VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁVRH GRAMATIKY A UŽIVATELSKÉHO ROZHRANÍ PRO FILTROVÁNÍ A VIZUALIZACI ČASOPROSTOROVÝCH DAT

DESIGN OF GRAMMAR AND USER INTERFACE FOR VISUALIZATION AND FILTRATION OF SPATIO-TEMPORAL DATA OF ROAD-USERS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. RICHARD HAUERLAND**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Hauerland Richard, Bc.**
Program: Informační technologie
Obor: Počítačová grafika a multimédia
Název: **Návrh gramatiky a uživatelského rozhraní pro filtrování a vizualizaci časoprostorových dat**
Design of Grammar and User Interface for Visualization and Filtration of Spatio-Temporal Data of Road-Users
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s problematikou vyhodnocování dopravních dat na základě analýzy trajektorií jednotlivých uživatelů silničního provozu. Zaměřte se na formální gramatiky používané v tomto oboru.
2. Navrhněte a popište formalismus, který umožní prostorovou filtraci (průjezd bránou, průjezd zónou apod.) a filtraci na základě statických a dynamických atributů (např. barva objektu, aktuální rychlost, délka stání, délka trvání atd.).
3. Navrhněte uživatelskou aplikaci s uživatelským rozhraním, která bude sloužit jako nástroj pro anyalýzu dat založenou na vašem formalismu. Nástroj umožní interaktivní analýzu a vizualizaci prostřednictvím uživatelského rozhraní. Zaměřte se na intuitivnost rozhraní při zachování dostatečné obecnosti v tvorbě dopravně analytických úloh.
4. Implementujte navržené uživatelské rozhraní a navržený formalismus.
5. Vyhodnoťte vlastnosti výsledného řešení. Testujte intuitivnost rozhraní na uživatelích. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu.

Literatura:

- Arnowitz J., M. Arent, and N. Berger, Effective Prototyping for Software Makers. Elsevier Science & Technology, 2007. ISBN: 978-0120885688
- Kuniavsky M., Observing the user experience: a practitioner's guide to user research. Elsevier, 2003. ISBN: 978-1466508910
- Ware, C.: Visual Thinking: for Design, Morgan Kaufmann, 2010, ISBN 978-0123708960
- Munzner T., Visualization analysis and design. AK Peters/CRC Press, 2014. ISBN: 978-1466508910

Při obhajobě semestrální části projektu je požadováno:

- První tři body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Cílem této práce je navrhnout gramatiku a uživatelské rozhraní pro filtrování a vizualizaci časoprostorových dat. Úvodním úkolem je seznámit se s problematikou vyhodnocování dopravních dat na základě analýzy trajektorií. Následuje návrh a popis formalismu, který umožňuje prostorovou filtraci a filtraci na základě statických a dynamických atributů. Podle vytvořeného formalismu je proveden návrh aplikace s uživatelským rozhraním určené k analýze dat. Před návrhem aplikace bylo provedeno srovnání významných existujících řešení. Aplikace byla implementována pomocí Qt frameworku s využitím programovacího jazyka C++ doplněného o použití jazyka QML.

Abstract

Objective of this thesis is about design of grammar and user interface for filtering and visualization of spatiotemporal data. The initial task is to get acquainted with evaluation of traffic data based on trajectory analysis. The next part is the design and description of a formalism which allows spatial filtering and filtering based on static and dynamic attributes. Based on the created formalism, data analysis application with a user interface is designed. Design process was preceded by a comparison of existing solutions. Application is implemented in Qt Framework using C++ and QML languages.

Klíčová slova

vizualizace, analýza trajektorií, filtrace, formalismus, uživatelské rozhraní

Keywords

visualization, trajectory analysis, filtration, formalism, user interface

Citace

HAUERLAND, Richard. *Návrh gramatiky a uživatelského rozhraní pro filtrování a vizualizaci časoprostorových dat.* Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Návrh gramatiky a uživatelského rozhraní pro filtrování a vizualizaci časoprostorových dat

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Richard Hauerland
19. května 2021

Poděkování

Chtěl bych poděkovat panu Ing. Jaroslavovi Rozmanovi, Ph.D. za odborné vedení práce a cenné rady, které mi pomohly tuto práci zkompletovat. Dále děkuji panu Ing. Davidu Hermanovi za jeho podporu a pomoc.

Obsah

Kapitola 1

Úvod

Použití motorových vozidel je v dnešní době pro většinu populace primárním způsobem dopravy. Vzhledem k vysokému počtu motorových vozidel na silnicích je nutné efektivně řídit dopravu a především zajistit maximální bezpečnost na silnicích. Dopravní situace bývá nejvíce problematická především ve velkých městech, kde často dochází k přetížení dopravy. Kromě zlepšení městské infrastruktury je dále možné optimalizovat dopravu pomocí monitorování a následné analýzy získaných dat [1]. Pro získání dat se používají různé typy senzorů, díky kterým jsme schopni efektivně sledovat dopravní situaci na silnicích. Jedním z nejvíce efektivních přístupů k monitorování dopravy je použití videokamer. Pomocí vhodně umístěných dopravních kamer jsme schopni monitorovat provoz na vozovce a získávat tak důležité informace o dopravní situaci [18]. Získané informace je následně možné využít k analýze, na základě které můžeme docílit nejenom zlepšení dopravní situace, ale také je možné zvýšit bezpečnost samotných řidičů a případně chodců.

Výsledkem monitorování dopravy jsou dopravní data, která je nutné vyhodnocovat. V této práci se budeme zabývat především analýzou trajektorií konkrétních uživatelů silničního provozu. Vzhledem k velkému množství získaných dat musíme být schopni provést filtraci trajektorií tak, abychom mohli efektivně přistupovat jen k těm informacím, které jsou pro nás důležité. Nad množinou získaných trajektorií je dále možné provádět prostorovou filtraci. Může nás například zajímat pouze jeden pruh vozovky nebo případně průjezd vozidel zvolenou zónou. Dále musíme být schopni filtrovat trajektorie na základě statických atributů, do kterých patří například barva a kategorie vozidla. Je také nutná podpora filtrace podle dynamických atributů zahrnujících zejména rychlost, zrychlení, dobu výskytu a délku stání vozidla [24]. Jednotlivé typy filtrů trajektorií je nutné implementovat a následně dokázat aplikovat tak, abychom zpracovávali jenom pro nás důležitá data. Tato práce se dále zabývá návrhem a popisem formalismu umožňujícího filtraci trajektorií. Následně bude proveden návrh a popis interaktivního uživatelského rozhraní, díky kterému jsme schopni vizualizovat analyzovaná data a efektivně aplikovat filtraci trajektorií.

Kapitola 2

Analýza provozu

Text v této podkapitole byl vytvořen na základě myšlenek popsaných v [18]. Dále bylo při tvorbě tohoto textu využito poznatků získaných v [24].

Motorová vozidla jsou pro velkou část lidí hlavním způsobem dopravy. Osobní automobily jsou bezpochyby nejpoužívanějším z motorových vozidel. Právě motorová vozidla mají nejvýraznější dopad na dopravní situaci ve velkých městech po celém světě. Dopravní situace samozřejmě neovlivňuje pouze zbylé účastníky provozu, ale také obyvatele těchto měst. Dopravní systém daného města je ovlivněn dalšími aspekty, mezi které patří samotná rozloha města, hustota zalidnění, stav silniční sítě a dopravní předpisy v daném městě. Dalším podstatným aspektem je monitorování dopravní situace včetně získávání statistik. Monitorování dopravy je důležitý způsob, díky kterému je možné nejenom docílit zlepšení dopravní situace v daném městě, ale hlavně může mít pozitivní dopady na bezpečnost provozu a obyvatel města. V Číně a Indii dohromady ročně zemře téměř půl milionu lidí v důsledku dopravní nehody. Tato data naznačují, že je nezbytné zvýšit úsilí věnované bezpečnosti provozu. Právě monitorování dopravy může výrazně pomoci zredukovat počet dopravních nehod.

Obrázek 2.1: Ukázka stavu dopravy na frekventované dálnici (převzato z [1]).

2.1 Monitorování dopravy

Aktuálně v dopravě převládá monitorování dopravní situace za pomoci fyzických senzorů umístěných na silnici. Díky těmto senzorům dochází k monitorování dopravní situace především na křižovatkách s cílem redukovat dopravní zácpy a přetížení. Nejpoužívanějším dopravním senzorem je dopravní indukční smyčka, což je elektronické zařízení sloužící ke zjištění přítomnosti vozidla na vozovce. Nainstalovaná indukční smyčka je znázorněna na obrázku 2.2. Při průjezdu vozidla křižovatkou dochází ke snížení indukčnosti a následnému zvýšení frekvence oscilátoru. Pokud daná frekvence překročí stanovený práh, tak je obvykle vnímána jako přítomnost vozidla na indukční smyčce. V případě umístění několika smyček za sebe do kaskády jsme schopni stanovit rychlost projíždějících vozidel [15]. Díky dopravním smyčkám je možné kontrolovat a následně optimalizovat dopravní situaci na křižovatkách. Použití dopravních smyček má ale několik značných nevýhod. Nevýhody vycházejí z nutnosti instalace indukčních smyček a dále z důvodu údržby již nainstalovaných smyček. Instalace a údržba smyček totiž způsobí dočasné omezení nebo dokonce přerušení provozu v daném jízdním pruhu. Indukční smyčku je pak nutné nainstalovat do každého jízdního pruhu, což je další značnou nevýhodou.

Obrázek 2.2: Detekční smyčka na pražské křižovatce v podobě černého obdélníku přes nápis „BUS“ (převzato z [15]).

Velmi zajímavou alternativou k dopravním smyčkám je použití dopravních kamer. Jedna kamera je obvykle schopna monitorovat provoz z více jízdních pruhů najednou, což je velká výhoda v porovnání s dopravní smyčkou [22]. Monitorování pomocí kamer poskytuje možnost složitější analýzy provozu v porovnání se smyčkou. Díky použití kamer je možné analyzovat nejenom přítomnost vozidla a jeho rychlost, ale je možné získat například barvu vozidla a také typ daného vozidla. Dále je možné analyzovat dynamické atributy vozidla, mezi které patří rychlost vozidla, zrychlení vozidla a případně doba výskytu a stání vozidla. Stěžejní výhodou dopravních kamer je skutečnost, že můžeme sledovat nejenom vozidla na vozovce, ale lze také sledovat například chodce na přechodech. Monitorování křižovatky je tedy možné provádět jako celek včetně semaforů a přechodů pro chodce. Tímto způsobem

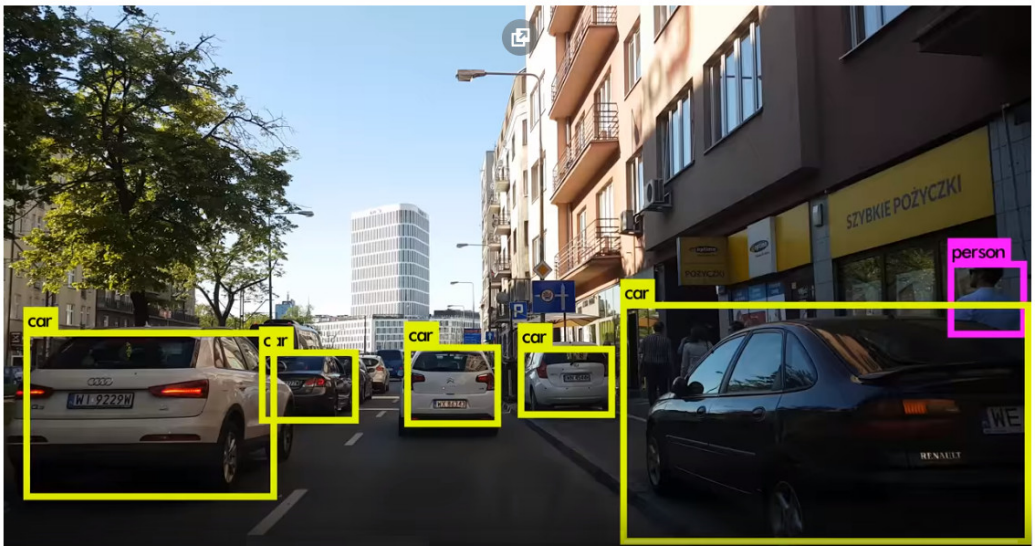
již nejsme omezeni pouze na jednotlivé pruhy jako v případě dopravních smyček, což je znázorněno na obrázku 2.1.

Vylepšení analýzy obrazu a videa jsou faktory, které přispívají k intenzivnějšímu výzkumu a vývoji aplikací v oblasti monitorování dopravy pomocí kamer. Analýza projíždějících vozidel prostřednictvím zpracování obrazu je komplikovaný proces, který zahrnuje porozumění nejenom účastníků provozu, ale především samotné scény prostředí. Obvykle je kromě monitorování vozidel a chodců nutné zohlednit také vozovku, jízdní pruhy, semafory a dopravní značky v rámci snímané scény. Konkrétní objekty zájmu, kterými jsou právě vozidla a chodci, je nutné oddělit od zbytku obrazu. Oddělení nemusí být úplně snadné vzhledem k tomu, že může docházet například ke vzájemnému překrývání objektů scény, anebo nedojde ke správné detekci objektu z důvodu nízké světelnosti vozovky a prostředí. Vzhledem k tomu, že scéna je obvykle venkovní, musí aplikace čelit různým výzvám včetně povětrnostních podmínek, které ovlivňují barvu a strukturu objektů ve scéně. Následně je potřeba zohlednit stíny způsobené budovami, stromy a dalšími prvky v rámci scény. Potom ještě musíme řešit okluzi objektů v důsledku úhlu kamery a změny jasu v průběhu dne.

2.2 Klasifikace objektů

Oddělení pozadí od objektů zájmu je důležitou technikou zpracování obrazu pro aplikace, které vyžadují detekci, segmentaci a následnou klasifikaci snímaných objektů. V rámci sledované scény vždy dochází k dělení objektů do dvou různých kategorií. První kategorii tvoří objekty zájmu, které chceme posléze analyzovat. Ve druhé kategorii jsou objekty pro nás nepodstatné s tím, že je naším cílem oddělit je od zájmových objektů. Obvykle tedy objekty zájmu řadíme do popředí scény. Jako pozadí scény pak uvažujeme zbylé nevýznamné objekty. Ve většině případů je proto důležitá naše schopnost oddělení prvků v popředí od pozadí sledované scény. Bez splnění této podmínky většinou není možné efektivně provádět monitorování zvolené scény.

Proces oddělení popředí a pozadí se zcela odlišuje podle toho, zda pracujeme se stacionární kamerou nebo naopak používáme kameru pohyblivou. Pohyblivé kamery je možné použít například v samotném vozidle a sledovat chování daného vozidla a reakce řidiče na různé dopravní situace. Použití tohoto typu kamery je znázorněno na obrázku 2.3. Kamery ve vozidle jsou nutností v případě monitorování chování autonomních vozidel. Efektivní detekce jiných vozidel na vozovce a také chodců na přechodech je v případě autonomních vozidel naprosto klíčová a úroveň bezpečnosti zde musí být výrazně vyšší. V případě této práce ale budeme pracovat pouze s kamerami stacionárními, které jsou obvykle dostačující pro monitorování provozu na křižovatkách.



Obrázek 2.3: Monitorování provozu pomocí kamery umístěné na palubní desce vozidla (převzato z [20]).

2.2.1 Odstranění pozadí

Prvním krokem úspěšné analýzy objektů v popředí je jejich oddělení od zbytku scény. Při typickém přístupu k oddělení popředí vycházíme z předpokladu, že objekty zájmu jsou v pohybu. Díky pohybu jsou objekty zájmu ve vztahu k pozadí, které je v případě stacionární kamery fixní a nemění se. Na základě tohoto vztahu k pozadí jsme schopni objekty oddělit od pozadí scény. Za normálních okolností můžeme každý statický objekt scény považovat za pozadí a každý pohyblivý objekt scény pak můžeme brát jako popředí scény. Základním přístupem k oddělení objektů popředí je uložení pozadí scény jako snímek obsahující pouze samotné pozadí bez objektů zájmu. Na základě tohoto přístupu jsme schopni pro každý pixel snímku videa $V(t)$ v konkrétním čase t provést výpočet rozdílu intenzity pixelu pro případ, kdy se jedná o pixel pokrytý objektem zájmu. Následně také zohledňujeme případ, kdy je objekt nepřítomen a jedná se tedy o výchozí stav pozadí scény pro daný pixel. Následující vztah znázorňuje výpočet zmíněného rozdílu:

$$F(x,y) = \begin{cases} foreground, & \text{if } |V(x,y,t) - B(x,y)| > T \\ background, & \text{otherwise} \end{cases} \tag{2.1}$$

V tomto vztahu hodnota $B(x,y)$ reprezentuje intenzitu pixelu pozadí na souřadnici obrazu (x,y). Hodnota $V(x,y,t)$ odpovídá hodnotě intenzity pixelu na souřadnici (x,y) snímku videa v konkrétním čase t, kde hodnota T odpovídá prahu tolerance pro porovnávání intenzit pozadí s pixelem v daném snímku videa. Je důležité si uvědomit, že operace odčítání je vlastní vůči barevnému modelu obrazu.

Nevýhodou výše zmíněného přístupu je skutečnost, že nejsme schopni reagovat na změny scény v průběhu času. Mezi tyto změny patří především změna světelných podmínek scény a také další průběžné aktualizace snímku pozadí. Mnohem více sofistikovaným přístupem je analyzovat každý pixel obrazu nezávisle. Na základně posledních N hodnot daných pixelů pak získáváme hustotu Gaussova rozdělení pravděpodobnosti, která je definována střední hodnotou μ a rozptylem σ^2. Je tedy nutné získat nejvíce pravděpodobnou barevnou hod-

notu pro každý pixel s tím, že vždy bereme v úvahu posledních N snímků. V rámci počáteční podmínky můžeme předpokládat intenzitu pixelu prvního snímku jako průměr a dále zvolenou výchozí hodnotu odchylky. Pro každý další snímek v čase t se hodnoty průměru a rozptylu aktualizují následujícím způsobem:

$$\begin{aligned} \mu_t &= \rho I_t + (1-\rho)\mu_{t-1} \\ \sigma^2 t &= d^2\rho + (1-\rho)\sigma^2_{t-1} \\ d &= |(I_t - \mu_t)| \end{aligned} \tag{2.2}$$

Kde d je euklidovská vzdálenost mezi hodnotou pixelu a střední hodnotou a ρ je časové okno, které určuje dopad na hustotu rozdělení každou novou aktualizací snímku obrazu. Pro hodnotu $\rho = 1$ je průměr hustoty rozdělení a rozptylu určen pouze aktuálním rámcem, a proto každý nový snímek patří na pozadí. Čím menší je hodnota ρ, tím větší je počet rámců použitých k výpočtu hustoty rozdělení. Díky prahové hodnotě k jsme schopni zjistit, zda hodnota pixelu spadá do požadovaného intervalu distribuce intenzit pixelů na pozadí. Pixely jsou na základě toho klasifikovány jako pozadí nebo popředí:

$$\begin{aligned} \frac{|(I_t - \mu_t)|}{\sigma_t} &> k \rightarrow foreground \\ \frac{|(I_t - \mu_t)|}{\sigma_t} &< k \rightarrow background \end{aligned} \tag{2.3}$$

Hlavní výhodou tohoto přístupu je možnost dynamické aktualizace pozadí scény. Pozadí se tedy s postupem času přizpůsobuje změnám ve scéně, které zahrnují například změnu osvětlení a také přítomnost dalších nestatických objektů. Níže můžeme vidět názornou ukázku použití tohoto přístupu při monitorování pouliční scény. Časový rozdíl mezi těmito dvěma snímky je přitom pouhých několik vteřin. Tato situace je přehledně znázorněna na obrázku 2.4.

Obrázek 2.4: Pozadí scény zachycené stacionární kamerou. a) Snímek scény s hodně chodci. b) Pozadí scény po analýze scény po dobu několika vteřin. (převzato z [18])

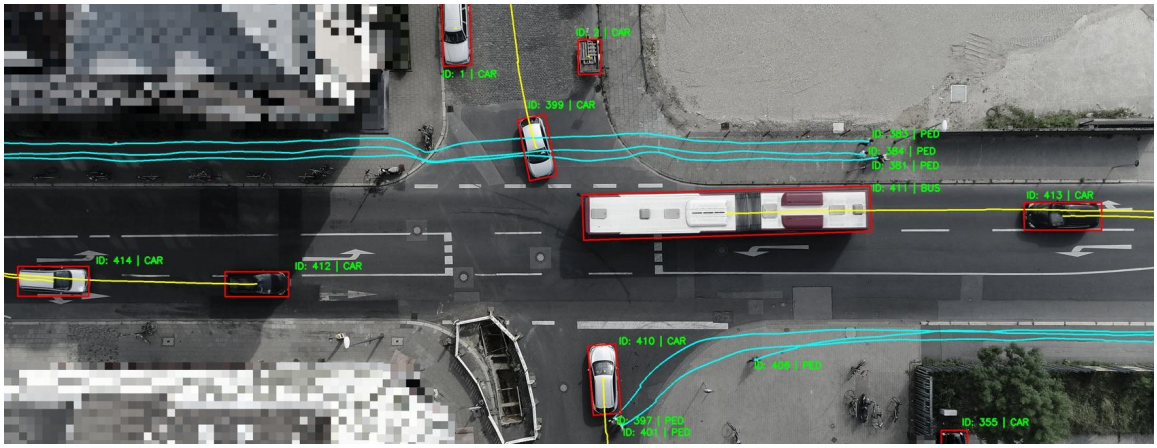
2.3 Detekce objektů

Text níže se zabývá detailnějším popisem detekce objektů ve sledované scéně. V rámci monitorování silničního provozu je důležité detekovat nejenom vozidla stojící na křižovatce, ale musíme být schopni detekovat také vozidla projíždějící. Je potřeba navrhnout takový algoritmus, pomocí kterého jsme schopni reagovat na náhlé zastavení vozidla a na jeho opětovný návrat do pohybu. V případě monitorování křižovatky můžeme například sledovat příjezd vozidla k semaforům a následné zastavení vozidla. Při rozjetí tohoto vozidla musíme být schopni zajistit, že se v rámci detekce bude jednat o ten stejný vůz s totožným identifikátorem. Kromě samotné detekce je nutné provádět sledování detekovaného vozidla a zaznamenávat jeho pohyb a dále atributy jeho pohybu. V případě každého detekovaného vozidla je tedy potřeba pracovat s trajektorií jeho pohybu [10]. Vzniklá trajektorie je dráhou odpovídající pohybu daného vozidla, která má tvar souvislé křivky. Na obrázku 2.5 lze vidět scénu včetně trajektorií detekovaných objektů. Při monitorování scény nás obvykle nebude zajímat pouze trajektorie pohybu vozidla, ale také rychlost, zrychlení, doba stání a délka výskytu vozidla.

