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

DETEKCE, SLEDOVÁNÍ A KLASIFIKACE AUTOMOBILŮ
DETECTION, TRACKING AND CLASSIFICATION OF VEHICLES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. RADEK VOPÁLENSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. ROMAN JURÁNEK, Ph.D.
SUPERVISOR

BRNO 2018

Zadání diplomové práce/20513/2017/xvopal00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Vopálenský Radek, Bc.**
Obor: Inteligentní systémy
Téma: **Detekce, sledování a klasifikace automobilů**
Detection, Tracking and Classification of Vehicles
Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte metody pro detekci, sledování a rozpoznání automobilů.
2. Pořiďte vhodný dataset (nahrávky dopravy za různých podmínek) a vytvořte jeho anotaci (bounding boxy automobilů, jejich orientace vůči kameře, typ,...).
3. Vytvořte fungující systém pro detekci, sledování a klasifikaci automobilů v reálném čase.
4. Experimentujte s detekcí, sledováním a klasifikací.
5. Vyhodnoťte výsledky.
6. Vytvořte prezentační materiály.

Literatura:

- Dle pokynů vedoucího

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% 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: **Juránek Roman, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

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

Cílem této diplomové práce je navrhnout a implementovat v jazyce C++ systém pro detekci, sledování a klasifikaci automobilů ze streamů nebo záznamů dopravních kamer. Systém běží na platformě robotického operačního systému a využívá knihovny OpenCV, FFmpeg, TensorFlow a Keras. Pro detekci je využit kaskádový klasifikátor, pro sledování Kalmanův filtr a pro klasifikaci konvoluční neuronová síť. Z celkového počtu 627 automobilů bylo správně sledováno 479. Z toho bylo klasifikováno celkem 458 (nejsou zahrnuty kamiony nebo nákladní automobily). Výsledný systém je možné využívat pro analýzu dopravy.

Abstract

The aim of this master thesis is to design and implement a system for the detection, tracking and classification of vehicles from streams or records from traffic cameras in language C++. The system runs on the platform Robot Operating System and uses the OpenCV, FFmpeg, TensorFlow and Keras libraries. For detection cascade classifier is used, for tracking Kalman filter and for classification of the convolutional neural network. Out of a total of 627 cars, 479 were tracked correctly. From this number 458 were classified (trucks or lorries not included). The resulting system can be used for traffic analysis.

Klíčová slova

doprava, automobily, detekce, sledování, klasifikace, zpracování obrazu, ROS, kaskádový klasifikátor, Kalmanův filtr, konvoluční neuronová síť

Keywords

traffic, vehicles, detection, tracking, classification, image processing, ROS, cascade classifier, Kalman filter, convolutional neural network

Citace

VOPÁLENSKÝ, Radek. *Detekce, sledování a klasifikace automobilů.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Juránek Roman.

Detekce, sledování a klasifikace automobilů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Romana Juránka, Ph.D.

.....
Radek Vopálenský
21. května 2018

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé práce Ing. Romanu Juránkovi, Ph.D. za odborné vedení, za čas, který mi věnoval a rady, které mi pomohly práci vést správným směrem.

Obsah

1 Úvod **2**

2 Existující metody pro získávání informací o objektech z videa **4**
2.1 Existující řešení 4
2.2 Detekce objektů 5
2.3 Sledování objektů 8
2.4 Klasifikace objektů 11

3 Systém pro detekci, sledování a klasifikaci automobilů **14**
3.1 Analýza problému 14
3.2 Nástroje 15
3.3 Návrh systému 17
3.4 Hlavní uzel 19
3.5 Čtení videa 22
3.6 Detekce automobilů 23
3.7 Sledování automobilů 26
3.8 Klasifikace automobilů 28
3.9 Logování výsledků 30

4 Experimenty a vyhodnocení **32**
4.1 Detekce automobilů 34
4.2 Sledování automobilů 39
4.3 Klasifikace automobilů 43

5 Závěr **48**

Literatura **50**

Přílohy **54**

A Obsah DVD **55**

Kapitola 1

Úvod

Tato práce se zabývá získáváním a využíváním informací o silničním provozu, což je v dnešní době velmi často probírané téma. Při analýze a řešení situací v dopravě je využití počítačového vidění považováno za jednu z nejlepších metod. Informace o aktuální dopravní situaci je potřeba získávat například pro sbírání dat potřebných ke zpracovávání analýz a statistik, pro efektivní ovládání dopravních signalizačních zařízení, pro výpočet tras v GPS navigacích, pro detekci nebezpečných situací na silnicích včetně dopravních nehod a podobně. Získávání těchto informací je v současnosti důležité hlavně proto, že se stále zvětšuje počet automobilů a tím houstne doprava na silnicích. Z toho důvodu jsou vyvíjeny různé systémy, které mají za úkol zefektivnit dopravu a zlepšit bezpečnostní situaci tím, že budou regulovat a řídit dopravu podle aktuální situace na komunikacích.

Ke sběru dat lze využít více technologií, například kamery, senzory nebo radary. Kamery se nejčastěji používají pro sledování dopravní situace. Videozáznamy pořízené pomocí kamer jsou následně zpracovávány a analyzovány. Získané výstupy se poté využívají pro zefektivnění a zrychlení dopravy. Tyto kamery mohou být umístěné staticky vedle nebo nad komunikací, také se používají mobilní kamery umístěné přímo v automobilech. Další možností jsou mobilní kamery nesené dronem. Pro sběr dat se dále využívají senzory zabudované ve vozovce nebo nad vozovkou, pomocí kterých lze například zjišťovat počty projíždějících automobilů, případně jejich hmotnost apod. Radary se využívají pro měření rychlosti vozidel na komunikacích.

Nejvíce informací lze získat z videozáznamů, protože automobily jsou kamerou snímány po celou dobu průjezdu sledovaným úsekem. Záznamy se poté analyzují a získaná data jsou dále využívána. Jedním z příkladů dalšího využití je klasifikace, jejíž pomocí lze zjistit o jaký druh vozidla se jedná (osobní, nákladní, autobus nebo kamion), u jednotlivých automobilů určit tovární značku a typ nebo rozpoznat konkrétní SPZ. Dále lze z videozáznamů určit aktuální rychlost automobilů nebo např. počet automobilů jedoucích v jednotlivých pruzích na komunikaci pro monitorování hustoty dopravy.

Toto zadání diplomové práce bylo zvoleno proto, že daná problematika je v současné době velmi aktuální a výsledný systém je využitelný v praxi. Systém běží na platformě *Robotického operačního systému*, který je detailněji popsán v kapitole 3.2. Úkolem systému je získávání informací o vozidlech, která projíždí v záběru dopravní kamery a následné určení jejich tovární značky a modelu. Data mohou být získána buď ze streamu nebo ze záznamu. Vozidlo je nejprve programem detekováno, během pohybu v rámci záběru kamery sledováno a následně je pomocí klasifikátoru určena jeho značka a typ.

Práce je rozdělena do dvou stěžejních kapitol a kapitoly, která se věnuje testování systému a získaným výsledkům. Kapitola 2 je v první části zaměřena na teoretický popis

existujících řešení a ve druhé na rozdělení a nejvyužívanější metody pro detekci, sledování a klasifikaci. V kapitole 3 je nejprve vysvětlena samotná analýza problému a následuje popis potřebných nástrojů pro správné fungování výsledného systému. Nejpodstatnější část této kapitoly tvoří návrh systému a popis hierarchie systému. Systém je rozdělen na bloky, které jsou následně v kapitole podrobněji rozebrány.

Kapitola 4 se věnuje samotnému testování v praxi. Jsou zde uvedeny a popsány výsledky testování a experimentů provedené na jednotlivých částech systému.

Závěrečná kapitola 5 obsahuje shrnutí autorovy práce na vývoji systému pro detekci, sledování a klasifikaci automobilů.

Kapitola 2

Existující metody pro získávání informací o objektech z videa

Cílem této kapitoly je seznámit čtenáře s postřehy získanými studiem odborných článků. Články se zabývají tvorbou systémů, které jsou podobné zadanému tématu diplomové práce. V dalších podkapitolách jsou pak popsány existující metody pro detekci, sledování a klasifikaci objektů. Tyto metody řešení jednotlivých problémů jsou nezbytné pro správnou funkčnost celého systému.

2.1 Existující řešení

Podobným zadáním se zabývá celá řada odborných článků. Nejčastěji jsou v nich zmiňována řešení, která využívají záznamy ze statických kamer, umístěných například u komunikací ve městech, případně na dálnicích. Tyto záznamy jsou potom využívány k měření nebo kontrole rychlosti, rozpoznání SPZ automobilů nebo rozeznávání druhu vozidla.

Článek [23] objasňuje zadanou problematiku o detekci a klasifikaci druhů automobilů. Na rozdíl od této práce program popsaný ve článku nevyhodnocuje konkrétní tovární značku a model automobilu, ale rozlišuje pouze druh vozidla (osobní automobil, autobus, kamion, atd.) a počítá četnost výskytů vozidel těchto jednotlivých kategorií. Autoři použili pro detekci Houghovu transformaci [19] a pro klasifikování vozidel kNN klasifikátor [24]. Pomocí experimentů autoři zjistili, že systém počítá četnost vozidel s 97% úspěšností a klasifikuje s úspěšností mezi 88 – 91 %.

Problematiku klasifikace automobilů, kdy je určena značka a typ vozidla, popisuje článek [11]. Autoři zde představují systém, který detekuje pouze zadní část automobilů pomocí určení polohy jejich SPZ a zadních světel. Klasifikaci automobilů provádí systém hierarchicky, kdy nejprve přiřadí automobilu tovární značku a následně určí konkrétní typ. Tento systém byl testován na vzorku 280 obrázků zachycujících zadní část automobilů a dosáhl úspěšnosti 96 %.

Pro získání informací o detekci a sledování automobilů je vhodný například i článek [39], který pochází přímo z Fakulty informačních technologií VUT v Brně. Autor pro detekci využívá metodu odečítání pozadí [33, 42] a pro sledování automobilů Kalmanův filtr [3, 14, 46].

Veškeré články pojednávající o zadané problematice mají jeden zásadní společný rys. Než je možné začít s klasifikací (automobilů nebo SPZ) nebo například s měřením rychlosti, musí se nejprve automobil v obraze detekovat a pak, v době kdy se objevuje v záběru

kamery, sledovat. Tento krok je důležitý proto, aby mohl program určit, zda se jedná o jedno konkrétní vozidlo. V opačném případě by v následujících snímcích sice bylo vozidlo detekováno, ale bylo by mu vždy přiděleno nové označení (ID) a pro analýzu dopravy by tyto informace byly nepoužitelné.

Proces získávání informací o vozidlech (značka a typ) z videa lze tedy rozdělit na tři části:

- **Detekce** – proces vyhledávání automobilů v obraze/videu.
- **Sledování** – proces lokalizace konkrétního pohybujícího se automobilu na všech po sobě jdoucích snímcích videa, dokud automobil neodjede ze záběru.
- **Klasifikace** – proces určení tovární značky a typu jednotlivých automobilů, které byly detekovány a sledovány v obraze/videu.

Autoři používají metody, které vždy odpovídají konkrétním požadavkům na systém. Pro detekci automobilů se využívají metody založené na pohybu [27, 33, 36, 37, 42] nebo na vzhledu [12, 20, 28, 31, 34, 35, 36, 37, 43]. Pro sledování automobilů se využívají metody, které sledují body [3, 14, 46, 47], siluety [9, 47], nebo sledují objekty pomocí jádra [27, 47]. Pro klasifikaci jsou využívány například klasifikátor SVM (z angl. Support Vector Machines) [26], AdaBoost (z angl. Adaptive Boosting) [44] nebo klasifikace pomocí konvolučních neuronových sítí [15].

Následující text se zabývá detailnějším popisem nejčastěji používaných metod pro detekci, sledování a klasifikaci objektů.

2.2 Detekce objektů

Detekce objektů v obraze je jeden ze základních úkonů v počítačové grafice. Aby bylo možné objekt z obrazu (videa) sledovat, klasifikovat nebo analyzovat, je třeba ho vždy nejdříve detekovat.

V současné době jsou vyvíjeny stále nové metody pro detekci objektů v obraze, takže jich aktuálně existuje celá řada. V této kapitole jsou popsány pouze některé z nich, které jsou zajímavé svým přístupem k problematice. Publikace [37] rozděluje metody detekce objektů do dvou hlavních skupin a to na metody založené na pohybu [27, 33, 36, 37, 42] a metody založené na vzhledu [12, 20, 28, 31, 34, 35, 36, 37, 43].

Metody založené na pohybu

Metody založené na snímání objektů během pohybu [27, 33, 36, 37, 42] potřebují pro detekci objektu sekvenci po sobě jdoucích snímků videa.

Do skupiny detektorů založených na pohybu patří například rozdílová metoda [27], metoda odečítání pozadí [33, 42] a detekce na základě pohybu významných bodů [36]. Tyto tři metody jsou používány nejčastěji.

Rozdílová metoda [27] pohybující se objekty detekuje pomocí rozdílu mezi dvěma do sobě jdoucími snímky videa. Výpočet je lehce implementovatelný a snadno adaptovatelný na různá prostředí. Nevýhodou této metody je chybná detekce pomalu se pohybujících objektů, kde může dojít k rozdělení na více segmentů nebo mohou detekce zaniknout. Další nevýhodou je, že metoda není úplně přesná, protože je problematické získat celý obrys pohybujícího se objektu.

Při využití **metody odečítání pozadí** [33, 42] může být videosekvence rozdělená na popředí (foreground) a pozadí (background). Popředí se skládá ze zájmových objektů a v pozadí jsou objekty nepotřebné k detekci. Postup je tedy takový, že se nejdříve odstraní pozadí, aby zůstalo pouze popředí obsahující objekty, které mají být detekovány. K tomuto úkonu se musí zjistit referenční model pozadí, který může být buď implicitně daný nebo se vypočítává. Nejdůležitější částí procesu detekce je tedy kvalitní tvorba modelu pozadí. Čím větší je jeho spolehlivost, tím lepší jsou výsledky detekce. Lze využít rekurzivní nebo nerekurzivní výpočet.

Rekurzivní výpočet rekurzivně aktualizuje model pozadí na základě každého snímku, a proto nepotřebuje buffer. Díky tomuto přístupu může mít i snímek z dávné minulosti vliv na aktuální model pozadí. Nevýhodou výpočtu je, že každá chyba v pozadí může přetrvávat velmi dlouhou dobu.

Nerekurzivní výpočet využívá pro odhad pozadí princip posuvného okna. V bufferu je uložený určený počet předchozích snímků a aktuální pozadí je odhadováno na základě změn konkrétního pixelu v průběhu času. Případnou nevýhodou je velikost bufferu. Oproti rekurzivnímu výpočtu má nerekurzivní výpočet větší paměťové nároky.

Detekce na základě pohybu významných bodů [36] se skládá ze dvou kroků. V prvním kroku se v referenčním obraze detekují významné body a k nim se ve druhém kroku hledají odpovídající body na následujících snímcích. Body, které se vyskytují v dostatečném množství po sobě jdoucích snímků, jsou označené za hledané objekty.

Metody založené na vzhledu

Metodám založeným na rozpoznávání objektů podle vzhledu [12, 20, 28, 31, 34, 35, 36, 37, 43] stačí pro detekování hledaného objektu jeden obraz. Výstupem všech detektorů je seznam souřadnic detekovaných objektů v jednotlivých snímcích.

K těmto metodám patří například detektory vzorů [31, 34], detektory tvarů [31, 34], bodové detektory [31, 34], detektory detekující podle barvy [31, 34] a detektory detekující pomocí klasifikátoru [31, 34].

Metody založené na vzhledu jsou v problematice detekce vozidel z videa využívané častěji, než metody založené na pohybu.

Vstupem **detektoru vzorů** [31, 34] je vzor objektu a tento vzor se snaží detektor nalézt v obraze. Při úspěšném hledání nalezne polohu části obrazu odpovídající vzoru. Pro porovnání lze využít dva principy, buď na základě korelace mezi vzorem a vybranou částí obrazu nebo odčítáním vzoru od vybrané části obrazu. Varianta s využitím korelace je odolnější vůči šumu nebo změně osvětlení, ale je náročnější na výpočet.

Detektor tvarů [31, 34] detekuje objekty v obraze pomocí tvarové analýzy a vyžaduje dobrou segmentaci obrazu. Obrys objektu je reprezentován konečnou množinou bodů, křivkami nebo regiony. Lze použít i zjednodušenou reprezentaci objektů, která ale musí obsahovat informace potřebné k opětovnému získání tvaru. Tato reprezentace se nazývá tvarový deskriptor. Metoda se využívá zejména pro jednodušší scény. U složitých scén dochází k problémům kvůli časté ztrátě viditelnosti objektu.

Princip **bodového detektoru** [31, 34] spočívá v tom, že v obraze hledá významné body. Tyto body mají ve svém okolí výraznou strukturu, což může být například textura. Výhodou této metody je, že je odolná proti globálním změnám obrazu. Například když dojde ke změně osvětlení, změně měřítka nebo pozice kamery. Mezi nejvýznamnější zástupce metody patří: SURF detektor [28], SIFT detektor [35] nebo Moravcův detektor rohů [12].

Metoda detekující podle barvy [31, 34] detekuje objekt v obraze pomocí barevných vlastností obrazu. Nejvíce se využívají modely `L*a*b` nebo `L*u*v` kvůli neuniformnosti RGB modelu. Detektor je sice výpočetně nenáročný, ale není příliš efektivní.

Princip **detekce pomocí klasifikátoru** [31, 34] spočívá v tom, že je možné natrénovat klasifikátor, který následně s určitou pravděpodobností rozhoduje, zda se hledaný objekt v obraze nachází nebo ne. Jedná se tedy o vytvoření funkce z poskytnutých dat, která na každý vstup (objekt) mapuje požadovaný výstup (nachází-li se objekt v obraze, vrátí jeho polohu). Klasifikátor se musí natrénovat do dvou tříd: do jedné třídy patří pouze obrázky hledaného objektu a do druhé všechny ostatní. Třídy se musí pro správnou funkčnost navzájem vylučovat. Na vstupu při trénování musí být k dispozici dostatečné množství dat pro obě třídy, aby mohly být vyhodnoceny jak pozitivní, tak negativní prvky (Obrázek 2.1).

Zástupcem této metody je **kaskádový klasifikátor** [20, 43], který je využit pro detekci automobilů v navrženém systému.

Obrázek 2.1: První dva obrázky vlevo zobrazují pozitivní prvky, tedy prvky které obsahují hledaný objekt. Další dva obrázky potom představují prvky negativní, tedy prvky, které obsahují buď pouze část hledaného objektu nebo jiné objekty.
[1]

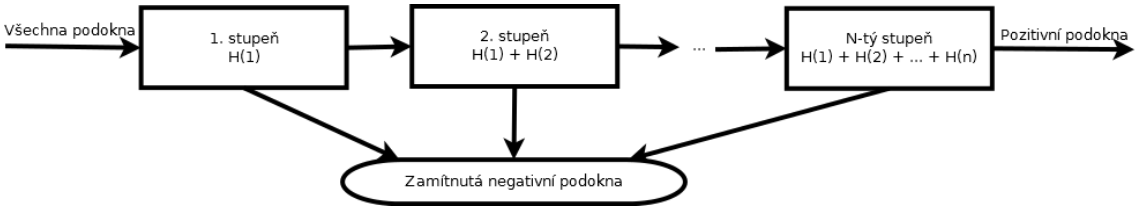
Kaskádový klasifikátor

Kaskádový klasifikátor [20, 43] se skládá z určitého počtu stupňů. Každý stupeň kaskády obsahuje určitý počet *slabých* klasifikátorů. Výsledkem je *silný* monolitický nelineární klasifikátor $H(x)$, který má svoji prahovou hodnotu P. Podle prahové hodnoty se klasifikátor rozhoduje, zda je aktuální podokno pozitivním či negativním prvkem. Účelem kaskády je zkrátit dobu detekce hledaného objektu v obraze. Na obrázku 2.2 je znázorněn a vysvětlen princip zapojení klasifikátorů do kaskády. Trénování kaskády může probíhat například metodou Viola-Jones [43].

