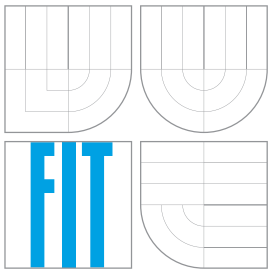
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PLÁNOVÁNÍ POHYBU OBJEKTU V 3D PROSTORU

PATH PLANNING IN 3D SPACE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. FRANTIŠEK NĚMEC

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV ROZMAN, Ph.D.

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Němec František, Bc.**
Obor: Inteligentní systémy
Téma: **Plánování pohybu objektu v 3D prostoru**
Path Planning in 3D Space
Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se s knihovnami DirectX a OpenGL. Nastudujte způsoby modelování jednoduchých geometrických objektů v programech jako je např. Blender. Dále nastudujte metody pro detekce kolizí.
2. Seznamte se s pravděpodobnostním plánováním pohybu ve 3D prostoru.
3. Navrhněte program umožňující uživateli zadat tvar objektu, rozmístění překážek a počáteční a koncový bod pohybu. Program pak sám cestu naplánuje a její průběh vizualizuje.
4. Program implementujte, otestujte na "bug trap" a "hedgehog" benchmarku a navrhněte možná vylepšení.
5. Vytvořte vlastní sadu testů s modely se vzrůstající složitostí, na kterých demonstrujete funkčnost algoritmu.

Literatura:

- Howie Choset et al.,Principles of Robot Motion, ISN 0-262-03327-5
- Radek Sasýn: Plánování pohybu objektu v 3D prostoru, diplomová práce, Brno, FIT VUT v Brně, 2013
- Benchmarky pro plánování 3D pohybu, https://parasol.tamu.edu/groups/amatogroup/benchmarks/mp/

Při obhajobě semestrální části projektu je požadováno:

- První tři body zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Rozman Jaroslav, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

L. S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tato práce se zabývá problémem plánování pohybu objektu ve 3D prostoru. Cílem je vytvořit program, který uživateli umožní vytvořit scénu, ve které bude hledání cesty probíhat, automaticky pak cestu vyhledá a nakonec ji bude vizualizovat. Práce je zaměřena na pravděpodobnostní algoritmy, které jsou spolu s důležitými pojmy popsány v teoretické části. V praktické části je popsán návrh a implementace aplikace. Nakonec je provedeno několik experimentů pro porovnání výkonu jednotlivých algoritmů a demonstraci funkčnosti programu.

Abstract

This paper deals with the problem of object path planning in 3D space. The goal is to create program which allows users to create a scene used for path planning, perform the planning and finally visualize path in the scene. Work is focused on probabilistic algorithms that are described in the theoretical part. The practical part describes the design and implementation of application. Finally, several experiments are performed to compare the performance of different algorithms and demonstrate the functionality of the program.

Klíčová slova

Plánování pohybu, hledání cesty, 3D prostor, pravděpodobnostní algoritmy, PRM, EST, RRT, R-RRT, RR-RRT.

Keywords

Motion planning, path planning, 3D space, probabilistic algorithms, sampling-based algorithms, PRM, EST, RRT, R-RRT, RR-RRT.

Citace

NĚMEC, František. *Plánování pohybu objektu v 3D prostoru*. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Rozman Jaroslav.

Plánování pohybu objektu v 3D prostoru

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
František Němec
22. května 2016

Poděkování

Tímto bych rád poděkoval Ing. Jaroslavu Rozmanovi, Ph.D. za pomoc a odborné vedení mé práce.

Obsah

Kapitola 1

Úvod

Roboti jsou čím dál složitější a pracují v komplexnějších podmínkách, takže je nutné vytvořit algoritmy, které se takovým podmínkám dokáží přizpůsobit. Jeden z důležitých problémů v robotice je automatické plánování pohybu, neboli hledání cesty. Cílem je specifikovat cíl na vyšší abstraktní úrovni a algoritmus automaticky vytvoří plán složený z jednoduchých úkonů jako posun, rotace nebo přímo ovládání jednotlivých motorů robota.

Jedním typem plánovacích algoritmů, které takový problém řeší, jsou pravděpodobnostní algoritmy. Pravděpodobnostní algoritmy pracují na principu vzorkování prostoru, čímž vytvoří graf prostoru, nad kterým lze hledat cestu pomocí klasických algoritmů hledání cesty v grafu. Plánovací algoritmy však nejsou omezeny jen na použití u robotů, ale nalezly využití i v jiných odvětvích, jako například verifikace průmyslových plánů, animace digitálních postav, mapování neprozkoumaného prostředí nebo návrh nových léků.

Cílem práce je vytvořit program pro řešení a demonstraci problému plánování pohybu ve 3D prostoru. K tomu je třeba se seznámit s grafickými knihovnami DirectX a OpenGL a nastudovat způsoby modelování 3D objektů v programech jako je např. Blender[1]. Dále je nutné nastudovat příslušné algoritmy pro plánování pohybu objektů ve 3D prostoru. Program musí obsahovat uživatelské rozhraní pro vytvoření scény (např. zadání tvaru objektu, rozmístění překážek, počáteční a koncový bod pohybu), nastavení parametrů algoritmu hledání cesty a nakonec vizualizaci nalezené cesty. Na závěr bude provedeno testování a experimentování na různých příkladech, které budou demonstrovat funkčnost výsledného programu.

První kapitola se zabývá možnými aplikacemi a využitím plánovacích algoritmů. Následující kapitola obsahuje popis vybraných pravděpodobnostních algoritmů, mezi které patří Probabilistic Roadmap (PRM), Expansive Spaces Trees (EST), Rapidly-exploring Random Trees (RRT), Retraction-based RRT (R-RRT), vlastní rozšíření Regulated R-RRT (RR-RRT) a Sampling-based Roadmap of Trees (SRT), spolu s popisem jednotlivých fází a možnými modifikacemi. Následuje kapitola o návrhu, implementaci a schopnostech vytvořeného programu. Na závěr je provedeno několik experimentů s cílem ověřit výkon implementovaných algoritmů a celkovou funkčnost programu.

[1] Blender 3D graphics and animation software: https://www.blender.org/

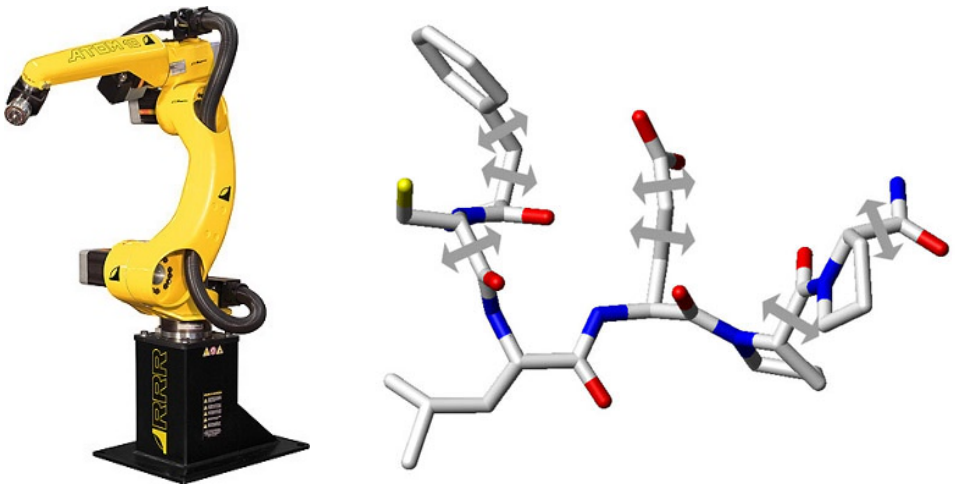
Kapitola 2

Aplikace plánovacích algoritmů

Plánovací algoritmy mají mnohá využití zejména v robotice. Typickým použitím je plánování pohybu robotického ramene u výrobní linky. Úkolem ramene je přesun/manipulace s objektem, přičemž se rameno musí vyhnout překážkám (ostatním konstrukcím montážní linky) a zároveň nesmí kolidovat samo se sebou. Lze řídit i pohyb několika robotů současně, například při součinnosti více robotických ramen. Plánování pohybu ramene je přitom optimalizováno s ohledem na co nejkratší trajektorii.

Dalším použitím je postup sestavení výrobku z několika mechanických součástek. Způsob sestavení výrobku je významnou částí výroby. Při výrobě např. nového motoru je většinou k dispozici i jeho model (např. CAD model). Tento model lze použít pro plánování sestavení celého motoru. Navíc se tímto způsobem dají odhalit nedostatky v návrhu ještě před samotnou výrobou, a tak předejít finančním a časovým ztrátám. Opačně může být model použit pro zjištění způsobu dekonstrukce výrobku při nutnosti údržby nebo opravy.

Nezanedbatelnou aplikací plánovacích algoritmů je biologie, zejména pak skládání a dokování proteinů. Způsob složení a výsledná struktura má velký vliv na funkci proteinu. Obal složeného proteinu totiž definuje, jak a s čím může protein reagovat. Způsob, jakým se protein skládá, je tedy důležitý k pochopení biomolekulárních interakcí. Pochopení takových interakcí může přispět k vývoji nových léků.



Obrázek 2.1: Vlevo je ukázka robotického ramene [11] a vpravo je *stick* reprezentace struktury proteinu spolu se šipkami, které značí možné rotace jednotlivých částí [7].

Kapitola 3

Pravděpodobnostní algoritmy

Plánováním cesty se rozumí nalezení posloupnosti kroků, kterými lze daný objekt (robota) bezkolizně přemístit z počáteční do cílové pozice. Důležitým faktorem je, zda je zkoumaný prostor diskrétní nebo spojitý. V případě diskrétního prostoru lze cestu hledat pomocí známých deterministických algoritmů hledání cesty v grafu. V druhém případě spojitého prostoru nelze algoritmy pro diskrétní prostor použít a je třeba zvolit jiný přístup.

Obecně pravděpodobnostní algoritmy pracují na principu vzorkování prostoru, čímž vytvoří graf prostředí, nad kterým lze hledat cestu pomocí algoritmů hledání cesty v grafu. Aby bylo možné hledat cestu ve spojitém prostředí pomocí pravděpodobnostních algoritmů, musí být toto prostředí předem známé, to znamená, že existuje model tohoto prostředí. Pokud tato podmínka není splněna, nelze pravděpodobnostní algoritmy použít [10]. Bylo dokázáno, že mnohé pravděpodobnostní algoritmy jsou *pravděpodobnostně kompletní*. To znamená, že pravděpodobnost nalezení řešení (v případě, že existuje) konverguje k jedné, jak čas postupuje k nekonečnu [4]. Pravděpodobnostní algoritmy lze rozdělit do tří základních kategorií:

- *Vícedotazové* algoritmy budují graf prostředí, nad kterým lze hledat cestu z různých počátečních a cílových konfigurací. Graf prostředí typicky pokrývá co nejvíce prostoru.
- *Jednodotazové* algoritmy budují strom z předem známé počáteční konfigurace a hledají cestu k cílové konfiguraci.
- *Kombinované* algoritmy kombinují oba předchozí algoritmy. Obecně na ně lze pohlížet jako na vícedotazové algoritmy, protože budují graf prostředí. Při konstrukci grafu využívají jednodotazové algoritmy.

Tato kapitola obsahuje popis konkrétních pravděpodobnostních algoritmů [4, 5, 10] *Probabilistic roadmap* (PRM), *Expansive-Spaces Trees* (EST), *Rapidly-exploring Random Trees* (RRT), *Retraction-based RRT* (R-RRT), *Regulated R-RRT* (RR-RRT) a *Sampling-based Roadmap of Trees* (SRT). Jako první je popsán základní PRM algoritmus. Po kterém následuje rozbor jednotlivých fází algoritmu (způsob vzorkování prostoru, propojovací funkce a optimalizace výsledné cesty) s možnými variantami, které lze aplikovat i na jiné algoritmy. Nakonec následuje popis ostatních algoritmů.

3.1 Základní pojmy

Tato sekce obsahuje popis důležitých pojmů, které se vyskytují v popisu jednotlivých pravděpodobnostních algoritmů.

- **Konfigurace** je stav objektu v prostoru (v následujícím textu se často vyskytuje pojem „uzel grafu“, který právě představuje konfiguraci).
- **Volná konfigurace** je taková konfigurace, při které objekt nekoliduje s žádnou překážkou.
- **Konfigurační prostor** Q je množina všech konfigurací objektu.
- **Volný konfigurační prostor** Q_{free} je množina všech volných konfigurací objektu.
- **Propojovací funkce** $\Delta : (q, q') \in Q_{free} \times Q_{free}$ vrací bezkolizní cestu z konfigurace q do q' nebo `NIL`, pokud taková cesta nemůže být nalezena. Funkce a její parametry jsou detailněji popsány v sekci 3.3.2.
- **Vzdálenostní funkce** $dist : Q \times Q \rightarrow R_0^+$ obvykle definována jako euklidovská vzdálenost: $dist(q', q'') = |emb(q') - emb(q'')|$, kde emb značí vzdálenost od počátku souřadné soustavy. Funkce může zahrnou i rotaci a být zobecněna na tvar $dist(q', q'') = w_t||X' - X''|| + w_r f(R', R'')$, kde w_r a w_t představují váhu rotace a translace, $||X' - X''||$ značí transpoziční vzdálenost a $f(R', R'')$ je funkce aproximace vzdálenosti rotace.

3.2 Probabilistic Roadmap

Probabilistic Roadmap (PRM) [8] patří mezi základní pravděpodobnostní algoritmy. PRM je vícedotazový algoritmus, který pracuje ve dvou fázích. V první fázi je vybudován graf prostředí (roadmap). Ve druhém kroku je vyhledána cesta mezi počáteční a cílovou konfigurací (uzly grafu).

3.2.1 Budování grafu prostředí

V první fázi je vytvořen neorientovaný graf (roadmap) prostoru $G = (V, E)$, kde V představuje množinu uzlů, neboli množinu volných konfigurací robota, tedy $V \subset Q_{free}$. Množina E obsahuje hrany (q_1, q_2), které značí bezkolizní cestu mezi konfiguracemi q_1 a q_2.

Postup vytvoření grafu, bez implementačních detailů, je popsán algoritmem 3.1. Vstupem algoritmu je počet uzlů výsledného grafu n a počet nejbližších sousedů, resp. počet spojení uzlu s ostatními uzly. Na počátku je inicializován samotný graf, neboli množina uzlů V a množina hran E. Následuje generování n volných konfigurací (uzlů grafu). Uzly jsou generovány náhodně z volného konfiguračního prostoru. Pokud je nově vygenerovaná konfigurace bezkolizní, je přidána do grafu. Existuje mnoho způsobů, jak uzly generovat, některé jsou popsány v následující sekci 3.3.1. Po vytvoření požadovaného počtu uzlů jsou uzly propojeny. Propojení probíhá na základě vzdálenostní funkce, která vybere k nejbližších uzlů a ty jsou propojeny podle spojovací funkce Δ (popsáné v sekci 3.3.2).

Je důležité vhodně zvolit vstupní parametry. V případě, že počet uzlů n bude příliš malý, nebude pracovní prostor dostatečně pokryt a cesta nemusí být nalezena. V opačném případě velkého počtu uzlů může být celý algoritmus zpomalen. Podobné je to s počtem

Algoritmus 3.1 Algoritmus pro vytvoření grafu prostředí

Vstup:

n: počet uzlů výsledného grafu

k: počet nejbližších sousedů každého uzlu

Výstup:

Graf $G = (V, E)$

1. $V \leftarrow \emptyset$
2. $E \leftarrow \emptyset$
3. **while** $|V| < n$ **do**
4. **repeat**
5. $q \leftarrow$ náhodná konfigurace z Q
6. **until** q je volná konfigurace
7. $V \leftarrow V \cup \{q\}$
8. **end while**
9. **for all** $q \in V$ **do**
10. $N_q \leftarrow k$ nejbližších uzlů podle vzdálenostní funkce *dist*
11. **for all** $q' \in N_q$ **do**
12. **if** $(q, q') \notin E$ **and** $\Delta(q, q') \neq$ `NIL` **then**
13. $E \leftarrow E \cup \{(q, q')\}$
14. **end if**
15. **end for**
16. **end for**

sousedů k, ale zde je nutné větší pozornosti. Právě nalezení nejbližších uzlů a jejich propojení je na celém PRM algoritmu výpočetně nejnáročnější.

3.2.2 Konstrukce cesty

Druhá fáze algoritmu zahrnuje napojení počátečního a cílového uzlu a nalezení samotné cesty. Protože je algoritmus vícedotazový, lze hledání cesty provádět z různých počátečních a cílových konfigurací, bez nutnosti pokaždé vytvářet nový graf prostředí.

Prvním krokem je napojení počátečního a cílového uzlu do grafu vytvořeného v první fázi. Propojení probíhá podobně jako při vytváření uzlů. Je vyhledán určitý počet sousedních uzlu, které se algoritmus snaží propojit s počátečním, resp. cílovým uzlem. Na rozdíl od propojení uzlů grafu, v tomto případě propojování končí při prvním úspěšném propojení. Po napojení počátečního a koncového uzlu je na řadě samotné vyhledání cesty. K tomu lze použít jakýkoli algoritmus hledání nejkratší cesty v grafu, např. Dijkstrův nebo A* algoritmus. Algoritmus 3.2 obsahuje postup napojení uzlů a následné vyhledání cesty. Pokud není cesta nalezena, prostor nejspíš nebyl dostatečně pokryt. Když tato situace nastane, vytvořený graf může být rozšířen o další uzly a algoritmus nalezení cesty znovu spuštěn.

Přirozeně bylo vytvořeno několik rozšíření PRM algoritmu. Mezi jedno rozšíření patří *Obstacle-Based PRM* (OBPRM) [2] algoritmus. Cílem algoritmu je generovat konfigurace blízko překážek. V první fázi algoritmus náhodně generuje několik konfigurací. Pro každou kolidující konfiguraci algoritmus zvolí náhodný směr a v tomto směru hledá volnou konfiguraci. Pokud je volná konfigurace nalezena, je provedeno binární hledání s cílem nalezení konfigurace co nejblíž k překážce.

Algoritmus 3.2 Algoritmus pro nalezení cesty mezi počáteční a cílovou konfigurací

Vstup:

q_{init}: počáteční konfigurace
q_{goal}: cílová konfigurace
k: počet nejbližších sousedů každého uzlu
$G = (V, E)$: graf prostředí vytvořen algoritmem 3.1

Výstup:

cesta P z q_{init} do q_{goal} v G nebo `failure` v případě neúspěchu

1. $N_{q_{init}} \leftarrow k$ nejbližších sousedů počáteční konfigurace q_{init} z V podle funkce $dist$
2. $N_{q_{goal}} \leftarrow k$ nejbližších sousedů cílové konfigurace q_{goal} z V podle funkce $dist$
3. $V \leftarrow V \cup \{q_{init}, q_{goal}\}$
4. $q' \leftarrow$ nejbližší sousední uzel uzlu q_{init} 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. $q' \leftarrow$ nejbližší sousední uzel uzlu q_{init} z $N_{q_{init}}$
10. **end if**
11. **until** uzel q_{init} byl úspěšně napojen **or** $N_{q_{init}} = \emptyset$
12. $q' \leftarrow$ nejbližší sousední uzel uzlu q_{goal} 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. $q' \leftarrow$ nejbližší sousední uzel uzlu q_{goal} z $N_{q_{goal}}$
18. **end if**
19. **until** uzel q_{goal} byl úspěšně napojen **or** $N_{q_{goal}} = \emptyset$
20. $P \leftarrow$ nejkratší cesta z q_{init} do g_{goal} v G
21. **if** $P \neq \emptyset$ **then**
22. **return** P
23. **else**
24. **return** `failure`
25. **end if**

3.3 Rozbor hlavních fází algoritmu

V popisu PRM algoritmu bylo zmíněno několik důležitých části, jako způsob vzorkování prostoru nebo způsob propojení jednotlivých konfigurací. Existují různé způsoby, jak k těmto problémům přistoupit a některé z nich jsou popsány v této sekci.

3.3.1 Vzorkování prostoru

Způsob vzorkování prostoru je jednou z nejdůležitějších, ne-li nejdůležitější částí algoritmu, protože na něm závisí jeho efektivita. Nejzákladnějším přístupem je generovat konfigurace s rovnoměrným rozložením pravděpodobnosti. Pro každý stupeň volnosti vygenerovat náhodnou hodnotu z povoleného rozsahu. Povolený rozsah je dán hranicemi prostoru nebo možnými rotacemi objektu. Efektivita tohoto přístupu klesá s větším počtem překážek a při výskytu tzv. *úzkých průchodů*. Čím více překážek se v prostoru vyskytuje, tím je větší

šance, že vygenerovaná konfigurace nebude volná a generování se musí opakovat. Pokud robot musí projít *úzkým průchodem*, rovnoměrné rozložení nemusí zajistit vygenerování konfigurace právě v tomto průchodu.