2.3.1 Získání trajektorií

Musíme počítat s tím, že projíždějící objekt nemusí být pouze osobní automobil, ale může to být například autobus, motocykl nebo dokonce cyklista. V důsledku toho se může změnit algoritmus pro detekci typu vozidla. V rámci detekce projíždějícího cyklisty je například potřeba počítat s tím, že algoritmus detekce může detekovat jízdní kolo jako vozidlo, ale daného cyklistu může detekovat jako chodce, protože se stále jedná o pohybující se osobu. Detekce barvy u cyklisty může být definována jako barva daného kola, ale také se může jednat o barvu oblečení daného cyklisty nebo dokonce kombinaci obojího. Detekce projíždějících objektů tedy není snadný úkol a existují různé přístupy k jeho řešení. V textu níže se budeme zabývat především detekcí osobních automobilů a dále nastíníme detekci chodců.

Detekované vozidlo tedy nemusí být pouhá trajektorie, ale jedná se o samostatný objekt, který může mít celou řadu dalších atributů. Tento detekovaný objekt se samozřejmě může vyskytovat v několika odlišných stavech. Stavy takového objektu mohou vycházet například z toho, do jakého intervalu rychlosti patří dané vozidlo. Objekt pak může být ve stavu statickém nebo například ve stavu rozjíždění, brzdění nebo souvislém pohybu s konstantní rychlostí [12]. Snímané objekty je dále možné klasifikovat na základě stavu nebo specifického atributu. Hlavními atributy detekovaného objektu budou jeho typ a barva. Mohou nás zajímat například jenom automobily červené barvy. Tímto přístupem je dokonce možné sestavovat pokročilé filtry nad detekovanými objekty. Aplikované filtry by bylo možné hierarchicky skládat do stromové struktury a dále by šlo aplikovat booleovské operace na více filtrů najednou. Filtrace by tehdy mohla zohledňovat například automobily modré barvy s průměrnou rychlostí nižší než je zvolený práh s tím, že pomocí booleovské operace sjednocení bychom ještě mohli zohledňovat všechny projíždějící autobusy. V dalších kapitolách se budeme věnovat filtraci vozidel detailněji a bude popsána i samotná implementace takových filtrů.



Obrázek 2.5: Znázornění typů a trajektorií monitorovaných objektů na vozovce (převzato z [2]).

2.3.2 Detekce vozidel

Bylo již zmíněno výše, že z vozidel se v tomto textu budeme zabývat především detekcí osobních automobilů. Samotné automobily jsou bezpochyby nejpoužívanějším typem vozidla, který se na silnicích vyskytuje, a proto si také zaslouží největší pozornost. Dále bylo řečeno, že analýza vozidel může být rozdělena do dvou na sebe navazujících fází. Ve fázi první je naším cílem vozidlo nejdříve detekovat. Teprve až po úspěšné detekci dojde k zahájení sledování tohoto vozidla. V rámci sledování detekovaného vozidla je nutné zohlednit nejenom souvislý pohyb vozidla, ale také rozjíždění a případné brzdění vozidla na vozovce. Výstupem této analýzy je tedy trajektorie vozidla.

Po úspěšném detekování vozidla je potřeba tomuto vozidlu přiřadit unikátní identifikační číslo, pomocí kterého jsme schopni vozidlo jednoznačně odlišit od zbylých vozidel. Výstupem detekce je počáteční souřadnice tohoto vozidla. V rámci následného sledování detekovaného vozidla postupně provádíme výpočet jeho aktuální souřadnice. Získání nové souřadnice vozidla je možné provést další analýzou pohybu, ale dále můžeme v rámci výpočtu využít minulé souřadnice získané v předchozích fázích sledování pohybu. Aktuální trajektorie pohybu vozidla může být využita pro zpřesnění výpočtu nové souřadnice pohybu. V rámci trajektorie vždy získáváme dráhu pohybu vozidla. Použití minulých souřadnic má nejenom pozitivní dopad na přesnost analýzy, ale může také urychlit samotné monitorování, protože jsme schopni výpočet nové trajektorie vyřešit efektivněji a tím případně snížit výpočetní nároky vyžadované daným algoritmem.

Při monitorování silničního provozu je nezbytné zajistit instalaci dopravních kamer na strategická místa infrastruktury města. Mezi tato strategická místa patří především vysoce frekventované silnice a křižovatky. Dále to mohou být taková místa, kde je potřeba dbát na zvýšenou bezpečnost řidičů a přecházejících osob. Na základě výše uvedeného textu je nutné zajistit, aby byl získaný obraz z kamer stabilní a nedocházelo k různým nežádoucím posuvům obrazu. Budou nás tedy zajímat záznamy pořízené ze stacionárních kamer. Pokud vezmeme v úvahu scény pořízené stacionárními kamerami, tak z pohledu zpracování obrazu mohou být vozidla reprezentována jako objekty v popředí, protože jsou obvykle v pohybu ve vztahu k prvkům na pozadí scény. Pro následnou segmentaci objektů v popředí lze použít techniky používané ve zpracování obrazu. Po oddělení objektů v popředí od pozadí scény je dalším krokem jejich klasifikace, která probíhá na základě vizuálních atributů nebo

pohybových vzorů [22]. Po klasifikaci lze získané objekty dále analyzovat, aby nám poskytly užitečné informace pro výslednou aplikaci.

Barva a textura jsou důležitými percepčními deskriptory při popisu objektů, a proto se běžně používají jako jeden z prvních diskriminačních prvků za účelem určení segmentů zájmu v obrazu. Vzhledem k tomu, že barva povrchu vozovky obvykle spadá do odstínů šedé barvy, tak v některých scénářích nám stačí pouze barva k detekci a segmentaci vozidel. Toto ale vyžaduje, aby sledované vozidlo bylo z hlediska barevnosti na vozovce výrazné a ideálně i kontrastní vůči povrchu vozovky. Nemalá část monitorovaných vozidel ale může mít barvu své karoserie také v odstínech šedi. Z tohoto důvodu je třeba vzít v úvahu další vlastnosti vozidla jako jsou jeho hrany a různé tvary karoserie [22]. V případě různých venkovních prostředí se však barva a struktura daného objektu mohou za různých povětrnostních a světelných podmínek jevit jako zcela odlišné a je třeba tyto podmínky zohlednit. Existují různé sofistikované přístupy, které umožňují pokročilou klasifikaci pixelů. Tyto klasifikátory jsou schopny určit, zda konkrétní pixel svou barevností odpovídá barevnosti povrchu vozovky. Na základě této klasifikace se provádí seskupování pixelů stejné kategorie do regionů, a právě na základě těchto regionů jsme schopni v rámci obrazu ignorovat celé shluky pixelů, které nás při detekci vozidel nezajímají. Zájmové regiony je posléze možné spojovat a získat tak výsledný tvar vozidla.

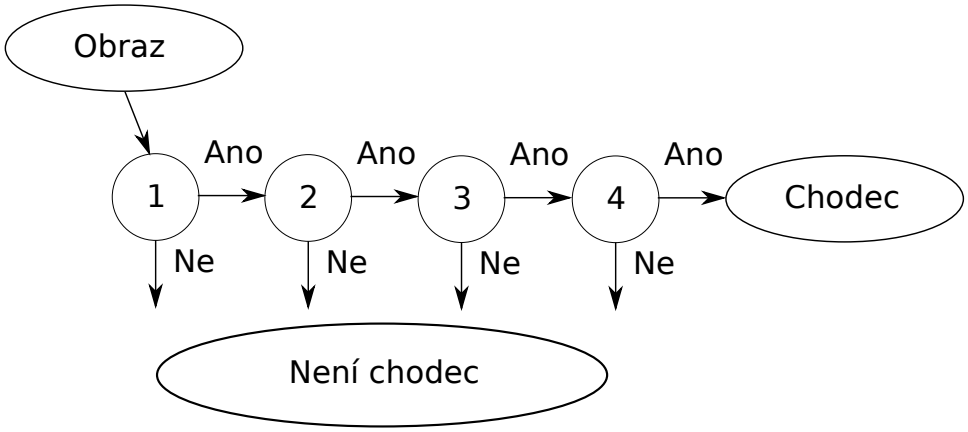
2.3.3 Detekce chodců

V předcházející části jsme se zabývali detekcí vozidel a to konkrétně osobních automobilů. Tento typ vozidla je bezpochyby nejvíce se vyskytujícím objektem na silnicích. Chodci na silnicích jsou v zásadě stejně důležití jako samotná vozidla, protože v reálném světě musí každá dopravní aplikace respektovat bezpečnostní opatření. V případě monitorování městské infrastruktury je detekce chodců přecházejících vozovku velmi užitečná. Důvodů, proč analyzovat pohyb chodců, je rovnou několik. Tím hlavním důvodem je samozřejmě jejich bezpečnost. Jsme například schopni vyhodnocovat četnost osob na přechodu pro chodce, který vede přes vysoce frekventovanou silnici. Pomocí získaných dat jsme schopni řídit a nastavovat jednotlivé intervaly na semaforech tak, abychom zlepšili provoz na křižovatce a zároveň se vyhnuli tomu, aby lidé museli dlouho čekat na přechodu. Dále je možné monitorovat osoby v rámci specifikovaného regionu. Tyto regiony mohou být třeba mnohoúhelníkové útvary, které ohraničující místa s vysokou četností chodců. Při analýze obrazu nás například zajímají jenom některé specifikované regiony a zbytek obrazu pro nás nemusí být podstatný. Detekce chodců je naprosto klíčová v případě autonomních vozidel. Kamera se pak nachází v samotném vozidle a schopnost efektivně detekovat osoby je v takové situaci z hlediska bezpečnosti povinná. Autonomní vozidlo musí totiž provádět velmi spolehlivá rozhodnutí především v situaci, kdy vozidlo přijde do přímého kontaktu s člověkem.

Detekce chodců je ale v porovnání s detekcí vozidel mnohem náročnějším úkolem, obzvlášť pak v reálném čase. Hlavní problém detekce chodců spočívá v samotném vzhledu chodců. Vozidla mají pevnou karoserii, která se v průběhu pohybu vozidla nijak nemění. Ve srovnání s vozidly se vzhled jednotlivých chodců může vzájemně velmi lišit. Chodci mohou nejenom nosit různé oblečení a další doplňkové předměty, ale také mohou v průběhu chůze postupně měnit své vystupování a dokonce dělat různé pózy nebo gesta. Detekce osob může probíhat například ve venkovních podmínkách s nadměrně přeplněným pozadím scény. Situaci může ještě více zkomplikovat měnící se osvětlení venkovní scény a různé povětrnostní podmínky. Ještě je nutné zmínit, že chodci jsou ve scéně obvykle prezentováni v nízkém rozlišení s ohledem na vzdálenost kamery. V případě výskytu davu lidí může docházet ke

vzájemnému překrývání osob. Vždy je nutné provést úspěšnou detekci a následné sledování chodce za pohybu a v různých úhlech pohledu. Prováděná detekce musí být použitelná v reálném čase, aby byla posléze použitelná v praktických aplikacích. V současné době se technologie pro detekci chodců využívají především v bankách, nákupních centrech, dopravě a pak hlavně na místech s vysokou ostrahou.

Pokud jde o segmentaci chodců ve scénách zachycených stacionárními kamerami, tak lze použít přístupy k modelování pozadí a segmentaci objektů v popředí podobně jako v předchozí části. Základní rozdíl v detekci vozidel a osob spočívá ve fázi klasifikace objektu. Ke klasifikaci chodců v rámci scény se v současnosti využívá různých typů klasifikátorů, které se od sebe odlišují především v přesnosti detekce [4]. Za účelem klasifikace lze využít například kaskádový klasifikátor. Při použití tohoto přístupu se postupně kombinuje více různých klasifikátorů, aby bylo možné odhadnout finální klasifikaci, jak je znázorněno na obrázku 2.6. Kombinací více klasifikátorů s menší sadou funkcí lze postupně získat pokročilejší a především efektivnější klasifikátor. V tomto případě klasifikátory používají hlavně pohybové vzory a také zájmové prvky vzhledu objektů. Sledovaný objekt je klasifikován jako chodec pouze v případě, že byl rozpoznán všemi použitými klasifikátory.



Obrázek 2.6: Ukázka kaskádového klasifikátoru určeného k detekci chodců.

Kapitola 3

Existující řešení

Účelem této kapitoly je představení již existujících aplikací, které umožňují analýzu provozu a zpracování trajektorií jednotlivých účastníků provozu. Hlavním důvodem ukázky alternativních řešení je možnost získat přehled o přístupech k analýze provozu, kterých tato řešení využívají. Následně je možné se těmito řešeními do jisté míry inspirovat a zakomponovat do své aplikace některé jejich zajímavé prvky jak z hlediska samotné funkcionality, tak z hlediska jejich uživatelského rozhraní. Uživatelské rozhraní těchto aplikací obvykle prošlo důkladným uživatelským testováním. Z tohoto důvodu pravděpodobně bude uživatelské rozhraní aplikace intuitivní a především pohodlné. Je tedy hned několik důvodů, proč se těmito aplikacemi inspirovat ve svém řešení. Nicméně je nutné zohlednit hlavní odlišnosti těchto alternativních aplikací, protože například funkcionalita se může v některých ohledech značně odlišovat. Z toho důvodu se může odlišovat také jejich uživatelské rozhraní.

Dále je potřeba zmínit vlastnosti a funkcionalitu, ve kterých mají dané alternativní aplikace omezení a nedostatky, a proč je vlastně potřeba implementovat novou aplikaci určenou k analýze provozu a zpracování trajektorií. Nově vzniklá konkurenční aplikace by měla v ideálním případě přinášet alespoň některé výhody a vylepšení v porovnání s již existujícími řešeními. V následujícím textu postupně představíme nejvýznamnější alternativní řešení, která slouží k analýze provozu stejně jako aplikace navržená v rámci této práce. Funkcionalita těchto programů obvykle nabízí mnohem více nástrojů a funkcí, protože se jedná o velmi rozsáhlé projekty vyvíjené zkušenými týmy vývojářů.

3.1 Miovision

Miovision [19] je společnost zabývající se získáváním a analýzou dopravních dat. Jedná se o společnost, která se problematikou dopravních dat zabývá velmi dlouhou dobu a poskytuje rovnou několik různých aplikací pro získávání a zpracování dopravních dat. Aplikace poskytované touto společností se zabývají především zpracováním dopravních dat v rámci městské infrastruktury s cílem zlepšit plynulost provozu ve velkých městech a docílit maximální bezpečnosti účastníků provozu. Dopravní data jsou získávána pomocí pořízených kamerových záznamů a společnost dokonce nabízí zařízení, díky kterým je možné automatizovaně získávat dopravní data.

Jedním z hlavních produktů společnosti Miovision je aplikace DataLink poskytující celou platformu pro práci s dopravními daty. Přístup k této platformě je možný pomocí DataLink portálu, což je prostředí určené k získávání dopravních dat a statistik v rámci uživatelem zvolených lokací. Tato platforma dále poskytuje správu projektů, na kterých se uživatel

v historii podílel, a s jakými dopravními daty pracoval. Hlavní myšlenkou této platformy je podání žádosti o dopravní data v lokaci, která je pro uživatele důležitá. Samozřejmě je možné získat nejenom statistická dopravní data, ale také obrazové záznamy ve zvolené oblasti. Cílovou lokaci si může uživatel jednoduše vybrat na mapě a následně zaslat žádost o odpovídající dopravní informace. U této platformy je uživatelům poskytováno úložiště získaných dat, které je organizováno do posloupnosti zmíněných projektů. Na obrázcích 3.1 a 3.2 lze vidět znázornění dopravních statistických dat získaných pomocí této platformy.

AVENUE RD & YORKVILLE AVE - TMC
Wed Nov 30, 2016
Full Length (7:30AM-9:30AM, 4PM-6PM)
All Classes (Articulated Trucks, Bicycles on Crosswalk, Bicycles on Road, Buses, Lights, Pedestrians, Single-Unit Trucks)
All Movements
ID: 369071, Location: 43.670335, -79.394808

miovision rethink traffic

Leg	AVENUE RD						YORKVILLE AVE						AVENUE RD						BUILDING ACCESS						
Direction	Southbound						Westbound						Northbound						Eastbound						
Time	R	T	L	U	App	Ped*	R	T	L	U	App	Ped*	R	T	L	U	App	Ped*	R	T	L	U	App	Ped*	Int
2016-11-30 7:30AM	1	342	0	0	**343**	11	7	2	31	0	**40**	16	0	130	1	0	**131**	17	2	0	3	0	**5**	15	**519**
7:45AM	1	364	0	0	**365**	18	13	2	30	0	**45**	19	0	140	0	0	**140**	12	1	0	1	0	**2**	25	**552**
Hourly Total	2	706	0	0	**708**	29	20	4	61	0	**85**	35	0	270	1	0	**271**	29	3	0	4	0	**7**	40	**1071**
8:00AM	5	368	0	0	**373**	27	14	2	34	0	**50**	20	0	129	0	0	**129**	7	1	0	3	0	**4**	24	**556**
8:15AM	2	378	0	0	**380**	27	19	0	45	0	**64**	31	0	154	0	0	**154**	18	0	0	2	0	**2**	47	**600**
8:30AM	3	377	0	0	**380**	47	22	1	43	0	**66**	30	0	120	0	0	**120**	29	4	0	1	0	**5**	43	**571**
8:45AM	3	384	0	0	**387**	46	25	0	38	0	**63**	27	0	104	0	0	**104**	18	2	0	4	0	**6**	58	**560**
Hourly Total	13	1507	0	0	**1520**	147	80	3	160	0	**243**	108	0	507	0	0	**507**	72	7	0	10	0	**17**	172	**2287**
9:00AM	3	317	0	0	**320**	31	22	1	42	0	**65**	32	0	97	1	0	**98**	21	2	0	3	0	**5**	46	**488**
9:15AM	4	339	0	0	**343**	29	22	1	33	0	**56**	32	0	140	0	0	**140**	22	0	0	2	0	**2**	39	**541**
Hourly Total	7	656	0	0	**663**	60	44	2	75	0	**121**	64	0	237	1	0	**238**	43	2	0	5	0	**7**	85	**1029**
4:00PM	0	166	0	0	**166**	48	38	1	46	0	**85**	46	0	261	0	0	**261**	22	4	0	4	0	**8**	52	**520**
4:15PM	2	179	0	0	**181**	30	42	2	46	0	**90**	33	0	244	0	0	**244**	38	2	0	4	0	**6**	37	**521**
4:30PM	5	159	0	1	**165**	41	38	0	50	0	**88**	44	0	227	0	1	**228**	26	3	0	1	0	**4**	49	**485**
4:45PM	4	172	0	0	**176**	53	48	4	52	0	**104**	70	0	257	0	0	**257**	36	1	0	4	0	**5**	38	**542**
Hourly Total	11	676	0	1	**688**	172	166	7	194	0	**367**	193	0	989	0	1	**990**	122	10	0	13	0	**23**	176	**2068**
5:00PM	1	190	0	0	**191**	43	41	1	46	0	**88**	52	0	249	1	0	**250**	47	4	0	5	0	**9**	74	**538**
5:15PM	0	156	0	1	**157**	45	33	1	51	0	**85**	60	0	239	0	0	**239**	26	0	0	1	0	**1**	39	**482**
5:30PM	0	184	0	1	**185**	38	43	2	50	0	**95**	48	0	260	1	0	**261**	35	2	0	2	0	**4**	41	**545**
5:45PM	1	190	0	1	**192**	44	43	3	46	0	**92**	60	0	262	1	0	**263**	34	2	0	3	0	**5**	61	**552**
Hourly Total	2	720	0	3	**725**	170	160	7	193	0	**360**	220	0	1010	3	0	**1013**	142	8	0	11	0	**19**	215	**2117**
Total	35	4265	0	4	**4304**	578	470	23	683	0	**1176**	620	0	3013	5	1	**3019**	408	30	0	43	0	**73**	688	**8572**
% Approach	0.8%	99.1%	0%	0.1%	-	-	40.0%	2.0%	58.1%	0%	-	-	0%	99.8%	0.2%	0%	-	-	41.1%	0%	58.9%	0%	-	-	-
% Total	0.4%	49.8%	0%	0%	**50.2%**	-	5.5%	0.3%	8.0%	0%	**13.7%**	-	0%	35.1%	0.1%	0%	**35.2%**	-	0.3%	0%	0.5%	0%	**0.9%**	-	-
Lights	35	4111	0	4	**4150**	-	449	20	645	0	**1114**	-	0	2908	5	1	**2914**	-	27	0	40	0	**67**	-	8245
% Lights	100%	96.4%	0%	100%	**96.4%**	-	95.5%	87.0%	94.4%	0%	**94.7%**	-	0%	96.5%	100%	100%	**96.5%**	-	90.0%	0%	93.0%	0%	**91.8%**	-	96.2%
Single-Unit Trucks	0	45	0	0	**45**	-	12	2	18	0	**32**	-	0	36	0	0	**36**	-	2	0	3	0	**5**	-	118
% Single-Unit Trucks	0%	1.1%	0%	0%	**1.0%**	-	2.6%	8.7%	2.6%	0%	**2.7%**	-	0%	1.2%	0%	0%	**1.2%**	-	6.7%	0%	7.0%	0%	**6.8%**	-	1.4%
Articulated Trucks	0	3	0	0	**3**	-	1	0	0	0	**1**	-	0	4	0	0	**4**	-	0	0	0	0	**0**	-	8
% Articulated Trucks	0%	0.1%	0%	0%	**0.1%**	-	0.2%	0%	0%	0%	**0.1%**	-	0%	0.1%	0%	0%	**0.1%**	-	0%	0%	0%	0%	**0%**	-	0.1%
Buses	0	36	0	0	**36**	-	2	0	1	0	**3**	-	0	30	0	0	**30**	-	0	0	0	0	**0**	-	69
% Buses	0%	0.8%	0%	0%	**0.8%**	-	0.4%	0%	0.1%	0%	**0.3%**	-	0%	1.0%	0%	0%	**1.0%**	-	0%	0%	0%	0%	**0%**	-	0.8%
Bicycles on Road	0	70	0	0	**70**	-	6	1	19	0	**26**	-	0	35	0	0	**35**	-	1	0	0	0	**1**	-	132
% Bicycles on Road	0%	1.6%	0%	0%	**1.6%**	-	1.3%	4.3%	2.8%	0%	**2.2%**	-	0%	1.2%	0%	0%	**1.2%**	-	3.3%	0%	0%	0%	**1.4%**	-	1.5%
Pedestrians	-	-	-	-	-	570	-	-	-	-	-	620	-	-	-	-	-	404	-	-	-	-	-	682	
% Pedestrians	-	-	-	-	-	98.6%	-	-	-	-	-	100%	-	-	-	-	-	99.0%	-	-	-	-	-	99.1%	
Bicycles on Crosswalk	-	-	-	-	-	8	-	-	-	-	-	0	-	-	-	-	-	4	-	-	-	-	-	6	
% Bicycles on Crosswalk	-	-	-	-	-	1.4%	-	-	-	-	-	0%	-	-	-	-	-	1.0%	-	-	-	-	-	0.9%	