Shrnutí metod pro detekci

Kromě výše zmíněných metod existují hybridní metody, které pro detekci objektů v obraze kombinují více způsobů a principů pro detekci objektů v obraze. Pro zrychlení a zefektivnění detektoru může programátor také určit, ve které oblasti obrazu chce hledaný objekt detekovat. Například pokud se sledovaný jízdní pruh nachází v levé polovině scény, stačí, když bude detektor hledat objekty pouze v této části obrazu. Tímto způsobem se také předchází nežádoucím nebo chybným detekcím.

[1] Obrázek 2.1 je získán z videa, pomocí kterého byl vyvíjen výsledný systém.



Obrázek 2.2: Schéma principu kaskádového klasifikátoru. Princip kaskádového klasifikátoru spočívá v tom, že na vstupu je velké množství podoken, kdy každý stupeň kaskády pošle pozitivní podokna na další stupeň a zamítne určité množství negativních podoken. První stupeň kaskády obsahuje pouze malé množství klasifikátorů a měl by zamítnout co nejvíce negativních snímků. Následující stupně vždy obsahují všechny klasifikátory ze stupňů předešlých. Každý stupeň je natrénován tak, aby dosahoval určité procentuální hodnoty pozitivních detekcí i hodnoty falešně pozitivních detekcí. Účelem je dosáhnout co největší hodnoty pozitivních detekcí a co nejmenší počet falešně pozitivních detekcí.

2.3 Sledování objektů

Cílem všech metod zabývajících se sledováním objektu je vygenerovat jeho trajektorii, která znázorňuje pohyb objektu ve videu.

Fáze detekce a generování trajektorie při sledování objektu mohou být spojené do jednoho kroku nebo rozdělené do dvou kroků. Metody spojující detekci a sledování objektů do jednoho kroku detekují objekty v každém snímku videa. Nalezené detekce potom propojují a tím vytváří trajektorie pohybu jednotlivých objektů. Při použití metod, které rozdělují detekci a sledování objektů do dvou samostatných kroků, je trajektorie získávána aktualizací pozice objektu v aktuálním snímku na základě jeho předešlých pozic v již vyhodnocených snímcích.

Metody pro sledování objektů lze rozdělit do tří základních skupin: sledování bodu (point tracking) [3, 14, 46, 47], sledování pomocí jádra (kernel tracking) [9, 47] a sledování siluety (silhouette tracking) [27, 47]. Podrobné rozdělení těchto metod je znázorněno na obrázku 2.3.

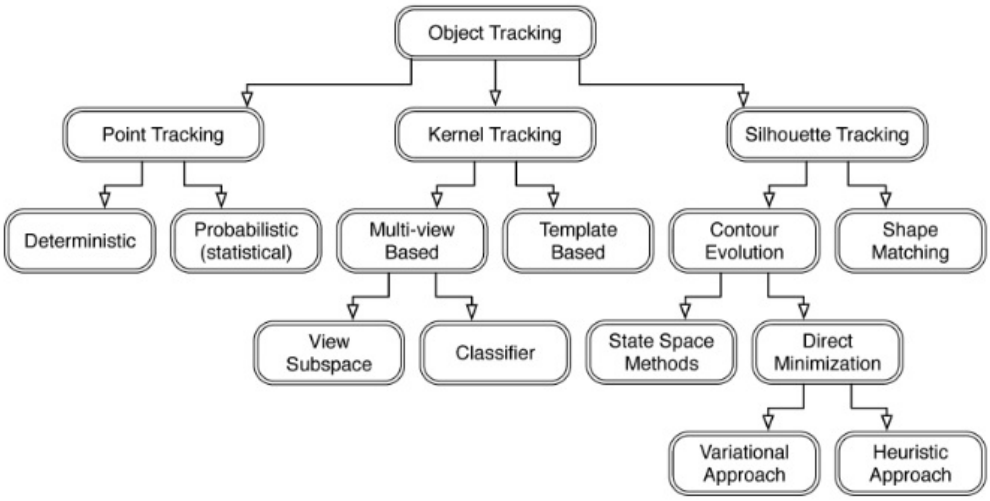
Sledování bodu

Metody pro sledování bodů [3, 14, 46, 47] je vhodné používat pro sledování objektů, které zabírají malou oblast ve snímku videa (např. hejna ptáků na obloze). Objekty jsou reprezentovány podle velikosti buď jedním centrálním bodem nebo souborem bodů. Při používání těchto metod je třeba provádět detekci objektu na každém snímku videa. Algoritmy pro nalezení odpovídajících dvojic bodů lze rozdělit na deterministické a statistické (pravděpodobnostní).

Deterministické metody [47] definují cenu přiřazení každého objektu v předešlém snímku (snímek v čase $t-1$) ke sledovanému objektu v aktuálním snímku (snímek v čase t) pomocí těchto omezení:

- **Blízkost (Proximity)** – situace, kdy se pozice objektu z předešlého snímku v aktuálním snímku příliš nezmění.

[2]Obrázek 2.3 je převzat z: [47].



Obrázek 2.3: Detailní rozdělení metod pro sledování objektů.
[2]

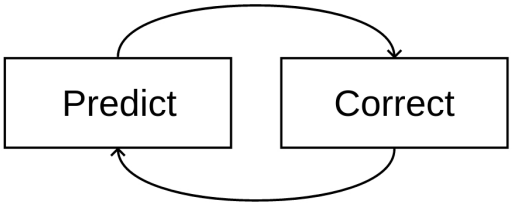
- **Maximální rychlost (Maximum velocity)** – omezuje pomocí poloměru r vzdálenost, kterou může objekt urazit.
- **Malá změna rychlosti (Small velocity change)** – zaručuje plynulost pohybu objektu tím, že se mezi snímky rychlost a směr pohybu objektu příliš nemění.
- **Společný pohyb objektů (Common motion)** – situace, kdy se sleduje více objektů najednou. Řeší se pomocí pravidla, kdy objekty zůstávají převážně v podobném sousedském rozestavení.
- **Tuhost (Rigidity)** – předpokládá, že objekty reálného světa nemění svůj tvar a proto vzdálenost mezi dvěma body jednoho objektu zůstává neměnná.
- **Blízká jednotnost (Proximal uniformity)** – kombinace blízkosti a malé změny rychlosti.

Tato omezení lze využít i u statistických metod.

Statistické metody [47] do sledování zanáší šum, aby simuloval klasický šum vzniklý snímáním kamery. Princip těchto metod je rozdělen do dvou modelů. První model popisuje vztah mezi předcházejícím stavem objektu s přidaným bílým šumem a novým aktuálním stavem. Druhý model zpracovává závislost aktuálního stavu se šumem měření. Klasickým představitelem statistických metod je **Kalmanův filtr** [3, 14, 46], který je využit k implementaci sledování automobilů v navrhovaném systému.

Kalmanův filtr

Kalmanův filtr [3, 14, 46] je rekurzivní filtr používaný k odhadu polohy objektu, kdy nejsou k dispozici skutečné hodnoty nebo je čas měření delší, než je rychlost zobrazování. Stav systému je reprezentován vektorem reálných čísel a kovariací mezi měřenými a odhadovanými hodnotami. Kalmanův filtr pracuje ve dvou základních stavech *Predict* a *Correct*, které se mezi sebou iterativně opakují (Obrázek 2.4).



Obrázek 2.4: Princip Kalmanova filtru. Fáze *Predict* slouží k předpovědi pravděpodobné pozice objektu v následujícím snímku videa a aktualizaci nové chyby měření. Výsledkem této fáze je tedy první odhad polohy objektu pro další krok (snímek v čase $t+1$). Druhou fází je fáze *Correct* zajišťující opravu predikce na základě měření, čili předpokládanou pozici objektu opraví na skutečnou pozici, kterou později (snímek v čase $t+1$) zjistí detektor. Dále tato fáze aktualizuje takzvaný *Gain* Kalmanova filtru, který představuje odhad stavu a chybu měření. Pokud dojde k detekci dosud nesledovaného objektu, je vytvořen nový Kalmanův filtr pro tento nově nalezený objekt. A naopak, pokud některý ze sledovaných předmětů nebyl na předem určeném počtu snímků detekován, Kalmanův filtr pro tento objekt zanikne a sledování tím skončí.

Ve stavu *Predict* jsou aplikovány pro získání predikce následující rovnice:

$$\hat{x}_k^{'} = F\hat{x}_{k-1}$$

$$P_k^{'} = FP_{k-1}F^T + Q$$

kde $\hat{x}_k^{'}$ je střední hodnota apriorního odhadu, $P_k^{'}$ je kovariace apriorního odhadu, Q je kovariace stavového šumu a předpokladem je, že střední hodnota šumu je nulová.

Ve druhém stavu (stav *Correct*) je opravena predikce na základě naměřených hodnot pomocí následujících rovnic:

$$K_k = P_k^{'}H^T(HP_k^{'}H^T + R)^{-1}$$

$$\hat{x}_k = \hat{x}_k^{'} + K_k(z_k - H\hat{x}_k^{'})$$

$$P_k = (I \ - K_kH)P_k^{'}$$

kde K_k je Kalmanův zisk, R je kovariace šumu měření (platí předpoklad, že šum má střední hodnotu nulovou), $\hat{x}_k$ je střední hodnota aposteriorního odhadu, P_k je kovariace aposteriorního odhadu a I je identická matice.

Vylepšením Kalmanova filtru je částicový filtr (particle filter) [3, 25], který se od Kalmanova filtru liší v tom, že není omezen sledováním právě jednoho kandidáta na sledovaný objekt v aktuálním snímku.

Sledování pomocí jádra

Metody sledování pomocí jádra [9, 47] pro sledování objektů využívají pro reprezentaci předmětu základní geometrická primitiva, tedy například obdélník nebo elipsu. Tyto metody se dají rozdělit na metody používající modely se vzhledem založeným na šablonách a hustotě (Template and Density-Based Appearance Models) [47] a na vícepohledové metody (Multiview Appearance Models) [47].

Mezi metody využívající modely se vzhledem založeným na šablonách a hustotě patří Mean-Shift [9] nebo KLT tracker [45]. Nevýhodou těchto metod je, že se jejich modely generují za běhu programu. Pokud tedy dojde k 3D rotaci objektu, rotace změní tvar sledovaného objektu v průběhu sledování a metody potom selhávají.

Problém s měnícím se tvarem objektu během jeho sledování řeší vícepohledové metody, které před začátkem sledování natočí model objektu z více různých pohledů. Zástupcem těchto metod je například SVM tracker [48].

Sledování siluety

Metody pro sledování siluety [27, 47] pracují s popisem tvaru objektu definovaného modelem. Model je generován na základě tvaru siluety a je reprezentován pomocí histogramu barev, obrysů objektu nebo pomocí hran. Tyto metody jsou vhodné pro sledování objektů, které mění tvar nebo se také využívají, pokud je třeba sledovat objekt s velkou přesností, což může být například pohyb ruky člověka. Sledování siluety lze dále rozdělit na metody porovnání tvarů (shape matching) [47] a na metody sledování obrysů (contour tracking) [47].

V algoritmech založených na **porovnání tvarů** je hledání provedeno výpočtem podobnosti modelu objektu s regiony v aktuálním snímku. Při výpočtu podobnosti se využívá například Hausdorffova metrika [18], která zajistí porovnání dvou množin bodů.

Princip **metody sledování obrysů** se od předchozí metody velmi liší. Zde dochází k iterativnímu vyvíjení obrysu získaného v předchozím snímku tak, aby se co nejvíce podobal obrysu v aktuálním snímku. Podmínkou je, aby se alespoň část objektu získaného z předchozího snímku zcela překrývala s částí objektu v aktuálním snímku.

2.4 Klasifikace objektů

Cílem klasifikace je navrhnout a vytvořit mechanismus, který na základě trénovacích dat nalezne určitou podobnost (vytvoří model) a po vytvoření modelu dokáže s co nejmenší možnou chybou rozdělit testovací data do modelem vytvořených tříd. Trénování klasifikátoru může probíhat pomocí metod, které buď využívají učitele, nebo které ho nepotřebují [50].

Výhodou metod trénování klasifikátoru s učitelem je, že mají k dispozici apriorní informaci o třídě. Cílem trénování je, aby klasifikační mechanismus pomocí trénovacích dat zobecnil model a díky tomu s co nejmenší chybou zařadil testovaná data do jednotlivých tříd.

Naproti tomu metody trénování klasifikátorů bez učitele nemají apriorní informaci o původu dat. Proto se snaží najít mezi daty podobnost, například pomocí eukleidovské vzdálenosti. Data jsou tedy rozdělena podle podobnosti do skupin, které vlastně samy o sobě tvoří jednotlivé třídy.

V dnešní době existuje pro rozpoznání objektů mnoho metod. Často se využívají samostatné klasifikátory, mezi které patří SVM [26] či AdaBoost [44]. Dále se pro klasifikaci také používají konvoluční neuronové sítě [15].

Klasifikátor SVM (z angl. Support Vector Machines) [26] je metoda strojového učení, kterou lze využít i pro klasifikaci objektů. Jelikož se jedná o metodu strojového učení, je pro jeho správnou funkčnost nezbytné ho nejprve natrénovat na trénovací množině dat. Cílem metody je oddělit dvě třídy objektů nadrovinou a to tak, aby mezi nimi byla co největší volná plocha. Plocha se spočítá jako vzdálenost nadroviny od nejbližšího objektu dané

třídy v n-dimenzionálním prostoru. Objekty nacházející se nejblíže specifikované rovině se nazývají *support vectors* a celá nadrovina je jimi definována pomocí vztahu 2.1.

$$g(x) = w^T x + w_0 = 0, \tag{2.1}$$

kde w je normálový vektor nadroviny a w_0 je skalární práh. Díky tomuto principu nemůže dojít k *přetrénování* klasifikátoru. Takto definovaných nadrovin může být nekonečně mnoho.

Princip metody **AdaBoost** (z angl. Adaptive Boosting) [44] je vytvoření klasifikátorů pomocí lineární kombinace *slabých* klasifikátorů, které mohou být reprezentovány pomocí rozhodovacího stromu nebo například perceptronem. Podmínkou je, aby jeho chyba byla menší než 0,5. V jednotlivých krocích je k výslednému klasifikátoru přidán *slabý* klasifikátor, který nejlépe přiřazuje špatně ohodnocená data. Avšak klasifikátor vytvořený pomocí tohoto principu už není lineární. Jeho přesnost je velmi vysoká a je označen jako *silný* klasifikátor.

Původně je AdaBoost navržen pro klasifikaci pouze do dvou tříd. Později byla ovšem vyvinuta různá rozšíření, která dovolují klasifikovat do více tříd. Výhodou metody je velká odolnost vůči *přetrénování* a poměrně snadná implementace. Nevýhodou je potřeba velkého množství dat pro trénovací proces.

Konvoluční neuronové sítě

Konvoluční neuronové sítě (*Convolutional Neural Networks* – CNN) [15] jsou specifickým druhem neuronových sítí, které jsou určeny pro zpracování dat s charakteristickou strukturou. V praxi jsou často využívány ke klasifikaci obrázkových dat ze snímků videa. Například pro klasifikaci automobilů, které byly detekovány a sledovány v jednotlivých snímcích videa. Proto jsou konvoluční neuronové sítě využity v tomto systému pro klasifikaci automobilů.

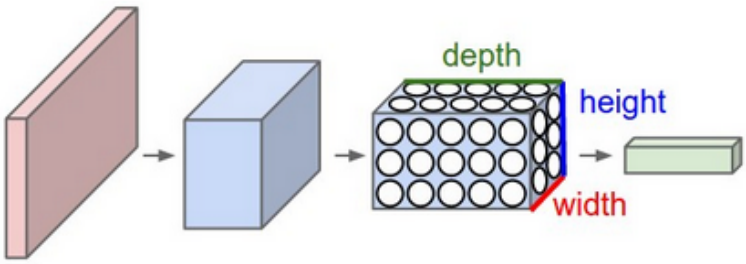
CNN mohou mít různou architekturu a kromě vstupní a výstupní vrstvy se skládají z dalších specifických vrstev. Mezi základní vrstvy patří:

- **Konvoluční vrstva** – je reprezentována obdélníkovou mřížkou neuronů a vyžaduje, aby její předcházející vrstva byla také reprezentována touto mřížkou, protože neurony dostávají na vstup data právě z obdélníkové podsekce předchozí vrstvy. Mřížek může být v konvoluční vrstvě více a vstupem každé z nich jsou data ze všech mřížek z předchozí vrstvy. Váhy jsou pro všechny neurony v této vrstvě stejné. Konvoluční vrstva je obrazem konvoluce předchozí vrstvy, kde váhy specifikují konvoluční filtr. Typicky tato vrstva obsahuje více konvolučních filtrů.

- **Vrstva poolingu** – v architektuře CNN se přidává za konvoluční vrstvu. Její účel spočívá v podvzorkování konvolučního obrazu na vstupu. Cílem této vrstvy je tedy dostat na vstup obdélníkové bloky z konvoluční vrstvy a tyto bloky poté sloučit do jednoho výstupu. Z n vstupů vznikne jeden výstup. Pomocí vrstvy *poolingu* se dá předcházet přeučení, protože díky ní se dá snižovat počet parametrů a množství výpočtů v síti. Pro *pooling* lze využít více metod, jako jsou například maximum nebo průměr. Nejčastěji se používá funkce pro maximum.

- **Klasifikační vrstva neboli plně propojená vrstva** – je nejvyšší vrstvou konvoluční neuronové sítě. Tato vrstva provádí vysokoúrovňové rozhodování a počet jejích neuronů se rovná počtu tříd, do kterých klasifikuje data ze vstupu. Vstupem této vrstvy jsou všechny výstupy (tedy příznakové vektory vstupního obrazu) z předchozí vrstvy.

Na obrázku 2.5 je znázorněna základní struktura konvoluční neuronové sítě. Vyobrazená síť obsahuje vrstvy s trojrozměrným uspořádáním neuronů. Každá vrstva převede trojrozměrný vstupní vektor na výstupní pomocí neuronových aktivací. Výstupem konvoluční vrstvy je souhrn všech tříd a k nim přiřazené hodnoty pravděpodobnosti. Tyto hodnoty určují, ke které třídě má být vstupní objekt přiřazen. Souhrn pravděpodobnostních hodnot musí být tedy logicky jedna. Pokud je konvoluční síť dobře natrénovaná, hodnoty s nejpravděpodobnější třídou jsou vyšší než 0,85. Je tedy velká pravděpodobnost správného zařazení vstupního obrazu.



