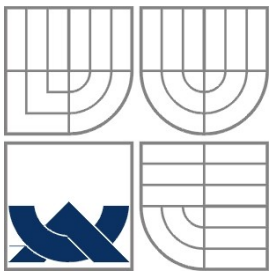
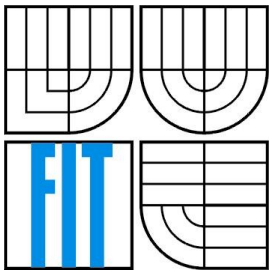


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

AUTOMATICKÝ ODHAD ROZESTUPU MEZI AUTOMOBILY

AUTOMATIC ESTIMATION OF DISTANCE BETWEEN VEHICLES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
MARTIN BERAN

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. SOCHOR JAKUB

BRNO 2016

Abstrakt

Tato bakalářská práce se zabývá automatickým odhadem rozestupu mezi jedoucími automobily. Výsledné soubory obsahují upravený videozáznam se zobrazením rozestupů. Řešení je implementováno v jazyce C++.

Abstract

This thesis deals with the automatic estimation of distance between moving vehicles. The resulting files contain edited video showing the distance. The solution is implemented in C++.

Klíčová slova

automatický, odhad, rozestup, C++, vozidlo, časový odstup

Keywords

automatic, estimation, distance, C++, vehicle, time difference

Citace

BERAN, Martin. Automatický odhad rozestupu mezi automobily. Brno 2016. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Sochor Jakub

Automatický odhad rozestupu mezi automobily

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jakuba Sochora. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Martin Beran
10. května 2015

Poděkování

Chtěl bych poděkovat vedoucímu své bakalářské práce Ing. Jakubovi Sochorovi za pomoc a cenné rady při konzultacích.

Obsah

1 Úvod

Cílem této bakalářské práce bylo vytvořit rozšíření pro již existující program na detekci vozidel z videozáznamu. Toto rozšíření zajišťuje automatický odhad rozestupů mezi jednotlivými vozidly. Odhady jsou implementované dvěma způsoby, a to výpočtem vzdálenosti v metrech a výpočtem vzdálenosti jako časového odstupu mezi průjezdy vozidel.

Implementace každého způsobu spočívá v několika krocích, jako je rozhodování o typu detekovaného objektu, výpočet středu souměrnosti vozidla, výběr nejbližšího vozidla, určení měřící oblasti pro časové vyhodnocení a samozřejmě i samotné postupy výpočtů vzdáleností.

Tato práce je rozdělena do kapitol, popisujících existující řešení některých částí, použité technologie a postupy, návrh algoritmů jednotlivých částí a jejich implementaci a také vyhodnocení úspěšnosti a přesnosti jednotlivých způsobů.

2 Důvody detekce vozidel a odhad rozestupů mezi nimi

Důvodů pro detekci vozidel a automatické odhady rozestupů mezi nimi je hned několik. Mezi nejzávažnější z nich pravděpodobně patří i vysoký počet vozidel.

Se stále zvyšujícím se počtem vozidel se také zvyšuje i počet dopravních nehod. Mezi nejčastější příčiny dopravních nehod patří právě nedodržení bezpečné vzdálenosti mezi vozidly, což mnohdy znemožňuje včasné zastavení. Následné zjišťování, zda k nehodě došlo právě z důvodu nedodržení bezpečné vzdálenosti je pak mnohdy velice zdlouhavé a s nejistým výsledkem.

Cílem práce je navrhnout a implementovat algoritmus pro automatický odhad rozestupů mezi jedoucími vozidly z videozáznamu. Pro tuto práci mi byla poskytnuta natočená videa z různých míst v ČR a dále již hotový program, který zajišťuje detekci samotných vozidel a zjišťování podrobných informací o každém z nich. I přes obrovské možnosti tohoto programu zde ovšem není podporován automatický odhad rozestupů mezi vozidly. Z tohoto důvodu jsem se rozhodl využít dostupný zdrojový kód a připojit k němu svoji práci.

3 Existující řešení

3.1 Postupy pro výpočet vzdálenosti mezi dvěma body

Vzhledem k tomu, že nelze počítat vzdálenost mezi dvěma vícerozměrnými objekty, je nutné tuto operaci provést pouze pro specifické body. O jaké body se jedná a jakým způsobem jsou určovány, je podrobně rozepsáno v kapitole 4.2.1.2. Poté, co jsou tyto body určeny, je možné vypočítat vzdálenost mezi nimi. Pro tyto výpočty existuje několik možných postupů.

Výpočet s pomocí Pythagorovy věty

Tento způsob využívá znalostí klasické Pythagorovy věty, která popisuje vztah platící mezi délkami stran pravoúhlých trojúhelníků. Umožňuje dopočítat délku třetí strany takového trojúhelníka, známe-li délky zbývajících dvou stran. Obecné vyjádření Pythagorovy věty zní následovně:

$$c^2 = a^2 + b^2$$

Oba potřebné body pro výpočet rozestupu objektů jsou zadány ve formátu: `A[A.x, A.y]` a `B[B.x, B.y]`. Délky odvěsen a a b by poté byly vyjádřeny jako rozdíly souřadnic jednotlivých bodů:

$$a = A.x - B.x, b = A.y - B.y$$

Po aplikování těchto skutečností do původního vzorce by poté výsledná vzdálenost byla vypočítána následovně:

$$vzdálenost = \sqrt{a^2 + b^2} = \sqrt{(A.x - B.x)^2 + (A.y - B.y)^2}$$

Toto řešení je výpočetně nejjednodušším řešením a po provedení testů je vidět, že je také časově nejméně náročné.

Manuální výpočet eukleidovské vzdálenosti

Eukleidovská vzdálenost bodů $A[A.x, A.y], B[B.x, B.y]$ je definována následujícím vztahem:

$$vzdálenost = \sqrt{(A.x - B.x)^2 + (A.y - B.y)^2}$$

Tato vzdálenost se rovná délce úsečky, která tyto body spojuje. V praxi je tento postup implementován tak, že je dojde nejprve ke vzájemnému odečtení bodů a až poté je pro výsledný bod vypočtena tato vzdálenost. Tím je způsobeno, že tento postup je nepatrně časově náročnější, než výpočet s využitím Pythagorovy věty, jak je vidět v níže uvedených testech.

OpenCV funkce cv::norm()

Tato funkce nabízí několik způsobů využití. Jedním z nich je také výpočet vzdálenosti mezi dvěma body. Tento výpočet využívá principu vyjádření eukleidovské vzdálenosti, proto má téměř stejnou časovou náročnost. Jelikož ale dochází ke kontrole správnosti a shody datových typů předávaných argumentů, trvá výpočet ještě déle než v předchozím případě.