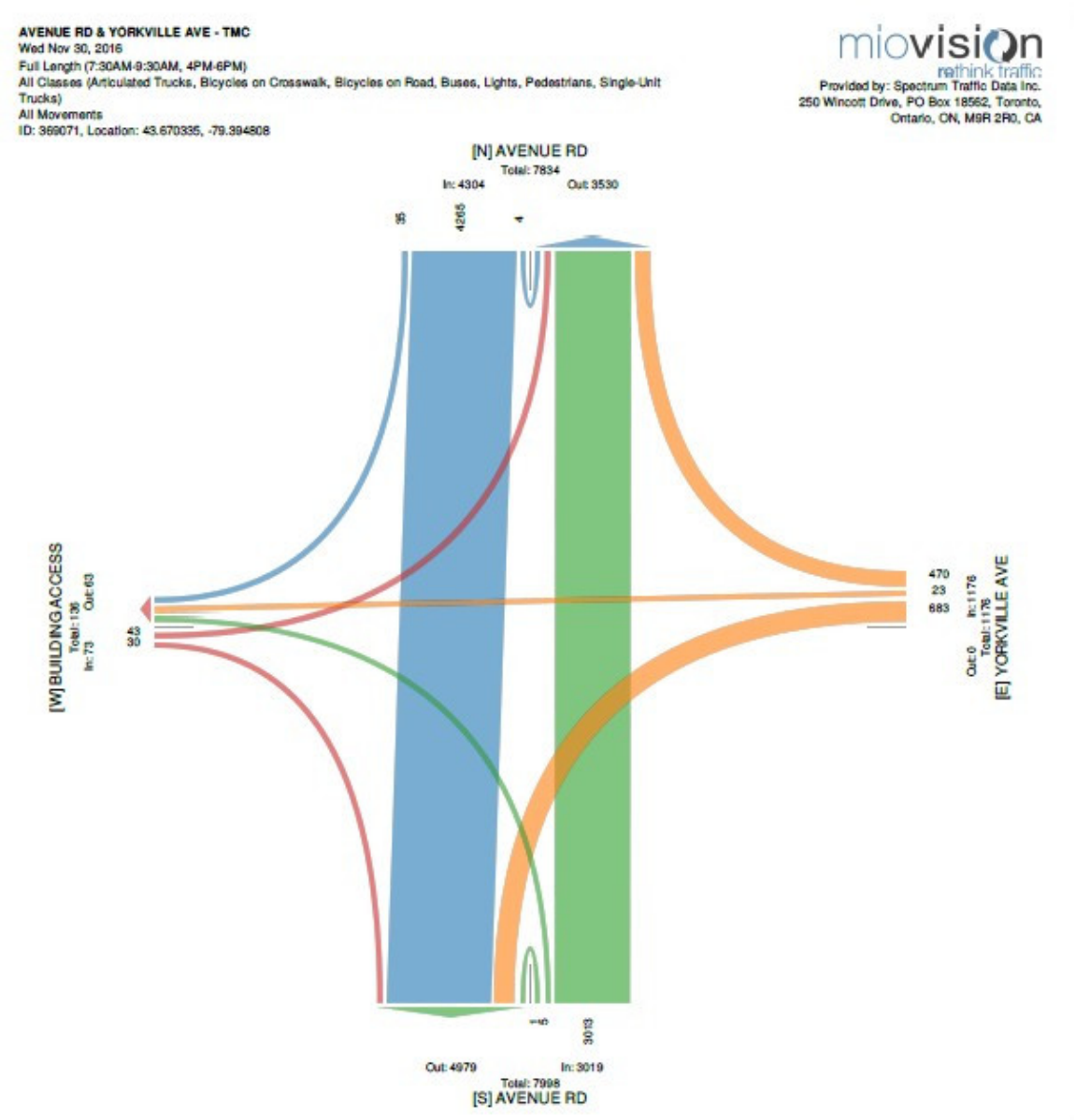
*Pedestrians and Bicycles on Crosswalk. L: Left, R: Right, T: Thru, U: U-Turn.

Obrázek 3.1: Ukázka statistických dopravních dat z aplikace od společnosti Miovision (převzato z [19]).

Dalším produktem této společnosti je aplikace TrafficLink, která je více zaměřena na optimalizaci dopravy ve velkých městech. Tato platforma je tedy více praktická a zaměřuje se na inteligentní řízení provozu v rámci městské infrastruktury. Řízení provozu je prováděno na základě dat získaných s pomocí umělé inteligence ze záznamů pořízených kamerami. Cílem této platformy je řídit dopravu automatizovaně a samozřejmě vzdáleně díky použití různých signálů. Tato společnost u svých platforem dále nabízí odolné mobilní zařízení zvané Scout, což je přenosné zařízení schopné získávat obrazové záznamy, na základě kterých je následně prováděna analýza provozu a získávání statistik. Kontrola tohoto zařízení je

ve velké míře automatizovaná a je samozřejmě možné ho ovládat vzdáleně. Cílem tohoto zařízení je zvýšit efektivitu samotné analýzy provozu a získat velmi přesná dopravní data.

Hlavní výhodou produktů od této společnosti je jejich intuitivnost a relativně snadné použití. Pro získání dopravních dat a statistik stačí pouze vybrat na mapě požadovanou lokaci. K získání statistik nemusí uživatel vlastnit kamerové vybavení schopné zachycovat dopravní situaci a ani nepotřebuje mít dopravní záznamy. Stačí si v portálu pouze zvolit pozici na mapě a následně mít rychlý přístup k dopravním statistikám. Nevýhodou této platformy je naopak značná závislost na technické podpoře společnosti. Je samozřejmě možné specifikovat oblast na mapě a poté specifikovat požadovaná data, ale získání dat a vypočtení statistik je následně úkolem na straně portálu a samotné společnosti.

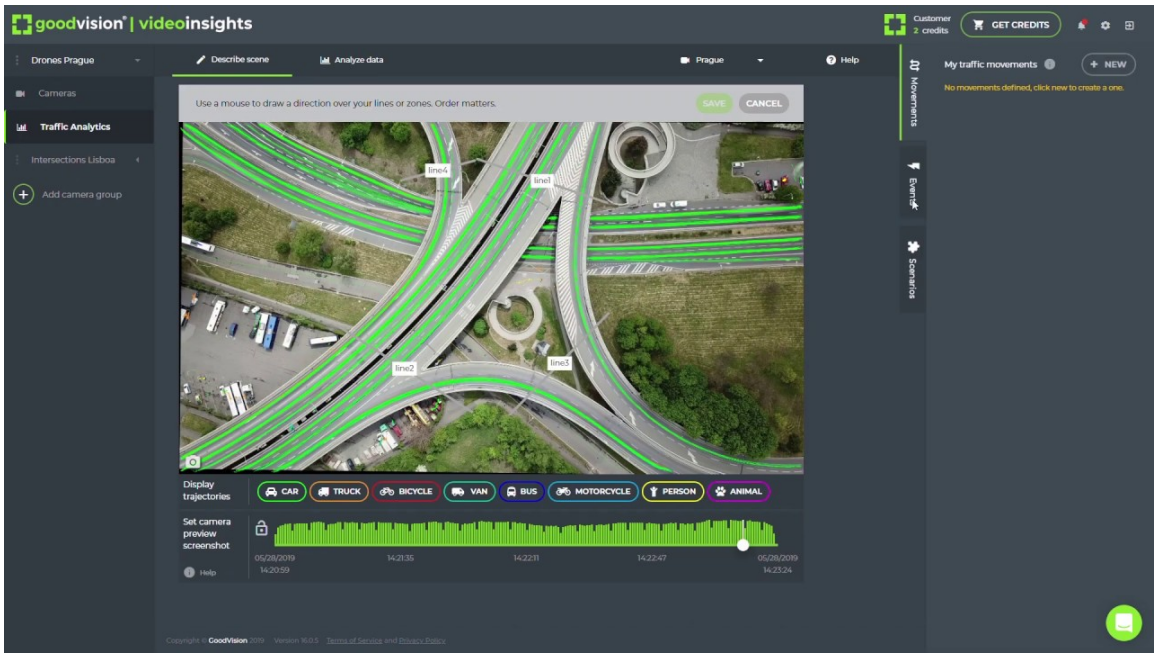


Obrázek 3.2: Ukázka grafu s dopravními daty z aplikace od společnosti Miovision (převzato z [19]).

3.2 GoodVision Video Insights

Společnost GoodVision [8] se zabývá vývojem interaktivní aplikace Video Insights, díky které je možné analyzovat získaná data účastníků provozu. Vstupem této aplikace je nahrávka pořízená pomocí dopravní kamery nebo letecký snímek pořízený dronem. Po volbě vstupního souboru dochází k automatické extrakci dopravních dat z dané nahrávky. Analýza vstupních dat probíhá pomocí proprietární umělé inteligence a algoritmů na zpracování obrazu, které jsou také vyvíjeny touto společností. Cílem společnosti je poskytnout rychlý přístup k detailním dopravním datům v porovnání s použitím indukčních smyček nebo dopravních radarů, které poskytují pouze omezené výstupy.

Aplikace poskytuje jak vizualizaci, tak také filtraci dopravních dat a následné zobrazení statistických dat. V rámci statistiky je umožněno získání nejenom počtu přítomných účastníků provozu, ale je možné zobrazit pokročilé metriky jako je vyhodnocení rychlosti vozidel. Je podporována analýza chodců a cyklistů stejně jako v případě vozidel. Samotné výstupy analyzovaných dat jsou nejenom v podobě počtů vozidel a jejich parametrů, ale data jsou vizualizována pomocí různých typů grafů. Díky těmto grafům a dalším diagramům lze názorně pozorovat vývoj dopravní situace v průběhu času analýzy. Vizualizace dopravních dat je detailní a především přehledná. Uživatel si může na časové ose zvolit libovolnou část nahrávky a případně si pozastavit analýzu a vytvořit snímek obrazovky včetně vykreslených trajektorií.

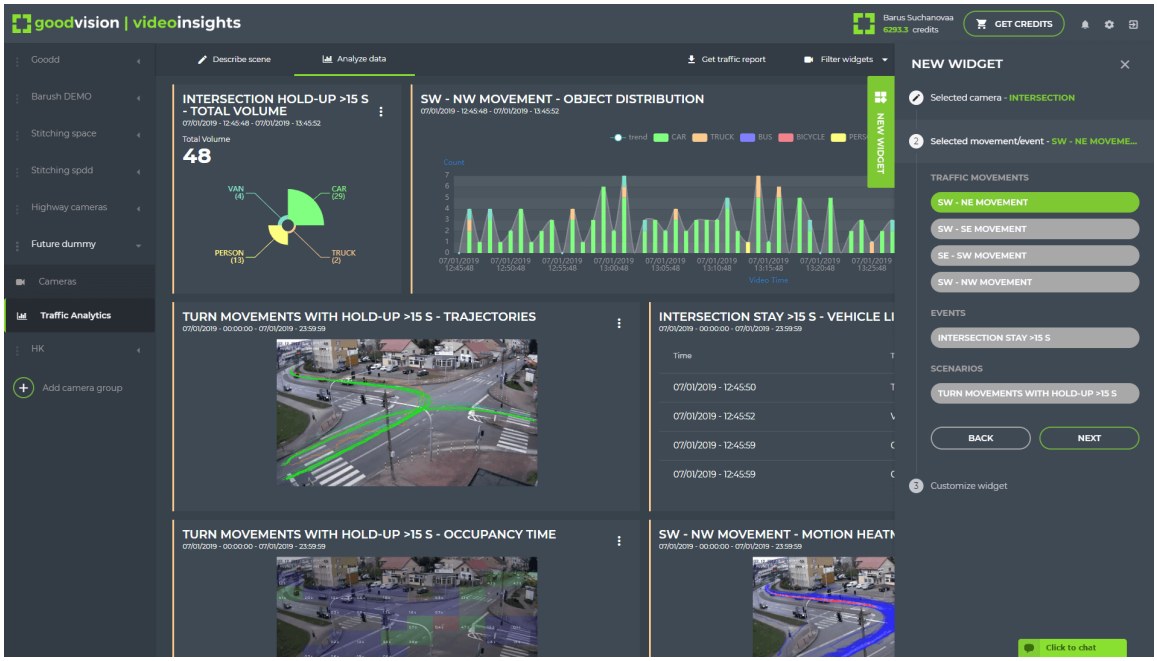


Obrázek 3.3: Filtrace pomocí aplikace Video Insights od společnosti GoodVision (převzato z [8]).

Samotná vizualizace účastníků provozu dále zahrnuje vykreslování trajektorií zachycených objektů na vozovce. V případě každého rozpoznaného objektu může být zobrazen typ objektu. Barva trajektorie objektu se vždy odvíjí od jeho typu. Získané trajektorie je poté možné filtrovat. Kromě filtrace na základě barvy a kategorie objektu je možné aplikovat prostorovou filtraci, která probíhá na základě pokročilých filtrů. Aplikace podporuje pokročilé

filtrování díky prostorovým elementům, mezi které patří zóna a dále pak směrový element, což je křivka znázorňující dráhu pohybu objektů. Ukázku filtrace pomocí této aplikace lze vidět na obrázku 3.3.

Výstupy získané filtrací lze přehledně vizualizovat díky různým typům widgetů, což je znázorněno na obrázku 3.4. Mezi podporované widgety patří například distribuční widget znázorňující četnosti objektů v závislosti na čase analýzy. Distribuce těchto hodnot je vizualizována v podobě grafu, ve kterém jsou barevně znázorněny četnosti objektů v závislosti na čase. Také je přítomen widget zobrazující množství různých typů objektů v průběhu zvoleného časového intervalu analýzy. Všechny vytvořené widgety jsou zobrazovány v rámci dopravní analytiky, kde lze widgety libovolně přeskupovat dle preferencí uživatele. Pomocí vytvořených widgetů lze rychle a přehledně prezentovat získané výsledky jiným uživatelům.



Obrázek 3.4: Vizualizace dat v aplikaci Video Insights od společnosti GoodVision (převzato z [8]).

Hlavní výhodou aplikace Video Insights je právě filtrace doplněná o přehlednou vizualizaci výstupů filtrace. Především prostorové filtrování je velmi užitečné a uživatel má možnost vytvářet nové elementy podle svých představ. Vizualizace analyzovaných dat je na velmi dobré úrovni. Především vytváření vlastních výstupních widgetů je velkou výhodou této aplikace. Uživatelské rozhraní aplikace je poměrně intuitivní a je cíleno především na pokročilé možnosti nastavení vizualizace filtrovaných dat. Mírným omezením této aplikace je nemožnost provádět pokročilou filtraci díky kombinování filtrů. Vytvořené filtry nelze snadno propojovat a vytvářet tak filtry složitější, které by mohly být v některých případech velmi užitečné.

3.3 BriefCam Video Analytics

BriefCam [3] je společnost zabývající se zpracováním videa pomocí neuronových sítí a algoritmů pro zpracování obrazu. V tomto textu si představíme jejich aplikaci Video Analytics. Nejedná se pouze o jedinou aplikaci, ale rovnou o celou platformu určenou ke zpracování obrazu díky použití umělé inteligence. Ze zpracovaného obrazu lze následně získat potřebná data a počty rozpoznaných objektů. Navíc je možné rychle a efektivně vyhledávat nad získanými daty. Hlavním cílem celé platformy je především detekce objektů. Je možné detekovat nejenom vozidla a chodce, ale jsou poskytovány nástroje na rozpoznání obličejů osob. Platforma samozřejmě podporuje detekci různých typů vozidel z dopravních záznamů.

V porovnání s konkurencí tato platforma nabízí možnost vyhledávání konkrétního objektu ve více záznamech. V případě detekce a vyhledávání chodců je například možná detekce pohlaví, barvy oblečení a dalších zvolených aspektů. Podporována je také opětovná detekce stejné osoby v celé množině záznamů. Monitorování dopravy naopak nabízí možnost rozpoznání poznávacích značek vozidel. Na základě značek je možné vyhledávat vozidla v celé databázi pořízených záznamů. Další užitečnou poskytovanou funkcí je výpočet vzdálenosti mezi detekovanými objekty, což zahrnuje jak vozidla, tak především chodce. Schopnosti detekování objektů jsou v případě této platformy opravdu velmi rozsáhlé.

Platforma dále poskytuje možnost seskupování a filtrace detekovaných objektů. Možnosti filtrace jsou opravdu pokročilé. Samozřejmostí je filtrace na základě různých atributů objektů, mezi které patří barva, kategorie, doba výskytu a další atributy. Je také podporována pokročilá prostorová filtrace. Lze filtrovat na základě směru pohybu a také je možné detekovat průnik zvolené oblasti. U všech typů filtrace je navíc možné nastavit hodnotu tolerance.

Obrázek 3.5: Vizualizace dat pomocí aplikace Video Analytics od společnosti BriefCam (převzato z [3]).

Detekce a filtrace je navíc doplněna o možnost přehledné vizualizace dat v několika vrstvách, v rámci kterých je možné zobrazovat grafy znázorňující průběh analýzy obrazu.

Ukázku vizualizace dat lze vidět na obrázku 3.5. Výhod této platformy je opravdu mnoho. Ve srovnání s konkurencí je navíc nabízena inovativní funkcionalita v podobě vyhledávání objektů ve více záznamech najednou a také sledování objektů napříč záznamy. Jistou nevýhodou celé platformy je její vysoká orientovanost na detekci a vyhledávání objektů v obrazových záznamech, což je na jednu stranu výhoda, ale slabinou celé platformy je relativně omezená filtrace objektů. Filtrování objektů je samozřejmě možné, ale není možné vytvářet pokročilé filtry jejich kombinováním a propojováním.

3.4 Shrnutí konkurence

Pro představená alternativní řešení je společné uživatelsky přívětivé rozhraní a ovládání aplikace. Právě při návrhu uživatelského rozhraní naší aplikace je doporučeno zohlednit přístupy používané konkurenčními aplikacemi. Především vizualizace dat pomocí widgetů je z hlediska prezentování výsledků analýzy velmi užitečná, a proto je dobré zvážit implementaci těchto vizualizačních prvků. Z hlediska funkcionality se v případě představených aplikací jedná o rozsáhlé projekty, které nabízejí velké množství funkcí. Všechna popsaná alternativní řešení mají omezení v případě pokročilé prostorové filtrace. Uživatel obvykle nemá možnost vytvořit pokročilý filtr složený z několika základních filtrů. Právě na tuto funkcionalitu je vhodné se zaměřit v případě implementace naší aplikace, protože právě díky ní se můžeme odlišit od konkurence.

Kapitola 4

Filtrace trajektorií

Při tvorbě textu v této kapitole bylo využito informací získaných v [26]. Také bylo využito některých principů popsaných v [12].

V rámci této práce je důležitým úkolem vytvoření formalismu, pomocí kterého je možné specifikovat filtrování výstupů získaných při analýze dopravy. Bez schopnosti filtrovat trajektorie se v zásadě nemůžeme obejít, protože ne vždy nás zajímají všechna získaná dopravní data, ale musíme mít možnost pracovat s jejich podmnožinou. Navíc zpracování celé množiny získaných dopravních dat může být velmi náročné na výpočetní výkon, což je dalším důvodem k využití filtrování výsledků.

4.1 Návrh formalismu

Jak již bylo zmíněno, tak provedením analýzy dopravy získáváme množinu trajektorií objektů, které byly součástí pořízeného záznamu. Těmito objekty jsou obvykle vozidla, která se pohybující v rámci snímku. Naším úkolem při filtraci trajektorií je reprezentace konkrétní trajektorie. Trajektorie pohybu objektu je sice pouze křivka znázorňující pohyb zachyceného objektu, ale u každé konkrétní trajektorie nás zajímá více atributů, které s ní souvisejí [2]. Těmito atributy jsou hodnoty popisující konkrétní zachycený objekt. Pokud křižovatkou projede automobil, tak nás nemusí zajímat pouze dráha jeho pohybu, ale také jeho typ, barva, rychlost, zrychlení a dále například délka stání vozidla. Je tedy potřeba navrhnout takovou strukturu, díky které bude možné popsat konkrétní objekt včetně množiny vyžadovaných atributů tohoto objektu.

4.1.1 Reprezentace získaných dat

Důležitým úkolem je efektivně reprezentovat samotnou dráhu pohybu vozidla. Při filtrování trajektorií musíme být schopni provést detekci protnutí brány nebo regionu danou trajektorií. Může nás totiž zajímat jenom konkrétní oblast snímku, a tudíž musíme být schopni zohlednit jenom takové trajektorie, které region skutečně protínají [10]. Zároveň nesmíme zapomínat na to, že můžeme mít velmi vysoké množství trajektorií, které je nutné uložit a efektivně zpracovat. K reprezentaci dráhy pohybu je nutné zvolit vhodnou strukturu, která bude efektivní na zpracování a nebude vyžadovat velké množství diskového prostoru k uložení. Tato dráha by se například dala reprezentovat jako sekvence vzájemně propojených úseček, které je možné uložit v podobě množiny řídících bodů. Pomocí této reprezentace jsme totiž schopni nejenom dostatečně přesně popsat směr pohybu, ale také jsme schopni efektivně implementovat detekci průniku s bránou či oblastí. Pro detekování průniku nám

stačí algoritmus na průnik polygonu s úsečkou. Je sice nutné porovnat průnik s každou úsečkou v sekvenci, ale tuto výpočetní úlohu je možné velmi efektivně rozdělit mezi více výpočetních jednotek a dosáhnout tak vysoké rychlosti zpracování. Pro uložení celé sekvence úseček nám bude stačit pouze N řídících bodů, kde $N - 1$ je počet všech úseček v sekvenci. Tímto jsme dokázali navrhnout reprezentaci trajektorie jednotlivých objektů, nicméně nyní je důležité provést návrh efektivního filtrování získaných trajektorií.

