VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

# VIZUALIZACE HLEDÁNÍ CESTY PRO ROBOTA

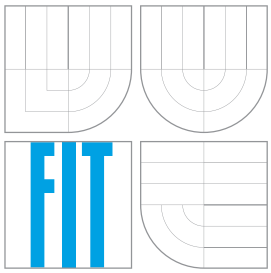
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MIROSLAV KVASNICA

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

# VIZUALIZACE HLEDÁNÍ CESTY PRO ROBOTA

VISUALISATION OF PATH-FINDING FOR ROBOT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MIROSLAV KVASNICA
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAROSLAV ROZMAN
SUPERVISOR

BRNO 2008

## Abstrakt

Tato bakalářská práce pojednává o pravděpodobnostních algoritmech pro hledání cesty robota. Obsahuje teoretický popis pravděpodobnostních algoritmů včetně diskuze nad implementačními detaily, přibližuje jejich využití a popisuje jednotlivé algoritmy – PRM, EST, RRT a SRT včetně jejich dalších modifikací. Součástí práce jsou java applety znázorňující vybrané algoritmy a také webové stránky věnované této problematice.



## Klíčová slova

robotika, hledání cesty, pravděpodobnostní metody, PRM, EST, RRT, SRT



## Abstract

This thesis deals with sampling-based algorithms for robot path planning. Theoretical principles of probabilistic path finding and its implementation details are discussed here. The second part focuses on individual algorithms – PRM, EST, RRT and SRT and its modifications. Java applets for visualisation of algorithms and web pages related with sampling-based algorithms are included on the CD.



## Keywords

robotics, path finding, roadmaps, sampling-based algorithms, PRM, EST, RRT, SRT



## Citace

Miroslav Kvasnica: Vizualizace hledání cesty pro robota, bakalářská práce, Brno, FIT VUT v Brně, 2008

# Vizualizace hledání cesty pro robota

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně
pod vedením pana Ing. Rozmana.

.....
Miroslav Kvasnica
22. července 2008

# Obsah


**1 Úvod** **3**
- 1.1 Hledání cesty robota v prostoru . . . 3
- 1.2 Rozdělení pravděpodobnostních algoritmů . . . 3
- 1.3 PA ve vícerozměrném prostoru . . . 4
- 1.4 Problém stěhování klavíru . . . 5
- 1.5 Vlastnosti PA . . . 6

**2 Využití pravděpodobnostních algoritmů** **7**

**3 Teorie implementace PA** **9**
- 3.1 Vzorkování náhodných bodů z konfiguračního prostoru . . . 9
- 3.2 Výběr nejbližších uzlů z grafu . . . 11
- 3.3 Vzdálenostní funkce . . . 12
- 3.4 Spojování dvou konfigurací . . . 13
- 3.5 Zkracování a vyhlazování cesty (postprocessing) . . . 14

**4 Popis implementace appletů** **15**
- 4.1 Ovládání appletů . . . 15
- 4.2 Hierarchie tříd . . . 17
- 4.3 Implementace řízení simulace . . . 18
- 4.4 Základní simulační funkce . . . 19
- 4.5 Grafické zobrazení simulace . . . 20
- 4.6 Základní nastavení simulace . . . 20

**5 PRM algoritmus – Probabilistic roadmaps** **22**
- 5.1 Vytváření grafu (roadmapy) . . . 22
- 5.2 Dotazovací fáze . . . 24
- 5.3 Implementace PRM appletu . . . 26

**6 EST algoritmus – Expansive-Spaces Trees** **27**
- 6.1 Vytváření stromu . . . 27
- 6.2 Výběr pravděpodobnostní funkce . . . 28
- 6.3 Napojení stromů . . . 29
- 6.4 SBL algoritmus . . . 30
- 6.5 Implementace EST appletu . . . 30

**7 RRT algoritmus – Rapidly-Exploring Random Trees** **34**
7.1 Vytváření stromů . . . . . 34
7.2 Volba délky kroku při vytváření stromů . . . . . 35
7.3 Spojování stromů . . . . . 36
7.4 Implementace RRT appletu . . . . . 37

**8 SRT algoritmus – Sampling-Based Roadmap of Trees** **40**
8.1 Vytvoření SRT grafu . . . . . 40
8.2 Dotazovací fáze . . . . . 41
8.3 Vlastnosti SRT . . . . . 41
8.4 Implementace SRT appletu . . . . . 42

**9 Závěr** **45**

**A Základní pojmy** **46**

**B Obsah přiloženého CD** **48**

# Kapitola 1

# Úvod

Tato bakalářská práce má za cíl přiblížit metody pro hledání cesty robota v prostoru, konkrétně tzv. pravděpodobnostní algoritmy.

Nabízí přehled základních metod a přístupů včetně teoretických implementačnách detailů a také rozbor praktické implementace uvedených metod ve formě java appletů.

## 1.1 Hledání cesty robota v prostoru

Jedním ze základních úkolů, které řeší robotika, je pohyb robotů a s ním spojené vyhledávání cesty ve spojitém prostoru.

Pohyb robota ve spojitém prostoru nelze řešit stejně snadno, jako pohyb v diskrétním prostoru (určeném grafem), pro který existují prověřené deterministické algoritmy. Jeden z možných přístupů tedy spočívá v navzorkování spojitého prostoru a následném vyhledání cesty právě pomocí deterministických algoritmů – a právě takto pracují i pravděpodobnostní algoritmy.

Pravděpodobnostní algoritmy (PA), jak už název napovídá, fungují na základě náhodného výběru bodů v prostoru a v jejich následném pospojování do grafu, ve kterém už se cesta vyhledá některým základním algoritmem (například Dijkstrovým).

Při hledání cesty robota ve spojitém prostoru rozlišujeme dva případy – hledání v neznámém prostoru nebo hledání cesty v prostoru, ve kterém známe rozmístění všech překážek. V případě pohybu v neznámém prostoru se používají například *bug algoritmy*, které pracují na principu postupného prozkoumávání prostoru až během pohybu robota, zatímco pro pohyb v předem známém prostoru užíváme pravděpodobnostní algoritmy. Pravděpodobnostní algoritmy lze také označit jako *off-line* algoritmy, protože výpočet hledání cesty proběhne ještě před samotným pohybem robota.

Existují i jiné algoritmy pro hledání cesty ve známém prostoru (například GVD, Generalized Voronoi Diagram), ale pravděpodobnostní algoritmy lze použít nejen pro jednoduché prostory, ale také pro komplikované hledání cesty ve vícedimenzionálním prostoru u robotů s více stupni volnosti.

## 1.2 Rozdělení pravděpodobnostních algoritmů

Pravděpodobnostní algoritmy lze rozdělit primárně podle toho, zda jimi vytvořený graf zachycující daný prostor můžeme využít opakovaně - tedy k hledání cesty mezi různými body:

- **Vícedotazové** – algoritmus nejprve vytvoří graf (síť bodů, **roadmap**) zachycující daný prostor a pomocí tohoto grafu lze *opakovaně* hledat cestu mezi libovolnými dvěma body prostoru. Patří sem PRM a jeho další modifikace.
- **Jednodotazové** – algoritmus vytváří strom mezi dvěma konkrétními body v prostoru a tento strom tedy nelze vícekrát použít k vyhledání cesty mezi jinými body. Do této kategorie patří RRT a EST.
- **Kombinované** – algoritmus SRT, který stojí na pomezí prvních dvou kategorií, protože jej lze využít k opakovanému vyhledávání, ale může být rychlejší než jednodotazový a navíc jednodotazové algoritmy interně využívá. SRT je obecně nejvýkonnějším algoritmem [2].

Ačkoliv jsou mezi konkrétními algoritmy rozdíly a také záleží na prostoru, ve kterém pracují, lze obecně říci, že jednodotazové algoritmy bývají rychlejší pro nalezení jedné konkrétní cesty mezi dvěma body. Pokud ale požadujeme hledání cest mezi různými body v jednom prostoru, jsou vhodnější vícedotazové algoritmy, kterým sice delší dobu trvá vytvoření grafu, ale ten lze potom použít opakovaně a další hledání jsou rychlejší než u jednodotazových metod [2].

## 1.3 PA ve vícerozměrném prostoru

Nejjednodušším problémem, který pravděpodobnostní algoritmy řeší, je vyhledání trajektorie pohybu statického robota ve dvourozměrném nebo trojrozměrném prostoru.
Jsou ale vhodné i pro komplikovanější pohyb robota složený z jeho posunu a pohybu jeho částí (například robotická ramena) – v takovém případě každý stupeň volnosti robota reprezentujeme jako další rozměr prostoru.

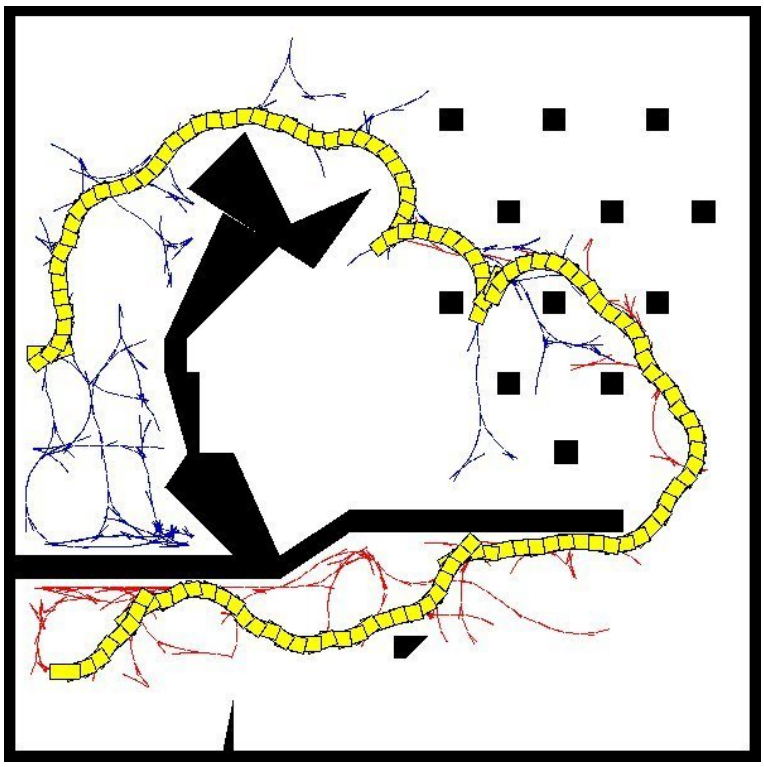

Obrázek 1.1: PA se používají i ke složitějším úkolům hledání cesty pro roboty s více stupni volnosti - například pro robotická ramena [8].

## 1.4 Problém stěhování klavíru

Pravděpodobnostní algoritmy lze použít při řešení typického úkolu hledání cesty, tzv. *problému stěhování klavíru* (*piano mover's problem*), který spočívá v nalezení volné cesty pro přesun klavíru z jednoho místa trojrozměrného prostoru s překážkami do druhého.

Za předpokladu, že známe topologii klavíru i daného prostoru s překážkami a že můžeme s klavírem libovolně manipulovat, lze polohu každého bodu klavíru jednoznačně určit pomocí šesti proměnných - souřadnic $x, y, z$ vybraného referenčního bodu klavíru a dále pak třemi údaji o natočení klavíru podle každé souřadné osy. Těchto šest proměnných tedy jednoznačně určuje konfiguraci robota a celý prostor s překážkami pak bude mít šest rozměrů [2].

Pro lepší představu je dobrá zjednodušená varianta problému stěhování klavíru - tzv. *problém stěhování pohovky* (*sofa mover's problem*). V tomto případě se jedná o stěhování pohovky po zemi, tedy pouze ve dvourozměrném prostoru. Konfigurace pohovky je tak určena dvěma souřadnicemi referenčního bodu plus jedním údajem o natočení pohovky. Tím dostaneme trojrozměrný prostor, který si lze představit jako plochu s překážkami „vytaženými nahoru" do třetího rozměru a každou pozici pohovky na podlaze také „vytaženou" do třetího rozměru, přičemž čímž „výše" pohovka je, tím více je otočená. V tomto trojrozměrném prostoru pak najdeme pravděpodobnostními algoritmy cestu a výsledek můžeme pro lepší představu promítnout zpátky do dvou rozměrů.



Obrázek 1.2: Ukázka průmětu autíčka ze 3D konfiguračního prostoru zpět do 2D prostoru, algoritmus RRT [4].

*Zobecněný stěhovací problém* (*generalized mover's problem*) zahrnuje oba předchozí problémy, protože se zabývá vyhledáním cesty pro robota s libovolným počtem stupňů volnosti.

Hledání cesty proto probíhá ve vícerozměrném prostoru a pro tyto výpočetně náročné úkoly jsou ideální právě pravděpodobnostní algoritmy [2].

## 1.5 Vlastnosti PA

Jedním z důležitých hledisek, kterým se hodnotí algoritmy, je jejich úplnost. Úplnost zaručuje, že algoritmus v konečném čase vždy najde řešení, pokud nějaké řešení existuje.

Pokud bychom chtěli pro hledání cesty použít nějaký úplný algoritmus, byli bychom kvůli výpočetní náročnosti v praxi omezeni zhruba deseti stupni volnosti robota [2]. Proto se k těmto složitejším výpočtům využívá pravděpodobnostních algoritmů, které jsou tzv. *pravděpodobnostně úplné*. Znamená to, že pokud existuje řešení (existuje cesta mezi startovní a cílovou konfigurací), algoritmus ji s určitou pravděpodobností vždycky najde – tedy čím déle necháme algoritmus běžet, tím je pravděpodobnější, že v hledání uspěje.

Výkonnost pravděpodobnostních algoritmů silně závisí na metodě rozhodující o tom, zda je daná náhodně vybraná konfigurace volná, nebo koliduje s překážkou. Výkonnost je také ovlivněna výběrem spojovací a vzdálenostní funkce.

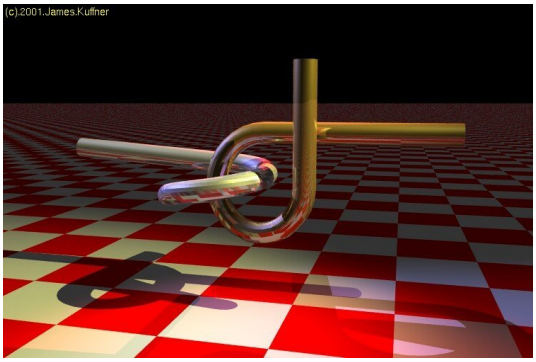
Obecně nejvýkonnějším pravděpodobnostním algoritmem je SRT, který kombinuje jednodotazové a vícedotazové algoritmy, avšak pro specifické účely můžou být vhodnější jiné algoritmy [2].

# Kapitola 2

# Využití pravděpodobnostních algoritmů

Pravděpodobnostní algoritmy mají velmi široké využití a dají se použít pro všechny roboty, kteří se pohybují ve známém stavovém prostoru.

PA se využívají pro řízení pohybu robotů ve dvourozměrném prostoru, ať už jde o neholonomické (všesměrové) roboty, nebo zjednodušené roboty typu auto. Při využití tzv. *control based* algoritmů je možné do hledání cesty zahrnout také dynamiku pohybu. Ve trojrozměrném prostoru lze pravděpodobnostními algoritmy řídit také pohyb robotů pohybujících se právě ve třech rozměrech, například humanoidních robotů nebo robotických ramen. [2].




Obrázek 2.1: Ukázka „rozpojovací“ úlohy – cílem je oddělit od sebe dva objekty ve tvaru písmene $\alpha$ [5].

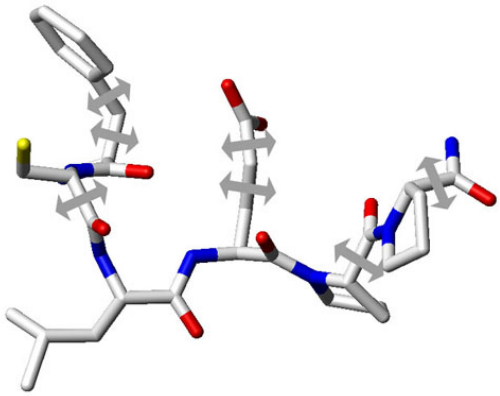
Typickým využitím PA jsou robotická ramena u výrobních linek. Ty mají často velmi mnoho stupňů volnosti (v [7] je zmíněn příklad robotické výrobní linky obsahující překážky složené z 43 530 polygonů a robotické rameno složené z 4 053 polygonů), takže PA jsou pro takové úlohy velmi vhodné.

Pomocí PA se řídí i současný pohyb několika robotů zároveň, například součinnost více robotických ramen u výrobní linky. V takových případech je možné hledat cestu všech robotů naráz (hledání tedy probíhá ve vícedimenzionálním prostoru daném součtem stupňů volnosti všech robotů), nebo pro každého robota zvlášť (po nalezení cest je pak nutné cestu každého robota korigovat, aby nekolidovala z ostatními) [2].

Další oblastí, kde PA nalézají uplatnění, jsou úlohy, při kterých robot manipuluje s předměty. U nich lze pohyb robota s předmětem nebo bez předmětu řídit i s uvažováním dynamiky pohybu (pomocí *control-based* algoritmů). Samozřejmě je také možné řídit robota, který musí přesunout více předmětů a je přitom v prostoru omezen překážkami i ostatními předměty, se kterými právě nemanipuluje [2].

