VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

PLÁNOVÁNÍ CESTY ROBOTU (RRT)

ROBOT PATH PLANNING (RRT)

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **LUKÁŠ KNISPEL**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADOMIL MATOUŠEK, Ph.D.**
SUPERVISOR

BRNO 2010

ZADÁNÍ ZÁVĚREČNÉ PRÁCE

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2009/2010

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Lukáš Knispel

který/která studuje v **bakalářském studijním programu**

obor: **Aplikovaná informatika a řízení (3902R001)**

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

Plánování cesty robotu (RRT)

v anglickém jazyce:

Robot Path Planning (RRT)

Stručná charakteristika problematiky úkolu:

Daná BP se bude zabývat návrhem a tvorbou linux aplikace implementující pokročilý RRT algoritmus pro plánování cesty všesměrového mobilního robotu. Daná práce bude zahrnuta jako součást řešení VZ Inteligentní systémy v automatizaci.

Cíle bakalářské práce:

- Knihovna pro implementaci RRT algoritmu.
- Vhodné GUI prostředí, včetně editoru „světa“.
- Parametrizace řešiče RRT algoritmu.
- Zahrnutí opravných algoritmů pro zefektivnění navržené trajektorie.
- Dávkový režim za účelem testování efektivnosti nastavení řešiče pro dané úlohy.
- Stručná e-dokumentace (může být součástí aplikace).

Seznam odborné literatury:
Lavalle, S.M. (1998). "Rapidly-exploring random trees: A new tool for path planning". Computer Science Dept, Iowa State University, Tech. Rep. TR: 98–11. http://citeseer.ist.psu.edu/311812.html. Retrieved 2008-06-30

Vedoucí bakalářské práce: Ing. Radomil Matoušek, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2009/2010.
V Brně, dne

L.S.

Ing. Jan Roupec, Ph.D. Ředitel ústavu	prof. RNDr. Miroslav Doupovec, CSc. Děkan fakulty

LICENČNÍ SMLOUVA

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Lukáš Knispel

Bytem:

Narozen (datum a místo):

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta strojního inženýrství

se sídlem Technická 2896/2, 616 69 Brno

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....

(dále jen „nabyvatel“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - □ disertační práce
   - □ diplomová práce
   - □ bakalářská práce
   - □ jiná práce, jejíž druh je specifikován jako .....

   (dále jen VŠKP nebo dílo)

Název VŠKP: _____

Vedoucí/ školitel VŠKP: _____

Ústav: _____

Datum obhajoby VŠKP: _____

VŠKP odevzdal autor nabyvateli v*:

- □ tištěné formě – počet exemplářů ……..
- □ elektronické formě – počet exemplářů ……..

* hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - □ ihned po uzavření této smlouvy
   - □ 1 rok po uzavření této smlouvy
   - □ 3 roky po uzavření této smlouvy
   - □ 5 let po uzavření této smlouvy
   - □ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: …….

…….. ……
Nabyvatel Autor

ABSTRAKT

Tato bakalářská práce se zabývá plánováním cesty všesměrového mobilního robotu pomocí algoritmu RRT (Rapidly-exploring Random Tree – Rychle rostoucí náhodný strom). V teoretické části dále popisuje základní algoritmy plánování cesty a prezentuje bližší pohled na RRT a jeho potenciál. Praktická část práce řeší návrh a tvorbu C++ linux aplikace v prostředí Ubuntu 9.10 za použití aplikačního frameworku Qt 4.6, která implementuje pokročilý RRT algoritmus s parametrizovatelným řešičem a dávkovým režimem za účelem testování efektivnosti nastavení řešiče pro dané úlohy.

ABSTRACT

This thesis deals with path plannig of omnidirectional mobile robot using the RRT algorithm (Rapidly-exploring Random Tree). Theoretical part also describes basic algorithms of path planning and presents closer view on RRT and its potential. Practical part deals with designing and creation of C++ linux application in Ubuntu 9.10 environment with Qt 4.6 application framework, which implements advanced RRT algorithm with user-programmable solver and batch mode in order to test efectivity of solver on given tasks.

KLÍČOVÁ SLOVA

Plánování cesty, mobilní robot, Rychle rostoucí náhodný strom, Qt, C++.

KEYWORDS

Path planning, mobile robot, Rapidly-exploring Random Tree, Qt, C++.

BIBLIOGRAFICKÁ CITACE PRÁCE

KNISPEL, L. *Plánování cesty robotu (RRT)*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2010. 47 s. Vedoucí bakalářské práce Ing. Radomil Matoušek, Ph.D.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že předložená bakalářská práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 28. května 2010 ……...