4.1.2 Statické atributy

Bylo již řečeno, že v rámci reprezentace objektu musíme ukládat atributy zachyceného objektu. Na základě právě těchto atributů musíme být schopni vybrat jen takové trajektorie, které jsou pro nás zajímavé. Pokud je cílem analýzy pouze detekování chodců na přechodu, tak není žádoucí zpracovávat trajektorie projíždějících vozidel. Je tedy vyžadován návrh a popis několika operátorů, pomocí kterých jsme schopni zohlednit jen zájmové trajektorie. Mezi základní typy operátorů budou patřit operátory typu a barvy objektu. V případě typu i barvy musíme být schopni akceptovat množinu atributů a ne jenom jeden konkrétní atribut. Musíme být schopni filtrovat například vozidla červené a modré barvy. Typ a barva objektu jsou statické atributy trajektorie, které se v průběhu analýzy nemění a jsou přímo vázány na konkrétní objekt. Ukázka trajektorií s různými statickými atributy je znázorněna na obrázku 4.1.

4.1.3 Dynamické atributy

V této podkapitole používáme některé z myšlenek popsaných v [23].

Kromě statických atributů nás budou zajímat atributy dynamické. Mezi dynamické atributy řadíme především rychlost, zrychlení, dobu stání vozidla a případně dobu výskytu objektu. Naše schopnost filtrovat trajektorie na základě dynamických atributů je naprosto klíčová. Musíme být například schopni spolehlivě a efektivně detekovat takové objekty, které překračují určený rychlostní práh. Pro všechny dynamické atributy je nutné provést popis operátorů s tím, že je potřeba provést filtraci v rámci zvoleného intervalu, což zahrnuje nastavení minimální a maximální akceptované hodnoty odpovídajícího atributu.

V případě dynamických atributů je situace poněkud složitější, než v případě atributů statických. Hodnota dynamického atributu se může průběžně měnit v závislosti na čase analýzy. Tato situace samozřejmě nastává velmi často, protože sledované vozidlo může například zastavit na křižovatce a následně začne zrychlovat po opuštění křižovatky. Rychlost daného vozidla je pak samozřejmě odlišná. Naším úkolem je reprezentace dynamických atributů v závislosti na čase. Za tímto účelem budeme definovat časovou masku, což je struktura, díky které jsme schopni popsat množinu všech časových úseků dané trajektorie. Jednotlivé časové úseky budeme popisovat pomocí dvojice, která se skládá z časové značky a dále délky trvání tohoto časového úseku. Délka trvání jednotlivých časových úseků samozřejmě nemusí být stejná a může se odvíjet na základě různých okolností. Časové úseky je možné prodloužit v případě nedostatku výpočetního výkonu. Úseky můžeme naopak zkrátit v situaci, kdy chceme získat co nejvyšší přesnost zpracování výsledků.

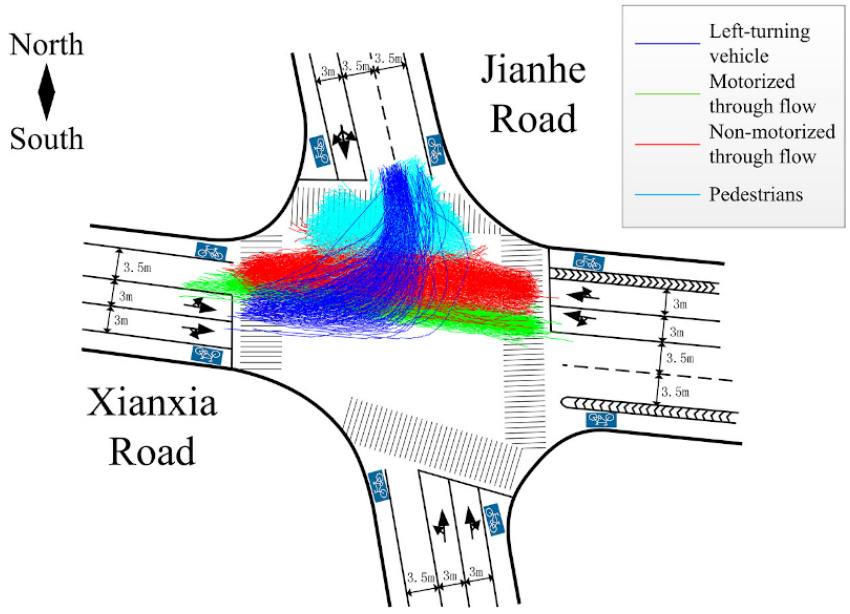
Díky časové masce jsme schopni pracovat se stavem trajektorie ve zvoleném časovém období. Toto období lze definovat například pomocí intervalu ohraničeného počáteční a koncovou časovou značkou. Samozřejmě nás může zajímat pouze jediný konkrétní okamžik v rámci celé časové masky. V případě pokročilé analýzy trajektorií bychom mohli pracovat s periodickou časovou maskou, kdy by bylo možné nastavení hodnoty této periody. Případně by bylo možné pracovat s celou historií trajektorie. Každý časový úsek trajektorie zahrnuje

zmíněné dynamické atributy. V případě atributu rychlosti jsme schopni získat průměrnou rychlost objektu v rámci daného časového úseku. Nemusíme být omezeni jen na průměrné hodnoty atributů, ale také na minimální, maximální a okamžité hodnoty daných atributů. V porovnání se statickými atributy je nutné zohlednit znatelně vyšší datovou náročnost reprezentace dynamických atributů.

Časovou masku je potřeba ukládat v rámci samotné reprezentace trajektorie. Filtrace na základě dynamických atributů tedy vyžaduje práci s časovou maskou. Na základě aktuálního času analýzy jsme schopni získávat atributy právě pomocí časové masky. Získání informace o aktuální rychlosti objektu vyžaduje přístup k aktuální časové značce v masce. Proces filtrace trajektorií zahrnuje nalezení odpovídajícího dynamického atributu z masky a ověření, zda získaná hodnota náleží zvolenému rozsahu hodnot. Tímto způsobem jsme schopni mít na výstupu operátoru jen takové trajektorie, které splňují zadaná kritéria dynamických hodnot. Formální definice časové masky trajektorií bude provedena v následujícím textu.

4.1.4 Prostorová filtrace

Mezi komplikovanější operátory budeme řadit operátory prostorové. Obtížnější není pouze návrh a popis takových operátorů, ale samotná filtrace může být výpočetně velmi náročná. Navržená filtrace proto musí být efektivní a hlavně dostatečně přesná. Mezi tyto operátory zařadíme operátor brány a dále pak složitější operátor regionu. Uživatel musí mít při filtraci možnost specifikovat bránu pomocí libovolné úsečky v prostoru. Případně bude možné nakreslit libovolný uzavřený polygon, který bude reprezentovat zmíněnou oblast v prostoru. Naším úkolem je navrhnout takový operátor, který zajistí spolehlivou detekci průniku.

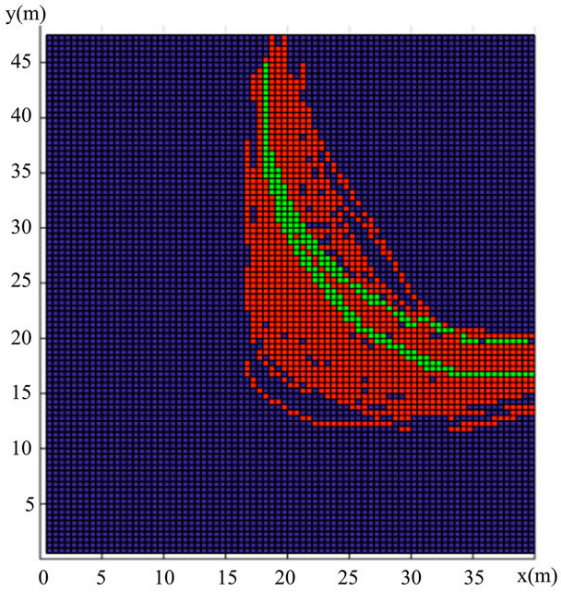


Obrázek 4.1: Ukázka získané množiny trajektorií různých typů objektů detekovaných při průjezdu křižovatkou (převzato z [26]).

Dráhu trajektorie budeme reprezentovat množinou řídících bodů, jak již bylo zmíněno výše. Díky zadaným řídícím bodům pak můžeme získat sekvenci úseček. Samotný region je reprezentován uzavřeným polygonem n-tého stupně. Uzavřený polygon budeme ukládat v podobě množiny řídících bodů jako v případě trajektorií. Detekci průsečíku trajektorie

s bránou tedy můžeme implementovat pomocí algoritmu pro nalezení průsečíku dvou úseček [17]. Protnutí polygonálního útvaru můžeme implementovat analogickým algoritmem, protože uzavřený polygon lze také brát jako množinu úseček. Algoritmus pro detekci průniku trajektorie s regionem by se tím pádem nemusel výrazně odlišovat od algoritmu určeného k průniku s bránou. Rozdílem by bylo, že v případě regionu je potřeba projít všechny úsečky daného polygonu. Jádrem celého algoritmu průniku je tedy nalezení průsečíku dvou úseček. Použití tohoto algoritmu je rozhodně výhodné z důvodu snadné implementace, ale bohužel je jeho velkou nevýhodou vysoká výpočetní náročnost v případě velkého množství trajektorií. Situace může být ještě horší v situaci, kdy analýza probíhá na počítači s velmi omezeným výpočetním výkonem. Naším cílem je navrhnout efektivnější přístup k detekci průniku s tím, že zůstane zachována dostatečná přesnost prostorové filtrace.

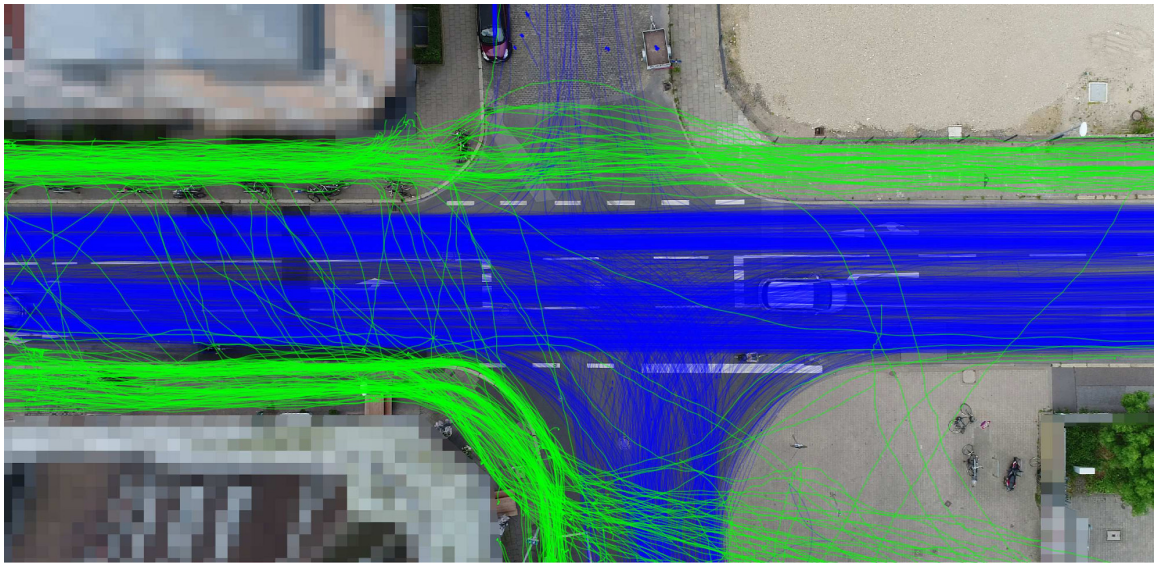
Za účelem optimalizace prostorové filtrace provedeme návrh a následný popis prostorové masky, která pracuje s binárními hodnotami a umožňuje nám převést detekci průsečíku na jednodušší a především výrazně rychlejší binární operace [7]. Samotná prostorová maska je dvourozměrná matice, která má ve svých buňkách pouze bitové hodnoty 0 a 1. Hlavní myšlenkou tohoto přístupu je reprezentovat prostor pomocí matice, kdy buňky s hodnotou 1 značí, že v odpovídající oblasti prostoru se nachází část dráhy trajektorie [9]. Buňky s hodnotou 0 naopak reprezentují oblasti, na kterých se trajektorie nenachází. Na základě zvolené přesnosti pracujeme s maticí, která reprezentuje prostor, v rámci kterého provádíme analýzu [14]. Prostor je vždy nutné rozdělit do množiny obdélníkových oblastí, kdy všechny vzniklé oblasti mají stejné rozměry, což je znázorněno na obrázku 4.2. Pro dosažení rychlejší prostorové filtrace je možné zvolit oblasti s větší plochou. Pokud naopak chceme dosáhnout vysoké přesnosti filtrace, tak je možné volit menší oblasti, na které prostor rozdělíme.



Obrázek 4.2: Ukázka dělení prostoru s množinou trajektorií na mřížku menších oblastí (převzato z [26]).

Tímto způsobem budeme samozřejmě reprezentovat také prostorové elementy. Bránu a polygonální region budeme ukládat v podobě odpovídajících buněk bitové matice, kdy oblast zabraná elementem bude popsána buňkami s hodnotou 1. Algoritmus detekce průsečíku elementu s trajektorií bude díky tomuto přístupu výrazně jednodušší, protože bude

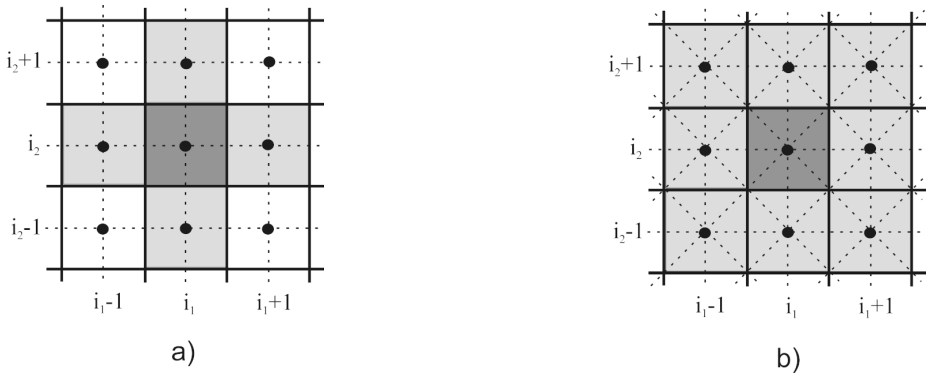
pracovat pouze s binárními maticemi. Nalezení průsečíku regionu s trajektorií bude vyžadovat pouze maticové operace, které lze velmi efektivně hardwarově akcelerovat. Pro samotné nalezení průsečíku dostačuje, aby se v matici elementu a matici trajektorie nacházela hodnota 1 na stejných indexech [9]. K detekci protnutí stačí nalezení pouze jediné shody. Na základě této myšlenky jsme schopni rychle provést prostorovou filtraci velkého množství trajektorií. Nalezení shody konkrétní trajektorie budeme také popisovat bitem, který má hodnotu 1 v případě detekce průniku [25]. Jednotlivé bity každé trajektorie budeme ukládat v podobě binárního vektoru, který bude popisovat pouze jedinou buňku matice prostorového elementu. Pokud budeme mít například region zabírající čtyři buňky matice, tak pro každou buňku potřebujeme binární vektor, který nám indikuje protnutí každé trajektorie s odpovídající oblastí v prostoru.



Obrázek 4.3: Ukázka scény s velkým množstvím trajektorií (převzato z [2]).

Velkou výhodou použití těchto bitových vektorů je provedení operace bitové disjunkce mezi všemi získanými vektory [9]. Pro úspěšné detekování průsečíku trajektorie s daným elementem nám stačí, aby došlo k nalezení shody pouze v jediné buňce matice. Díky této bitové operaci jsme velmi rychle schopni získat výsledek. Po aplikaci operace disjunkce získáme jediný bitový vektor, kdy každý n-tý bit tohoto vektoru odpovídá konkrétní trajektorii v prostoru. Zjištění toho, zda regionem prošla právě n-tá trajektorie v prostoru, vyžaduje pouze získání hodnoty bitu na odpovídajícím indexu ve vektoru. Protnutí regionu bude samozřejmě značeno hodnotou 1 daného bitu a analogicky hodnotou 0 v opačném případě. Tento navržený postup lze ještě rozšířit o použití operace bitové konjunkce, kterou lze využít v případě výskytu více prostorových elementů v rámci jedné oblasti [9]. Díky této operaci jsme schopni zjistit, zda trajektorie protíná každý z množiny elementů v dané oblasti. Tuto operaci aplikujeme na všechny bitové vektory získané pro každý element. Veškeré algoritmy na nalezení průsečíku dvou přímek jsme tedy schopni nahradit touto kombinací maticových a bitových operací, které je navíc možné spustit paralelně a docílit ještě rychlejší prostorové filtrace. Nevýhodou tohoto přístupu je samozřejmě jeho menší přesnost. Může například nastat situace, kdy je detekováno protnutí regionu trajektorií, ačkoliv ve skutečnosti k průniku nedošlo.

Pro využití tohoto přístupu k prostorové filtraci je nutné nejdříve vytvořit odpovídající prostorové masky, což zahrnuje správné nastavení bitů v buňkách matic. Bez korektní inicializace matic nelze správně provést prostorovou filtraci. Důležitým krokem při vytváření prostorové masky je určení toho, jakou část prostoru vlastně daná trajektorie zabírá [25]. Dráha trajektorie je vždy dána pouze množinou řídících bodů. Naším úkolem je reprezentace řídících bodů pomocí prostorové masky, což odpovídá nalezení všech takových oblastí v prostoru, na kterých se daná trajektorie nachází. Nalezení všech těchto oblastí vyžaduje postupný průchod řídících bodů trajektorie a získání množiny oblastí v prostoru, na kterých se nachází alespoň jeden řídící bod. Po získání této množiny oblastí je nutné zachovat postupnou návaznost jednotlivých oblastí. Musíme vždy zaručit, aby nedošlo k přerušení návaznosti mezi oblastmi. V případě porušení tohoto pravidla bychom získali reprezentaci více než jedné trajektorie, což by neodpovídalo zadaným řídícím bodům trajektorie. Pro zachování souvislosti trajektorie v masce je nutné pro každou buňku naleznout alespoň jednu sousední buňku v okolí všech osmi sousedních buněk. Získáváme tedy Moorovo sousedství dané buňky v mřížce s tím, že je nutné prohledat vždy čtyři sousední buňky po stranách a dále čtyři sousední buňky přes vrchol [16]. Tento typ sousedství je přehledně znázorněn na obrázku 4.3. V případě porušení souvislosti je ideální doplnit pouze jedinou sousední buňku, abychom zredukovali počet operací provedených při následné prostorové filtraci. Při doplnění sousedních buněk je možné přidávat sousední buňky přes hranu matice. Dráha trajektorie může například projít hranou a následně opět navazovat na opačném konci matice. Návaznost buněk je tedy nutná, ale počet buněk v matici nutných k popisu trajektorie by měl být co nejnižší, abychom snížili výpočetní nároky prostorové filtrace. Po inicializaci odpovídající prostorové masky trajektorie je dále potřeba inicializovat prostorové matice pro všechny prostorové elementy.



Obrázek 4.4: Znázornění typů sousedství ve dvourozměrném prostoru. a) von Neumannovo sousedství. b) Moorovo sousedství. (převzato z [27])

V případě vytváření prostorové matice pro bránu a region bude postup pro inicializování matice trochu odlišný od trajektorií. Oba prostorové elementy musí být samozřejmě souvislé, ale v porovnání s trajektorií nesmí buňky navazovat za hranou. Je nežádoucí, aby byl region rozdělen hranou matice a musíme se této situaci vyhnout. Vytvořený region musí být vždy uzavřený, a proto budeme doplňovat sousední buňky pouze v rozsahu čtyř sousedních buněk po stranách [27]. Tento typ sousedství je označován jako von Neumannovo sousedství, které je znázorněno na obrázku 4.3. Není tedy možné navazovat sousedství pomocí sousedních buněk přes vrchol, ale pouze po stranách [16]. Právě tímto způsobem zajistíme uzavřenost prostorového elementu a vyhneme se případným neuzavřeným hranám

elementů. V případě polygonálních regionů je uzavřenost oblasti dostačující a již není potřeba vyplňovat případný zbytek vnitřní plochy regionu, protože nás zajímá pouze průsečík trajektorie s ohraničením polygonu.

Po procesu vytváření prostorových masek je nutné provést jejich uložení. V případě trajektorie budeme prostorovou masku ukládat ve struktuře dané trajektorie stejně jako ukládáme masku časovou. U prostorových elementů je vždy nutné ukládat prostorovou masku daného elementu. Každá brána nebo region jsou vždy závislé na odpovídajícím prostorovém operátoru, a proto je prostorová maska součástí právě tohoto operátoru. U každého prostorového operátoru je dále nutné uložit vypočtené bitové vektory prostorové filtrace. Na základě těchto vektorů dochází k určení toho, které trajektorie budou ve výstupní množině operátoru. Pomocí popsané masky jsme tedy schopni vymaskovat jen takovou část trajektorie, v rámci které dochází k průsečíku s prostorovým elementem.

4.1.5 Kombinace operátorů

Při analýze provozu ale může nastat situace, kdy bude naším úkolem provést filtraci pouze červených osobních automobilů, které mají průměrnou rychlost menší než je zvolený práh. Na takovou filtraci nám jediný operátor nestačí a je potřeba použít tři různé operátory najednou. V případě každého operátoru získáme množinu filtrovaných trajektorií, nicméně dále je potřeba výsledky reprezentovat pouze jedinou množinou výstupních trajektorií. Za tímto účelem provedeme návrh booleovských operátorů, díky kterým je možné shromažďovat výsledky z různých operátorů dohromady. Pro realizaci výše zmíněné filtrace provedeme návrh booleovského operátoru průniku, který bude mít na svém vstupu množiny akceptovaných trajektorií z více různých operátorů. V takovém případě budou na výstupu tohoto operátoru jenom takové trajektorie, které se vyskytují v každé jeho vstupní množině trajektorií. Analogicky je možné provést definici booleovského operátoru sjednocení, kdy výstupem tohoto operátoru bude množina takových trajektorií, které se nacházejí alespoň v jedné vstupní množině operátoru. Třetím z booleovských operátorů bude operátor doplňku. Na výstupu tohoto operátoru budou takové trajektorie, které se nenacházejí ve vstupní množině operátoru. Operace doplňku je prováděna vůči originální množině všech trajektorií v rámci analýzy.

