím zajistíme, že již přišly všechny parkety, a každá minuta je tak zpracována právě a pouze jednou.

[4] 30 sekund je vybráno proto, že v základu se Worker Manager aktualizuje každých 20s. Pokud tedy spustíme Worker proces, nejpozději za 20s by tento požadavek měl být zpracován a za 30s by jsme již měli mít informaci o úspěchu/neúspěchu a můžeme jej zobrazit při další aktualizaci.

[5] 67 sekund je zvoleno tak aby se požadavky se seznamem procesů nesynchronizovaly po nejdelší možnou dobu. Navíc se předpokládá, že stav clusterů není potřeba aktualizovat tak často.



Obrázek 5.2: ER diagram

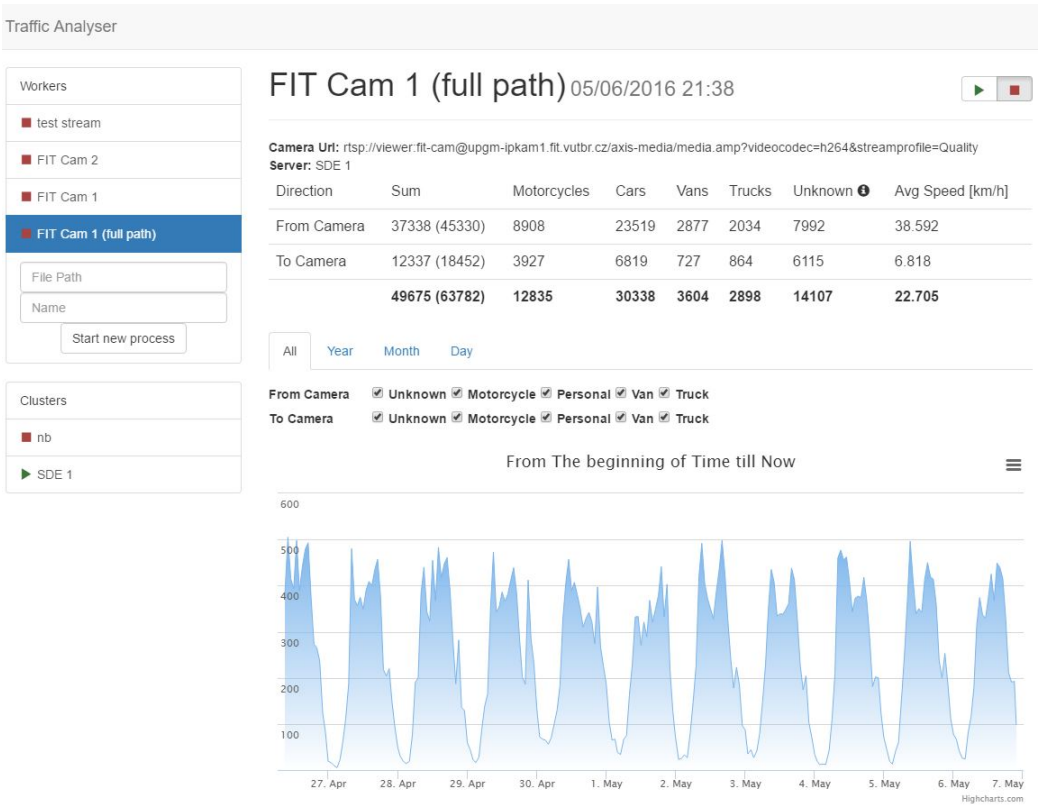
v případě clusterů).

Po výběru procesu se zobrazí další údaje, jako url zpracovávané kamery, na kterém clusteru běží, a nejnovější celkové hodnoty. Zároveň je k dispozici několik grafů zobrazujících průběh. Své detailnější zobrazení má také cluster, kde se uživatel dozví aktuální zatížení a má k dispozici graf historie zatížení.

Aplikace pak využívá dvou základních typů grafů. Prvním z nich je celkový graf (zobrazen na obr. 5.3), ve kterém je možné myší vybrat daný úsek a tím jej přiblížit. Tento graf funguje s dynamickou agregací, tedy v základu agreguje data po dnech, ale při výběru 30 nebo méně dnů najednou již agreguje po hodinách. Při výběru pod 5 dnů agreguje po 10 minutách a při výběru pod 12 hodin po 1 minutě. Druhým typem grafů je graf omezen na určité období (rok, měsíc, den, hodina) s rozdělením dle typů vozidel (obr. 5.5). Zde po kliknutí na jednotlivé hodnoty v grafu se přesuneme na další graf v pořadí s detailnějším rozložením (tedy při kliknutí na 28. dubna v měsíčním pohledu, se přesuneme na stejný den v denním pohledu apod.). U obou grafů je pak možnost filtrovat dle směru vozidel a/nebo jednotlivých typů vozidel. Celá výsledná aplikace je zobrazena na obr. 5.3 a 5.4.

5.7 Jak to vše funguje

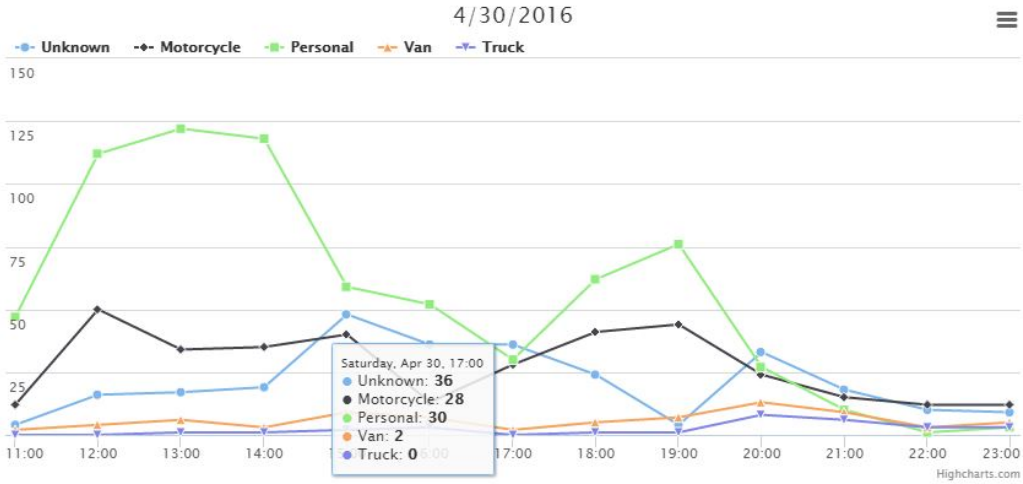
Nyní, když všechny části byly popsány, již můžu popsat spolupráci všech součástí. Celý postup je také znázorněn na obr. 5.1.



