Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

DIPLOMOVÁ PRÁCE

Plzeň, 2015

Michal Jirovský

PROHLÁŠENÍ

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

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

V Plzni dne .....

.....

vlastoruční podpis

Poděkování

Na tomto místě bych rád poděkoval Ing. Miroslavu Flídrovi Ph.D za vedení práce a za cenné podněty, které vedly k vypracování této diplomové práce. Rád bych také poděkoval své rodině za podporu během studia.

Abstrakt

Tato práce se zabývá plánováním trajektorie mobilního robotu ve spojitém prostředí. Plánování trajektorie je provedeno pomocí RRT algoritmu a Voroného diagramu. Nalezené trajektorie jsou optimalizovány pomocí kubického interpolačního splinu nebo Bézierovy křivky. V závěru práce jsou uvedeny výsledky získané při simulaci algoritmů.

Klíčová slova

Plánování trajektorie, RRT algoritmus, Voroného diagram, mobilní robot, optimalizace naplánovaných cest

Abstract

This thesis deals with path planning of mobile robot in a continuous environment. Path planning is realize using RRT algorithm and Voronoi diagram. Found paths are optimized using a cubic spline interpolation or Bezier curves. End of work contains the results of simulation.

Keywords

Path planning, RRT algorithm, Voronoi diagram, mobile robot, optimalization of planned paths

Obsah

Seznam obrázků

1 Úvod

Náplní této diplomové práce je úloha plánování trajektorie mobilního robotu. Hlavní náplní práce bylo seznámit se s metodami pro úlohu plánování trajektorie, implementace zvolených metod a simulační ověření metod pro hledání trajektorie.

Plánování trajektorie je v současné době velmi rozvíjející se obor v automatickém řízení. Své uplatnění může nalézt například v medicíně, vojenství nebo průmyslových odvětví. Díky velkému zájmu o tuto problematiku bylo v posledních letech vyvinuto mnoho nových a efektivních algoritmů a některé z nich budou uvedeny v této práci. Hlavním cílem plánování trajektorie je nalezení cesty z počátečního do cílového bodu. Tato cesta nesmí kolidovat s překážkami, které se v prostoru vyskytují a cesta je zároveň v jistém smyslu optimální. Optimální může být z hlediska minimální vzdálenosti, kterou mobilní robot musí urazit nebo minimálního času stráveného průjezdem mezi dvěma body. Aby trasa robotu nekolidovala s překážkami, robot musí znát oblast, ve které se nachází. Tato oblast může být předem určena mapou daného prostředí nebo si robot mapu získává průběžně sám pomocí senzorů.

V první části této práce se čtenář seznámí se základními pojmy a metodami používanými při plánování trajektorie. Ve druhé části jsou podrobně popsány dvě zvolené a jedna modifikovaná metoda plánování. Tyto metody byly zvoleny k implementaci a proto je zde uveden popis samotné implementace nejprve bez řízení konkrétního robotu a poté i s řízením, kdy je stanoven rozměr robotu a cílem je naplánovat bezkolizní trajektorii pro tyto rozměry robotu. Dále je zde také uvedena kapitola zabývající se metodami pro úpravu nalezených cest. Celá práce je zakončena shrnutím výsledků obdržených při simulacích jednotlivých metod.

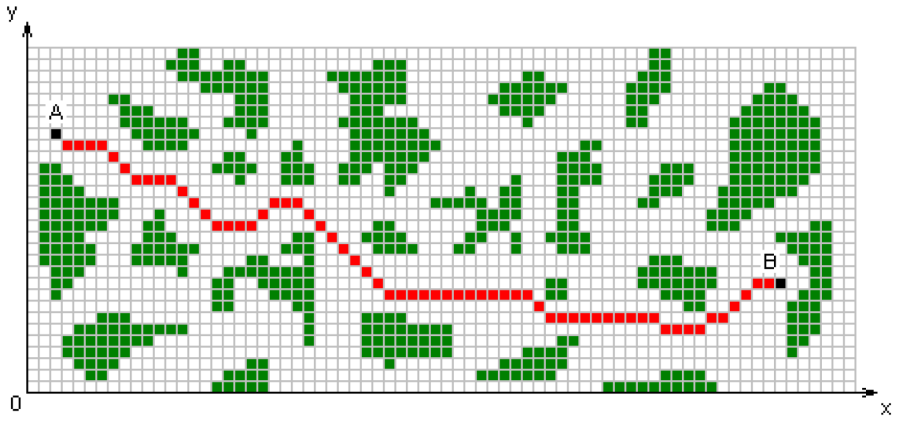
2 Základní pojmy pro plánování trajektorie

V této kapitole bude nadefinováno pár základních a důležitých pojmů, které se budou později využívat pro řešení práce nebo budou součástí metod při zpracování.

Pozici každého reálného objektu lze popsat v určitém prostředí. Pokud se bude jednat o polohu robotu, tímto prostředím je prostor, ve kterém plní robot požadované úkoly. Pro zprostředkování orientace je nutné dané prostředí popsat pomocí zjednodušeného modelu, který by měl být co nejvíce podobný skutečnosti. Čím přesnější je popis takového prostředí, tím větší jsou ale výpočetní nároky a naopak. Reprezentace zmíněného prostoru může být diskrétní nebo spojitá.

Diskrétní prostředí

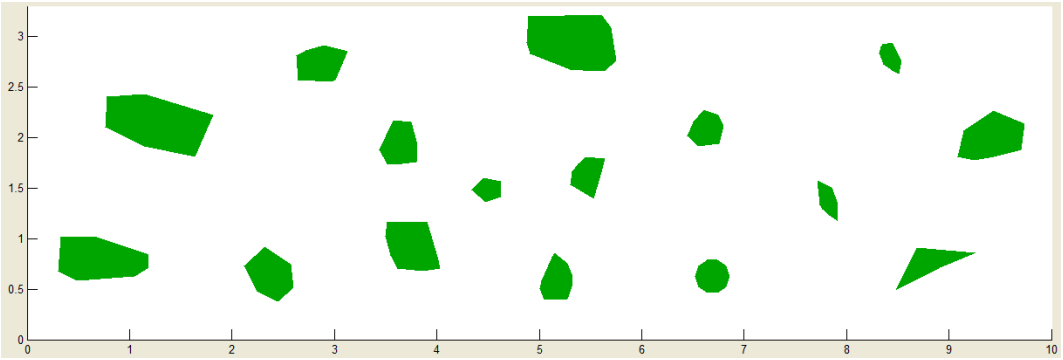
Popis takového prostředí může být zprostředkován pomocí buněk. Počet buněk bude závislý na přesnosti popisu překážek, na velikosti modelovaného prostoru či velikosti robotu. Jak bylo zmíněno výše, se zvyšujícím se počtem buněk dochází k přesnějšímu popisu, ale zároveň se zvýší výpočetní náročnost. Buňky mohou mít různé tvary a velikosti. Ukázka diskrétního popisu se čtvercovými buňkami je na obrázku (2.1).



Obrázek 2.1: Ukázka diskrétně popsaného prostředí

Spojité prostředí

Druhým typem popisu prostředí je spojité prostředí. V tomto případě se jedná o kontinuální prostor, který není rozdělen na jednotlivé buňky jako v diskrétním případě. Díky tomu lze popsat překážky přesněji a navržená cesta se tak bude více blížit skutečnosti. Oproti diskrétnímu prostředí jsou výpočetní nároky při prohledávání náročnější. Tento popis prostředí bude využíván i při dalším zpracování této práce. Ukázka spojitého prostoru je na obrázku (2.2).



Obrázek 2.2: Ukázka spojitě popsaného prostředí

Pracovní prostor

Na začátku kapitoly bylo zmíněno, že se robot pohybuje v jakémsi prostředí. Toto prostředí se nazývá pracovní prostor, který bude značen jako W. Každý takový pracovní prostor lze reprezentovat jako N-rozměrný Euklidovský prostor

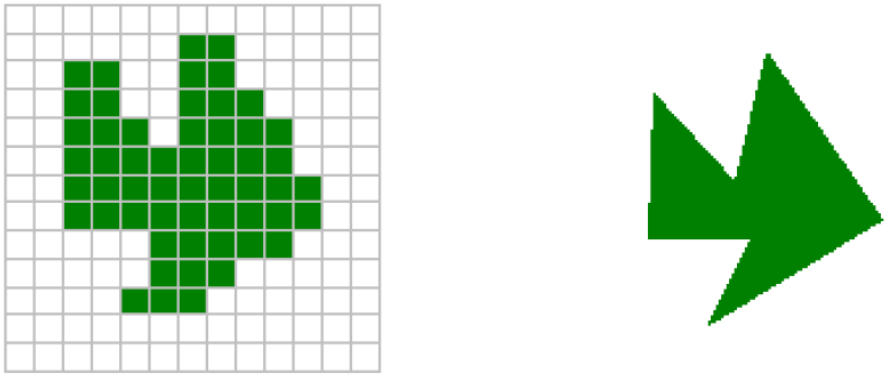
$$W = \mathbb{R}^N, \text{ kde } N \in \{2, 3\} \tag{2.1}$$

Pokud bude v dalších kapitolách řeč o pracovním prostoru, bude používán výhradně pracovní prostor $W = \mathbb{R}^2$. V takto definovaném prostoru se v reálných situacích vyskytují překážky, kterými nelze procházet a robot je musí objet. Tyto překážky patří do množiny O.

Překážky

Překážky vyskytující se v pracovním prostoru W představují libovolné objekty různých velikostí a tvarů, které se vyskytují ve skutečném světě a omezují pohyb robotu. Zobrazení překážek v pracovním prostoru souvisí s reprezentací prostředí. Podle zvoleného způsobu popisu jsou popsány i překážky.

Jejich popis může být bud'to diskrétní nebo spojitý, viz obr. (2.3). Dále lze dělit překážky do několika kategorií. Jednou z nich je například rozlišení statických a dynamických překážek. Statické překážky nemění svou polohu nebo velikost v čase. Naopak dynamické překážky mohou měnit svůj tvar, velikost i polohu v čase, což stěžuje výpočet cesty robotu, který se musí v případě překážky v cestě přizpůsobit podle její aktuální polohy.



Obrázek 2.3: Reprezentace stejné překážky diskrétní a spojitou formou

Konfigurační prostor

Překážky tedy tvoří množiny prostoru, ve kterých se robot pohybovat nemůže. Musí se tedy existovat i opačná množina, která je charakterizována tím, že se v ní robot vyskytovat musí. Tato množina se nazývá volný konfigurační prostor. Konfigurační prostor C je množina všech přípustných stavů robotu, ve kterých se může robot pohybovat. Představme si robota pohybujícího se v prostoru mezi stacionárními, neprůchozími překážkami O. Hlavní myšlenkou konfiguračního prostoru je reprezentace robotu jako bodu a mapování překážek do pracovního prostoru. Oblastí překážek značenou C_{obst} nazýváme takovou část stavového prostoru, která je obsazena překážkami a lze jí reprezentovat vztahem (2.2).

$$C_{obst} = \{q \in C : R(q) \cap O \neq 0\} \tag{2.2}$$

kde $R(q)$ je podmnožina pracovního prostoru W obsazená robotem R s konfigurací q. Vektor q jednoznačně určuje pozici a orientaci robotu v prostoru.

Doplňkem množiny C_{obst} do C je množina C_{free} určující volný konfigurační prostor. Je to prostor všech konfigurací, které nekolidují s žádnou překážkou a které vyhovují všem omezením robotu. Prostřednictvím konfiguračního prostoru je úloha plánování trasy robotu redukována na plánování pohybu uvnitř volného konfiguračního prostoru C_{free}.

Metrika

Dalším užitečným pojmem, bez kterého by se tato práce neobešla se nazývá metrika. Stejně jako v reálném světě má každý objekt své rozměry, tak i zde musí být rozměry zahrnuty. Aby bylo možné využívat algoritmy pro plánování tras, je nutné znát vzdálenost d, která určuje vzdálenost dvou konfigurací robotu v konfiguračním prostoru C. Tato vzdálenost je určena tzv. *metrikou* značenou ρ. Je to zobrazení $\rho : C \times C \rightarrow R$, kdy pro každou konfiguraci q_i, q_j nebo $q_k \in C$ platí následující axiomy:

1) Axiom totožnosti - $\rho(q_i, q_j) = 0$, jestliže $q_i = q_j$

2) Axiom symetrie - $\rho(q_i, q_j) = \rho(q_j, q_i)$

3) Trojúhelníkové nerovnosti - $\rho(q_i, q_k) \leq \rho(q_i, q_j) + \rho(q_j, q_k)$

4) Nezápornost vyplývající z axiomů 1-3 - $\rho(q_i, q_j) \geq 0$

Hodnota $\rho(q_i, q_j)$ je nazývána vzdálenost d.

Metrika je vždy definována pro určitý prostor a vzhledem k tomu, že prostor W byl obecně definován jako N-rozměrný Euklidovský prostor, tak pro výpočet d bude definována tzv. Euklidovská metrika ρ, která popisuje vzdálenost dvou konfigurací v pracovním prostoru nejlépe. Na množině $\mathbb{R}^N$ lze nadefinovat takovou Euklidovskou metriku, která bude vyjadřovat přesnou vzdálenost mezi dvěma body X a Y, pro které platí $X = (x_1, x_2, ..., x_N) \in \mathbb{R}^N$ a $Y = (y_1, y_2, ..., y_N) \in \mathbb{R}^N$. Vzdálenost d se poté vypočte podle vztahu (2.3).

$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_N - y_N)^2} \tag{2.3}$$

Detekce kolizí

Z předchozích pojmů už bylo stanoveno, že pracovní prostor W se skládá z překážek O patřící do množiny C_{obs} a volného konfiguračního prostoru C_{free}. Aby robot při průjezdu prostředím nekolidoval s překážkami, musí v

sobě plánovací algoritmus zahrnovat funkci určující, zda se daná konfigurace robotu nachází ve volném konfiguračním prostoru C_{free}. Tato funkce $D : C_{free} \rightarrow \{True, False\}$ je nazývána detektorem kolizí.

Existuje mnoho metod, jak vyřešit kolize s překážkami. Záleží také na způsobu, kterým je reprezentována samotná překážka. Pokud je popsána diskrétně, detektor pouze vyřeší, jestli je daná buňka volná nebo obsazená překážkou. Zde je ovšem nutné mít co nejlepší popis prostředí, aby nedocházelo k omylům, kdy by detektor vyhodnotil kolizi jako $False$, ale v reálném případě by robot kolidoval s překážkou. V případě spojitého popisu jsou překážky reprezentovány jednotlivými úsečkami. Robot je rozdělen na úsečky (hrany) popisující jeho tvar a ty jsou testovány s úsečkami (hranami) překážek. Pokud některá z hran robotu protíná hranu překážky, je detekována kolize. Tato metoda je vhodná pro jednoduché překážky a výhodou je její použití jak na konvexní tak i na konkávní polygony bez nutnosti převádění konkávních polygonů na konvexní.

3 Základní metody plánování trajektorie

Po úvodní kapitole, kde bylo nadefinováno pár užitečných pojmů se nyní bude práce zabývat konkrétními metodami pro hledání trajektorie v pracovním prostoru. Pro řízení pohybu mobilního robotu je důležité plánování trasy tak, aby byl splněn zadaný požadavek v nejkratším možném čase a s minimálními náklady na cestu. Plánování může být globální nebo lokální. Globální navrhuje trasu robotu z počátečního do cílového bodu, poté robot trasu vykoná. Naopak lokální plánování se provádí během pohybu robotu, kdy robot pomocí senzorů nachází překážky, které nebyly doposud známy a je tak nucen měnit původní trasu. Pro plánování trajektorie robotu byla vyvinuta řada metod. V této kapitole je uveden průřez základních metod pro plánování trajektorie robotu.

Princip metod spočívá ve vytvoření vnitřního modelu pracovního prostoru, ve kterém je trajektorie hledána. Metody pro vytvoření vnitřního modelu pracovního prostoru se dělí na tři základní přístupy:

- Dekompozice buněk
- Mapy cest
- Potenciálová pole

Tyto přístupy budou nyní popsány a dále budou také uvedeny konkrétní metody, které patří do jednotlivých skupin.

3.1 Dekompozice buněk

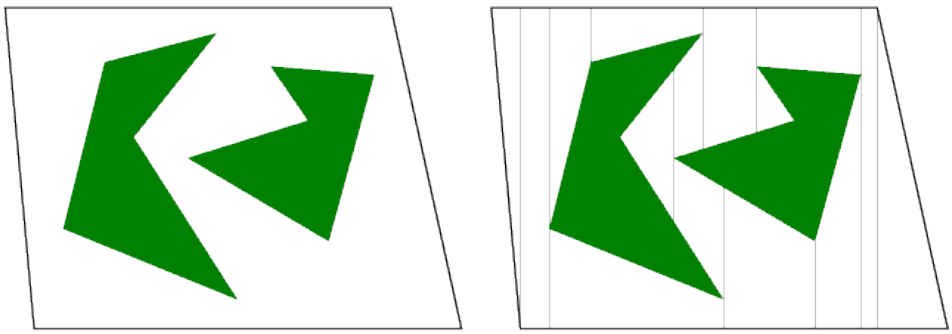
Cílem této metody je dekompozice volného konfiguračního prostoru C_{free} do množiny buněk a vytvoření grafu sousednosti. Graf tvoří vrcholy, které jsou reprezentované volnými buňkami, a hrany mezi dvěma vrcholy tvoří přechod mezi dvěma buňkami. Plánování trasy nebo-li vyhodnocení je poté otázkou nalezení takové posloupnosti buněk, pro které bude platit, že:

- První buňka obsahuje počáteční konfiguraci q_{init}
- Následuje posloupnost vzájemně sousedících buněk
- Poslední buňka obsahuje cílovou konfiguraci q_{goal}

Takto vzniklá posloupnost buněk se nyní transformuje pomocí prohledávacího algoritmu na cestu. Metody dekompozice se dále dělí na dva přístupy - exaktní a aproximativní.

Exaktní dekompozice

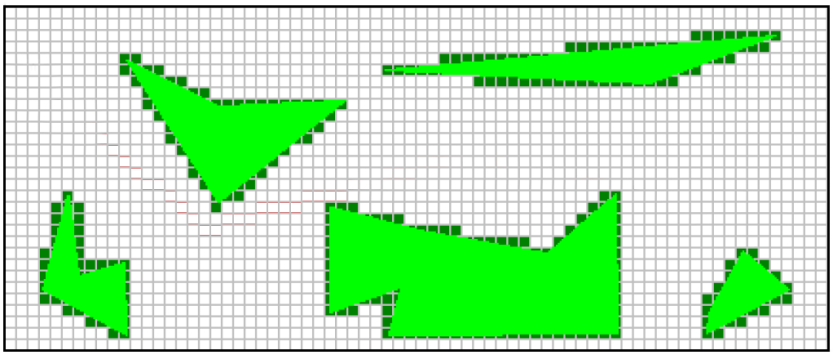
Principem dekompozice je rozložení volného konfiguračního prostoru do sousedících a vzájemně se nepřekrývajících buněk ve tvaru konvexních mnohoúhelníků. Vrcholy mnohoúhelníků jsou tvořeny vrcholy jednotlivých překážek. Existuje mnoho konkrétních variant takového rozkladu, nejčastěji se používá lichoběžníkový nebo trojúhelníkový tvar. Ukázka této dekompozice do lichoběžníkového tvaru je na obr. (3.1).



Obrázek 3.1: Exaktní lichoběžníková dekompozice

Aproximativní dekompozice

U metody aproximativní dekompozice je konfigurační prostor rozdělen do buněk stejného tvaru. Tvar buněk bývá zpravidla čtvercový, ale mohou se vyskytovat výjimky, kdy buňky reprezentují i jiné tvary. V případě dvourozměrného prostoru je pak mapa prostředí tvořena maticí buněk. Ke každé buňce je rekurzivně přiřazena hodnota reprezentující stav v daném místě. Pokud se buňka nachází v místě volného konfiguračního prostoru C_{free} je buňka označena jako prázdná, pokud je zcela překryta překážkou dostává označení obsazená a v případě kombinace částečně volného i obsazeného prostoru se pak jedná o označení kombinovaná. Přesnost takového rozkladu opět hodně závisí na velikosti jednotlivých buněk a jejich celkovém počtu. Trasa je pak v takto nadefinovaném prostoru nalezena pomocí vhodného prohledávacího algoritmu, např.: algoritmus A*, který bude podrobně popisován dále.



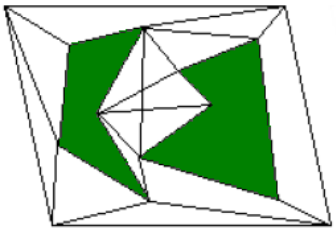
Obrázek 3.2: Aproximativní dekompozice

3.2 Mapy cest

Další skupina metod se řadí do kategorie nazývané mapy cest. Tyto metody vytváří mapu cest v podobě grafu reprezentujícího volný pracovní prostor. Hrany grafu tvoří cesty, po kterých se může robot pohybovat. Vrcholy těchto map jsou pak tvořeny opět vrcholy překážek a počáteční a cílovou konfigurací. Za použití vhodně zvoleného algoritmu je pak hledána cesta v grafu.