PA jsou také schopny řešit „rozpojovací“ úlohy, jejichž typickým příkladem jsou dětské hlavolamy obsahující dvě samostatné, ale propojené části (viz obr. 2.1).

Dokážou také řešit manipulaci s deformovatelnými objekty. V takových případech je nutné deformovatelný objekt aproximovat několika stavy, které může nabývat a uvažovat přitom deformační energii (více viz [2]).



Obrázek 2.2: PA se využívají při modelování molekul – na obrázku ukázka molekuly, jejíž pohyblivé části jsou znázorněny šipkami [1].

Významným přínosem PA je *zarovnávání* (*dokování*) objektů. Příkladem může být zachycení družice na oběžné dráze pomocí robotického ramena raketoplánu a následné uložení družice do nákladového prostoru [2].

K podobným úkolům se PA používají v biochemii, kde se pomocí nich zkoumají možnosti vazeb u složitých molekulárních struktur nebo také pohyb jednotlivých částí molekul při návratu molekuly do své výchozí konfigurace (viz obr. 2.2) [2].

# Kapitola 3

# Teorie implementace PA

Pravděpodobnostní algoritmy jsou složeny z několika základních metod, které zajišťují práci s grafem, jako je přidávání uzlů, hran a určování vzdáleností mezi těmito uzly.

## 3.1 Vzorkování náhodných bodů z konfiguračního prostoru

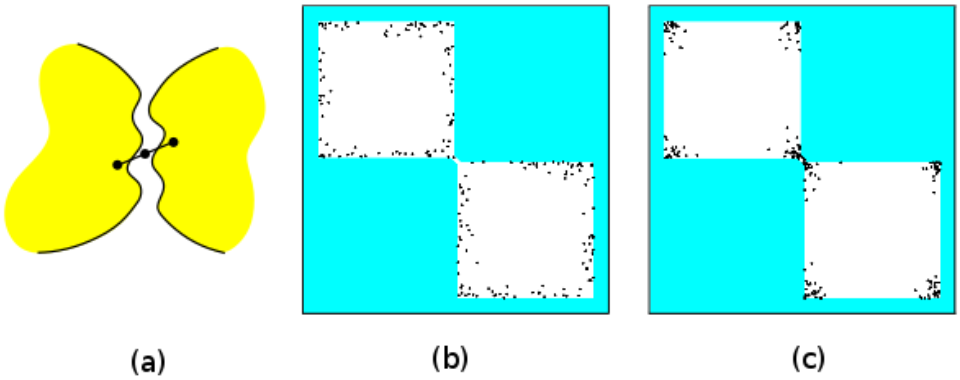
Výběr konfigurací z konfiguračního prostoru je velmi důležitý, neboť na něm závisí budoucí pokrytí prostoru grafem.

Nejjednodušším, avšak dostačujícím způsobem výběru náhodných konfigurací je obyčejné generování náhodných hodnot s rovnoměrným rozložením pravděpodobnosti. Pro každý stupeň volnosti generujeme náhodnou hodnotu z jejího povoleného rozsahu. Omezením jsou okraje prostoru, ve kterém se robot pohybuje, nebo rozsah úhlů, ve kterém se může natáčet samotný robot nebo jeho klouby [2]. U úhlů natočení je třeba vyloučit případné duplicitní konfigurace s určitými hodnotami (například pokud je robot ve 2D prostoru oválný a nepotřebujeme rozlišit jeho přední a zadní část, pracujeme s jeho natáčením pouze mezi 0° a 180°).

Ačkoliv vzorkování s rovnoměrným rozložením pravděpodobnosti funguje vždy, v některých specifických situacích je málo efektivní, protože může být potřeba vzorkovat stavový prostor v některých určitých hustěji. Takovým případem je hledání cesty v prostoru, který obsahuje tzv. *úzké průchody* (*narrow passages*) – ty vznikají mezi překážkami, které od sebe odděluje úzký koridor volného prostoru. Pokud robot musí nutně tímto úzkým průchodem projít, je rovnoměrné rozložení pravděpodobnosti často nedostatečné, protože je malá pravděpodobnost, že bude zvolena náhodná konfigurace právě z úzkého průchodu (případně že hrana mezi dvěma body z opačných stran úzkého průchodu nebude kolidovat s překážkami) [2].

Nedostatky vzorkování v okolí úzkých průchodů se řeší dvěmi způsoby: filtrovacími nebo retrakčními algoritmy. Ty první se snaží pomocí náhodných vzorků zachytit různými technikami volný prostor v okolí úzkých průchodů, zatímco ty druhé pracují na principu „zmenšení“ překážek a tím zvětšení úzkých průchodů a následné úpravě grafu podle původni velikosti překážek [2].

Jedním z fitrovacích algoritmů je RBB – Randomized Bridge Builder [3]. Ten vychází z předpokladu, že úzké průchody jsou oblasti, kterými robot může projít jen v jednom směru a nesmí se odchylovat do stran kolmo k tomuto směru. Využívá takzvaný „most“, což je úsečka mezi dvěma překážkami, jejíž konce („nosníky mostu“) se nachází uvnitř překážek a středový bod úsečky leží ve volném prostoru. RBB tedy generuje vždy dva náhodné body




Obrázek 3.1: Ukázka filtrovacích algoritmů pro vzorkování v okolí úzkých průchodů. Most přes úzký průchod (a) a porovnání bodů vygenerovaných ve volném prostoru pomocí Gaussovského vzorkování (b) a algoritmu RBB (c) [3].

s rovnoměrným rozložením pravděpodobnosti. Pokud oba dva body kolidují s překážkami, algoritmus otestuje bod uprostřed jejich spojnice a pokud ten leží ve volném prostoru, přidá jej do grafu.

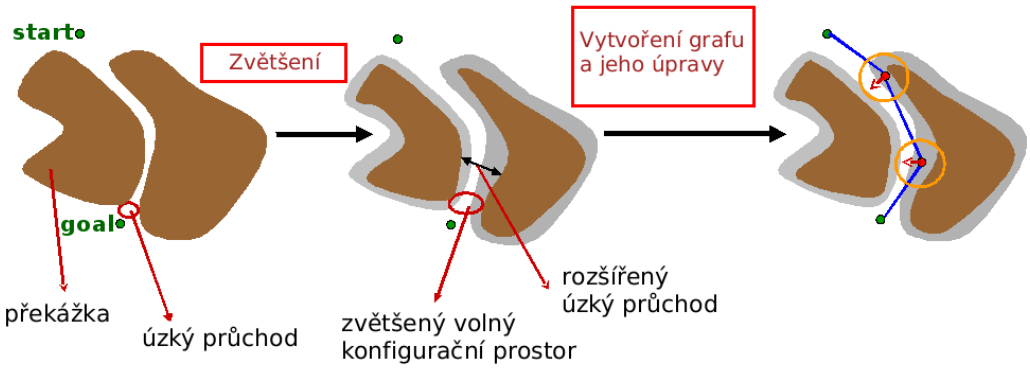
**Algoritmus 3.1 RBB – Randomized Bridge Builder [3].**

*1:* ***repeat***
*2:* *náhodně vyber $x$ z konfig. prostoru $C$ s rovnoměrným rozložením pravděpodob.*
*3:* ***if*** $x \notin C_{free}$ ***then***
*4:* *náhodně vyber $x'$ z okolí $x$ pomocí pravděpodob. funkce $\lambda_x$*
*5:* ***if*** $x' \notin C_{free}$ ***then***
*6:* $p \leftarrow$ *bod ve středu úsečky* $x, x'$
*7:* ***if*** $p \in C_{free}$ ***then***
*8:* *přidat $p$ do grafu jako nový uzel*
*9:* ***end if***
*10:* ***end if***
*11:* ***end if***

Další možností vzorkování v okolí úzkých průchodů je využití tzv. *Gaussovského vzorkování* [2]. To pracuje na principu přidávání do grafu pouze takových bodů, u kterých je pravděpodobné, že se nachází u hranic překážek. Nejprve je vygenerová náhodný bod $q_1$ s rovnoměrným rozložením pravděpodobnosti a poté je určena délka kroku *step* s gaussovským rozložením pravděpodobnosti. Ve vzdálenosti *step* od $q_1$ je pak v náhdném směru vygenerován nový bod $q_2$. Pokud jsou oba body $q_1$ i $q_2$ uvnitř překážky, nebo naopak oba ve volném prostoru, algoritmus do grafu nepřidá ani jeden z nich a pokračuje dále. Pouze v případě, že jeden z bodů se nachází uvnitř překážky a druhý ve volném prostoru, je ten z volného prostoru přidán do grafu.

Mezi výkonné retrakční způsoby hledání cesty přes úzké průchody patří SSRP (Small-Step Retraction Planner, [7]). Ten nejdříve provede „zmenšení překážek“, čímž rozšíří volný konfigurační prostor včetně úzkého průchodu. Poté snáze vyhledá cestu v tomto prostoru a na závěr provede upravení nalezené cesty tak, aby vyhovovala původnímu volnému pro-

storu. Vlastní upravování cesty se provádí buď při umisťování bodů do grafu („pesimistická“ varianta), nebo až po nalezení cesty („optimistická“ varianta), přičemž druhá varianta je rychlejší, ale není pravděpodobnostně úplná. Jako nejvhodnější se tedy jeví využít „optimistickou“ variantu a až v případě jejího neúspěchu použít „pesimistickou“.




Obrázek 3.2: Algoritmus SSRP je určen pro hledání cesty v prostoru s úzkými průchody. Obrázek zachycuje fáze tohoto algoritmu – 1. zmenšení prostoru překážek, $2+3$. vytvoření grafu a jeho úprava pro původní prostor [7].

Problémem pro rovnoměrné rozložení pravděpodobnosti nemusejí být jen úzké průchody, ale také větší množství obyčejných překážek. V takovém případě je vhodné použít modifikaci PRM, algoritmus OBPRM.

Algoritmus OBPRM (Obstacle PRM, [2]) se snaží do grafu přidávat body na okrajích překážek, protože ty hrají významnou roli při zachycení volného konfiguračního prostoru. Dosahuje toho nálsedujícím způsobem: pro každou náhodně vygenerovanou konfiguraci $q_{in}$, která koliduje s překážkou, najde v náhodném směru volnou konfiguraci $q_{out}$. Poté dělením intervalu mezi $q_{in}$ a $q_{out}$ najde volnou konfiguraci $q$ co nejblíže od okraje překážky a právě tu přidá dá grafu.

Existují i další složitější postupy pro efektivní vzorkování prostorů s úzkými průchody, například pomocí obecných Voronoi diagramů (GVD, Generalized Voronoi Diagrams) [2].

## 3.2 Výběr nejbližších uzlů z grafu

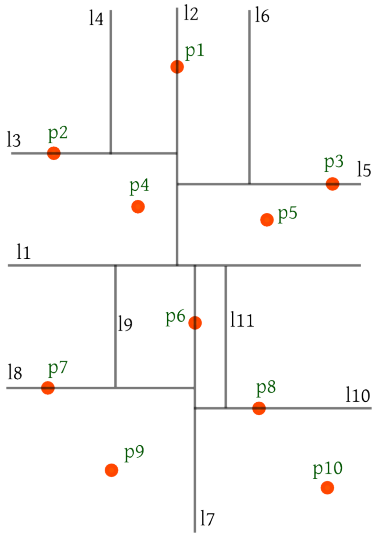
Pro spojování uzlů do grafu je důležitá funkce, která vyhledá $n$ nejbližších uzlů ke každému zadanému uzlu. Pojem *nejbližší* závisí na metrice daného prostoru a bude dále diskutován v části 3.3 o vzdálenostních funkcích.

Nejbližší body můžeme hledat buď *prostým měřením vzdálenosti* všech bodů, nebo některými sofistikovanějšími postupy.

První případ je jednoduchý - změříme od vybraného bodu vzdálenost ke všem bodům grafu a zapamatujeme si přitom $n$ nejbližších bodů. Tato metoda je ale pochopitelně velmi

neefektivní a se vzrůstajícím počtem bodů v grafu nebo se vzrůstajícím počtem dimenzí zpomaluje celý pravděpodobnostní algoritmus [2].

Sofistikovanější metodou je využití kd-stromu. Kd-strom je vytvářen tak, že se rekurzivně dělí $d$-rozměrný graf na dvě části se stejným počtem bodů do té doby, než je v každé části takto rozděleného prostoru právě jeden bod. Takto vzniklá struktura je uložena do binárního stromu, v němž jsou nekoncové uzly tvořeny dělícími plochami a listy stromu tvoří body [2].




Obrázek 3.3: Kd-strom – ukázka dělení dvourozměrného prostoru s body a jemu odpovídající binární strom.

Kd-strom má tu výhodu, že celý graf najednou zpracuje a uloží do struktury stromu (s časovou složitostí $O(d * n \log n)$), ve které pak všechna vyhledávání nejbližších bodů probíhají s časovou složitostí $O(n^{1-1/d} + m)$, kde $d$ je počet dimenzí prostoru, $n$ počet bodů v něm a $m$ je počet nejbližších sousedů, které chceme vybrat [2].

## 3.3 Vzdálenostní funkce

Vzdálenostní funkce (označovaná jako *dist*, $C \times C \leftarrow \Re^+ \cup 0$), je důležitá pro výběr nejbližších uzlů grafu. Základním přístupem je zobrazení konfigurací robota ve vícerozměrném prostoru a spočtení jejich euklidovské vzdálenosti:

$$dist(q', q'') = \mid emb(q') - emb(q'') \mid$$

(funkce *emb()* je euklidovská vzdálenost bodu od počátku souřadné soustavy) [2]

Pro statické roboty, kteří nemají žádné klouby, se vzdálenost může vyjádřit pomocí následující funkce:

$$dist(q', q'') = w_t \times \mid X' - X'' \mid + w_r \times f(R', R'')$$

($\mid X' - X'' \mid$ je euklidovská vzdálenost dvou konfigurací, má tedy význam translace robota; $f(R', R'')$ je funkce, která vrací přibližnou „rotační“ vzdálenost - například délku křivky, po které se pohybuje vybraný bod robota mezi dvěma konfiguracemi; $w_t$ a $w_r$ jsou

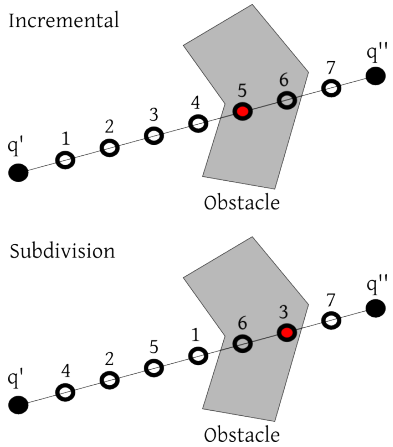
pak koeficienty upravující vzájemnou významnost složek translace a rotace pro celkovou vzdálenost) [2].

## 3.4 Spojování dvou konfigurací

Funkce spojování dvou konfigurací ($\Delta$, *local planner*) má za úkol v grafu spojit dvě konfigurace robota, přičemž jím nalezená cesta nesmí kolidovat s překážkami.

Hlavním kritériem spojovacích funkcí je jejich výkonnost v tom smyslu, že jsou schopny účinně najít volnou cestu mezi dvěma konfiguracemi. Nejjednodušší možnou spojovací funkcí je spojování dvou konfigurací v prostoru úsečkou – ta ale samozřejmě není příliš výkonná, protože cestu nenajde, pokud se mezi danými konfiguracemi nachází překážka. Pokud tedy požadujeme výkonnou spojovací funkci, můžeme zvolit některý z jednodotazových pravděpodobnostních algoritmů, jako jsou EST nebo RRT [2].

Proti výkonnosti spojovacích funkcí působí jejich časová a výpočetní náročnost. Jednodotazové algoritmy sice najdou cestu mezi dvěma body i v komplikovanějších případech, ale zase jsou oproti pouhému spojování úsečkou pomalejší a náročnější, takže v praxi musíme zvolit, která vlastnost je pro danou implementaci pravděpodobnostního algoritmu významnější. PRM algoritmus využívá spojování úsečkou, SRT zase jednodotazové algoritmy.




Obrázek 3.4: Srovnání dvou spojovacích funkcí - inkrementální a rekurzivního dělení. Zatímco první zjistí po pěti krocích, že cesta mezi body $q'$ a $q''$ koliduje s překážkou, druhá to zjistí už po třech krocích.

U spojovacích funkcí sledujeme také další dvě důležité vlastnosti – zda je symetrická a deterministická. Symetrický algoritmus zaručuje, že pro hledání cesty z bodu A do bodu B najde stejnou cestu jako pro hledání z B do A. V praxi jde o obvyklý případ, ale s nesymetrickými algoritmy se můžeme také setkat – například pokud máme robota, který má asymetrický půdorys (kupříkladu vyčnívající rameno na jedné straně), tak projde kolem překážky jen v jednom směru. Pokud potřebujeme v takovém případě uchovávat informace o možném spojení dvou vrcholů grafu, musí být tento graf orientovaný.

