FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Spojité plánování pohybu pro autonomní vozy na křižovatkách
Student:	Ladislav Miklík
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je navrhnout techniky pro naplánování současného bezkolizního průjezdu více autonomních automobilů křižovatkou. Předpokládáme, že u autonomních vozů lze na dopravní předpisy pohlížet novým způsobem a například na křižovatkách tak nebude nutné dávat přednost ani respektovat semafory, ale místo toho budou moci všechna vozidla projíždět křižovatkou současně. Plánovací techniky by měly pracovat přímo na úrovni spojitého prostoru a brát v úvahu kinematiku automobilu. Úkoly řešitele jsou následující:

1. Prostuduje relevantní algoritmy pro plánování spojitého pohybu více entit, jak je například studováno v mobilní robotice a v multi-robotických systémech.
2. Na základě provedené rešerše navrhněte vlastní metodu vhodnou pro případ křižovatky s více autonomními automobily.
3. Návrh implementuje jako softwarový prototyp a otestujte jej v syntetických scénářích, například v on-line režimu, kdy křižovatku vozidla opouštějí a přijíždějícím vozidlům je potřeba vytvořit plán průjezdu.

–

[1] Shravan Krishnan, R. Govind Aadithya, Rahul Ramakrishnan, Vijay Arvindh, Sivanathan K: A Look at Motion Planning for AVs at an Intersection. ITSC 2018: 333-340
[2] Rahul Shome, Kiril Solovey, Andrew Dobson, Dan Halperin, Kostas E. Bekris: dRRT*: Scalable and informed asymptotically-optimal multi-robot motion planning. Auton. Robots 44(3-4): 443-467 (2020)
[3] Steven M. LaValle: Planning algorithms. Cambridge University Press 2006, ISBN 978-0-521-86205-9, pp. I-XVI, 1-826

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 25. listopadu 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Spojité plánování pohybu pro autonomní vozy na křižovatkách

Ladislav Miklík

Katedra aplikované matematiky
Vedoucí práce: doc. RNDr. Pavel Surynek, Ph.D.

10. února 2022

Poděkování

Tímto bych chtěl poděkovat vedoucímu této práce, panu doc. RNDr. Pavlovi Surynkovi, Ph.D., za ochotu a pomoc skrze konzultace a rady, které mi pomohly vydat se správným směrem. Dále bych chtěl poděkovat své rodině za podporu a trpělivost při mém studiu. Bez nich bych práci nenapsal.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 10. února 2022 . . . . . .

Odkaz na tuto práci

Miklík, Ladislav. *Spojité plánování pohybu pro autonomní vozy na křižovatkách.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato práce seznamuje čtenáře s problematikou spojitého plánování více entit na křižovatce a představuje vlastní metodu pro řešení této problematiky. Implementovaná metoda je založená na reprezentování proveditelných stavů autonomního vozidla pomocí stavové mřížky, diskretizaci vzniklého koordinačního prostoru a následné jeho prohledání pomocí A* algoritmu pro optimální cestu. Podařilo se dosáhnout propustnosti 2,93 aut za sekundu u optimálnějšího přístupu a 2,47 aut za sekundu při počítání v reálném čase (7,6ms pro výpočet).

Klíčová slova spojité plánování, stavová mřížka, A*, diskretizace, multiagentní, Godot

Abstract

This work serves as an introduction into the topic of continuous multi-agent path planning and presents its own solution to this problem. The implemented method is based on the representation of feasible paths of a vehicle, discretization of a coordination space and searching this space for an optimal path utilizing A* algorithm. A throughput of 2.93 cars per second was reached using an optimal approach and 2.47 cars per second using a real-time computation (7.6ms to compute).

Keywords continuous planning, state lattice, A*, discretization, multi-agent, Godot

Obsah

Úvod **1**

1 Cíl práce **3**

I Teorie **5**

2 Teoretický základ **7**
2.1 Plánování 7
2.2 Geometrická reprezentace 8
2.3 Konfigurační prostor 9
2.4 Modely vozidla 11
2.4.1 Differential drive 11
2.4.2 Unicycle model 11
2.4.3 Bicycle model 12

3 Plánovací algoritmy **13**
3.1 Probabilistic road map 13
3.2 Rapidly-exploring Random Tree 15
3.3 Artificial Potential Field 16

4 Problém křižovatky **19**
4.1 Multi-agentní plánování 19
4.2 Současná řešení 21

II Vlastní metoda **25**

5 Návrh algoritmu **27**
5.1 Požadavky 27

5.2 Metoda 28
5.3 Bézierovy křivky 29
5.4 A* 29

6 Implementace **31**
6.1 Použité technologie 31
6.2 Křižovatka 31
6.3 Model auta 32
6.4 Detekce kolizí 34
6.5 Plánovač 34
6.5.1 Fáze zachycení 34
6.5.2 Fáze plánování 35

7 Výsledky **37**
7.1 Srovnání 37
7.2 Experimenty 39

Závěr **41**

Bibliografie **43**

A Seznam použitých symbolů **49**

B Seznam použitých zkratek **51**

C Obsah přiloženého CD **53**

Seznam obrázků

2.1 Konfigurační prostor tuhých těles 10
2.2 Bicycle model 12

3.1 Srovnání RRT a RRT* 16
3.2 Složky U_{att}, U_{rep} a konstrukce U 18

6.1 Metoda v akci 33
6.2 Stavová mřížka 33

Seznam tabulek

3.1 Porovnání PRM, RRT, RRT* 16

7.1 Porovnání algoritmů 37
7.2 Průměrný čas strávený na křižovatce 40
7.3 Průměry časů výpočtu plánu 40
7.4 Propustnost 40

Úvod

Pohyb a jeho plánování je pro člověka triviální záležitost. Je to pro nás něco tak přirozeného, že si většina z nás ani neuvědomí, že k nějakému plánování vůbec došlo. Vezměme si například situaci, kdy se chceme napít vody ze skleničky. Lidský mozek v mžiku vyhodnotí, že nejlepší volbou je natáhnout ruku ke skleničce, uchopit ji a dát ji pod úhlem k puse tak, aby voda vtekla rovnou do pusy. Co když je ale mezi námi a skleničkou nějaká menší překážka jako třeba kniha o plánování pohybu? Žádný problém. Prostě natáhneme ruku přes knihu. Ani ve snu by nás nenapadlo, že bychom měli natáhnout ruku skrze knihu, protože víme, že se akorát srazíme. Tohle ovšem humanoidní robot provádějící stejnou akci neví a je pouze na nás, abychom ho naučili, co je v našem světě možné a že zkrátka nejde, aby se pokoušel natahovat ruce skrze knihy. Analogicky funguje i plánování pohybu auta na křižovatce, ať už ho ovládá člověk nebo robot. Řidič auta musí správně reagovat na okolní prostředí a další účastníky provozu tak, aby nedošlo ke kolizi. Ovšem s přibývajícím počtem aut na silnicích se stává tento problém pouze těžší a těžší.

Současné studie předpovídají, že dopravní zácpy se navýší až o 60% do roku 2030 [1]. Hlavním viníkem těchto zácp je způsob, jakým je řízena doprava v oblasti křižovatky. Současným řešením je řízení pomocí světelných signálů. Ty jsou ovšem neefektivní v případě, kdy je objem aut přijíždějících na křižovatku příliš vysoký [2]. Navíc více jak 44% všech dopravních nehod je způsobeno v oblasti křižovatky [3]. Má tedy smysl se dívat po lepších řešeních. Jednou z alternativ je zbavit se světelných signálů (a lidí) úplně a ponechat veškeré řízení na umělé inteligenci. Lidé by se pouze nechali vozit jako pasažéři a veškerá zodpovědnost řízení by zůstala na robotech. Toto není tak vzdálená budoucnost, jak by se mohlo na první pohled zdát. Se společnostmi jako je Tesla Inc. vedoucí výzkum autonomizace dopravy můžeme říct, že se jedná o velice atraktivní oblast výzkumu a je pouze otázka času, než uvidíme první plně automatizované křižovatky.

Stále tady ale zůstává otázka, jak něco tak „triviálního“ jako je řízení auta

na křižovatce naučit roboty? V této práci se budu snažit zodpovědět tuto otázku a doufám, že přečtením této práce si čtenář odnese alespoň základní poznatky z oblasti spojitého plánování pohybu více entit.

Kapitola 1

Cíl práce

Cílem práce je seznámit čtenáře s teoretickými poznatky z oblasti plánování pohybu a multiagentního plánování. Představíme si koncept plánování, jak matematicky modelovat roboty ve světě a způsob, jak se v tomto světě mohou pohybovat pomocí abstrakce v konfiguračním prostoru. Dále se seznámíme s populárními plánovacími algoritmy jako jsou *Probabilistic Road Map*, *Rapidly-exploring Random Tree* a *Artificial Potential Field*. Ukážeme si jak tyto metody můžeme použít v případě, kdy musíme naplánovat pohyb pro více robotů kde tito roboti musí spolu nějakým způsobem spolupracovat, aby nedošlo ke srážce.

V druhé části práce navrhneme vlastní algoritmus pro plánování pohybu více aut na křižovatce a ukážeme si jeho implementaci v herním enginu Godot.

Od čtenáře se nepředpokládají žádné rozsáhle znalosti a i laický čtenář by měl být schopen práci pochopit. Předpokládaná je pouze znalost časové a paměťové asymptotické složitosti.

Část I

Teorie

Kapitola 2

Teoretický základ

V této kapitole se seznámíme se základními pojmy spojenými s plánováním pohybu a multi-agentním navigováním. Představíme si také odbornou literaturou, která se věnuje dané problematice. Tato kapitola vychází především z *LaValleho* práce [4] a uvádí pouze to nejnutnější pro úvod do problematiky plánování pohybu. Pro hlubší porozumění doporučuji si přečíst právě zmíněnou knihu, avšak pro pochopení této práce to není nutné.

2.1 Plánování

Tato práce se zabývá plánováním pohybu. Je tedy nutné si sjednotit, co si představujeme pod pojmem *plánování*. Pro naše potřeby si plánování definujeme následovně:

Definice 1 (Plánování) je proces hledání *časově* označených posloupností *akcí*, které vedou z počátečního *stavu* do koncového, volené podle *kritéria*.

V naší definici můžeme najít několik pojmů, které je vhodné si ujasnit, co znamenají:

Stav — Stav je prvek *stavového prostoru*. Stavový prostor reprezentuje všechny možné situace, které mohou při plánování nastat. Může být *diskrétní*, např. všechny možné šachové pozice tvoří stavový prostor a konkrétní šachová pozice je stav. Nebo může být *spojitý*, např. pozice a směr vozidla na silnici.

Čas — Všechny plánovací problémy mají nějaké vnímání času. Ten může být *explicitní*, jako třeba přesně označené pozice aut při průjezdu křižovatkou, nebo může být *implicitní*, kdy nám stačí vědět, v jakém pořadí se mají akce vykonat.

Akce — Plán generuje akce, které manipulují stav. Při formulaci plánu musíme někde specifikovat, jak se změní stav, když provedeme danou akci. Toho můžeme docílit definováním přechodové funkce u diskrétních problémů nebo definováním diferenciálních rovnic u spojitých problémů.

Počáteční a koncový stav — Problém plánování většinou obsahuje nějaký počáteční stav, ze kterého se snažíme pomocí posloupnosti akcí dostat do specifického koncového stavu nebo do nějakého stavu z množiny koncových stavů.

Kritérium — Kritérium nám říká, jaký plán se má vygenerovat s ohledem na stav a akce, které se mají vykonat. Většinou nás zajímají plány, které splňují jedno z následujících dvou kritérií:

1. *Proveditelnost* — Vygeneruj plán, který se dostane do cílového stavu bez ohledu na jeho efektivnost.

2. *Optimálnost* — Vygeneruj proveditelný plán, který se dostane do cílového stavu efektivně. Zde je efektivita nějaká námi specifikovaná vlastnost, kterou se snažíme optimalizovat. Formálně je to nějaká účelová funkce, kterou se snažíme minimalizovat (případně maximalizovat).

Vlastnosti plánování uvedené výše jsou aplikovatelné na širokou škálu plánovacích problémů — řešení logických hádanek, stěhování nábytku, vaření oběda. Můžeme říct, že téměř vše, co v životě děláme, jsou pouze akce nějakého plánu. Stejně tak se vlastnosti výše vztahují k plánování pohybu, kterým se budeme zabývat v této práci.

Základní příklad plánování pohybu je *Piano mover's problem*. Neformálně řečeno jde o problém, kdy dostaneme 3D model piana a snažíme se ho přestěhovat z jedné místnosti do druhé bez toho, aniž bychom do něčeho narazili. Složitost problému spočívá v netriviálním tvaru piana a prostředí, ve kterém piano přesouváme. Jedná se samozřejmě pouze o abstrakci. Pod pianem si budeme moct představit autonomní vozidlo a prostor domu bude křižovatka. Formálně si problém definujeme v sekci 2.3.

