VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

PLÁNOVÁNÍ TRAJEKTORIE MOBILNÍHO ROBOTU S DIFERENCIÁLNÍM PODVOZKEM

DIFFERENTIAL CHASSIS MOBILE ROBOT TRAJECTORY PLANNING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Michael Chochel
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Jiří Krejsa, Ph.D.
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Michael Chochel**
Studijní program:	Strojírenství
Studijní obor:	Základy strojního inženýrství
Vedoucí práce:	**doc. Ing. Jiří Krejsa, Ph.D.**
Akademický rok:	2016/17

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Plánování trajektorie mobilního robotu s diferenciálním podvozkem

Stručná charakteristika problematiky úkolu:

Při plánování trajektorie mobilního robotu ve vnitřním prostředí hraje podstatnou roli neholonomní omezení pohybu. I v případě robotu s diferenciálním podvozkem existují taková omezení v případě, že osa poháněné nápravy není v ose souměrnosti robotu a tento nemá kruhový půdorys. Podstatou úkolu je zjistit trajektorii optimální z hlediska bezpečnosti. Prostředí je jasně definováno v podobě křižovatky tvaru T.

Cíle bakalářské práce:

1. Proveďte rešerši možných způsobů plánování trajektorie robotu s přihlédnutím k daným omezením.
2. Najděte optimální trajektorii pro průjezd křižovatky tvaru T, bez ohledu na výpočetní nároky.
3. Navrhněte jednoduchý plánovač trajektorie, který se bude optimální trajektorii blížit.

Seznam literatury:

S.M. LaValle: Planning algorithms, Cambridge University Press, 2006

Termín odevzdání bakalářské práce ›ven časovým plánem akademického roku 2016/17.

V Brně, dne

L. S.

.....	.....
prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt

Tato práce pojednává o návrhu trajektorie mobilního robota s neholonomním omezením pohybu a s diferenciálním podvozkem při průjezdu křižovatkou tvaru písmene „T“. V teoretické části budou popsány některé metody, kterými lze řešit úlohu plánování trajektorie robota. V praktické části bude vytvořen algoritmus pro nalezení optimální trajektorie robota pro průjezd křižovatkou.

Abstract

This work speaks about a suggestion of trajectory of mobile robot with nonholonomic movement limits and differential chassis while passing through a “T“ shaped crossroad. In the theoretical part some methods generally used to solve this task of planning a trajectory of the robot will be explained. In the practical part this task will be solved by an algorithm for searching the optimal trajectory to pass through the crossroad.

Klíčová slova

mobilní robot, diferenciální podvozek, neholonomní, plánování

Key words

mobile robot, differential chassis, nonholonomic, planning

Bibliografická citace mé práce:

CHOCHEL, M. *Plánování trajektorie mobilního robotu s diferenciálním podvozkem.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2017. XY s. Vedoucí bakalářské práce doc. Ing. Jiří Krejsa, Ph.D..

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování použil nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

.....

Poděkování

Velice děkuji vedoucímu mé bakalářské práce, panu doc. Ing. Jiřímu Krejsovi, Ph.D., za ochotu, trpělivost, vstřícnost a udávání směru, kterým se dát, když nevím, kudy kam.

Obsah

Úvod

Hledání trajektorií robotů je nedílnou a velice důležitou součástí robotického světa. Bez plánování trajektorie by žádný robot nebyl pořádným robotem. Plánování je ukazatel robotické inteligence. Nástroj k plnění veškerých úkonů.

Při hledání vhodné trajektorie hledáme cestu, která se vyhne všem překážkám, na které může robot během své cesty narazit. Cesta má vždy nějaký začátek i konec. Úkolem plánování trajektorie je spojit tyto dva body. Trajektorie, která body spojuje, je většinou ovlivněna nějakými požadavky uživatele. Někdy je vyžadována trajektorie co nejkratší, jindy co nejrychlejší nebo nejbezpečnější. Tato omezení si člověk více méně volí sám, ale existují i další vlivy, které ovlivňují to, jak bude trajektorie vypadat, ale my je nijak nezměníme.

Těmito vlivy, které mnohdy situaci spíše komplikují než naopak, jsou třeba kinematická omezení pro pohyb robota. Ovlivňují, jak rychle nebo jakým směrem se může robot pohybovat. Zástupcem stroje s takovým omezením je neholonomní robot.

Tradičním příkladem neholonomního robota je auto. To známe všichni, ale z hlediska omezení pohybu je třeba zmínit, že se auto nemůže otočit na místě a ani nemůže jet přímo do strany. Směr jeho pohybu je omezen tím, jak moc se mohou natočit jeho přední kola. To jsou typické znaky neholonomního omezení. Kdyby bylo auto holonomní, podélné parkování by bylo mnohem snazší. Po odbornější stránce lze říci, že neholonomní robot má menší počet říditelných stupňů volnosti, než je jejich celkový počet.

1 Kinematická omezení

V každém okamžiku se musí robot rozhodovat, jakou rychlostí se bude pohybovat a jakým směrem se vydá.

Zde jsou omezení týkající se rychlosti pohybu robota, se kterými se musí počítat, pokud nechceme, aby došlo k nějaké nehodě. Podle nich je vybírána vhodná akce u.

Omezení pohybu robota může být zadáno buď parametricky, nebo implicitně.

1.1 Implicitní omezení pohybu

Jsou použity souřadnice [x,y], kde x je směr dopředu a dozadu a y je směr doleva a doprava:

1) $\dot{x} > 0$ – ve směru y může robot mít libovolnou rychlost, ale ve směru x se může pohybovat pouze tím směrem, kterým se právě pohybuje, nikoliv proti stávajícímu směru,

2) $\dot{x} \geq 0$ – robot může během svého pohybu ve směru x změnit svou rychlost na nulu, teprve potom může, bude-li potřeba, změnit svou rychlost opačným směrem,

3) $\dot{x}^2 + \dot{y}^2 < 1$ – udává omezení rychlosti pohybu robota při vykonávání pohybu po křivce, aby se zamezilo převrhnutí robota vlivem vysoké rychlosti v zatáčce.

1.2 Parametrická omezení pohybu

Množina dovolených rychlostí je určena funkcí $f(q,u)$, která prohledává množinu všech přípustných akcí U, které může robot vykonat, a vybírá nejvhodnější možnosti v závislosti na aktuální konfiguraci robota.

K určení množiny dovolených rychlostí se může přistupovat dvěma způsoby:

1) Polohu q bereme jako neměnnou veličinu. Potom funkce f prohledává množinu U a stanoví všechny přípustné rychlosti dle každé dovolené akce $u \in U$.

2) Akci u bereme jako neměnnou veličinu. Potom funkce f prohledává všechny polohy q v konfiguračním prostoru pro každou neměnnou akci $u \in U$.

1.3 Omezení čtyřkolového neholonomního robota

Jak už bylo řečeno v úvodu této práce, náš robot má neholonomní omezení pohybu, nemůže se tedy libovolně pohybovat do stran. Pohyb, který není přímočarý, je omezen schopností robota natočit kola o určitý maximální úhel.

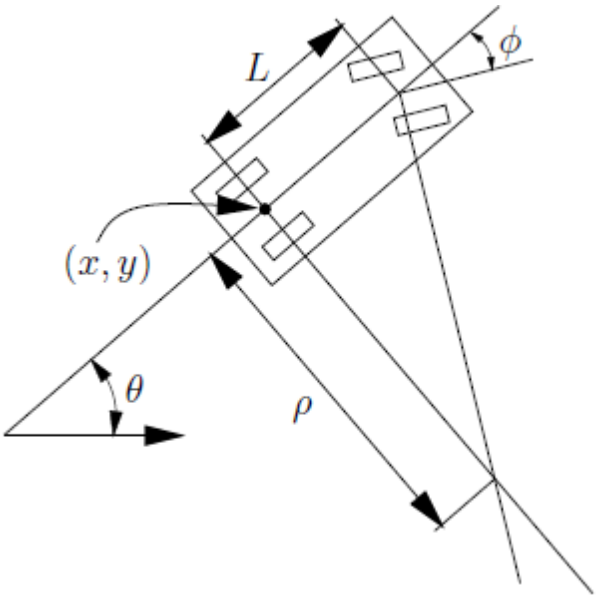
Aktuální konfigurace robota je určena jako $q = f(x,y,\phi)$, přičemž platí:

$$\dot{x} = f_1(x, y, v, \phi) \quad (1.1)$$
$$\dot{y} = f_2(x, y, v, \phi) \quad (1.2)$$
$$\dot{\theta} = f_3(x, y, v, \phi) \quad (1.3)$$

kde x, y jsou souřadnice osy poháněné nápravy, v je rychlost pohybu a ϕ je natočení robota.

Osa poháněné nápravy opisuje úhel $\rho = \frac{L}{\tan(\phi)}$, kde L je vzdálenost osy přední a zadní nápravy kol a ϕ je úhel řízení. Pohybem urazí dráhu $ds = \rho d\phi$, odtud získáme vztah pro změnu úhlu $d\phi = \frac{1}{L} * \tan(\phi)\, ds$. Poté bude přechodová rovnice konfigurace vypadat následovně:

$$\dot{x} = v * \cos(\phi) \quad (1.4)$$
$$\dot{y} = v * \sin(\phi) \quad (1.5)$$
$$\dot{\theta} = \frac{v}{L} * \tan(\phi) \quad (1.6)$$



obr. 1.1 Grafické znázornění podvozku čtyřkolového robota. [1]

2 Plánování na diskrétním stavovém prostoru

V této kapitole si uvedeme několik metod prohledávání stavového prostoru robota. Obecně algoritmy diskrétního plánování řeší problém tak, že prochází kompletně celý stavový prostor a posuzují stav po stavu, dokud nenajdou ten, který je označen jako cíl, resp. konečný stav, čímž vytvoří plán akcí, které robot musí vykonat. Pro začátek je nutné uvést některé pojmy:

- *stav* – konfigurace robota, jeho pozice a orientace (značí se $\mathbf{x}$)
- *počáteční stav* – startovní konfigurace robota (značí se $\mathbf{x_1}$)
- *konečný stav* – cílová konfigurace robota (značí se $\mathbf{x_G}$); může to být jediná konfigurace nebo i množina konfigurací, záleží na zadaných požadavcích
- *stavový prostor* – množina všech stavů robota (značí se X)
- *akce* – pomocí akce se mění stav $\mathbf{x_1}$ na stav $\mathbf{x_2}$ (značí se u)
- *přechodová funkce* – určuje, jak bude vypadat stav $\mathbf{x_2}$ (značí se $\mathbf{x_2} = f(\mathbf{x_1}, u)$)
- *akční prostor stavu* – množina aplikovatelných akcí na stav (značí se $U(\mathbf{x})$)
- *plán* – soubor akcí, které robot provede, aby se dostal ze stavu $\mathbf{x_1}$ do stavu $\mathbf{x_G}$.

