Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

Plánování pohybu kolových mobilních robotů

PLZEŇ, 2021 Lukáš Vladař

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Lukáš VLADAŘ**
Osobní číslo:	**A18B0557P**
Studijní program:	**B3918 Aplikované vědy a informatika**
Studijní obor:	**Kybernetika a řídicí technika**
Téma práce:	**Plánování pohybu kolových mobilních robotů**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1. Seznamte se s různými přístupy k plánování cesty (trajektorie) pro kolové mobilní roboty.
2. Podrobně popiště a analyzujte plánovací algoritmy RRT, RRT* a CL-RRT (Closed-Loop Rapidly-exploring Random Tree [3]).
3. Nalezněte vhodnou implementaci plánovacích algoritmů v jazyku Python a zhodnoťte její možné využití v systému REXYGEN.
4. V případě možnosti ověřte navržený plánovací algoritmus na miniaturním modelu vozidla TnT firmy REX Controls, s. r. o.

Rozsah bakalářské práce: **30-40 stránek A4**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

[1] ELBANHAWI, Mohamed a SIMIC, Milan. Sampling-Based Robot Motion Planning: A Review. *IEEE Access*. 2014, vol. 2, s. 56-77. ISSN 2169-3536.
[2] HOLMER, Olov. *Motion Planning for a Reversing Full-Scale Truck and Trailer System*. Linköping, 2016. Master of Science Thesis. Department of Electrical Engineering, Linköping University.
[3] LJUNGQVIST, Oskar. *Motion Planning and Stabilization for a Reversing Truck and Trailer System*. Linköping, 2015. Examensarbete. Department of Electrical Engineering, Linköping University.

Vedoucí bakalářské práce: **Prof. Ing. Miloš Schlegel, CSc.**
Katedra kybernetiky

Datum zadání bakalářské práce: **15. října 2020**
Termín odevzdání bakalářské práce: **24. května 2021**

Západočeská univerzita v Plzni
L.S.
Fakulta aplikovaných věd

Doc. Dr. Ing. Vlasta Radová
děkanka

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 15. října 2020

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne .....

Lukáš Vladař

Poděkování

Chtěl bych tímto poděkovat panu prof. Ing. Miloši Schlegelovi, CSc. za odborné vedení této práce, ale také za vstřícný a trpělivý přístup a množství věnovaného času. Velké díky patří rovněž mým rodičům, kteří mne po dobu celého studia podporovali.

Abstrakt

Cílem této práce je nalezení algoritmu plánujícího cestu pro kolové mobilní roboty. První část práce se zabývá formulací problému plánování a jsou zde představeny základní přístupy použitelné v případě plánování pohybu holonomních robotů. Druhá část je věnována praktickému návrhu algoritmů RRT a CL-RRT v případě automobilu a kloubového vozidla typu TnT. Je zde rovněž popsána uzavřená smyčka zajišťující pohyb daného robotu po naplánované cestě.

Klíčová slova

Plánování pohybu, kolové mobilní roboty, TnT, RRT, CL-RRT, strategie přímého pronásledování

Abstract

The aim of this thesis is to find a motion planning algorithm for wheeled mobile robots. The first part of the thesis covers the formulation of the motion planning problem and some basic approaches applicable for holonomic robots. The next part is focused on the practical design of RRT and CL-RRT algorithms for a car and an articulated vehicle TnT. A closed loop ensuring that the robot follows the planned path is also described.

Keywords

Motion planning, wheeled mobile robots, TnT, RRT, CL-RRT, pure pursuit control

Obsah

1 Úvod

Jednou ze základních schopností většiny živočichů je schopnost pohybovat se, tedy měnit svou polohu v čase. Tato dovednost jim umožňuje obstarávat si potravu, navazovat kontakty s ostatními příslušníky svého druhu či např. uprchnout v případě hrozícího nebezpečí.

Přemisťování se pomocí vlastních pohybových orgánů s sebou však přináší též jistá omezení. Jedná se o omezenou rychlost pohybu, množství a hmotnost nákladu, který může živočich v danou chvíli přenášet a v neposlední řadě též o skutečnost, že při pohybu musí živočich vydávat vlastní energii, po určité době se vyčerpá a musí v pohybu ustat, aby si odpočinul.

Jelikož je však člověk inteligentní tvor, vynalezl množství alternativních způsobů pohybu, které předchozí nevýhody alespoň do určité míry eliminují. Zprvu se jednalo o využívání síly ostatních živočichů či přírodních vlivů jako je vítr, později byly vynalezeny dopravní prostředky poháněné sofistikovanými zařízeními, jako je parní stroj, spalovací motor, či elektromotor.

Díky velkému rozvoji elektrotechniky a výpočetní techniky je v dnešní době možné posunout způsob dopravy osob či nákladu na zcela novou úroveň, a sice prostřednictvím tzv. autonomních vozidel. Tato vozidla využívají, stejně jako dříve používané způsoby přepravy, vlastní zdroj energie, takže se člověk při jejich užívání nevysílí, zásadním přínosem je však především fakt, že člověk není zapotřebí ani při jejich řízení.

Tato skutečnost umožňuje pohyb vozidel v prostředí, ve kterém by se člověk pohybovat nemohl (např. hluboko pod vodní hladinou či v radioaktivním prostředí), existují též každodenní situace, ve kterých přímá účast člověka není nezbytná, a proto jej autonomně řízené stroje mohou zcela nahradit (za zmínku stojí např. autonomní vysavače či sekačky na trávu).

Vytvoření takovéhoto autonomního vozidla však vyžaduje poměrně složité softwarové vybavení, neboť je třeba, aby se vozidlo dokázalo pohybovat po určené cestě a zároveň aby bylo schopno tuto cestu samo naplánovat.

Plánování pohybu je pro člověka zcela přirozenou činností, kterou provádí obvykle podvědomě. Chceme-li však tento proces zautomatizovat, ukazuje se, že se jedná o dosti složitou úlohu. Plánování pohybu se tak stává jedním z klíčových problémů řešených v robotice, využití však nalézá i v jiných aplikacích, např. při tvorbě počítačových her.

2 Formulace problému

2.1 Prostředí robotu

Uvažujme mobilní robot pohybující se ve spojitém prostředí W. V praktických aplikacích je W zpravidla omezená množina a platí $W \subset \mathbb{R}^2$ (v případě robotů pohybujících se v rovině, jako jsou např. kolové roboty, jimiž se zabývá tato práce) či v obecném případě $W \subset \mathbb{R}^3$ (pohybuje-li se robot např. ve vzduchu nebo pod vodou).

V praxi se obvykle setkáváme se situací, kdy se v uvažovaném prostředí nalézají překážky. Označme tyto překážky symbolem $\mathcal{O}_i$, kde $i \in \{1, 2, ..., k\}$, přičemž platí $\mathcal{O}_i \subseteq W$ pro každé $i \in \{1, 2, ..., k\}$. Definujme dále množinu $\mathcal{O}$ jako sjednocení všech těchto překážek, tedy platí:

$$\mathcal{O} = \bigcup_{i=\{1,2,...,k\}} \mathcal{O}_i \subseteq W \tag{2.1}$$

V této práci bude uvažováno, že množina $\mathcal{O}$ není závislá na čase, v takovém případě mluvíme o tzv. *statickém prostředí*. V obecném případě se však množina $\mathcal{O}$ může v čase měnit, k tomu dochází v situacích, kdy v prostředí existují další pohybující se objekty. Takové prostředí nazýváme *dynamickým prostředím* [1].

2.2 Konfigurační prostor

Konfiguraci robotu lze v libovolném časovém okamžiku popsat pomocí vektoru $\boldsymbol{q}(t) = \begin{bmatrix} q_1(t) & q_2(t) & ... & q_n(t) \end{bmatrix}^T$, který je prvkem tzv. konfiguračního prostoru Q [1] (v literatuře je tento prostor též často označován písmenem C [2], [3]). Poznamenejme, že dimenze prostoru Q odpovídá počtu stupňů volnosti robotu. V případě automobilu má např. konfigurační prostor dimenzi 3, neboť pro kompletní popsání konfigurace automobilu v prostoru $\mathbb{R}^2$ jsou třeba dvě polohové souřadnice $x(t)$, $y(t)$, a dále informace o úhlu natočení automobilu $\theta_0(t)$. Konfigurační vektor $\boldsymbol{q}(t)$ pak lze zapsat ve tvaru $\boldsymbol{q}(t) = \begin{bmatrix} x(t) & y(t) & \theta_0(t) \end{bmatrix}^T$.

Některé konfigurační vektory však nejsou pro robot přípustné, proto je třeba definovat množinu konfigurací $Q_{free} \subseteq Q$, která obsahuje pouze ty konfigurační vektory, ve kterých se robot smí nacházet. Uvažujme zobrazení $\mathcal{A} : Q \to W$ takové, že $\mathcal{A}(\boldsymbol{q})$ je množina všech bodů v prostředí W, ve kterých se robot nachází v okamžiku, kdy je jeho konfigurace popsána vektorem $\boldsymbol{q}$. Potom lze podle [2] definovat prostor všech konfigurací, při nichž se robot nachází v kolizi s některou z překážek, takto:

$$Q_{obs} = \{\boldsymbol{q} \in Q : \mathcal{A}(\boldsymbol{q}) \cap \mathcal{O} \neq \emptyset\} \tag{2.2}$$

Robot by se v žádném okamžiku neměl nacházet v konfiguraci, která leží v prostoru Q_{obs}. I mezi konfiguracemi, kdy robot nekoliduje s žádnou překážkou, se však mohou nacházet konfigurace, které jsou z určitého hlediska nepřípustné. Uvažujme např. automobil táhnoucí přívěs. V takovéto situaci jsou nepřípustné všechny konfigurace, pro které úhel mezi osou automobilu a osou přívěsu překračuje určitou mez, poněvadž by byl robot v kolizi sám se sebou. Označíme-li množinu takto nepřípustných konfigurací Q_N, můžeme vyjádřit množinu všech přípustných konfigurací následujícím způsobem:

$$Q_{free} = Q - Q_{obs} - Q_N \tag{2.3}$$

2.3 Problém plánování pohybu

Problém plánování pohybu je v literatuře mnohdy označován jako tzv. problém stěhování klavíru (*Piano Mover's Problem* [4]). Představme si, že je třeba přesunout velké klavírní křídlo z jedné místnosti do druhé. Tato úloha je dosti složitá, neboť je třeba pečlivě naplánovat trajektorii pohybu klavíru tak, aby během stěhování nenarazil do žádné překážky. Velmi podobná situace nastává rovněž při plánování pohybu pro mobilní robot.

V praxi bývají často zaměňovány pojmy *cesta* a *trajektorie*. V této práci budeme stejně jako v [2] chápat cestu jako posloupnost bodů $\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_p$, kde $\boldsymbol{c}_i \in W$ pro každé $i \in \{1, 2, ..., p\}$, zatímco trajektorií se bude rozumět libovolná funkce $\boldsymbol{q}(t)$, kde pro každý časový okamžik t platí $\boldsymbol{q}(t) \in Q$.

Na základě [5] nyní formulujme úlohu plánování trajektorie. Mobilní robot lze chápat jako dynamický systém, jehož konfigurace $\boldsymbol{q}(t)$ se řídí soustavou diferenciálních rovnic 1. řádu ve tvaru

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{f}\Big(\boldsymbol{q}(t), \boldsymbol{u}(t)\Big), \tag{2.4}$$

kde $\boldsymbol{u}(t)$ je vstup systému a $\boldsymbol{f}\Big(\boldsymbol{q}(t), \boldsymbol{u}(t)\Big)$ je libovolná funkce vstupu a konfigurace robotu, přičemž je dána počáteční podmínka $\boldsymbol{q}(0) = \boldsymbol{q}_0$. Na vstup systému jsou v praktických aplikacích kladena omezení ve tvaru $\forall t : \boldsymbol{u}(t) \in U$, kde U je vhodně zvolená omezená množina.

Pro hledání trajektorie definujeme množinu konfigurací, které jsou považovány za cílové, označme tuto množinu $Q_{goal} \subseteq Q_{free}$. Úlohu plánování trajektorie pak lze chápat jako úlohu nalezení vstupu $\boldsymbol{u}(t)$ na intervalu $t \in \langle 0; t_f \rangle$, kde $t_f \in \langle 0; +\infty)$, tak aby výsledné řešení diferenciální rovnice (2.4) $\boldsymbol{q}(t)$ minimalizovalo hodnotu kritéria

$$\int_0^{t_f} \Gamma\Big(\boldsymbol{q}(t)\Big) dt + \Phi\Big(\boldsymbol{q}(t_f), Q_{goal}\Big) \tag{2.5}$$

za dodržení omezujících podmínek ve tvaru:

$$\begin{aligned} \forall t \in \langle 0; t_f \rangle : \boldsymbol{u}(t) &\in U \\ \forall t \in \langle 0; t_f \rangle : \boldsymbol{q}(t) &\in Q_{free} \\ \boldsymbol{q}(t_f) &\in Q_{goal} \\ \boldsymbol{q}(0) &= \boldsymbol{q}_0, \end{aligned} \tag{2.6}$$

kde $\Gamma\Big(\boldsymbol{q}(t)\Big)$ je funkce hodnotící danou konfiguraci $\boldsymbol{q}(t)$ (použijeme-li např. $\Gamma\Big(\boldsymbol{q}(t)\Big) = 1$, jsou všechny konfigurace ohodnoceny stejně a první část kritéria (2.5) hodnotí pouze čas nutný k vykonání pohybu po nalezené trajektorii) a $\Phi\Big(\boldsymbol{q}(t_f), Q_{goal}\Big)$ je funkce hodnotící optimalitu cílové konfigurace $\boldsymbol{q}(t_f)$.

Analytické řešení tohoto problému je velmi složité, proto se v praxi používají numerické metody, které poskytují suboptimální řešení [6].

3 Plánování pohybu pro holonomní roboty

3.1 Holonomní a neholonomní roboty

Často se setkáváme se situací, kdy je pohyb robotu jistým způsobem omezen. Tato omezení lze rozdělit do dvou skupin, na *holonomní* a *neholonomní* vazby.

Uvažujme robot, jehož konfigurace je popsána vektorem $\boldsymbol{q}(t)$. Holonomními vazbami rozumíme omezení, která lze vyjádřit ve tvaru

$$f_H(\boldsymbol{q}) = 0 \tag{3.1}$$

Existuje-li pro robot holonomní vazba, pak lze s využitím vztahu (3.1) vyjádřit jednu z konfiguračních souřadnic pomocí ostatních souřadnic, čímž se sníží dimenze konfiguračního prostoru o 1 [1].

Oproti tomu neholonomními vazbami se rozumí vazby ve tvaru

$$f_N(\boldsymbol{q}, \dot{\boldsymbol{q}}) = 0, \tag{3.2}$$

kde funkci $f_N(\boldsymbol{q}, \dot{\boldsymbol{q}})$ nelze integrovat podle času (je-li rovnice (3.2) integrovatelná podle času, pak ji lze transformovat na holonomní vazbu ve tvaru (3.1)) [1]. Neholonomní vazby neumožňují zjednodušení konfiguračního prostoru, kladou pouze omezení na derivaci konfiguračního vektoru, což se v praxi může projevit např. tím, že se robot dokáže pohybovat pouze určitými směry.

Holonomním robotem budeme rozumět robot, ve kterém neexistují neholonomní vazby, pojmem *neholonomní robot* budeme naopak označovat robot, pro který existuje alespoň jedna neholonomní vazba.

Je-li robot holonomní, pak tato skutečnost zásadním způsobem usnadňuje úlohu plánování pohybu, neboť neexistují žádná omezení na změnu konfiguračního vektoru, a tudíž stačí nalézt libovolnou trajektorii, která leží kompletně v prostoru Q_{free}, a tato trajektorie bude robotem realizovatelná.