Díky přítomnosti booleovských operátorů jsme dokonce schopni vytvářet stromovou strukturu složenou z různých operátorů. Každým uzlem tohoto stromu bude konkrétní operátor. Listovými uzly takového stromu pak budou právě takové operátory, jejichž výstupy nás budou zajímat v rámci zpracování výsledků analýzy. Kořenem stromové struktury bude vstupní množina trajektorií získaných při analýze provozu. Z každého uzlu stromu jsme schopni propojit výstupní množinu odpovídajícího operátoru se vstupem operátoru jiného. Za pomoci booleovských operátorů naopak můžeme získané výstupy spojit dohromady do jediné výstupní množiny. Velkou výhodou takové stromové struktury je její poměrně snadná implementace. Vyhledávání v této struktuře je velmi rychlé a režie spojená s uložením je naopak nízká. Úprava již existujícího stromu je poměrně snadná. Přidávání a odstraňování listových uzlů je rychlé a jednoduché. Úprava již existujících uzlů stromu je samozřejmě možná. Libovolný operátor je možné kdykoliv nahradit operátorem jiným a to bez narušení konceptu této stromové struktury. Další velkou výhodou tohoto přístupu aplikování operátorů je jeho intuitivnost. Vytvořit jednodušší filtrovací struktury by mohla zvládnout většina uživatelů bez výraznějších problémů. V následujícím textu provedeme formální popis jednotlivých operátorů a také všech jejich atributů.

4.2 Popis formalismu

Cílem této kapitoly je definice formalismu umožňujícího filtraci trajektorií. Formalismus budeme popisovat v návaznosti na jeho návrh v předcházející části textu. Postupně budeme definovat jednotlivé základní struktury, ze kterých budeme následně konstruovat struktury složitější. Budeme zde pracovat s obecným počtem prvků ve strukturách, kdy skutečný počet prvků se bude moci měnit v závislosti na použité aplikaci. Hlavním účelem definice níže uvedeného formalismu je jeho podklad pro návrh a následnou implementaci grafického uživatelského rozhraní, které bude z formalismu vycházet a bude také umožňovat filtraci trajektorií. V této kapitole budeme definovat jenom teoretickou část filtrace. Samotným návrhem a implementací zmíněného uživatelského rozhraní se budeme zabývat až v následujících kapitolách, kdy se budeme k myšlenkám a postupům popsaných v této kapitole opět vracet.

4.2.1 Prostorové elementy

Bod v prostoru můžeme zadefinovat jako uspořádanou dvojici horizontální a vertikální souřadnice naší scény a to následujícím způsobem:

$P = (X, Y)$ kde:

- X: horizontální souřadnice bodu v prostoru, kde $X \in N$,
- Y: vertikální souřadnice bodu v prostoru, kde $Y \in N$

Pomocí uspořádané dvojice počátečního a koncového bodu budeme reprezentovat úsečku v prostoru:

$L = (P_1, P_2)$ kde:

- P_1: počáteční bod úsečky,
- P_2: koncový bod úsečky

Region v prostoru budeme reprezentovat jako uzavřený polygon alespoň třetího stupně. Samotnou definici polygonu provedeme pomocí uspořádané n-tice bodů, které jsou propojeny úsečkami. Vždy je nutné propojit počáteční a koncovou úsečku, aby došlo k uzavření vzniklé oblasti. Níže můžeme vidět definici regionu:

$R = (P_1, P_2, \ldots, P_n)$, kde $n \in \mathbb{N} \wedge n \geq 3 \wedge \exists L_n = (P_n, P_1) \wedge \forall i \in \mathbb{N}, i < n : \exists L_i = (P_i, P_{i+1})$

4.2.2 Výčtové typy

V rámci každé nově získané trajektorie musíme uchovávat typ objektu, jehož dráhu v prostoru získaná trajektorie reprezentuje. Následující množina reprezentuje typy objektů, s nimiž při analýze provozu budeme pracovat:

$Types = \{$“car”,“van”,“$truck$”,“bus”,“$motorbike$”,“$bicycle$”,“$pedestrian$”,“$unknown$”$\}$

U všech získaných trajektorií musíme vždy ukládat barvu sledovaného objektu. Množina podporovaných barev je následující:

$Colors = \{$“$black$”,“$white$”,“red”,“$green$”,“$blue$”,“$yellow$”,“$violet$”,“$orange$”,“$silver$”, “$brown$”,“$gray$”,“$unknown$”$\}$

4.2.3 Filtrační masky

Pomocí následující uspořádané pětice budeme definovat časový úsek trajektorie, který tvoří časová značka úseku, doba trvání daného úseku a dále pak hodnoty dynamických atributů:

$S = (Tim, Dur, Vel, Acc, Sta)$ kde:

- Tim: časová značka daného časového úseku uložená v podobě hodnoty v milisekundách, kde $Tim \in \mathbb{N}$,
- Dur: doba trvání daného časového úseku v milisekundách, kde $Dur \in \mathbb{N}$,
- Vel: průměrná rychlost objektu v rámci časového úseku, kde $Vel \in \mathbb{R}$,
- Acc: průměrné zrychlení objektu v rámci časového úseku, kde $Acc \in \mathbb{R}$,
- Sta: doba stání objektu v rámci časového úseku, kde $Sta \in \mathbb{N}$

Je potřeba provést definici časové masky, abychom mohli filtrovat trajektorie na základě dynamických atributů. Musíme být schopni pracovat s rychlostí a zrychlením sledovaného objektu. Dále nás bude zajímat doba výskytu objektu v rámci scény. V některých případech může být užitečná také doba stání objektu. Časová maska trajektorie je definována jako uspořádaná n-tice navazujících časových úseků:

$TimeMask$: časová filtrační maska v podobě sekvence časových úseků, kde $TimeMask = (S_1, S_2, \ldots, S_n)$ pro $n \in \mathbb{N}$

Prostorová filtrace trajektorie vyžaduje definici prostorové masky. Na základě této masky jsme schopni popsat trajektorii nebo prostorový element v prostoru a následně vymaskovat jen určitou část trajektorie. Níže lze vidět samotnou definici prostorové masky, která je reprezentována binární maticí:

$AreaMask$: matice prostorové filtrační masky, kde $AreaMask = (a_{i,j}), i, j = 1, \ldots, n \wedge \forall i, j \in \mathbb{N} : a_{i,j} \in \{0, 1\}$

4.2.4 Trajektorie

Trajektorii budeme reprezentovat jako uspořádanou n-tici. Jsou zahrnuty všechny potřebné atributy trajektorie, se kterými budeme pracovat. Vždy proto budeme ukládat typ a barvu sledovaného objektu, na základě kterých je možné provádět statickou filtraci. Za účelem dynamické filtrace je nutné uložit časovou masku trajektorie. Vždy je potřeba uložit prostorovou masku, abychom mohli aplikovat prostorovou filtraci trajektorie. Důležitou informací o trajektorii je dráha, která reprezentuje pohyb sledovaného objektu. Dráhu trajektorie budeme definovat jako uspořádanou n-tici řídících bodů trajektorie. Tyto řídící body jsou propojeny sekvencí navazujících úseček, pomocí kterých je určen směr pohybu objektu v prostoru. Níže můžeme vidět výslednou reprezentaci trajektorie:

$T = (Path, Type, Color, TimeMask, AreaMask)$ kde:

- $Path = (P_1, P_2, \ldots, P_n)$, kde $n \in \mathbb{N} \wedge n \geq 2 \wedge \forall i \in \mathbb{N}, i < n : \exists L_i = (P_i, P_{i+1})$ je uspořádaná n-tice řídících bodů reprezentujících trajektorii v prostoru,
- $Type$: typ objektu, kde $Type \in Types$,
- $Color$: barva objektu, kde $Color \in Colors$,
- $TimeMask$: časová maska trajektorie,
- $AreaMask$: prostorová maska trajektorie

Výsledkem analýzy provozu je zdrojová množina vstupních trajektorií. Naším cílem je nad touto množinou provést filtrování a získat pouze její podmnožinu, která je pro nás podstatná. Zdrojová množina trajektorií se samozřejmě liší v závislosti na zvolené konfiguraci. Každá konfigurace zahrnuje pouze aktuální množinu zdrojových trajektorií. Dále je v konfiguraci zahrnutý aktuální úsek časové masky trajektorií. V rámci konfigurace tedy známe časovou značku a také dobu trvání dané konfigurace. Díky těmto hodnotám a časové masce trajektorie jsme schopni získat aktuální hodnoty dynamických atributů trajektorie. Následujícím způsobem budeme definovat konfiguraci jako uspořádanou n-tici:

$C = (Input, Tim, Dur)$ kde:

- $Input$: zdrojová množina vstupních trajektorií konfigurace, kde $Input = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N}\}$,
- Tim: časová značka konfigurace v podobě hodnoty v milisekundách, kde $Tim \in \mathbb{N}$,
- Dur: doba trvání dané konfigurace v milisekundách, kde $Dur \in \mathbb{N}$

4.2.5 Statické operátory

V rámci analýzy provozu bude nutná implementace operátorů určených k filtraci získaných trajektorií na základě statických atributů. Uživateli bude umožněno zvolení množiny podporovaných hodnot daného statického atributu. Zvolená množina hodnot bude podmnožinou odpovídajícího výčtového typu operátoru. Na základě náležitosti atributu do zvolené množiny dojde k filtraci vstupní množiny trajektorií a získáme tak množinu výstupních trajektorií. Obecná definice statického operátoru je následující:

$Operator = (In, Out, Enum)$ kde:

- In: množina vstupních trajektorií operátoru,
- Out: množina všech výstupních trajektorií operátoru,
- $Enum$: množina akceptovaných hodnot statického atributu

Budeme implementovat rovnou několik statických operátorů. První z těchto operátorů slouží k ověření toho, zda trajektorie náleží objektům, které mají pouze určitou barvu. Akceptované barvy budeme reprezentovat množinou zvoleného počtu barev. Níže lze vidět samotnou definici operátoru barvy:

$ColorOperator = (In, Out, Enum)$ kde:

- In: množina vstupních trajektorií operátoru, kde $In = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N}\}$,
- Out: množina všech výstupních trajektorií operátoru, kde $Out = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq n : T_i \in In \wedge \pi 3(T_i) \in Enum\}$,
- $Enum$: množina akceptovaných barev objektů, kde $Enum \subseteq Colors$

Dalším podstatným statickým operátorem je operátor typu objektu. Je samozřejmě možné akceptovat více různých typů objektů. Definice tohoto operátoru je následující:

$TypeOperator = (In, Out, Enum)$ kde:

- In: množina vstupních trajektorií operátoru, kde $In = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N}\}$,
- Out: množina všech výstupních trajektorií operátoru, kde $Out = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq n : T_i \in In \wedge \pi 2(T_i) \in Enum\}$,
- $Enum$: množina akceptovaných typů objektů, kde $Enum \subseteq Types$

4.2.6 Dynamické operátory

Filtrace trajektorií na základě dynamických atributů vyžaduje definici dynamických operátorů. V případě těchto operátorů provádíme filtraci vstupní množiny trajektorií podle náležitosti hodnoty dynamického atributu do zvoleného intervalu. Současnou hodnotu odpovídajícího atributu je možné získat pomocí časové masky konkrétní trajektorie. Pro získání aktuální hodnoty atributu bude u tohoto typu operátorů volána funkce, která vrací současnou hodnotu na základě trajektorie specifikované v jejím parametru. Obecná definice dynamického operátoru je následující:

$Operator = (In, Out, Atr_{min}, Atr_{max})$ kde:

- In: množina vstupních trajektorií operátoru,
- Out: množina všech výstupních trajektorií operátoru,
- Atr_{min}: minimální akceptovaná hodnota dynamického atributu,
- Atr_{max}: maximální akceptovaná hodnota dynamického atributu

Důležitým dynamickým operátorem je operátor rychlosti, díky kterému jsme schopni akceptovat jenom ty objekty, které se pohybují v rámci definovaného intervalu rychlosti. Kompletní definici operátoru rychlosti můžeme vidět níže:

$VelocityOperator = (In, Out, Vel_{min}, Vel_{max})$ kde:

- In: množina vstupních trajektorií operátoru, kde $In = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N}\}$,
- Out: množina všech výstupních trajektorií operátoru, kde $Out = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq n : T_i \in In \wedge Vel_{min} \leq velocity(T_i) \leq Vel_{max}\}$,
- Vel_{min}: minimální průměrná rychlost objektu, kde $Vel_{min} \in \mathbb{R}$,
- Vel_{max}: maximální průměrná rychlost objektu, kde $Vel_{max} \in \mathbb{R}$

Pro akceptování objektů, které zrychlují pouze v rámci nadefinovaných rozsahů zrychlení, budeme používat dynamický operátor zrychlení. Ten je definován následovně:

$AccelerationOperator = (In, Out, Acc_{min}, Acc_{max})$ kde:

- In: množina vstupních trajektorií operátoru, kde $In = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N}\}$,
- Out: množina všech výstupních trajektorií operátoru, kde $Out = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq n : T_i \in In \wedge Acc_{min} \leq acceleration(T_i) \leq Acc_{max}\}$,
- Acc_{min}: minimální průměrné zrychlení objektu, kde $Acc_{min} \in \mathbb{R}$,
- Acc_{max}: maximální průměrné zrychlení objektu, kde $Acc_{max} \in \mathbb{R}$

Níže můžeme vidět reprezentaci dynamického operátoru doby výskytu objektu:

$OccurrenceOperator = (In, Out, Occ_{min}, Occ_{max})$ kde:

- In: množina vstupních trajektorií operátoru, kde $In = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N}\}$,
- Out: množina všech výstupních trajektorií operátoru, kde $Out = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq n : T_i \in In \wedge Occ_{min} \leq occurrence(T_i) \leq Occ_{max}\}$,
- Occ_{min}: minimální doba výskytu objektu v milisekundách, kde $Occ_{min} \in \mathbb{R}$,
- Occ_{max}: maximální doba výskytu objektu v milisekundách, kde $Occ_{max} \in \mathbb{R}$

Tímto způsobem provedeme definici dynamického operátoru doby stání objektu:

$StationaryOperator = (In, Out, Sta_{min}, Sta_{max})$ kde:

- In: množina vstupních trajektorií operátoru, kde $In = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N}\}$,
- Out: množina všech výstupních trajektorií operátoru, kde $Out = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq n : T_i \in In \wedge Sta_{min} \leq stationary(T_i) \leq Sta_{max}\}$,
- Sta_{min}: minimální doba stání objektu, kde $Sta_{min} \in \mathbb{N}$,
- Sta_{max}: maximální doba stání objektu, kde $Sta_{max} \in \mathbb{N}$

4.2.7 Prostorové operátory

Při analýze provozu pro nás bude velmi užitečná prostorová filtrace trajektorií. Pro získání výstupní množiny trajektorií musíme provést výpočet prostorové filtrace nad vstupní množinou trajektorií. K výpočtu průsečíku trajektorií s prostorovým elementem využijeme možností prostorové masky. Výstupem provedené prostorové filtrace je množina binárních vektorů, pomocí kterých filtrujeme jen takové vstupní trajektorie, které elementem prochází. Obecná definice prostorového operátoru je následující:

$Operator = (Element, In, AreaMask, BitMask, Out)$ kde:

- $Element$: zvolený prostorový element operátoru,

- In: množina vstupních trajektorií operátoru,
- $AreaMask$: prostorová maska zvoleného elementu,
- $BitMask$: množina binárních vektorů po provedení filtrace trajektorií,
- Out: množina všech výstupních trajektorií operátoru

Prvním prostorovým operátorem bude operátor brány. Brána bude reprezentována úsečkou, kterou musí získaná trajektorie protnout, aby došlo k akceptování dané trajektorie. Součástí operátoru musí samozřejmě být prostorová maska elementu, abychom byli schopni provést prostorovou filtraci trajektorií a následně vytvořit potřebné binární vektory. Musíme být schopni určit, zda opravdu došlo k protnutí brány některou z trajektorií, čehož dosáhneme právě pomocí prostorových masek trajektorií a masky daného elementu. V závislosti na vzniklé množině binárních vektorů je možné získat výstupní množinu trajektorií operátoru. Níže se nachází definice tohoto prostorového operátoru:

$GateOperator = (L, In, AreaMask, BitMask, Out)$ kde:

- L: úsečka reprezentující bránu,
- In: množina vstupních trajektorií operátoru, kde $In = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N}\}$,
- $AreaMask$: prostorová maska brány,
- $BitMask = \{B_1, B_2, \ldots, B_n \mid n \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq n : B_i = (b_1, b_2, \ldots, b_k)$, kde $k = |In| \wedge \forall j \in \mathbb{N}, j \leq k : b_j = \{0, 1\}\}$ je výsledná množina binárních vektorů po provedení filtrace trajektorií pomocí prostorové masky,
- Out: množina všech výstupních trajektorií operátoru, kde $Out = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq n : T_i \in In \wedge \exists B \in BitMask : \pi_i(B) = 1\}$

Pokročilejším prostorovým operátorem bude operátor regionu. Tento operátor bude akceptovat pouze takové trajektorie, které prochází uživatelem definovanou polygonální oblastí. Je potřeba zjistit, zda došlo k průchodu některých trajektorií vytvořeným regionem. Takto vypadá definice tohoto prostorového operátoru:

$RegionOperator = (R, In, AreaMask, BitMask, Out)$ kde:

- R: ohraničující region operátoru,
- In: množina vstupních trajektorií operátoru, kde $In = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N}\}$,
- $AreaMask$: prostorová maska regionu,
- $BitMask = \{B_1, B_2, \ldots, B_n \mid n \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq n : B_i = (b_1, b_2, \ldots, b_k)$, kde $k = |In| \wedge \forall j \in \mathbb{N}, j \leq k : b_j = \{0, 1\}\}$ je výsledná množina binárních vektorů po provedení filtrace trajektorií pomocí prostorové masky,
- Out: množina všech výstupních trajektorií operátoru, kde $Out = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq n : T_i \in In \wedge \exists B \in BitMask : \pi_i(B) = 1\}$

4.2.8 Booleovské operátory

Pro vytváření hierarchie operátorů potřebujeme mít možnost propojit výstupy několika operátorů tak, aby šel výsledek použít jako vstup jiného operátoru. Z tohoto důvodu je nutná implementace booleovských operátorů, díky kterým bude možné spojovat výstupy operátorů a dále s nimi pracovat. Obecná definice booleovského operátoru je následující:

$Operator = (In, Out)$ kde:

- In: množina množin vstupních trajektorií operátoru,
- Out: sjednocení všech vstupních množin trajektorií

Prvním booleovským operátorem je operátor sjednocení. Níže lze vidět definici tohoto operátoru:

$UnionOperator = (In, Out)$ kde:

- In: množina množin vstupních trajektorií operátoru, kde $In = \{In_1, In_2, \ldots, In_m \mid m \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq m : In_i = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N}\}\}$,
- Out: sjednocení všech vstupních množin trajektorií, kde $Out = \bigcup_{i=1}^{\infty} In_i$

Analogicky k operátoru sjednocení provedeme definici operátoru průniku:

$IntersectionOperator = (In, Out)$ kde:

- In: množina množin vstupních trajektorií operátoru, kde $In = \{In_1, In_2, \ldots, In_m \mid m \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq m : In_i = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N}\}\}$,
- Out: průnik všech vstupních množin trajektorií, kde $Out = \bigcap_{i=1}^{\infty} In_i$

Následně musíme provést definici operátoru doplňku. Při práci s doplňkem množiny trajektorií je nutné získat celkovou množinu trajektorií v aktuální konfiguraci, kterou získáme pomocí volání odpovídající funkce. Samotná definice tohoto operátoru je následující:

$ComplementOperator = (In, Out)$ kde:

- In: množina vstupních trajektorií operátoru, kde $In = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N}\}$,
- Out: množina všech výstupních trajektorií operátoru, kde $Out = \{T_1, T_2, \ldots, T_n \mid n \in \mathbb{N} \wedge \forall i \in \mathbb{N}, i \leq n : T_i \notin In \wedge T_i \in input()\}$

Kapitola 5

Návrh uživatelského rozhraní

V rámci této kapitoly se budeme zabývat návrhem grafického uživatelského rozhraní určeného k vizualizaci a filtraci dopravních dat a trajektorií. Následující kapitola se bude zabývat implementací aplikace na základě navrženého uživatelského rozhraní. Navržené uživatelské rozhraní bude vycházet z formalismu, který byl popsán v kapitole 4. Jednotlivé komponenty uživatelského rozhraní tedy budou stavěny na odpovídajících strukturách z formalismu.

5.1 Vizualizace scény

Hlavní součástí uživatelského rozhraní je vizualizace scény. Samotná vizualizace scény se dělí na vykreslování dopravního obrazového záznamu a dále na vizualizace získaných trajektorií. Obrazový záznam budeme v případě navrhované aplikace reprezentovat jako pozadí scény. V popředí scény budou umístěny konkrétní trajektorie účastníků provozu. Důvodem k tomuto dělení scény je skutečnost, že na popředí této scény bude moci uživatel aplikovat prostorovou filtraci trajektorií, která samozřejmě může mít vliv na počet výsledných trajektorií. Právě z tohoto důvodu budeme na trajektorie nahlížet jako na součást popředí scény. Je nutné uživateli umožnit manipulaci s vykreslovanou scénou. Uživatelské rozhraní musí poskytovat možnost přibližování a oddalování pohledu na scénu, které bude doplněno možností posunutí pohledu na scénu ve všech směrech. Uživateli je následně potřeba umožnit přiblížení pouze konkrétní oblasti scény s cílem detailního náhledu na tuto oblast, což může být užitečné především v situaci, kdy vykreslujeme detailní záznam ve vysokém rozlišení. V rámci scény budou přítomna tlačítka umožňující zmíněnou úpravu přiblížení pohledu.