Deterministický algoritmus najde při každém svém volání pro dva body A, B stejnou cestu, zaručuje tedy časovou invarianci. Díky této vlastnosti nemusíme uchovávat dodatečnou informaci o cestě (hraně) mezi dvěma uzly grafu.

Spojovací funkce musí nějakým způsobem interpolovat všechny konfigurace mezi dvěma krajními a ty kontrolovat, zda nekolidují s překážkami. Algoritmy EST a RRT, které se dají použít jako spojovací funkce, budou podrobně rozebrány v samostatných kapitolách, ale i základní funkci spojující dvě konfigurace úsečkou lze implementovat více způsoby - inkrementálně a rekurzivním dělením [2].

Obě dvě funkce spojující konfigurace úsečkou nejprve danou úsečku mezi body $q'$ a $q''$ diskretizují na posloupnost hodnot $q_1, q_2 \dots q_n$ – liší se pouze ve způsobu této diskretizace. Obě dvě potom interpolují odděleně translační a rotační složku konfigurace v bodech $q_1, q_2 \dots q_n$ a v nich potom testují konfigurace robota, zda jsou volné, nebo kolidují s překážkou. Inkrementální verze postupně interpoluje všechny možné konfigurace mezi počáteční a koncovou konfigurací s určitou délkou kroku. Verze používající rekurzivní dělení pracuje následovně: rozdělí úsečku mezi počáteční a koncovou konfigurací na polovinu, otestuje prostřední konfiguraci na kolizi s překážkami a rekurzivně se zavolá pro obě poloviny rozdělené úsečky. Algoritmus končí ve chvíli, kdy narazil na konfiguraci kolidující s překážkou, nebo pokud všechna rekurzivní volání skončí bez kolizí s úsečkou kratší než je délka kroku [2].

U obou metod je důležitá volba délky kroku. Měla by být v rozumné míře pokud možno co nejmenší, což sice zvyšuje výpočetní náročnost algoritmu, ale pokud bychom zvolili větší délku kroku, metoda by nefungovala správně pro překážky menší než rozlišovací schopnost metody. Mohlo by se totiž stát, že mezi dvěma sousedními testovanými konfiguracemi by se vyskytovala překážka menší než délka kroku a metoda by ji vůbec nedetekovala.

Rekurzivní dělení bývá obecně efektivnější, protože dříve odhalí kolize. Je to z toho důvodu, že delší cesty mají větší pravděpodobnost, že budou kolidovat s překážkou - a subdivision pracuje s delšími cestami, zatímco inkrementální přístup postupuje po krátkých úsecích [2].

## 3.5 Zkracování a vyhlazování cesty (postprocessing)

Když kterýmkoliv pravdpodobnostním algoritmem nalezneme cestu mezi počátečním a cílovým bodem, můžeme tuto cestu dále upravovat, aby vyhovovala určitým požadavkům.

Nejčastějším požddadavkem je nalezenou cestu zkrátit. Nesousední body cesty je totiž často možné přímo spojit, ale během konstrukce grafu spojeny nebyly z důvodu, že jsou příliš daleko na to, aby byly navzájem zahrnuty do seznamu svých nejbližších bodů. Kratší cestu získáme v případě, že se nám podaří spojit dva nesousední body cesty. Existuje více variant, jak vybírat dva nesousední body cesty - například vybrat dva náhodné body cesty nebo tzv. *hladový přístup* [2].

Hladový přístup funguje tak, že se snažíme z počátečního bodu postupně přímo (úsečkou, pokud tedy nepracujeme s jinou spojovací funkcí) spojit cílový bod a pokud neuspějeme, tak předposlední bod atd. Pokud nelze počáteční bod spojit s žádným jiným uzlem, než s jeho přímým následníkem v cestě, pokračujeme stejným způsobem dále, jen se pokoušíme spojovat body směrem od cíle s tímto následníkem počátku atd. [2]

Dalším požadavkem je vyhladit nalezenou cestu. K tomu je vhodné použít body nalezené cesty jako řídící body pro spline a poté testovat body na takto vzniklé křivce, zda nekolidují s překážkami [2].

Zkracování i vyhlazování cesty zpomaluje celý pravděpodobnostní algoritmus, takže je vhodnější tyto optimalizace provádět až po nalezení cesty a nikoliv už během vytváření grafu [2].

# Kapitola 4

# Popis implementace appletů

Součástí této bakalářské práce je také vytvoření java appletů demonstrujících jednotlivé pravděpodobnostní algoritmy.

V této kapitole bude popsáno ovládání appletů a hlavně implementace základní funkčnosti appletů – řízení simulace, obecné funkce pro práci se simulačním prostorem, grafické zobrazení simulace a popis základních nastavení. Vzhledem k využití dědičnosti i kompozice je tato základní funkčnost využita u všech appletů. Detaily implementace jednotlivých appletů jsou popsány samostatně v kapitolách u příslušných algoritmů.

Programová dokumentace bakalářské práce vygenerovaná nástrojem Javadoc je uložena na přiloženém CD.

## 4.1 Ovládání appletů

Applety jsou vertikálně rozvrženy na tři hlavní části – nahoře ovládací prvky, uprostřed zobrazovací plocha simulačního prostoru a dole pak stavová lišta, která zobrazuje informace o probíhající simulaci nebo tipy pro ovládání appletu.

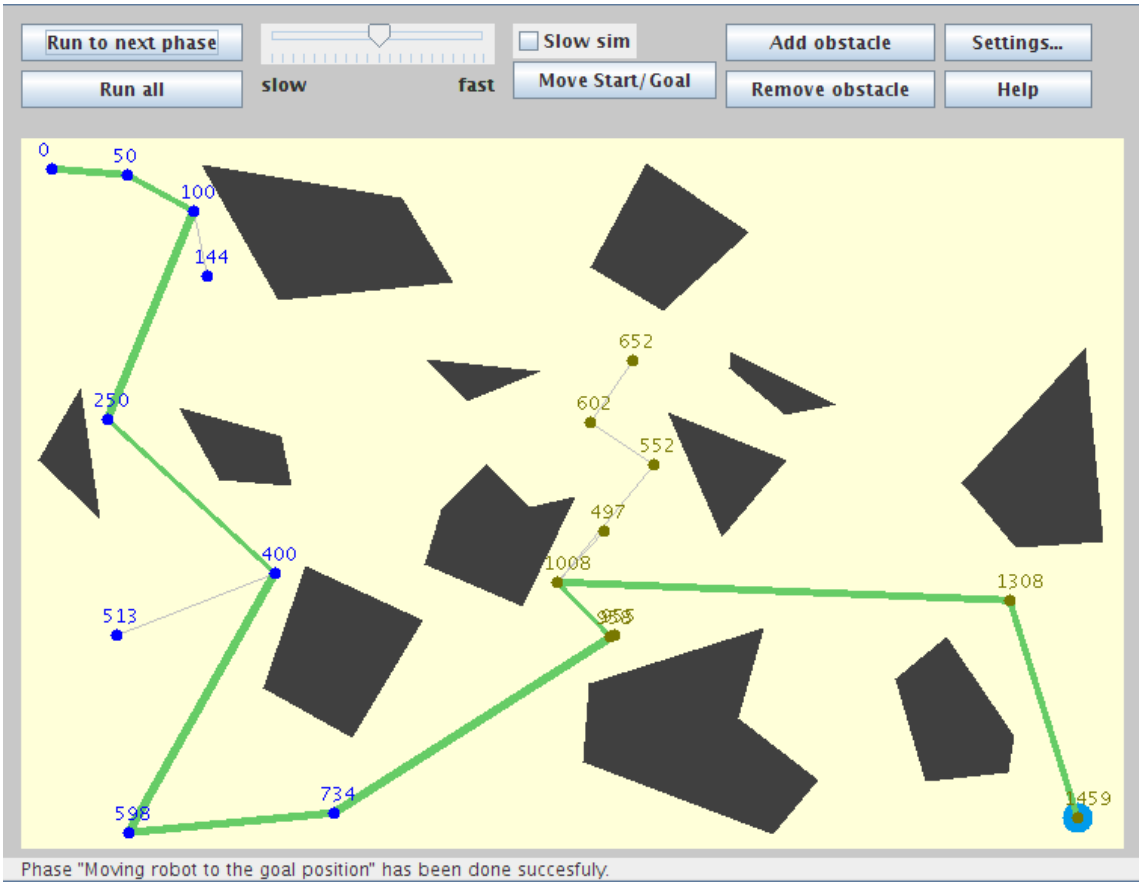
Z *ovládacích tlačítek* jsou první čtyři určeny k řízení simulace – *Run to next phase/Stop*, *Run all/Reset*, zatržítko *Slow sim* a jezdec pro ovládání rychlosti. Vpravo se nachází tlačítko *Settings* pro nastavení appletu (to je přístupné pouze před začátkem nebo po ukončení celé simulace) a *Help* pro nápovědu. Během simulace se v závislosti na průběhu objevují mezi jednotlivými simulačními fázemi další ovládací tlačítka, například pro úpravu překážek nebo editaci grafu.

Uprostřed appletu je světle vykreslena **simulační plocha** se šedě zbarvenými překážkami a případně s vytvářenými grafy. Pokud je to možné, tak se u bodů (uzlů) grafu zobrazují čísla vyjedřující vzdálenost od počátečního bodu. Je-li zatržena volba *Slow sim*, plocha se průběžně překresluje v souladu s informacemi zobrazovanými ve stavové liště.

Ve **stavové liště** se zobrazují aktuální informace o průběhu simulace nebo o možnostech nastavení a ovládání appletu.

Simulaci lze spustit buď tlačítkem **Run to next phase**, které provede simulaci pouze jedné fáze (viz 4.3) a pak ji zastaví, nebo tlačítkem **Run all**, které provede simulaci až do konce v případě, že v žádné fázi nenastala chyba. Rychlost simulace lze ovlivňovat jezdcem a také zatžítkem *Slow sim*, po jehož deaktivaci se přestanou zobrazovat jednotlivé průběžné kroky simulace.

Během simulace je možné její běh zastavit tlačítkem *Stop*, které ukončí právě probíhající




Obrázek 4.1: Uživatelské rozhraní appletů. Nahoře ovládací prvky, uprostřed zobrazovací plocha simulačního prostoru a dole stavová lišta zobrazující informace o simulaci.

fázi. Mezi fázemi je aktivní tlačítko *Reset* (nachází se na místě tlačítka *Run all*), které resetuje celou simulaci, takže je možné ji začít znovu od začátku.

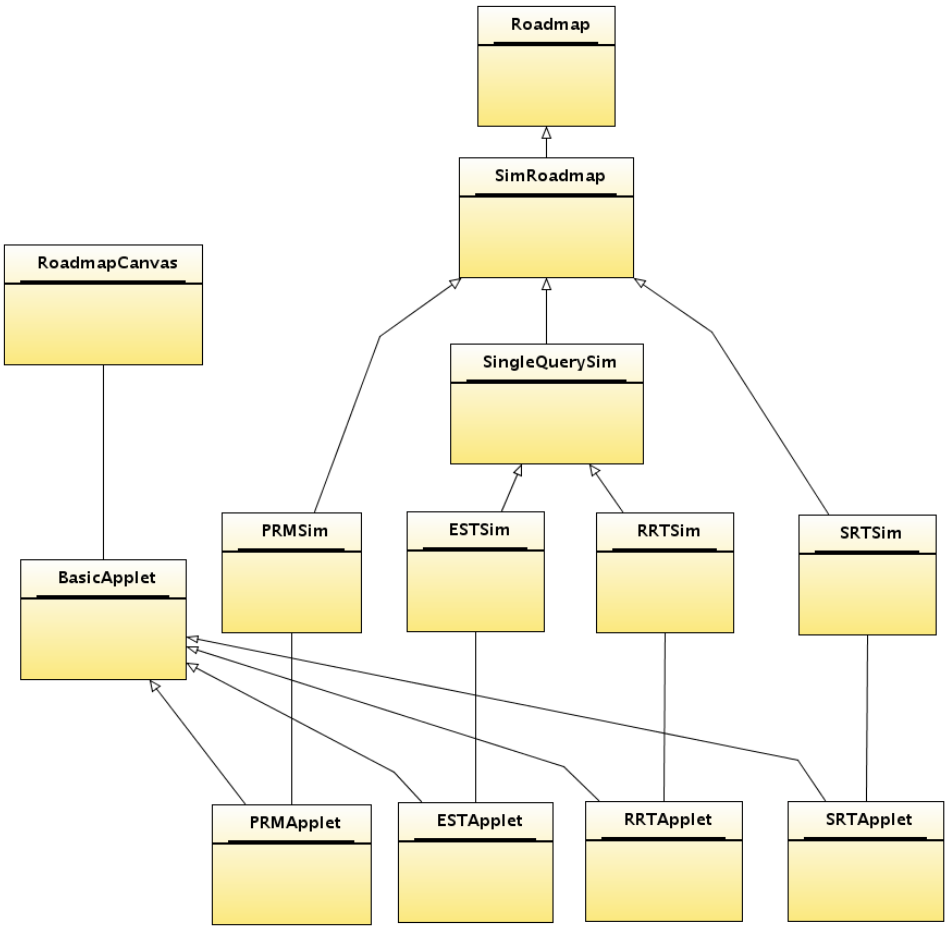
Kromě předem nastavených parametrů před začátkem simulace (pomocí tlačítka *Settings*) umožňují applety mezi některými fázemi tyto akce:

- **Změnit polohu startu nebo cíle (Move Start/Goal)** – umožňuje kliknutím levým tlačítkem myši do volného prostoru simulační plochy změnit polohu startovního bodu robota. Kliknutím pravým tlačítkem lze změnit polohu cíle. Tuto akci je možné provádět pouze před zahájením a po skončení simulace.
- **Přidat překážku (Add Obstacle)** – umožňuje do simulační plochy vložit nový polygon překážky. Kliknutím do simulační plochy je přidán nový bod polygonu, dvojklikem se polygon uzavře (spojením posledního bodu s prvním) a je přidán do prostoru jako nová překážka. Tuto akci je možné provádět pouze před zahájením a po skončení simulace.
- **Odstranit překážku (Remove Obstacle)** – umožňuje smazat libovolnou překážku kliknutím kamkoliv do této překážky v simulační ploše. Tuto akci je možné provádět pouze před zahájením a po skončení simulace.

- **Přidat bod do grafu (Add Point)** – kliknutím do simulační plochy je přidán do grafu nový bod, pokud leží ve volném konfiguračním prostoru. Tuto akci je možné provádět pouze po ukončení fáze generování nových bodů, která se vyskytuje u více-dotazových algoritmů – PRM a SRT.
- **Vyhladit nalezenou cestu (Smooth Path)** – provede vyhlazovací algoritmus pro zkrácení nalezené cesty (viz 3.5). Tuto akci je možné provádět až poté, kdy byla cesta od startu k cíli úspěšně nalezena.

## 4.2 Hierarchie tříd

Všechny třídy, vytvořené v rámci této bakalářské práce, se nacházejí v balíku SBAApplets (Sampling-Based Algorithms Applets) a lze je zjednodušeně rozdělit do těchto kategorií:




Obrázek 4.2: Schéma zachycující hierarchii nejdůležitějších tříd implmentace appletů. Šipky naznačují dědičnost, spojovací čáry kompozici.

- **Jednotlivé applety** – tyto třídy spojují všechny níže uvedené třídy do konkrétních appletů a umožňují spuštění simulace.

Do této kategorie patří *PRMApplet*, *ESTApplet*, *RRTApplet* a *SRTApplet*.

- **Jednotlivé simulační třídy** – jedná se o třídy, které implementují konkrétní pravděpodobnostní algoritmy. Tyto třídy budou popsány podrobně u jednotlivých PA v příslušných kapitolách této práce.

  Do této kategorie patří *PRMSim*, *ESTSim*, *RRTSim* a *SRTSim*.

- **Třídy pro nastavení jednotlivých appletů** – vytvářejí grafické uživatelské rozhraní pro nastavování parametrů jednotlivých appletů s pravděpodobnostními algoritmy. Dále také umožňují zobrazení nápovědy ke konkrétním nastavením.

  Do této kategorie patří *PRMSettingsPanel*, *ESTSettingsPanel*, *RRTSettingsPanel* a *SRTSettingsPanel*.

- **Základní simulační třídy** – tyto třídy poskytují obecné funkce pro práci se simulačním prostorem (například pro práci s grafy) a také funkce pro řízení simulace. Jsou využívány všemi konkrétními simulačními třídami jednotlivých PA.

  Do této kategorie patří třídy *Roadmap* a *SimRoadmap* a také *SingleQuerySim*.

- **Pomocné třídy pro práci s grafy** – jsou využívány k uchovávání informací o grafech, hranách a uzlech grafu. Jedná se o třídy *Graph*, *Edge* a *Vertex*.

- **Základní vykreslovací třídy** – do této kategorie patří jediná třída – *RoadmapCanvas*. Ta poskytuje funkce pro vykreslování grafů, překážek a dalších objektů v simulačním prostoru. Dále také umožňuje grafickou editaci prostoru, jako například přesun startovní pozice nebo přidávání nebo uzlů do grafu.