2.2 Geometrická reprezentace

Předtím, než budeme moct začít plánovat pohyb vozidel na křižovatce, je potřeba si představit, jak taková křižovatka vypadá z matematické perspektivy. Prvním krokem je zadefinování *světa*, ve kterém se křižovatka nachází. Pro konzistenci ve značení s [4] budeme svět značit $\mathcal{W}$. Nejpraktičtější světy pro plánování pohybu na křižovatce jsou 2D nebo 3D, tedy $\mathcal{W} = \mathbb{R}^2$ nebo $\mathcal{W} = \mathbb{R}^3$. Svět zpravidla obsahuje:

1. *Překážky* — Oblast světa, která je trvale obsazena. Může se jednat o fyzickou překážku jako například zdi budov, ale můžeme brát jako překážku i něco víc abstraktnějšího, jako nějakou oblast, kam robot nesmí vjet. Například chodník či tráva v případě autonomních vozidel na křižovatce.

2. *Roboty* — Tělesa ovladatelné pohybovým plánem. V kontextu plánování je tento termín zaměnitelný se slovy *agent* či v angličtině *decision maker*. My navíc budeme používat termín *autonomní vozidlo* (zkráceně AV)

V kontextu geometrické reprezentace není mezi překážkami a roboty žádný rozdíl. Obecně si můžeme u obojího vybrat mezi *mezní reprezentací* nebo *pevnou reprezentací*. [4] Mezní reprezentace modeluje pouze hranice objektu zatímco pevná reprezentace modeluje všechny jeho body.

Pro zadefinování Piano mover's problem budeme dále potřebovat znát semi-algebraické množiny: „Množina bodů vyjádřena jediným primitivním polynomem se nazývá *algebraická množina*; množina bodů, která se dá získat konečným počtem sloučení a průniků algebraických množin se nazývá *semi-algebraická množina*“ [4, vlastní překlad]

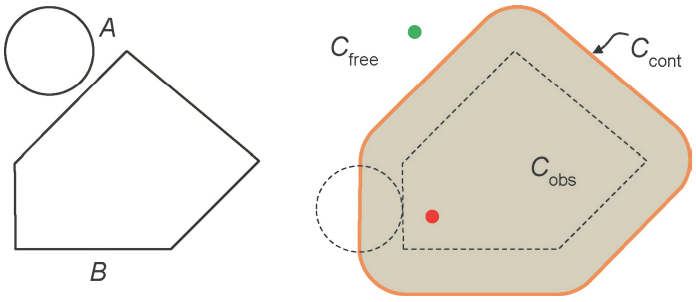
2.3 Konfigurační prostor

„Plánování pohybu si můžeme rozdělit na dva kroky: 1) Konstrukce datové struktury reprezentující geometrické omezení světa $\mathcal{W}$ a 2) Prohledání této datové struktury k nalezení řešení.“ [5, vlastní překlad] Konfigurační prostor, poprvé představen *Lozano-Perezem* v [6], je právě jedna z takových struktur. Běžně se značí $\mathcal{C}$ nebo C-Space a je to prostor všech konfigurací robota. Pro představu si to ukážeme na příkladu. Mějme libovolné tuhé těleso v 3D prostoru, například stavební cihlu. Tuto cihlu můžeme jednoznačně reprezentovat třemi souřadnicemi x, y, z pro její polohu a třemi hodnotami θ, ϕ, ψ její úhlové výchylky. Konfigurace této cihly je potom uspořádaná 6-tice $(x, y, z, \theta, \phi, \psi)$, kde tyto parametry jednoznačně určují pozici každého bodu cihly ve světě $\mathcal{W}$. Množina všech konfigurací této cihly potom tvoří konfigurační prostor dimenze 6 — $\mathcal{C} = \mathbb{R}^3 \times \mathbb{S}^3$. Konfigurační prostor nám ale nemusí reprezentovat pouze tuhá tělesa. Vezmeme-li si například průmyslového robota, zafixované rameno se dvěma rotačními klouby, pak jeho konfigurační prostor bude mít tvar toroidu $\mathcal{C} = \mathbb{S}^2$ a tedy jakákoliv konfigurace takového ramena se dá vyjádřit jako bod na povrchu toroidu. Obecně platí, že pro každý stupeň volnosti robota potřebujeme jeden parametr v konfiguraci a to znamená dimenzi navíc v konfiguračním prostoru.

Pokud navíc budeme uvažovat prostor s překážkami, objevíme skutečnou sílu konfiguračního prostoru. Konfigurační prostor $\mathcal{C}$ si rozdělíme na dvě podmnožiny $\mathcal{C}_{obs}$ a $\mathcal{C}_{free}$. $\mathcal{C}_{obs}$ je množina konfigurací, kdy robot koliduje s překážkou. Naopak $\mathcal{C}_{free}$ je množina konfigurací, kdy nedochází k žádné kolizi. Z definice tedy platí, že $\mathcal{C} = \mathcal{C}_{free} \cup \mathcal{C}_{obs}$. Výhodou této abstrakce je, že problém plánování pohybu se nám tímto zredukuje na nalezení spojité křivky v $\mathcal{C}_{free}$ reprezentující posloupnost nekolizních konfigurací robota.

Vybízí se otázka, jak takový prostor zkonstruovat. Tímto se zabývá *Pan* a *Manocha* v [7]. Nechť $\mathcal{A} \subset \mathcal{W}$ je robot a $\mathcal{O} \subset \mathcal{W}$ je oblast překážek ve světě $\mathcal{W}$. Ve speciálním případě, kdy $\mathcal{A}$ a $\mathcal{O}$ jsou obojí tuhé tělesa, $\mathcal{C}_{obs}$ se rovná

Minkowského sumě mezi $\mathcal{A}$ a $\mathcal{O}$: $\mathcal{C}_{obs} = \mathcal{A} \oplus (-\mathcal{O}) = \{x = x_a + x_o \mid x_a \in \mathcal{A}, x_o \in -\mathcal{O}\}$. [7]. Příklad takového prostoru můžeme vidět na obrázku 2.3. I v tomto speciálním případě je výpočetní složitost vysoká — $O(mn)$, pokud jsou $\mathcal{A}$ a $\mathcal{O}$ konvexní objekty [8], a $O(m^3n^3)$, pokud jsou nekonvexní, kde m a n je počet trojúhelníku v $\mathcal{A}$ a $\mathcal{O}$. [7]



Obrázek 2.1: Konfigurační prostor tuhých těles

Explicitní reprezentace konfiguračního prostoru není vždy nutná a v mnoha případech není ani možná. V případě, kdy naši roboti nejsou pouze tuhá tělesa (což je většina případů), potřebujeme použít jinou techniku. Tou budou *vzorkovací algoritmy* v kapitole 3

Se znalostí konfiguračního prostoru si můžeme formálně zadefinovat *Piano mover's problem* poprvé představen v části 2.1

Definice 2 (Piano mover's problem [4])

1. *Svět* $\mathcal{W}$, kde $\mathcal{W} = \mathbb{R}^2$ nebo $\mathcal{W} = \mathbb{R}^3$
2. Semialgebraická oblast překážek $\mathcal{O}$ ve světě $\mathcal{W} \Leftrightarrow \mathcal{O} \subset \mathcal{W}$
3. Semialgebraický model robota ve světě $\mathcal{W}$. Ten může být definován jako tuhé těleso $\mathcal{A}$ nebo jako kolekce n spojnic $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$
4. Konfigurační prostor $\mathcal{C}$ definovaný množinou všech možných transformací, které lze aplikovat na robota. Z tohoto odvodíme $\mathcal{C}_{free}$ a $\mathcal{C}_{obs}$
5. Konfiguraci $q_I \in \mathcal{C}_{free}$ označíme jako *počáteční konfiguraci*
6. Konfiguraci $q_G \in \mathcal{C}_{free}$ označíme jako *koncovou konfiguraci*. Počáteční a koncová konfigurace se často označují jako *dotazovací pár* (q_I, q_G)
7. Úplný algoritmus vypočítá spojitou dráhu $\tau : [0, 1] \to \mathcal{C}_{free}$ takovou, kde $\tau(0) = q_I$ a $\tau(1) = q_G$ nebo správně vypoví, že taková dráha neexistuje

Všimněme si, že naše definice *Piano mover's problem* zmiňuje pouze jednoho robota. To není omyl. V sekci 4.1 si ukážeme přímočaré rozšíření pro multi-agentní plánování.

2.4 Modely vozidla

Specifická vlastnost pro vozidla je jejich neschopnost pohybovat se do stran. Takovéto systémy nazýváme *underactuated* [9] a vznikají v situacích, kdy ovladatelný stupeň volnosti je menší, než celkový stupeň volnosti. V této části textu si představíme matematické modely, které splňují tato omezení a simulují chování různých robotů.

2.4.1 Differential drive

První model, který si představíme, je *differential drive*. Auto reprezentované takovým modelem má pouze 2 kola, kde každé kolo je připojeno k vlastnímu motoru operující nezávisle na druhém. Jednoduše řečeno se jedná o tank. Trošku umírněnějším příkladem by mohl být například OZOBOT – jednoduchý programovatelný robot určený především pro výuku programování.

Pro konstrukci takového modelu potřebujeme znát pouze vzdálenost mezi dvěma koly robota L a poloměr kol r. Differential drive má pouze dva kontrolní prvky – úhlová rychlost levého a pravého kola (u_l, u_r). Pokud $u_l = u_r \neq 0$, pak se robot pohybuje dopředu či dozadu ve směru kol. Pokud $u_t = -u_r \neq 0$, pak se robot otáčí kolem středového bodu nápravy. Podle těchto poznatků by přechodová funkce mohla vypadat následovně [4]:

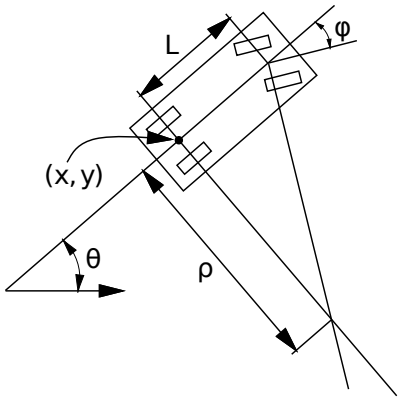
$$\begin{aligned}\dot{x} &= \frac{r}{2}(u_l + u_r)\cos\theta \\ \dot{y} &= \frac{r}{2}(u_l + u_r)\sin\theta \\ \dot{\theta} &= \frac{r}{L}(u_r - u_l)\end{aligned}$$

2.4.2 Unicycle model

Unicycle model je model simulující jednokolku. V mnoha ohledech je podobný differential drive modelu, ovšem rotace tentokrát nevzniká z rozdílu úhlových rychlostí kol, ale je tvořena přímo jezdcem. Přechodová funkce vypadá následovně [4]:

$$\begin{aligned}\dot{x} &= r u_\sigma \cos\theta \\ \dot{y} &= r u_\sigma \sin\theta \\ \dot{\theta} &= u_\omega\end{aligned}$$

kde u_σ je úhlová rychlost kola a u_ω je rotace vyvolaná jezdcem. Všimněme si, že pokud použijeme substituci $u_\sigma = \frac{u_l+u_r}{2}$, $u_\omega = r(u_r - u_l)$ a $L = 1$, dostaneme přechodovou funkci pro differential drive. Pomocí differential drive můžeme tedy simulovat chování jednokolky.



Obrázek 2.2: Bicycle model; Zdroj: [4]

2.4.3 Bicycle model

Bicycle model, někdy nazývaný „Simple Car“ [4] nebo také „Car-like robot“ [10], je jeden z nejjednodušších kinematických modelů, kterým můžeme reprezentovat auto ve světě. Zároveň je to první námi představený model, který už se alespoň trochu podobá chování reálného auta. Budeme-li na chvíli ignorovat rolující omezení, konfigurační prostor auta je stejný jako tuhé těleso v rovině $\mathcal{W} = \mathbb{R}^2$, tedy $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$. Jakoukoliv konfiguraci $q \in \mathcal{C}$ takového vozidla tedy můžeme značit jako $q = (x, y, \theta)$, kde x, y je pozice středu zadní nápravy a θ je směr vozu. Nechť ϕ reprezentuje úhel předních kol relativní ke směru vozu θ a L je vzdálenost mezi přední a zadní nápravou. Pokud ponecháme úhel zatáčení ϕ stejný, trajektorie auta bude kružnice o poloměru ρ viz. diagram 2.2. Přechodová funkce vypadá následovně [4]:

$$\begin{aligned} \dot{x} &= u_s \cos\theta \\ \dot{y} &= u_s \sin\theta \\ \dot{\theta} &= \frac{u_s}{L} \tan\phi \end{aligned}$$

kde u_s je rychlost vozidla a u_ϕ je úhel zatáčení. *Polack et al.* v [11] srovnávali bicycle model s reálnějšími modely s až 9 stupni volnosti. Ukázali, že bicycle model je vhodný model pro plánování pohybu, pokud se laterální zrychlení limituje na $0.5g$.

