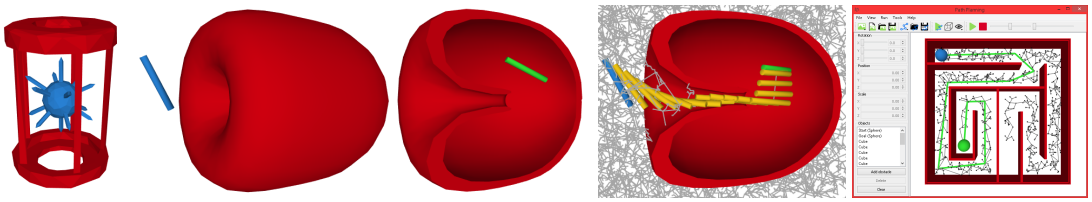
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2016
http://excel.fit.vutbr.cz

Plánování pohybu objektu v 3D prostoru

František Němec*



Abstrakt

Práce je zaměřena na plánování pohybu objektu ve 3D prostoru. Ve spojitém prostoru klasické algoritmy hledání cesty selhávají a je nutné najít jiný způsob řešení problému. Jedním takovým řešením je použití tzv. pravděpodobnostních algoritmů. Cílem této práce je vytvořit program pro řešení a demonstraci problému plánování pohybu ve 3D prostoru pomocí pravděpodobnostních algoritmů. Konkrétně pomocí algoritmů Probabilistic Roadmap (PRM), Expansive Spaces Trees (EST), Rapidly-exploring Random Trees (RRT), rozšířením Retraction-based RRT (R-RRT) a vlastním vylepšením Regulated R-RRT (RR-RRT). Program musí umožnit uživateli vytvořit scénu, ve které bude plánování pohybu probíhat, automaticky pak cestu vyhledá a nakonec ji bude vizualizovat.

Klíčová slova: Plánování pohybu — Pravděpodobnostní algoritmy — 3D prostor — PRM — EST — RRT — R-RRT — RR-RRT

Přiložené materiály: Demonstrační video — Aplikace pro Windows

*xnemec61@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Úvod

Práce se zabývá plánováním pohybu objektu ve 3D prostoru. Cílem plánování pohybu je vytvořit plán složený z jednoduchých úkonů jako posun, rotace nebo v reálném použití přímo ovládání jednotlivých motorů robota. Plánovací algoritmus má přitom informace jen o rozmístění překážek, počáteční a cílové pozici.

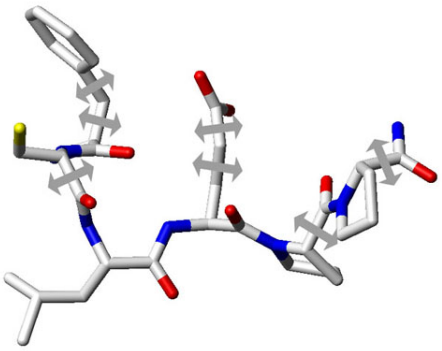
Plánování pohybu má využití v mnoha odvětvích. Typickým použitím je plánování pohybu robotického ramene. Úkolem ramene je přesun/manipulace objektu, přičemž se rameno musí vyhnout překážkám (ostatním konstrukcím montážní linky) a zároveň nesmí kolidovat samo se sebou.

Dalším použitím je postup sestavení výrobku z několika mechanických součástek. Způsob sestavení výrobku je významná část 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

Obrázek 1. Robotické rameno [1]

dají odhalit nedostatky v návrhu ještě před samotnou výrobou a tak předejít finančním a časovým ztrátám. Nezanedbatelnou aplikací plánovacích algoritmů je biologie, zejména pak skládání a dokování proteinů

[2]. 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. *Stick* reprezentace struktury proteinu spolu se šipkami, které značí možné rotace jednotlivých částí [3].