Stavy ve stavovém prostoru můžeme rozdělit na následující typy:

- *nenavštívené* – stavy, které dosud nebyly prohledány algoritmem prohledávání
- *mrtvé* – stavy, které byly klasifikovány jako stavy, které nevedou k cíli
- *živé* – stavy, které byly navštíveny a vedou ke stavům nenavštíveným, ale ještě nemůžeme určit, zda vedou k cíli či nikoliv.

Jiné dělení:

- *expandované* – stavy, na kterých byly provedeny všechny možné akce (stavy živé i mrtvé)
- *neexpandované* – jsou vždy živé, dokud se neukáže, jestli vedou nebo nevedou k cíli.

2.1 Plánování na mřížce

Stavový prostor je rozdělen na čtvercovou síť. Jedna buňka je jeden stav. Každé buňce je přiřazen charakter volného prostoru nebo překážky, a to podle toho, zda překážka do dané buňky zasahuje či nikoliv.

Přesnost mřížky se zvyšuje s nárůstem počtu buněk. Nejjednodušší jsou mřížky čtvercové.

Řešení se hledá nejčastěji pomocí algoritmů diskrétního vyhledávání.

- ***Vzorkování uvnitř úzkých průchodů***

  Navzorkují se dvě nepovolené konfigurace a hledá se povolená konfigurace, která se nachází mezi nimi. Taková konfigurace je zapsána do mapy.

- ***Pseudonáhodné vzorkování***

  Stavový prostor se rozdělí na různě veliké buňky.

Zde je vhodné vysvětlit následující pojmy:

disperze – charakterizuje velikost největšího prázdného místa z celého stavového prostoru,
diskrepance – charakterizuje, jak rovnoměrně je prostor pokryt vzorky.

Cílem je dosáhnout co nejmenší hodnoty disperze a co největší hodnoty diskrepance.

- ***Van der Corputova posloupnost***
  Princip podobný půlení intervalů. Posloupnost funguje pouze na jedinou jednotku. Vzorky se ukládají doprostřed mezi další dva vzorky v každé ose. Tohle vede k rovnoměrnému a hustému zaplnění jednotky o malé disperzi a velké diskrepanci.

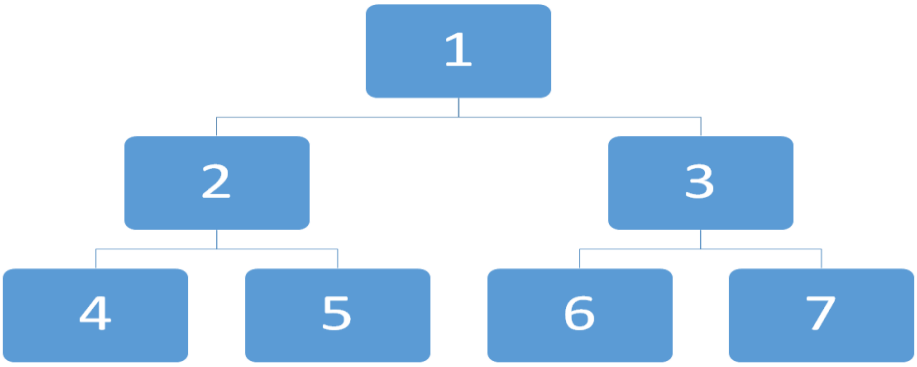
2.2 Neinformované algoritmy

Vyznačují se tím, že ignorují situaci, kdy řeší stav, který už řešily někdy předtím. Navíc prohledávají zbytečně velký stavový prostor a celý proces prohledávání trvá příliš dlouho.

Druhy algoritmů:
- *dopředný* – řešení začíná v $\mathbf{x_1}$ a hledá se cesta k $\mathbf{x_G}$,
- *zpětný* – řešení začíná v $\mathbf{x_G}$ a hledá se cesta k $\mathbf{x_1}$,
- *oboustranný* – řešení se hledá jak od stavu $\mathbf{x_1}$, tak od stavu $\mathbf{x_G}$. Přidává se další krok v hledání řešení, který zajišťuje spojení prohledaných stavových prostorů.

Při uplatňování těchto algoritmů se uvažují principy FIFO a LIFO:

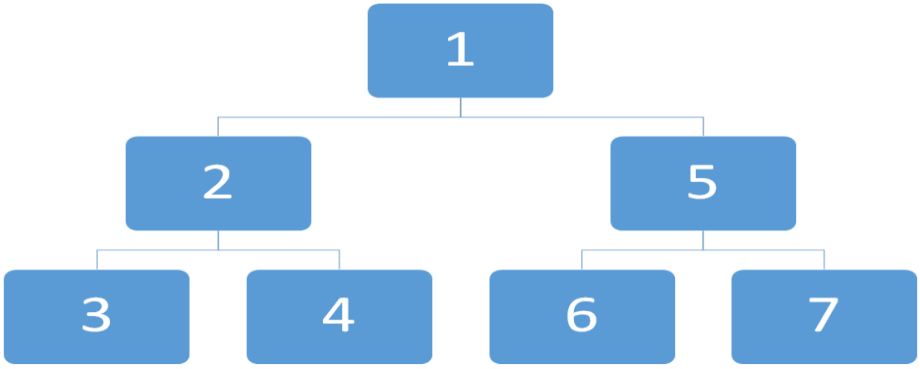
- ***Dopředné prohledávání do šířky***
  Prohledávání probíhá na principu FIFO (First-in, First-out).. Nejdříve se posuzují ty stavy, které přímo navazují na stav předcházející. Jakmile jsou tyto stavy vyřešeny, začnou se posuzovat stavy, které po nich následují. Pokud bude nalezen cílový stav, tak bude co nejblíže ke stavu počátečnímu.



obr. 2.1 Prohledávání do šířky.

- ***Dopředné prohledávání do hloubky***
  Prohledávání probíhá na principu LIFO (Last-in, First-out). Přímočařejší přístup, zasahuje hlouběji do grafu. Z počáteční pozice se posuzuje jen jeden stav, z něj zase jen jeden stav a tak to jde dál, dokud existují další navazující stavy k prohledání. Pokud ne, algoritmus se ve stejné větvi vrací zpět a zkoumá další stavy. Nevýhodou je,

že algoritmus může prozkoumat zbytečně velkou část stavového prostoru a zatíží tím paměť, nicméně samotný výpočet je rychlejší než u prohledávání do šířky.



obr. 2.2 Prohledávání do hloubky.

Rozdíl mezi těmito algoritmy spočívá pouze v pořadí posuzování stavů ze seznamu neexpandovaných stavů. Kromě toho fungují tyto algoritmy stejně.

V případě FIFa se jako první posuzuje počáteční stav $\mathbf{x_1}$. Poté se provede *akce* **U(x)**, podle níž se určí další stav x´, u kterého se posoudí, zda je cílovým stavem $\mathbf{x_G}$, který hledáme. Pokud ne, označí se jako mrtvý nebo expandovaný. Tento proces se opakuje pořád dokola, dokud není seznam neexpandovaných stavů prázdný nebo dokud není nalezen stav, který je označen jako cílový.

Algoritmus dopředného prohledávání:

```
1.   Q.Insert(x1) and mark x1 as visited
2.   while Q not empty do
3.       x ← Q.GetFirst()
4.       if x∈XG
5.           return SUCCESS
6.       forall u∈U(x)
7.           x´←f(x, u)
8.           if x´ not visited
9.               Mark x´ as visited
10.              Q.Insert(x´)
11.  return FAILURE
```

2.3 Informované algoritmy

Tyto algoritmy berou v potaz informaci, kde se cíl nachází. Narozdíl od neinformovaných algoritmů v sobě mají zahrnutý krok, který má na starosti duplicitu (situaci, kdy se řeší stav, který už se někdy dříve řešil). Navíc můžeme v těchto algoritmech zavést požadavky pro optimální cestu, který bude splňovat daná kritéria. Optimální cesta má co nejnižší možnou „cenu“.

- *Cena* – funkce, která je závislá na stavu a prováděné akci (značíme $g = f(\mathbf{x},u)$). Každý přechod z jednoho stavu do druhého má svoji cenu a celková cena za uraženou dráhu je sumou všech dílčích cen, které bylo nutno „uhradit“ za přesuny mezi stavy (značíme $G(\mathbf{x})$).

Informovaný algoritmus:

```
1.    Q.Insert(x1) and mark x1 as visited
2.    while Q not empty do
3.        x ← Q.GetFirst()
4.        if x∈XG
5.            return SUCCESS
6.        forall u∈U(x)
7.            x´←f(x, u)
8.            if x´ not visited
9.                Mark x´ as visited
10.               Q.Insert(x´)
11.           else
12.               Resolve duplicate x´
11.   return FAILURE
```

Mezi informované algoritmy patří:

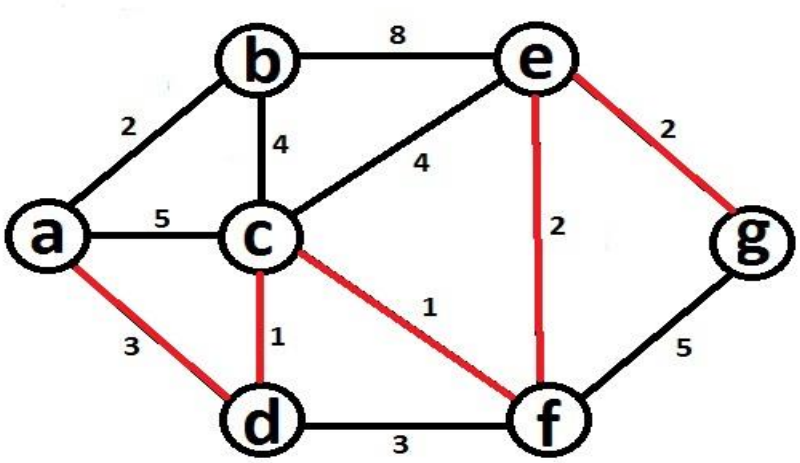
- ***Dijkstrův algoritmus***
  Funkce, podle které je vybrán další stav x´, je závislá na ceně každého stavu. Cena je dána náročností přechodu z jednoho stavu do druhého. Většinou je účelem vybrat ten prvek, za který platíme cenu nejnižší. Tato funkce má však neblahý následek na rychlost výpočtů a za zvýšenou optimalitu trajektorie tedy platíme zvýšením výpočetního času.

- ***Algoritmus uspořádaného prohledávání***
  Jeho prioritou není konkrétní optimální cesta, pouze jen její odhad (značíme h(**x**)). Algoritmus vybírá ze seznamu neexpandovaných stavů ten, který má hodnotu buď největší (maximalizace) nebo nejmenší (minimalizace). Hodnotu vybírá podle požadavků uživatele. Hodnota se stanovuje většinou podle vzdálenosti aktuálního stavu od $\mathbf{x}_G$.

- ***Algoritmus A****
  Kombinuje Dijkstrův algoritmus s algoritmem uspořádaného prohledávání (značíme V(**x**) = G(**x**) + h(**x**)). Odhadne celkovou cenu cesty z počátečního do koncového bodu.