Obrázek 2.5: Vrstvy konvoluční neuronové sítě. V této síti reprezentuje červená vstupní vrstva obrázek, který bude klasifikován a výška se šířkou této vrstvy odpovídá rozlišení vstupního obrázku. Modrý kvádr reprezentuje skrytou vrstvu, kde její hloubku určuje počet barevných kanálů. V tomto případě je využit barevný model RGB, čili hloubka je rovna třem. Vrstvy obsahují proměnlivé počty příznakových map, kde příznaky mohou být různé kombinace barev na různých částech obrazu. Zelený kvádr znázorňuje konvoluční vrstvu. Výstupem vrstvy je informace, kde je ke každé třídě přidělena pravděpodobnost jejího přiřazení ke vstupnímu obrazu.
[3]

[3]Obrázek 2.5 je převzat z: [15].

Kapitola 3

Systém pro detekci, sledování a klasifikaci automobilů

Zadáním diplomové práce je navrhnout a implementovat systém pro detekci, sledování a klasifikaci vozidel.

V této kapitole je popsána analýza zadaného problému, dále jsou zde zmíněny nástroje používané při vývoji systému a nástroje potřebné pro jeho správný chod. Ve druhé části kapitoly je vysvětlen konkrétní návrh systému a je zde popsána hierarchie celého systému. Jednotlivé bloky schématu systému jsou vysvětleny v samostatných podkapitolách. Tam jsou také popsány zajímavosti ohledně implementace systému.

3.1 Analýza problému

Jedním z cílů vývojářů zabývajících se zefektivněním dopravy (Inteligentní dopravní systémy [29]) je, aby řidiči nemuseli trávit zbytečně dlouhý čas na křižovatkách nebo v kolonách a aby se zvýšila bezpečnost. Příkladem neefektivně vytvořeného systému je křižovatka řízená semafory, kde je přepínání barev nastaveno staticky na určitý časový interval. Tedy i když je na vozovce jediné vozidlo, řidič musí předem stanovený časový interval čekat na zelenou. Moderní systém pro analýzu dopravy by v této situaci řidiči, který je na křižovatce sám, sepnul zelenou v jeho směru jízdy okamžitě.

V současnosti, kdy jsou informační technologie na vysoké úrovni, se stále častěji využívají počítačové systémy, které jsou schopné na základě dat z kamer získat a využít více různých informací najednou. V minulosti bylo nutné pro získání každé informace použít jiný postup (např. rychlost bylo možné měřit pouze pomocí radaru [38] nebo efektivní řízení křižovatek na základě aktuální dopravní situace museli provádět policisté osobně).

Proces zpracování dat lze rozdělit na dvě části, na sběr dat (z kamery) a software (počítačový program, který provádí na základě získaných dat požadované úkony). Pro co nejlepší výsledky je důležitá jak kvalita získaných dat, tak i kvalita vytvořeného softwaru.

Analyzovat dopravu pomocí kamer je také výhodné z hlediska snadné instalace. Stačí nainstalovat kameru na místo, kde je žádoucí dopravní situaci sledovat a mít k dispozici funkční systém pro určenou problematiku. Další výhodou je, že získaná data mohou být zálohována na požadovanou dobu a následně je možné je využít pro zpracování různých statistik (např. počet projetých automobilů, nejčastější značky automobilů, nejrozšířenější barva vozidel, atd.) v určených časových intervalech (denně, týdně, měsíčně, atd.).

Pro získání kvalitních výsledků je rozhodující odpovídající kvalita záznamu (záleží na technických parametrech kamery) a zároveň i vhodné umístění kamery. Umístění kamer není omezeno pevně daným úhlem, ze kterého snímají vozovku, je tedy možné kamery umístit na budovy, mosty nebo jiné konstrukce. Důležité je však umístit kamery v dostatečné výšce nad komunikacemi tak, aby se snímaná vozidla při průjezdu sledovaným úsekem co nejméně vzájemně zakrývala. Při častém zakrývání vozidel by nebyly zakryté automobily zaneseny do statistik a výsledky by byly zkreslené.

Mezi další faktory, které ovlivňují správnou činnost systému patří meteorologické podmínky. Například silný sluneční svit může zastiňovat vozidla, nebo může záběry z kamer značně znepřehlednit hustý déšť nebo sněžení.

3.2 Nástroje

V této podkapitole jsou stručně popsány nástroje a knihovny, které jsou využity pro implementaci a správný chod celého systému. Mezi ty patří *Robotický operační systém*, *Docker* a knihovny *FFmpeg*, *OpenCV*, *TensorFlow* a *Keras*.

Robotický operační systém

Hlavním cílem *Robotického operačního systému* (ROS) [1, 30] je umožnit vývojářům rychlé a snadné vytváření modulů pro aplikace na společné platformě. Nabízí vývojářům hardwarovou abstrakci, ovladače zařízení, knihovny, vizualizéry, předávání zpráv mezi procesy a správu balíčků. Aplikace vytvářená v ROS se skládá z více částí, které se nazývají *nody*. Jednotlivé *nody* jsou opakovaně použitelné pro další nové systémy. ROS je licencován pod open source na základě BSD licence a je rozšířen po celém světě. Dále jsou popsány mechanismy ROS, které jsou využity pro komunikaci systému.

Node

Každý *node* [1, 30] musí být pojmenovaný. *Node* je vlastně proces, který provádí výpočet. To znamená, že pomocí jednoduchých *nodů* je implementována část řešení aplikace, která je v ROS tvořena mnoha *nody*. Spousta *nodů* pro různá využití už je vytvořena. Jsou volně dostupné a lze je použít pro vlastní řešení. Jednotlivé *nody* mezi sebou komunikují pomocí vzájemného zasílání zpráv na *topicy* nebo pomocí *service*.

Topic

Topic [1, 30] je pojmenovaný komunikační kanál mezi publisherem a subscriberem, přes který jednotlivé *nody* komunikují. Název *topicu* se používá k identifikaci obsahu zprávy. *Node* (publisher) vysílá zprávu tak, že zveřejní dotyčné informace na daný *topic*. *Node* (subscriber), který má zájem o určitý druh dat, se připojí na příslušný *topic* a z něho čte potřebná data. Na jediný *topic* může být zároveň připojeno více publisherů i subscriberů. Jeden *node* může zveřejnit a nebo se přihlásit k odběru z více *topiců*. Pro zjištění dostupných *topiců* slouží příkaz: `rostopic list`.

Message

Message [1, 30] je datová struktura, která reprezentuje *topic*. Tato datová struktura může obsahovat celé nebo desetinné číslo, pole, atd. Zprávy mohou také obsahovat libovolně

vnořené struktury. Výpis zpráv z jednotlivých *topiců* lze vypsat pomocí příkazu: `rostopic echo jméno_topicu`.

Service

Service [1, 30] funguje na principu dotaz-odpověď. To znamená, že server přijme požadavek od klienta a *node* podporující tento požadovaný *service* ho provede. Klient následně obdrží odpověď. Struktura *service* je dána souborem typu *srv*. Pro zjištění dostupných *service* slouží příkaz: `rosservice list`.

Rosrun

Rosrun [1, 30] slouží k spouštění jednotlivých *nodů*, které jsou implementovány v jazyce Python. Formát příkazu pro spuštění: `rosrun adresář jméno_nodu.py`.

Roslaunch

Roslaunch [1, 30] spouští předem definované *nody* s parametry, vlastními jmennými prostory a přemapováním *topiců* na vstupy/výstupy. Tyto informace jsou uloženy v jednom *launch* souboru. Formát příkazu pro spuštění: `roslaunch adresář jméno_launche.launch`.

Docker

Docker [6, 21] je určený primárně k usnadnění vývoje, údržby, dodávání a provozování aplikací pomocí aplikačních kontejnerů. Tedy umožňuje aplikaci izolovat v rámci kontejneru, společně se všemi jejími závislostmi, potřebnými knihovnami a programovým prostředím, od ostatních procesů na systému hostitele. Jedná se o open source platformu.

FFmpeg

FFmpeg [7] je open source multimediální framework pod licencí LGPL nebo GPL, který umožňuje nahrávání, přehrávání, konverzi a streamování digitálního zvuku a obrazu. Jedná se o velmi rychlý audio/video konvertor. Jako zdroj pro *FFmpeg* může posloužit téměř cokoliv, od jednoduchého video souboru až po vysílání streamu. Knihovna obsahuje velké množství různých kodeků pro dekódování videa, jako jsou například `MPEG-4, H.264, AAC, FLAC`.

Princip činnosti spočívá v počátečním volání knihovny *libavformat* k načtení vstupních snímků a k získání dat pomocí demuxeru. Kódované pakety jsou potom poslány do dekodéru. Dekodér produkuje nekomprimované rámce (raw video, PCM audio, atd.), které mohou být zpracované filtrováním. Následně po filtrování jsou data předána do kodéru, kde jsou zakódována a poslána jako kódované pakety. Nakonec jsou tyto pakety vstupem muxeru, který je zapíše do výstupního souboru.

OpenCV

OpenCV (z angl. Open Source Computer Vision Library) [5] je open source knihovna pod licencí BSD nabízející implementaci široké škály algoritmů z oboru počítačového vidění, zpracování obrazu a strojového učení. Vyřeší problematiku jako je filtrace a segmentace obrazu, rozpoznávání a sledování objektů, kalibrace kamer, spojování snímků, analýzy pohybu nebo 3D rekonstrukce scény. Tato knihovna byla navržena s důrazem na výpočet v reálném čase.

TensorFlow

TensorFlow [2] je open source software knihovna pro strojovou inteligenci. Provádí numerické výpočty pomocí grafu toků dat. Uzly v grafu představují matematické operace a hrany grafu reprezentují multidimenzionální datová pole neboli tenzory. Flexibilní architektura určitého prostředí umožňuje nasazení na jeden popřípadě více CPU nebo GPU. *TensorFlow* byl vyvinut vědci a inženýry pracujícími pro Google. Dnes je ovšem volně dostupný.

Keras

Keras [16] je open source vysokoúrovňová knihovna pro práci s neuronovými sítěmi. Pro složité výpočty využívá *TensorFlow*, které bylo vyvinuto s důrazem na umožnění rychlého experimentování. Tato knihovna je schopná běžet bez problémů na CPU nebo na GPU. Mezi hlavní výhody patří snadná rozšiřitelnost. Nové moduly lze bez problémů přidat ke stávajícímu projektu. Modulem je graf nebo sekvence grafů a měl by být krátký a jednoduchý.

3.3 Návrh systému

Aplikace běží na platformě *Robotického operačního systému* a je implementována v programovacím jazyce C++, který ROS plně podporuje. Pouze *node*, který klasifikuje automobily je implementován s využitím Python rozhraní systému ROS. Tento způsob je jednodušší než implementace v jazyce C++, protože je možné přímo využít knihovny *Keras* a *TensorFlow*.

ROS jsem vybral vzhledem k tomu, že podporuje škálovatelnost aplikace na jednotlivé bloky (*nody*), které mezi sebou komunikují pomocí dvou mechanismů pojmenovaných *topic* a *service*. Dalším důvodem je také podpora práce s knihovnami *OpenCV, FFmpeg, TensorFlow* nebo *Keras*, které jsou využity pro implementaci tohoto systému.

Vzhledem k náročné a mnohdy zdlouhavé instalaci ROS jsem zvolil jeho použití v rámci *docker* kontejneru. *Docker* umožňuje snadnou instalaci, přenositelnost a znovupoužití jeho *image*. Pomocí *Dockeru* je ROS velmi rychle zprovozněn a připraven k okamžitému použití.

V použitém *docker image* byly k dispozici pouze knihovny *FFmpeg* a *OpenCV*. Knihovny *TensorFlow* a *Keras*, což jsou prerekvizity pro klasifikaci, musely být dodatečně doinstalovány. Výhoda použití *Dockeru* spočívá v tom, že se dá *image* vytvořený ke správnému chodu systému uložit. Uživatelé, kteří systém budou používat, mohou rovnou namapovat vytvořený *image*. Systém lze poté využívat okamžitě, bez nutnosti další instalace prerekvizit.

Systém je navržen tak, že sbírá data z video souborů, které jsou pořízeny statickými kamerami umístěnými nad dopravními komunikacemi. Informace získané z těchto kamer jsou potom zpracovávány v reálném čase (live streamy) nebo jsou záznamy z nich uložené pro budoucí zpracování. Vstupem je URL streamu nebo uloženého videa. Navržený systém je primárně určen pro videa, kdy je automobil snímán zepředu a jede směrem ke kameře.

Výstupem jsou jednotlivé logovací soubory zvlášť pro každý úkon (detekce, sledování a klasifikace vozidel). Pro detekci jsou tyto soubory ve formátu: číslo snímku a pozice bounding boxu, což je rámeček, který ohraničuje nalezené vozidlo ve videu.

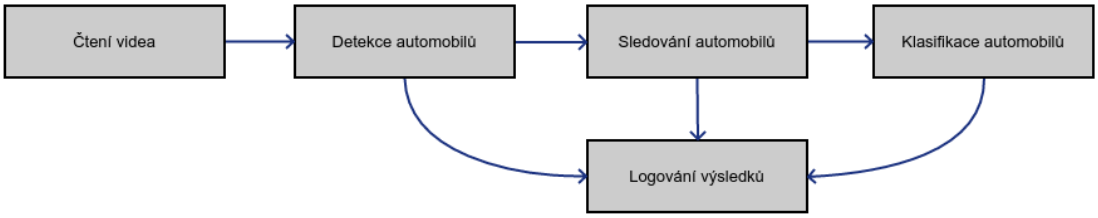
Pro sledování automobilů jsou logovací soubory ve formátu: číslo snímku, na kterém byl daný automobil poprvé detekován. Následuje pole, které obsahuje souřadnice detekcí. Pole obsahuje pouze jeden bod určený osami X a Y, protože se pracuje se středem bounding boxu.

Klasifikační logovací soubor je ve stejném formátu jako výsledný soubor pro tracking. Navíc jsou zde uloženy informace o klasifikaci, tedy číslo přiřazené třídy, která reprezentuje tovární značku a typ daného automobilu.

Na obrázcích níže jsou zachyceny různé typy chyb a situací, ke kterým docházelo během implementace systému. Při vývoji bylo použito jiné video než pro testování.

Navržený systém může být součástí systému pro analýzu dopravy, který jsem implementoval společně se spolužákem Pavlem Hájkem. Tento systém jsme prezentovali na konferenci Excel@Fit 2017. V README systému je podrobněji popsána naše spolupráce.

Jak bylo uvedeno v podkapitole Existující řešení, lze řešení problému rozčlenit na bloky pro detekci, sledování a klasifikaci vozidel. Pro správné fungování systému je nutný také blok pro otevření a překódování vstupního videa a blok pro logování výsledků. Na obrázku 3.1 je zobrazeno základní blokové schéma celého systému se správným vnitřním zapojením jednotlivých bloků a tím jsou znázorněny i komunikační kanály mezi jednotlivými *nody*.



Obrázek 3.1: Blokové schéma vnitřního zapojení celého systému. Jednotlivé bloky představují v ROS *nody*.

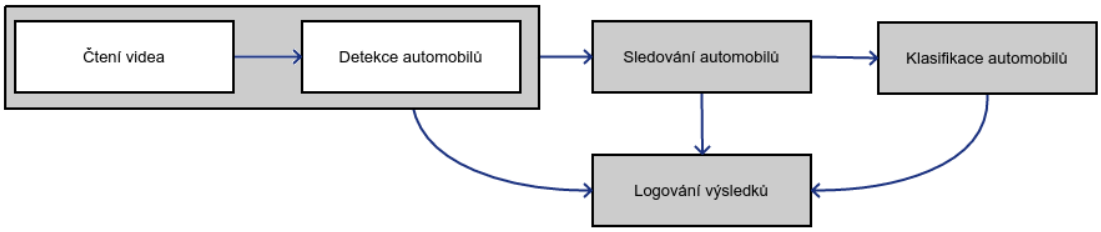
Celý systém je tedy rozdělen na následující ROS balíčky:

- **Hlavní uzel** – `traffic` – základní blok, který sdružuje potřebné informace od všech ostatních *nodů* a na jejich základě řídí celý systém. Také zajišťuje vizualizaci aktuální scény a logování dosažených výsledků.
- **Čtení videa** – `traffic_decode` – otevře, čte a dekóduje vstupní video. Jeho vstupem je URL streamu nebo video souboru a výstupem potom jednotlivé snímky videa zapisované na zadaný *topic*.
- **Detekce automobilů** – `traffic_detection` – detekuje automobily v jednotlivých snímcích videa. Jeho vstupem jsou tedy snímky videa a jako výstup vrací informace o nalezených automobilech ve formě jejich bounding boxů. Všechny tyto informace jsou postupně publikovány na *topic*.
- **Sledování automobilů** – `traffic_tracking` – vstupem tohoto bloku jsou informace z *topicu*, který publikuje výstup detektoru a na výstupu publikuje výsledek sledování automobilů. Tento balíček tedy doplňuje ID jednotlivých vozidel a tím spojuje více bouding boxů z různých snímků. Díky tomu je vytvořena trajektorie dráhy pohybu jednotlivých automobilů.
- **Klasifikace automobilů** – `traffic_classification` – vstupem jsou snímky (výřezy) jednotlivých vozidel a výstupem jsou ID třídy, které reprezentují tovární značku a model automobilu, jehož snímek byl vstupem.

Při vývoji systému se ukázalo, že toto vnitřní blokové zapojení způsobovalo, že dosažené výsledky nebyly uspokojivé.

Problém spočíval v tom, že detekční blok nestíhal zpracovávat všechny uložené snímky vstupního videa. Bylo to způsobené tím, že detekce jednoho snímku trvala kolem 60 ms, což byl příliš dlouhý čas. Během zpracování jednoho snímku přicházely na určený *topic* následující snímky videa. Jelikož ale stále ještě nebyl zpracován aktuální snímek, docházelo k tomu, že nebyly zpracovány některé snímky a po dokončení detekce byl použit poslední snímek, který byl na *topic* uložen. Například byl tedy detekován každý desátý snímek videa a ostatní snímky byly zahozeny. To se negativně projevovalo ve výsledných trajektoriích sledovaných vozidel.

Jako správné řešení tohoto problému, se ukázalo spojit bloky pro čtení videa a detekci do jednoho společného bloku. Hlavní změnou proti předchozí verzi je, že jednotlivé snímky videa jsou předávány funkci pro detekování automobilů pomocí parametru a ne pomocí *topicu*. Tímto způsobem jsou detekovány všechny příchozí snímky. Jinak tento nový blok zapisuje informace na příslušné *topicy* stejně jako předtím dva samostatné bloky. Stále jsou tedy snímky videa ukládány na *topic*, ale jsou využívány pouze pro vizualizaci. Aktuální podoba schématu výsledného systému je znázorněna na obrázku 3.2.



Obrázek 3.2: Výsledné blokové schéma vnitřního zapojení systému pro detekci, sledování a klasifikaci automobilů.

Spojení těchto dvou bloků tedy vedlo k získání více detekcí pro každý automobil. Tento rozdíl je znázorněn na obrázku 3.3.