Obrázek 5.3: Ukázka webové aplikace při detailu procesu, v horní části jsou celkové informace a ve spodní se nachází část s grafy.

Obrázek 5.4: Ukázka webové aplikace při detailu clusteru s historií počtu spuštěných Worker procesů.

Předpokládejme, že je aktivních několik clusterů a procesů. Pokud se uživatel rozhodne některý z procesů zastavit (či zapnout), je tento požadavek směrován na API server, který v případě požadavku k zastavení nastaví příslušnou informaci v DB. V případě spuštění



Obrázek 5.5: Ukázka spojnicového grafu aplikace.

nejprve nalezne nejvhodnější cluster a opět nastaví příslušnou informaci v DB. Uživateli se zobrazí zpráva o čekání na vyřízení jeho požadavku.

Při další aktualizaci vybraného Worker Manager procesu se dozví o změnách, vykoná je a následně informuje API server zda požadované procesy běží. Ten je při příští aktualizaci webové aplikace zobrazí uživateli. V ideálním případě a s výchozím nastavením by celý tento proces neměl trvat déle než půl minuty.

V případě zapnutí Worker procesu (ať už nového či starého), se tento po zapnutí přihlásí Data Processing serveru se zprávou o zpracovávané kameře. Ta si uloží záznam o IP adrese a portu ze kterého mu uvítací zpráva přišla. Tak dokáže při dalších UDP zprávách rozpoznat, ke kterému procesu patří. Jelikož daný vstup může být zpracováván pouze jednou, při příští uvítací zprávě obsahující stejné ID kamery je záznam o IP adrese a portu přepsán. Jakékoliv pakety ze starého procesu dále nejsou využívány a proto nedojde ke zdvojení informací. Tohoto může být využito při předávání procesu z jednoho clusteru na druhý pro rozprostření zátěže mezi všechny běžící stroje.

Jelikož API server pouze odpovídá na dotazy, ale sám se nespouští, je provedena kontrola na běžící clustery při každé aktualizační zprávě. V tu chvíli se zjistí jestli existuje cluster, který se nepřihlásil déle než 5 minut. Pokud takový existuje, je označen jako neběžící a všechny jemu přiřazené procesy (které byly aktivní) jsou, pokud možno, přerozděleny ostatním clusterům, čímž dojde k minimálnímu výpadku (z pohledu dnů až měsíců je výpadek na několik minut zanedbatelný). V případě, že není volný žádný cluster je proces označen jako neběžící a tato informace je zobrazena uživateli ve webové aplikaci.

Název a Typ	**Parametry**	**Funkce (F), Odpověď (R)**
CreateProcess POST	command name	**F:** Spustí Worker proces s url `command` a jménem `name` **R:** jméno clusteru, na kterém se spustil Worker proces.
StopProcess GET	CamId	**F:** Zastaví Worker proces dle `CamId` kamery.
SchedulerSignIn POST	name capacity	**F:** Uvítací zpráva pro Worker Manager proces. **R:** seznam Worker procesů, které by měl spustit.
SchedulerUpdate POST	name <*data*>	**F:** Aktualizační zpráva Worker Manager procesu. <*data*> obsahují informace o právě spuštěných Worker procesech. **R:** seznam Worker procesů, které by měl spustit / vypnout.
ServerSignIn GET		**F:** Přihlášení Data Processing Serveru. **R:** aktuální stav hodnot ze všech Worker procesů.
ServerUpdate POST	CamId <*data*>	**F:** Vložení zpracovaných dat za jednu minutu.
ListProcesses GET		**R:** seznam všech Worker procesů a jejich stav.
ListServers GET		**R:** seznam všech Worker Manager procesů a jejich stav.
Info GET	CamId	**R:** podrobné informace o stavu a data Worker procesu dle `CamId`.
GraphDataYear GET	CamId dir year	**R:** data potřebné pro zobrazení grafu Worker procesu pro daný rok.
GraphData-Month GET	CamId dir year month	**R:** data potřebné pro zobrazení grafu Worker procesu pro daný měsíc.
GraphDataDay GET	CamId dir year month day	**R:** data potřebné pro zobrazení grafu Worker procesu pro daný den.
GraphDataHour GET	CamId dir year month day hour	**R:** data potřebné pro zobrazení grafu Worker procesu pro danou hodinu.
GraphDataAll GET	CamId dir min, max	**R:** data potřebné pro zobrazení celkového grafu Worker procesu s ohraničením na datum a čas *min* a *max*.
GraphDataAll-Servers GET	CamId	**R:** data potřebné pro zobrazení grafu Worker Manager procesu.

Tabulka 5.1: Popis poskytovaného REST API.

Kapitola 6

Implementace, testování a výsledky

6.1 Implementace

K implementaci základního programu (Worker Process) a dvou serverových (Worker Manager, Data Processing Server) bylo využito jazyka C++11 na dvou operačních systémech a s využitím knihoven OpenCV a Boost. Zde byl ručně přidán neoficiální Boost modul *process*[1], který byl dále upraven přidáním funkce pro zjištění, zda daný proces stále běží či nikoliv, pro oba operační systémy.

- Windows 10 – sestava pro vývoj
  - OpenCV (2.4.11)
  - Boost (1.59)
  - Visual Studio 2013
- Debian 8.1.0 – sestava pro testování (virtuální server)
  - OpenCV (2.4.9)
  - Boost (1.55)

Dále bylo využito PHP 5.5 pro API server, a JavaScript s knihovnami jQuery (2.1.1), Facebook React (0.13.3), Bootstrap (3.3.6) a HighCharts (využito pomocí CDN, v aktuální verzí 4.2.4) pro webovou aplikaci.