  Jakmile se do cesty vloží nějaká překážka, celková cena musí zákonitě vzrůst a algoritmus A* hledá cestu s cenou, která bude původní celkové ceně co nejbližší.

  Od Dijkstrova algoritmu se odlišuje ve způsobu výběru stavu ze seznamu neexpandovaných. Výběr je závislý na parametrech zvolených uživatelem.



obr. 2.3 Znázornění ceny přestupu z jedné konfigurace do druhé. [10]

- ***Paprskovité prohledávání***

  Je to modifikace algoritmu uspořádaného prohledávání. Uživatel zvolí velikost paprsku, resp. velikost prohledávaného stavového prostoru, ze kterého si vybírá stavy k expanzi.

  Výhodou je menší náročnost na paměť a vyšší rychlost, jelikož prohledává jen omezený výběr stavů. Za to ale platíme rizikem, že nebude řešení vůbec nalezeno, protože cílový stav nemusí být vůbec vybrán do seznamu stavů k expanzi.

3 Plánování na spojitém stavovém prostoru

U spojitého plánování je vhodné přejít z reálného prostoru, kde se robot pohybuje, do konfiguračního prostoru, kde se dá s pohybem robota lépe pracovat. Zavádí se sem omezení ve formě překážek nebo kinematická omezení. S každým omezením se rozměr konfiguračního prostoru snižuje o jeden rozměr.

- *Konfigurační prostor* – prostor, kde je robot považován za bod a kde je tolik os, kolik je stupňů volnosti.

Plánovací algoritmy můžeme rozdělit podle toho, jak plán sestavují, na dva typy:

- *kombinatorické plánování* – postupně se vytváří diskrétní reprezentace daného problému, který je potřeba vyřešit, a pomocí prohledávacího algoritmu stavového prostoru se buď najde, nebo nenajde řešení (výhodou je, že se vždy dozvíme, zda ono řešení opravdu existuje, nebo ne = plánování je úplné),

- *pravděpodobnostní plánování* – stavový prostor je vzorkován a rozdělen na menší části, které jsou poté zase prohledány pomocí prohledávacího algoritmu stavového prostoru. Řešení se hledá, dokud se nenajde přibližné řešení, avšak možnost, že řešení neexistuje, se bohužel nedozvíme, vzniká tedy řešení pouze pravděpodobně úplné.

3.1 Kombinatorické plánování

Toto plánování je úplné. Vždy se dozvíme, zda cíl existuje či ne. Důležitým znakem tohoto typu plánování je vytvoření cestovní mapy při opakovaném prohledávání. Konfigurační prostor je rozdělen na buňky, které lze prohledat nějakou z diskrétních metod. Prostor musí být zmapován v co největší míře, aby byla chybovost maximálně snížena, jinak by to nebylo plánování úplné. Mezi buňkami musí existovat snadný přechod a musí být známo, která z buněk je počáteční a která cílová.

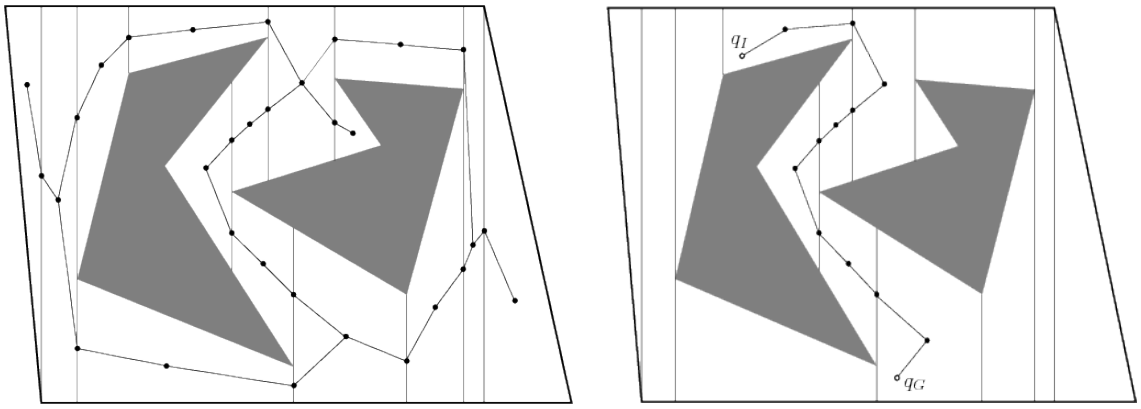
3.1.1 Metody vytváření cestovních map:

- ***Vertikální dekompozice***

  Rozděluje zmapovaný prostor na lichoběžníky. Jednotlivé překážky jsou znázorněny jako polygony a jsou charakteristické svými vrcholy.

  Uplatňuje se metoda pročesávací přímky. Prohledávaným prostorem prostupuje přímka, která se zastaví na každém vrcholu v pořadí vzestupném podle velikosti x-ové souřadnice vrcholů. V případě vertikální dekompozice se stane to, že se přímka v místě, kde je vrchol, rozdělí a putuje směrem dolů a nahoru. Poté se šíří v prostoru dál a stejným způsobem se rozdělí u všech vrcholů.

  Dále se konkrétní cesta mezi jednotlivými lichoběžníky hledá jako spojnice $\mathbf{x_1}$ a středu nejbližší vertikální úsečky nebo těžiště sousedícího lichoběžníku. Tento střed nebo těžiště se poté spojí s dalším charakteristickým bodem. Toto spojování pokračuje dál, dokud nedojde ke spojení s $\mathbf{x_G}$.

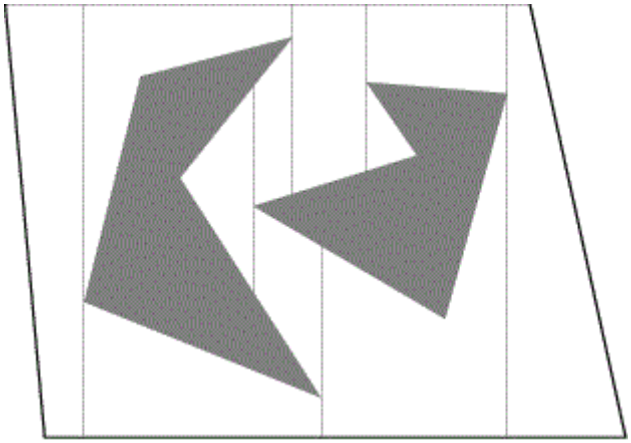


a) b)

obr. 3.1a) Rozdělení prostoru na lichoběžníky. Vyznačení všech bodů cestovní mapy černými tečkami na vertikálních úsečkách a v těžištích. [1]
obr. 3.1b) Vyznačení optimální cesty. [1]

- ***Bustrofedonová dekompozice***

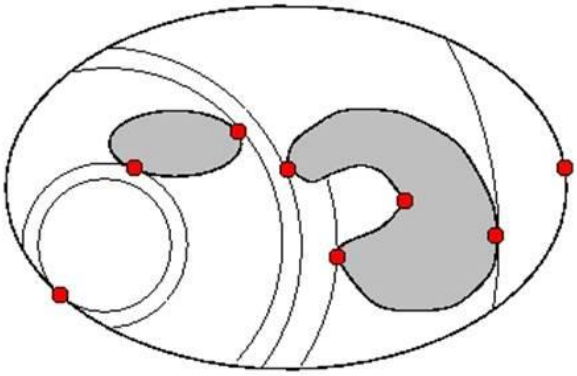
  Rozděluje zmapovaný prostor na lichoběžníky stejným způsobem jako dekompozice vertikální. Rozdíl je v tom, že vertikální úsečky se vedou pouze z „kritických bodů“. To jsou vrcholy, ze kterých je možné vést vertikální úsečku oběma směry až k okraji mapy. Pokud by se měla úsečka vést jenom jedním směrem nebo žádným, pak nevznikne. Výhodou je vytvoření menšího počtu lichoběžníků k nalezení cesty se stejnou efektivitou jako u vertikální dekompozice.



obr. 3.2 Znázornění vztyčení úseček, které vedou pouze z kritických bodů. [1]

- ***Morseho dekompozice***

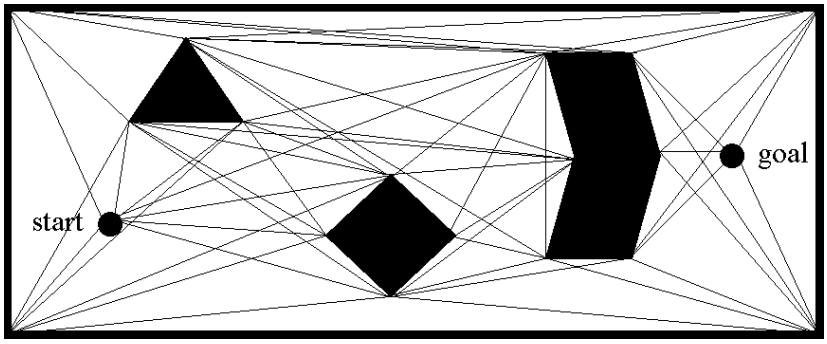
  Rozděluje zmapovaný prostor na základě tvaru funkce, která funguje na stejném principu jako pročesávací přímka. Mohou to být kružnice, přímky, čtverce nebo jiné geometrické útvary. Pomocí těchto bodů se poté vytvoří buňka. V předchozích dekompozicích měla tvar lichoběžníků, tady to může být jeden z mnoha tvarů. Prohledávací algoritmus poté hledá cestu mezi buňkami stejným způsobem jako u předcházejících dekompozic.



obr. 3.3 Morseho dekompozice pomocí kružnic. [15]

- ***Graf viditelnosti***

  Slouží k nalezení nejkratší možné cesty. Ta leží většinou těsně u překážek. Tento graf vytvoříme spojením startovního a cílového bodu a vrcholů překážek, přičemž žádná ze spojnic neprochází skrz překážku, resp. jednotlivé body spojené spojnicí se navzájem vidí.

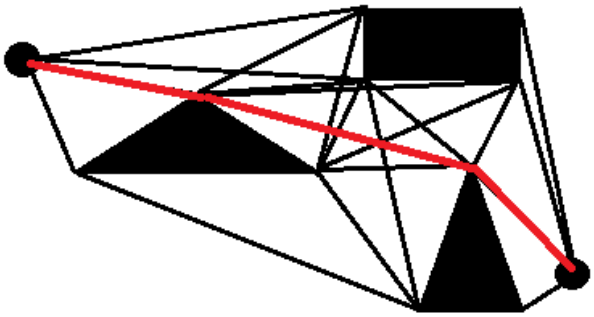


obr. 3.4 Graf viditelnosti. [14]

- ***Redukovaný graf viditelnosti***

  Je modifikací grafu viditelnosti. Vznikne odstraněním spojnic takových bodů, z nichž alespoň jeden leží na hranici prohledávané oblasti.