Existují dále 2 varianty bicycle modelu. Reeds-Sheppovo auto [12] je varianta s omezením na rychlosti $u_s \in \{-1, 0, 1\}$. Odstraněním možnosti jet dozadu z Reeds-Sheppova auta dostaneme Dubinovo auto. V roce 1957, *L. E. Dubin* dokázal [13], že nejkratší cesta mezi dvěma stavy (x, y, θ) dopředu se pohybujícího vozidla s minimálním poloměrem otáčení r_{min} je složena z né více, jak 3 kruhových křivek minimálního poloměru r_{min} a rovných úseček.

Kapitola 3

Plánovací algoritmy

V této části textu se seznámíme s populárními přístupy řešení problému plánování pohybu.

3.1 Probabilistic road map

V roce 1996 *Kavraki et al.* představili svou metodu pro plánování pohybu vhodnou především pro roboty s vysokým stupněm volnosti zvanou *probabilistic road map* (PRM) [14]. Patří do rodiny vzorkovacích algoritmů. Metoda má dvě fáze — fáze *učení* a fáze *dotazování*. Takhle je alespoň nazývá *Kavraki* ve své práci, ovšem zjednodušeně řečeno se jedná pouze o konstrukci grafu a následné hledání nejkratší cesty v tomto grafu.

Ve fázi učení se vytvoří mapa (graf) generováním náhodných konfigurací. Každá z těchto konfigurací se pak spojí buďto s k nejbližšími sousedními konfiguracemi nebo se všemi konfiguracemi v dané vzdálenosti jednoduchým a rychlým *lokálním plánovačem*. Vygenerovanou mapu pak lze použít pro dotazování na plán pohybu robota. Konstrukce PRM je nastíněna v algoritmu 1.

V algoritmech 1 a 2 je Δ zobrazení $\mathcal{C}_{free} \times \mathcal{C}_{free} \to \{0, 1\}$, které nám říká, zdali lokální plánovač je schopen najít proveditelnou cestu mezi 2 konfiguracemi dané jako argument a *dist* je metrika určující vzdálenost mezi dvěma konfiguracemi, např. Eukleidovská metrika

Výsledkem algoritmu 1 je neorientovaný graf, kde uzly reprezentují konfigurace robota a hrany reprezentují změnu konfigurace pohybem robota. Hledání plánu pohybu pak vypadá následovně: Pro jakoukoliv nekolizní počáteční a koncovou konfiguraci q_{init} a q_{goal} se najdou jejich nejbližší sousedé, ke kterým zároveň existuje proveditelná cesta (potencionální hrana v grafu). Tato hrana se přidá do mapy a pak se prohledávacím algoritmem jako je $A*$ nebo *Dijkstra* najde nejkratší cesta z q_{init} do q_{goal} pokud existuje. Tím dostaneme náš pohybový plán nebo chybu, pokud taková cesta nad naší vygenerovanou mapou

Algoritmus 1: Probabilistic Road Map — konstrukce

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
while $|V| < n$ **do**
  repeat
    $q \leftarrow$ náhodná konfigurace v Q
  until q *je bezkolizní*
  $V \leftarrow V \cup \{q\}$
for $\forall q \in V$ **do**
  $N_q \leftarrow k$ nejbližších sousedů q vybrané z V podle *dist*
  for $\forall q' \in N_q$ **do**
    if $(q, q') \notin E$ ***and*** $\Delta(q, q')$ **then**
      $E \leftarrow E \cup \{(q, q')\}$

Algoritmus 2: Probabilistic Road Map — dotazování

$N_{q_{init}} \leftarrow k$ nejbližších sousedů q_{init} z V podle *dist*
$N_{q_{goal}} \leftarrow k$ nejbližších sousedů q_{goal} z V podle *dist*
$V \leftarrow \{q_{init}\} \cup \{q_{goal}\} \cup V$
$q' \leftarrow$ nejbližší soused q_{init} z $N_{q_{init}}$
repeat
  if $\Delta(q_{init}, q')$ **then**
    $E \leftarrow (q_{init}, q') \cup E$
  else
    $q' \leftarrow$ následující nejbližší soused q_{init} z $N_{q_{init}}$
until *spojení bylo úspěšné* ***or*** $|N_{q_{init}}| = 0$
$q' \leftarrow$ nejbližší soused q_{goal} z $N_{q_{goal}}$
repeat
  if $\Delta(q_{goal}, q')$ **then**
    $E \leftarrow (q_{goal}, q') \cup E$
  else
    $q' \leftarrow$ následující nejbližší soused q_{goal} z $N_{q_{goal}}$
until *spojení bylo úspěšné* ***or*** $\left|N_{q_{goal}}\right| = 0$
$P \leftarrow$ nejkratší cesta(q_{init}, q_{goal}, G) `/* A*, Dijkstra, atd. */`
if $|P| > 0$ **then**
  return P
else
  return *chyba*

neexistuje. Algoritmus pro dotazování zapsaný v pseudokódu najdeme v algoritmu 2.

„Vytváření náhodných konfigurací: Uzly R by měli tvořit spíše jednotný náhodný výběr $\mathcal{C}_f$." [14, s.569, vlastní překlad], kde $\mathcal{C}_f$ je $\mathcal{C}_{free}$ a R je vygenerovaná mapa. Toto tvrzení nepřímo vyvrací v [15], kde využívají kvazináhodného vzorkování $\mathcal{C}$ pro generování náhodných konfigurací pro klasické PRM [14] i pro variantu Lazy-PRM [16] s lepšími výsledky v obou případech.

3.2 Rapidly-exploring Random Tree

Rapidly-exploring Random Tree (RRT) [17] je metoda podobná PRM v tom smyslu, že se pro její konstrukci také generují náhodné konfigurace, kterými se snažíme zmapovat $\mathcal{C}$. Liší se ovšem ve způsobu, jakým se tyto vygenerované konfigurace spojují. Zatímco v případě PRM to bylo k nejbližších sousedů, u RRT je to pouze ten nejbližší. Navíc oproti PRM, které je *multi-query*, je RRT *single-query*. To znamená, že pro každý dotaz na plán pohybu se musí vygenerovat nový strom (graf). U PRM se jednou vygenerovala mapa a ta se pak dala znovu použít pro následující dotazy. Výhodou RRT je jeho přímočaré rozšíření pro neholonomní roboty a kinodynamické plánování.

Metoda funguje následovně: Začneme s grafem (stromem), který obsahuje pouze počáteční stav. Poté n-krát vygenerujeme náhodnou konfiguraci q_{rand}, vezmeme si nejbližší uzel ze stromu q_{near} a vygenerujeme nový uzel „posunutý" o $stepsize$ směrem ke q_{rand}. Ten přidáme do stromu, pouze pokud je spojením mezi těmito dvěma konfiguracemi možné, tj. splňuje všechny kinodynamické podmínky zadané uživatelem a prochází pouze v C_{free}. Algoritmus zapsaný v pseudokódu najdeme v algoritmu 3.

Algoritmus 3: Rapidly-exploring random tree — konstrukce

$V \leftarrow \{q_0\}$
$E \leftarrow \emptyset$
for $i \in \{1, \ldots, n\}$ **do**
  $q_{rand} \leftarrow$ náhodně vybraná nekolizní konfigurace
  $q_{near} \leftarrow min(\{\forall q_{iter} \in V \ |\Delta(q_{rand}, q_{iter})\})$
  $q_{new} \leftarrow q_{near}$ posunutá o $stepsize$ směrem k q_{rand}
  if q_{new} *je bezkolizní* **then**
    $V \leftarrow V \ \cup \{q_{new}\}$
    $E \leftarrow E \cup \{(q_{near}, q_{new})\}$

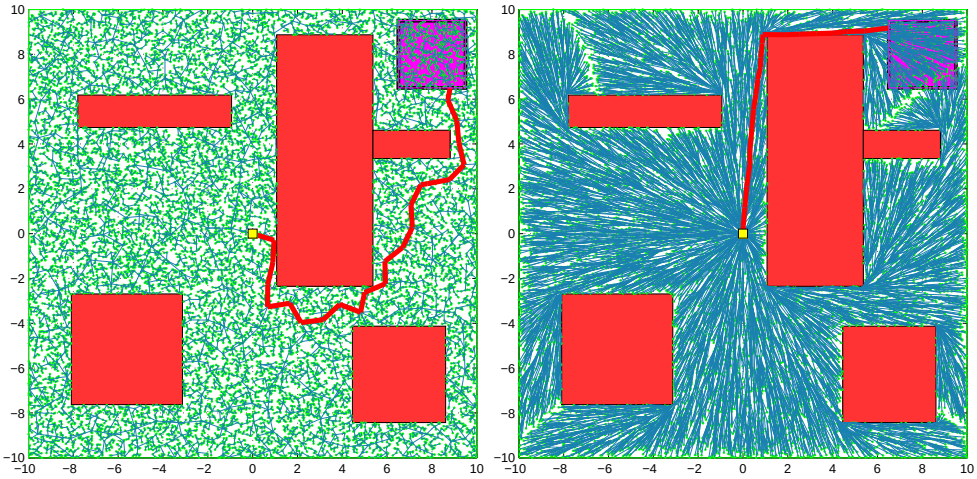
LaValle zmiňuje několik pěkných vlastností RRT: „1) Expanze RRT je silně orientovaná směrem k neprozkoumaným částem stavového prostoru; 2) distribuce hran v RRT se přibližuje k distribuci vzorkování, což vede ke konzistentnímu chování; 3) RRT je pravděpodobnostně kompletní za velice obecných

Algoritmus	Pravděpodobnostní úplnost	Asymptoticky optimální	Monotónní konvergence	Časová složitost		Paměťová složitost
				Výpočet	Dotaz	
PRM	Ano	Ne	Ano	$O(n \log n)$	$O(n \log n)$	$O(n)$
RRT	Ano	Ne	Ano	$O(n \log n)$	$O(n)$	$O(n)$
RRG	Ano	Ano	Ano	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
RRT*	Ano	Ano	Ano	$O(n \log n)$	$O(n)$	$O(n)$

Tabulka 3.1: Porovnání PRM, RRT, RRT*, Zdroj: [19]

podmínek; 4) RRT algoritmus je relativně jednoduchý, což usnadňuje analýzu výkonnosti (tohle je také preferovaná vlastnost PRM); 5) RRT zůstává vždy propojen, i přestože je počet hran minimální; 6) RRT může být považován jako modul plánování pohybu, který můžeme adaptovat a začlenit do široké škály plánovacích systémů; 7) celé plánovací algoritmy mohou být sestrojeny bez nutnosti přepínání systému mezi dvěmi předepsanými stavy, což velmi rozšiřuje aplikovatelnost RRT“ [17, str.2, vlastní překlad]

Díky populárnosti se RRT dočkalo několika rozšíření a variant. Jmenovitě například *Rapidly-exploring Random Graph* (RRG) a jeho stromová varianta RRT* [18], které konvergují k optimálnímu řešení.



Obrázek 3.1: Srovnání RRT(nalevo) a RRT*(vpravo). Oba algoritmy použili stejnou sekvenci 20000 vzorků. Cena nejlepší cesty u RRT byla 21,02 a 14,51 u RRT*; Zdroj: [18]

3.3 Artificial Potential Field

Poslední metodu plánování pohybu, kterou si představíme, je *Artificial Potential Field* (APF). Metodu poprvé představil *O. Khatib* v [20]. Je založena

na jednoduché myšlence — cílem plánování pohybu je dostat se z jedné konfigurace do druhé bez toho, aniž bychom přitom prošli skrz nevalidní konfiguraci, tj. vyhnout se překážkám. Můžeme se tedy dívat na cílovou konfiguraci, jako něco dobrého, co nás přitahuje a čeho chceme dosáhnout a na překážky, potažmo kolizní konfigurace, naopak jako něco odpudivého, čemu se chceme vyhnout. Analogicky z fyziky se můžeme dívat na počáteční konfiguraci, jako pozitivně nabitou částici, která hledá energetické minimum, tj. cílovou konfiguraci. Překážky jsou pak také pozitivně nabité oblasti, kterým se naše částice snaží vyhnout.