V případě hledání cesty ve spojitém prostoru nelze použít naivní způsoby jako průchod všemi stavy systému. To z důvodu vysoké časové složitosti a hlavně z důvodu, že takový stavový prostor není konečný. Základním přístupem může být rozdělení prostoru na pravidelnou mřížku. Nicméně takový způsob nezachytí povahu prostoru, protože stejným způsobem vzorkuje otevřený prostor i úzké průchody, které vyžadují větší pozornost. Časová složitost takových algoritmů je také obecně příliš vysoká.

Jedním způsobem řešení jsou *pravděpodobnostní algoritmy* [4]. Obecně pravděpodobnostní algoritmy pracují na principu náhodného vzorkování prostoru, čímž vytvoří graf prostředí, nad kterým lze hledat cestu pomocí klasických algoritmů hledání cesty v grafu. Budou-li vzorky vhodně rozmístěné, což je zatíženo náhodou a povahou konkrétního algoritmu, lze nalézt cestu mezi počáteční a cílovou pozicí mnohem rychleji, než při procházení velkého množství pravidelně navzorkovaných bodů. 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].

Cílem práce je vytvořit program pro řešení a demonstraci problému plánování pohybu ve 3D prostoru. Program musí umožnit uživateli vytvořit scénu, ve které bude plánování pohybu probíhat, automaticky pak cestu vyhledá a nakonec ji bude vizualizovat. Plánování pohybu je řešeno pomocí již zmíněných pravděpodobnostních algoritmů, konkrétně Probabilistic Roadmap (PRM) [5], Expansive Spaces Trees (EST), Rapidly-exploring Random Trees (RRT) [6], rozšířením Retraction-based RRT (RRRT) [7] a vlastním vylepšením Regulated R-RRT (RR-RRT)

První kapitola obsahuje popis pravděpodobnostních algoritmů. 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.

2. Pravděpodobnostní algoritmy

Pravděpodobnostní algoritmy patří do kategorie algoritmů pro plánování pohybu, které pracují na principu náhodného vzorkování prostoru. Vzorkování je přitom řízeno, např. generováním více vzorků ve členitější oblasti prostoru. Vzorky jsou následně propojeny, čímž je vytvořen graf, který zachycuje prostředí v diskrétní formě. Nad tímto grafem lze lehce hledat konkrétní cestu, neboli naplánovat pohyb tělesa z počáteční k cílové pozici.

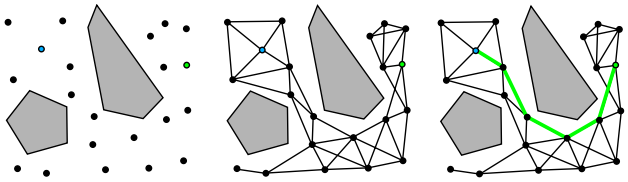
2.1 Probabilistic Roadmap

Jedním ze základních pravděpodobnostních algoritmů je *Probabilistic Roadmap* (PRM) [5]. Základní myšlenka je popsána v algoritmu 1.

V první fázi je generován určitý počet volných konfigurací (umístění robota tak, že nekoliduje s žádnou překážkou), které jsou uloženy do grafu. Volné konfigurace jsou následně propojeny hranami. Základním způsobem propojení je spojení dvou konfigurací úsečkou a kontrola, zda žádná konfigurace na úsečce nekoliduje s překážkami. Na konci každého cyklu je proveden test propojitelnosti počáteční a cílové konfigurace. Pokud lze konfigurace propojit, zbývá jen najít konkrétní cestu. Pro hledání cesty je možné použít známé algoritmy hledání cesty v grafu jako např. Dijkstrův nebo A* algoritmus.

Algoritmus 1 Probabilistic Roadmap

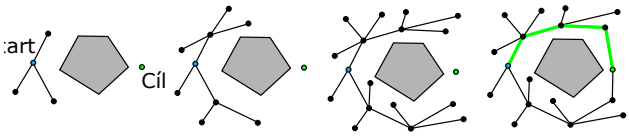
Vstup:
n : počet uzlů jednoho cyklu
q_{init} : počáteční konfigurace
q_{goal} : cílová konfigurace
Výstup:
Cesta z q_{init} do q_{goal}
1. **repeat**
2. Generování n volných konfigurací (uzlů)
3. Propojení vygenerovaných uzlů
4. **until** existuje sled z q_{init} do q_{goal}
5. Sestroj posloupnost uzlů z počátečního k cílovému uzlu jako výslednou cestu



Obrázek 3. Jednotlivé fáze PRM algoritmu: generování volných konfigurací, propojení konfigurací a hledaní konkrétní cesty.