  Sníží se tím počet buněk, které je nutno prohledávat, a tím dojde ke zvýšení rychlosti nalezení nejvhodnější cesty.



obr. 3.5 Redukovaný graf viditelnosti se zvýrazněnou nejkratší cestou.

- ***Voroného diagramy***

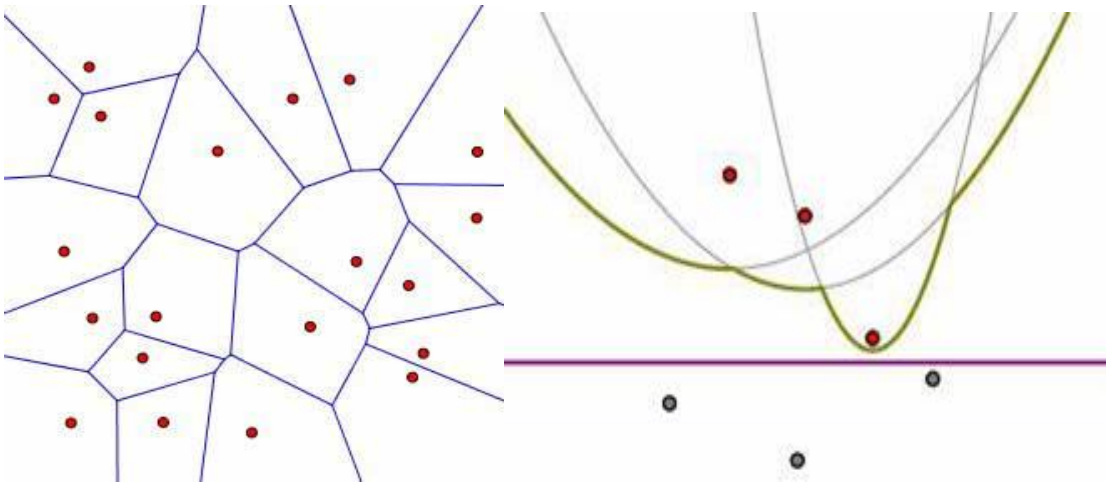
  Slouží k nalezení nejbezpečnější cesty mezi překážkami. Tyto diagramy fungují pouze v případě, že překážky zde vystupují jako body. Pro další případy tvaru překážek slouží Zobecněné Voroného diagramy.

  Vytvoří se nalezením dvou bodů, které mají od dvou nebo tří bodových překážek stejnou vzdálenost. Tyto body jsou následně spojeny, čímž se vytvoří dráha mezi překážkami, která má od jednotlivých překážek stejnou vzdálenost. Dráha je i nejbezpečnější cestou mezi těmito překážkami. Tyto spojnice jsou jedna po druhé spojeny, čímž se vytvoří Voroného diagram.

  Duálním grafem k Voroného diagramu je Delaunayho triangulace. Ta spočívá v sestrojení opsaných kružnic trojúhelníků, jejichž vrcholy jsou tři překážky, přičemž musí být dodržena podmínka, že uvnitř těchto kružnic se nenachází žádná překážka. Středy těchto kružnic jsou koncové body spojnic ve Voroného diagramu. Z toho plyne, že se Delaunayho triangulace dá použít ke konstrukci Voroného diagramu.

  Pro konstrukci Voroného diagramů se používají například následující metody:

  - *inkrementální metoda* – začne se sestrojením Voroného diagramu pro tři překážky. Poté se přidávají další překážky,
  - *metoda rozděl a panuj* – všechny překážky jsou rozděleny na několik skupin. Pro každou skupinu je vytvořen jeden Voroného diagram a nakonec se všechny zkonstruované Voroného diagramy spojí do jednoho,
  - *metoda pročesávací přímky* – při postupu pročesávací přímky je zachována posloupnost parabolických oblouků pomocí „přílivové vlny“. Průsečíky těchto oblouků tvoří hrany Voroného diagramu.



a) b)

obr. 3.6a) Voroného diagram s vyznačenými bodovými překážkami a cestami rovnoměrně vzdálenými od všech překážek. [12]
obr. 3.6b) Znázornění přílivové vlny, která vznikne při pohybu fialové pročesávací přímky. [12]

- ***Zobečněné Voroného diagramy***

  Pracují velice podobně jako Voroného diagramy. Jsou schopny pracovat i s překážkami jako jsou přímky, mnohoúhelníky a složitější tvary.

Metody konstrukce jsou stejné, pouze se provádí více operací:

- *inkrementální metoda* – nejdříve vytvoří Voroného diagram pro dvě bodové překážky. Poté k ním přídává další body a vytváří úsečky. Pro každou iteraci vytvoří Voroného diagram a ze všech diagramů složí ten výsledný Voroného diagram,
- *metoda rozděl a panuj* – body tvořící překážku jsou rozděleny do skupin po dvou bodech, pro něž je vytvořen jejich Voroného diagram. Výsledný Voroného diagram je zkonstruován spojením všech dílčích Voroného diagramů,
- *metoda pročesávací přímky* – překážka je rozdělena na úsečky a každá úsečka na body. Tyto body se poté zpracovávají stejně jako u Voroného diagramů zachováním posloupnosti parabolických oblouků.

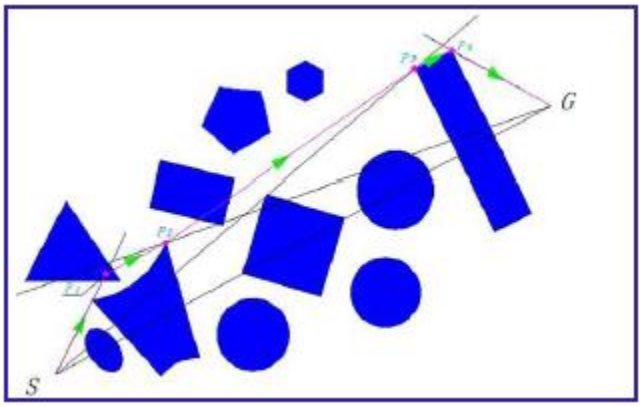
- ***Algoritmus pro plánování ve vektorově zadaném prostoru***

  Podle článku [9] není nutné na vektorově zadaném prostoru využívat dekompozici. Uvedený algoritmus najde cestu z počátečního do cílového bodu mezi nepohyblivými, vektorově zadanými, překážkami. Spojí počáteční (S) a cílový (G) bod úsečkou a tu poté upravuje tak, aby neprocházela žádnou překážkou.

  Celý proces vytváření trajektorie se dělí do několika kroků:

  1) Úsečkou se spojí body S a G.
  2) V případě, že spojnice prochází přes překážku, tak se zjistí, kterou stranou se překážka objede a použije se „plán pro vyhnutí se dvěma věžím“.

     Vyhnutí se dvěma věžím – Podle zvoleného směru objezdu těchto překážek se posuzuje levá a pravá část překážek. Vyberou se dvě překážky (dvě věže), které od spojnice vyčnívají nejvíce při pohledu od bodů S a G na zvolené straně. Poté se sestrojí tečny k těmto dvěma překážkám tak, aby procházely body S a G. Body, pro které jsou tečny sestrojeny, spojíme mezi sebou a s body S a G. Na obr. 10 jsou průsečíky spojnic vyznačeny fialovými tečkami. První ze spojnic mezi těmito body se stane částí trajektorie pro robota. U ostatních se vyšetří, zda procházejí překážkou a pokud ano, pak jsou odstraněny.



obr. 3.7 Průběh plánu vyhnutí se dvěma věžím. [9]

  3) Body P_1 a P_2 se stávají „náhradníky“ bodů S a G pro další iteraci. Až nastane případ, kdy na spojnici těchto bodů nebude žádná překážka, je tento úsek označen jako část výsledné trajektorie.

  Výsledná trajektorie je optimalizována s přihlédnutím k rozměrům robota.

3.2 Pravděpodobnostní plánování

Plánování, u kterého nevíme jistě, zda řešení určitě existuje nebo ne. Řešení není úplné, ale jen pravděpodobně úplné. Konfigurační prostor se rozdělí na vzorky, které postupně nějakým způsobem zpracovává a z výsledků vyhodnocuje nějaký pravděpodobný závěr.

- ***Potenciálové pole***

  Princip je takový, že cílové buňce přiřadíme nejnižší možnou hodnotu, nejčaštěji nulu, a počáteční buňce hodnotu nejvyšší. Jak moc velká tato hodnota bude, je určeno velikostí vzdálenosti od překážky. Následuje postup napříč buňkami použitím gradientní metody, tedy postup k buňkám s čím dál menší hodnotou.

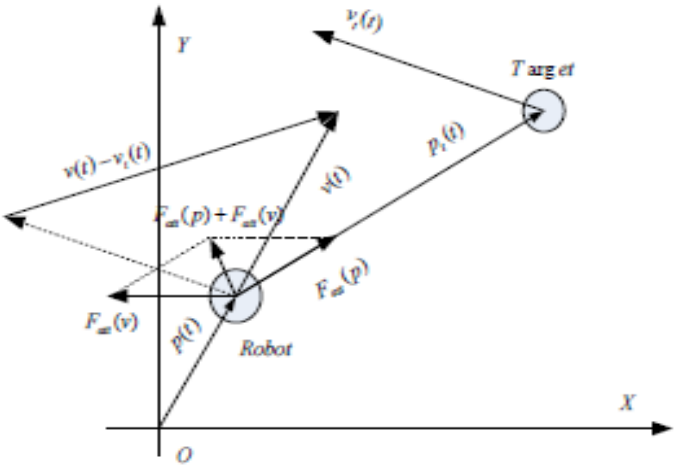
  Kvůli riziku uvíznutí v lokálním minimu se tato metoda kombinuje s některým z diskrétních algoritmů, které jsou schopny nalézt cestu z lokálního minima ven.

obr. 3.8. Přiřazení hodnot jednotlivým buňkám. Počátek má hodnotu 12, cíl hodnotu 0. [17]

- ***Potenciálové pole v dynamickém prostředí***

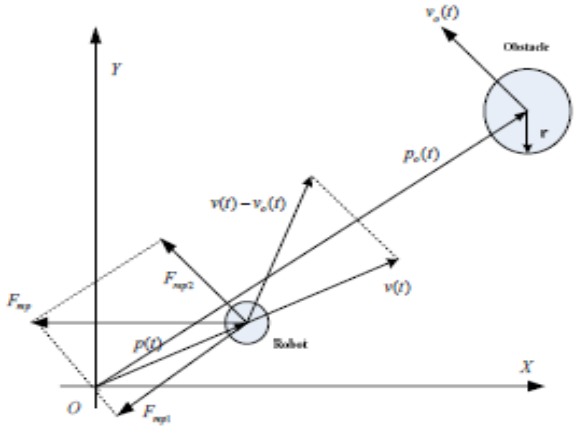
  Podle článku [8] se dá tento problém řešit přístupem přitažlivých a odpudivých sil, kde se zohledňuje poloha a rychlost robota vůči překážkám.