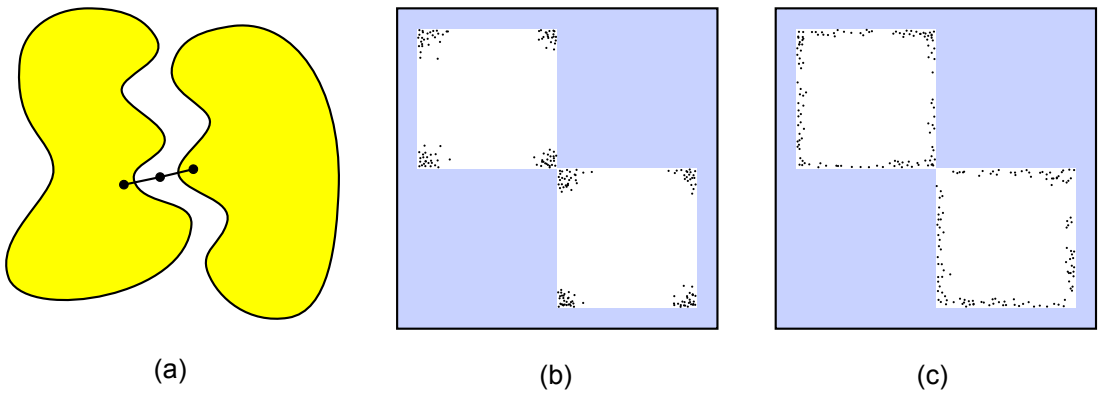
Nedostatečné vzorkování u překážek nebo v *úzkých průchodech* lze řešit několika způsoby: filtrovacími, dilatačními nebo retrakčními algoritmy. Filtrovací a retrakční algoritmy se snaží zachytit více prostoru u překážek nebo v *úzkých průchodech* ve fázi generování nových konfigurací. Dilatační algoritmy nejdříve zmenší všechny překážky, čímž se zvýší šance na vygenerování volné konfigurace.

Jedním z filtrovacích algoritmů je *Randomized Bridge Builder* algoritmus [6]. Prvně jsou vygenerovány dvě náhodné konfigurace s rovnoměrným rozložením q' a q''. Pokud obě konfigurace kolidují s překážkami, je vytvořena konfigurace q_m ve středu úsečky mezi q' a q''. V případě, že je q_m volná konfigurace, je přidána do grafu prostředí. Tímto způsobem jsou generovány konfigurace uvnitř *úzkých průchodů*.

Algoritmus 3.3 Algoritmus Randomized Bridge Builder

Vstup:
n: počet generovaných konfigurací

1. **repeat**
2. $q' \leftarrow$ náhodná konfigurace z Q
3. $q'' \leftarrow$ náhodná konfigurace z Q
4. **if** $q' \notin Q_{free}$ **and** $q'' \notin Q_{free}$ **then**
5. $q_m \leftarrow$ konfigurace uprostřed úsečky mezi q' a q''
6. **if** $q_m \in Q_{free}$ **then**
7. q_m je přidána do grafu prostředí
8. **end if**
9. **end if**
10. **until** bylo vygenerováno n konfigurací

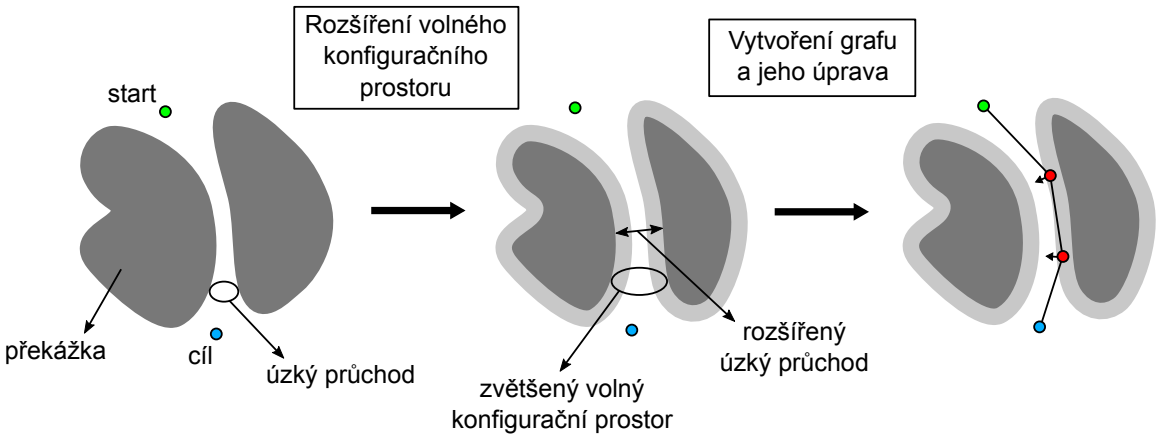


Obrázek 3.1: Filtrovací algoritmy pro vzorkování úzkých průchodů a okolí překážek. Most přes úzký průchod RRB algoritmu (a), Randomized Bridge Builder (b) a Gaussovské vzorkování (c).

Dalším filtrovacím algoritmem, podobným k RBB, je *Gaussovské vzorkování* [3]. To pracuje na principu přidávání takových konfigurací, které leží blízko překážky. Nejprve je vygenerována konfigurace q_1 s rovnoměrným pravděpodobnostním rozložením. Následně je zvolena hodnota *step* podle normálního rozložení, která slouží pro generování nové konfigurace

q_2, která se nachází ve vzdálenosti *step* od konfigurace q_1. Pokud obě konfigurace q_1 i q_2 leží ve volném konfiguračním prostoru nebo obě kolidují s překážkami, jsou zahozeny. Konfigurace je uložena jen v případě, že je volná a druhá konfigurace koliduje s překážkou.

Dilatační algoritmy [12] pracují na principu zmenšení překážek (obrázek 3.2), čímž rozšíří volný konfigurační prostor a zvětší tak úzké průchody. V takto upraveném prostoru je jednodušší nalézt cestu. Po nalezení cesty je cesta upravena, aby vyhovovala původnímu prostoru. Vlastní úprava cesty se provádí při umisťování uzlů do grafu (pesimistická varianta), nebo až po nalezení kompletní cesty (optimistická varianta). Nicméně implementace takových algoritmů je náročná z důvodu složité operace zmenšení překážek.



Obrázek 3.2: Postup dilatačních algoritmů. Nejprve jsou zmenšeny všechny překážky, resp. zvětšen volný konfigurační prostor. Následně jsou v takto upraveném prostoru vygenerovány volné konfigurace. Na závěr je výsledná cesta upravena tak, aby vyhovovala původnímu neupravenému prostoru.

Retrakční algoritmy fungují podobně jako filtrovací algoritmy. Jejich cílem je generovat konfigurace ve větší blízkosti požadované oblasti. To může znamenat co nejblíže k překážce nebo naopak do pozice, která se nachází ve stejné vzdálenosti od všech okolních překážek. Konkrétní retrakční algoritmus *Retraction-based RRT* [14, 15] je popsán v sekci 3.7.

3.3.2 Propojovací funkce

Propojovací funkce (Δ, *local planner*) má za úkol spojit dvě volné konfigurace. To je provedeno vytvořením hrany mezi danými uzly. Hrana přitom musí umožnit bezkolizní přesun objektu ze zdrojové do cílové konfigurace.

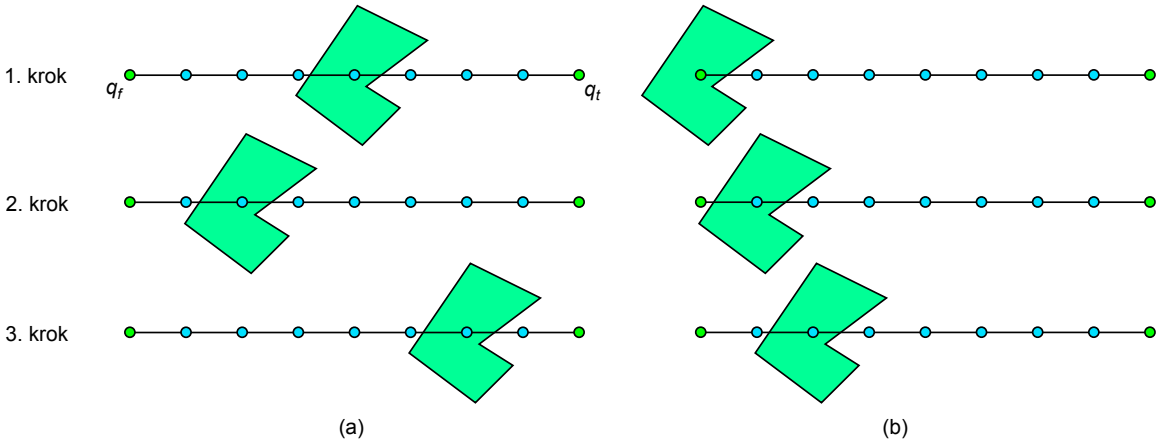
Dvě konfigurace mohou být propojeny prostou úsečkou, ale lze použít i složitější křivky (např. NURBS nebo Bézierovy křivky). Nicméně použití úsečky je časově efektivnější a celková efektivita algoritmu je důležitý parametr. Oba přístupy lze kombinovat a např. v případě nenalezení cesty úsečkou může být provedeno propojení křivkami, které už cestu mohou nalézt.

Obecně propojovací funkce interpoluje všechny konfigurace mezi zdrojovou a cílovou konfigurací. Propojovací funkce křivku postupně vzorkuje a kontroluje, zda je interpolovaná konfigurace bezkolizní. Existují dva základní přístupy k interpolaci:

- *Inkrementální interpolace*: funkce křivku vzorkuje od zdrojové do cílové konfigurace s krokem $step_{size}$ a postupně kontroluje, zda konfigurace na úsečce nekolidují s žádnou překážkou.

- *Interpolace rekurzivním dělením*: křivka je rozdělena na dvě poloviny a prostřední konfigurace je testována na kolizi s překážkami. Pokud konfigurace nekoliduje, jsou obě poloviny znovu rozděleny a vše se rekurzivně opakuje. Dělení končí v případě kolize s překážkou nebo pokud je velikost rozdělené části menší jak $step_{size}$.

Interpolace rekurzivním dělením je obecně rychlejší, protože rychleji odhalí kolizi s překážkou. Výhodou inkrementální interpolace je však možnost uložení poslední nekolizní konfigurace. Důležitým parametrem je velikost $step_{size}$, která by měla být co nejmenší, aby nevznikla situace, kdy by kvůli velkém kroku nebyla detekována překážka. Na druhou stranu při příliš krátkém kroku může být algoritmus znatelně zpomalen.



Obrázek 3.3: Provedení prvních tří kroků interpolačních algoritmů. Interpolace rekurzivním dělením (a) a inkrementální interpolace (b) mezi uzly q_f a q_t.

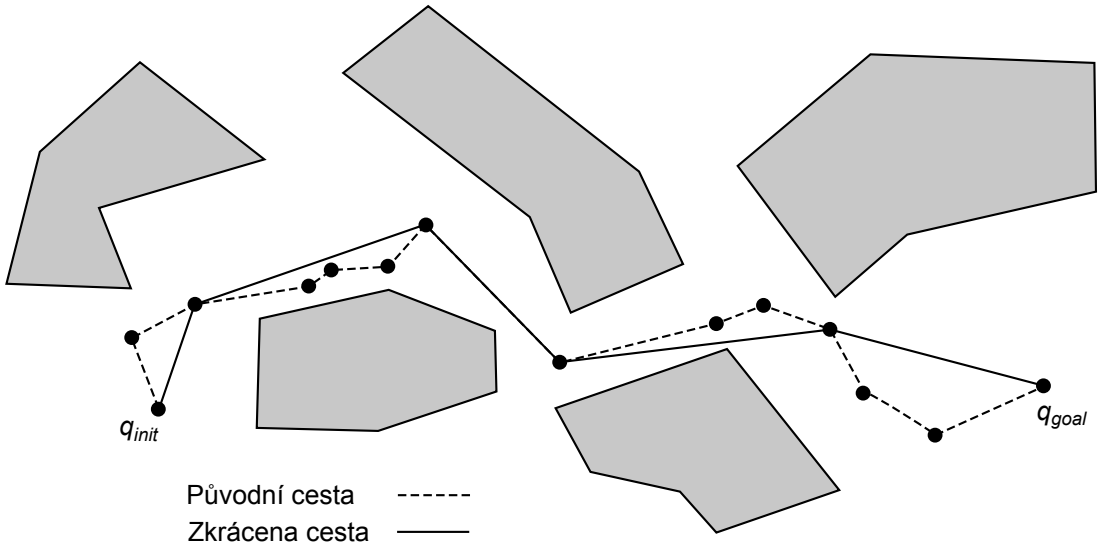
Dalším parametrem propojovací funkce je její symetričnost, tedy zda $\Delta(q, q') = \Delta(q', q)$. Je také nutné brát v úvahu, zda je funkce deterministická. V případě, že není, je třeba se samotným faktem, že lze dvě konfigurace propojit, ukládat i nalezenou cestu pro další použití.

Za účelem zrychlení algoritmu nalezení cesty byla vytvořena tzv. *lazy evaluation* modifikace, která v průběhu vyvářeení uzlů a hran nekontroluje, zda jsou hrany bezkolizní. Test je proveden až v poslední fázi hledání konkrétní cesty, kdy už je graf prostředí vytvořen. Celý algoritmus je tak nezanedbatelně zrychlen, protože detekce kolizí patří mezi výpočetně nejnáročnější operace algoritmu.

3.3.3 Optimalizace cesty

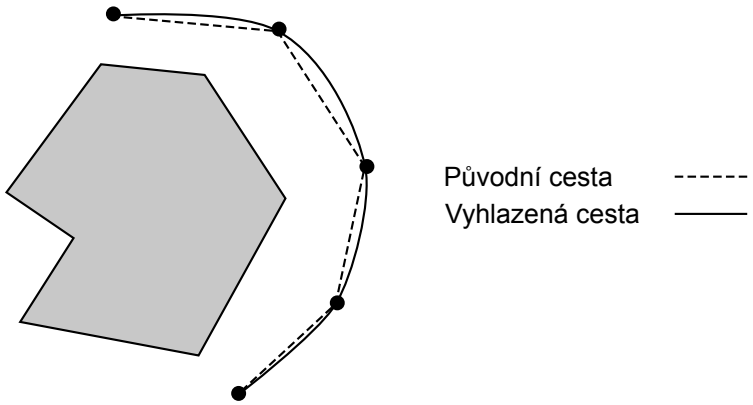
Optimalizace cesty je vylepšení výsledné cesty. Po nalezení konkrétní cesty lze cestu upravit a vylepšit. Optimalizace cesty je časově náročná operace a je vhodné ji provádět až po dokončení celého algoritmu, než při jeho běhu.

Zkrácením cesty se rozumí odstranění nebo propojení uzlů, které nebyly v průběhu plánování propojeny, ale jejich propojení vede ke zlepšení cesty. Jedním z nejjednodušších způsobů, jak cestu zkrátit, je *hladový přístup* (obrázek 3.4). Ten nejdříve zkontroluje, zda je možné propojit počáteční a cílový uzel. Pokud počáteční a cílový uzel nelze propojit, pokračuje se pokusem o propojení počátečního a předposledního uzlu. Po testu propojení počátečního uzlu se všemi ostatními uzly se pokračuje druhým uzlem a je aplikován stejný postup jako u počátečního uzlu. Tímto způsobem jsou zpracovány všechny uzly.



Obrázek 3.4: Zkrácení nalezené cesty pomocí hladového algoritmu

Další úprava se týká vyhlazení cesty. Jednotlivé uzly cesty lze použít jako řídící body některé *spline křivky* (obrázek 3.5) a tím tak celou cestu vyhladit. Vyhlazenou cestu je nutné následně otestovat na kolizi s překážkami.



Obrázek 3.5: Vyhlazení cesty

3.4 Expansive-Spaces Trees

Expansive-Spaces Trees (EST) je jednodotazový algoritmus, který se snaží co nejrychleji pokrýt prostor mezi počáteční a cílovou konfigurací. Na rozdíl od PRM algoritmu, který buduje graf, EST algoritmus buduje strom, který je zakořeněn v počáteční konfiguraci. EST může současně budovat dva stromy, jeden z počáteční konfigurace a jeden z cílové konfigurace. Dvoustromová verze je většinou efektivnější, ale složitější na implementaci. EST algoritmus lze rozdělit na tři hlavní části:

1. Tvorba stromu – generování konfigurací volného prostoru z počáteční konfigurace. V případě dvou stromů současné generování z počáteční a koncové konfigurace.

2. Propojení stromu – propojení stromu s cílovou konfigurací nebo v případě dvoustromové verze propojení obou stromů.

3. Nalezení cesty – pokud je strom propojen s cílovou konfigurací nebo když jsou dva stromy propojeny, cesta z počátku do cíle jistě existuje a zbývá ji jen ve vytvořeném stromu najít. Pro tento účel lze stejně jako u PRM použít jakýkoli algoritmus hledání cesty, např. Dijkstrův nebo A* algoritmus. Nicméně v případě stromu je snadnější a rychlejší výslednou cestu nalézt jako sekvenci konfigurací od listového (cílové) uzlu ke kořenovému (počátečnímu) uzlu. Jelikož každý uzel ve stromu obsahuje jen jeden rodičovský uzel, je výslednou cestou posloupnost rodičovských uzlů.

3.4.1 Tvorba stromu

Přidávání uzlů do stromu je popsáno v algoritmu 3.4. Vstupem algoritmu je počáteční konfigurace q_0, která slouží jako kořen stromu, a počet pokusů o expanzi stromu n. Při rozšíření stromu je vybrán některý z uzlů q_{rand} a ten je expandován. Výběr je proveden podle pravděpodobnostní funkce π_T. V okolí tohoto uzlu je vygenerován nový uzel q_{new} a je proveden pokus o jeho propojení s uzlem q_{rand}. V případě úspěchu je nový uzel (spolu s hranou) přidán do stromu. Tento proces je opakován, dokud nebylo dosaženo požadovaného počtu uzlů.

Algoritmus 3.4 Algoritmus konstrukce EST stromu.

Vstup:
q_0: kořen stromu (počáteční uzel/konfigurace)
n: počet pokusů o expanzi stromu
Výstup:
Strom $T = (V, E)$ s kořenem q_0 a počtem uzlů $\leq n$

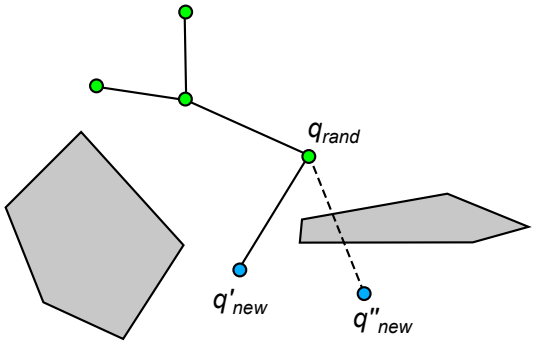
1. $V \leftarrow \{q_0\}$
2. $E \leftarrow \emptyset$
3. **for** $i = 1$ do n **do**
4. $\quad q_{rand} \leftarrow$ náhodně zvolená konfigurace z V s pravděpodobností $\pi_T(q_{rand})$
5. $\quad q_{new} \leftarrow$ náhodná bez-kolizní konfigurace v okolí konfigurace q_{rand}
6. $\quad$ **if** $\Delta(q_{rand}, q_{new}) \neq$ `NIL` **then**
7. $\quad\quad V \leftarrow V \cup \{q_{new}\}$
8. $\quad\quad E \leftarrow E \cup \{(q_{rand}, q_{new})\}$
9. $\quad$ **end if**
10. **end for**
11. **return** T

Nejdůležitější částí jak z pohledu efektivity, tak výpočetní náročnosti, je pravděpodobnostní funkce π_T. Funkce by měla vybírat uzly s cílem co největšího pokrytí volného konfiguračního prostoru. Dobrou volbou je vybírat uzly s co nejmenší hustotou sousedních uzlů. Jedním způsobem měření takové hustoty je přiřadit každému uzlu váhu $w_T(q)$, která počítá počet sousedních uzlů v předem definovaném okolí. Pravděpodobnostní funkci $\pi_T(q)$ pak lze přímo definovat jako inverzní hodnotu k $w_T(q)$.

3.4.2 Spojení stromů

V případě jednostromové verze se algoritmus snaží propojit nějaký z uzlů stromu s cílovým uzlem. Pokud propojení uspěje, cesta zajisté existuje a zbývá jen najít konkrétní posloupnost konfigurací. U dvoustromové verze je třeba propojit některý uzel z prvního stromu s uzlem ze druhého stromu. Propojení probíhá následovně:

1. Pomocí pravděpodobnostní funkce π_T je vybrán některý uzel q_{rand} z prvního stromu.
2. K uzlu q_{rand} je nalezen nejbližší sousední uzel druhého stromu, který je propojen spojovací funkcí Δ.
3. Pokud je spojení úspěšné, stromy jsou navzájem propojeny a tím pádem existuje cesta z počátečního do koncového uzlu.
4. Při neúspěchu je stejný postup aplikován na druhý strom.



Obrázek 3.6: Přidání nové konfigurace do EST. Uzel q_{rand} je vybrán pro expanzi. V okolí uzlu q_{rand} je vygenerován nový uzel q'_{new}, ten je úspěšně propojen s q_{rand} a spolu s hranou je přidán do výsledného stromu. V případě, že by byl vygenerován uzel q''_{new}, nebylo by možné ho s q_{rand} propojit a žádný uzel ani hrana by nebyla do stromu přidána.

3.5 Single-Query Bi-Directional Lazy Collision Checking

Přímou aplikaci *lazy evaluation* způsobu propojovací strategie, popsané v sekci 3.3.2, na EST, vznikl SBL [13] algoritmus. SBL algoritmus, stejně jako EST, vytváří nové uzly, ale netestuje, zda lze uzly propojit. Test je proveden až v konečné fázi hledání posloupnosti uzlů, které potenciálně tvoří výslednou cestu. Takový postup výrazně snižuje časovou náročnost algoritmu za cenu zvýšení složitosti.