Pro řešení problému plánování pohybu pro holonomní roboty bylo navrženo velké množství metod. Tyto metody nalézají uplatnění i mimo oblast robotiky, např. při vývoji počítačových her, neboť postavy v těchto hrách se zpravidla mohou pohybovat všemi směry a lze je tedy rovněž považovat za holonomní systémy. Některé z nejobvyklejších metod holonomního plánování jsou popsány v této kapitole.

3.2 Algoritmy typu Bug

Algoritmy typu Bug jsou typickými představitely tzv. *neinformovaného plánování*, tedy úlohy, kdy není apriorně známé prostředí robotu [3] nebo se toto prostředí dynamicky mění [1]. V takovém případě musí robot pomocí svých senzorů shromažďovat informace o překážkách v reálném čase a plánovat svůj pohyb pouze na základě částečné znalosti prostředí.

Algoritmy typu Bug jsou použitelné též v situaci, kdy je prostředí předem známé, ovšem jejich nevýhodou je skutečnost, že nalezené řešení v obecném případě zdaleka není optimální. Oproti tomu výhodou těchto algoritmů je, že nejsou výpočetně náročné a vyžadují minimální paměť [1], za výhodu lze rovněž považovat fakt, že robot může zahájit pohyb okamžitě po zadání cílové konfigurace a není nutné nejprve čekat na naplánování trajektorie. Následující podkapitoly popisují princip fungování těchto algoritmů na základě [1].

3.2.1 Algoritmus Bug0

Tento algoritmus je velmi jednoduchý, neboť po robotu vyžaduje pouze dvě opakující se činnosti: pohyb směrem k cíli a pohyb podél překážky. Implementace tohoto algoritmu pro mobilní robot pohybující se v rovině je formou pseudokódu zapsána v Algoritmu 3.1, výsledná cesta v případě jednoduchého prostředí je zobrazena v levé části Obrázku 3.1.

Algoritmus 3.1: Bug0

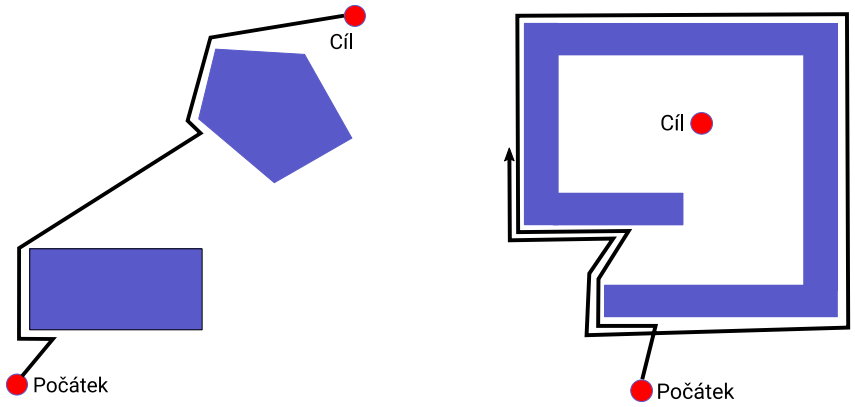
```
1 while Není dosažen cíl do
2 |   if Není detekována překážka then
3 |   |   Pohybuj se po přímce směrem k cíli
4 |   else
5 |   |   Pohybuj se podél detekované překážky ve směru hodinových ručiček
6 |   end
7 end
```

[1] uvádí, že při použití algoritmu Bug0 není garantováno nalezení cesty, v případě, že tato cesta existuje. Příklad selhání algoritmu je znázorněn v pravé části Obr. 3.1.

Poznamenejme, že nezáleží na směru objíždění překážkek (řádek 5), je však třeba dodržovat zvolený směr konzistentně v každé iteraci [1].

3.2.2 Algoritmus Bug1

Algoritmus Bug1 na rozdíl od výše zmíněného algoritmu Bug0 garantuje nalezení cesty v případě, že cesta existuje. Jeho nevýhodou je však nepatrně vyšší výpočetní složitost a v mnoha případech především neoptimální délka nalezené cesty.



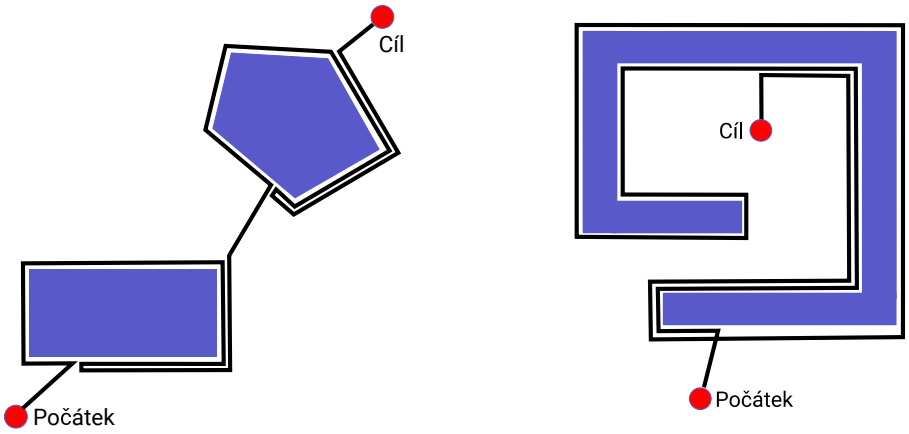
Obrázek 3.1: Ilustrace fungování algoritmu Bug0 za standardních podmínek (vlevo) a v situaci, kdy řešení není nalezeno (vpravo)

Postup hledání cesty je naznačen v Algoritmu 3.2. Stejně jako v případě předchozího algoritmu se robot pohybuje po přímce, která jej spojuje s cílem. Ve chvíli, kdy detekuje překážku, však tuto překážku nejprve celou objede (řádky 7 – 14), a poté se vrátí do bodu na obvodu překážky, který leží nejblíže k cíli (řádky 15 – 17) a pokračuje v pohybu směrem k cíli.

Algoritmus 3.2: Bug1

```
1  while Není dosažen cíl do
2      if Není detekována překážka then
3          Pohybuj se po přímce směrem k cíli
4      else
5          bodNejblizeKCili ← aktuální poloha
6          minVzdalenostDoCile ← vzdálenost z aktuální polohy do cíle
7          while Není znovu dosažen bod příjezdu k překážce do
8              Pohybuj se podél detekované překážky ve směru hodinových
                ručiček
9              vzdalenostDoCile ← vzdálenost z aktuální polohy do cíle
10             if vzdalenostDoCile < minVzdalenostDoCile then
11                 bodNejblizeKCili ← aktuální poloha
12                 minVzdalenostDoCile ← vzdalenostDoCile
13             end
14         end
15         while Není dosažen bod bodNejblizeKCili do
16             Pohybuj se podél překážky ve směru, ve kterém je možné se
                dostat do bodu bodNejblizeKCili za kratší dobu
17         end
18     end
19 end
```

Příklad cest nalezených algoritmem Bug1 je znázorněn na Obr. 3.2.



Obrázek 3.2: Ilustrace fungování algoritmu Bug1 ve dvou prostředích

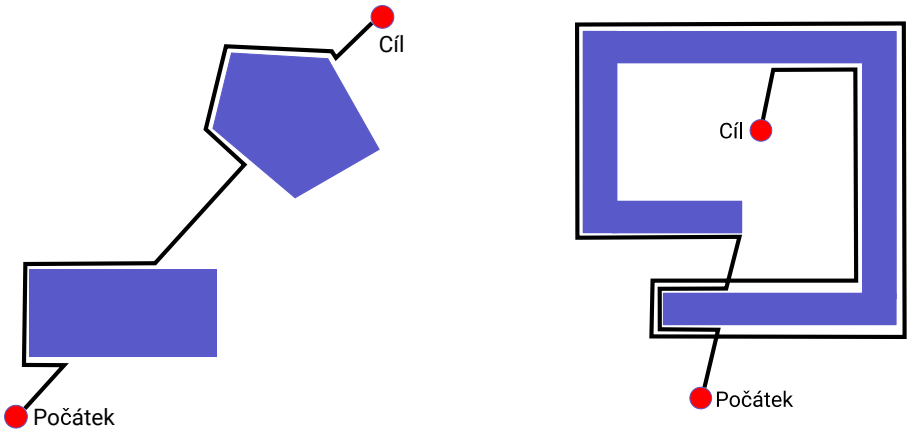
3.2.3 Algoritmus Bug2

Algoritmus Bug2 do určité míry spojuje výhody dříve zmíněných algoritmů, tedy garanci nalezení řešení a relativně krátkou nalezenou cestu. Jeho princip spočívá v tom, že se robot pohybuje po přímce spojující počáteční a cílový bod a detekuje-li překážku, objíždí ji v určeném směru až do chvíle, kdy se znovu ocitne na přímce spojující počátek a cíl, tentokrát ovšem blíže k cíli (viz Algoritmus 3.3).

Algoritmus 3.3: Bug2

```
1  while Není dosažen cíl do
2      if Není detekována překážka then
3          Pohybuj se po přímce procházející počátkem a cílem směrem k cíli
4      else
5          vzdalenostDoCilePriPrijezdu ← vzdálenost z aktuální polohy do cíle
6          while True do
7              if Robot se nachází na přímce spojující počátek a start and
                 vzdálenost z aktuální polohy do cíle <
                 vzdalenostDoCilePriPrijezdu then
8                  break
9              else
10                 Pohybuj se podél překážky ve směru hodinových ručiček
11             end
12         end
13     end
14 end
```

Ačkoliv je algoritmus obvykle efektivnější nežli Bug1, v určitých situacích dochází ke zbytečnému opakování částí cesty [1], viz Obr. 3.3.



Obrázek 3.3: Ilustrace fungování algoritmu Bug2 ve dvou prostředích

3.3 Metoda PRM

Další z postupů použitelných při plánování pohybu pro holonomní roboty je metoda PRM (z angl. *Probabilistic RoadMap*). Jedná se o nedeterministický algoritmus, který v konečeném čase negarantuje nalezení řešení, v určitých situacích je však jeho použití velmi výhodné. Metoda PRM je prováděna offline a pro výpočet je tedy třeba apriorní znalost celého prostředí.

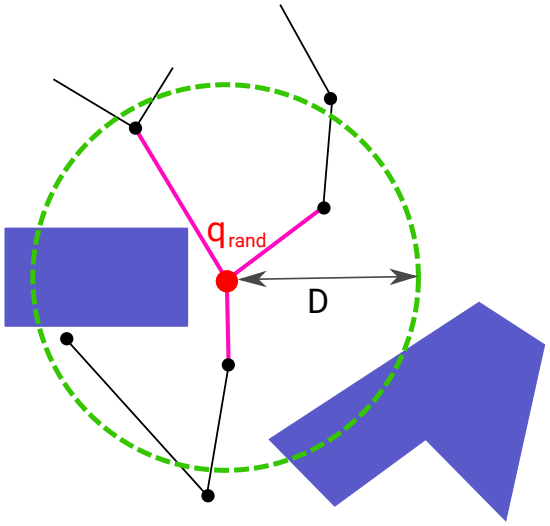
Plánování pohybu metodou PRM probíhá ve dvou fázích. První z nich je fáze učení (*learning phase*), ve které je konstruován neorientovaný graf, jehož uzly odpovídají bodům ve volné části konfiguračního prostoru Q_{free} (viz Kapitola 2.2) a hrany reprezentují spojnice těchto bodů, které nekolidují s žádnou z překážek (leží tedy též kompletně v Q_{free}) [1]. Příklad implementace fáze učení je uveden v Algoritmu 3.4

Algoritmus 3.4: PRM – fáze učení

```
1  uzly ← ∅
2  hrany ← ∅
3  for i in {1, 2, ..., n} do
4  |  q_rand ← Náhodný bod v prostoru Q_free
5  |  for uzel in uzly do
6  |  |  if Vzdálenost mezi body q_rand a uzel ≤ D and Úsečka mezi vrcholy
   |  |   q_rand a uzel leží v Q_free then
7  |  |  |  Vlož úsečku mezi vrcholy q_rand a uzel do seznamu hrany
8  |  |  end
9  |  end
10 |  Vlož q_rand do seznamu uzly
11 end
```

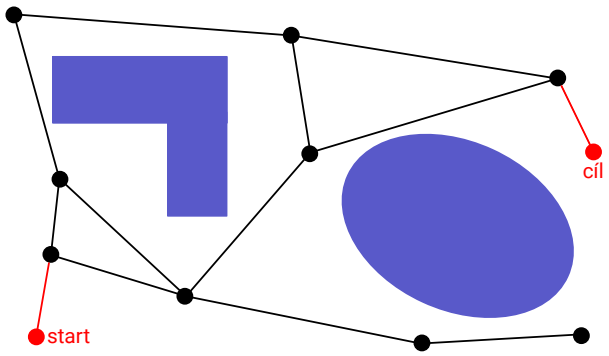
Fáze učení probíhá iterativním způsobem, přičemž počet iterací n zadá uživatel jako parametr algoritmu. V každé iteraci je zvolena náhodná konfigurace ve volném prostoru Q_{free} (řádek 4). Následně jsou vybrány všechny uzly v grafu, jejichž Eukleidovská vzdálenost od bodu q_{rand} nepřekračuje uživatelem zadaný parametr D, a

leží-li jejich spojnice s bodem q_{rand} kompletně v prostoru Q_{free}, pak je tato spojnice vložena do grafu (řádky 5 – 9). Nakonec je mezi uzly grafu vložen též bod q_{rand} (řádek 10). Příklad jedné iterace tohoto algoritmu je znázorněn na Obrázku 3.4.



Obrázek 3.4: Fáze učení algoritmu PRM. Růžovou barvou jsou vyznačeny úsečky vkládané do grafu během jedné iterace.

Po zkonstruování grafu následuje fáze hledání. Počáteční i cílová konfigurace je spojena hranou s nejbližším uzlem, pro který je toto spojení možné (tj. pro který celá příslušná úsečka leží v prostoru Q_{free}), viz Obr. 3.5. Řešení je poté nalezeno některou z metod pro hledání cesty v grafu.



Obrázek 3.5: Fáze hledání algoritmu PRM

Jak uvádí [1], fáze učení a hledání mohou být spouštěny též iterativně. Pokud není řešení nalezeno, pak se při dalším spuštění fáze učení graf rozroste, čímž se zvýší pravděpodobnost úspěchu při dalším průběhu fáze hledání.

Výhodou metody PRM je zejména skutečnost, že fáze učení nevyužívá informaci o počáteční a cílové konfiguraci. Tuto fázi lze tedy pro dané prostředí spustit pouze jednou a pro libovolnou počáteční a cílovou konfiguraci poté stačí provést pouze fázi hledání.

Algoritmus PRM mnohdy selhává v úloze plánování pohybu úzkou uličkou mezi

dvěma překážkami. Tento nedostatek však lze vyřešit např. přidáním dalších uzlů metodou tzv. *bridge test*u [1].

3.4 Algoritmus RRT

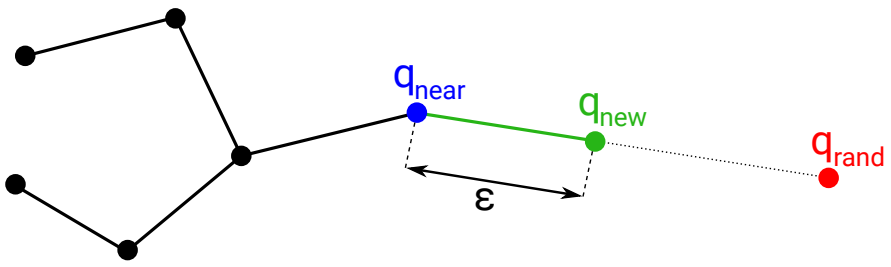
RRT, neboli metoda *Rychle rostoucích náhodných stromů* (angl. *Rapidly-Exploring Random Trees*) je algoritmus použitelný při plánování pohybu pro neholonomní roboty, existuje však i jeho jednodušší varianta používaná v úloze holonomního plánování, kterou popisuje např. [1].