  Cíl k sobě přitahuje robota svým přitažlivým potenciálním polem, které má za následek vznik přitažlivé síly. Přitažlivá síla má dvě složky. F(p), která je závislá na poloze robota vůči cíli a F(v), která je závislá na rychlosti robota vůči cíli. Výsledná přitažlivá síla je určena vektorovým součtem těchto dvou složek, což je znázorněno na obr. 12.



obr. 3.9 Znázorňuje výpočet výsledné přitažlivé síly cílového bodu na robota. [8]

Překážky od sebe naopak robota odpuzují odpudivou silou F_{rep}. Ta je dána vektorovým součtem sil F_{rep1} a F_{rep2}, což je znázorněno na obr. 13. F_{rep1} je síla, jejíž nositelka leží na spojnici robota a překážky, a má směr od překážky. Nositelka síly F_{rep2} je kolmá na spojnici robota a překážky a orientovaná tak, aby robot objížděl překážku ve stejném smyslu, v jakém už ji objíždí z předchozí iterace.



obr. 3.10 Znázorňuje výpočet výsledné odpudivé síly překážky na robota.

Výsledný pohyb potom odpovídá směru a velikosti síly, která vznikne součtem obou výslednic.

3.2.1 Pravděpodobnostní mapy

Cílem je zjistit, zda je daná konfigurace v kolizi s překážkou nebo ne. Pokud není v kolizi s překážkou, je daná konfigurace považována za povolenou.

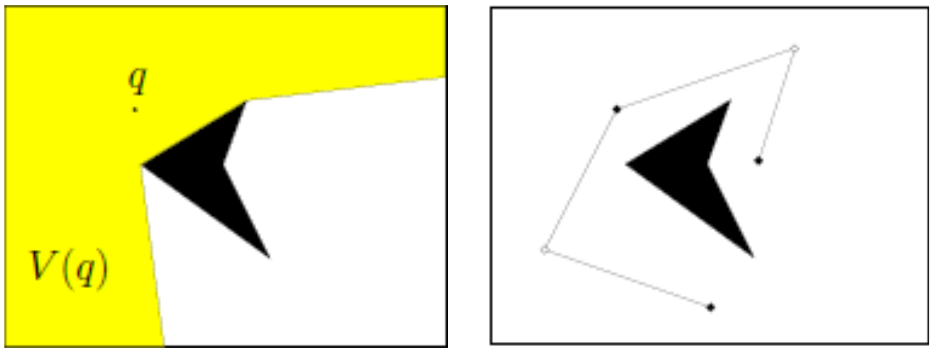
Cestovní mapa se vytváří spojením vzorků, které reprezentují povolené konfigurace robota, a jejich spojnicemi. Po vytvoření grafu se do algoritmu zahrne počáteční a koncový bod a následně se hledá nejkratší spojnice v grafu, která by spojovala tyto dva body.

Pro kontrolu kolize spojnice vrcholů grafu s překážkou se používá algoritmus pro detekci kolize s překážkou. To se provádí buď postupně, kdy se kontroluje jedna konfigurace za druhou, přičemž vzdálenost mezi konfiguracemi je konstatní, anebo půlením intervalu, kdy se jde od prostředku až po počáteční a koncové konfigurace.

- ***Pravděpodobnostní mapy založené na viditelnosti***

  Sestrojení mapy založené na dvou konfiguracích robota. Je to buď *strážce* nebo *spojka*. Strážcem se stává konfigurace, ze které nejde vidět žádný jiný strážce. Pokud algoritmus řeší konfiguraci, ze které nejde vidět na žádného strážce, stává se z této konfigurace další strážce. Pokud z této konfigurace jde vidět na dva strážce, stává se spojkou mezi těmito strážci.

  Tímto způsobem se mapa navzorkuje poměrně rychle, neboť počet uzlů mapy je malý a paměť namáhá méně.



a) b)

obr. 3.11a) Znázornění viditelnosti jednoho strážce. [1]
obr. 3.11b) Znázornění cestovní mapy vytvořené třemi strážci (plné tečky) a dvěma spojkami (kruhy). [1]

- ***Optimalizace mravenčí kolonií***

  Uplatňuje se princip ze života mravenců. Nová kolonie začne prohledávat okolí mraveniště ve snaze najít zdroje jídla. Klíčem je schopnost mravenců uvolňovat feromon, který jsou ostatní mravenci schopni cítit.

  Mějme jedno naleziště jídla, ke kterému vedou dvě cesty. Při cestě za jídlem se mravenci rozdělí na dvě skupiny a každá jde jiným směrem. Pokud je jedna z cest kratší než druhá, pak mravenci, kteří šli po delší cestě, budou během zpáteční cesty do mraveniště následovat své druhy, kteří k jídlu dorazili dříve.

  Během jejich pohybu mravenci uvolňují feromon. Koncentrace feromonu bude hustější na kratší cestě, neboť se po této cestě vydá více mravenců. Jiní mravenci tento feromon vycítí a rovnou se vydají cestou, kde je koncetrace feromonu větší. Tím se vytvoří nejkratší spojnice naleziště jídla a mraveniště.

  Stejný princip je použit v článku [7], kde se robot snaží nalézt nejkratší cestu z počátku do cíle. Celý stavový prostor je na mřížce. Jako první krok je vždy nalezena nejkratší vzdálenost z buňky mřížky do další buňky. To se provede několikrát pro několik generací mravenců. Při přechodu z buňky do buňky je propočítána funkce, která funguje jako zanechání feromonu na cestě. Zároveň je zavedena funkce, která simuluje vymizení stopy feromonu v čase, pokud se v daném místě dlouho žádný mravenec nenachází. V závěru je nalezena nejkratší cesta podle koncentrace feromonu mezi uzly jednotlivých tras mravenců.

  V článku [10] je tato metoda vylepšena, aby bylo zajištěno, že algoritmus bude konvergovat a výsledná trasa bude ještě kratší a hladší.

  Docílilo se toho pomocí úpravy výpočtu posunu do další konfigurace. Je zaveden vliv úhlového faktoru. Mravenci mají kvůli tomuto faktoru omezenou viditelnost a mohou si tedy vybírat z omezeného počtu míst, kam se vydají v další iteraci. Tato úprava má za následek snížení výpočetního času a omezení konvergence v lokálních optimálních řešeních.

Na konci každé iterace se posuzuje, zda došlo k nalezení cíle. Pokud cíl zatím nebyl nalezen, tak se celý proces opakuje znovu od počátečního bodu. Hodnoty feromonové funkce zůstavají pro dosud navštívené body uloženy. V každé iteraci se tedy rozroste počet bodů, ve kterých byl proveden výpočet feromonové funkce. Podle feromonů je nakonec určena optimální trajektorie.

- ***Algoritmus skákající žáby s posunutím***

  V článku [6] je uveden algoritmus, který spočívá ve vytváření skupin žab, jež skáčou směrem k cíli. V každé iteraci je náhodně vytvořena populace P žab. Poté jsou žáby seřazeny podle toho, jak svou polohou vyhovují požadavkům optimální trajektorie a poté jsou jim sestupně přiřazena čísla 1, 2, ...P. Všechny žáby jsou následně rozděleny do několika oblastí. Optimalita polohy žáby se posuzuje z hlediska vzdálenosti od cíle a překážek. Vzdálenost k cíli musí být co nejmenší, zatímco vzdálenost od překážek co největší. V každé oblasti navíc nezávisle probíhá další vyhledávací algoritmus, aby pro danou oblast našel nejlepší řešení.

  Dalším krokem je označení nejvíce, resp. nejméně, vyhovující žaby značkou X_b, resp. X_w. Žába, která z celé populace žab vyhovuje nejvíce je označena X_G.

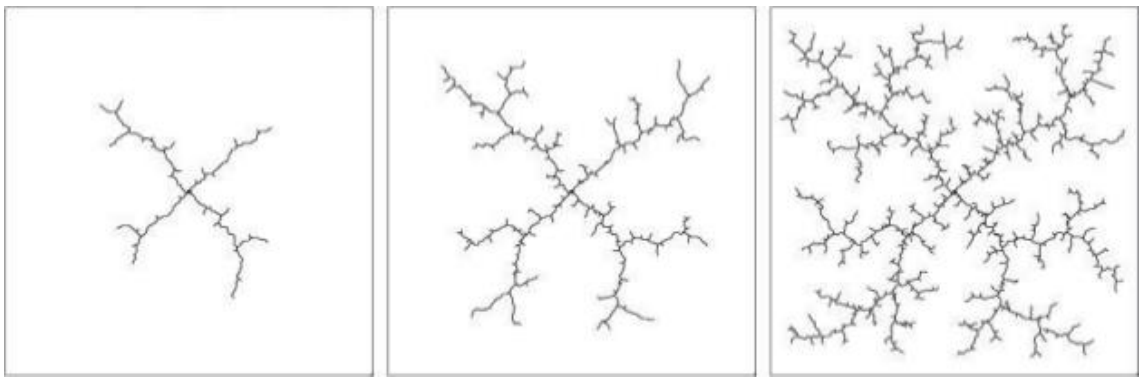
  V každé oblasti se provede posunutí žáby X_w podle vzorce $X_\Delta = rand()*(X_b - X_w)$, kde rand() je náhodně zvolené číslo v intervalu (0;1). Pokud nová pozice pořád nevyhovuje, aplikuje se stejný vzorec, ale místo X_b se použije X_G. Pokud ani tehdy nová pozice nevyhovuje, je žába smazána a vytvořena nová.

  Tímto skákáním a posouváním kupředu je postupně vytvořena trajektorie robota ze startovního do cílového bodu bez kolizí s překážkou.

- ***Rychlé náhodné stromy***

  Metoda je principiálně postavená na tom, že se po konfiguračním prostoru rozrůstá strom, jehož kořen je v počáteční konfiguraci a snaží se dostat až k cílové konfiguraci.

  Na počátku je tedy určen kořen a z něj určen další bod vzdálený o předem určenou délku kroku ε, která je určena uživatelem. Tyto dva body vytvoří základ kmene, který se dále rozšiřuje pomocí větví, které rostou do volného prostoru. Takto se strom pořád rozrůstá. Vždy se vytvoří další bod vzdálený od předchozí konfigurace o vzdálenost ε. Pokud tato vzdálenost zasahuje v daném směru do překážky, je změněn směr a bod se vytvoří tak, aby do překážky nezasahoval. Řešení této situace zajišťuje algoritmus pro řešení kolize, který také určuje, jak blízko se mohou jednotlivé konfigurace přiblížit k překážce.