V EST algoritmu je při rozhodování, který uzel rozšířit, hledáno n nejbližších sousedů. To je ale časově náročná operace a SBL používá jiný přístup. SBL rozděluje celý konfigurační prostor do mřížky a pro expanzi vybírá uzly z buňky s nejmenším počtem uzlů. Tento postup lze aplikovat u ostatních algoritmů.

3.6 Rapidly-exploring Random Trees

Rapidly-exploring Random Trees (RRT) [9] je jednodotazový algoritmus podobný EST algoritmu. Stejně jako EST používá dva stromy, jeden s kořenem v počátečním uzlu a druhý

strom z koncového uzlu. Existuje i jednostromová varianta. Algoritmus oba stromy buduje zároveň, přičemž se je snaží propojit. Propojení stromů značí konec algoritmu a úspěšné nalezení cesty.

3.6.1 Budování stromů

První fáze začíná vytvořením stromů T_{init} a T_{goal} z počáteční konfigurace q_{init} a cílové konfigurace q_{goal} (algoritmus 3.5 a 3.6). V každé iteraci je vygenerována náhodná konfigurace q_{rand} z volného konfiguračního prostoru Q_{free}. K této konfiguraci je z aktuálně expandovaného stromu nalezena nejbližší konfigurace q_{near}. Uzly jsou propojeny úsečkou a ve vzdálenosti $step_{size}$ od q_{near} je na úsečce vytvořen nový uzel q_{new}. Pokud lze uzly spojit propojovací funkcí Δ, je uzel q_{new} přidán do stromu.

Parametr $step_{size}$ lze zvolit jako konstantní hodnotu nebo hodnotu dynamicky měnit na základě vzdálenosti q_{near} a q_{rand}. V případě konstantní hodnoty je nutné hodnotu volit s ohledem na konkrétní úlohu. Je-li zvolená délka kroku příliš malá, algoritmus bude generovat příliš mnoho uzlů, což může algoritmus neúnosně zpomalit.

Algoritmus může být dále rozšířen generováním nových uzlů po krocích o velikosti $step_{size}$ do doby, než narazí na překážku nebo pokud je pokryt celý prostor mezi q_{near} a q_{rand}. Nicméně je důležité negenerovat příliš mnoho uzlů. Rozšíření lze vylepšit uložením jen uzlu, který je v největší vzdálenosti od q_{near}, resp. nejblíž k q_{rand}, což má za následek snížení časových i paměťových nároků.

Algoritmus 3.5 Algoritmus budování RRT stromu

Vstup:

q_0: počáteční konfigurace
n: počet pokusů o expanzi stromu

Výstup:

Strom $T = (V, E)$ s kořenem q_0 a počtem uzlů $\leq n$

1. $V \leftarrow \{q_0\}$
2. $E \leftarrow \emptyset$
3. **for** $i = 1$ do n **do**
4. $\quad q_{rand} \leftarrow$ náhodně vybraná volná konfigurace
5. $\quad extendRRT(T, q_{rand})$
6. **end for**
7. **return** T

Způsob generování uzlů velice ovlivňuje celkovou efektivitu algoritmu. Lze použít stejné techniky jako u PRM algoritmu popsané v sekci 3.3.1. Dalším vylepšením může být pokaždé negenerovat nový uzel, ale snažit se stromy rozšířit přímo k cílové, resp. počáteční konfiguraci. Jinými slovy v algoritmu 3.5 na řádku 4 zaměnit q_{rand} za q_{init}, resp. q_{goal}. Pravděpodobnost záměny by ale neměla být příliš velká, protože by hledání cesty mohlo lehce uvíznout v lokálním extrému. Tuto myšlenku lze ještě dále rozšířit a přímo nezaměňovat q_{rand}, ale nový uzel generovat v okolí q_{init}, resp. q_{goal}.

Algoritmus 3.6 Algoritmus expanze RRT stromu (*extendRRT*)

Vstup:

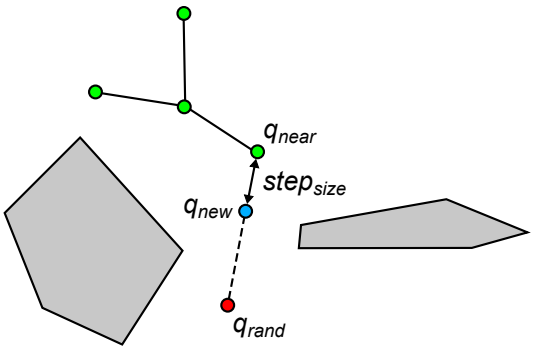
$T = (V, E)$: RRT strom

q: konfigurace, ke které strom T poroste

Výstup:

Nová konfigurace q_{new} směrem ke q nebo `NIL` v případě neúspěchu

1. $q_{near} \leftarrow$ nejbližší uzel z V ke konfiguraci q
2. $q_{new} \leftarrow$ bod na úsečce mezi uzly q_{near} a q_q ve vzdálenosti $step_{size}$ od q_{near}
3. **if** $\Delta(q_{near}, q_{new}) \neq$ `NIL` **then**
4. $\quad V \leftarrow V \cup \{q_{new}\}$
5. $\quad E \leftarrow E \cup \{(q_{near}, q_{new})\}$
6. $\quad$ **return** q_{new}
7. **else**
8. $\quad$ **return** `NIL`
9. **end if**



Obrázek 3.7: Přidání nové konfigurace RRT. Z Q_{free} byl vygenerován uzel q_{rand} a k němu byl nalezen nejbližší uzel q. Nový uzel q_{new} je vytvořen posunutím uzlu q směrem k q_{rand} ve vzdálenosti $step_{size}$.

3.6.2 Spojení stromů

Ve fázi propojení (algoritmus 3.7) se RRT algoritmus snaží propojit stromy T_{init} a T_{goal} vytvořené v předchozí fázi. Nejprve je vytvořen nový uzel q_{rand}, který je propojen s jedním ze stromů. Pokud je připojení uzlů úspěšné, následuje pokus o propojení q_{rand} s druhým stromem. Když je uzel připojen k oběma stromům, stromy byly úspěšně propojeny a algoritmus končí. V opačném případě, kdy připojení uzlu selhalo a nebyl proveden maximální počet pokusů o propojení, jsou stromy vyměněny a proces se opakuje.

Algoritmus 3.7 Algoritmus spojení RRT stromů

Vstup:

$T_1 = (V_1, E_1)$: první RRT strom
$T_2 = (V_2, E_2)$: druhý RRT strom
l: počet pokusů o propojení stromů T_1 a T_2

Výstup:

`merged` pokud je spojení úspěšné, jinak `failure`

1. **for** $i = 1$ **to** l **do**
2. $q_{rand} \leftarrow$ náhodně zvolená volná konfigurace
3. $q_{new,1} \leftarrow extendRRT(T_1, q_{rand})$
4. **if** $q_{new,1} \neq$ `NIL` **then**
5. $q_{new,2} \leftarrow extendRRT(T_2, q_{rand})$
6. **if** $q_{new1} = q_{new2}$ **then**
7. **return** `merged`
8. **end if**
9. $SWAP(T_1, T_2)$
10. **end if**
11. **end for**
12. **return** `failure`

3.7 Retraction-based RRT

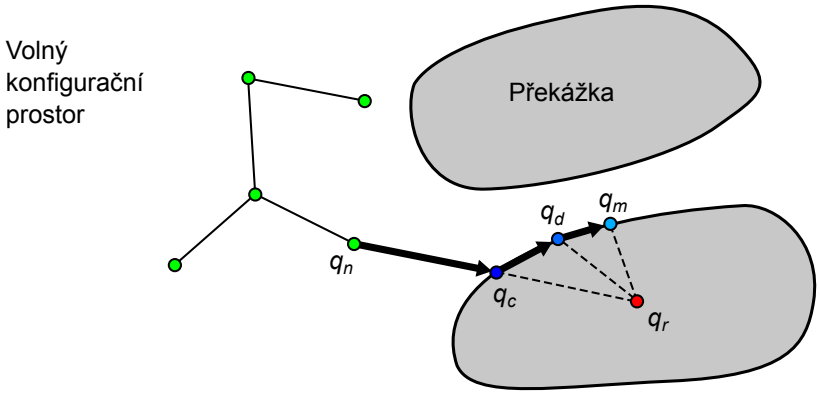
Retraction-based RRT (R-RRT) [14, 15] je jednodotazový retrakční algoritmus, který rozšiřuje RRT algoritmus o efektivnější způsob vzorkování úzkých průchodů. Algoritmus pracuje podobně jako OBPRM, který se snaží generovat více konfigurací na okraji překážek. R-RRT ke generování konfigurací na okraji překážek používá tzv. *retrakční krok*. Ten lze formálně definovat jako optimalizační problém:

$$q_m = \arg\min_q \delta(q, q_r), q \in C_{contact} \tag{3.1}$$

kde δ značí vzdálenostní metriku mezi dvěmi konfiguracemi a $C_{contact}$ značí volný konfigurační prostor na okraji překážek. Cílem je tedy nalézt konfiguraci q_m, která leží na hranici překážky a zároveň je co nejblíže k cílové konfiguraci q_r. Retrakční krok pracuje následovně:

1. Nalezení konfigurace q_c, která leží mezi uzly q_n a q_r a na hranici překážky.
2. Prohledávání lokálního prostoru za účelem nalezení nové nekolidující konfigurace q_d, která se nachází blíže ke q_r podle metriky δ.
3. Přiřazení $q_n = q_d$ a skok na krok 1.

Optimalizace je ukončena v případě, že nelze nalézt žádnou bližší konfiguraci, nebo bylo dosáhnuto předem definovaného počtu pokusů. Na obrázku 3.8 je grafické znázornění retrakčního kroku.



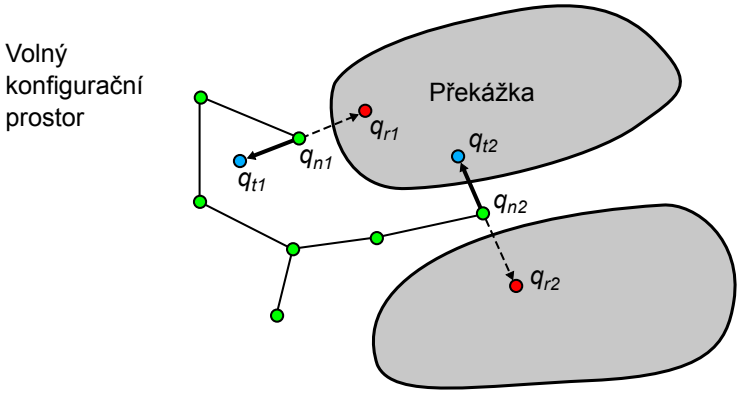
Obrázek 3.8: Retrakční krok R-RRT algoritmu. Generování sekvence nekolidujících konfigurací q_c, q_d, q_m, které se postupně blíží k q_r.

3.8 Regulated R-RRT

Na základě experimentů bylo navrženo vylepšení R-RRT algoritmu, které bylo pojmenováno *Regulated R-RRT* (RR-RRT). R-RRT algoritmus sice pracuje lépe v úzkých prostorech, použitím retrakčního kroku ale retrakční krok provádí, i když to není potřeba. Celý algoritmus je pak v některých scénách znatelně zpomalen. RR-RRT používá jednoduchý test úzkého průchodu. Pokud nelze dvě konfigurace q_n a q_r propojit, provede se test, na jehož základě se rozhodne o spuštění retrakčního kroku. Test pracuje následovně:

1. Výpočet konfigurace q_t, která leží v opačném směru cílové konfigurace q_r v pevné vzdálenosti od q_n.
2. Pokud q_t koliduje s překážkou, test označí oblast jako úzký průchod.

Retrakční krok je pak spuštěn jen v případě, že test označí okolí jako úzký průchod. Test nemusí vždy správně úzký průchod identifikovat, ale filtruje velké množství případů, kdy by se retrakční krok spustil zbytečně a jen by celý algoritmus zpomalil.



Obrázek 3.9: Test úzkého průchodu RR-RRT algoritmu. V případě q_{n1} a q_{r1} testovací konfigurace q_{t1} nekoliduje s překážkou, a je tak zamezeno zbytečnému použití retrakčního kroku. V druhém případě q_{t2} koliduje, což má za následek spuštění retrakčního kroku.

Na obrázku 3.9 je ukázka dvou testů. V prvním případě nastala situace, kdy je třeba propojit q_{n1} a q_{r1}. Je vygenerována testovací konfigurace q_{t1}, která však nekoliduje s žádnou překážkou, a tak není rektrakční krok spuštěn. V druhém případě (q_{n2} a q_{r2}) testovací konfigurace q_{t2} koliduje s překážkou a je pozitivně detekován úzký průchod a následně spuštěn retrakční krok.

3.9 Sampling-based Roadmap of Trees

Sampling-based Roadmap of Trees (SRT) [1] je kombinovaný plánovací algoritmus, který využívá již zmíněné jednodotazové a vícedotazové algoritmy PRM, EST a RRT. Algoritmus pracuje ve dvou hlavních fázích. V první fázi je vytvořen graf volného konfiguračního prostoru, který je použit ve druhé fázi při hledání cesty.

3.9.1 Tvorba grafu prostředí

Graf volného konfiguračního prostoru G_T je tvořen kombinací jednodotazových a vícedotazových algoritmů následujícím způsobem:

1. Generování n volných konfigurací v prostoru s rovnoměrným rozložením. Cílem je pokrýt co největší část prostoru. Vzorkování může být řízeno rovnoměrným rozložením nebo může být provedeno sofistikovanějšími přístupy popsané v sekci 3.3.1.

2. Z n vygenerovaných uzlů (konfigurací) jsou vytvořeny uzly stromů $T_1, \ldots, T_n$, které jsou přidány do grafu G_T.

3. Pro každý strom (v této fázi zatím jen kořenový uzel) $T_1, \ldots, T_n$ je spuštěn některý jednodotazový algoritmus, např. EST nebo RRT.

4. Rozšířené stromy $T_1, \ldots, T_n$ jsou navzájem propojeny.

Propojení stromů je popsáno algoritmem 3.8. Vstupem algoritmu je graf (množina stromů V_T) vytvořený jednodotazovými algoritmy. Parametry jsou počet nejbližších stromů k a počet náhodných stromů n k propojení. Algoritmus pro každý strom vybere k nejbližších a n náhodných stromů a snaží se je propojit. Pokud nejsou stromy propojeny, je vybrán seznam náhodných uzlů z prvního stromu a ty se snaží přímo spojit hranou s nejbližšími uzly z druhého stromu propojovací funkcí Δ. Pokud přímé propojení selže, je aplikováno sofistikovanější propojení (v algoritmu 3.8 označeno jako $MergeTrees$), např. propojení složitější křivkou nebo jednodotazovým algoritmem.

Výpočet vzdálenosti stromů je ideálně nejmenší vzdálenost ze všech dvojic uzlů. Takový přístup je však výpočetně náročný a aproximace je na místě. V extrémním případě lze použít jen kořeny stromu. Lepší aproximací je pro každý strom vypočíst střed jako průměr pozic všech uzlů stromu a ze středů stromů přímo určit jejich vzdálenost.

3.9.2 Vyhledání cesty

Stejně jako u PRM, SRT umožňuje nalezení cesty pro různé počáteční a koncové konfigurace. Nejprve je vytvořen strom T_{init} z počáteční konfigurace q_{init} a strom T_{goal} z koncové konfigurace q_{goal}. Tyto stromy jsou následně připojeny do grafu prostředí G_T, stejně jako při konstrukci grafu algoritmem 3.8. Pokud jsou oba stromy ve stejné komponentě, cesta zajisté existuje a stačí jen nalézt příslušnou posloupnost konfigurací.

Algoritmus 3.8 Algoritmus propojení jednotlivých SRT stromů

Vstup:

V_T: množina stromů
k: počet nejbližších sousedních stromů k napojení
r: počet náhodných stromů k napojení

Výstup:

Graf $G_T = (V_T, E_T)$ prostředí

1. $E_T \leftarrow \emptyset$
2. **for all** $T_i \in V_T$ **do**
3. $N_{T_i} \leftarrow k$ nejbližších a r náhodných sousedních stromů stromu T_i
4. **for all** $T_j \in N_{T_i}$ **do**
5. **if** T_i a T_j nejsou součástí stejné komponenty grafu G_T **then**
6. $merged \leftarrow$ `FALSE`
7. $S_i \leftarrow$ seznam náhodně vybraných uzlů ze stromu T_i
8. **for all** $q_i \in S_i$ **and** $merged =$ `FALSE` **do**
9. $q_j \leftarrow$ nejbližší uzel k uzlu q_i ze stromu T_j
10. **if** $\Delta(q_i, q_j) \neq$ `NIL` **then**
11. $E_T \leftarrow E_T \cup \{(q_i, q_j)\}$
12. $merged \leftarrow$ `TRUE`
13. **end if**
14. **end for**
15. **if** $merged =$ `FALSE` **and** $MergeTrees(T_i, T_j)$ **then**
16. $E_T \leftarrow E_T \cup \{(T_i, T_j)\}$
17. **end if**
18. **end if**
19. **end for**
20. **end for**

3.9.3 Parametry

Zajímavou vlastností SRT algoritmu je, že pro určité parametry degeneruje na základní algoritmy PRM, EST nebo RRT. Pokud je velikost stromů nastavena na jedna, nebude provedeno jejich rozšíření a ve výsledku budou propojeny jen jednotlivé kořeny, což je přesně PRM algoritmus. V případě, kdy je počet stromů nastaven na nulu, SRT odpovídá EST nebo RRT algoritmu v závislosti na zvoleném typu stromu.

Algoritmus je také lehce paralelizovatelný. Na rozdíl od EST nebo RRT algoritmů, kde je generování uzlů závislé na předchozím vygenerovaném uzlu, u SRT algoritmu lze stromy budovat nezávisle. Paralelizace propojení stromů už není tak jednoduchá, protože v průběhu propojení stromů se přidáváním nových hran stromy mění.

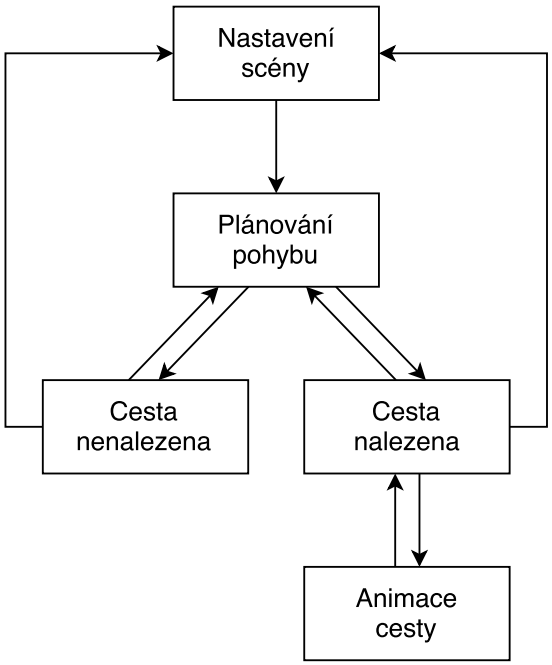
Kapitola 4

Návrh

Tato kapitola obsahuje návrh programu. Aplikace obsahuje uživatelské rozhraní pro definici úlohy (nastavení scény) a následné zobrazení nalezeného řešení. V následující sekci jsou popsány jednotlivé stavy programu, které definují stav uživatelského rozhraní a jaké akce lze vykonat.

4.1 Stavy programu

Jádrem programu je stavový automat, který určuje, jaké akce může uživatel vykonat a které komponenty uživatelského rozhraní mají být aktivní. Program se může nacházet v jednom z pěti stavů, které jsou uvedeny na obrázku 4.1.



Obrázek 4.1: Stavový automat programu

4.1.1 Nastavení scény

Prvním krokem při plánování pohybu objektu je nastavení scény, které je zároveň výchozím stavem při spuštění programu. Do tohoto stavu program automaticky přejde, pokud uživatel provede jakoukoli změnu scény. To zahrnuje např. načtení nové scény, změnu polohy objektu nebo odstranění překážky.

Scénu lze vytvořit vkládáním nových objektů a nastavením jejich pozice, rotace a velikosti. Program obsahuje několik přednastavených modelů. Další modely lze vytvořit v jakémkoli 3D editoru, který umožňuje uložit model ve *Wavefront OBJ* formátu. Nový model lze následně načíst, čímž se zpřístupní v seznamu modelů, a je možné ho použít jako překážku nebo pohyblivý objekt.

Vytvořenou scénu lze uložit a načíst při budoucím spuštění aplikace. Formát souboru se scénou je jednoduché XML. Konkrétní formát souboru se scénou je popsán v sekci 4.3. Program přímo obsahuje několik předpřipravených scén pro rychlou demonstraci programu. Mezi možné operace nastavení scény patří následující:

- Určení počáteční a cílové pozice (konfigurace) robota
- Nastavení modelu robota
- Rozmístění překážek (model a konfigurace překážky)
- Nastavení hranic prostoru

4.1.2 Plánování pohybu

Druhým krokem a hlavní částí programu je samotné hledání cesty. Plánování pohybu, neboli hledání cesty, se aktivuje přes dialog, kde uživatel vybere konkrétní plánovací algoritmus z nabídky implementovaných algoritmů. Spolu s typem algoritmu uživatel nastaví parametry, jako např. u PRM algoritmu počet generovaných uzlů grafu nebo počet sousedních uzlů.