- **Třídy pro základní funkčnost appletů** – patří sem třída *BasicApplet*, která poskytuje základní kostru pro všechny applety. Dále sem patří *UpdateThread*, která ve zvláštním vlákně překresluje applet během simulace a také malá komponenta sloužící pro výběr sady překážek – *ObstacleSet*.

- **Třídy pro nápovědu** – jedná se o třídy využívané v uživatelském rozhraní appletů pro zobrazení nápovědy. Patří sem *HelpPanel*, *HelpTexts* a *HelpButton*.

Applety dědí své rozvržení a ovládací prvky ze třídy BasicApplet, vykreslovací funkce přebírají z RoadmapCanvas, obecné simulační funkce z Roadmap a další simulační funkce závislé na vykreslování ze třídy SimRoadmap. Schéma těchto závislostí zachycuje diagram 4.2.

S přihlédnutím k širší možnosti využití jsem applety včetně dokumentace vytvořil v anglickém jazyce.

Velká většina algoritmů zabudovaných v appletech je nevhodná k vlastní simulaci pravděpodobnostních metod, protože vzhledem k povaze zadání bylo nutné během provádění algoritmů zobrazovat informace o průběhu simulace a průběžně simulaci vizualizovat.

## 4.3 Implementace řízení simulace

Nejdůležitější součástí appletů jsou samozřejmě třídy zajišťující běh simulace.

Všechny applety pracují na principu rozdělení algoritmů do několika samostatných fází, jako je například vytváření grafu, vyhledávání cesty v něm apod. Seznam těchto fází je uveden u každého apletu v nápovědě (tlačítko *Help*).

Při inicializaci appletu se vytvoří nové simulační vlákno konkrétního pravděpodobnostního algoritmu – tedy *PRMSim*, *ESTSim*, *RRTSim* nebo *SRTSim*. Toto vlákno je po svém startu uspáno a probuzeno může být až stisknutím stačítka *Run all* nebo *Run to next phase*.

Po takovéto aktivaci je proveden jeden krok simulace (pokud bylo spuštění vyvoláno tlačítkem *Run to next phase* nebo pokud byl první krok neúspěšný), nebo celá simulace.

Spuštění simulačního kroku probíhá konkrétně následovně: po aktivaci simulačního vlákna je nastaven příznak probíhající simulace a počítadlo aktuálně probíhající fáze je zvětšen o jedna. Poté je zavolána funkce *executePhase()*, která právě na základě počítadla probíhající fáze volá funkce odpovídající dané fázi simulace, jako například generování nových bodů u PRM, spojování stromů u SRT, reset simulace před první fází a podobně.

Informace o úspěšném provedení funkce volané pro danou fázi je předána zpět funkci *executePhase()* a ta ji dále předá do simulačního vlákna. To na jejím základě může umožnit spuštění další fáze, nebo vlákno znovu uspí a čeká na případné znovuspuštění simulace novým probuzením tohoto vlákna.

Jednotlivé fáze simulace mají samozřejmě vliv i na grafické uživatelské rozhraní appletu, konkrétně na zobrazování různých tlačítek pro editaci simulačního prostoru, jak bylo popsáno v části 4.1.

Během simulace většina funkcí vykonávajících příslušné fáze umožňuje zobrazovat jednotlivé probíhající kroky (pomocí zatržítka *Slow sim*), jako například přidání bodu do grafu a podobně. V průběhu těchto funkcí je proto nutné zobrazovat informace do stavového řádku appletu a na standardní výstup a hlavně průběžně překreslovat plátno se simulačním prostorem. K tomuto všemu slouží funkce *updateCanvas()* ze třídy *SimRoadmap*, která vytváří nové vlákno třídy *UpdateThread*, jež se postará o překreslení plátna (případně zadané části plátna) a aktualizaci textu ve stavové liště. Po vytvoření tohoto vlákna je samotná simulace na několik milisekund uspána v závislosti na nastavené simulační rychlosti.

Aby bylo možné simulaci kdykoliv v jejím průběhu zastavit, musí simulační funkce pravidelně testovat příznak zastavení. Není totiž možné simulaci zastavit v libovolném okamžiku, protože by mohla být narušena konzistance simulačních dat.

## 4.4 Základní simulační funkce

Základní simulační funkce jsou využívány všemi pravděpodobnostními algoritmy.

Ve třídě *Roadmap* se nachází ty pomocné funkce pro simulaci PA, které jsou nezávislé na prezentačním výstupu. Neobsahují tedy žádné průběžné zobrazování stavu simulace uživateli ani funkce pro řízení simulace. Díky tomu je lze třídu *Roadmap* přímo využít i k případné jiné implementaci PA mimo tyto demonstrační applety.

Nejdůležitější funkce ze třídy *Roadmap* poskytují základní abstrakci pro práci s grafy. Patří sem booleovské funkce pro zjištění, zda zadaný bod leží ve volném konfiguračním prostoru (*isPointCollisionFree()*) nebo zda ve volném konfiguračním prostoru leží celá zadaná hrana grafu (*isEdgeCollisionFree()*). Dále *Roadmap* poskytuje funkce pro výběr uzlů nejbližších zadanému bodu (*selectNeighbors()*), pro výběr sousedních uzlů grafu spojených se zadaným uzlem hranami (*pointGraphNeighbors()*), pro vyhledání cesty v grafu pomocí Dijkstrova algoritmu (*dijkstraSearch()*), pro vytváření cesty (*addPathPoint()*), pro základní vyhlazení cesty „hladovým přístupem“ (*smoothPath()*) a také pro reset všech simulačních proměnných (*resetSimulation()*).

Třída *SimRoadmap* rozšiřuje třídu *Roadmap* o další pomocné simulační funkce, ale tyto funkce už závisejí na prezentačním výstupu. Jedná se o funkce pro řízení simulace, pro vykonávání některých simulačních fází a také o další pomocné funkce pro práci se simulačním prostorem.

Řízení celé simulace (viz 4.3) má v *SimRoadmap* na starosti metoda *run()*, ve které běží v nekonečné smyčce posloupnost volání úkonů k provádění jednotlivých fází. Důležitá je také již zmíněná funkce *updateCanvas* pro aktualizaci zobrazení simulace.

Funkce pro vykonávání simulačních fází implementuje *SimRoadmap* dvě - *moveRobot()* pro pohyb robota po cestě od počátečního do cílového bodu a *smoothPath()*, která překrývá funkci ze třídy *Roadmap* a oproti ní umožňuje zobrazování průběžných informací při vyhlazování cesty.

Mezi funkce třídy *SimRoadmap* pro práci se simulačním prostorem lze zařadit funkci *moveRobotOnEdge()*, která zajišťuje pohyb robota po jedné hraně grafu a jeho průběžné překreslování, dále pak funkci *setObstacleSet()* pro přidání předpřipravené sady překážek do simulace, *setWaitTime()* pro nastavení rychlosti simulace a také *resetSimulation()*, která překrývá metodu z *Roadmap*. Třída *SimRoadmap* také obsahuje několik metod pro práci se zvýrazněnými částmi grafu - *setHighlightedPoint()*, *resetHighlightedPoint()* apod.

## 4.5 Grafické zobrazení simulace

Cílem vytvořených appletů je názorně vizualizovat principy pravděpodobnostních algoritmů, takže grafické zobrazení probíhající simulace je velmi důležité.

Všechny prvky simulace se zobrazují v simulačním prostoru, který je reprezentován třídou *RoadmapCanvas*, odvozenou od plátna (*Canvas*) z balíku *java.awt*. Při svém překreslování tedy *RoadmapCanvas* vykresluje překážky (tmavě šedou barvou), robota (světle modrou), cílovou pozici (zelenou, označena písmenem „G“), v případě potřeby mřížku s hodnotou funkce hustoty (pouze pro algoritmus EST) a hlavně grafy navzorkovaného prostoru pravděpodobnostních algoritmů. Tyto grafy jsou reprezentovány uzly (vertices, vykresleny tmavě modrou nebo tmavě hnědou barvou) a hranami (edges, vykresleny světle šedou) a pokud v simulaci již lze určit vzdálenost bodů od startovní pozice robota (v případě dvoustromových verzí algoritmů EST a RRT též od cílové), zobrazuje se u bodů grafu číselná informace o jejich vzdálenosti právě od počáteční (resp. cílové) pozice.

Kromě základní vykreslovací barvy je u většiny těchto prvků umožněno jejich zvýrazněné zobrazení jinou barvou, což je užitečné právě pro zpomalenou vizualizaci průběhu simulace algoritmů.

Třída *RoadmapCanvas* kromě vykreslování také poskytuje funkce pro úpravu simulačního prostoru pomocí myši. Mezi tyto funkce patří obsluha přidávání a mazání překážek, změna počáteční a cílové pozice a ruční přidávání bodů do grafu. Podrobněji o těchto akcích pojednává část 4.1.

## 4.6 Základní nastavení simulace

Všechny applety lze po stisknutí tlačítka *Settings* před zahájením a po ukončení simulace konfigurovat.

Většina nastavení je specifická pro každý applet, ale následujících pět proměnných je společných všem:

- **Počet bodů v grafu** (Points to generate in the roadmap) – určuje, kolik bodů se má snažit simulace vytvořit ve volném konfiguračním prostoru.

  Pokud tato hodnota není dosažena a další body se už se nedaří vytvořit, pak je daná fáze simulace neúspěšně ukončena. Při zadání nízké hodnoty vznikají „řídké“ grafy, které mohou nedostatečně zachycovat volný konfigurační prostor. Při příliš vysokých hodnotách je zase vygenerováno mnoho bodů, což může být zbytečné a může to zpomalovat simulaci.

  U jednodotazových algoritmů má tato hodnota odlišný význam - určuje naopak horní mez počtu bodů ve stromu (případně v obou stromech dohromady) a pokud není nalezena cesta pomocí stromů dříve, než počet bodů ve stromě (stromech) dosáhne této hodnoty, tak je fáze simulace neúspěšně ukončena.

  U algoritmu SRT znamená tato hodnota zároveň také počet stromů, které mají být do simulace přidány.

- **Maximální počet pokusů pro nalezení volné konfigurace** (Max. attempts to find free configuration) – tato hodnota určuje maximální počet pokusů při hledání nové konfigurace z volného konfiguračního prostoru. Pokud není ani po tomto počtu pokusů volná konfigurace nalezena, je daná fáze simulace neúspěšně ukončena.

- **Průměr bodů grafu** (Point size (diameter)) – určuje průměr bodů grafu, jak budou vykreslovány v simulačním prostoru.

- **Průměr robota** (Robot size) – určuje průměr robota. Tato hodnotá má vliv na zjišťování, zda libovolný bod koliduje s překážkou. Nekontroluje se totiž jen pozice středu bodů, ale pozice celé plochy robota, který se v tomto bodu může potencionálně nacházet. Samozřejmě má tato hodnota vliv také na vykreslování robota.

- **Sada překážek** (Obstacle set) – hodnota předdefinovaných sad překážek, mezi nimž je například labyrint, prostor s úzkým průchodem a několik náhodných sad překážek.

Všechny údaje o vzdálenostech a rozměrech v simulaci jsou uvedeny v pixelech.

# Kapitola 5

# PRM algoritmus – Probabilistic roadmaps

PRM algoritmus (Probabilistic roadmaps) je základním pravděpodobnostním algoritmem, který dobře ilustruje principy pravděpodobnostních algoritmů a existuje velké množství jeho modifikací a vylepšení. PRM je vhodný pro vícerozměrné problémy (roboti s více stupni volnosti), ve své základní podobě zhruba až pro 12 stupňů volnosti [2].

Jedná se o vícedotazový algoritmus a pracuje ve dvou fázích:

1. **Vytvoření grafu (roadmapy)** Tato fáze se dá také nazvat jako *učící fáze*. Jsou vybírány náhodné konfigurace z volného konfiguračního prostoru a ty jsou přidávány do grafu a poté vzájemně spojovány. Vzniklý graf by měl co nejlépe zachycovat volný konfigurační prostor, aby v něm bylo možné v další fázi vyhledat cestu [2].

2. **Vyhledání cesty** Druhá fáze - vyhledávání cesty v grafu - se může opakovat libovolněkrát, protože PRM je vícedotazový a hledání cesty pracuje s grafem vytvořeným v první části, který by měl dobře zachycovat celý volný konfigurační prostor a měl by tedy být použitelný pro jakékoliv hledání mezi dvěma libovolnými body. V této fázi proběhnou nejprve pokusy o napojení požadovaného počátečního a cílového bodu a pokud se tyto body povede napojit do grafu, proběhne vyhledání cesty v tomto grafu, například pomocí Dijkstrova algoritmu [2].

## 5.1 Vytváření grafu (roadmapy)

V první fázi algoritmu PRM se vytváří neorientovaný graf $G = (V, E)$. $V$ je množina uzlů grafu, tedy konfigurací robota v prostoru (v základním případě se jedná o volné konfigurace). $E$ je množina hran spojujících uzly (viz níže – spojovací funkce) tak, že daná hrana (cesta) nekoliduje s překážkami.

### Důležité dílčí funkce pro vytváření grafu:

1. **Spojovací funkce** ($\Delta$, *local planner*)
   Pro vstupní uzly $q', q'' \in C_{free}$ vrátí buď cestu spojující tyto dva body a nekolidující s překážkami, nebo $NIL$ v případě, že tyto body nelze kvůli překážkám spojit. Pro základní variantu PRM je to funkce spojující dva uzly úsečkou, tedy deterministická a symetrická.

2. **Vzdálenostní funkce** (*dist*)
   $C \times C \leftarrow \Re^+ \cup 0$
   Funkce vracející vzdálenost libovolných dvou uzlů grafu, je využívána při vyhledávání nejbližších uzlů. Podrobněji viz 3.3.

**Algoritmus 5.1 Algoritmus pro konstrukci grafu PRM [2].**

***Vstup:***
*n: počet uzlů, kolik má mít graf po provedení algoritmu*
*k: počet sousedních uzlů určených pro napojení*
***Výstup:***
*Graf* $G = (V, E)$

```
1:  V ← ∅
2:  E ← ∅
3:  while | V |< n do
4:    repeat
5:      q ← náhodná konfigurace z C
6:    until q je volná konfigurace
7:    V ← V ∪ {q}
8:  end while
9:  for all q ∈ V do
10:   N_q ← k nejbližších sousedů podle vzdálenostní funkce dist
11:   for all q' ∈ N_q do
12:     if (q, q') ∉ E and Δ(q, q') ≠ NIL then
13:       E ← E ∪ {(q, q')}
14:     end if
15:   end for
16: end for
```

Algoritmus pro konstrukci grafu (roadmapy) funguje následovně:
Na počátku máme v grafu prázdnou množinu vrcholů i hran. Postupně generujeme souřadnice náhodného bodu (konfigurace robota) z $C$ a určíme, zda bod koliduje s překážkou. Pokud tento bod s překážkou nekoliduje (leží v $C_{free}$), přidáme jej jako nový vrchol do grafu.
Takto postupujeme až do chvíle, než graf $G$ obsahuje právě požadovaných $n$ bodů.

Ve druhé části algoritmu už pak pracujeme s body vloženými do grafu $G$ v první části. Postupně ke každému bodu z grafu najdeme $k$ jemu nejbližších bodů $q'$ pomocí funkce pro výběr nejbližších sousedů (3.2). Potom testujeme pomocí spojovací funkce (3.4) cestu mezi vybraným bodem $q$ a každým jeho nejbližším sousedem $q'$. V případě nalezení nekolizní cesty je tato úsečka přidána jako nová hrana do grafu. Pokud bychom použili místo spojování úsečkou nějakou nedeterministickou funkci, museli bychom její výsledek uchovávat pro další použití, avšak to už by se nejednalo o základní PRM algoritmus.

Volba počtu bodů grafu $n$ a počtu nejbližších bodů $k$ pro pokus o spojení je klíčová, protože právě na těchto hodnotách závisí úspěšnost pokrytí konfiguračního prostoru. Pokud zvolíme tyto hodnoty nižší, algoritmus bude méně výpočetně náročný, avšak na úkor úspěšnosti hledání cesty. Pro vyšší hodnoty $n$ a $k$ bude výsledný graf obsahovat více bodů a hran a bude tedy vyšší pravděpodobnost, že se algoritmu podaří najít požadovanou cestu [2].