Graf viditelnosti

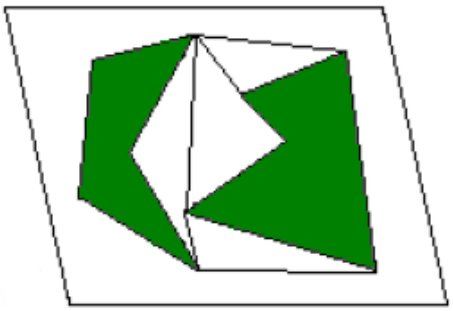
Tuto metodu lze zvolit, pokud se jedná o konfigurační prostor reprezentovaný polygony. V takovém případě je možné zobrazit viditelnostní vztahy mezi vrcholy polygonů. Každý vrchol polygonu může být spojen s jiným vrcholem polygonu pomocí hrany a to v případě, že je z jednoho vrcholu vidět druhý vrchol. Obecně řečeno - aby mezi dvěma vrcholy mohla vzniknout hrana, nesmí se na jejich spojnici vyskytovat žádná překážka. Výsledná cesta je pak určena pomocí posloupnosti hran grafu z počáteční do cílové konfigurace. Pokud by byl prostor reprezentován nepolygonálními překážkami, bylo by nutné tyto tvary převést na polygony, které nejvíce odpovídají původnímu tvaru. Graf viditelnosti je vyobrazen na obr. (3.3).



Obrázek 3.3: Graf viditelnosti

Graf tečen

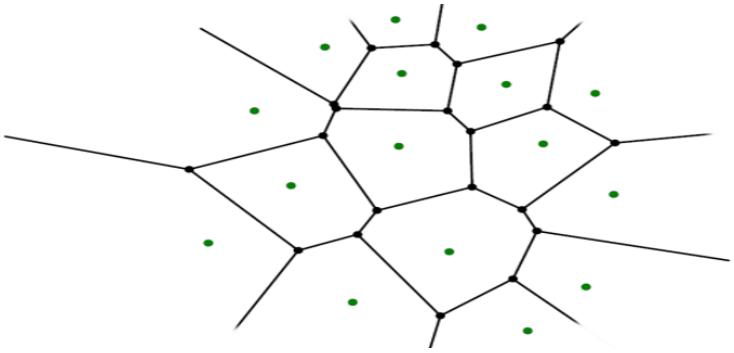
Graf viditelnosti ve skutečnosti obsahuje velké množství nadbytečných hran a proto vznikla metoda grafu tečen. V tomto případě je graf tvořen pouze takovými hranami, které jsou tečnami mezi polygony. To znamená, že mohou "jít za roh", takže pokud je hrana mezi vrcholy prodloužena, nesmí kolidovat s překážkou, přičemž hranice konfiguračního prostoru je považována také za překážku.



Obrázek 3.4: Graf tečen

Voroného diagram

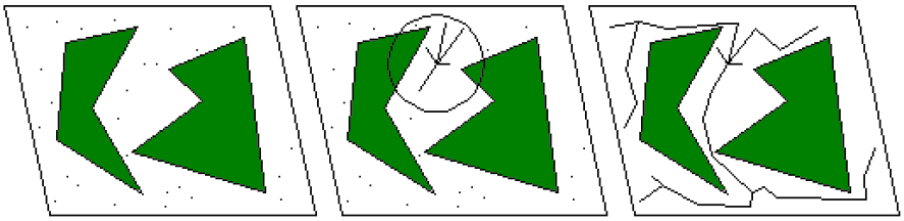
Tato metoda popsaná v článku [8] je specifická v tom, že předpokládá nenulový rozměr robotu ba dokonce rozměr takový, že překážky lze v základní verzi tohoto algoritmu reprezentovat pouze pomocí bodů. Diagram tvoří geometrická struktura, která je generována na základě bodů umístěných ve stejných vzdálenostech od dvou nebo více překážek. Tyto centroidní body tvoří vrcholy diagramu a hrany tvoří propojení těchto bodů. Robot se tak může pohybovat bez kolize s překážkou. Pokud se do grafu zahrne i počáteční a cílová konfigurace, je možné spočítat nejkratší bezkolizní vzdálenost mezi těmito překážkami. V reálu ovšem nelze předpokládat minimální velikosti překážek oproti robotu a tak existují i jisté možnosti výpočtu segmentu mezi dvěma polygony. Výpočtem Voroného diagramu se bude podrobněji zabývat kapitola 5, protože tato metoda byla vybrána ke zpracování. Na obrázku (3.5) je vidět Voroného diagram pro překážky reprezentované pomocí zelených bodů.



Obrázek 3.5: Voroného diagram

Pravděpodobnostní mapy cest

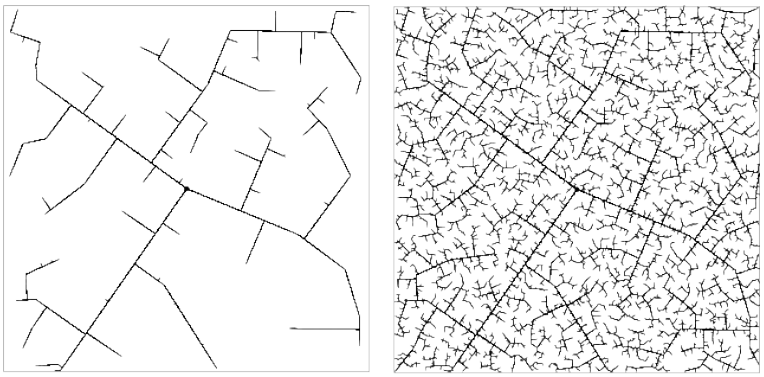
Metoda plánování podle pravděpodobnostní mapy cest se skládá ze dvou částí. V první části, která se nazývá učící, je základní motivací náhodné rozprostření bodů do konfiguračního prostoru. Jediná nutná podmínka je, aby se tyto body vyskytovaly ve volném konfiguračním prostoru. V závislosti na počtu těchto bodů (volí uživatel) je dobré, aby pokrývaly co nejlépe pracovní prostor. Čím více je takovýchto bodů zvoleno, tím lepší výsledky budou dosaženy. Při každém nově vygenerovaném bodu se testuje, zdali je možné jej přímo propojit s již existujícím bodem. Jestliže nenastane kolize a body jsou od sebe vzdáleny méně než je povolená maximální vzdálenost, jsou tyto dva body propojeny. V odborném názvosloví budou náhodné body opět nazvány jako vrcholy a propojení mezi vrcholy nazvány jako hrany. Po propojení je obdržen graf, pomocí kterého je možné při definici počáteční a cílové konfigurace určit cestu průjezdu konfiguračním prostorem. Základní princip je znázorněn ve třech krocích na obrázku (3.6).



Obrázek 3.6: Pravděpodobnostní mapa cest

Pravděpodobnostní stromy

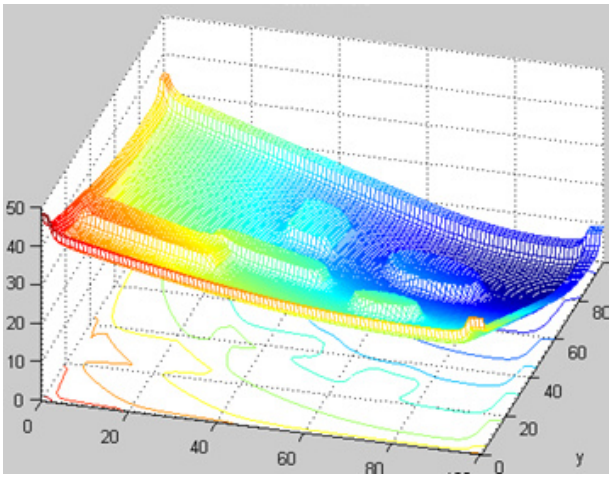
U této metody je základním cílem vytváření prohledávacího stromu, který rovnoměrně prohledává volný pracovní prostor C_{free} a snaží se najít cílovou konfiguraci. Vytváření stromu je součástí iteračního cyklu, který vygeneruje novou náhodnou konfiguraci a ta je propojena s nejbližší existující konfigurací, kterou strom obsahuje a je zařazena do stromu vygenerovaných konfigurací. Tyto metody jsou navrženy pro neholonomní plánování cesty robotů, kteří mají vysoký počet stupňů volnosti. Podrobněji se touto metodou v podobě RRT algoritmu bude zabývat kapitola 4. Na obrázku (3.7) je vidět struktura stromu při 45 iteracích a 2345 iteracích.



Obrázek 3.7: Pravděpodobnostní mapa cest

3.3 Potenciálová pole

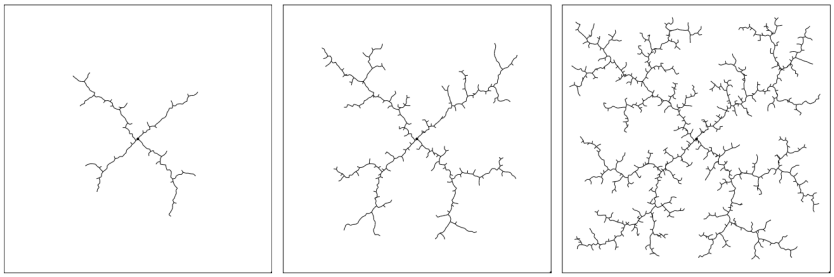
Cílem metody je pokrytí konfiguračního prostoru potenciálovým polem, které je definováno funkcí $E(x, y)$ určující míru potenciálu v daném místě. Počáteční konfigurace robotu je ohodnocena vyšší hodnotou potenciálu, než je cílová konfigurace, která je ohodnocena globálním minimem potenciálu. Prostory s překážkami C_{obs} jsou ohodnoceny vyšším potenciálem, než má volný prostor C_{free}. Nalezení cesty spočívá v nalezení globálního minima. Robot se tedy bude pohybovat ve směru záporného gradientu. Nevýhodou této metody je možnost uvíznutí v některém z lokálních minim. Touto metodou se zabývá článek [7]. Ukázka potenciálového pole je na obrázku (3.8).



Obrázek 3.8: Ukázka potenciálového pole aplikovaného na pracovní prostor

4 Metody plánování založené na metodě RRT

Pro implementaci metod plánování trajektorie byly zvoleny dva přístupy. V této kapitole bude podrobně popsán přístup pomocí rychle rostoucích stromů a jeho modifikace. Název této metody vznikl z převzetí anglického spojení Rapidly exploring Random Tree (zkráceně RRT). Algoritmus byl poprvé představen v roce 1998 Stevem LaVallem, článek [1]. Technika algoritmu byla charakterizována jako náhodně vytvořená datová struktura pro prohledávání n-dimenzionálního, stavového, nekonvexního prostoru značeného C, která slouží pro plánování trajektorií. I když je RRT vhodný pro širokou skupinu problémů plánování cesty, navržen byl především se zaměřením na neholonomní plánování, včetně dynamických omezujících podmínek, robotů s vysokým počtem stupňů volnosti. Myšlenkou je postupné vytváření nových uzlů, které tvoří prohledávací strom a snaží se rychle a rovnoměrně prohledat předem nakonfigurovaný stavový prostor. Pokud je vytváření uzlů stromu náhodné, jedná se právě o strategii RRT. Klíčovou výhodou RRT je rychlé rozrůstání k neprozkoumaným oblastem konfiguračního prostoru C. Na obrázku (4.1) je zaznamenán vývoj RRT stromu s přibývajícími iteracemi výpočtu.

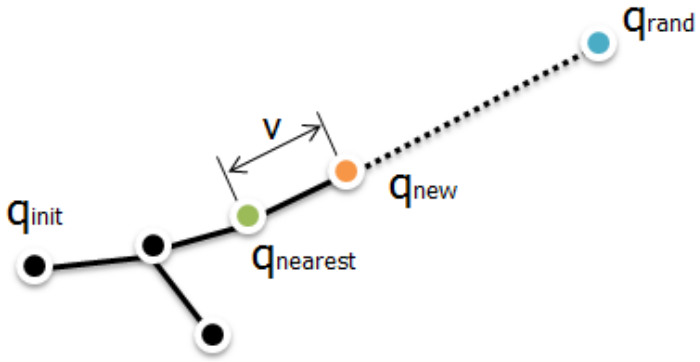


Obrázek 4.1: Vývoj rozrůstající se struktury RRT

4.1 Základní princip metody RRT

Výpočet RRT začíná vytvořením základního uzlu ve startovní konfiguraci q_{init} a během expanze se snaží nalézt cílovou konfiguraci q_{goal}. V každém kroku se náhodně vygeneruje konfigurace q_{rand}, od které se hledá nejbližší sousední bod $q_{nearest}$ existujícího RRT stromu. Nový bod q_{new} je generován ve vzdálenosti v od nejbližšího souseda ve směru náhodně vygenerované konfigurace

q_{rand}. V průběhu hledání jsou vrcholy stromu tvořeny takovým způsobem, že všechny uzly náleží C_{free}. Z tohoto důvodu musí existovat možnost testovat, zda určitý stav neleží v C_{obs}. Stavy v C_{obst} mohou mít v závislosti na aplikaci různý význam. Nejčastěji C_{obs} představuje konfiguraci, kdy je možná trasa v kolizi s překážkou. Pro možnost zachování uzlu q_{new} ve stavu C_{free} je nutné testovat, jestli samotný q_{new} není v kolizi s překážkou a také jestli s překážkou nekoliduje propojení těchto dvou konfigurací. Pokud testovaný uzel vyhoví těmto podmínkám, je zařazen do stromové struktury. Pokud uzel testování neprojde, je vyřazen a proces rozrůstání RRT pokračuje. V případě, že se nově vytvořený uzel q_{new} přiblíží cílové konfiguraci q_{goal}, je testováno, zdali je možné propojení těchto konfigurací. Pokud se v cestě spojení vyskytuje překážka nebo konfigurace q_{goal} není ve vzdálenosti d od q_{new}, je nutné pokračovat v generování dalších konfigurací q_{rand} a následně uzlů q_{new}, dokud není podmínka propojení bez kolize, či blízké vzdálenosti ke q_{goal} splněna. Proces vytváření RRT struktury je vyobrazen na obrázku (4.2).



Obrázek 4.2: Proces vytváření struktury RRT

Algoritmus RRT

Nyní bude popsán obecný algoritmus 4.1, který slouží jako podklad pro implementaci ve výpočetním programu Matlab. Vstupem algoritmu je počáteční stav q_{init}, cílový stav q_{goal} a definice konfiguračního prostředí C. Výstupem je výsledná trajektorie, pokud je nalezena.

Nejprve je nutné vytvořit mapu prostoru, ve kterém se bude trasa hledat. Tato mapa je součástí proměnné *mapa*. Pro vytvoření mapy je nutná znalost parametrů jako jsou rozměry mapy, počet a velikosti překážek nebo výběr jedné z předdefinovaných map. Podrobněji se na vytvoření mapy pracovního

prostoru bude zaměřovat kapitola 7.2. Na dalších dvou řádcích jsou proměnné pro přiřazení počáteční a cílové pozice, mezi kterými se bude cesta hledat. V samotné implementaci umožňuje uživateli pomocí příkazu *ginput* zvolit počáteční a cílový bod v mapě. Výstupem jsou souřadnice popisující volbu bodu, které jsou uloženy do struktury stromu *tree*, kam se později budou zapisovat souřadnice vygenerovaných bodů a také ceny cest od startovního bodu q_{init} a odkaz na předchůdce k novým bodům. Následuje cyklus *while*, který se opakuje do doby, dokud není nalezena trasa mezi q_{init} a q_{goal} nebo dokud není překročen maximální počet iterací. Existuje totiž možnost, že mezi body nelze nalézt žádnou trasu. Jediným příkazem, který je vykonáván uvnitř této smyčky je volání funkce pro expanzi stromu, která strom rozšiřuje o další nově vygenerovaný bod, který je zařazen do struktury *tree*. Pokud je nově vygenerovaný bod q_{new} možné spojit s cílovým bodem q_{goal}, cyklus *while* je ukončen. V opačném případě pokračuje expanze stromu další iterací. V poslední části kódu se při nalezení trasy pomocí funkce pro hledání cesty určí cesta, která propojí startovní a cílový bod. V opačném případě může být vypsána varovná hláška, že žádná trasa nebyla nalezena. V případě složitě definovaného prostředí s mnoha překážkami neznamená, že trasa mezi body neexistuje, ale že mohl být překročen maximální počet expanzí stromu *tree*, což je druhá podmínka pro ukončení smyčky *while*.

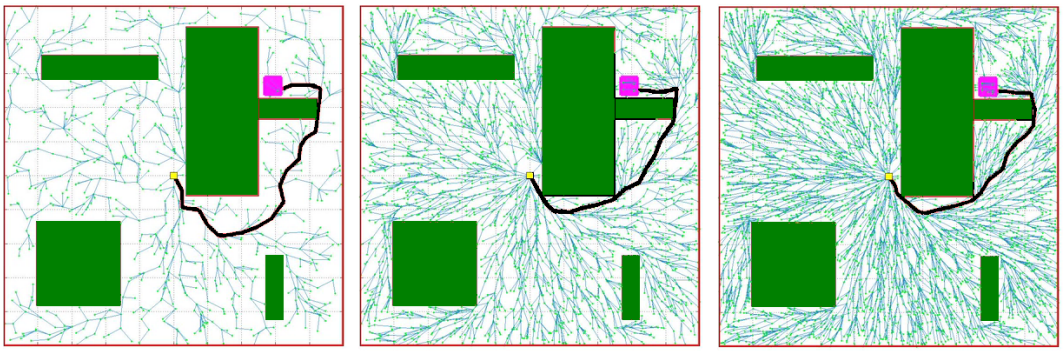
4.1: RRT

```
1  mapa = vytvoreni pracovniho prostoru
2  q_init = urceni pocatecni pozice
3  q_goal = urceni cilove pozice
4
5  while (neni nalezen cilovy bod nebo neni prekrocen
       pocet iteraci)
6      expandujStromRRT()
7  end
8
9  if(cilovy bod nalezen)
10     najdi minimalni cestu a vykresli
11 else
12     informuj uzivatele, ze trasa nebyla nalezena
13 end
```

4.2 Algoritmus RRT*

Algoritmus RRT* je rozšířenou modifikací základního algoritmu RRT. I tato modifikace byla testována a v další části práce bude porovnávána s ostatními metodami a proto budou nyní uvedeny odlišnosti od základní verze RRT a způsob výpočtu trasy. Tato metoda vychází z článku [5].

Metoda pracuje na stejném principu generování náhodných uzlů, které následně tvoří strom. Rozšířením této verze oproti verzi základní je výpočet ceny z počátečního do aktuálního uzlu a následné přepočítání, zdali neexistuje v okolí nově vygenerovaného uzlu další uzel, jehož cena by případným spojením s nově vytvořeným uzlem byla menší, než cena aktuálního propojení. Základní verze algoritmu RRT ceny spojení, které byly již dříve stanoveny nepřepočítává. Na rozdíl od RRT metoda RRT* neukončí výpočet prohledávání při nalezení cesty. Výpočet je ukončen dosažením maximální iterace. Na obrázku (4.3) je vidět, že čím vyšší iterace algoritmu právě probíhá, tím kratší je díky přepočítávání cesta ze startovního do libovolného uzlu generovaného stromu. To platí i pro spojení startovní - cílový uzel, kde se výsledná cesta se zvyšující se iterací zkracuje na minimum.



Obrázek 4.3: Vývoj rozrůstající se struktury RRT*

Princip metody RRT*

Protože proces vytváření struktury tohoto algoritmu je stejný jako jeho základní verze, není nutné jej opět popisovat a text se může rovnou zaměřit na popis celého algoritmu.