V průběhu hledání cesty se do textového pole vypisují informace o aktuální fázi algoritmu, např. inicializace algoritmu, generování nových uzlů, optimalizace nalezené cesty atd. Ve scéně se zároveň postupně zobrazuje aktuální graf/strom prostředí, jak jej daný algoritmus buduje. V této fázi lze sledovat, na jakém principu algoritmy pracují.

Po dokončení plánování pohybu je zobrazen výsledný graf prostředí. V případě, že byla cesta nalezena, jsou vypsány základní informace o cestě, jako její délka nebo počet uzlů. Mimo informací o cestě jsou také vypsány informace o grafu prostředí (počet uzlů, počet hran), který byl vygenerován v průběhu hledání cesty. Na závěr je vypsána doba běhu algoritmu. V případě, že cesta byla úspěšně nalezena, program přechází do stavu nalezené cesty. V opačném případě neúspěchu nalezení cesty program přechází do stavu nenalezené cesty.

4.1.3 Cesta nalezena

Pokud algoritmus úspěšně nalezne cestu, přechází program do tohoto stavu. V tomto stavu je zobrazen výsledek hledání cesty, což zahrnuje graf prostředí spolu s vyznačenou cestou (posloupnost hran grafu). Z tohoto stavu je možné přejít do stavu animace nalezené cesty.

Plánování pohybu v komplexním prostředí může být časově náročné a aby nebylo nutné při každém spuštění aplikace cestu opakovaně hledat, je možné nalezenou cestu uložit a později načíst. Stejně jako u souboru se scénou i soubor s cestou je uložen ve formátu XML a je detailněji popsán v sekci 4.3.

4.1.4 Cesta nenalezena

V případě neúspěchu nalezení cesty lze zobrazit jen graf prostředí. Tento stav slouží zejména jako mezi-stav mezi změnou scény nebo znovu-spuštěním hledání cesty jiným algoritmem nebo s jinými parametry.

4.1.5 Animace cesty

Po úspěšném nalezení cesty přichází na řadu její vizualizace. Vizualizace cesty je provedena animací pohybu objektu z počáteční do cílové konfigurace. Při animaci lze měnit rychlost pohybu objektu nebo jeho aktuální pozici na cestě. Když objekt dorazí do cílové pozice, je přesunut na počátek cesty a celá animace se opakuje.

4.2 Uživatelské rozhraní

Účelem uživatelského rozhraní je umožnit uživateli vytvořit 3D scénu z několika objektů, vybrat algoritmus pro hledání cesty spolu s jeho parametry a nakonec nalezenou cestu (pokud byla nalezena) vizualizovat. Bez ohledu na to, zda byla cesta nalezena, by mělo být možné zobrazit graf (uzly a hrany), který plánovací algoritmus vytvořil.

Okno programu je rozděleno na tři části. Hlavní částí je pohled na 3D scénu. Druhou částí je boční panel pro nastavení pozice, rotace a velikosti objektu spolu se seznamem překážek. Třetí částí je ovládací lišta pro zobrazení okna pro plánování pohybu, nastavení hranic prostoru a ovládání animace výsledné cesty.

4.2.1 Scéna

Scéna je hlavní částí okna aplikace. Scéna obsahuje počáteční a cílovou pozici, rozmístění překážek, graf prostředí generovaný plánovacími algoritmy, výslednou cestu a animaci objektu po nalezené cestě. Při spuštění aplikace scéna obsahuje výchozí nastavení počáteční a cílové pozice spolu s překážkami.

Manipulovat lze s aktivním objektem, který byl vybrán v levém menu. Nově přidaný objekt je také automaticky nastaven na aktivní objekt. Aktivní objekt je zvýrazněn hraniční obálkou. Konfiguraci aktivního objektu lze měnit pomocí levého panelu anebo s ním lze manipulovat myší a klávesnicí, které je blíže popsáno v sekci 4.2.4.

4.2.2 Panel objektů

Levý panel slouží pro správu objektů ve scéně. Panel obsahuje komponenty pro nastavení polohy, rotace a velikosti aktivního objektu. Aktivní objekt lze vybrat ze seznamu všech objektů ve scéně, který obsahuje počáteční pozici, cílovou pozici a všechny překážky. Pravým tlačítkem myši na položku v seznamu objektů lze vyvolat kontextové menu, které umožňuje změnit model objektu. Pokud je označený objekt překážka, lze ji odstranit anebo vytvořit její kopii. Panel dále obsahuje tlačítka pro přidání nové překážky, odstranění aktuálního objektu a odstranění všech překážek.

4.2.3 Ovládací panel

Ovládací panel je umístěn v horní části okna a je rozdělen do několika sekcí:

- **Scéna:** sekce pro scénu obsahuje tlačítka pro vytvoření nové scény, uložení scény do souboru, načtení scény ze souboru a načtení ukázkové scény, která je uložena přímo v programu.
- **Cesta:** podobná sekce jako pro scénu. Sekce obsahuje tlačítka pro uložení a načtení cesty ze souboru a načtení uložené cesty, takže není nutné při každém spuštění znovu cestu hledat.
- **Plánování pohybu:** zde jsou komponenty pro otevření dialogu pro započetí hledání cesty a dialogu pro nastavení hranic scény. Sekce dále obsahuje menu s ovládáním, jaké objekty mají být zobrazeny (hranice prostoru, uzly a hrany grafu, nalezená cesta a další). Možnosti zobrazení jsou podrobněji popsány v sekci 5.2.
- **Animace:** sekce s ovládacími prvky pro spuštění, pozastavení a zastavení animace. Sekce dále obsahuje posuvník pro ovládání rychlosti animace a posuvník pozice objektu na cestě. Úplně levá pozice posuvníku je počáteční pozicí cesty a úplně pravá pozice posuvníku je cílovou pozicí cesty.

4.2.4 Ovládání scény

Scénu lze pomocí myši a klávesnice posouvat, rotovat a přibližovat. Stejně tak lze ovládat aktivní objekt. Konkrétní parametry objektů lze zároveň nastavit přes levý panel, který je popsaný v sekci 4.2.2. Ovládání scény a objektů klávesnicí a myší je uvedeno v tabulce 4.1.

Akce	**Funkce**
Levé tlačítko myši	posun scény
Pravé tlačítko myši	rotace scény
Kolečko myši	přiblížení scény
CTRL + levé tlačítko myši	posun aktivního objektu
CTRL + pravé tlačítko myši	rotace aktivního objektu podle os X a Y
CTRL + SHIFT + pravé tlačítko myši	rotace aktivního objektu podle os X a Z
CTRL + kolečko myši	změna velikosti aktivního objektu

Tabulka 4.1: Způsob ovládání scény a objektů

4.3 Formát souboru se scénou a cestou

Aby nebylo nutné při každém spuštění vytvářet novou cestu, je potřeba navrhnout formát souboru, ve kterém bude scéna uložena. Scéna je uložena v XML formátu a schéma souboru je následující (ukázka souboru je v příloze B.1):

- `Models` – seznam modelů použitých ve scéně:
  - `Model` – model obsahuje název, kterým je odkazován při použití u robota anebo překážek. Dále obsahuje cestu k souboru s definicí modelu ve Wavefront OBJ formátu.

- `Space` – definice prostoru:
  - `Boundary` – hranice prostoru, která je definována minimální a maximální hodnotou pozic pro všechny dimenze prostoru.
  - `Robot` – robot, neboli objekt, jehož pohyb je plánován. Robot obsahuje název modelu a velikost modelu.
  - `Start` – počáteční pozice robota, která zahrnuje pozici v prostoru a rotaci.
  - `Goal` – cílová pozice robota, která zahrnuje pozici v prostoru a rotaci.
  - `Obstacles` – seznam překážek ve scéně:
    - `Obstacle` – překážka, která obsahuje název modelu překážky, polohu, rotaci a velikost modelu.

Důležitější než uložení scény je uložení nalezené cesty. Hledání cesty totiž ve složitých scénách může trvat dlouhou dobu a je nepraktické při každém spuštění programu již nalezené cesty znovu hledat. Stejně jako scéna i cesta je uložena v XML formátu. Schéma cesty je následující (ukázka souboru je v příloze B.2):

- `Scene` – cesta k souboru se scénou, ve které byla cesta nalezena.
- `Nodes` – posloupnost uzlů cesty:
  - `Node` – uzel cesty definovaný pozicí a rotací.

Kapitola 5

Implementace

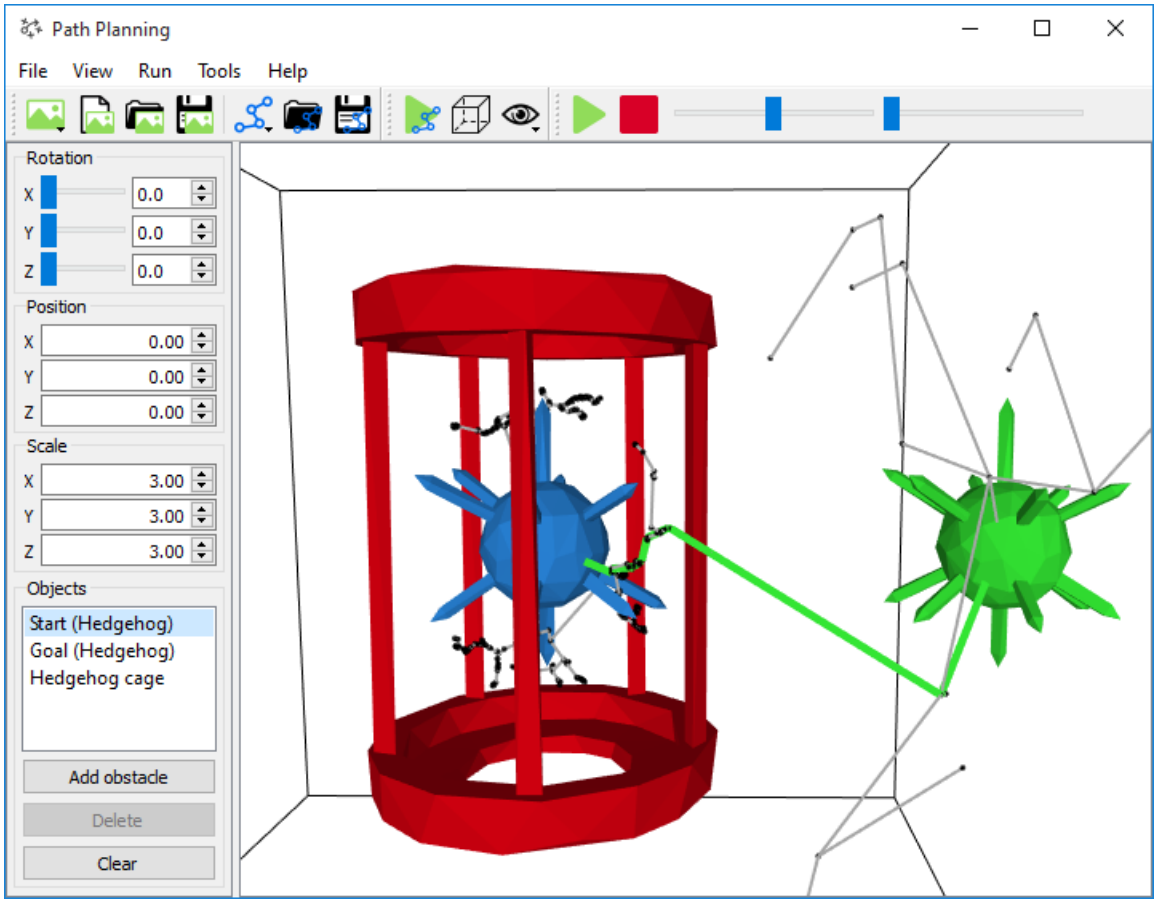
V této kapitole je popsána implementace programu, což zahrnuje implementaci uživatelského rozhraní a plánovacích algoritmů. Hlavní důraz je kladen na popis implementačních detailů plánovacích algoritmů. Některé části jsou totiž v teoretické části popsány na vyšší úrovni a jejich implementace vyžaduje detailnější popis.

Program je implementován v jazyce C++ a pro uživatelské rozhraní je použit *Qt framework*[1]. Pro zobrazení 3D rozhraní bylo na výběr z *OpenGL* nebo *Microsoft DirectX*. Bylo zvoleno OpenGL, kvůli možnosti spuštění aplikace na jiných operačních systémech než Windows. Qt framework navíc poskytuje několik modulů pro práci s OpenGL. Detekce kolizí je realizována volně dostupnou knihovnou *ColDet 3D*[2]. Dále je použita matematická knihovna *OpenGL Mathematics (GLM)*[3], ze které jsou použity např. struktury pro práci s vektory nebo kvaterniony. Všechny použité knihovny jsou multiplatformní, a tak je možné aplikaci spustit na různých operačních systémech.

Implementaci programu lze rozdělit na dvě hlavní části, kterými jsou:

- **Uživatelské rozhraní:** třídy pro ovládání aplikace a vizualizaci scény. Qt třídy pro hlavní okno programu a její komponenty, jako OpenGL komponenta pro zobrazení 3D scény a ovládací panely. Dalšími třídami uživatelského rozhraní je ovládání animace cesty (rychlosti, polohy) a v neposlední řadě moduly pro uložení a načtení scény a cesty.
- **Plánování pohybu:** implementace jednotlivých pravděpodobnostních algoritmů. Přičemž některé části jsou společné pro všechny algoritmy, jako způsob propojení dvou konfigurací (detekce kolizí), generování náhodných konfigurací, struktury pro výsledek hledání cesty a komunikační rozhraní s vizualizační částí.

[1] Qt framework: https://www.qt.io/
[2] Amir Geva, ColDet 3D Collision Detection: https://sourceforge.net/projects/coldet/
[3] OpenGL Mathematics (GLM): http://glm.g-truc.net/



Obrázek 5.1: Ukázka uživatelského rozhraní programu

5.1 Hlavní třídy programu

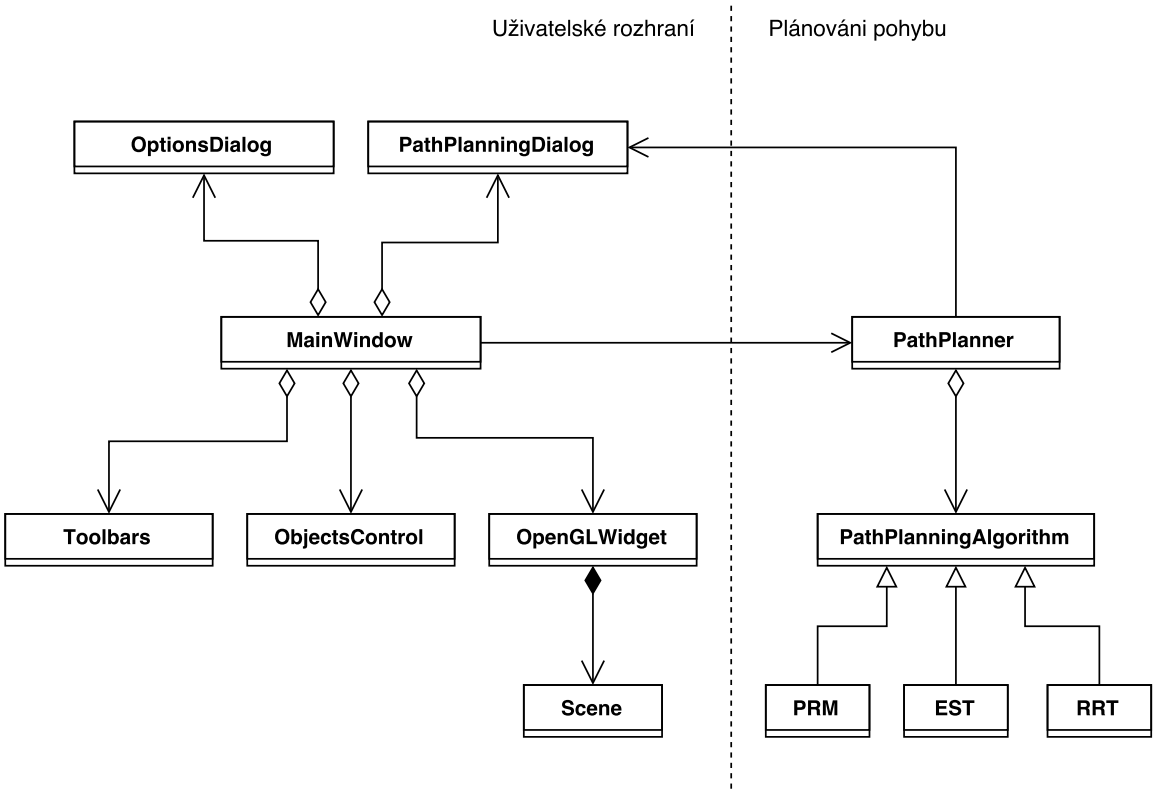
Tato sekce obsahuje popis nejdůležitějších tříd uživatelského rozhraní a plánovacích algoritmů. Obrázek 5.2 obsahuje diagram hlavních tříd programu a jejich rozdělení na uživatelské rozhraní a plánování pohybu.

5.1.1 Uživatelské rozhraní

Uživatelské rozhraní obsahuje třídy pro vizualizaci scény, ovládání scény a zobrazení výsledků plánování cesty. Většina komponent rozšiřuje některou Qt komponentu, jako dialog nebo OpenGL komponentu.

MainWindow

Hlavní třída programu. `MainWindow` zastřešuje všechny hlavní komponenty a dialogy pro nastavení programu a plánování pohybu. Obsahuje komponenty OpenGL vizualizace 3D scény, postranní panel ovládání objektů a horní panel ovládání aplikace.



Obrázek 5.2: Zjednodušený diagram hlavních tříd programu

OpenGLWidget

Komponenta pro zobrazení 3D scény s překážkami a výsledky hledání cesty. `OpenGLWidget` složí jen jako kontejner s OpenGL funkcemi. Samotné kreslení objektů provádí následující třída `Scene`. `OpenGLWidget` zajišťuje předzpracovaní uživatelských vstupů myši, které přeposílá scéně. Třída obsahuje časovač pro periodické překreslování scény. Frekvenci překreslení scény lze změnit v nastavení programu.

Scene

Třída `Scene` zajišťuje samotné zobrazení 3D scény pomocí OpenGL funkcí. Obsahuje aktuální pozici kamery, funkce pro přepočet uživatelského vstupu na změnu polohy kamery a změnu konfigurace aktivního objektu. Hlavní funkcí je zobrazení všech objektů, jako počáteční a cílové pozice, překážek, nalezené cesty a grafu prostředí.

ObjectsControl

`ObjectsControl` je panel pro správu objektů ve scéně. Obsahuje komponenty pro nastavení pozice, rotace a velikosti aktuálně označeného objektu. Dále obsahuje seznam všech objektů ve scéně. V seznamu lze označit aktivní objekt anebo změnit model objektu. Překážky lze přes seznam odstranit nebo duplikovat.

Toolbars

Program obsahuje tři na sobě nezávislé ovládací panely:

- `SaveOpenToolBar` obsahuje tlačítka pro uložení a načtení scény a cesty ze souboru. Dále obsahuje menu pro načtení předpřipravených scén a uložených nalezených scén.
- `PathPlanningToolBar` obsahuje tlačítka pro zobrazení dialogu plánování pohybu `PathPlanningDialog`. Dále obsahuje tlačítko pro nastavení hranic prostoru a v neposlední řadě obsahuje menu pro nastavení objektů, které mají být ve scéně zobrazeny. Jednotlivé volby zobrazení jsou popsány v sekci 5.2.
- `AnimationToolBar` obsahuje tlačítka pro spuštění, pozastavení a zastavení animace. Dále obsahuje posuvník rychlosti animace a posuvník s aktuální pozicí animovaného objektu na cestě.

PathPlanningDialog

`PathPlanningDialog` je dialog pro výběr algoritmu plánování pohybu spolu s jeho parametry. Po výběru algoritmu a nastavení parametrů může uživatel spustit hledání cesty pro aktuálně nastavenou scénu. Plánování pohybu lze kdykoli přerušit. V dialogu se v průběhu plánování vypisují informace o aktuálním stavu hledání cesty. Po nalezení cesty jsou zde vypsány informace o nalezené cestě.

OptionsDialog

`OptionsDialog` je dialog pro změnu nastavení programu. Pokud právě neprobíhá plánování pohybu, tak uživatel může změnit různá nastavení. Popis jednotlivých voleb nastavení je uveden v sekci 5.3.

5.1.2 Plánování pohybu

V této sekci jsou krátce popsány hlavní třídy pro plánování pohybu spolu s důležitými strukturami, které jsou při plánování pohybu použity.

PathPlanner

Třída `PathPlanner` slouží hlavně jako komunikační rozhraní mezi samotným plánováním pohybu a uživatelským rozhraním. Další funkcí třídy je spuštění plánovacího algoritmu v novém vlákně, aby při hledání cesty nebylo zablokováno uživatelské rozhraní.

PathPlanningAlgorithm

Všechny plánovací algoritmy jsou potomky bázové třídy `PathPlanningAlgorithm`. Třída definuje komunikační rozhraní a obsahuje základní funkce, mezi které patří: inicializace struktur, propojovací funkce, test kolize objektu v dané konfiguraci s překážkami, optimalizační algoritmus pro zkrácení cesty a Dijkstrův algoritmus.

Object

Třída `Object` obsahuje všechna potřebná data o objektu. Obsahuje typ objektu: počáteční pozice, cílová pozice, překážka a robot. Dále obsahuje konfiguraci objektu, což zahrnuje pozici, rotaci a velikost objektu. Pro zobrazení objektu ve 3D scéně obsahuje model načtený z .OBJ souboru a barvu, která je přiřazena na základě typu objektu. Poslední důležitou položkou je kolizní model, který je vytvořen z modelu objektu. Samotnou detekci kolizí zajišťuje knihovna ColDet 3D.