Jedná se opět o nedeterministický iterační algoritmus. Postup konstrukce stromu RRT je popsán v Algoritmu 3.5. Při inicializaci je do stromu vložena počáteční konfigurace (řádek 1), následně je spuštěn cyklus, jehož počet iterací je uživatelem zadávaný parametr (řádky 2 – 9). V každé iteraci je zvolena náhodná konfigurace (řádek 3), je vyhledán nejbližší bod ve stromě (řádek 4), a poté je vygenerován konfigurační vektor q_{new}, který leží na polopřímce dané body q_{near} a q_{rand} v uživatelem stanovené vzdálenosti ϵ od bodu q_{near} (řádek 5). Leží-li spojnice bodů q_{near} a q_{new} v prostoru Q_{free}, pak je bod q_{new} přidán do stromu (řádek 7). Ilustrace jedné iterace algoritmu je znázorněna na Obr. 3.6.

Algoritmus 3.5: Konstrukce stromu RRT pro holonomní roboty

```
1 RRT ← počáteční konfigurace
2 for i in {1, 2, ..., n} do
3 |   q_rand ← konfigurace náhodně vybraná z prostoru Q
4 |   q_near ← uzel ze stromu RRT, který je nejblíže k bodu q_rand
5 |   q_new ← konfigurace na polopřímce q_near q_rand ve vzdálenosti ϵ
6 |   if Úsečka q_near q_new leží celá v prostoru Q_free then
7 |   |   Vlož uzel q_new do stromu RRT jako potomka uzlu q_near
8 |   end
9 end
```

Poté, co je strom zkonstruován, se algoritmus pokusí připojit ke stromu cílový stav [1]. V případě úspěchu byla cesta nalezena.



Obrázek 3.6: Ukázka konstrukce stromu RRT pro holonomní roboty

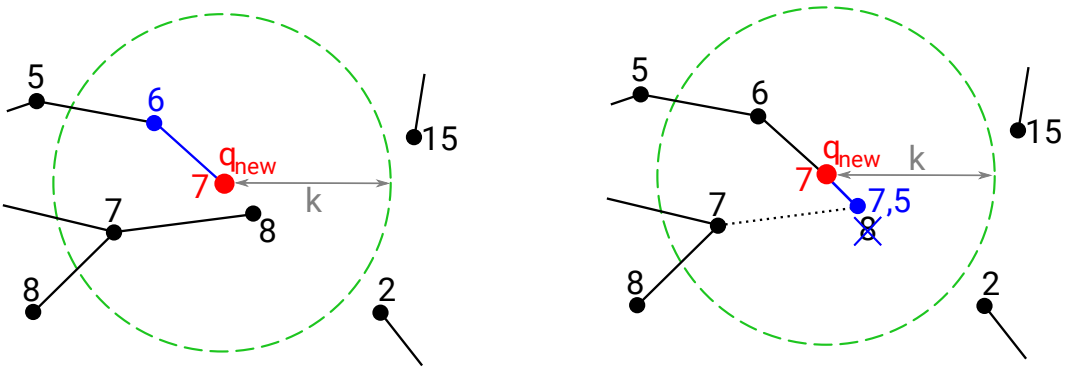
Algoritmus RRT nalezl uplatnění v mnoha praktických aplikacích a byly rovněž vyvinuty jeho modifikace, které v určitých případech zdokonalují jeho vlastnosti.

[2] uvádí např. modifikaci *Goal Bias*, která každou k-tou iteraci namísto generování náhodné konfigurace expanduje strom směrem ke konfiguraci cílové. Další možnou modifikací je konstruovat kromě stromu vycházejícího z počáteční konfigurace ještě druhý strom, jehož kořenem je cílová konfigurace [3]. Řešení je pak nalezeno v okamžiku, kdy se podaří oba stromy propojit.

3.5 Algoritmus RRT*

Nevýhodou algoritmu RRT je, že nalezená cesta obvykle není příliš optimální. Tento nedostatek se pokouší odstranit modifikace tohoto algoritmu nazvaná RRT*.

Na rozdíl od algoritmu RRT je každý uzel ve stromě ohodnocen svou cenou (cenou se obvykle rozumí délka nalezené cesty z počáteční konfigurace do daného uzlu). Při vkládání nového bodu q_{new} do stromu pak není tento bod prohlášen za potomka nejbližšího uzlu ve stromě, ale namísto toho jsou nalezeny všechny uzly, které se nacházejí ve vzdálenosti maximálně k od bodu q_{new}, a jako předek bodu q_{new} je vybrán ten z nich, který má nejnižší cenu [7] (viz levá část Obr. 3.7).



Obrázek 3.7: Ukázka fungování algoritmu RRT*. U každého uzlu je vyznačena jeho cena

Dalším krokem je tzv. předrátování (angl. *rewiring*). Všechny uzly ve vzdálenosti maximálně k od bodu q_{new} jsou otestovány, zda by nedošlo ke snížení jejich ceny, pokud by byly potomkem uzlu q_{new}. Je-li výsledná cena opravdu nižší, jsou dané body prohlášeny za potomky uzlu q_{new} (v pravé části Obrázku 3.7 např. došlo ke změně předka modře zvýrazněného uzlu).

4 Plánování pohybu automobilu pomocí algoritmu RRT

4.1 Motivace

Kapitola 3 popisovala některé z nejběžněji používaných algoritmů tzv. holonomního plánování. Tyto techniky nalézají využití v široké škále aplikací, v oblasti zabývající se mobilními roboty se však setkáváme převážně s neholonomními systémy.

V případě neholonomních robotů je úloha plánování pohybu značně zkomplikována, neboť na rozdíl od holonomních robotů neplatí tvrzení, že libovolná trajektorie ležící kompletně ve volné části konfiguračního prostoru Q_{free} je robotem realizovatelná. Proto je nutné plánovat trajektorii na základě matematického modelu uvažovaného robotu.

Typickým představitelem neholonomních mobilních robotů je automobil. Uvažujme automobil zobrazený na Obrázku 4.3. Označme polohové souřadnice středu zadní nápravy x a y a orientaci automobilu θ_0. Uvažujeme-li, že při jízdě nedochází se smyku kol, pak se automobil může pohybovat pouze směrem vpřed, platí tedy:

$$\forall t \in \langle 0; +\infty \rangle \exists k : \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \end{bmatrix} = k \cdot \begin{bmatrix} \cos\left[\theta_0(t)\right] \\ \sin\left[\theta_0(t)\right] \end{bmatrix} \tag{4.1}$$

Jednoduchými úpravami obdržíme vztah

$$\frac{\dot{x}(t)}{\cos\left[\theta_0(t)\right]} - \frac{\dot{y}(t)}{\sin\left[\theta_0(t)\right]} = 0, \tag{4.2}$$

který odpovídá standardní formě zápisu neholonomní vazby, viz (3.2).

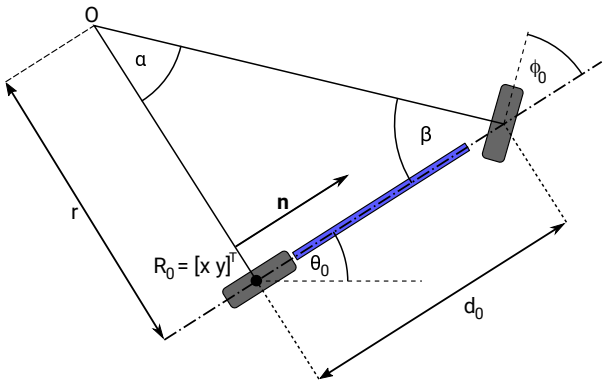
Tato kapitola je věnována praktickému řešení úlohy plánování pohybu pro automobil s využitím plánovacího algoritmu RRT.

4.2 Odvození matematického modelu

4.2.1 Bicykl

Odvoďme nejprve kinematický model bicyklu, který se automobilu blízce podobá. Jedná se o robot se dvěma koly, přičemž zadní kolo je poháněno a natáčením předního kola lze ovládat směr jízdy, viz Obr. 4.1. Uvažujme, že střed zadního kola (tj. bod

$R_0 = \begin{bmatrix} x & y \end{bmatrix}^T$) se pohybuje rychlostí u_1 a celý bicykl se otáčí po kružnici se středem v bodě O. Matematický model pak lze odvodit na základě [1].



Obrázek 4.1: Nákres bicyklu

Zanedbáme-li skutečnost, že může docházet ke smyku kol, pak směr pohybu obou kol musí být kolmý na jejich spojnici s bodem O. Časová derivace orientace bicyklu je pak rovna derivaci úhlu mezi těmito spojnicemi, tj.

$$\dot{\theta_0} = \dot{\alpha} \tag{4.3}$$

Pro úhlovou rychlost $\dot{\alpha}$ přitom platí:

$$\dot{\alpha} = \frac{u_1}{r} \tag{4.4}$$

Vyjádřeme nyní vzdálenost r pomocí úhlu natočení předního kola ϕ_0:

$$r = \frac{d_0}{\mathrm{tg}(\alpha)} = \frac{d_0}{\mathrm{tg}(\frac{\pi}{2} - \beta)} = \frac{d_0}{\mathrm{tg}[\frac{\pi}{2} - (\frac{\pi}{2} - \phi_0)]} = \frac{d_0}{\mathrm{tg}(\phi_0)} \tag{4.5}$$

Dosazením do (4.4) a (4.3) dostáváme vztah pro vývoj orientace bicyklu:

$$\dot{\theta_0} = u_1 \cdot \frac{\mathrm{tg}(\phi_0)}{d_0} \tag{4.6}$$

Rovnice pro polohu středu zadního kola získáme ze znalosti, že se zadní kolo pohybuje ve směru orientace bicyklu. Jednotkový vektor ve směru pohybu bicyklu získáme takto:

$$\boldsymbol{n} = \begin{bmatrix} \cos\theta_0 \\ \sin\theta_0 \end{bmatrix}, \tag{4.7}$$

přičemž rychlost pohybu zadního kola je rovna vstupu u_1, platí tedy:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = u_1 \cdot \boldsymbol{n} = u_1 \cdot \begin{bmatrix} \cos\theta_0 \\ \sin\theta_0 \end{bmatrix} \tag{4.8}$$

Z rovnic (4.6) a (4.8) lze sestavit stavový model bicyklu:

$$\begin{aligned} \dot{x} &= u_1 \cos\theta_0 \\ \dot{y} &= u_1 \sin\theta_0 \\ \dot{\theta}_0 &= u_1 \frac{\mathrm{tg}(\phi_0)}{d_0} \end{aligned} \tag{4.9}$$

Za vstupy systému jsou považovány rychlost pohybu bicyklu u_1 a úhel natočení předního kola ϕ_0. [8] uvádí možnost přidání čtvrté rovnice ve tvaru

$$\dot{\phi_0} = -\frac{1}{\tau}\phi_0 + \frac{1}{\tau}u_2, \tag{4.10}$$

neboť úhel ϕ_0 nelze řídit libovolně rychle, za vstup je zde tedy považován požadovaný úhel natočení předního kola u_2. Je-li však časová konstanta τ dostatečně malá, lze používat pouze zjednodušený model (4.9) [8].

4.2.2 Tříkolka

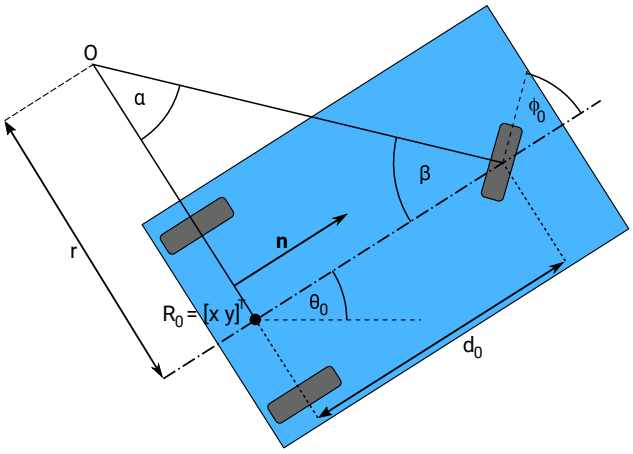
Rozšiřme nyní bicykl na kolový robot se třemi koly, viz Obrázek 4.2. Směr jízdy je stejně jako v případě bicyklu ovládán otáčením předního kola, jediný rozdíl spočívá ve skutečnosti, že se na zadní nápravě nacházejí dvě kola. Systém však vykazuje stejné chování jako bicykl a lze jej rovněž popsat modelem (4.9).

[1] uvádí, že mobilní roboty v podobě tříkolky jsou v praxi běžnější než roboty typu bicykl, neboť jsou díky třem kolům stabilní.

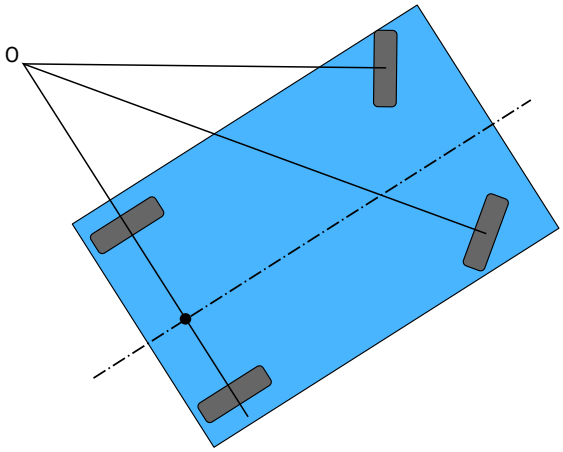
4.2.3 Automobil využívající Ackermannovo řízení

Při řízení automobilů se čtyřmi koly je zpravidla využíván tzv. Ackermannův princip řízení [1].

Tento princip vychází z požadavku, aby během pohybu automobilu nedocházelo ke smykům, neboť při smyku jsou rychleji opotřebovávány pneumatiky. Chceme-li smykům zamezit, musí být směr pohybu obou předních kol kolmý na jejich spojnici se středem otáčení O, z čehož vyplývá, že každé z předních kol musí být natočeno vzhledem k ose automobilu o jiný úhel (viz Obr. 4.3).



Obrázek 4.2: Schéma kolového robotu se třemi koly



Obrázek 4.3: Jízda automobilu využívajícího Ackermannův princip řízení

Chování takto navrženého systému se shoduje s vlastnostmi výše popsaného tříkolového robotu, díky tomu lze automobil rovněž popsat matematickým modelem bicyklu (4.9).

4.3 Formulace úlohy plánování pohybu pro automobil

Formulujme nyní úlohu plánování pohybu pro automobil. Uvažujme automobil, tedy systém zobrazený na Obrázku 4.3, který je popsán matematickým modelem (4.9), se známou vzdáleností přední a zadní nápravy d_0. Předpokládejme, že směr natočení předních kol je omezen, máme tedy zadáno číslo ϕ_0^{max} a požadujeme, aby pro každý časový okamžik platilo $|\phi_0| \leq \phi_0^{max}$.

Mějme dále definováno prostředí $W \subset \mathbb{R}^2$ a množinu překážek $\mathcal{O} \subset W$, přičemž překážky jsou reprezentovány konvexními mnohoúhelníky.

Cílem této úlohy je vytvořit algoritmus, který pro uživatelem zadanou počáteční konfiguraci $\boldsymbol{q}_0$ nalezne trajektorii, která je automobilem realizovatelná, leží kompletně

v prostoru Q_{free} a její koncový stav $\boldsymbol{q}_\tau$ leží v množině Q_{goal} definované následujícím způsobem:

$$Q_{goal} = \{\boldsymbol{q} = \begin{bmatrix} x & y & \theta_0 \end{bmatrix}^T : \sqrt{(x - x_{goal})^2 + (y - y_{goal})^2} \leq \tilde{d}; |\theta_0 - \theta_{0goal}| \leq \tilde{\theta}_0\} \tag{4.11}$$

Jedná se tedy o množinu všech konfigurací, pro něž Eukleidovská vzdálenost od požadovaného cílového bodu $\begin{bmatrix} x_{goal} & y_{goal} \end{bmatrix}$ nepřevyšuje parametr $\tilde{d}$ a odchylka od požadované orientace vozidla v cílové konfiguraci je rovna maximálně uživatelem zadanému číslu $\tilde{\theta}_0$.

Popsaný problém bude v této kapitole řešen pomocí algoritmu RRT.

4.4 Využití algoritmu RRT pro neholonomní roboty

V Kapitole 3 byl prezentován algoritmus RRT jako metoda plánování pohybu pro holonomní roboty. Faktem ovšem je, že tento algoritmus lze použít rovněž v úloze neholomního plánování. Pseudokód algoritmu RRT přizpůsobeného úloze neholonomního plánování je na základě [9] zapsán v Algoritmu 4.1.

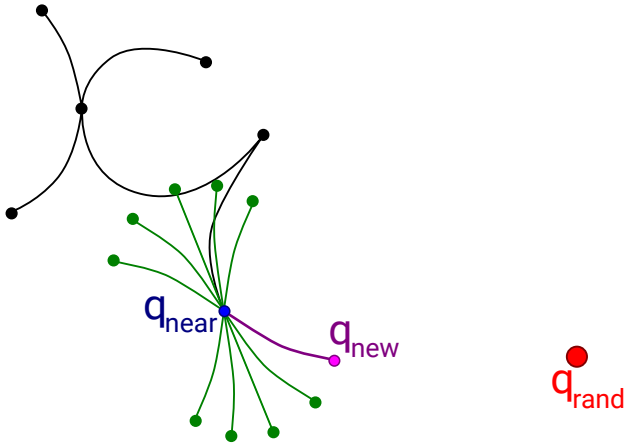
Algoritmus 4.1: Plánování pohybu neholonomních robotů metodou RRT

```
1  RRT ← q_start
2  for i in {1, 2, ..., k} do
3  |  q_rand ← náhodná konfigurace z prostoru Q
4  |  q_near ← uzel stromu RRT, který má nejmenší vzdálenost ke q_rand
5  |  mozniPotomci ← ∅
6  |  for u in U do
7  |  |  q_u ← konfigurace dosažitelná z bodu q_near aplikováním vstupu u po
   |  |   dobu dt
8  |  |  Přidej q_u do seznamu mozniPotomci
9  |  end
10 |  q_new ← Konfigurace ze seznamu mozniPotomci, která má nejmenší
   |   vzdálenost od q_rand a leží v Q_free
11 |  if Uvažovaná trajektorie z q_near do q_new leží v Q_free then
12 |  |  Vlož q_new do stromu RRT
13 |  |  if q_new je cílovým stavem then
14 |  |  |  break
15 |  |  end
16 |  end
17 end
```

Při inicializaci algoritmu je do stromu RRT vložena pouze počáteční konfigurace q_{start} (řádek 1). Následně je spuštěn cyklus, v jehož každé iteraci je zvolena náhodná konfigurace q_{rand} ležící v konfiguračním prostoru Q (řádek 3), a poté je ze stromu

RRT vybrán uzel q_{near}, který má od této konfigurace nejmenší vzdálenost z hlediska zvolené metriky (řádek 4). Poté je po předem stanovenou dobu dt simulován pohyb robotu z konfigurace q_{near} za aplikování různých vstupů, čímž obdržíme množinu konfigurací $mozniPotomci$, do kterých se lze za dobu dt z konfigurace q_{near} dostat (řádky 5 – 9). Z této množiny je dále vybrána konfigurace ležící ve volném prostoru Q_{free}, která se nachází nejblíže ke konfiguraci q_{rand} (řádek 10). Následně je proveden test, zda simulovaná trajektorie z bodu q_{near} do bodu q_{new} leží celá v prostoru Q_{free}. V případě úspěchu je konfigurace q_{new} vložena do stromu jako potomek konfigurace q_{near} (řádek 12). V případě, že je stav q_{new} cílovým stavem, tedy platí $q_{new} \in Q_{free}$, je řešení problému nalezeno a algoritmus končí (řádek 14). Algoritmus je rovněž ukončen v případě, že ani po vykonání maximálního počtu iterací není nalezeno řešení.

Jedna iterace algoritmu v případě dvourozměrného konfiguračního prostoru je zobrazena na Obr. 4.4.



Obrázek 4.4: Ukázka jedné iterace algoritmu RRT pro neholonomní roboty. Červenou barvou je označena náhodně zvolená konfigurace q_{rand}, modrou barvou nejbližší konfigurace ve stromě q_{near}. Jsou spočítány konfigurace, do kterých se lze dostat z konfigurace q_{near} (označeny zeleně) a je vybrána konfigurace q_{new}, která z nich leží nejblíže ke q_{rand} a zároveň se nachází v prostoru Q_{free} (vyznačena fialovou barvou)

Implementace tohoto algoritmu vyžaduje vyřešení několika zásadních problémů jako je způsob simulace pohybu automobilu, hledání nejbližšího uzlu ve stromě, otestování, zda při pohybu po určité trajektorii nedojde ke kolizi s překážkami apod. Všechny tyto dílčí problémy budou dále diskutovány v této kapitole.

4.5 Diskretizace množiny vstupů

Algoritmus v každé iteraci vybere jednu z konfigurací uložených ve stromě a aplikuje na ni všechny možné vstupy. Problém je, že množina přípustných vstupů U je zpravidla spojitá, je tedy nutné ji vhodným způsobem diskretizovat.

V případě automobilu existují dvě vstupní proměnné: rychlost jízdy u_1 a úhel na-

točení předních kol ϕ_0 (viz matematický model (4.9)). V úloze plánování pohybu nezáleží na rychlosti jízdy, neboť je-li pohyb po určité cestě realizovatelný rychlostí v, pak je realizovatelný také pro libovolnou rychlost $k \cdot v$, kde $k > 0$ (tato skutečnost vyplývá z toho, že rychlost u_1 vystupuje ve stavovém modelu (4.9) jako koeficient násobící směr pohybu ve stavovém prostoru, ovlivňuje tedy pouze rychlost pohybu v prostoru Q, nikoliv směr). Díky tomu lze na model aplikovat pouze dvě hodnoty vstupu u_1, kupříkladu $u_1 \in \{\pm 1\}$.

Pro úhel předních kol máme zadáno omezení $\phi_0 \in \langle -\phi_0^{max}; \phi_0^{max} \rangle$. Tento interval tedy diskretizujeme např. pěti vzorky. Jelikož máme množinu úhlů předních kol reprezentovanou pěti hodnotami a rychlost pohybu dvěma hodnotami, budeme v každé iteraci testovat celkově deset hodnot vstupních veličin.

4.6 Volba metriky

V každé iteraci algoritmu RRT dochází několikrát k výpočtu vzdáleností určitých konfigurací (viz řádky 4 a 10 v Algoritmu 4.1). Je třeba zvolit metriku, na základě které budou tyto vzdálenosti určovány. V této práci je uvažována Eukleidovská vzdálenost bodů určujících polohu automobilu, pro dvě konfigurace $\boldsymbol{q}_1 = \begin{bmatrix} x_1 & y_1 & \theta_{01} \end{bmatrix}^T$ a $\boldsymbol{q}_2 = \begin{bmatrix} x_2 & y_2 & \theta_{02} \end{bmatrix}^T$ je tedy vzdálenost ρ určena takto:

$$\rho(\boldsymbol{q}_1, \boldsymbol{q}_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{4.12}$$

Existují však též další možnosti volby funkce $\rho(\boldsymbol{q}_1, \boldsymbol{q}_2)$, například tzv. Manhattanská metrika [10].

4.7 Úloha hledání nejbližšího bodu

Pro hledání nejbližší konfigurace ve stromě byla určena metrika v podobě Eukleidovské vzdálenosti bodů reprezentujících polohu automobilu (viz výše). Dostáváme tedy úlohu nalezení bodu $\boldsymbol{P}^*$, který má ze zadané množiny $B = \{\boldsymbol{P}_i \in \mathbb{R}^2; i \in \{1, 2, ..., n\}\}$ nejmenší Eukleidovskou vzdálenost od určitého bodu $\boldsymbol{P}$.

Intuitivním řešením je spočítat vzdálenost bodu $\boldsymbol{P}$ od všech bodů $\boldsymbol{P}_i \in B$, a poté vybrat bod $\boldsymbol{P}^* \in B$, pro který je tato vzdálenost minimální. Takovýto jednoduchý algoritmus má však kvadratickou složitost a pro rozsáhlejší množiny B trvá výpočet poměrně dlouho. Lepším řešením je hledání nejbližšího souseda pomocí tzv. čtyřstromu (angl. *Quadtree*), neboť tento algoritmus poskytuje složitost $\mathcal{O}(n \log n)$ [11].

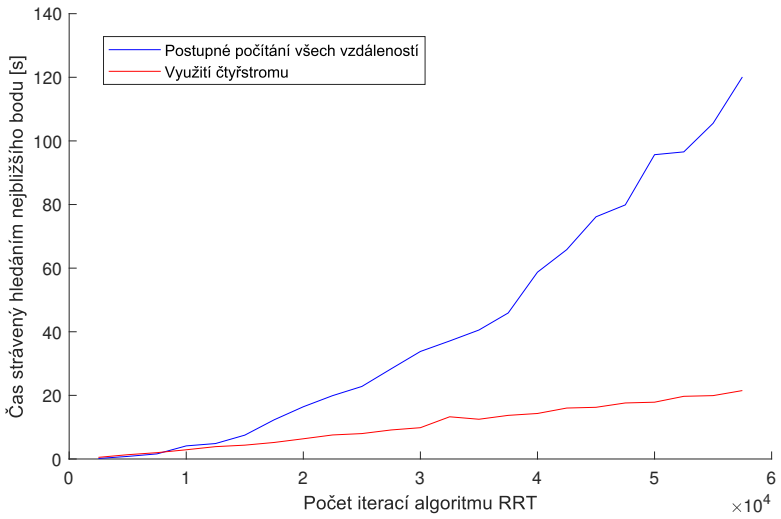
Zmíněný algoritmus využívá ke hledání strom, jehož uzly reprezentují čtverce v prostoru $\mathbb{R}^2$, přičemž každý vnitřní uzel má právě čtyři potomky. Každý z uzlů, pro které platí, že v příslušném čtverci leží některý bod z množiny B, má tzv.

reprezentanta, čímž je právě jeden z bodů, které v tomto čtverci leží. Strom je navíc konstruován tak, že každý bod z množiny B je reprezentantem právě jednoho listového uzlu.

Tento strom lze budovat iterativně. V každé iteraci algoritmu RRT je v případě úspěšné simulace nalezena nová konfigurace q_{new} (viz řádek 12 algoritmu 4.1) a příslušná poloha automobilu může být ihned vložena do čtyřstromu.

Vložení bodu $\boldsymbol{P}_{new}$ do čtyřstromu je popsáno Algoritmem 4.2. Hledání nejbližšího bodu ve čtyřstromu pak využívá skutečnosti, že není třeba prohledávat všechny části stromu. Nachází-li se např. reprezentant kořenového uzlu ve vzdálenosti d od bodu $\boldsymbol{P}$, pak není nutné prohledávat podstromy, jejichž kořenový uzel reprezentuje čtverec nacházející se ve vzdálenosti větší než d od bodu $\boldsymbol{P}$. Přesný postup prohledávání čtyřstromu je uveden v [11].

Byl proveden experiment, při němž byl měřen čas, který algoritmus RRT stráví hledáním nejbližšího bodu ve stromě, v závislosti na počtu iterací algoritmu RRT. Výsledek experimentu je vykreslen na Obr. 4.5. Z grafu je zřejmé, že využití čtyřstromu je pro vyšší počet provedených iterací algoritmu RRT znatelně výhodnější.



Obrázek 4.5: Srovnání časové náročnosti algoritmů pro hledání nejbližšího bodu ve stromě RRT

4.8 Simulace pohybu robotu

Algoritmus RRT odhaduje vývoj konfigurace robotu na základě prováděné simulace. Je tedy třeba navrhnout vhodnou simulační techniku, která pro zadanou konfiguraci $\boldsymbol{q}(T)$ a vstup $\boldsymbol{u}$ nalezne konfiguraci $\boldsymbol{q}(T+\tau)$, do které se robot dostane za dobu τ při aplikování konstantního vstupu $\boldsymbol{u}$. Jelikož známe matematický model robotu ve tvaru $\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u})$ (4.9), lze na tento problém použít libovolnou metodu pro numerické řešení diferenciálních rovnic.

Algoritmus 4.2: Přidání bodu $\boldsymbol{P}_{new}$ do čtyřstromu *tree*

```
1  if Strom tree je prázdný then
2  |  Přidej do tree nejmenší čtverec obsahující celé prostředí W s
   |   reprezentantem P_new
3  else
4  |  u ← kořen stromu tree
5  |  while u není listový uzel stromu tree do
6  |  |  u ← potomek uzlu u, ve kterém leží bod P_new
7  |  end
8  |  if Uzel u nemá reprezentanta then
9  |  |  Prohlaš P_new reprezentantem uzlu u
10 |  else
11 |  |  if Vzdálenost P_new od reprezentanta čtverce u je menší než ε then
12 |  |  |  return
13 |  |  else
14 |  |  |  R ← reprezentant čtverce u
15 |  |  |  u_1 ← u
16 |  |  |  u_2 ← u
17 |  |  |  while u_1 == u_2 do
18 |  |  |  |  Rozděl u_1 na čtyři shodně velké čtverce
19 |  |  |  |  u_1 ← nově vzniklý čtverec, ve kterém leží P_new
20 |  |  |  |  u_2 ← nově vzniklý čtverec, ve kterém leží R
21 |  |  |  |  Prohlaš P_new reprezentantem čtverce u_1
22 |  |  |  |  Prohlaš R reprezentantem čtverce u_2
23 |  |  |  end
24 |  |  end
25 |  end
26 end
```

[9] navrhuje použití Eulerovy metody, tj. metody ve tvaru

$$\boldsymbol{q}(t+\tau) = \boldsymbol{q}(t) + \tau \cdot \boldsymbol{f}\Big(\boldsymbol{q}(t), \boldsymbol{u}\Big), \tag{4.13}$$

zároveň však uvádí, že v praxi bývá často dávána přednost metodám vyššího řádu, např. metodám Runge-Kuttova typu. V této práci byla použita Heunova metoda, která je dána předpisem

$$\boldsymbol{q}(t+\tau) = \boldsymbol{q}(t) + \frac{\tau}{2}\left[\boldsymbol{f}\Big(\boldsymbol{q}(t), \boldsymbol{u}\Big) + \boldsymbol{f}\Big(\boldsymbol{q}(t) + \tau \cdot \boldsymbol{f}(\boldsymbol{q}(t), \boldsymbol{u}), \boldsymbol{u}\Big)\right] \tag{4.14}$$

Aplikací tohoto vztahu na stavový model (4.9) obdržíme metodu ve tvaru

$$\begin{aligned} x(t+\tau) &= x(t) + \frac{\tau}{2}\left[u_1 \cos\theta_0(t) + u_1 \cos\left(\theta_0(t) + \frac{u_1\tau}{d_0}\mathrm{tg}(\phi_0(t))\right)\right] \\ y(t+\tau) &= y(t) + \frac{\tau}{2}\left[u_1 \sin\theta_0(t) + u_1 \sin\left(\theta_0(t) + \frac{u_1\tau}{d_0}\mathrm{tg}(\phi_0(t))\right)\right] \\ \theta_0(t+\tau) &= \theta_0(t) + \frac{u_1\tau}{d_0}\mathrm{tg}(\phi_0(t)) \end{aligned} \tag{4.15}$$

4.9 Testování volnosti cesty

Uvažujme, že byla algoritmem RRT nalezena trajektorie z konfigurace $\boldsymbol{q}_{near}$ do konfigurace $\boldsymbol{q}_{new}$. Aby byl koncový bod $\boldsymbol{q}_{new}$ vložen do stromu, je nutné nejprve otestovat, zda se do tohoto bodu robot skutečně může dostat, tj. zda při pohybu po nalezené trajektorii nenarazí do některé z překážek (viz řádek 11 Algoritmu 4.1).

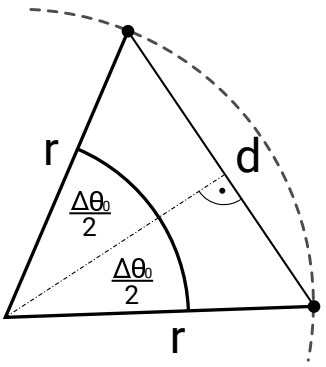
Předpokládejme, že dokážeme spočítat vzdálenost robotu v dané konfiguraci $\boldsymbol{q}$ od nejbližší překážky, označme tuto vzdálenost $D(\boldsymbol{q})$ (postup pro výpočet této vzdálenosti bude popsán dále v této kapitole). Spočítáme-li vzdálenost robotu nacházejícího se v počáteční konfiguraci od překážek $D(\boldsymbol{q}_{near})$, pak víme, že se libovolný bod robotu může posunout až o vzdálenost $D(\boldsymbol{q}_{near})$, aniž by došlo ke kolizi s některou z překážek. Můžeme tedy pomocí Heunovy metody simulovat pohyb robotu o vzdálenost $D(\boldsymbol{q}_{near})$, čímž obdržíme novou konfiguraci $\boldsymbol{q}'$. Spočítáme opět vzdálenost robotu od překážek $D(\boldsymbol{q}')$ a simulujeme pohyb o tuto vzdálenost.

Tento postup lze opakovat, dokud celková doba simulace nepřekročí simulační krok τ. Pokud je simulace úspěšná, lze prohlásit, že při pohybu po uvažované trajektorii nedojde ke kolizi s překážkami. Vyjde-li však pro některou z konfigurací vzdálenost od překážek menší než zadaný práh $\delta > 0$, prohlásíme, že příslušná cesta není volná.

Nabízí se otázka, jak určit simulační krok τ', aby se robot posunul maximálně o vzdálenost $D(\boldsymbol{q})$. Výpočet simulačního kroku se opírá o [4].

Uvažujme, že robot vykonává pouze translační pohyb. Potom se každý jeho bod za čas τ' posune o vzdálenost $\sqrt{\Delta x^2 + \Delta y^2}$, kde $\Delta x = x(t+\tau') - x(t)$ a $\Delta y = y(t+\tau') - y(t)$ reprezentují posun středu zadní nápravy automobilu v jednotlivých souřadnicích.

Dochází-li naopak pouze k rotačnímu pohybu, pak se o největší vzdálenost posune bod, který se nachází nejdále od středu otáčení (za střed otáčení považujeme střed zadní nápravy R_0). Nachází-li se tento bod ve vzdálenosti r od středu otáčení, pak při otočení o úhel $\Delta\theta_0$ urazí vzdálenost $2r|\sin\frac{\Delta\theta_0}{2}|$, jak je patrné z Obr. 4.6.



Obrázek 4.6: Odvozování vzdálenosti uražené při rotačním pohybu

Pomocí odvozených vztahů lze říci, že robot nenarazí do překážky, jestliže platí

$$\sqrt{\Delta x^2 + \Delta y^2} + 2r\Big|\sin\frac{\Delta\theta_0}{2}\Big| \leq D\Big(\boldsymbol{q}(t)\Big), \tag{4.16}$$

kde $\boldsymbol{q}(t)$ je konfigurace, ve které začíná simulace. Poznamenejme, že vztah (4.16) je postačující, nikoliv nutnou podmínkou toho, že nedojde k nárazu do překážky. Vyjádříme-li přírůstky Δx, Δy a $\Delta\theta_0$ pomocí Heunovy metody (4.15), obdržíme vztah

$$|u_1|\tau'\sqrt{1+\cos\Big(\frac{u_1\tau'}{d_0}\mathrm{tg}\phi_0\Big)} + 2r\Big|\sin\Big(\frac{u_1\tau'}{2d_0}\mathrm{tg}\phi_0\Big)\Big| \leq D\Big(\boldsymbol{q}(t)\Big) \tag{4.17}$$

Z této nerovnice potřebujeme získat simulační krok τ', ovšem analytické řešení této nerovnice je velmi složité. Použijme proto následující zjednodušení:

$$\forall x \in \mathbb{R} : \cos x \leq 1; |\sin x| \leq |x| \tag{4.18}$$

Aplikujeme-li vztahy (4.18) na nerovnici (4.17), obdržíme již snadno řešitelnou nerovnici

$$|u_1|\tau'\sqrt{2} + 2r\Big|\frac{u_1\tau'}{2d_0}\mathrm{tg}\Phi_0\Big| \leq D\Big(\boldsymbol{q}(t)\Big), \tag{4.19}$$

jejíž řešením dostaneme množinu přípustných simulačních kroků:

$$\tau' \leq \frac{D\Big(\boldsymbol{q}(t)\Big)d_0}{|u_1|(r|\mathrm{tg}\phi_0| + d_0\sqrt{2})} \tag{4.20}$$

4.10 Výpočet vzdálenosti robotu od překážek

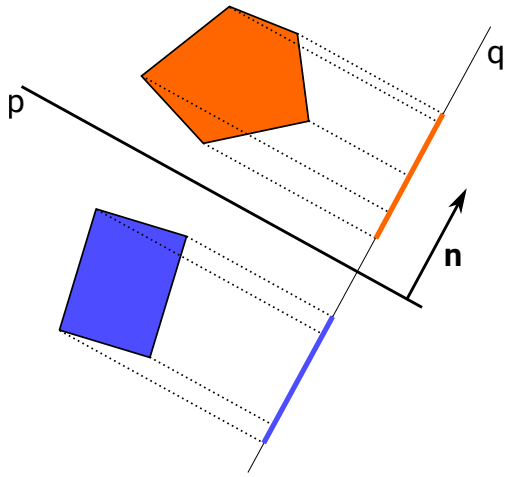
V minulé podkapitole bylo uvažováno, že dokážeme pro libovolnou konfiguraci robotu $\boldsymbol{q}$ spočítat hodnotu funkce $D(\boldsymbol{q})$, která určuje vzdálenost robotu od nejbližší překážky. Tento výpočet však není zcela triviální, a proto bude popsán níže.

Výpočet probíhá ve dvou fázích: Nejprve je prováděn test kolize automobilu s překážkami. Existuje-li průnik robotu s některou z překážek, pak platí $D(\boldsymbol{q}) = 0$, v opačném případě je provedena druhá část výpočtu, která počítá přesnou vzdálenost od překážek.

4.10.1 Test kolize robotu s překážkami

Mějme robot ve tvaru konvexního mnohoúhelníku (v případě automobilu se zpravidla jedná o obdélník) a množinu překážek, které jsou rovněž reprezentovány konvexními mnohoúhelníky (v praxi mohou existovat též nekonvexní překážky, ty však vždy lze rozložit na množinu konvexních mnohoúhelníků). Použitá metoda je podrobně popsána v [12] a zakládá se na skutečnosti, že dva konvexní mnohoúhelníky nemají společný průnik právě tehdy, existuje-li přímka p, která tyto dva mnohoúhelníky odděluje (viz Obrázek 4.7).

Skutečnost, zda daná přímka p mnohoúhelníky odděluje, je testována následujícím způsobem: Je spočítán ortogonální průmět každého vrcholu z obou mnohoúhelníků do přímky q kolmé na přímku p (tento průmět spočítáme jednoduše jako skalární součin $\boldsymbol{x} \cdot \boldsymbol{n}$, kde $\boldsymbol{x}$ je daný vrchol a $\boldsymbol{n}$ je normálový vektor přímky p). Tímto způsobem získáme dva intervaly reprezentující průměty mnohoúhelníků do přímky q (na Obr. 4.7 znázorněny barevnými úsečkami). Jestliže tyto intervaly nemají společný průnik, pak ani celé mnohoúhelníky vzájemně nekolidují.



Obrázek 4.7: Princip testu průniku dvou konvexních mnohoúhelníků

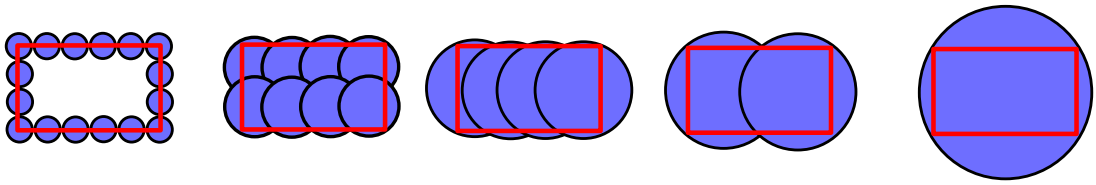
Jestliže existují přímky oddělující dané mnohoúhelníky, pak alespoň jedna z nich je rovnoběžná s některou hranou jednoho z mnohoúhelníků, stačí tedy uvažovat ty vektory $\boldsymbol{n}$, které jsou kolmé na některou z hran. [12]

4.10.2 Výpočet vzdálenosti robotu od překážek

Nedošlo-li ke kolizi robotu s překážkou, je nutné spočítat vzdálenost robotu od nejbližší překážky.

Pro výpočet vzdálenosti dvou mnohoúhelníků lze použít hierarchickou metodu [4] (tato metoda je podrobně popsána v [13] pro výpočet vzdálenosti trojrozměrných objektů, velmi dobře ji však lze použít i v případě mnohoúhelníků v rovině). Tato metoda je založena na skutečnosti, že je velmi jednoduché spočítat vzdálenost dvou kruhů.

Obvod obou mnohoúhelníků je pokryt malými kruhy (viz Obr. 4.8 vlevo) a vzdálenost robotu od překážky je pak aproximována vzdáleností nejbližších kruhů k_1 a k_2, kde k_1 leží na obvodu robotu a k_2 na obvodu překážky. Aby nebylo nutno počítat vzdálenosti všech těchto kruhů, je pro robot i pro každou překážku vystavěn binární strom, jehož uzly odpovídají jednotlivým kruhům, přičemž listové uzly jsou již zmíněné kruhy pokrývající obvod mnohoúhelníku a kořen stromu reprezentuje kruh opsaný celému objektu (viz Obr. 4.8). [13] pak popisuje algoritmus prohledávání těchto stromů, který zajistí, že jsou prohledávány pouze části stromu, které mohou poskytnout menší vzdálenost robotu od překážky nežli tu, která již byla spočtena.

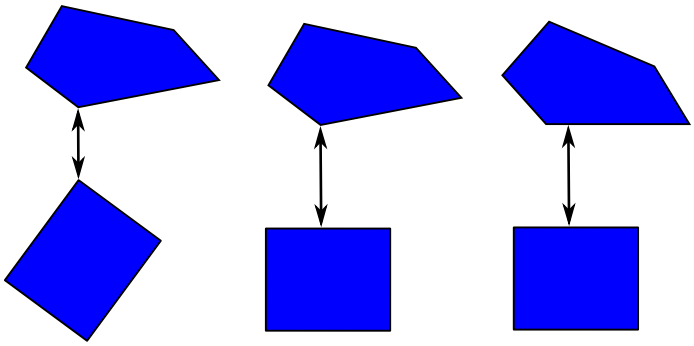


Obrázek 4.8: Aproximace objektu pomocí kruhů

Tento algoritmus je poměrně složitý, jeho nevýhodou navíc je, že vypočtená vzdálenost robotu od překážek je pouze odhadovaná a skutečná vzdálenost je pravděpodobně větší, proto byla v této práci hierarchická metoda nahrazena jiným způsobem výpočtu vzdálenosti robotu od překážek.

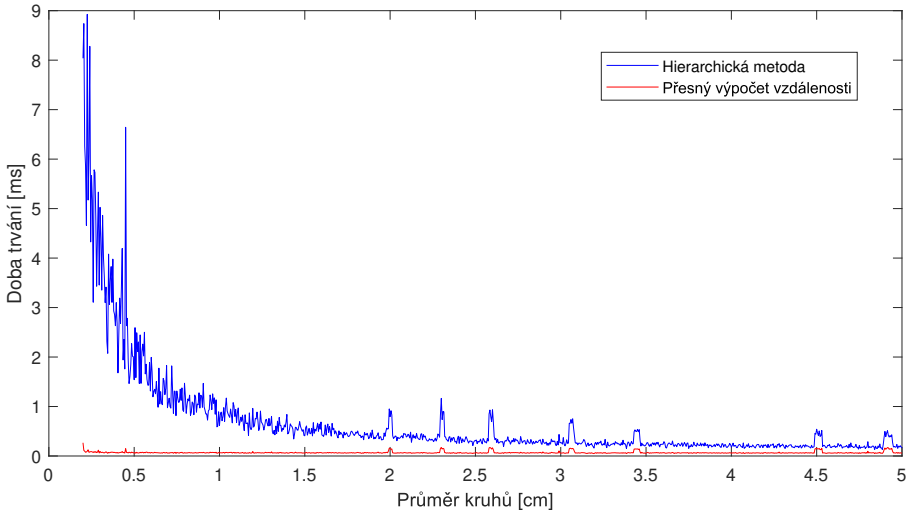
Vzdálenost dvou mnohoůhelníků je vždy rovna vzdálenosti dvou jejich vrcholů, vzdálenosti jednoho vrcholu a jedné hrany či vzdálenosti mezi dvěma rovnoběžnými hranami (viz Obrázek 4.9). Nejprve je tedy spočítána vzdálenost všech dvojic vrcholů, následně vzdálenost všech dvojic vrchol – hrana, kde existuje kolmice na danou hranu, která tuto hranu protíná a zároveň prochází uvažovaným vrcholem, a nakonec jsou též určeny vzdálenosti všech dvojic rovnoběžných hran, pro které existuje kolmice protínající obě tyto hrany. Z vypočítaných vzdáleností je poté určena minimální hodnota, která je označena za vzdálenost těchto dvou mnohoúhelníků.

Byla měřena průměrná doba výpočtu vzdálenosti pomocí hierarchické metody a rovněž pomocí výše popsané metody přesného určování vzdálenosti. Doba výpočtu hierarchické metody závisí na průměru kruhů pokrývajících obvod mnohoúhelníků, při experimentu se však ukázalo, že pro běžné hodnoty tohoto průměru je přesný výpočet vždy rychlejší (viz Obrázek 4.10), jeho hlavní výhodou je navíc již zmíněný fakt, že výsledná vzdálenost je přesná, nikoliv jen odhadovaná jako v případě



Obrázek 4.9: Znázornění vzdálenosti dvou mnohoúhelníků

hierarchické metody.



Obrázek 4.10: Porovnání rychlosti algoritmů pro výpočet vzdálenosti dvou mnohoúhelníků

5 Plánování pohybu pomocí algoritmu CL-RRT

5.1 Algoritmus CL-RRT

V minulé kapitole bylo popsáno plánování trajektorie pomocí algoritmu RRT. Tato metoda plánování využívá znalost matematického modelu robotu, díky čemuž generuje trajektorie, které jsou robotem realizovatelné. Problém nastává, chceme-li naplánovat pohyb pro nestabilní robot, neboť algoritmus RRT aplikoval na robot vždy po částech konstantní vstupy a řídit tímto způsobem nestabilní systém zpravidla není možné, není-li simulační krok velmi krátký.

Řešení přináší modifikovaný algoritmus CL-RRT (*Closed-Loop Rapidly-exploring Random Trees*), který nesimuluje pouze chování robotu samotného, nýbrž simuluje celou uzavřenou smyčku, která je navržena k řízení robotu.

Princip plánování je formou pseudokódu na základě [5] a [6] zapsán v Algoritmu 5.1. Uzly stromu CL-RRT jsou tvořeny uspořádanými dvojicemi $(w, \boldsymbol{q})$, kde $\boldsymbol{q}$ je konfigurace, do které se robot dostane z konfigurace odpovídající rodičovskému uzlu, zadáme-li mu jako referenční bod, ke kterému se má pohybovat, bod w.

Při inicializaci algoritmu je do stromu vložena počáteční konfigurace a počáteční poloha robotu (řádek 1). Následně je v každé iteraci náhodně vygenerován bod $w_{rand} \in W$ a uzly ve stromu jsou seřazeny vzestupně podle ceny cesty vedoucí z příslušného uzlu do bodu w_{rand} (k výpočtu této ceny se využívá heuristika, viz dále). Počínaje od uzlu s nejnižší cenou jsou poté simulovány trajektorie vedoucí do bodu w_{rand} (řádek 6). Je-li trajektorie volná, pak je vložena do stromu (řádek 9) a dále je simulován pohyb z nově získané konfigurace do cílového bodu (řádek 10). Je-li i tato propagace úspěšná, je nově získaná konfigurace rovněž vložena do stromu (řádek 13), a pokud splňuje požadavky na cílovou konfiguraci, pak je algoritmus ukončen, neboť byla nalezena hledaná trajektorie.

5.2 Simulace pohybu a test volnosti trajektorie

Mějme stavový model $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, u)$ popisující uzavřenou smyčku, tj. model robotu a zároveň veškerých řídicích algoritmů, kde vstupem u zpravidla rozumíme cestu, kterou má robot sledovat. Pro naplánování trajektorie je nutné dokázat tento systém simulovat (viz Algoritmus 5.1, řádky 6 a 10) a otestovat, zda simulovaná trajektorie leží ve volné části konfiguračního prostoru Q_{free} (řádky 8 a 12).

K tomuto účelu využijeme opět Heunovu metodu (4.14), musíme však určit simulační krok τ tak, aby při pohybu po simulované trajektorii nedošlo ke kolizi s překážkami. V Kapitole 4 byl tento problém vyřešen explicitním vyjádřením času τ ze stavového

Algoritmus 5.1: CL-RRT

```
1  CL-RRT ← (počáteční konfigurace, počáteční poloha robotu)
2  for i in {1, 2, ..., n} do
3      w_rand ← náhodný bod v prostředí ℝ²
4      Q_sorted ← pomocí heuristiky seřaď uzly stromu CL-RRT vzestupně
         podle ceny expandování do bodu w_rand
5      for (q_poc, w_poc) in Q_sorted do
6          σ ← trajektorie, po které by se robot s počáteční konfigurací q_poc
             pohyboval při sledování cesty w_poc w_rand
7          q_new ← konfigurace na konci trajektorie σ
8          if Trajektorie σ je volná then
9              Vlož do stromu CL-RRT dvojici (q_new, w_rand) jako potomka uzlu
                 (q_poc, w_poc)
10             σ_cil ← trajektorie, po které by se robot s počáteční konfigurací
                 q_new pohyboval při sledování cesty vedoucí z w_poc do cílové
                 polohy
11             q_goal ← konfigurace na konci trajektorie σ_cil
12             if Trajektorie σ_cil je volná then
13                 Vlož do stromu CL-RRT dvojici (q_goal, cílová poloha) jako
                     potomka uzlu (q_new, w_rand)
14                 if q_goal splňuje požadavky na cílovou konfiguraci then
15                     return
16                 end
17             end
18             break
19         end
20     end
21 end
```

modelu robotu, v případě uzavřené smyčky je však analytické vyjádření simulačního kroku velmi složité, proto bylo přistoupeno k experimentální volbě simulačního kroku.

Zvolíme-li maximální simulační krok τ_{max} a maximální počet dělení toho kroku, můžeme k simulaci uzavřené smyčky a ověřování, zda uvažovaná cesta neprochází přes překážky, použít Algoritmus 5.2. Tento algoritmus spočítá vzdálenost robotu od překážek, a poté se pokouší provést simulaci uzavřené smyčky s maximálním simulačním krokem. Pokud se během této simulace robot posune o větší vzdálenost, než je vzdálenost nejbližší překážky v počátečním bodě, pak je simulační krok vydělen dvěma a simulace je provedena znovu. Maximální počet rozdělení simulačního kroku je definován jako parametr algoritmu.

Pokud i při použití minimálního simulačního kroku převyšuje uražená vzdálenost vzdálenost od nejbližší překážky, pak algoritmus prohlásí, že simulovaná trajektorie není volná (viz řádek 11). V opačném případě pokračuje simulace dalším simulačním krokem. Nedojde-li ke změně konfiguračního vektoru, znamená to, že robot dorazil

do cílové konfigurace a již se nepohybuje, algoritmus tedy končí úspěchem (řádek 14).

Algoritmus 5.2: Simulování uzavřené smyčky

```
1  x_1 ← počáteční stav uzavřené smyčky
2  while True do
3  |  D ← vzdálenost robotu od nejbližší překážky, nachází-li se uzavřená
   |   smyčka ve stavu x_1
4  |  for i in {0, 1, 2, ..., maxPocetDeleni} do
5  |  |  x_2 ← simuluj uzavřenou smyčku z bodu x_1 s krokem τ_max / 2^i
6  |  |  if Robot mezi stavy x_1 a x_2 neurazil vzdálenost větší než D then
7  |  |  |  break
8  |  |  end
9  |  end
10 |  if Robot mezi stavy x_1 a x_2 urazil vzdálenost větší než D then
11 |  |  return Trajektorie není volná
12 |  end
13 |  if Během simulace nedošlo ke změně konfigurace robotu then
14 |  |  return x_2 (Trajektorie je volná)
15 |  else
16 |  |  x_1 ← x_2
17 |  end
18 end
```

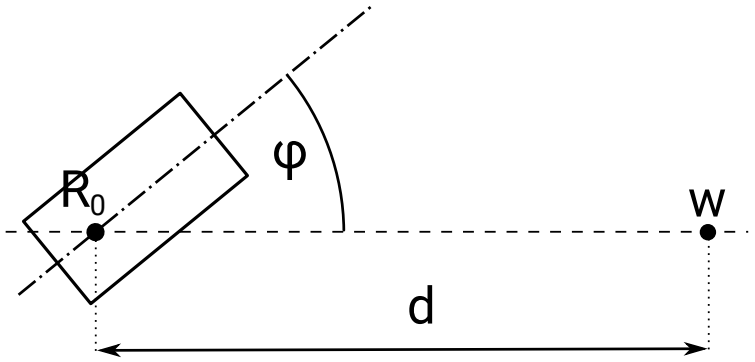
5.3 Heuristika

Po zvolení náhodného bodu $w_{rand} \in W$ je nutné seřadit konfigurace ve stromu CL-RRT vzestupně podle ceny cesty vedoucí z bodu odpovídajícího dané konfiguraci do bodu w_{rand} (viz Algoritmus 5.1, řádek 4). V případě algoritmu RRT byla konfigurace nejvhodnější k propagaci vybírána pouze na základě Eukleidovské vzdálenosti příslušné polohy robotu od bodu w_{rand}, to však není ideální, neboť délka a náročnost cesty nezávisí pouze na počáteční poloze robotu, ale také na ostatních konfiguračních souřadnicích.

Definujme tedy heuristickou funkci $H(\boldsymbol{q}, w)$, která určuje cenu cesty vedoucí z konfigurace $\boldsymbol{q}$ do bodu w. Tato funkce byla zvolena jednoduše jako délka dané cesty, může však hodnotit např. také hladkost cesty [6].

V případě automobilu např. závisí délka cesty pouze na vzdálenosti bodu w a na počáteční orientaci automobilu vzhledem ke spojnici s bodem w (viz Obr. 5.1), dostáváme tedy funkci $H = H(d, \varphi)$. Jak uvádí [4], výpočet této funkce je dosti časově náročný, proto jsou hodnoty funkce simulačně spočítány v určitých bodech offline, a následně je funkční hodnota v libovolném bodě odhadována metodou *bilineární interpolace*.

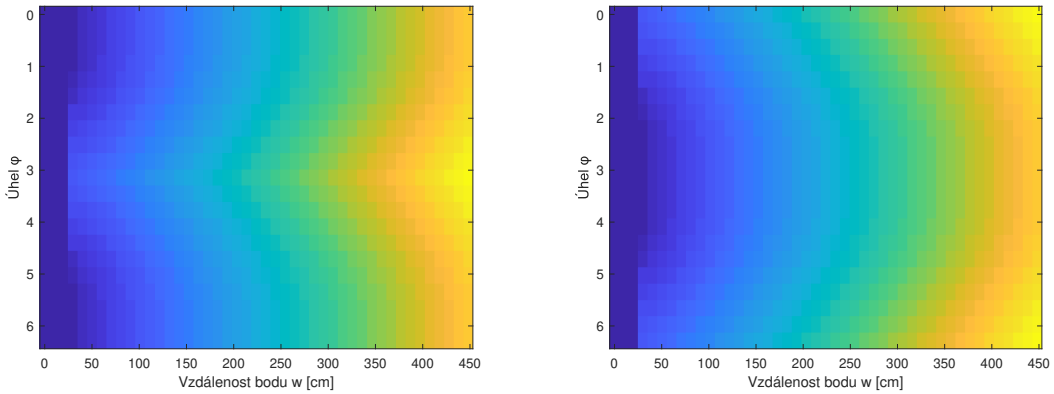
Ve skutečnosti byly počítány hodnoty dvou funkcí $H_{vpred}(d, \varphi)$ a $H_{vzad}(d, \varphi)$, které



Obrázek 5.1: Určování heuristické funkce pro automobil

stanovují cenu cesty při pohybu robotu vřed či při pohybu vzad. Výsledná hodnota funkce $H(d, \varphi)$ je pak dána minimem z těchto dvou funkčních hodnot. Výhodou tohoto přístupu je, že uživatel může definovat omezení, že se robot smí po nalezené cestě pohybovat pouze směrem vpřed či pouze směrem vzad. Pak lze za hodnotu heuristické funkce považovat přímo hodnotu $H_{vpred}(d, \varphi)$, resp. $H_{vzad}(d, \varphi)$.

Příklad simulačně nalezené heuristické funkce pro pohyb automobilu vpřed a vzad je zobrazen na Obr. 5.2.



Obrázek 5.2: Ukázka heuristické funkce hodnotící cenu cesty při pohybu automobilu vpřed (vlevo) a vzad (vpravo)

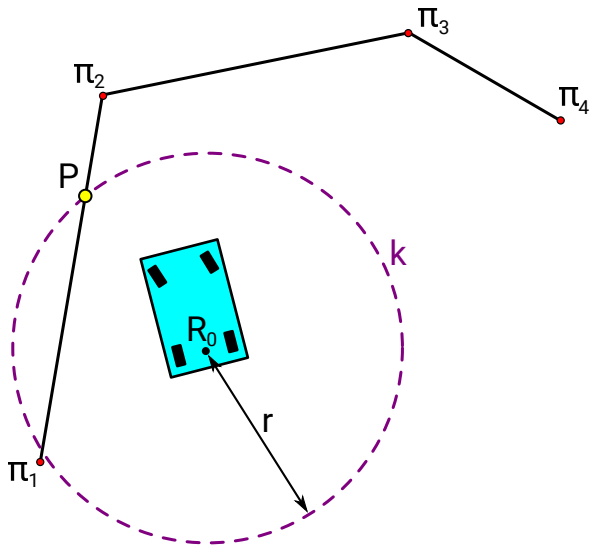
6 Uzavřená smyčka pro řízení automobilu

Ačkoliv je algoritmus CL-RRT určen primárně pro plánování pohybu nestabilních robotů, lze jej úspěšně použít i v případě stabilních robotů. V rámci této práce byl algoritmus implementován pro úlohu plánování pohybu automobilu.

Jak bylo popsáno v Kapitole 5, při plánování pohybu pomocí algoritmu CL-RRT je nutné znát matematický model celé uzavřené smyčky. Model samotného automobilu (4.9) byl již odvozen, je však třeba popsat řídicí algoritmus generující řídicí veličiny $u_1(t)$ a $\phi_0(t)$. Analýzou uzavřené smyčky se zabývá tato kapitola.

6.1 Strategie přímého pronásledování

Strategie přímého pronásledování (*Pure Pursuit Control*) je intuitivní algoritmus, který řídí robot tak, aby se pohyboval po požadované cestě. Uvažujme, že požadovaná cesta Π je tvořena posloupností bodů $\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, ..., \boldsymbol{\pi}_n$. Základním principem tohoto algoritmu je, že automobil v každém časovém okamžiku pronásleduje průsečík referenční cesty Π a kružnice k, jejíž střed je umístěn ve středu zadní nápravy automobilu (viz Obrázek 6.1) .



Obrázek 6.1: Princip fungování strategie přímého pronásledování

Pronásledování průsečíku P spočívá ve vypočtení úhlu předních kol ϕ_0, který musí být aplikován, aby robot po určitém čase dojel do bodu P za předpokladu, že by se tento bod nepohyboval. Vztah pro výpočet tohoto úhlu je podrobně odvozen v [5].

Existuje však ještě intuitivnější způsob pronásledování bodu P: Nastavme v každém časovém okamžiku úhel natočení předních kol tak, aby kola směřovala přímo

směrem k bodu P (viz Obr. 6.2). Pohybuje-li se robot pozadu, pak spočítáme požadovanou orientaci automobilu, při které by zadní část vozidla směřovala k bodu P. Jelikož však orientace automobilu není ovladatelným vstupem robotu, regulujeme ji na požadovanou hodnotu pomocí PI regulátoru. Tento způsob sledování cesty je implementován v Algoritmu 6.1.

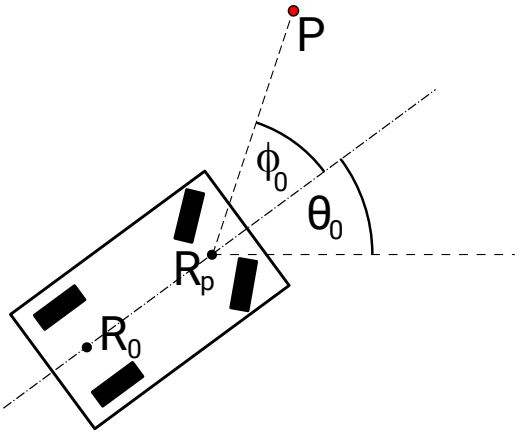
Algoritmus 6.1: Strategie přímého pronásledování

```
for i in 1, 2, ..., n − 1 do
    while True do
        usecka ← úsečka π_i π_{i+1}
        if Neexistuje průsečík úsečky usecka a kružnice k then
            usecka ← úsečka R_0 π_{i+1}
        end
        P ← průsečík úsečky usecka s kružnicí k, který se nachází blíže k
          bodu π_{i+1}
        if Bod P leží za bodem π_{i+1} then
            break
        else
            if Bod P leží před bodem R_0 then
                φ_0 ← úhel předních kol, pro který kola směřují k bodu P
                Zapiš φ_0 na vstup robotu jako úhel natočení předních kol
                Zapiš +1 na vstup robotu jako směr jízdy
            else
                θ_0^* ← požadovaná orientace automobilu, při níž zadní část
                  vozidla směřuje k bodu P
                Zapiš θ_0^* jako referenční hodnotu PI regulátoru
                Zapiš výstup PI regulátoru na vstup robotu jako úhel
                  natočení předních kol
                Zapiš −1 na vstup robotu jako směr jízdy
            end
        end
    end
end
Zapiš 0 na vstup robotu jako směr jízdy
```

Výpočet úhlu předních kol při pohybu vpřed lze odvodit s využitím schématu znázorněného na Obr. 6.2. Nechť $R_p = \begin{bmatrix} r_{px} & r_{py} \end{bmatrix}^T$ je střed přední nápravy automobilu, dále označme $P = \begin{bmatrix} p_x & p_y \end{bmatrix}^T$. Zřejmě platí

$$\begin{aligned} \phi_0 + \theta_0 &= \operatorname{atan2}(p_y - r_{py}, p_x - r_{px}) \\ \phi_0 &= \operatorname{atan2}(p_y - r_{py}, p_x - r_{px}) - \theta_0 \end{aligned} \tag{6.1}$$

Poznamenejme, že funkci arctg(...) v tomto případě nelze použít, neboť v tom případě by vztah platil pouze pro $\phi_0 + \theta_0 \in \langle -\frac{\pi}{2}, \frac{\pi}{2} \rangle$. Oproti tomu atan2(...) je rozšířením funkce arctg(...) pro interval $\langle -\pi, \pi \rangle$ [5].



Obrázek 6.2: Výpočet úhlu natočení předních kol

Uveďme dále, že poloměr kružnice k může být obecně funkcí rychlosti pohybu robotu, jelikož však v úloze plánování uvažujeme konstantní rychlost pohybu, pak můžeme poloměr kružnice k chápat jako parametr algoritmu přímého pronásledování.

Nabízí se otázka, zda je výhodnější použití původního algoritmu přímého pronásledování, který je popsán v [5], či uvedená modifikace spočívající v přímém natočení předních kol k pronásledovanému bodu. Zásadní rozdíl spočívá v tom, že v případě původního algoritmu přijíždí robot do pronásledovaného bodu po kružnici, zatímco ve druhém zmíněném algoritmu probíhá pohyb po přímce.

Byl proveden experiment, při němž byl z různých počátečních konfigurací simulován pohyb robotu po zadané úsečce při využití obou zmíněných postupů. Při každém experimentu byla spočtena hodnota následujících kritérií:

$$K_1 = \int_0^{+\infty} \left| o\Big(\boldsymbol{q}(t)\Big) \right| dt \qquad K_2 = \int_0^{+\infty} |\dot{\phi}_0(t)| dt, \tag{6.2}$$

kde $o\Big(\boldsymbol{q}(t)\Big)$ reprezentuje odchylku robotu od referenční cesty v čase t. Ve všech testovaných případech vyšlo kritérium K_1 vyšší pro původní algoritmus přímého pronásledování, zatímco vyšší hodnota kritéria K_2 byla obdržena pro modifikovaný algoritmus. Při použití modifikovaného algoritmu se tedy robot pohybuje blíže referenční cestě, dochází však k rychlejším změnám úhlu natočení předních kol.

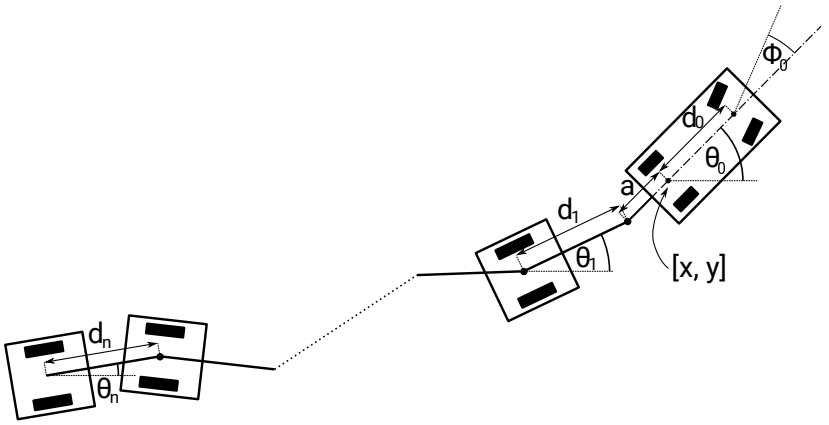
6.2 Regulátor orientace automobilu

Pro řízení úhlu natočení automobilu v režimu couvání je použit jednoduchý PI regulátor. Jelikož existují omezení na akční veličinu ϕ_0, je výstup regulátoru saturován a z toho důvodu je vhodné přidat ošetření unášení integrační složky.

7 Uzavřená smyčka pro řízení kloubového vozidla s n přívěsy

7.1 Matematický model kloubového vozidla

Uvažujme kloubové vozidlo složené z tahače a n přívěsů označované zkratkou TnT (*Truck with n trailers*). Schéma tohoto robotu je znázorněno na Obrázku 7.1.



Obrázek 7.1: Schéma kloubového vozidla TnT

Matematický model tohoto systému je odvozen v [8]. Zavedeme-li úhly mezi osami jednotlivých částí vozidla ve tvaru $\phi_i = \theta_{i-1} - \theta_i$ a úhel mezi osou tahače a osou posledního přívěsu $\phi = \sum_{i=1}^{n} \phi_i$, lze robot popsat následujícím stavovým modelem:

$$
\begin{aligned}
\dot{x} =& u_1 \cos\theta_0 \\
\dot{y} =& u_1 \sin\theta_0 \\
\dot{\theta_0} =& \frac{u_1}{d_0}\mathrm{tg}\phi_0 \\
\dot{\phi_1} =& \frac{u_1}{d_0}\mathrm{tg}\phi_0 - \frac{u_1}{d_1}\Big[\sin\phi_1 - \frac{a}{d_0}\cos\phi_1\mathrm{tg}\phi_0\Big] \\
\dot{\phi_2} =& \frac{u_1}{d_1}\Big[\sin\phi_1 - \frac{a}{d_0}\cos\phi_1\mathrm{tg}\phi_0\Big] - \frac{u_1}{d_2}\sin\phi_2\Big[\cos\phi_1 + \frac{a}{d_0}\sin\phi_1\mathrm{tg}\phi_0\Big] \\
\dot{\phi_3} =& u_1\Big[\frac{1}{d_2}\sin\phi_2 - \frac{1}{d_3}\sin\phi_3\cos\phi_2\Big]\Big[\cos\phi_1 + \frac{a}{d_0}\sin\phi_1\mathrm{tg}\phi_0\Big] \\
\dot{\phi_4} =& u_1\Big[\frac{1}{d_3}\sin\phi_3\cos\phi_2 - \frac{1}{d_4}\sin\phi_4\cos\phi_2\cos\phi_3\Big]\Big[\cos\phi_1 + \frac{a}{d_0}\sin\phi_1\mathrm{tg}\phi_0\Big] \\
& \ldots \\
\dot{\phi_n} =& u_1\Big[\frac{1}{d_{n-1}}\sin\phi_{n-1}\cos\phi_2\cos\phi_3\ldots\cos\phi_{n-2} - \frac{1}{d_n}\sin\phi_n\cos\phi_2\cos\phi_3\ldots\cos\phi_{n-1}\Big]\cdot \\
& \cdot\Big[\cos\phi_1 + \frac{a}{d_0}\sin\phi_1\mathrm{tg}\phi_0\Big]
\end{aligned}
\tag{7.1}
$$

7.2 Modifikovaný algoritmus přímého pronásledování

Pro sledování referenční cesty použijme opět modifikovaný algoritmus přímého pronásledování, který byl popsán v Kapitole 6, je však nutné tento algoritmus přizpůsobit kloubovému vozidlu.

Podstatnou modifikací je rozhodování, kterým směrem se bude robot pohybovat. Toto rozhodování probíhá na základě vzdálenosti automobilu a posledního přívěsu od konce úsečky, po které se robot pohybuje. Nachází-li se střed zadní nápravy automobilu k tomuto bodu blíže než střed nápravy posledního přívěsu, pak se robot pohybuje vpřed, v opačném případě je zvolen pohyb vzad.

Je-li zvolen pohyb vpřed, probíhá řízení stejně jako v případě automobilu – přední kola jsou natočena směrem k pronásledovanému bodu P (viz Kapitola 6). Pohybuje-li se však robot dozadu, je nutné nasměrovat poslední n-tý přívěs tak, aby směřoval k bodu P. Určíme-li tedy požadovanou orientaci tohoto přívěsu θ_n^*, pak požadujeme splnění rovnice

$$\theta_n \stackrel{!}{=} \theta_n^* \tag{7.2}$$

Jelikož úhly ϕ_i jsou definovány vztahem $\phi_i = \theta_{i-1} - \theta_i$, platí $\theta_i = \theta_{i-1} - \phi_i$ a lze postupně odvodit:

$$\begin{aligned} \theta_n &= \theta_{n-1} - \phi_n = \theta_{n-2} - \phi_{n-1} - \phi_n = ... = \\ &= \theta_1 - \phi_2 - \phi_3 - ... - \phi_n = \theta_0 - \sum_{i=1}^{n} \phi_n = \theta_0 - \phi \end{aligned} \tag{7.3}$$

Po dosazení vztahu (7.3) do (7.2) dostáváme požadavek ve tvaru

$$\phi \stackrel{!}{=} \theta_0 - \theta_n^* \tag{7.4}$$

Úhel mezi osou automobilu a osou zadního přívěsu ϕ je řízen zpětnovazebním regulátorem, jeho referenční hodnotou tedy bude $\phi_{sp} = \theta_0 - \theta_n^*$.

7.3 Regulátor úhlu mezi automobilem a posledním přívěsem

Algoritmus přímého pronásledování poskytuje v případě couvání požadovanou hodnotu úhlu ϕ, který svírá osa tahače s osou posledního přívěsu (viz výše). Je třeba implementovat zpětnovazební regulátor, který bude na základě měřených úhlů mezi částmi vozidla $\phi_i, i \in \{1, 2, ..., n\}$ generovat akční veličinu ϕ_0 tak, aby úhel ϕ odpovídal požadované hodnotě, tj. aby robot couval pod správným úhlem.

V [8] je ukázáno, že při pohybu vzad je robot nestabilní, což činí úlohu regulace složitější. Popsaný regulátor je navržen na základě [8].

Pro řešení úlohy regulace úhlu ϕ nejsou informace o poloze a orientaci automobilu potřebné, první tři rovnice stavového modelu (7.1) tedy nemusíme uvažovat. Rozšíříme-li takto redukovaný systém o integrátor úhlu ϕ, obdržíme systém, jehož stavový vektor $\boldsymbol{z}$ má tvar

$$\boldsymbol{z} = \begin{bmatrix} \phi_1 & \phi_2 & \dots & \phi_n & \int_0^t \phi(\tau)d\tau \end{bmatrix}^T \tag{7.5}$$

Uvažujme, že výstupní úhel ϕ regulujeme na požadovanou hodnotu ϕ_{sp}. Pak můžeme s využitím stavového modelu (7.1) určit hodnoty ϕ_0^*, ϕ_1^*, ... ϕ_n^*, které odpovídají hodnotám úhlů ϕ_0, ϕ_1, ... ϕ_n v rovnovážném stavu za podmínky $\phi = \phi_{sp}$. Z těchto veličin sestavme vektor $\boldsymbol{z}^*$ ve tvaru

$$\boldsymbol{z}^* = \begin{bmatrix} \phi_1^* & \phi_2^* & \dots & \phi_n^* & \int_0^t \phi_{sp}(\tau)d\tau \end{bmatrix}^T \tag{7.6}$$

Regulátor úhlu ϕ pak generuje řídicí veličinu ϕ_0 na základě následujícího zákonu řízení:

$$\phi_0 = \boldsymbol{F}^T(\boldsymbol{z} - \boldsymbol{z}^*) + \phi_0^* = \boldsymbol{F}^T\boldsymbol{z} + (\phi_0^* - \boldsymbol{F}^T\boldsymbol{z}^*), \tag{7.7}$$

kde $\boldsymbol{F}$ je vektor obsahující $n+1$ koeficientů. Jedná se tedy o lineární stavový regulátor, který je doplněn o dopřednou vazbu ve tvaru $\phi_0^* - \boldsymbol{F}^T\boldsymbol{z}^*$. Nechť je systém rozšířený o integrátor úhlu ϕ ve stavové reprezentaci zapsán jako

$$\dot{\boldsymbol{z}} = \boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}\phi_0 \tag{7.8}$$

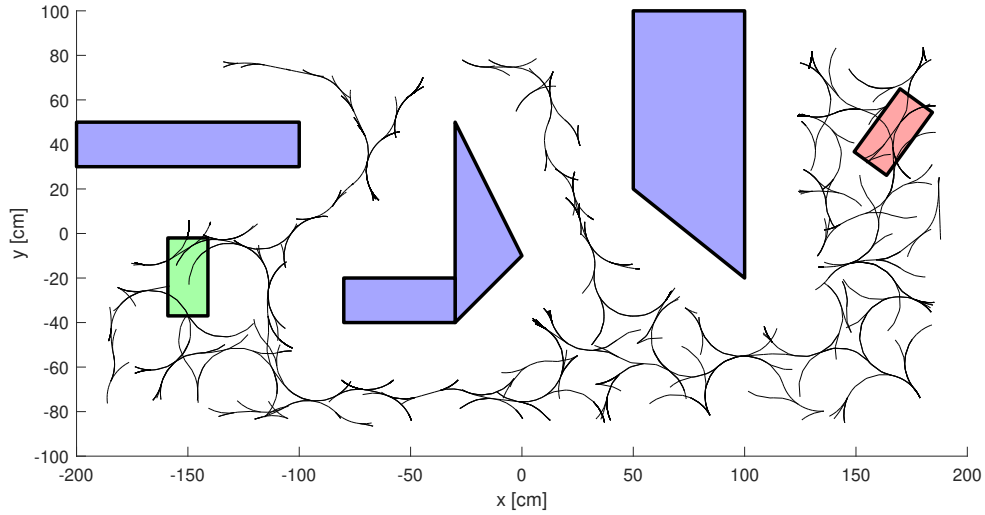
Pak lze prvky vektoru $\boldsymbol{F}$ spočítat přiřazením vlastních čísel matice $\boldsymbol{A} + \boldsymbol{B}\boldsymbol{F}^T$. Stavový regulátor lze navrhnout například tak, aby byla výsledná uzavřená smyčka stabilní a energie akční veličiny $\int_0^{+\infty} \boldsymbol{F}^T\boldsymbol{z}(t)dt$ byla minimální. Jak je uvedeno v [8], v takovém případě jsou vlastní čísla matice $\boldsymbol{A} + \boldsymbol{B}\boldsymbol{F}^T$ zrcadlově symetrická podle imaginární osy k pólům otevřené smyčky.

Jelikož dochází k saturaci akční veličiny ϕ_0, je regulátor ošetřen proti unášení integrační složky. Integrál úhlu ϕ a ϕ_{sp} je navíc z praktických důvodů vynulován při každé změně směru jízdy, aby se na akční veličině při jízdě vzad neprojevovala odchylka integrovaná během jízdy vpřed.

8 Výsledky experimentů

Tato práce se zabývá především plánovacími algoritmy RRT a CL-RRT. Existuje velké množství implementací těchto algoritmů, např. v jazyce Python. V rámci této práce však byla vytvořena vlastní implementace obou algoritmů v softwarovém nástroji MATLAB. Tato kapitola prezentuje výsledky dosažené pomocí těchto algoritmů.

Kapitola 4 byla věnována plánování pohybu pro automobil pomocí algoritmu RRT. Princip algoritmu spočívá v náhodném generování realizovatelných trajektorií. Příklad takto nalezených trajektorií je znázorněn na Obr. 8.1.

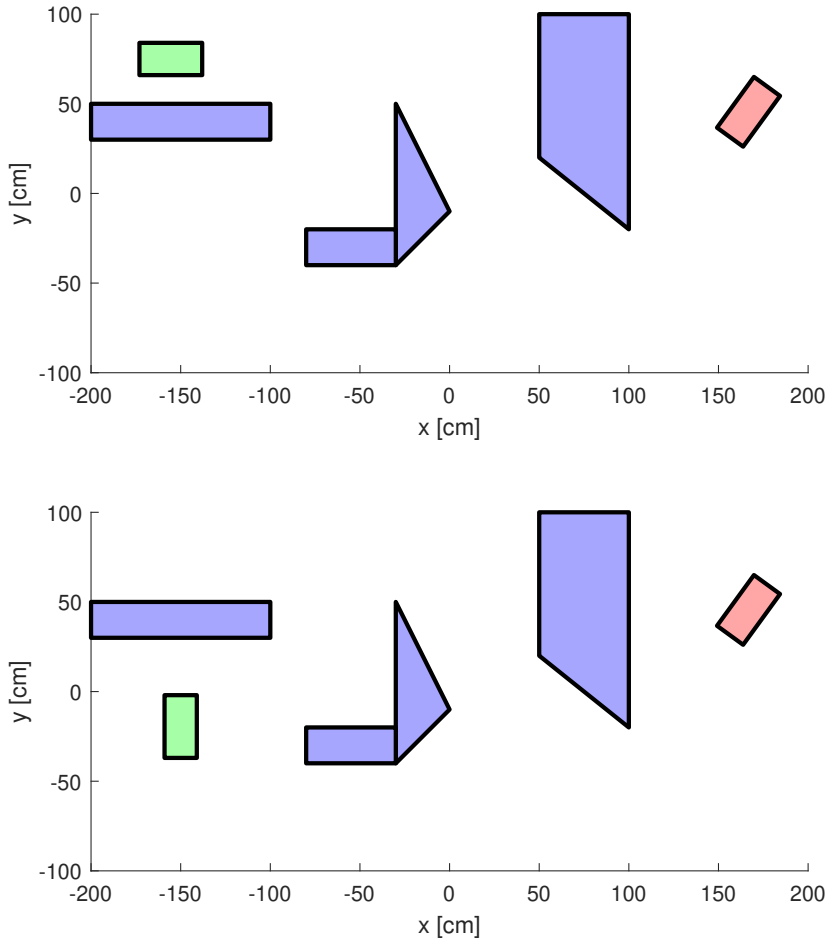


Obrázek 8.1: Ukázka trajektorií testovaných algoritmem RRT. Červenou barvou je vyznačena počáteční konfigurace robotu, zelenou barvou finální konfigurace

Dále bylo popisováno plánování pohybu pro automobil s využitím algoritmu CL-RRT. Zásadní odlišností tohoto přístupu je, že je simulována celá uzavřená smyčka řídící robot.

Porovnejme nyní tyto dvě metody pro plánování pohybu automobilu. Algoritmy byly srovnávány při řešení dvou úloh. Použité prostředí je znázorněno na Obrázku 8.2. Počáteční konfigurace i překážky jsou v obou úlohách nastaveny stejně, cílová konfigurace robotu se však liší. Oběma algoritmům byly tyto dvě situace předloženy k řešení a byl měřen čas výpočtu, délka nalezené cesty a optimalita konfigurace na konci trajektorie (tj. vzdálenost od požadovaného cílového bodu a odchylka orientace automobilu). Pro algoritmus RRT byl zadán maximální počet iterací 50 000, v případě algoritmu CL-RRT byla pro maximální počet iterací zvolena hodnota 10 000. Jelikož RRT i CL-RRT jsou nedeterministické algoritmy, byl každý experiment proveden třicetkrát a získané hodnoty byly následně zprůměrovány, aby byl minimalizován vliv náhody. Výsledky experimentů jsou uvedeny v Tabulce 8.1.

Objektivní porovnání algoritmů RRT a CL-RRT je velice komplikované, neboť jejich



Obrázek 8.2: Prostředí použité při porovnávání algoritmů RRT a CL-RRT. Červenou barvou je vyznačena počáteční konfigurace robotu, zelenou barvou finální konfigurace

Tabulka 8.1: Výsledky porovnávání algoritmů RRT a CL-RRT v úloze plánování pohybu pro automobil

	Úloha 1	Úloha 2
RRT	Úspěšnost: 73% Doba výpočtu: 4,6 s Délka nalezené cesty: 598 cm Odchylka polohy cílové konfigurace: 14,9 cm Odchylka orientace cílové konfigurace: 11,9°	Úspěšnost: 100% Doba výpočtu: 3,5 s Délka nalezené cesty: 511 cm Odchylka polohy cílové konfigurace: 17,2 cm Odchylka orientace cílové konfigurace: 12,4°
CL-RRT	Úspěšnost: 97% Doba výpočtu: 2,9 s Délka nalezené cesty: 526 cm Odchylka polohy cílové konfigurace: 1,3 cm Odchylka orientace cílové konfigurace: 5,9°	Úspěšnost: 100% Doba výpočtu: 1,5 s Délka nalezené cesty: 465 cm Odchylka polohy cílové konfigurace: 6,3 cm Odchylka orientace cílové konfigurace: 11,8°

chování je závislé na mnoha faktorech, jako je délka simulačního kroku, maximální počet iterací apod. Na základě výsledků uvedených v Tabulce 8.1 však lze vyslovit domněnku, že algoritmus CL-RRT funguje lépe, v obou testovaných úlohách totiž

poskytnul lepší výsledek, a to ve všech ohledech – pravděpodobnost nalezení řešení je vyšší, výpočet trvá kratší dobu, nalezená cesta je kratší a konec trajektorie lépe odpovídá uživatelem zadané cílové konfiguraci.

Rychlá konvergence algoritmu CL-RRT je z velké části způsobena skutečností, že se algoritmus v každé iteraci pokouší o propagaci do cíle. V jednoduchých úlohách je tedy možné při vhodné volbě náhodného bodu nalézt řešení i během pouhé jedné iterace.

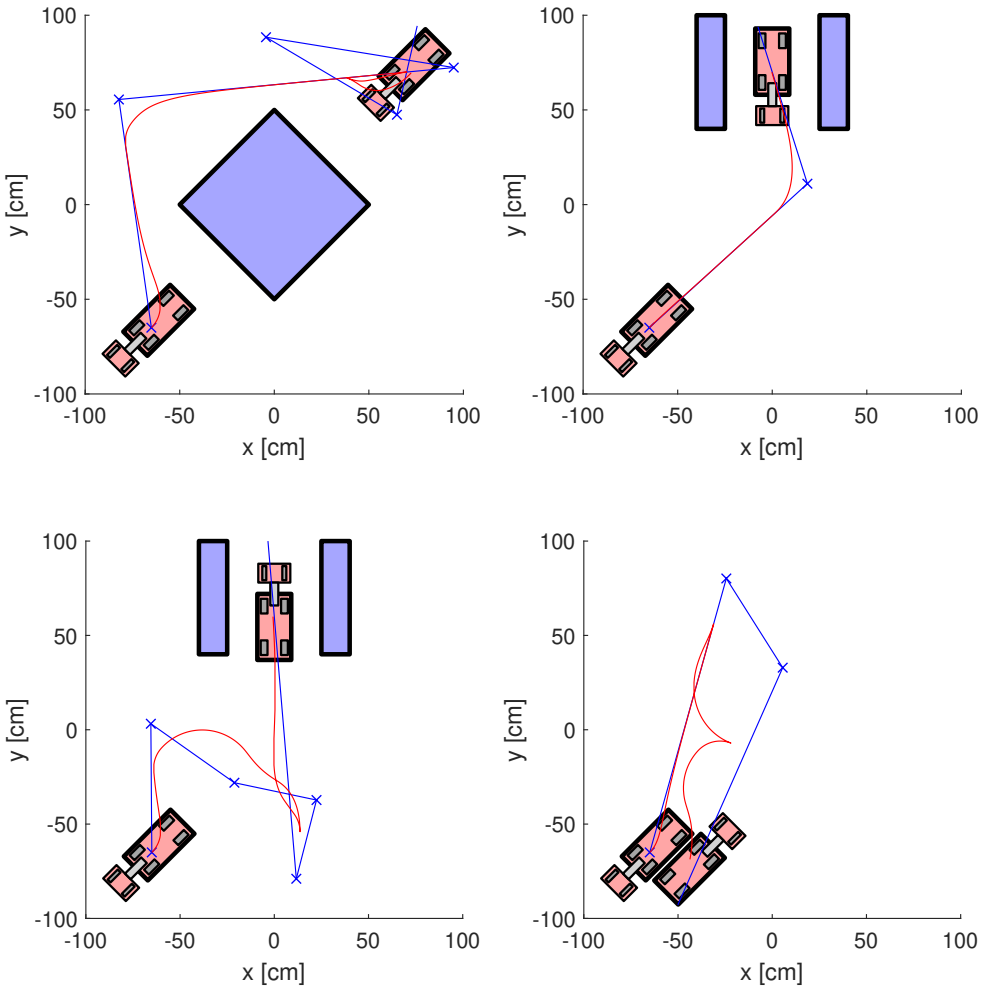
Za hlavní výhodu algoritmu CL-RRT lze zřejmě považovat skutečnost, že zde dochází k simulaci chování celé uzavřené smyčky. Algoritmus RRT nalézá cestu, která je daným robotem realizovatelná, neposkytuje však informaci o tom, jak má robot tuto cestu sledovat. Oproti tomu algoritmus CL-RRT prověřuje trajektorii, po které se robot skutečně bude pohybovat, proto prakticky nehrozí riziko, že robot během sledování naplánované cesty narazí do překážky, neboť celý tento pohyb byl předem simulován ve fázi plánování.

Plánovací algoritmus CL-RRT byl implementován rovněž pro úlohu plánování pohybu automobilu s jedním přívěsem. I v tomto případě algoritmus vykazuje velmi dobré vlastnosti. Na Obrázku 8.3 je zobrazeno řešení některých běžných situací, konkrétně objíždění překážky, zajíždění do garáže popředu i pozadu a otáčení na místě. Poznamenejme, že algoritmus je jednoduše rozšiřitelný také pro tahač s více přívěsy.

Algoritmus CL-RRT byl navržen tak, aby byl schopen v případě potřeby naplánovat cestu, po které se bude robot pohybovat pouze popředu či naopak pouze pozadu. Byla dále vytvořena animace, která vizualizuje pohyb robotu po nalezené cestě, což uživateli umožňuje vizuální kontrolu realizovatelnosti nalezené trajektorie.

Oba výše popsané algoritmy mají jednu zásadní nevýhodu – neposkytují optimální řešení. Požadujeme-li, aby nalezená cesta byla alespoň do určité míry optimální, lze plánovací algoritmus spustit několikrát, a poté vybrat nejkratší cestu.

RRT i jeho modifikace CL-RRT jsou neúplnými algoritmy, neboť při omezeném počtu iterací nelze garantovat nalezení řešení. [10] však uvádí, že pokud existuje realizovatelná trajektorie, pak pravděpodobnost, že ji RRT a jemu podobné algoritmy nenaleznou, s počtem iterací jdoucím do nekonečna klesá limitně k nule.



Obrázek 8.3: Ukázka cest nalezených pro automobil s přívěsem. Modrou barvou je vyznačena referenční cesta sledovaná regulátorem, červeně pak cesta, po které se robot skutečně pohybuje

9 Závěr

Tato práce byla zaměřena na problém plánování pohybu kolových mobilních robotů. Byly prezentovány některé běžně používané postupy, které umožňují naplánovat cestu pro holonomní roboty, hlavní důraz však byl kladen na problém tzv. neholonomního plánování.

Byly představeny dva algoritmy, které umožňují tuto úlohu vyřešit – RRT a CL-RRT. Výhodou algoritmu CL-RRT je, že dokáže plánovat pohyb i pro nestabilní roboty, během experimentů se však ukázalo, že oproti algoritmu RRT disponuje ještě mnoha dalšími benefity.

Prakticky byla řešena úloha plánování pohybu pro dva typy robotů – pro automobil a pro kloubové vozidlo T1T, tedy tahač s jedním přívěsem. Algoritmy byly implementovány v softwarovém nástroji MATLAB a byla ověřena jejich funkčnost pro různá prostředí robotu.

Jako možné rozšíření této práce se nabízí otestování plánovacích algoritmů na fyzickém modelu kloubového vozidla. Algoritmy lze implementovat např. v softwaru REXYGEN firmy REX Controls, s. r. o., který slouží k návrhu pokročilých řídících systémů a umožňuje uživateli definovat své vlastní funkční bloky. Tímto způsobem by bylo možné přímo do řídicí jednotky robotu implementovat program, který by z aktuální pozice robotu nalezl vhodnou cestu do uživatelem zadané cílové konfigurace a následně by řídil robot tak, aby se po nalezené cestě pohyboval.

Seznam použité literatury

[1] Klančar, G., Zdešar, A., Blažič, S., & Škrjanc, I. (2017). *Wheeled Mobile Robotics*. Oxford, Velká Británie: Elsevier.

[2] Vaněk, P. (2010). *Plánování pohybu technikami RRT* (Bakalářská práce). České vysoké učení technické v Praze, Fakulta elektrotechnická, Česká republika.

[3] Knispel, L. (2012). *Advanced Robot Path Planning (RRT)* (Diplomová práce). Vysoké učení technické v Brně, Fakulta strojního inženýrství, Česká republika.

[4] LaValle, S. M. (2006). *Planning Algorithms*. Cambridge, Velká Británie: Cambridge University Press.

[5] Ljungqvist, O. (2015). *Motion Planning and Stabilization for a Reversing Truck and Trailer System* (Diplomová práce). Linköping University, Department of Electrical Engineering, Švédsko.

[6] Holmer, O. (2016). *Motion Planning for a Reversing Full-Scale Truck and Trailer System* (Diplomová práce). Linköping University, Department of Electrical Engineering, Švédsko.

[7] Noreen, I., Khan, A., & Habib, Z. (2016). Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions. *International Journal of Advanced Computer Science and Applications*, *7*(11), 97–107.

[8] Schlegel, M. (2020). *Amazing Reversing T3T. Popis matematického modelu a stabilizujícího regulátoru kloubových vozidel typu TnT*. Plzeň, Česká republika: REX Controls, s. r. o.

[9] LaValle, S. M. (1998). *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc /download?doi=10.1.1.35.1853&rep=rep1&type=pdf

[10] Li, J., Liu, S., Zhang, B., & Zhao, X. (2014). *RRT-A* Motion Planning Algorithm for Non-holonomic Mobile Robot*. doi: 10.1109/SICE.2014.6935304

[11] Varadarajan, K. (2013). *All Nearest Neighbours via Quadtrees*. Dostupné z: http://homepage.divms.uiowa.edu/~kvaradar/sp2012/daa/ann.pdf

[12] Eberly, D. (2001). *Testing for Intersection of Convex Objects: The Method of Separating Axes*. Dostupné z: https://booksite.elsevier.com/ 9781558605930/revisionnotes/MethodOfSeperatingAxes.pdf

[13] Quinlan, S. (1994). *Efficient Distance Computation between Non-Convex Objects*. doi: 10.1109/ROBOT.1994.351059

Seznam obrázků