Zapsáno formálněji, naši částici budou ovlivňovat dva potenciály — kladný U_{att} a záporný U_{rep}. U je zobrazení určující potenciál pro jakoukoliv konfiguraci $U : \mathcal{C} \to \mathbb{R}$. Finální pole potenciálů tedy bude kombinací kladného a záporného potenciálu:

$$\forall q \in \mathcal{C} : U(q) = U_{att}(q) + U_{rep}(q)$$

Ukázku konstrukce U můžeme najít na obrázku 3.3. Vhodný kandidát pro kladný potenciál U_{att} by mohla být například vzdálenost mezi současnou konfigurací q a cílovou konfigurací q_{goal}:

$$U_{att}(q) = \zeta\rho(q, q_{goal})$$

nebo jeho kvadratická alternativa [21]:

$$U_{att}(q) = \frac{1}{2}\xi\rho^2(q, q_{goal})$$

kde parametry ζ a ξ jsou kladné škálovací konstanty a ρ je vzdálenost mezi dvěma konfiguracemi. Záporný potenciál by mohl vypadat následovně:

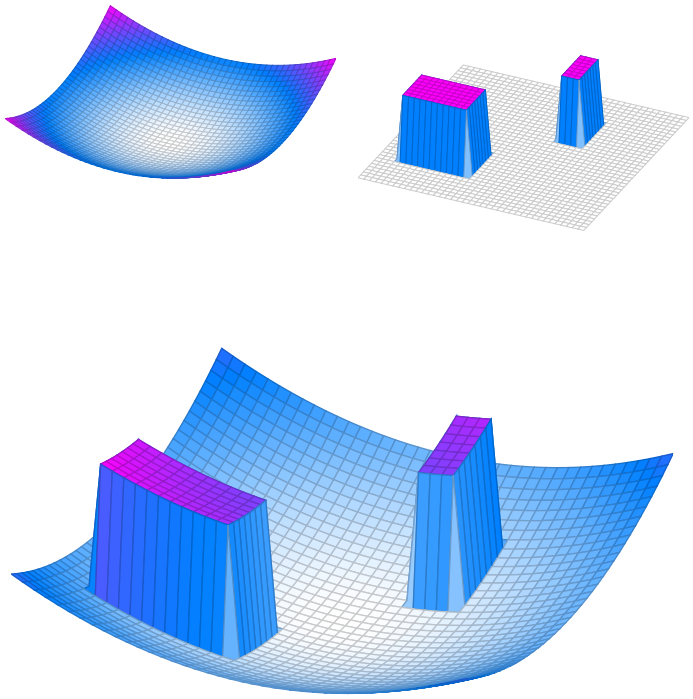
$$U_{req}(q) = \begin{cases} \frac{1}{2}\eta(\frac{1}{\rho_o(q)} - \frac{1}{\sigma_o})^2\rho^2(q, q_{goal}) & \rho(q) \leq \sigma_o \\ 0 & \rho(q) > \sigma_o \end{cases}$$

kde η je znovu kladná škálovací konstanta, σ_o je kladná konstanta vymezující vzdálenost vlivu překážek a $\rho_o(q)$ je vzdálenost konfigurace q od nejbližší kolizní konfigurace. Z pole potenciálů sestrojíme pole sil, které budou působit na robota:

$$\begin{aligned} \vec{F}(q) &= -\vec{\nabla}U(q) \\ &= \vec{F}_{att}(q) + \vec{F}_{rep}(q) \end{aligned}$$

kde přitažlivá a odpudivá síla jsou:

$$\begin{aligned} \vec{F}_{att}(q) &= -\vec{\nabla}U_{att}(q) \\ &= -\vec{\nabla}\frac{1}{2}\xi\rho^2(q, q_{goal}) \\ \vec{F}_{rep}(q) &= -\vec{\nabla}U_{rep}(q) \\ &= -\vec{\nabla}\frac{1}{2}\eta(\frac{1}{\rho_o(q)} - \frac{1}{\sigma_o})^2\rho^2(q, q_{goal}) \end{aligned}$$



Obrázek 3.2: Složky U_{att}, U_{rep} a konstrukce U

Největší problém spojený s APF je problém *lokálních minim*. Může nastat situace, kdy se přitažlivé síly rovnají těm odpudivým a výsledný vektor pohybu je tedy roven nulovému vektoru. Plán se tak zasekne v lokálním minimu a nikdy nedosáhne cílové konfigurace. Tímto problémem se zabývají v [22], kde se snaží najít praktickou cestu v poli potenciálů pomocí gradientního sestupu (PGDA) a přidáváním odpudivého potenciálu v případě blokujících konfigurací. Když PGDA dosáhne globálního minima, sestrojí se nové pole potenciálů, které má svoje globální minimum umístěné v cílové konfiguraci. Pro rozhodnutí dosažitelné trajektorie se nakonec udělá ještě druhá iterace PGDA.

Kapitola 4

Problém křižovatky

4.1 Multi-agentní plánování

Doposud jsme se dívali na problém plánování pohybu jednoho robota ve spojitém prostoru, který je sám o sobě dost složitý. Problém křižovatky je ovšem ještě o trochu složitější, jelikož už si nevystačíme pouze se statickými překážkami, ale musíme brát v potaz i pohyby ostatních robotů, kteří se také snaží projet křižovatkou. Předefinujme si tedy problém plánovaní pohybu představený v 2.

Definice 3 (Piano mover's problem [4])

1. Svět $\mathcal{W}$ and oblast překážek $\mathcal{O}$ je stejná jako v 2.

2. Mějme m robotů, $\mathcal{A}^1, \ldots, A^m$, kde každý může být sestaveno z jednoho či více těl.

3. Každý robot $\mathcal{A}^i$, kde i jde od 1 do m, má přiřazený konfigurační prostor $\mathcal{C}^i$.

4. Stavový prostor X je definovaný jako kartézský součin

$$X = \mathcal{C}^1 \times \mathcal{C}^2 \times \cdots \times \mathcal{C}^m$$

5. Oblast překážek v X je

$$X_{obs} = \left(\bigcup_{i=1}^{m} X_{obs}^i \right) \bigcup \left(\bigcup_{i \neq j} X_{obs}^{ij} \right)$$

   kde X_{obs}^i jsou kolize robotů s překážkami a X_{obs}^{ij} jsou kolize robotů s roboty.

6. Stav $x_I \in X_{free}$ je počáteční stav, kde $x_I = (q_I^1, \ldots, q_I^m)$

7. Stav $x_G \in X_{free}$ je koncový stav, kde $x_G = (q_G^1, \ldots, q_G^m)$

8. Cílem je vypočítat spojitou křivku $\tau : [0, 1] \to X_{free}$ takovou, kde $\tau(0) = x_I$ a $\tau(1) = x_G$

Na první pohled se může zdát, že je tato definice zbytečná, protože bychom mohli docílit stejného výsledku i s definicí 2. Stačilo by se dívat na jednotlivé roboty jako jednoho velkého robota s velkým stupněm volnosti. Ovšem největším problémem je, že dimenze X roste lineárně s počtem robotů a bude tak neúnosně velká. Máme-li například 10 tuhých těles v 3D prostoru, tj. každý robot má konfigurační prostor $\mathcal{C}^i = SE(3)$, pak dimenze X je $6 \cdot 10 = 60$. „Kompletní algoritmy vyžadují čas, který je alespoň exponenciální, což je dělá nepravděpodobnými kandidáty pro takové problémy. Vzorkovací algoritmy se budou pravděpodobněji lépe škálovat v případě vícero robotů, ale dimenze X stále může být příliš velká.“ [4, vlastní překlad]

Jedním ze způsobů, jak vyřešit problém vysoké dimenze X, je použití *odděleného plánování*. Takové plánovače nejdřív navrhnou plán pohybu pro každého robota zvlášť, aniž by brali v potaz ostatní roboty. Poté se až začnou zvažovat kolize mezi roboty. Nastane-li problém, možnosti jednotlivých robotů jsou v tu chvíli již omezeny původně navrženými plány a není tedy možné vrátit se do stavu, kde by problém nenastal. V takovém případě bohužel ztrácíme úplnost řešení. Avšak tyto metody jsou velice praktické a v mnoha případech se dá úplnost obnovit.

Příklad odděleného plánování může být třeba *prioritizované plánování* [23, 24]. Funguje následovně: Každý robot dostane přiřazenou prioritu a roboti se vybírají podle snižující priority. Vybranému robotu se naplánuje trajektorie, která se vyhýbá statickým překážkám i trajektoriím robotů vybraných před ním. Předchozí roboti tak fungují jakožto dynamické překážky.

Fixed-path coordination [25] je dalším příkladem odděleného plánování. Každý robot je omezen následovat pouze jednu určitou trasu. Tu můžeme vygenerovat jakýmkoliv plánovačem z kapitoly 3. Pro m robotů vytvoříme m-dimenzionální stavový prostor zvaný *koordinační prostor*. V tomto prostoru se plánuje pořadí průjezdů jednotlivých robotů. Čas je v tomto prostoru reprezentován pouze implicitně. Algoritmus musí vypočítat trasu v koordinačním prostoru, z které se pak dají získat explicitní časy. Pro hledání trasy v koordinačním prostoru můžeme znovu využít standardních plánovacích algoritmů představených dříve. Fixed-path coordination je možné rozšířit, aby se roboti namísto jedné cesty mohli pohybovat po mapě stejně jako u PRM. Taková metoda se nazývá *fixed-roadmap coordination*. Její výhodou je, že výsledný koordinační prostor má stále dimenzi m, tedy jedna dimenze za každého robota a zároveň se blíží více úplnému algoritmu.

4.2 Současná řešení

V této části práce se podíváme, jak se k problému plánovaní pohybu na křižovatce staví současná literatura a pokusíme se na základě představených metrik je nějakým způsobem porovnat. *Shravan et al.* ve své práci [26] navrhují následující metriky:

1. *Škálovatelnost* — Plánovač by měl být schopen zvládnout velký objem aut stejně nebo podobně efektivně, jako při menším provozu.

2. *Čas strávený na křižovatce* — AV by měli trávit co nejméně času v křižovatce, tedy v nejlepším případě pouze čas potřebný pro průjezd křižovatkou s nulovým provozem.

3. *Hladkost* — Jak moc je jízda v daném vozidle pohodlná (respektive nepohodlná) pro pasažéry. Ve své podstatě pouze porovnáváme vyšší derivace pozice v čase, jako jsou zrychlení a ryv.

4. *Využití prostoru křižovatky* — Kolik celkového prostoru křižovatky je nevyužito při nejvyšším vytížení.

5. *Heterogenita* — Schopnost plánovače vytvářet trajektorii pro různorodá auta s různými jízdními a fyzikálními vlastnostmi.

6. *Robustnost* — Plánovač nemusí od jednotlivých AV dostávat přesná data. Robustnost je schopnost a způsob, jak se plánovač s touto nejistotou vypořádá. S přibývajícím časem se zvětšuje i nejistota v datech. Jeden způsob, jak se s tímto problémem vypořádat je mít více konzervativní přístup při plánování trajektorií. To vede k delšímu času stráveným na křižovatce. Tedy klíčové je najít rovnováhu mezi těmito metrikami.

7. *Všestrannost* — Aplikovatelnost plánovacího algoritmu na různé typy křižovatek. Většina plánovačů se vyvíjí pro specifické druhy křižovatek. Je tedy důležité se dívat a měřit, jak si tyto plánovače vedou v neznámém prostředí.

Shravan et al. dále navrhují 2 metriky, které se namísto generování trajektorií zabývají sdílení dat mezi AV:

1. *Potřebná data* — Některým plánovačům stačí vědět pozici a směr auta pro naplánování trajektorie. Jiné zase potřebují navíc znát rychlost a jiné kinematické vlastnosti auta. S vyššími požadavky plánovače na potřebná data se samozřejmě zvedá i zátěž na síť, přes kterou plánovač komunikuje s AV. To vede ke zvýšené pravděpodobnosti ztráty dat a zvýšené odezvě.

2. *Ochrana osobních údajů* — Velkým tématem posledního desetiletí je otázka ochrany osobních údajů, viz legislativa Evropské unie GDPR. Je

tedy nutné porovnávat jednotlivé algoritmy z hlediska jejich způsobu sdílení a ochrany osobních dat.