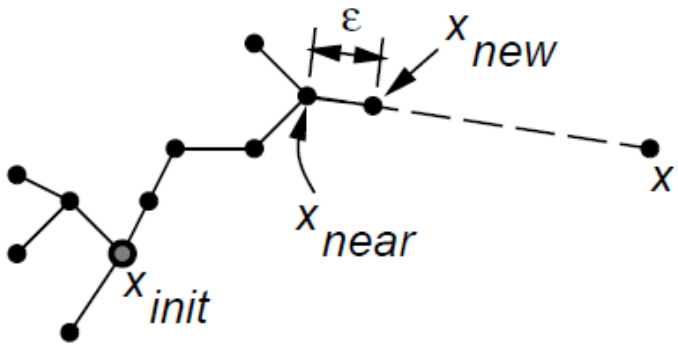


obr. 3.12 Znázornění růstu stromů. [15]

Základní algoritmus RRT:

```
BUILD_RRT(x_init)
1    G.init(x_init);
2    for k=1 to K do
3            x_rand ← RANDOM-STATE();
4            EXTEND(G,x_rand);
5    Return G

EXTEND(G,x)
1    x_near ← NEARES_NEIGHBOR(x,G);
2    if NEW_STATE(x,x_near,x_new,u_new) then
3            G.add_vertex(x_new);
4            Gadd_edge(x_near,x_new,u_new);
5            if x_new = x then
6                    Return Reached;
7            else
8                    Return Advanced;
9            Return Trapped;
```



obr. 3.13 Postup hledání nových bodů do stromu. [3]

V každé iteraci je strom rozšířen o další náhodně vybranou konfiguraci. Funkce EXTEND vybírá konfiguraci ve stromu x_{near}, resp. v množině *G*, která je nejblíže k této náhodně vybrané konfiguraci *x*. To, jak blízko u sebe mají být, je dáno uživatelem.

Funkce NEW_STATE vykonává pohyb ve směru ke konfiguraci *x* v závislosti na akci *u*. Tato akce je vybrána náhodně nebo vybrána po vyzkoušení všech možností a uznána jako nejvhodnější pro výběr nejbližší konfigurace. Pokud je funkce úspěšně provedena, pak je zaznamenána nová hodnota x_{new} a vložena do množiny *G*.

Dále může nastat jedna ze tří možností. Pokud je x_{new} naše cílová konfigurace, pak je vrácena hodnota *Reached* a celý cyklus je ukončen. Pokud x_{new} není cílová konfigurace, pak je vrácena hodnota *Advanced*, x_{new} je spojena s x_{near} a algoritmus pokračuje dále v hledání cesty. Pokud x_{new} leží v překážce a algoritmus už nemá možnost nalézt cestu k cíli, pak je vrácena hodnota *Trapped*. Algoritmus nehledá cestu zpět, jelikož funguje tak, aby prohledával místa, kde ještě nebyl.

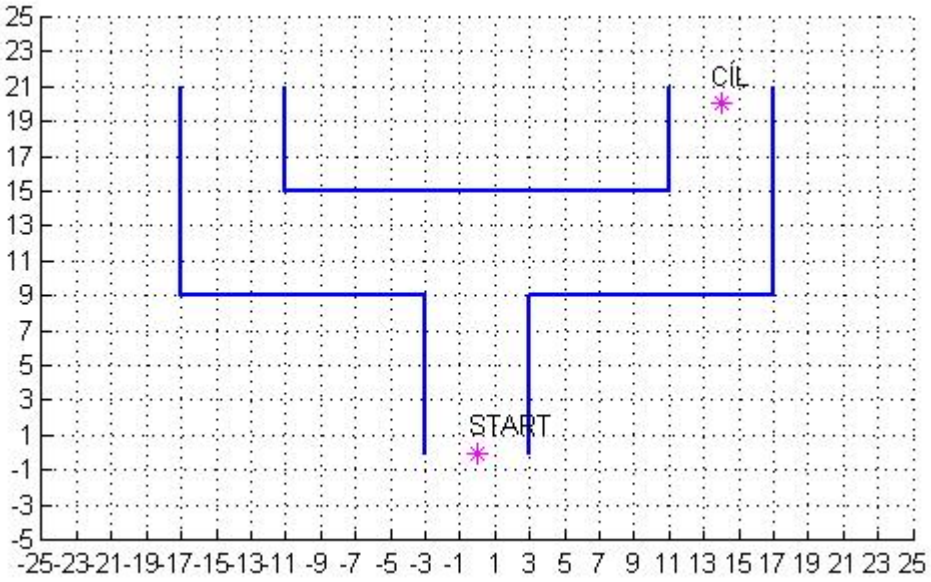
Někdy se používá metoda řešení pomocí dvou a více stromů. Tato metoda je rychlejší, vyplatí se ve složitých terénech, avšak musí se vyřešit problém navázání stromů v momentě, kdy se setkají.

4 Hledání optimální trajektorie

K nalezení optimální trajektorie byl sepsán algoritmus v programu Matlab R2013a. Jeho simulace se spouští v souboru Spustit_Optimalni.m.

Jelikož má být optimální cesta ta nejbezpečnější, pak musí trajektorie ležet uprostřed mezi stěnami, aby měla od každé zdi chodby stejnou vzdálenost. Vlivem neholonomního omezení robota se celá úloha dost komplikuje a proto přistupujeme na řešení, kdy má robot téměř stejnou vzdálenost od obou stěn.

Znovu připomeňme, že podle zadání má křižovatka tvar písmene „T“. Šířka chodby je stanovena na 6 jednotek. Řešení je pojato obecně, proto je použita obecná míra délky „jednotka“. Přesnější rozměr (mm, cm, m) bude použit v závislosti na konkrétním problému nebo zadání. Robot projíždí chodbou tohoto tvaru takovým způsobem, aby se dostal ze startovní do cílové konfigurace a udržoval si přitom co největší odstup od zdí nalevo i napravo od něj.

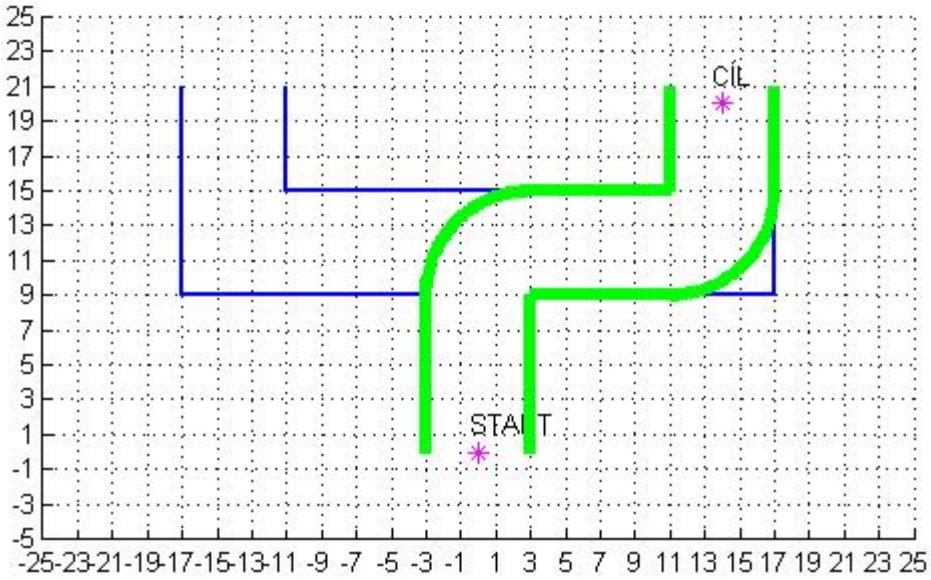


obr. 4.1 Zadání řešené úlohy. Křižovatka tvaru písmene „T“ a dva body, které označují počátek a konec cesty robota.

4.1 Úprava zdí

Pro zjednodušení a zrychlení výpočtů začíná algoritmus krokem separace zdí. V tomto kroku se z mapy odstraní ty zdi, kolem kterých robot nebude projíždět během cesty ke svému cíli.

V tomto případě robot odbočí určitě doprava, tudíž zdi nalevo se mohou odstranit. Zbylé zdi spojíme do dvou křivek tak, aby vymezovaly hranice jakési imaginární dráhy pro našeho robota. V tuto chvíli se už robot nepohybuje podle původních překážek, ale podle nově vytvořených, které o něco lépe určí tvar optimální trajektorie.



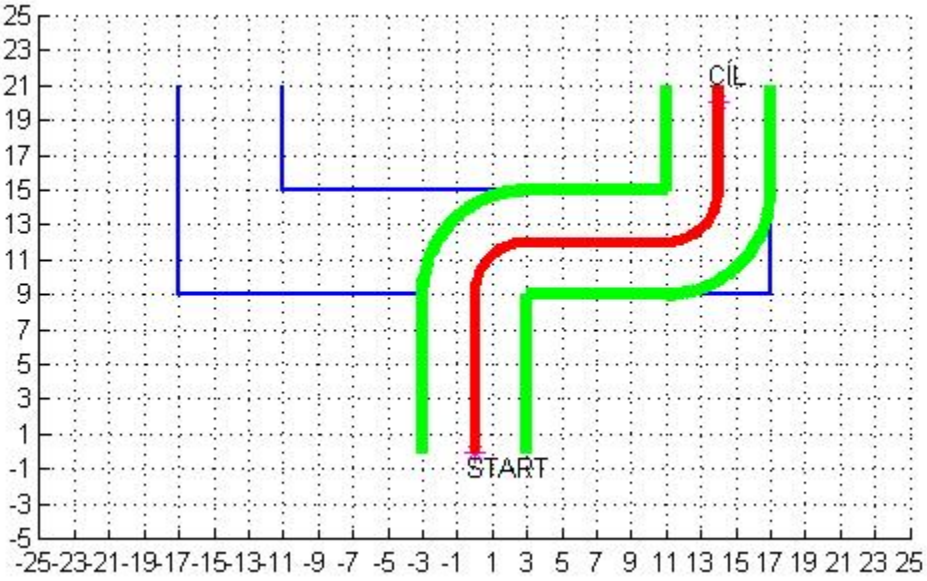
obr. 4.2 Zelené křivky vymezují nově vytvořenou dráhu pro robota.

4.2 Určení trasy uprostřed chodby

Pro určení optimální trajektorie je nutné určit středovou čáru po celé délce vytvořené dráhy.

Každá rovná část krajnice naší vytvořené dráhy je naplněna stejným počtem bodů. Tyto body jsou spojeny s body v krajnici naproti. Celá dráha je tedy zaplněna spojnicemi, které se táhnou napříč celou délkou. V případě zatáček jsou body krajnice spojeny s protilehlým rohem.