Výstupem vizualizace scény tedy bude množina trajektorií vykreslovaná na zvoleném obrazovém záznamu. Jednotlivé trajektorie budeme vykreslovat v podobě křivek popsaných množinou řídících bodů, jak již bylo zmíněno v kapitole 4. Jednotlivé křivky trajektorií je nutné vhodným přístupem rasterizovat před vykreslením na obrazovku. Pomocí procesu rasterizace jsme schopni převést geometrickou reprezentaci trajektorie na množinu pixelů obrazu. Důležitá je správná volba barvy vykreslené trajektorie. Naším cílem je zvolit takovou barvu, která bude kontrastní vůči samotnému obrazu, na který budeme trajektorii vykreslovat. Alternativní řešení obvykle využívají trajektorie zelené, fialové a v některých případech také modré nebo žluté barvy. Vždy je vhodné volit barvu trajektorie v závislosti na typu rozpoznaného objektu. Můžeme například volit zelenou barvu trajektorií vozidel a fialovou barvu trajektorií chodců. Trajektorie různých kategorií vozidel je navíc možné vykreslovat v různých odstínech zelené barvy. Díky použití více druhů barev jsme schopni zlepšit čitelnost samotného výstupu vizualizace scény. Uživatelské rozhraní scény bude do-

plněno o lištu s přehledem počtu filtrovaných trajektorií. Uvnitř této lišty pak bude pro každou podporovanou kategorii objektů zobrazen aktuální počet trajektorií.

Je potřeba zmínit skutečnost, že celá vstupní množina trajektorií bude rovnou načtena ze zdrojového souboru při spuštění aplikace. Scéna tedy rovnou bude vykreslována v čase konce analýzy se všemi trajektoriemi a se snímkem scény z konce nahrávky. Aplikace tedy nebude podporovat vykreslování videa, ale pouze statického ukázkového snímku.

5.2 Prostorové elementy

Hlavní funkcionalitou navržené aplikace musí být možnost prostorové filtrace. Tato filtrace vyžaduje vytvoření odpovídajících prostorových elementů v rámci scény. V naší aplikaci budeme podporovat dva různé typy prostorových elementů. Jedná se o bránu a polygonální region. Brána je úsečka, kterou mají filtrované trajektorie protínat. U každé brány si bude uživatel moci nastavit směr filtrace. Druhým podporovaným prostorovým elementem bude region. Jedná se o polygonální útvar, který musí filtrované trajektorie protnout.

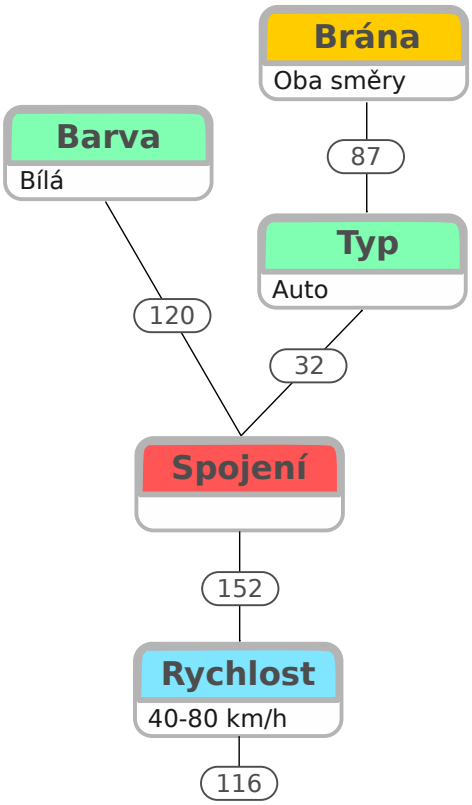
Uživateli je potřeba umožnit vytvoření libovolného tvaru podporovaných elementů na zvolené pozici v rámci scény. Naším cílem je navrhnout takový přístup k vytváření elementů, který bude intuitivní a především rychlý. V případě tvorby regionu je samozřejmě možné nechat uživatele vybrat stupeň polygonu a následně rozměru nového útvaru. Nevýhodou tohoto přístupu může být jeho komplikovanost a případně nízká rychlost samotného procesu tvorby. V naší aplikaci použijeme odlišný přístup k vytváření elementů, při kterém si uživatel nakreslí otevřenou nebo uzavřenou čáru na obrazovce. Pro vytvoření brány stačí nakreslit rovnou čáru nebo otevřenou křivku. Následně dojde k vytvoření úsečky, která vzniklou křivku protíná. Pomocí vhodného algoritmu následně dojde k vytvoření dvou koncových bodů úsečky. Obdobným způsobem bude možné vytvořit element regionu s tím rozdílem, že uživatel na obrazovku nakreslí uzavřenou křivku. Opět bude potřeba implementovat algoritmus vytvářející polygon, který svým tvarem zhruba odpovídá nakreslené křivce.

Může případně nastat situace, kdy uživatel vytvoří takový tvar prostorového elementu, který v některých aspektech neodpovídá jeho původní představě. Každý vytvořený element vždy bude možné ručně editovat. Každý řídící bod útvaru bude možné libovolně přesunovat pomocí stisku tlačítka myši. Dále bude možné dvojitým stiskem tlačítka myši odstranit libovolný řídící bod. Dvojitým stiskem tlačítka na hranu regionu pak dojde k vytvoření nového řídícího bodu. V případě elementu brány je možné pouze přesunovat oba řídící body bez možnosti body přidávat nebo mazat. Následně bude uživatelům umožněno přesunovat celé prostorové elementy stiskem tlačítka myši a zvolením výsledné pozice.

5.3 Filtrační strom

V důsledku návrhu na základě vytvořeného formalismu nestačí pouze přidat prostorový element k tomu, aby došlo k zahájení prostorové filtrace pomocí daného elementu. Pro zahájení filtrace bude uživatel muset vytvořit k danému elementu nový prostorový operátor. Pro vytvoření operátoru je potřeba přesunout daný element stiskem tlačítka myši na plátno, které se nachází vedle scény. Tímto způsobem následně dojde k vytvoření nového operátoru na plátně, což je prostor určený k vytváření nových operátorů. Vytvořený operátor budeme zobrazovat pomocí obdélníkového útvaru, který bude obsahovat typ, název a další atributy odpovídající danému typu operátoru. V závislosti na typu operátoru se bude lišit barva tohoto operátoru s cílem dosažení co nejvyšší přehlednosti.

Přidání dalších typů operátorů bode možné přetažením zvoleného operátoru z panelu na plátno. Panel operátorů je vizuální prvek nacházející se na okraji plátna, ke kterému je připnutý. Součástí panelu operátorů jsou všechny podporované operátory rozdělené do několika kategorií podle jejich typu. Mezi typy podporovaných operátorů v panelu patří statické, dynamické a dále booleovské operátory. Z panelu lze tedy přesunout zvolený operátor na plátno za současného stisku tlačítka myši. Následně dojde k vytvoření zvoleného operátoru na plátně. Samotné plátno je prázdná plocha, která stejně jako plocha scény podporuje interaktivní ovládání pomocí myši. Je nutné poskytnout libovolné přibližování a oddalování plátna a případnou úpravu náhledu na plátno. Cílem této funkcionality je umožnit uživateli vytvoření libovolného náhledu na vytvořené operátory.



Obrázek 5.1: Ukázka filtračního stromu umožňujícího pokročilou filtraci.

Na základě definovaného formalismu bude implementována podpora pro vytváření pokročilejších operátorů. Vytváření takových operátorů bude realizováno strukturou, kterou budeme označovat filtračním stromem. Tento strom je množina různých typů operátorů, které vzájemně tvoří stromovou strukturu. Uzly této stromové struktury jsou jednotlivé operátory. Uzly stromu budeme propojovat pomocí propojovacích čar, které si uživatel bude moci vytvářet. Spojením dvou operátorů vzniká závislost výstupu prvního operátoru na vstupu operátoru druhého. Filtrační strom může být tvořen pouze jediným kořenovým operátorem, který je implicitní. Postupným připojováním dalších operátorů vzniká pokročilejší filtrační strom. V případě propojení dvou operátorů dochází k tomu, že všechny trajektorie filtrované prvním operátorem musí být dále filtrované operátorem druhým, aby byly na výstupu takového filtračního stromu. Z uživatelského hlediska bude mít každý operátor vstupní a výstupní spojení, na které bude možné připojovat další operátory. K propojení

dvou operátorů stačí pouze navázat vstupní spojení jednoho operátoru na výstupní spojení operátoru druhého s tím, že je potřeba v průběhu této akce držet tlačítko myši. Samozřejmostí je možnost navázat výstup jednoho operátoru na více vstupů různých operátorů. Podmínkou propojování operátorů je to, že nelze vytvářet cyklické spoje mezi stejným operátorem.

U každého operátoru v rámci plátna bude navíc zobrazen počet jeho výstupních trajektorií. V případě filtračního stromu bude zobrazen počet výstupních trajektorií u každého uzlu tohoto stromu. Z hlediska procesu filtrace jsou pro uživatele podstatné výstupní počty trajektorií u listových uzlů filtračního stromu. Počty trajektorií u nelistových a kořenových uzlů stromu jsou pouze informační. Na obrázku 4.4 je znázorněn filtrační strom umožňující pokročilou filtraci trajektorií takových vozidel, které mají bílou barvu nebo protínají vytvořenou bránu s tím, že jejich průměrná rychlost je v rámci zvoleného intervalu rychlosti.

5.4 Nastavení operátorů

Uživatel musí mít možnost nastavení atributů vytvořených operátorů. K nastavení těchto atributů bude určen dialog nastavení operátoru, který bude zobrazen po dvojitém klepnutí na zvolený operátor z plátna. Pro všechny typy operátorů bude společné nastavení názvu operátoru. Dále pak v dialogu bude možná konfigurace atributů podporovaných daným operátorem. Pro statické operátory se jedná o nastavení zvolených atributů z daného výčtového typu. V případě dynamických operátorů dojde k zobrazení posuvníků k nastavení rozsahu intervalu hodnot daného atributu. V dialogu je potřeba zobrazit tlačítka pro potvrzení a případně zrušení provedených změn. Také je nutná přítomnost tlačítka určeného k odstranění tohoto operátoru, čímž dojde k odstranění všech jeho vstupních a výstupních propojení. Dialogové okno bude implementováno jako modální. Je tedy nutné dokončit úpravy, než bude možné opět ovládat hlavní okno aplikace.

5.5 Okno aplikace

Postupně jsme popsali hlavní komponenty uživatelského rozhraní naší aplikace. Všechny komponenty je následně nutné zakomponovat do výsledného okna aplikace. Hlavní komponentou je tedy samotná vizualizace naší scény s trajektoriemi. Tuto komponentu umístíme do levé části okna aplikace. Poté je potřeba správně umístit plátno s operátory, které se bude nacházet v pravé části okna aplikace hned vedle scény. Důvodem tohoto rozmístění komponent je poskytnutí rychlého vytváření prostorových operátorů. Panel operátorů je vhodné implementovat jako samostatnou komponentu, kterou lze v rámci okna aplikace přesunovat. Díky tomu si může uživatel panel s operátory libovolně přesunout podle jeho vlastních preferencí. Výchozí umístění panelu bude v levém horním rohu plátna s operátory tak, aby uživatel mohl rychle vytvářet nové operátory. Poměr velikosti scény a plátna musí být uživatelsky nastavitelný pro případy, kdy chceme maximalizovat velikost jedné z těchto komponent. Velikost okna aplikace bude přizpůsobitelná na základě rozlišení obrazovky a ručního nastavení uživatelem.

Kapitola 6

Implementace

Navržená aplikace by tedy měla být schopna provádět filtraci trajektorií. Již při návrhu samotné filtrace bylo naším cílem vytvořit efektivní a rychlý přístup k filtraci trajektorií. Z tohoto důvodu je také vhodné aplikaci implementovat v jazyce, který se vyznačuje vysokou rychlostí a efektivitou. Jazyk C++ tyto vlastnosti splňuje, a proto v něm také byla aplikace implementována. Následně bylo nutné najít vhodný framework, ve kterém bude možné vytvořit uživatelské rozhraní naší aplikace. Zvolili jsme právě framework Qt, který umožňuje pohodlné vytváření uživatelských rozhraní s tím, že v něm lze navíc vyvíjet úplně zdarma. Tento framework je také implementován v jazyce C++ a dále nabízí pokročilé vývojové prostředí Qt Creator [5]. Hlavní výhodou celého frameworku je pak podpora všech významných operačních systémů. Pro nás jsou pak důležité operační systémy Microsoft Windows a Linux, na kterých musí naše aplikace bezproblémově fungovat.

V jazyce C++ je implementováno filtrování trajektorií a obecně chování aplikace. Pro implementaci samotného uživatelského rozhraní bylo využito možností deklarativního jazyka QML[1], který je také součástí Qt frameworku a je implementován v rámci knihovny Qt Quick. Jeho hlavní výhodou v porovnání se starší knihovnou Qt Widgets je rychlejší tvorba prvků uživatelského rozhraní. Jazyk QML má syntaxi velmi podobnou jazyku JSON a podporuje vytváření vlastních funkcí v jazyce JavaScript. Samotný dokument v tomto jazyce je pak popisován hierarchickým objektovým stromem tvořeným jeho jednotlivými komponentami. Návrh okna se všemi ovládacími prvky je velmi pohodlný a především ho lze rychle upravit. Podstatnou výhodou je možnost překladu QML kódu a vlastních JavaScript funkcí do nativní binární C++ aplikace, což nám umožňuje dosáhnout vysoké rychlosti vytvořené aplikace. Vývoj v QML je ve srovnání s Qt Widgets nejenom rychlejší a pohodlnější, ale především jsou poskytovány nové uživatelské prvky a komponenty. Jazyk má v sobě například integrovanou podporu pro dotykové ovládání, tvorbu animací a dále také aplikaci grafických efektů v podobě stínů a průhlednosti.

V příloze B lze pak najít podrobný návod, jak vytvořenou aplikaci přeložit a spustit. V průběhu celé implementace bylo nutné ze správných zdrojů získávat informace ohledně různých součástí frameworku. Všechny poznatky a informace použité v této kapitole jsou získány z knih [6] a [13].

[1] Qt Modeling Language

6.1 Vizualizace scény

V této sekci se budeme zabývat popisem vizualizace dopravní scény, která je hlavní komponentou celé vytvořené aplikace. Samotná vizualizace zahrnuje jak vykreslení obrazu z kamery, tak také množiny získaných trajektorií. Trajektorie budeme vykreslovat na samotný obraz, čímž vznikne výsledný snímek. Postupně budou vysvětleny hlavní principy, díky kterým jsme schopni scénu zobrazovat.

6.1.1 Získání trajektorií

Naprosto klíčovou částí implementace bylo získání a načtení množiny trajektorií účastníků dopravy. Vytvořená aplikace bude načítat množinu trajektorií z JSON souboru s trajektoriemi. Daný soubor již tedy zahrnuje celou historii trajektorií v rámci naší scény. Všechny trajektorie tedy budou vykresleny rovnou při spuštění aplikace. Za účelem načtení parametrů trajektorií byla provedena implementace třídy `TrajectoryAccessor` určené k přečtení zdrojového souboru s trajektoriemi. Samotný zdrojový JSON soubor je tvořen hlavičkou na prvním řádku. V hlavičce se nachází obecné informace o scéně. Mezi tyto informace patří především snímková frekvence zdrojového videa a dále pak rozlišení scény. Každý další řádek zdrojového souboru pak popisuje jednu trajektorii jako samostatný JSON objekt. Každý objekt má základní atributy trajektorie, mezi které patří barva a kategorie objektu. Dále tento objekt zahrnuje JSON seznam všech prostorových dat dané trajektorie a seznam identifikátorů jednotlivých časových úseků. Souřadnice každého úseku trajektorie jsou v rozlišení obrazu a úseky jsou časově seřazeny již v samotném souboru. Ve třídě `TrajectoryAccessor` jsou tedy implementovány metody schopné načíst všechny souřadnice do objektů třídy `Trajectory`, což je bázová třída popisující trajektorii a její časové i prostorové atributy.

6.1.2 Vykreslení scény

Po procesu získání souřadnic konkrétní trajektorie je potřeba realizovat její vykreslení. Pro rasterizaci trajektorií bylo využito modulu `QPainter` z frameworku Qt, který poskytuje rasterizaci různých typů křivek. Konkrétně se využilo metody `cubicTo` ze třídy `QPainterPath` k vykreslení křivky podle množiny souřadnic načtených ze zdrojového souboru. Každá křivka je navíc vykreslena v barvě odpovídající dané kategorii objektu, kterému trajektorie náleží. Tato třída nám navíc umožňuje vykreslení křivek rovnou na snímek pozadí scény. Následně bylo potřeba vytvořený obraz zobrazit v aplikaci a umožnit uživateli pohodlnou manipulaci s náhledem na scénu. Za tímto účelem bylo využito QML komponenty zvané `VideoOutput`, která nám umožňuje zobrazovat snímek s podporou pro přibližování, oddalování a posunování náhledu scény. Podporu všech zmíněných gest nám zajistila komponenta `Flickable`, která obaluje vykreslený snímek a implementuje zpracování událostí myši nebo dotykové obrazovky. Je velmi důležité zmínit, že vytvořená aplikace rovnou při spuštění zobrazuje všechny zdrojové trajektorie na statickém snímku pozadí scény. Nedochází tedy k přehrávání dopravního videa jako v případě existujících konkurenčních aplikací. Důvodem je především velmi obtížná implementace vykreslování videa, která by navíc vyžadovala synchronizaci obrazu s filtrovací logikou, aby nedocházelo k nepřesným výsledkům filtrace. Pro účely demonstrace filtrování trajektorií je ale vykreslování statického snímku dostatečné a stále jsme schopni aplikovat všechny podporované typy filtrů a získat správné výsledky.

6.2 Filtrace

Implementace filtrace trajektorií je realizována hned několika třídami pro každý typ operátoru s tím, že jejich bázovou třídou je vždy třída `Filter`. Tato třída zahrnuje všechny atributy společné pro zbylé typy operátorů, mezi které patří především pomocné parametry pro práci s filtračním stromem. Dále jsou zde také potřebné signály, které se odesílají po provedení vyhodnocení daného operátoru. Postupně budou vysvětleny hlavní myšlenky, díky kterým jsme schopni aplikovat různé typy filtrace.

6.2.1 Atributy trajektorie

Atributy každé trajektorie se načítají ze zdrojového souboru. Jedná se o barvu a kategorii detekovaného objektu. Ve třídě `ColorFilter` je implementováno chování operátoru barvy. Nacházejí se zde potřebné atributy pro filtraci na základě barvy. Výčet uživatelem zvolených barev se ukládá pomocí binární hodnoty, kde jednotlivé bity postupně reprezentují zvolené barvy z podporovaného výčtu. Celkem je podporováno dvanáct barevných odstínů a uživatel si pouze vybere jejich libovolnou podmnožinu. Analogicky je definována třída `CategoryFilter` s tím rozdílem, že v ní pracujeme s osmi podporovanými kategoriemi objektů. Bázovou třídou obou zmíněných tříd je pak třída `AttributeFilter`.

6.2.2 Časová filtrace

Implementace časových filtrů byla nejkomplikovanější částí celé práce a bylo potřeba vyřešit hned několik problémů s filtrací na základě dynamických atributů. Před samotnou implementací tohoto typu filtrace bylo totiž potřeba dynamické atributy nějak získat, protože se ve zdrojovém souboru s trajektoriemi nenachází. Za tímto účelem byla implementována sada metod, pomocí kterých lze vypočítat všechny požadované dynamické atributy ze zdrojového souboru. Konkrétně se jedná o atributy, mezi které které patří rychlost, zrychlení, délka stání a celková doba výskytu objektu. Tyto atributy nejsou ukládány do zdrojového soubory především z důvodu jejich náročnosti na diskový prostor. V případě zdrojových souborů s velkým množstvím trajektorií by ukládání všech těchto atributů způsobilo výrazné zvětšení velikosti souboru.

K výpočtu atributů nám totiž dostačuje znalost snímkové frekvence scény a souřadnice každého stavu trajektorie. Díky znalosti snímkové frekvence nejdříve vypočítáme dobu trvání každého stavu trajektorie, kdy je nutné nejprve vypočítat celkovou dobu trvání trajektorie. Následně dochází k výpočtu odhadu časových atributů objektu pomocí nalezení hranic stavu a pak celé trajektorie. Je potřeba iterativně hledat začátek a konec trvání trajektorie. V průběhu hledání obou hranic trajektorie provádíme výpočet součtu hodnot rychlosti, normálového zrychlení a tečného zrychlení v rámci daného úseku trajektorie [21]. Získání potřebných dynamických atributů vyžaduje výpočet rozdílu mezi předchozí a následující pozicí úseku trajektorie, čímž získáváme vektor rychlosti a zrychlení. Pomocí funkce `cv::norm` z knihovny OpenCV vypočítáme hodnotu rychlosti z vektoru rychlosti [11]. Obdobně vypočítáme tečné zrychlení pomocí skalárního součinu s vektorem zrychlení a normálové zrychlení pomocí vektorového součinu s vektorem zrychlení. Následně vypočítáme úhel rychlosti pomocí rozdílu hodnot v horizontální a vertikální souřadné ose [21]. Na závěr zohledníme hodnotu snímkové frekvence a získáme tak požadované dynamické atributy pro daný stav trajektorie. Je potřeba zmínit, že získané dynamické atributy rychlosti a zrychlení jsou pouze v jednotkách pixelů. Důvodem je skutečnost, že pro získání skutečných hodnot

bychom museli provést výpočet georeferencování dané scény, což je poměrně komplikovaný proces přesahující rámec této práce.

Po implementaci metod určených k získání dynamických atributů trajektorie byla zahájena implementace tříd všech dynamických operátorů. Bázovou třídou tohoto typu operátorů je třída `SpatiotemporalFilter` zahrnující proměnné pro porovnání časových atributů v rámci uživatelem zvoleného intervalu. Je tedy implementován výpočet náležitosti hodnoty časového atributu do definovaného rozsahu hodnot. Implementace operátoru rychlosti se pak nachází ve třídě `SpeedFilter`, kde je navíc implementována podpora pro filtraci podle aktuální, průměrné, minimální a maximální hodnoty rychlosti daného objektu. Analogicky jsou pak definovány třídy `AccelerationFilter`, `DurationFilter` a `StationaryFilter`. Tyto třídy postupně implementují operátory zrychlení, doby výskytu a délky stání objektu.