Velice populárním přístupem řešení problému křižovatky jsou tzv. *slot based systémy* neboli *rezervační systémy*. Tyto plánovače si rozdělí křižovatku na jednotlivé sektory, které potom přiřazují přijíždějícím AV nebo si je AV sami rezervují. Jedním z takových systémů je i [27] představený *Dresnerem* v roce 2008. V jeho přístupu vozidlo přijíždějící na křižovatku vyšle signál plánovači, který mu poté zašle odpověď s rychlostí, trajektorií a pruhem, do kterého se má zařadit. Samotný plánovač si rozdělí křižovatku do mřížky a detekování kolizí se tak omezí pouze na hrany mřížky. S vylepšenou verzí přišli *Levin* a *Rey* v [28], kde se zbavili nutnosti AV dotazovat se křižovatky na alokovaný prostor. Místo toho využili lineárního celočíselného programování pro optimalizování rezervací. V [29] navrhují sadu pravidel založených na čínských silničních pravidlech, které udávají pořadí průjezdu jednotlivých AV. Pokud přijíždějící AV potřebuje pustit jiné AV, algoritmus mu navrhne vhodnou míru brždění tak, aby nedošlo ke kolizi. Na rozdíl od předchozích prací, tato metoda nevyužívá centrálního plánování, ale místo toho komunikace probíhá pouze mezi jednotlivými AV, tzv. vehicle-to-vehicle komunikace (V2V). V [30] *Fayazi et al.* využili pro plánování průjezdu smíšeného lineárního celočíselného programování (MILP) s obousměrnou komunikací mezi AV. Po přijetí zprávy s žádostí o průjezd a stavu vozidla od AV plánovač spočítá ideální příjezdový plán snažící se minimalizovat celkový čas přístupu AV ke křižovatce. V [31] navrhli distribuovaný plánovač pro případ, kdy V2V komunikace je nespolehlivá a má časté výpadky. V [32] se snaží zbavit nutnosti měnit pruh v případě odbočování a navrhují plánovač, který umožní jet z jakéhokoliv pruhu do kteréhokoliv jiného. Pro zjednodušení problému se AV pohybují po předdefinovaných cestách a křižovatkou projíždí konstantní rychlostí. V [33] *Levin et al.* navrhují metodu pro optimalizaci rezervačních algoritmů, které využívají trajektorií založenou na diskretizaci prostoru křižovatky. Konkrétně se zaměřují na to, jak vyřešit problémová místa konfliktu, kde by mohlo dojít ke kolizím. Navrhují také celočíselný program, který řeší právě tato problémová místa. V [34] navrhují rezervační systém využívající teorie hromadné obsluhy. Jejich výsledky ukazují, že s použitím jejich metody se kapacita křižovatky až zdvojnásobí oproti konvenčním křižovatkám se světelnými signály.

Výše zmíněné metody využívají strategie pro vyhnutí kolizí založené na komunikaci mezi AV, které však neberou v potaz, jak se může trajektorie AV v průběhu času měnit. K tomuto účelu můžou lépe posloužit decentralizované nebo smíšené systémy.

V [35] se zaměřují na více-účelové evoluční algoritmy pro vývoj fuzzy systémů [36] vhodných především pro spravování rychlosti AV. Hlavní zvláštností této práce je spojení AV s manuálně řízenými auty. Cesta AV přijíždějící ke křižovatce je totiž zkřížená pruhem manuálně řízených aut a je tedy povinností AV aby se projíždějícím autům vyhnul a na vzniklé situace nějak

zareagoval. Metoda je založena na velmi známé metodě určené pro řešení víceúčelových úloh SPEA2 [37].

Zbavovat se světelných signálů není vždy nutností. Například v [38] šli cestou teorie her a formulovali křižovatku jakožto nekooperativní hru mezi různými hráči. Na základě žádostí o průjezd křižovatkou od různých hráčů pak akorát mění světla na semaforu. Výhodou tohoto přístupu je fakt, že křižovatku pak můžou využívat AV i manuálně řízená auta.

V [39] řeší problém křižovatky vyřešením subproblému optimálního řízení pro všechny permutace sekvencí projíždění. Ve své práci ukazují, že pro vybranou sekvenci průjezdu může být jeho podproblém optimálního řízení formulován jako konvexní program. Navrhovaná metoda konvertuje problém z původní časové domény na prostorovou. Na této metodě založili svoji práci *Riegger et al.* v [40], kde problém formulovali jako Model Predictive Control (MPC) úlohu řešenou kvadratickým programem tak, aby ho bylo možné vyřešit v reálném čase. Metodu implementovali v CarMaker. Metoda také získává časové odstupy mezi AV tak, aby se optimalizační problém nestal neproveditelným v případě chybných dat.

V [41] využívají poměr rychlostí AV k vyhnutí kolizí. Účelovou funkci definovali jako součet výchylek současných rychlostí. Minkowského suma je použita pro reprezentaci vozidel jako body. V [42] navrhují nekonvexní distribuovaný MPC model využívající paralelní optimalizaci. Distribuce je dosažena tím, že se každému AV dá jeho lokální cíl a globální omezení čímž se původně centralizovaný optimalizační problém změní v kvadraticky omezený kvadratický program. Aby se vyrovnali s nekonvexními omezeními, aplikují semidefinitivní programovou relaxaci k nalezení proveditelných řešení. V [43] formulují kombinatorický optimalizační problém který rozdělili na dva podproblémy. Jeden je najít centrálně optimální koordinaci vozidel a druhý vyřešit pro všechny AV lokální řídící problém závislí na systémových omezení a časových úsecích specifických pro každé AV.

Některé práce jako například [28] využívají techniku *vzdalujícího se horizontu* [44] kdy pro vysokou výpočetní náročnost není nutné ani vhodné hledat úplné řešení. V takovém případě hledáme řešení pouze několik časových iterací dopředu až po námi určený časový horizont. Ten s každým dalším krokem posuneme o fixní vzdálenost a znovu hledáme řešení.

Část II

Vlastní metoda

Kapitola 5

Návrh algoritmu

V praktické části práce navrhneme vlastní metodu pro řešení problému navigace více AV na křižovatce a otestujeme v on-line scénáři, kdy v pravidelných intervalech přijíždí AV ke křižovatce a naším úkolem je vytvořit jim plán pohybu. Nakonec porovnáme naše řešení pomocí metrik představených v části 4.2.

5.1 Požadavky

Při vytváření vlastního řešení jsem bral v potaz hlavně využitelnost v praxi. Řídil jsem se tedy především následujícími požadavky:

- *Bezpečnost* – Plánovač by měl především zajistit bezpečný průjezd pro všechny účastníky provozu a až poté zvažovat jakékoliv jiné metriky. Bezpečnost je tedy na prvním místě. Model, který navrhuje kolizní průjezd křižovatkou je v praxi zbytečný.
- *Existence řešení* – Plánovač by měl být schopen najít pokaždé řešení bez ohledu na počet či konfiguraci přijíždějících AV.
- *Rychlost* – Plánovač by měl být schopen použit v praxi, kde je rychlost nalezení řešení zásadní. To znamená, že výpočet plánu by měl být v řádu milisekund.
- *Plynulost jízdy* – Tento požadavek jde ruku v ruce s hladkostí křivek z části 4.2. Cestujícím by se nemělo dělat špatně při jízdě v AV a tedy bychom se měli snažit limitovat boční zrychlení vozu.

Jedním ze způsobů, jak zaručit existenci řešení, je vpouštět AV do prostoru křižovatky po jednom a zarezervovat jim celý prostor křižovatky jenom pro ně. Tím sice zaručíme existenci řešení a vyhneme se jakýmkoliv kolizím, avšak takové řešení nebude zvlášť dobré. Využitelnost křižovatky, metrika z 4.2,

bude velmi slabá a po čase se nám před křižovatkou začne vytvářet kolona i při sebemenším provozu. Určitě dokážeme vymyslet i něco lepšího. Nicméně si můžeme něco vzít z tohoto řešení. To je nevpouštět AV do křižovatky, pokud jim nejsme schopni zaručit bezpečný průjezd. Tím zajistíme bezpečnost i existenci řešení.

Co se plynulosti jízdy týče, tak ta závisí na trajektorii vozu projíždějící křižovatkou. Zvažoval jsem použití RRT pro generování proveditelných křivek, ovšem ty mnohdy mohou vypadat nepřirozeně a klikatě a mohly by tedy vyvolat nevolnost u pasažérů. Rozhodl jsem se tedy pro řešení pomocí ručně namodelované *stavové mřížky* neboli *state lattice* inspirované prací [45]. Toto názvosloví může být trošku zavádějící, jelikož se úplně nejedná o stavovou mřížku jako takovou. Sdílí sice podobné vlastnosti, ovšem její vytvoření probíhá trošku jinak. Nemodeluji totiž mřížku pro každé přijíždějící auto zvlášť s ohledem na jeho kinodynamické vlastnosti, ale naopak nejdřív modeluji stavovou mřížku pro křižovatku a jakékoliv přijíždějící auto musí splňovat takové vlastnosti, které mu umožní průjezd křižovatkou. To se na první pohled může zdát jako velké omezení, ale ve výsledku to pouze znamená, že jakékoliv přijíždějící auto musí mít dostatečný poloměr zatáčení. Auta se takhle efektivně chovají jako vlaky na kolejích.

5.2 Metoda

Hlavní princip mé metody by se dal rozdělit do následujících čtyř kroků:

1. Konstrukce stavové mřížky pro křižovatku pomocí Bézierových křivek.
2. Vytvoření koordinačního prostoru pro přijíždějící auta v bezprostřední blízkosti křižovatky.
3. Diskretizace koordinačního prostoru.
4. Prohledání koordinačního prostoru pomocí prohledávacích algoritmů jako je třeba A*.

Tato metoda je přímo inspirovaná částí *La Valleho* práce: „Předpokládejme, že každý robot $\mathcal{A}^i$ je omezen následováním cesty $\tau_i : [0, 1] \to \mathcal{C}^i_{free}$, která může být vypočítána běžnými technikami pro plánování pohybu. Pro m robotů, m-dimenzionální stavový prostor nazývaný *koordinační prostor* je definován, který plánuje pohyby robotů podél jejich cest tak aby nekolidovali ... Pro velice náročné koordinační problémy mohou vzorkovací řešení nést praktické výsledky. Možná jedno z nejjednodušších řešení je proložit X mřížkou a adaptovat klasické prohledávací algoritmy ...“ [4, sekce 7.2.2, vlastní překlad]

5.3 Bézierovy křivky

Pro konstrukci stavové mřížky jsem využil Bézierových křivek. Bézierovy křivky jsou parametrické křivky hojně využívané především v grafice [46]. Například *Pierre Bézier*, po kterém jsou tyto křivky pojmenovány, je použil pro modelování karosérie aut Renault v 60. letech.

Mějme dva různé body P_0 a P_1. Lineární Bézierova křivka je potom jednoduchá úsečka mezi těmito dvěma body:

$$B(t) = P_0 + t(P_1 - P_0) = (1-t)P_0 + tP_1, \quad 0 \leq t \leq 1$$

Toto odpovídá lineární interpolaci. Pro křivku mezi třemi body P_0, P_1, P_2 se používá kvadratická Bézierova křivka a ta vypadá následovně:

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2, \quad 0 \leq t \leq 1$$

Kvadratickou Bézierovou křivku si můžeme představit jako lineární interpolaci dvou lineárních interpolací. Na tomto principu je postaven algoritmus *De Casteljau*, který rekurzivně vyhodnocuje lineární interpolace. Ukázka na kubické Bézierove křivce pro 4 body:

```
A = lerp(P0, P1, t)
B = lerp(P1, P2, t)
C = lerp(P2, P3, t)
D = lerp(A, B, t)
E = lerp(B, C, t)
F = lerp(D, E, t)
```

Obecně můžeme pro libovolný počet bodů vyjádřit Bézierovu křivku následovně:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i$$

5.4 A*

Pro prohledání koordinačního prostoru byla použita variace A*. Poprvé představen *Hartem* v [47], A* je algoritmus podobný *Dijkstrovu algoritmu* sloužící pro hledání optimálních cest v kladně ohodnocených grafech. Na rozdíl od Dijkstrova algoritmu využívá navíc heuristické funkce, podle které se určuje pořadí prohledávaných uzlů společně s cenou. Níže je uveden algoritmus A* zapsaný v pseudokódu.