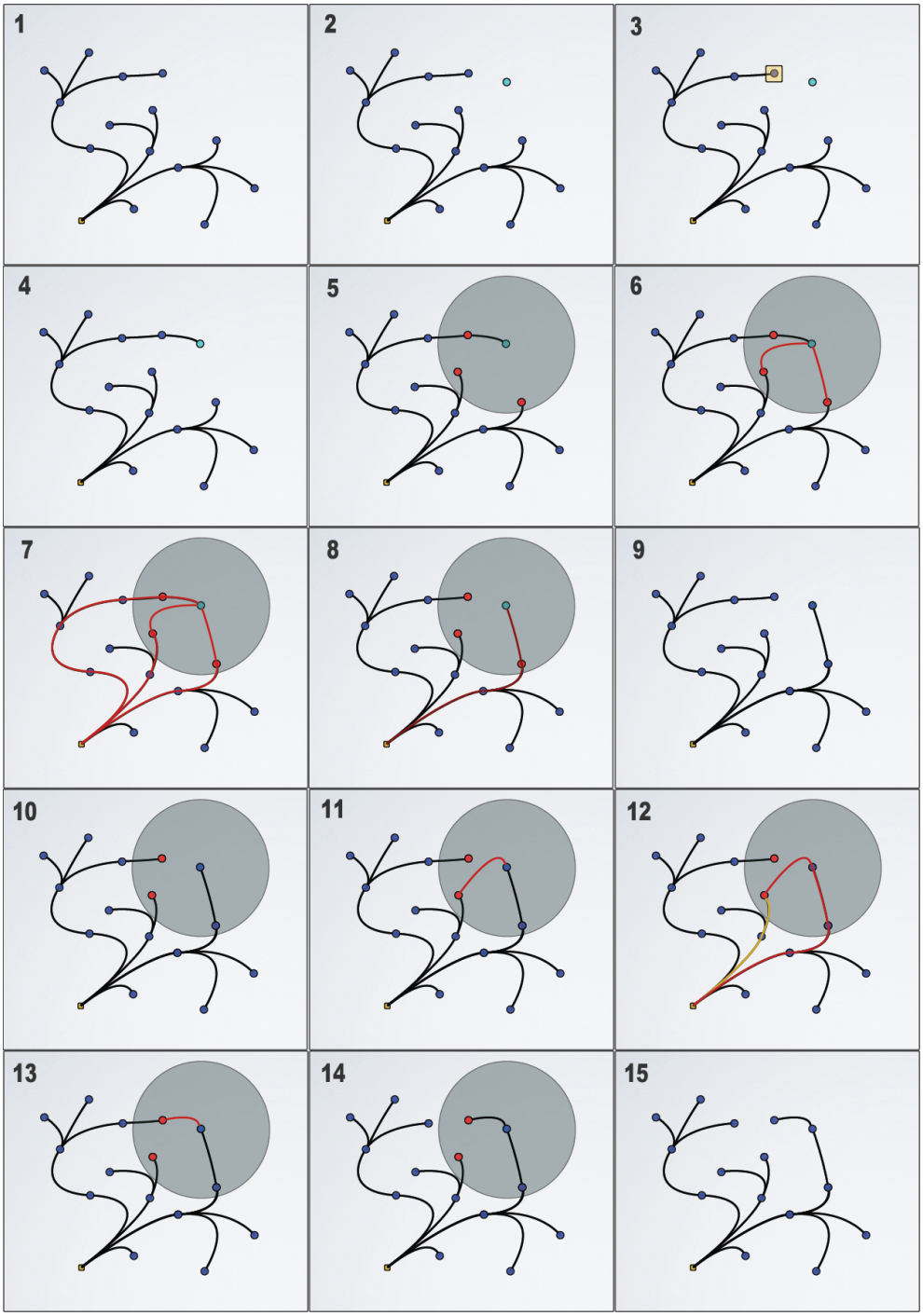
Začátek samotného algoritmu RRT* se neliší od základního algoritmu RRT. Nejdříve je vytvořen nebo načten pracovní prostor a je zvolena počá-

teční inicializace uzlů q_{init} a q_{goal}. Při probíhající expanzi uzlů se opět vygeneruje náhodný uzel q_{rand} a hledá se nejbližší sousední uzel $q_{nearest}$, ke kterému je dále ve vzdálenosti v a ve směru k bodu q_{rand} vygenerován uzel q_{new}. Po připojení q_{new} ke $q_{nearest}$ nastává změna, kdy se hledají všechny uzly značené q_x ve vzdálenosti v od q_{new}. Pokud žádné takové uzly nejsou výpočet probíhá dále podle algoritmu RRT beze změny. Pokud jsou ovšem v této vzdálenosti další uzly, je nutné zjistit, jestli není cena cesty nějakého z nalezených uzlů nižší, pokud by byl propojen s q_{new}. To se zjistí tak, že se k ceně cesty do q_{new} přičte teoretická cena cesty z q_{new} do q_x. Dále se porovná nově navržená cena do q_x s cenou aktuální. Pokud je nová cena nižší, je přepsán předchůdce q_x na q_{new}. Pokud je stará cena nižší, než nově navrhovaná, změna je zamítnuta a předchůdce q_x zůstává stejný. Tento výpočet je posléze proveden pro všechny nalezené uzly ve vzdálenosti v od q_{new}. Když jsou vyčerpány všechny nalezené možnosti uzlů q_x, algoritmus pokračuje další iterací. Princip expanze a přepočítávání je názorně zobrazen na obrázku (4.4).

Algoritmus RRT* 4.2 se liší od základní verze 4.1 opravdu pouze v detailech. Ukončení smyčky *while* proběhne pouze za podmínky, že byla dovršena poslední iterace výpočtu, která byla uživatelem dovolena. Aby bylo vidět, že se nalezená cesta s přibývajícími iteracemi neustále zlepšuje, je možné smyčku *while* rozšířit o vykreslení aktuální, nově nalezené trasy. Tím se naskytuje možnost upravit podmínku za smyčkou *while*, která bude kontrolovat pouze případ, že by nebyla nalezena žádná cesta mezi q_{init} a q_{goal}.

4.2: RRT*

```
1  mapa = vytvoreni pracovniho prostoru
2  q_init = urceni pocatecni pozice
3  q_goal = urceni cilove pozice
4
5  while (neni prekrocen pocet iteraci)
6      expandujStromRRT()
7      if(nalezena nova trasa)
8          vykresli novou trasu
9      end
10 end
11
12 if(cilovy bod nenalezen)
13     informuj uzivatele, ze trasa nebyla nalezena
14 end
```



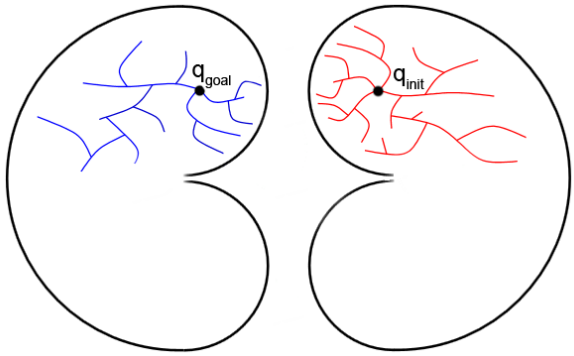
Obrázek 4.4: Princip výpočtu RRT*, Zdroj [5]

4.3 Ostatní modifikace algoritmu RRT

Modifikace základní verze RRT uvedená v předchozí sekci není modifikací jedinou. Existuje mnoho dalších úprav základního algoritmu RRT. Dvě z mnoha dalších variant jsou níže uvedeny. Tyto níže uvedené modifikace jsou zde zařazeny pouze přehledově a nebyly již voleny pro implementaci. Další metody jsou uvedené v práci [17].

RRT Bidirect

První takovou modifikací, která je zde uvedena, se nazývá RRT Bidirect, což v překladu znamená obousměrný. V tomto případě jsou vytvářeny současně dva stromy a snahou je nalézt vhodné propojení těchto dvou stromů pomocí takových uzlů, které jsou od sebe vzdáleny méně, než je maximální povolená vzdálenost spojení. Počátečními konfiguracemi stromů jsou počáteční a cílový stav robotu. Tento algoritmus se používá z pravidla v takových mapách, kde je jak k počátečnímu, tak k cílovému stavu složitý přístup. Oba tyto stavy mohou být například obklopeny překážkou a přístup je možný pouze pomocí malé přístupové cesty. Tento případ je znázorněn na obrázku (4.5).



Obrázek 4.5: Ukázka použití RRT Bidirect

Hlavním principem vytváření těchto stromů je vygenerování náhodného bodu q_{rand} a k tomuto bodu se pak oba stromy rozšiřují. V některých případech nastává problém při propojování stromů. Z hlediska kinematiky některých robotů totiž není možné cestu v místě spojení projet. Řešení by mohlo spočívat například ve snížení maximální možné vzdálenosti propojení těchto stromů, potom by ale nalezení dvou uzlů, které by stromy propojily, bylo časově náročnější. Další variantou je použití aproximace, která by navrhovanou trajektorii upravila tak, aby jí robot mohl bezproblémově aplikovat.

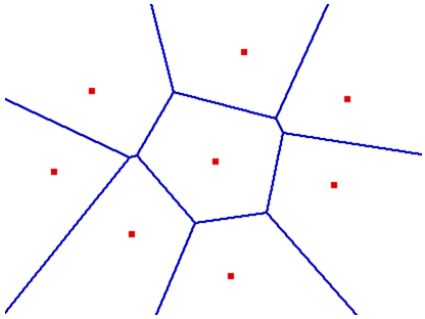
RRT Connect

Další modifikace se v odborné literatuře nazývá RRT Connect, což znamená v překladu spojení. Odlišnost od základního algoritmu je v tom, že po vygenerování náhodného bodu q_{rand} je snahou rozšiřovat prohledávací strom, dokud se ke q_{rand} nepřiblíží. V případě dosažení náhodného bodu algoritmus pokračuje další iterací. Nastat může ale i opačný případ, kdy se strom k náhodnému bodu nerozšíří a to z důvodu možnosti propojení s cílovým stavem (v tomto případě je algoritmus ukončen) nebo kolize s překážkou. V případě kolize jsou ve stromu nově vygenerované uzly ponechány až do doby kolize a algoritmus pokračuje další iterací. Aplikace algoritmu je vhodná pro konfigurační prostor, který obsahuje menší počet překážek. Tento algoritmus je možné aplikovat spolu s algoritmem RRT Bidirect.

5 Voroného diagram

Další metodou, která byla vybrána k implementaci je metoda Voroného diagramu. Tato metoda vznikla v roce 1908, kdy se jistý matematik Georgiji Feodosjeviči Voroné zabýval studií zobecněné n-rozměrné formy dělení prostoru. Podle tohoto matematika byl také Voroného diagram pojmenován. Metoda se zabývá problematikou dělení prostoru v závislosti na zadané množině bodů, které tvoří překážky v prostoru. Od prohledávání pomocí RRT se liší tím, že se nejprve pomocí diagramu rozdělí celý pracovní prostor a poté je hledána optimální cesta mezi dvěma body. Podrobněji se touto metodou zabývají také články [8] a [13].

Dělení prostoru probíhá obecně tak, že ke každému bodu p_i z množiny $P = p_1, ..., p_n$, kde $2 < n < \inf$ je přidělena oblast bodů $V(p_i)$ tak, že každý libovolný bod patřící do oblasti $V(p_i)$ byl z pohledu euklidovské vzdálenosti blíže k bodu p_i než k jakémukoliv jiném bodu z množiny P. Průnikem dvou oblastí $V(p_i)$ a $V(p_j)$ jsou body, mající stejnou vzdálenost od p_i a p_j a tato vzdálenost je nejkratší od všech zbylých bodů z množiny P. Právě tyto průniky mezi všemi oblastmi tvoří zmiňovaný Voroného diagram. Vše je názorně vidět na obrázku (5.1), kde jsou červeně vyznačeny body z množiny P a modře je vyobrazen Voroného diagram přičemž v místech, kde se diagram větví se jedná o vrcholy grafu a ty jsou spojeny modrými hranami.

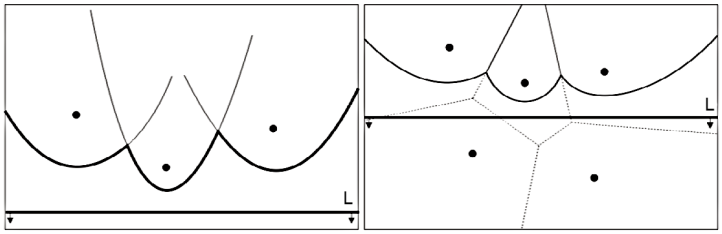


Obrázek 5.1: Struktura Voroneho diagramu

Dále bude vysvětleno, jak se takový diagram může spočítat. Existuje mnoho metod, jak lze najít Voroného diagram. Mezi nejznámější metody patří: Zametací algoritmus, Inkrementální algoritmus, Delaunayho triangulace, Metoda zdvihu, Metoda kružnic nebo třeba metoda Rozděl a panuj. Tato práce se zaměří na Zametací algoritmus, který bývá někdy nazýván jako Fortunův algoritmus.

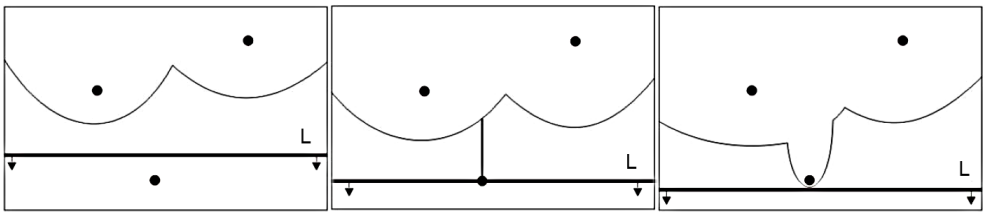
5.1 Zametací algoritmus

Jak již název napovídá, algoritmus je zaměřen na „zametání“ roviny. Po rovině se pohybuje přímka L ve směru od shora dolů přes všechny body z množiny P. Během posunu se udržuje informace o části již vytvořené oblasti $V(P)$ nad přímkou L, která nemůže být dále ovlivněna body, kterými přímka L teprve projde. Pro polohy bodů z množiny P, které leží pod aktuální polohou přímky, není dostupná žádná informace. Oblast $V(p)$ nad L je ohraničena sekvencí parabol. Body tvořící tuto parabolu mají stejnou vzdálenost od L a bodu p_i. Průsečíky jednotlivých parabol na L tvoří hrany $V(P)$. S každým dalším posunem přímky se vytvoří nové paraboly nad L a jejich pozměněné průsečíky budou opět tvořit hrany $V(P)$. Během posunu přímky L se udržuje sekvence parabol. Ukázka výpočtu sekvencí parabol je zobrazena na obrázku (5.2). Změna této sekvence nastává v okamžiku vzniku nové paraboly nebo zániku paraboly existující. Metoda zametacího algoritmu nebo také Fortuneho algoritmu je popsána v článku [13]



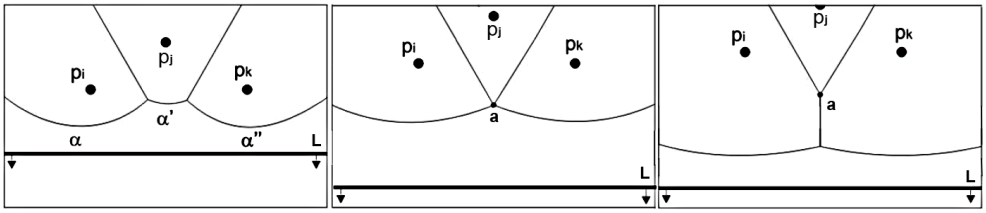
Obrázek 5.2: Vpravo sekvence parabol kolem bodů z P, vlevo sekvence parabol s naznačeným budoucím Voroného diagramem

Nová parabola o nulové šířce vznikne v okamžiku, kdy L dosáhne dalšího bodu, jedná se tedy pouze o vertikální úsečku spojující nový generátor se sekvencí parabol. Průsečík této nově vzniklé paraboly (úsečky) s parabolou ze sekvence parabol udává počáteční pozici nově vznikající hrany $V(P)$. Při dalším posunu L se tato parabola dále rozšiřuje (viz 5.3) a opět postupně mění tvar sekvence parabol jak bylo popsáno v předchozím odstavci.



Obrázek 5.3: Vpravo vznik nové sekvence parabol po zahrnutí nového bodu

Další možností je zánik jedné z parabol. Taková událost může nastat tehdy, pokud jsou nad L tři body z množiny P a prostřední bod má největší vzdálenost od L. Celá situace je znázorněna na obrázku (5.4). Vzniklý bod a má stejnou vzdálenost od L a od všech tří bodů p_i,p_j a p_k. Lze tedy říci, že existuje kružnice taková, která má střed v a a prochází těmito třemi body a přímka L je tečnou v její nejnižší části. Zároveň se uvnitř kružnice nemůže vyskytovat žádný jiný bod z množiny P, protože by byl blíže k bodu a než a k L a to by odporovalo faktu, že bod a leží na sekvenci oblouků parabol. Nově vzniklý bod a je vrcholem grafu Voroného diagramu.



Obrázek 5.4: Změna sekvence parabol - zánik paraboly

5.1: Voroného diagram

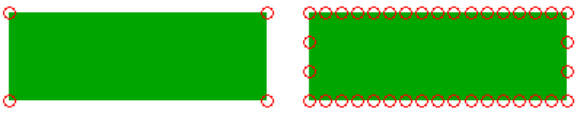
```
P = vektor s body definujicimi prekazky
S = prazdny, dvojite souvisly seznam
T = prazdny binarni vyhledavaci strom
Q = fronta s body z P serazena sestupne podle
   souradnice Y

while (Q je neprazdna)
  vyber udalost s nejvetsi prioritou z Q
  if je vybrana udalost pi bod z mnoziny P
    proved algoritmus zpracuj_udalost_bodu(pi)
  else
    proved algoritmus zpracuj_udalost_kruznice(pi)
  end
end

najdi dostatecne velky ctverec zobrazujici vsechny
   vrcholy V-diagramu
najdi pruniky nekonecnych hran diagramu s timto
   ctvercem
uprav dvojite souvisly seznam S pro hrany majici
   prunik se ctvercem
```

```
18 function zpracuj_udalost_bodu(pi)
19  if T je prazdny
20    vloz pi do T
21  else
22    najdi v T list tvorici parabolu nad bodem pi
23  end
24  if list reprezentujici parabolu obsahuje odkaz na
       kruhovou udalost
25    vymaz tuto kruhovou udalost z fronty udalosti Q
26  end
27  nahrad v T list reprezentujici parabolu podstromem se
       tremi listy
28  proved vyvazeni stromu T
29  vytvor zaznam do S pro hrany, ktere budou vytvareny
       pruseciky danych dvou parabol
30  zkontroluj vznik nove trojice bodu tvorici kruhovy
       bod
31  if vznik nove trojice
32     pridej novy kruhovy bod do Q
33     do T pridej odkaz noveho bodu k listu
          reprezentujicimu novou parabolu
34  end
35
36 function zpracuj_udalost_kruznice(pi)
37  vymaz z T list tvorici parabolu, ktera zanika
       pruchodem bodu pi
38  proved vyvazeni stromu T
39  z Q vymaz udalosti tvorene pomoci zanikle paraboly
40  v S vytvor zaznam pro vrchol danny stredem kruznice
41  vytvor zaznamy v S pro hrany, ktere jsou urceny
       novymi pruseciky parabol
42  v S uprav zaznamy tri hran koncicich ve vzniklem
       vrcholu
43  zkontroluj vznik nove trojice bodu tvorici kruhovy
       bod
44  if vznik nove trojice
45     pridej novy kruhovy bod do Q
46     do T pridej odkaz noveho bodu k listu
          reprezentujicimu novou parabolu
47  end
48 end
```

5.2 Voroného diagram pro polygony

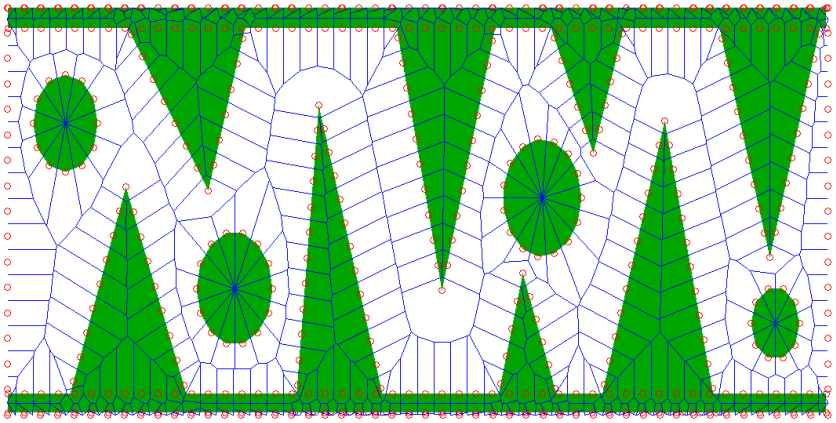
V předchozích částí této kapitoly bylo popsáno, co je Voroného diagram a jak se vypočítá pro množinu bodů, které bude obklopovat. Bohužel, v této práci nejsou překážky tvořené pomocí bodů, ale pomocí různých polygonů a kružnic. Je tedy nutné upravit výpočet tak, aby získaný diagram správně obklopoval daný pracovní prostor s překážkami. Výpočet vychází z práce [13].