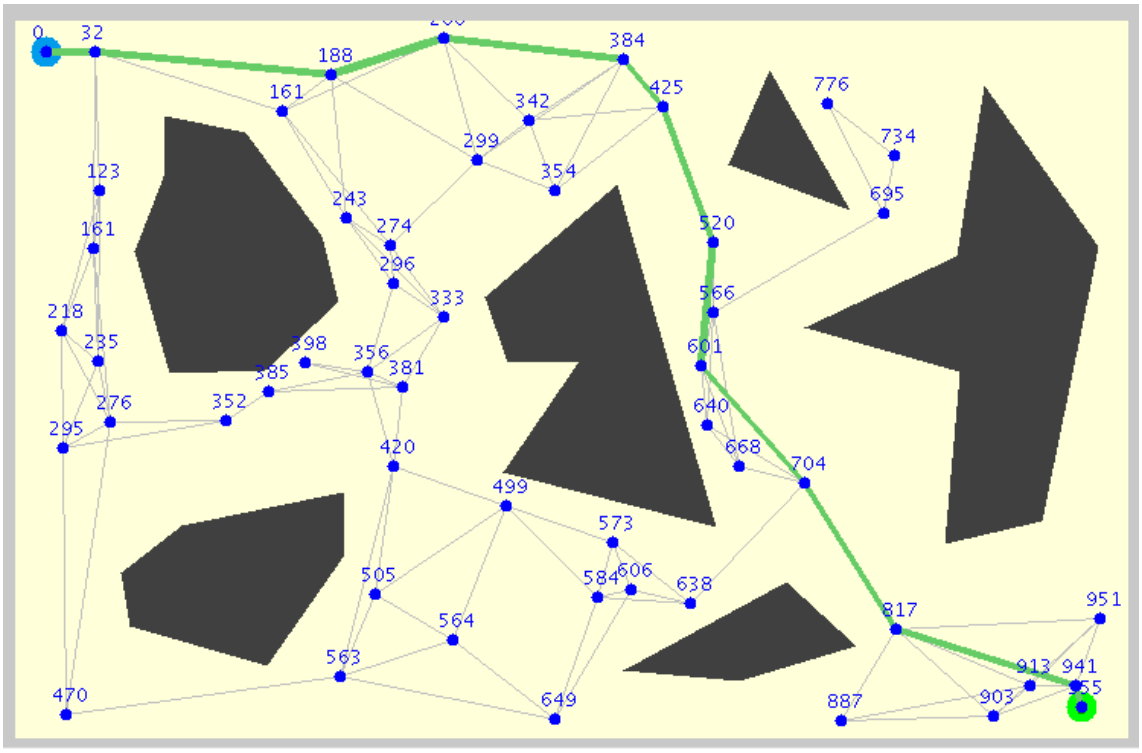
## 5.2 Dotazovací fáze

Druhá fáze algoritmu PRM se může libovolněkrát opakovat – pro každý dotaz lze totiž využít graf vytvořený už v první fázi PRM. K tomuto grafu se ve druhé fázi algoritmus snaží napojit počáteční $q_{init}$ a cílový $a_{goal}$ bod.

**Algoritmus 5.2 Algoritmus pro nalezení cesty v grafu PRM [2].**

***Vstup:***
$q_{init}$*: počáteční konfigurace robota*
$q_{goal}$*: cílová konfigurace robota*
$k$*: počet sousedních uzlů určených pro napojení*
$G = (V, E)$*: Graf vytvořený algoritmem 5.1*
***Výstup:***
*Cesta z* $q_{init}$ *do* $q_{goal}$
1: $N_{q_{init}} \leftarrow k$ *nejbližších sousedů uzlu* $q_{init}$ *z* $V$ *podle vzdálenostní funkce dist*
2: $N_{q_{goal}} \leftarrow k$ *nejbližších sousedů uzlu* $q_{goal}$ *z* $V$ *podle vzdálenostní funkce dist*
3: $V \leftarrow V \cup \{q_{init}\} \cup \{q_{goal}\}$
4: $q' \leftarrow$ *nejbližší sousední uzel uzlu* $q_{init}$ *vybraný z* $N_{q_{init}}$
5: ***repeat***
6: ***if*** $\Delta(q_{init}, q') \neq NIL$ ***then***
7: $E \leftarrow E \cup (q_{init}, q')$
8: ***else***
9: $q' \leftarrow$ *další sousední uzel uzlu* $q_{init}$ *vybraný z* $N_{q_{init}}$
10: ***end if***
11: ***until*** *uzel* $q_{init}$ *byl úspěšně napojen nebo* $N_{q_{init}}$ *je prázdný*
12: $q' \leftarrow$ *nejbližší sousední uzel uzlu* $q_{goal}$ *vybraný z* $N_{q_{goal}}$
13: ***repeat***
14: ***if*** $\Delta(q_{goal}, q') \neq NIL$ ***then***
15: $E \leftarrow E \cup (q_{goal}, q')$
16: ***else***
17: $q' \leftarrow$ *další sousední uzel uzlu* $q_{goal}$ *vybraný z* $N_{q_{goal}}$
18: ***end if***
19: ***until*** *uzel* $q_{goal}$ *byl úspěšně napojen nebo* $N_{q_{goal}}$ *je prázdný*
20: $P \leftarrow$ *nejkratší cesta z* $q_{init}$ *do* $q_{goal}$ *pomocí* $G$
21: ***if*** $P$ *není prázdná* ***then***
22: ***return*** $P$
23: ***else***
24: ***return*** $failure$
25: ***end if***

Nejprve je potřeba napojit počáteční a cílový bod do grafu $G$. To se provádí podobně jako napojování uzlů v první fázi – funkcí pro výběr nejbližších sousedů získáme $k$ nejbližších bodů a ty se pak postupně snažíme úsečkou spojit s počátečním (resp. cílovým) bodem tak, aby úsečka (cesta) nekolidovala s překážkami. Hodnota $k$ v této fázi nemusí být stejná jako ta z fáze minulé, protože v tomto případě je výběr a pokus o napojení nejbližších bodů proveden pouze dvakrát a případná vyšší hodnota $k$ tedy nemá prakticky žádný vliv na výkonnost algoritmu.




Obrázek 5.1: Ukázka appletu s nalezenou cestou v grafu PRM. Počáteční bod je vlevo nahoře, cílový vpravo dole. Graf je znázorněn jednotlivými uzly (u každého uzlu je zobrazena vzdálenost od počátečního bodu, spočítaná při vyhledávání nejkratší cesty) a hranami, překážky šedými polygony a nalezená nejkratší cesta je vyznačena silnější čarou od počátku do cíle.

Pokud se podaří napojit počáteční a zároveň i cílový bod do grafu, zbývá jen použít libovolný algoritmus pro vyhledání (nejkratší) cesty v grafu. Nejčastěji se používá Dijkstrův algoritmus (ten je vhodný pro hledání nejkratší cesty v *neměnném* grafu, což je právě tento případ), případně A*. Nalezenou cestu lze potom tedy popsat seznamem po sobě jdoucích vrcholů grafu $G$, mezi kterými existují v tomto grafu hrany.

Uvedený algortmus pro dotazovací fázi PRM ale nezaručuje, že po napojení startovního a cílového bodu bude cesta mezi nimi vždy nalezena. Pokud totiž graf nedostatečně zachycuje volný konfigurační prostor a je tvořen oddělenými komponentami, může se stát, že počáteční bod je napojen k jiné komponentě než cílový a neexistuje tedy mezi nimi cesta. V takovém případě jsou dvě možnosti jak cestu nalézt – přidáním dalších bodů do grafu nebo pokusem napojit oba body ke stejné komponentě [2].

První možnost spočívá v přidání dalších bodů do grafu. Pokud je vůbec možné oddělené komponenty grafu propojit (tedy pokud se nejedná například o konfigurační prostor oddělený překážkou na dvě části), pak přidáním dalších bodů do grafu zvyšujeme pravděpodobnost propojení komponent. Body můžeme přidat opakováním algoritmu 5.1 s tím rozdílem, že na začátku neinicializujeme prázdný graf (řádek 1 a 2), ale pracujeme s již dříve vytvořeným grafem.

Druhou možností je pokusit se napojit počáteční a cílový bod ke stejné komponentě grafu $G$. Můžeme procházet postupně všechny komponenty grafu a ty předat jako vstup

algoritmu 5.2. Pokud dojde k napojení obou bodů k jedné komponentě, pak je zaručeno, že mezi nimi existuje cesta [2].

## 5.3 Implementace PRM appletu

Protože PRM je základním a nejjednodušším pravděpodobnostním algoritmem, byla jeho implementace díky už přepraveným základním funkcím pro práci s grafem, které jsou implementovány ve třídách *Roadmap* a *SimRoadmap*, triviální.

Simulace PRM algoritmu probíhá v pěti fázích:

1. Generování náhodných bodů
2. Spojení bodů do grafu
3. Napojení starovního a cílového bodu
4. Vyhledání cesty a její případné vyhlazení
5. Přesun robota do cíle

Všechny funkce, které realizují tyto fáze, jsou přímo součástí třídy *PRMSim*, kromě poslední, která je stejně jako u ostatních appletů vykonávána funkcí *moveRobot()* zděděnou ze *SimRoadmap* (viz. 4).

První fáze simulace, generování náhodných bodů ve volném konfiguračním prostoru, je implementována ve funkci *generatePoints()*. Ta probíhá přesně podle prvních osmi řádků algoritmu 5.1 pro konstrukci PRM grafu a při vygenerování každého nového bodu umožňuje při zpomalené simulaci tento bod zvýraznit a zobrazit o něm další informace ve stavové liště. Druhá fáze spočívá v pospojování blízkých bodů grafu hranami. Tu provádí funkce *connectRoadmap()* opět podle algoritmu 5.1, podle 9. až 16. řádku. Při zpomalené simulaci tato funkce zobrazuje všechny pokusy o napojení sousedních bodů.

Třetí simulační fáze se pokouší napojit nejprve počáteční a poté cílový bod do vzniklého grafu. Tuto akci obstarává funkce *addStartAndGoal()* podle algoritmu 5.2. Čtvrtá fáze probíhá v režii funkce *searchPath()*, která je implementací Dijkstrova algoritmu doplněného o průběžné zobrazování vzálenosti již určených bodů.

Poslední fáze je stejně jako u ostatních appletů prováděna funkcí *moveRobot()*, která je zděděna ze třídy *SimRoadmap*.

Co se týče speciálních nastavení simulace, tak PRM applet obsahuje oproti základním nastavením (zmíněným v části 4.6) pouze jedno. Tím je hodnota *Počet nejbližších sousedů pro napojení* (*Closest neighbors to examine*), která udává, s kolika nejbližšími sousedy se má každý uzel pokoušet spojit hranou. Vyšší hodnota tohoto nastavení sice zvyšuje pravděpodobnost, že daný uzel bude spojen alespoň s jedním sousedem, ale také zvyšuje náročnost simulace, protože vzniká více zbytečných hran.

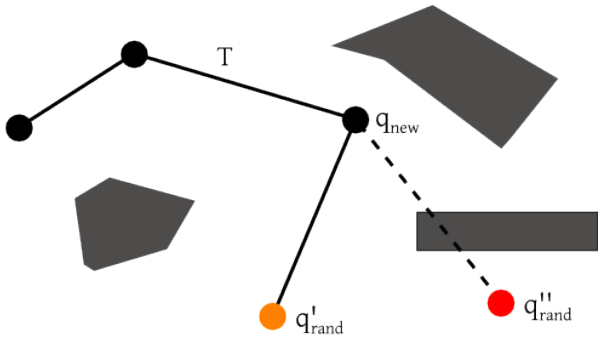
# Kapitola 6

# EST algoritmus – Expansive-Spaces Trees

EST je výkonný jednodotazový algoritmus, který se často využívá pro řešení kinodynamických úloh. Pracuje, podobně jako RRT, se stromem ukotveným v počátečním (a v případě dvoustromové verze i v cílovém) bodě a tento strom se postupně větví do volného konfiguračního prsotoru.

## 6.1 Vytváření stromu

Algoritmus EST může pracovat ve dvou variantách – buď vytváří dva stromy (ukotvené v počáteční a cílové pozici), nebo pracuje jen s jedním stromem (ukotvený v počáteční pozici), což je vhodné pro kinodynamické úlohy [2].




Obrázek 6.1: Ukázka způsobu přidávání nového uzlu do stromu EST. Bod $q$ byl vybrán ze stromu na základě pravděpodobnostní funkce $\pi_T$ a $q'_{rand}$ je náhodně zvolený bod z volného konfiguračního prostoru. Pro takto vybraný bod $q'_{rand}$ dojde k rozšíření stromu – je do něj přidán právě bod $q'_{rand}$ a hrana $(q, q'_{rand})$. Pokud jako náhodný bod zvolili $q''_{rand}$, k rozšíření stromu by nedošlo.

Přidávání nových uzlů a hran do stromu se řídí algoritmem *Build EST* (6.1) a *Extend EST* (6.2):

Pomocí pravděpodobnostní funkce $\pi_T$ (viz dále) náhodně vybereme jeden z už existujících uzlů stromu a k němu vygenerujeme nový, náhodný bod z jeho okolí. Poté pomocí

spojovací funkce (úsečkou) zjistíme, zde mezi nimi existuje cesta a pokud ano, nový bod i tuto cestu (hranu) přidáme do stromu.

Oproti PRM tedy přidáváme do grafu jen ty uzly, které se podaří napojit do stromu – pokud tedy v závěrečné fázi připojíme do stromu i počáteční a cílový bod, pak je jisté, že hledaná cesta existuje.

**Algoritmus 6.1 Build EST – algoritmus pro konstrukci stromu EST [2].**

***Vstup:***
*$q_0$: kořenový uzel stromu*
*$n$: maximální počet pokusů o rozšíření stromu*
***Výstup:***
*Strom $T = (V, E)$, který má kořen $q_0$ a maximálně $n$ uzlů*
*1:* $V \leftarrow \{q_0\}$
*2:* $E \leftarrow \emptyset$
*3:* ***for*** $i = 1$ *to* $n$ ***do***
*4:* $q_{rand} \leftarrow$ *volná konfigurace náhodně vybraná s pravděpodobností* $\pi_T(q_{rand})$
*5:* *extendEST($T$, $q_{rand}$)*
*6:* ***end for***
*7:* ***return*** $T$

**Algoritmus 6.2 Extend EST – algoritmus pro expanzi EST stromu [2].**

***Vstup:***
*Strom $T = (V, E)$ (EST strom))*
*$q$: konfigurace $\in V$, ze které bude strom expandovat*
***Výstup:***
*Nová konfigurace (uzel) $q_{new}$ v okolí bodu $q$, nebo $NIL$ v případě neúspěchu*
*1:* $q_{new} \leftarrow$ *náhodně vybraná volná konfigurace v okolí bodu* $q$
*2:* ***if*** $\Delta(q, q_{new})$ ***then***
*3:* $V \leftarrow V \cup \{q_{new}\}$
*4:* $E \leftarrow E \cup \{(q, q_{new})\}$
*5:* ***return*** $q_{new}$
*6:* ***end if***
*7:* ***return*** $NIL$

## 6.2 Výběr pravděpodobnostní funkce

Při vytváření stromu je nejdůležitější účinně vybírat ze stromu náhodné body, určené pro připojování dalších nových bodů. Pravděpodobnostní funkce totiž nesmí způsobovat „převzorkování“ (oversampling) v některých místech (zejména kolem počátku, případně cíle), protože by to snižovalo efektivitu algoritmu. Je tedy potřeba zvolit takovou pravděpodobnostní funkci $\pi_T$, která bude zaručovat generování nových náhodných bodů s vyšší pravděpodobností v prostoru méně pokrytém stomem [2].

V praxi se osvědčilo uchovávat o každém bodu stromu informaci $w_T(q)$, která vyjadřuje hustotu okolních bodů. Ta se dá spočítat nejjednodušeji výčtem bodů v předem definovaném

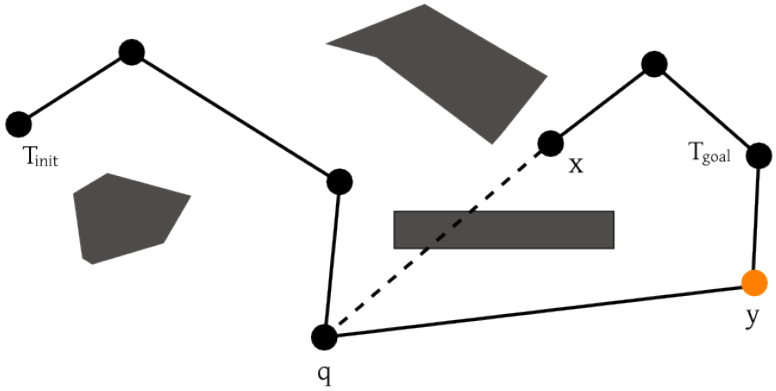
okolí každého bodu. Pokud rozložení pravděpodobnosti definujeme jako inverzní k této funkci hustoty okolí, získáme pravděpodobnostní rozložení $\pi_T$, které upřednostňuje méně pokryté oblasti a umožňuje tak stromu expandovat do neprozkoumaných částí prostoru [2].

Rozumnou aproximací toho způsobu je pak metoda pracující s prostorem rozděleným na pravidelnou mřížku [2], která uchovává informaci o počtu uzlů stromu v každé buňce této mřížky. Nové uzly jsou pak přidávány s větší pravděpodobností do méně „obsazených“ buněk. U tohoto způsobu se jednoduše aktualizuje mřížka po přidání nového uzlu do stomu a je tedy vhodnější pro náročnější úlohy.

Protože pravděpodobnostní funkce $\pi_T$ pro generování nových vzorků je pro EST algoritmus zásadní, existuje mnoho dalších variant. Můžeme kromě počtu sousedů zohledňovat také pořadí, ve kterém byly uzly přidány do stromu, vzdálenost uzlů od počátku nebo můžeme využít $A^*_{cost}$ funkci [2], která kombinuje hodnotu vzdálenosti od počátku s očekávanou hodnotou vzdálenosti k cíli. Při implementaci EST appletu jsem mimo jiné použil i modifikaci výše zmíněného „mřížkového“ přístupu, jak bude podrobněji popsáno v 6.5

## 6.3 Napojení stromů

V případě jednostromové verze je možné algoritmus EST ukončit v případě, že nově přidaný bod lze přímo spojit s cílovým.