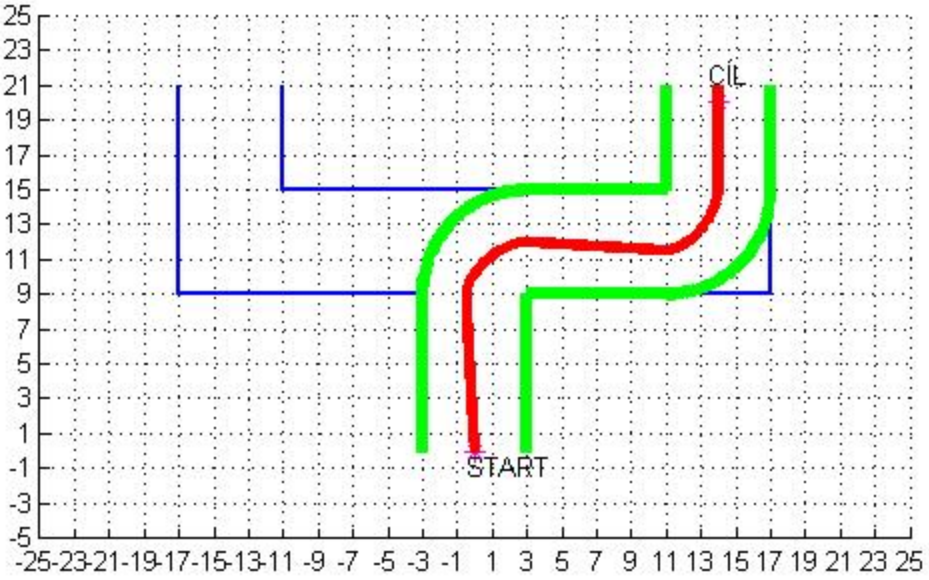
Když jsou vytvořeny všechny spojnice, najdeme střed každé ze spojnic. Tyto středové body spojnic spojíme, čímž vytvoříme křivku táhnoucí se přímo uprostřed dráhy.



obr. 4.3 Znázornění optimální trasy pro neholonomního robota.

Po vytvoření křivky je potřeba ji ještě upravit. Na mnoha místech získané trajektorie není potřeba nic měnit, ale například v zatáčkách to je potřeba.

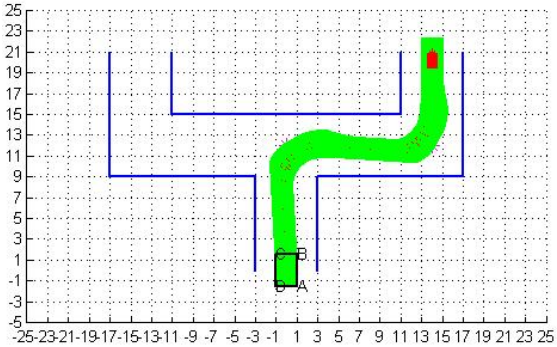
Proto je na zatáčky aplikována korekce. Ta je závislá na rozměrech robota a poloměru zakřivení zatáčky. Korekce posune body trajektorie v zatáčkách. Pokud by se vliv korekce zanedbal, robot by se příliš vychýlil a hrozil by náraz do zdi. Finální podoba optimální trajektorie je znázorněna na obr. 4.4.



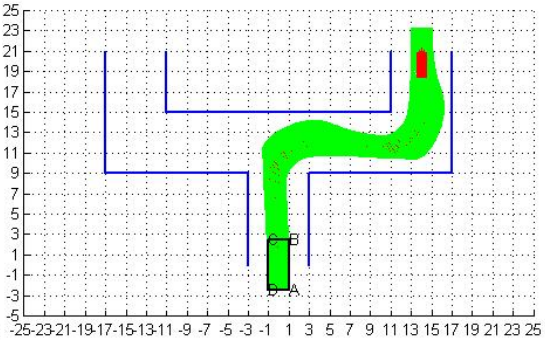
obr. 4.4

4.3 Ověření funkčnosti

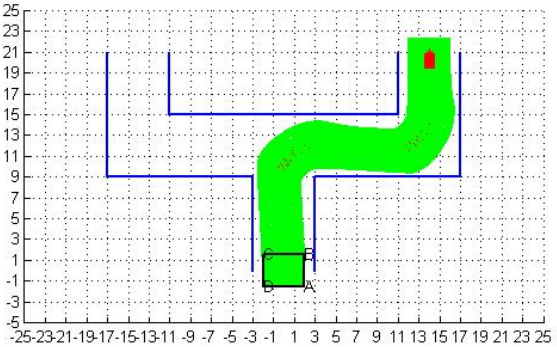
Pro ověření správnosti výpočtu optimální trajektorie bylo provedeno měření pro čtyři různé rozměry robota. Každý z těchto robotů projel zadanou křižovatku po optimální trajektorii s posunutím v zatáčkách. Robotům byly dány rozměry a) 2x3 jednotky, b) 2x5 jednotky, c) 4x3 jednotky, d) 4x5 jednotky. Šířka chodby je 6 jednotek.



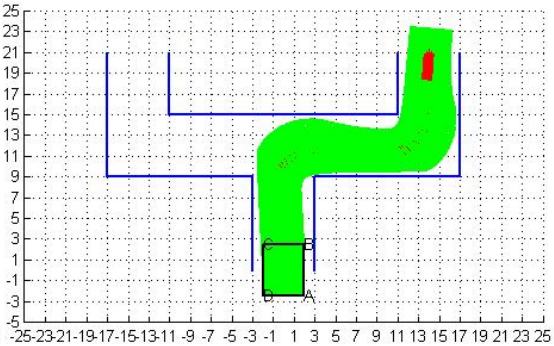
obr. 4.5a)



obr. 4.5b)



obr. 4.5c)



obr. 4.5d)

Z obrázků 4.5 a) až d) jde vidět, že v oblastech, kde není zatáčka, se robot pohybuje o něco blíže k jedné ze stěn chodby. Je to kvůli tomu, aby mohl robot projet zatáčkou a zároveň být v bezpečné vzdálenosti od všech stěn. Musí si do zatáčky „najet“.

Ná základě těchto měření jsme tedy schopni stanovit přibližnou minimální vzdálenost od zdí, kterou by měl robot během své cesty dodržovat. Dále už chybí jen navrhnout plánovač, který vytvoří trajektorii s požadovanými vlastnostmi a zároveň nebude tak moc náročný na výpočty.

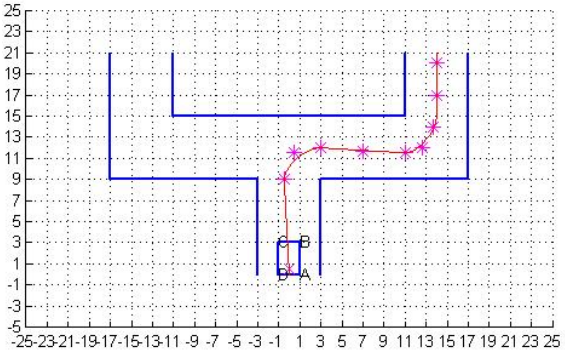
5 Plánovač trajektorie

Jelikož výpočet optimální trajektorie pracuje s příliš mnoha informacemi a její výpočet je tedy náročný a příliš zdlouhavý, je třeba navrhnout plánovač, který přinese přibližně stejný výsledek, ale výpočetní techniku zatíží méně.

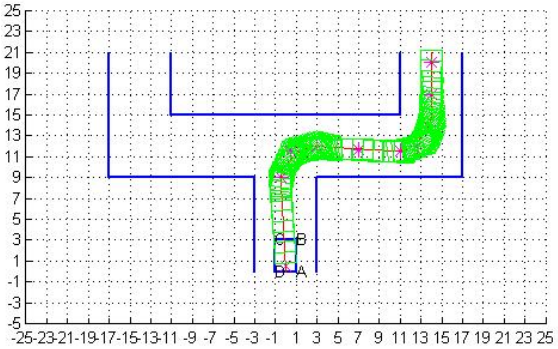
5.1 Pohyb robota po mapě

Princip pohybu robota po mapě spočívá v tom, že se na mapě rozmístí několik bodů, kterými musí robot projet. Těmito body jsou start a cíl a několik dalších bodů, které jsou v algoritmu pojmenovány „vodítka“. Název pochází z jejich účelu navádět robota během jeho cesty k cíli.

Bylo provedeno patnáct měření, kde se zkoušely sestavy o různém počtu vodítek s různou polohou. Výsledky měření jsou zapsány v souboru Záznamy_vzdáleností.xlsx. Pro zjednodušení výpočtu plánovače je vhodné, aby byl počet vodítek co nejmenší. Z měření vyplynulo, že aby se robot pohyboval v dostatečné vzdálenosti od stěn chodby, stačí použít 9 vodítek. Jejich rozmístění je znázorněno na obrázku 5.1. a cesta robota naváděného těmito vodítky na obrázku 5.2. Na obou obrázcích je vidět, kudy vede optimální trajektorie a jak moc se trajektorie plánovače liší.



obr. 5.1



obr. 5.2

5.2 Pohyb robota z jedné konfigurace do druhé

Jak již bylo zmíněno, pohyb robota je ovlivněn rozmístěnými body po mapě, vodítky. Robot jimi projíždí postupně až dokud nezbývá samotný cíl cesty.

Poloha vodítek je předem známá. Aby se robot ke každému z nich bezpečně a jistě dostal, musí v každé iteraci proběhnout proces posunutí a v případě potřeby i natočení robota.

Maximální rychlost posuvu robota je stanovena už na začátku cesty. Robot začne výrazně zpomalovat pouze tehdy, je-li nutné prudce změnit směr.

Maximální změna směru pohybu je stanovena na $\frac{\pi}{64}$. Pokud je potřeba změnit směr natočení robota, pak robot podle velikosti změny směru zpomalí. Zpomalení je při změně směru nutné zavést kvůli působení setrvačných sil.

Robot se tedy na začátku každé iterace natočí směrem k vodítku, ke kterému zrovna jede. Jakmile je robot natočen přímo na vodítko, jede k němu plnou rychlostí.

Stav, zda je robot natočen přímo na vodítko, se posuzuje pomocí výpočtu obecných rovnic přímek. Počítáme rovnice pro dvě přímky, kde každé přímce náleží jedna ze dvou stěn robota. Když jsou určeny obecné rovnice těchto přímek, dosadí se

souřadnice vodítka. Po dosazení souřadnic vodítka do rovncie přímky, vyjde číslo, u kterého se posuzuje, jestli je větší nebo menší než 0. Jinými slovy se určí, zda je vodítko napravo nebo nalevo od přímky. Naším cílem je zajistit, aby bylo vodítko nalevo od pravé strany robota a napravo od levé strany robota. Potom je robot natočen na vodítko.

V momentě, kdy se střed podvozku robota přiblíží dostatečně blízko k vodítku, je jako nový cíl zvoleno další vodítko v pořadí.

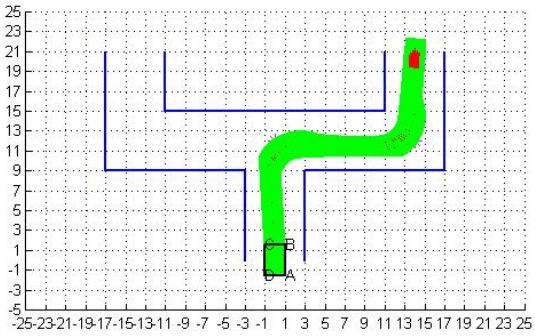
Samotný proces posuvu a natočení robota do další iterace je řešen pomocí matice rotace kolem osy z, která prochází osou otáčení robota. Ta se nachází uprostřed spojnice zadních kol, která jsou umístěna v první šestině délky podvozku.