První úprava se netýká algoritmu samotného, ale definování překážek. Pokud by byly polygony reprezentovány pouze svými vrcholy, při výpočtu diagramu by byla nízká hodnota informace o tvaru překážky, protože vrcholy by od sebe byly umístěné příliš daleko. Je tedy nutné překážky definovat tak, aby byly popsány pomocí více vrcholů. Vzdálenost vrcholů je určena pomocí parametru K. Řešení problému je vidět na obrázku (5.5). Vlevo je zobrazena překážka v základní struktuře a vpravo je rozšířený popis překážky pomocí více vymezujících bodů.



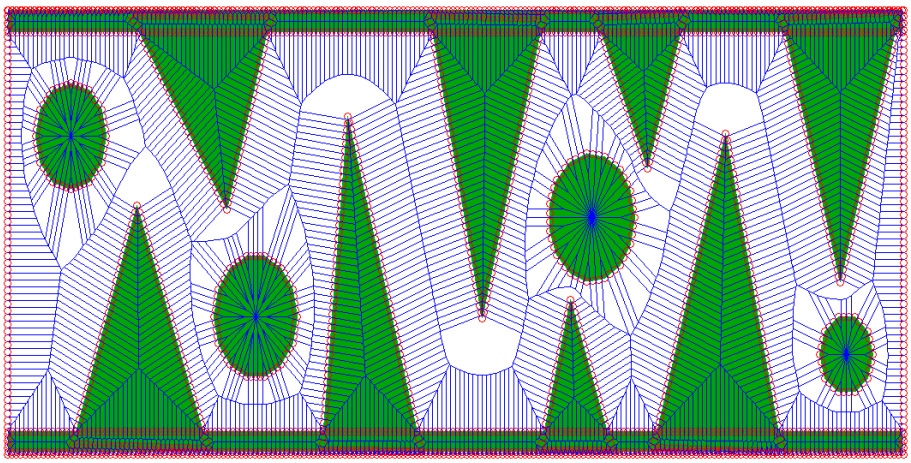
Obrázek 5.5: Struktury překážky na základě velikosti parametru K

Pokud bude nyní aplikován zametací algoritmus na vzorovou mapu překážek, výsledek bude takový, že mnoho hran diagramu bude kolidovat se samotnými překážkami. Pokud by se robot po takové hraně pohyboval, výsledkem by byla kolize robotu samotného s překážkou. Celá situace je znázorněna na obrázku (5.6).



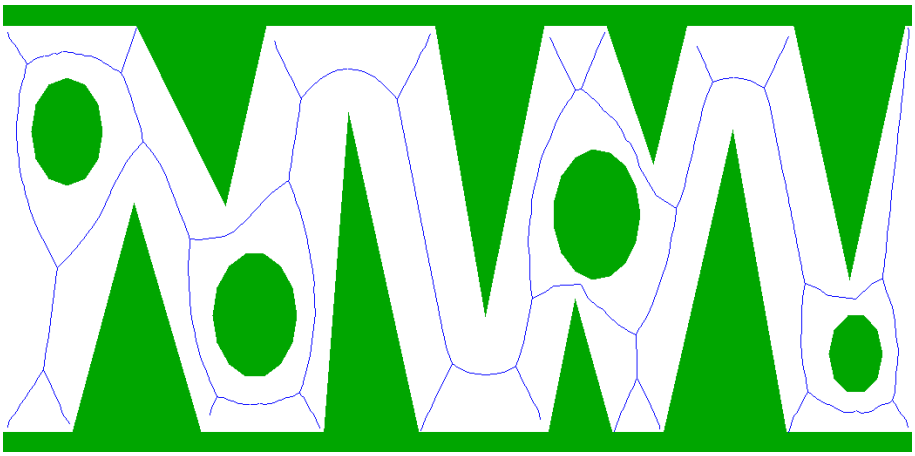
Obrázek 5.6: Voroného diagram s nadbytečnými hranami

U této situace je také dobré ukázat výhodu lepší struktury popisu jednotlivých překážek. S vyšším parametrem K bude výpočet Voroného diagramu kvalitnější. Důkaz je na obrázku (5.7). Oproti obrázku (5.6) je vidět, že s vyšší hodnotou K je počet vrcholů diagramu vyšší a tím pádem je získán lepší popis samotného pracovního prostoru.



Obrázek 5.7: Voroného diagram s lepší strukturou překážek

Nyní je zapotřebí se zaměřit na hrany diagramu, které kolidují s překážkou nebo se v překážce nacházejí. Takové hrany musí být odstraněny. Výsledkem je konečný Voroného diagram (obr. 5.8), ve kterém lze nalézt optimální cestu.



Obrázek 5.8: Voroného diagram pro pracovní prostor s polygonovými překážkami

5.3 Hledání cesty pomocí algoritmu A*

Pomocí Voroného diagramu byl získán graf možných cest, na kterých se robot může pohybovat, ale tato práce se zabývá úlohou nalezení optimální cesty mezi dvěma body a tuto úlohu už samotný algoritmus voroného diagramu neřeší. Je tedy nutné použít jinou metodu, která optimální cestu mezi těmito body nalezne. Existuje mnoho algoritmů, jak lze takovou cestu najít. Jedním z nejznámějších a nejvyužívanějších metod pro hledání cesty v kladně ohodnoceném grafu je algoritmus nazývaný A*, více na stránkách [9].

Tento algoritmus používá hladový princip pro nalezení optimální cesty z daného počátečního uzlu do požadovaného koncového uzlu. Hladový algoritmus vybírá v každém svém kroku lokální minimum a tento postup provádí, dokud výpočet nedosáhne bodu cílového. Optimální cestou se rozumí nejkratší, nejrychlejší nebo nejlevnější cesta v závislosti na reprezentaci hodnot vah hran v grafu. Z pohledu hledání optimální cesty v této práci jde o cestu nejkratší.

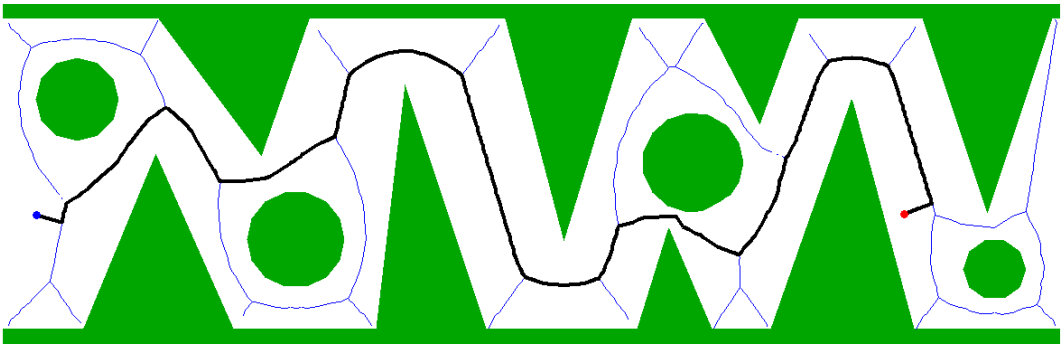
Pro výpočet je nutné definovat funkci značenou $f(n)$, která ohodnocuje ceny jednotlivých uzlů a podle jejich výše je vybírán vždy takový uzel s nejnižší cenou. Výpočet této funkce je je proveden podle vztahu

$$f(n) = g(n) + h(n) \tag{5.1}$$

kde funkce $g(n)$ představuje cenu z počátečního uzlu do aktuálního uzlu n a funkce $h(n)$ představuje heuristickou funkci. Tato funkce odhaduje správnost postupu při vyhledávání optimální cesty za pomoci vzdálenosti z aktuálního uzlu n do uzlu cílového. Hodnota této funkce byla stanovena jako přímá vzdálenost mezi aktuálním a cílovým bodem.

Samotný algoritmus 5.2 je implementován následovně. Jsou vytvořeny fronty otevřených uzlů (ještě nebyly navštíveny) a uzavřených uzlů (již byla určena hodnota $f(n)$). Z otevřené fronty se odebere takový uzel, jehož cena $f(n)$ je nejnižší, vloží se do uzavřené fronty a je spočítána hodnota funkce $f(n)$ pro všechny jeho sousední uzly. Tyto sousední uzly jsou pak vloženy do fronty otevřených uzlů. Pokud se uzel v této frontě již vyskytuje a nová hodnota $f(n)$ je nižší, tak se odhadovaná cena pouze přepíše. Takto algoritmus pokračuje, dokud nemá konečný uzel menší hodnotu $f(n)$, než libovolný jiný uzel z fronty, nebo dokud není tato fronta prázdná. Hodnota $f(n)$ koncového uzlu je poté délkou nejkratší cesty grafem, protože hodnota funkce $h(n)$ je nulová. Aby bylo možné dále rekonstruovat takto vypočtenou cestu, je nutné

si pro každý uzel pamatovat svého předchůdce. Poté se nechá od cílového bodu zpětně získat seznam všech uzlů, které tvoří nejkratší cestu mezi body. Na obrázku (5.9) je vyobrazen Voroného diagram s nalezenou cestou mezi dvěma body pomocí takto popsaného algoritmu.



Obrázek 5.9: Nalezená cesta ve Voroného diagramu pomocí algoritmu A*

5.2: A*

```
1  closedset = prazdna mnozina
2  openset = mnozina obsahujici pouze pocatecni uzel
3  g_skore[start] = 0
4  h_skore[start] = heuristicky_odhad_vzdalenosti(start,
       cil)
5  f_skore[start] = h_skore[start]
6
7  while openset neni prazdna
8      x = otevreny uzel s nejmensi hodnotou f_skore[]
9
10     if x = cil
11         return rekonstruuj_cestu(prisel_z[cil])
12     end
13
14     vyjmi x z openset
15     pridej x do closedset
16
17     for y : sousedni_uzly(x)
18         if y in closedset
19             continue
20         end
21         stavajici_g_skore = g_skore[x] + d(x, y)
22
```

```
23          if y not in openset
24              add y to openset
25              stavajici_je_lepsi = true
26          elseif stavajici_g_skore < g_skore[y]
27              stavajici_je_lepsi = true
28          else
29              stavajici_je_lepsi = false
30          end
31          if stavajici_je_lepsi = true
32              prisel_z[y] = x
33              g_skore[y] = stavajici_g_skore
34              h_skore[y] = heuristicky_odhad_vzdalenosti
                    (y, cil)
35              f_skore[y] = g_skore[y] + h_skore[y]
36          end
37      end
38  end
39
40  function rekonstruuj_cestu(aktualni_uzel)
41      if prisel_z[aktualni_uzel] is set
42          p = rekonstruuj_cestu(prisel_z[aktualni_uzel])
43          return (p + aktualni_uzel)
44      else
45          return aktualni_uzel
46      end
47  end
```

6 Úprava nalezených cest pro mobilní roboty

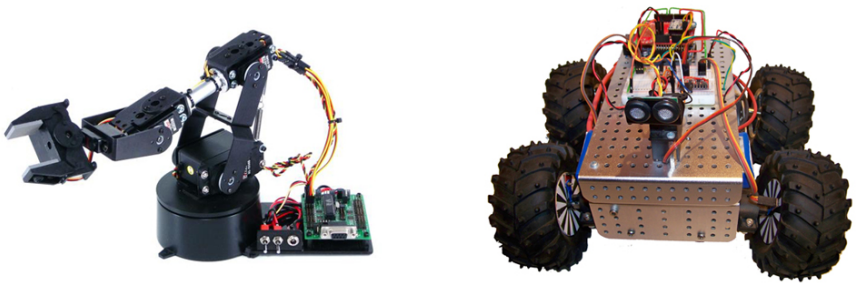
Pomocí metod uvedených v kapitolách 4 a 5 byly nalezeny cesty, které ale nejsou z hlediska vzdálenosti nejkratší možné a jejich průběhy nejsou hladké (v některých místech neexistují druhé derivace). Takové trasy by reálný robot nebyl schopen projet a proto je dobré tyto navrhované trasy matematicky upravit tak, aby byly hladké, z hlediska vzdálenosti optimální a realizovatelné pomocí robotu s konkrétním podvozkem. Tímto problémem a jeho řešením se bude zabývat tato kapitola. Nejprve ale bude řečeno, co je mobilní robot, jaké jsou varianty jeho podvozku a poté budou uvedeny metody, pomocí kterých lze trasu upravit. Porovnání metod upravujících trajektorie bude uvedeno v kapitole 8. Na konci kapitoly pak bude vysvětleno, jak byl vyřešen problém s řízením mobilního robotu s konkrétními rozměry.

6.1 Mobilní robot a jeho typy podvozků

V předchozích kapitolách bylo často používáno slovo robot. Nyní bude vysvětleno, co je to robot, s jakými typy robotů se můžeme v současnosti setkat a jaký typ podvozku robotu byl zvolen pro ověření plánování trajektorie.

Co je tedy již mnohokrát zmiňovaný robot? Robotem lze nazvat inteligentní stroj vykonávající určené úkoly. Tyto úkoly jsou vykonávány předepsaným způsobem a při různých mírách potřeby interakce s okolním světem a se zadavatelem, který může zasáhnout do vykonávání úkolu. Pokud se jedná o autonomní verzi robotu, lze říci, že takový robot svou práci vykonává zcela samostatně, bez cizího zásahu, pomocí řídícího počítače, který je mozkem takového robotu.

Podle schopnosti pohybu v prostoru lze roboty dělit na stacionární a mobilní. Stacionární robot je takový robot, který se nepohybuje z místa na místo a svou činnost vykonává většinou pomocí kloubového ramene. V praxi se tento typ robotů vyskytuje často v průmyslu u manipulátorů, které jsou určeny k jedné konkrétní činnosti (svařování, frézování, přesun výrobků z jedné linky na druhou apod.). Mobilní roboty jsou takové roboty, které se mohou pohybovat ve svém pracovním prostoru a nejsou fyzicky ukotveny na místě. Jako příklad lze uvést kolové či létající roboty, které se pohybují volně ve svém pracovním prostoru. Na tento typ robotů je zaměřena i tato práce.



Obrázek 6.1: Ukázka robotů - vpravo stacionární, vlevo mobilní

Mobilní roboty lze dále dělit podle dvou kritérií. Prvním kritériem dělení může být typ prostředí, ve kterém se robot pohybuje. Podle typu prostředí pak mohou být roboty pracující na zemi, ve vodě, ve vzduchu nebo ve vesmíru. Dalším kriteriem pak může být způsob kontaktu podvozku s povrchem. Do této kategorie lze zařadit kolové podvozky, pásové podvozky nebo kráčející podvozky.

Senzorický systém robotů

Aby mohl robot reagovat na aktuálně vzniklé situace a plnit požadavky, je nutné doplnit fyzickou strukturu takzvaným senzorickým systémem, který bude poskytovat zpětnou vazbu řídícímu systému a ten tak bude moci správně reagovat. Senzorickým systémem se rozumí soustava vhodných čidel, které snímají a následně posílají informace řídícímu systému z okolního prostředí. Pro lepší představu lze uvést například detektor pohybu. Když senzor zaznamená pohyb, je vyslán pokyn k rozsvícení světla nebo spuštění alarmu.

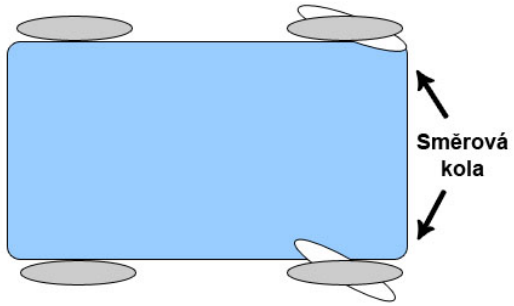
Druhým systémem, který tvoří strukturu robotů, je tzv. kognitivní nebo také řídící systém. Tento systém zpracovává a vyhodnocuje informace získané ze senzorů a podle těchto dat je schopen určit, jak se má v daných situacích robot zachovat - vydává řídicí pokyny akčním prvkům (motor, servo).

Rozdělení podvozků mobilních robotů

Ne každý mobilní robot je schopen projet libovolnou naplánovanou trajektorii. Schopnost projetí nalezené trajektorie je závislá na typu podvozku robotu a proto nyní budou představeny typy jednotlivých podvozků, které jsou u mobilních robotů využívány. Podrobněji se typy podvozků mobilních robotů a jejich modelováním zabývá práce [10].

Ackermanův podvozek

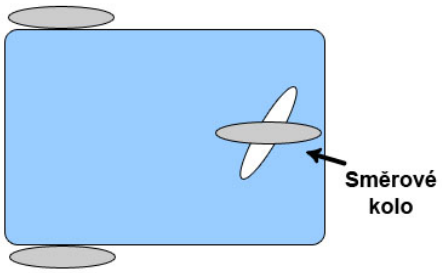
Zřejmě nejrozšířenějším a nejznámějším podvozkem je Ackermanův podvozek. Setkat se s ním můžeme u drtivé většiny automobilových podvozků. Skládá se ze dvou zadních a dvou předních směrových kol, kde každé kolo je natočeno pod jiným úhlem. Hnacími koly mohou být jak přední tak zadní a někdy i všechna čtyři kola podvozku. Omezením tohoto podvozku je pohyb do boku. Díky tomu jsou kladena omezení i na tvar trajektorie. Příklad podvozku je na obrázku (6.2).



Obrázek 6.2: Ukázka Ackermanova podvozku

Trojkolový podvozek s řízeným předním kolem

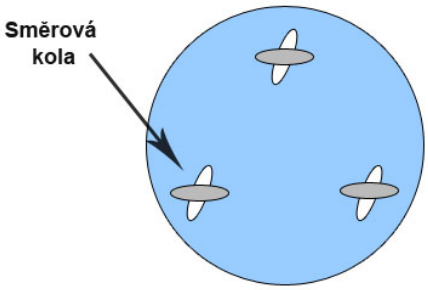
Konstrukce takového podvozku se nejčastěji skládá ze dvou zadních hnacích kol a předního směrového kola. V dnešní době se ale hojně vyskytují podvozky, kde je koncepce kol opačná (jedno zadní hnací kolo a dvě směrová kola vpředu). Stejně jako v prvním případě se z fyzických zákonů tento podvozek nedokáže otočit na místě (pokud ovšem nemá každé hnací kolo svůj pohon, viz diferenciální podvozek). Toto omezení klade i přísnější požadavky na tvar vykonávané trasy. Rozdílem mezi Ackermanovým podvozkem je pouze počet kol a směrová kola nejsou zároveň hnací a naopak. Trojkolový podvozek s jedním směrovým a dvěma hnacími koly je na obrázku (6.3).



Obrázek 6.3: Ukázka trojkolového podvozku

Synchronní podvozek

Nejčastější uspořádání podvozku tvořené ze tří kol se dvěma stupni volnosti je vidět na obrázku (6.4). Díky možnosti natočení kol se robot s takovým podvozkem může pohybovat libovolným směrem za předpokladu, že jsou všechna kola stejně natočena a točí se stejnou rychlostí. To znamená, že robot takovýmto podvozkem je schopen vykonat libovolnou trasu. Výhodou podvozku oproti předchozím dvěma typům je tedy dobrá manévrovatelnost.

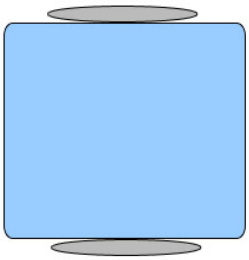


Obrázek 6.4: Ukázka synchronního podvozku

Diferenciální podvozek

Tento podvozek se vyznačuje tím, že, stejně jako každé z kol, má pouze jeden stupeň volnosti a žádné kolo není směrově orientované. Zatímco u jiných typů podvozku je smyk nežádoucím faktorem, zde je využíván při zatáčení. Náprava podvozku může být tvořena pomocí kol nebo pásů. Obě varianty bývají hojně využívány v reálných konstrukcích u manipulačních, zemědělských nebo vojenských prostředků. Pokud je rychlost na obou těchto nápravách stejná, robot se pohybuje přímočaře a pokud jsou rychlosti odlišné, robot zatáčí na stranu, kde je rychlost nápravy menší. Specifickou vlastností těchto

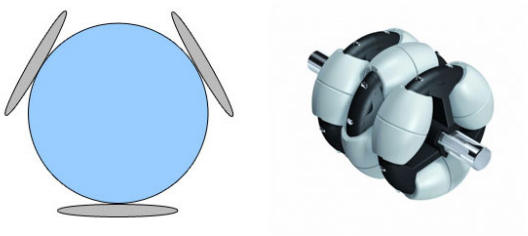
podvozků je, že pokud je rychlost obou náprav stejná, ale opačně orientovaná, robot se může otáčet na místě. Tato vlastnost se dá využít při manipulaci robotu v úzkém pracovním prostoru, kde by se roboti s jinými podvozky pohybovat nedokázaly. Jediným omezením tohoto podvozku je pohyb do boku. Díky dvěma stupňům volnosti se ale robot může na místě natočit do libovolného směru, kterým poté pojede, což je výhodné v ostrých změnách směru trajektorie. Další výhodou podvozku je velmi dobrá průchodnost a manévrovatelnost v obtížném terénu. Z těchto důvodů jsou tyto typy podvozků k vidění i v horských oblastech ve využití pro pohyb na sněhovém podkladu a nebo ve vojenských prostředcích jako podvozek tanku. Pro tento typ podvozku se bude zabývat ověření řízení mobilního robotu v další části práce. Modelováním a řízením diferenciálního podvozku se zabývá práce [16].