Níže jsou uvedené testy časové náročnosti pro jednotlivé implementace výše popsaných algoritmů. Algoritmy byly testovány postupně na množině patnácti tisíc bodů, přičemž k určení těchto bodů byl použit uniformní generátor náhodných čísel, který je dostupný v knihovně OpenCV. Jak je vidět, implementace výpočtu pomocí Pythagorova teorému je výpočetně nejjednodušší a tudíž i časově nejméně náročná.
Výsledky testů časové náročnosti implementací existujících postupů pro výpočet vzdálenosti mezi dvěma body vypadají následovně:

```
pythagoreanDistance time[s]:    4.34338
euclideanDistance time[s]:      5.76784
cv::norm() distance time[s]:    6.38777
```

3.2 Způsoby serializace dat

Kvůli velkému objemu dat při zpracování snímku videozáznamu ve vysoké kvalitě není možné je ukládat pouze v paměti programu. Je nutné tato data převést do podoby, ve které je bude možno uložit do souboru, tzv. serializovat. Způsobů serializace je několik. Liší se zejména časovou náročností, velikostí výsledného souboru a možnostmi serializovat složitější datové struktury.

Memcpy
Tento způsob je využíván hlavně v jazyce C a je již značně zastaralý a pro serializaci složitějších datových struktur je téměř nepoužitelný. Bylo by totiž nutné rekurzivně serializovat každý složitější či uživatelsky definovaný prvek struktury zvlášť, což je časově i paměťově velice náročné. V dnešní době se tento způsob již příliš nepoužívá.

Boost::Serialization a Boost::Archive
Serializace prostřednictvím těchto dvou knihoven je sice poměrně časově náročná, nicméně je u tohoto způsobu umožněno relativně jednoduchým způsobem serializovat i složitější a uživatelem definované struktury. Je ovšem nutné pro tyto struktury implementovat přetěžující metody pro způsob jejich serializace.

Hlavní výhodou použití souboru knihoven Boost je, že neslouží pouze pro serializaci objektů, ale například také obsahuje knihovny usnadňující práci při implementaci řešení geometrických problémů (Boost::Geometry), práci s grafy (Boost::Graph) nebo také při práci s vícerozměrnými poli (Boost::Multi_array).

Při implementaci své bakalářské práce jsem využil možností těchto knihoven hlavně pro jednoduchost jejich použití a také pro snadnou integraci dalších modulů tohoto souboru při případném rozšiřování programu.

Cereal
Další možností serializace dat je prostřednictvím knihovny Cereal. Na rozdíl od souboru knihoven Boost je tato knihovna podporována až v C++ verze 11 a její syntaxe je velice podobná knihovně Boost::Serialization. Podporuje konverzi dat do mnoha formátů, například XML, JSON nebo do binárního kódování.

Její hlavní výhodou oproti Boost::Serialization je především její rychlost. Knihovna Cereal provádí serializaci téměř dvakrát rychleji, než knihovna Boost. Bohužel tato knihovna obsahuje pouze

funkce pro serializaci dat, takže její použití v projektech, kde již jsou využívány jiné knihovny souboru Boost je, alespoň dle mého názoru, téměř zbytečné.

4 Návrh řešení

V následující kapitole bude čtenář seznámen s návrhem vlastního algoritmu pro automatický odhad rozestupů mezi vozidly, postupem výběru nejbližšího vozidla vzhledem k aktuálně zpracovávanému, postupem výpočtu středového bodu vozidla, výpočtu vzdálenosti vozidel v metrech a také časového odstupu vozidel.

4.1 Použité technologie

Před samotným začátkem realizace programu bylo potřeba si určit nástroje a postupy, které budou použity pro implementaci vlastního algoritmu. Technologie byly vybrány po analýze problematiky s ohledem na co nejnižší časovou složitost a paměťové nároky programu. V případech, kdy se nabízelo více možností, byl zvolen nástroj dle vlastního uvážení s ohledem na dosavadní zkušenosti s daným nástrojem a s ohledem na případné další využití daného nástroje při případném dalším rozšiřování možností programu.

4.1.1 Programovací jazyk C++

Jako první bylo potřeba zvolit vhodný programovací jazyk. S přihlédnutím k faktu, že je celý program konzolový, jako nejlepší varianta se naskytl jazyk C++, se kterým jsem měl již zkušenosti. Učební literaturou se staly knihy *Moderní programování v C++: návrhové vzory a generické programování v praxi* [1], *Mistrovství v C++* [2] a referenční příručka , vytvořená komunitou[1], dále také oficiální dokumentace knihoven OpenCV[2] a Boost[3]. Pro pochopení principů automatické kalibrace dopravní dohledové kamery a detekci a sledování vozidel ve videu pak články Automatic Camera Calibration for Traffic Understanding [3] a Fully Automatic Roadside Camera Calibration for Traffic Surveillance [4]

Tento jazyk byl zvolen také z důvodu, že již celý původní program byl v tomto jazyce psán a tudíž by v integraci mého rozšíření do původního programu neměly provázet nějaké větší komplikace.

Pro můj program byly využity následující knihovny třetích stran:

- OpenCV[4] - knihovna pro zpracování obrazu a práci s obrazovými prvky
- Boost[5] - soubor víceúčelových knihoven pro C++

4.1.2 Knihovna OpenCV

Tato volně dostupná multiplatformní knihovna je zaměřená především na zpracování obrazu v reálném čase a také na implementaci počítačového vidění. Je psána v jazycích C a C++, proto ji lze použít na různých platformách a operačních systémech. Mimo jiné poskytuje mnoho specifických datových typů a funkcí, které umožňují, výrazně usnadňují a zefektivňují zpracování obrazu v reálném čase.

1 http://www.cplusplus.com/
2 http://docs.opencv.org/
3 http://www.boost.org/doc/
4 http://opencv.org/
5 http://www.boost.org/

4.1.3 Soubor knihoven Boost

V rámci snahy o dosažení korektních výsledků při výpočtu časového odstupu vozidel je nutné tyto výpočty zahájit až v okamžiku, kdy jsou detekovány vodící čáry jednotlivých jízdních pruhů. Data, zpracovaná do té doby je však potřeba uchovat. Jako řešení se nabízely následující možnosti:

a) Uchovat dosavadní získaná data v paměti programu

b) Data uspořádat a uložit jako formát XML

c) Tato data zakódovat a uložit do binárního souboru

S přihlédnutím na možná vysoká rozlišení videozáznamů jsem z důvodu šetření operační paměti zvolil kódováni do binárního souboru, konkrétně pomocí knihoven boost archive[6] a boost serialization[7].

4.2 Návrh programu

Po stanovení a upřesnění používaných technologií, bylo třeba sestavit koncept vlastního programu. Jedná se o nezbytnou součást vývoje každého projektu, i když bývá často opomíjena, mnohdy i zkušenými vývojáři. Dobře připravený návrh může přinášet i značnou úsporu času a tím i redukovat prostředky vynaložené pro vývoj projektu.