2.2 Expansive Spaces Trees

Druhým základním algoritmem je *Expansive Spaces Trees* (EST), který se snaží co nejrychleji pokrýt prostor mezi počáteční a cílovou konfigurací. Postup EST je v algoritmu 2. Na rozdíl od PRM algoritmu, který buduje graf, EST buduje strom, který je zakořeněn v počáteční konfiguraci. Algoritmus podle pravděpodobnostní funkce π_T vybírá již vytvořené konfigurace q_{rand} a v jejich okolí generuje nové konfigurace q_{new}. Pokud lze konfigurace q_{rand} a q_{new} propojit, je konfigurace q_{new} přidána do stromu. Algoritmus je ukončen v případě, že lze ke stromu připojit cílovou konfiguraci. Nakonec zbývá najít konkrétní posloupnost uzlů od počátečního k cílovému uzlu.



Obrázek 4. EST algoritmus. Postupné rozšiřování stromu generováním nových konfigurací.

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. To znamená vybírat nejvíce osamocené uzly. Jedním způsobem je definovat míru osamocenosti jako počet sousedních uzlů v nějakém okolí. Nicméně to vyžaduje výpočet vzdáleností mezi všemi uzly, což je výpočetně náročné (v následujícím textu je tato varianta označena jako EST-Dist). Méně přesným, ale rychlejším způsobem je rozdělit prostor do mřížky a vybírat uzly z buňky s nejmenším počtem uzlů (varianta EST-Grid).

2.3 Rapidly-exploring Random Trees

Rapidly-exploring Random Trees (RRT) [6] algoritmus je velice podobný EST algoritmu. Základní RRT algoritmus pracuje následovně. Na počátku je vytvořen strom T s kořenovým uzlem v počáteční konfiguraci. Algoritmus iterativně přidává uzly do stromu. V každé iteraci je vygenerována náhodná konfigurace q_{rand} a RRT algoritmus se ji pokusí propojit úsečkou s nejbližší konfigurací q_{near}. Pokud lze q_{rand} a q_{near} propojit bezkolizní cestou, je strom T rozšířen o konfiguraci

Algoritmus 2 Expansive Spaces Trees

Vstup:
$T = (V, E)$: Strom s počáteční konfigurací jako kořenový uzel
q_{goal} : cílová konfigurace
n : počet uzlů jednoho cyklu

Výstup:
Cesta z počáteční do cílové konfigurace

1. **repeat**
2. **for** $i = 1$ do n **do**
3. $q_{rand} \leftarrow$ náhodně zvolená konfigurace z V s pravděpodobností $\pi_T(q_{rand})$
4. $q_{new} \leftarrow$ náhodná bez-kolizní konfigurace v okolí konfigurace q_{rand}
5. **if** q_{rand} a q_{new} lze propojit **then**
6. Ulož uzel q_{new} spolu s hranou (q_{rand}, q_{new}) do stromu T
7. **end if**
8. **end for**
9. **until** lze ke stromu připojit q_{goal}
10. Sestroj posloupnost uzlů z počátečního k cílovému uzlu jako výslednou cestu

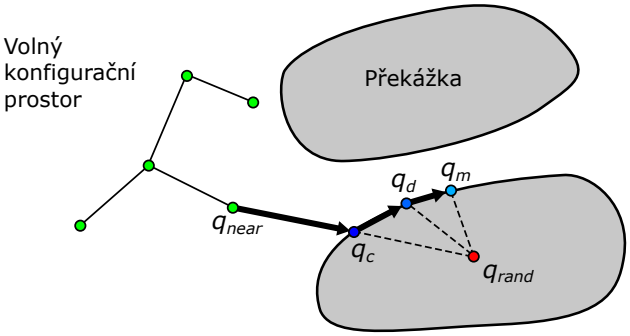
q_{rand}. V opačném případě je vypočtena první kontaktní konfigurace q_c, která leží na úsečce z q_{near} do q_{rand} a ta je uložena do stromu.