Lukáš Knispel

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat vedoucímu mé bakalářské práce, Ing. Radomilu Matouškovi, Ph.D., za odborné vedení, cenné rady, připomínky a čas, který mi věnoval. Děkuji také svým rodičům a přítelkyni za pomoc a podporu, bez které by následující stránky vznikaly jen obtížně.

Obsah:

1 ÚVOD

V posledních letech dochází s prudkým rozvojem informačních technologií i k rozvoji robotiky a k velkému pokroku v oblasti plánování cesty mobilních robotů. V současné době dokáží roboty řešit problémy v mnoha složitých prostředích, vlastnost robotů pracovat v cizím nebo agresivním prostředí otevírá široké možnosti aplikace robotů a jejich náhrady za lidskou práci. Plánování cesty se tak uplatňuje v mnoha odvětvích, jako např. průmyslová robotika, automatizace, robotická chirurgie, automatický průzkum prostoru. Dalším významným uplatněním je oblast počítačové grafiky a animace.

V oblasti umělé inteligence se pod pojmem *plánování* obecně rozumí vyhledávání sekvence operací, která transformuje počáteční pozici stavu (polohy) objektu v prostředí do požadovaného cílového stavu (polohy). *Plánování cesty* robotu se zabývá generováním vhodné trajektorie tak, aby se robot při svém pohybu prostorem po této trajektorii nesetkal s překážkami. Dalšími souvisejícími úlohami s plánováním je zahrnutí opravných algoritmů pro zefektivnění navržené trajektorie nebo zohlednění kinematiky robotu.

Cílem této bakalářské práce je přiblížení problematiky hledání cesty pro všesměrového mobilního robotu pomocí algoritmu Rychle rostoucího náhodného stromu. Dále pak tvorba linux aplikace v jazyce C++ za použití aplikačního frameworku Qt 4.6 v prostředí Ubuntu 9.10. Aplikace umožní bližší poznávání vlastností RRT algoritmu pomocí uživatelem parametrizovatelného řešiče a dávkového módu pro opakovaná měření a další zpracování výstupních statistických dat. Součástí aplikace bude vhodné grafické uživatelské rozhraní včetně pohodlného „editoru světa“ s možností ukládání a načítání map, volitelným měřítkem vzhledem ke skutečnosti, exportem obrázků a exportem výsledků dávkového módu v podobě grafu a souboru s tabulkou hodnot a významnými statistickými hodnotami.

V práci jsou vysvětleny základní pojmy nutné k pochopení zadaného tématu, přiblížena problematika plánování cesty. Následuje popis a rozdělení základních algoritmů plánování cest. Zde jsou představeny z oblasti informovaného hledání cesty metody rastrové a dekompoziční, metody map cest nebo pravděpodobnostní. Z oblasti neinformovaného hledání cesty pak práce popisuje Bug algoritmy. Na dalších stranách je čtenář podrobně informován o samotném algoritmu RRT, o variantách jeho provedení (jedno-, dvou- a případně i více stromové struktuře), možnostech a příkladech využití. Následuje popis funkcí a ovládání linux aplikace RRT Explorer, popis implementace vytvořeného software a v neposlední řadě také závěry měření vybraných nastavení řešiče.

2 ZÁKLADNÍ POJMY

Tato kapitola definuje, případně blíže představuje pro čtenáře pojmy: *robot*, *překážka*, *plánování cesty* a *reprezentace prostředí*.

2.1 Robot

„Robot je stroj pracující s určitou mírou samostatnosti, vykonávající určené úkoly, a to předepsaným způsobem a při různých mírách potřeby interakce s okolním světem a se zadavatelem. Robot je schopen své okolí vnímat pomocí senzorů, zasahovat do něj, případně si o něm vytvářet vlastní představu, model.“ [1]

Roboty dle jejich schopnosti přemisťovat dělíme na:

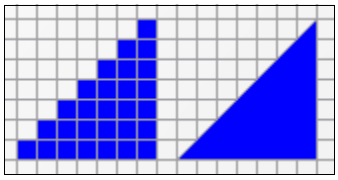
- *stacionární roboty* – nemohou se pohybovat z místa na místo. Většinou průmyslové manipulátory (ramena, která zajišťují polohování uchopeného předmětu ve 3D prostoru).

- *mobilní roboty* – mohou se přemisťovat v prostoru. Dále je můžeme (z hlediska autonomie) rozdělit na dvě skupiny:
  - *řízené dálkově* – pohybují se a pracují na základě instrukcí operátora, mnohdy nejsou vybaveny prakticky žádnou inteligencí.
  - *autonomní* – pohybují se na základě autonomních algoritmů plánování cesty a k práci často využívají prvky umělé inteligence. Většinu úkolů zvládají bez pomoci člověka.