Celý systém (každá jeho část) je multiplatformní (přinejmenším běží na testovaných systémech s Windows a Linux) a díky síťové komunikaci může být každá část spuštěna nejen na jiném fyzickém stroji ale i jiném operačním systému.

6.1.1 Pomocné aplikace

Pro potřeby testování byly vytvořeny dvě malé webové aplikace vytvořené v JavaScriptu. První z nich je aplikace pro anotaci videa. Tato aplikace umožňuje načíst video, přehrávat jej, příp. zastavovat nebo posouvat čas. Zároveň aplikace umožňuje anotovat jaký typ vozidla jede v jakém pruhu pomocí kláves q - p (pro výběr typu vozidla) a s - l (pro výběr

[1] Ke stažení z http://www.highscore.de/boost/process.zip, celý modul i s mými změnami je také na přiloženém CD

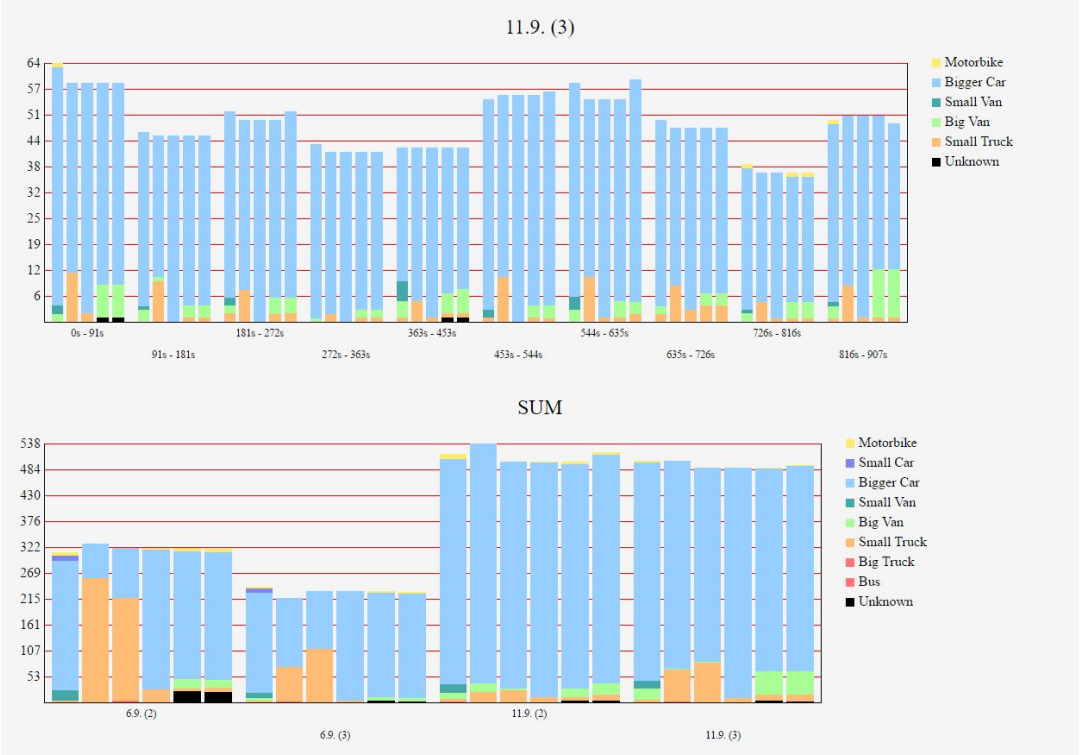
jízdního pruhu). Aplikace sama zaznamená čas ve kterém byly zmáčknuty klávesy. Jednotlivé záznamy jsou ihned zobrazeny, přičemž uživatel je může smazat, čímž se vrátí do daného času ve videu a tak může opravit předchozí, špatně anotované vozidlo. Výstupem pak je JSON struktura využívaná v druhé pomocné aplikaci. Náhled této aplikace je na obr. 6.1.

Obrázek 6.1: Aplikace pro anotaci videa. V levé horní polovině je přehrávané video, v levé dolní polovině pak ovládací prvky s přiřazenými klávesami a v pravé části pak seznam posledních anotovaných vozidel.

Druhou pomocnou aplikací je pak zobrazení grafů s možností srovnání anotovaných (tedy přesných dat) s nově získanými. Jedná se o soustavu grafů, které mají schopnost zobrazit data ve formě skládaného sloupcového grafu, a zároveň ve standardní formě sloupcových grafů (tedy více sloupců vedle sebe pro jednu hodnotu X souřadnice). Této vlastnosti bylo využito při zobrazení všech hodnot z více měření do stejného grafu, kde pro jeden časový úsek lze zobrazit všechny měření vedle sebe, a zároveň údaje četnosti jednotlivých typů vozidel nad sebou. Jako vstup pak slouží již zmíněné JSON struktury z anotace a stejné struktury z aplikace zpracovávající dané video. Náhled této aplikace je pak na obr. 6.2.

6.2 Testování

Pro účely testování bylo pořízeno 6 videí o délce 10 – 20 minut. Všechny byly pořízeny z nadchodu přes rychlostní silnici E461 mezi zastávkou Červinkova a fakultami FSI/FEKT VUT. Jednotlivá videa byla pořízena ve dvou dnech a to 6.9.2015 (Neděle) a 11.9.2015 (Středa) vždy po třech kusech označených (1), (2), (3) podle pořadí ve kterém byly natočeny.



Obrázek 6.2: Dva grafy z aplikace pro zobrazení grafů. Horní graf rozepisuje hodnoty pro jednotlivé časové úseky jednoho videa (každé video má jeden takovýto graf) a dolní graf zobrazuje celkový počet vozidel pro všechny videa.

Každé video zabírá silnici z jiné strany či úhlu (snímky z jednotlivých videí jsou na obr. 6.3 s jejich popisem v tabulce 6.1).