Dále se řeší velikost a směr vektoru posunutí robota. Tento vektor charakterizuje rychlost robota. Jeho složky jsou ovlivněny změnou směru vektoru, resp. sinem a cosinem úhlu, o který se mění natočení robota v další iteraci.

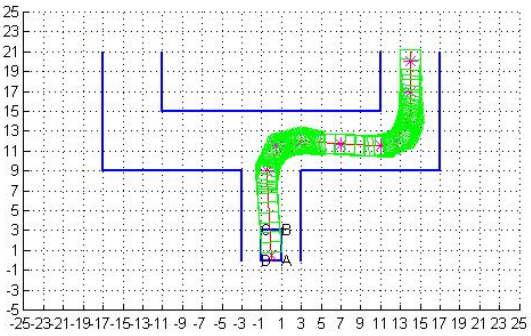
6 Naměřené údaje

6.1 Porovnání optimální trajektorie s trajektorií vypočítanou plánovačem.

Jako kritéria srovnání se použijí vypočítáné hodnoty bezpečnosti, resp. vzdáleností vrcholů a stěn robota od zdí a také výpočetní čas při výpočtu optimální trajektorie, obr. 6.1, a trajektorie vypočítané plánovačem, obr. 6.2. Cílem je najít největší hodnoty bezpečnosti a nejkratší výpočetní čas.



obr. 6.1



obr. 6.2

Nejmenší naměřená vzdálenost vrcholu podvozku robota od některé ze zdí chodby má hodnotu 1,4165 jednotky. Nejmenší hodnota vzdálenosti stěny robota od některé ze zdí chodby byla naměřena 1,8278 jednotky. Čas výpočtu trajektorie je 2,965 sekundy.

Pokud hledáme vhodnou trajektorii navrženou plánovačem, pak se tato trajektorie musí blížit těmto naměřeným hodnotám.

Vhodná sestava vodítek odpovídá rozmístění dle sestavy číslo 15, viz. soubor Záznamy_Vzdáleností.xlsx. Výsledek této trajektorie lze vidět na obr. 6.2. Zde má nejmenší naměřená vzdálenost vrcholu od zdí hodnotu 1,3747 jednotky. Hodnota nejnižší vzdálenosti stěny robota od zdí je 1,7011 jednotky. Výpočet této trajektorie trvá 0,672 sekundy. Čas ale při výběru vhodného rozložení vodítek nehrál příliš velkou roli. Až na vyjímky byly naměřené časy srovnatelné.

6.2 Hodnoty pro různé návrhy trajektorií

Bylo provedeno patnáct měření. První bylo tvořeno pouze třemi vodítky včetně cíle. Zbylá dvě vodítka byla umístěna do zatáček, jakožto nejzajímavějších oblastí. Podle naměřených hodnot je tato sestava vodítek nedostatečná, neboť se jeden z bodů během cesty nacházel příliš blízko zdi.

V dalším postupu se přidávalo více vodítek, jelikož jen se třemi se dostatečně bezpečná trajektorie nedá nalézt. Během měření se častokrát objevila sestava vodítek, kde naměřené hodnoty dalece přesahovaly požadavky na bezpečnost trajektorie. Nicméně vždy aspoň jedna z hodnot byla silně nedostačující.

V posledním měření byla nalezena trajektorie, která splňovala požadavky ve všech směrech. Kvůli tomu byla také označena jako nejvhodnější trajektorie pro průjezd chodbou zadaného tvaru.

Závěr

Tato bakalářská práce pojednává o možnostech nalezení optimální trajektorie pro neholonomního robota při průjezdu křižovatkou tvaru „T“.

V první části práce jsou uvedeny metody řešení hledání trajektorie robota od startovního bodu k cílovému. Jsou uvedeny metody pracující v diskrétním a spojitém prostoru pro holonomní i neholonomní roboty.

V rámci druhé části je sepsán algoritmus v programu MATLAB. Algoritmus vypočítá optimální trajektorii pro pohyb robota. Jako kritérium optimality je zvolena bezpečnost při průjezdu chodbou, což je charakterizováno nejnižšími naměřenými vzdálenostmi podvozku robota od zdí chodby. Čím nižší hodnoty se naměří, tím nižší je celková bezpečnost trajektorie. V případě zatáček je trajektorie částí kružnice. Body této části kružnice jsou posunuty v závislosti na poloze jeho osy otáčení a poloměru kružnice.

Výsledná trajektorie je křivka, po které se musí pohybovat střed podvozku robota, aby se robot pohyboval optimálně. Pravdivost tohoto tvrzení byla ověřena aplikací algoritmu na několik dalších robotů o různých rozměrech. Nejmenší vzdálenosti robota od stěn chodby byly napravo i nalevo od robota, v rámci tolerance, stejné. Z toho plyne, že se robot pohyboval uprostřed chodby.

Ve třetí části se hledá trajektorie robota, která se optimální trajektorii co nejvíce blíží. Tato trajektorie prochází předem zvolenými body. Bylo provedeno 15 měření pro stanovení potřebného počtu těchto bodů a také jejich vhodného rozložení na mapě. Nakonec byla jako vhodný kandidát vybrána soustava devíti vodítek.

Vzhledem ke kinematickým omezením robota jsou tyto body voleny tak, aby si robot do každé zatáčky najel a zároveň si zachoval bezpečnou vzdálenost od zdí. Během měření byly zaznamenány vzdálenosti vrcholů podvozku robota od stěn chodby a také vzdálenost stěn robota od rohu, které v zatáčkách objíždí. Správné rozestavění bodů, kterými má robot projet, bylo určeno na základě těchto hodnot.

Seznam použitých zdrojů

[1] LAVALLE, Steven Michael. *Planning algorithms* [online]. New York: Cambridge University Press, 2006 [cit. 2016-05-20]. ISBN 978-052-1862-059.

[2] KAVRAKI, Lydia E. a Steven Michael LAVALLE. Motion Planning. In: SICILIANO, Bruno a Oussama KHATIB. *Springer Handbook of Robotics*. Berlin: Springer-Verlag Berlin Heidelberg, c2008, s. 109-128. ISBN 978-3-540-23957-4. Dostupné z: http://msl.cs.illinois.edu/~lavalle/papers/KavLav08.pdf

[3] LAVALLE, Steven M. a James J. KUFFNER. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and Computational Robotics: New Directions* [online]. 2001, 293-308 [cit. 2016-05-20]. Dostupné z: http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01.pdf

[4] ŠINDELÁŘ, Jiří. *PLÁNOVÁNÍ CESTY NEHOLONOMNÍHO MOBILNÍHO ROBOTU*. Brno, 2010. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství. Vedoucí práce Petr Krček.

[5] KRČEK, Petr. *PLÁNOVÁNÍ CESTY AUTONOMNÍHO LOKOMOČNÍHO ROBOTU NA ZÁKLADĚ STROJOVÉHO UČENÍ*. Brno, 2010. Disertační práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství. Vedoucí práce Jiří Dvořák.

[6] ZHANG, Zheng Ran a Ji Ying YIN. The Study on Mobile Robot Path Planning Based on Frog Leaping Algorithm. *Advanced Materials Research* [online]. 2012, **2012**(490-495), 808-812 [cit. 2017-05-16]. DOI: 10.4028/www.scientific.net/AMR.490-495.808. ISBN 10.4028/www.scientific.net/AMR.490-495.808. Dostupné z: http://www.scientific.net/AMR.490-495.808

[7] ZHU, Qian, Wei SUN, Zhi Wei ZHOU a Su Wei ZHANG. Path Planning for the Chassis of Duct-Cleaning Robot Based on Ant Colony Algorithm. *Applied Mechanics and Materials* [online]. 2012, **2012**(190-191), 715-718 [cit. 2017-05-16]. DOI: 10.4028/www.scientific.net/AMM.190-191.715. ISBN 10.4028/www.scientific.net/AMM.190-191.715. Dostupné z: http://www.scientific.net/AMM.190-191.715

[8] SHI, Pu a Jian Ning HUA. Mobile Robot Dynamic Path Planning Based on Artificial Potential Field Approach. *Advanced Materials Research* [online]. 2012, **2012**(490-495), 994-998 [cit. 2017-05-16]. DOI: 10.4028/www.scientific.net/AMR.490-495.994. ISBN 10.4028/www.scientific.net/AMR.490-495.994. Dostupné z: http://www.scientific.net/AMR.490-495.994

[9] SONG, Li Fei, Ming Rui TANG a Nan LI. A Global Path Planning Algorithm for AGV Based on the Shapefile Vectorgraph. *Advanced Materials Research* [online]. 2014, **2014**(889-890), 1117-1120 [cit. 2017-05-16]. DOI: 10.4028/www.scientific.net/AMR.889-890.1117. ISBN 10.4028/www.scientific.net/AMR.889-890.1117. Dostupné z: http://www.scientific.net/AMR.889-890.1117

[10] HUANG, Min, Ping DING a Jiao Xue HUAN. Global Path Planning for Mobile Robot Based on Improved Ant Colony Algorithms. *Applied Mechanics and Materials* [online]. 2013, **2013**(418), 15-19 [cit. 2017-05-16]. DOI: 10.4028/www.scientific.net/AMM.418.15. ISBN 10.4028/www.scientific.net/AMM.418.15. Dostupné z: http://www.scientific.net/AMM.418.15

[11] Dijkstra-Algorithmus. In: *Informatik-Forum.at* [online]. El Segundo (California): vBulletin, c2016 [cit. 2016-05-20]. Dostupné z: https://www.informatik-forum.at/showthread.php?92172-Dijkstra-Algorithmus

[12] Voronoi Diagrams and a Day at the Beach. *American Mathematical Society* [online]. Providence (RI USA): American Mathematical Society, c2016 [cit. 2016-05-21]. Dostupné z: http://www.ams.org/samplings/feature-column/fcarc-voronoi

[13] KOSECKA, Jana. *Potential Field Methods*. Fairfax (VA). Dostupné také z: http://cs.gmu.edu/~kosecka/cs685/cs685-potential-fields.pdf

[14] Creating all possible polyline routes.. In: *Geographic Information Systems* [online]. New York (NY): Stack Exchange, c2016 [cit. 2016-05-20]. Dostupné z: http://gis.stackexchange.com/questions/53534/creating-all-possible-polyline-routes-connecting-many-point-locations-around-pol

[15] Coverage Research - Biorobotics. *Biorobotics Laboratory* [online]. Pittsburgh: Carnegie Mellon University, c2000-2013 [cit. 2017-05-12]. Dostupné z: http://biorobotics.ri.cmu.edu/research/complete.php

[16] Robot Navigation - Rapidly-Exploring Random Tree Algorithm. In: *Taner Güngör* [online]. TANER GÜNGÖR, 2011 [cit. 2017-05-12]. Dostupné z: http://tanergungor.blogspot.cz/2015/05/robot-navigation-rapidly-exploring.html

[17] Robotika.cz [online]. Grada, 2005 [cit. 2017-05-21]. Dostupné z: https://robotika.cz