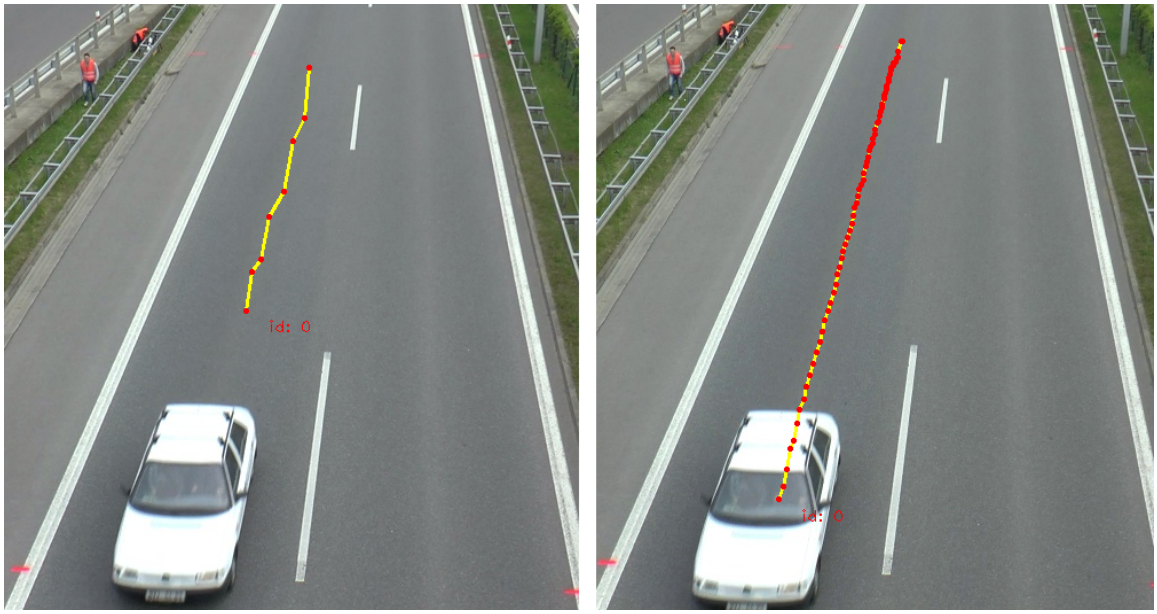
Komunikace

Systém pro komunikaci mezi *nody* využívá mechanismy ROS, jednotlivé bloky mezi sebou komunikují pomocí *topicu* nebo mechanismu *service*. Pro tento účel je nutné definovat nové datové typy. Definice těchto datových typů jsou umístěny v hlavním balíčku *traffic*. V tabulce 3.1 je zobrazen formát komunikace a sdílení dat mezi jednotlivými bloky. Podrobněji je komunikace mezi *nody* popsána v následujícím textu, který se věnuje jednotlivým blokům systému.

3.4 Hlavní uzel

Hlavní uzel (*traffic*) ovládá činnost celého systému a obsahuje veškeré soubory pro nastavení a spuštění aplikace a také soubory definující strukturu mechanismů pro komunikaci jednotlivých bloků.

Tento uzel obsahuje *launch* soubor (spouštěcí soubor). Aby uživatel nemusel jednotlivé *nody* spouštět samostatně, existují v ROS *launch* soubory, které spustí všechny nadefinované



Obrázek 3.3: Na snímku vlevo je zobrazena situace, která nastávala při počátečním zapojení bloků. Na snímku vpravo je naopak zobrazena situace po spojení bloků pro čtení videa a detekci vozidel. Protože v druhém případě byly k detekci použity všechny snímky videa, jsou získané trajektorie tvořeny z více bodů. Výsledkem je přesnější trajektorie, která neobsahuje výkyvy jako původní trajektorie z počátku vývoje systému. Rozdíl mezi oběma trajektoriemi je viditelný pouhým okem.

Tabulka 3.1: Při čtení videa je každý jeho snímek uložen na *topic* s informacemi o jeho časové stopě a pořadovém čísle. Detekci a sledování mají na starosti dva odlišné *nody*, ovšem oba dva používají stejný datový typ. Detektor ukládá na *topic* informace o bounding boxu a aktuální výřez daného vozidla. Tracker si postupně načítá informace od detektoru a ty si přeuloží na svůj *topic* a doplní každé detekci ID automobilu, čímž postupně spojuje detekce a tím vytváří jednotlivé trajektorie. Klasifikátor jako jediný *node* využívá pro komunikaci s hlavním blokem mechanismus *service* ve formátu, kdy vstupem klasifikačního uzlu je seznam výřezů a jako výstup posílá vítěznou třídu modelu, která reprezentuje tovární značku a typ daného automobilu.

Informace (médium)	Položky
video (*topic*)	snímek pořadové číslo snímku čas snímku
detekce a sledování (*topic*)	ID automobilu bounding box výřez automobilu
klasifikace (*service*)	seznam výřezů automobilu ID třídy modelu

nody najednou. Dále také načte konfigurační soubor (podrobněji je konfigurační soubor popsán dále), pomocí kterého si uživatel nastaví všechny parametry nezbytné pro správný chod systému.

Další skupinou souborů, které je zde třeba zmínit a vysvětlit, jsou soubory pro komunikaci *nodů* mezi sebou. Jsou zde soubory typu *msg*, reprezentující strukturu jednotlivých *topiců*, na kterých jsou sdružovány potřebné informace. Dalšími soubory, které jsou potřebné pro chod aplikace, jsou soubory typu *srv*. Tyto soubory reprezentují strukturu zpráv pro komunikační mechanismus *service*.

Kromě zmíněných specifických souborů, se zde nachází zdrojové kódy hlavního *nodu*. Jedná se o klasický C++ soubor a jeho hlavičkový soubor. Pomocí těchto dvou souborů jsou ovládány všechny ROS balíčky znázorněné v blokovém schématu. Z uvedených *nodů* získává postupně hlavní uzel potřebné informace pro řízení systému. *Node* je schopný komunikovat se všemi bloky systému. Ostatní bloky komunikují pouze se svými sousedy.

Během běhu systému jsou získané informace ukládány do třídy *Car*. Třída *Car* uchovává všechny potřebné informace o jednotlivých vozidlech. Při volání destruktoru jsou spuštěny funkce pro logování výsledků.

Je zde také implementován *timer*, který každou vteřinu zjišťuje, zda nebylo dokončeno sledování jednoho nebo více automobilů. Pokud bylo dokončeno, jsou zkompletovány informace o daném vozidle, které jsou nutné pro proces klasifikace. Výstupem je seznam vozidel, která už opustila prostor snímaný kamerami a tato vozidla jsou následně klasifikována. Informace potřebné pro klasifikaci (ID a výřezy automobilu) jsou uloženy do fronty. Tam jsou postupně ukládány údaje o všech vozidlech, která už nejsou v záběru kamery. Do fronty jsou informace o jednotlivých automobilech ukládány proto, že pro klasifikaci se nejprve musí načíst model, pomocí kterého jsou automobily zařazovány do jednotlivých tříd. K načtení modelu je potřeba určitý časový úsek, proto pro klasifikaci není k dispozici hned po spuštění systému. Ve chvíli, kdy je model načten, začíná proces klasifikace jednotlivých informací o vozidlech z fronty. Protože klasifikace je náročný proces, běží paralelně s hlavním uzlem v jiném vlákně.

Díky rozdělení běhu systému do dvou vláken nedochází ke zpomalení ostatních částí systému. Zpomalení bylo znát také při vizualizaci funkčnosti systému. Ta se při běhu systému na jednom vlákně prakticky nepohybovala a tím pádem nebylo možné takovou vizualizaci používat. Zároveň se systém ani nepřibližoval běhu v reálném čase.

Po přijetí posledního snímku videa do hlavního uzlu a jeho zpracování končí činnost všech *nodů* kromě klasifikace. Ta začala se zpožděním po načtení potřebného modelu a také je její vykonávání výpočetně náročné. Hlavní uzel čeká, až se ukončí klasifikace a fronta údajů o neoklasifikovaných vozidlech bude prázdná. Pokud se jedná o video záznam, tak v tomto okamžiku hlavní uzel ukončí činnost celého systému. Při použití živého streamu běží systém do té doby, než ho uživatel vypne.

Vizualizace

Hlavní uzel také zajišťuje vizualizaci systému. Postupně zobrazuje snímky, které si načítá z *topicu*, na kterém jsou uloženy jednotlivé snímky videa. Do obrazu kromě snímků videa vykresluje i další informace. V levém horním rohu je zobrazena časová stopa videa a číslo aktuálního snímku. Dále jsou detekované a sledované automobily označené ID a jejich trajektorie je žlutá čára, kde červené tečky označují předešlé polohy automobilu.

Konfigurační soubor

Pomocí konfiguračního souboru, který je ve formátu YAML, se nastavují veškeré parametry jednotlivých bloků:

- **Čtení videa** – URL videa, hodnota *drop_frame* (počet vynechaných snímků vstupního videa), maximální šířka videa (při půlení rozlišení) a název *topicu*.
- **Detekce** – cesta ke kaskádě, cesta ke vstupní masce videa, parametry detektoru a název *topicu*.
- **Sledování** – zapnutí/vypnutí kontroly směru jízdy jednotlivých automobilů, parametry trackeru a název *topicu*.
- **Klasifikace** – cesta ke klasifikačnímu modelu, maximální počet výřezů pro klasifikaci jednoho automobilu, způsob určení výsledné třídy a název *topicu*.
- **Logování** – umístění a názvy logovacích souborů a umístění, kam se mají ukládat výřezy sledovaných automobilů.

Vždy se jedná o název *topicu*, na který se mají získané informace zapisovat jednotlivými *nody*.

3.5 Čtení videa

Pro implementaci otevření, čtení a dekódování vstupního videa je využita knihovna *FFmpeg*. Tato knihovna obsahuje všechny potřebné nástroje pro práci s videem a umožňuje otevřít jak videa ze záznamu, tak i živé streamy. Výhodou knihovny *FFmpeg* je, že otevře jakékoliv video bez nutnosti parsovat zadané vstupní URL.

Tento blok postupně čte a dekóduje jednotlivé rámce videa na snímky. Vstupem je tedy URL video souboru, který bude zpracováván. Výsledkem je sekvence snímků vstupního videa, které jsou postupně zapisovány na zadaný *topic*. Dále jsou snímky předávány detekční části jako parametr.

Systém uchovává o jednotlivých snímcích následující informace: ID snímku a čas, ve kterém se snímek objevuje ve videu (v sekundách a milisekundách).

ROS má větší režii, zejména v případě kopírování snímků na *topic*, a důsledkem toho je, že systém neběží v reálném čase pro videa, která mají vysokou hodnotou FPS. Délka času uložení snímku na *topic* je závislá na jeho velikosti. Jako zrychlení se ukázala komprimace snímku z formátu BGR na JPEG. Komprimace má také určitou režii, ale při vývoji se ukázalo, že i přes to je vhodné ji využít, protože se tak razantně zvýší rychlost uložení snímku na *topic*.

Další zrychlení přineslo půlení rozlišení vstupního videa. Rozlišení je děleno dokud neklesne šířka snímku pod nastavenou hranici počtu pixelů (zde 720 pixelů). U testovaných videí je rozlišení děleno pouze jednou (scale = 2). Program dále i přes půlení rozlišení pracuje s původními hodnotami velikostí souřadnic videa.

Aby systém fungoval co nejrychleji a odpovídal reálnému času, musí být v určitých případech (například málo výkonný PC) vynechávány některé pořízené snímky videa. Testovaná videa mají vysokou hodnotu FPS, při které je vynechání menšího počtu snímků zanedbatelné. Počet vynechaných snímků (kladné celé číslo) lze přizpůsobit podle zadaných požadavků na systém (uživatel může požadovat rychlost nebo naopak co největší přesnost získaných informací i s vědomím, že systém nepoběží v reálném čase).

3.6 Detekce automobilů

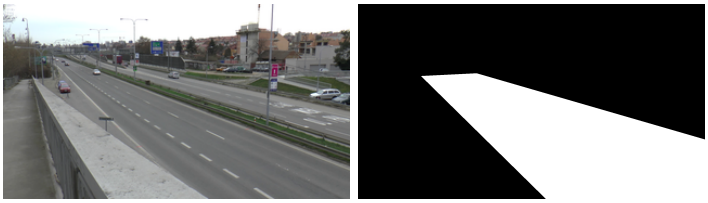
Pro detekování automobilů byl zvolen kaskádový klasifikátor [20, 43], který pro detekci potřebuje natrénovanou kaskádu.

Princip detekce automobilů

Na vstupu detektoru musí být k dispozici sekvence snímků vstupního videa. Snímky jsou detektoru postupně předávány pomocí parametru z funkce pro čtení videa. Další podmínkou pro fungování detektoru je mít na vstupu natrénovanou kaskádu pro vyhledávání objektů (v tomto případě automobilů) ve snímcích videa.

Detektor tedy prochází jednotlivé snímky a pro každý snímek pomocí kaskády vytvoří seznam nalezených automobilů. Tyto informace posílá v jednotlivých iteracích přes určený *topic* bloku systému, který obstarává sledování automobilů. Na *topic* se ukládá poloha jednotlivých automobilů ve formě bounding boxů. Bounding box obsahuje souřadnice levého horního rohu rámečku (X a Y) a informace o jeho velikosti (šířka a výška). Pomocí získaných bounding boxů nalezených automobilů jsou ze snímků videa získávány výřezy aktuálně detekovaných vozidel. Tyto výřezy jsou spolu s informacemi o bounding boxech dále pomocí hlavního uzlu načteny z *topicu* a ukládány do třídy *Car*, protože jsou v dalším průběhu zpracování využity pro klasifikaci.

Ke vstupnímu streamu nebo video souboru lze přidat do konfiguračního souboru i vstupní masku tohoto videa (Obrázek 3.4), pomocí které se detektor rapidně zrychlí, neboť hledá automobily pouze tam, kde by se měly vyskytovat, tedy pouze na silnicích ve vybraných jízdních pruzích a směrech, které mají být analyzované. Použitím vstupní masky se také zvyšuje přesnost detektoru, protože nejsou vyhledávány automobily, které nejsou součástí analyzovaného dopravního pruhu.



Obrázek 3.4: Levý obrázek zobrazuje screen videa a pravý obrázek masku tohoto videa. V případě přítomnosti masky vyhledává detektor automobily pouze v její bílé oblasti.

Před spuštěním systému je možné na základě použitých videí vyladit detektor pomocí dvou stěžejních parametrů:

- **scale_factor** (double) – určuje, o kolik procent je velikost snímku snížena na obrazovém měřítku a pomocí toho určuje rychlost a důkladnost detekce. Pro příklad scale_factor = 1.03 znamená, že je velikost snímku snížena o 3 %.
- **min_neighbors** (int) – určuje, kolik sousedů si má každý kandidát udržet, čímž určuje přesnost a počet detekcí. Při nastavení parametru na hodnotu 4, která je v navrhovaném systému používána, je detekce přesnější a počet detekcí menší. Je to dáno tím, že jsou detekovány pouze objekty, u kterých je vyšší pravděpodobnost, že se skutečně jedná o automobily.

Popis kaskády pro detekci

Kaskáda je ve formátu XML. V navrženém systému je používána kaskáda, jež byla natrénována na Fakultě informačních technologií VUT v Brně a je dostupná pro její studenty.

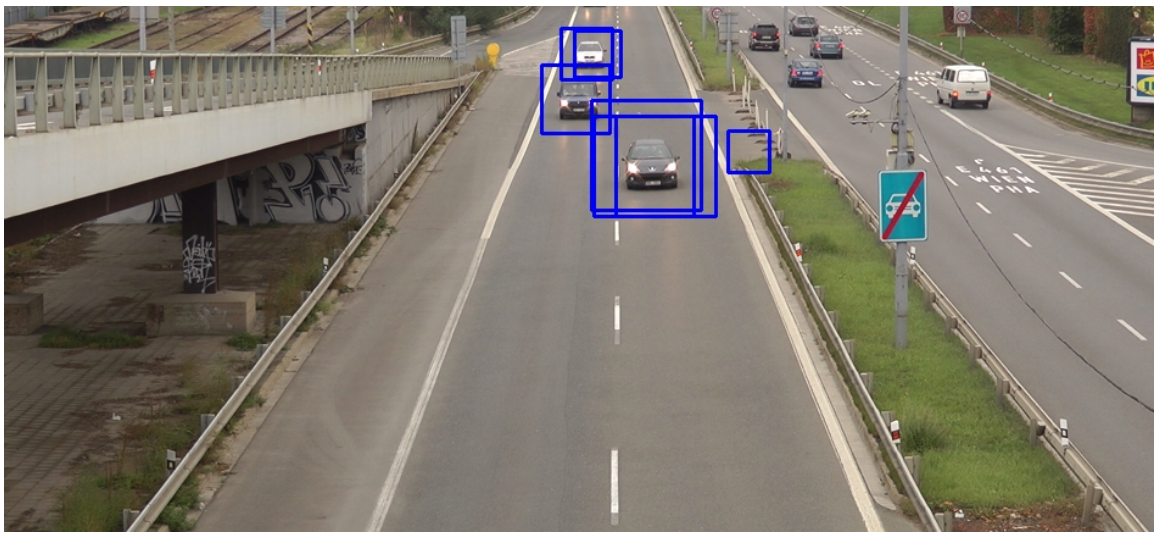
Tato kaskáda je natrénována na fakultním datasetu *COD20K* [13]. Tento dataset obsahuje přibližně 10 000 trénovacích a přes 1 100 testovacích snímků s anotacemi popisujícími polohy automobilů.

Existence spolehlivé a funkční kaskády byla jedním z důvodu pro výběr metody detekování objektů pomocí kaskády. Kaskádový klasifikátor je implementován knihovnou *OpenCV*[1], která je podporována platformou ROS. Což byl další důvod, proč zvolit tuto metodu.

Odstranění chyb detektoru

Při práci detektoru docházelo k následujícím chybám: detekování jiných objektů, než vozidel (false positive), detekování jednoho automobilu v rámci jednoho snímku vícekrát (duplicity) a poslední chybou bylo nedetekování automobilu v aktuálním snímku (false negative).

Problém s detekováním jiných objektů než vozidel (Obrázek 3.5) je řešen tak, že pokud se detekce objevila nahodile pouze v jednom snímku a v dalších snímcích už ne, je tato detekce vypuštěna a dále se s ní nepracuje. Pokud je ovšem parametr detektoru, konkrétně *min_neighbors* nastaven na vyšší hodnotu (v tomto systému defaultně na hodnotu 4), pak se tyto špatné detekce prakticky neobjevují.



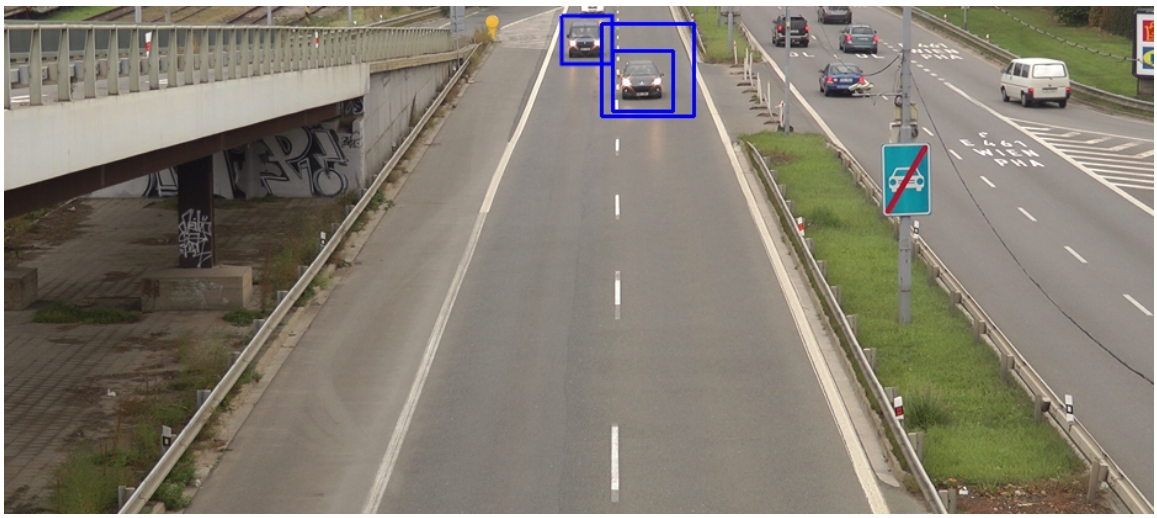
Obrázek 3.5: Ukázka špatné funkčnosti detektoru, kdy byl nastaven parametr *min_neighbors* na hodnotu 0. Na tomto obrázku je dobře vidět, že při špatně nastavené hodnotě tohoto parametru detektor detekuje i objekty, které nejsou vozidla a zároveň detekuje jednotlivá vozidla v rámci jednoho snímku vícekrát.

Vícenásobná detekce jednoho automobilu v jednom snímku se sice během testování na testovacích videích neobjevovala příliš často, avšak i malé množství vzniklých chyb bylo třeba odstranit, zejména kvůli správné funkčnosti sledování vozidel. Pokud by vícenásobná

[1] http://docs.opencv.org/2.4/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html

detekce nebyla odstraněna, bylo by jednomu automobilu přiděleno více identifikačních čísel a automobil by potom byl i vícekrát sledován a klasifikován.

Tato chyba (Obrázek 3.6) je odstraněna tak, že po získání seznamu detekovaných vozidel v jednom snímku, systém tento seznam projde znovu. V případě, že je nalezen jeden nebo více bouding boxů uvnitř většího bouding boxu, jsou tyto vnitřní bounding boxy ze seznamu detekcí odstraněny. A protože při vícenásobné detekci nemusí být vždy bounding boxy zcela uvnitř největšího, ale mohou být pouze z části, je zde implementována druhá kontrola, která opět odstraní bounding boxy, které jsou ve velmi blízké sousednosti tak, aby zůstal pouze jeden.



Obrázek 3.6: Ukázka dvojité detekce, kde byl automobil značky Peugeot detekován dvakrát v rámci jednoho snímku.

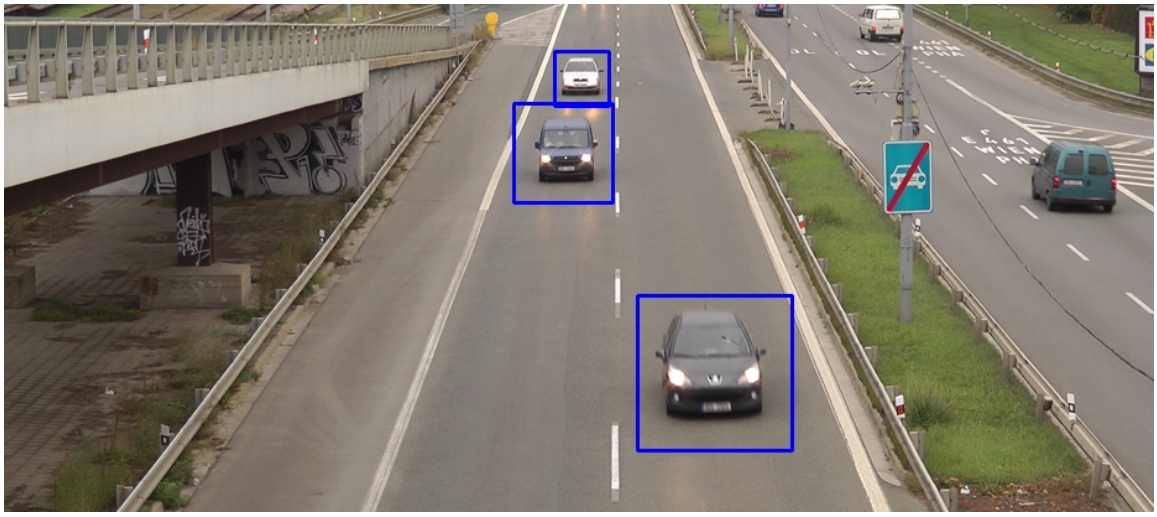
Poslední problém, kterým je nedetekování vozidla v aktuálním snímku, vyřeší *node* pro sledování automobilů. Pokud chybí aktuální informace (bounding box) ke konkrétnímu automobilu, který už je sledován, tento *node* polohu dopočítá. Činnost sledování automobilů je podrobněji popsána v následujícím textu o sledování automobilů.

Po odstranění chyb, které byly popsány v předchozích odstavcích, se dá předpokládat správná činnost detektoru. Správně pracující detektor je znázorněn na obrázku 3.7.

K nedetekování vozidla může dojít i jinak, než chybou detektoru. Nejčastější příčinou bývá zastínění hledaného objektu, tedy v případě, že je mezi objektem a kamerou další předmět. Může to být jiné vozidlo, sloup nebo například zábradlí. Tato chyba vzniká tím, že detektor neumí detekovat částečně zastíněné automobily.

Dále bylo potřeba zamezit detekování automobilů, které stojí na krajnici a nezasahují do aktuální dopravní situace, protože by negativně ovlivňovaly výsledky systému. Tuto situaci řeší načtení masky vstupního videa. Použití vstupní masky také výrazně zrychlí činnost detektoru.

Vyhodnocení a experimenty s detekcí jsou podrobněji popsány v kapitole 4. Detektor není závislý na použité kaskádě, proto lze při běhu systému bez problému používat jinou kaskádu (nastavení cesty ke zvolené kaskádě se provádí v konfiguračním souboru).



Obrázek 3.7: Ukázka správné funkčnosti detektoru, kde je každý automobil detekován pouze jednou a nedochází k žádným nahodilým detekcím.

3.7 Sledování automobilů

Dalším blokem navrhovaného systému je blok pro sledování automobilů. Tento blok zajišťuje, aby každý automobil měl přiřazeno právě jedno ID a jednu trajektorii.

Pro sledování automobilů jsem jako vhodnou metodu vyhodnotil Kalmanův filtr [3, 14, 46]. Kalmanův filtr byl zvolen proto, že není výpočetně náročný a navíc je tato metoda podporována výše zmíněnou knihovnou *OpenCV*.

Princip bloku pro sledování automobilů

Bouding boxy detekovaných automobilů jsou postupně ukládány na *topic*, ze kterého se tyto informace dostanou na vstup *nodu* pro sledování vozidel. Ten přijaté informace o polohách automobilů zkopíruje na výstupní *topic* a k jednotlivým bounding boxům a výřezům automobilů přiřadí ID. Přiřazením ID tedy spojuje více detekcí z jednotlivých předchozích snímků a tím sleduje jednotlivé automobily a vytváří trajektorie jejich pohybu.

Node, pomocí kterého je metoda Kalmanův filtr implementována, během své činnosti udržuje informace o aktuálních automobilech na scéně. Při příchodu nových detekcí z aktuálního snímku videa porovná odhady poloh jednotlivých vozidel, které vypočítal, s právě příchozími detekcemi. Pomocí *Hungarian algoritmu* [17], který slouží k párování na základě nejnižší ceny, přiřadí tyto příchozí detekce k jednotlivým automobilům. Pokud by cena byla příliš vysoká (nad stanovený práh), znamená to, že tato detekce je nový automobil, který ještě není sledován. Tracker s danou detekcí nakládá jako s prvním výskytem nového automobilu a vytvoří nový Kalmanův filtr, který bude nově detekované vozidlo po dobu jeho průjezdu sledovaným úsekem sledovat. Po tom, co jsou všechny detekce správně přiřazeny, tracker vypočítá pomocí existujících Kalmanových filtrů odhady nových poloh, které budou v dalším cyklu opět porovnávány s příchozími detekcemi.

Při implementaci tohoto bloku byla kostra trackeru převzata z již existující implementace metody Kalmanův filtr[2] pro využití sledování vozidel. Tato implementace byla určena pro sledování náhodně se pohybujících kuliček v obraze.

[2] https://github.com/Smorodov/Multitarget-tracker

Kostra byla výrazně upravena a rozšířena pro správné použití v navrhovaném systému. Nejprve bylo potřeba implementaci přizpůsobit a upravit pro správnou činnost a funkčnost na platformě ROS. Třída pro ukládání informací o jednotlivých trajektoriích musela být rozšířena o možnost ukládání jednotlivých výřezů vozidel.

Další úpravy byly aplikovány v rámci zpřesnění a odstranění chyb, které nastávaly. Jejich řešení a další úpravy jsou popsány níže.

Dále bylo potřeba správně přizpůsobit parametry, na základě kterých se tato metoda rozhoduje při vytváření jednotlivých trajektorií. Mezi parametry pro správné sledování automobilů patří:

- **dist_thres** (double) – vzdálenost, o kterou se automobil může v rámci dvou po sobě jdoucích snímků posunout jakýmkoliv směrem (vzdálenost mezi detekcemi ve dvou po sobě jdoucích snímcích),
- **max_trace_length** (int) – maximální počet snímků videa, po kterých je automaticky ukončeno sledování automobilu (maximální délka trajektorie),
- **max_allowed_skip_frame** (int) – maximální počet po sobě jdoucích snímků videa, ve kterých nemusí být již sledovaný automobil detekován, aby nebylo jeho sledování zrušeno.

Systém podporuje více formátů vstupního videa, tudíž se může stát, že budou mít spouštěná videa rozdílné rozlišení nebo jiné specifikace. Z tohoto důvodu si uživatel individuálně nastaví tři výše uvedené parametry v konfiguračním souboru. Nejvíce proměnný parametr je *dist_thres*, kde například záleží na rychlosti automobilů. Na dálnici mohou automobily za stejný časový úsek ujet mnohonásobně větší vzdálenost, než automobily ve městě.

Jak již bylo uvedeno dříve, detektor nemusí vždy detekovat automobil na všech snímcích, ve kterých se vyskytuje. A kvůli tomu musel být tracker náležitě upraven, aby se s touto situací uměl vyrovnat a daný automobil sledoval i potom, co vyjede ze zákrytu. Tato situace může nastat například pokud je automobil na chvíli zastíněn jiným předmětem. Automobil je sice stále v obraze, ale díky stínění detektor nepozná, že se jedná o vozidlo. Pokud je překročen počet po sobě jdoucích snímků, na kterých není automobil detekován, tracker automobil přestane sledovat a tím jeho Kalmanův filtr zaniká. Tato situace také nastává, když automobil odjede z prostoru snímaného kamerou, ovšem v tomto případě je konec sledování jednotlivých automobilů žádoucí. Sledování je ukončeno a automobil může být zařazen na konec fronty, ve které jsou uloženy informace potřebné pro klasifikaci jednotlivých dosud neklasifikovaných automobilů.

Parametry se nastavují pro chod systému bez vynechávání snímků (*drop_frame* = 0). Pokud by měly být některé snímky přeskakovány, parametry trackeru jsou automaticky přepočítány, aby odpovídaly aktuální konfiguraci systému.

Odstranění chyb trackeru

Největší chyba, ke které v průběhu testování docházelo, nastávala v hustém silničním provozu, kdy automobily jezdily blízko sebe. V tomto případě docházelo k jevu, kdy byly jedním ID střídavě chybně označovány různé automobily. K této situaci dochází, protože tracker vyhledává nejbližší polohu nalezeného vozidla v následujícím snímku. Může se tedy stát, že bližší poloha, než poloha původně sledovaného automobilu v následujícím snímku, je poloha automobilu jedoucího vedle nebo za sledovaným automobilem, a proto tracker chybně spojí jednou trajektorií dva různé automobily.

Tato situace je vyřešena úpravou implementace Kalmanova filtru tak, že během sledování jednotlivých automobilů systém kontroluje jejich směr v osách X a Y. Z počátečních detekcí je každému automobilu zjištěn směr jízdy. Při přiřazování nových souřadnic potom tracker kontroluje, zda jsou přiřazeny souřadnice, které odpovídají původnímu směru jízdy automobilu. Pokud nově přidělená detekce nepokračuje ve správném směru, je vyzkoušena hodnota předpovězená Kalmanovým filtrem. Pokud předpovězená hodnota představuje pohyb automobilu správným směrem, je mu tato hodnota přiřazena. Jestliže ani tato hodnota není řešením, nebude k trajektorii v tento krok iterace přiřazen žádný nový bod. Díky této kontrole nedochází k velkým výkyvům trajektorie a částečně se zamezí přeskakování jedné trajektorie na více automobilů při hustém provozu (Obrázek 3.8).

Provedené experimenty a dosažené výsledky tohoto bloku jsou podrobněji popsány v kapitole 4.

3.8 Klasifikace automobilů

Poté, co automobil odjede ze scény, je ukončeno jeho sledování a tím jsou pro tento automobil k dispozici všechny informace potřebné pro klasifikaci (ID vozidla a seznam jeho výřezů). Klasifikace je řešena pomocí konvolučních neuronových sítí [15]. Pro jejich implementaci je využíván *Keras*, což je vysokoúrovňová knihovna pro práci s těmito sítěmi. *Keras* jsem vybral z důvodu jeho jednoduché implementace a dobré úspěšnosti dosahovaných výsledků. V tomto systému je pro klasifikaci automobilů použit fakultní natrénovaný model.

Princip bloku pro klasifikaci

Ve chvíli, kdy sledované vozidlo opustí zorné pole kamery, hlavní uzel zkompletuje potřebné informace tohoto automobilu ke klasifikaci (ID vozidla a seznam jeho výřezů) z *topicu*, na který zapisuje tracker. Následně jsou tyto informace uloženy do fronty. Protože je doba průjezdu a dráha jednotlivých automobilů ve videu proměnlivá, je počet snímků (výřezů automobilů) zredukován maximálně na deset (hodnotu lze přenastavit v konfiguračním souboru) tak, aby vybrané snímky reprezentovaly celou sledovanou dráhu vozidla. Zároveň jsou snímky vynechávány také proto, že by bylo zbytečné klasifikovat výřezy ze dvou po sobě jdoucích snímků videa, kdy se obvykle vlastnosti výřezů téměř nezmění. Systém tedy postupně vynechává předem určený počet snímků (minimálně čtyři – opět lze přenastavit v konfiguračním souboru) ze seznamu výřezů a do redukovaného seznamu ukládá například každý pátý snímek. Pokud se automobil objeví v záběru kamery pouze na krátkou chvíli (je pořízeno méně než pět jeho výřezů), jsou pro klasifikaci použity výřezy dva a to první a poslední. Po redukci seznamu výřezů je tento seznam poslán do klasifikačního *nodu*. Vždy se jedná o seznam pouze jediného automobilu a jednotlivé výřezy jsou postupně posílány na *node*, který je klasifikuje. *Nody* pro klasifikaci komunikují pomocí mechanismu *service*.

Před samotnou klasifikací je důležité každý výřez převést na konstantní velikost, kterou vyžaduje klasifikátor. A protože ROS používá při komunikaci mezi *nody* barevný model BGR a klasifikátor pracuje s modelem RGB, je tedy nutné získané výřezy převést z BGR na model RGB. Klasifikátor na vstupu očekává 4D matici ve tvaru [N, width, height, M], kde **N** je počet snímků, které se posílají na vstup klasifikátoru současně, **width, height** je šířka a výška jednotlivých obrázků (klasifikátor očekává 224 x 224) a **M** je počet barevných kanálů (zde RGB). V navrhovaném systému se vždy na vstup klasifikátoru posílají výřezy jednotlivě, tedy ve 4D matici s rozměry [1, 224, 224, 3]. Po vytvoření je tato matice poslána ke klasifikaci pomocí funkce *predict_on_batch (x)*, kde parametr x je právě tato 4D matice.

Obrázek 3.8: Na horním obrázku běží tracker bez kontroly směru a je vidět, že trajektorie přeskočila z levého osobního automobilu na pravé (žlutá část trajektorie). Také jsou na tomto screenu zřetelné výkyvy trajektorie. Zatímco na spodním obrázku, kde jsou vozidla sledována za pomocí kontroly směru, je zobrazena správná činnost trackeru. Nedochází zde k žádnému přeskakování a každý automobil má správně přiřazenou jednu trajektorii, která neobsahuje větší výkyvy.

Funkce *predict_on_batch ()* se nachází v knihovně *Keras*. Klasifikátor na základě natrénovaného modelu přiřadí pravděpodobnost shody ke každé třídě z natrénovaného modelu.

Způsoby vyhodnocení výsledné třídy jednotlivých automobilů

Z pole, které obsahuje jednotlivé pravděpodobnosti pro každou třídu, lze výslednou vítěznou třídu pro aktuální vozidlo získat více způsoby. V tomto systému jsou implementovány dva z nich.

První způsob je ten, že výsledná třída každého výřezu je ukládána do pole a po skončení klasifikace všech výřezů určitého automobilu je na toto pole vítězných tříd aplikován *modus*. Nejčastější hodnota, tedy vítězná třída, je poslána zpět hlavnímu uzlu jako výsledek. Tento postup se opakuje jednotlivě pro všechny automobily, které byly detekovány v analyzovaném video souboru.

Druhý způsob získává vítěznou třídu tak, že se pro každý výřez ukládá pole se všemi pravděpodobnostmi u všech tříd modelu. Po klasifikování posledního výřezu aktuálně zpracovávaného vozidla, jsou pravděpodobnosti získané v jednotlivých iteracích sečteny a třída s největší hodnotou pravděpodobnosti je vybrána jako vítězná a stejně jako v prvním případě poslána hlavnímu uzlu, který ji uloží do třídy *Car*.

Kterým způsobem se bude klasifikace vyhodnocovat si uživatel nastavuje v konfiguračním souboru. Úspěšnost těchto dvou způsobů pro určení nejpravděpodobnější třídy je detailně popsána v kapitole 4.

Popis modelu pro klasifikaci

Natrénovaný model pro klasifikaci je stejně jako kaskáda pro detektor k dispozici na Fakultě informačních technologií. Model je ve formátu H5. Použitý model je rozdělen do 107 tříd, kde každá třída reprezentuje konkrétní typy modelů jednotlivých továrních značek automobilů. Index třídy s největší hodnotou je i nejpravděpodobnějším výsledkem. Nejvyšší hodnoty se pohybují nejčastěji v intervalu 80 – 95 % a ostatní třídy mají pravděpodobnost v rozsahu desetin procent. Model je zaměřen na automobily nejčastěji se vyskytující v České republice, proto mají největší zastoupení automobily značky Škoda, Volkswagen nebo Ford, kdy u natrénovaných značek automobilů rozeznává rozdíl mezi verzemi combi, hatchback nebo sedan. Dále je klasifikátor pomocí tohoto modelu schopen rozpoznat například u automobilů značky Škoda první, druhou nebo třetí generaci.

Model je natrénován na fakultním datasetu *BoxCars116k*. Tento dataset obsahuje 116 000 snímků automobilů zabraných z různých kamer a úhlů [41].

Přesné výsledky testování jsou popsány a zhodnoceny v kapitole 4.

3.9 Logování výsledků

Posledním celkem blokového schématu (Obrázek 3.1) je logování. Tento blok jako jediný nepředstavuje samostatný *node*, ale stará se o něj hlavní uzel, který má k dispozici všechny informace o běhu systému.

Po skončení klasifikace zbývá uložit dosažené výsledky do jednotlivých logovacích souborů, které jsou ve formátu JSON.

Logování detekce probíhá při běhu systému. Po příchodu seznamu detekcí pro aktuální snímek na *topic* je tento seznam uložen v již popsaném formátu. Logovací funkce pro sledování a klasifikaci automobilů jsou volány z destruktoru třídy *Car*. Po zalogování už nejsou informace o dříve sledovaných automobilech potřeba, proto jsou smazány. Do těchto dvou logovacích souborů jsou tedy postupně zapisovány informace o projetých vozidlech.

Všechny potřebné informace pro logování jsou získány ze třídy *Car*. Jelikož informace o výsledku klasifikace (vítězná třída) je v této třídě uložena jako číselný údaj (*int*), může se toto číslo (identifikátor třídy klasifikátoru) pomocí připravené funkce *classToName ()* převést na *string*, který reprezentuje tovární značku a model vozidla.

Mimo logovacího souboru jsou do vybrané složky uloženy výřezy všech vozidel, která se objevila v záběru vstupního videa. Snímky jednotlivých automobilů jsou roztříděny do systémem vytvořených podsložek. Tyto podsložky jsou pojmenovány podle ID automobilů a jednotlivé výřezy jsou vždy pojmenovány číslicemi vzestupně od nuly. Výřezy se ukládají do třídy *Car*. Jakmile automobil opustí zorné pole kamery, z výřezů se vyberou vhodní kandidáti na klasifikaci a ostatní výřezy jsou ze třídy smazány. Před smazáním ze třídy jsou všechny výřezy uloženy na disk. Velké množství výřezů (ty, které nejsou určeny ke klasifikaci) je tedy rychle mazáno pro efektivnější práci s pamětí.

Cestu k umístění logovacího souboru a k podsložkám s výřezy jednotlivých automobilů si uživatel libovolně nastaví pomocí konfiguračního souboru.

Kapitola 4

Experimenty a vyhodnocení

Tato kapitola popisuje všechny provedené experimenty a shrnuje dosažené výsledky pro jednotlivé části. Experimenty byly rozděleny do tří hlavních kategorií a to detekci, sledování a klasifikaci.

Výsledný systém byl testován na souboru fakultních videí nazvaném *BrnoCompSpeed* [40]. Byla použita videa z prvních pěti setů. Tato videa jsou pořízena na různých čtyřproudových dopravních komunikacích, kde jsou sledovány automobily jedoucí ve dvou pruzích směrem ke kameře. Každý set obsahuje tři videa, kde je kamera umístěna z levé nebo pravé strany vozovky a nebo přímo nad ní. Většina videí má hodnotu FPS = 50. Video záznamy jsou dlouhé přibližně jednu hodinu a rozlišení mají 1920x1080. Ukázky testovacích videí jsou znázorněny na obrázcích 4.1 – 4.5.

Systém byl vyvíjen a následně testován na notebooku Lenovo ThinkPad E530c s procesorem Intel Core i5 2520M Sandy Bridge a operační pamětí 4GB DDR3 800 MHz. Na tomto počítači byl nainstalován operační systém Linux Ubuntu 16.04.

Po úspěšném dokončení implementace systému jsem nejprve vytvořil skripty pro porovnání skutečných hodnot s hodnotami, které systém během svého běhu uložil do logovacích souborů. Dále bylo potřeba pro tento testovací dataset vytvořit anotace pro detekci, sledování a klasifikaci automobilů. Vytvořené anotace jsou k dispozici na přiloženém DVD.

Pro jednotlivé testy byly implementovány skripty v programovacím jazyce Python. Skripty jsou robustní a připravené na odlišné testy s různými kritérii. Ke každému skriptu existuje soubor *config.py*, kde si uživatel nastaví veškeré potřebné parametry a cesty k testovaným anotacím, respektive logovacím souborům. Principy skriptů jsou popsány v dalších podkapitolách věnovaných vyhodnocení výsledků bloků pro detekci, sledování a klasifikaci automobilů.

Obrázek 4.1: Ukázka Session1.

Obrázek 4.2: Ukázka Session2.

Obrázek 4.3: Ukázka Session3.

Obrázek 4.4: Ukázka Session4.

Obrázek 4.5: Ukázka Session5.

Popis vytvořených anotací

K vytvoření anotací pro detekční část systému jsem využil program *labelIMG* (pro jeho jednoduchou instalaci jsem opět použil dříve zmíněný mechanismus *Docker*). Pomocí tohoto programu jsem v každém snímku videa označil právě projíždějící vozidla obdélníkem, který reprezentuje bounding box. Program pro každý anotovaný snímek vytvoří XML soubor, kde jsou uloženy hodnoty jednotlivých bounding boxů (osy X a Y, šířka a výška). Aby byly skutečné anotace ve stejném formátu jako výsledné logovací soubory, vytvořil jsem skript (*xml_to_json.py*), který umí převést všechny XML soubory jednoho videa do jednoho souboru ve formátu JSON. Struktura souboru je ve formě: pořadové číslo snímku, osa X, osa Y, šířka a výška. Pro každé video tedy vznikl jeden soubor se skutečnými polohami bouding boxů projetých automobilů.

Vytvořené anotace se dají použít pouze pro vyhodnocení úspěšnosti detektoru. Proto je potřeba následně tyto anotace převést na anotace použitelné pro vyhodnocení činnosti bloku pro sledování automobilů. K této transformaci mi pomohl mnou vytvořený skript (*detection_to_tracking.py*). Anotace pro tracking jsou opět ve formátu JSON a ve formě:

ID automobilu, snímek videa, na kterém začíná trajektorie, pole hodnot X a Y. Hodnoty X a Y určují střed bouding boxů jednotlivých vozidel. Se středy bounding boxů se pracuje proto, aby výsledné trajektorie vycházely ze středu automobilu a ne z levého horního rohu bounding boxu.

Anotace pro klasifikaci vycházejí z anotací pro sledování vozidel. K těmto anotacím je doplněna třída, která reprezentuje značku a typ daného vozidla. Protože je systém vytvářen na sledování automobilů zepředu, je v některých případech těžké rozpoznat, jestli se jedná o hatchback, sedan nebo combi. Například jestli projela Škoda Fabia hatchback nebo Škoda Fabia combi, nebyl klasifikátor schopen rozpoznat. Ovšem správně určil, že se jedná o automobil Škoda Fabia. Klasifikátor ovšem označil jednu z těchto možností největší pravděpodobností a pokud by označil právě druhou možnost, skript pro vyhodnocení by určil výsledek klasifikace tohoto vozu jako negativní, což není správné rozhodnutí. Proto byla do anotací přidána další informace a tou je alternativní třída. Alternativní třída slouží právě pro výše zmíněný případ, kdy pomocí této třídy zjistí vyhodnocující skript, že klasifikátor správně určil, že se jedná o automobil Škoda Fabia. Zda se jedná o hatchback nebo combi už neřeší, protože tato informace v tomto případě není důležitá.

Pokud daný automobil nepatří do skupiny automobilů, na které byl používaný model pro klasifikaci natrénován, je označen třídou 107. Pokud k automobilu neexistuje alternativní třída je v kolonce pro tuto hodnotu uvedena třída 108.

Vytvoření anotací pro celá videa by bylo z důvodu jejich vysoké hodnoty FPS velmi časově náročné. Byly tedy vytvořeny anotace pouze pro jejich části, což dle mého názoru na testování funkčnosti celého systému postačuje. Ke každému videu jsou k dispozici anotace pro prvních 7500 snímků a to jak pro detekci a sledování, tak pro klasifikaci automobilů. Celkově jsem vytvořil anotace pro 627 automobilů z různých situací, které těchto patnáct videí nabízí.

4.1 Detekce automobilů

Pro detekci a získání bouding boxů projíždějících vozidel z jednotlivých snímků videa se volá funkce *detectMultiScale ()*, kde je potřeba správně nastavit dva parametry a to *scale_factor* a *min_neighbors*.

Parametr *scale_factor* se ve studovaných systémech pohyboval v intervalu 1.0 – 1.2. Při těchto hodnotách je detekce dostatečně důkladná. Vyšší hodnoty tohoto parametru zajistí detektoru vyšší rychlost, ale úspěšnost detektoru při detekci vozidel je v tomto případě nižší.

Druhý parametr funkce pro detekci *min_neighbors* je třeba nastavit na vyšší hodnotu než je nula. Výsledek při nastavení parametru na hodnotu nula je zobrazen na obrázku 3.5, kde je na první pohled zřejmé, že takto nastavený detektor nelze použít.

Pro správnou a co nejpřesnější detekci byla provedena řada experimentů, při kterých bylo vyzkoušeno více kombinací hodnot těchto parametrů na všech testovaných videích. K nalezení nejlepší kombinace těchto parametrů, testování a vyhodnocení výsledků detektoru byly použity následující metodiky.

PRC křivka

PRC (z angl. Precision Recall Curve) [8, 10] je křivka generovaná pro grafické zobrazení vyhodnocení výkonu detektoru na daném datasetu. Při vyhodnocování výsledků mohou nastat tyto situace:

- **TP** (true positive) – vyjadřuje počet správně detekovaných automobilů.
- **TN** (true negative) – vyjadřuje počet správně nedetekovaných automobilů, tedy počet kandidátů (objektů), které detektor správně neoznačil za automobil.
- **FP** (false positive) – vyjadřuje počet chybně detekovaných automobilů, tedy objektů které byly chybně označeny za automobil.
- **FN** (false negative) – vyjadřuje počet chybně nedetekovaných automobilů, tedy těch, které nebyly detekovány.

Skript je při porovnání skutečných a předpovězených bounding boxů schopen zjistit pouze hodnoty **TP**, **FP** a **FN**. Proto je zde použita PRC křivka, kde jsou potřeba k sestrojení grafu právě tyto tři hodnoty. Na osy křivky se vynášejí zjištěné hodnoty *Precision* a *Recall*.

Precision je definován jako podíl počtu správně nalezených objektů vůči celkovému počtu nalezených objektů. Výpočet je znázorněn vztahem 4.1 a vyjadřuje tedy přesnost daného výsledku.

$$precision = \frac{TP}{TP + FP} \tag{4.1}$$

Recall potom říká, kolik bylo správně nalezeno objektů z celkového počtu hledaných objektů. Je definován (Vztah 4.2) jako podíl počtu správně nalezených objektů a celkový počet hledaných objektů.

$$recall = \frac{TP}{TP + FN} \tag{4.2}$$

Ze souřadnic *precision* a *recall*, pomocí kterých se zaznamenávají jednotlivé body do grafu, lze vypočítat hodnotu F (Vztah 4.3). Bod grafu, ve kterém dosáhne F nejvyšší hodnotu je považován za nejlepší. Zároveň tento bod reprezentuje nejlepší nastavení detektoru.

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4.3}$$

Metrika IoU

IoU (z angl. Intersection over Union) [32, 49] je vyhodnocovací metrika používaná k měření přesnosti detektoru na konkrétní datové množině. Pomocí této metriky lze tedy vypočítat, jak moc je bounding box získaný systémem identický se skutečným bounding boxem. Výsledkem je hodnota v intervalu od nuly do jedné, kde jednička označuje stoprocentní shodu porovnávaných obdélníků. Výpočet je zobrazen pomocí vztahu 4.4 a na obrázku 4.6 jsou ilustrovány definované oblasti pro tento výpočet.

$$IoU = \frac{oblast\ překrytí\ (průnik\ bounding\ boxů)}{oblast\ spojení\ (sjednocení\ bounding\ boxů)} \tag{4.4}$$

[1] Obrázek 4.6 je převzat z [32]

Obrázek 4.6: Na levém obrázku je zobrazena oblast překrytí a na pravém oblast spojení. [1]

Úspěšnost klasifikace v závislosti na hodnotě IoU

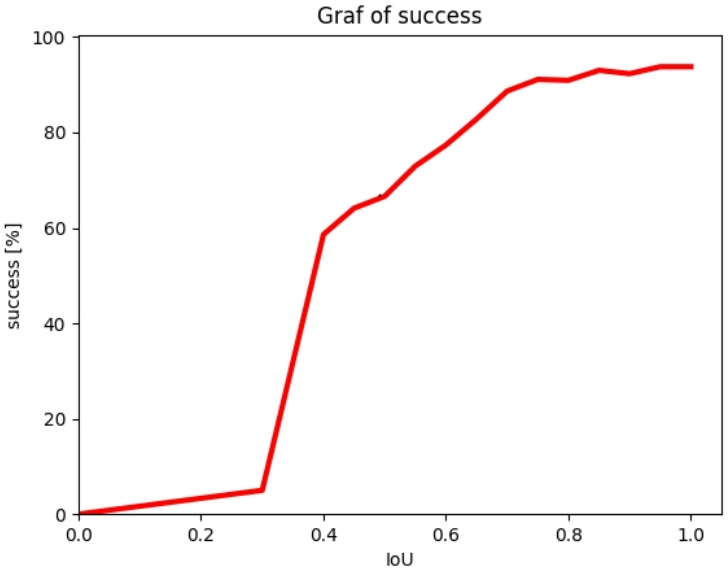
Pro účel testování úspěšnosti klasifikace v závislosti na hodnotě IoU byl vytvořen skript (*eval_class_iou.py*), který se pokouší klasifikovat 32 automobilů. Skript vezme na začátku snímek automobilu (IoU = 1, při této hodnotě dosahovala klasifikace úspěšnosti kolem 95 %) a náhodně ho posunuje tak, aby vytvořil snímky s různou hodnotou IoU. Hodnoty spadají postupně do intervalů 0.4 – 0.45 až 0.95 – 1.0. Velikost intervalů je nastavena na 0.05. Pro každý automobil je vytvořeno 100 nových snímků pro každý interval. Tyto snímky jsou klasifikovány a podle výsledků je vypočtena procentuální úspěšnost klasifikace. Následně je pro jednotlivé intervaly pomocí průměru získána úspěšnost pro všech 32 automobilů. Získané výsledky jsou zaneseny v tabulce 4.1 a výsledný graf je zobrazen na obrázku 4.7.

Tabulka 4.1: Tabulka znázorňující úspěšnost klasifikace v závislosti na velikosti hodnoty IoU.

Interval hodnoty IoU	Průměrná úspěšnost
0.40 – 0.45	58.62 %
0.45 – 0.50	64.13 %
0.50 – 0.55	66.62 %
0.55 – 0.60	72.96 %
0.60 – 0.65	77.37 %
0.65 – 0.70	82.82 %
0.70 – 0.75	88.68 %
0.75 – 0.80	91.12 %
0.80 – 0.85	90.94 %
0.85 – 0.90	93.03 %
0.90 – 0.95	92.34 %
0.95 – 1.00	93.79 %

Vyhodnocení detektoru na datasetu BrnoCompSpeed

Pro vyhodnocení správné činnosti detektoru jsem implementoval skript (*eval_detection.py*), který porovnává skutečné hodnoty s hodnotami předpovězenými systémem. Skript bere postupně jednotlivé snímky originálních anotací, na kterých se vyskytl alespoň jeden automobil a pomocí pořadového čísla snímku nalezne odpovídající hodnoty, které pro tento snímek předpověděl vyvinutý systém. Následně zjistí počet skutečných a předpovězených bounding boxů. Podle zjištěného počtu může nastat situace, kdy je více skutečných bounding boxů, tím pádem se o tento počet zvedá hodnota FN nebo naopak pokud je více předpovězených bounding boxů, zvedá se o tento počet hodnota proměnné FP.



Obrázek 4.7: Na ose Y je zobrazena úspěšnost klasifikace [%] závislá na hodnotě metriky IoU na ose X [-]. Z grafu je tedy patrné, že například při minimální hodnotě IoU = 0.6, by se měla úspěšnost klasifikace v ideálním případě pohybovat mezi 79 – 100 %. Ideálním případem je myšleno, že každý detekovaný snímek bude mít při úplné shodě (IoU = 1) téměř 100 % úspěšnost klasifikace. Graf je vytvořen přímo skriptem *eval_class_iou.py* implementovaném v jazyce Python.

Skript dále k předpovězeným bounding boxům hledá nejvíce odpovídající skutečný bounding box. Velikosti latence, o kolik se polohy bounding boxů mohou lišit, jsou počítány pomocí velikosti skutečných bounding boxů a nastaveného poměru.

Když je nalezen odpovídající skutečný bounding box, jsou oba porovnány pomocí metriky IoU. Pokud je hodnota IoU větší nebo rovna nastavené minimální povolené hodnotě IoU, je předpovězený bounding box označen za správný a je inkrementována hodnota TP. Jestliže je hodnota nižší než tato nastavená hodnota, je inkrementována hodnota FP.

Skript pro vyhodnocení úspěšnosti detektoru znázorňuje jako výsledek počet TP, FP a FN. Pro testování jsou k dispozici výsledky systému, který byl spuštěn s parametrem pro vynechávání snímků *drop_frame* = 0. Systém tedy zpracoval všechny snímky vstupního videa. Výsledky jednotlivých videí jsou zapsány v tabulce 4.2 a výsledná PRC křivka znázorněna na obrázku 4.8.

Vyhodnocení detektoru na datasetu COD20K

Kromě testovaného datasetu (*BrnoCompSpeed*) jsem provedl vyhodnocení pomocí stejného skriptu (se stejnými parametry) i na datasetu *COD20K*, na které byla trénována kaskáda pro detekci. Dataset *COD20K* obsahoval anotace v jiné formě, než kterou je skript pro vyhodnocení schopen zpracovat. Proto musely být skutečné anotace převedeny pomocí skriptu *txt_to_json.py* do formátu JSON. Do anotací jsou započítána pouze vozidla, která byla součástí sledovaných jízdních pruhů. Automobily, které stály u silnic například na parkovištích, byly z anotací odstraněny. Snímky byly spouštěny se vstupní maskou, která definovala detektoru hledanou vybranou oblast, tedy pouze jízdní pruhy.

Tabulka 4.2: Tabulka s výsledky experimentů detekce provedených na jednotlivých videích. Minimální hodnota metriky IoU při porovnání skutečného a předpovězeného bounding boxu musí být vyšší nebo rovna hodnotě 0.45, aby byl předpovězený bounding box označen za TP. Tato minimální hranice byla nastavena na 0.45, protože už při této hodnotě dosahovala klasifikace dobrých výsledků (Graf 4.7). Předpovězené bounding boxy jsou získány pomocí detektoru s nejlepším nastavením parametrů. Nejlepší nastavení bylo určeno pomocí PRC křivky (Graf 4.8).