Algoritmus 4: A* [48]

```
open ← init_priority_queue()
dist ← init_table()
prev ← init_table()
enqueue(open, I, h(I))
dist[I] ← 0
while ¬empty(open) do
    x ← dequeue(open)
    if x ∈ G then
        return reconstruct_path(prev, x)
    for ∀y ∈ neighbors(x) \closed do
        d′ ← dist[x] + c((x, y))
        if y ∉ open or dist[y] > d′ then
            dist[y] ← d′
            prev[y] ← x
            if y ∉ open then
                enqueue(open, y, d′ + h(y))
            else
                update_key(open, y, d′ + h(y))
    closed ← closed ∪ {x}
```

Kapitola 6

Implementace

6.1 Použité technologie

První prototyp projektu vznikal v herním enginu Unity. Ovšem jelikož tento text vznikal v LaTexu v linuxovém prostředí, zároveň jsem neslyšel dobré recenze o Unity v linuxu a chtěl jsem mít prostředí sjednocené jak pro text, tak pro projekt, rozhodl jsem se, že se přikloním linuxu a použiji jinou technologii pro projekt. Druhý pokus o prototyp byl v jazyce Python s použitím knihovny Pygame. Python je znám pro svou schopnost rychlého prototypování. Navíc už jsem pár projektů v Pythonu psal, takže se to zdálo, jako logická volba. Jak je ale Python známý pro svou schopnost rychle v něm zkonstruovat prototyp, je také znám pro svou rychlost běhu. Python je totiž dynamický interpretovaný jazyk a je tedy velice pomalý. To se ukázalo jako zásadní slabina, když jsem napsal modul pro detekci kolizí. Ten se volá při každé iteraci prohledávání konfiguračního prostoru a musí tedy být rychlý. Byl jsem tedy nucen znovu změnit technologie. Moje třetí a poslední volba byl herní engine Godot a jeho Pythonu podobný jazyk GDScript. Byl to ideální kompromis mezi rychlosti psaní v Pythonu a rychlostí běhu Unity.

6.2 Křižovatka

Metodu jsem se rozhodl implementovat pro klasickou dvouproudou křižovatku tj. 2 pruhy příjezdové a 2 pruhy odjezdové pro každou světovou stranu. Toto rozhodnutí je založeno především na faktu, že se jedná o jednu z nejběžnějších křižovatek [49] a je tedy praktické hledat řešení právě pro tuto křižovatku. Zároveň se ve své podstatě jedná o nejhorší možný případ křižovatky, jelikož zde nejsou žádné dodatečné odbočovací pruhy či nájezdy a tedy počet míst, kde může dojít ke konfliktu, je tak maximální.

Pro konstrukci stavové mřížky byli použity Bézierovy křivky. Tady nám Godot usnadnil práci, jelikož má v sobě již zabudovaný objekt Path2D, pomocí jehož jsem mohl ručně navrhnout stavovou mřížku křižovatky. Z každého

příjezdového pruhu by se AV měl být schopen dostat do jakéhokoliv odjezdového pruhu. Nebereme-li v potaz otáčení na křižovatce, znamená to, že každému příjezdovému pruhu náleží 6 odjezdových. To je celkem 48 křivek modelující stavovou mřížku pro všech 8 příjezdových pruhů.

6.3 Model auta

Pro model auta jsem se rozhodl použít bicycle model představený v 2.4.3. Godot nám zde znovu usnadnil práci, jelikož má v sobě již zabudovaný kinematický model určený právě pro simulování jízdy automobilu. Úpravou pozice kinematického modelu vzhledem ke středu scény jsme schopni určit pozici středu zadní nápravy. Auto se musí v jakýkoliv moment nacházet v jednom z následujících stavů:

1. *IN QUEUE* – Toto je první stav, do kterého se auto dostane hned po vytvoření. Cílem auta nacházející se v tomto stavu je postupně se přibližovat k prostoru křižovatky a udržovat bezpečný odestup od auta jedoucí před ním.

2. *APPROACHING INTERSECTION* – Potom, co se auto dostane do blízkosti křižovatky a je zachyceno oblastí k tomu určenou, auto buď to zůstane ve stavu IN QUEUE pokud se před ním nachází nějaké neodbavené vozidla a nebo začne zpomalovat až dokud nezastaví před křižovatkou. Vzdálenost od křižovatky pro zahájení zpomalení můžeme snadno vypočítat pomocí

$$d = \left| v_i \cdot t + \frac{a \cdot t^2}{2} \right|$$

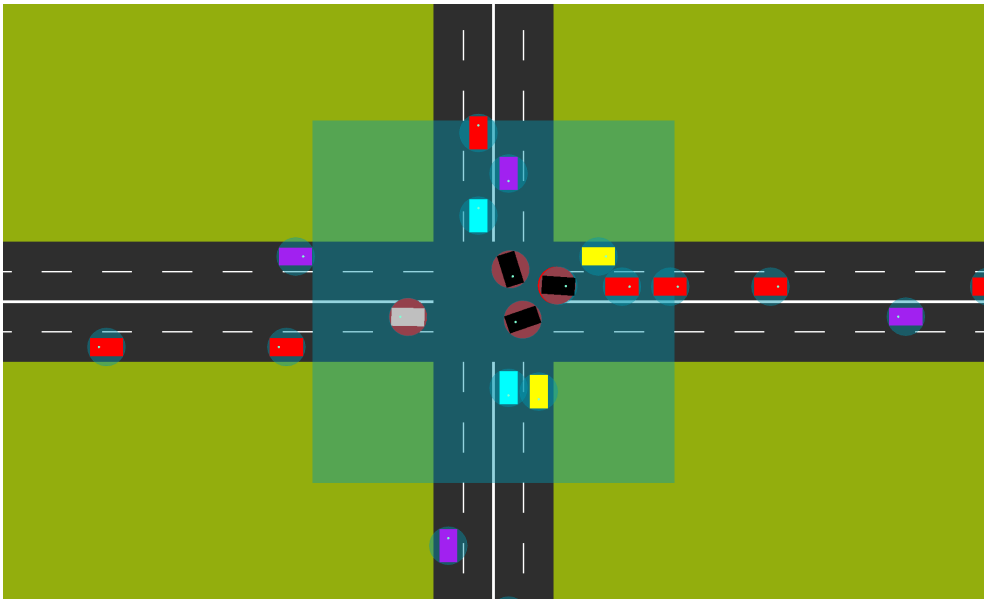
   kde t je čas potřebný ke změně rychlosti $t = \frac{(v_f - v_i)}{a}$, v_i počáteční rychlost, v_f koncová rychlost a a míra zrychlení. Jelikož v_f bude vždy 0, můžeme výraz zjednodušit na

$$d = \frac{v_i^2}{2a}$$

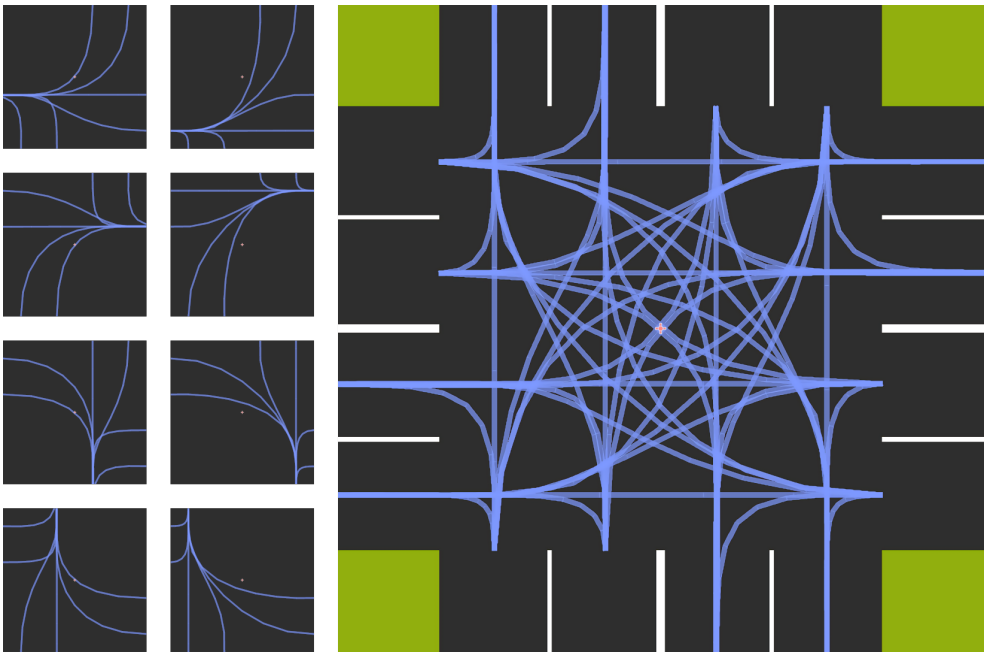
3. *WAITING FOR TIMING* – Auto čeká na správné načasování, kdy může vjet do křižovatky, které obdržel od plánovače.

4. *FOLLOWING PLAN* – Auto následuje plán, který obdržel od plánovače.

5. *FINISHED* – Auto dokončilo průjezd křižovatkou a už je pouze na něm, jak bude dál pokračovat. V mojí implementaci pokračuje ve svém pruhu dokud nevyjede z mapy.



Obrázek 6.1: Metoda v akci. Modře zvýrazněná oblast uprostřed je oblast zachycení. Auta jsou barevně rozlišena podle stavu, ve kterém se nachází



Obrázek 6.2: Stavová mřížka

6.4 Detekce kolizí

Detekce kolizí je důležitá součást každého plánovače. Především musí být velice rychlá, jelikož se volá v řádu tisíců až statisíců volání za jeden dotaz na plán.

V mém řešení jsem se nechal inspirovat prací *Zieglera* [50] a pro detekci kolizí jsem použil aproximaci pomocí disků. Výhodou této metody je, že na rozdíl od netriviálních útvarů jako jsou mnohoúhelníky, je detekce kolizí pro disky triviální. Vystačíme si zde pouze se 6 údaji. Jmenovitě poloha dvou disků x_1, y_1, x_1, y_2 a poloměry disků r_1, r_2. Výpočet kolize pak vypadá následovně:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} - r_1 - r_2 \leq 0$$

Godot má sice v sobě zabudované prvky pro detekci kolizí (i mnohoúhelníků), ovšem tyto prvky jdou použít pouze v reálném čase při běhu simulace a né v mezi-snímku tj. tam kde bude reálně probíhat veškeré plánování. Je to způsobeno tím, že Godot detekuje jakékoliv kolize až na konci snímku a né dřív. Byl jsem proto nucen si napsat vlastní modul pro detekci kolizí s použitím rovnice zmíněné výše. I přestože Godot nedetekuje kolize uprostřed snímku, pozice těchto kolizních prvků se aktualizuje okamžitě a pro modelování kolizních disků jsem tedy mohl použít již zabudovaný prvek *CollisionShape2D*.

6.5 Plánovač

Plánovač je motorem celého algoritmu. Je zodpovědný za naplánování individuálních profilů zrychlení pro přijíždějící AV. Tyto individuální plány vypadají jako vektor uspořádaných dvojic, kde každá dvojice reprezentuje akci a, kterou má auto provést a čas t, do kdy má tuto akci vykonávat.

$$P = ((a_0, t_0), (a_1, t_1), \ldots, (a_n, t_n))$$

Akce a je celé číslo z množiny $a \in \{-1, 0, 1\}$, kde auto i plánovač interpretují -1 jako příkaz „zpomal“, 0 jako „udržuj rychlost“ a 1 jako „zrychli“. Časy t jsou přirozená čísla odpovídající času od vjezdu do křižovatky.

Strukturálně si můžeme plánovač rozdělit na dvě fáze — fáze *zachycení* a fáze *plánování*. Ve fázi zachycení si plánovač vybere vhodnou podmnožinu přijíždějících aut vzhledem k současným požadavkům na rychlost běhu programu a optimalitu výsledku. Ve fázi plánování se vytvoří jednotlivé plány.

6.5.1 Fáze zachycení

Základním kamenem plánovače je *oblast zachycení*. To je oblast, kde pokud auto do ní vjede, tak tím oznamuje plánovači, že chce projet křižovatkou. V tuto chvíli je povinností plánovače, aby autu naplánovala trasu. Oblast zachycení můžeme vidět na obrázku 6.1 jako modrou oblast.

Po vjetí auta do oblasti zachycení se plánovač dotáže auta, kudy chce jet, jaká je jeho maximální rychlost, zrychlení a v jakou dobu je nejdřív schopen dojet ke křižovatce. Plánovač si zároveň uloží do paměti všechna auta, které požádali o naplánování trasy od poslední fáze plánování. Poté, co první auto z těchto uložených aut dojede ke křižovatce nebo je dosažen limit aut k naplánování, plánovač začne plánovat všem uloženým autům individuální plán. Plánovač se tak dostane do fáze plánování.

6.5.2 Fáze plánování

V této fázi má plánovač k dispozici auta, která požádali o průjezd křižovatkou a jejich kinematické vlastnosti jako je zrychlení, současná rychlost a maximální rychlost. Dále má k dispozici údaje o tom, kdy jednotlivá auta dorazí ke křižovatce. Na základě těchto údajů je potom schopen vytvořit jednotlivé plány.