Obrázek 6.2: Spojování dvou EST stromů. Po přidání nového uzlu $q$ do stromu $T_{init}$ se tento uzel pokouší spojit s nejbližšími sousedními body ze druhého stromu $T_{goal}$. Pro nejbližší bod $x$ se spojení nepovede, ale díky dalšímu blízkému bodu $y$ jsou oba dva stromy úspěšně spojeny.

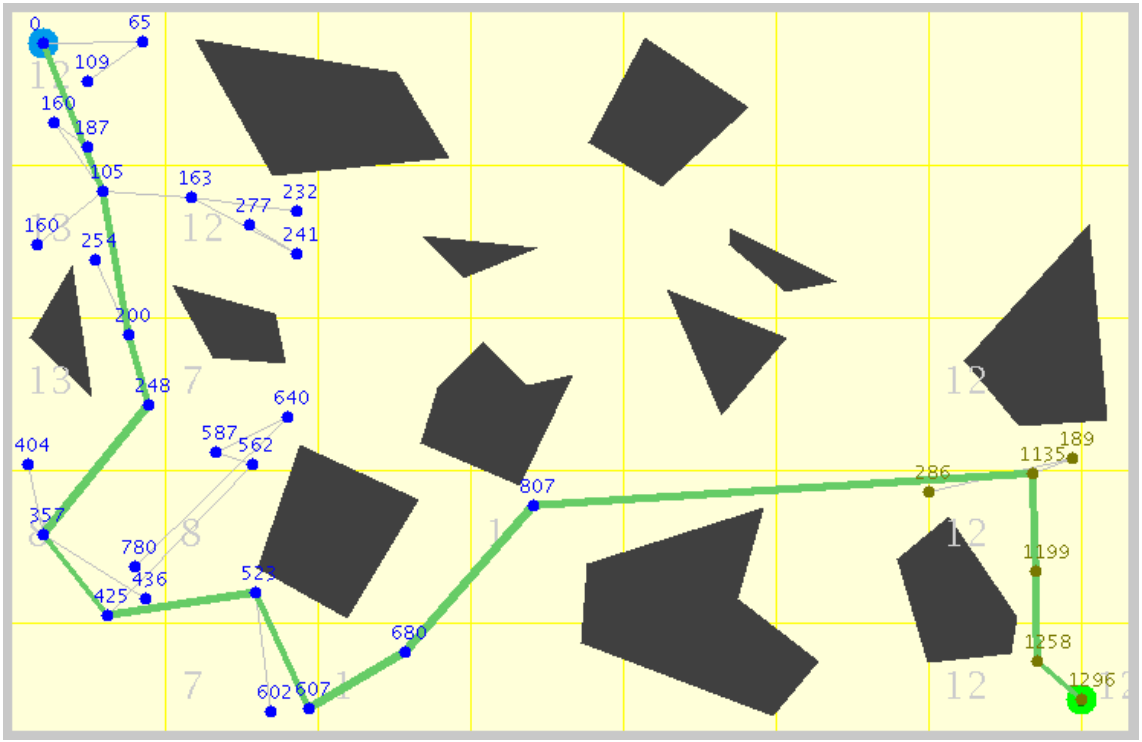
Pokud EST pracuje se dvěma stromy, je potřeba, podobně jako u RRT algoritmu, stromy spojit a tím zajistit existenci cesty mezi počátkem a cílem. Stromy jsou úspěšně spojeny v případě, že je nalezena cesta mezi libovolným bodem z prvního stromu a jiným bodem z druhého stromu. Pomocí stejné pravděpodobnostní funkce $\pi_T$ jako při generování EST stromu vybereme jeden uzel z prvního stromu. Pak najdeme k němu nejbližší uzel z druhého stromu a pokusíme se je spojit úsečkou. Pokud se spojení bodů podaří, tak oba dva stromy mohou být propojeny a cesta mezi počátkem a cílem se získá spojením posloupností mezi kořeny stromů a těmito spojovacími uzly. V případě neúspěšného propojení stromů se

stromy vymění a celý spojovací postup se opakuje [2].

## 6.4 SBL algoritmus

SBL (Single-query, Bi-directional, Lazy-collision checking, [2]) je jednodotazový algoritmus, který pracuje podobně jako EST. SBL také pracuje se dvěma stromy, ukotvenými v počátku a cíli, ale liší se ve způsobu přidávání nových bodů a hran do stromu – SBL totiž netestuje, zde mezi novým bodem a vybraným bodem ze stromu existuje hrana nekolidující s překážkou. SBL algoritmus nový bod a hranu přidá do stromu vždy, takže vzniklé stromy neleží jen ve volném konfiguračním prostoru, ale můžou kolidovat i s překážkami.

Vyhodnocování legitimity hran probíhá až na závěr a to jen u těch hran, které jsou součástí hledané cesty. Díky této vlastnosti algoritmus ušetří prostředky, které by jinak zabralo vyhodnocování všech hran.




Obrázek 6.3: Ukázka nalezené cesty pomocí EST appletu pracujícího s pravděpodobnostní funkcí založenou na mřížce.

Pokud zvolíme pro SBL pravděpodobnostní funkci založenou na mřížce (6.2) a vybíráme pro napojování přednostně buňky mřížky obsahující méně uzlů stromu, je SBL algoritmus velmi rychlý a výkonný.

## 6.5 Implementace EST appletu

Specialitou EST algoritmu oproti ostatním PA je pravděpodobnostní funkce pro výběr uzlů, ze kterých se má EST strom rozšiřovat.

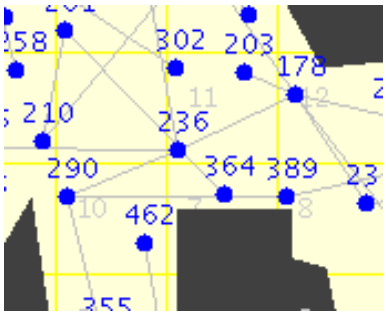
V simulační třídě EST appletu (*ESTSim*) jsou implementovány dva přístupy pro výběr uzlů – na základě počtu uzlů v okolí a pomocí mřížky.

První přístup s vyšší pravděpodobností vybírá uzly s menší hustotou okolních bodů, přičemž pojem „okolí“ je prostor kolem každého bodu do vzdálenosti určené proměnnou *generateNeighborhoodSize*. Hustota okolí je pak pro každý bod dána jako

$$\Sigma_{x'}(generateNeighborhoodSize - dist(x, x'))$$

, kde $x'$ jsou všechny body v daném okolí bodu $x$. Hodnoty hustoty jsou uchovávány ve zvláštní hashovací tabulce, ve které je jako klíč použita hash hodnota uzlu. Pro přidávání uzlů do grafu je proto využívána funkce *recomputePointDensity*, která zároveň s přidáním bodu přepočítá hustotu okolních bodů v této tabulce, a pro náhodný výběr bodu (s větší pravděpodobností bodu s řídkým okolím) pak funkce *getSparsePoint*.

Druhý přístup dělí simulační prostor pomocí mřížky na menší čtvercové oblasti – buňky. Výběr bodu pro expanzi v tomto případě spočívá nejprve ve výběru buňky s nejmenší hustotou bodů uvnitř a poté je náhodně vybrán bod z této buňky. Hustota buněk je dána počtem bodů, které se uvnitř ní nachází. Pro práci s buňkami jsem ve třídě *ESTSim* vytvořil vnořenou třídu *Cell*, která uchovává informace o bodech v této buňce, hustotě a o pozici v mřížce. Dále poskytuje metody pro přidání bodu do buňky a pro výběr náhodného bodu z ní – *addVertex()* a *getRandomVertex()*.




Obrázek 6.4: Bod ve vzdálenosti 364 se nachází v buňce, ve které je sice jediný a proto bude při základním přístupu tato buňka často vybírána pro expanzi, ale kvůli překážce už v této buňce není mnoho místa pro další body, takže budou expanze neúspěšné.

Po implementaci obou zmíněných přístupů jsem zjistil, že efektivita pravděpodobnostní funkce při generování stromů je nižší, pokud se oblast vymezená pro určování hustoty (tedy okolí bodu nebo buňka) nenachází celá ve volném konfiguračním prostoru. Pokud tedy například polovinu buňky zabírá překážka, nemůže být v této buňce nikdy tolik bodů, jako v buňce bez překážek a dojde dříve ke stavu „zahlcení“ buňky, kdy už není možné do ní přidávat další body. Buňky s menší plochou volného konfiguračního prostoru proto průměrně obsahují méně bodů a jsou tím pádem častěji vybírány pro expanzi, i když expanze z nich už nemusí být možná (viz obrázek 6.4). Podobně je tomu také u přístupu s počítáním hustoty na základě bodů v okolí bodu, kdy se v tomto okolí nachází překážka.

Aby se předešlo tomuto neefektivnímu výběru bodů pro expanzi, bylo by možné hustotu vyjadřovat se zohledněním poměru plochy volného konfiguračního prostoru k celé ploše dané buňky/okolí bodu. Další možností je nastavení vzdálenosti pro generování nových bodů výrazně větší, než rozměry buňky, což ale řeší problém jen částečně. V EST appletu jsem

| Rozložení překážek | Základní přístup | Vylepšený přístup |
|---|---|---|
| Náhodné I. | 145 | 106 |
| Náhodné II. | 381 | 261 |
| Náhodné III. | 130 | 96 |
| Úzký průchod | 217 | 148 |
| Mnoho překážek | 339 | 226 |
| Labyrint | 2034 | 1086 |
| Celkem průměrně | 649 | 385 |

Tabulka 6.1: Srovnání dvou přístupů pro výběr buněk pro pokus o expanzi EST stromu – základního (podle počtu bodů, které obsahují) a vylepšeného (podle počtu bodů a neúspěšných pokusů o expanzi z buňky) přístupu. Tabulka zachycuje naměřené hodnoty celkového počtu pokusů o expanzi, než byla nalezena cesta. Tyto hodnoty byly u každého rozložení překážek získány jako průměr z 200 měření (jedna polovina při simulaci s jedním stromem, druhá se dvěma stromy).

pro výběr buněk použil ještě jiný přístup – po každé neúspěšné expanzi z dané buňky je inkrementováno její počítadlo bodů. Toto počítadlo pak tedy vyjadřuje součet počtu bodů v buňce a počtu neúspěšných pokusů o expanzi této buňky. Při generování stromu jsou díky tomu častěji vybírány buňky s menším počtem bodů a s menším počtem neúspěšných pokusů o expanzi, což při rovnosti počtu bodů upřednostňuje pro výběr nověji přidané buňky, které mají menší hodnotu neúspěšných pokusů.

Na toto vylepšní jsem nikde v literatuře nenarazil, takže jsem se svůj předpoklad, že opravdu zefektivňuje EST algoritmus, snažil experimentálně ověřit. Porovnání tohoto přístupu s obyčejným výběrem buněk podle počtu bodů v nich je zachyceno v tabulce 6.1, ze které plyne, že tato modifikace je pro výběr buněk opravdu efektivnější.

Samotná simulace EST algoritmu probíhá ve třech fázích:

1. Vytváření EST stromů
2. Vyhledání cesty a její případné vyhlazení
3. Přesun robota do cíle

První dvě fáze jsou realizovány funkcemi přímo ze třídy *ESTSim*, třetí je jako u ostatních appletů vykonávána funkcí *moveRobot()* zděděnou ze *SimRoadmap* (viz. 4).

Nejdůležitější – první – fázi provádí funkce *buildEST()*. Ta pracuje podobně, jako je popsáno v algoritmu 6.1. Narozdíl od tohoto algoritmu ale po každém úspěšném přidání nového bodu do grafu testuje, zda už nelze generování úspěšně ukončit. Při jednostromové variantě je to realizováno pokusem o spojení nového bodu s cílovým pomocí funkce *mergeWithGoal()*, u dvoustromové verze pokusem o spojení nového bodu s kterýmkoliv bodem ze druhého stromu, což provádí funkce *mergeEST()*.

Pokud je generování EST stromu/stromů ukončeno úspěšně, ve druhé fázi je pomocí funkce *searchPath()* vyhledána cesta. V případě jednostromové verze pomocí posloupnosti cesty od cíle ke kořeni stromu (tedy do počátečního bodu robota), v případě dvoustromové varianty pomocí funkce *searchTwoTrees()* ze třídy *SingleQuerySim*, která cestu vytvoří spojením dvou cest od bodu, který stromy spojuje, do jejich kořenů.

EST applet obsahuje oproti základním nastavením (zmíněným v části 4.6) několik nastavení speciálních. Je možné si vybrat mezi jedno- a dvoustromovou verzí simulace (*Single tree* nebo *Two trees (bidirectional)*), mezi pravděpodobnostní funkcí založenou na hustotě bodů v okolí bodu (*Simple approach*) nebo založenou na mřížce (*Grid approach*).

Lze nastavit zobrazování všech bodů, které jsou náhodně generovány, i když potom nejsou přidány do grafu (*Show all random points*). Při vybrané pravděpodobnostní funkci založené na mřížce je možné aktivovat optimalizaci zohledňující i počet neúspěšných expanzí z buňky, jak bylo popsáno výše v této části. K tomu slouží zatžítko *Progressive cell select*.

Další nastavení se týká velikosti buňky mřížky (*Cell size*), velikosti okolí pro generování nového bodu (*Neighborhood size to generate point*) a poloměr oblasti pro výpočet hustoty v okolí bodu (*Neighborhood size for dist function*).

# Kapitola 7

# RRT algoritmus – Rapidly-Exploring Random Trees

Pravděpodobnostní algoritmus PRM rozebíraný v minulé kapitole byl ukázkou vícedotazového algoritmu. Pokud ale potřebujeme provést jen jedno hledání mezi dvěma konkrétními body prostoru, může být efektivnější použít některý z jednodotazových algoritmů. Vyhneme se tím fázi budování grafu zachycujícího *celý* konfigurační prostor, protože k nalezení cesty mezi dvěma body stačí vytvořit graf jen mezi těmito body a zbytek konfiguračního prostoru už nás obvykle nezajímá.

Jedním z prvních jednodotazových algoritmů byl RPP [2], který využíval k hledání cesty potenciálová pole mezi překážkami a pokud hledání uvízlo v lokálním minimu, algoritmus se z něj pokusil dostat pomocí náhodných pohybů.

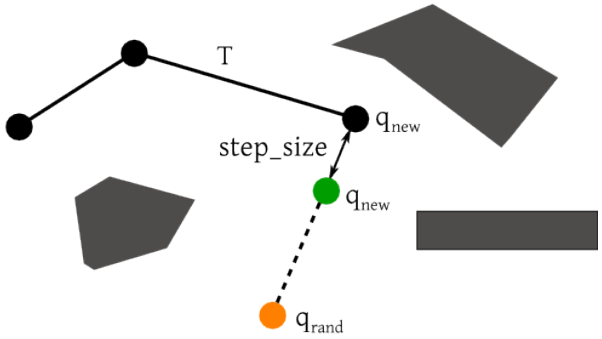
Dalšími zástupci jednodotazových algoritmů jsou RRT (**Rapidly-exploring Random Trees**) a EST (**Expansive-Spaces Trees**). Oba dva pracují se dvěma stromy, jejichž kořeny jsou v počátečním a cílovém bodě. Tyto dva stromy se postupně rozšiřují po volném konfiguračním prostoru a algoritmy končí ve chvíli, kdy lze oba stromy spojit. Místo dvou stromů je možné pracovat jen s jedním, který by pak měl kořen v počátečním stavu, a proto jsou oba algoritmy vhodné pro tzv. *kinodynamické* (*kinodynamic*, [6]) úlohy, což jsou úlohy, kdy musíme při hledání cesty uvažovat rychlost a zrychlení robota.

## 7.1 Vytváření stromů

Základním principem algoritmu RRT je využití dvou stromů $T_{init}$ a $T_{goal}$, jejichž kořenem jsou startovní a cílový bod pro hledání cesty – $q_{init}$ a $q_{goal}$. V první fázi algoritmu je tedy potřeba vytvořit tyto dva stromy.

Oba dva stromy se budují zároveň – postupně se přidává jeden nový uzel do $T_{init}$ a vzápětí do $T_{goal}$. Uzel se přidává podle algoritmu *Build RRT* (7.1) a *Extend RRT* (7.2):

Nejprve vygenerujeme náhodný bod $q_{rand}$ z volného konfiguračního prostoru. K němu pomocí vzdálenostní funkce vybereme nejbližší bod $q_{near}$ ze stromu, se kterým právě pracujeme, a ve vzdálenosti *step_size* pak získáme nový bod $q_{new}$. Potom pomocí spojovací funkce zjistíme, zda je možné tento nový bod přidat do grafu – tedy jestli existuje nekolizní cesta mezi $q_{near}$ a $q_{new}$. Pokud taková cesta existuje, přidáme ke stromu tuto hranu a také bod $q_{new}$ jako nový list.