Jelikož se jedná pouze o rozšíření původního programu, nebylo třeba vytvářet žádné grafické rozhraní aplikace, pouze upravit informace zobrazované ve výsledném videozáznamu.

Cílová skupina

Vzhledem k tomu, že je i celý původní program konzolový, lze předpokládat, že bude využíván pouze úzkou skupinou lidí, která bude mít pokročilé znalosti v užívání konzolových aplikací. Celé měření rozestupů mezi vozidly bylo vzhledem k požadavkům nutno rozdělit na dvě části, a to:

- Měření rozestupů v metrech
- Měření rozestupů jako časový odstup vozidel

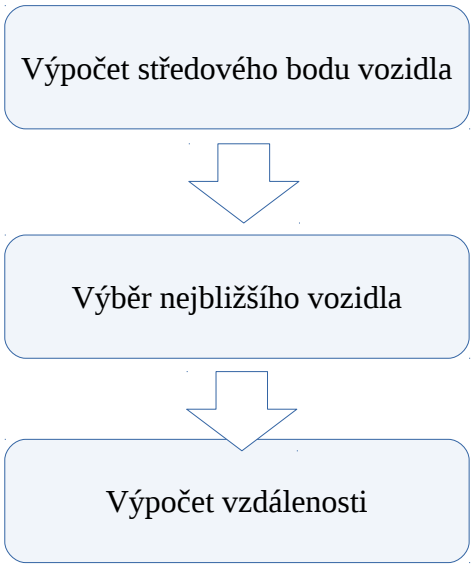
Každá část je podrobněji popsána níže.

4.2.1 Měření rozestupu v metrech

Aby se při měření vzdáleností mezi vozidly předešlo zkreslení výsledků, musí mít toto měření několik omezení. Vzdálenost se musí měřit pouze mezi bezprostředně za sebou jedoucími vozidly. Ta samozřejmě musí jet ve stejném jízdním pruhu. Další podmínkou je, aby byl pro každé vozidlo použit stejný měřící bod, nezávisle na rozměrech vozidla. V této části budou blíže popsány algoritmy pro

6 http://www.boost.org/doc/libs/1_60_0/libs/serialization/doc/archive_reference.html
7 http://www.boost.org/doc/libs/1_60_0/libs/serialization/doc/serialization.html

výběr nejbližšího vozidla dle daných podmínek vzhledem k referenčnímu, výpočet středových bodů zpracovávaných vozidel a dále také výpočet vlastní vzdálenosti.



4.2.1.1 Postup výběru nejbližšího vozidla

Na úspěšnosti tohoto algoritmu záviselo celé měření, bylo proto nutno ho navrhnout velice přesně. Nejprve je vybráno první vozidlo, které je označeno jako refferenceCar. Poté je na všechny ostatní objekty, označené jako currentCar, aplikován postup, znázorněný v níže uvedeném pseudokódu. V první řadě bylo nutno vyloučit detekované objekty, které nesplňovaly podmínky vozidel. Dále bylo nutno vyloučit vozidla jedoucí v opačném směru či v jiném jízdním pruhu. Po provedení tohoto kroku již zbývalo pouze vybrat nejbližší vozidlo. Výběr probíhal určením středového bodu každého vozidla a následným výpočtem vzdáleností pro každé vozidlo. Jako nejbližší bylo poté vybráno vozidlo s nejmenší vzdáleností od referenčního.

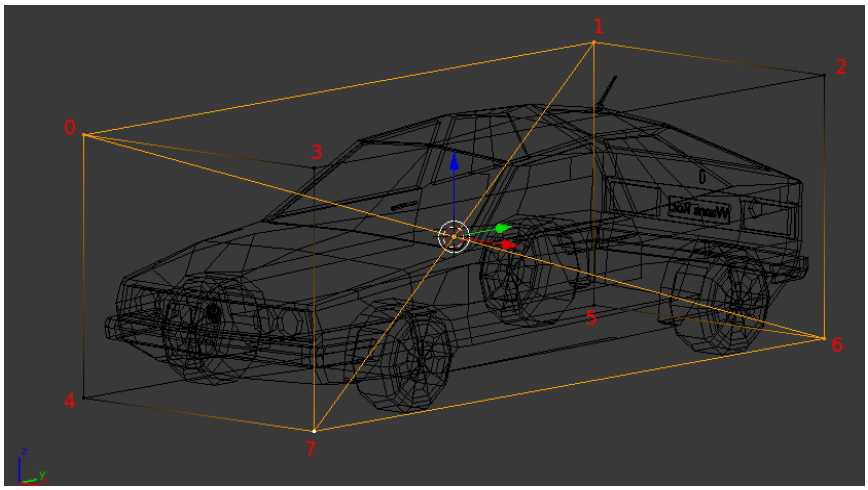
Pseudokód:

1. **pro všechna vozidla V na snímku**
2. { referenceCar = V;
3. **pro všechna ostatní vozidla na snímku**
4. {
5. získání středového bodu refferenceCar
6. **if(**(objekt je vozidlo) && (ID pruhu vozidla == ID pruhu refferenceCar) && (správný směr vozidla)**)**
7. získání středového bodu vozidla
8. výpočet vzdálenosti mezi středovými body
9. **if(**vzdálenost < nejmenší vzdálenost**)**

10. nejmenší vzdálenost = vzdálenost

4.2.1.2 Výpočet středového bodu vozidla

Pro přesné měření vzdáleností bylo nutné pro každý uvažovaný objekt přesný bod, který bude pro měření použit. Jako tento bod jsem zvolil střed souměrnosti vozidla, identifikovaný jako průsečík tělesových úhlopříček boxu, který ohraničuje každé vozidlo, tzv. „Bounding Box“. Pro urychlení algoritmu nebyly při určování průsečíku použity všechny čtyři tělesové úhlopříčky, ale pouze dvě navzájem protilehlé, vedené mezi body 0 → 6 a 7 → 7 (obr. 4.1).



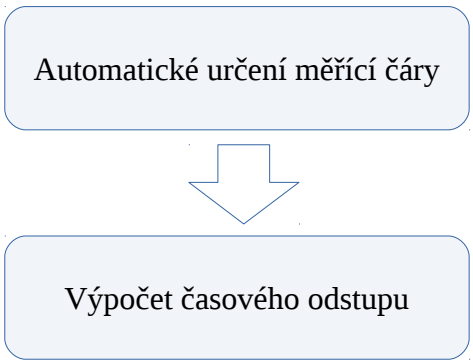
Obr 4.2 - Tělesové úhlopříčky vozidla

4.2.1.3 Výpočet vzdálenosti

Algoritmus pro vlastní výpočet vzdáleností mezi vozidly probíhal na základě určených středových bodů jednotlivých vozidel. Došlo k jednoduchému odečtení bodů reprezentovaných objekty Point knihovny OpenCV. Po tomto odečtení je nutné převést výsledek do absolutní hodnoty, aby se předešlo záporným hodnotám vzdáleností. Následně bylo nutné tuto obrazovou vzdálenost převést na skutečnou. Protože výpočty vzdáleností probíhaly mezi středovými body vozidel, bylo nutné obrazové rozměry těchto vozidel převést na skutečné a poté polovinu délky každého vozidla odečíst od skutečné vzdálenosti, aby se předešlo zkreslení dosažených výsledků.