Další testovací videa v délkách pár minut až hodinu, která byla pořízena z ulice Sportovní v Brně a dálnice u Prahy, mi poskytl vedoucí práce. Tato videa jsem využil pouze pro ověření, že systém funguje jak má na neznámých záběrech (SVM klasifikátor byl trénován na mnou natočených záběrech a proto tyto představovaly další úroveň ověření systému). Jelikož tyto záběry nebyly anotovány (a vlastně byly využity pouze jednorázově), nejsou ani zařazeny ve zmíněné tabulce.

6.2.1 Testování celého cloudového systému

Pro testování cloudového systému jsem využil jeden virtuální server, na kterém současně běží Worker Manager i Data Processing Server. Webová část (tedy MySQL databáze, API Server i Webová aplikace) se nachází na školním serveru[2]. Toto rozdělení (a vlastně i návrh architektury) vychází z faktu, že tento virtuální stroj je schován za firemní VPN a tudíž není zvenčí dostupný.

Pro testování známého záběru jsem pak využil mnou natočených dat, které jsem pomocí programu VLC media player streamoval protokolem RTSP z osobního notebooku (připojeného přes firemní VPN). Stream dat tedy proudil přes veřejnou síť a dokonale tak simuloval reálný provoz. Z tohoto testu také vyšel jeden zajímavý fakt a to, že při ztrátě nebo poškození části dat se může zhoršit kvalita části obrazu, čímž dojde k nesprávnému

[2] http://www.stud.fit.vutbr.cz/~xvalch01/DP/

Název	Délka	Popis
6.9. (1)	14:57	Záběr je pořízen uprostřed vozovky, vozidla by se tak měla překrývat jen minimálně. Bohužel toto video nebylo zařazeno do testování z důvodu poškození a nemožnosti přehrání pomocí OpenCV. Obr. 6.3a.
6.9. (2)	14:55	Zaběr je pořízen z nižší výšky a pod velkým úhlem. Cílem tohoto záběru je ověřit možnosti celého systému. Bohužel zde nastává problém s 3. jízdním pruhem (počítáno směrem od kamery), který je z větší části zakryt svodidly. Obr. 6.3b.
6.9. (3)	10:02	Záběr je pořízen tak aby byl levý jízdní pruh co nejrovněji s osou kamery (čímž se mírně srovná efekt zatáčky v pozadí). V pravých jízdních pruzích dochází k překrytí sloupem, či jinými vozidly. Obr. 6.3c.
11.9. (1)	19:57	Podobný záběr jako u videa 6.9. (3). Tento záběr je však delší a natočen mírně vpravo od původního místa (levé pruhy jsou tak rovnější než v předchozím případě). Obr. 6.3d.
11.9. (2)	14:59	Záběr je pořízen nad nejpravějším jízdním pruhem. Zde vznikají problémy se stromem zasahujícím do záběru (při větru vznikají falešné detekce). Obr. 6.3e.
11.9. (3)	15:08	Záběr z protější strany (oproti předchozímu), tedy nad nejlevějším jízdním pruhem. Obr. 6.3f.

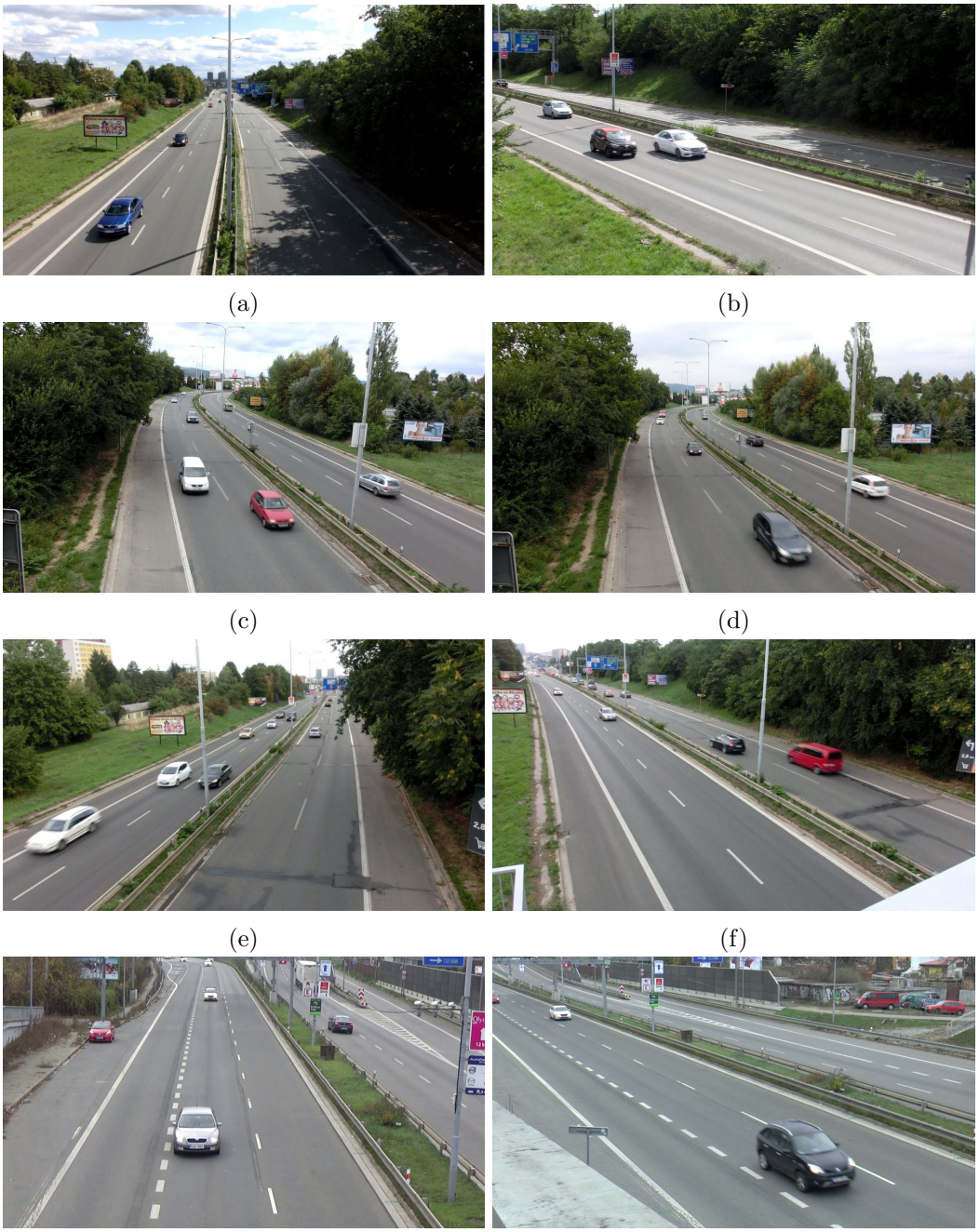
Tabulka 6.1: Podrobnosti jednotlivých záznamů.