Obrázek 7.1: Ukázka způsobu přidávání nového uzlu do stromu RRT. Bod $q_{rand}$ je náhodně vybrán z volného konfiguračního prostoru a poté je ze stromu $T$ nalezen k němu nejbližší bod $q_{near}$. Nová konfigurace $q_{new}$ pro přidání do stromu pak leží ve vzdálenosti *step_size* od $q_{near}$ ve směru ke $q_{rand}$.

**Algoritmus 7.1 Build RRT – algoritmus pro konstrukci stromu RRT [2].**

***Vstup:***
*$q_0$: kořenový uzel stromu*
*$n$: maximální počet pokusů o rozšíření stromu*
***Výstup:***
*Strom $T = (V, E)$, který má kořen $q_0$ a maximálně $n$ uzlů*

```
1:  V ← {q_0}
2:  E ← ∅
3:  for i = 1 to n do
4:      q_rand ← náhodně vybraná volná konfigurace
5:      extendRRT(T, q_rand)
6:  end for
7:  return T
```

## 7.2 Volba délky kroku při vytváření stromů

Zvláštní roli u algoritmu RRT hraje volba délky kroku *step_size*. Pokud pracujeme s konstantní délkou kroku, neměla by být ani příliš krátká (potom by vznikalo velké množství uzlů stromu blízko sebe a algoritmus by byl méně výkonný), ani příliš dlouhá (pak by při velkém množství překážek mohlo být obtížné přidat nový uzel a algoritmus *Build RRT* (7.1) by mohl neúspěšně skončit) [2].

Délka kroku může být také dynamicky volená na základě vzdálenosti $q$ a $q_{near}$. Pokud můžeme umístit $q_{new}$ do větší vzdálenosti od stromu, strom pak rychleji expanduje do volného prostoru [2].

Další možností je využít algoritmus *Connect RRT* (7.3). Ten spočívá v opakování základního algoritmu *Extend RRT* (7.2) tak dlouho, dokud je možné postupně posouvat $q_{new}$ směrem k náhodnému bodu $q_{rand}$ o konstantní hodnotu *step_size*. Pokud bychom chtěl zabránit přílišnému „zahušťování“ grafu novými body směrem ke $q_{rand}$, je možné přidávat pouze poslední bod, který ještě nekoliduje s překážkou [2]. Tento algoritmus je také využíván pro spojování stromů, jak bude vysvětleno v následující části.

**Algoritmus 7.2 Extend RRT – algoritmus pro expanzi RRT stromu [2].**

***Vstup:***
*Strom $T = (V, E)$ (RRT strom))*
*$q$: konfigurace, ke které bude strom expandovat*
***Výstup:***
*Nová konfigurace (uzel) $q_{new}$ (ve směru ke $q$), nebo $NIL$ v případě neúspěchu*

```
1:  q_near ← nejbližší soused bodu q ze stromu T
2:  q_new ← bod na úsečce q_near, q ve vzdálenosti step_size od near
3:  if q_new nekoliduje s překážkou then
4:      V ← V ∪ {q_new}
5:      E ← E ∪ {(q_near, q_new)}
6:      return q_new
7:  end if
8:  return NIL
```

**Algoritmus 7.3 Connect RRT – algoritmus pro expanzi stromu RRT [2].**

***Vstup:***
*Strom $T = (V, E)$ (RRT strom))*
*$q_{rand}$: konfigurace, ke které bude strom expandovat*
***Výstup:***
*connected pokud se uzel $q$ podaří napojit; jinak $failure$*

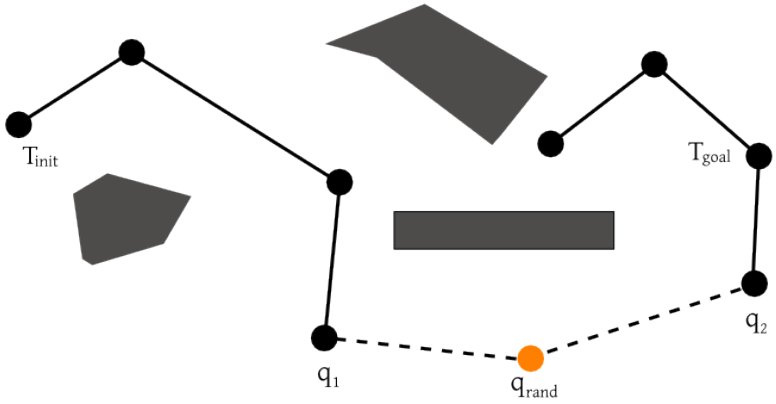
```
1:  repeat
2:      q_new ← extendRRT(T, q)
3:  until (q_new = q or q_new = NIL)
4:  if q_new = q then
5:      return connected
6:  else
7:      return failure
8:  end if
```

## 7.3 Spojování stromů

Algoritmus RRT, respektive jeho hlavní část – vytváření stromů – končí ve chvíli, kdy se podaří oba dva stromy spojit. Protože jeden ze stromů má kořen v počátečním a druhý v cílovém bodě, jejich spojením dostaneme graf obsahující hledanou cestu mezi těmito body. Tu oproti algoritmu PRM nemusíme vyhledávat žádným dalším algoritmem, ale dostaneme ji pouhým spojením cest od listů, které spojily stromy, ke kořenům.

Spojení dvou stromů můžeme zkoušet po každém přidání nového bodu $q_{new}$ do jednoho ze stromů. Je možné pokoušet se napojit tento nově přidaný bod se kterýmkoliv z bodů ze druhého stromu, nebo lze využít algoritmu *Connect RRT* (7.3).

Ke spojování stromů za využití agoritmu *Connect RRT* (7.3) slouží algoritmus *Merge RRT* (7.4). Nejprve je do jednoho ze stromů přidán nový bod $q_{new}$ (pomocí *Extend RRT*) a ten je vzápětí použit jako vstup $q_{rand}$ opět pro *Extend RRT*, který je teď ale volán pro druhý strom. Pokud je možné spojit druhý strom s $q_{new}$, je tento bod přidán i do druhého stromu a pomocí něj jsou oba stromy spojeny a celý algoritmus úspěšně končí. V případě




Obrázek 7.2: Spojování dvou RRT stromů. Bod $q_{rand}$ je náhodně vybrán z volného konfiguračního prostoru a následně jsou k němu rozšiřovány oba dva stromy (ze svých nejbližších bodů – $q_1$ a $q_2$). Pokud obě rozšiřování skončí úspěšně v bodě $q_{rand}$, jsou stromy tímto bodem spojeny.

neúspěchu je možné oba dva stromy vyměnit a *Merge RRT* opakovat s novým $q_{rand}$ na počátku, ale nemá smysl jej opakovat mnohokrát, protože by to zpomalovalo celý RRT algoritmus [2].

U algoritmu *MergeRRT* je možné místo volání *Extend RRT* volat *ConnectRRT*, aby se stromy podařilo spojit dříve ještě v situacích, kdy jsou od sebe vzdálenější, než je *step_size*. Je také možné zvolit kompromis a *Connect RRT* volat jen na jednom z řádků 3 a 5 algoritmu *Merge RRT* (7.4) [2].

## 7.4 Implementace RRT appletu

RRT algoritmus probíhá ve třech fázích:

1. Vytváření RRT stromů
2. Vyhledání cesty a její případné vyhlazení
3. Přesun robota do cíle

První dvě fáze jsou realizovány funkcemi přímo ze třídy *RRTSim*, třetí je jako u ostatních appletů vykonávána funkcí *moveRobot()* zděděnou ze *SimRoadmap* (viz. 4).

Nejdůležitější – první – fázi provádí funkce *buildRRT()*. Ta pracuje podobně, jako je popsáno v algoritmu 7.1. Je přitom možné využít jak přidávání bodů pouze do vzdálenosti *stepSize* (funkce *extendRRT()* podle algoritmu 7.2), tak „hladový“ přístup (funkce *connectRRT* podle algoritmu 7.3). U hladového přístupu je možné nastavit (volba *Add only end point if greedy*), zda se budou do stromu přidávat všechny body směrem k náhodnému bodu, dokud to bude možné – pokud je tato volba aktivována, je přidán pouze poslední bod.

Funkce *buildRRT* po přidání každého nového bodu do stromu zkouší, zda už nelze generování úspěšně ukončit. Při jednostromové variantě je to realizováno pokusem o spojení

**Algoritmus 7.4 Merge RRT – algoritmus pro spojení dvou RRT stromů [2].**

***Vstup:***
*$T_1$: první RRT strom*
*$T_2$: druhý RRT strom*
*$l$: maximální počet pokusů o spojení stromů $T_1$ a $T_2$*
***Výstup:***
*merged pokud se podaří stromy spojit; jinak $failure$*

```
1:  for i = 1 to l do
2:    q_rand ← náhodně vybraný bod z volného konfiguračního prostoru
3:    q_new,1 ← extendRRT(T_1, q_rand)
4:    if q_new,1 ≠ NIL then
5:      q_new,2 ← extendRRT(T_2, q_new,1)
6:      if q_new,1 = q_new,2 then
7:        return merged
8:      end if
9:      Swap(T_1, T_2)
10:   end if
11: end for
12: return failure
```

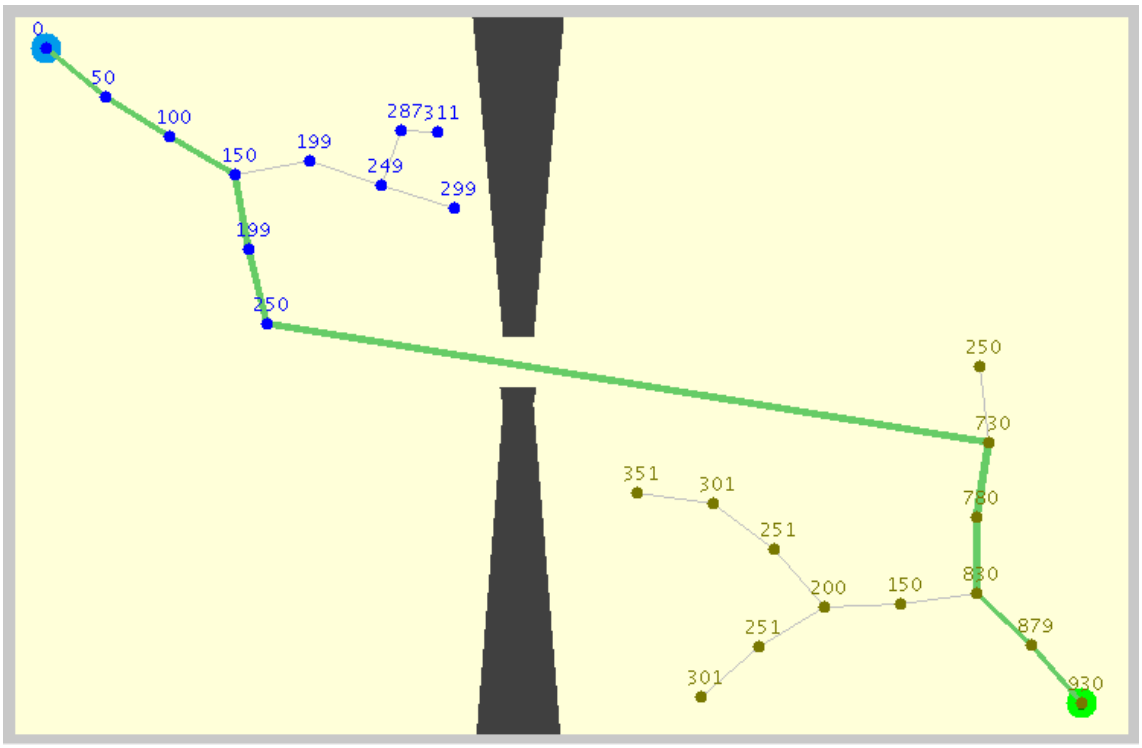
nového bodu s cílovým pomocí funkce *mergeWithGoal()*. U dvoustromové verze se při hladovém přístupu využívá funkce *mergeRRTGreedy()* (viz algoritmus 7.4), při normálním pak *mergeRRTNotGreedy()*, která se snaží spojit nový bod s některým bodem ze druhého stromu.

Pokud je generování RRT stromu/stromů ukončeno úspěšně, ve druhé fázi je vyhledána cesta od startu k cíli. V případě jednostromové simulace pomocí *searchPath()* jako posloupnost cesty od cíle ke kořeni stromu (tedy do počátečního bodu robota), v případě dvoustromové simulace pak pomocí funkce *searchTwoTreesGreedy()* pro hladový přístup, nebo funkcí *searchTwoTrees()* pro normální. U obou přístupů dvoustromových simulací je cesta získána podobně – spojením dvou cest od bodu, který stromy spojuje, do jejich kořenů.

RRT applet obsahuje oproti základním nastavením (zmíněným v části 4.6) několik speciálních. Jedná se o výše zmíněný výběr jedno- nebo dvoustromové simulace (*Ariadne's Clew* nebo *Two trees (bidirectional)*), možnost nastavení hladového přístupu při expanzi stromu (*Greedy step size*), nastavení přidávání pouze posledního bodu při hladovém přístupu (*Add only end point if greedy*) a stjně jako u EST volba *Show all random points*, která povoluje zobrazování všech bodů, které jsou náhodně generovány, i když potom nejsou přidány do grafu.

Pro expanzi RRT stromu je důležitá hodnota délky kroku (*Step size*), která definuje, v jaké vzdálenosti od bodu nejbližšího náhodnému bodu se bude algoritmus pokoušet vytvořit nový bod pro přidání do stromu. Pokud je tato hodnota příliš malá, bude se vytvářet zbytečně mnoho bodů a strom bude růst ve volném prostoru pomaleji. Pokud by však tato hodnota byla příliš velká, vzrostla by pravděpodobnost, že body v této vzdálenosti budou kolidovat s překážkami.

Posledním speciálním nastavením RRT appletu je hodnota *Max. attempts to merge trees*, která určuje maximum pokusů o spojení stromů ve funkci *mergeRRTGreedy()*.




Obrázek 7.3: Ukázka appletu s nalezenou cestou pomocí dvou stromů RRT. Počáteční bod je vlevo nahoře, cílový vpravo dole – na druhé straně prostoru rozděleného úzkým průchodem. Stromy jsou znázorněny jednotlivými uzly (u každého uzlu je zobrazena vzdálenost od počátečního bodu, spočítaná při vyhledávání nejkratší cesty) a hranami, překážky šedými polygony a nalezená nejkratší cesta je vyznačena silnější čarou od počátku do cíle. V této ukázce byl pro expanzi stromů použit pouze algoritmus 7.2.

# Kapitola 8

# SRT algoritmus – Sampling-Based Roadmap of Trees

Algoritmus SRT (Sampling-Based Roadmap of Trees) stojí na pomezí mezi jednodotazovými a vícedotazovými algoritmy. Jeho principem je totiž vytvoření grafu (roadmapy), přičemž k jeho konstrukci se jako *spojovací funkce* používá některý z jednodotazových algoritmů.

Z tohoto pohledu je tedy SRT vícedotazový – jakmile je jednou vytvořen graf, můžeme jej opakovaně využívat k hledání cesty mezi libovolnými body prostoru. Na druhou stranu je ale SRT vhodný i k použití jen pro jeden dotaz, neboť může být efektivnější, než samotné jednodotazové algoritmy [2].

## 8.1 Vytvoření SRT grafu

Jako u všech pravděpodobnostních algoritmů, je i u SRT nutné nejprve vytvořit graf (roadmapu). Na začátku pomocí některé vzorkovací funkce, například s rovnoměrným rozložením pravděpodobnosti, postupně vygenerujeme $n$ volných konfigurací v prostoru. Potom položíme každý z těchto bodů jako kořen stromu $T_1, \ldots, T_n$, ty přidáme do grafu $G_T$ a u každého stromu provedeme $l$ pokusů o jeho rozšíření pomocí některého z jednodotazových algoritmů, nejčastěji pomocí EST nebo RRT.

Po této úvodní fázi máme tedy graf $G_T$ obsahující jako vrcholy jednotlivé stromy $T_1, \ldots, T_n$ ukotvené v původně navzorkovaných bodech prostoru. Dalším krokem je propojení těchto stromů, které se provádí podle algoritmu *Connect SRT* (8.1):

Postupně pro každý strom $T_i$ najdeme jeho $k$ nejbližších sousedů a $r$ dalších náhodných stromů a označíme je jako $N_{T_i}$. Vzdálenost stromů určujeme zprůměrováním všech bodů stromu do jednoho fiktivního řídícího a pak už určením vzdálenosti těchto dvou řídících bodů stromů. Poté postupně zpracováváme všechny stromy $T_j$ v $N_{T_i}$ – zkontrolujeme, zda už nejsou spojeny s $T_i$ a v případě že ne, tak ze zpracovávaného stromu $T_i$ náhodně vybereme několik bodů a ty se pokoušíme napojit na jim nejbližší bod z každého stromu $T_j$. Pokus o spojení stromů provádíme nejprve rychlým jednoduchým spojovacím algoritmem (spojujícím body úsečkou) a teprve v případě neúspěchu pak robustním algoritmem pro spojování dvou stromů, například *Connect RRT* (7.3,).