4.2.2 Měření rozestupu jako časový odstup vozidel

Druhou částí mé bakalářské práce bylo odhadování rozestupů jako časový rozdíl průjezdů jednotlivých vozidel. V této části je popsán algoritmus pro automatické určení oblasti, ve které bude měření časů průjezdů probíhat a dále také algoritmus použitý pro vlastní výpočet časového rozdílu od průjezdu posledního vozidla.



Vzhledem k rozdělení vozovky do několika jízdních pruhů bylo nezbytné rozdělit projíždějící vozidla do jednotlivých jízdních pruhů a informace o průjezdu posledního vozidla uchovávat pro každý pruh zvlášť.

Kvůli ušetření paměťového prostoru je celý vstupní videozáznam programem zpracováván po jednotlivých snímcích. Aby nemusely být tyto údaje přepočítávány pro každý zpracovávaný snímek zvlášť bylo nutné vytvořit datovou strukturu mimo zpracovávané snímky. Tato struktura má formát asociativního pole. Jako identifikační klíče v tomto poli byly použity indexy jízdních pruhů. Ukládaná hodnota pro každý klíč je pak čas průjezdu posledního vozidla pro tento pruh.

Kvůli možnému vícenásobnému počtu výskytů stejného vozidla v měřící oblasti bylo nezbytné uchovávat indexy zpracovaných vozidel a jejich časy průjezdů touto oblastí. Takto zaznamenaná vozidla byla při zpracování dalších snímků automaticky vyřazována.

Pseudokód:

1. **pro všechna vozidla V na snímku**
2. { referenceCar = V;
3. **pro všechna ostatní vozidla na snímku**
4. {
5. **if(**(objekt je vozidlo) && (ID pruhu vozidla == ID pruhu refferenceCar) && (správný směr vozidla)**)**
6. **if(**průjezd v jízdním pruhu inicializován**)**
7. výpočet časového odstupu od posledního průjezdu
8. aktualizace průjezdu pro jízdní pruh
9. **else**
10. inicializace průjezdu pro jízdní pruh

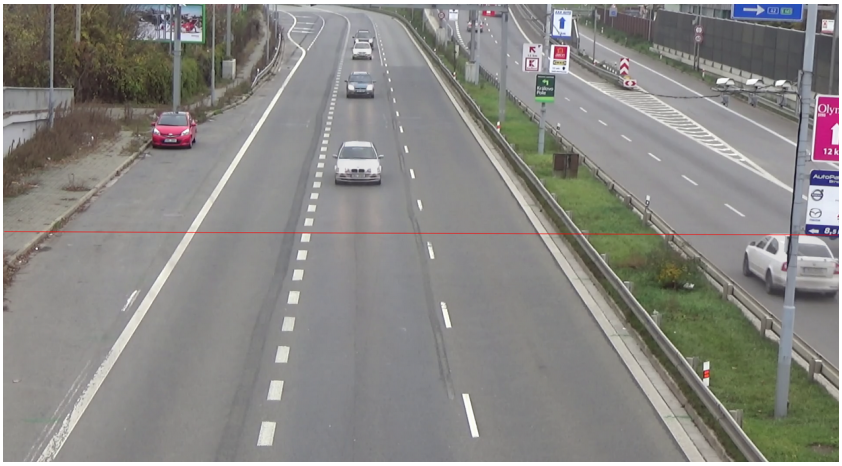
4.2.2.1 Automatické určení měřící oblasti

Určení měřící oblasti bez nutnosti jakéhokoliv zásahu uživatele se již od samotného začátku jevilo jako stěžejní problém. Hlavním úkolem bylo určit výchozí bod pro konstrukci přímky, v jejíž oblasti bude měření prováděno.

Zvolil jsem možnost, kdy je vybrán specifický bod přímky reprezentující jízdní pruh, který je z pohledu kamery nejvíce vlevo. Za tímto účelem bylo nezbytné zpracovávat snímky až po určení jízdních pruhů. K uložení již získaných informací o snímcích a vozidlech jsem zvolil možnost jejich zakódování do binárního souboru. Postup samotného kódování je popsán v kapitole 5.1.

Po získání potřebného bodu bylo nutné určit hraniční body této přímky. Z těch je poté vypočítán výchozí bod pro určení měřící oblasti. Nachází se přesně v polovině přímky reprezentující jízdní pruh. Druhým hraničním bodem je pak druhý bod zmizení, obsahující důležité informace týkající se kalibrace kamery.

Výsledná přímka je pak kolmá na vozovku a je tedy vhodná k měření časového odstupu vozidel (obr. 4.2).



Obr 4.2 - přímka reprezentující měřící oblast

4.2.2.2 Výpočet časového odstupu

Po získání měřící přímky byl již samotný výpočet časového odstupu velice jednoduchý. Nejprve bylo potřeba zjistit, kolik času uplyne mezi dvěma snímky videozáznamu. Poté bylo nutné zkontrolovat, zda byl pro daný jízdní pruh již inicializován index snímku průjezdu prvního vozidla.

V takovém případě došlo k odečtení indexů snímků a vynásobení tohoto čísla dobou přechodu na další snímek. Nakonec musí být aktualizován index snímku pro daný jízdní pruh.

V případě, že index snímku pro jízdní pruh zatím nebyl inicializován, časový odstup vozidel se nepočítá a pouze dojde k inicializaci.

5 Implementace

Hlavním úkolem této bakalářské práce bylo implementovat rozšíření, které má za úkol automaticky odhadovat rozestupy mezi jednotlivými vozidly. Toto rozšíření bylo integrováno do původního programu na detekci a sledování vozidel. Implementace byla realizována podle návrhu, který je popsaný v kapitole 4.2.

5.1 Serializace snímků a jejich uložení do binárních souborů

První částí implementace byla nutnost začít s odhadováním rozestupů až po inicializaci jízdních pruhů. Informace o snímcích, které byly zpracované do té doby, bylo potřeba uchovat. Pro toto jsem využil možností souboru knihoven Boost, konkrétně knihovny Boost::Archive a Boost::Serialization. Pro specializované datové typy OpenCV, jako je například Point, Rect, Mat a další a jiné, uživatelem definované, struktury a objekty, mezi něž například patří struktura CarData, která obsahuje kompletní informace o všech detekovaných objektech na daném snímku, bylo nutné pro bezchybnou serializaci přetížit předem definované metody serialize, save a load.

Vzhledem k vysokému objemu dat, zvláště pak samotného snímku reprezentovaného datovým typem cv::Mat, je tato část celého procesu nejpomalejší. Takto serializované snímky jsou pak uloženy do binárních souborů. Pro každý snímek je vygenerován vlastní soubor jehož název je „data_XX“, kde XX značí číslo daného snímku.

Vytvoření binárního souboru, archivace a uložení serializovaných dat do tohoto souboru probíhá následujícím způsobem:

1. pomocí funkce std::ofstream s použitím příznaku ios::binary je vytvořen binární soubor připravený pro zápis dat.
2. S využitím metody binary_oarchive knihovny Boost::Archive, která při svém vyvolání automaticky využívá přetížené metody serializace, jsou veškerá data snímku serializována, archivována a uložena do určeného binárního souboru.

5.2 Implementace časového zpoždění zpracování odhadů

Veškerá data od samotného začátku videozáznamu jsou zpracována způsobem, který je popsán v kapitole 5.1 až do doby, kdy počet detekovaných objektů ve videozáznamu dosáhne počtu, který je uveden v inicializačním souboru default.ini. V tu chvíli je spuštěna metoda RebuildClusters, třídy LanesClustering, která zajišťuje přepočítání jízdních pruhů. Do této funkce jsem přidal následující úsek kódu:

```
Line l = Line(Init→GetFirstVanishingPoint(),
Clusters.front().ClusterCenter);
auto points =
l.ImageEdgesCrossing(Init→GetBackground().clone().size());
cv::Point2d drawPoint;
drawPoint.x = ((points.first.x + points.second.x) / 2.0);
drawPoint.y = ((points.first.y + points.second.y) / 2.0);
std::ofstream laneFile("laneFile.txt");
laneFile << drawPoint.x << " " << drawPoint.y << std::endl;
```

Tento kód vytváří čáru reprezentovanou objektem třídy Line. Čára je vytvořena ze dvou bodů, kterými jsou tzv. „první bod zmizení“ a střed jízdního pruhu, který je z pohledu kamery nejvíce vlevo. Následně je tato čára pomocí metody ImageEdgesCrossing upravena tak, aby sahala až k okrajům snímku na pozadí. Návratovou hodnotou této metody jsou body, kde čára protíná hranice snímku.

Z těchto dvou bodů je poté vypočítán a následně uložen do souboru laneFile.txt bod, který je použit pro vytvoření měřící oblasti pro časové měření rozestupů. Při přechodu na zpracování dalšího snímku je po jeho obvyklém uložení provedena kontrola, zda byl vytvořen tento soubor. V případě úspěchu program vstupuje do podmínky, kde vybrán první ze seznamu snímků. Tam je otevřen soubor s daným číslem snímku a jeho obsah je deserializován pomocí metody boost::archive::binary_iarchive a nahrán do proměnné myData. Poté je soubor daného snímku smazán, do dat snímku jsou dodatečně přidány informace o měřící oblasti,které jsou potřebné pro časové měření a následně je vyvolána funkce compute_distance třídy Distance, která zajišťuje veškeré výpočty vzdáleností. Nakonec je snímek pomocí funkce VideoOutput→AddFrame přidán do výsledného výstupního videa.

Po uplynutí vstupního videa je potřeba ještě zpracovat zbylé snímky. To provádí konečný for cyklus, jehož tělo je totožné s tělem podmínky, kde se zpracovávají snímky.

5.3 Postup výpočtu středového bodu vozidla

Při obou způsobech měření rozestupů je potřeba určit středový bod každého vozidla. Tuto úlohu zajišťuje metoda getPoint třídy Distance. Funkce jako argument přijímá strukturu std::vector<BoundingBox3D>, což je v podstatě pole „klecí“. Každá tato klec je složena z pole osmi bodů, jejichž propojením se kolem vozidla vytvoří. Pro výpočet středového bodu vozidla jsou použity body s indexy 0, 1, 6 a 7. Jejich propojením dojde k vytvoření dvou tělesových úhlopříček, které jsou dostačující pro určení středu souměrnosti vozidla (obr 5.1).

Níže zobrazený kód popisuje princip a postup určení středu souměrnosti z dostupných tělesových úhlopříček.

```
cv::Point2d CarsDistance::getPoint(std::vector<BoundingBox3D>
BoundingBoxes){
      BoundingBox3D BoundingBox = BoundingBoxes.back();

      Line body_diag1(BoundingBox[0], BoundingBox[6]);
      Line body_diag2(BoundignBox[1], BoundingBox[7]);
      cv::Point2d centerPoint = body_diag1.Intersection(body_diag2);
      return centerPoint;
}
```

Nejprve je potřeba získat aktuální Bounding Box. Ten je uložen v předávaném argumentu na posledním místě pole. Získal jsem ho díky vestavěné C++ funkci std::vector.back(), umožňující přístup k poslednímu prvku vektoru. Následně jsem vytvořil dva objekty třídy Line, reprezentující přímku, v tomto případě tělesové úhlopříčky. Díky funkci Line::Intersection je poté získán střed souměrnosti vozidla.

5.4 Postup výpočtu vzdálenosti v metrech a výběru nejbližšího vozidla

Ve snaze předejít zbytečnému zpoždění při vyloučení objektů, které nesplňovaly podmínky pro vozidla a rozdělování vozidel do jednotlivých pruhů, jsou tyto tyto požadavky kontrolovány pro každý zpracovávaný objekt zvlášť.

```
for(auto &refCar : data.Cars){  /*For each car in frame*/
    distance = 0.0;
    real_distance = std::numeric_limits<double>::infinity();
    closest_distance = std::numeric_limits<double>::infinity();
    CarData refferenceCar = refCar.second;
    if(refferenceCar.IncludedToStats == true){
      CarData closestCar;
      cv::Point2d curCarPoint;
      cv::Point2d refCarPoint;
      refCarPoint = getPoint(refferenceCar.BoundingBoxes);
      for(auto const &curCar : data.Cars){
        if(curCar.second.IncludedToStats == true){
          if(refferenceCar.ID != curCar.first){
            CarData currentCar = curCar.second;
            if(refferenceCar.LaneClusterIndex ==
              currentCar.LaneClusterIndex&&!refferenceCar.WrongWay&&
              !currentCar.WrongWay){
                 curCarPoint=getPoint(currentCar.BoundingBoxes);
```

Výše zmíněný úsek kódu zajišťuje výpočet vzdálenosti v metrech pro každý objekt, který vyhovuje podmínkám a nejbližšího objektu. Každý objekt z data.Cars, který je zahrnut do statistik a tudíž je klasifikován jako vozidlo je postupně ukládán do proměnné refferenceCar. Poté se znovu prohledává data.Cars, kde je aktuální vozidlo reprezentováno jako currentCar.

Pokud je objekt currentCar klasifikován jako vozidlo, nemá stejné ID jako refferenceCar, jede ve stejném jízdním pruhu jako refferenceCar a nejede v protisměru, je mezi těmito vozidly spočítána vzájemná vzdálenost. V případě, že je nižší než vzdálenost od předchozího vozidla, jsou aktuální currentCar a vzdálenost refferenceCar od něj uloženy do closestCar a closest_distance.

```
double xDiff=refCarPoint.x-curCarPoint.x;
double yDiff=refCarPoint.y-curCarPoint.y;

distance=std::sqrt((xDiff*xDiff)+(yDiff*yDiff));
if(distance<closest_distance){
   closest_distance=distance;
   closestCar=currentCar;
```

Vlastní výpočet vzdálenosti je pak implementován podle algoritmu vycházejícího již existujícího řešení vyjádření vzdálenosti dvou bodů pomocí Pythagorovy věty, který je podrobně vysvětlen v kapitole 3.1.

Proměnné `xDiff` a `yDiff` reprezentují jednotlivé odvěsny pravoúhlého trojúhelníku, do proměnné distance je pak uložena výsledná vzdálenost mezi vozidly. V případě, že vypočítaná vzdálenost je menší, než ta doposud nejmenší, je tato vzdálenost i aktuální vozidlo za původní hodnoty.

5.5 Určení počátečního bodu pro měřící oblast

Správné určení tohoto bodu je velmi důležité pro celé časové měření rozestupů vozidel. Níže uvedený úsek kódu celou tuto operaci popisuje podrobně. Tento kód je integrován do hlavního programu v metodě RebuildClusters.

Tato metoda je členem třídy LanesClustering, která zabezpečuje detekci jízdních pruhů a jejich postupné upřesňování v průběhu zpracování vstupního videozáznamu. Při každém volání metody RebuildClusters dochází k přepočítání trajektorií jízdních vzhledem k předchozím hodnotám a také k určení počátečního bodu měřící oblasti a uložení jeho souřadnic do textového souboru s názvem laneFile.txt. Tento soubor, stejně jako ostatní dočasné soubory využívané při běhu je před ukončením programu smazán.

```
Line l = Line(Init->GetFirstVanishingPoint(),
Clusters.front().ClusterCenter);

auto points =
l.ImageEdgesCrossing(Init→GetBackground().clone().size());

cv::Point2d drawPoint;

drawPoint.x = ((points.first.x + points.second.x) / 2.0);
drawPoint.y = ((points.first.y + points.second.y) / 2.0);

std::ofstream laneFile("laneFile.txt");
laneFile << drawPoint.x << " " << drawPoint.y << std::endl;
```

Uvedený kód popisuje výpočet a uložení počátečního bodu pro určení měřící oblasti. V první řadě je vytvořen objekt třídy Line, který reprezentuje přímku, vedenou tzv. „prvního bodu zmizení“ do středu jízdního pruhu.

Při detekci a přepočítávání jsou jízdní pruhy do struktury ukládány tak, v jakém pořadí se nachází zleva od pohledu kamery.

Pro můj výpočet je tedy použit střed jízdního pruhu, který je nejvíce vlevo. Následně dochází k protažení této přímky až k okrajům snímku metodou ImageEdgesCrossing. Návratovou hodnotou této metody je dvojice bodů, ve kterých přímka proťala hranice snímku. Z těchto bodů je poté vypočítán hledaný počátek přímky pro měřící oblast. Nakonec jsou souřadnice bodu uloženy do textového souboru.

V hlavní větvi programu jsou při zpracování dalšího snímku tyto souřadnice získány a uloženy do proměnné distanceLanePoint. Druhým bodem pro určení měřící přímky je pak „druhý bod zmizení“. Oba tyto body jsou uloženy do vektoru distanceDetectLane, který je součástí struktury FrameData. Celá tato struktura je pak jako argument předána funkci Distance→compute_distance.

Uvnitř této funkce jsou pak tyto body převedeny na přímku, a ta je opět protažena až k okrajům snímku. Následně je vytvořen objekt třídy cv::LineIterator, který rozděluje přímku na jednotlivé pixely a umožňuje tak přístup k souřadnicím každého z nich.

5.6 Detekce průjezdu vozidla měřící oblastí

Po umožnění přístupu k souřadnicím každého bodu měřící přímky je možné stanovit oblast, ve které bude detekce vozidla probíhat. Vzhledem k tomu, že není zpracováván každý jednotlivý snímek videa je jasné, že není možné kontrolovat pouze souřadnice bodů měřící čáry. Je potřeba kontrolovat také oblast okolo těchto bodů. Samotnou detekci, zda vozidlo projelo příslušnou oblastí, zajišťuje metoda Distance::checkCoordsToleration.

```
bool CarsDistance::checkCoordsToleration(cv::Point2d carPoint,
cv::Point2d lanePoint){
     double coordsTolerationX = 5.0;
     double coordsTolerationY = 10.0;

     if((carPoint.x >= (lanePoint.x - coordsTolerationX) &&
carPoint.x <= (lanePoint.x + coordsTolerationX)) && (carPoint.y >=
(lanePoint.y - coordsTolerationY) && carPoint.y <= (lanePoint.y +
coordsTolerationY))){
          return true;
     }
     return false;
}
```

Jako vstupní argumenty přijímá pozici středového bodu aktuálně zpracovávaného vozidla jednoho bodu měřící čáry. Oba tyto argumenty jsou reprezentovány datovým typem cv::Point2d. Návratovou hodnotou je true v případě, že bylo vozidlo detekováno v měřící oblasti. V opačném případě je vracena hodnota false. Samotná detekce probíhá pouze porovnáváním, zda se souřadnice středového bodu vozidla vyskytuje v okolí bodu měřící přímky. Toto okolí jsem po sérii testů zvolil pět bodů pro souřadnice X a deset bodů pro souřadnice Y.

5.7 Výpočet časového odstupu průjezdů vozidel

Po úspěšné detekci vozidla v měřící oblasti je nutné provést kontrolu, zda byl průjezd pro daný jízdní pruh již inicializován. Níže uvedený úsek kódu podrobně popisuje postup inicializace průjezdu měřící oblastí a určování časového odstupu mezi vozidly.

```
double timeBetweenFrames = 1/Config.VideoInput.DowngradeToFPS;

if(ClustersCrossingTime.count(refferenceCar.LaneClusterIndex) != 1){
     ClustersCrossingTime[refferenceCar.LaneClusterIndex] = frame;
     processedCarsIDs.push_back(refferenceCar.ID);
}else{
     refferenceCar.timeDistance=(frame-
ClustersCrossingTime[refferenceCar.LaneClusterIndex])*timeBetweenFra
mes;
     processedCars[refferenceCar.ID]=(frame-
ClustersCrossingTime[refferenceCar.LaneClusterIndex])*timeBetweenFra
mes;
     if(refferenceCar.timeDistance < 1){
          refferenceCar.toClose = true;
     }
     ClustersCrossingTime[refferenceCar.LaneClusterIndex]  =  frame
     processedCarsIDs.push_back(refferenceCar.ID);
}
break;
```

V proměnné timeBetweenFrames je uložen časový odstup mezi jednotlivými zpracovávanými snímky. Tento čas je spočítán podle vztahu

Poté proběhne kontrola, zda byl průjezd pro daný jízdní pruh již inicializován. Tato kontrola

$$\frac{1}{\textit{počet zpracovaných snímků za vteřinu}}$$

je provedena pokusem o nalezení indexu daného jízdního pruhu v asociativním poli ClustersCrossingTime. Položky tohoto pole, indexovaného podle indexů jízdních pruhů, obsahují čísla snímků posledního průjezdu vozidel daným pruhem. Pokud položka s daným indexem neexistuje, daný jízdní pruh nebyl zatím inicializován.

V takovém případě se provede inicializace jednoduchým uložením čísla aktuálního snímku na daný index pole. Dále je také vektoru processedCarsIDs uloženo ID detekovaného vozidla, aby toto vozidlo již nebylo v dalších snímcích zpracováváno.

V opačném případě dojde k výpočtu časového odstupu od průjezdu posledního vozidla. Tento výpočet je proveden odečtením čísla aktuálního snímku od snímku uloženého v ClustersCrossingTime na příslušném indexu a následným vynásobením proměnnou timeBetweenFrames. Výsledkem je časový odstup uvedený ve vteřinách. Do ClustersCrossingTime na index příslušného pruhu je uloženo číslo aktuálního snímku, do processedCarsIDs pak ID detekovaného vozidla a do struktury std::map processedCars na index detekovaného vozidla je uložen zjištěný časový odstup. Pokud je časový odstup kratší, než jedna vteřina, je indikátor vozidla refferenceCar.toClose nastaven na hodnotu true. Tento indikátor značí, že je vozidlo příliš blízko předchozímu a časový odstup je ve videozáznamu zobrazen odlišnou barvou.

Tyto struktury jsou přístupné mimo zpracovávané snímky. Jsou využívané pro přístup k informacím o časovém odstupu v každém zpracovávaném snímku a pro vyloučení vozidel, která již byla detekována.

5.8 Kompilace programu

Předtím, než bude možné výsledný program používat, je nutné ho zkompilovat. Tato kompilace spočívá v několika krocích.

Nejprve je nutné se přesunout do adresáře TrafficAnalyser, který se nachází v hlavním adresáři aplikace. Zde je následně nutné provést tyto příkazy zajišťující jak kontrolu přítomnosti všech potřebných prostředků, tak i samotnou kompilaci:

- `mkdir -p build`: Vytvoření adresáře, do které budou uloženy informace o vlastní kompilaci a také výsledné spustitelné soubory
- `cd build`: Přesun do kompilačního adresáře
- `cmake ..`: Kontrola přítomnosti všech potřebných prostředků pro úspěšnou kompilaci a spuštění programu
- `make`: Spuštění kompilace

5.9 Spuštění programu

Jak jsem již uvedl, má práce spočívala v rozšíření původního programu o automatický odhad rozestupů mezi vozidly. Toto rozšíření i původní program vyžadují svoji funkčnost přítomnost několika nástrojů. Informace, o jaké nástroje se jedná a popis jejich získání naleznete v příloze

Ovlivnění práce programu přepínači

Přepínače patří mezi standardní prostředky pro nastavení parametrů spouštěného souboru a také se jejich prostřednictvím specifikují cesty k vstupním a výstupním souborům. Původní program je schopný zpracovat několik přepínačů, které upřesňují, jaká zdrojová data budou použita. Každý přepínač má dvě alternativy zápisu. Akceptované přepínače jsou popsány v následujícím seznamu:

- `-h`, `--help`: V případě přítomnosti tohoto přepínače je na standardní výstup pouze vytištěna nápověda k programu
- `-i`, `--input`: Tento povinný argument specifikuje cestu ke vstupnímu videozáznamu
- `-c`, `--config`: Nepovinný argument, který upřesňuje konfigurační soubor, který bude při zpracování použit. Pokud není uveden, je použit standardní soubor default.ini, uložený v adresáři Configuration, který je přístupný z kořenového adresáře programu

Příklady spuštění aplikace

Program je spouštěn z adresáře `build`, který je vytvořen kompilaci programu. Postup kompilace je podrobně popsán v kapitole 5.8. Pro spuštění je potřeba základní soubor programu se jménem `TrafficAnalyser`. Tento soubor zajišťuje kompletní spuštění aplikace. Nyní je už načase uvést základní příklady samotného spuštění. Za proměnnou `FILE` lze dosadit relativní cestu k videozáznamu, který chceme zpracovávat, vzhledem k aktuální složce. Kdybychom tedy měli v kořenovém adresáři aplikace video `test_video.avi`, které bychom chtěli zpracovat, obsahovala by proměnná `FILE` následující cestu: `../../test_video.avi`.

- `./TrafficAnalyser -h`: Vytiskne nápovědu na standardní výstup a ukončí program
- `./TrafficAnalyser -i FILE`: Základní spuštění programu
- `./TrafficAnalyser -i FILE -c custom_cfg_file`: Ovlivnění použitého konfiguračního souboru

6 Vyhodnocení výsledků

Po dokončení implementace rozšíření bylo potřeba analyzovat úspěšnost vyvinutých algoritmů pro odhady rozestupů, zejména co se týče odhad časového odstupu.

Úspěšnost odhadu časového odstupu byla testována na pěti různých videozáznamech. Výsledky tohoto testování jsou zaznamenány v níže uvedené tabulce:

	1	2	3	4	5	průměr
Velikost (AVI) [MB]	575,6	580,5	579,6	579,7	578,2	578,72
Délka videa [min]	15:00	15:00	15:00	15:00	15:00	15:00
Počet detekovaných vozidel	1250	1375	1967	1906	1472	1594
Doba zpracování	4:02:50	4:26:12	5:01:39	4:49:05	4:28:21	4:33:25
Počet zpracovaných vozidel	1182	1217	1743	1692	1265	1420
Úspěšnost [%]	94,56	88,5	88,6	88,7	85,9	89,25

Za účelem přesnějšího vyhodnocení dosažených výsledků, byly používané testovací videozáznamy zkráceny na jednotnou délku patnácti minut.

Jak je patrné z výsledků testování, implementovaný algoritmus pro odhad rozestupů je poměrně vysoce časově náročný a jeho průměrná úspěšnost je 89,25%. Tato fakta jsou ovlivněna hned několika faktory.