2.4 Retraction-based RRT

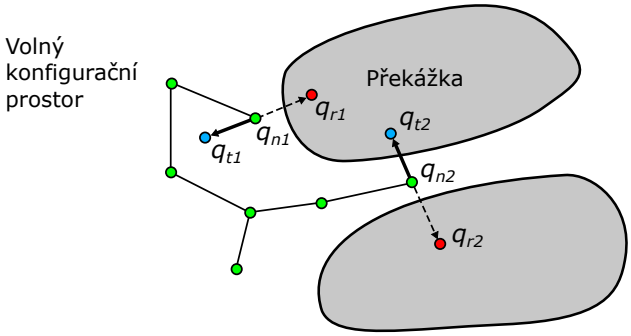
Jedním z hlavních problémů pravděpodobnostních algoritmů jsou úzké průchody. Pokud prostor obsahuje úzký průchod a robot ním musí projít, je malá šance, že nová konfigurace bude vygenerována právě uvnitř průchodu. Tento problém se snaží řešit *Retraction-based RRT* (R-RRT) [7, 8] generováním více konfigurací na okraji překážek. R-RRT pracuje stejně jako RRT, ale v případě, že dvě konfigurace nelze přímo propojit, vykoná se tzv. *retrakční krok*. Retrakční krok pracuje následovně:

1. Nalezení konfigurace q_c, která leží mezi uzly q_{near} a q_{rand} a na hranici překážky.
2. Prohledávání lokálního prostoru za účelem nalezení nové nekolidující konfigurace q_d, která se nachází blíže k q_{rand}.
3. Přiřazení $q_n \leftarrow q_d$ a skok na krok 2.

Retrakční krok je ukončen v případě, že nelze nalézt žádnou bližší konfiguraci, nebo bylo dosaženo předem definovaného počtu pokusů. Vizuální reprezentace retrakčního kroku je na obrázku 5.



Obrázek 5. Retrakční krok R-RRT algoritmu. Generování sekvence nekolidujících konfigurací q_c, q_d, q_m, které se postupně blíží k q_{rand}.



Obrázek 6. 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 pozitivní výsledek testu a spuštění retrakčního kroku.

2.5 Regulated R-RRT

Na základě experimentů jsem navrhl vylepšení R-RRT algoritmu, které jsem pojmenoval *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.

Na obrázku 6 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. Návrh programu

Okno programu je rozděleno na tři části. Hlavní částí je pohled na 3D scénu a boční panel pro nastavení pozice, rotace a měřítka objektu spolu se seznamem překážek. Třetí částí je ovládací lišta pro zobrazení okna pro plánování pohybu, nastavení hranic prostoru a ovládání animace výsledné cesty. Program se může nacházet v jednom ze čtyř stavů:

- **Nastavení scény:** v tomto stavu je možné nastavit počáteční a cílovou pozici hledané cesty, rozmístění překážek a hranice prostoru. Model překážek a robota lze načíst ze souboru v Wavefront OBJ formátu. Scénu lze uložit a načíst při budoucím spuštění aplikace. Formát souboru se scénou je jednoduchý XML. Program přímo obsahuje několik předpřipravených scén pro rychlou demonstraci programu.
- **Výběr algoritmu:** uživatel má na výběr z několika algoritmů popsaných v teoretické části. Spolu s typem algoritmu může uživatel nastavit některé parametry, jako například u PRM algoritmu počet generovaných uzlů v jednom cyklu.
- **Plánování pohybu (hledání cesty):** provádění vybraného algoritmu. Ve scéně se zároveň zobrazuje aktuální graf/strom prostředí, jak jej daný algoritmus buduje. V této fázi lze sledovat na jakém principu algoritmy pracují. Ukázka řešení úlohy je na obrázku 9. Stejně jako vytvořenou scénu i nalezenou cestu lze uložit a později načíst, takže není nutné ji při novém spuštění znovu plánovat.
- **Animace cesty:** po úspěšném nalezení cesty ji lze vizualizovat. Vizualizace je provedena animací pohybu objektu po nalezené trajektorii. Rychlost animace lze regulovat, nebo lze přímo nastavit na jakém bodě cesty se má objekt zobrazit.