Plánovač funguje na principu prohledávání koordinačního prostoru X. Ten je popsán v *LaValleho* práci: „Pro m robotů, koordinační prostor X je definován jako m-dimenzionální jednotková hyperkrychle $X = [0,1]^m$ ... i-tá souřadnice X reprezentuje doménu $S_i = [0,1]$ cesty τ_i. Nechť s_i značí bod v S_i (je to také i-tý komponent x). Stav $x \in X$ značí konfiguraci každého robota. Pro každého robota i, konfigurace $q^i \in C^i$ je dána jako $q^i = \tau_i(s_i)$. Ve stavu $(0,\ldots,0) \in X$ jsou všichni roboti ve svých počátečních konfiguracích $q_I^i = \tau_i(0)$ a ve stavu $(1,\ldots,1) \in X$ jsou všichni roboti v jejich cílových konfiguracích $q_G^i = \tau_i(1)$. Jakákoliv spojitá cesta $h : [0,1] \to X$, pro kterou $h(0) = (0,\ldots,0)$ a $h(1) = (1,\ldots,1)$, dostane roboty do jejich cílové konfigurace.“ [4, vlastní překlad] Bohužel takovýto koordinační prostor definuje pouze pozici a natočení každého robota. V mém řešení si potřebuji navíc ukládat rychlost. Koordinační prostor potom tedy vypadá jako m-tice dvojic pozice a rychlosti:

$$X = (s,v)^m \quad s \in [0,1], v \in [0, v_{max}]$$

Prohledávání začíná v počáteční konfiguraci $x = ((0,0)_1, (0,0)_2, \ldots, (0,0)_m)$. Cílových konfigurací existuje nekonečně mnoho, jelikož je nám jedno, jakou budou mít jednotlivá auta na konci rychlost. Prohledávání probíhá zkoušením různých profilů zrychlení $A = (a_1, a_2, \ldots, a_m)$ po nějakou dobu t, kde $a_i \in \{-1, 1\}$ je příkaz pro auto i, jestli má zrychlit či zpomalit. Auta tedy můžou v jeden moment buďto zrychlovat nebo zpomalovat. Tímto nám vzniká graf, kde uzly odpovídají jednotlivým konfiguracím a hrany profilům zrychlení. Vhodným zvolením t jsme pak schopni ovládat míru diskretizace koordinačního prostoru a tedy i velikost grafu. Dále bylo nutné naimplementovat omezení, aby rychlost jednotlivých aut zůstala v rozsahu $v_i \in [0, v_{max}]$. Tedy auto bude zrychlovat, potažmo zpomalovat, pouze pokud je to možné.

Prohledávají se nejdřív konfigurace s nejmenším součtem ceny a heuristické funkce. Jelikož v nejlepším případě by auta projíždějící křižovatkou

pouze zrychlili na svojí maximální rychlost a zůstali by tak až dokud by nedosáhli konce své cesty, konfigurace s nejmenší cenou jsou zde definovány jako nejmenší kumulativní počet instancí, kdy auta byla nucena zpomalit. K tomu je ještě připočtena heuristická funkce, která počítá vzdálenost současné konfigurace od cílové. Jelikož konfigurace nejblíž cílové konfiguraci jsou zároveň nejdál od počáteční, při implementaci jsem volil druhý způsob a pro každou konfiguraci jsem si ukládal počet kroků z počáteční konfigurace do cílové. Vyhnul jsem se tak explicitnímu počítání heuristické funkce jelikož stačilo seřadit konfigurace nejdřív podle nejnižší ceny a pak konfigurace se stejnou cenou podle vzdálenosti od počáteční konfigurace.

V potaz jsem taky musel brát fakt, že auta přijíždějící na křižovatku nebudou přijíždět ke křižovatce „pěkně“ a jelikož nemá smysl, aby první auto, co dojede ke křižovatce, muselo čekat na ostatní, počáteční konfigurace tak není úplně pravdivá, jelikož se některé auta ještě nenachází na svých počátečních pozicích. Z tohoto důvodu se ve fázi zachycení plánovač dotazoval aut na jejich odhadovanou dobu příjezdu aby tyto časování mohl vzít v potaz a neplánoval jim, jestli mají zrychlovat či zpomalovat ještě předtím, než budou vůbec schopni dorazit ke křižovatce. Analogicky stejné platí i pro cílovou konfiguraci. Jakmile auta dosáhnou konce cesty, nemusí čekat na ostatní, ale prostě pokračují dál v jízdě.

Po dosažení jedné z cílových konfigurací plánování končí. Dále už jen rekonstruujeme jednotlivé plány zpětným následováním grafu zpět do počáteční konfigurace. Nakonec plánovač odešle jednotlivé plány zpět autům a zároveň jim dodá nové časování, kdy by měli vjet do křižovatky, pokud musí počkat než projedou ostatní auta.

Kapitola 7

Výsledky

V této části práce porovnáme náš navržený plánovač se současnými řešeními a otestujeme náš algoritmus na několika experimentech.

7.1 Srovnání

K porovnání se současnými řešeními jsem použil metriky od *Shravana* představené v 4.2. Výsledky jsou zapsané v tabulce 7.1. Jelikož se jedná o dost subjektivní metriky, v následující části textu se pokusím popsat a odůvodnit tyto výsledky.

Škálovatelnost je na jednu stranu dobrá v tom, že nejsme omezeni počtem přijíždějících aut. Můžeme mít natvrdo určený limit, kolik jsme v jeden moment schopni pojmout maximální počet aut. Dále pak můžeme určit jemnost diskretizace koordinačního prostoru. Díky těmto dvěma ovladatelným prvkům jsme schopni metodu využít i na větších křižovatkách s hustším provozem, ovšem za cenu slabší optimálnosti. Je tedy nutné zvážit výpočetní sílu plánovače a rozhodnout se mezi rychlostí hledání řešení a optimálností nalezeného řešení. Z tohoto důvodu jsem bod škálovatelnosti určil jako střední.

Algoritmus	Škálovatelnost	Strávený čas	Hladkost	Využití prostoru	Heterogenita
[27]	Skvělá	Střední	Zrychlení	Střední	Vysoká
[34]	Skvělá	Nízký	Zrychlení	Střední	Vysoká
[40]	Slabá	Vysoký	Ryv	Konzervativní	Nízká až Střední
Naše metoda	Střední	Nízký	Zrychlení	Konzervativní	Střední

Algoritmus	Robustnost	Všestrannost	Potřebná data	Ochrana dat
[27]	Ztráta zprávy	Upravitelná	Střední	Slabá
[34]	Nízká	Střední	Nízká	Limitovaná
[40]	Žádná	Nízká	Vysoká	Limitovaná
Naše metoda	Nízká	Upravitelná	Nízká	Slabá

Tabulka 7.1: Porovnání algoritmů s využitím metrik z 4.2, Zdroj: [26]

Čas strávený na křižovatce je už z definice našeho řešení minimální. Plánovač hledá pomocí diskretizace koordinačního prostoru výsledky s nejmenším počtem kumulovaného zpomalení všech aut. Kdyby výpočetní výkon nebyl problém, mohli bychom jemnost diskretizace ponechat na minimální možné, tj. každý snímek, a tím najít lokálně optimální řešení. Bylo by pouze lokálně optimální jelikož bereme v potaz pouze určitou podmnožinu přijíždějících aut a né všechny.

Hladkost je přímočará. Bereme v potaz pouze pohyby s konstantním zrychlením $a \in \{-1, 0, 1\}$ škálované nějakou konstantou, tedy hladkost je zrychlení. Tato metrika bohužel nebere v potaz hladkost laterálního zrychlení, což je škoda, protože s ručně navrženými křivkami jsme schopni docílit nejen efektivního průjezdu křižovatkou, ale díky limitování maximální povolené rychlostí i pocit příjemné jízdy.

Využití prostoru je díky limitování se pouze na stavovou mřížku a neuvažování všech možných proveditelných cest poněkud konzervativní. K tomu nedopomáhá ani fakt, že detekce kolizí je založená pouze na hrubých odhadech pomocí disků a tedy auto tedy při plánování zabírá více místa, než by doopravdy měl. To je bohužel nutný ústupek pro vytváření plánu v reálném čase.

Co se týče heterogenity, tak tady jsme schopni pojmout auta libovolných tvarů a velikostí. Ovšem tato auta musí splňovat kinodynamické vlastnosti navržené stavové mřížky a připravíme se tím tak o určitou množinu aut. Z tohoto důvodu jsem určil heterogenitu jako střední.

V mém řešení téměř vůbec neberu v potaz, že by auta nebyla schopna následovat svou určenou trasu navrženou plánovačem. Jedinou záchranou je zde již dříve zmíněná nepřesnost v detekci kolizí, která testuje auta větší, než doopravdy jsou a tedy s určitou mírou nepřesnosti jsme si ještě schopni poradit.

Všestrannost je upravitelná, jelikož jsme sice schopni navrhnout stavovou mřížku pro jakýkoliv druh křižovatky, ovšem tento krok není automatizovaný a je zapotřebí člověka aby nám ji navrhl. Tento krok by se dal potencionálně automatizovat vzali bychom auto s průměrnými kinodynamickými vlastnostmi a podle něj navrhli stavový prostor pro celou křižovatku podobně jako v [45].

Navrhovaná metoda nepotřebuje ke svému běhu téměř žádná data. Vystačíme si pouze se zrychlením, maximální rychlostí a odhadovaným příjezdem ke křižovatce. O vše ostatní se už postará plánovač. Zátěž na potřebná data je tedy nízká.

Poslední metrika, která nám zbývá, je ochrana dat. Vnitřní vadou všech centralizovaných řešení je fakt, že data se musí posílat plánovači někam na centralizovaný server, čímž do systému zavádíme různé slabiny a potencionální místa útoku. Ochrana dat je tedy slabá.

7.2 Experimenty

Plánovač jsem podrobil několika experimentům — průměrný čas strávený na křižovatce, průměrná doba trvání výpočtu plánu a propustnost křižovatky v závislosti na jemnosti diskretizace a maximálním počtu plánovaných aut. Propustností je myšleno kolik aut projede křižovatkou za jednu sekundu. Všechny měření probíhaly za stejných podmínek s maximální rychlostí $v_{max} = 4$ a intervalem generování aut zaručující zahlcení křižovatky $t = 250$ms po dobu 50 sekund. Zároveň byli auta generovány pseudonáhodně, tj. auta přijížděli ve stejném pořadí pro všechna měření. Propustnost se začala měřit až po projetí prvního auta křižovatkou. Výsledky experimentů jsou zapsány v heat mapách 7.2, 7.3 a 7.4. V tabulce je zapsaná jemnost diskretizace jako frekvence. Je tím myšlena frekvence uzlů diskretizovaného grafu vzhledem ke spojitému, tj. graf, kdy by každá hrana odpovídala jednomu snímku. V tabulkách chybí data u měření, kde výpočet jediného plánu trval déle jak minutu.

Z výsledků měření je zřejmé, že pokud je naším cílem optimálnost nalezeného plánu, je nejlepší volit co nejmenší jemnost diskretizace a co největší počet aut braný v potaz při výpočtu plánu. V reálném světě ovšem potřebujeme zohlednit i čas potřebný k výpočtu plánu. Zde se zdá být efektivnější zvětšit jemnost diskretizace než plánovat pro více aut zároveň. To je docela očekávaný výsledek, jelikož každé auto navíc znamená další dimenzi v koordinačním prostoru, čímž se problém značně ztíží.

	Maximální počet aut				
Frekvence	1	2	3	4	5
100	3.886948	3.719979	3.646858	3.520490	3.637231
50	3.048324	2.791661	2.778490	2.790080	2.736090
20	2.645412	2.437302	2.238708	2.300929	2.169250
10	2.445365	2.106272			
5	2.113249				

Tabulka 7.2: Průměrný čas strávený na křižovatce

	Maximální počet aut				
Frekvence	1	2	3	4	5
100	0.005105	0.011098	0.015279	0.021137	0.022577
50	0.005258	0.011841	0.026054	0.035841	0.024062
20	0.007923	0.038461	0.132248	1.938664	1.555990
10	0.018707	0.465379			
5	0.415512				

Tabulka 7.3: Průměry časů výpočtu plánu

	Maximální počet aut				
Frekvence	1	2	3	4	5
100	1.752216	1.810021	1.868182	1.874510	1.790575
50	2.127677	2.233949	2.233839	2.338038	2.382613
20	2.472183	2.555453	2.737928	2.712895	2.861891
10	2.611324	2.891715			
5	2.913879				

Tabulka 7.4: Propustnost

Závěr

Cílem této práce bylo prostudovat algoritmy plánování spojitého pohybu více entit a na základě provedené rešerše navrhnout vlastní metodu. V teoretické části práce jsme zjistili, co každý z těchto pojmů obnáší. Představili jsme si různé matematické reprezentace modelů aut, moderní plánovací algoritmy a současná řešení problému křižovatky. Navrhli jsme vlastní metodu postavenou na konstrukci stavové mřížky, diskretizaci koordinačního prostoru a prohledání tohoto prostoru pomocí A*.