Obrázek 6.5: Ukázka diferenciálního podvozku

Všesměrový podvozek

Stejně jako diferenciální se i tento podvozek dokáže otáčet kolem své osy na místě. Ovšem zatímco diferenciální podvozek se dokáže pohybovat pouze v jedné ose a to v ose, která je kolmá ke kolům (pohybu do strany brání uspořádání kol), tak všesměrový podvozek se může díky své koncepci pohybovat ve dvou osách. To znamená, že robot s takovým podvozkem není v pohybu omezován a může se pohybovat libovolným směrem. Konstrukce takového podvozku spočívá ve spojení několika pasivních válečků s hlavním oběžným kolem, jejichž stupeň volnosti je s hlavním oběžným kolem kolmý. Tento typ podvozku nemá žádné omezení při svém pohybu a proto je schopen projet libovolně tvarovanou trajektorii. Uspořádání takového podvozku a ukázka všesměrového kola je vidět na obrázku (6.6).



Obrázek 6.6: Ukázka všesměrového podvozku a všesměrového kola

6.2 Metody pro úpravu nalezených trajektorií

Jak bylo výše řečeno, většina podvozků má nějaké pohybové omezení, což je příčina toho, že robot s určitým podvozkem není schopen vykonat libovolnou trajektorii. Z ekonomických důvodů však není běžné podle nalezené trasy volit typ podvozku, který bude schopen tuto trasu vykonat. Nalezené trasy se tak dále upravují pomocí matematických metod které zaručí, že robot s dostupným podvozkem je schopen upravenou trasu vykonat. Nyní budou podrobně představeny tři základní metody používané pro úpravu nalezené trajektorie nejen pro diferenciálně řízený podvozek. Tuto část bude společně s textem doprovázet nalezená trajektorie pomocí algoritmu RRT, která bude postupně upravována podle požadavků, které byly zmíněny na začátku této kapitoly. Nalezená trajektorie je uvedena na obrázku (6.7). Úpravou nalezených cest se zabývá článek [3] a [4].

Obrázek 6.7: Nalezená cesta pomocí RRT

Pohledem na tento obrázek je zřejmé, že díky ostrým změnám směru trajektorie by tuto trasu mohl vykonat pouze podvozek všesměrový, synchronní a případně i diferenciální. V případě diferenciálního podvozku by projetí takové trasy nebylo časově optimální, protože v každém místě ostrého přechodu by musel robot nejprve zastavit, natočit se do nového směru a poté se znovu rozjet. Ackermanův nebo trojkolový podvozek by projetí této trajektorie díky ostrým přechodům nezvládly.

6.2.1 Vyhlazování cesty

První možnou metodou, která se používá v oblasti plánování je metoda vyhlazování cesty. Tato metoda zaručí, že nalezená cesta bude po částech hladká. To znamená, že bude minimalizován počet jednotlivých lineárních úseků trajektorie. Na již zmíněném obrázku (6.7) je vidět, že nalezená cesta je velmi kostrbatá a pokud by jí měl robot projet, mohl by mít z fyzikálních omezení svého podvozku problémy s průjezdem v místech ostrých přechodů v jednotlivých částech trasy a důsledkem by bylo nedodržení stanoveného průběhu trajektorie při průjezdu robotem. V některých případech by hrozila i kolize s překážkou. Metodou vyhlazování cesty se zabývá článek [3] ve druhé kapitole nazvané Path Planning v sekci Path Prunning.

Princip metody vyhlazování spočívá v hledání nejvzdálenějšího bodu cesty od bodu cílového, přičemž spojnice mezi těmito dvěma body nesmí procházet žádnou překážkou. V prvním kroku je tedy vybrán počáteční a cílový bod trasy a je zjištěno, jestli spojnice mezi body nekoliduje s žádnou překážkou. Pokud nekoliduje, body se propojí a výpočet končí. V opačném případě, kdy se spojnice kříží s libovolnou překážkou je vybrán následovník počátečního bodu a opět je testována spojnice nově zvoleného bodu s bodem cílovým. Tato smyčka probíhá do doby, kdy je nalezen první bod, který lze s bodem cílovým spojnicí propojit. Po nalezení takového bodu vzniká první maximální možný lineární úsek trajektorie, který nekoliduje s překážkami. V dalším kroku je cílový bod nahrazen bodem nalezeným v předchozím kroku a je pro něj opět hledán od začátku trajektorie takový bod, jehož spojnice s prvním možným bodem nekoliduje. Výpočet končí v případě, že je propojen počáteční bod trasy s některým ze svých následovníků. Výsledek aplikace algoritmu (6.1) je na obrázku (6.8).

6.1: Algoritmus vyhlazování

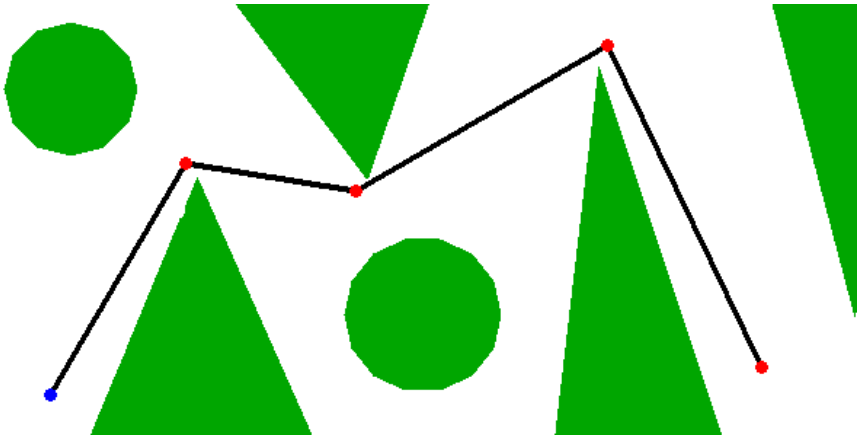
```
puvodni_cesta = [x1,x2,...,xN]
j = N
while j se nerovna 1
    for i = 1:(j-1)
        if spojnice xi a xj je bez kolize
            nova_cesta = [nova_cesta; xi]
            j = i
            break
        end
    end
end
```

Obrázek 6.8: Po částech vyhlazená cesta

Po aplikaci této metody se cesta výrazně zjednodušila a zkrátila, stále se ale nedá mluvit o ideální a především hladké cestě, kterou by byl každý robot schopen projet, protože v některých místech se opět nachází ostré změny směru trajektorie. Tuto trajektorii by opět zvládly projet roboty se všesměrovým, synchronním a diferenciálním podvozkem. V případě diferenciálního podvozku, by projetí trasy bylo mnohem rychlejší než v původním řešení, protože se zde nachází mnohem méně ostrých přechodů. Ackermanův nebo trojkolový podvozek by ale takovouto trajektorii stále neprojely. Řešení zajistí další dvě dále uvedené metody, které využijí vlastností vyhlazené cesty.

6.2.2 Kubický interpolační spline

V případě první metody lze říct, že se jednalo o lineární interpolaci, protože jednotlivé úseky cesty byly nahrazeny lineárními spojnicemi. Použitím kubické interpolace vycházející z článku [11] bude zaručeno, že nově vzniklá trasa bude procházet všemi body, které tvoří vyhlazenou trajektorii a nově vzniklá trajektorie bude po celé své délce hladká, bez ostrých přechodů. Problém lze popsat pomocí obrázku (6.9).



Obrázek 6.9: Trasa, která bude interpolována

Úkolem je nalézt takovou křivku (cestu), která bude procházet červeně vyznačenými body, v celé své délce bude hladká a její první i druhá derivace bude spojitá. Řešením těchto požadavků je interpolace pomocí kubického splinu, který se vyznačuje tím, že na každém intervalu mezi body $P_0, ..., P_n$, které vymezují nalezenou trajektorii, se souřadnicemi x a y je křivka popsána polynomem třetího stupně ve tvaru (6.1)

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{6.1}$$

kde $i = 1, ..., n$ a parametry a_i, b_i, c_i a d_i jsou hledány. Dále musí kubický spline vyhovovat podmínkám spojitosti, které jsou následující (rovnice 6.2)

$$\begin{aligned} S_i(x_{i+1}) &= S_{i+1}(x_{i+1}) \\ S_i'(x_{i+1}) &= S_{i+1}'(x_{i+1}) \\ S_i''(x_{i+1}) &= S_{i+1}''(x_{i+1}) \end{aligned} \tag{6.2}$$

Dále je nezbytné pro určení kubického splinu určení okrajových podmínek. Existuje více možností, jak zvolit okrajové podmínky, v tomto případě budou voleny jako $S''(x_0) = 0$ a $S''(x_n) = 0$. Nyní lze přistoupit k nalezení

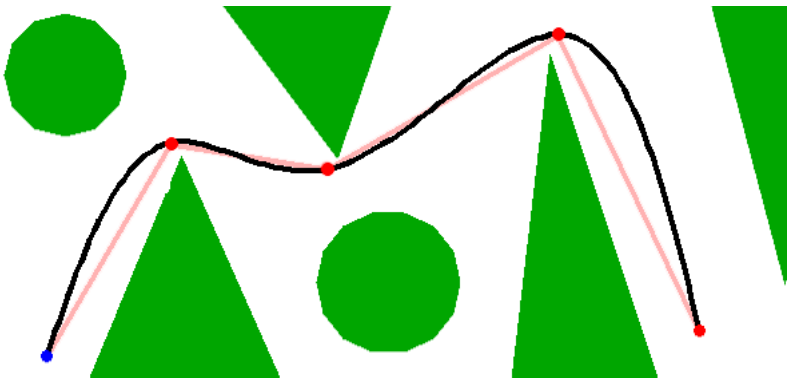
kubického splinu. Nejprve budou hledány koeficienty c_i polynomu $S_i(x)$. Ty jsou nalezeny vyřešením soustavy rovnic ve tvaru (6.3)

$$\begin{aligned}
c_0 &= \frac{y_0''}{2} \\
h_0c_0 + 2(h_0 + h_1)c_1 + h_1c_2 &= 3(\frac{\Delta y_1}{h_1} - \frac{\Delta y_0}{h_0}) \\
h_1c_1 + 2(h_1 + h_2)c_2 + h_2c_3 &= 3(\frac{\Delta y_2}{h_2} - \frac{\Delta y_1}{h_1}) \\
&\cdot \\
&\cdot \\
&\cdot \\
h_{n-2}c_{n-2} + 2(h_{n-2} + h_{n-1})c_{n-1} + h_{n-1}c_n &= 3(\frac{\Delta y_{n-1}}{h_{n-1}} - \frac{\Delta y_{n-2}}{h_{n-2}}) \\
c_n &= \frac{y_n''}{2}
\end{aligned} \tag{6.3}$$

kde $h_i = x_{i+1} - x_i$ a $\Delta y_i = y_{i+1} - y_i$. Nyní zbývá díky znalosti koeficientu c_i dopočítat ostatní koeficienty spline polynomu podle vztahů (6.4)

$$\begin{aligned}
a_i &= f_i \\
b_i &= \frac{f(x_{i+1}) - f(x_i)}{h_i} - \frac{c_{i+1} + 2c_i}{3}h_i \\
d_i &= \frac{c_{i+1} - c_i}{3h_i}
\end{aligned} \tag{6.4}$$

kde $i = 0, ..., (n-1)$. Použitím těchto výpočtů jsou obdrženy polynomy $S_i(x)$, pomocí kterých lze dopočíst interpolační křivku. Výsledná trajektorie je zachycena na obrázku (6.10).



Obrázek 6.10: Použití kubického interpolačního splinu

Trasu získanou pomocí kubického interpolačního splinu by byly schopny bez problému aplikovat všechny uvedené podvozky. V případě Ackermanova a trojkolového podvozku by záleželo, jaký minimální rádius jsou schopny vykonat. Podle tohoto parametru by byla kladena další omezení při výpočtu trajektorie pomocí této metody.

Kolize křivky s překážkami

Výše definovaný přístup na výpočet kubické interpolační křivky neobsahuje řešení pro případ, že nově interpolovaný úsek cesty koliduje s překážkou. To je ovšem velmi důležité pro úlohy plánování trajektorií. Pokud by vznikla kolize s překážkou, robot by naplánovanou trasu nemohl absolvovat. Je tedy nutné zajistit, aby takto interpolovaná trasa s překážkami nekolidovala. Pravděpodobnost kolize se v tomto případě zvyšuje za podmínky, že odchylka mezi interpolovanou a po částech vyhlazenou cestou se zvětšuje. To byla zásadní myšlenka pro hledání řešení tohoto problému. Pokud na některém intervalu trasa koliduje, byl tento interval rozdělen na dva stejně dlouhé intervaly, čímž vznikl nový řídící bod mezi původní dvojicí bodů a pro nově vzniklé dva intervaly byla opět spočítána nová křivka. Díky vlastnosti, že křivka prochází všemi řídícími body tak byla zmenšena odchylka od původní trajektorie a bylo opět zjištěno, jestli mezi nově vzniklými intervaly křivka nekoliduje s překážkami. Tento postup byl opakován do doby, dokud nebyla celá křivka součástí volného pracovního prostoru C_{free}.

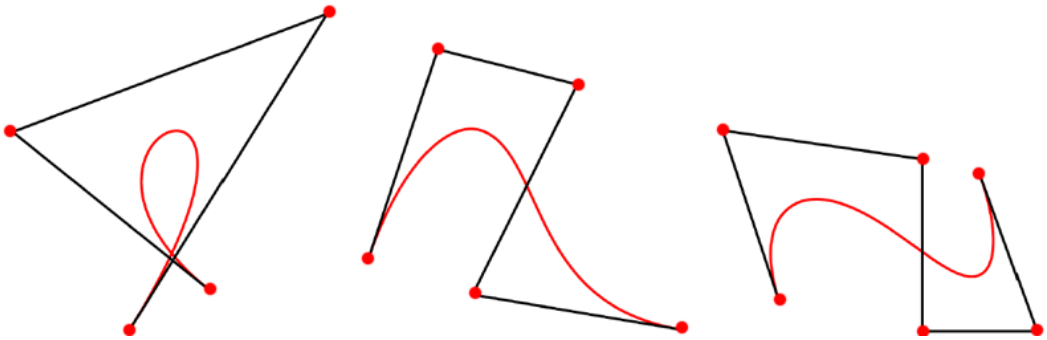
6.2: Algoritmus interpolačního splinu

```
cesta = [x1,x2,...,xN]
kolize = 1
while(kolize ~= 0)
  nova_cesta = vypocet_splinu(cesta)
  [kolize, pozice] = zjisteni_kolize_nove_cesty
  if (kolize == 1)
    i = index_vymezujiciho_bodu_pred_mistem_kolize
    novy_bod = (cesta(i) + cesta(i+1))/2
    cesta = [x1,...,xi,novy_bod,x(i+1),...,xN]
  end
end
```

Díky funkci *cscvn*, která automaticky vypočítá interpolační kubický spline, nebylo nutné implementovat výše popsaný výpočet polynomu $S_i(x)$.

6.2.3 Bézierova křivka

V této části bude popsán přístup aproximace cesty, který se v této disciplíně používá nejčastěji. Řeč bude o Bézierově křivce [4] a [12], která je pojmenovaná po francouzském inženýru Pierru Bézierovi. Tato metoda umožňuje interaktivně vytvářet křivku a modifikovat její tvar. Na rozdíl od předchozí metody je tato křivka aproximační, což znamená, že výsledná křivka nemusí procházet řídícími body, což je zřetelné z obrázku (6.11).



Obrázek 6.11: Ukázka Bezierových křivek

Křivka řídícími body sice neprochází, ale znalost těchto bodů, které tvoří řídící polygon (cestu) a budou zde značeny $P_0, ..., P_n$, je pro výpočet velmi důležitá.

Metoda de Casteljau

Tento algoritmus se používá pro výpočet parametrů Bézierovy křivky [12]. Je založen na opakovaném použití lineární interpolace a zobecňuje speciální případ konstrukce paraboly pro křivky vyšších stupňů. Výpočet křivky bude ukázán na polygonu se 4 řídícími body, viz obrázek (6.12). Pro výpočet je nutné definovat parametr t, který bude nabývat hodnot z intervalu $\langle 0, 1\rangle$. V prvním kroku algoritmu je provedena lineární interpolace podle následujících vztahů (6.5) pro všechny dvojice po sobě jdoucích bodů a jsou získány body nové.

$$\begin{aligned} P_0^1(t) &= (1-t)P_0 + tP_1 \\ P_1^1(t) &= (1-t)P_1 + tP_2 \\ P_2^1(t) &= (1-t)P_2 + tP_3 \end{aligned} \tag{6.5}$$

Stejný postup je aplikován ve druhém kroku. Pro nově získané parametry z kroku předchozího jsou jsou opět vypočítány nové body podle vztahů (6.6).

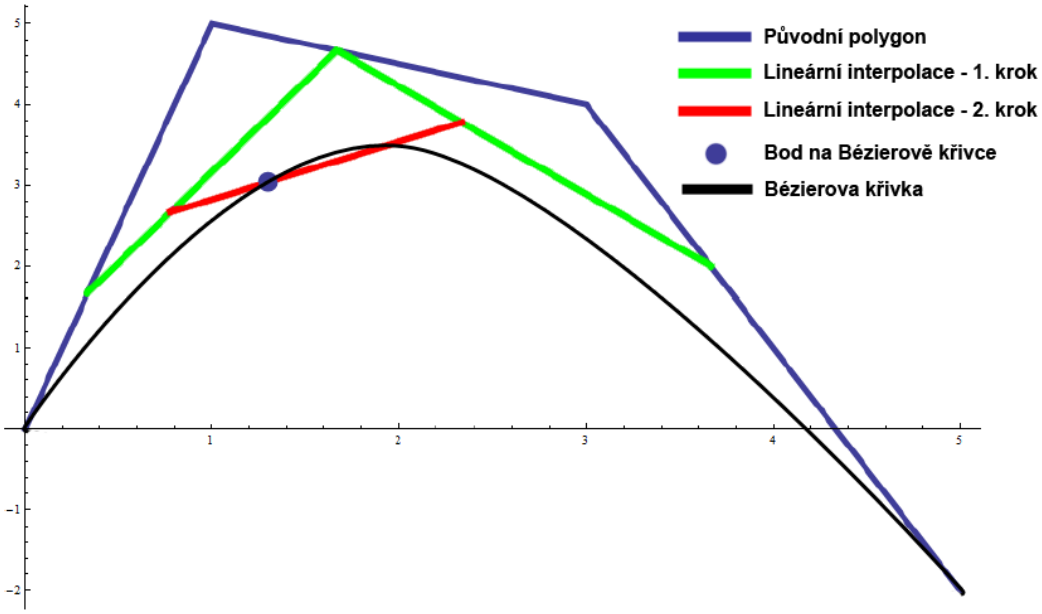
$$P_0^2(t) = (1-t)P_0^1 + tP_1^1$$
$$P_1^2(t) = (1-t)P_1^1 + tP_2^1 \tag{6.6}$$

V posledním kroku se pro dva nové body postup opět opakuje podle vztahu (6.7) a je získán bod na Bézierově křivce.

$$P(t) = P_0^3(t) = (1-t)P_0^2 + tP_1^2 \tag{6.7}$$

Zároveň je nutné zdůraznit, že pro všechny tyto kroky musí být použita stejná hodnota parametru t. Po dokončení posledního kroku je celý algoritmus opakován pro všechna $t \in \langle 0, 1\rangle$. Počet kroků pro každé t je závislý na počtu řídících bodů v polygonu. V případě, že polygon má $n+1$ řídících bodů, počet kroků je n. Pro libovolný polygon vždy platí, že krajní body polygonu jsou vždy totožné s krajními body Bézierovy křivky, protože platí $P(0) = P_0$ a $P(1) = P_3$. Pokud se vztahy (6.5 a 6.6) dosadí do (6.7) je získán obecný předpis metody de Casteljau pro výpočet parametrů křivky (6.8).

$$P(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i \tag{6.8}$$



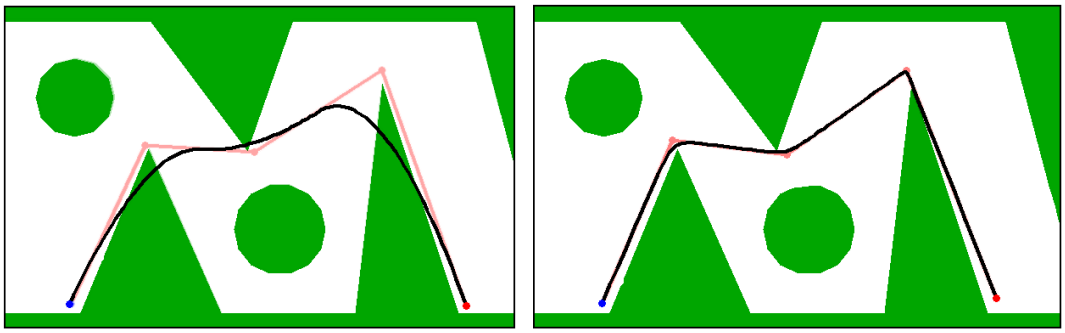
Obrázek 6.12: Konstrukce Bézierovy křivky pro ukázkový polygon

Kolize Bézierovy křivky s překážkami

Po výpočtu Bézierovy křivky je získána aproximace křivky s původními řídícími body. Stejně jako v předchozím případě tento algoritmus nezajišťuje bezkolizní trajektorii. V tomto případě problém kolize nastává z toho důvodu, že aproximace po částech lineární cesty pomocí Bézierovy křivky není vázána průchodem řídících bodů původní cesty. Řešením je definování váhy každého bodu. Pokud nějaká část křivky prochází překážkou, zvolí se vyšší váha bodu, pro jehož úsek aproximace s překážkou koliduje. Díky větší váze je poté aproximovaná cesta blíže k vymezujícímu bodu. Postup zvyšování váhy bodů je aplikován do té doby, kdy konkrétní části trasy s překážkou nekolidují. Výsledný vzorec pro výpočet křivky se po přidání jednotlivých vah bodů změní do tvaru (6.9)

$$P(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \omega_i P_i \tag{6.9}$$

kde ω_i určuje váhy jednotlivých bodů. Celá takto popsaná situace je opět doplněna obrázkem (6.13).



Obrázek 6.13: Vlevo křivka kolidující s překážkami, vpravo křivka bez kolize

I tuto trasu by pak mohly projet všechny podvozky, u některých by byl ale opět kladen požadavek na minimální rádius zatáčení, který by musela trasa splňovat.

6.3: Algoritmus výpočtu Bézierovy křivky

```
cesta = [x1,x2,...,xN]
w = [1,1,...,1] %vahy bodu cesty
while(kolize ~= 0)
  nova_cesta = vypocet_bezierovy_krivky(cesta,w)
  [kolize, pozice] = zjisteni_kolize_nove_cesty
  if (kolize == 1)
    zvyseni vahy bodu, v jehoz okoli vznikla kolize
  end
end
```

6.3 Řízení modelu mobilního robotu

Tato část kapitoly bude zaměřena na samotnou implementaci řízení. Doposud bylo v předchozích kapitolách popsáno, jak hledat a upravovat v pracovním prostoru trasu robotu, který má nulové rozměry. Ve skutečnosti ovšem robot nulové rozměry nemá a proto je nutné dodržovat patřičný odstup od překážek, aby nedocházelo ke kolizím. Díky možné rotaci kolem těžiště se robot může otáčet na místě. To byla důležitá vlastnost, které bylo využito při implementaci řízení, protože nebylo nutné řešit rychlé směrové změny trajektorie a jediné omezení, které bylo nutné aplikovat, bylo zamezení kolize s překážkou při daných rozměrech robotu.

Způsoby, jak této kolizi zamezit, jsou minimálně dva. První možnou variantou je zjišťovat při expanzi stromů RRT vzdálenost nově vygenerovaného bodu q_{new} od překážky. Pokud je tato vzdálenost menší, než je maximální možná vzdálenost určená rozměry robotu, tak tento bod není přidán do struktury stromu a je vygenerován nový bod q_{new}. Obdobně by tento výpočet probíhal i u metody Voroného diagramu. Nevýhodou u takového výpočtu je jeho časová náročnost, protože by se pro každý bod musela počítat vzdálenost od každé překážky a to by celý výpočet výrazně zpomalilo. Lepším řešením je „rozšíření“ samotných překážek o určitý rozměr daný rozměry robotu a následně prohledávat pracovní prostor reprezentovaný rozšířenými překážkami. V tomto případě, kdy se předpokládá, že robot bude mít obdélníkový tvar, bude velikost rozšíření určena polovinou délky uhlopříčky, tedy od těžiště robotu k jednomu z rohů jeho konstrukce. Tím je zajištěno, že robot nebude kolidovat s překážkami a dokonce se bude moci v případě potřeby v libovolné části volného pracovního prostoru otočit na místě, aniž by kolidoval s překážkou. Ukázka pracovního prostoru s rozšířenými rozměry překážek je obrázku

(6.14). Zeleně jsou vyznačeny původní překážky a červeně je vykresleno rozšíření. Toto rozšíření určuje, jak daleko od překážky se musí těžiště robotu nacházet, aby nedošlo ke kolizi konstrukce robotu s překážkou. Trajektorie při řízení robotu tedy nesmí zasahovat do červených oblastí. Tím je zaručen bezkolizní průjezd.

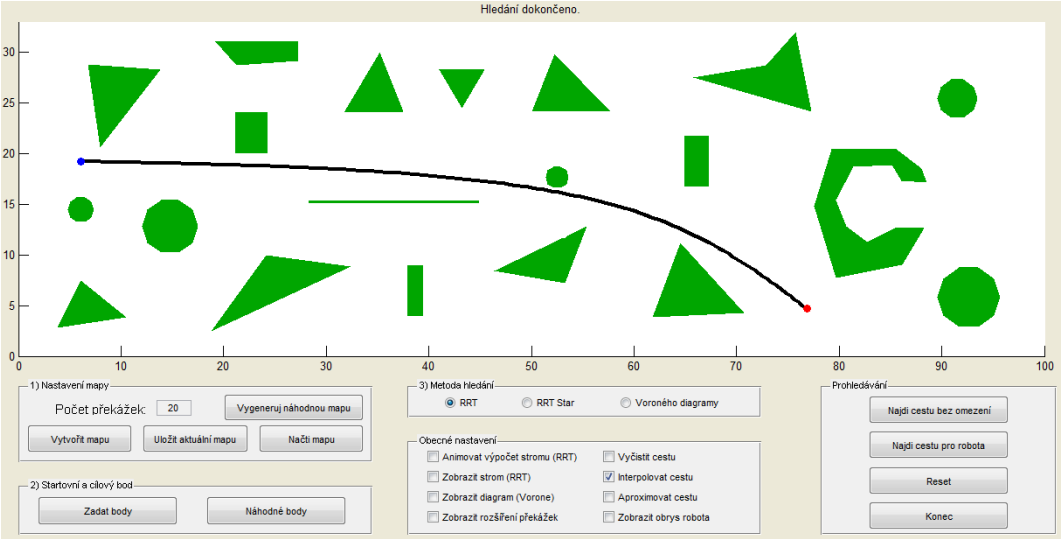
Obrázek 6.14: Ukázka zvětšení rozměrů překážek

Z hlediska časové náročnosti je výpočet pomocí RRT pro takto definovaný pracovní prostor delší než pro prostor původní, protože plocha volného pracovního prostoru C_{free} je menší a tím pádem je více bodů q_{new} zamítnuto, protože se nachází v překážce. U prohledávací metody Voroného diagramu je časová náročnost také vyšší, protože překážky jsou reprezentovány složitějším strukturou. Na takto nově definovaný pracovní prostor je možné nyní aplikovat prohledávací metody. Důkaz o časové náročnosti bude uveden v poslední kapitole 8.

7 Vizualizační prostřední pro metody plánování trajektorie

Jak fungují algoritmy bylo popsáno v předchozích kapitolách. Nyní je třeba se zaměřit na praktickou implementaci algoritmů a především na grafické zpracování výstupů. Aby mohly být algoritmy implementovány, bylo nutné zvolit vhodné programové prostředí. Tímto prostředím bylo zvoleno interaktivní programové prostředí Matlab. Dále bylo vhodné navrhnout grafické prostředí, pomocí kterého budou jednotlivé algoritmy řízeny a pomocí kterého budou zobrazovány výsledky simulací. Na grafické prostředí a zobrazení výsledků jednotlivých algoritmů bude zaměřena tato kapitola.

Pro simulaci implementovaných metod bylo vytvořeno v MATLABu grafické prostředí pro obecné nastavení parametrů a zobrazení výsledků simulace. Spuštění tohoto apletu spočívá v otevření souboru *run.m*. Uživatel si v tomto prostředí může vygenerovat mapu vlastního prostoru, zadat počáteční a cílový bod, zvolit metodu prohledávání nebo třeba upravit průběhy nalezených trajektorií. Celé základní prostředí je zobrazeno na obrázku (7.1).

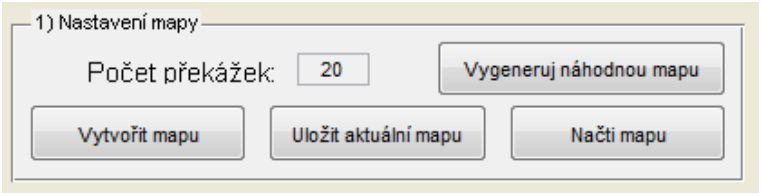


Obrázek 7.1: Grafické prostředí pro simulaci algoritmů

V horní části je zobrazen pracovní prostor s překážkami a ukázkovou cestou mezi dvěma body. Ve spodní části jsou skupiny tlačítek a zaškrtávácích políček, kterými lze měnit obecné nastavení simulací. Této části grafického prostředí se bude text nyní věnovat.

7.1 Volba parametrů simulace

První krokem, který musí uživatel provést je vygenerování dvojrozměrné mapy, ve které bude hledání vhodné trajektorie probíhat. Pro vygenerování mapy slouží skupina prvků nazvaná *Nastavení mapy* viz obrázek (7.2). V případě, že uživatel nemá konkrétní specifické požadavky na parametry pracovního prostoru, může si nechat vygenerovat náhodnou mapu. V tomto případě si nastaví pouze počet překážek, které se mají v pracovním prostoru vykreslit. Stisknutím tlačítka *Vygeneruj náhodnou mapu* je mapa zobrazena v okně pracovního prostoru výše. Druhou možností je návrh vlastního pracovního prostoru, který provede uživatel sám. K tomuto případu slouží tlačítko *Vytvořit mapu* a více o ručním vytvoření pracovního prostoru bude napsáno na konci této kapitoly. Pokud si uživatel vytvoří nebo vygeneruje mapu, která je vhodná i pro případné další použití, může kliknutím na *Uložit aktuální mapu* mapu uložit a později pomocí tlačítka *Načti mapu* opět tuto mapu načíst.



Obrázek 7.2: Definování pracovního prostoru s překážkami

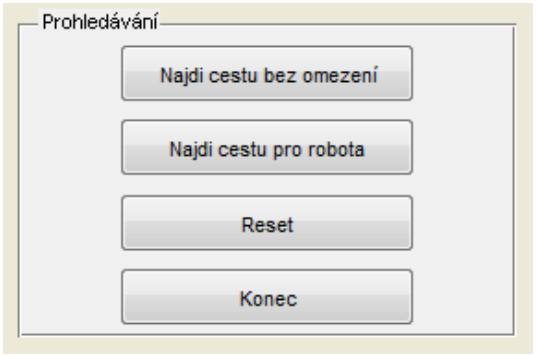
Po vygenerování mapy je třeba zadat počáteční q_{init} a cílovou pozici robotu q_{goal}. Uživatel si může vybrat, jestli se body vygenerují sami nebo jestli je sám zadá do pracovního prostoru. Je nutné, aby byl bod umístěn do volného pracovního prostoru. Pokud je náhodný bod vygenerován, či sám uživatel by zadal pozici bodu do místa, kde se nachází překážka, bod nebude přijat a zadání pozice tohoto bodu bude nutné opakovat. Počáteční pozice je v mapě reprezentována modrým bodem a cílová bodem červeným.

Obrázek 7.3: Tlačítka pro výběr počátečního a cílového bodu

Třetím krokem, který musí uživatel provést před samotným spuštěním hledání trajektorie je volba metody, pomocí které má být pracovní prostor prohledáván. Na výběr je ze tří možností viz obrázek (7.4). Jak tyto metody fungují bylo popsáno v kapitolách (4 a 5)

Obrázek 7.4: Volba metody prohledávání prostoru

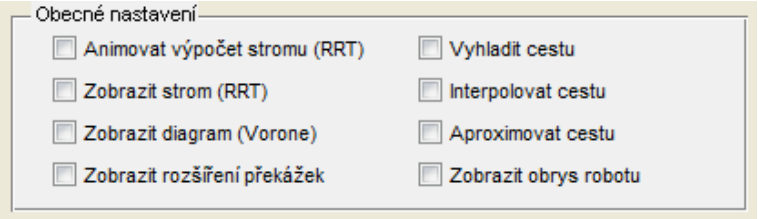
Nakonec uživatel zvolí typ akce, která má být provedena. Na výběr jsou akce pro nalezení neomezené trasy, nalezení trasy pro konkrétní specifikaci robotu, resetování pracovního prostoru a ukončení programu, viz obrázek (7.5). V případě resetování se anulují všechny proměnné v programu mimo mapy s překážkami, která zůstane stejná jako v předchozím prohledávání a počátečního a cílového bodu, které budou také stejné, případně může uživatel parametry změnit za nové pomocí příslušných tlačítek.



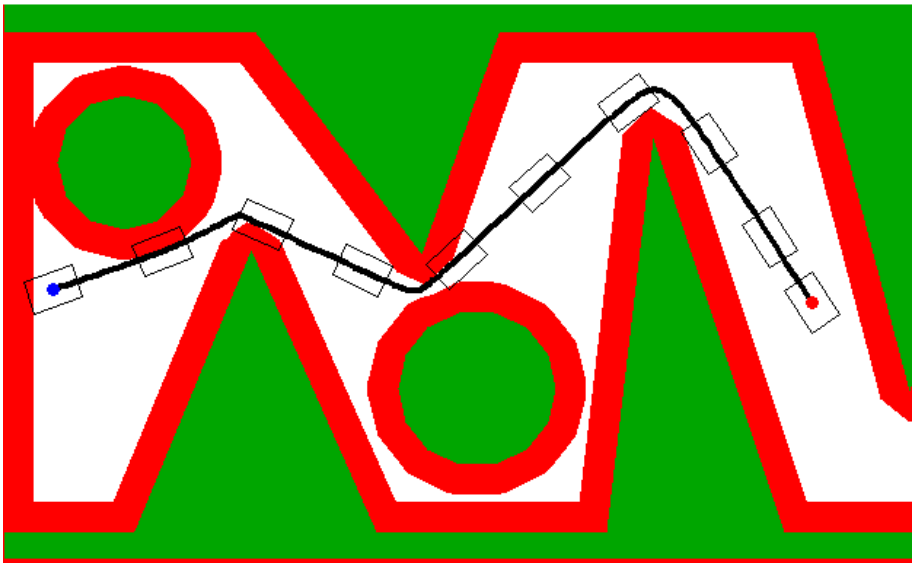
Obrázek 7.5: Možnosti pro volbu akcí

Poslední oknem v grafickém prostředí programu je obecné nastavení, které obsahuje skupinu takzvaných „check boxů“ - zaškrtávacích okének. Prvním z nich je check box pro animaci expanze stromu RRT a RRT*. Pokud jej uživatel zaškrtne, při následném prohledávání pracovního prostoru pomocí metody RRT nebo RRT* je uživateli průběžně vykreslována grafická podoba rozrůstajícího se stromu. Pokud uživatel nechce sledovat průběh expanze stromu pomocí těchto metod, ale po dokončení prohledávání by chtěl zobrazit jeho strukturu, je zde druhé políčko s názvem *Zobrazit strom*. Zvolením této možnosti se vykreslí struktura stromu, která byla zaznamenána v dokončeném prohledávání. Třetí políčko se týká možnosti, kdy byla cesta hledána pomocí Voroného diagramu. Pokud byla cesta hledána právě touto metodou, zaškrtnutím tohoto check boxu je vykreslen společně s výslednou trajektorií i celý Voroného diagram. Posledním políčkem v prvním sloupci je *Zobrazit rozšíření překážek*. Pokud je toto políčko zaškrtnuto, zobrazí se mapa rozšířených překážek (viz obr. 6.14). Uživatel si tak může ověřit, že nalezená trasa pro

robotu nekoliduje s překážkami. Ve druhém sloupci plní první tři check boxy funkce pro úpravu nalezené trajektorie pomocí metod uvedených v kapitole 6.2. Zaškrtnutím jednoho z těchto políček je vždy zobrazena upravená příslušná cesta. Pokud není zaškrtnuto žádné tlačítko, je vykreslena původní nalezená trajektorie. Posledním check boxem je *Zobrazit obrys robotu*. Pokud jej uživatel zaškrtne, na trase se zobrazí obdélníkově naznačený tvar robotu se zadanými rozměry. Na začátku kódu v Matlabu si pak uživatel může zvolit počet obrysů, které se na nalezené cestě mají vykreslit. Stejně tak je možné zadat i rozměry robotu. Příklad je vidět na obrázku (7.7).



Obrázek 7.6: Další možnosti zobrazení výsledků



Obrázek 7.7: Nalezená trajektorie s vykreslením obrysu robotu

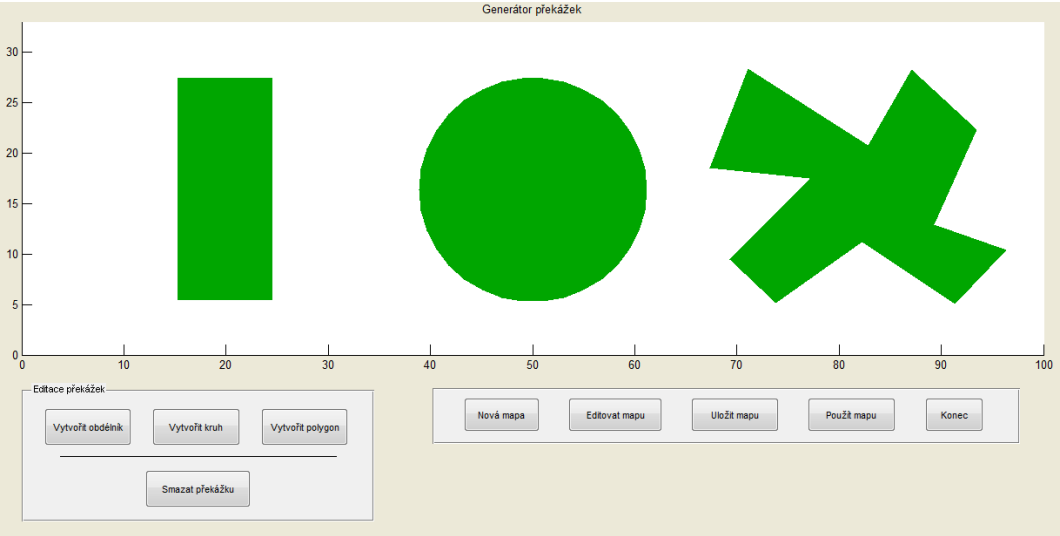
7.2 Vytvoření prostoru s překážkami

Jak bylo řečeno v kapitole 2, kde byla popsána reprezentace prostředí, pro úlohu hledání optimální trajektorie je nutné definovat pracovní prostor, ve kterém se nachází překážky, jimž se robot musí vyhnout. Jedna z možností, jak takovou mapu vytvořit, je jejich náhodné vygenerování, které, až na počet generovaných překážek, není ovlivněno uživatelem. Tento způsob ale není vhodný v případě, že je třeba navrhnout složité prostředí nebo model skutečného prostředí s překážkami. Efektivnější možností je, že si uživatel může nadefinovat vlastní mapu s libovolným počtem a tvarem překážek. V práci je možné využít obou variant, ale v této části bude popsána druhá varianta, kde si uživatel může vytvořit vlastní mapu.

Do generátoru překážek se uživatel dostane ze základního prostředí kliknutím na tlačítko "Vytvořit mapu". Poté se na obrazovce otevře nové okno, ve kterém již uživatel může vytvářet vlastní tvary překážek, viz obr. (7.8). Na výběr jsou tři základní tvary - obdélník, kruh a polygon. Pokud chce tedy uživatel vytvořit obdélník, klikne na příslušné tlačítko *Vytvořit obdélník* a poté v mapě zvolí dva body, které budou tvořit uhlopříčku obdélníkové překážky. Po zadání těchto dvou bodů je překážka v mapě vykreslena. Další možností je vytvoření kruhu, které probíhá obdobně jako v předchozím případě až na výjimku, že dva zadané body netvoří uhlopříčku, ale poloměr kruhu. Prvně určený bod tvoří střed kruhu a druhý leží na kružnici s daným poloměrem. Poslední a nejzajímavější možností vytvoření překážky je vytvoření polygonu libovolného tvaru s n vrcholy. Počet vrcholů polygonu není omezen a ukončení zadávání vrcholů je provedeno zadáním posledního bodu v mapě na pozici prvního bodu aktuálně vytvářeného polygonu. Vzhledem k tomu, že každý bod je v mapě reprezentován křížkem, uživatel ví, kam první vrchol polygonu zanesl. Vytváření překážek není omezeno celkovým počtem a mohou se v tomto případě libovolně překrývat. Pokud byla vytvořena nežádoucí překážka, je možné ji smazat. To lze provést kliknutím na tlačítko *Smazat překážku* a kliknutím do mapy na místo, kde se překážka nachází.

Pokud je navržená mapa z pohledu uživatele podle představ, může s ní v tomto editoru provádět úkony jako je uložení mapy pro pozdější práci s ní nebo použití aktuální mapy pro hledání cesty v ní (tlačítko *Použít mapu*). Pokud existuje již dříve vytvořená mapa a nyní by bylo třeba ji editovat, je možné ji pomocí tlačítka *Editovat mapu* načíst a upravit překážky podle představ. Dále je zde tlačítko pro otevření nového prostoru pro novou mapu a pro ukončení tohoto editoru. Okno generátoru překážek s mapu, která reprezentuje všechny varianty překážek, je zobrazeno na obrázku (7.8). V horní části

je pracovní prostor s navrženými překážkami, v dolní části jsou dvě skupiny tlačítek. Skupina vlevo obsahuje tlačítka pro návrh nových překážek, skupina vpravo obsahuje tlačítka pro práci s navrženým pracovním prostorem.

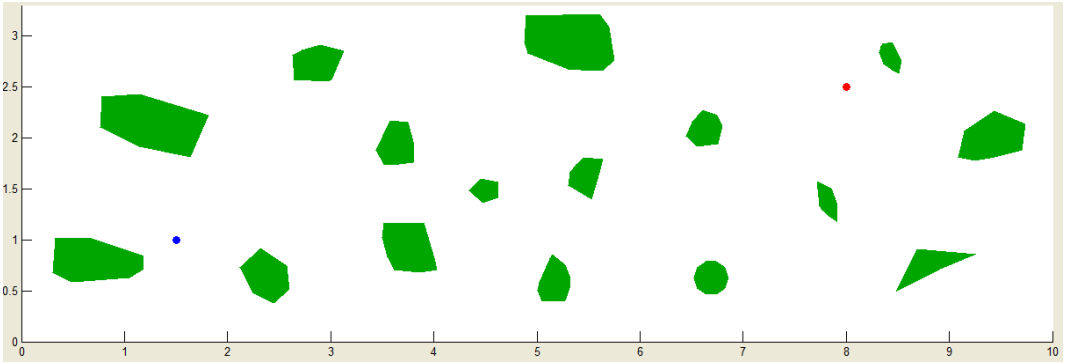


Obrázek 7.8: Generátor překážek v pracovním prostoru

8 Výsledky simulací metod pro plánování trajektorie

V předchozích kapitolách byly popsány metody pro nalezení trasy mezi dvěma body a způsoby, jak tuto nalezenou trasu upravit, aby byla hladká a zároveň optimální z hlediska vzdálenosti pomocí vhodných metod pro úpravy trajektorie. Doposud ale v textu nebyla žádná zmínka o tom, které metody dosahují nejlepších výsledků z hlediska vzdálenosti, časového výpočtu, či počtu iterací výpočtu. Na toto porovnání je zaměřena tato kapitola, ve které budou porovnány výsledky jak pro hledání trajektorie bez omezení, tak s omezením. Toto omezení spočívá v simulaci a naznačení projetí trajektorie pomocí obrysu robotu při znalosti jeho reálných rozměrů, z nichž je počítán jeho odstup od překážek tak, aby nevznikaly kolize. Nejprve budou reprezentovány výsledky každé metody samostatně a dále se získané výsledky jednotlivých metod vzájemně porovnají. Toto porovnání bude provedeno pouze pro trajektorie, které byly získány pomocí jednotlivých metod bez následných úprav popsaných v kapitole 6.2. Toto porovnání bude provedeno na konci kapitoly nezávisle na získaných výsledcích jednotlivých simulací metod pro hledání trajektorie.

Vzhledem k tomu, že se budou dále porovnávat metody hledání trajektorie mezi sebou, pro všechny simulace byl zvolen stejný pracovní prostor s totožnými počátečními a koncovými body pro každou simulaci, viz obr. (8.1).



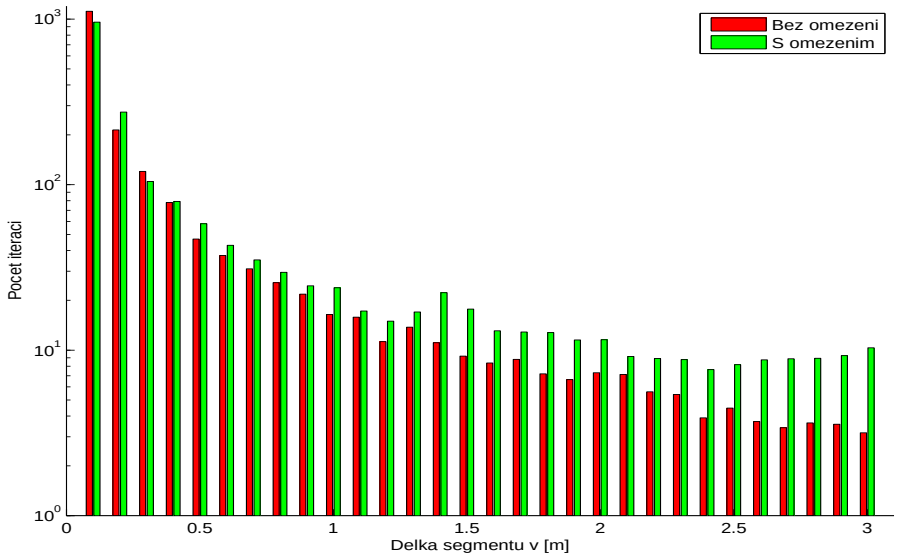
Obrázek 8.1: Pracovní prostor pro porovnání metod

Výsledky simulací metody RRT

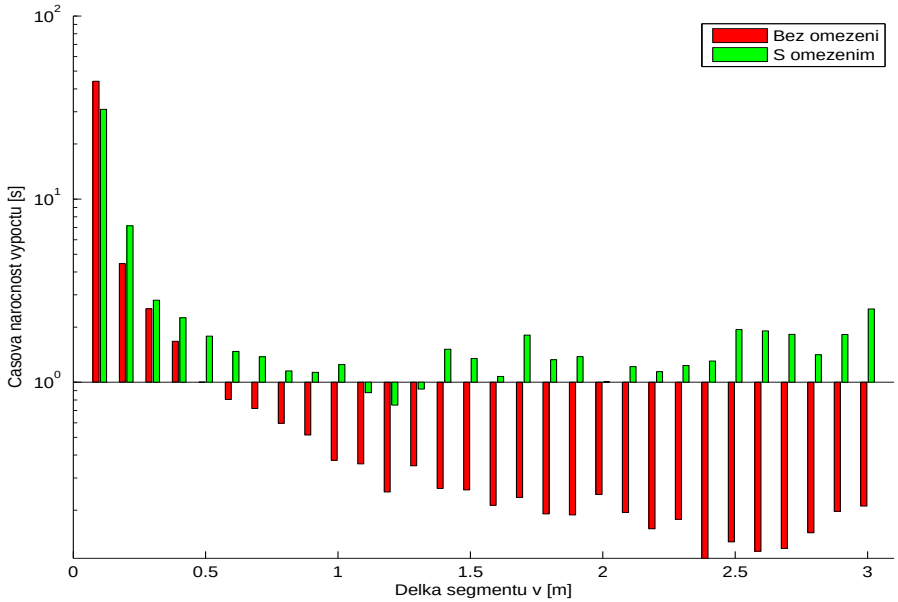
U této metody byly porovnávány 3 parametry. Bylo zkoumáno, jak je závislá délka výpočtu na délce segmentu v (vzdálenost nově vytvořeného bodu q_{new} od bodu existujícího $q_{nearest}$, viz kapitola 4.1). Dále byla měřena závislost počtu iterací smyčky pro výpočet stromu RRT a výsledná délka nalezené trajektorie na délce v. Pro každou hodnotu délky v z množiny $V = \{0.1m, 0.2m, 0.3m, ..., 3m\}$ bylo provedeno třicet simulací hledání cesty a nakonec byl proveden odhad středních hodnot ve formě aritmetického průměru. Tento způsob měření byl použit z důvodů náhodného generování bodů při expanzi stromu. Jelikož je každý výpočet ovlivněn různými realizacemi náhodných veličin, změřené výsledky by dosáhly při každé simulaci jiných hodnot a pokud by byl proveden pouze jeden výpočet pro každou hodnotu z množiny V, výsledky by byly zatížené velkou chybou. Odhad naměřených hodnot pomocí aritmetického průměru tuto chybu částečně eliminuje. Změřené výsledky byly zaznamenány a zaneseny do grafů. Tyto grafy budou nyní vyhodnoceny.

Na prvním grafu (8.2) jsou vyneseny hodnoty počtu iterací v závislosti na délce v s omezením a bez omezení. Z průběhu je vidět, že se zvyšující se délkou v je počet iterací nižší. To však platí do vzdálenosti přibližně $v = 3m$. Při vyšších hodnotách v nebyly pro zvolenou mapu v patřičných vzdálenostech nalezeny žádné úseky nekolidující s překážkami a nebyly tak nalezeny ani příslušné trajektorie. Dále je z grafu zřejmé, že pokud je hledána trajektorie pro konkrétní rozměry robotu (v tomto případě 30x20cm) je počet průměrných iterací s vyšší hodnotou v vyšší než při hledání trajektorie bez omezení. To je způsobeno zmenšením volného pracovního prostoru robotu, který se musí pohybovat v určitých vzdálenostech od překážek. S vyššími rozměry robotu by se počet iterací výpočtu dále zvyšoval.

Na druhém grafu (8.3) je porovnání průměrné doby výpočtu v závislosti na délce segmentu v. I v tomto případě bylo zjištěno, že se zvyšující se délkou v se do jisté hodnoty doba výpočtu snižuje a průměrná časová náročnost výpočtu bez omezení je menší než u výpočtu s omezením. Díky těmto dvěma grafům (8.2 a 8.3) se tak naplnil předpoklad, že časová náročnost výpočtu je závislá na počtu iterací. S délkou segmentu v vyšší než 3m by se časová náročnost výpočtu zvyšovala, protože by bylo složité najít takový segment, který by odpovídal dané délce a nekolidoval by s překážkami.

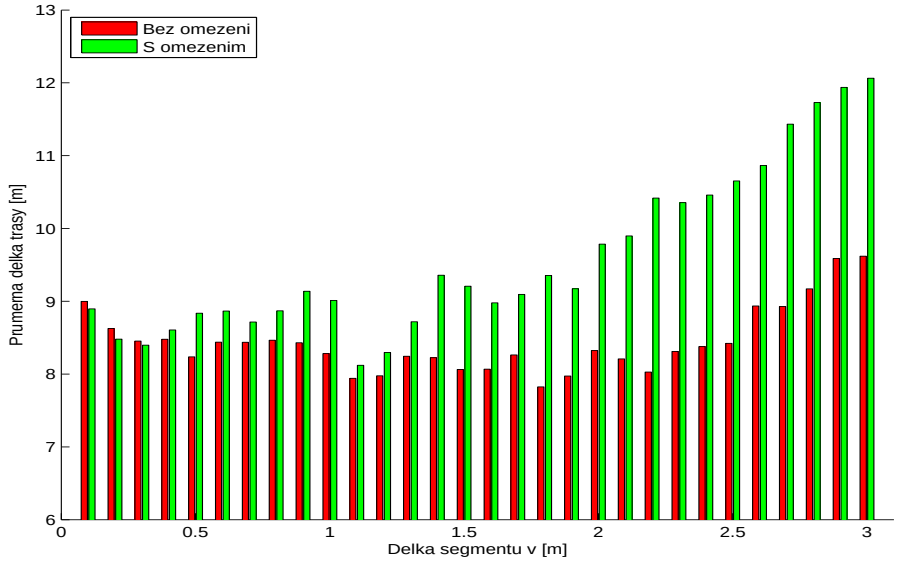


Obrázek 8.2: Závislost počtu iterací algoritmu na délce segmentu v



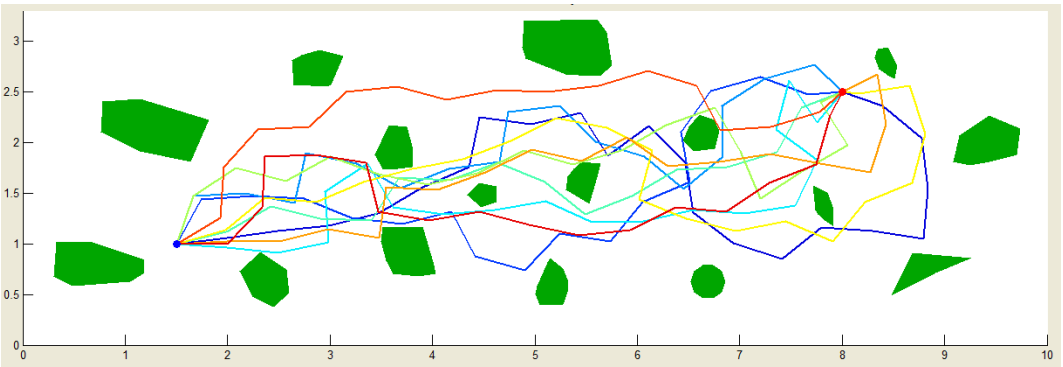
Obrázek 8.3: Závislost průměrného času výpočtu na délce segmentu v

Jako poslední byla zjišťována závislost mezi délkou segmentu v a délkou nalezené trajektorie. Podle grafu (8.4) nejprve průměrná délka nalezených trajektorií do určitých hodnot v mírně klesá, následně se ale se zvyšujícím v délka trajektorie zvyšuje.



Obrázek 8.4: Závislost délky trasy na délce segmentu v

Pro mapu zvolenou k testování tak byly nalezeny nejkratší trajektorie pomocí metody RRT pro hodnoty $v = 1.5m$ v případě simulace bez omezení a $v = 0.7m$ v případě simulace s omezením. Je ale nutné říci, že tyto vypočtené hodnoty nejsou nejlepší z hlediska délky trajektorie. Obecně platí předpoklad, že čím složitější je pracovní prostor robotu, tím menší se volí délka segmentu v a naopak. Dále je z uvedených grafů zřejmé, že čím delší v je voleno, tím je hledání cesty rychlejší, ale nalezená cesta je naopak delší. V opačném případě, kdy by byla voleno kratší v, by výpočet trval déle a nezaručoval by nalezení nejkratší cesty.



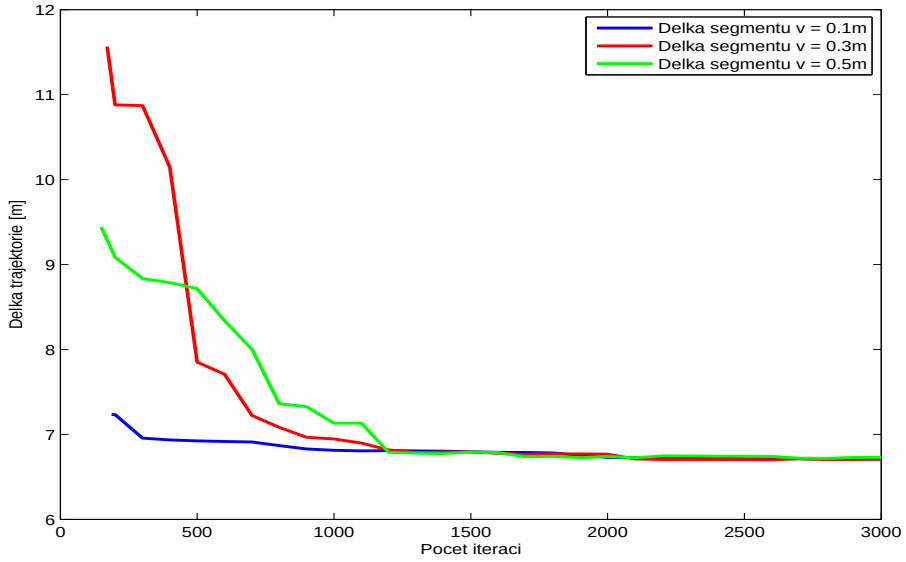
Obrázek 8.5: Ukázka nalezených cest pomocí RRT

To, že nalezená trajektorie pomocí této metody není nejkratší vystihuje i obrázek (8.5), kde je vykresleno deset nalezených cest pro hledání pomocí RRT s délkou segmentu *0.5m* a simulace bez omezení.

Metoda RRT*

Jak bylo řečeno v kapitole 4.2, výpočet pomocí RRT* na rozdíl od RRT nekončí po nalezení první možné trasy, ale pokračuje do maximální povolené iterace a trasa je neustále vylepšována. S tím souvisí i časové nároky na výpočet, které budou pro stejnou velikost segmentu v a stejné simulační podmínky pokaždé téměř stejné. Naopak bude zajímavé se zaměřit na časové rozdíly pro různé délky v. V tomto případě byla množina segmentů V volena pouze v rozmezí $V = \{0.1m, 0.2m...0.5m\}$.

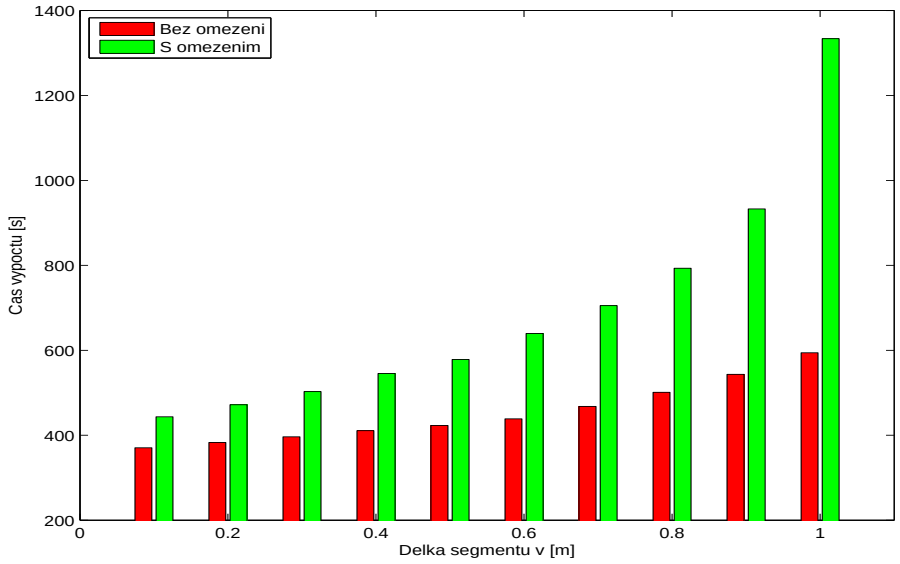
V grafu (8.6) jsou zaznamenány průběhy vývoje délky trajektorie v závislosti na počtu iterací. Tyto průběhy jsou zaznamenány pro tři různé délky segmentu v. Z grafu je vidět, že se zvyšujícím se počtem iterací je trajektorie kratší. Čím více se pak blíží maximální povolené iteraci 3000, tím více se blíží nejkratší možné cestě, v tomto případě $6.7m$.



Obrázek 8.6: Vývoj délky trasy na počtu iterací výpočtu

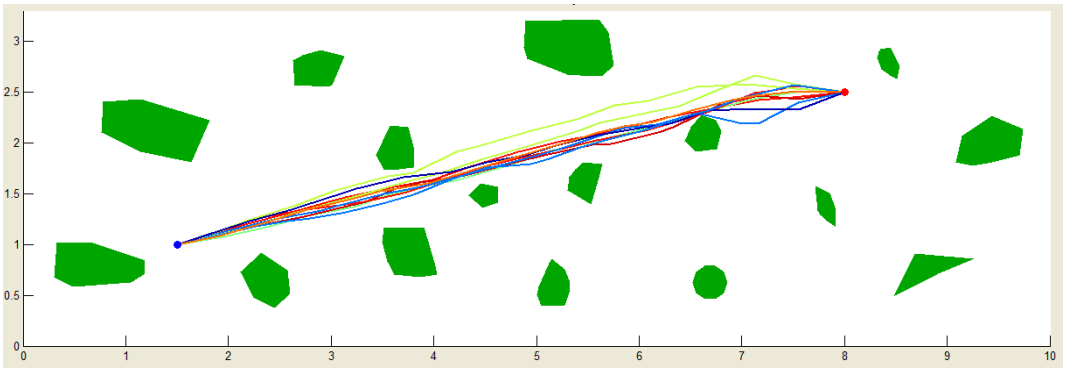
Na dalším grafu (8.7) jsou zaznamenány průběhy časové náročnosti výpočtu v závislosti na délce segmentu v. Opět jsou zde uvedeny průběhy pro hledání trajektorie s omezením a bez omezení. Z průběhů je možné vypo-

zorovat, že s vyšším v se časové nároky na výpočet v obou případech exponenciálně zvyšují. To je způsobeno rychlejším pokrytím celého pracovního prostoru při vytváření stromu a pro nově vygenerované body se musí provádět s přibývajícím časem více přepočítávání (viz obr. 4.4), které se projeví ve výsledné časové náročnosti celého výpočtu. Stejně jako u metody RRT i zde platí, že výpočet pro omezení je časově náročnější než pro výpočet bez omezení. Zároveň je z tohoto grafu zřejmé, proč byla v tomto případě volena množina V s maximální hodnotou *0.5m*. Pro tuto hodnotu trval výpočet 10 minut pro nalezení trajektorie bez omezení a přibližně 20 minut pro trajektorii s omezením. Pro vyšší hodnoty v by byly měřeny časové nároky v řádech hodin.



Obrázek 8.7: Závislost délky výpočtu na délce segmentu v pro 3000 iterací

Nakonec jsou opět uvedeny průběhy nalezených cest (8.8) pomocí metody RRT*. Maximální počet iterací simulace bez omezení byl v tomto případě stanoven na 3000 a délka segmentu byla $v = 0.5m$. Na rozdíl od trajektorií nalezených pomocí metody RRT je vidět, že všechny tyto trajektorie mají velmi podobné průběhy a průměrná délka těchto trajektorií je nižší, než u RRT.



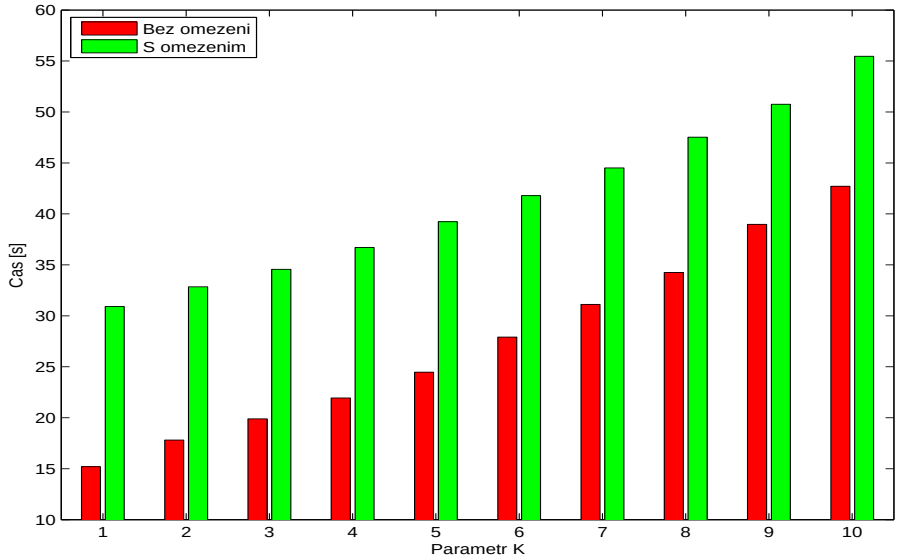
Obrázek 8.8: Nalezené cesty pomocí metody RRT*

Metoda Voroného diagramu

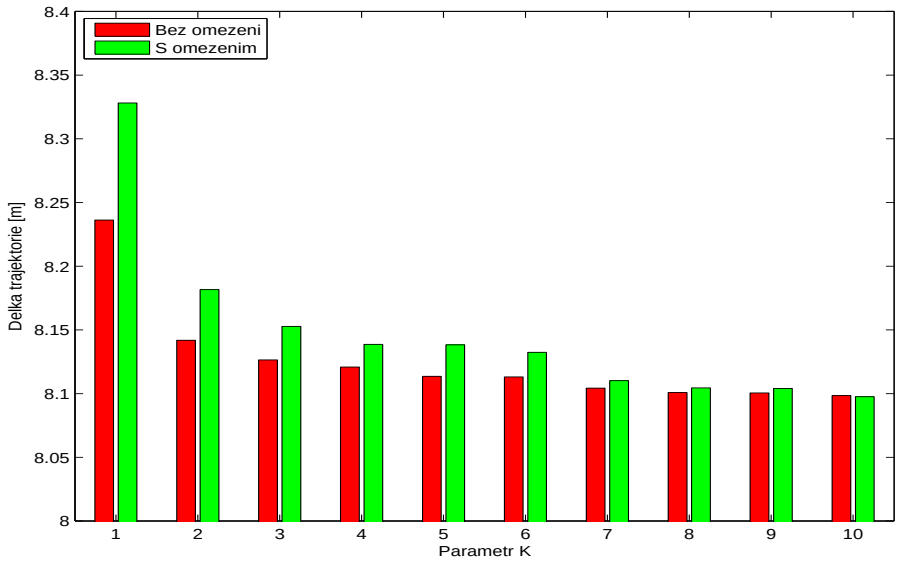
Poslední testovanou metodou je metoda Voroného diagramu. Odlišností metody Voroného diagramu od metod předchozích je vlastnost, že délka nalezené trajektorie pomocí této metody je při každé simulaci stejná, protože tato trajektorie je hledána pomocí grafu vytvořeného na základě znalosti polohy překážek a na rozdíl od metody rychle rostoucích stromů se zde nevyskytuje žádná generování náhodných veličin. Jedinou vlastností, která v tomto případě stojí za povšimnutí, je ovlivnění velikosti parametru K na časových nárocích výpočtu a délky nalezené trajektorie. O významu parametru K bylo vše důležité zmíněno v kapitole 5.2.

Na obrázku (8.9) je vykreslen graf s výpočetními nároky při měnícím se parametru K. Z vývoje grafu je vidět, že čím vyšší hodnota K je, tím déle opět výpočet trvá. Dále opět platí, že výpočet s omezením trvá déle, než výpočet bez omezení. To je způsobeno popisem překážek při řízení robotu, které jsou definovány pomocí více vrcholů než pro hledání cesty bez omezení.

Délka nalezené trajektorie pro stejný parametr K je vždy stejná, ale pokud se hodnota K mění, délka trajektorie se mění také. V kapitole 5.2 bylo řečeno, že parametr K zpřesňuje výpočet Voroného diagramu. Z grafu (8.10) je patrné, že čím přesnější je výpočet diagramu, tím kratší je nalezena cesta.

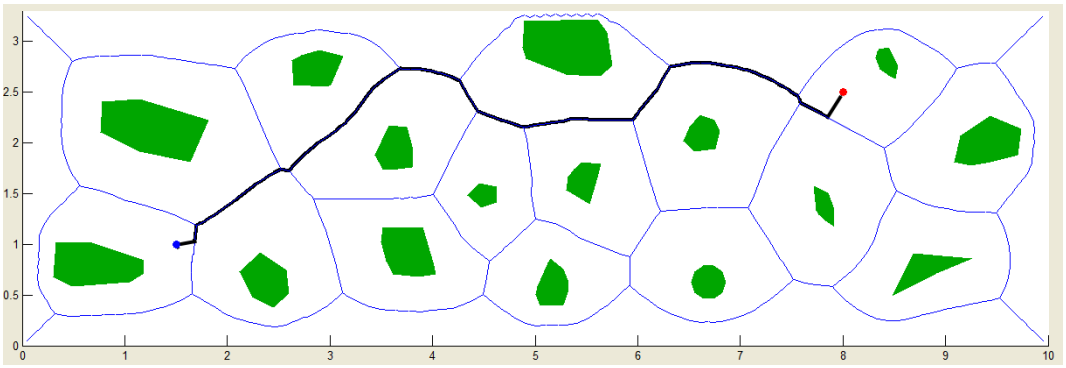


Obrázek 8.9: Vývoj časové náročnosti výpočtu s měnícím se parametrem K



Obrázek 8.10: Vývoj délky trajektorie s měnícím se parametrem K

Toto zlepšení je ale v řádech desetin metru díky lepšímu popisu, tvar trajektorie však zůstává stejný. Nalezená trajektorie pomocí Voroného diagramu je i s diagramem samotným vyobrazena na obrázku (8.11). Tato nalezená trasa má délku 8.1m, což je patrné i z grafu (8.10).



Obrázek 8.11: Nalezená trasa pomocí Voroného diagramu pro $K = 10$

Společné porovnání metod pro hledání trajektorie

Nyní budou porovnány výsledky dosažené implementovanými metodami hledání trajektorie.

Z výsledků simulací dosažených v předchozích kapitolách je zřejmé, že z hlediska časových nároků na výpočet je nejlepší zvolit metodu RRT s délkou segmentu $v = 0.5m$ a více, protože právě pro tyto parametry proběhlo hledání trajektorie v řádech desetin vteřiny. Bohužel, takto nalezené trasy nejsou z hlediska své délky optimální. Pokud by bylo zapotřebí dosáhnout nejkratší možné trajektorie, bylo by vhodné zvolit metodu RRT*, která poskytuje optimální výsledky z hlediska délky naplánované trajektorie. Nevýhodou této metody jsou její velké časové nároky na výpočet. Pro nalezení jedné trajektorie touto metodou bylo zapotřebí hledat trasu v řádech minut pro různé délky v. Kompromisem mezi těmito metodami je použití metody Voroného diagramu, kde byla nalezena trajektorie o délce 8.1m v řádově desítkách sekund pro různé parametry K.

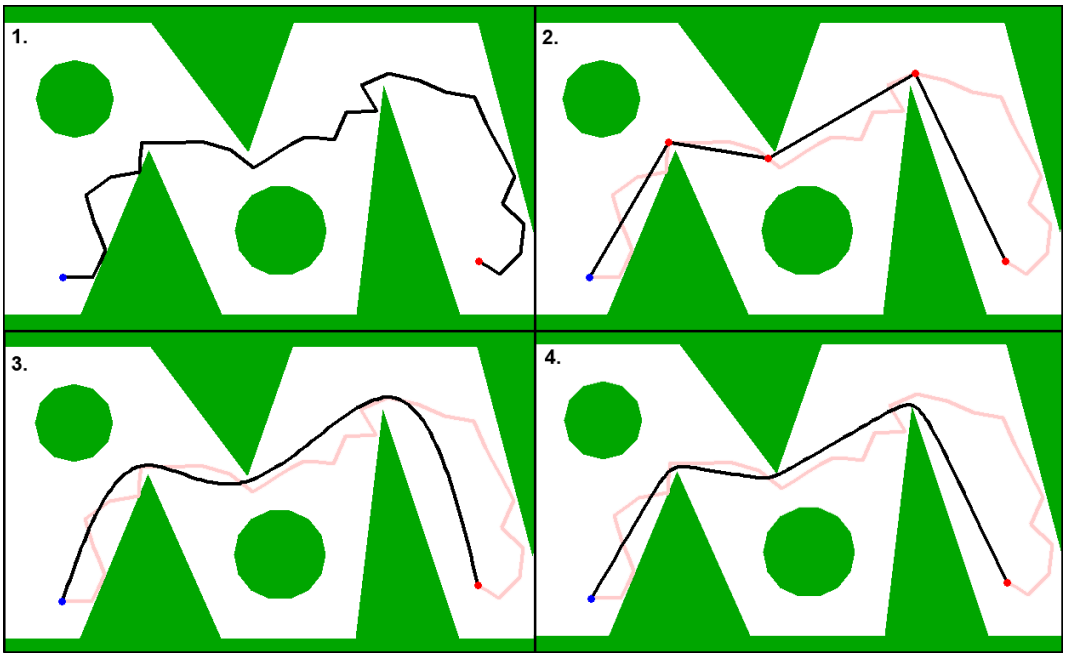
Dále bylo zjištěno, že při výpočtu trajektorie s omezením jsou časové nároky vyšší, než v případě výpočtu bez omezení. Tento fakt je závislý na velikosti robotu. Čím vyšší budou jeho rozměry, tím déle bude samotný výpočet u všech metod trvat.

Porovnání metod upravujících trajektorie

V kapitole 6.2 byly představeny metody upravující tvar a délku nalezených cest. Každá metoda byla aplikována na nalezenou referenční trajektorii pomocí metody RRT, viz obrázek (6.7). Nyní budou trajektorie získané pomocí metod z kapitoly 6.2 porovnány. Tyto trajektorie jsou vyobrazeny na (8.12)

společně s původní nalezenou trajektorií. V části č.1 je původní neupravovaná trajektorie, v části č.2 je trajektorie po vyhlazení, v části č.3 je interpolovaná trajektorie pomocí kubického interpolačního splinu a jako poslední, v části č. 4, je aproximovaná trajektorie pomocí Bézierovy křivky.

Z tohoto přehledu je vidět, že pomocí metod byly získány trajektorie, které jsou oproti původní trajektorii mnohem lépe realizovatelné robotem s diferenciálním podvozkem. Původní trajektorie měřila 9.32m a pomocí metod pro úpravy byla zkrácena na 6.36m v případě aplikace metody pro vyhlazování, 6.47m pro metodu interpolačního kubického splinu a v případě použití Bézierovy křivky měřila trajektorie 6.31m. Z těchto hodnot je dále patrné, že nejkratší cesta byla dosažena úpravou pomocí Bézierovy křivky, ovšem rozdíl mezi všemi upravenými délkami byl pouze 0.16m. Upravené trajektorie byly použitím metod zkráceny přibližně o třetinu původní délky.



Obrázek 8.12: Referenční a upravené trajektorie

Co se týče průběhu upravených trajektorií, je vidět, že metoda kubického interpolačního splinu poskytuje trajektorii, kterou by za určitých okolností byly schopny projet roboty se všemi typy podvozků. Zároveň je také dobré zmínit časovou náročnost při průjezdu jednotlivých tras robotem. V tomto ohledu bude pro diferenciálně řízeného robota vždy vhodná trasa získaná úpravou právě pomocí interpolačního kubického splinu, protože změny dru-

ých derivací v průběhu trajektorie jsou oproti zbývajícím metodám velmi malé a robot tuto trasu může projet ve vyšší rychlosti. U trajektorie získané pomocí Bézierovy křivky se jsou velké změny druhých derivací v místech, kde pro vyhlazenou trajektorii byly definovány vymezující body. V těchto místech by tedy robot musel hodně zpomalit v případě vyhlazené trajektorie dokonce zastavit, natočit se správným směrem a až poté pokračovat v jízdě. Z těchto důvodů by byl průjezd takové trasy časově náročnější.

Z hlediska časové náročnosti výpočtu bylo použití těchto metod velmi úsporné, protože výpočet každé metody trval přibližně vteřinu. Doba výpočtu je opět závislá na složitosti vyhlazené trajektorie. Čím více pak obsahuje vyhlazená trajektorie bodů vymezujících trajektorii, tím déle výpočet zbývajících metod trvá.

Na závěr této kapitoly je dobré říci, že v průběhu vypracování této práce bylo zjištěno, že nejvhodnější je v úloze plánování trajektorie použít pro nalezení trasy metodu RRT s vhodnou délkou segmentu v a následně na nalezenou trajektorii aplikovat metodu kubického interpolačního splinu, díky které je nalezená trasa zkrácena a zároveň vyhlazena. V případě, že by byla pro hledání trajektorie použita například metoda RRT*, hledání trasy by bylo časově daleko náročnější a pokud by se následně na nalezenou trasu aplikovala metoda kubického interpolačního splinu, byl by získán velmi podobný výsledek průběhu trajektorie. To samé by platilo pro kombinaci metody Voroného diagramu a kubického interpolačního splinu, kde by byl hlavním výsledným rozdílem čas výpočtu.

9 Závěr

Tato diplomová práce se zabývala řešením úlohy plánování trajektorie pro mobilní roboty. Nejprve se práce věnovala obecnému studiu problematiky plánování trajektorií a poté byly zvoleny tři vhodné metody plánování, které byly detailně popsány a následně implementovány ve výpočetním programu Matlab. Pro provedení výpočetních experimentů bylo vytvořeno vlastní grafické prostředí, ve kterém byly jednotlivé metody testovány.

Zvolené metody jako takové ovšem neposkytovaly vhodné výsledky z hlediska tvaru nalezených trajektorií, které nebyly hladké a v případě metod RRT a Voroného diagramu ani délkově optimální. Proto byla navržena řešení pro úpravu trajektorií. Pomocí metod vyhlazování, kubického interpolačního splinu a Bezierových křivek byly nalezené trajektorie vyhlazeny a zároveň také zkráceny.

Dále bylo navrženo řešení pro plánování trajektorie pro model diferenciálního mobilního robotu s určitými rozměry. Za předpokladu, že konstrukce robotu je obdélníkového tvaru bylo navrženo rozšíření překážek v pracovním prostoru o polovinu úhlopříčky z rozměru robotu. Výhoda tohoto řešení spočívala především v tom, že samotné implementace jednotlivých metod pro hledání trajektorie nemusely být upravovány a trajektorie byla hledána pouze pro jiný popis pracovního prostoru.

Na závěr byly vyhodnoceny získané výsledky pomocí implementovaných metod. Tyto výsledky jsou závislé na složitosti pracovního prostoru, na velikosti překážek a na počtu možných variant trajektorií, které lze z počátečního bodu do cílového bodu nalézt. Obecně ale platí, že nejrychlejší metodou hledání trajektorie z metod, které byly v práci popsány, je metoda RRT a nejlepší metodou z hlediska délky trajektorie je metoda RRT*. Metoda Voroného diagramu je kompromisem mezi těmito metodami a její největší předností je robustnost nalezené trasy, která je pro dané rozměry robotu nejkratší možná a zároveň by se robot pohyboval v maximálních odstupech mezi překážkami.

Literatura

[1] LAVALLE S. M.: Rapidly-exploring random trees: A new tool for path planning. 1998

[2] LAVALLE S. M.: Motion planning: The Essentials. Robotics and Automation Magazine, IEEE, 2011, 18.1: 79-89.

[3] Kwangjin Yang, Sukkarieh S.: 3D Smooth path planning for a UAV in cluttered natural environmentsIn: Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on. IEEE, 2008. p. 794-800.

[4] Kwangjin Yang, Daehan Jung, Sukkarieh S.: Continuous curvature path-smoothing algorithm using cubic B zier spiral curves for non-holonomic robots. Advanced Robotics, 2013, 27.4: 247-258.

[5] Karaman S., Walter M., Perez A., Frazzoli E. AND Teller S.: Anytime motion planning using the RRT*. In: Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011. p. 1478-1483.

[6] Berard R.: Path Planning, Dostupné z: http://eceuavweb.groups.et.byu.net/labs/guidance_labs/PathPlanning/index.html

[7] Martin S. C., Hillier N., Corke P.: Practical application of pseudospectral optimization to robot path planning. In: Australasian Conference on Robotics and Automation 2010. Australian Robotics and Automation Association, 2010.

[8] Subhas C. N.: Voronoi diagram, Advanced Computing and Microelectronics Unit, Indian Statistical Insituta, Kolkata, July 2008

[9] Wikipedie: Algoritmus A* [online]. 2015 [citováno 2015-04-30]. Dostupné z: http://cs.wikipedia.org/wiki/A*

[10] Sypták M.: Modelování a řízení mobilních kolových robotů [online]. 2009 [cit. 2015-04-30]. Diplomová práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Vedoucí práce František Šolc.

[11] Hlavičková I.: Interpolace pomocí splajnu, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Přednáška z předmětu BMA3, Brno 2009, Dostupné z: http://www.umat.feec.vutbr.cz/∼hlavicka/vyuka/BMA3_predn/2009/prednaska08.pdf

[12] Bastl B.: Bézierova křivka, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, KMA/GPM, Plzeň, Dostupné z: http://geometrie.kma.zcu.cz/index.php/www/content/download/1152/3264/file/GPM_Bezier.pdf

[13] Švec P.: Plánování trasy robota pomocí Voroného diagramů a dalších prostředků výpočetní geometrie [online]. 2006 [cit. 2015-04-30]. Diplomová práce. Masarykova univerzita. Fakulta informatiky. Vedoucí práce Petr Tobola. Dostupné z: http://is.muni.cz/th/39233/fi_m/.

[14] Gross T.: Plánování cesty mobilního robota. 2008 [cit. 2015-04-30]. Diplomová práce. Vysoké učení technické v Brně. Fakulta strojního inženýrství. Vedoucí práce: Jiří Dvořák

[15] Šindelář J.: Plánování cesty neholonomního mobilního robotu. 2010 [cit. 2015-04-30]. Diplomová práce. Vysoké učení technické v Brně. Fakulta strojního inženýrství. Vedoucí práce: Petr Krček

[16] Jirovský M.:Návrh, modelování a řízení diferenciálně řízeného čtyřkolového mobilního robotu. 2013. Bakalářská práce, Západočeská univerzita v Plzni, Katedra kybernetiky, Vedoucí práce: Miroslav Flídr

[17] Vaněk P.:Plánování pohybu technikami RRT. 2010. Bakalářská práce, České vysoké učení technické v Praze, Fakulta elektrotechnická, Vedoucí práce: Jan Faigl