SpaceConfiguration

Třída `SpaceConfiguration` obsahuje popis scény (model scény). Scéna obsahuje objekty jako počáteční pozici, cílovou pozici a seznam překážek. Hranice scény jsou definovány třídou `Boundary`, která obsahuje minimální a maximální hodnoty pro všechny souřadnice prostoru x,y,z.

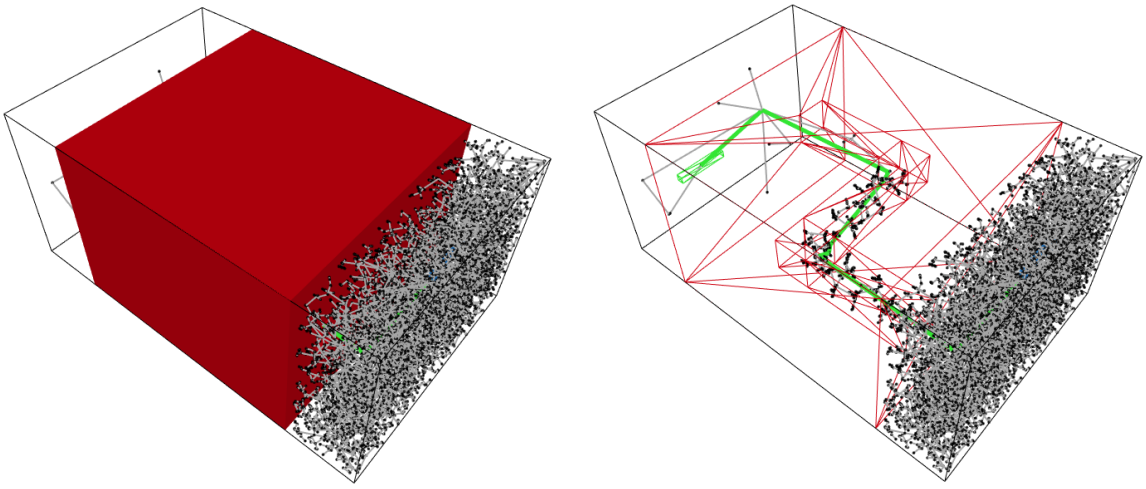
Result

`Result` představuje výsledek hledání cesty plánovacích algoritmů, což zahrnuje graf prostředí vytvořený daným algoritmem a nalezenou cestu, a to v optimalizované i neoptimalizované verzi.

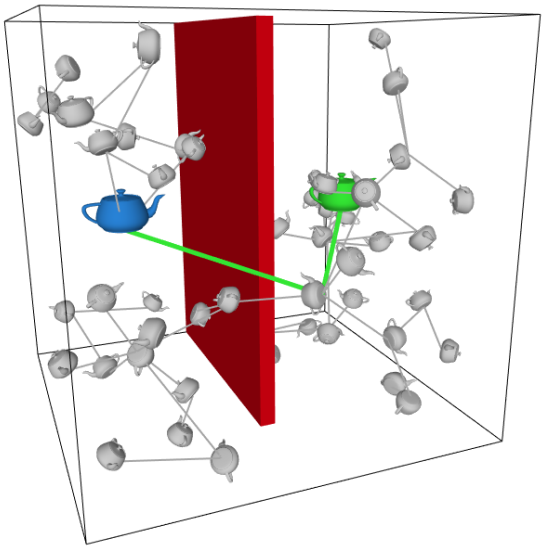
5.2 Možnosti zobrazení scény

Menu ovládání zobrazení umožňuje měnit zobrazení některých objektů a obsahuje následující položky:

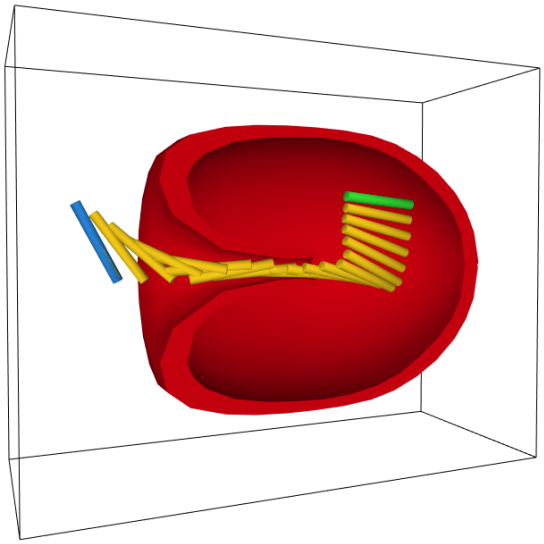
- **Boundary** – zobrazení hraniční obálky prostoru
- **Edges** – zobrazení hran grafu prostředí
- **Nodes** – zobrazení uzlů grafu prostředí
- **Model nodes** – jelikož samotné body neposkytují informaci o rotaci objektu, je možné uzly grafu nahradit zmenšeninami robota. Na obrázku 5.4 je ukázka vytvořeného grafu prostředí s uzly jako modely.
- **Path** – zobrazení nalezené cesty
- **Wireframe objects** – některé překážky jsou uzavřené objekty s vnitřním průchodem. Aby bylo možné sledovat cestu i uvnitř objektu, lze objekty zobrazit jako drátový model (obrázek 5.3).
- **Path preview** – při nalezení cesty lze zobrazit statický náhled animace pohybu objektu, jak je na obrázku 5.5. Krok náhledů lze změnit v nastavení programu. Pokud je krok náhledu animace stejný jako interpolační krok propojovací funkce Δ, pak náhled animace přímo odpovídá kolizním testům, které propojovací funkce provedla.



Obrázek 5.3: Vlevo je standardní zobrazení objektů a vpravo je zobrazení drátových modelů objektů.



Obrázek 5.4: Graf prostředí, kde jsou uzly zobrazeny jako zmenšeniny objektu.



Obrázek 5.5: Statický náhled animace nalezené cesty

5.3 Nastavení programu

Nastavení programu je rozděleno do tří sekcí:

1. **Obecná nastavení:**
   - Zobrazení neoptimalizované cesty. Ve výchozím stavu je zobrazena jen optimalizovaná cesta.
   - Sestavení cesty při zastavení algoritmu. V případě přerušení algoritmu se vytvoří cesta od počáteční konfigurace ke konfiguraci nejblíže cílové konfiguraci.

- Zobrazení aktuální podoby grafu prostředí (průběžného grafu). V případě rozsáhlých grafů může být plánování pohybu značně zpomaleno. Uživatel tedy může zakázat zobrazení průběžného grafu prostředí. Pokud je průběžný graf povolen, je zde možnost nastavení intervalu odesílání grafu v milisekundách.
- Velikost interpolačního kroku propojovací funkce Δ
- Krok náhledu animace nalezené cesty
- Inicializace generátoru náhodných čísel aktuálním časem

2. **Nastavení zobrazení:**

   - Frekvence překreslení 3D scény ve snímcích za vteřinu
   - Velikost bodu zastupující uzel grafu
   - Velikost modelu zastupující uzel grafu. Hodnota značí relativní velikost vzhledem k modelu robota. Pro hodnotu např. 0,5 bude model uzlu poloviční velikosti oproti modelu robota.
   - Šířka křivky nalezené cesty
   - Šířka hrany grafu

3. **Nastavení barev:**

   - Graf prostředí: hrany, uzly a modely
   - Objekty: počáteční pozice, cílová pozice, animovaný objekt, překážky a zvýraznění aktivního objektu
   - Výsledek: nalezená cesta a neoptimalizovaná cesta
   - Scéna: pozadí a hranice scény

Typ objektu	**Barva**
Počáteční pozice	modrá
Cílová pozice	zelená
Překážka	červená
Nalezená cesta	zelená
Graf prostředí	šedá

Tabulka 5.1: Výchozí barvy hlavních objektů ve scéně

5.4 Plánovací algoritmy

V programu jsou implementovány algoritmy PRM, EST, RRT a variace R-RRT a RR-RRT. Konkrétní implementační detaily jednotlivých algoritmů jsou popsány v následujících sekcích. Spolu s popisem algoritmů každý algoritmus obsahuje obrázek s ukázkou běhu algoritmu na scéně s bludištěm, která ukazuje, jakým způsobem algoritmus pracuje.

Před samotným spuštěním plánovacího algoritmu je provedena inicializace zdrojů, která je stejná pro všechny algoritmy. Spolu s inicializací jsou provedeny některé operace, které tvoří základní kostru plánovacího algoritmu:

1. Nastavení parametrů algoritmu.

2. Test kolize počáteční a cílové konfigurace s překážkami. Pokud některá konfigurace koliduje s překážkou, nelze cestu v takovém prostředí nalézt a algoritmus nebude spuštěn.

3. Inicializace struktur pro plánování pohybu, což zahrnuje graf prostředí, strukturu pro výslednou cestu a vytvoření kolizních modelů.

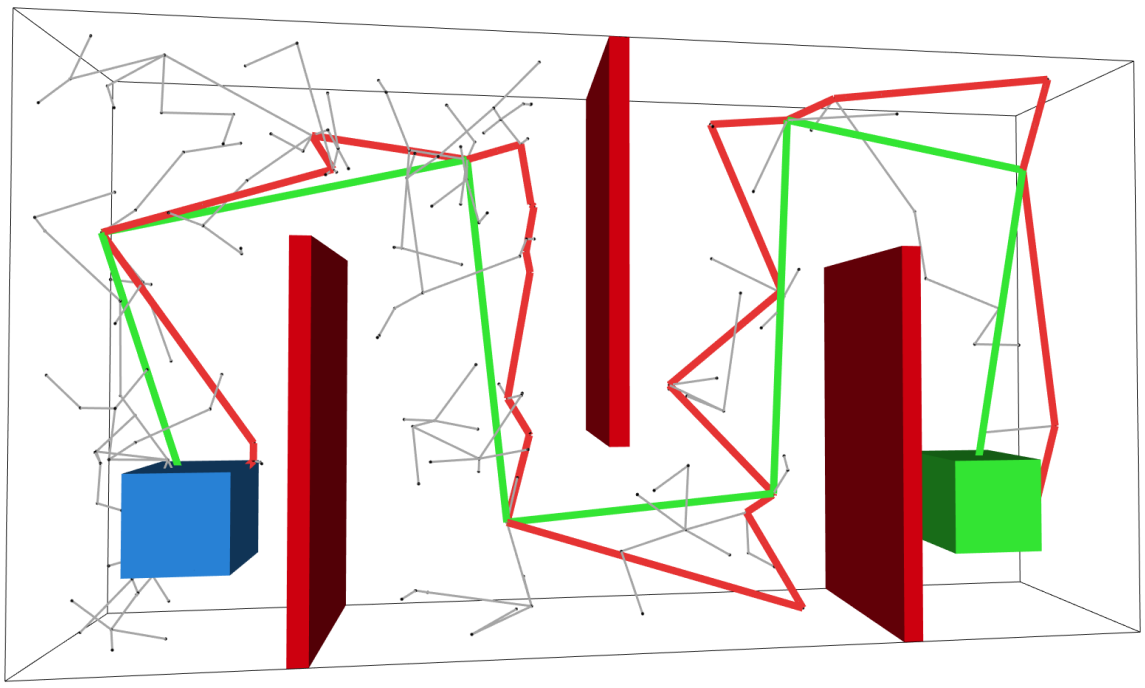
4. Inicializace časovačů. Při hledání cesty jsou periodicky vypisovány informace o aktuálním počtu vytvořených uzlů. V průběhu hledání cesty se také odesílá kopie grafu prostředí vizualizační části pro zobrazení ve scéně.

5. Běh konkrétního plánovacího algoritmu.

6. Pokud byla cesta nalezena, je optimalizována hladovým algoritmem popsaným v sekci 3.3.3. Na obrázku 5.6 je ukázka optimalizace cesty.

7. Odeslání nalezené cesty spolu s konečným grafem prostředí.

Všechny implementované algoritmy pracují do doby, než je nalezena výsledná cesta. V případě, že cesta neexistuje, algoritmy nikdy neskončí a je nutný zásah uživatele, který algoritmus musí ručně zastavit.



Obrázek 5.6: Ukázka optimalizace nalezené cesty. Neoptimalizovaná cesta je označena červeně a optimalizovaná cesta je označena zeleně.

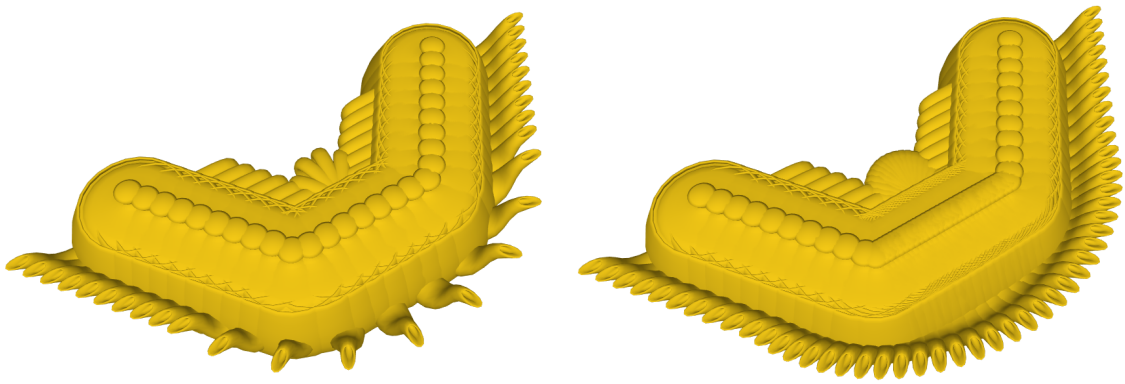
5.4.1 Propojovací funkce

Důležitou částí algoritmů je konkrétní implementace propojovací funkce Δ. Propojovací funkce má za úkol spojit dvě volné konfigurace. To je provedeno vytvořením křivky mezi

dvěma konfiguracemi. Křivka přitom musí umožnit bezkolizní přesun objektu ze zdrojové do cílové konfigurace. Jako propojovací křivka je zvolena úsečka, kvůli její jednoduchosti. Je použita lineární propojovací funkce, protože ta v případě neúspěchu propojení konfigurací umožňuje vrátit poslední nekolidující konfiguraci v blízkosti překážky. V případě rekurzivní propojovací funkce je třeba provést další výpočet. Poslední bezkolizní konfigurace je zejména potřeba u RRT a EST algoritmů.

Jemnost interpolace je řízena interpolačním krokem, v teoretické části (sekce 3.3.2) pojmenovaným jako $step_{size}$. Hodnotu parametru $step_{size}$ lze upravit v nastavení programu. Krok by měl být co nejmenší, aby úsečka byla dostatečně pokryta. Naopak by ale krok neměl být příliš malý, aby nebylo provedeno více kroků, než je potřeba. Velikost kroku má obrovský vliv na čas výpočtu. Detekce kolizí je výpočetně náročná operace a je žádané minimalizovat počet nutných testů kolize. Je ale lepší volit menší hodnotu kroku i za cenu delšího výpočtu, než zvolit hodnotu příliš vysokou a následně špatně detekovat překážky.

Velmi důležité je při krokování brát v úvahu i rotaci objektu. V případě, kdyby byla brána v úvahu jen translační vzdálenost, nebude proveden dostatečný počet kroků, který by rovnoměrně pokryl celou úsečku. Na obrázku 5.7 je porovnání propojovací funkce v případě použití pouze translační vzdálenosti a při použití kombinace s rotační vzdáleností. Problém je velice markantní v situacích, kdy objekt rotuje na místě nebo je nutná manipulace objektu ve stísněném prostoru. V tom případě by funkce založená pouze na translační vzdálenosti zajisté špatně detekovala překážky.



Obrázek 5.7: Porovnání interpolačních kroků při testování bezkoliznosti cesty. Na levém obrázku je způsob interpolace, kdy je použita pouze translační vzdálenost. Na pravém obrázku je interpolace pro translační i rotační vzdálenost. Z levého obrázku je na první pohled znatelné, že pokud není zahrnuta rotační vzdálenost, je téměř jisté, že propojovací funkce nebude správně pracovat.

Základem propojovací funkce je výpočet počtu interpolačních kroků, který je nutné provést, aby žádný z vrcholů objektu mezi kroky nepřekročil vzdálenost danou parametrem $step_{size}$. Před samotným výpočtem je třeba definovat translační vzdálenost d_{trans} a rotační vzdálenost $d_{angular}$ dvou konfigurací. Translační vzdálenost d_{trans} přímo odpovídá euklidovské vzdálenosti pozic konfigurací. Rotační vzdálenost $d_{angular}$ je vypočtena složitěji. Základem je rozdíl rotací cílové konfigurace q_{to} a zdrojové konfigurace q_{from}:

$$rot_{diff} = q_{to}.rotation - q_{from}.rotation \tag{5.1}$$

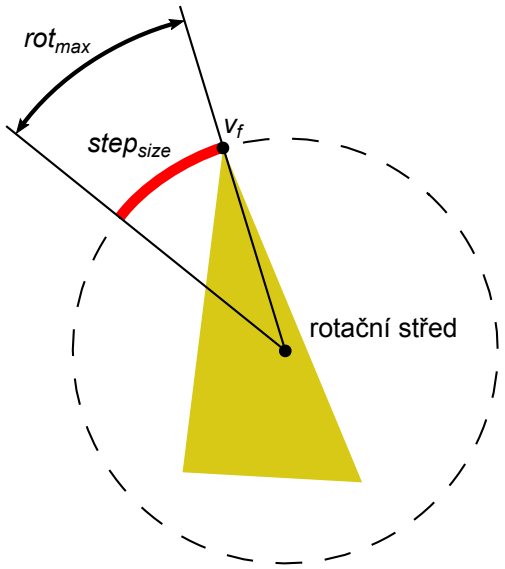
Vektor rot_{diff} je následně normalizován, aby hodnoty všech složek ležely v intervalu $\langle -180, 180 \rangle$. Výsledná rotační vzdálenost $d_{angular}$ pak odpovídá absolutní hodnotě maximální složky:

$$d_{angular} = \max(|rot_{diff}.x|, |rot_{diff}.y|, |rot_{diff}.z|) \tag{5.2}$$

Počet interpolačních kroků je závislý na zdrojové konfiguraci, cílové konfiguraci a na tvaru objektu, který interpolaci provádí. Na základě modelu objektu, který úsečku interpoluje, je proveden výpočet maximální rotace objektu během jednoho kroku rot_{max} (obrázek 5.8):

1. Výpočet nejvzdálenějšího vrcholu objektu v_f od rotačního středu
2. Výpočet obvodu kruhu p, který vrchol v_f vytvoří při rotaci objektu o 360°
3. Maximální rotace rot_{max} (ve stupních) během jednoho kroku je pak určena jako:

$$rot_{max} = 360 \cdot (step_{size}/p) \tag{5.3}$$



Obrázek 5.8: Vizuální reprezentace výpočtu maximální rotace jednoho kroku interpolace rot_{max} při délce kroku interpolace $step_{size}$.

Na základě translační vzdálenosti d_{trans}, rotační vzdálenosti $d_{angular}$, maximální rotaci během jednoho kroku rot_{max} a délce interpolačního kroku $step_{size}$ je pak celkový počet interpolačních kroků $steps_{total}$, které musí propojovací funkce provést, vypočten následovně:

1. Počet kroků pro translační vzdálenost:

$$steps_{trans} = d_{trans}/step_{size} \tag{5.4}$$

2. Počet kroků pro rotační vzdálenost:

$$steps_{angular} = d_{angular}/rot_{max} \tag{5.5}$$

3. Celkový počet kroků:

$$steps_{total} = \sqrt{steps_{trans}^2 + steps_{angular}^2} \tag{5.6}$$

5.4.2 Vzdálenostní funkce

Jako vzdálenostní funkce je použita euklidovská vzdálenost, která je vypočtena z poloh konfigurací. Z optimalizačních důvodů je vynechána odmocnina, která při vzájemném porovnání vzdáleností nemá žádný vliv. Optimalizace je velice znatelná u PRM a EST algoritmů, které často počítají vzdálenosti konfigurací.

Na rozdíl od propojovací funkce není zahrnuta rotace konfigurací, jelikož zde nehraje kritickou roli. Její zahrnutí by zesložitilo výpočet vzdálenosti a hlavně by se znatelně prodloužil její výpočet a tím i celkový běh algoritmů.

5.4.3 Vzorkovací funkce

Vzorkovací funkce, neboli generování náhodných konfigurací, je řízeno rovnoměrným rozložením. Pozice nové konfigurace je vygenerována v rozsahu hranic prostoru, které uživatel zadal při nastavení scény. Rotace je také generována rovnoměrným rozložením v intervalu $\langle 0, 360 \rangle$.