určení rozměrů vozidel a tím pádem i k horší/špatné klasifikaci. V mnou testovaném případě tak došlo k znatelnému rozdílu výsledků mezi zpracováním na notebooku a na vzdáleném serveru. Oba procesy přitom zpracovávali současně stejný RTSP stream (pro notebook data nešla přes veřejnou ani místní síť).

Po ověření funkčnosti celku na známých videích jsem přistoupil k online video streamu. K dispozici byly dvě kamery na naší fakultě (snímky z těchto kamer jsou na obr. 6.4). Tyto byly zpracovávány nepřetržitě i několik dní. Případné přerušení bylo nutné kvůli aktualizaci aplikací. V rámci tohoto testu byla totiž objevena spousta chyb, které se neprojeví při 20 minutovém videu.

Jedním z problémů, jehož řešení je mimo rámec této práce, jsou noční záběry (viz obr. 6.5a), na které kamera není stavěná. Projíždějící vozidla svými světly osvítí velkou část vozovky, čímž naruší výpočet velikosti a tím i klasifikaci. Problém je o to horší, že aplikace se adaptivně přizpůsobuje a mění odhad reálných rozměrů. Ráno tedy byly tyto hodnoty porušené a proto aplikace určovala osobní automobily jako motocykly atd. (viz graf na obr. 6.5c). Z tohoto důvodu bylo nastavení pro další testy ručně uzamčeno, čímž bohužel mohou vznikat nepřesnosti v různých částech dne (např. kvůli směru stínu, který zvětší rozměry vozidla v jedné ose, obr. 6.5b a graf 6.5d).

Možným vylepšením by zde mohlo být zlepšení logování událostí a jejich zpřístupnění přes webovou aplikaci, a také možnost změnit konfigurační soubor pomocí webové aplikace pro každý Worker proces zvlášť.



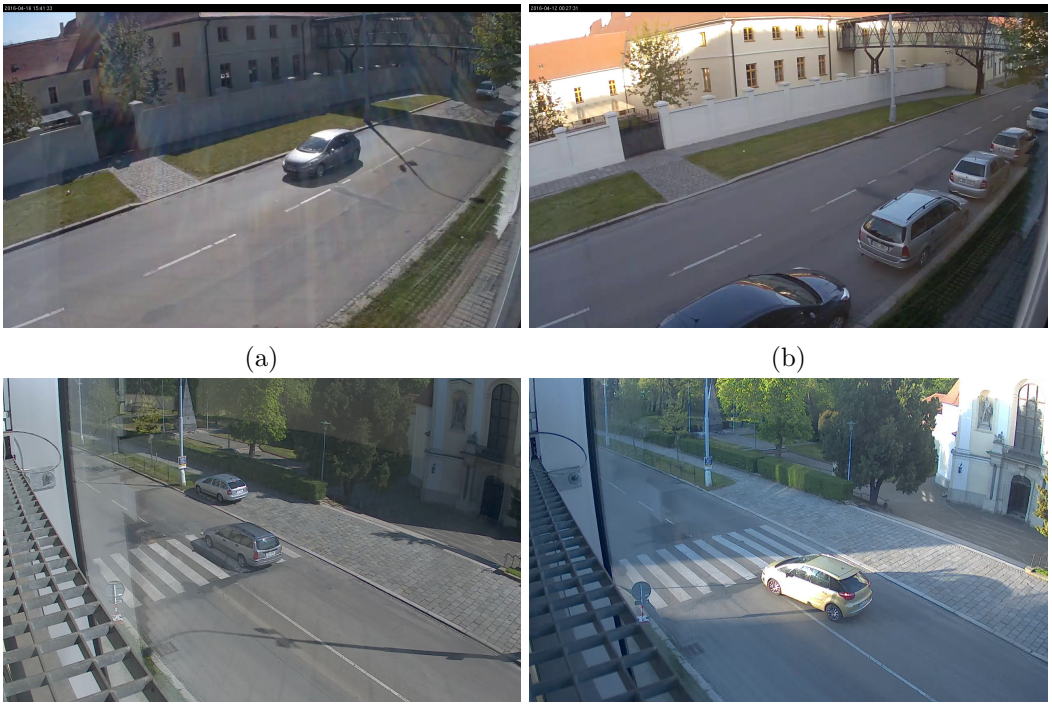
(a) (b)

(c) (d)

(e) (f)

(g) (h)