6.2.3 Prostorová filtrace

Filtrace pomocí prostorových elementů vyžaduje implementaci algoritmu, který naplní prostorovou masku trajektorií a přítomných prostorových elementů. Reprezentace trajektorie v prostorové masce vyžaduje zajištění návaznosti jednotlivých úseků v buňkách matice. Při načítání souřadnic trajektorie ze zdrojového souboru musíme respektovat identifikátory každého úseku a ukládat identifikátor předcházejícího stavu do každého nově vytvořeného stavu. Při plnění prostorové masky jsme totiž schopni ručně zajistit spojitost trajektorie doplněním prázdných buněk tak, aby již trajektorie byla spojitá. Třída `TrajectorySet` pak reprezentuje samotnou prostorovou masku, pomocí které je realizována prostorová filtrace. Ve třídě `WorkSpace` se nachází implementace samotných metod, které prostorovou filtraci umožňují. Výpočet rastrové operace průniku pro výpočet průsečíku prostorových masek je realizován pomocí třídy `std::bitset`.

Bázovou třídou prostorových operátorů je třída `SpatialFilter`, která zahrnuje všechny společné atributy pro podporované prostorové elementy. Mezi tyto atributy patří řídící body elementu a dále samozřejmě jeho prostorová maska. Implementace prostorového operátoru brány se nachází ve třídě `LineFilter`, která zahrnuje atributy pro směr a úhel brány. Druhým prostorovým operátorem je operátor regionu, jehož implementaci lze nalézt ve třídě `PolygonalFilter`, která navíc obsahuje atributy určené k rasterizaci ohraničení polygonu a také hraniční body v prostorové masce.

6.2.4 Vyhodnocení stromu

Vyhodnocení filtračního stromu vyžaduje průchod všech operátorů v uzlech stromu a získání jejich výstupních počtů trajektorií. Za tímto účelem byla provedena implementace třídy `ExpressionTree`, kde uzly této struktury implementuje třída `ExpressionNode`. Vždy tedy pracujeme se seznamem uzlových objektů. Samotné větvení stromu je realizováno pomocí ukazatelů na předcházející a následující uzly konkrétního uzlu. Komplikovanější je práce s kořenovým uzlem, protože tento uzel je implicitní a reprezentuje celou vstupní množinu trajektorií. Kořenový uzel pro uživatele tedy není viditelný, ale v rámci struktury s ním musíme počítat. Tato skutečnost je pak ošetřena při vykreslování celého stromu.

V rámci každého uzlu máme k dispozici množinu filtrovaných trajektorií, která se vypočítá při vyhodnocení tohoto uzlu. Vyhodnocení vždy vyžaduje nalezení všech předchozích uzlů a zavolání vyhodnocovací metody také pro každého potomka uzlu. Jedná se tedy o rekurzivní algoritmus pro vyhodnocení uzlů. Všechny uzly připravené k vyhodnocení ukládáme do fronty s tím, že dochází k vyhodnocení prvního uzlu ve frontě. Vyhodnocení je ukončeno v případě, kdy je fronta uzlů prázdná. Z důvodu prevence zacyklení výpočtu

nebo velmi dlouhých výpočtů je velikost každé fronty omezena na 1000 uzlů. Celý filtrační strom je pak zapouzdřen do třídy `TrafficExpressionTree`, která se stará o detekci cyklů ve stromě a zapisování výsledků vyhodnocení.

6.3 Prostorové elementy

Vizualizace dopravní scény dále vyžaduje vykreslení množiny prostorových elementů určených k prostorové filtraci trajektorií. Jednotlivé elementy se nacházejí v popředí této scény a uživatel s nimi může libovolně manipulovat, a proto potřebujeme efektivně reprezentovat tyto elementy, kterých může být obecně libovolné množství.

6.3.1 Vykreslení elementů

Třída `SpatialElementModel` reprezentuje model všech přítomných prostorových elementů, které je potřeba vykreslovat. Tato třída přímo dědí ze třídy `QAbstractListModel`, která nám poskytuje implementaci listového modelu nad grafickými prvky. Přidání, odebrání a úprava každého elementu je zpracována touto třídou. Nás zajímají specifické role elementů, pomocí kterých jsme schopni získat atributy každého elementu přímo v QML. Celý model je tedy přístupný z QML a jsme schopni získat přesné souřadnice jednotlivých řídících bodů, abychom mohli element vykreslit přímo v QML. Za účelem manipulace s elementy byla vytvořena QML komponenta `SpatialFilterView` poskytující přístup k našemu modelu elementů. Tato komponenta dále zahrnuje prvek `Repeater` obsahující všechny elementy z modelu. QML definice konkrétního elementu je v komponentě `SpatialFilterShape`, který je delegátem prvku `Repeater`, jež se nachází uvnitř samotného prvku s vykreslenou scénou. Manipulaci s elementem a jeho řídícími body implementuje třída `SceneManager`, ve které se nacházejí odpovídající sloty pro události myši při úpravách elementu.

6.3.2 Vytváření elementů

Vytváření nových prostorových elementů je také zajištěno prvkem `SpatialFilterView`. Při aktivaci režimu tvorby nového elementu dojde k vykreslování cesty pohybu kurzoru myši při současném stisku levého tlačítka myši. Průběžné vykreslování cesty je zajištěno QML prvkem `Canvas` a postupně dochází k ukládání každé nové souřadnice pohybu. Dokončení tvorby nového elementu se provádí uvolněním stisknutého tlačítka s tím, že následně dojde v metodě třídy `SceneManager` k vytvoření nového prostorového elementu zvoleného typu a přidání tohoto elementu do modelu `SpatialElementModel`, který současně zajistí jeho vykreslení. Každý vytvořený prostorový element lze stiskem tlačítka označit a pracovat s jeho jednotlivými řídícími body. Body lze přidávat, mazat a samozřejmě také přesunovat. Manipulace s body elementů je také implementována v rámci třídy `SceneManager`.

6.4 Filtrační strom

Po implementaci chování a vyhodnocení filtračního stromu bylo potřeba realizovat vykreslení samotné stromové struktury. Z hlediska vizualizace lze filtrační strom rozdělit na jednotlivé operátory a spojení mezi dvojicí operátorů. Každé spojení pak také zahrnuje obdélníkovou vlajku s výstupním počtem trajektorií. Tyto vlajky je dále potřeba vykreslovat u každého listového uzlu stromu, které už spojení nemají a reprezentují výstup celé filtrace.

6.4.1 Reprezentace operátoru

Vizuální reprezentaci každého operátoru implementuje třída `GuiFilterWrapper`. Kromě samotného typu jsou v atributech této třídy ukládány další vizuální parametry operátoru, mezi které řadíme pozici, velikost, barvu, zvýraznění, datový model a pak také seznam všech jeho vstupních a výstupních spojení. Datový model operátoru je seznam s detailnější specifikací parametrů, které se liší v závislosti na jeho typu. V datovém modelu zobrazujeme například zvolený rozsah časových atributů, vybrané kategorie a případně také zvolenou množinu barev. Podobně jako v případě prostorových elementů byl vytvořen model s operátory označovaný jako `OperatorModel`, který realizuje přidávání, mazání a upravování existujících operátorů. Vizualizaci každého prvku v tomto modelu pak zajišťuje komponenta `OperatorDelegate`, ve které pomocí rolí z modelu nastavujeme pozici, velikost, barvu a také datový model každého operátoru.

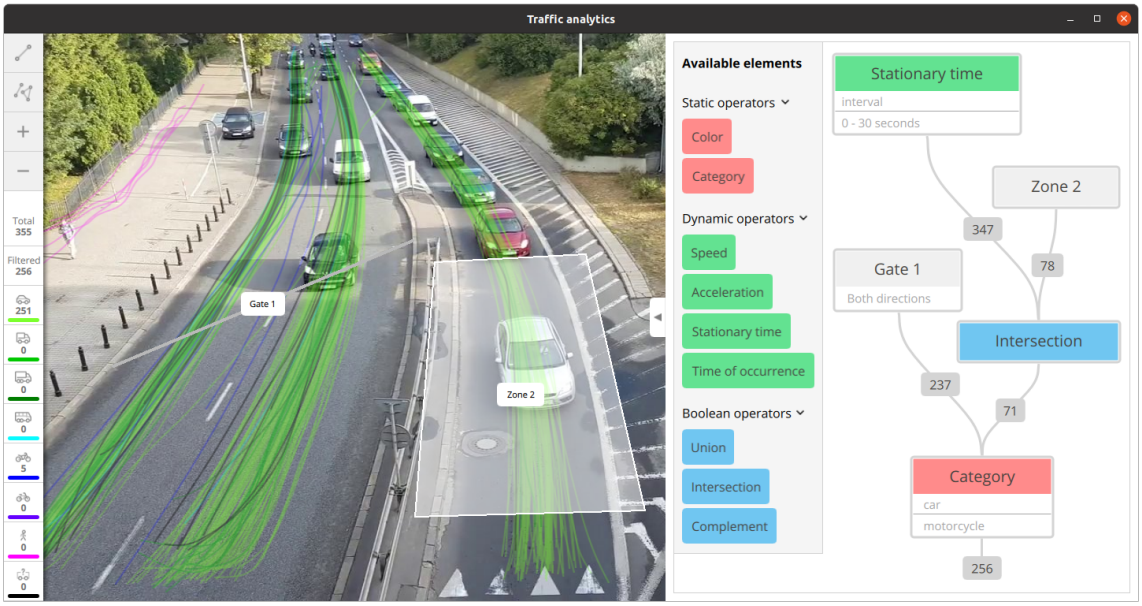
6.4.2 Vytváření operátorů

Nový operátor lze vytvořit jeho přetažením z panelu operátorů na plátno. Samotný panel operátorů reprezentuje QML komponenta `ElementsTab`, ve které jsou dostupné typy operátorů seřazeny v prvku `ListView`. Každý typ operátoru je pak delegátem z modelu implementovaného v rámci třídy `FilterGroupModel`. Každý delegát je vizualizován prvkem `ProgrammingElement`, což je obdélníkový útvar, který lze myší přesunout na plátno, čímž vznikne nový operátor daného typu. Proces přesunu operátoru včetně jeho vizualizace je realizován prvkem `DraggableProgrammingElement`.

V případě vytváření prostorových elementů je situace poněkud odlišná. Existující prostorový element je totiž potřeba přesunout na plátno ze samotné scény s elementy. Zbytek procesu vytváření je již stejný jako v případě zbylých typů operátorů. Proces přesunu elementu včetně jeho vizualizace je implementován v rámci prvku `DraggableSpatialFilterShape`. Tyto prvky jsou v podobě delegátů také umístěny uvnitř prvku `SpatialFilterView`. Prostorový element je tedy nutné nejdříve přesunout na plátno, aby byl vázaný na operátor a byl použitelný k filtraci. Ke každému prostorovému elementu lze vytvořit více instancí operátorů, které je navíc možné přejmenovat pro větší přehlednost.

6.4.3 Vykreslení stromu

Vykreslení všech operátorů na plátno realizuje QML komponenta `AnalyticView`, která obsahuje prvek `Repeater` s delegáty umístěnými v modelu `OperatorModel`. Delegátem je již zmíněný prvek `OperatorDelegate` s tím, že manipulace s operátorem je implementována přímo v QML pomocí prvků `DropArea` a `MouseArea`, které poskytují sloty na zpracování událostí z myši. Mezi dvojicí operátorů pak může být spojení, které také musíme vykreslovat. Třída `OperatorConnectionModel` implementuje samotná spojení operátorů a dědí ze třídy `QAbstractListModel`. U každého spojení jsou v tomto modelu uloženy výstupní počty trajektorií a čtveřice řídících bodů. V této třídě pak provádíme pomocí metody `cubicTo` ze třídy `QPainterPath` výpočet ohraničujících obdélníků křivky, pomocí kterých jsme schopni rozpoznat označení spojení myší. Zobrazení celé množiny spojení je opět realizováno prvkem `Repeater`, jehož delegáty jsou jednotlivé spojení z modelu. Každé spojení je vykreslováno na samostatný prvek `Canvas`, což je plátno určené k rasterizaci vzniklé křivky a dále obdélníkové vlajky s počtem výstupních trajektorií. Vzniklý filtrační strom je zapouzdřený v komponentě `Flickable` poskytující podporu pro přibližování, oddalování a posunování náhledu na strom.



Obrázek 6.1: Ukázka výsledné aplikace s prostorovými elementy a filtračním stromem.

Kapitola 7

Testování

Závěrečnou částí této práce bylo provedení uživatelského testování. Hlavním smyslem samotného testování je získat názory jiných uživatelů na vytvořenou aplikaci. V rámci uživatelského testování je potřeba získat zpětnou vazbu jak k uživatelskému rozhraní aplikace, tak k její funkcionalitě. Vytvořená aplikace tedy musí mít nejenom moderní a přehledné uživatelské rozhraní, ale dále musí být intuitivní a mít komfortní ovládání. Informace získané v rámci uživatelského testování mohou pomoci nalézt potíže a nedostatky aplikace, ale také mohou být podkladem pro navazující práci. Samotné testování pak probíhalo na základě navržené testovací procedury, která se skládala z několika navazujících fází.

Před zahájením testování bylo nutné oslovit několik testerů, kteří byli ochotni si vytvořenou aplikaci vyzkoušet a následně nám dát zpětnou vazbu. Pro uživatelské testování bylo vybráno celkem šest testerů. Důvodem k volbě právě tohoto počtu testerů je především vhodná kombinace dostatečného počtu názorů a přijatelné časové náročnosti testovací procedury. Cílem bylo vybrat takové kandidáty, kteří mají hodně praktických zkušeností s vývojem aplikací. Právě tito lidé totiž mají hodně zkušeností s různými uživatelskými rozhraními. Jejich názor na vytvořenou aplikaci má tedy velkou váhu z hlediska intuitivnosti a vzhledu uživatelského rozhraní. Dále také budou očekávat dostatečnou funkcionalitu a spolehlivost od naší aplikace. Všichni testeři tedy mají zkušenosti s aplikacemi různých kategorií s tím, že tři z nich již mají zkušenosti s konkurenčními aplikacemi určenými k filtrování trajektorií. Právě tato trojice testerů byla schopna provést přímé srovnání naší aplikace s konkurencí, což byl hlavní cíl celého testování.

7.1 Návrh testování

Pro získání lepší zpětné vazby bylo samotné testování rozděleno do dvou navazujících fází, které se v některých ohledech hodně lišily. V první fázi testování měl každý tester možnost vyzkoušet si aplikaci s tím, že mu byl sdělen pouze její účel. Dozvěděl se tedy, že se jedná o aplikaci určenou k filtrování trajektorií, ale další informace mu již nebyly poskytnuty. Cílem této fáze bylo zjistit, jak jsou testeři schopni pracovat bez cizí pomoci. V průběhu celé fáze byly sledovány akce, které tester činí, a jak je schopen aplikaci sám ovládat a používat. Poté následovala fáze druhá, ve které měl tester opět možnost vyzkoušet si práci s aplikací. Rozdíl byl v tom, že druhé fázi předcházelo vysvětlení všech aspektů aplikace testerovi. Daný člověk se tedy před zahájením druhé fáze zeptal na různé detaily a nechal si vysvětlit logiku celého uživatelského prostředí. V průběhu druhé fáze testování pak bylo naším cílem sledovat, jak se liší testerovi schopnosti používání aplikace ve srovnání s fází první. Závěrem

celého testování pak bylo pokládání doplňujících otázek danému testerovi. Účelem těchto otázek bylo získat detailnější odpověď daného člověka na otázky, na které jsme se v průběhu testování snažili odpovědět. Některé otázky se pak také týkaly budoucích rozšíření vytvořené aplikace. V příloze C pak lze nalézt kompletní seznam všech položených otázek.

Naším cílem bylo umožnit testerům vyzkoušet si všechny komponenty aplikace a zeptat se na případné detaily. Nejdůležitější komponentou v rámci testování byl samozřejmě filtrační strom. V případě stromu nás zajímal první dojem testerů na tuto strukturu s tím, že bylo naším cílem zjistit, zda je podle nich použití stromové struktury dobrý nápad či nikoliv. Dále bylo důležité získat od testerů informace o tom, zda je vizuální zpracování filtračního stromu zdařilé. Zajímal nás názor jak na vzhled celkové struktury filtračního stromu, tak také jednotlivých operátorů v uzlech tohoto stromu. Se samotným filtračním stromem pak dále souvisí panel s operátory a plátno, na které je strom vykreslován. V rámci panelu s operátory jsme museli zjistit, zda je vytváření nových prostorových elementů intuitivní, rychlé a především pohodlné. Po dotazech na vytváření operátorů následovaly otázky týkající se upravování vytvořeného stromu a manipulace s jednotlivými operátory a spojeními. Dále jsme se pokusili zjistit, zda je vizualizace výstupů filtrace přehledná, a jestli jsou vykreslované vlajky s výstupy dobře čitelné. Další komponentou pak samozřejmě byla vizualizace scény s trajektoriemi. Zajímala nás zpětná vazba jak na vykreslování trajektorií, tak také na vytváření nových prostorových elementů. V rámci vizualizace trajektorií bylo potřeba zjistit, zda testerům vyhovují zvolené barvy trajektorií. Následně jsme se pokusili zjistit, jaký je názor testerů na vzhled a význam postranního menu s přehledem kategorií podporovaných typů objektů. Na závěr nás ještě zajímal názor na celkový vzhled vytvořené aplikace a jejího prostředí.

7.2 Průběh testování

V rámci první fáze testování byla většina testerů zhruba na stejné úrovni z hlediska jejich schopností ovládat aplikaci. Testeři byli již v této fázi schopni vytvářet prostorové elementy a aplikovat filtraci trajektorií pomocí vlastního filtračního stromu. Vytvoření nového jednoduššího stromu nedělalo testerům potíže. Každý tester dokázal nějakým způsobem filtrovat vstupní množinu trajektorií, ačkoliv zpočátku měl jisté problémy s kombinováním různých typů operátorů. Další komplikace byly způsobeny především díky značnému rozdílu mezi vytvářením prostorových operátorů ve srovnání s operátory v panelu. Dále testery mátla vizualizace kategorií objektů v postranním panelu, kdy si přehled trajektorií daného typu objektu pletli s jejich filtrací. Další výhradou testerů byla poměrně malá šířka plátna pro filtrační strom a panel s operátory, kdy dle jejich názoru mělo být plátno raději trochu širší, aby nemuseli ručně rozšiřovat plochu pomocí přítomného tlačítka.

Při diskuzi s testery došlo k vysvětlení a upřesnění tvorby filtračního stromu a manipulace s operátory. Poté následovala druhá fáze testování, v rámci které si testeři mohli opět vyzkoušet práci s aplikací. Testeři po diskuzi pak bez problémů vytvářeli vlastní filtrační stromy a také správně používali booleovské operátory ke kombinování výstupů. Nikdo neměl potíže s úpravou náhledu na vykreslenou scénu a filtrační strom. Testeři si scénu vždy přiblížili dle svých představ a vytvořili prostorové elementy na správných místech. Obecně již po vytvoření vlastního stromu neměl žádný z nich potíže s úpravou existujících prvků. Někteří testeři zpočátku nedokázali pochopit, proč jsou jednotky rychlosti a zrychlení v pixelech. Po vysvětlení problematiky s georeferencováním scény již testeři dále neměli s jednotkami potíže.

V příloze D lze nalézt detailní vyhodnocení testování včetně odpovědí na všechny položené otázky. Níže pak můžeme vidět seznam nejčastějších připomínek a návrhů:

- Poměrně malá výchozí šířka plátna pro zobrazení filtračního stromu včetně panelu s operátory.
- V rámci každého operátoru by měla být možnost nastavení vlastního barevného provedení.
- Jednotlivé typy prostorových elementů by bylo lepší vykreslovat v jiných barevných odstínech.
- Chybějící možnost vytvářet vlastní kombinované operátory z existujících filtračních stromů.
- Po vytvoření operátoru by měl být automaticky ukončen režim kreslení, aby člověk omylem dále nekreslil.
- Vytváření spojení operátorů je nepřesné a měl by být zvětšen konektor a přidána možnost tvorby spojení rovnou z vlajky.
- Chybí podpora pro rotaci prostorových elementů tak, jak je to typické v jiných grafických nástrojích.
- Zadání rozsahu časových atributů musí jít napsat ručně a ne pouze nastavením pomocí posuvníku.
- Vytvoření prostorových elementů by mělo jít také provést klepnutím na pozice, kde mají být řídící body elementu.
- Přidat podporu pro další prostorový element, který by reprezentoval směr pohybu objektů.
- Implementace nového typu operátoru, který by filtroval na základě limitu vzájemné vzdálenosti objektů scény.

7.3 Výsledky testování

Uživatelské testování dopadlo úspěšně a každý tester si v dostatečné míře vyzkoušel práci s naší aplikací. Testeři měli možnost se k aplikaci vyjádřit a sdělit nám své připomínky a návrhy nové funkcionality. Následně bylo potřeba zpracovat výsledky uživatelského testování. Veškerá zpětná vazba testerů tedy byla vyhodnocena a došlo k vytvoření odpovědí na všechny položené otázky. Velmi dobrou zprávou je, že se podle testerů jedná o kvalitní aplikaci, která by mohla být v praktickém použití hodně užitečná. Na veškeré připomínky testerů by bylo vhodné reagovat a aplikaci rozšířit nebo upravit. Celkový počet připomínek byl poměrně nízký a jednalo se věci, jejichž vyřešení by nezabralo mnoho času. Také je dobré uvažovat nad implementací nové funkcionality, kterou testeři v průběhu testování navrhli.

7.4 Navazující práce

Z hlediska budoucí navazující práce je hned několik rozšíření, které by bylo vhodné implementovat. Některé rozšíření nejsou implementovány z jejich časové náročnosti. Zbylé rozšíření naopak zmínili testeři při uživatelském testování aplikace.

Nejdůležitějším rozšířením je implementace podpory pro zobrazení videa v rámci scény. Tato funkcionalita vyžaduje načtení a přehrání záznamu s tím, že množina trajektorií se může měnit v závislosti na aktuálním čase scény. Trajektorie tedy postupně přibývají v průběhu přehrávání záznamu. Filtrace trajektorií je v takovém případě komplikovanější, protože musí být filtrování synchronizované se snímkem obrazu, který se může měnit v závislosti na snímkové frekvenci. Samotný proces filtrace a vyhodnocení stromu je pak samozřejmě znatelně výpočetně náročnější, protože se provádí velmi často.