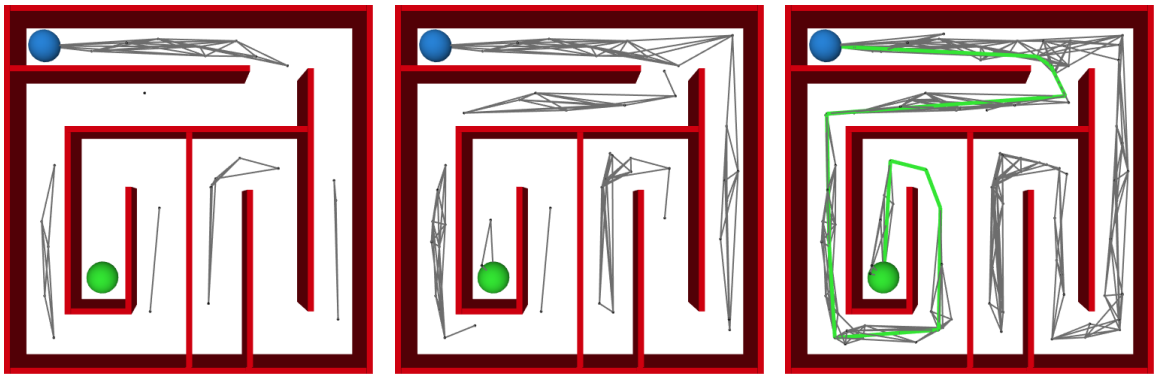
Pokud je třeba generovat novou konfiguraci v okolí konfigurace q_{ref} při velikosti okolí n_{size}, je vygenerována konfigurace q_{new} s náhodnou rotací stejně, jak je uvedeno výše. Pro každou dimenzi pozice je vygenerována hodnota rovnoměrným rozložením v intervalu:

$$\langle q_{ref}.position - n_{size}, q_{ref}.position + n_{size} \rangle \tag{5.7}$$

5.4.4 PRM

PRM algoritmus je implementován v základní formě. Základní verze algoritmu pracuje na principu vytvoření určitého počtu konfigurací, které následně navzájem propojí. Nakonec zkontroluje, zda existuje cesta mezi počáteční a cílovou konfigurací. Pokud cesta nebyla nalezena, je algoritmus ukončen.

Implementovaná verze pracuje v cyklech. Pokud nebyla cesta nalezena, algoritmus není ukončen, ale vygenerují se další konfigurace a celý algoritmus se opakuje. Algoritmus je ukončen jen v případě úspěšného nalezení cesty, anebo pokud uživatel algoritmus přeruší. Konkrétní implementace je uvedena v algoritmu 5.1. Algoritmus je parametrizován počtem generovaných uzlů v jenom cyklu a počtem nejbližších uzlů při propojování jednotlivých uzlů.



Obrázek 5.9: Průběh postupného budování grafu prostředí PRM algoritmu na úloze bludiště. PRM se již od začátku snaží pokrýt celý prostor. EST a RRT algoritmy na druhou stranu postupně pokrývají prostor od počáteční pozice.

Algoritmus 5.1 Implementace PRM algoritmu

Vstup:

Parametry PRM

Scéna s překážkami, počáteční a cílovou konfigurací

Výstup:

Cesta z počáteční do cílové konfigurace (pokud cesta existuje)

1. Nastavení parametrů
2. Inicializace zdrojů
3. **while** v grafu prostředí neexistuje cesta mezi počáteční a cílovou konfigurací **do**
4. Generování náhodných volných konfigurací
5. **for** každá nová konfigurace **do**
6. Vytvoření seřazeného seznamu *list* všech konfigurací podle vzdálenostní funkce
7. **while** *list* je neprázdný **and** nebyl propojen dostatečný počet konfigurací **do**
8. Výběr první položky (nejbližší konfigurace) ze seznamu *list*
9. Pokud lze novou a nejbližší konfiguraci propojit, je mezi nimi vytvořena hrana
10. Odstranění první položky ze seznamu *list*
11. **end while**
12. **end for**
13. Připojení počáteční a cílové konfigurace ke grafu prostředí stejným způsobem jako nově vygenerované konfigurace
14. **end while**
15. Hledání nejkratší cesty pomocí Dijkstrova algoritmu
16. Optimalizace nalezené cesty

5.4.5 EST

Hlavní schéma algoritmu je totožné s algoritmem popsaným v teoretické části. Algoritmus cyklicky generuje nové konfigurace a postupně je připojuje ke stromu prostředí. Po vygenerování určitého počtu konfigurací je proveden test připojení cílové konfigurace. Pokud je cílová konfigurace připojena ke stromu prostředí, tak zbývá nalézt konkrétní posloupnost konfigurací výsledné cesty a cestu optimalizovat. Implementace algoritmu je detailněji popsána v algoritmu 5.2 a 5.3. EST algoritmus je implementován ve dvou variantách. Varianty se liší v implementaci pravděpodobnostní funkce π_T pro výběr uzlů pro expanzi.

První varianta (označena jako EST-Dist) vybírá uzly podle počtu okolních uzlů. Velikost okolí lze nastavit jako parametr před samotným spuštěním algoritmu. Tato varianta však vyžaduje počítání n nejbližších sousedů v každém cyklu, což je výpočetně náročná operace a po delším běhu se algoritmus neúnosně zpomalí.

V důsledku pomalosti algoritmu při experimentech je implementována část SBL algoritmu popsaného v sekci 3.5. Modifikace spočívá v rozdělení prostoru pravidelnou mřížkou. Pravděpodobnostní funkce pak vybírá uzly z buňky s nejmenším počtem uzlů. Rozdělení prostoru zajišťuje struktura, jejímž základem je trojrozměrný seznam buněk prostoru a jednorozměrný prioritní seznam, ve kterém jsou uloženy seřazené odkazy na buňky spolu s počtem uzlů v dané buňce. Pro nalezení buňky s nejmenším počtem uzlů pak stačí přečíst první položku seznamu. V případě, kdy více buněk obsahuje minimální počet uzlů, je vybrána náhodná buňka. Ze zvolené buňky je následně náhodně vybrán konkrétní uzel, který je použit pro expanzi stromu.

Druhá varianta (označena jako EST-Grid) je výrazně rychlejší než první varianta pracující na principu okolních uzlů. Nicméně není tak přesná. Velice záleží na velikosti mřížky (počtu buněk v každé dimenzi). Pokud bude počet buněk příliš nízký, bude se výběr uzlů blížit náhodnému výběru, protože pokaždé bude vybrán uzel ze stejné buňky. Varianta s mřížkou dobře pracuje ve stísněných prostorech, protože zvládne generovat velké množství uzlů, a tak rychle pokrýt úzké místo. Parametry EST algoritmu:

- Počet generovaných konfigurací v jenom cyklu, neboli po kolika uzlech bude proveden test připojení cílové konfigurace.
- Velikost okolí, ve kterém jsou generovány nové uzly
- Typ pravděpodobnostní funkce:
  - Počet okolních uzlů: velikost okolí
  - Rozdělení prostoru mřížkou: velikost mřížky

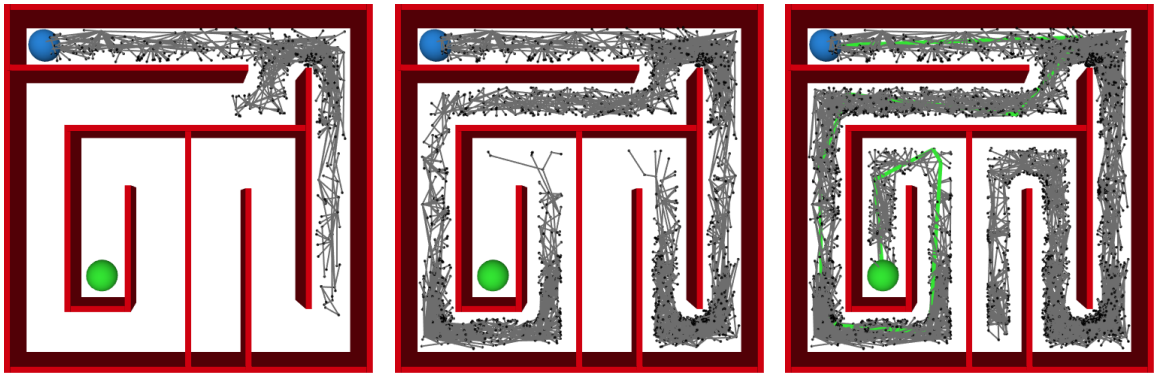
Algoritmus 5.2 Implementace EST algoritmu

Vstup:
Parametry EST
Scéna s překážkami, počáteční a cílovou konfigurací
Výstup:
Cesta z počáteční do cílové konfigurace (pokud cesta existuje)

1. Nastavení parametrů
2. Inicializace zdrojů
3. Uložení počáteční konfigurace do grafu prostředí
4. **while** cílová konfigurace q_{goal} nelze připojit ke stromu prostředí **do**
5. **for** počet konfigurací jednoho cyklu **do**
6. *generateNodesEST()*
7. **end for**
8. Pokus o připojení cílové konfigurace
9. **end while**
10. Konstrukce posloupnosti uzlů cesty výsledné cesty
11. Optimalizace nalezené cesty

Algoritmus 5.3 Implementace generování uzlů EST algoritmu (*generateNodesEST*)

1. $q_{expand} \leftarrow$ výběr uzlu pro expanzi stromu prostředí podle pravděpodobnostní funkce π_T. Úplně na počátku běhu bude ve stromu jen počáteční uzel. Zde je rozdíl výše popsaných variant EST-Dist a EST-Grid.
2. $q_{new} \leftarrow$ nová volná konfigurace vygenerovana v okolí uzlu q_{expand}
3. **if** q_{expand} a q_{new} lze propojit **then**
4. Uložení q_{new} do stromu prostředí spolu s hranou (q_{expand}, q_{new})
5. **else**
6. Výpočet nejvzdálenější bezkolizní konfigurace $q_{contact}$
7. Uložení $q_{contact}$ do stromu prostředí spolu s hranou (q_{expand}, $q_{contact}$)
8. **end if**



Obrázek 5.10: Průběh postupného budování stromu prostředí EST-Grid algoritmu na úloze bludiště. Algoritmus postupně pokrývá prostor od počáteční k cílové pozici. Na první pohled je znatelné, že EST-Grid generuje velké množství uzlů.

5.4.6 RRT

RRT algoritmus je implementován ve třech verzích. První verzí je základní RRT algoritmus. Druhou verzí je modifikace Retraction-based RRT (R-RRT), který se snaží řešit problém úzkých průchodů. Poslední verzí je Regulated R-RRT (RR-RRT), který se snaží odstranit negativní vlastnosti R-RRT algoritmu.

Implementovaná verze RRT algoritmu se trochu liší od algoritmus popsaného v teoretické části. Základní algoritmus vygeneruje náhodnou konfiguraci q_{rand} a k ní nalezne nejbližší konfiguraci q_{near}. Následně je vytvořena nová konfigurace q_{new}, která leží v pevné vzdálenosti od q_{near} mezi q_{rand}. Až s touto konfigurací q_{new} je proveden test propojitelnosti a případně je přidána do grafu. Při experimentech ale pracovala lépe verze, kdy se propojuje přímo konfigurace q_{rand} a generování mezi-konfigurace v pevné vzdálenosti je vynechána.

Druhým implementačním detailem je, jak se má algoritmus zachovat, pokud nelze dvě konfigurace propojit. Základní verze v tomto případě znovu vygeneruje novou náhodnou konfiguraci a test propojitelnosti se opakuje. Situace je implementována tak, že pokud nelze dvě konfigurace propojit, nalezne se nejvzdálenější nekolidující konfigurace (konfigurace nejblíže překážce) a ta se spolu s hranou uloží do stromu.

Základní verze i její rozšíření jsou implementovány pomocí jedné třídy a jednotlivé varianty jsou aktivovány pomocí parametrů. Základ RRT algoritmu (algoritmus 5.4) je téměř totožný s EST algoritmem. Hlavní rozdíl je ve způsobu generování nových uzlů/konfigurací. Algoritmus 5.5 obsahuje implementaci generování nových uzlů spolu s RRT variantami, které jsou popsány dále.

Výhodou RRT je, že na rozdíl od EST nepotřebuje téměř žádné parametry. EST totiž vyžaduje definici velikosti okolí nových konfigurací a parametry pravděpodobnostní funkce π_T. Špatná volba parametrů může velice negativně ovlivnit celkový čas hledání cesty. Jediným parametrem RRT algoritmu je počet generovaných uzlů v jednom cyklu, neboli po kolika uzlech bude proveden test připojení cílové konfigurace. Tento parametr navíc žádným zásadním způsobem neovlivňuje běh algoritmu.

Algoritmus 5.4 Implementace RRT algoritmu

Vstup:

Parametry RRT

Scéna s překážkami, počáteční a cílovou konfigurací

Výstup:

Cesta z počáteční do cílové konfigurace (pokud cesta existuje)

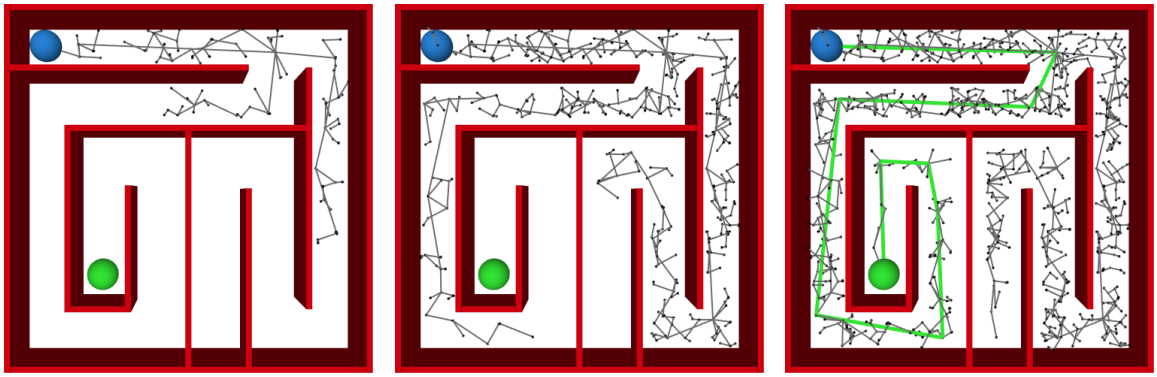
1. Nastavení parametrů
2. Inicializace zdrojů
3. Uložení počáteční a cílové konfigurace do grafu prostředí
4. **while** cílová konfigurace q_{goal} nelze připojit ke stromu prostředí **do**
5. **for** počet konfigurací jednoho cyklu **do**
6. *generateNodesRRT()*
7. **end for**
8. Pokus o připojení cílové konfigurace
9. **end while**
10. Konstrukce posloupnosti uzlů cesty výsledné cesty
11. Optimalizace nalezené cesty

Algoritmus 5.5 Implementace generování uzlů RRT algoritmu (*generateNodesRRT*)

1. $q_{rand} \leftarrow$ náhodně generovaná konfigurace
2. $q_{near} \leftarrow$ nejbližší konfigurace k q_{rand}
3. **if** q_{rand} a q_{near} lze propojit **then**
4. Uložení q_{rand} do stromu prostředí spolu s hranou (q_{near}, q_{rand})
5. **else**
6. Výpočet nejvzdálenější bezkolizní konfigurace $q_{contact}$
7. **if** R-RRT modifikace **then**
8. **if** RR-RRT modifikace **then**
9. **if** *narrowTest*$(q_{contact}, q_{rand})$ **then**
10. *retractionStep*$(q_{contact}, q_{rand})$
11. **end if**
12. **else**
13. *retractionStep*$(q_{contact}, q_{rand})$
14. **end if**
15. **else**
16. Uložení $q_{contact}$ do stromu prostředí spolu s hranou $(q_{near}, q_{contact})$
17. **end if**
18. **end if**

R-RRT

R-RRT rozšiřuje základní RRT v případě kolize s překážkou spuštěním retrakčního kroku. Myšlenkou retrakčního kroku je detailnější prohledávání prostoru blízko překážky. To je provedeno iterativním generováním několika konfigurací v blízkém okolí místa kolize s překážkou. Takto je provedeno několik iterací, přičemž v každé iteraci by se nové konfigurace měly postupně blížit cílové konfiguraci. Optimalizace je ukončena při provedení zadaného počtu iterací nebo pokud lze propojit cílová konfigurace.



Obrázek 5.11: Průběh postupného budování stromu prostředí RRT algoritmu na úloze bludiště.

Konkrétní implementace je uvedena v algoritmu 5.6. Retrakční krok je parametrizován počtem iterací $iter_{count}$, počtem vygenerovaných konfigurací v jedné iteraci q_{count} a velikostí okolí generovaných konfigurací. Retrakční krok postupně hledá blížící se konfigurace k cílové konfiguraci a všechny průběžné konfigurace ukládá do seznamu. Po provedení všech iterací pak zkusí připojit každou konfiguraci ke stromu prostředí. Připojení je provedeno základním způsobem přes nejbližší konfiguraci.

Algoritmus 5.6 Implementace retrakčního kroku R-RRT algoritmu (*retractionStep*)

Vstup:

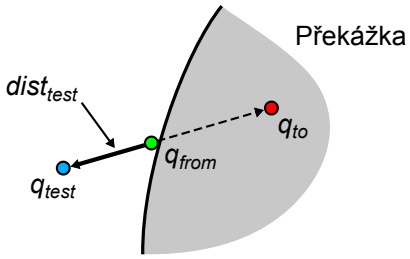
q_{from}: zdrojová konfigurace
q_{to}: cílová konfigurace

1. $q_{current} \leftarrow q_{from}$
2. **for** počet iterací retrakčního kroku $iter_{count}$ **do**
3. Generování q_{count} náhodných konfigurací v okolí q_{from}
4. $q_{nearest} \leftarrow$ nejbližší konfigurace k q_{to}
5. **if** vzdálenost mezi $q_{nearest}$ a q_{to} je menší než $q_{current}$ a q_{to} **then**
6. $q_{current} \leftarrow q_{nearest}$
7. Uložení konfigurace $q_{nearest}$ do seznamu optimalizovaných konfigurací
8. **end if**
9. **if** $q_{current}$ lze propojit s q_{to} **then**
10. Uložení konfigurace q_{to} do seznamu optimalizovaných konfigurací
11. `break`
12. **end if**
13. **end for**
14. **for all** konfigurace ze seznamu optimalizovaných konfigurací **do**
15. Připojení konfigurace ke stromu standardním přípojením přes nejbližší konfiguraci
16. **end for**

RR-RRT

RR-RRT algoritmus se snaží odstranit nedostatek R-RRT algoritmu, kdy se provádí retrakční krok při každé kolizi s překážkou, což nemusí nutně znamenat, že se algoritmus nachází v úzkém průchodu. Retrakčnímu kroku tak předchází test, který určí, zda se algoritmus pravě nachází v úzkém průchodu a je žádoucí provést detailnější vzorkování prostoru.

Konkrétní implementace je uvedena v algoritmu 5.7. Algoritmus při kolizi s překážkou vygeneruje novou konfiguraci q_{test} ve směru q_{to} k q_{from}. Testovací konfigurace je generována v pevné vzdálenosti od q_{from} a lze ji nastavit před spuštěním algoritmu. Velikost vzdálenosti prakticky určuje, jak velký úzký průchod bude detailněji prohledáván.



Obrázek 5.12: Vizuální reprezentace testu úzkého průchodu

Algoritmus 5.7 Implementace testu úzkého průchodu RR-RRT algoritmu (*narrowTest*)

Vstup:

q_{from}: zdrojová konfigurace
q_{to}: cílová konfigurace

Výstup:

Zda se konfigurace nachází v úzkém průchodu

1. $q_{test} \leftarrow$ extrapolace konfigurace ve směru q_{to} do q_{from} ve vzdálenosti $dist_{test}$ od q_{from}
2. Rotace konfigurace q_{test} je stejná jako rotace cílové konfigurace q_{to}
3. **return** konfigurace q_{test} koliduje s překážkou

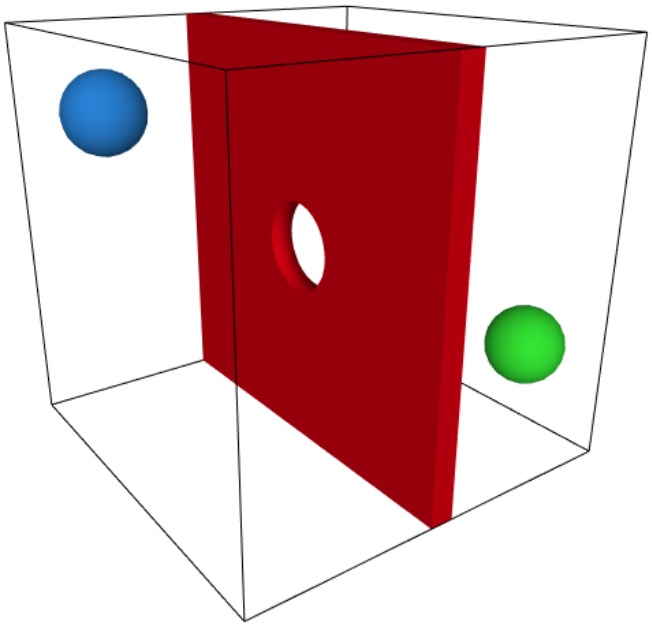
Kapitola 6

Experimenty

Cílem experimentů je ověřit funkčnost programu a výkon jednotlivých algoritmů. Experimenty jsou rozděleny do dvou sad. První sada obsahuje stejnou scénu se zvyšující se složitostí. Druhá sada obsahuje soubor několika různých scén. Pro každý experiment bylo provedeno 30 běhů. Ze všech běhů byl vypočítán průměr a výsledné časy jsou uvedeny v jednotlivých tabulkách. Parametry algoritmů pro každý experiment jsou uvedeny na konci kapitoly. Experimenty proběhly na procesoru Intel Core i5-4460 3,2GHz.

6.1 První sada experimentů

První sada experimentů je provedena na jednoduché scéně se stěnou s otvorem (obrázek 6.1). Přičemž je provedeno několik experimentů s různou velikostí otvoru. Konkrétně s velikostmi otvoru 1,25, 1,1, 1,05 a 1,01. Velikost označuje, kolikrát je otvor větší oproti kouli, která musí otvorem projít. Velikost 1,25 znamená, že průměr otvoru je o 25 % větší než průměr koule. Výsledky experimentů jsou uvedeny v tabulce 6.1.