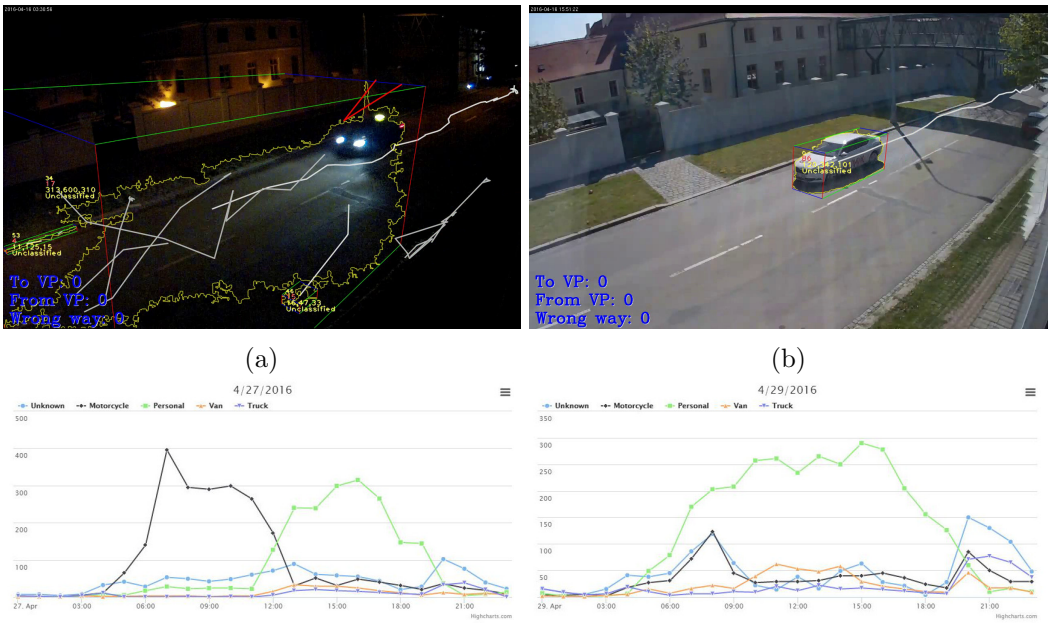
Obrázek 6.3: Ukázky z jednotlivých videí. Postupně 6.9. (1) - (3) a 11.9. (1) - (3) a nakonec záběry z ulice Sportovní v Brně.



(a) (b)

(c) (d)

Obrázek 6.4: Ukázky z reálných kamer na fakultě FIT VUT zabírající ulici Božetěchova vždy ráno (vlevo) a odpoledne (vpravo).



(a) (b)

(c) (d)

Obrázek 6.5: Vlevo noční záběr a výsledný graf se špatným faktorem pro přepočet na reálnou délku, kde v půlce dne došlo k dalšímu přepočtu a návratu na správnou hodnotu. Vpravo ranní záběr s ostrým stínem a výsledný graf, kde kolem 8 hodin ráno je neúměrné množství motocyklů.

6.3 Výsledky

Během vývoje jednotlivých modulů bylo provedeno testování na 4 základních testovacích videích (6.9. (2), (3) a 11.9. (2), (3)) jejichž výsledky byly zobrazeny v grafech na obr. 6.2. Všechny dílčí výsledky jsou uvedeny v tabulce 6.2. V ní si můžeme všimnout, že původní klasifikace na základě HOG deskriptoru není vůbec špatná např. na videu 11.9. (2), u ostatních (a obzvláště u 6.9. (2)) naopak výsledky nejsou příliš povzbudivé. Toto je způsobeno nepřetrénováním na mých datech (tedy snímky vozidel byly pod jiným úhlem). Naproti tomu klasifikace pomocí rozměrů není závislá na úhlu pohledu kamery, a tak i po natrénování SVM klasifikátoru na dvou videích dává vcelku dobré výsledky i na těch dalších.

Dále si můžeme všimnout, že celkový počet vozidel se (většinou) nezměnil mezi testem bez modulů, a sestav s novými moduly pro klasifikaci a odstranění překážek. U těchto sestav dochází pouze k jiné distribuci mezi typy vozidel. Teprve až sestava s modulem pro rozdělení aut zasahuje do samotné detekce a může tak změnit celkový počet vozidel[3].

6.3.1 Výsledky z reálných kamer

Detekce z reálného streamu má několik nedostatků z důvodu chyb v přenosu videa či světelných podmínek (rozhození klasifikace v noci). Řešení je ale plně funkční. Jednotlivé procesy dokáží běžet několik dní v kuse (zatím jsem je vždy zastavil sám z důvodu aktualizace aplikace) a není potřebný jakýkoliv zásah uživatele mimo webovou aplikaci.

Na obr. 6.6a je zobrazen výsledek měření z kamery FIT 1 (mířící směrem k fakultě) po dobu 5 dní, kde lze vidět fluktuace vozidel v průběhu různých dní. Nejvyšší byl pohyb v úterý a postupně pomalu klesá. Je zajímavé sledovat zvýšený pohyb vozidel projíždějících v noci z Pátku na Sobotu a ze Soboty na Neděli oproti jiným nocím. Dále pak na obr. 6.6b a 6.6c je porovnání jednoho dnu z obou kamer. Zde sice celkový počet vozidel nesedí, protože kamera FIT 2 směřuje na křižovatku, ve které větší polovina vozidel odbočí a proto nejsou zaznamenány (systém dokáže detekovat a klasifikovat pouze vozidla jedoucí rovně po dostatečně rovném úseku), ale pokud zanedbáme absolutní počet projetých vozidel a porovnáme pouze tvar křivek, zjistíme, že jsou si docela podobné.