Dalším rozšířením je pak podpora pro georeferencování zobrazené scény. Hlavní myšlenkou tohoto procesu je vytvoření souvislosti mezi zobrazovanou scénou v aplikaci a místem, na kterém se tato scéna skutečně odehrává [9]. Rozšíření by tedy vyžadovalo implementaci mapování obrazu v některém geografickém souřadnicovém systému na naši vykreslovanou scénu. Tento obraz je pak definován souřadnicemi v tomto souřadnicovém systému [9]. V rámci tohoto rozšíření by také bylo potřeba přidat podporu pro konfiguraci georeferencování a ruční nastavení mapování obrazu na scénu. Díky tomuto rozšíření bychom následně byli schopni pracovat se skutečnými časovými atributy. Rychlost a zrychlení by pak měly metrické jednotky místo pixelů.

Mezi zbylé rozšíření pak řadíme věci, které navrhli sami testeři při uživatelském testování. Nejdůležitějším z těchto rozšíření je určitě implementace podpory pro vytváření vlastních kombinovaných operátorů z filtračního stromu. Dále by bylo velmi užitečné implementovat podporu pro nastavení vlastního barevného provedení u operátorů. Kompletní seznam všech možných rozšíření lze nalézt v příloze D.

Kapitola 8

Závěr

Cílem této práce byl návrh gramatiky a uživatelského rozhraní pro filtrování a vizualizaci časoprostorových dat. Před samotným návrhem bylo provedeno seznámení se s vyhodnocováním dopravních dat na základě analýzy trajektorií jednotlivých uživatelů silničního provozu. Následoval návrh a popis formalismu umožňujícího filtraci na základě statických a dynamických atributů a dále také filtraci prostorovou. Postupně byly popsány a definovány všechny komponenty navrženého formalismu a také byly vysvětleny hlavní myšlenky, podle kterých je formalismus vytvořen.

Poté bylo provedeno představení nejvýznamnějších existujících aplikací, které umožňují vizualizaci a filtraci dopravních dat. U každé z těchto aplikací byl proveden popis funkcionality a uživatelského rozhraní. Dále byly shrnuty výhody a nevýhody každé aplikace. Na základě vytvořeného formalismu byl proveden návrh aplikace s uživatelským rozhraním, která je určena k vizualizaci a analýze dopravních dat. Shrnutí vlastností existujících aplikací bylo podkladem tohoto návrhu uživatelského rozhraní. Postupně byl proveden popis chování a ovládání každé hlavní komponenty navržené aplikace. Závěrečnou částí byla implementace navržené aplikace umožňující filtraci trajektorií. Aplikace byla implementována pomocí Qt frameworku. Byl zvolen programovací jazyk C++ doplněný o použití jazyka QML. Tato volba nám zajišťuje vysoký výkon výsledné aplikace a dále podporu všech hlavních operačních systémů.

Po dokončení implementace bylo potřeba provést uživatelské testování vytvořené aplikace, v rámci kterého bylo nutné získat zpětnou vazbu. Samotné testování vyžadovalo oslovení několika testerů, kteří byli ochotni si aplikaci vyzkoušet. Naším cílem bylo získat od testerů důležité informace o uživatelském rozhraní a funkcionalitě aplikace. Testování bylo rozděleno do několika fází, v rámci kterých byla provedena diskuze s testery. Uživatelské testování dopadlo úspěšně s tím, že byly získány všechny potřebné informace od testerů.

Literatura

[1] Afrin, T. a Yodo, N. A Survey of Road Traffic Congestion Measures towards a Sustainable and Resilient Transportation System. *Sustainability*. Červen 2020, sv. 12, s. 4660. DOI: 10.3390/su12114660.

[2] Bock, J., Krajewski, R., Moers, T., Runde, S., Vater, L. et al. The inD Dataset: A Drone Dataset of Naturalistic Road User Trajectories at German Intersections. Listopad 2019.

[3] *BriefCam: Transforming video into actionable intelligence*. [online]. [cit. 2021-05-16]. Dostupné z: https://www.briefcam.com.

[4] Chen, Y., Guo, S., Zhang, B. a Du, K.-L. A Pedestrian Detection and Tracking System Based on Video Processing Technology. In:. Prosinec 2013, s. 69–73. DOI: 10.1109/GCIS.2013.17. ISBN 978-1-4799-2886-6.

[5] Chroboczek, M. *Grafická uživatelská rozhraní v Qt a C++*. Computer Press, 2013. ISBN 978-80-251-4124-3.

[6] Eng, L. Z. *Qt5 C++ GUI Programming Cookbook - Second Edition*. Packt Publishing, 2019. ISBN 9781789803822.

[7] Folger, P. Geospatial information and geographic information systems (GIS): Current issues and future challenges. Leden 2011, s. 1–34.

[8] *GoodVision Video Insights - Advanced Traffic Analytics Platform*. [online]. [cit. 2021-05-16]. Dostupné z: https://goodvisionlive.com.

[9] Huisman, O. a By, R. de. *Principles of geographic information systems : an introductory textbook*. Leden 2009. 258 s.

[10] Jacob, B. a Violette, E. Vehicle Trajectory Analysis: An Advanced Tool for Road Safety. *Procedia - Social and Behavioral Sciences*. Prosinec 2012, sv. 48, s. 1805–1814. DOI: 10.1016/j.sbspro.2012.06.1155.

[11] Kaehler, A. a Bradski, G. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. 1st. O'Reilly Media, Inc., 2016. ISBN 1491937998.

[12] Kanagaraj, V., Asaithambi, G., Toledo, T. a Lee, T.-C. Trajectory Data and Flow Characteristics of Mixed Traffic. *Transportation Research Record: Journal of the Transportation Research Board*. Říjen 2015, sv. 2491, s. 1–11. DOI: 10.3141/2491-01.

[13] Lazar, G. *Mastering Qt 5 : create stunning cross-platform applications using C++ with Qt widgets and QML with QT Quick*. Second edition. Packt, 2018. ISBN 1-78899-389-6.

[14] Loidl, M., Wallentin, G., Cyganski, R., Graser, A., Scholz, J. et al. GIS and Transport Modeling—Strengthening the Spatial Perspective. *ISPRS International Journal of Geo-Information*. Červen 2016, sv. 5, s. 84.

[15] *Detekční smyčka*. [online]. [cit. 2021-05-16]. Dostupné z: https://cs.wikipedia.org/wiki/Detek%C4%8Dn%C3%AD_smy%C4%8Dka.

[16] Maignan, L. a Yunès, J.-B. Moore and Von Neumann Neighborhood N-Dimensional Generalized Firing Squad Solutions Using Fields. In:. Prosinec 2013. DOI: 10.1109/CANDAR.2013.98.

[17] Martin, J., Delgado Martin, G. a Garcia Asuero, A. Intersecting Straight Lines: Titrimetric Applications. In:. Září 2017. DOI: 10.5772/intechopen.68827. ISBN 978-953-51-3523-4.

[18] Mccullagh, B. a Ambrósio Archanjo, G. Traffic Analysis: Basic Concepts and Applications. In:. Listopad 2018. DOI: 10.1201/9781351032742. ISBN 9781351032735.

[19] *Miovision: Traffic Systems Management*. [online]. [cit. 2021-05-16]. Dostupné z: https://miovision.com.

[20] Oosterhuis, T. a Schomaker, L. "Who is Driving around Me?"Unique Vehicle Instance Classification using Deep Neural Features. Únor 2020.

[21] Opoku Sarkodie, R. a Acheampong, E. Vector-Valued Function Application to Projectile Motion. Květen 2015, sv. 2.

[22] Quek, C., Pasquier, M. a Lim, B. POP-TRAFFIC: A novel fuzzy neural, approach to road traffic analysis and prediction. *Intelligent Transportation Systems, IEEE Transactions on*. Červenec 2006, sv. 7, s. 133 – 146. DOI: 10.1109/TITS.2006.874712.

[23] Santhosh, K. K., Dogra, D. P., Roy, P. P. a Chaudhuri, B. B. Trajectory-Based Scene Understanding Using Dirichlet Process Mixture Model. *IEEE Transactions on Cybernetics*. 2019, s. 1–14. DOI: 10.1109/TCYB.2019.2931139.

[24] Wang, Z., Lu, M., Yuan, X., Zhang, J. a Wetering, H. Visual Traffic Jam Analysis Based on Trajectory Data. *IEEE transactions on visualization and computer graphics*. Prosinec 2013, sv. 19, s. 2159–68. DOI: 10.1109/TVCG.2013.228.

[25] Warfield, J. N. Binary Matrices in System Modeling. *IEEE Transactions on Systems, Man, and Cybernetics*. 1973, SMC-3, č. 5, s. 441–449. DOI: 10.1109/TSMC.1973.4309270.

[26] Xu, Y., Ma, Z. a Sun, J. Simulation of turning vehicles' behaviors at mixed-flow intersections based on potential field theory. *Transportmetrica B: Transport Dynamics*. Březen 2018, sv. 7, s. 1–21. DOI: 10.1080/21680566.2018.1447407.

[27] Zaitsev, D. K-neighborhood for Cellular Automata. *ArXiv*. Květen 2016, abs/1605.08870.

Příloha A

Obsah CD

Struktura přiloženého CD je následující:

/text/xhauer02-traffic.pdf	Text diplomové práce ve formátu pdf
/text/zadani.pdf	Zadání diplomové práce ve formátu pdf
/text/	Zdrojové soubory textu diplomové práce v LaTeX
/doc/	Programová dokumentace vytvořená aplikací Doxygen
/readme.txt	Návod na sestavení aplikace
/src/	Zdrojové kódy výsledné aplikace
/src/images/	Všechny ikony použité v aplikaci

Příloha B

Návod na sestavení aplikace

Cílem této kapitoly je poskytnout návod, pomocí kterého lze vytvořenou aplikaci přeložit na svém počítači. Přeložení aplikace je možné provést pomocí aplikace Qt Creator, kterou je nutné na daný počítač nejdříve nainstalovat. Překlad aplikace dále vyžaduje přítomnost knihovny OpenCV na daném počítači. V projektovém souboru je proto potřeba nastavit správné umístění knihovny OpenCV v závislosti na operačním systému. Po správném nastavení tohoto umístění by již aplikace měla být přeložitelná.

Překlad vyžaduje otevření celého projektu v aplikaci Qt Creator, což zahrnuje zvolení projektového souboru pomocí zobrazeného dialogového okna. Následně dojde k automatickému otevření celé struktury projektu včetně všech zdrojových souborů. V horní liště pak lze otevřít nabídku s možnostmi překladu, kde se také nachází možnost přeložení celého projektu. Provedením této akce dojde k zahájení překladu, kdy po jeho dokončení lze aplikaci spustit odpovídajícím tlačítkem v levé liště.

Příloha C

Testovací protokol

1. Lze uživatelské rozhraní považovat za přehledné a intuitivní?
2. Je uživatelské rozhraní vhodně vizuálně zpracované?
3. Je uživatelské rozhraní dostatečně propracované?
4. Je vykreslování trajektorií přehledné a dobře vizuálně zpracované?
5. Líbí se vám zvolená množina barev trajektorií podle kategorií?
6. Zdá se vám vytváření prostorových elementů intuitivní?
7. Je tvorba prostorových elementů dostatečně přesná a pohodlná?
8. Máte nějaký lepší nápad, jak by šlo vytvářet prostorové elementy?
9. Jak se vám manipuluje s jednotlivými body prostorových elementů?
10. Máte nějaké potíže s úpravou náhledu na vykreslenou scénu?
11. Jaký máte názor na postranní lištu a její rozložení?
12. Jaký je váš první dojem na vzhled filtračního stromu?
13. Přijde vám použití filtračního stromu jako dobrý nápad?
14. Zdá se vám vytváření nových operátorů intuitivní?
15. Je manipulace s operátory dostatečně přesná a pohodlná?
16. Zdá se vám grafické provedení panelu s operátory zdařilé?
17. Jak se vám líbí provedení dialogů pro nastavení operátorů?
18. Jak se vám líbí vizualizace výstupních počtů trajektorií?
19. Zdá se vám přesnost prostorové filtrace dostatečná?
20. Je podle vás absence vykreslování videa velkým problémem?
21. Jaký máte názor na jednotky v pixelech u dynamických operátorů?

22. Je nějaký další typ prostorového elementu, který by měl být přidán?
23. Bylo by podle vás dobré doplnit některé typy operátorů?
24. Chybí vám v aplikaci nějaká další funkcionalita nebo nástroj?

Příloha D

Vyhodnocení testování

1. Lze uživatelské rozhraní považovat za přehledné a intuitivní?

   - Nejdelší diskuze v rámci celého uživatelského testování náležela odpovědi právě na první otázku. Jisté připomínky k přehlednosti uživatelského rozhraní mělo všech šest testerů. Nejvíce otázek pak bylo k filtračnímu stromu, kdy testeři zpočátku moc nechápali, k čemu vlastně strom slouží. Stejná situace byla také v případě panelu s operátory a vytváření nových operátorů. Všichni testeři se ale poměrně rychle dokázali s uživatelským rozhraním seznámit a byli schopni vytvářet vlastní filtrační strom. Dále některým testerům činilo potíže odlišit význam přehledu kategorií v postranní liště od samotné filtrace podle kategorií. Někteří testeři dále navrhovali přesun postranní lišty pod samotnou scénu. Po pochopení principu rozhraní již testeři dále neměli potíže s ovládáním aplikace a samotné rozhraní následně označili jako vydařené a úsporné z hlediska potřebného prostoru v okně aplikace.

2. Je uživatelské rozhraní vhodně vizuálně zpracované?

   - Testerům se líbilo barevné provedení jak vizualizace trajektorií, tak také celého uživatelského rozhraní. Odstíny barev prvků v uživatelském rozhraním jsou podle testerů správně zvoleny a celé prostředí je podle nich přehledné a konzistentní právě díky volbě odstínů šedé a bílé barvy. Barvy operátorů jednotlivých kategorií jsou podle testerů také v pořádku a dobře vyniknou na pozadí plátna.

3. Je uživatelské rozhraní dostatečně propracované?

   - Z hlediska propracovanosti uživatelského rozhraní bylo několik poznámek a návrhů, ale jinak testerům nic nescházelo. Mezi poznámkami byla například poměrně malá šířka plátna pro filtrační strom a panel s operátory. Dále by měla být přítomna možnost dočasného zabalení celého panelu s operátory tak, že by byl viditelný pouze nadpis. Toto by mohlo být užitečné především v případě, kdy chceme mít na plátně co nejvíce prostoru pro vizualizaci.

4. Je vykreslování trajektorií přehledné a dobře vizuálně zpracované?

   - Někteří testeři měli poznámku ohledně situace, kdy trajektorie byly vykreslovány ve velkém množství v jednom pruhu. V takové situaci je údajně trochu problematické rozeznat trajektorii konkrétního typu vozidla v celém shluku. V tento moment bylo poukázáno na vizualizaci kategorií v postranní liště, kdy testeři následně uznali užitečnost této vizualizace.

5. Líbí se vám zvolená množina barev trajektorií podle kategorií?

   - V rámci odpovědi na tuto otázku nebyly ze strany testerů nějaké připomínky. Volba různých barev trajektorií dle kategorií je podle nich dobrý nápad.

6. Zdá se vám vytváření prostorových elementů intuitivní?

   - Vytváření nových prostorových elementů pomocí kreslení vlastní křivky je dle testerů velmi intuitivní a snadno pochopitelné i pro běžné uživatele. Dle testerů by bylo užitečné, kdyby šlo prostorovými elementy rotovat tak, jak je to běžné v grafických editorech. Dva testeři dále navrhli vykreslování zón a regionů v jiných barevných odstínech, což by údajně vedlo k lepší přehlednosti scény.

7. Je tvorba prostorových elementů dostatečně přesná a pohodlná?

   - Testeři pochválili přesnost tvaru nově vzniklého regionu, který opravdu odpovídá nakreslené křivce. Některým z nich trochu vadilo to, že často vzniká region s velmi ostrými hranami. Tito lidé také navrhli implementaci ořezání ostrých hran u nových regionů. Podle dvou testerů by bylo dobré vypnout režim tvorby elementu po jeho vytvoření, protože si to někdy neuvědomili a místo posunu náhledu dále kreslili.

8. Máte nějaký lepší nápad, jak by šlo vytvářet prostorové elementy?

   - Jednomu testerovi by se líbila možnost vytvoření polygonu pomocí klepnutí myší na body scény, kde by se nacházely řídící body regionu. V aplikaci by přitom mohlo jít snadno rozeznat kreslení od vytváření řídících bodů.

9. Jak se vám manipuluje s jednotlivými body prostorových elementů?

   - Testerům se líbila možnost vytváření nových řídících bodů dvojitým poklepáním na hranu regionu. Stejně také v případě mazání řídících bodů. Testerům se poměrně často stávalo, že hranu regionu minuli a došlo k odznačení elementu, což je dle nich docela nepříjemné. Tato situace by se dala řešit více způsoby. Například by šlo rozšířit plochu hrany regionu pro větší přesnost.

10. Máte nějaké potíže s úpravou náhledu na vykreslenou scénu?

    - Testeři neměli potíže s přiblížením, oddálením a posunem náhledu na vykreslovanou scénu. Citlivost přibližování je podle nich ideální.

11. Jaký máte názor na postranní lištu a její rozložení?

    - Testerům se postranní lišta zdá v pořádku z hlediska vzhledu a volby barev. Nicméně zpočátku nedokázali rozlišit mezi přehledem kategorií v postranní liště a samotnou filtrací trajektorií podle kategorie. Po vysvětlení principu vizualizace kategorií v postranní liště již testeři neměli potíže pochopit její význam.

12. Jaký je váš první dojem na vzhled filtračního stromu?

    - První dojmy vývojářů na filtrační strom byly pozitivní. Líbil se jim vzhled samotného stromu a volba barev operátorů, spojení a také plátna. Nicméně jistou dobu trvalo, než pochopili samotný účel filtračního stromu.

13. Přijde vám použití filtračního stromu jako dobrý nápad?

    - Myšlenka sestavování vlastní filtrační struktury testery překvapila. Líbila se jim především myšlenka přizpůsobit si filtrační strom svým potřebám a zvolené scéně. Dále měl jeden člověk dotaz, zda by šlo realizovat funkci, která efektivním způsobem znovu seřadí existující strom tak, aby byly shodné jeho výstupy, ale jeho vyhodnocení by bylo rychlejší. Tento nápad by určitě šlo implementovat v navazující práci.

14. Zdá se vám vytváření nových operátorů intuitivní?

    - Testeři byli poměrně rychle schopni vytvářet nové operátory jejich přesunem z panelu na plátno. Situace byla trochu komplikovaná v případě vytváření prostorových operátorů. Princip přesunu elementu na plátno bylo nejdříve potřeba lidem upřesnit. Po vysvětlení tohoto principu vytváření prostorových elementů již bylo všechno v pořádku. Vytváření spojení operátorů je podle dvou testerů poměrně nepřesné. Situaci by údajně zlepšil větší konektor operátorů a také možnost tvorby spojení rovnou z vlajky operátoru.

15. Je manipulace s operátory dostatečně přesná a pohodlná?

    - Manipulace se samotnými operátory je dle testerů naprosto v pořádku. Trochu jim vadila malá výchozí šířka plátna, což je často nutilo používat tlačítko na rozšíření šířky plátna. Dále také jeden tester navrhl funkcionalitu, v rámci níž by bylo možné nahradit konkrétní uzel filtračního stromu. Při přesunu operátoru z plátna by dle něho pouze stačilo najet na zvolený uzel, který by byl nahrazen novým operátorem. Dvojice testerů dále přišla s myšlenkou vytvářet si vlastní kombinované operátory, které jsou tvořeny celým filtračním stromem, ale lze s nimi manipulovat jako s klasickým operátorem. Tato funkcionalita by se rozhodně dala implementovat v rámci rozšíření aplikace, protože by mohla být opravdu užitečná.

16. Zdá se vám grafické provedení panelu s operátory zdařilé?

    - Panel s operátory je dle testerů také v pořádku. Možnost přesunu panelu je dle nich dobrý nápad. Jeden člověk dále navrhl možnost skrytí celého panelu tak, aby zůstal viditelný jenom nadpis. Tímto způsobem by pak mohlo být ušetřeno místo na plátně v případě vizualizace filtračního stromu.

17. Jak se vám líbí provedení dialogů pro nastavení operátorů?

    - Zadání rozsahu časových atributů musí jít napsat ručně a ne pouze nastavením pomocí posuvníku. Tato připomínka zazněla od většiny testerů. Dále by bylo dobré přidat tlačítko na obrácený výběr množiny barev a kategorií v dialogu.

18. Jak se vám líbí vizualizace výstupních počtů trajektorií?

    - Vizualizace výstupních počtů trajektorií ve stromě je podle testerů naprosto v pořádku a navíc je vše dostatečně přehledné.

19. Zdá se vám přesnost prostorové filtrace dostatečná?

- Testeři u prostorové filtraci nerozeznali omezenou přesunout. Následovalo upřesnění situace s maticovou reprezentací scény a omezením přesnosti prostorové filtrace. Testeři uznali, že by filtrace mohla být výpočetně náročná při rozsáhlých scénách. Také uvítali možnost zvolení míry přesnosti filtrace v konfiguračním souboru.

20. Je podle vás absence vykreslování videa velkým problémem?

    - Testeři byli poměrně překvapeni po zjištění toho, že nedochází k vykreslování videa, ale pouze ukázkového snímku. Následně bylo lidem upřesněno, že implementace vykreslování scény včetně videa je komplikovaná a byla by potřeba synchronizace obrazu a filtrace. Dle testerů je pro účely demonstrace filtrace dostatečná aktuální implementace. Nicméně vykreslování videa je funkcionalita, kterou by šlo implementovat v navazující práci.

21. Jaký máte názor na jednotky v pixelech u dynamických operátorů?

    - Většina testerů neměla problém s jednotkami operátorů rychlosti a akcelerace, které jsou v pixelech. Jeden tester očekával metrické jednotky rychlosti, které odpovídají dané scéně. Následně mu bylo vysvětleno, že by bylo nutné implementovat georeferencování scény a komplikovanost tohoto procesu.

22. Je nějaký další typ prostorového elementu, který by měl být přidán?

    - Testeři navrhovali přidat podporu pro směrový element, který by reprezentoval směr pohybu objektů. Tento element by mohlo jít vytvářet analogicky jako v případě existujících prostorových elementů.

23. Bylo by podle vás dobré doplnit některé typy operátorů?

    - Jeden tester navrhl přidání podpory pro operátor, který by filtroval na základě limitu vzájemné vzdálenosti objektů scény.

24. Chybí vám v aplikaci nějaká další funkcionalita nebo nástroj?

    - Tři testeři by uvítali možnost nastavení vlastních barevných odstínů pro jednotlivé operátory. Momentálně je barva operátoru určena jeho kategorií a nelze ji ručně změnit.