Obrázek 6.1: Stěna s otvorem

	Stěna 1,25	**Stěna 1,1**	**Stěna 1,05**	**Stěna 1,01**
PRM	25,5s	–	–	–
EST-Dist	33,9s	342,1s	–	–
EST-Grid	1,5s	6,2s	38,5s	–
RRT	0,3s	1,7s	3,8s	15,9s
R-RRT	13,3s	21,5s	28,8s	40,4s
RR-RRT	0,5s	5,7s	8,8s	19,4s

Tabulka 6.1: Průměrné časy algoritmů na sadě experimentů se stěnou s otvorem. Pomlčky značí, že algoritmus nebyl schopen úlohu vyřešit do jedné hodiny.

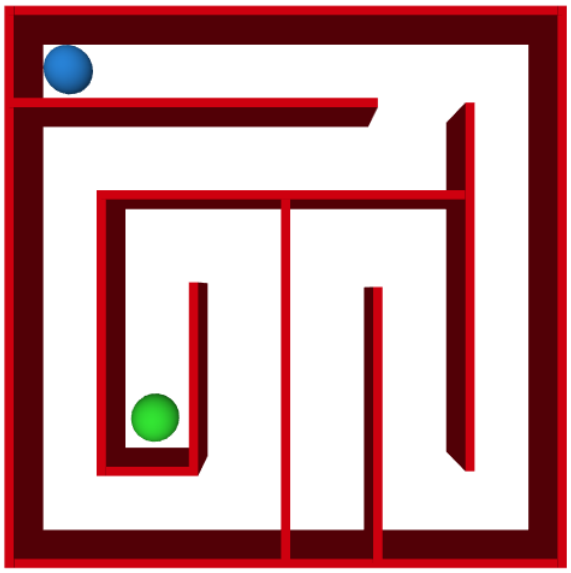
PRM spolehlivě vyřešil jen stěnu 1,25 a přitom skoro nejpomaleji. PRM byl schopný vyřešit stěnu 1,1, ale velice záleželo, kdy byla vygenerována konfigurace uvnitř otvoru. Pokud byla konfigurace vygenerována na počátku běhu, zbývalo jen navzorkovat okolí stěny a řešení bylo nalezeno. Pokud ale konfigurace nebyla vygenerována na počátku, následné generování bylo natolik zpomaleno, že se potřebná konfigurace nevygenerovala v rozumném čase. V několika případech byl algoritmus schopen cestu nalézt během 20 sekund, ale ve většině případů ji nenašel ani během hodiny. U stěn s menšími otvory nebyl PRM schopný cestu nalézt do jedné hodiny. Neúspěch a pomalost PRM algoritmu tkví v častém počítání n nejbližších sousedů, což je časově náročná operace. Druhým, možná i větším zpomalením, je testování, zda je počáteční a cílová konfigurace propojena (na grafu prostředí s desítkami tisíc uzlů). EST a RRT algoritmy tímto netrpí, protože ty nalezení cesty detekují prostým napojení cílové konfigurace ke stromu.

Časová složitost EST-Dist je podobná jako u PRM, takže jsou podobné i výsledky. EST-Grid zlepšuje špatnou časovou složitost EST-Dist, a tak dokáže vyřešit i scény s menším otvorem. Nicméně stěnu 1,01 nebyl schopen vyřešit do jedné hodiny. Navíc při dlouhém běhu už začínala být zjevná prostorová složitost. Pro extrémně dlouhé doby výpočtu by bylo nutné sledovat zaplnění paměti.

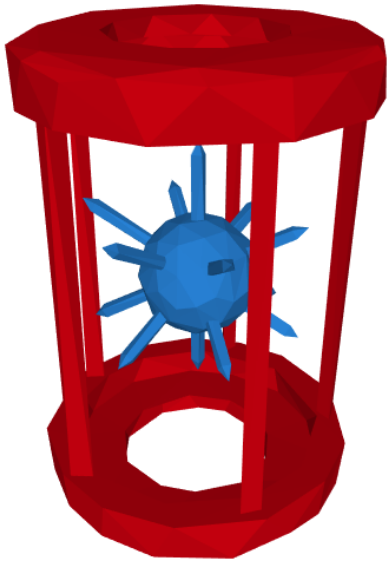
Nejlepší výsledky, trochu překvapivě, podává základní RRT. R-RRT algoritmus, jak už bylo zmíněno, špatně pracuje na scénách bez úzkého průchodu a výsledné časy tomu odpovídají. Scéna sice obsahuje úzký průchod, ale zároveň obsahuje místa s překážkou a bez úzkého průchodu, kde je detailnější vzorkování spouštěno zbytečně a celý algoritmus je jen zpomalen. RR-RRT však výrazně R-RRT vylepšuje filtrováním případů, kdy by se detailnější vzorkování spouštělo zbytečně. Režie spojená s filtrováním případů a fakt, že ne všechny případy jsou filtrovány, mají za následek, že výsledky nejsou úplně porovnatelné se základní verzí RRT. Se snižováním velikosti otvoru se však nutná režie testování úzkého průchodu v celkovém času ztrácí.

6.2 Druhá sada experimentů

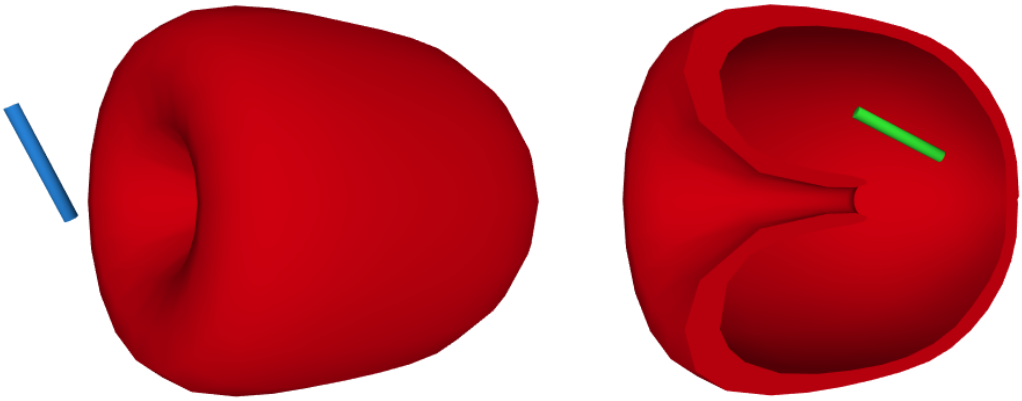
Druhá sada experimentů obsahuje několik složitějších scén. Prvním experimentem je jednoduché bludiště (obrázek 6.2). Druhý experiment je proveden na často se vyskytujícím problému „past na brouka“ (obrázek 6.4). Cílem je nalézt cestu pro brouka dovnitř pasti. Brouk přitom musí projít úzkým průchodem pasti, ale zároveň obsahuje prostor, kterým brouk vůbec nemusí projít. Posledním experimentem je hlavolam „ježek v kleci“ (obrázek 6.3). Cílem je vytáhnout ostnatou kouli (ježka) mimo klec. Ježek v kleci je relativně náročný na vyřešení, jelikož vyžaduje manipulaci objektu v malém prostoru. Výsledky druhé sady experimentů jsou uvedeny v tabulce 6.2.



Obrázek 6.2: Jednoduché bludiště



Obrázek 6.3: Hlavolam ježek v kleci



Obrázek 6.4: Past na brouka. Past je označena červenou barvou (vpravo je průřez pasti), počáteční pozice brouka modře a cílová pozice zeleně.

PRM algoritmus vyřešil bludiště nejrychleji, ale jen díky souvislé struktuře bludiště, kde není nutné projít úzkými průchody. Past na brouka a ježka v kleci nebyl algoritmus schopný vyřešit do jedné hodiny. Důvodem je, že např. u ježka v kleci algoritmus vzorkuje hlavně okolní prostor a vnitřek klece je málo pokrytý. Výhody EST a RRT algoritmů, oproti PRM, jsou právě v procházení jen dosažitelného prostoru.

EST-Dist stejně jako PRM vyřešil pouze bludiště, protože pro nalezení cesty není nutné vytvořit mnoho uzlů a tím pádem se na malém počtu uzlů neukáže špatná časová složitost. Past na brouka a ježek v kleci jsou pro algoritmus příliš složité úlohy. Rychlost nalezení cesty v bludišti EST-Grid algoritmem je znatelně pomalejší než rychlost PRM nebo EST-Dist. Avšak EST-Grid algoritmus dokázal vyřešit ježka v kleci, ale jen díky generování velkého množství uzlů (zhruba 100krát více uzlů než u R-RRT).

	Bludiště	Past na brouka	Ježek v kleci
PRM	1,3s	–	–
EST-Dist	2,5s	–	–
EST-Grid	5,1s	–	472,1s
RRT	1,4s	143,2s	385,1s
R-RRT	37,2s	336,2s	41,0s
RR-RRT	2,0s	114,0s	29,9s

Tabulka 6.2: Průměrné časy algoritmů na druhé sadě experimentů. Pomlčky značí, že algoritmus nebyl schopen úlohu vyřešit do jedné hodiny.

RRT a její varianty podaly nejlepší výsledky. R-RRT algoritmus, v porovnání s RRT, podal lepší výsledek jen u ježka v kleci, kde je znatelně rychlejší. Nicméně ve scénách, kde není nutné projít úzkým průchodem, výkon R-RRT algoritmu výrazně degraduje, protože vzorkuje okolí překážek, i když to není nutné. To je vidět na prvním experimentu bludiště, kde byl R-RRT algoritmus mnohonásobně pomalejší než jeho základní verze nebo RR-RRT. Past na brouka sice obsahuje úzký průchod, ale zároveň obsahuje velkou oblast volného prostoru, kterým brouk vůbec nemusí projít. R-RRT podává lepší výsledky, jen pokud je nutné projít úzkým průchodem a pokud je objekt uzavřen v malém prostoru. Úplně nejlepší výsledky podal RR-RRT, který odstraňuje nedostatek R-RRT přílišného vzorkování překážek. V případě bludiště a pasti na brouka je vylepšení velice znatelné. U ježka v kleci není zlepšení tak markantní, ale i přesto nezanedbatelné.

6.3 Parametry algoritmů

V tabulce 6.3 jsou uvedeny parametry algoritmů, které byly použity při každém experimentu. V tabulce 6.4 je souhrnný popis významu jednotlivých parametrů.

Parametry		Stěna	Bludiště	Past na brouka	Ježek v kleci
PRM	$node_{count}$	50	50	50	250
	$edge_{count}$	5	5	5	5
EST-Dist	$node_{count}$	50	50	50	500
	$area_{size}$	0,5	2	1	0,5
	$neigh_{size}$	1	2	1	1
EST-Grid	$node_{count}$	50	50	50	500
	$area_{size}$	0,5	5	1	0,5
	$grid_{size}$	10	10	10	25
RRT	$node_{count}$	50	50	50	250
R-RRT	$iter_{count}$	3	3	3	3
	q_{count}	15	15	15	15
	$area_{size}$	1	1	1	1
RR-RRT	$dist_{test}$	0,25	0,25	0,25	0,25

Tabulka 6.3: Parametry algoritmů každého experimentu

Parametr	Význam
$node_{count}$	počet generovaných konfigurací v jednom cyklu
$edge_{count}$	počet nejbližších uzlů, s kterými bude nově vygenerovaný uzel propojen
$area_{size}$	velikost okolí náhodně generovaných uzlů
$neigh_{size}$	velikost okolí při počítání počtu sousedních uzlů
$grid_{size}$	velikost mřížky, neboli počet buněk pro každou dimenzi mřížky
$iter_{count}$	počet iterací retrakčního kroku
q_{count}	počet generovaných konfigurací v jedné iteraci retrakčního kroku
$dist_{test}$	vzdálenost konfigurace testu úzkého průchodu

Tabulka 6.4: Souhrn parametrů a jejich význam

6.4 Shrnutí

Jelikož je PRM algoritmus implementován v základní verzi, podal tak celkově nejhorší výsledky, přičemž složitější úlohy nebyl schopen vůbec vyřešit. Algoritmus není vhodný pro úlohy se stísněným prostorem, jako např. ježek v kleci nebo past na brouka. Algoritmus se totiž snaží pokrýt co největší prostor, a málo se tak soustředí na uzavřené prostory. Na úloze ježka v kleci to znamená, že je hojně pokryto okolí klece, ale vnitřní prostor klece je pokryt příliš málo na to, aby byla úloha vyřešena. Stejně tak algoritmus dobře nepracuje ve scénách, které obsahují robotem nedosažitelná místa, protože je i tak vzorkuje.

EST algoritmus na zvolených scénách pracoval lépe něž PRM algoritmus, ale pořád by bylo vhodné implementovat důmyslnější řešení překážek. Nejdůležitější částí EST je pravděpodobnostní funkce π_T, která zásadním způsobem ovlivňuje celkový výkon algoritmu. Avšak vymyslet a implementovat funkci tak, aby vybírala nejosamocenější uzly a zároveň byla výpočetně efektivní, je náročné. Implementovaná varianta EST-Dist s počítáním sousedních uzlů je časově neefektivní a výsledky jsou podobné jako u PRM algoritmu. Druhá varianta EST-Grid, která rozděluje prostor mřížkou, pracuje sice lépe, ale pořád nedosahuje výsledků RRT algoritmu. Navíc varianta generuje velké množství uzlů a při dlouhém běhu už je potřeba brát ohled na velikost paměti.

Celkově nejúspěšnější byl algoritmus RRT a jeho varianty. To částečně vysvětluje, že velká část článků zabývající se plánováním pohybu rozšiřuje právě RRT algoritmus. Na jednoduchých scénách dobře pracuje i základní RRT algoritmus. Vylepšení R-RRT celkově pracuje hůře než jeho základní verze, jelikož detailněji prohledává každou překážku bez ohledu na to, zda je to právě nutné. Nicméně dobře pracuje ve scénách, které vyžadují manipulaci objektu ve stísněném prostoru a zároveň neobsahují volný prostor. Nedostatek R-RRT částečně odstraňuje RR-RRT algoritmus, který detailnější prohledávání provádí jen v úzkých průchodech. Na jednoduchých scénách bez úzkých průchodů je pořád lepší základní RRT, ale u složitějších scén s kombinací úzkých průchodů je vylepšení velice znatelné.

Kapitola 7

Závěr

Cílem této práce bylo vytvořit program pro řešení a demonstraci problému plánování pohybu ve 3D prostoru pomocí pravděpodobnostních algoritmů. Uživatel pak vybere některý plánovací algoritmus spolu s jeho parametry a program automaticky cestu vyhledá. Po nalezení cesty by měl program umožnit výslednou cestu vizualizovat.

Výsledný program umožňuje vytvořit scénu z překážek, jejíž modely lze načíst ze souboru Wavefront OBJ formátu. Vytvořenou scénu lze následně uložit do souboru v XML formátu a později načíst. Uživatel má na výběr z několika implementovaných pravděpodobnostních algoritmů, mezi které patří PRM, EST, RRT, R-RRT a vlastní rozšíření RR-RRT. Po výběru algoritmu a nastavení parametrů program sám cestu vyhledá. V průběhu hledání cesty se zároveň ve scéně zobrazuje graf prostředí, jak jej daný algoritmus buduje. Na grafu prostředí lze pozorovat, na jakém principu jednotlivé algoritmy pracují. V případě nalezení cesty lze cestu vizualizovat. Vizualizace je provedena animací objektů po nalezené cestě. Animaci lze kdykoli pozastavit, změnit rychlost pohybu objektu nebo nastavit konkrétní pozici na cestě. Stejně jako vytvořenou scénu i nalezenou cestu lze uložit a později načíst, takže není nutné při každém spuštění aplikace cesty znovu hledat.

Na implementovaných algoritmech bylo provedeno několik experimentů. Prvním experimentem byla stěna s otvorem, přičemž velikost otvoru se v jednotlivých experimentech měnila. Dalšími experimenty bylo jednoduché bludiště a často vyskytující se problém „past na brouka“ a hlavolam „ježek v kleci“. Experimenty byly vyhodnoceny a byly popsány některé nedostatky algoritmů. Nejlepší výsledky podal algoritmus RRT a jeho varianty R-RRT a RR-RRT. Dobrý výkon algoritmů se odráží ve faktu, že velká část článků a prací o plánování pohybu je založena právě na RRT algoritmu.

Pro návrh a implementaci uživatelského rozhraní bylo nutné nastudovat práci s 3D modely a OpenGL. Dále bylo nutné nastudovat již zmíněné algoritmy. Výsledný program využívá multiplatformních knihoven, takže jej lze použít na různých operačních systémech.

Vhodným rozšířením by bylo implementovat K-D strom nebo jiný mechanismus, který by urychlil hledání nejbližších sousedů, čímž by se výrazně algoritmy urychlily, zejména EST a PRM. Bez ohledu na použitý algoritmus je výpočetně nejnáročnější částí detekce kolizí s překážkami (zhruba 80% celkového času výpočtu). Velkým urychlením by bylo implementovat paralelní detekci kolizí nebo vytvořit implementaci pro grafické karty.

Literatura

[1] AKINC, M.; BERKIS, K.; CHEN, B.; aj. : Probabilistic Roadmaps of Trees for Parallel Computation of Multiple Query Roadmaps. *Robotics Research. The Eleventh International Symposium*, 2005.

[2] AMATO, N. M.; BAYAZIT, O. B.; DALE, L. K. : OBPRM: An Obstacle-Based PRM for 3D Workspaces. *Robotics: The Algorithmic Perspective*, 1998: s 156–168.

[3] BOOR, V.; OVERMARS, M.; VAN DER STAPPEN, A. : The Gaussian sampling strategy for probabilistic roadmap planners. *IEEE International Conference on Robotics and Automation*, roč. 2, 1999: s 1018–1023, ISSN 1050-4729.

[4] CHOSET, H. M. *Principles of robot motion : theory, algorithms, and implementation*. Cambridge, Mass.: MIT Press, 2005. ISBN 02-620-3327-5.

[5] CORROCHANO, E. B. *Handbook of geometric computing : applications in pattern recognition, computer vision, neuralcomputing, and robotics*. New York: Springer, 2005. ISBN 35-402-0595-0.

[6] HSU, D.; JIANG, T.; REIF, J.; aj. : The bridge test for sampling narrow passages with probabilistic roadmap planners. *IEEE International Conference on Robotics and Automation*, roč. 3, 2003, ISSN 1050-4729.

[7] KAVRAKI, L. E. : Protein-Ligand Docking, Including Flexible Receptor-Flexible Ligand Docking [online]. [cit. 20.4.2016]. Dostupné z: <http://cnx.org/content/m11456/>

[8] KAVRAKI, L. E.; SVESTKA, P.; LATOMBE, J.-C.; aj. : Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, roč. 12, č. 4, 1996: s 566–580, ISSN 1042-296X.

[9] KUFFNER, J.; LAVALLE, S. : RRT-connect: An efficient approach to single-query path planning. *IEEE International Conference on Robotics and Automation*, roč. 2, 2000: s 995–1001, ISSN 1050-4729.

[10] LAVALLE, S. M. *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006. ISBN 05-218-6205-1.

[11] RRRobotica : Articulated industrial robot ATOM 10 [online]. [cit. 20.4.2016]. Dostupné z: <http://www.rrrobotica.it/atom10_e.htm>

[12] SAHA, M.; LATOMBE, J.-C. : Finding narrow passages with probabilistic roadmaps: the small step retraction method. *International Conference on Intelligent Robots and Systems*, 2005: s 622–627.

[13] SÁNCHEZ, G.; LATOMBE, J.-C. : On Delaying Collision Checking in PRM Planning. *Robotics Research. The Tenth International Symposioum*, 2003: s 403–417, ISSN 1610-7438.

[14] ZHANG, L.; HUANG, X.; KIM, Y. J.; aj. : D-Plan: Efficient Collision-Free Path Computation for Part Removal and Disassembly. *Journal of Computer-Aided Design and Applications*, roč. 5, č. 5, 2008: s 774–786.

[15] ZHANG, L.; MANOCHA, D. : An efficient retraction-based RRT planner. *IEEE International Conference on Robotics and Automation*, 2008: s 3743–3750, ISSN 1050-4729.

Přílohy

Seznam příloh

Příloha A

Obsah CD

Přiložené CD obsahuje zdrojové soubory aplikace, přeložený program pro systém Windows se všemi potřebnými knihovnami, technickou zprávu a zdrojové LaTeX soubory technické zprávy. Dále CD obsahuje prezentační materiály pro konferenci Excel@FIT2016, na které byla práce prezentována.