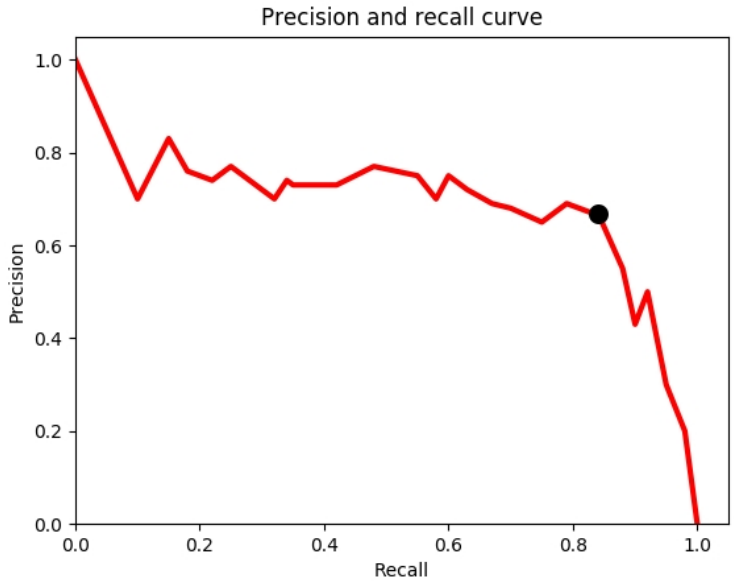
Název videa	TP	FP	FN
Session1_center	1 555	94	77
Session1_left	1 583	580	153
Session1_right	1 262	296	405
Session2_center	3 228	536	346
Session2_left	1 991	670	731
Session2_right	1 754	611	481
Session3_center	357	134	17
Session3_left	426	336	50
Session3_right	655	461	176
Session4_center	2 866	699	147
Session4_left	2 971	1 760	277
Session4_right	1 450	2 030	117
Session5_center	3 031	1 195	251
Session5_left	2 663	1 396	1 040
Session5_right	1 566	2 491	615
Shrnutí – center	11 037	2 658	838
Shrnutí – left	9 634	4 742	2 251
Shrnutí – right	6 687	5 889	1 794
Shrnutí	27 358	13 289	4 883

Jelikož tento dataset neobsahuje video soubory, ale pouze snímky videí, není zde potřeba blok pro čtení videa. Pro získání předpovězených bounding boxů použitým detektorem byl použit speciální program, který spouští jednotlivé snímky videa ve smyčce a postupně na nich hledá automobily. Zdrojové soubory k tomuto programu jsou také k dispozici na přiloženém DVD.

Výsledky testu na datasetu *COD20K* jsou zaznamenány v tabulce 4.3.

Tabulka 4.3: Tabulka s výsledky experimentů s detekcí provedených na datasetu *COD20K*, na kterém byla natrénována kaskáda pro detektor. Výsledky v tabulce jsou zobrazeny po porovnání s metrikou IoU nastavenou na minimální hodnotu 0.45.

TP	FP	FN
1302	567	602



Obrázek 4.8: Jako nejlepší bod byl zvolen bod s hodnotami $recall = 0.84$ a $precision = 0.67$. Kdy $F = 0.75$, což byla nejvyšší získaná hodnota. Parametry popsané výše byly nastaveny takto: $factor\_scale = 1.1$ a $min\_neighbors = 4$.

Shrnutí experimentů

Výsledky testování detektoru jsou zobrazeny v tabulce 4.2. Z důvodu velkého množství testovaných kombinací nastavení parametrů (prahu detektoru) je zde uvedena pouze tabulka s nejlepšími výsledky. Tyto výsledky byly pořízeny při následujícím nastavení parametrů: $scale\_factor = 1.1$ a $min\_neighbors = 4$, které byly vybrány pomocí PRC křivky.

Úspěšnost detektoru snižovala zejména velká vozidla (nákladní vozidla, kamiony), proto je úspěšnost lehce proměnlivá v závislosti na počtu výskytů těchto vozidel v daném testovaném úseku videa. Jak je vidět z tabulky, pro každé video se výsledky detektoru mění v závislosti na konkrétních podmínkách dopravních situací v testovacích videích.

Nejlepší výsledky dosahoval detektor na videích **center** a naopak nejhorší na videích **right**, kde úspěšnost nejvíce negativně ovlivňují videa **Session4_right** a **Session5_right**.

4.2 Sledování automobilů

Pro vyhodnocení úspěšnosti sledování automobilů jsem se inspiroval metrikou *MOT16*.

MOT16

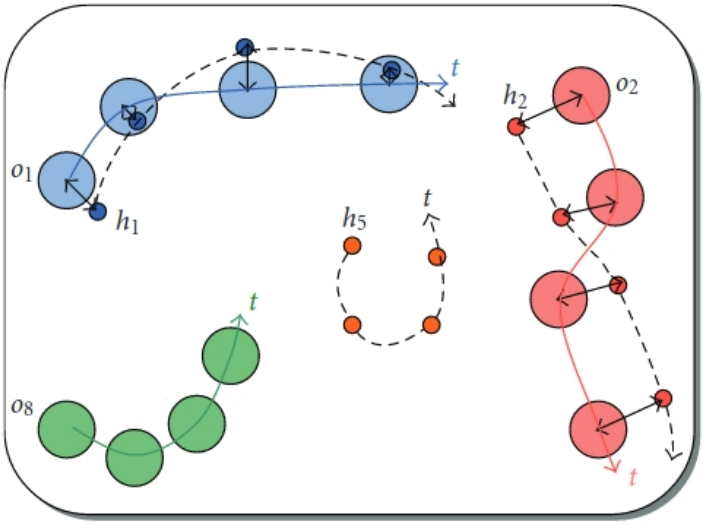
MOT (z angl. Multi Object Tracking) [4, 22] je metrika určená pro vyhodnocení sledování různých objektů. Pomocí této metriky je skript pro vyhodnocení trackeru schopen určit, zda je předpovězená trajektorie správná, tedy odpovídá skutečné (TP), nebo zda je předpovězená trajektorie falešný poplach (FP). Tento rozhodovací proces se obvykle provádí na základě prahových hodnot založených na definované vzdálenosti d. Vzdálenost reprezentuje největší možnou odchylku mezi body předpovězené a skutečné trajektorie na stejném snímku, aby byly body považovány za shodné. Pokud není v aktuálně zpracovávaném

snímku videa nalezen bod předpovězené trajektorie, znamená to, že pro aktuální snímek nebyla zaznamenána detekce (FN). Pro dobrý výsledek a tedy k prohlášení, že předpovězená trajektorie odpovídá skutečné, se očekává co nejmenší počet FP a FN.

Podobně jako jednotlivé body trajektorií jsou zařazovány do kategorií také trajektorie jako celek. Pro vyhodnocení lze tedy opět využít základní kategorie:

- **TP** (true positive) – vyjadřuje počet trajektorií automobilů, které byly vytvořeny správně.
- **FP** (false positive) – vyjadřuje počet trajektorií automobilů, které byly vygenerovány špatně.
- **FN** (false negative) – vyjadřuje počet trajektorií automobilů, které nebyly systémem vytvořeny.

Toto rozdělení je znázorněno a popsáno na obrázku 4.9.



Obrázek 4.9: Na obrázku plné šipky označené písmenem o (objekt) znázorňují skutečné trajektorie a přerušované šipky označené písmenem h (hypotéza) trajektorie předpovězené. Pomocí modrých bodů je naznačena trajektorie, která byla označena za správnou (TP), neboť některé body jsou téměř identické a některé se liší o malou hodnotu, která je menší než definovaná vzdálenost d. V případě červených bodů už není shoda taková a na zařazení této trajektorie bude mít vliv velikost parametru maximální vzdálenosti d. Řekněme, že hodnota d je nízká a tři ze čtyř bodů nespadají do povolené tolerance. Trajektorie tedy bude označena jako FP. Do kategorie FP bude také zařazena trajektorie $h5$ (oranžové body), která v nejbližším okolí nemá skutečnou trajektorii, se kterou by mohla být porovnána. Do kategorie FN patří poslední situace z obrázku (zelený objekt). Zde je, na rozdíl od předchozího případu, pouze skutečná trajektorie, ke které není vhodný kandidát z předpovězených trajektorií.

[2]

[2]Obrázek 4.9 je převzat z [4]

Pro vyhodnocení *nodu* pro sledování automobilů jsem implementoval skript pojmenovaný *eval_tracking.py*, který porovnává skutečné anotace s výsledky, které systém získal během zpracování videa a uložil do logovacích souborů.

Skript pracuje na podobném principu jako skript pro vyhodnocení detekce a vyhodnocení určuje podle principu metriky MOT16. Nejprve zjistí počet skutečných a předpovězených trajektorií a poté případně zjistí, kolik předpovězených trajektorií chybí. Postupně prochází odpovídající snímky skutečných a předpovězených hodnot, kdy si vždy pro každý snímek zjistí počet trajektorií, které na tomto snímku začínají. Pokud je více předpovězených trajektorií, tak o tolik se zvedne hodnota falešných trajektorií (FP). K analyzované trajektorii je nalezena ta nejbližší a následně je nalezen společný bod obou trajektorií. Od tohoto bodu jsou tyto dvě trajektorie porovnány. Porovnání probíhá podle atributů, které byly popsány výše v textu o metrice MOT16. Na základě tohoto porovnání je předpovězená trajektorie při splnění parametrů označená jako TP, v opačném případě jako FP. Ve skriptu je implementována kontrola délky předpovězené trajektorie. Pokud je kratší, než je nastavená minimální hodnota, je tato trajektorie automaticky označena jako FP. Po zanalyzování vstupních souborů skript vypíše souhrnné výsledky úspěšnosti pro jednotlivá videa.

Hodnota vzdálenosti d, tedy maximální vzdálenost, o kterou se mohou body lišit je nastavena opět v konfiguračním souboru. Rozdílná je také hodnota při hledání společného počátečního bodu trajektorií. Tato počáteční hodnota je menší, neboť trajektorie začínají ve vzdálenějším horním rohu. Konce trajektorií jsou blízko kamery, tedy automobily jsou zde větší a může být mezi vzdálenostmi větší rozdíl. Vzdálenost d se nastavuje zvlášť pro souřadnice X a Y. U souřadnice X se také rozlišuje druh videa. U typu **center** je maximální vzdálenost menší než u videí **left** nebo **right**.

Procentuální hodnota poměru mezi jednotlivými body trajektorie označenými jako TP a FP, aby byla vygenerovaná trajektorie považována za správnou, je další parametr v konfiguračním souboru. Udává se zde maximální odchylka. Tedy pokud je požadovaný poměr 80:20 – nastaví se tato proměnná na hodnotu 20. Tento poměr byl nastaven při testování navrženého systému.

Dále může uživatel specifikovat vyhodnocující skript nastavením minimálního počtu bodů označených za TP, aby byla trajektorie vůbec považována za kandidáta na TP. Další nastavitelná proměnná vytváří zarážku ve vyhodnocení aktuální trajektorie a tedy i zrychlí vyhodnocení. Zarážka je aktivována, pokud je určený počet za sebou jdoucích bodů označených jako FN. Pokud dojde k zarážce, je trajektorie zkontrolována, jestli není příliš krátká a pokud ne, je vyhodnocena jako ostatní trajektorie, u kterých nebyla zarážka aktivována.

V tabulce 4.4 jsou zobrazeny výsledky jednotlivých videí.

Shrnutí experimentů

Z tabulek je zřejmé, že při slabém provozu (**Session_3**) je úspěšnost Kalmanova filtru vyšší než při husté dopravě, kdy tracker dosahuje horších výsledků.

V hustém provozu může nastat situace, kdy je sledovaný automobil na určitou dobu zastíněn. Tím jsou místo jedné trajektorie získány dvě, které jsou sice správné, ale skript je vyhodnotí jako špatné, neboť jsou krátké. Pro zlepšení úspěšnosti sledování je možné implementovat funkci, která by takto přerušené trajektorie detekovala, zpětně spojila a zapsala do logovacího souboru jako jednu trajektorii. Takto spojenou trajektorii už by systém mohl vyhodnotit jako správnou.

Tabulka 4.4: Tabulka znázorňující získané výsledky bloku pro sledování automobilů, kde $\sum$ značí skutečný počet projetých automobilů.

Název videa	$\sum$	TP	FP	FN
Session1_center	27	25	2	0
Session1_left	31	24	8	1
Session1_right	27	21	6	0
Session2_center	50	35	17	4
Session2_left	42	30	8	4
Session2_right	43	28	12	5
Session3_center	6	6	0	0
Session3_left	6	6	0	0
Session3_right	9	7	2	0
Session4_center	56	49	7	1
Session4_left	55	51	5	2
Session4_right	43	24	20	11
Session5_center	78	74	5	1
Session5_left	73	64	11	3
Session5_right	81	35	59	13
Shrnutí – center	217	189	31	6
Shrnutí – left	207	175	32	10
Shrnutí – right	203	115	99	29
Shrnutí	627	479	162	45

Další chybou v hustém provozu je přiřazování jednoho ID více vozidlům (přeskakování trajektorie z jednoho automobilu na druhý). Častěji tato chyba vzniká u záznamů ze strany (zejména u videí **right**). K tomu dochází proto, že v těchto videích se zvětšují vzdálenosti mezi jednotlivými detekcemi sledovaného automobilu. Na těchto záznamech automobil urazí větší vzdálenost za stejný čas, než na záznamech pořízených kamerou, která snímá automobily výhradně zepředu (**center**). Problémy se spojením dvou vozidel do jednoho ID nastávají, když dva automobily jedou blízko sebe ve stejném směru jak v ose X tak v ose Y, tedy ani kontrola směru tuto chybu nemusí detekovat. Dalším důvodem je také to, že se v těchto videích automobily vzájemně více zastiňují. Tento úkaz je také zřetelný z tabulky, kdy nejvyšší úspěšnost sledování byla prokázána u videí s označením **center**.

Celkem bylo anotováno 627 automobilů a z toho bylo 479 označeno správnou trajektorií. Tracker tedy dosáhl průměrné úspěšnosti 76.40 %. Celkovou úspěšnost nejvíce snižují videa **right**, konkrétně nejvíce video **Session5_right**.

Nejvyšší úspěšnost vykázala videa **center**, kde bylo z 217 automobilů správně sledováno 189. O trochu nižší úspěšnost videa **left**, kde projelo 207 automobilů a 175 bylo správně sledováno. V obou případech se úspěšnost správně sledovaných automobilů vyšplhala nad 84 %. Zatímco videa **right**, kde bylo celkem anotováno 203 automobilů, dosáhla úspěšnost správně sledovaných automobilů pouze 56.65 %. Správně zde bylo sledováno pouze 115 vozidel.

Ve výsledcích detekce a sledování lze najít společné rysy v dosažené úspěšnosti. Pokud jsou získané výsledky detekce horší, lze očekávat i horší výsledky úspěšnosti sledování

vozidel. Nejvíce viditelné je to u videí **Session4_right** a **Session5_right**. Tato videa vykazovala při experimentech nejhorší výsledky.

Jak bylo již uvedeno výše, aby systém pracoval v reálném čase pro testovaná videa musí být vynechávány některé snímky videa. Během testů jsem zjistil, že je systém schopen na testovaném stroji v reálném čase zpracovat videa s hodnotou FPS = 15. Pokud bude systém spuštěn na počítači s lepšími parametry, než je zmíněný, ne příliš graficky výkonný notebook, nebude nutné vynechat tolik snímků nebo žádný snímek a systém bude pracovat v reálném čase.

Pokud by uživatel z různých důvodů při používání navrhovaného systému vynechával určitý počet snímků, je vyhodnocovací skript připraven i na testování sledování při vynechání několika snímků analyzovaného videa. Stačí v konfiguračním souboru přenastavit parametr *drop_frame*.

4.3 Klasifikace automobilů

Pro klasifikaci bylo třeba nejprve určit, kolik výřezů je vhodné posílat na vstup klasifikátoru pro jednotlivé automobily. Dále bylo třeba stanovit minimální vzdálenost výřezů mezi snímky videa. Správné nastavení těchto parametrů zajistí vyšší úspěšnost klasifikátoru a zamezí zbytečně dlouhé době klasifikování jednotlivých automobilů, která by nastala v případě klasifikování všech získaných výřezů pro jednotlivá vozidla. Úkolem tedy bylo zjistit, kolik snímků a s jakým rozestupem je ideální uložit, aby se klasifikovalo co nejméně obrázků při zachování vysoké úspěšnosti.

Po sérii experimentů byla určena jako minimální vzdálenost mezi výřezy čtyři snímky videa, tedy každý pátý výřez se klasifikuje. Při tomto rozestupu se začaly odlišovat pravděpodobnosti jednotlivých tříd modelu. Jako maximální počet výřezů pro každý automobil byla zvolena hodnota deset. Při klasifikaci více než deseti výřezů pro jednotlivé automobily testy v drtivé většině ukázaly, že více výřezů určení výsledné třídy, která představuje tovární značku a typ automobilu, nezmění. Při detekování vozidla na více než padesáti snímcích (deset výřezů, kdy se klasifikuje každý pátý) je mezera mezi výřezy konstantně zvětšena tak, aby bylo klasifikováno deset výřezů a jejich pomocí byla reprezentována celá dráha vozidla. Pokud se automobil neobjevil na více než pěti snímcích vstupního videa, je klasifikován první a poslední výřez tohoto vozidla.

Jak bylo zmíněno v kapitole 3, je systém při klasifikaci jednotlivých automobilů vybaven dvěma módy pro vyhodnocení vítězné třídy. Jedná se o tyto módy:

- **Suma_prediction** – během klasifikace jednotlivých výřezů daného automobilu si uchovává výsledky pravděpodobnosti pro všechny třídy a při klasifikaci posledního výřezu pravděpodobnosti sečte. Třída s největší hodnotou je označena jako vítězná.
- **Modus_win_class** – během klasifikace jednotlivých výřezů daného automobilu si uchovává pouze třídu s největší pravděpodobností a po klasifikaci posledního výřezu aplikuje systém na tyto nejlepší třídy *modus*. Třída s největším zastoupením je označena jako vítězná.

Pro dané video lze získat dva logovací soubory pro klasifikaci s mírně odlišnými výsledky. Tyto dva typy vyhodnocení jsou dále porovnávány.

Po získání hodnot mohla být otestována klasifikace na všech testovacích videích. Experimenty jako u předchozích bloků provádí skript. Je zde využíván skript pojmenovaný *eval_classification.py*.

Skript pro získání úspěšnosti klasifikace bere postupně jednotlivé automobily ze skutečných anotací a k nim hledá příslušné automobily v logovacím souboru. Je pracováno pouze s těmi automobily, které byly při vyhodnocení sledování označeny jako správně sledované. Tento počet může být u klasifikace nižší, protože se pro vyhodnocení klasifikace neberou v úvahu kamiony a nákladní vozidla, které se v testovaných videích také objevují. Po nalezení shodné dvojice jim jsou porovnány třídy a podle toho jsou zařazeny do skupiny OK nebo KO.

Úspěšnost je tedy počítána pouze pro osobní automobily a dodávky. Protože se ve videích vyskytují i automobily, na které není model natrénován, bylo vyhodnocení úspěšnosti rozděleno do více kategorií:

- **All** – pracuje se všemi automobily,
- **Train** – pracuje se s automobily, na které je model natrénován,
- **Skoda** – pracuje se pouze s automobily tovární značky Škoda (protože natrénovaný model obsahuje nejvíce tříd právě této značky).

Tabulka 4.5: Tabulka znázorňující získané výsledky bloku pro klasifikaci automobilů ve videích Session1.

Název videa	Kategorie	$\sum$	Mód	OK	KO	Úspěšnost
Session1_center	All	25	Suma_prediction	15	10	60.00 %
			Modus_win_class	17	8	68.00 %
	Train	18	Suma_prediction	10	8	55.56 %
			Modus_win_class	12	6	66.67 %
	Skoda	12	Suma_prediction	9	3	75.00 %
			Modus_win_class	10	2	83.33 %
Session1_left	All	23	Suma_prediction	14	9	60.87 %
			Modus_win_class	14	9	60.87 %
	Train	18	Suma_prediction	12	6	66.67 %
			Modus_win_class	12	6	66.67 %
	Skoda	7	Suma_prediction	5	2	71.43 %
			Modus_win_class	5	2	71.43 %
Session1_right	All	20	Suma_prediction	15	5	75.00 %
			Modus_win_class	14	6	70.00 %
	Train	13	Suma_prediction	9	4	69.23 %
			Modus_win_class	8	5	61.54 %
	Skoda	10	Suma_prediction	7	3	70.00 %
			Modus_win_class	7	3	70.00 %

Tabulka 4.6: Tabulka znázorňující získané výsledky bloku pro klasifikaci automobilů ve videích Session2.