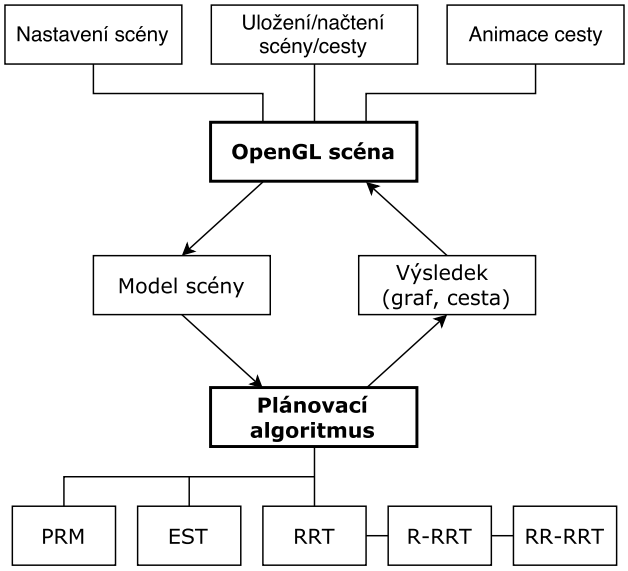
4. Implementace programu

Program je implementován v jazyce C++ a pro uživatelské rozhraní je použit Qt framework[1]. Pro vykreslení 3D scény bylo zvoleno OpenGL, přičemž Qt framework poskytuje několik modulů pro práci s OpenGL.

[1] Qt framework: https://www.qt.io/

Detekce kolizí je realizována volně dostupnou knihovnou ColDet[2]. Implementace je rozdělena na dvě hlavní části (obrázek 7):

- **Vizualizace:** zejména třídy uživatelského rozhraní. Qt třídy pro hlavní okno programu a její komponenty jako OpenGL komponenta pro vykreslení 3D scény nebo ovládací panely. Vykreslení scény zastřešuje vykreslení modelu, posun a rotaci objektů na základě uživatelského vstupu. Dalšími třídami uživatelského rozhraní je ovládání animace cesty (rychlosti, poloha) 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.

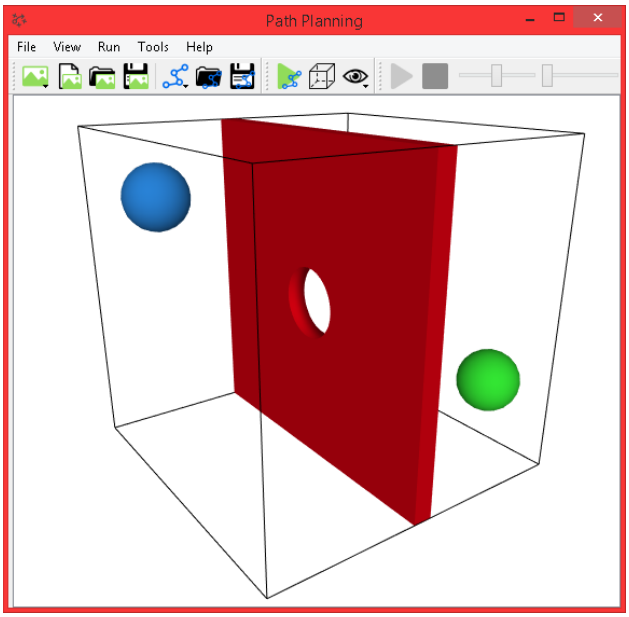


Obrázek 7. Hlavní části implementace programu

5. Experimenty

Cílem experimentů je ověřit funkčnost programu a výkon jednotlivých algoritmů. Pro každý experiment jsem provedl 30 běhů. Ze všech běhů jsem vypočítal průměr a výsledné časy jsou uvedeny v jednotlivých tabulkách. Experimenty proběhly na procesoru Intel Core i5-4460 3,2GHz. Experimenty jsou rozděleny do dvou sad.