Pro implementaci byl použit herní engine Godot a skriptovací jazyk GDScript. To se nakonec ukázalo jako docela dvousečná zbraň, jelikož na jednu stranu jsme dostali jednoduchost a rychlost psaní kódu podobnou Pythonu, ale na druhou stranu jsme také dostali rychlost exekuce kódu podobnou Pythonu. Tohle je zkrátka nevyhnutelná slabina při používání interpretovaných jazyků jako je GDScript. Naštěstí díky vhodnému zvolení parametrů algoritmu jsme byli schopni i tuto slabinu překonat a dostat tak plánovač, který je schopen pracovat v reálném čase ačkoliv se slabší optimalitou.

Jednou z potenciálních optimalizací je namísto GDScriptu použít rozhraní GDNative pro psaní kódu v C či C++. Minimálně modul pro detekci kolizí by z této změny mohl značně benefitovat. Potencionálně by se mohlo zvážit při znatelném zrychlení i použití více disků pro jedno auto pro lepší aproximaci tvaru auta. Avšak tuto změnu jsem nenaimplementoval a míra potencionálního zrychlení tak zůstává otázkou budoucí práce.

Bibliografie

1. RAFTER, Craig B; ANVARI, Bani; BOX, Simon. Traffic responsive intersection control algorithm using GPS data. In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. 2017, s. 1–6.
2. SHI, Ming-kang; JIANG, Hong; LI, Song-huan. An intelligent traffic-flow-based real-time vehicles scheduling algorithm at intersection. In: *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2016, s. 1–5.
3. AZIMI, Reza; BHATIA, Gaurav; RAJKUMAR, Ragunathan Raj; MUDALIGE, Priyantha. STIP: Spatio-temporal intersection protocols for autonomous vehicles. In: *2014 ACM/IEEE international conference on cyber-physical systems (ICCPS)*. 2014, s. 1–12.
4. LAVALLE, Steven M. *Planning algorithms*. Cambridge university press, 2006.
5. LOZANO-PEREZ. Spatial Planning: A Configuration Space Approach. *IEEE Transactions on Computers*. 1983, roč. C-32, č. 2, s. 108–120. Dostupné z DOI: 10.1109/TC.1983.1676196.
6. LOZANO-PEREZ, Tomas. Spatial planning: A configuration space approach. In: *Autonomous robot vehicles*. Springer, 1990, s. 259–271.
7. PAN, Jia; MANOCHA, Dinesh. Efficient Configuration Space Construction and Optimization for Motion Planning. *Engineering*. 2015, roč. 1, č. 1, s. 046–057. ISSN 2095-8099. Dostupné z DOI: https://doi.org/10.15302/J-ENG-2015009.
8. HALPERIN, Dan. Robust geometric computing in motion. *The International Journal of Robotics Research*. 2002, roč. 21, č. 3, s. 219–232. Dostupné z DOI: 10.1177/027836402320556412.
9. SPONG, Mark W. Underactuated mechanical systems. In: *Control problems in robotics and automation*. Springer, 1998, s. 135–150.

10. LAUMOND, Jean-Paul; SEKHAVAT, Sepanta; LAMIRAUX, Florent. Guidelines in nonholonomic motion planning for mobile robots. In: *Robot motion planning and control*. Springer, 1998, s. 1–53.

11. POLACK, Philip; ALTCHÉ, Florent; D'ANDRÉA-NOVEL, Brigitte; LA FORTELLE, Arnaud de. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In: *2017 IEEE intelligent vehicles symposium (IV)*. 2017, s. 812–818.

12. REEDS, James; SHEPP, Lawrence. Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*. 1990, roč. 145, č. 2, s. 367–393.

13. DUBINS, Lester E. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*. 1957, roč. 79, č. 3, s. 497–516.

14. KAVRAKI, Lydia E; SVESTKA, Petr; LATOMBE, J-C; OVERMARS, Mark H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*. 1996, roč. 12, č. 4, s. 566–580.

15. BRANICKY, Michael S; LAVALLE, Steven M; OLSON, Kari; YANG, Libo. Quasi-randomized path planning. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*. 2001, sv. 2, s. 1481–1487.

16. BOHLIN, Robert; KAVRAKI, Lydia E. Path planning using lazy PRM. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. 2000, sv. 1, s. 521–528.

17. LAVALLE, Steven M et al. Rapidly-exploring random trees: A new tool for path planning. 1998.

18. KARAMAN, Sertac; FRAZZOLI, Emilio. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*. 2010, roč. 104, č. 2.

19. KARAMAN, Sertac; FRAZZOLI, Emilio. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*. 2011, roč. 30, č. 7, s. 846–894.

20. KHATIB, Oussama. Real-time obstacle avoidance for manipulators and mobile robots. In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. 1985, sv. 2, s. 500–505.

21. FU-GUANG, Ding; PENG, Jiao; XIN-QIAN, Bian; HONG-JIAN, Wang. AUV local path planning based on virtual potential field. In: *IEEE International Conference Mechatronics and Automation, 2005*. 2005, sv. 4, s. 1711–1716.

22. BOUNINI, Farid; GINGRAS, Denis; POLLART, Herve; GRUYER, Dominique. Modified artificial potential field method for online path planning applications. In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. 2017, s. 180–185.

23. ERDMANN, Michael; LOZANO-PEREZ, Tomas. On multiple moving objects. *Algorithmica*. 1987, roč. 2, č. 1, s. 477–521.

24. VAN DEN BERG, Jur P; OVERMARS, Mark H. Prioritized motion planning for multiple robots. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2005, s. 430–435.

25. O'DONNELL, Patrick A; LOZANO-PÉREZ, Tomás. Deadlock-free and collision-free coordination of two robot manipulators. In: *1989 IEEE International Conference on Robotics and Automation*. 1989, s. 484–489.

26. KRISHNAN, Shravan; GOVIND AADITHYA, R; RAMAKRISHNAN, Rahul; ARVINDH, Vijay; SIVANATHAN, K. A Look at Motion Planning for AVs at an Intersection. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, s. 333–340. Dostupné z DOI: 10.1109/ITSC.2018.8569244.

27. DRESNER, Kurt; STONE, Peter. A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research*. 2008, roč. 31, s. 591–656.

28. LEVIN, Michael W; REY, David. Conflict-point formulation of intersection control for autonomous vehicles. *Transportation Research Part C: Emerging Technologies*. 2017, roč. 85, s. 528–547.

29. LU, Guangquan; LI, Lumiao; WANG, Yunpeng; ZHANG, Ran; BAO, Zewen; CHEN, Haichong. A rule based control algorithm of connected vehicles in uncontrolled intersection. In: *17th international IEEE conference on intelligent transportation systems (itsc)*. 2014, s. 115–120.

30. FAYAZI, S. Alireza; VAHIDI, Ardalan; LUCKOW, Andre. Optimal scheduling of autonomous vehicle arrivals at intelligent intersections via MILP. In: 2017, s. 4920–4925. Dostupné z DOI: 10.23919/ACC.2017.7963717.

31. SAVIC, Vladimir; SCHILLER, Elad M; PAPATRIANTAFILOU, Marina. Distributed algorithm for collision avoidance at road intersections in the presence of communication failures. In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. 2017, s. 1005–1012.

32. HE, Zhengbing; ZHENG, Liang; LU, Lili; GUAN, Wei. Erasing lane changes from roads: A design of future road intersections. *IEEE Transactions on Intelligent Vehicles*. 2018, roč. 3, č. 2, s. 173–184.

33. LEVIN, Michael W; FRITZ, Hagen; BOYLES, Stephen D. On optimizing reservation-based intersection controls. *IEEE Transactions on Intelligent Transportation Systems*. 2016, roč. 18, č. 3, s. 505–515.

34. TACHET, Remi; SANTI, Paolo; SOBOLEVSKY, Stanislav; REYES-CASTRO, Luis Ignacio; FRAZZOLI, Emilio; HELBING, Dirk; RATTI, Carlo. Revisiting street intersections using slot-based systems. *PloS one*. 2016, roč. 11, č. 3, e0149607.

35. ONIEVA, Enrique; HERNÁNDEZ-JAYO, Unai; OSABA, Eneko; PERALLOS, Asier; ZHANG, Xiao. A multi-objective evolutionary algorithm for the tuning of fuzzy rule bases for uncoordinated intersections in autonomous driving. *Information Sciences*. 2015, roč. 321, s. 14–30.

36. ZADEH, L.A. Fuzzy sets. *Information and Control*. 1965, roč. 8, č. 3, s. 338–353. ISSN 0019-9958. Dostupné z DOI: `https://doi.org/10.1016/S0019-9958(65)90241-X`.

37. ZITZLER, Eckart; LAUMANNS, Marco; THIELE, Lothar. SPEA2: Improving the strength Pareto evolutionary algorithm. *TIK-report*. 2001, roč. 103.

38. BUI, Khac-Hoai Nam; JUNG, Jason J. Cooperative game-theoretic approach to traffic flow optimization for multiple intersections. *Computers & Electrical Engineering*. 2018, roč. 71, s. 1012–1024.

39. MURGOVSKI, Nikolce; CAMPOS, Gabriel Rodrigues de; SJÖBERG, Jonas. Convex modeling of conflict resolution at traffic intersections. In: *2015 54th IEEE conference on decision and control (CDC)*. 2015, s. 4708–4713.

40. RIEGGER, Lea; CARLANDER, Markus; LIDANDER, Niklas; MURGOVSKI, Nikolce; SJÖBERG, Jonas. Centralized mpc for autonomous intersection crossing. In: *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)*. 2016, s. 1372–1377.

41. BELKHOUCHE, Fethi. Control of autonomous vehicles at an unsignalized intersection. In: *2017 American Control Conference (ACC)*. 2017, s. 1340–1345.

42. KATRINIOK, Alexander; KLEIBAUM, Peter; JOŠEVSKI, Martina. Distributed model predictive control for intersection automation using a parallelized optimization approach. *IFAC-PapersOnLine*. 2017, roč. 50, č. 1, s. 5940–5946.

43. HULT, Robert; CAMPOS, Gabriel R; FALCONE, Paolo; WYMEERSCH, Henk. An approximate solution to the optimal coordination problem for autonomous vehicles at intersections. In: *2015 American Control Conference (ACC)*. 2015, s. 763–768.

44. SETHI, Suresh; SORGER, Gerhard. A theory of rolling horizon decision making. *Annals of Operations Research*. 1991, roč. 29, č. 1, s. 387–415.

45. PIVTORAIKO, Mihail; KELLY, Alonzo. Efficient constrained path planning via search in state lattices. In: *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*. 2005, s. 1–7.

46. MORTENSON, Michael E. *Mathematics for computer graphics applications*. Industrial Press Inc., 1999.

47. HART, Peter E; NILSSON, Nils J; RAPHAEL, Bertram. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*. 1968, roč. 4, č. 2, s. 100–107.

48. SURYNEK, Pavel. *Introduction to artificial intelligence - kam.fit.cvut.cz* [online]. 2015 [cit. 2022-02-10]. Dostupné z: `https://kam.fit.cvut.cz/bi-zum/media/en/lectures/03-heuristicsearch-noanim.pdf`.

49. *DelDOT road design manual* [online]. DELAWARE DEPARTMENT OF TRANSPORTATION, 2006 [cit. 2022-01-24]. Dostupné z: `https://deldot.gov/Publications/manuals/road_design/pdfs/revisions062811/07_Intersections.pdf`.

50. ZIEGLER, Julius; STILLER, Christoph. Fast collision checking for intelligent vehicle motion planning. In: *2010 IEEE intelligent vehicles symposium*. 2010, s. 518–522.

Příloha **A**

Seznam použitých symbolů

$\mathcal{W}$ svět

$\mathcal{C}$ konfigurační prostor

$\mathcal{C}_{obs}$ kolizní konfigurace

$\mathcal{C}_{free}$ nekolizní konfigurace

X koordinační prostor

$\mathbb{R}$ reálná čísla

$\mathbb{S}$ kružnice

$\mathcal{O}$ oblast překážek

$\oplus$ Minkowského suma

q konfigurace

q_I počáteční konfigurace

q_G koncová / cílová konfigurace

τ spojitá dráha

U potenciál

U_{att} kladný potenciál

U_{rep} záporný potenciál

Příloha B

Seznam použitých zkratek

AV autonomní vozidlo

PRM probabilistic road map

RRT rapidly-exploring random tree

RRG rapidly-exploring random graph

RRT* stromová varianta RRT

GDPR general data protection regulation

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `project`
  - `src` ..... zdrojové kódy implementace
  - `dist` ..... adresář se spustitelnou formou implementace
- `thesis`
  - `src` ..... zdrojová forma práce ve formátu LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF