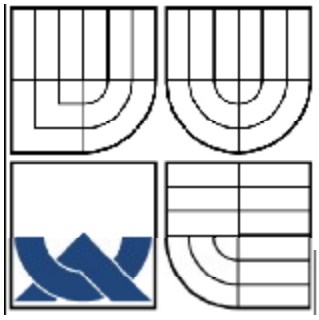


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

PLÁNOVÁNÍ CESTY NEHOLONOMNÍHO MOBILNÍHO ROBOTU

PATH PLANNING OF A NONHOLONOMIC MOBILE ROBOT

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE — JIŘÍ ŠINDELÁŘ
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR KRČEK
SUPERVISOR

BRNO 2010

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2009/2010

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Jiří Šindelář

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Aplikovaná informatika a řízení (3902T001)**

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

Plánování cesty neholonomního mobilního robotu

v anglickém jazyce:

Path planning of a nonholonomic mobile robot

Stručná charakteristika problematiky úkolu:

Úkolem plánování cesty robotu je nalezení cesty z počáteční do cílové pozice bez kolize se známými překážkami tak, aby ohodnocení cesty bylo minimální. Ohodnocení cesty je určeno hlavně délkou cesty a muže zahrnovat i další aspekty, jako např. obtížnost a riziko cesty. Jestliže jsou na pohyb robotu kladena nějaká dodatečná omezení (vyjma těch, která jsou spojena se způsobem modelování prostředí), pak je robot charakterizován přívlastkem neholonomní (nonholonomic).

Cíle diplomové práce:

1. Analyzovat dosavadní přístupy k plánování cesty robotu.
2. Implementovat vybrané metody plánování cesty neholonomního robotu.
3. Provést srovnávací experimenty implementovaných metod.

Seznam odborné literatury:

1. LaValle, S. M. & Kuffner, J. J. (2001) Rapidly-Exploring Random Trees: Progress and Prospects.

2. LaValle, S. M. (2006) Planning Algorithms.

Vedoucí diplomové práce: Ing. Petr Krček

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2009/2010.

V Brně, dne

L.S.

prof. RNDr. Ing. Miloš Šeda, Ph.D. Ředitel ústavu	doc. RNDr. Miroslav Doupovec, CSc. Děkan fakulty

LICENČNÍ SMLOUVA

(na místo tohoto listu vložte vyplněný a podepsaný list formuláře licenčního ujednání)

ABSTRAKT

Tato práce se zabývá plánováním cesty robota pomocí vybraných algoritmů. Konkrétně se jedná o metody: RRT (Rapidly-Exploring Random Trees), IGPPR (The Intelligent Global Path Planner With Replanning) a ISSD (Incorporating State Space Discretization). Teoretická část je věnována dosavadním přístupům k plánování cesty a vysvětlení výše zmíněných algoritmů. Praktická část pak popisuje implementaci jednotlivých metod, které jsou aplikovány na neholonomního mobilního robota, pracujícího ve dvourozměrném pracovním prostoru se statickými překážkami.

ABSTRACT

This thesis deals with robot path planning by means of selected methods. Specifically by the methods RRT, IGPPR and ISSD. The theoretical part contains the overview of existing methods for path planning and description of previously mentioned methods. The practical part describes implementation of each methods which are applied to nonholonomic mobile robot working in 2D workspace with static obstacles.

KLÍČOVÁ SLOVA

Plánování, robot, neholonomní, RRT, IGPPR, ISSD.

KEYWORDS

Planning, robot, nonholonomic, RRT, IGPPR, ISSD.

Obsah

1 ÚVOD

Tato diplomová práce se zabývá plánováním cesty robotu. Základní podstatou plánování pohybu je nalezení nekolizní cílové cesty mezi překážkami (ať už statickými nebo dynamicky se měnícími). To je čistě geometrický problém, který vypadá zdánlivě jednoduše. Ve skutečnosti, vyjma u robotů s několika málo stupni volnosti, je výpočetně velmi náročný.

V této oblasti byl za poslední desetiletí učiněn velký pokrok a bylo uvedeno velké množství nových algoritmů pro plánování pohybu, které jsou úspěšně aplikovány pro řešení náročných problému plánování pohybu robotu i s mnoha stupni volnosti. Autonomní roboty mohou v současné době řešit složité úlohy v komplikovaných prostředích. Plánování pohybu se uplatňuje v různých oborech, např. u počítačové simulace nebo v chirurgii (příkladem může být plánovač použitý neurochirurgy k výpočtu pohybu kloubového robotu s lineárním pohonem pro operaci nádorů mozku).

Samotné plánování pohybu se u každé metody odlišuje, proto musíme velkou vahou dbát také na vhodném výběru plánovače. Tento výběr nám ovlivňuje např. typ robotu (holonomní či neholonomní) nebo požadavek na kvalitu či optimalizaci nalezené cesty.

Cílem této diplomové práce bylo analyzovat dosavadní přístupy k plánování cesty robotu a navrhnout simulační program, který za pomocí implementovaných metod nalezne cestu ze startovní pozice do cílové. Pro implementaci byla vybrána metoda RRT (*Rapidly-Exploring Random Trees*), IGPPR (*The Intelligent Global Path Planner With Replanning)* a ISSD (*Incorporating State Space Discretization*). Nejprve je vysvětlen samotný princip těchto metod. V implementační části je uvedeno konkrétní řešení či možné vylepšení pro danou metodu. V experimentální části pak tyto metody byly porovnány a vyhodnoceny z hlediska efektivnosti metod.

2 ROBOT A OKOLNÍ PROSTŘEDÍ

2.1 Robot

Pod pojmem robot si můžeme představit stroj, který je schopen vykonávat zadané úkoly s určitou mírou samostatnosti, podle algoritmů definovaných zadavatelem.

Podle schopnosti robotu pohybovat se v prostředí rozlišujeme roboty na stacionární a mobilní. Stacionární roboty se nemohou pohybovat z místa na místo (jedná se o průmyslové manipulátory, které tvoří kloubová ruka). Jejich aplikace je dnes hojně využívána v mnoha odvětvích (např. při svařování karoserie automobilů, manipulaci, lakování atd.). Oproti tomu mobilní robot má schopnost přesouvat se po svém pracovním prostředí a není fixován na jednu fyzickou lokaci.

Mezi nejběžnější typy mobilních robotů patří roboty kolové, chodící a v neposlední řadě roboty létající (např. bezpilotní letadla).

Robot může pracovat pod přímým řízením člověka (manipulátory) nebo autonomně.

Pracuje-li robot autonomně, pak robotem rozumíme inteligentní stroj, který je schopen zadané úkoly vykonávat zcela samostatně, tedy bez zásahu člověka. Za tuto schopnost vděčí svému mozku – počítači, který řídí veškerou činnost robotu.

Pro některé druhy robotů s inteligencí se používá přesnějšího označení:

- *Droid* – jakýkoliv inteligentní a samočinný robot.
- *Android* – robot podobný člověku – obvykle se očekává biologické složení.
- *Humanoid* – robot podobný člověku principiální stavbou těla a zejména způsobem pohybu.
- *Kyborg* (kybernetický organismus) – živá bytost obohacená o mechanické či elektronické součástky. V extrémním případě může z původní bytosti zůstat pouze mozek.

Obr. 1 Humanoidní robot ASIMO od společnosti Honda [11]

Obr. 2 Android [11]

Obr. 3 Robotický pes Aibo [7]

Struktura robotu se obvykle skládá z těchto subsystémů [8]:

- **Senzorický systém:** Soustava technických zařízení, která snímají a následně zpracovávají údaje o okolním prostředí (příkladem takového senzoru může být třeba i obyčejný spínač použitý jako nárazník pro informaci, že se v cestě nachází překážka).

- **Kognitivní systém:** Systém, který zpracovává a vyhodnocuje informace ze senzorů. Uskutečňuje inteligentní chování robotu. Zahrnuje v sobě subsystémy jako jsou:
  - Interní model prostředí: Shromažďuje poznatky o uspořádání a vlastnostech prostředí, ve kterém robot vykonává úkoly.
  - Plánovač: Tato část vytváří posloupnosti přípustných akcí, vedoucích od aktuálního stavu robotu a prostředí ke stavu cílovému. Plánovač pracuje s informacemi obsaženými ve vnitřním modelu prostředí.
  - Realizátor plánů: Podrobně rozpracovává provedení jednotlivých akcí do podoby vhodné pro řízení aktuátorů.

- Aktuátory: Umožňují fyzické působení robotu na prostředí. Jsou to lokomoční systémy (kola, pásy, nohy) a manipulační systémy (ruce, pracovní nástroje apod.)
- Komunikační systém: Umožňuje komunikaci s nadřazeným operátorem nebo jiným inteligentním systémem.

2.2 Modely kolových vozidel

Jelikož je tato práce zaměřena na autonomní mobilní roboty, přiblížíme si jejich základní typy. Většinu mobilních robotů můžeme rozdělit do těchto tří kategorií [10]:

Auto

Tato kategorie zahrnuje roboty, které se nedokážou otáčet na místě. Pro změnu jejich orientace vyžadují natočení kol či nápravy a následný pohyb vpřed nebo vzad. Tento způsob pohybu je označován jako *Ackermanovo* řízení, které zahrnuje většinu dnešních automobilů.

Tank

Kategorie tank zahrnuje všechny typy robotů, které jsou schopné otáčet se na místě kolem své osy. Jejich pohyb je omezen pouze vpřed nebo vzad. Často je nazýváme diferenciálně řízenými, jelikož změna orientace závisí na rozdílu rychlosti levého a pravého kola/pásu.

Všesměroví

Všesměroví roboti mají schopnost pohybovat se všemi směry bez omezení, a tudíž nemají problém otočit se na místě. Tato kategorie není příliš rozšířená, i když poslední dobou začíná být populární v robotickém fotbalu – např. tým fu-fighters. Často je nalezneme pod označením omnidrive.

Jednotlivé kategorie si popíšeme podrobněji [10].

Auto

Tento typ robotu patří do skupiny tzv. ***neholonomních*** (*nonholonomic*) robotů. Robot je označen jako neholonomní, je-li počet řiditelných stupňů volnosti menší než jejich celkový počet. Jednoduše řečeno, na pohyb neholonomního robotu jsou kladena nějaká dodatečná omezení (vyjma těch která jsou spojena se způsobem modelování prostředí). Naopak, ***holonomní*** *(holonomic)* vozidlo může měnit svoji rychlost nezávisle ve všech směrech. Pokud je tedy počet řiditelných stupňů volnosti stejný jako jejich celkový počet, pak je robot holonomní.

Příkladem neholonomního robotu je právě robot typu auto, jelikož je jeho pohyb do stran omezen maximálním natočením předních kol.

Konstrukce jednoduchého modelu auta je tvořena dvěma zadními koly, která jsou pevně spojena na společné nápravě, a dvěma předními natáčecími koly, která určují úhel φ a tím pádem směr pohybu (obr.4). Toto natáčení je omezené maximálním povoleným úhlem φ_{max}, jehož hodnota leží v intervalu $[0, \pi/2]$ a $|\varphi| \leq \varphi_{max}$.

Auto se může pohybovat vpřed i vzad a podle natočení předních (řídících) kol se pohybuje buď po přímce (jsou-li natáčecí kola rovnoběžně s osou auta) anebo po kružnici. Manévrovací schopnost auta závisí jednak na vzdálenosti mezi osami L (čím kratší tím se snáze zatáčí) a dále na limitním úhlu natočení předního kola.

Nechť referenčním bodem auta je střed nápravy pevných kol. Protože pozice a orientace robotu v konfiguračním prostoru je dána souřadnicemi *x* a *y*, a celkovým natočením θ, pohyb auta můžeme reprezentovat soustavou rovnic

$$\dot{x} = f_1(x, y, \theta, s, \varphi) \qquad (2.1)$$
$$\dot{y} = f_2(x, y, \theta, s, \varphi)$$
$$\dot{\theta} = f_3(x, y, \theta, s, \varphi).$$

Pro malý časový interval *dt*, který se limitně blíží k nule platí následující vztah

$$\tan\theta = \frac{dy}{dx} = \frac{\dot{y}}{\dot{x}}\,. \qquad (2.2)$$

Z trigonometrie víme, že $\tan q = \frac{\sin\theta}{\cos\theta}$ a dosazením tohoto vztahu do (2.2) obdržíme

$$-\dot{x}.\sin\theta + \dot{y}.\cos\theta = 0\,. \qquad (2.3)$$

Toto omezení je splněno pokud $\dot{x} = \cos\theta$ a $\dot{y} = \sin\theta$. Protože je tato změna přímo úměrná rychlosti *s* auta, první dvě skalární složky pohybové rovnice jsou

$$\dot{x} = s.\cos\theta \qquad (2.4)$$
$$\dot{y} = s.\sin\theta$$

Předpokládáme-li nenulové natočení řídícího kola ($\varphi \neq 0$), a tedy pohyb po kružnici, zajímá nás střed a poloměr této kružnice ρ. Jak je vidět z obrázku 4, střed kružnice leží v průsečíku pevné a natáčecí osy nápravy (vlevo, či vpravo podle směru natočení). Poloměr této kružnice je roven $\rho = L / \tan\varphi$. Nechť *w* označuje ujetou vzdálenost autem (integrál rychlosti), pro kterou platí $dw = \rho d\theta$. Dosazením vztahu pro poloměr kružnice ρ obdržíme

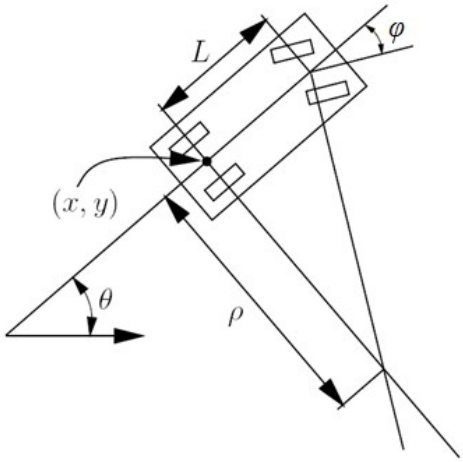
$$d\theta = \frac{\tan\varphi}{L}\, dw \qquad (2.5)$$

a vydělením obou stran rovnice *dt* a využitím faktu $\dot{w} = s$ dostáváme

$$\dot{\theta} = \frac{s}{L}\tan\varphi \qquad (2.6)$$

Pohyb jednoduchého auta je tedy popsán následujícími pohybovými rovnicemi:

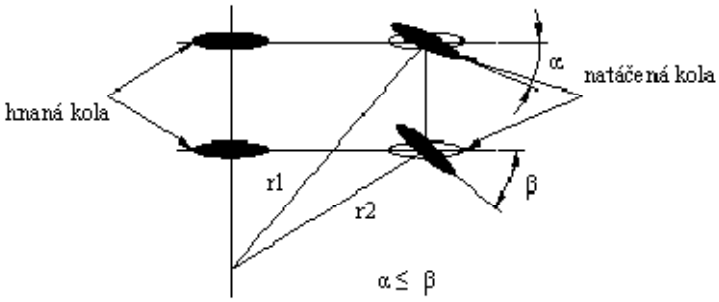
$$\dot{x} = s.\cos\theta \qquad (2.7)$$
$$\dot{y} = s.\sin\theta$$
$$\dot{\theta} = \frac{s}{L}.\tan\varphi$$



Obr. 4 Model podvozku auta [10]

Ackermanovo řízení

Ackermanovo řízení je aplikováno u většiny dnešních automobilů. Mechanismus tohoto řízení spočívá v uspořádání kol tak, že přední řídící kola jsou natáčena každé pod jiným úhlem. Aby žádné kolo nebylo ve smyku, je nutné, aby vnitřní kolo zatáčelo více než vnější.



Obr. 5 Schéma Ackermanova podvozku [8]

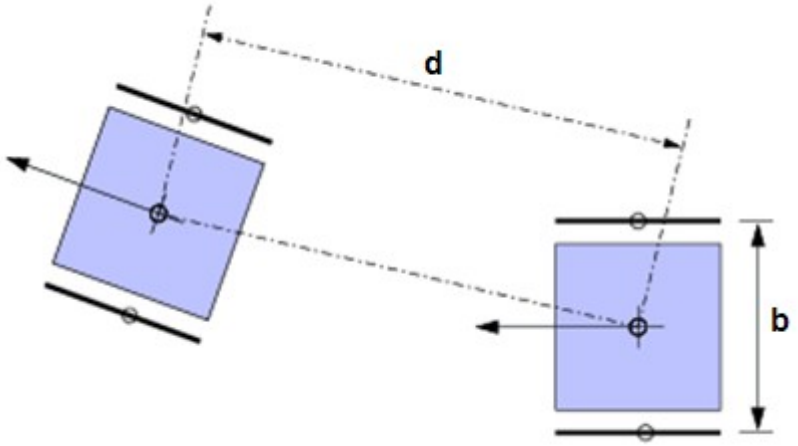
Tank

Základním rysem modelu tanku jsou dvě nezávisle poháněná kola/nápravy. Pokud se obě točí stejně rychle stejným směrem, robot jede rovně. Pokud se naopak točí opačným směrem, točí se celý robot na místě kolem středu mezi nápravami. To je také nejčastější referenční bod.

Pokud je rozdíl rychlostí mezi nápravami nenulový, ale obě nápravy se točí na stejnou stranu, pohybuje se i tento robot po kružnici. Vzdálenost středu této kružnice je dána poměrem obou rychlostí: $R = \frac{1}{2} \cdot b \cdot (vL + vR) / (vL - vR)$, kde b je vzdálenost kol od sebe (rozchod) a vL a vR jsou rychlosti.

Pokud levé kolečko ujede vzdálenost dL a pravé dR změní se orientace o úhel θ (v radiánech) $\theta = (dL - dR)/b$, kde b je již zmiňovaný rozchod. Celková ujetá vzdálenost d (počítaná pro střed poháněné osy) je pak $d = (dL + dR)/2$.

Natočení robota závisí pouze na rozdílu celkové ujeté vzdálenosti pravého a levého kolečka a nikoliv na průběhu jednotlivých změn. K výpočtu směru tedy stačí i jednoduchý čítač, který lze snadno realizovat i v jednočipu (nejsou třeba goniometrické funkce).

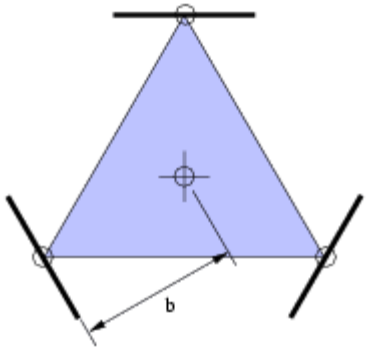


Obr. 6 Model podvozku tanku [10]

Všesměroví

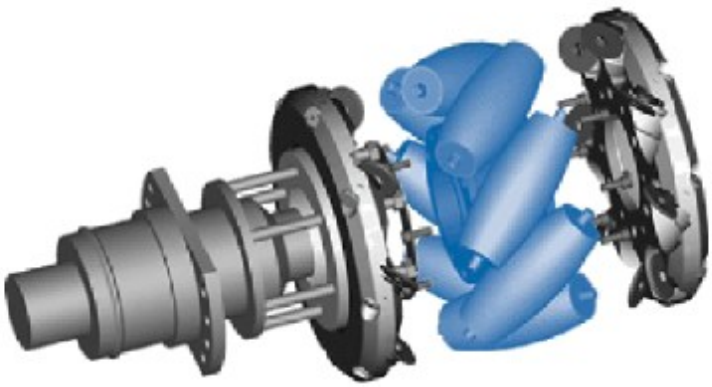
Doposud zmiňované modely byly neholonomní. Všesměrový podvozek (*omnidirectional drive*) je speciálním případem obecnějšího konceptu holonomicity.

Diferenciálně řízení roboti se sice mohou otáčet na místě, ale stále se mohou pohybovat pouze směrem kolmým k ose kol. Pohybu do strany brání kola. Co kdybychom obyčejná kola nahradili koly speciálními, která by místo pneumatiky měla spoustu nezávislých koleček (tato kolečka je možné si představit jako nařezanou pneumatiku). Pokud by na robota byla vyvinuta nějaká vnější síla působící do strany, nic by nyní pohybu nebránilo. Vhodnějším uspořádáním kol a zvýšením jejich počtu minimálně na 3 (přece jenom chceme řídit 3 stupně volnosti) dosáhneme požadované všesměrovosti. Minimální varianta všesměrového podvozku je na obrázku 7.



Obr. 7 Model všesměrového podvozku [10]

V praxi se můžeme setkat se dvěma typy všesměrových kol. U prvního typu jsou na obvodu běžného kola umístěny pasivní válečky, které jsou natočeny pod stejným úhlem. Druhý typ má válečky uspořádány do kruhu kolmo na obvod kola. Oba typy uspořádaní mají minimálně tři pasivní válečky. [8]



Obr. 8 Uspořádání všesměrového kola [8]

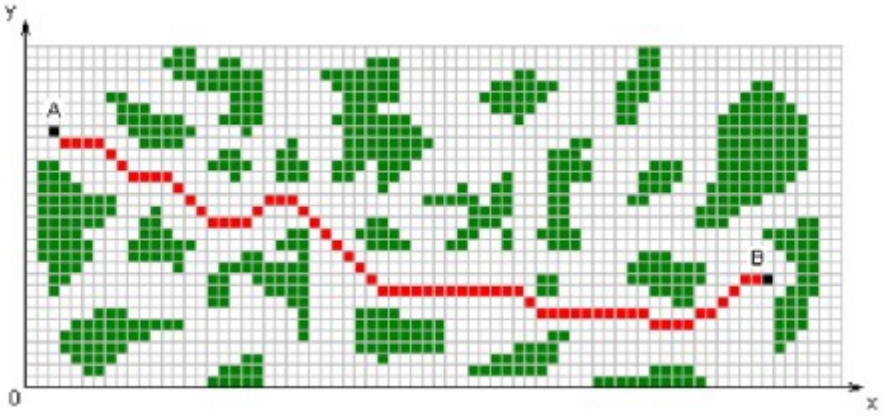
Obr. 9 Aplikace všesměrového kola a jeho detail [7]

2.3 Reprezentace prostředí

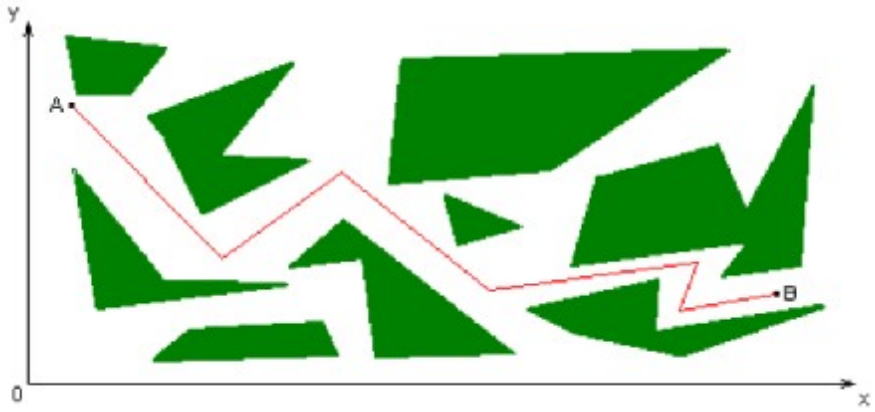
Prostředí můžeme reprezentovat buď diskrétně anebo spojitě. [9]

Diskrétní prostředí je prostor skládající se z buněk. Počet buněk závisí na velikosti scény, velikosti robota a na požadavku přesnosti překážek. Velký počet buněk zajistí větší členitost a přesnost zobrazení překážek, ale zvýší výpočetní náročnost. Velikost buňky nesmí být menší, než je velikost daného robota. Buňky mohou mít různé tvary a velikosti.

Spojité prostředí je kontinuální prostor, tudíž není rozdělen na buňky jako u diskrétního prostředí, tudíž lze překážky mapovat přesněji. Směr pohybu robota ani tvar překážek není omezen na čtvercovou síť. Oproti diskrétnímu prostředí se však obecně vyznačuje složitější náročností na prohledávání.



Obr. 10 Diskrétní prostředí [9]



Obr. 11 Spojité prostředí [9]

Stavový prostor

Stavový prostor je definován jako čtveřice X= [S, P, s_0, C], kde
S - je množina stavů
P - je množina přechodů mezi stavy
s_0 - je počáteční stav
C - je množina cílových stavů.

Stavový prostor *X* bývá nejčastěji reprezentován orientovaným grafem. V tomto grafu pak uzly představují jednotlivé stavy a orientované hrany přechod mezi stavy. Úlohu plánování pohybu tedy můžeme formulovat jako hledání cesty v grafu stavového prostoru.

Jelikož je pro robota určující prostor kde plní zadané úkoly, potřebuje mít k dispozici jeho zjednodušený model, aby se v tomto prostoru orientoval. Čím přesnější tento model je, tím stoupá i jeho výpočetní náročnost. Pro usnadnění vytvoření takového modelu bylo zavedeno několik pojmů [6]:

Pracovní prostor

Jedná se o prostor, ve kterém se robot pohybuje a vykonává úkoly. Označuje se *W* a lze ho reprezentovat jako N-rozměrný euklidovský prostor R^N, kde $N = 2, 3$ v závislosti na tom, je-li robot schopen překonávat různé překážky, případně stoupání nebo klesání.

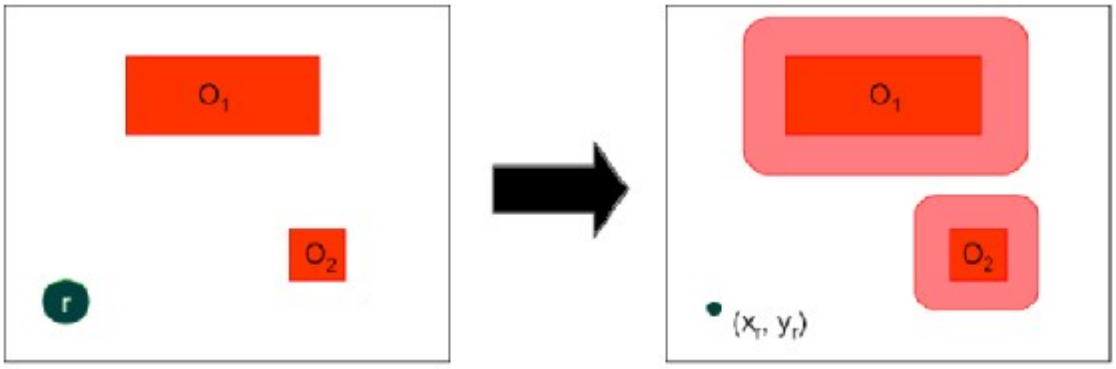
V pracovním prostoru *W* se navíc mohou vyskytovat neprůchozí objekty (překážky), které omezují pohyb robota. Překážky rozdělujeme na statické, které nemění svoji polohu nebo velikost v čase (např. budovy, ploty apod.) a dynamické, u kterých musí robot určit jejich vlastnosti v době setkání a podle nich se zařídit (např. lidé či další roboti).

Aby bylo možné matematicky popsat pozici a orientaci robotu v pracovním prostoru, byl zaveden pojem konfigurační prostor *C*.

Konfigurační prostor

Robota můžeme chápat jako systém, který se nachází v určitém stavu. Konfigurační prostor je pak množina všech přípustných stavů (konfigurací), ve kterých se může robot nacházet.

Hlavní myšlenkou konfiguračního prostoru je reprezentace robotu jako bodu, čímž se problém značně zjednoduší. Při této redukci se překážky do prostoru zmapují o zvětšený rozměr, který robot zabírá v pracovním prostoru.



Obr. 12 Redukce robotu z pracovního prostoru do konfiguračního [6]

Jelikož pracovní prostor v sobě zahrnuje překážky, je vhodné si vymezit prostor, ve kterém jsou konfigurace robotu přípustné. Tento prostor nazýváme ***volným konfiguračním prostorem*** C_{free}. Jedná se o množinu všech konfigurací nekolidujících s překážkami a současně vyhovující všem omezením kladených na robota (např. maximální úhel natočení kloubu či kol).

Oproti tomu část stavového prostoru, která je obsazena překážkami označujeme jako ***kolizní konfigurační prostor*** C_{obst}. Jedná se o prostor všech konfigurací robotu, při kterých dochází ke kolizi robotu s některou překážkou či při porušení omezení kladených na pohyb robotu.

Prostřednictvím konfiguračního prostoru tedy redukujeme úlohu plánování pohybu robotu na plánování pohybu bodu uvnitř robotova volného konfiguračního prostoru. Nutno ovšem poznamenat, že toto platí pouze u typů robotů, kde lze zvětšit rozměr překážek.

Aby plánování mohlo být vůbec úspěšné, musí startovní i cílová konfigurace náležet volnému konfiguračnímu prostoru.

Konfigurací robotu q rozumíme vektor, jehož složky jednoznačně určují pozici a orientaci robota v konfiguračním prostoru. Robot pracující v 2D pracovním prostoru, který může provádět translaci a rotaci má tři stupně volnosti (translace je možná v osách x a y, rotace okolo osy z). Jeho konfigurační prostor je tudíž třírozměrný.

$$\mathbf{q} = (\mathbf{x}, \mathbf{y}, \mathrm{q}) \in R^2 \times [\,0, 2\pi) \qquad (1)$$

kde: x, y - osy souřadnice
q - úhel natočení robota

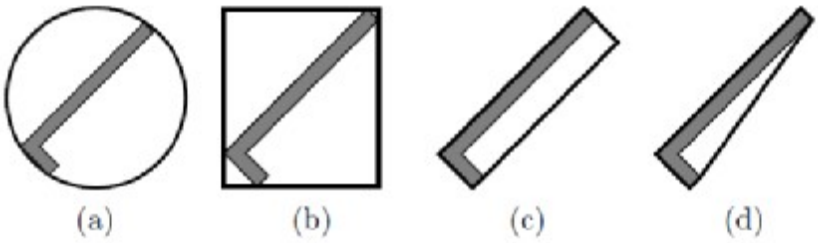
2.4 Detekce kolizí

Aby se robot dokázal vyhnout překážkám v pracovním prostoru, musí v sobě plánovací algoritmus zahrnovat určitou funkci, která určí, zda se daná konfigurace robotu nachází ve volném konfiguračním prostoru. Tuto funkci $D: C_{free} \rightarrow \{True, False\}$ označujeme jako detektor kolizí.

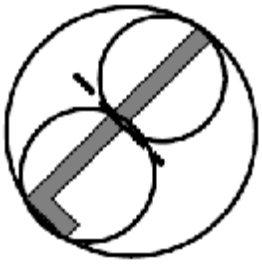
Existuje více metod jak tuto problematiku řešit. Záleží na reprezentaci daného pracovního prostoru i samotného robotu. Jsou-li hrany překážek i robotu reprezentovány pomocí úseček ve spojitém prostředí, nejjednodušším způsobem je *testování průsečíku* těchto úseček. Robot se rozdělí na jednotlivé úsečky, které se následně testují s úsečkami všech překážek. Pokud některá z hran robotu protíná některou 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.

V případě složitějších prostředí a robotů složitého tvaru by bylo nutné použít např. metodu s hierarchií ohraničujících těles (*Bounding Volume Hierarchy*). [1]

Hierarchické metody obvykle rozloží každý objekt (překážky či robota) do stromu. Každý vrchol stromu reprezentuje ohraničenou oblast, která obsahuje některou podmnožinu objektu. Ohraničující oblast kořenového vrcholu (*root*) obsahuje celé tělo. Na obrázku 12.1 můžeme vidět čtyři různé druhy ohraničující oblastí: (a) koule, (b) ohraničený box zarovnaný podle os, (c) orientovaný ohraničený box a (d) konvexní trup. Každá vymezená oblast obvykle poskytuje vyšší přesnost vůči zkoumanému subjektu než ta předchozí, ale zároveň je náročnější na testování překrývání párů. Na obrázku 12.2 můžeme pozorovat rozdělení ohraničující oblasti objektu tvaru L. Velký kruh znázorňuje ohraničující oblast pro vrchol, který pokrývá celý objekt. Po rozdělení podél přerušované čáry jsou používány dva menší kruhy. Každý kruh odpovídá vrcholu potomka.



Obr. 12.1 Čtyři různé druhy ohraničující oblastí [1]

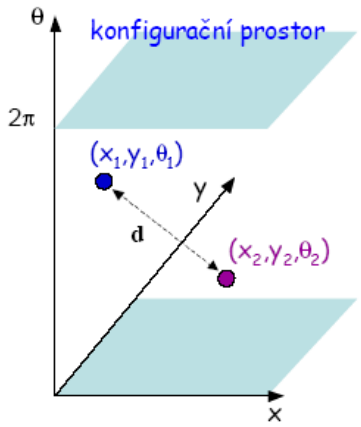


Obr. 12.2 Rozdělení ohraničené oblasti [1]

Pro určení zda se dva objekty (označme jako E a F) nacházejí v kolizi, se testuje, zda se ohraničující oblasti kořenů stromů T_E a T_F protínají. Pokud se neprotínají, pak objekty E a F nejsou v kolizi. Pokud se ovšem ohraničující oblasti protínají, pak jsou porovnávány ohraničující oblasti potomků T_E s ohraničujícími oblastmi T_F.

2.5 Metrika v konfiguračním prostoru

Nezbytnou částí plánovacího algoritmu je funkce, která dokáže určit vzdálenost mezi dvěma body (konfiguracemi) v konfiguračním prostoru *C*. Tato vzdálenost je určena tzv. *metrikou*.



Obr. 13 Určení vzdálenosti mezi dvěmi konfiguracemi [6]

Metrika ρ je zobrazení $\rho: C \times C \rightarrow R$, kdy pro každé $q_1, q_2, q_3 \in C$ platí následující axiomy [1]:

1) Axiom totožnosti
   $\rho(q_1, q_2) = 0$, jestliže $q_1 = q_2$
2) Axiom symetrie
   $\rho(q_1, q_2) = \rho(q_2, q_1)$
3) Trojúhelníkové nerovnosti
   $\rho(q_1, q_3) \leq \rho(q_1, q_2) + \rho(q_2, q_3)$
4) Z axiomů 1 a 3 vyplývá také nezápornost
   $\rho(q_1, q_2) \geq 0$

Hodnotu $\rho(q_1,q_2)$ potom nazýváme vzdáleností *d*.

Metrika je vždy definována pro určitý prostor. V závislosti na konkrétní aplikaci se volí taková metrika, která vzdálenost dvou konfigurací v pracovním prostoru popisuje nejlépe. Pro euklidovský prostor je zavedena euklidovská metrika ρ, definována následujícím vztahem:

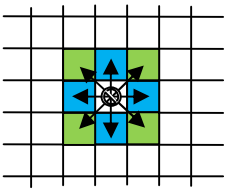
$$\rho(X,Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + .. + (x_n - y_n)^2} \qquad (2)$$

kde: $X = (x_1, x_2, \ldots, x_n) \in R^n$

$Y = (y_1, y_2, \ldots, y_n) \in R^n$

3 PLÁNOVÁNÍ POHYBU ROBOTU

Jelikož se v daném pracovním prostředí robotu vyskytují překážky, pohyb robotu je omezený. Toto omezení je současně závislé na rychlosti robotu (konstantní nebo proměnné) a na způsobu reprezentace prostředí. U spojitého prostředí není pohyb robotu nijak omezen. Naopak u diskrétní reprezentace prostředí je pohyb robotu omezen strukturou daného prostředí. Robot si následně vybírá směr nekolidující s překážkami. Často bývá povoleno pouze 8 možných směrů pohybu (viz obr.14).



Obr. 14 Možné směry pohybu v 2D diskrétním prostředí

Problematika plánování pohybu robotu spočívá ve schopnosti vytvořit si vnitřní model prostředí a v něm nalézt takovou posloupnost akcí, aby se při jejich provedení dostal z počátečního stavu do cílového. Tato posloupnost akcí se nazývá plán.

Plánování se skládá ze dvou kroků a to *globálního* a *lokálního* plánování.

Globální plánování – toto plánování se provádí ještě před samotným vykonáním pohybu robotu. Jeho úkolem je nalézt nekolizní cestu z počáteční pozice do cílové, proto před samotným plánováním vyžaduje model daného pracovního prostředí.

Lokální plánování – toto plánování má na starosti vlastní řízení pohybu robota po nalezené cestě, určené globálním plánováním. Proto musí lokální plánovač také zohledňovat případná omezení (omezení pohybu robota, nové překážky apod.).

Zná-li robot mapu prostředí, ve kterém se pohybuje, řeší problém vlastní lokalizace, případně problém plánování pohybu na cestě k cíli. Nemá-li robot k dispozici mapu prostředí, musí řešit lokalizaci a plánování souběžně.

3.1 Metody pro plánování cesty robota

Samotných plánovacích metod existuje nespočetné množství, jelikož různé z nich vznikají určitou modifikací původního algoritmu. Následující přehled proto uvádí základní typy těchto metod.

1) **Metody rozkladu do buněk** (*Cell decomposition*)
2) **Mapy cest** (*Roadmaps)*
3) **Potenciálová pole** (*Potential fields)*

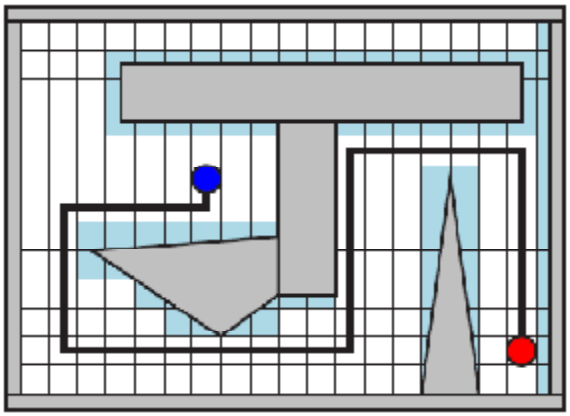
Tyto metody se vždy skládají ze dvou fází. V první fázi se provádí tzv. *předzpracování*, kdy se popisuje pracovní prostor pomocí grafu, mřížky nebo funkce. Pak následuje fáze tzv. *dotazovací*, kde je již hledána cesta mezi počáteční a cílovou konfigurací robota. [6]

3.1.1 Metody rozkladu do buněk

Princip této metody spočívá v rozkladu daného prostředí do buněk, které jsou určitého tvaru a velikosti. U všech buněk se určí, zda náleží do volného nebo obsazeného prostoru a rovněž se určí jejich sousedé. Z těchto informací se vytvoří tzv. graf sousednosti, jehož vrcholy představují buňky, které neobsahují překážku a hrany představují spojnice mezi těmito vrcholy. Pro rozložení daného prostředí se používá bud exaktního anebo aproximativního rozkladu.

3.1.1.1 Aproximativní rozklad

Celé prostředí je rozděleno do buněk stejného tvaru (nejčastěji čtvercového). Rozdělení prostoru převádí daný prostor na diskrétní tvar. Buňky malého rozměru zpřesňují daný prostor, avšak zvyšují výpočetní a paměťovou náročnost. Parametr buněk bývá většinou konstantní, ale je možno použít i buňky různých velikostí. Obsahuje-li buňka alespoň jeden bod nějaké překážky, je celá označená jako obsazená. Z tohoto důvodu se může stát, že metoda nemusí nalézt cestu k cíli, když ve skutečnosti existuje. Proto není tato metoda kompletní.

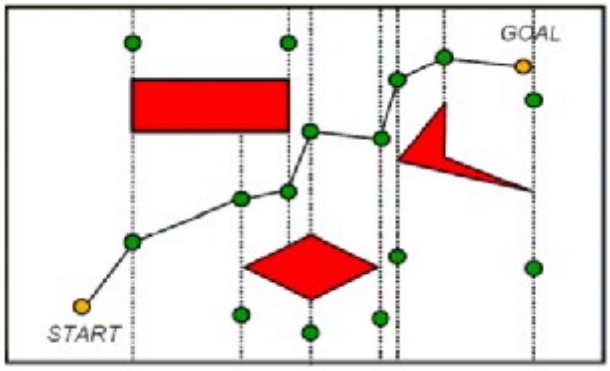


Obr. 15 Aproximativní rozklad na čtvercové buňky konstantní velikosti [6]

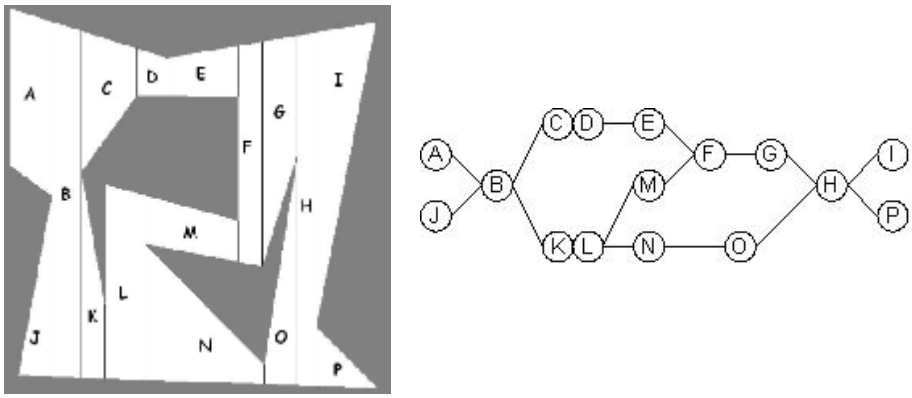
3.1.1.2 Exaktní rozklad

Principem tohoto rozkladu je rozdělit volný prostor do množiny nepřekrývajících se buněk jednoduchého tvaru. Často se využívá lichoběžníků nebo trojúhelníků. Výsledná cesta z počátečního bodu do cílového se skládá z těchto bodů a bodů přechodu, které robot využívá k bezpečnému pohybu mezi překážkami. Bod přechodu se nalézá vždy na hranici mezi jednotlivými buňkami.

Exaktní metoda je metodou kompletní, což znamená, že vždy nalezne cestu (pokud existuje), v opačném případě ověří, že neexistuje.



Obr. 16 Cesta nalezená pomocí exaktního rozkladu [6]



Obr. 17 Prostor rozdělený lichoběžníkovou dekompozicí a výsledný graf sousednosti [12]

3.1.2 Mapy cest

Tyto metody využijeme především, víme-li, že daným prostředím budeme plánovat více cest. Princip metody spočívá ve vytvoření mapy cest (*roadmap*), která má podobu grafu reprezentujícího volný pracovní prostor. Mapa cest obsahuje pouze hlavní cesty, zajišťující bezpečný průchod robota mezi překážkami. Jedná se o obdobu mapy dálnic, drážní mapy kolejních koridorů či mnoho dalších. [12]

Vrcholy grafu reprezentují různé body v pracovním prostoru (podle konkrétní metody). Hrany grafu pak tvoří cesty, po kterých se robot může pohybovat. Přidáme-li do grafu počáteční a cílovou pozici robota jako další vrcholy grafu, problém nalezení cesty ze startu do cíle přechází na problém hledání cesty v grafu za použití některého ze známých algoritmů.

Vlastnosti

Metody mapy cest mají velice příznivé vlastnosti, týkající se nalezení cesty. Pro tyto metody platí následující [12]:

- existuje-li cesta, je nalezena
- je-li cesta nalezena, jedná se o nejoptimálnější cestu vzhledem k ohodnocení hran
- není-li cesta nalezena, pak cesta neexistuje.

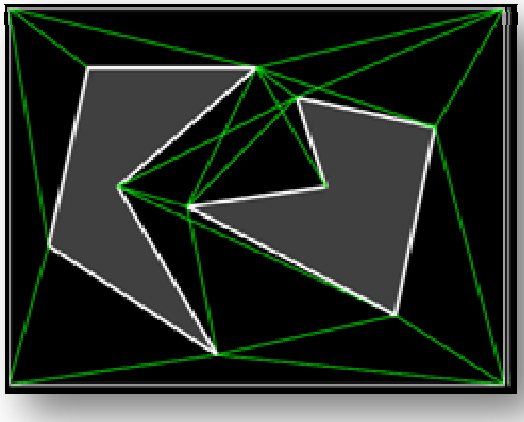
Metody v základním rozdělení dělíme na *deterministické* a *pravděpodobnostní*.

3.1.2.1 Deterministické metody

Graf viditelnosti

Graf viditelnosti je roadmapa definovaná jako dvojdimenzionální polygonální zobrazení prostoru.

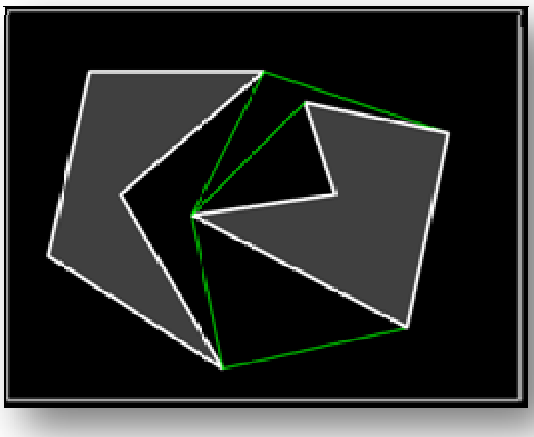
Vrcholy grafu tvoří startovní a cílový bod robota a dále vrcholy všech překážek v pracovním prostoru. Hrany grafu tvoří spojnice mezi vrcholy překážek. Tato spojnice ovšem nesmí protínat žádnou překážku. Hranami grafu jsou také samotné hrany překážek. Aplikace tohoto grafu je možná pouze u pracovního prostoru s polygonálními překážkami. U nepolygonních překážek (např. kružnice) se musí provést transformace na polygon, jehož tvar co nejvíce odpovídá dané překážce. Při tomto nahrazení ovšem narůstá počet vrcholů, což má vliv na dobu při sestavování grafu viditelnosti.



Obr. 18 Graf viditelnosti

Graf tečen

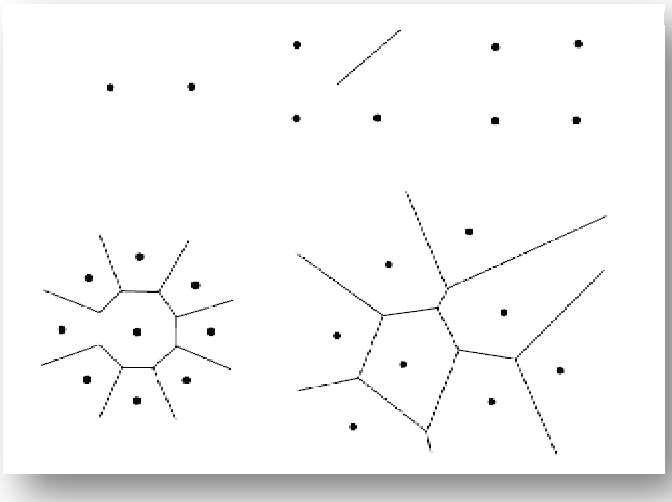
Tento graf je redukcí grafu viditelnosti. Vznikne odstraněním takových hran grafu, jež nejsou tečnami jednotlivých vrcholů. Touto redukcí docílíme rychlejšího prohledávání grafu.



Obr. 19 Redukovaný graf viditelnosti

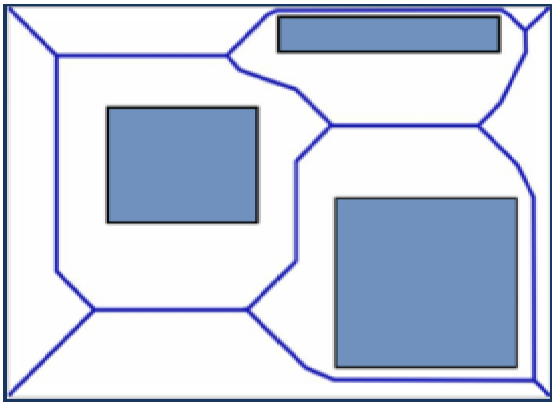
Voronoiův diagram

Jedná se o rovinnou geometrickou strukturu tvořenou body mající stejnou vzdálenost od dvou nebo více překážek. Hranami diagramu pak jsou spojnice těchto bodů. V případě robota zde hrany tvoří nekolizní cestu mezi překážkami. Graf z Voronoiova diagramu se vytvoří pomocí vrcholů představujících body, které mají stejnou vzdálenost od třech nebo více překážek a pomocí hran, které vzniknou spojením bodů, mající stejnou vzdálenost od dvou překážek. Do grafu se dále napojí počáteční a cílový vrchol. Následným řešením obdržíme nejkratší nalezenou cestu v grafu. [9]



Obr. 20 Ukázky Voronoiových diagramů [5]

Pro reálnější prostředí složené například z několika obdélníkových překážek v místnosti by mohl Voronoiův diagram vypadat nějak takto:



Obr. 21 Zjednodušený Voronoiův diagram [12]

3.1.2.2 Pravděpodobnostní metody

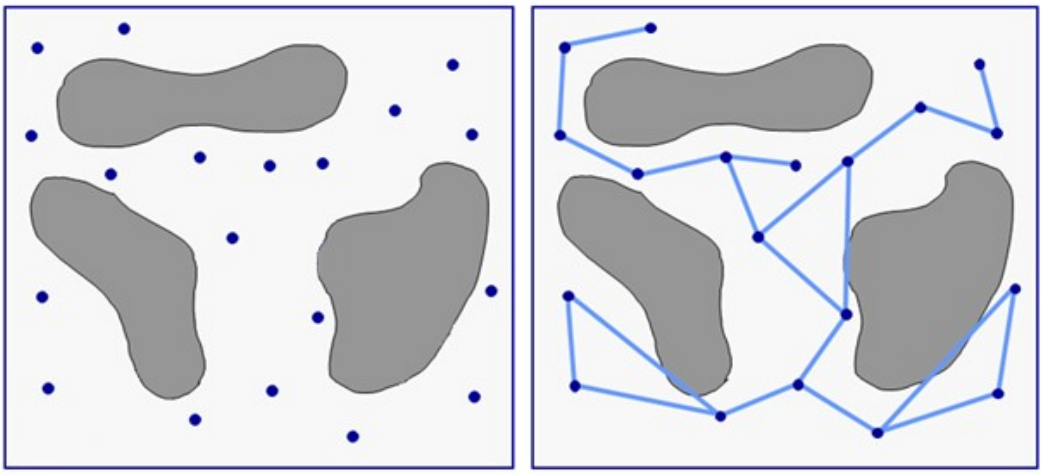
Pravděpodobnostní mapy cest (Probabilistic Roadmaps – PRM)

Samotný algoritmus se skládá ze dvou fází, a to *učící* a *dotazovací*.

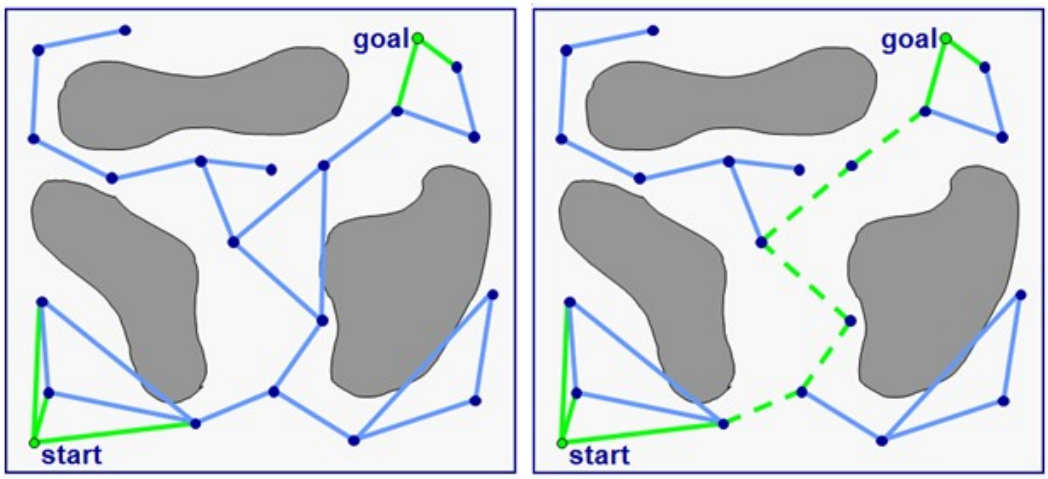
V první fázi se vytvoří pravděpodobnostní graf, jehož vrcholy představují náhodně vybrané nekolizní konfigurace robotu. Hranami grafu pak jsou nekolizní spojnice mezi těmito vrcholy. Tato fáze se opakuje, dokud není dostatečně popsán volný pracovní prostor.

Ve druhé fázi se do grafu napojí počáteční a cílový vrchol a propojí se možnými cestami. Následně se hledá cesta grafem. V případě nenalezení řešení je potřeba dostatečně pokrýt daný pracovní prostor a první fázi opakovat.

Pravděpodobnostní mapy jsou vhodné především u prostředí se statickými překážkami, kdy učící fázi stačí provést jen jednou a dotazovací fáze se několikrát opakuje.

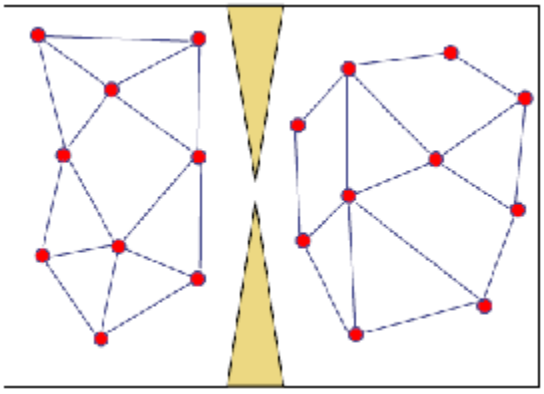


Obr. 22 Učící fáze



Obr. 23 Dotazovací fáze

Nevýhodou pravděpodobnostních map je, že v případě překážek s úzkými průchody dochází k rozpadu roadmapy na několik nepropojených segmentů (viz. obr.24). Je-li následně počáteční a cílový vrchol každý v jiném segmentu, pak mezi nimi nelze najít cestu. Proto se problém musí řešit dodatečnou metodou, která rozdělené segmenty opět propojí. [6]

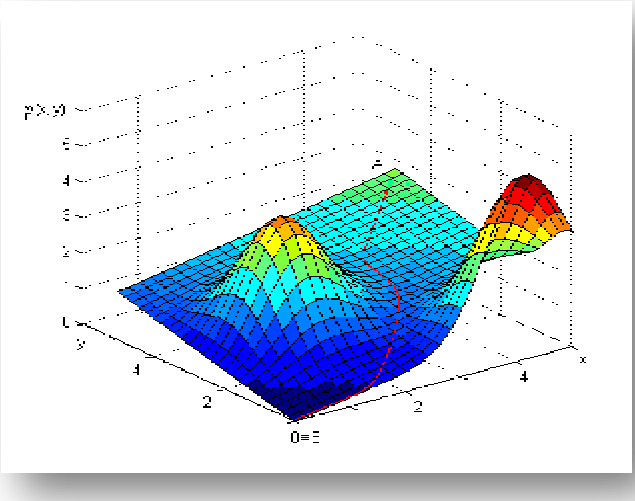


Obr. 24 Rozpad roadmapy na segmenty [6]

Alternativní metodu, která zachovává propojení všech vrcholů v souvislém stromu, představují pravděpodobnostní stromy. Jelikož v diplomové práci byla tato metoda vybrána pro implementaci, bude podrobněji popsána v další kapitole.

3.1.3 Potenciálová pole

Princip této metody spočívá v pokrytí pracovního prostoru potenciálovým polem definovaným potenciálovou funkcí $\varphi(x,y)$. Počáteční pozice robotu má vyšší potenciál než pozice cílová. Jelikož cílová pozice leží v globálním minimu, je dosaženo toho, že ve scéně působí homogenní přitažlivé pole se silovým vektorem směřujícím k tomuto cíli. Překážky mají vyšší potenciál než volný pracovní prostor. Jsou obklopeny odpudivým polem, jehož intenzita se vzdáleností od překážky klesá (lze si tedy představit jako kopce). Robot hledající cestu se pohybuje ve směru opačného gradientu potenciálové funkce.



Obr. 25 Potenciálové pole [6]

4 PRAVDĚPODOBNOSTNÍ STROMY *(RAPIDLY-EXPLORING DENSE TREES - RDT)*

Tuto rodinu metod představil poprvé v roce 1998 Steve M. LaValle. Jedná se o metody, které byly navrženy zejména pro úlohy neholonomního plánování, avšak bez problému je lze aplikovat u široké skupiny problematiky plánování cesty. Myšlenkou je postupné vytváření prohledávacího stromu, jenž se snaží rychle a rovnoměrně prohledat konfigurační prostor. Jestliže posloupnost tohoto vytváření probíhá náhodně, potom hovoříme o metodě RRT (*Rapidly-Exploring Random Trees*). Obecně je tato rodina stromů, ať už se jedná o deterministickou či náhodnou posloupnost, nazývána RDT, což má naznačovat, že se jedná o metodu rychlého prozkoumání prostoru pomocí hustého stromu.

RRT metoda je v podstatě stromová struktura, která se postupně rozrůstá vytvářením nových náhodně vybraných vrcholů (konfigurací robotu). Hrany stromu pak představují jednotlivé akce robotu. Počátek stromu je ve startovní konfiguraci q_{init}, který se následně v každé iteraci rozrůstá do náhodné nekolizní konfigurace q_{rand} o pevně zvolený krok ε (viz. obr. 26). [8]

Pro správnou funkci algoritmu je potřeba určit následující parametry [2]:

1) **Konfigurační prostor** C

2) **Počáteční a cílová konfiguraci**
   $q_{init} \in C$ a $q_{goal} \subset C$

3) **Detektor kolizí**
   Funkce $D: C \rightarrow \{True, False\}$, která určuje, zda konfigurace $q \notin C_{free}$.

4) **Množina vstupů** U
   Specifikuje množinu řízení či akcí, které mohou ovlivnit konfiguraci robota (např. maximální úhel natočení kol, směr jízdy, rychlost apod.).

5) **Inkrementální simulátor**
   Na základě řešení diferenciálních pohybových rovnic robota určuje Konfiguraci robota v čase t. Pro integraci se používá některá numerická metoda (např. Eulerova nebo Runge-Kutta).

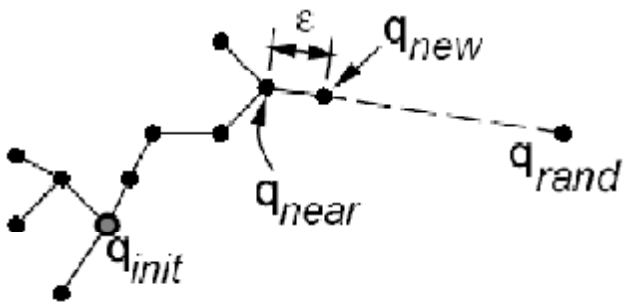
6) **Metrika** ρ
   Jedná se o funkci, určující vzdálenost dvou konfigurací robota.

4.1 Základní algoritmus RRT

[2] popisuje algoritmus takto:

function **BUILD_RRT**(q_{init})
1) T.init(q_{init});
2) **for** k = 1 to K **do**
3) $q_{rand} \leftarrow$RANDOM_CONF( );
4) EXTEND(T, q_{rand});
5) return T.

function **EXTEND**(T, q_{rand})
1) $q_{near} \leftarrow$NEAREST_NEIGHBOR(q_{rand}, T);
2) **if** NEW_CONF(q_{rand}, q_{near}, q_{new}, u_{new}) **then**
3) T.add_vertex(q_{new});
4) T.add_edge(q_{near}, q_{new}, u_{new});
5) **if** $q_{new} = q_{rand}$ **then**
6) return Reached;
7) **else**
8) return Advanced;
9) return Trapped;



Obr. 26 Princip RRT [2]

Popis základního algoritmu RRT

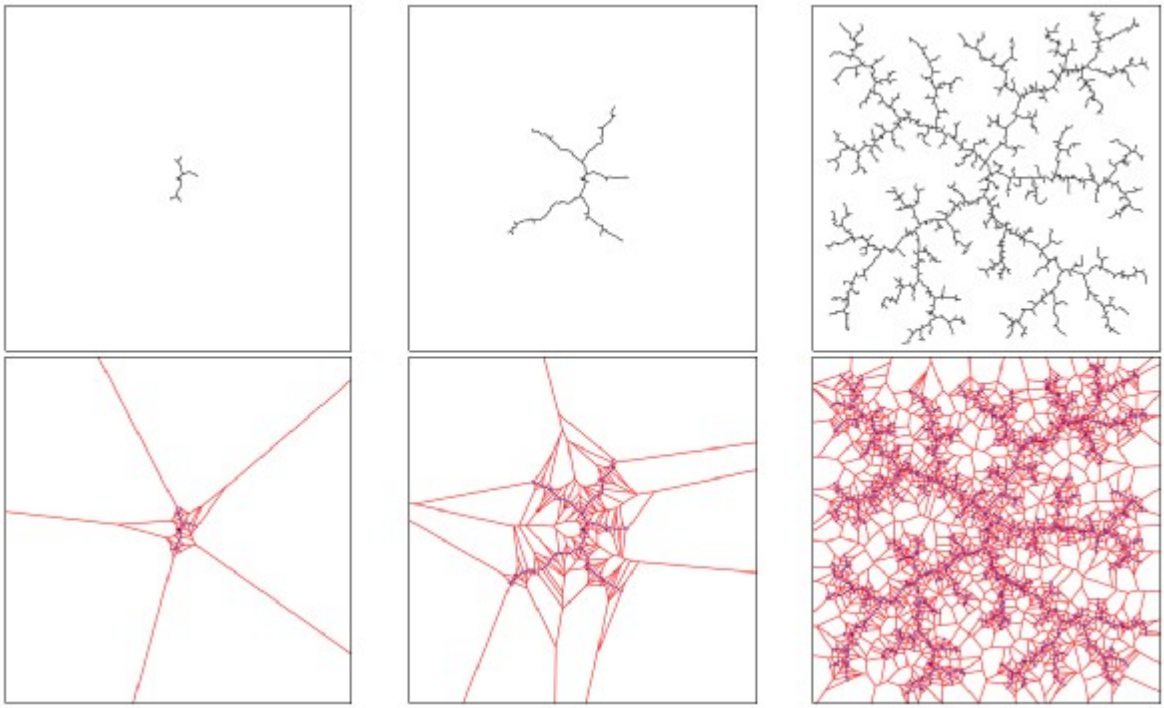
V prvním kroku se určí kořen stromu T, čímž je počáteční konfigurace q_{init}. V následujícím cyklu, kde K je počet iterací, se volají funkce RANDOM_CONF a EXTEND. Funkce RANDOM_CONF vygeneruje náhodnou konfiguraci q_{rand}, nacházející se v konfiguračním prostoru. Funkce EXTEND pak zajistí růst stromu směrem ke q_{rand}. Nejprve se volá funkce NEAREST_NEIGHBOR, která vrací nejbližší konfiguraci q_{near} k náhodně vybrané konfiguraci q_{rand}. Zde aplikujeme metriku, pro výpočet vzdálenosti mezi dvěma konfiguracemi. Dále následuje funkce NEW_CONF, která vrací nově spočtenou konfiguraci q_{new}, vypočítanou pomocí inkrementálního simulátoru. Vstupem simulátoru je konfigurace q_{near} , na kterou aplikuje vstup $u \in U$ po určitý časový inkrement Δt (časový inkrement může být jiný než v integrační metodě). Vstup u může být vybrán náhodně z množiny U, nebo se zkusí všechny možné vstupy a z nich se vybere ten, který je nejblíže ke q_{rand}. Jestliže je množina U nekonečná, pak se použije aproximace. Po spočtení nové konfigurace q_{new} se ještě musí otestovat, zda vyhovuje všem omezením (tj. zda neleží v kolizní oblasti C_{obst}). Toto má na starosti detektor kolizí. Pokud nový uzel omezení nesplňuje, je jednoduše vyřazen a proces rozrůstání RRT pokračuje. Pokud je ovšem konfigurace nekolizní, funkce NEW_CONF vrátí *True* a q_{new} se přidá do stromu

jako nový vrchol. Současně se přidá i hrana s ohodnocením u_{new}, vzniklá spojením vrcholu q_{near} s vrcholem q_{new}. Jestliže nový vrchol q_{new} je totožný s q_{rand} (porovnává se vzdálenost mezi těmito konfiguracemi a zvoleným parametrem δ), funkce EXTEND jako *result* vrací Reached (dosaženo), v opačném případě Advanced (rozšířeno). Pokud se funkci NEW_CONF nepodařilo nalézt novou konfiguraci, vrací Trapped (uvězněn).

Podle počtu iterací *K* se celý cyklus opakuje čímž se RRT struktura rozrůstá a vyplňuje konfigurační prostor. Čím vyšší počet iterací, tím hustěji je prostor pokryt, avšak nevýhodou jsou samozřejmě vyšší výpočetní a paměťové nároky.

RRT algoritmus je za obecných podmínek pravděpodobnostně kompletní, což je slabší varianta kompletního plánovacího algoritmu. Tedy pokud existuje cesta vedoucí k cíli, pravděpodobnost, že bude nalezena, konverguje k jedné, jak se počet iterací blíží k nekonečnu. Jinými slovy, nalezení cesty závisí na počtu vrcholů stromu. [8]

Jelikož prohledávací strom má silnou tendenci růst směrem k dosud neprozkoumaným oblastem, činí jej ideálně vhodným nástrojem pro řešení široké škály praktických problémů plánování cesty. Tato tendence je způsobena velikostí Voronoiových oblastí pro jednotlivé vrcholy stromu. Největší Voronoivy oblasti patří vrcholům, které představují listy stromu. Algoritmus RRT vybere pro růst stromu ten vrchol, který je nejblíže k náhodně vygenerovanému vrcholu. V případě rovnoměrného rozložení pravděpodobnosti je pravděpodobnost, že tento náhodně vygenerovaný vrchol bude ležet uvnitř nějaké Voronoiovy oblasti přímo úměrná velikosti této oblasti. Tendenci stromu růst směrem k ještě neprozkoumaným oblastem prostoru můžeme vidět na obr. 27. [6]



Obr. 27 Voronoivy oblasti odpovidajici vrcholům RRT stromu [2]

Na druhou stranu, tím, že se strom rozrůstá na všechny strany víceméně rovnoměrně, jeho hlavním problémem je příliš pomalá konvergence k cíli, tudíž se v praxi základní algoritmus RRT příliš nepoužívá. Z tohoto důvodu byly navrženy různé upravené verze.

4.2 Plánovače RRT

[2,4,6] rozšiřuje základní algoritmus takto:

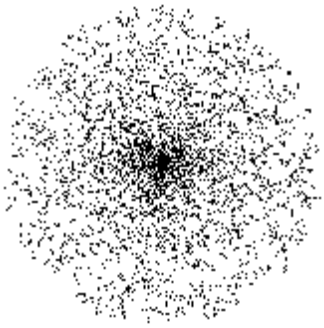
Plánovač se spádem k cíli (RRT-GoalBias)

Zrychlení konvergence stromu k cíli je dosaženo úpravou funkce RANDOM_CONF v základním algoritmu RRT. Tato úprava spočívá v definování malé hodnoty pravděpodobnosti (např. 0,1), s kterou bude tato funkce vracet cílovou konfiguraci q_{goal} (namísto náhodné konfigurace obdržené použitím funkce *random*). Avšak pravděpodobnost nesmí být příliš velká, jinak může dojít k uváznutí stromu v lokálním minimu.

Plánovač se zaměřením na cíl (RRT-GoalZoom)

Tato varianta krom zajištění zrychlení konvergence k cíli zároveň zmenšuje riziko uváznutí v lokálním minimu, oproti předešlé variantě plánovače. Úprava spočívá opět v modifikaci funkce RANDOM_CONF a to tak, že funkce sice vrací náhodné konfigurace pomocí funkce *random*, ale ty jsou generovány z určitého okolí kolem cílové konfigurace q_{goal}. To znamená, že je nastavena maximální povolená vzdálenost od cílové konfigurace q_{goal}, ve které se náhodná konfigurace q_{rand} může nacházet. Na začátku je toto okolí velké a jak se blíží strom k cíli, tak se velikost okolí zmenšuje. Stále ovšem hrozí uvíznutí v lokálním minimu. Všeobecně se zdá nejlepší nahradit funkci RANDOM_CONF funkcí generující náhodné konfigurace robota s nerovnoměrnou hustotou pravděpodobnosti, která roste směrem k cíli.

Obrázek 28 znázorňuje příklad RRT stromu zkonstruovaného pomocí funkce, jenž generuje stavy s rovnoměrnou hustotou pravděpodobnosti, čímž se strom utváří do kruhového prstence.



Obr. 28 rovnoměrný RRT strom [2]

Dvousměrný plánovač (RRT-Bidirectional)

Tento plánovač při prohledávání konfiguračního prostoru využívá růstu dvou stromů. První strom má kořen v q_{init} a druhý v q_{goal}. Řešení je nalezeno, pokud se oba stromy setkají. Algoritmus dvousměrného RRT plánovače vypadá takto:

function **RRT_BIDIRECTIONAL** (q_{init}, q_{goal})
1) T_a.init(q_{init}); T_b.init(q_{goal});
2) **for** k = 1 to K **do**
3) $q_{rand} \leftarrow$RANDOM_CONF();
4) **if not** (EXTEND(T_a, q_{rand}) = Traped) **then**
5) **if** (EXTEND(T_b, q_{new}) = Reached) **then**
6) return PATH(T_a, T_b);
7) SWAP(T_a, T_b);
8) return Failure.

Funkce RRT_BIDIRECTIONAL dělí výpočetní čas mezi dva procesy. První část výpočtu tvoří prohledávání konfiguračního prostoru a druhou část snaha o spojení obou stromů. Oba stromy T_a a T_b jsou po celou dobu zachovávány, dokud se nenajde řešení a nespojí se. V každé iteraci se jeden strom rozroste o nový vrchol q_{new} směrem k náhodné konfiguraci q_{rand} a druhý strom se pokouší tento vrchol napojit se svým nejbližším vrcholem. Potom se oba stromy prohodí a výpočet se opakuje. Cesta je nalezena, když dojde ke spojení obou stromů.

Po řadě experimentů, se ukázalo, že v případě potřeby, je použití dvousměrného přístupu mnohem účinnější než při využití jediného RRT stromu. Avšak pro aplikaci u neholonomního plánování je napojení stromů na sebe mnohem obtížnější.

Ostatní vylepšení

Další možností jak zajistit rychlejší konvergenci k cíli je dynamicky měnit velikost kroku ε použitého pro rozrůstání RRT stromu. Pomocí zvolené metriky se zjistí vzdálenost aktuální konfigurace robota od nejbližší překážky. Pokud je robot ve větší vzdálenosti od překážky, může být krok ε větší, než kdyby byl blízko překážky.

Úpravou zrychlující konvergenci stromu k cíli může být také změna funkce EXTEND na CONNECT . Princip spočívá v tom, že místo rozšiřování RRT inkrementálně po krocích, může být funkce EXTEND volána opakovaně, dokud strom nedosáhne náhodné konfigurace q_{rand} nebo nedojde ke kolizi s překážkou. Algoritmus funkce CONNECT vypadá následovně:

function **CONNECT** (T, q_{rand})

1) **repeat**
2) $S \leftarrow$EXTEND(T, q_{rand});
3) **until not** (S = Advanced);
4) return S;

Výhodou je možnost vytváření delších cest za pomoci jediného volání funkce NEAREST_NEIGHBOR. Po experimentování se ukázalo, že funkce CONNECT dosahuje nejlepších výsledků u aplikací holonomního plánování a funkce EXTEND je vhodnější pro neholonomní plánování.

5 INTELIGENTNÍ GLOBÁLNÍ PLÁNOVAČ S PŘEPLÁNOVÁNÍM *(THE INTELLIGENT GLOBAL PATH PLANNER WITH REPLANNING)*

Další plánovací metodu, kterou si představíme z důvodu jejího vybrání pro implementaci, se nazývá IGPPR (*Intelligent Global Path Planner with Replannig*). V roce 1998 ji poprvé představil Leszek Podsedkowski v [13]. Jedná se o metodu vyvinutou pro roboty typu Auto s tří-dimenzionálním konfiguračním prostorem, ale může být použita i pro jiné typy robotů s neholonomním omezením. Metoda je obzvláště vhodná u aplikací s potřebou plánování cesty on-line v částečně známém prostředí.

Metoda je založena na algoritmu A* prohledávání grafu s vrcholy umístěnými v diskrétním konfiguračním prostoru a je vybavena procedurou přeplánování cesty, což je velmi užitečné v částečně známých pracovních prostorech.

Tuto metodu charakterizují dvě základní výhody. První z nich je, že vygenerovaná cesta je vždy optimální pro daný stav znalosti svého prostředí. Další výhodou této metody je vysoká rychlost procesu plánování cesty. Obzvláště rychlost přeplánování je velmi důležitá, pokud chceme aplikovat on-line řízení bez zastavení robota při změně pracovního prostoru. Časy potřebné pro plánování cesty jsou kratší či srovnatelné s lidskou reakcí. Metoda je obzvláště výhodná u velmi složitých pracovních prostorů (jako jsou např. bludiště), kde jiné metody nefungují efektivně.

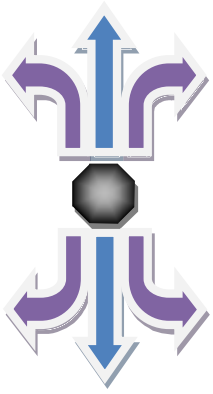
Tato kapitola je zpracována na základě [3].

5.1 Možnosti řešení

Metoda IGPPR vychází z metod vytvoření a prohledávání grafu. Mnoho z nich, jako např. graf viditelnosti či Voronoiho diagram, nejsou vhodné pro roboty s neholonomními omezeními. Grafy, které lze uvažovat při hledání cesty pro takové roboty pomocí IGPPR jsou: graf, v němž je robot reprezentován jako *bod s omezeným poloměrem zatáčení* nebo tzv. *grafy se základními skoky*.

První metoda předpokládá, že překážky jsou polygonální. Tento předpoklad snižuje použitelnost metody, protože reálná prostředí mají často bitmapovou reprezentaci. Během prohledávání grafu se algoritmus pokouší spojit každé dva vrcholy a pak kontroluje kolizi vytvořené hrany s polygonálními překážkami.

Druhá metoda rozděluje konfigurační prostor robota do matice buněk a umisťuje vrcholy grafu do každé buňky. Hrany grafu jsou vytvořeny základními skoky. Existuje šest takových skoků: rovně dopředu, dopředu vlevo, dopředu vpravo, rovně dozadu, dozadu vlevo a dozadu vpravo.



Obr. 29 Možné směry pohybu robota typu Auto

Definice hran a vrcholů grafu je pouze prvním krokem plánování cesty. Druhým krokem je prohledání grafu, vedoucí k nalezení cesty. Existuje mnoho metod prohledávání grafu. Např. hledání do hloubky (*depth-first search*), hledání do šířky (*breadth-first search*) - Dijkstrův algoritmus, či A* algoritmus s různými heuristicky ohodnocujícími funkcemi. Tyto algoritmy iteračně prozkoumávají graf *G*, začínající v počátečním uzlu N_{root} a končící dosažením cílového uzlu N_{fruit}. Obvykle uzly N_{root} a N_{fruit} nahrazujeme uzly N_{start} a N_{goal}.

5.2 Princip metody

Jelikož jsou prostředí obvykle popsána mřížkovými mapami, pro generování grafu je použito procedury základních skoků. Pozice vrcholů nejsou centrovány do středů mřížek (buněk) konfiguračního prostoru. To způsobuje proměnlivou strukturu grafu, ale velikost buněk může být srovnatelná s délkou skoku.

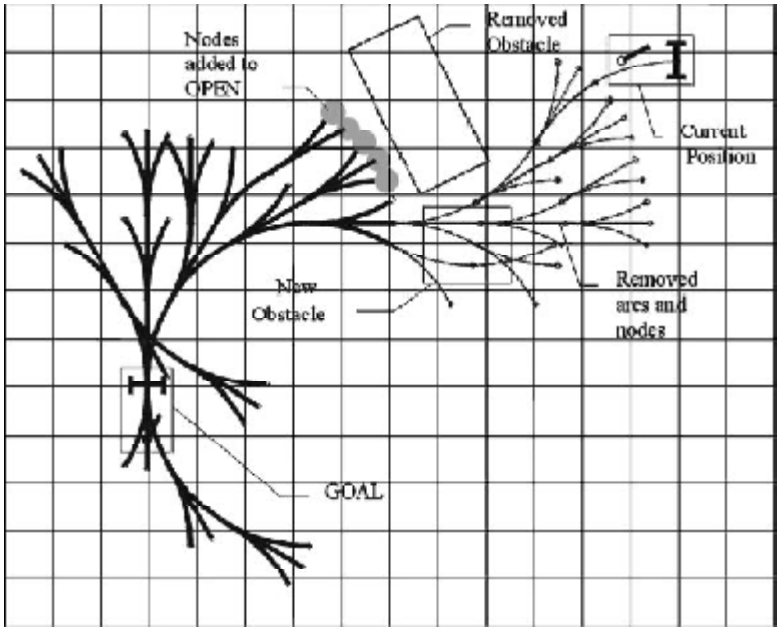
První změna v modifikaci algoritmu A* se týká směru prohledávání grafu. To umožňuje využít část grafu v proceduře přeplánování. Dále byly analyzovány a porovnány různé heuristické funkce ohodnocení.

5.2.1 Směr prohledávání grafu

Pokud je potřeba nalézt cestu v prostředí pouze jednou, směr hledání není důležitý. Uzel N_{root} může být ve startovní pozici, a uzel N_{fruit} v cílové. Zcela jiná situace nastane, kdy robot musí přeplánovat cestu. To se stane, když robot detekuje změny v pracovním prostoru nebo když vyjede z dráhy na cestě k cíli (způsobené např. prokluzem kol nebo naměření chyby). V takové situaci, kdy se kořen grafu nachází v počáteční pozici, musí být celý graf přestavěn v průběhu přeplánování. Všechny dříve uložené údaje, se stávají neplatné. Naopak, když kořen grafu je v cílové pozici, velká část grafu je stále platná. Tohoto je využito v algoritmu IGPPR.

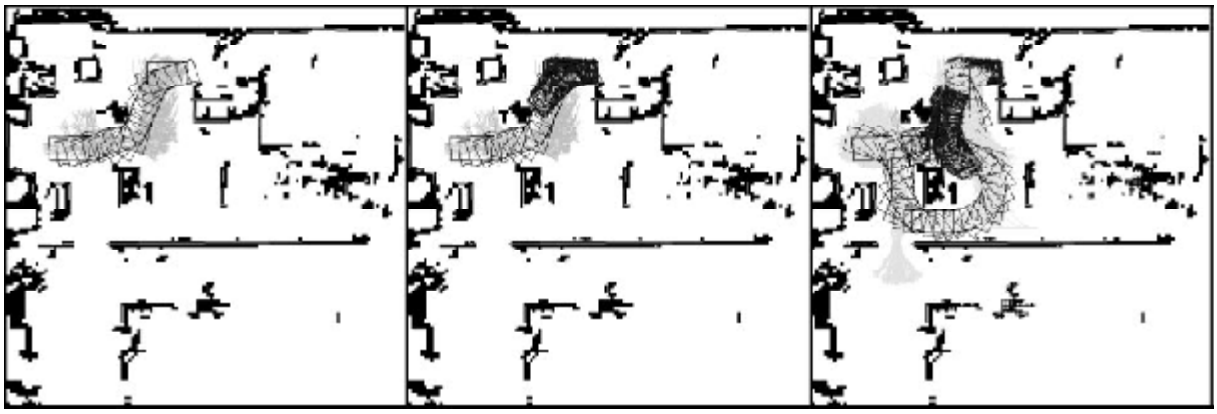
Během procedury přeplánování musíme obnovit jen část grafu. Před tím musí být odstraněny neplatné části grafu. Obecně platí, že v situaci, kdy robot detekuje na cestě nové překážky, algoritmus IGPPR odstraní z paměti všechny uzly kolidující s překážkou a všechny jejich následníky, a vloží všechny předchůdce odstraněných uzlů do seznamu OPEN. Z tohoto seznamu se pak následně vybírají uzly pro expandování prohledávaného grafu.

Jestliže robot detekuje absenci dříve zapamatovaných překážek, vloží do seznamu OPEN všechny uzly, které mohou mít následníky v místě těchto překážek (viz obr.30). Po těchto operacích, algoritmus IGPPR pokračuje v generování grafu základními skoky a v prohledávání grafu stejným způsobem jako v algoritmu A*. K zajištění této změny, je nutné, aby byly všichni možní předchůdci i následníci uzlu grafu zachovávány (nejen předchůdci s nejnižším ohodnocením jako v klasické A* metodě).



Obr. 30 Modifikace grafu během procedury přeplánování [3]

Na obrázku 31 můžeme pozorovat přeplánování cesty, způsobené uzavřením průchodu na předem spočtené cestě.



Obr. 31 Změna cesty při uzavření průjezdu [3]

Na obrázku 32 můžeme pozorovat přeplánování cesty, způsobené otevřením průchodu, které přináší nové možnosti nalezení optimální cesty.

Obr. 32 Změna cesty při otevření průjezdu [3]

5.2.2 Urychlení prohledávání grafu – heuristická funkce ohodnocení

Velmi důležitým parametrem popisujícím kvalitu algoritmu je jeho rychlost. V rámci on-line řízení robota se předpokládá, že doba hledání musí být kratší než několik sekund. Je-li doba delší, pak se algoritmus nehodí pro real-time aplikace. Složitost vyhledávání pro A* algoritmus může být odhadnuta jako $O(n_c \log n_c)$, kde n_c je maximální počet vrcholů nebo počet iterací.

A* algoritmus přiřazuje každému prozkoumanému vrcholu N ohodnocení pomocí funkce:
$$f(N) = g(N) + h(N). \qquad (2)$$
Tato funkce je odhadem minimálních nákladů na cestu spojující N_{root} s N_{fruit} v grafu G a procházející skrz vrchol N.

$g(N)$ představuje cenu (ohodnocení) cesty mezi vrcholy N_{root} a N. Ohodnocení cesty je určeno hlavně délkou cesty, ale může zahrnovat i další aspekty, jako např. obtížnost a riziko cesty.

$h(N)$ představuje heuristickou aproximaci ohodnocení $h^*(N)$ minimálních nákladů na cestu z vrcholu N do N_{fruit}.

Vzhledem k přípustné heuristické funkci můžeme předpokládat, že ohodnocení f každého vrcholu není vždy menší než ohodnocení jeho předchůdce. Počet iterací je roven počtu vrcholů, z kterých jsou vytvořeni následníci. Protože první vrchol vybraný z OPEN je N_{root} a poslední N_{fruit} je možné zapsat:
$$f(N_{root}) < f(N) < f(N_{fruit}). \qquad (3)$$
Protože $f(N_{root}) = h(N_{root})$ a $f(N_{fruit}) = g(N_{fruit}) = h^*(N_{root})$, obdržíme
$$h(N_{root}) < f(N) < h^*(N_{root}). \qquad (4)$$

Jestliže hodnota $h(N_{root})$ je blízká $h^*(N_{root})$, pak menší počet uzlů má ohodnocení f, splňující rovnici (4) a počet iterací algoritmu klesá (avšak toto neplatí obecně).

Analyzujme dva typy heuristických funkcí:

1. Ohodnocení $h(N)$ se rovná euklidovské vzdálenosti mezi vrcholy N a N_{fruit}.
2. Ohodnocení $h(N)$ se vypočítá jako nejkratší vzdálenost k vrcholu N_{fruit} procházející kolem překážek.

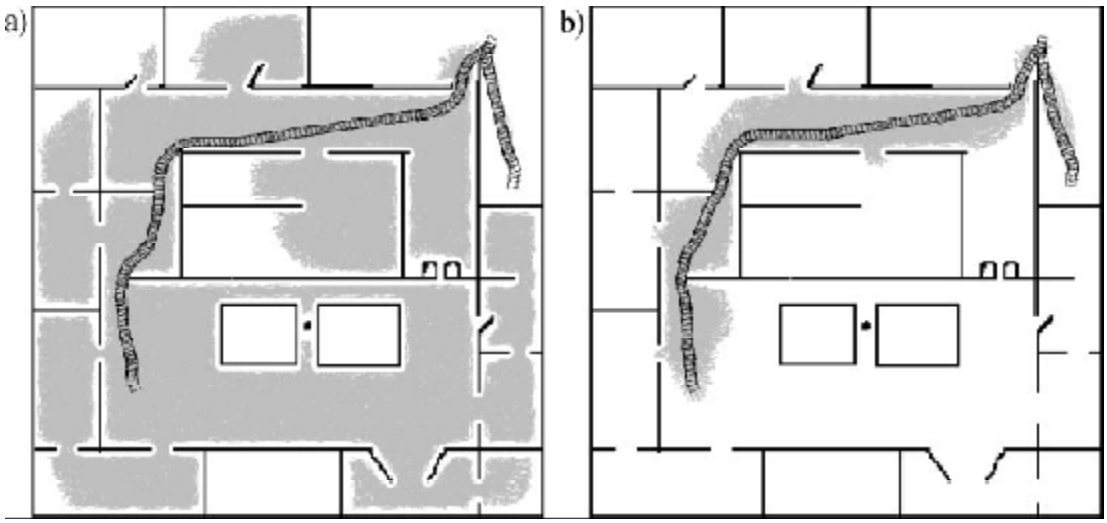
Druhý typ metody vyžaduje předem spočtenou mapu ohodnocení (*heuristic cost map*). Ta je vytvořena s využitím 4, 8 nebo 16 směrové metody šíření, kterou si popíšeme dále. Použijeme-li druhou metodu, tak během prvního kroku (čímž je vytvoření heuristické ohodnocující mapy) řešíme problém plánování cesty bez zohlednění neholonomních omezení. Tato mapa nám pomáhá v druhém kroku (prohledávání grafu) tím, že řídí prohledávání a prochází prostor tam, kde je možné najít cestu. Samozřejmě změna N_{fruit} (N_{root}) během pohybu vyžaduje přestavbu mapy ohodnocení.

U prvního typu heuristické funkce závisí ohodnocení prohledávané oblasti na (4). Vzhledem k tomu, že $g(N)$ je větší než euklidovská vzdálenost $vzd(N, N_{root})$ a $h(N)$ se rovná vzdálenosti $vzd(N, N_{fruit})$, můžeme upravit pravou stranu vztahu (4) na
$$vzd(N, N_{root}) + vzd(N, N_{fruit}) < h^*(N_{root}) \qquad (5)$$

Tuto nerovnici můžeme reprezentovat pomocí elipsy s ohnisky v N_{root} a N_{fruit} a hlavní osou délky $h^*(N_{root})$. Počet uzlů (a iterací) n_c , je omezen hodnotou n_m, která odpovídá povrchu elipsy.

U druhého typu funkce a pracovního prostoru bez překážek má prohledávaná oblast také tvar elipsy. Pro pracovní prostor s překážkami můžeme použít následující metodu: deformujeme-li pracovní prostor takovým způsobem, že nejkratší vzdálenost mezi N_{root} a N_{fruit} procházející kolem překážek bude přímá (vzdálenosti k překážkám jsou zachovávány), situace se stane obdobnou k předchozí. Při použití druhého typu heuristické funkce je počet prohledaných uzlů výrazně nižší. V závislosti na poloze překážek a konfigurací N_{root} a N_{fruit} může rozdíl dosáhnout až 20tinásobku. V některých situacích nemusí být rozdíl tak vysoký. Pokud jsou konfigurace N_{root} a N_{fruit} umístěny tak, že nalezená cesta prochází téměř celým pracovním prostorem, pak u obou typů ohodnocující funkce budou prohledány téměř všechny konfigurace.

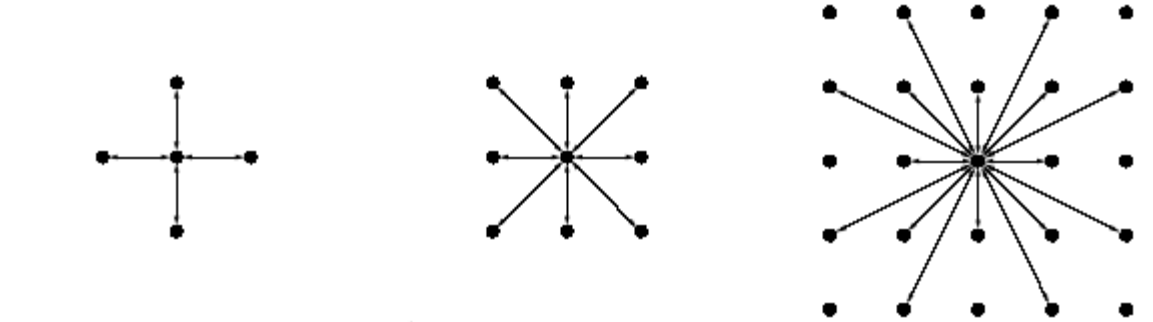
Příklad simulace plánování cesty za použití obou metod heuristických funkcí je prezentován na obrázku 33. Nalezená cesta je vyznačena černě a prohledaná oblast šedivě.



Obr. 33 Srovnání prohledané oblasti pro první (a) a druhý (b) typ heuristické funkce [3]

5.2.3 Šestnácti-směrová metoda šíření

Princip spočívá ve vytvoření rastrové mapy CB_{min} z pracovního prostoru s překážkami navýšenými o rozměr r_{min} (poloměr opsané kružnice robota). Zvětšením překážek zamezíme procházení robota příliš blízko překážek. Šestnácti-směrová metoda šíření používá matici o velikosti CB_{min} přiřazující každému prvku směr šíření a vzdálenost (heuristické ohodnocení) ke konfiguraci N_{fruit}. Jelikož každá buňka v mapě má osm sousedů, existuje osm základních směrů šíření. Ohodnocení pro sousedy v kolmém směru je navýšeno o jednotku, diagonálně o $\sqrt{2}$. Změnou směru šíření (známo díky zapamatování si šíření směru) můžeme obdržet dalších osm šiřitelných směrů o vzdálenosti $\sqrt{5}$ (viz. obr. 34c). V takových situacích by mělo být ohodnocení zvýšeno o $(\sqrt{5}-\sqrt{2})$ či $(\sqrt{5}-1)$. Celkově máme tedy 16 možných směrů šíření. Jelikož v praxi počítačový software pracuje mnohem rychleji při zpracování celých čísel, nahrazujeme čísla $1, \sqrt{2}, \sqrt{5}$ čísly 17, 24 a 38. Na obr.34 je porovnání odhadu vzdálenosti pro 4 (a), 8 (b) a 16 (c) směrovou metodou šíření. Maximální chyby u těchto metod jsou 41, 8 a 3%.



a)

136	119	102	85	68	85	102	119	136
119	102	85	68	51	68	85	102	119
102	85	68	51	34	51	68	85	102
85	68	51	34	17	34	51	68	85
68	51	34	17	0	17	34	51	68
85	68	51	34	17	34	51	68	85
102	85	68	51	34	51	68	85	102
119	102	85	68	51	68	85	102	119
136	119	102	85	68	85	102	119	136

b)

96	89	82	75	68	75	82	89	96
89	72	65	58	51	58	65	72	89
82	65	48	41	34	41	48	65	82
75	58	41	24	17	24	41	58	75
68	51	34	17	0	17	34	51	68
75	58	41	24	17	24	41	58	75
82	65	48	41	34	41	48	65	82
89	72	65	58	51	58	65	72	89
96	89	82	75	68	75	82	89	96

c)

96	86	76	72	68	72	76	86	96
86	72	62	55	51	55	62	72	86
76	62	48	38	34	38	48	62	76
72	55	38	24	17	24	38	55	72
68	51	34	17	0	17	34	51	68
72	55	38	24	17	24	38	55	72
76	62	48	38	34	38	48	62	76
86	72	62	55	51	55	62	72	86
96	86	76	72	68	72	76	86	96

Obr. 34 Porovnání 4, 8 a 16 směrové metody šíření [3]

6 ZAČLENĚNÍ DISKRETIZACE STAVOVÉHO PROSTORU (INCORPORATING STATE SPACE DISCRETIZATION)

Poslední metodou, vybranou k implementování do plánovače, je metoda ISSD (*Incorporating State Space Discretization*). Tato metoda, uvedená v [1], rozděluje stavový prostor robotu do malých buněk, ve kterých je povolen maximálně jeden vrchol prohledávaného grafu. To brání prohledávacímu algoritmu před neustálým chodem, jestliže má prohledávaný graf nekonečný počet vrcholů v některé ohraničené oblasti. K prohledávání grafu využívá jakéhokoli systematického prohledávacího algoritmu.

6.1 Princip metody

6.1.1 Rozklad stavového prostoru do buněk

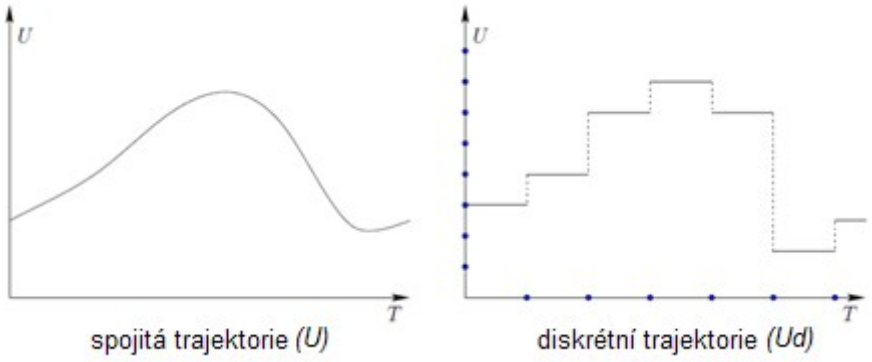
Na počátku se stavový prostor *X* rozloží do kolekce (matice) buněk *D* o rozměru *n*. To znamená, že při definování *D* ignorujeme detekci kolizí a ostatní omezení na stavovém prostoru.

Nejčastější rozklad stavového prostoru do buněk se provádí pomocí kubického rozdělení. Dalším možným způsobem může být například rozklad prostoru do Voronoiových oblastí. V případě kubického rozdělení, patří body na společných hranicích buněk vždy pouze jedné sousední buňce.

6.1.2 Prohledávání

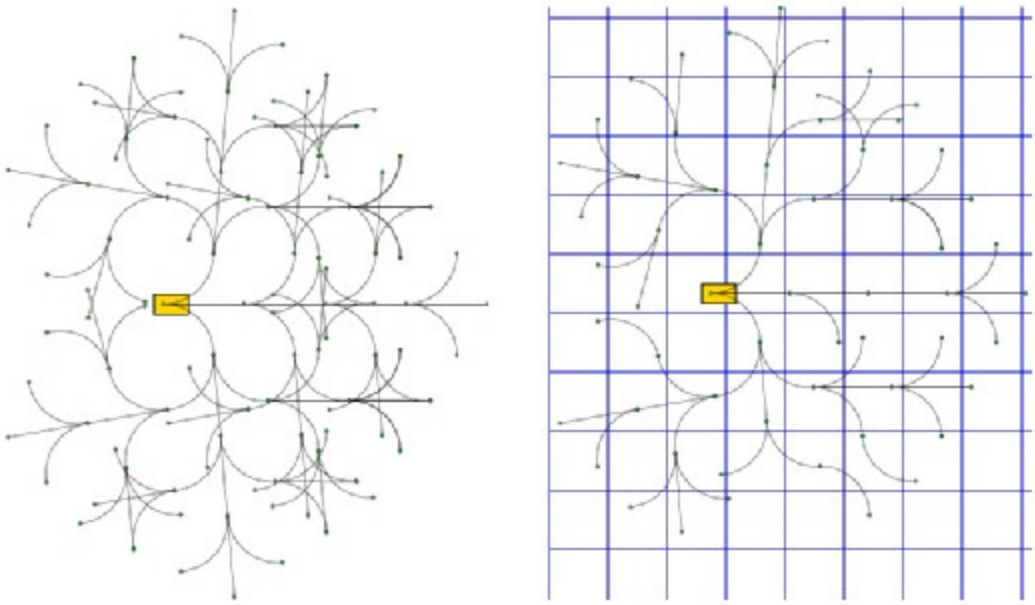
K prohledávání grafu z počátečního stavu x_I je využito jakéhokoli systematického prohledávacího algoritmu. Předpokladem je, že systém byl nějakým způsobem zdiskretizován. Nejčastěji se při diskretizaci prostředí používá pevně zvolený krok robotu Δt a konečná množina akcí U_d. Na obrázku 35 je znázorněn diskrétní model, který je charakterizován třemi aspekty:

1. Čas *T* je rozdělen do intervalů délky Δt (umožňuje krokování).
2. Konečná podmnožina U_d akčního prostoru U ($U_d \subset U$).
3. Akce $u(t) \in U_d$ musí zůstat konstantní během každého časového intervalu.



Obr. 35 Znázornění diskretizace vstupní množiny akcí [1]

U algoritmu ISSD je důležité sledovat, které buňky již byly navštíveny. Buňka *D* se nazývá ***„navštívená“*** (*visited*) pokud prohledávací graf obsahuje vrchol umístěný v buňce *D*, jinak buňku označujeme jako ***„nenavštívenou“*** (*unvisited*). Zpočátku je jako navštívená buňka označena pouze ta, která obsahuje počáteční stav x_I. Všechny ostatní buňky jsou inicializovány jako nenavštívené. Tohoto značení je využito k prořezání grafu během vyhledávání. Na obrázku 36a jsou znázorněny první čtyři etapy dosažitelného grafu pro tzv. *Dubinsovo* auto, které se může pohybovat pouze vpřed. Na vedlejším obrázku pak vidíme jeden z možných prohledávacích grafů, který obdržíme prořezáním větví původního grafu pomocí přiřazení nanejvýše jednoho vrcholu každé buňce.



(a) (b)

Obr. 36 Dosažitelný graf pro Dubinsovo auto [1]

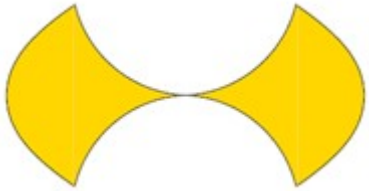
6.2 Základní algoritmus [1]

function **CELL_BASED_SEARCH** (x_I, x_G)

1) Q.insert (x_I);
2) Q.init (x_I);
3) **while** $Q \neq \emptyset$ **and** x_G is *unvisited*
4) $x_{cur} \rightarrow Q$.pop ( );
5) **for each** ($\tilde{u}_t$,x) ∈ REACHED (x_{cur})
6) **if** x is *unvisited*
7) Q.insert (x);
8) G.add_vertex (x);
9) G.add_edge ($\tilde{u}_t$);
10) Mark cell that contains x as *visited*;
11) Return *G*;

Nechť Q představuje prioritní frontu, v níž prvky jsou vrcholy prohledávaného grafu. Pokud je vyžadována optimalizace nákladů vynaložených na cestu, může být fronta Q seřazena podle funkčních ohodnocení. Tyto náklady mohou být stanoveny různými způsoby. Mohou představovat čas (počet Δt kroků) či je možné počítat, kolikrát se změnila akce.

Do grafu jsou uloženy pouze ty vrcholy, které leží v buňce označené jako nenavštívená a zároveň neobsazené překážkou. Současně se vrcholy umístí do fronty Q. Po tomto kroku se buňka označí jako navštívená. Funkce REACHED generuje množinu segmentů trajektorie nekolidující s překážkami. V rámci diskrétního modelu to znamená uplatnění každého $u \in U_d$ na x_{cur} v čase Δt a navrácení nových stavů x, u kterých nedošlo k porušení omezení kladených na robota (včetně zamezení kolize).



Obr. 37 Dosažitelná množina trajektorií pro jednoduché auto (časově-omezená) [1]

Problém může nastat, pokud je velikost buňky ve srovnání s cílovou oblastí X_G příliš velká. Pokud cílová oblast dostatečně zahrnuje celé buňky, pak se tomuto problému vyhneme. Je-li cíl definován pouze jediným bodem x_G, algoritmus musí akceptovat určitou toleranci.

6.3 Zachování buněk

Existuje několik alternativ pro uchovávání buněk. Hlavní operací, která musí být prováděna efektivně je tzv. lokalizace bodu (*point location*). Tato funkce nám určuje, která buňka obsahuje daný bod (stav). Pro uchování buněk můžeme využít n-rozměrného pole. Kontrola kolizí s překážkami pak může být provedena v předstihu. Jakákoli buňka, obsahující alespoň jeden bod z X_{obs} je označena jako obsazená. To nám umožňuje vyhnout se buňkám, které obsahují kolizní konfigurace, aniž bychom museli následně volat modul detekce kolizí.

Alternativou je použití hash tabulky k udržování kolekce buněk, které jsou označeny jako navštívené. To může být obzvláště cenné v případě, kdy není důležitá optimalizace a jestliže se očekává, že řešení bude nalezeno před dosažením většiny buněk.

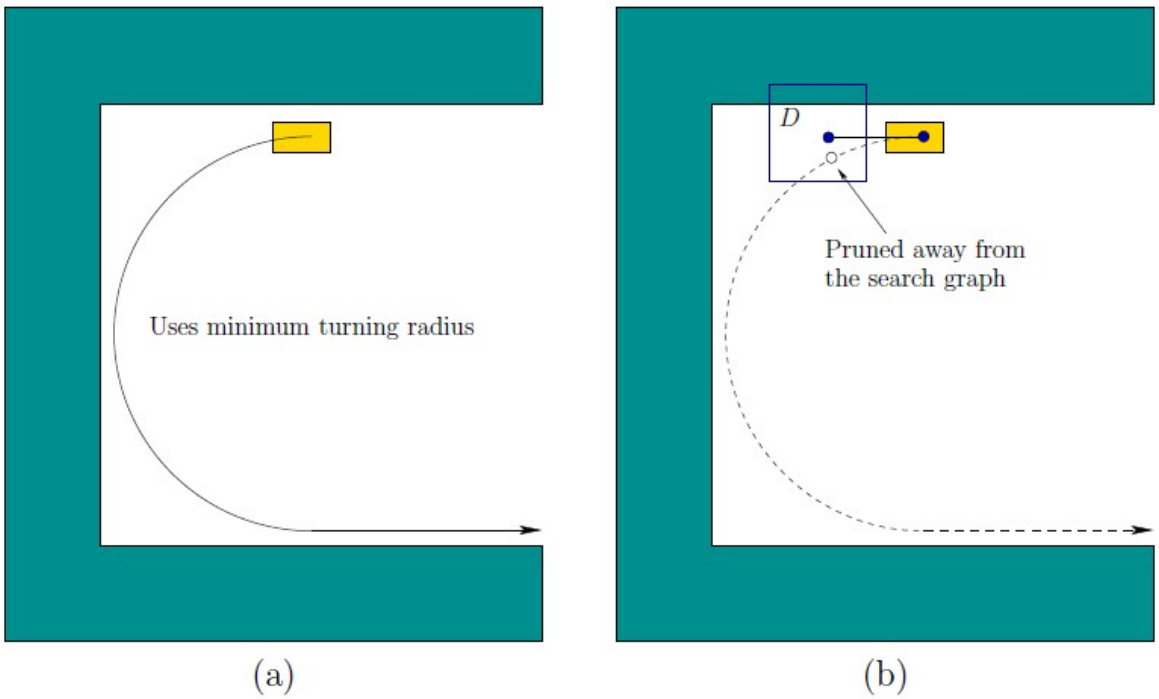
6.4 Problémy při řešení

Jednou z hlavních komplikací při použití stavové diskretizace je, že existují tři prostory, ve kterých dochází ke vzorkování: čas, akční prostor a stavový prostor. Předpokládejme, že je používán diskrétní model. Pro optimální řešení je důležité používat velmi malé buňky. To ovšem omezuje jeho použití na méně-rozměrné stavové prostory. Časový interval Δt by měl být rovněž malý, ale pokud je příliš malý vzhledem k velikosti buněk, pak může být nemožné opustit buňku. Jestliže je jediným požadavkem proveditelnost, pak mohou být použity větší buňky, a Δt musí být odpovídajícím

způsobem navýšeno. Průběh kvantování *U* může způsobit nezdar řešení, zvláště pokud je interval Δ*t* velký. Čím menší Δ*t* je, tím se počet vzorků v U_d stává méně důležitým.

K získání úspěšného řešení je třeba zlepšit vzorkování pokaždé, když vyhledávání selže. Pokaždé, kdy se spustí vyhledávání, by měl být snížen alespoň jeden parametr vzorkování. Můžeme redukovat časový interval Δ*t* či přidat další akce do U_d.

Obrázek 38 znázorňuje, jak chybějící možná akce může zapříčinit vážné problémy o mnoho kroků později. Při použití maximálního natočení kol by Dubinsovo auto bylo schopno se zcela otočit, jelikož však jedna z buněk na trajektorii neobsahuje potřebný vrchol, není možné tento manévr provést.



Obr. 38 Ilustrace problému z důvodu prořezání grafu [1]

7 IMPLEMENTACE

V diplomové práci byly v rámci plánování cesty neholonomního robotu implementovány a porovnány tyto vybrané metody:

- Pravděpodobnostní stromy (*The Rapidly Exploring Dense Tree*)
- Začlenění diskretizace stavového prostoru (*Incorporating State Space Discretization*)
- Inteligentní globální plánovač (*The Intelligent Global Path Planner with Replanning*)

Navržená aplikace byla naprogramována v prostředí Borland Delphi 7 a základní stavbou navazuje na aplikaci z diplomové práce [6]. Aplikace je vytvořena za použití technik objektově orientovaného programování. Uživatelské rozhraní je napsáno pomocí Win32 API (*Windows Application Programming Interface*) a pro grafické výstupy používá knihovnu OpenGL.

Aplikace simuluje plánování cesty mobilnímu neholonomnímu robotu typu Auto. K nalezení cílové cesty v pracovním prostoru robota používá výše zmíněné metody. Algoritmus musí vracet bezkolizní cestu, po které se robot dostane ze startovní pozice do cílové. Pro tvorbu samotného pracovního prostředí byl naprogramován *Editor překážek*, který vytváří 2D mapu konfiguračního prostoru s polygonálními překážkami. Před samotným plánováním cesty je tedy již dopředu znán prostor, ve kterém se robot nachází.

7.1 Model robotu

Implementovaným modelem robotu je mobilní neholonomní robot, konstrukcí odpovídající jednoduchému autu. Robot se může pohybovat vpřed i vzad a může natáčet přední kola podle požadovaného směru. Toto natáčení je omezené maximálním povoleným úhlem. V implementaci byla zvolena defaultní hodnota tohoto úhlu $\varphi_{max} = 35°$, ovšem v *Nastavení* programu lze tento úhel měnit. Referenční bod, který určuje pozici robotu v konfiguračním prostoru, se nachází ve středu zadní nápravy auta.

Dále stanovme vstupní množinu akcí U. Ta je specifikována akčními proměnnými $u = (u_s, u_\varphi)$, kde $u \in U$ a u_s značí rychlost a u_φ úhel natočení kol φ.

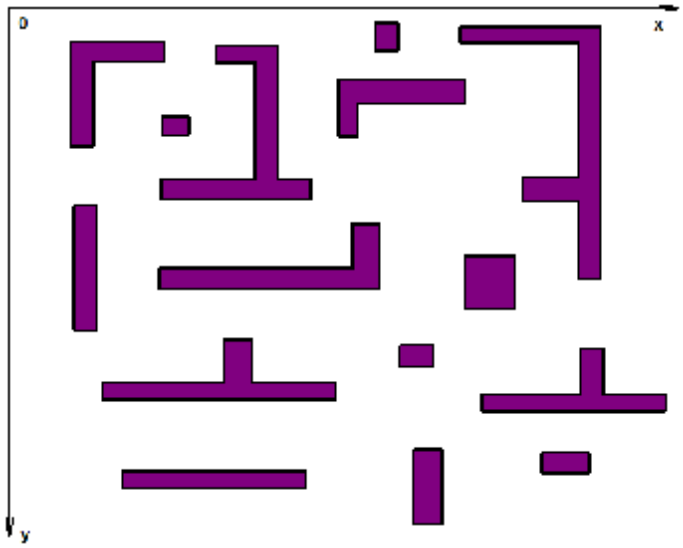
V reálné situaci není rychlost auta konstantní, avšak my pro zjednodušení modelu zanedbáme dynamiku a předpokládáme pomalý bezpečný pohyb. Rychlost robotu tedy může být –1, 0, či 1 podle toho zda auto couvá, stojí nebo jede vpřed. Vstupem u pak modifikujeme konfigurace robotu.

Pohyb implementovaného robotu je popsán následujícími konfiguračními pohybovými rovnicemi, které popisují auto s náhonem na přední kola [4]. Odvození pohybových rovnic robotu typu auta je uvedeno v kapitole 2.2.

$$\begin{aligned} \dot{x} &= u_s . cos(\varphi) . cos(\theta) \\ \dot{y} &= u_s . cos(\varphi) . sin(\theta) \\ \dot{\theta} &= \frac{u_s}{L} . sin(\varphi) \end{aligned} \qquad (6)$$

7.2 Model prostředí

Prostředí je řešeno jako 2D spojitá oblast omezená svými hranicemi. V tomto prostředí se mohou nacházet překážky, které jsou reprezentovány svými vrcholy a hranami, spojující tyto vrcholy. Překážky mohou být tvořeny jednoduchými polygony (konkávními nebo konvexními), nemění svoji polohu v čase a robot je může pouze obcházet (statické, neprostupné).



Obr. 40 Grafická reprezentace prostředí

7.3 Metrika

Implementovaná metrika je využita z práce [4], jelikož lépe vystihuje vzdálenost mezi konfiguracemi robotu než standardní Euklidovská metrika.
Rovnice metriky:

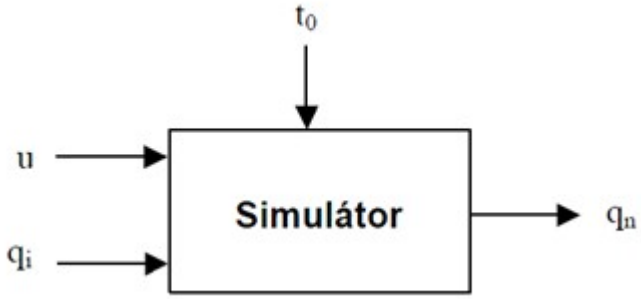
$$d(p,q) = \sqrt{(p.x - q.x)^2 + (p.y - q.y)^2 + L^2.\min\left[(p.\theta - q.\theta), (p.\theta - q.\theta + 2\pi), (p.\theta - q.\theta - 2\pi)\right]^2}$$

kde p a q jsou konfigurace robotu a L vzdálenost os robota. *(7)*

7.4 Inkrementální simulátor

Jelikož jsou pohybové rovnice vyjádřeny pomocí diferenciálních rovnic, pro výpočet nových konfigurací po určitém čase Δt potřebujeme nějaký integrátor. Tento modul nazýváme systémovým simulátorem. Simulátor spočte novou konfiguraci q_n aplikováním vstupu u na konfiguraci q_i po dobu t_0. Tato doba nám určuje vzdálenost, kterou robot ujede, a tím i krok, o který se prohledávaný graf může rozrůst během jedné iterace. Integrační časovou konstantu t_0 je možno měnit v *Nastavení* aplikace u příslušné metody.

Pro integraci pohybových rovnic máme na výběr ze dvou numerických metod a to metodu Runge-Kutta 4. řádu či jednodušší Eulerovu metodu. U srovnávacích experimentů bylo využito přesnější metody Runge-Kutta.



Obr. 41 Schéma inkrementálního simulátoru

7.5 Detektor kolizí

RRT

Pro detekci kolize robotu s překážkami je použito testování průsečíku úseček. Toto testování probíhá vůči obdélníku, který představuje implementovaného robota a jeho rozměry jsou 20 x 25 pixelů. Obdélník se rozdělí na čtyři úsečky, které se následně testují se všemi hranami překážek. Pokud některá z úseček obdélníku protíná některou hranu překážky, je detekována kolize.

Při použití tohoto způsobu detekce kolizí ovšem nastává problém, jestliže velikost kroku robotu (velikost rozrůstání stromu RRT) je příliš velká. Jelikož se vůči překážkám netestuje trasa, ujetá robotem v daném kroku, ale obdélník představující robota, mohlo by dojít k „přeskočení“ robotu přes hranu překážky, čímž by se dostal do vnitřního prostoru této překážky a kolize by nebyla detekována. Z tohoto důvodu je v použité implementaci RRT metody omezena horní hranice integrační časové konstanty t_0 hodnotou 15.

IGPPR a ISSD

Jelikož u plánovacích metod IGPPR a ISSD je pracovní prostor reprezentován diskrétně (prostorem skládajícím se z buněk), detektor kolizí voláme již při inicializaci konfiguračního prostoru. Na rozdíl od metody RRT, kdy musíme v průběhu rozrůstání stromu otestovat každý nově vytvořený vrchol grafu (konfiguraci robotu), u těchto metod testujeme samotné buňky.

U metody IGPPR mají zpočátku všechny buňky přiřazené ohodnocení -1. Při vytváření mapy ohodnocení se pomocí směrové metody postupně ohodnocují všechny buňky, vyjma těch, které spadají do C_{obs}. Kromě ohodnocení se zároveň provádí testování buňky pomocí detektoru kolizí. Velikost každé buňky je navýšena o rozměr robotu, z důvodu zabránění situacím, kdy je robot příliš blízko u překážek. Samotná kolize buňky s překážkami se pak zjišťuje jako u metody RRT pomocí testování průsečíků úseček. Buňky, kterým po ohodnocení a otestování na kolizi zůstane hodnota ohodnocení -1, jsou prohlášeny za kolizní.

Jelikož je ovšem počátek vytváření mapy ohodnocení v buňce, ve které se nachází cílová pozice, musí se tato pozice nacházet v C_{free}. Pokud by se cílová pozice nacházela uvnitř nějaké překážky, dojde k ohodnocení pouze vnitřního prostoru dané překážky a zbylá část konfiguračního prostoru by byla považována za C_{obs}.

U metody ISSD jsem pro určení volného konfiguračního prostoru C_{free} využil 4směrové metody z algoritmu IGPPR. Jelikož metoda ISSD nevyužívá ohodnocení buněk, pomocí směrové metody buňky testujeme pouze na kolizi s překážkami. Detekce kolizí buněk vůči překážkám probíhá opět pomocí testování průsečíků úseček. Je-li testovaná buňka kolizní, její parametr *occupied* je nastaven na *true*. Počátek vytvářené „mapy ohodnocení“ u metody ISSD se nachází v cílové pozici, proto před spuštěním prohledávacího algoritmu musíme dbát na umístění této konfigurace v nekolizní oblasti C_{free}. Zde ovšem nedochází k vytváření nové mapy ohodnocení před každým spuštěním prohledávacího algoritmu jako u IGPPR, z důvodu absence ohodnocení buněk a neměnícího se konfiguračního prostoru. „Mapa ohodnocení“ se vytváří pouze při prvním spuštěním algoritmu, s kterou se nadále pracuje. Novou mapu ohodnocení vytvoříme automaticky při načtení nových překážek nebo při změně velikosti buněk pracovního prostoru.

7.6 Metoda RRT

Implementovaný RRT algoritmus používá pro prohledávání konfiguračního prostoru jeden strom. Jak již bylo řečeno v teoretické části, u dvou-stromové varianty u aplikací neholonomního plánování je napojení stromů na sebe mnohem obtížnější. Ke zrychlení konvergence k cíli byla využita kombinace funkcí GoalBias a GoalZoom z práce [4]. Tato funkce generuje náhodnou konfiguraci q_{rand}, za kterou je vybrána s pravděpodobnosti PG cílová konfigurace q_{goal} a s pravděpodobností PG1 náhodná konfigurace ležící v určitém okolí D od q_{goal}. V ostatních případech je q_{rand} zcela náhodná a může se nacházet kdekoli v konfiguračním prostoru.

Pro kvalitnější nalezení cesty bylo dále využito funkce Connect, zlepšující konvergenci k cíly. Ta je ovšem od původní funkce omezena maximálním počtem iterací R (nastavitelná hodnota) z důvodu zbytečného prodlužování cesty vytvářením tras tvaru kružnic.

V případě, kdy rostoucí RRT strom je ještě relativně daleko od q_{goal}, je žádoucí, aby spád k cíli nebyl příliš velký a tím se strom rozrůstal do všech stran víceméně rovnoměrně. Při příliš silném spádu by hrozilo uvíznutí v lokálním minimu a strom by se zastavil o některou z překážek, kde by se stále snažil růst nejkratší cestou k cíli a nemohl by tak překážku obejít. Naopak nachází-li se RRT strom relativně blízko cílové konfiguraci, je vhodné tento spád k cíly zvýšit. Z tohoto důvodu algoritmus funguje ve dvou módech konvergence (*Normal Mode* a *Goal Mode*), v nichž se nastavení hodnot PG, PG1 a D liší. [6]

Jelikož u neholonomního robotu je v podstatě nemožné dosáhnout přesné cílové konfigurace v rozumném čase, musíme nastavit určitou toleranci, pomocí které vyhodnocujeme, zda danou konfiguraci můžeme pokládat za cílovou. Touto tolerancí je hraniční vzdálenost D_{End} od cíle. Pokud je tedy vzdálenost mezi q_{new} a q_{goal} menší než D_{End} můžeme tvrdit, že cílové konfigurace bylo dosaženo.

Nalezená cesta z q_{start} do q_{end} (nejbližší konfigurace ke q_{goal}) se vyhledá v obráceném směru od q_{end} do q_{start} pomocí předchůdců a následně se otočí. Výsledná cesta je potom tvořena posloupností konfigurací robotu a vstupů, které reprezentují přechody mezi jednotlivými konfiguracemi.

Algorimus **BUILD_RRT** (q_{start}, q_{goal}) [6]

1. Nastav „normální“ mód konvergence (*Normal Mode*)
2. Jako kořen stromu použij q_{start}
3. V K iteracích opakuj
   a. Pomocí funkce **GoalBiasedConf** vygeneruj náhodnou konfiguraci q_{rand}
   b. Najdi nejbližší vrchol stromu q_{near} ke q_{rand}
   c. Pomocí funkce **OptmInput** (q_{rand}, q_{near}) vypočítej optimální vstup $u = (u_s, u_\varphi)$
   d. Jestliže má spočtený vstup nulovou rychlost, pak přeskoč na další iteraci (krok 3.a)
   e. Spočti novou konfiguraci robota q_{new} aplikací vstupu u na q_{near} (pomocí inkrementálního simulátoru)
   f. Přidej vrchol q_{new} do stromu
   g. Přidej hranu do stromu spojující q_{new} a q_{near} s ohodnocením u
   h. Jestliže vzdálenost mezi q_{new} a q_{goal} je menší než D_{End} (cílová konfigurace q_{goal} byla dosažena) ukonči cyklus a skoč na krok 4
   i. Jestliže vzdálenost mezi q_{new} a q_{goal} je menší než $D_{GoalMode}$, aktivuj „cílový“ mód konvergence (*Goal Mode*)
   j. V R iteracích opakuj
      i. Poslední konfiguraci q_{new} označ jako q_{rand}
      ii. Spočti novou konfiguraci q_{new} aplikací již vypočítaného vstupu u na q_{rand}
      iii. Jestliže vzdálenost mezi q_{new} a q_{goal} je větší než v předchozí iteraci nebo q_{new} je kolizní, pak ukonči cyklus
      iv. Přidej vrchol q_{new} do stromu
      v. Přidej hranu do stromu spojující q_{new} a q_{near} s ohodnocením u
      vi. Jestliže vzdálenost mezi q_{new} a q_{goal} je menší než D_{End} (cílová konfigurace q_{goal} byla dosažena) ukonči cyklus a skoč na 4
      vii. Jestliže vzdálenost mezi q_{new} a q_{goal} je menší než $D_{GoalMode}$, aktivuj „cílový“ mód konvergence (*Goal Mode*)
4. Najdi vrchol stromu q_{end}, který je nejblíže ke q_{goal}
5. Najdi cestu stromem z jeho kořene q_{start} do q_{end}.

Podstatnou úlohou RRT algoritmu je nalezeni nekolizního optimálního vstup u, který minimalizuje vzdálenost mezi q_{near} a q_{rand}. Toto má na starosti funkce OptmInput.

V prvním kroku funkce nejprve spočte optimální vstup u pomocí minimalizace funkce metriky. Tento optimalizační problém hledá extrém rovnice metriky (7) - derivace podle $d\varphi$ rovna nule, kde p je nově hledaná konfigurace a q náhodně vygenerovaná konfigurace. V druhém kroku se tento vstup aplikuje pomocí inkrementálního simulátoru na q_{near}, čímž se získá nová konfigurace q_{new}. Je-li tato konfigurace nekolizní, funkce vstup u navrátí a skončí.

Je-li spočtená konfigurace kolizní, pak se upraví daný vstup u tak, že se po malých krocích $\Delta\varphi$ mění natočení předních kol, dokud úhel φ nepřesahuje maximální povolený φ_{max}, a následně se spočte nová konfigurace. Pokud se funkci OptmInput nepodaří najít žádný nekolizní vstup u, nastaví nulovou rychlost akce ($u_s = 0$).

function **OptmInput** (q_{rand} , q_{near}) [6]

1. Najdi optimální vstup u minimalizací funkce pro výpočet vzdálenosti
2. Aplikuj vstup u na q_{near} a výslednou konfiguraci označ jako q_{new}
3. Pokud není q_{new} kolizní, vrať u a ukonči funkci
4. Opakuj, dokud $u\varphi < \varphi_{max}$
   a. $u\varphi = u\varphi + \Delta\varphi$ (kde $\Delta\varphi$ je přičítaný malý úhel)
   b. Aplikuj nový vstup u na q_{near} a výslednou konfiguraci označ jako q_{new}
   c. Pokud není q_{new} kolizní, vrať u a ukonči funkci
5. Opakuj, dokud $u\varphi > \varphi_{max}$
   a. $u\varphi = u\varphi - \Delta\varphi$ (kde $\Delta\varphi$ je odečítaný malý úhel)
   b. Aplikuj nový vstup u na q_{near} a výslednou konfiguraci označ jako q_{new}
   c. Pokud není q_{new} kolizní, vrať u a ukonči funkci
6. Překážce se nelze vyhnout, nastav $us = 0$, vrať u a skonči.

7.7 Metoda IGPPR

Implementovaný algoritmus IGPPR je založen na metodě prohledávání grafu s vrcholy, umístěnými v diskrétním konfiguračním prostoru. Jedná se o modifikaci algoritmu A*. Jelikož se robot nachází ve statickém a neměnícím se prostředí, nevyužíváme zde procedury přeplánování, a tudíž nemusíme uchovávat všechny možné následníky.

Pro ohodnocení vrcholů grafu metoda používá funkci *Evaluate*, která spočte ohodnocení *f* pomocí druhého typu heuristické funkce zmíněné v teoretické části.

$f(N) = g(N) + h(N)$

kde: $g(N)$ – ohodnocení cesty z vrcholu q_{root} do N (počet provedených akcí z vrcholu q_{root} do N (hloubka uzlu) vynásobený krokem dt)

$h(N)$ – ohodnocení buňky, ve které se uzel nachází (pomocí směrové metody).

Pro vytvoření mapy ohodnocení a zároveň definování volného prostoru C_{free} používá metoda IGPPR 4 nebo 8 směrovou metodu šíření, které jsou plně dostačující.

Algorimus **IGPPR** (q_{root}, q_{fruit})

1. Pomocí procedury **CreateCostMap** vytvoř mapu ohodnocení pro daný prostor
2. Do seznamu *OPEN* přidej q_{root} a přiřaď *found = False*
3. Dokud (*found = False*) a (*Open*.Count > 0) opakuj
   a. Ze seznamu *OPEN* vyber vrchol s nejlepším ohodnocením a označ ho jako q_{exp}
   b. Smaž q_{exp} z *OPEN* a přidej do *CLOSED*
   c. V K iteracích opakuj
      i. Pomocí funkce **ActionInput** (K, q_{exp}) vrať optimální vstup u = $(us, u\varphi)$
      ii. Pokud u=***nil*** přeskoč na další iteraci (krok 3.c)
      iii. Spočti novou konfiguraci q_{new} aplikací vstupu u na expandující konfiguraci q_{exp}
      iv. Pomocí funkce **Evaluate** ohodnoť q_{new}
      v. Přidej vrchol q_{new} do grafu
      vi. Přidej do grafu hranu spojující q_{exp} a q_{new} s ohodnocením u

vii. Jestliže vzdálenost mezi q_{new} a q_{fruit} je menší než D_{End} (dosažení cílové konfigurace q_{fruit}), přiřaď ***found*** = ***True***, ukonči cyklus a q_{new} označ jako q_{end}

viii. Pokud hloubka nově přidávaného vrcholu q_{new} je menší než maximální povolená, pomocí funkce **SortedList** přidej vrchol do seznamu *OPEN*

4. Je-li dosaženo cílové konfigurace, najdi cestu grafem z q_{root} do q_{end}.

V prvním kroku algoritmu se nejprve vytvoří mapa ohodnocení pomocí procedury CreateCostMap. Ta rozdělí prostor robota do matice buněk velikosti *dt*. Všechny buňky se zinicializují s ohodnocením -1, mimo buňky obsahující cílovou konfiguraci q_{fruit}. Ta je ohodnocena 0. Pak pomocí vybrané směrové metody šíření se postupně ohodnotí všechny buňky v konfiguračním prostoru (počátek šíření je v cílové buňce).

V dalším kroku se vždy vybere vrchol s nejlepším ohodnocením, který se bude dále expandovat.

Počet iterací *K* je stanoven na 6 - šest možných směrů pohybu robota typu Auto (viz. obr.29). Tyto směry pak odpovídají vstupním akcím *u*, které generuje funkce ActionInput.

Vstupní množinou akcí je tedy:

$$U = \{ (u.s = 1, u.\varphi = 0), (u.s = 1, u.\varphi = -\varphi max), (u.s = 1, u.\varphi = +\varphi max),$$
$$(u.s = -1, u.\varphi = 0), (u.s = -1, u.\varphi = -\varphi max), (u.s = -1, u.\varphi = +\varphi max) \}.$$

Funkce ActionInput vrací pouze takové vstupní akce *u*, které vedou k nalezení nové nekolizní konfigurace a současně vrcholu s lepším ohodnocením než má jeho předchůdce. V opačném případě funkce navrátí vstupní akci s hodnotou ***nil*** a pokračuje se další iterací. Při objevení nové nekolizní konfigurace se daný vrchol ohodnotí použitím funkce Evaluate a přidá se do grafu i s hranou ohodnocenou dle *u*. Pokud hloubka nově přidávaného vrcholu q_{new} je menší než maximální povolená, vrchol se současně přidá do seznamu *OPEN* pomocí funkce SortedList, která setřiďuje seznam od nejhoršího ohodnocení po nejlepší.

Stejně jako u předchozí metody pomocí hraniční vzdálenosti (tolerance) D_{End} určujeme, kdy můžeme prohlásit danou konfiguraci za cílovou. Je-li cílové konfigurace dosaženo, cesta z q_{root} do q_{end} se nalezne pomocí předchůdců.

7.8 Metoda ISSD

Implementovaný algoritmus rozkládá pracovní prostor robota pomocí kubického rozdělení. V každé buňce o velikosti *dt* je povolen maximálně jeden vrchol grafu. Prohledávaný graf je konstruován inkrementálně z počátečního stavu q_{start} použitím prohledávacího algoritmu A*, který pro ohodnocení vrcholů grafu používá funkce *Evaluate*. Ohodnocení *f* se spočte pomocí funkce:

$$f(N) = g(N) + h(N)$$

kde: $g(N)$ – ohodnocení cesty z vrcholu q_{root} do N (počet provedených akcí z vrcholu q_{root} do N (hloubka uzlu) vynásobený krokem *dt*)

$h(N)$ – vzdálenost mezi konfiguracemi N a q_{goal} (využití metriky)

Kontrola kolizí s překážkami je provedena v předstihu, což nám umožňuje vyhnout se buňkám, které obsahují kolizní konfigurace, aniž bychom museli následně volat modul detekce kolizí.

Algorimus **ISSD** (q_{start}, q_{goal})

1. Pomocí procedury **Decomposing** rozlož pracovní prostor
2. Do seznamu *OPEN* přidej q_{start} a přiřaď *found* = *False*
3. Dokud (*found* = *False*) a (*Open*.count > 0) opakuj
   a. Ze seznamu *OPEN* vyber vrchol s nejlepším ohodnocením a označ ho jako q_{exp}
   b. Smaž q_{exp} z *OPEN* a přidej do *CLOSED*
   c. V K iteracích opakuj
      i. Pomocí funkce **AddNodeToGraph** (q_{exp} , *K*) spočti novou konfiguraci q_{new}
      ii. Pokud (q_{new} = ***nil***) přeskoč na další iteraci (krok 3.c)
      iii. Jestliže vzdálenost mezi q_{new} a q_{goal} je menší než D_{End} (dosažení cílové konfigurace q_{goal}), přiřaď *found* = *True*, ukonči cyklus a q_{new} označ jako q_{end}
      iv. Pokud hloubka nově přidávaného vrcholu q_{new} je menší než maximální povolená, pomocí funkce **SortedList** přidej vrchol do seznamu *OPEN*
4. Je-li dosaženo cílové konfigurace, najdi cestu grafem z q_{start} do q_{end}.

Algoritmus nejprve pomocí procedury Decomposing rozloží pracovní prostor robota na matici buněk rozměru *dt*, které uchovává v n-rozměrném poli. Všem buňkám se zinicializují parametry (*cell.visited* = ***False***), (*cell.occupied* = ***True***) a (*cell.node* = ***nil***). Buňce obsahující startovní konfiguraci q_{start} se nastaví parametry *visited* na ***True*** a *occupied* na ***False***. Použitím 4 směrové metody šíření se pak postupně testují jednotlivé buňky na kolize s překážkami pomocí Detektoru kolizí (počátek šíření je v cílové buňce).

Po zdiskretizování prostoru robota se do seznamu OPEN přidá startovní konfigurace q_{start} a v *K* iteracích se pak expanduje. Počet iterací *K* je stanoven na 6 - šest možných směrů pohybu robota typu Auto (viz. obr.29) . Vstupní akční množinou tedy je:

$$U = \{ (u.s = 1, u.\varphi = 0), (u.s = 1, u.\varphi = -\varphi max), (u.s = 1, u.\varphi = +\varphi max),$$
$$(u.s = -1, u.\varphi = 0), (u.s = -1, u.\varphi = -\varphi max), (u.s = -1, u.\varphi = +\varphi max) \}.$$

Funkce AddNodeToGraph má za úkol generovat nové konfigurace. Zahrnuje v sobě také generátor vstupních akcí. Ten podle aktuální iterace *K* vybere odpovídající vstupní akci *u*, kterou aplikuje na vrchol q_{exp}. Poté se pomocí funkce PointLocation nalezne buňka cell, ve které by se měl nacházet nově generovaný vrchol. Je-li buňka cell nekolizní a nebyla ještě navštívena, přidá se do ní nový ohodnocený vrchol q_{new} (*cell.node* = q_{new}) a buňka se označí jako navštívená. V případě, že daná buňka již obsahuje nějaký vrchol (*cell.visited* = ***True***), porovná se ohodnocení tohoto vrcholu s nově přidávaným a pokud má nový vrchol lepší ohodnocení, přidá se do buňky. Po uložení nového vrcholu do *cell.node* se následně přidá do grafu vrchol s příslušnou hranou. Má-li q_{new} ohodnocení horší, funkce AddNodeToGraph se ukončí s návratovou hodnotou ***nil*** a do grafu se nic nepřidá.

Pokud hloubka nově přidávaného vrcholu q_{new} je menší než maximální povolená, vrchol se současně přidá do seznamu *OPEN* pomocí funkce SortedList, která setřiďuje seznam od nejhoršího ohodnocení po nejlepší.

Po dosažení cílové konfigurace se cesta z q_{start} do q_{goal} nalezne pomocí předchůdců.

function **AddNodeToGraph** (q_{exp},K)

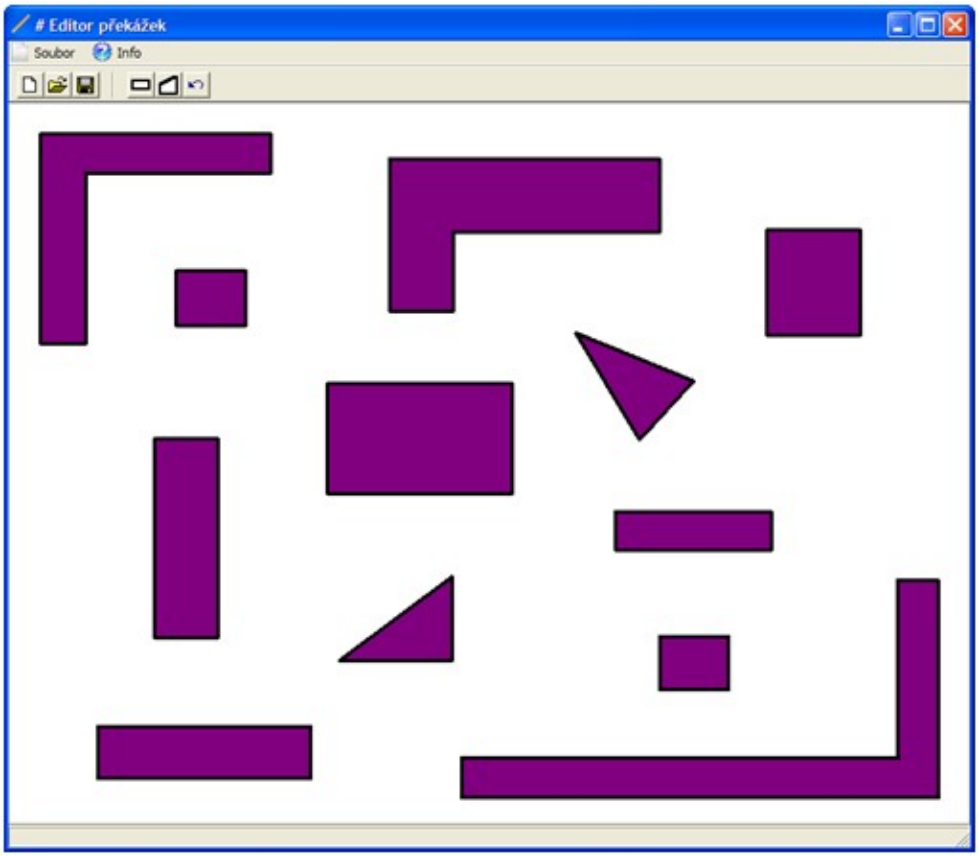
1. Podle iterace K vyber ze vstupní množiny akci u
2. Aplikuj akci u na q_{exp} a novou konfiguraci označ jako q_{new}
3. Pomocí funkce **PointLocation** zjisti buňku, v které by se měla nacházet konfigurace q_{new} a označ ji jako *cell*
4. Pokud je buňka *cell* nekolizní pak
    a. Ohodnoť vrchol q_{new} funkcí **Evaluate**
    b. Pokud je buňka *cell* již navštívená pak
        i. Je-li ohodnocení vrcholu q_{new} horší než ohodnocení vrcholu v buňce *cell* pak ukonči funkci ***(result = nil)***
        ii. Do buňky *cell* ulož nový vrchol q_{new}
    c. Pokud je buňka *cell* nenavštívená pak
        i. Buňku *cell* označ jako navštívenou
        ii. Ulož do ní vrchol q_{new}
    d. Přidej vrchol q_{new} do grafu
    e. Přidej do grafu hranu spojující vrcholy q_{exp} a q_{new} s ohodnocením u
    f. Přiřaď q_{new} návratové hodnotě funkce

Z důvodu zbytečného prohledávání celé oblasti a rychlejšího nalezení kvalitnější cesty, funkce AddNodeToGraph vrací jen vrcholy s lepším ohodnocením oproti předchůdci. Jelikož ovšem u této metody prořezáváme prohledávací graf (povolením maximálně jednoho vrcholu v každé buňce), ne vždy se musí podařit nalézt cestu do cíle. Proto funkce AddNodeToGraph vrací vrcholy s lepším ohodnocením až od určité hloubky vrcholů (nastaveno na hloubku 10). Tímto umožníme zpočátku rozrůstání stromu grafu i do oblastí s horším ohodnocením a následné možné rozrůstání grafu z těchto oblastí v případě nenalezení cesty (či uvíznutí) striktním dodržováním výběru nejlépe ohodnocených vrcholů.

Avšak i přesto nemusí být cesta ze startovní pozice do cílové vždy nalezena. V případě nenalezení cesty většinou stačí nepatrně změnit startovní či cílovou pozici a spustit algoritmus znovu. Nepomůže-li ani toto, musíme zlepšit vzorkování redukováním časového intervalu Δt.

8 EDITOR PŘEKÁŽEK

Navržená aplikace byla naprogramována v prostředí Borland Delphi 7 pomocí šablony SDI (*Single Document Interface*), která vytváří aplikační rozhraní pro jeden dokument. Editor překážek slouží pro tvorbu samotného pracovního prostředí robota neboli mapu, kterou lze následně použít v simulačním programu. V editoru lze vytvářet jak jednoduché tak i komplexní polygony, ovšem simulační program podporuje pouze ty jednoduché (konvexní, konkávní). Velikost pracovního prostředí je *800x600* bodů.



Obr. 44 Aplikace Editoru překážek

Popis programu

Práce s aplikací je podobná jako s klasickým grafickým editorem. Mimo ukládání a otevírání map umožňuje kreslit obdélníky a obecné polygony.

 *Obdélník* - nakreslí překážku tvaru obdélníku pomocí 2 vrcholů zadaných postupným stisknutím levého tlačítka myši na pracovní ploše.

 *Polygon* - kreslí překážky tvaru polygonu postupným zadávání vrcholů levým tlačítkem myši. Ukončení a spojení posledního vrcholu s prvním se provádí stisknutím pravého tlačítka myši.

 *Zpět* - vrátí se o krok zpět, neboli smaže posledně vytvořený objekt.

9 SIMULAČNÍ PROGRAM

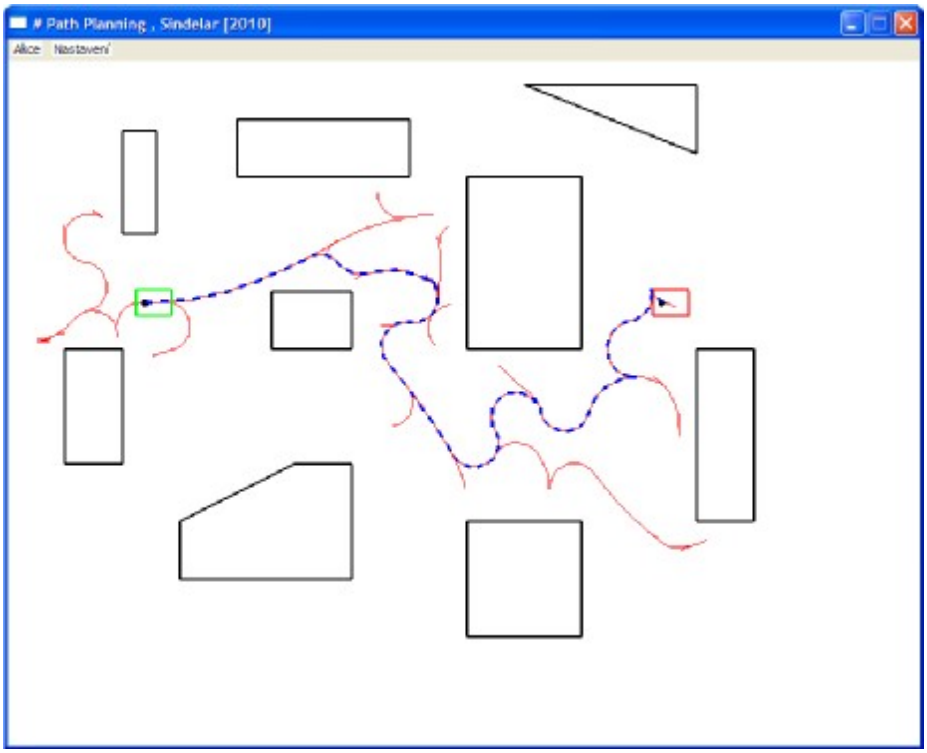
9.1 Popis programu

Navržená simulační aplikace slouží k plánování cesty holonomního robotu v pracovním prostředí, které je definováno mapou, nakreslenou v Editoru překážek. K plánování využívá dříve zmíněné metody (*RRT, IGPPR, ISSD*).

V okně aplikace se zobrazuje mapa pracovního prostředí, kterou lze načíst ze souboru s příponou **.obs* (*Nastavení->Načíst překážky*). Defaultně je načtena mapa s označením *data.obs*, která se nachází ve stejné složce jako samotná aplikace. Hlavní menu aplikace obsahuje dvě podnabídky. V nabídce *Akce* spouštíme výpočet vybrané metody. V nabídce *Nastavení* lze pak jednotlivým metodám měnit konkrétní parametry.

Robot v počáteční a cílové konfiguraci je reprezentován obdélníkem, navýšeným o určitý rozměr. Zelený obdélník značí počáteční konfiguraci a červený značí cílovou konfiguraci. Černá tečka v obdélníku představuje konfiguraci auta (souřadnice *x* a *y*) a je umístěna ve středu zadní nápravy. Počátek souřadného systému se nachází v levém horním rohu pracovní plochy. Pozice startovní konfigurace se zadává levým tlačítkem myši a cílová pozice tlačítkem pravým. Celkové natočení robota lze měnit kolečkem myši. Natočení se provádí vůči středu zadní nápravy auta. Pro změnu natočení cílové pozice robota je třeba ještě použít klávesu CTRL. Překážky jsou vykreslovány černou barvou.

Pro úspěšnost plánování cesty se předpokládá umístění počáteční i cílové oblasti v nekolizní překážkové oblasti. Při spuštění některé vyhledávací metody se v okně zobrazí červenou barvou prohledávací graf a v případě úspěšného nalezení cílové konfigurace se modrou barvou vyznačí nalezená cesta. Následně, je-li zapnuto simulování pohybu robota, se spustí simulace auta, která znázorňuje, jak by po nalezené cestě projížděl skutečný robot. Povolení simulace a vykreslování grafu lze nastavovat v menu *Nastavení -> Ostatní*.



Obr. 45 Hlavní okno aplikace

9.2 Nastavení parametrů metod

Pro specifické nastavení jednotlivých metod slouží podnabídka *Nastavení*. Po výběru některé metody se otevře dialogové okno, v němž lze měnit určité parametry metody. V následujícím textu si tyto parametry popíšeme.

9.2.1 Nastavení RRT

***Odchylka od cíle** (End)* – povolená cílová tolerance D_{End} (vzdálenost mezi dosaženou q_{end} a skutečnou cílovou q_{goal} konfigurací). Čím je tato tolerance vyšší, tím snadněji dosáhneme „cílové“ konfigurace (minimální povolená hodnota je 5).

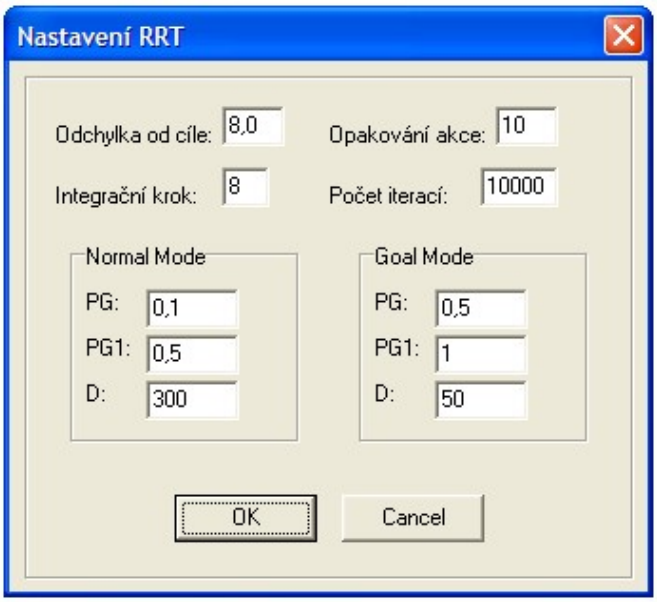
***Opakování akce** (R)* – počet opakování optimální akce. Vyšší hodnoty zvyšují konvergenci k cíly, ovšem za cenu delší cesty (optimálně nastaveno hodnotou 10).

***Počet iterací** (K)* – počet iterací RRT algoritmu. Po dosažení této hodnoty výpočet končí. Je-li hodnota příliš malá, může se stát, že cílová konfigurace nebude nalezena (maximální povolený počet je 30 000).

***Integrační krok** (dti)* – integrační časová konstanta (velikost kroku robota). S vyšší hodnotou dosáhneme rychlejšího nalezení cílové konfigurace. Defaultně nastavena optimální hodnota 8 (možný rozsah nastavení: 3 - 15).
(*Poznámka*: změnou tohoto parametru je potřeba přizpůsobit případný čas u simulace).

Normal Mode – nastavení „normálního” módu
PG - pravděpodobnost výběru cílové konfigurace (rozsah nastavení: 0 – 1)
PG1 - pravděpodobnost výběru konfigurace nacházející se do vzdálenosti *D* od cíle
D - povolená vzdálenost od cílové konfigurace (minimální povolená hodnota je 30).

Goal Mode – nastavení „cílového” módu (PG, PG1, D – viz. *Normal Mode*).



Obr. 46 Dialog pro nastavení metody RRT

9.2.2 Nastavení IGPPR

***Velikost buňky** (dt)* – nastavení velikosti buněk pracovního prostoru.
(*Poznámka*: příliš malé velikosti buněk zpomalují výpočet z důvodu rozkladu pracovního prostoru do objemnějšího n-rozměrného pole. Příliš vysoké hodnoty zase zpomalují konvergenci k cíly zbytečným prohledáváním okolního prostoru, čímž narůstá výpočetní čas. Z tohoto důvodu byl možný rozsah nastavení omezen hodnotami: 5 – 25).

***Integrační krok** (dti)* – integrační časová konstanta (velikost kroku robota). S vyšší hodnotou dosáhneme rychlejšího nalezení cílové konfigurace. Defaultně nastavena optimální hodnota 8 (možný rozsah nastavení: 5 - 30).
(*Poznámka*: změnou tohoto parametru je potřeba přizpůsobit případný čas simulace).

***Maximální hloubka** (MaxDepth)* – nastavení maximální povolené hloubky vrcholů prohledávaného grafu.

***Mapa ohodnocení** (CostMap)* – nastavení čtyř nebo osmi směrové metody šíření pro diskretizaci pracovního prostoru.
(*Poznámka*: Osmi směrová metoda poskytuje přesnější ohodnocení buněk, na druhou stranu ovšem nepatrně prodlužuje dobu výpočtu.)

***Odchylka od cíle** (End)* – povolená cílová tolerance D_{End} (vzdálenost mezi dosaženou q_{end} a skutečnou cílovou q_{goal} konfigurací). Čím je tato tolerance vyšší, tím snadněji dosáhneme "cílové" konfigurace (minimální povolená hodnota je 5).

Obr. 47 Dialog pro nastavení metody IGPPR

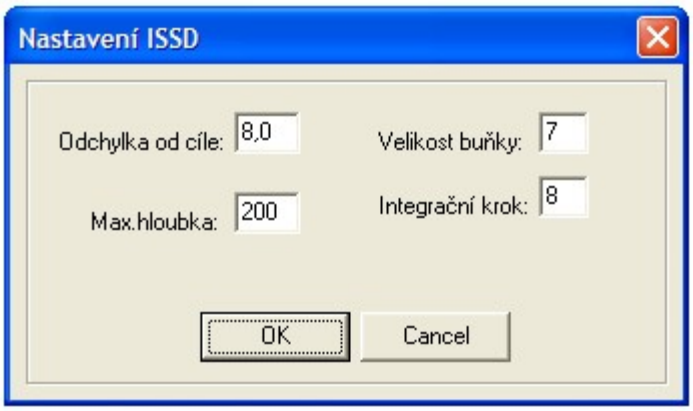
9.2.3 Nastavení ISSD

***Velikost buňky** (dt)* – nastavení velikosti buněk pracovního prostoru.
(*Poznámka*: příliš malé velikosti buněk zpomalují výpočet z důvodu rozkladu pracovního prostoru do objemnějšího n-rozměrného pole. Příliš vysoké hodnoty zase způsobí neúspěšné nalezení cesty do cíle. Jelikož u metody ISSD je povolen maximálně jeden vrchol v každé buňce, velikost buňky musí být menší nebo rovno oproti integračnímu kroku. Možný rozsah nastavení je omezen hodnotami: 5 – 25).

***Integrační krok** (dti)* – integrační časová konstanta (velikost kroku robota). S vyšší hodnotou dosáhneme rychlejšího nalezení cílové konfigurace. Defaultně nastavena optimální hodnota 8 (možný rozsah nastavení: 5 - 30).
(*Poznámka*: změnou tohoto parametru je potřeba přizpůsobit případný čas simulace).

***Maximální hloubka** (MaxDepth)* – nastavení maximální povolené hloubky vrcholů prohledávaného grafu. Dosáhne-li vrchol této hloubky, dál se již neexpanduje.

***Odchylka od cíle** (End)* – povolená cílová tolerance D_{End} (vzdálenost mezi dosaženou q_{end} a skutečnou cílovou q_{goal} konfigurací). Čím je tato tolerance vyšší, tím snadněji dosáhneme cílové konfigurace (minimální povolená hodnota je 5).



Obr. 48 Dialog pro nastavení metody ISSD

9.2.4 Ostatní nastavení

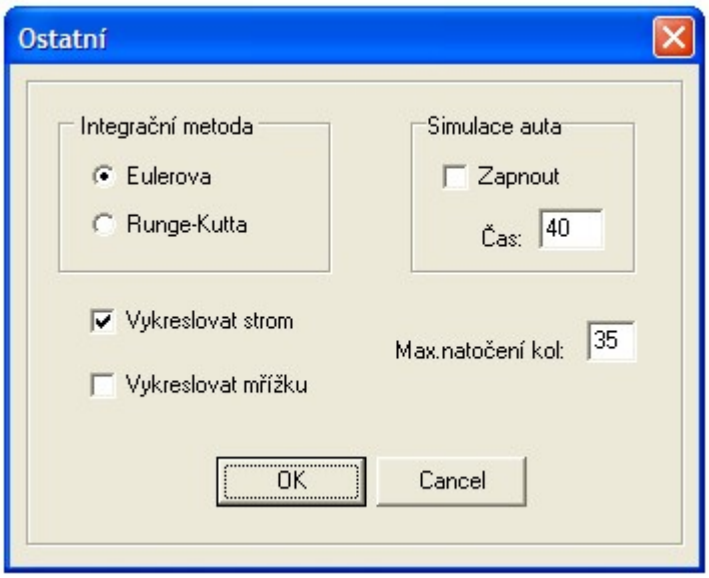
Integrační metoda – pro integraci diferenciálních pohybových rovnic robota si můžeme vybrat jednu ze dvou integračních metod (*Eulerovu* nebo *Runge-Kutta*).

Vykreslovat strom – zaškrtnutím povolíme vykreslování grafu v průběhu výpočtu. (*Poznámka*: vypnutím snížíme paměťové nároky a celkový výpočetní čas).

Vykreslovat mřížku – zaškrtnutím povolíme vykreslování mřížky u metod IGPPR a ISSD (zobrazí jednotlivé buňky pracovního prostoru s jejich „ohodnocením“).
(*Poznámka*: Zobrazování mřížky je vhodné u vyšších velikostí buněk, vypnutím výrazně snížíme celkový výpočetní čas).

Simulace auta – zaškrtnutím povolíme simulaci jízdy robota (znázorňuje pohyb robota po nalezené cestě). Při zapnuté simulaci můžeme specifikovat dobu trvání průjezdu cesty [ms].

Max. natočení kol – maximální povolený úhel natočení předních kol robota. Hodnota parametru musí ležet v intervalu [0, 90]°.



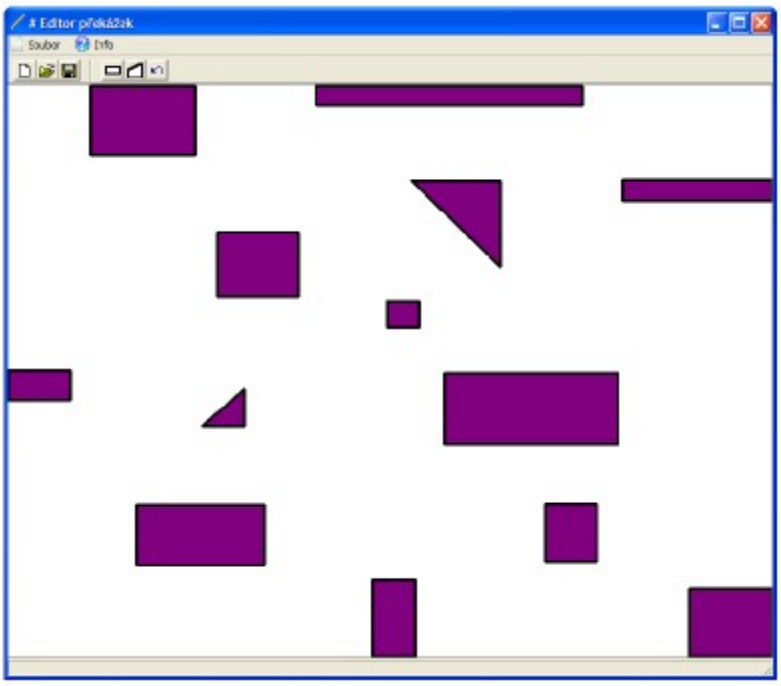
Obr. 49 Dialog pro nastavení metody ISSD

9.3 Požadavky programu na PC

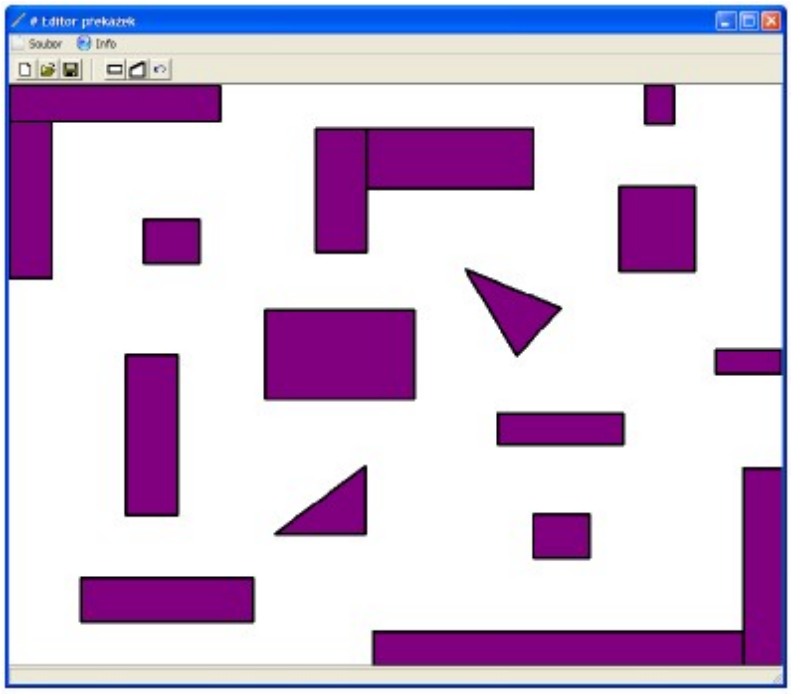
Podporované OS: Windows 2000/XP /Vista
Minimální rozlišení: 1024x768
Operační paměť: 512 MB
Grafická karta : podpora OpenGL

10 EXPERIMENTY

Před samotnými srovnávacími experimenty implementovaných metod byly stanoveny optimální testovací parametry pro nastavení všech metod. Následně byly provedeny srovnávací testy pro zjištění efektivnosti a výkonu jednotlivých prohledávacích algoritmů. Testování probíhalo ve dvou prostředích (jednodušší a složitější varianta) vytvořených pomocí Editoru překážek. Veškeré experimenty probíhaly na PC s konfigurací: AMD Athlon 64 X2 (5000+), 4 GB RAM, Radeon HD 2400XT 256 MB a OS Windows XP SP2.



Obr. 50 Pracovní prostor robota (mapa1)



Obr. 51 Pracovní prostor robota (mapa2)

10.1 Stanovení optimálních parametrů

Aby výsledné srovnání prohledávacích metod bylo objektivní, u všech metod byly při srovnávacích experimentech nastaveny stejné hlavní parametry. Konkrétně se jedná o: integrační krok (*dti*) a odchylku od cíle (*End*). U obou map byly tyto parametry experimentálně měněny a pozorováním kvality nalezené cesty byly určeny společné optimální hodnoty pro následné testování. U metody RRT se zdá optimální nastavení parametrů na hodnoty: *dti* = 5 a *End* = 8. U metod IGPPR a ISSD se tímto optimem jeví nastavení hodnot na: *dti* = 8 a *End* = 5. Metoda RRT má zvýšenou potřebu pro vyšší povolenou odchylku od cíle z důvodu náhodného vybírání vrcholů pro expanzi, čímž je obtížnější dosáhnout přesné cílové pozice. Naopak metoda IGPPR a ISSD vyžaduje vyšší integrační krok z důvodu úplného expandování vrcholů na všechny jejich možné následníky použitím vstupní množiny akcí. S menším integračním krokem je pak potřeba povolit vyšší maximální hloubku vrcholů, čímž zase zvyšujeme výpočetní čas a paměťové nároky. Obzvláště metoda ISSD vyžaduje pro úspěšné nalezení cesty do cíle zvýšený integrační krok oproti metodě RRT. To je dáno tím, že v každé buňce je povolen maximálně jeden vrchol. Proto zde také hraje důležitou roli rozměr buněk, který musí být menší než integrační krok. Velikost buněk má vliv na kvalitu nalezené cesty. Jelikož optimální integrační krok byl zvolen na hodnotu 8, rozměr buněk *dt* byl stanoven hodnotou 7.

Z těchto důvodů byly testovací hodnoty parametrů voleny takto:

- integrační krok: *dti* = 8
- velikost buněk: *dt* = 7
- odchylka od cíle: *End* = 8.

Ostatní hodnoty byly v defaultním nastavení.

10.2 Srovnávací experimenty

10.2.1 Metoda RRT

Opakování akce

V každé mapě byl 20krát spuštěn prohledávací algoritmus RRT pro nalezení cesty z výchozí pozice k cíly a to při vypnuté a následně povolené funkci *Repeat*. Tato funkce slouží pro opakování vstupní akce na nově nalezený vrchol. Hodnota funkce *Repeat* byla optimálně nastavena na *R*=10. Pro každou nalezenou cestu byla zaznamenána doba jejího výpočtu. Nutno podotknout, že do hodnocení výpočtu nebyly brány v potaz případy uvíznuté v lokálním minimu, a že testování probíhalo se zapnutým vykreslováním stromu. Následující tabulky obsahují průměrné hodnoty těchto časů.

Mapa 1	s opakováním akce	bez opakování akce
Doba výpočtu [s]	0,13	0,41
Počet uvíznutí [-]	1	0

Tab. 1 Funkce Repeat (mapa1)

Mapa 2	s opakováním akce	bez opakování akce
Doba výpočtu [s]	0,52	1,11
Počet uvíznutí [-]	3	1

Tab. 2 Funkce Repeat (mapa2)

Z výsledků je vidět, že průměrná doba výpočtu, potřebná k nalezení cesty do cíle, je při vhodném použití funkce pro opakování akce výrazně kratší. Na druhou stranu zvyšuje možný počet neúspěšných případů.

Vykreslování grafu

V následujícím testu se porovnávali průměrné hodnoty z 20 naměřených výpočetních časů s povoleným a následně zakázaným vykreslováním stromu RRT.

Mapa 1	s vykreslováním stromu	bez vykreslování stromu
Doba výpočtu [s]	0,19	0,12

Tab. 3 Funkce DrawTree (mapa1)

Mapa 2	s vykreslováním stromu	bez vykreslování stromu
Doba výpočtu [s]	0,56	0,21

Tab. 4 Funkce DrawTree (mapa2)

Z výsledků je patrné, že dalším urychlením vedoucím k menšímu výpočetnímu času je zakázání vykreslování stromu RRT.

10.2.2 Srovnání IGPPR a ISSD

Experiment spočíval v nalezení cílové cesty se stejně zadanými počátečními a cílovými konfiguracemi pro algoritmus IGPPR i ISSD. Následně byla porovnána délka nalezené cesty a čas potřebný pro její výpočet. Nutno podotknout, že do doby potřebné k nalezení cílové cesty je zahrnuto i vytváření mapy ohodnocení.

Z důvodu efektivního srovnání obou metod byla u metody IGPPR pro rozklad pracovního prostoru použita 4směrová metoda šíření, jakožto u metody ISSD. Dále ještě musel být implementovaný algoritmus ISSD upraven tak, aby se „mapa ohodnocení“ vytvářela při každém spuštění prohledávacího algoritmu. Tato úprava v programu ovšem není dostupná, neboť toto vytváření je potřeba jen při prvním užití algoritmu ISSD. Testování probíhalo se zapnutým vykreslováním grafu.

Mapa 1	IGPPR		ISSD	
Cesta	Délka cesty	Doba výpočtu [s]	Délka cesty	Doba výpočtu [s]
1.	81	0,43	80	0,43
2.	82	1,49	85	0,98
3.	82	1,53	85	0,94
4.	119	0,96	133	1,2
5.	77	0,94	89	1,19
6.	72	1,1	105	0,81
7.	90	1,16	138	0,84
8.	88	1,1	77	1
9.	122	0,59	78	0,43
10.	119	0,58	110	0,48

Tab. 5 IGPPR vs. ISSD (mapa1)

Mapa 2	IGPPR		ISSD	
Cesta	Délka cesty	Doba výpočtu [s]	Délka cesty	Doba výpočtu [s]
1.	102	1	103	1,06
2.	93	1,21	93	1,49
3.	123	1,1	106	1,14
4.	118	0,91	84	1,37
5.	133	1,24	152	1,19
6.	114	0,48	138	0,46
7.	120	0,48	133	1,3
8.	114	0,46	154	0,46
9.	109	0,51	119	1,19
10.	109	0,65	104	0,49

Tab. 6 IGPPR vs. ISSD (mapa2)

Porovnáním hodnot můžeme pozorovat, že u jednodušší mapy dosahuje algoritmus ISSD nepatrně lepších výpočetních časů. U složitější mapy se ovšem výpočetní doba algoritmu ISSD spíše zhoršuje. Na druhou stranu tím, že v implementaci definujeme, aby se „mapa ohodnocení“ vytvářela jen při prvním spuštění prohledávacího algoritmu ISSD, snížíme celkový čas výpočtu o dobu potřebnou na vytvoření této mapy. Tím dostává algoritmus ISSD převahu nad algoritmem IGPPR. Doba spočtení „mapy ohodnocení“ závisí na velikosti buněk pracovního prostoru. Celkový výpočetní čas se dá u obou metod ještě snížit vypnutím vykreslování grafu.

U délek nalezených cílových cest nelze jednoznačně určit, který algoritmus nalezne kratší cestu do cíle. Co se týče kvality nalezené cesty, jsou oba algoritmy srovnatelné. Nalezená cílová cesta pomocí algoritmu ISSD se jeví někdy kvalitnější (plynulejší) než u metody IGPPR, z důvodu menšího počtu změn po cestě. Tomuto přičítám hlavní podstatu ISSD, jelikož povoluje maximálně jeden vrchol v každé buňce a tím redukuje prohledávací graf.

Nutno ovšem podotknout, že hlavní podstata algoritmu ISSD je zároveň jeho největším problémem u aplikací ve složitějších prostředích (zejména typu labyrint), kde už algoritmus hledá cestu do cíle obtížněji (zdali ji vůbec najde), z důvodu omezenějšího pohybu.

10.2.3 Porovnání všech metod

Pro porovnání všech tří metod bylo použito průměrné výpočtové doby algoritmu potřebné k nalezení cesty do cíle. Startovní a cílová pozice byla vždy pro všechny metody stejná. Následně se porovnávaly výpočtové časy všech algoritmů a délky nalezených cest. Jelikož se ovšem u metody RRT generují vrcholy náhodně a tudíž výsledná nalezená cesta do cíle je pokaždé jiná, pro přibližné srovnání algoritmů se u metody RRT vždy vypočetl průměr z 10 naměřených časů a délek potřebných pro nalezení jednotlivých cílových cest. Celkové opakování měření probíhalo 10krát. Nutno poznamenat, že do průměrné hodnoty doby výpočtu RRT se nezapočítávaly stavy, kdy RRT strom uvíznul v lokálním minimu (celkem 16krát). Testování probíhalo opět se zapnutým vykreslováním grafu.

Mapa 1	RRT	IGPPR	ISSD
Doba výpočtu [s]	0,26	1,17	1,17
Délka cesty	155	121	131

Tab. 7 Výsledky měření (mapa1)

Mapa 2	RRT	IGPPR	ISSD
Doba výpočtu [s]	0,43	1,26	1
Délka cesty	175	157	146

Tab. 8 Výsledky měření (mapa2)

Z výsledků je vidět, že s metodou RRT dosáhneme cílové pozice nejrychleji. Musíme ovšem brát v potaz, že se jedná o průměrnou hodnotu náhodně generovaných cest. Obecně se však dá říci, že pokud metoda RRT neuvízne v lokálním minimu, pracovní prostor prohledá nejrychleji. Uvízne-li ovšem, pak doba výpočtu rapidně narůstá, a pokud dojde k maximálnímu povolenému počtu iterací, cesta k cíly nebude nalezena. Pokud se podíváme na kvalitu a délku cesty, tak u metody RRT rozhodně nemůžeme očekávat optimálně nalezenou cestu k cíly. U metod IGPPR a ISSD se používá heuristické ohodnocující funkce, která robota vede co nejefektivněji přímo k cíly a zajišťuje tak optimalizaci nalezené cesty.

11 ZÁVĚR

Cílem této práce bylo analyzovat dosavadní přístupy k plánování cesty robotu a dále implementovat a provést srovnávací experimenty u vybraných metod pro neholonomního robota.

Z výsledků experimentů vyplývá, že metoda RRT při vhodném použití funkce pro opakování akcí a při zakázaném vykreslování stromu dosahuje nejlepších výpočetních časů potřebných k nalezení cílových cest. Musíme ovšem brát v potaz, že se jedná o průměrnou hodnotu náhodně generovaných cest. Problémem algoritmu RRT však mohou být lokální minima. I když v nich RRT neuvázne a cesta je nakonec úspěšně nalezena, znamenají výrazné prodloužení doby potřebné pro nalezení cesty do cíle. Toto riziko hrozí především u složitějších pracovních prostorů s velkým množstvím překážek. V případě častého uvíznutí je vhodné zastavit výpočet tlačítkem Stop a algoritmus spustit znovu. Možného zlepšení lze dosáhnout snížením spádu k cíli, čímž se ale mírně prodlouží doba výpočtu pro všechny hledané cesty.

Co se týče kvality a délky nalezené cesty, tak ta je jednoznačně lepší u metod IGPPR a ISSD z důvodu implementované heuristické funkce pro ohodnocení vrcholů. Pro nalezení cílové cesty u jednoduchých prostředí se jeví nejvhodnějším algoritmem ISSD. Oproti algoritmu IGPPR má tu výhodu, že u něho není třeba pokaždé vytvářet mapu ohodnocení, čímž výpočetní čas narůstá. Dobu pro spočtení mapy ohodnocení můžeme snížit zvětšením velikosti buněk, ale tím ovšem můžeme zase prodloužit výpočetní čas potřebný pro nalezení cílové cesty z důvodu méně přímého vedení k cíly. Kvalita cest u obou metod je srovnatelná. Avšak u složitějších prostředí metoda ISSD nemusí vždy nalézt cestu z důvodu prořezávání prohledávaného grafu, proto je zde vhodnější použít metodu IGPPR.

SEZNAM POUŽITÝCH ZDROJŮ

[1] LaValle, S. M.: Planning Algorithms. Cambridge University Press, 2006.

[2] LaValle, S. M. & Kuffner, J. J.: Rapidly-Exploring Random Trees: Progress and Prospects, 2001.

[3] Podsedkowski, L.: A new solution for path planning in partially known or unknown environment for nonholonomic mobile robots. Robotics and Autonomous Systems 34, 2001.

[4] Jun Qu: Nonholonomic Mobile Robot Motion Planning – Motion Strategy Project, 1999. Dostupné z WWW: http://msl.cs.uiuc.edu/~lavalle/cs576_1999/projects/junqu

[5] Šeda, M.: Teorie grafů. FSI VUT Brno, 2003.

[6] Šíma, M.: Plánování cesty robota ve spojitém prostředí. Diplomová práce. ÚAI FSI VUT Brno, 2006.

[7] Šolc, F. Žalud, L.: Robotika. FEKT VUT Brno, 2002.

[8] Svidenská, A.: RRT plánování pohybu robota aplikovaná ve 2D prostoru. Bakalářská práce. ÚAI FSI VUT Brno, 2007.

[9] Gross, T.: Plánování cesty mobilního robota. Diplomová práce. ÚAI FSI VUT Brno, 2007.

[10] Webové stránky Robotika.cz. Dostupné z WWW: http://robotika.cz

[11] Webová encyklopedie Wikipedia. Dostupná z WWW: http://www.wikipedia.org

[12] Sykala, V.: Hledání cesty robotem. FIT VUT Brno. Dostupné z WWW: http://www.stud.fit.vutbr.cz/~xsykal00/IBP/index.php?co=1

[13] Podsedkowski, L.: Path planner for nonholonomic mobile robot with fast replanning Procedure. Proceedings of the 1998 IEEE International Conference on Robotics and Automation.