První sada experimentů je provedena na jednoduché scéně se stěnou/zdí s otvorem (obrázek 10). 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 1.



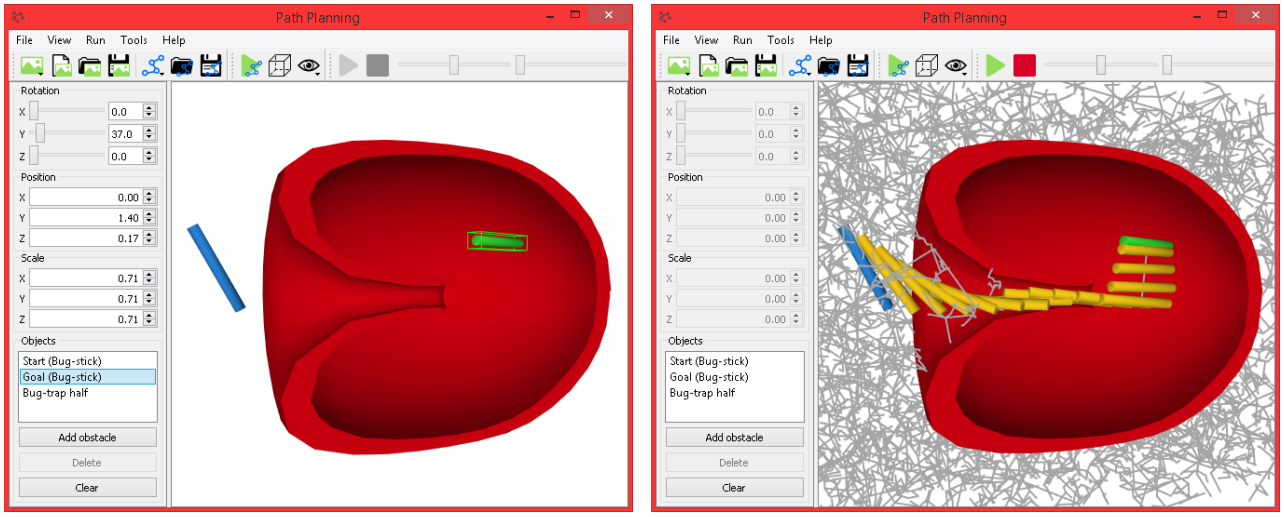
Obrázek 10. Stěna s otvorem

Tabulka 1. Průměrné časy algoritmů na sadě experimentů se stěnou/zdí s otvorem. Pomlčky značí, že algoritmus nebyl schopen úlohu vyřešit do jedné hodiny.