Mezi nejhlavnější z nich patří úhel pohledu kamery, který výrazně ovlivňuje už samotnou detekci a sledování vozidel. Hlavním faktorem vysoké časové je pak to, že odhady vzdáleností je potřeba provádět až po detekci vodících čar jízdních pruhů. Do té doby je nutno zpracované snímky uchovávat. Jelikož je pro každý snímek vytvářen speciální soubor, dochází k zápisu a čtení vysokého objemu dat na disk, což jej zatěžuje a zpomaluje. Dalším faktorem zpomalujícím zpracování je určitě ne zcela optimalizovaná implementace výpočtu rozestupů. Tyto a jistě i další faktory by byly vhodným námětem pro případná rozšíření.

7 Závěr

Výsledkem této bakalářské práce je funkční rozšíření původního programu pro detekci a sledování vozidel ve video záznamu. Toto rozšíření zajišťuje automatický odhad rozestupů mezi detekovanými vozidly a využívá funkcí knihovny OpenCV verze 2.4.9. Rozšíření má dvě hlavní části, odhad rozestupů v metrech a odhad rozestupů vyjádřený jako časový odstup mezi průjezdy jednotlivých vozidel měřící oblastí. Pro odhad rozestupů v metrech byly použity tři algoritmy a výsledkem tohoto procesu je zobrazení zjištěných informací pro každé detekované vozidlo ve výstupním videozáznamu.

Zadání práce zahrnovalo nastudování algoritmů pro detekci a sledování vozidel ve videu, algoritmů pro automatickou kalibraci dopravní dohledové kamery, návrh a implementace systému pro automatický odhad rozestupů mezi detekovanými vozidly a vyhodnocení přesnosti těchto odhadů. Jednotlivé požadavky zadání byly splněny tak, že byly nastudovány jednotlivé algoritmy pro detekci vozidel a kalibraci dopravní kamery, byl navrhnut a implementován systém pro automatický odhad rozestupů a jeho úspěšná integrace do původního programu a výsledky byly popsány v kapitole 6. K práci byl vytvořen také prezentační plakát a video.

Program poskytuje možnosti případných rozšíření v podobě snížení časové náročnosti, zvýšení úspěšnosti zpracování detekovaných objektů, provádění dynamických výpočtů bezpečné vzdálenosti mezi vozidly, ukládání snímků vozidel, která tuto vzdálenost nedodržela nebo jiných úprav.

Literatura

[1] ALEXANDRESCU, Andrei. Moderní programování v C++: návrhové vzory a generické programování v praxi. Brno: Computer Press, 2004. ISBN 80-251-0370-6.
[2] PRATA, Stephen. Mistrovství v C++. 4., aktualiz. vyd. Překlad Boris Sokol. Brno: Computer Press, 2013. Bestseller (Computer Press). ISBN 978-80-251-3828-1.
[3] DUBSKÁ Markéta, SOCHOR Jakub a HEROUT Adam. Automatic Camera Calibration for Traffic Undestanding. In: Proceedings of BMVC 2014. Nottingham: The British Machine Vision Association and Society for Pattern Recognition, 2014, s. 1-10. ISBN 1-901725-49-9.
[4] DUBSKÁ Markéta, HEROUT Adam, JURÁNEK Roman a SOCHOR Jakub. Fully Automatic Roadside Camera Calibration for Traffic Surveillance. IEEE Transactions on Intelligent Transportation Systems, 2014, s. 1 - 10.

Seznam příloh

Příloha 1: Obsah CD

- `/src/` – adresář se zdrojovými soubory a modifikovanými soubory původního programu
- `/doc/` – zdrojový soubor písemné zprávy
- `/xberan34-BP.pdf` – elektronická verze písemné zprávy
- `/poster.pdf` – plakát k programu
- `/video.avi` – doprovodné video k programu
- `/out.mkv` – ukázkový výstup programu

Příloha 2: Instalace potřebných knihoven

Pro úspěšný překlad a spuštění programu je nutné zajistit podporu některých knihoven. Všechny uvedené příkazy jsou platné pro operační systémy Ubuntu 14.04 a Ubuntu 16.04, na kterých vývoj rozšíření programu probíhal. Níže uvádím seznam potřebných knihoven a postup k jejich instalaci:

Boost – Při serializaci objektů a práci se soubory jsou využívány knihovny Boost::Filesystem, Boost::Serialization a Boost::Archive. Tyto knihovny jsou obsaženy v základních repositářích a je možno nainstalovat je pomocí příkazů:

```
sudo apt-get install libboost-dev
sudo apt-get install libboost-filesystem-dev
sudo apt-get install libboost-serialization-dev
```

OpenCV – Knihovna pro práci s obrazovými prvky. Níže je uvedený podrobný návod k jejímu stažení a instalaci

```
sudo apt-get update
sudo apt-get upgrade

sudo apt-get install build-essential libgtk2.0-dev libjpeg-dev
libtiff4-dev libjasper-dev libopenexr-dev cmake python-dev python-
numpy python-tk libtbb-dev libeigen3-dev yasm libfaac-dev
libopencore-amrnb-dev libopencore-amrwb-dev libtheora-dev libvorbis-
dev libxvidcore-dev libx264-dev libqt4-dev libqt4-opengl-dev sphinx-
common texlive-latex-extra libv4l-dev libdc1394-22-dev libavcodec-
dev libavformat-dev libswscale-dev default-jdk ant libvtk5-qt4-dev

cd ~
wget http://sourceforge.net/projects/opencvlibrary/files/opencv-
unix/2.4.9/opencv-2.4.9.zip
unzip opencv-2.4.9.zip
cd opencv-2.4.9

mkdir build
cd build
cmake -D WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON
-D INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D
BUILD_EXAMPLES=ON -D WITH_QT=ON -D WITH_OPENGL=ON -D WITH_VTK=ON ..

make
sudo make install
```

Po provedení výše uvedených příkazů je ještě nutné OpenCV konfigurovat. Je nutné otevřít a editovat soubor `opencv.conf`. K tomu slouží příkaz

```
sudo gedit /etc/ld.so.conf.d/opencv.conf
```

A na jeho úplný konec vložit řádek `/usr/local/lib` a tento soubor uložit.

Dále je potřeba provést konfiguraci knihovny příkazem

```
sudo ldconfig
```

Poté je potřeba ještě přimět systém uvažovat cestu ke knihovně. Po otevření souboru

```
sudo gedit /etc/bash.bashrc
```

Se musí na jeho úplný konec vložit následující řádky:

```
PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
export PKG_CONFIG_PATH
```

Posledním krokem instalace knihovny je odhlášení a znovu přihlášení uživatele, případně restart zařízení, na které je OpenCV instalováno. Bez provedení tohoto kroku nebude knihovna správně fungovat.