Název videa	Kategorie	$\sum$	Mód	OK	KO	Úspěšnost
Session2_center	All	37	Suma_prediction	26	11	70.27 %
			Modus_win_class	25	12	67.57 %
	Train	27	Suma_prediction	21	6	77.78 %
			Modus_win_class	22	5	81.48 %
	Skoda	15	Suma_prediction	11	4	73.33 %
			Modus_win_class	12	3	80.00 %
Session2_left	All	28	Suma_prediction	21	7	75.00 %
			Modus_win_class	17	11	60.71 %
	Train	18	Suma_prediction	13	5	72.22 %
			Modus_win_class	13	5	72.22 %
	Skoda	10	Suma_prediction	8	2	80.00 %
			Modus_win_class	8	2	80.00 %
Session2_right	All	28	Suma_prediction	20	8	71.43 %
			Modus_win_class	17	11	60.71 %
	Train	18	Suma_prediction	15	3	83.33 %
			Modus_win_class	15	3	83.33 %
	Skoda	12	Suma_prediction	11	1	91.67 %
			Modus_win_class	11	1	91.67 %

Tabulka 4.7: Tabulka znázorňující získané výsledky bloku pro klasifikaci automobilů ve videích Session3.

Název videa	Kategorie	$\sum$	Mód	OK	KO	Úspěšnost
Session3_center	All	6	Suma_prediction	4	2	66.67 %
			Modus_win_class	4	2	66.67 %
	Train	3	Suma_prediction	2	1	66.67 %
			Modus_win_class	2	1	66.67 %
	Skoda	-	Suma_prediction	-	-	-
			Modus_win_class	-	-	-
Session3_left	All	6	Suma_prediction	5	1	83.33 %
			Modus_win_class	5	1	83.33 %
	Train	3	Suma_prediction	2	1	66.67 %
			Modus_win_class	2	1	66.67 %
	Skoda	-	Suma_prediction	-	-	-
			Modus_win_class	-	-	-
Session3_right	All	7	Suma_prediction	5	2	71.43 %
			Modus_win_class	5	2	71.43 %
	Train	4	Suma_prediction	3	1	75.00 %
			Modus_win_class	3	1	75.00 %
	Skoda	-	Suma_prediction	-	-	-
			Modus_win_class	-	-	-

Tabulka 4.8: Tabulka znázorňující získané výsledky bloku pro klasifikaci automobilů ve videích Session4.

Název videa	Kategorie	$\sum$	Mód	OK	KO	Úspěšnost
Session4_center	All	45	Suma_prediction	28	17	62.22 %
			Modus_win_class	27	18	60.00 %
	Train	26	Suma_prediction	21	5	80.77 %
			Modus_win_class	22	4	84.62 %
	Skoda	17	Suma_prediction	12	5	70.59 %
			Modus_win_class	13	4	76.47 %
Session4_left	All	45	Suma_prediction	28	17	62.22 %
			Modus_win_class	26	19	57.78 %
	Train	26	Suma_prediction	20	6	76.92 %
			Modus_win_class	21	5	80.77 %
	Skoda	15	Suma_prediction	11	4	73.33 %
			Modus_win_class	12	3	80.00 %
Session4_right	All	19	Suma_prediction	11	8	57.89 %
			Modus_win_class	12	7	63.16 %
	Train	11	Suma_prediction	8	3	72.73 %
			Modus_win_class	8	3	72.73 %
	Skoda	5	Suma_prediction	4	1	80.00 %
			Modus_win_class	4	1	80.00 %

Tabulka 4.9: Tabulka znázorňující získané výsledky bloku pro klasifikaci automobilů ve videích Session5.

Název videa	Kategorie	$\sum$	Mód	OK	KO	Úspěšnost
Session5_center	All	71	Suma_prediction	42	29	59.15 %
			Modus_win_class	44	27	61.97 %
	Train	50	Suma_prediction	33	17	66.00 %
			Modus_win_class	35	15	70.00 %
	Skoda	28	Suma_prediction	20	8	71.43 %
			Modus_win_class	21	7	75.00 %
Session5_left	All	63	Suma_prediction	40	23	63.49 %
			Modus_win_class	41	22	65.08 %
	Train	41	Suma_prediction	30	11	73.17 %
			Modus_win_class	33	8	80.49 %
	Skoda	26	Suma_prediction	21	5	80.77 %
			Modus_win_class	24	2	92.31 %
Session5_right	All	35	Suma_prediction	22	13	62.86 %
			Modus_win_class	20	15	57.14 %
	Train	21	Suma_prediction	14	7	66.67 %
			Modus_win_class	13	8	61.90 %
	Skoda	12	Suma_prediction	10	2	83.33 %
			Modus_win_class	9	3	75.00 %

Tabulka 4.10: Tabulka znázorňující souhrnné výsledky bloku pro klasifikaci automobilů.

Název videa	Kategorie	$\sum$	Mód	OK	KO	Úspěšnost
Shrnutí	All	458	Suma_prediction	296	162	64.62 %
			Modus_win_class	288	170	62.88 %
	Train	297	Suma_prediction	213	84	71.71 %
			Modus_win_class	221	76	74.41 %
	Skoda	169	Suma_prediction	129	40	76.33 %
			Modus_win_class	136	33	80.47 %

Shrnutí experimentů

V porovnání obou způsobů výběru vítězné třídy si na všech projetých automobilech vedla lépe metoda **Suma_prediction**. Úspěšněji zvládla rozlišit hranici mezi automobily, na které je klasifikační model natrénován a na které ne.

Při porovnání výsledků pouze na automobilech, na které je klasifikační model natrénován, byla úspěšnější metoda **Modus_win_class**. V kategoriích Train a Skoda vždy o pár procent zvítězila. Přesné výsledky jsou uvedeny v tabulkách 4.5 – 4.10.

Experimenty potvrdily, že v reálném případě je úspěšnost nižší než v případě ideálním. Je to tím, že systém při běhu nezná hodnotu IoU žádného bounding boxu, proto na klasifikátor mohou být poslány i bounding boxy s nízkou hodnotou této metriky. Úspěšnost může také snižovat fakt, že trajektorie prohlášena za správnou může obsahovat určité procento bodů, které byly při vyhodnocení trackeru označeny jako FP. V tomto případě opět systém při běhu neví, jestli je vybraný bod ve skutečnosti TP nebo FP.

Kapitola 5

Závěr

Cílem této práce bylo navrhnout, vytvořit a otestovat systém pro detekci, sledování a klasifikaci automobilů. Práce byla zahájena studiem stěžejních bodů zadání této diplomové práce, popsaných v kapitole 2. Po nastudování všech citovaných materiálů mohla začít tvorba návrhu hierarchie systému.

Pro tento typ systému, který je vhodné rozdělit do více bloků, se nabízelo využití ROS, který má výbornou podporu škálovatelnosti bloků (*nodů*) a podporuje práci se všemi knihovnami využitými při implementaci systému. Protože jsou všechny získané informace ze všech uzlů uchovávány na jednom místě, je velmi jednoduché systém rozšířit. Například o měření velikosti rozestupů mezi automobily.

Implementace systému byla pomocí návrhu řešení rozdělena do více bloků. Skládá se ze tří stěžejních bloků, a to bloků pro detekci, sledování a klasifikaci automobilů a dále bloků pro čtení videa a logování výsledků. Pro detekci byl využit kaskádový klasifikátor, který pracuje s natrénovanou fakultní kaskádou. Pro sledování automobilů byla zvolena metoda Kalmanův filtr, kterému při přiřazování detekcí k jednotlivým sledovaným automobilům pomáhá Hungarian algoritmus. Klasifikaci provádí konvoluční neuronová síť, která klasifikuje automobily pomocí natrénovaného modelu. Úkolem této práce bylo vlastně propojit existující metody pro detekci, sledování a klasifikaci do výsledného systému, přizpůsobit tyto metody na platformu ROS a odstranit co nejvíce nedostatků, aby výsledný systém dosahoval co nejlepších výsledků. Jednotlivé metody byly implementovány do samostatných *nodů* a pomocí mechanismů *topic* a *service* mezi nimi byla zajištěna bezproblémová komunikace.

Experimenty bylo ověřeno, že je systém stabilní a dosahuje dobrých výsledků. Detektor označil 27 358 bounding boxů jako správné, 13 289 jako špatné (malá hodnota metriky IoU) a 4 883 skutečných bounding boxů detektor vůbec nezaznamenal. Z celkového počtu 627 automobilů bylo správně sledováno 479. Úspěšnost sledování byla v průměru 76.40 %. Z 479 správně sledovaných automobilů bylo klasifikováno celkem 458 (nejsou zahrnuty kamiony nebo nákladní automobily). V módu All dosáhl klasifikátor průměrnou úspěšnost 63.75 %, v módu Train 73.06 % a v módu Skoda 78.40 %.

Systém je primárně implementován na detekci, sledování a klasifikaci automobilů zepředu. Pokud by chtěl uživatel systém využít například pro americký trh, stačí načíst kaskádu pro detekci a model pro klasifikaci vozidel vyskytujících se na zpracovávaných video souborech. Systém tedy není závislý pouze na jedné kaskádě a jednom klasifikačním modelu. V případě budoucího použití v praxi by bylo nutné je pravidelně aktualizovat v závislosti na stále se rozšiřujícím počtu typů vozidel.

Další vývoj systému bude spočívat zejména v jeho optimalizaci a dalším zpřesnění. Cílem je také natrénovat model, jehož pomocí bude klasifikátor schopen správně klasifikovat více typů automobilů, tím by vzrostla úspěšnost klasifikace.

Literatura

[1] WWW stránky: ROS [online]. http://www.ros.org/, dostupné online 12.9.2017.

[2] Abadi, M.; Barham, P.; Chen, J.; aj.: TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*, 2016.

[3] Arulampalam, M. S.; Maskell, S.; Gordon, N.; aj.: A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on signal processing*, ročník 50, č. 2, 2002: s. 174–188.

[4] Bernardin, K.; Stiefelhagen, R.: Evaluating multiple object tracking performance: the CLEAR MOT metrics. *Journal on Image and Video Processing*, ročník 2008, 2008: str. 1.

[5] Bradski, G.; Kaehler, A.: *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[6] Chamberlain, R.; Schommer, J.: Using Docker to support reproducible research. *DOI: https://doi. org/10.6084/m9. figshare*, ročník 1101910, 2014: str. 44.

[7] Cheng, Y.; Liu, Q.; Zhao, C.; aj.: Design and Implementation of Mediaplayer Based on FFmpeg. *Software Engineering and Knowledge Engineering: Theory and Practice*, 2012: s. 867–874.

[8] Clark, A. F.; Clark, C.: Performance Characterization in Computer Vision. *European Union's IST programme*, 1999.

[9] Comaniciu, D.; Ramesh, V.; Meer, P.: Kernel-Based Object Tracking. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, ročník 25, č. 5, May 2003: s. 564–577.

[10] Davis, J.; Goadrich, M.: The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*, ACM, 2006, s. 233–240.

[11] Emami, H.; Fathi, M.; Raahemifar, K.: Real Time Vehicle Make and Model Recognition Based on Hierarchical Classification. *International Journal of Machine Learning and Computing*, ročník 4, č. 2, April 2014: s. 142–145, ISSN 2010-3700.

[12] Harris, C.; Stephens, M.: A combined corner and edge detector. In *Alvey vision conference*, ročník 15, Citeseer, 1988, s. 10–5244.

[13] Juránek, R.; Herout, A.; Dubská, M.; aj.: Real-Time Pose Estimation Piggybacked on Object Detection. In *ICCV*, 2015.

[14] Kalman, R. E.: A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, ročník 82, č. 1, 1960: s. 35–45.

[15] Karpathy, A.: Convolutional Neural Networks for Visual Recognition. *Unversity Lecture*, 2016.

[16] Ketkar, N.: Introduction to Keras. In *Deep Learning with Python*, Springer, 2017, s. 95–109.

[17] Kuhn, H. W.: The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, ročník 2, č. 1–2, 1955: s. 83–97.

[18] Kůrková, V.: Fraktální geometrie. *Pokroky matematiky, fyziky a astronomie*, ročník 34, č. 5, 1989: s. 267–277.

[19] Li, L.; Chen, L.; Huang, X.; aj.: A traffic congestion estimation approach from video using time-spatial imagery. In *Intelligent Networks and Intelligent Systems, 2008. ICINIS'08. First International Conference on*, IEEE, 2008, s. 465–469.

[20] Lienhart, R.; Kuranov, A.; Pisarevsky, V.: Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *Joint Pattern Recognition Symposium*, Springer, 2003, s. 297–304.

[21] Matthias, K.; Kane, S. P.: *Docker: Up and Running*. O'Reilly Media, 2015, ISBN 978-1-491-91757-2.

[22] Milan, A.; Leal-Taixé, L.; Reid, I.; aj.: MOT16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831*, 2016.

[23] Mithun, N. C.; Rashid, N. U.; Rahman, S. M. M.: Detection and Classification of Vehicles From Video Using Multiple Time-Spatial Images. *IEEE Transactions on Intelligent Transportation Systems*, ročník 13, č. 3, September 2012: s. 1215–1225, ISSN 1524-9050.

[24] Morris, B. T.; Trivedi, M. M.: Learning, modeling, and classification of vehicle track patterns from live video. *IEEE Transactions on Intelligent Transportation Systems*, ročník 9, č. 3, 2008: s. 425–437.

[25] Okuma, K.; Taleghani, A.; De Freitas, N.; aj.: A boosted particle filter: Multitarget detection and tracking. In *European conference on computer vision*, Springer, 2004, s. 28–39.

[26] Parasuraman, K.; Subin, P.: SVM Based License Plate Recognition System. *International Conference on Computational Intelligence and Computing Research*, 2010.

[27] Parekh, H. S.; Thakore, D. G.; Jaliya, U. K.: A Survey on Object Detection and Tracking Methods. *International Journal of Innovative Research in Computer and Communication Engineering*, ročník 2, č. 2, 2014: s. 2970–2979.

[28] Pimenov, V.: Fast image matching with visual attention and SURF descriptors. In *Proceedings of the 19th International Conference on Computer Graphics and Vision*, 2009, s. 49–56.

[29] Přibyl, P.; Svítek, M.: *Inteligentní dopravní systémy*. BEN-technická literatura, 2001.

[30] Quigley, M.; Conley, K.; Gerkey, B.; aj.: ROS: an Open-Source Robot Operating System. In *ICRA Workshop on Open Source Software*, ročník 3, Kobe, 2009, str. 5.

[31] Ragland, K.; Tharcis, P.: Survey on Object Detection, Classification and Tracking Methods. *International Journal of Engineering Research and Technology (IJERT)*, ročník 3, č. 11, November 2014: s. 622–628.

[32] Rosebrock, A.: Intersection over Union (IoU) for object detection. 2016. URL https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[33] Savić, N.; Junghans, M.; Krstić, M.: Traffic Data Collection Using Tire Pressure Monitoring System. In *International Conference on Transport Systems Telematics*, Springer, 2014, s. 19–28.

[34] Shantaiya, S.; Verma, K.; Mehta, K.: Survey on Approaches of Object Detection. *International Journal of Computer Applications*, ročník 65, č. 18, March 2013: s. 14–20.

[35] Sirmacek, B.; Unsalan, C.: Urban-area and building detection using SIFT keypoints and graph theory. *IEEE Transactions on Geoscience and Remote Sensing*, ročník 47, č. 4, 2009: s. 1156–1167.

[36] Sivaraman, S.; Trivedi, M.: Looking at Vehicles on the Road: A Survey of Vision-Based Vehicle Detection, Tracking, and Behavior Analysis. *IEEE Transactions on Intelligent Transportation Systems*, ročník 14, č. 4, December 2013: s. 1773–1795, ISSN 1524-9050.

[37] Sivaraman, S.; Trivedi, M.: A Review of Recent Developments in Vision-Based Vehicle Detection. *Intelligent Vehicles Symposiu (IV)*, ročník 4, June 2013: s. 310–315, ISSN 1931-0587.

[38] Skolnik, M. I.: Introduction to Radar. *Radar Handbook*, ročník 2, 1962.

[39] Sochor, J.: Fully Automated Real-Time Vehicles Detection and Tracking with Lanes Analysis. In *Proceedings of The 18th Central European Seminar on Computer Graphics*, Technical University Wien, 2014, ISBN 978-3-9502533-3-7.

[40] Sochor, J.; Juránek, R.; Špaňhel, J.; aj.: BrnoCompSpeed: Review of Traffic Camera Calibration and Comprehensive Dataset for Monocular Speed Measurement. 2017, arXiv:1702.06441.

[41] Sochor, J.; Špaňhel, J.; Herout, A.: BoxCars: Improving Fine-Grained Recognition of Vehicles Using 3-D Bounding Boxes in Traffic Surveillance. *IEEE Transactions on Intelligent Transportation Systems*, ročník PP, č. 99, 2018: s. 1–12, ISSN 1524-9050, doi:10.1109/TITS.2018.2799228.

[42] Stauffer, C.; Grimson, W. E. L.: Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, ročník 2, IEEE, 1999, s. 246–252.

[43] Viola, P.; Jones, J. M.: Robust Real-Time Face Detection. *International Journal of Computer Vision*, ročník 57, č. 2, May 2014: s. 137–154, ISSN 1573-1405.

[44] Viola, P.; Jones, M.: Rapid Object Detection Using a Boosted Cascade of Simple Features. *Accepted Conference on Computer Vision and Pattern Recognition*, 2001.

[45] Wang, H.; Kläser, A.; Schmid, C.; aj.: Action recognition by dense trajectories. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, IEEE, 2011, s. 3169–3176.

[46] Welch, G.; Bishop, G.: An Introduction to the Kalman Filter. 1995.

[47] Yilmaz, A.; Javed, O.; Shah, M.: Object Tracking: A Survey. *ACM Comput. Surv.*, ročník 38, December 2006: s. 1773–1795, ISSN 0360-0300.

[48] Yu, Q.; Dinh, T. B.; Medioni, G.: Online tracking and reacquisition using co-trained generative and discriminative trackers. In *European conference on computer vision*, Springer, 2008, s. 678–691.

[49] Zitnick, C. L.; Dollár, P.: Edge boxes: Locating object proposals from edges. In *European Conference on Computer Vision*, Springer, 2014, s. 391–405.

[50] Španěl, M.: Klasifikace a rozpoznávání. VUT FIT UPGM, 2009, str. 7.

Přílohy

Příloha A

Obsah DVD

Přiložené DVD obsahuje technickou zprávu ve formátu PDF a zdrojové soubory ve formátu LATEX, ze kterých byla zpráva vytvořena.

Dále jsou zde uloženy zdrojové soubory výsledného systému, kde složky představují jednotlivé *nody* a soubor README, ve kterém je popsáno, jak se systém spouští. K testovaným videím jsou zde uloženy vstupní masky.

Na DVD jsou také uloženy adresáře s vyhodnocujícími skripty, README a skutečnými i systémem předpovězenými anotacemi.

DVD obsahuje i video, které prezentuje vizuální funkčnost systému.