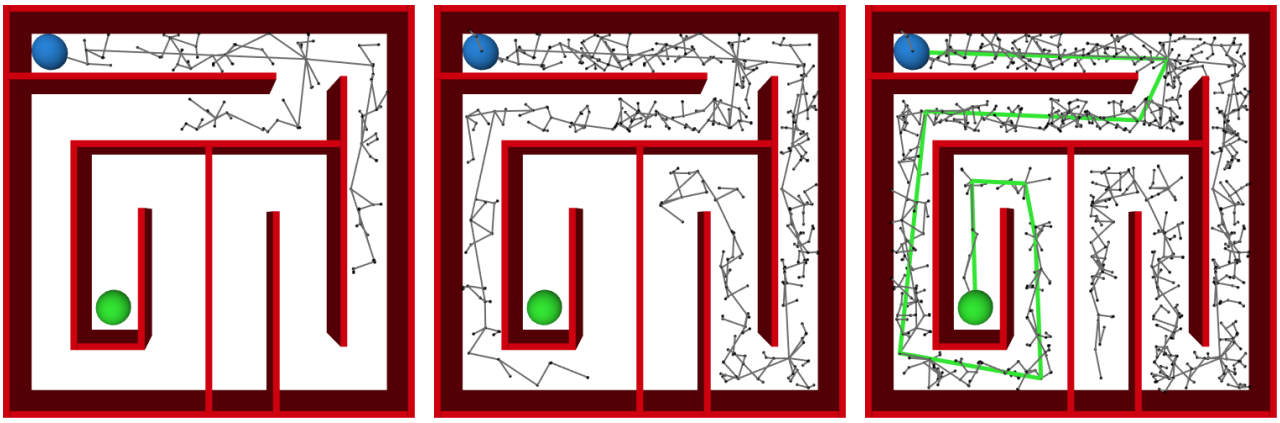
	Zeď 1,25	Zeď 1,1	Zeď 1,05	Zeď 1,01
PRM	25,5s	?	–	–
EST-Dist	33,9s	342,1s	–	–
EST-Grid	1,5s	6,2s	38,5s	–
RRT	0,3s	1,7s	3,8s	15,9s
R-RRT	13,3s	21,5s	28,8s	40,4s
RR-RRT	0,5s	5,7s	8,8s	19,4s

PRM spolehlivě vyřešil jen zeď 1,25 a přitom skoro nejpomaleji. PRM byl schopný vyřešit zeď 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

[2] Amir Geva: ColDet 3D Collision Detection



Obrázek 8. Ukázka uživatelského rozhraní s úlohou past na brouka. Na levém obrázku je zobrazeno nastavení scény. Na pravém obrázku je nalezená cesta spolu s hranami grafu prostředí.

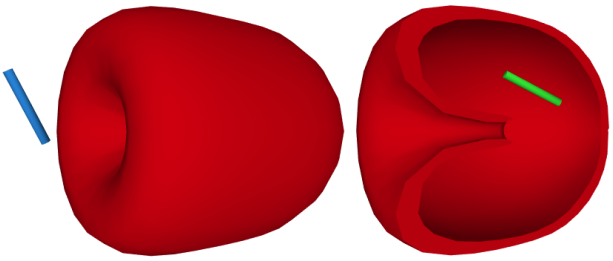


Obrázek 9. Průběh postupného budování stromu prostředí RRT algoritmu na úloze bludiště. Stěny bludiště jsou označeny červeně, počáteční pozice je vyznačena modře a cílová pozice zeleně. Šedé křivky představují hrany stromu prostředí a zelená křivka na posledním obrázku je nalezená cesta.

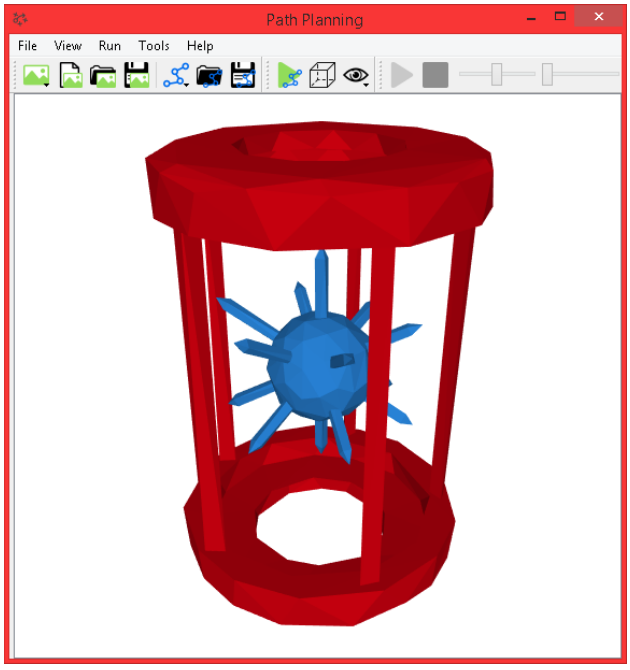
časově náročná operace. Druhým, možná i větším zpomalením, je testování zda je počáteční a cílová konfigurace propojena (na grafu prostředí s desítkami tisíci uzly). EST a RRT algoritmy tímto netrpí, protože ty nalezení cesty detekují prostým napojení cílové konfigurace ke stromu.