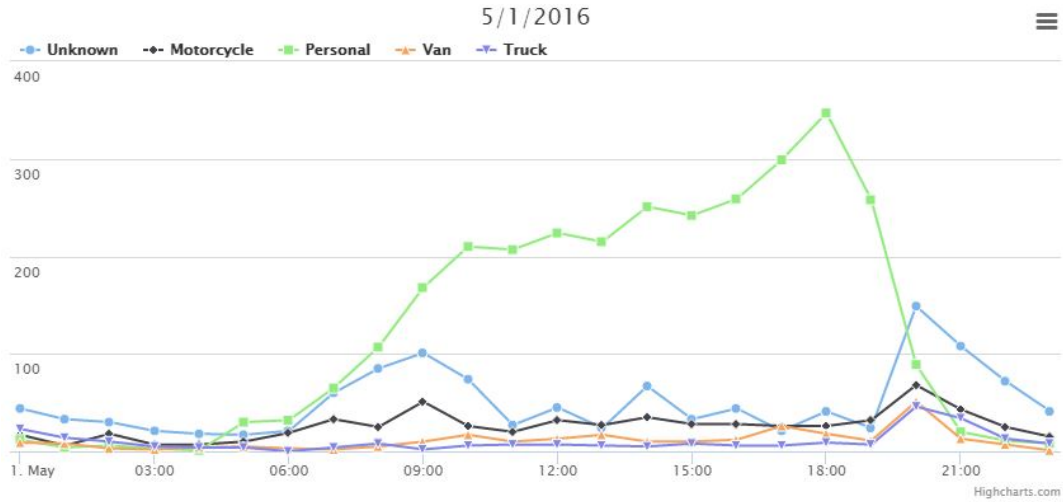
Na výše zmíněných grafech si také můžete všimnout nezvykle vyššího počtu motocyklů. To bohužel není způsobeno motorkářským klubem projíždějícím stále dokola, ale umístěním kamer v místech s vysokým pohybem osob a schované za sklem. To potvrzuje také snímek 6.7a, kde byl jako motorka označen odraz psa ve skle, za kterým je kamera umístěna. Na protějším chodníku (ale občas také uprostřed cesty) je pak vysoký pohyb osob, kde pokud jdou 2 – 3 osoby blízko sebe v určité formaci, mohou být považovány za pomalu jedoucí motorku. Také samotné sklo a odlesky vozidel v něm způsobují další falešné detekce. Obr. 6.7 pak zobrazuje snímky více jednotlivých vozidel s jejich klasifikací tak jak je systém zařadil.

[3] Teoreticky je tohoto schopen i modul pro odstranění překážek ale na mých testovacích videích se u menších vozidel projevilo to, že je překážka rozdělí jen na 1 – 2 snímky (z čehož se algoritmus sám dostane) a u větších vozidel se po rozdělení jedna část před nebo za překážkou zahodila z důvodu např. krátké ujeté vzdálenosti.

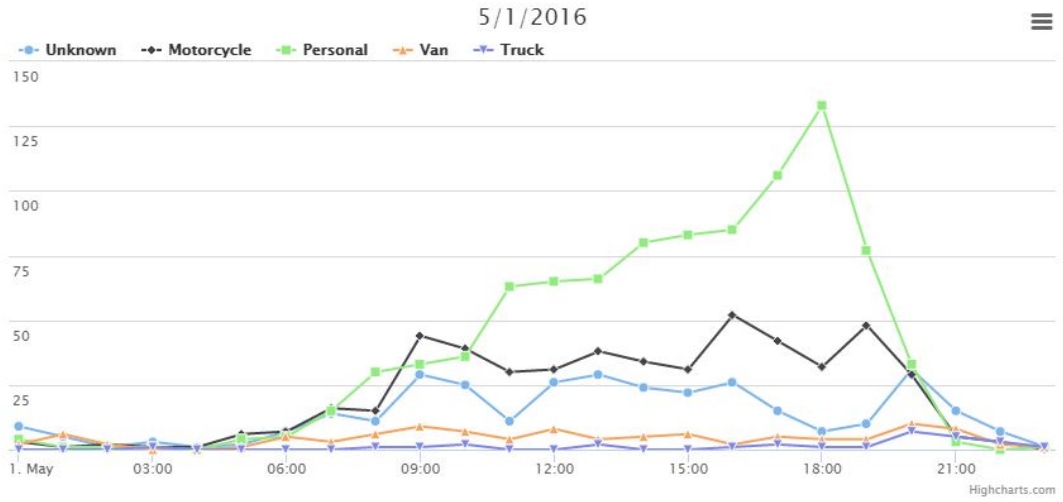
Video	Typ	Skut. hodnoty	Bez modulů[1]	Klasifikace[2]	Klasifikace + překážky[3]	Klasifikace + překážky + rozdělení vozidel[4]
6.9. (2)	Motorcycle	7	0	3	**7**	**7**
	Personal	280	104	291	**265**	**265**
	Van	21	0	0	**20**	16
	Truck	4	217	27	**6**	9
	Unknown	0	0	0	23	22
	Celkem	312	321	321	298 (321)	297 (319)
6.9. (3)	Motorcycle	3	0	0	**3**	**4**
	Personal	218	120	227	**217**	**216**
	Van	15	0	0	**7**	**7**
	Truck	5	111	**4**	0	0
	Unknown	0	0	0	4	3
	Celkem	241	231	231	227 (231)	227 (230)
11.9. (2)	Motorcycle	10	0	1	**6**	**6**
	Personal	469	**471**	488	**466**	**474**
	Van	31	5	0	17	**24**
	Truck	7	25	12	**8**	13
	Unknown	0	0	0	4	4
	Celkem	517	501	501	497 (501)	517 (521)
11.9. (3)	Motorcycle	3	0	0	**1**	**1**
	Personal	454	404	**478**	423	**428**
	Van	39	0	0	**48**	**48**
	Truck	7	82	**9**	13	14
	Unknown	0	0	0	2	2
	Celkem	503	486	487	485 (487)	491 (493)

Tabulka 6.2: Přesnost detekce v závislosti na zapnutých modulech. Modře jsou označeny hodnoty nejblíže skutečnosti (v případě stejného nebo velmi podobného výsledku je označeno více sestav).
(1) Původní aplikace.
(2) S první verzí klasifikace pomocí rozměrů (ručně zadané meze).
(3) S modulem pro detekci překážek a druhou verzí klasifikace pomocí rozměrů (SVM bez přepočtu na reálnou délku).
(4) S modulem pro detekci překážek, rozdělení aut a s třetí verzí klasifikace pomocí rozměrů (SVM s přepočtem na reálnou délku).

(a)



(b)



(c)

Obrázek 6.6: Celkový graf průjezdu vozidel kolem FITu za 5 dní (a), a pohled na den 1.5.2016 z kamery 1 (b) a kamery 2 (c).

(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

(m) (n) (o)

Obrázek 6.7: Snímky vozidel a jejich klasifikace: Motorcycle (a), Personal (b) – (i), Van (j) – (l), Truck (m) – (o).