- `Aplikace/` – soubory aplikace
  - `PathPlanning/` – spustitelný program pro platformu Windows
  - `src/` – zdrojové soubory aplikace
    - `NetBeans/` – projekt ve vývojovém prostředí NetBeans
    - `QtCreator/` – projekt ve vývojovém prostředí QtCreator
    - `README.txt` – informace pro překlad programu
- `Excel@FIT2016/` – prezentační materiály pro konferenci Excel@FIT2016
  - `diplomy/` – získaná ocenění
  - `src/` – zdrojové soubory prezentačních materiálů
  - `clanek.pdf` – článek
  - `nahled.png` – ikona práce
  - `plakat.pdf` – A3 plakát práce
  - `shrnujici_snimek.png` – shrnující snímek práce
  - `video.mp4` – video s ukázkou programu
- `Zprava/` – soubory technické zprávy
  - `LaTeX/` – zdrojové soubory technické zprávy
  - `zprava.pdf` – technická zpráva

Příloha B

Ukázky XML souborů

B.1 Scéna

```
<scene>
  <models> <!-- Seznam modelů pro robota a překážky -->
    <model name="cube" filepath="./cube.obj"/>
    <model name="sphere" filepath="./sphere.obj"/>
  </models>
  <space>
    <boundary> <!-- Hranice prostoru -->
      <x min="-8" max="8"/>
      <y min="-8" max="8"/>
      <z min="-8" max="8"/>
    </boundary>
    <robot> <!-- Nastavení robota -->
      <model name="sphere"/>
      <scale x="1" y="1" z="1"/>
    </robot>
    <start> <!-- Počáteční pozice -->
      <position x="-5" y="0" z="0"/>
      <orientation x="45" y="45" z="0"/>
    </start>
    <goal> <!-- Cílová pozice -->
      <position x="5" y="0" z="0"/>
      <orientation x="0" y="60" z="30"/>
    </goal>
    <obstacles> <!-- Seznam překážek -->
      <obstacle>
        <model name="cube"/>
        <position x="0" y="0" z="-1.5"/>
        <orientation x="0" y="0" z="0"/>
        <scale x="0.2" y="8" z="6.5"/>
      </obstacle>
    </obstacles>
  </space>
</scene>
```

B.2 Cesta

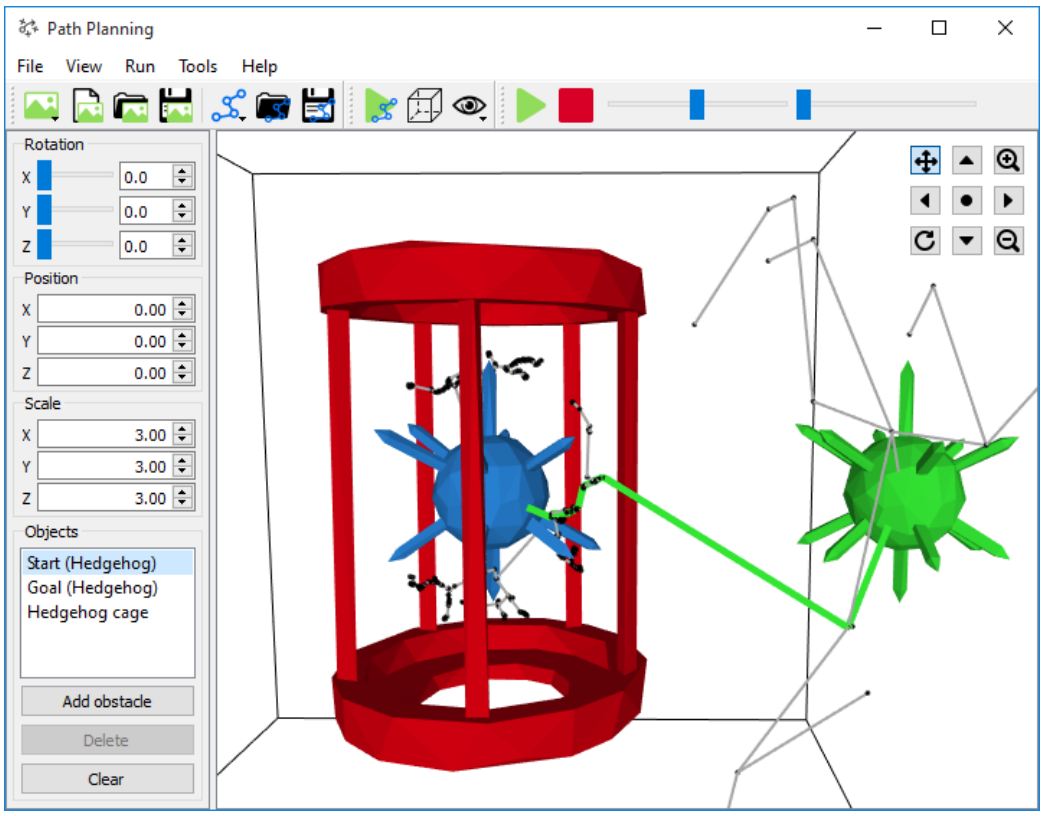
```
<path>
  <scene file="./sceneWall.xml"/>
  <nodes>  <!-- Seznam uzlů cesty -->
    <node>
      <position x="-5" y="0" z="0"/>
      <orientation x="45" y="45" z="0"/>
    </node>
    <node>
      <position x="-2.53518" y="-1.29151" z="5.65552"/>
      <orientation x="70.3838" y="170.797" z="341.866"/>
    </node>
    <node>
      <position x="-0.156866" y="-0.360711" z="6.55"/>
      <orientation x="8.1724" y="295.703" z="65.833"/>
    </node>
    <node>
      <position x="7.11636" y="-2.64136" z="5.78804"/>
      <orientation x="15.2105" y="249.095" z="157.197"/>
    </node>
    <node>
      <position x="5" y="0" z="0"/>
      <orientation x="60" y="30" z="0"/>
    </node>
  </nodes>
</path>
```

Příloha C

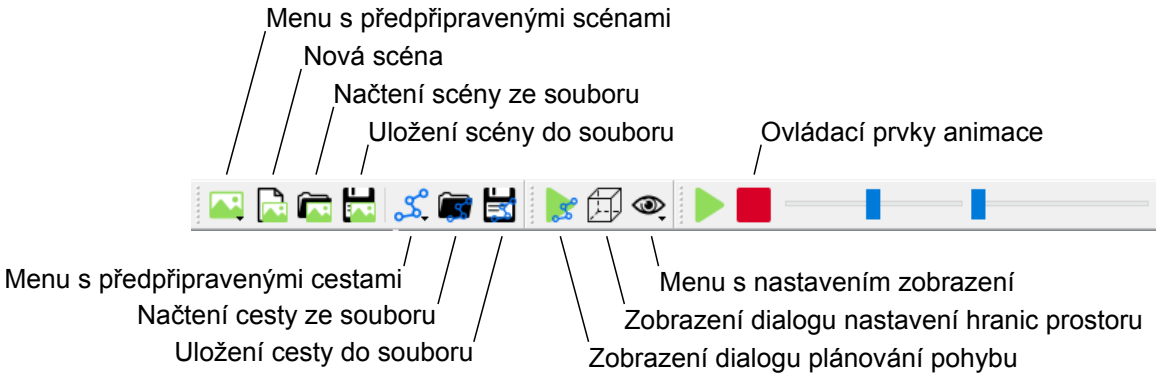
Manuál

Okno programu je rozděleno na tři hlavní části:

1. OpenGL komponenta s 3D zobrazením scény
2. Levý panel pro nastavení konfigurace objektů (poloha, rotace a velikost) spolu se seznamem všech objektů ve scéně
3. Horní panel s prvky pro načtení a uložení scény a cesty, ovládání plánování pohybu a ovládání animace nalezené cesty



Obrázek C.1: Hlavní okno programu



Obrázek C.2: Horní ovládací panel

C.1 Ovládání

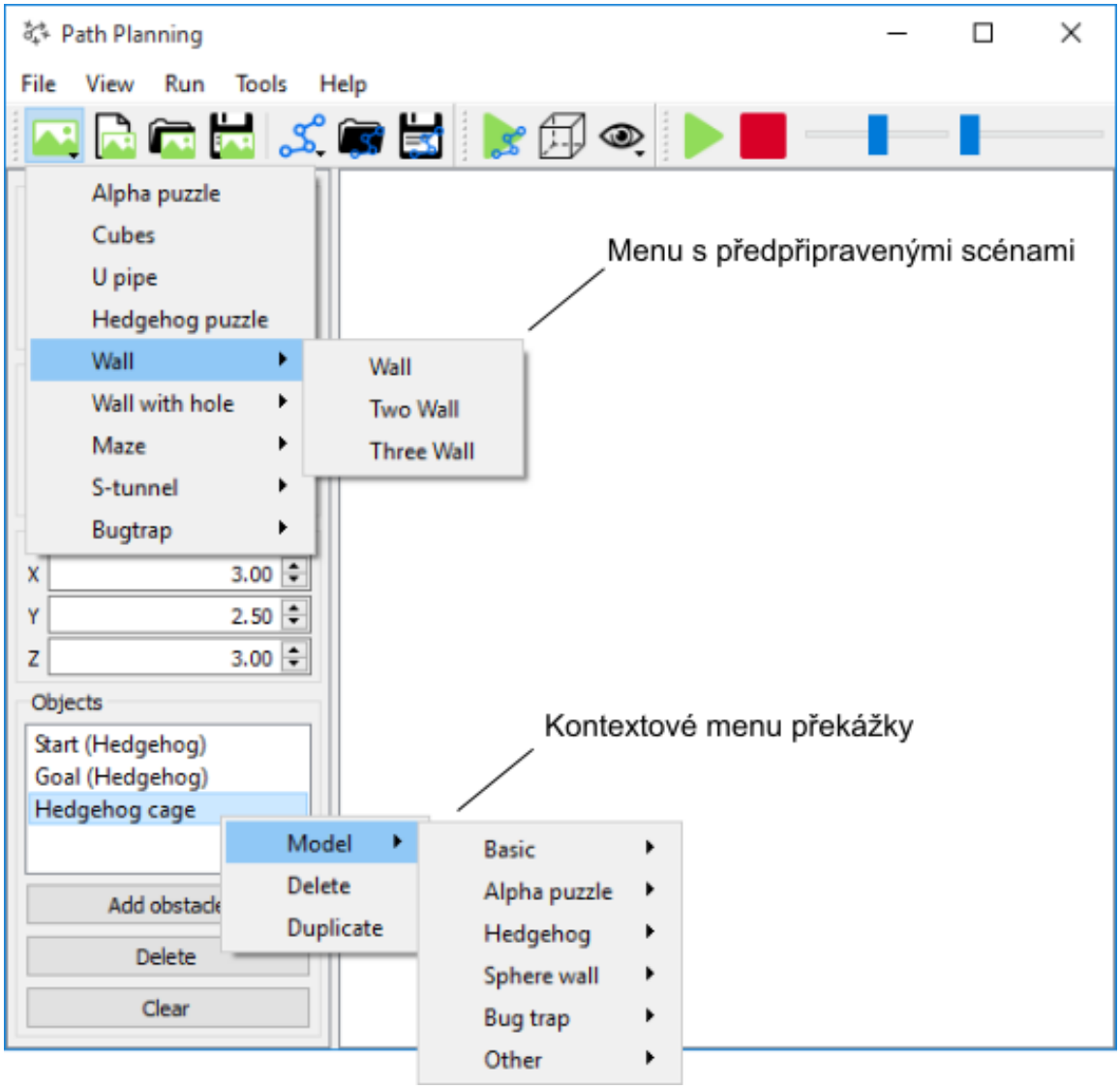
Scénu lze pomocí myši a klávesnice posouvat, rotovat a přibližovat. Stejně tak lze ovládat aktivní objekt. Konkrétní parametry objektů lze zároveň nastavit přes levý panel. Ovládání scény a objektů klávesnicí a myší je uvedeno v tabulce C.1.

Akce	**Funkce**
Levé tlačítko myši	posun scény
Pravé tlačítko myši	rotace scény
Kolečko myši	přiblížení scény
CTRL + levé tlačítko myši	posun aktivního objektu
CTRL + pravé tlačítko myši	rotace aktivního objektu podle os X a Y
CTRL + SHIFT + pravé tlačítko myši	rotace aktivního objektu podle os X a Z
CTRL + kolečko myši	změna velikosti aktivního objektu

Tabulka C.1: Způsob ovládání scény a objektů

C.2 Nastavení scény

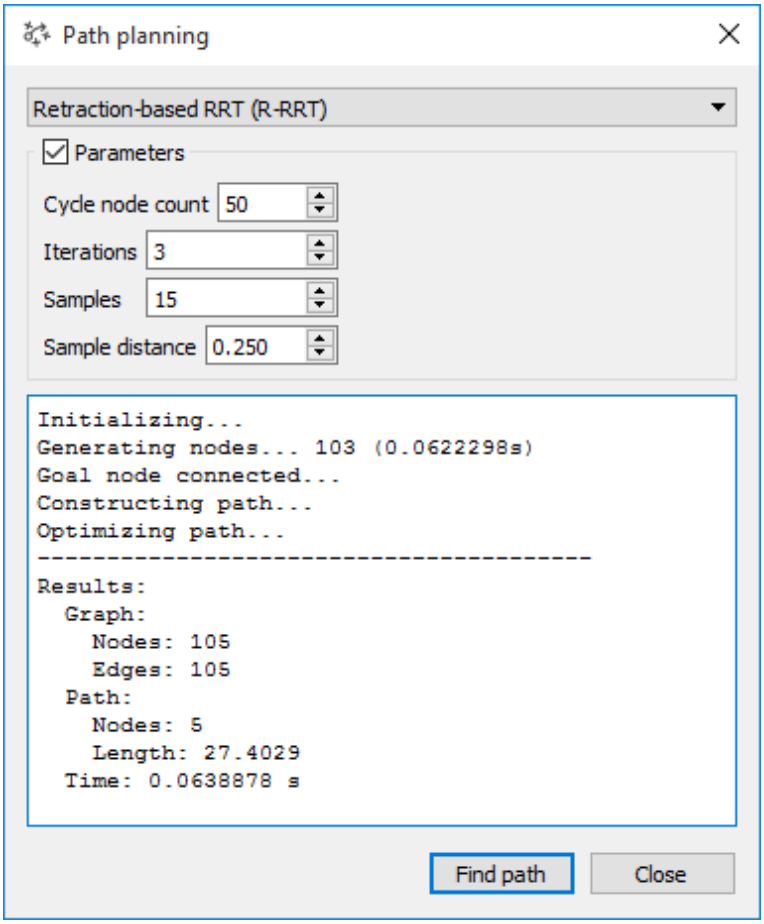
Prvním krokem je nastavení scény. Nastavení scény zahrnuje nastavení počáteční pozice, cílové pozice a rozmístění překážek. V levém panelu lze nastavit pozici, rotaci a velikost aktuálně označeného objektu. Při kliknutí pravým tlačítkem myši na položku v seznamu objektů se vyvolá kontextové menu, které umožňuje změnit model objektu. Pokud je označeným objektem překážka, lze ji odstranit nebo duplikovat. Objekty počáteční nebo cílové pozice nelze odstranit. Program také obsahuje několik předpřipravených scén, které lze kdykoli načíst.



Obrázek C.3: Okno aplikace s menu pro načtení předpřipravené scény a s kontextovým menu pro změnu modelu překážky

C.3 Plánování pohybu

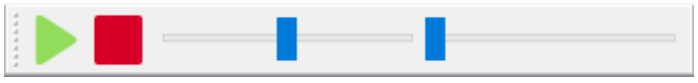
Samotné plánování pohybu je provedeno přes dialog plánování pohybu, který lze aktivovat tlačítkem v horním panelu, přes hlavní menu **Run** → **Find path** anebo klávesou **F5**. Dialog plánování pohybu obsahuje komponenty pro výběr algoritmu, nastavení parametrů vybraného algoritmu a textové pole s aktuálními informacemi o průběhu hledání cesty. V průběhu plánování pohybu se také ve scéně zobrazuje aktuální podoba grafu prostředí. Po dokončení hledání cesty jsou do textového pole vypsány informace o nalezené cestě a vytvořeném grafu prostředí.



Obrázek C.4: Dialog plánování pohybu

C.4 Animace cesty

Pokud byla cesta nalezena, lze animovat pohyb objektu po nalezené cestě. Animaci lze ovládat prvky v horní části okna. Animace lze spustit, pozastavit anebo úplně zastavit. Rychlost animace lze regulovat levým posuvníkem. Pravý posuvník slouží pro zobrazení a nastavení aktuální pozice objektu na cestě.



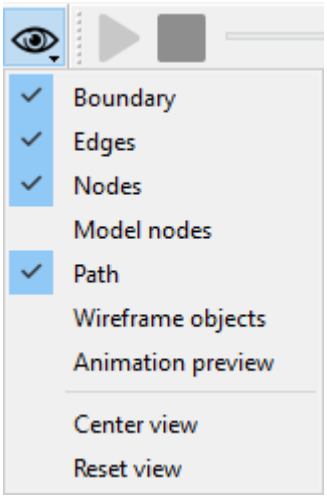
Obrázek C.5: Ovládací prvky animace pohybu objektu po nalezené cestě

C.5 Nastavení zobrazení

V horním panelu lze vyvolat menu zobrazení, které umožňuje měnit zobrazení některých objektů a obsahuje následující položky:

- **Boundary** – zobrazení hraniční obálky prostoru
- **Edges** – zobrazení hran grafu prostředí

- **Nodes** – zobrazení uzlů grafu prostředí
- **Model nodes** – jelikož samotné body neposkytují informaci o rotaci objektu, je možné uzly grafu nahradit zmenšeninami robota.
- **Path** – zobrazení nalezené cesty
- **Wireframe objects** – některé překážky jsou uzavřené objekty s vnitřním průchodem. Aby bylo možné sledovat cestu i uvnitř objektu, lze objekty zobrazit jako drátový model.
- **Path preview** – při nalezení cesty lze zobrazit statický náhled animace pohybu objektů. Krok náhledů lze změnit v nastavení programu. Pokud je krok náhledu animace stejný jako interpolační krok propojovací funkce Δ, pak náhled animace přímo odpovídá kolizním testům, které propojovací funkce provedla.



Obrázek C.6: Menu s nastavením zobrazení objektů

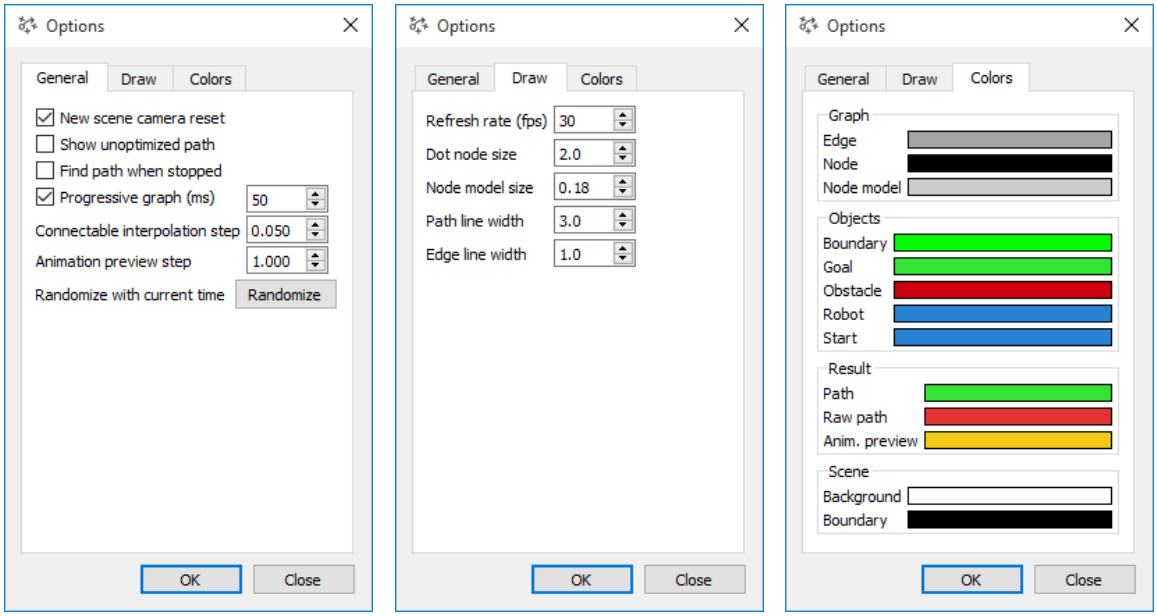
C.6 Nastavení programu

Program obsahuje dialog nastavení programu s různými volbami. Nastavení je rozděleno do tří sekcí:

1. **Obecná nastavení:**
   - Zobrazení neoptimalizované cesty. Ve výchozím stavu je zobrazena jen optimalizovaná cesta.
   - Sestavení cesty při zastavení algoritmu. V případě přerušení algoritmu se vytvoří cesta od počáteční konfigurace ke konfiguraci nejblíže cílové konfiguraci.
   - Zobrazení aktuální podoby grafu prostředí (průběžného grafu). V případě velkých grafů může být plánování pohybu značně zpomaleno. Uživatel tedy může zakázat zobrazení průběžného grafu prostředí. Pokud je průběžný graf povolen, je zde možnost nastavení intervalu odesílání grafu v milisekundách.

- Velikost interpolačního kroku propojovací funkce Δ
- Krok náhledu animace nalezené cesty
- Inicializace generátoru náhodných čísel aktuálním časem

2. **Nastavení zobrazení:**
   - Frekvence překreslení 3D scény ve snímcích za vteřinu
   - Velikost bodu zastupující uzel grafu
   - Velikost modelu zastupující uzel grafu. Hodnota značí relativní velikost vzhledem k modelu robota. Pro hodnotu např. 0,5 bude model uzlu poloviční velikosti oproti modelu robota.
   - Šířka křivky nalezené cesty
   - Šířka hrany grafu
3. **Nastavení barev:**
   - Graf prostředí: hrany, uzly a modely
   - Objekty: počáteční pozice, cílová pozice, animovaný objekt, překážky a zvýraznění aktivního objektu
   - Výsledek: nalezená cesta a neoptimalizovaná cesta
   - Scéna: pozadí a hranice scény



Obrázek C.7: Dialog nastavení programu