Časová složitost EST-Dist je podobná jako PRM, takže jsou podobné i výsledky. EST-Grid odstraňuje špatnou časovou složitost EST-Dist, a tak dokáže vyřešit i scény s menším otvorem. Nicméně zeď 1,01 nebyl schopen vyřešit do jedné hodiny. 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. RR-RRT však výrazně R-RRT vylepšuje, ne však na úroveň RRT, ale i tak je vylepšení znatelné. Vylepšení bude ještě znatelnější na druhé sadě experimentů.

Druhá sada experimentů obsahuje několik složitějších scén. Prvním experimentem je jednoduché bludiště (obrázek 9). Druhý experiment je proveden na často se vyskytujícím problému „past na brouka“ (obrázky 8 a 11). 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 12). Cílem je vytáhnout ostnatou kouli (ježka) mimo klec. Ježek v kleci je relativně náročný na vyřešení, jelikož vyžaduje manipulaci objektu v malém prostoru.



Obrázek 11. Past na brouka. Past je označena červenou barvou (vpravo je průřez pasti), počáteční pozice brouka modře a cílová pozice zeleně.



Obrázek 12. Model hlavolamu ježka v kleci

Tabulka 2. Průměrné časy algoritmů na druhé sadě experimentů. Pomlčky značí, že algoritmus nebyl schopen úlohu vyřešit do jedné hodiny.

	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

PRM algoritmus vyřešil bludiště nejrychleji, ale to 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. 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. EST-Grid algoritmus dokázal vyřešit ježka v kleci, ale to jen díky generování velkému množství uzlů (zhruba 100krát více uzlů než R-RRT).

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 tedy 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. Závěr

Cílem této práce bylo vytvořit program pro řešení a demonstraci problému plánování pohybu ve 3D prostoru pomocí pravděpodobnostních algoritmů. Program musel umožnit uživateli vytvořit scénu, ve které bude plánování pohybu probíhat, automaticky pak cestu vyhledá a nakonec ji vizualizuje.

Pro návrh a implementaci uživatelského rozhraní bylo nutné nastudovat práci s 3D modely a OpenGL. Dále byly nastudovány některé pravděpodobnostní algoritmy, mezi které patří PRM, EST, RRT, R-RRT a vlastní rozšíření RR-RRT. Nakonec bylo provedeno několik experimentů s cílem ověřit funkčnost programu a výkon algoritmů. Experimenty prokázaly funkčnost programu a poukázaly na nedostatky některých algoritmů. Výsledný program lze použít pro demonstraci pravděpodobnostních algoritmů plánování pohybu nebo řešení některých plánovacích úloh.

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

[2] J. Cortes, L. Jaillet, and T. Simeon. Molecular disassembly with rrt-like algorithms. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3301–3306, 2007.

[3] Lydia E. Kavraki. Protein-ligand docking, including flexible receptor-flexible ligand docking [online]. [cit. 20.4.2016]. Dostupné z: http://cnx.org/content/m11456/.

[4] Howie M. Choset. *Principles of robot motion*. MIT Press, Cambridge, Mass., 2005.

[5] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[6] J.J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. *IEEE International Conference on Robotics and Automation*, 2:995–1001, 2000.

[7] Liangjun Zhang and D. Manocha. An efficient retraction-based rrt planner. *IEEE International Conference on Robotics and Automation*, pages 3743–3750, 2008.

[8] Liangjun Zhang, Xin Huang, Young J. Kim, and Dinesh Manocha. D-plan: Efficient collision-free path computation for part removal and disassembly. *Journal of Computer-Aided Design and Applications*, 5(5):774–786, 2008.