**Algoritmus 8.1 Connect SRT – algoritmus pro propojení stromů [2].**

***Vstup:***
*$V_T$: seznam všech stromů*
*$k$: počet nejbližších sousedních stromů určených pro napojení*
*$r$: počet náhodných stromů určených pro napojení*
***Výstup:***
*Graf $G_T = (V_T, E_T)$ složený ze stromů*

```
1:  E_T ← ∅
2:  for all T_i ∈ V_T do
3:      N_Ti ← k sousedních a r náhodných stromů z V_T stromu T_i
4:      for all T_j ∈ N_Ti do
5:          if T_i a T_j nejsou součástí jedné komponenty grafu G_T then
6:              merged = FALSE
7:              S_i ← seznam náhodně vybraných bodů z T_i
8:              for all q_i ∈ S_i and merged = FALSE do
9:                  q_j ← bod z T_j nejbližší bodu q_i
10:                 if Δ(q_i, q_j) then
11:                     E_T ← E_T ∪ {(T_i, T_j)}
12:                     merged = TRUE
13:                 end if
14:             end for
15:             if merged = FALSE and MergeTrees(T_i, T_j) then
16:                 E_T ← E_T ∪ {(T_i, T_j)}
17:             end if
18:         end if
19:     end for
20: end for
```

## 8.2 Dotazovací fáze

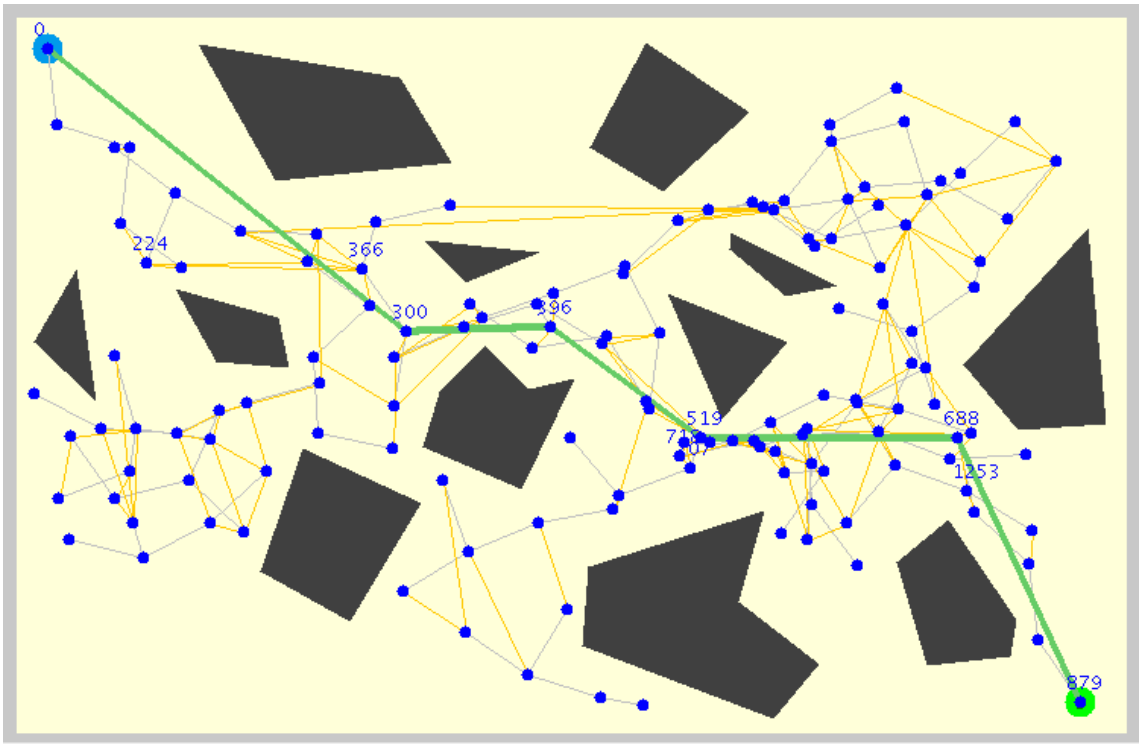
Výsledkem algoritmu *Connect SRT* (8.1) je graf, který je složen z několika, nebo v ideálním případě z jedné komponenty (spojených stromů). Pro vyhledání cesty v prostoru pak stačí napojit počáteční a cílový bod do tohoto grafu.

Napojení se dá provést opět pomocí stromů. Do $q_{init}$ a $q_{goal}$ umístíme kořeny nových stromů a v několika krocích do nich přidáme uzly. Potom můžeme tyto stromy, podobně jako v algoritmu *Connect SRT* (8.1), zkusit napojit k ostatním stromům z grafu $G_T$. Pokud se povede počáteční i cílový strom napojit ke stejné komponentě grafu, pak je hledaná cesta úspěšně nalezena [2].

## 8.3 Vlastnosti SRT

SRT algoritmus zaujímá tak trochu výsadní postavení mezi pravděpodobnostními algoritmy, protože jednou z jeho zajímavých vlastností je, že po nastavení určitých parametrů degenruje na PRM, nebo také na EST, případně RRT algoritmus [2].

Pokud nastavíme počet kroků pro počáteční expanzi stromů na nulu, pak z každého stromu zůstane pouze kořen, tedy jeden, bod, stejně jako u PRM. A pokud dále nastavíme




Obrázek 8.1: Ukázka SRT appletu.

počet opakování druhé spojovací funkce (řádek 15 algoritmu *Connect SRT*, 8.1) také na nulu, budou se body spojovat pouze pomocí základní spojovací funkce – úsečkou (řádek 10). S takovými parametry se pak ze SRT stává obyčejný PRM algoritmus.

Druhým extrémem je nastavit na počátku nula bodů a tedy žádný strom pro přidání do grafu. V takovém případě neproběhne vytvoření grafu a algoritmus přejde rovnou k dotazovací fázi, ve které se použije některý jednodotazový algoritmus pro vyhledání cesty. Při této variantě pak SRT degeneruje na algoritmus použitý v dotazovací fázi, tedy na EST nebo RRT.

SRT je sám o sobě velmi výkonný algoritmus, navíc je vhodný pro paralelizaci. Vytváření grafu totiž spočívá v přidávání nových bodů k navzájem nezávislým stromům. V první fázi tak lze úplně oddělit tyto výpočty, což je výhodou zejména pro náročnější vícedimenzinální úlohy [2].

## 8.4 Implementace SRT appletu

Implementace SRT appletu kombinuje upravené algoritmy použité jak u PRM, tak u RRT, který je interně využit pro expanzi stromů.

SRT pracuje převážně na úrovni grafů, takže pro uchování dalších informací o nich (pozice těžiště pro výpočet vzdáleností grafů, informace o komponentě, ke které graf náleží, vzdálenost od počátečního bodu, informace o napojení na sousední grafy...) byla v *SRTSim* vytvořena vnitřní třída *SRTGraph*.

Vlastní SRT algoritmus probíhá v šesti fázích:

1. Generování náhodných bodů jako kořenů nových stromů
2. Expanze stromů
3. Spojení stromů do jednoho grafu (roadmapy)
4. Přidání stromů z počátečního a cílového bodu do grafu
5. Vyhledání cesty a její případné vyhlazení
6. Přesun robota do cíle

Kromě poslední jsou všechny fáze realizovány funkcemi přímo ze třídy *RRTSim*. Poslední je stejně jako u ostatních appletů vykonávána funkcí *moveRobot()*, zděděnou ze *SimRoadmap* (viz. 4).

V první fázi jsou, podobně jako u PRM algoritmu, náhodně vygenerovány body ve volném konfiguračním prostoru. Tyto body nejsou ale jako u PRM vkládány do jednoho grafu, ale každý z nich se stává kořenem jiného grafu. První fázi realizuje funkce *generatePoints()*.

Ve druhé fázi funkce *expandTrees()* postupně expanduje všechny body a z každého z nich se snaží vytvořit pomocí algoritmu RRT strom obsahující $pointsInTree$ bodů.

Třetí fáze, realizovaná funkcí *connectSRT()*, postupně ke každému stromu vybírá určitý počet jeho nejbližších sousedů a dalších náhodných stromů. Poté vždy náhodně vybere několik bodů z daného stromu a k němu najde nejbližší sousední body z vybraného sousedního/náhodného stromu. Pokud lze tyto dva body spojit, stromy jsou propojeny do jedné komponenty a je v nich uchována informace o bodech, které je propojují.

Funkce *addStartAndGoal()* ve čtvrté fázi vytváří stromy z počátečního a cílového bodu, podobně jako u ostatních stromů ve druhé fázi, a pokouší se je připojit (jako ve třetí fázi) k ostatním stromům. Pokud je tato fáze úspěšná a grafy navíc zachycují volný konfigurační prostor natolik dobře, že jsou všechny spojeny v jednu komponentu, pak je zaručeno, že existuje cesta od počátku do cíle.

Vyhledání cesty má na starost funkce *searchPath()*. Ta nejprve vyhledá nejkratší cestu na úrovni stromů Dijkstrovým algoritmem (pro vyhledávání využívá abstrakci stromů jako bodů – jejich těžiště) a finální cestu získá pomocí funkce *computeLocalTreePath()*, která vrací cestu jedním stromem od jeho jedné hranice (bodu) se sousedním stromem ke druhé.

Poslední fáze je, stejně jako u ostatních pravděpodobnostních algoritmů, realizována funkcí *moveRobot()* ze třídy *SimRoadmap*.

U SRT appletu je možné měnit základní parametry, které už byly zmíněny v kapitole 4, a také několik dalších, souvisejících se samotnými grafy.

Počet stromů, které mají být v simulaci (*Trees to generate in the roadmap*), je jinými slovy také počet kořenových bodů, které se mají v první fázi vygenerovat. Tato hodnota tedy odpovídá proměnné *pointCount*, jak již bylo zmíněno v kapitole 4.

Dále je možné nastavit počet bodů, který se má generovat v každém stromu (*pointsInTree*). Stejně jako u předchozího nastavení je nutné najít kompromis mezi příliš nízkou hodnotou (pak by stromy byly méně rozvětvené a bylo by obtížnější je propojit) a zbytečně vysokou hodnotou (ta by způsobovala přehuštění simulačního prostoru, což by mělo za následek pomalejší běh simulace).

Délka kroku (*Step size*) se vztahuje k RRT algoritmu, který je využíván pro expanzi jednotlivých stromů. Podrobnosti k tomuto nastavení byly uvedeny v části 7.4.

Další nastavení souvisí s grafy: počet sousedních (*Neighbor trees to select*) a náhodných (*Random trees to select*) stromů pro pokus o napojení a počet bodů z každého stromu, které se budou napojování účastnit (*Random points to select from tree*).

Součástí nastavení SRT appletu je, stejně jako u obou jednodotazových algoritmů, možnost zobrazování všech bodů, které jsou náhodně generovány, i když potom nejsou přidány do grafu – *Show all random points*.

# Kapitola 9

# Závěr

V rámci bakalářské práce jsem vytvořil čtyři demonstrační java applety. Tyto applety pro vizualizaci algoritmů hledání cesty pracují se základními druhy pravděpodobnostních algoritmů – PRM, EST, RRT a SRT.

Při implementaci appletů jsem se soustředil zejména na jejich názornost a možnosti nastavení parametrů. Během simulace je možné zpomaleně zobrazovat jednotlivé probíhající kroky a informace o nich. Názornost podporuje také zvýrazňování právě zpracovávaných uzlů, hran a grafů v simulaci. U každého appletu je možné experimentovat se všemi důležitými nastaveními, takže jsou vhodné jak pro základní seznámení se s principy pravděpodobnostních algoritmů, tak pro vyzkoušení vlivu různých nastavení na jejich chování. Z výše uvedených vlastností appletů vyplývá, že jejich implementace PA není vhodná pro využití v praxi, ale hodí se k výukovým účelům.

Při vytváření EST appletu jsem použil vlastní modifikaci pro výběr buněk k expanzi stromu, která zefektivňuje celý algoritmus. Experimentální srovnání této modifikace a jeho rozbor se nachází v části 6.5.

Bakalářská práce kromě appletů také zahrnuje teoretický popis pravděpodobnostních algoritmů včetně diskuze nad implementačními detaily, přibližuje využití PA a popisuje jednotlivé algoritmy. Tato textová část práce byla společně s applety a jejich zdrojovými kódy umístěna na vytvořené webové stránky na adrese
`http://www.stud.fit.vutbr.cz/~xkvasn03/bakalarka/`.

# Dodatek A

# Základní pojmy

## Konfigurace robota

Stav robota, který lze přesně popsat určitým počtem proměnných. Počet proměnných odpovídá dimenzím prostoru, ve kterém bude algoritmus hledání cesty pracovat (podrobněji viz 1.3) a závisí na dimenzích prostoru, ve kterém se robot pohybuje, na dimenzích robota a případně na počtu kloubů robota. Počet proměnných tak odpovídá počtu stupňů volnosti robota.

## Volná konfigurace (collision-free configuration)

Je to taková konfigurace, ve které robot žádným bodem svého objemu nekoliduje překážkou.

## Konfigurační prostor (configuration space)

$N$-rozměrný prostor, který zahrnuje všechny možné konfigurace robota. Značí se $C$.

## Volný konfigurační prostor (free configuration space)

$N$-rozměrný prostor, který zahrnuje všechny možné volné konfigurace robota. Značí se $C_{free}$.

## Cesta

V kontextu pravděpodobnostních algoritmů se jedná o posloupnost volných konfigurací robota, která spojuje startovní a cílovou pozici.

## Roadmap

Graf zachycující konkrétní konfigurační prostor. Obvykle zachycuje volný konfigurační prostor, ale u některých *lazy* modifikací algoritmů jsou tolerovány i stavy, kdy části grafu kolidují s překážkami.

## Strom

Speciální typ souvislého grafu, který neobsahuje kružnice. Jeho dobře využitelnou vlastností je, že z každého jeho uzlu vede právě jedna cesta ke kořeni.

## Stupeň volnosti (degree of freedom, DOF)

Odpovídá nezávislé proměnné popisující konfiguraci robota. Příkladem jsou tři stupně volnosti u *problému stěhování pohovky* (*sofa mover's problem*) – dva popisují polohu řídícího bodu ve dvourozměrném prostoru a třetím stupněm volnosti je údaj o natočení pohovky (podrobněji viz 1.3).

## Lazy modifikace algoritmů

Jedná se o takové modifikace algoritmů, ve kterých jsou do grafu (roadmapy) přidávány i konfigurace, které kolidují s překážkami.

## Algoritmus spojování dvou konfigurací ($\Delta$, local planner)

Algoritmus, který má za úkol v grafu spojit dvě konfigurace robota. V nejjednodušším případě je symetrický (najde stejnou cestu při hledání z A do B i z B do A) a deterministický (při každém volání pro dva stejné body vrátí stejnou cestu), tedy se například snaží spojit dvě konfigurace v prostoru úsečkou. Výkonnější *local planner* může vytvářet složitější křivky, nebo může být složen z některého jednodotazového pravděpodobnostního algoritmu (podrobněji viz 3.4).

## Holonomický a neholonomický robot (holonomic, nonholonomic robot)

Pohyblivé roboty lze rozdělit podle toho, zda se dokáží bez omezení pohybovat do všech směrů prostoru, ve kterém se vyskytují. „Příkladem holonomického robota je všesměrový robot, neholonomický je robot pohybující se jako obyčejné auto, protože jeho pohyb do stran je omezen maximálním natočením předních kol“ [9].

# Dodatek B

# Obsah přiloženého CD

Disk CD přiložený k této bakalářské práci obsahuje následující složky:

- **applety** – zdrojové kódy všech java appletů (Java verze 1.6) a jejich součástí
- **technicka_zprava** – zdrojový text této technické zprávy ve formátu LaTeX včetně všech souborů potřebných pro její překlad
- **web** – webové stránky ve formátu HTML (s použitím PHP verze 5); webové stránky jsou též dostupné na adrese `http://www.stud.fit.vutbr.cz/~xkvasn03/bakalarka/`

# Literatura


[1] Choset, H. a kol.: *Principles of Robot Motion: Theory, Algorithms, and Implementations*, MIT Press, June 2005, ISBN 0-262-03327-5.

[2] Kavraki, L. E.: Protein-Ligand Docking, Including Flexible Receptor-Flexible Ligand Docking. [online] `http://cnx.org/content/m11456/latest/`, 2007.

[3] Hsu, D. a kol.: The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners. In *IEEE International Conference on Robotics and Automation*, 2003, str. 25.

[4] LaValle, S. M.: The RRT Page. [online] `http://msl.cs.uiuc.edu/rrt/gallery_car.html`.

[5] LaValle, S. M.: The RRT Page. [online] `http://msl.cs.uiuc.edu/rrt/gallery_alpha.html`.

[6] LaValle, S. M.: *Planning Algorithms*. Cambridge University Press, 2006, 791 s.

[7] Mitul, S.; Latombe J.-C.: Finding Narrow Passages with Probabilistic Roadmaps: The Small-Step Retraction Method. [online] `http://ai.stanford.edu/ latombe/projects/iros05.ppt`, 2005.

[8] RRRobotica, W.: [online] `http://www.rrrobotica.it/atom10_e.htm`.

[9] Šíma, M.: *Plánování cesty robota ve spojitém prostředí*. Diplomová práce, Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2006.