Kapitola 7

Závěr

V rámci této práce jsem se seznámil s možnostmi detekce a klasifikace vozidel z dopravních kamer. Detailně jsem se pak zabýval aplikací, kterou vyvinul Ing. Jakub Sochor. Tu jsem rozšířil o nové moduly, které zlepšují detekci a rozpoznávací schopnost systému. Zároveň jsem si rád vyzkoušel řešení tak obtížného problému jakým je rozdělení dvou blízko sebe jedoucích vozidel.

Další částí bylo vytvoření Cloudového řešení schopného zpracovávat několik vstupů najednou. Díky zvolené architektuře je systém dobře škálovatelný (stačí přidat nový fyzický stroj a automaticky bude zařazen do systému) a díky využití HTTP zpráv i dobře rozšířitelný a udržovatelný. Samozřejmostí je možnost sledovat dění prakticky v reálném čase, se zpožděním maximálně dvou minut. Systém byl také otestován jak offline, na předem natočených záběrech, tak online po dobu dvou týdnů.

Možnost pokračování v této práci pak vidím především ve spojení dat z několika kamer a rozpoznávání vozidel napříč jimi. Ve složení s Cloudovým řešením by mohl vzniknout skvělý nástroj pro monitorování dopravy ve větším úseku a přispěl by k lepšímu pochopení jednotlivých cest vozidel.

Literatura

[1] Ballard, D. H.: Readings in Computer Vision: Issues, Problems, Principles, and Paradigms. kapitola Generalizing the Hough Transform to Detect Arbitrary Shapes, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987, ISBN 0-934613-33-8, s. 714–725.

[2] Bas, E.; Crisman, J.: An easy to install camera calibration for traffic monitoring. In *Intelligent Transportation System, 1997. ITSC '97., IEEE Conference on*, Nov 1997, s. 362–366, doi:10.1109/ITSC.1997.660502.

[3] Dalal, N.; Triggs, B.: Histograms of Oriented Gradients for Human Detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, CVPR '05, Washington, DC, USA: IEEE Computer Society, 2005, ISBN 0-7695-2372-2, s. 886–893.

[4] Dubská, M.; Herout, A.: Real Projective Plane Mapping for Detection of Orthogonal Vanishing Points. In *Proceedings of BMVC 2013*, The British Machine Vision Association and Society for Pattern Recognition, 2013, ISBN 1-901725-49-9, s. 1–10.

[5] George S. K. Fung, G. K. H. P., Nelson H. C. Yung: Camera calibration from road lane markings. 2003.

[6] He, X.-C.; Yung, N. H. C.: A Novel Algorithm for Estimating Vehicle Speed from Two Consecutive Images. In *WACV*, IEEE Computer Society, ISBN 978-0-7695-2794-9, str. 12.

[7] Hough, P.: Machine analysis of bubble chamber pictures. 1959.

[8] Jakub Sochor, A. H.: Unsupervised Processing of Vehicle Appearance for Automatic Understanding in Traffic Surveillance. In *Proceedings of DICTA 2015*, Australian Pattern Recognition Society, 2015, ISBN 978-1-4673-6795-0, s. 1–8.

[9] Kanhere, N. K.; Birchfield, S. T.: Real-Time Incremental Segmentation and Tracking of Vehicles at Low Camera Angles Using Stable Features. *Trans. Intell. Transport. Sys.*, ročník 9, č. 1, Březen 2008: s. 148–160, ISSN 1524-9050, doi:10.1109/TITS.2007.911357.

[10] Kanhere, N. K.; Birchfield, S. T.; Sarasua, W. A.: Automatic Camera Calibration Using Pattern Detection for Vision-Based Speed Sensing.

[11] Lazaros Grammatikopoulos, E. P., George Karras: Automatic estimation of vehicle speed from uncalibrated video sequences. 2005.

[12] Luvizon, D.; Nassu, B.; Minetto, R.: Vehicle speed estimation by license plate detection and tracking. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, May 2014, s. 6563–6567, doi:10.1109/ICASSP.2014.6854869.

[13] Markéta Dubská, A. H., Jakub Sochor: Automatic Camera Calibration for Traffic Understanding. In *Proceedings of BMVC 2014*, The British Machine Vision Association and Society for Pattern Recognition, 2014, ISBN 1-901725-49-9, s. 1–10.

[14] Markéta Dubská, R. J. J. S., Adam Herout: Fully Automatic Roadside Camera Calibration for Traffic Surveillance. *IEEE Transactions on Intelligent Transportation Systems*, ročník 2014, č. 1, 2014: s. 1–10, ISSN 1524-9050.

[15] Sochor, J.: *Traffic Analysis from Video*. Diplomová práce, Brno University of Technology, Faculty of Information Technology, 2014.

[16] Sonka, M.; Hlavac, V.; Boyle, R.: *Image Processing, Analysis, and Machine Vision*. Nelson Education Limited, 2008, ISBN 9780495082521.

[17] Theodoridis, S.; Koutroumbas, K.: *Pattern Recognition, Third Edition*. Orlando, FL, USA: Academic Press, Inc., 2006, ISBN 0123695317.

[18] Tuan Hue Thi, J. Z., Sijun Lu: Self-calibration of traffic surveillance camera using motion tracking.

[19] Wang, L.-L.; Tsai, W.-H.: Camera Calibration by Vanishing Lines for 3-D Computer Vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, ročník 13, č. 4, Duben 1991: s. 370–376, ISSN 0162-8828.

[20] Welch, G.; Bishop, G.: SIGGRAPH 2001 Course 8 An Introduction to the Kalman Filter.

[21] Yang, J.; Wang, Y.; Sowmya, A.; aj.: Vehicle detection and tracking with low-angle cameras. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, Sept 2010, ISSN 1522-4880, s. 685–688, doi:10.1109/ICIP.2010.5649575.

[22] Yu, S.-H.; Hsieh, J.-W.; Chen, Y.-S.; aj.: An Automatic Traffic Surveillance System for Vehicle Tracking and Classification. In *SCIA*, editace J. Bigün; T. Gustavsson, Lecture Notes in Computer Science, Springer, ISBN 3-540-40601-8, s. 379–386.