Tato práce se bude zabývat aplikací pro mobilní autonomní roboty, konkrétně plánováním cesty ve 2D prostředí. Stacionární roboty obvykle vyžadují plánování ve 3D (polohování ramene).

2.2 Překážka

Překážka představuje objekt reálného prostředí, který omezuje směr popř. způsob pohybu robotu. Zobrazení překážky je závislé na modelu prostředí. Ve spojitém modelu jsou polygoniální překážky reprezentovány vrcholy a hranami, nepolygoniální překážky mohou být popsány pomocí křivek. V diskrétním modelu je překážka reprezentována jednou nebo více elementárními buňkami modelu.



Obr. 1 Reprezentace překážky v diskrétním modelu (vlevo) a spojitém modelu (vpravo).

2.3 Plánování cesty

Klíčovou úlohou, kterou řešíme v souvislosti s autonomními roboty, je jejich pohyb v prostoru. Cílem je schopnost robotu naplánovat trasu k zadanému cíli – z místa A na místo B. V závislosti na prostředí obklopující robot můžeme dle [2] metody plánování cesty rozdělit na:

- Plánování se statickými překážkami ve známém prostředí.
- Plánování se statickými překážkami v neznámém nebo částečně známém prostředí.
- Plánování s dynamickými překážkami ve známém prostředí.
- Plánování s dynamickými překážkami v neznámém nebo částečně známém prostředí.

V reálných příkladech nás dle [3] ve známém prostředí se statickými překážkami (častý případ pro autonomní mobilní roboty a zaměření této práce) zajímají řešení následujících problémů:

- Jak se dostat z bodu A do bodu B.
- Jak se vyhnout překážkám na cestě.
- Jak najít pokud možno nejkratší cestu k cíli.
- Jak najít příslušnou cestu rychle.

Plánování cesty obvykle začíná tak, že robot obdrží mapu prostoru (rozmístění překážek) spolu s polohou startovního a cílového stavu. Robot pohybující se po této trase, se do cíle může dostat mnoha různými způsoby. Schopnost plánování tak představuje výpočet více či méně optimální cesty k cíli. Optimálnost plánování nejčastěji posuzujeme podle rychlosti výpočtu nebo délky trajektorie.

2.4 Reprezentace prostředí

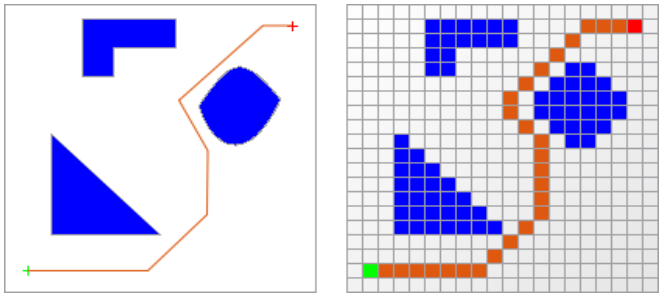
Pro dosažení určité přesnosti by měl být *model prostředí* pokud možno co nejvíce podobný reálnému prostředí. Přesnosti modelu je ovšem úměrná výpočetní náročnost. V praxi je třeba mezi požadavky přesnosti modelu prostředí a výpočtové náročnosti problému volit vhodný kompromis.

Z hlediska dimenzí může být model prostředí dvourozměrný nebo třírozměrný, v závislosti na pohybových schopnostech robotu.

Z hlediska reprezentace můžeme prostředí rozdělit na:

- *spojité prostředí* – modelem je spojitý prostor, který je velmi podobný reálnému prostředí. Umožňuje přesné zadávání překážek bez zjednodušení jejich tvaru. Směr pohybu robotu není nijak omezen. Implementace kvalitního editoru modelu takového prostředí není jednoduchá.

- *diskrétní prostředí* – modelem může být graf nebo prostor složený z jednotlivých buněk. Počet buněk se odvíjí od požadavku na přesnost modelu a velikosti robotu. Minimální velikost buňky odpovídá maximálnímu rozměru robotu. Dostatečně velký počet buněk zajistí vyšší přesnost a věrnější zobrazení členitosti překážek, ale přináší vyšší výpočetní náročnost. Implementace editoru modelu prostředí je jednodušší, spočívá ve čtverečkované síti a multiselekci jednotlivých políček.



Obr. 2 Spojité prostředí s nepolygoniální překážkou (vlevo) a jemu odpovídající diskrétní aproximované prostředí (vpravo) .

Aplikace vyvíjená v rámci této práce používá kombinaci obou přístupů – model prostředí je zadáván pomocí diskrétní sítě, ale plánovací algoritmus se dále pohybuje jako ve spojitém prostředí.

3 ALGORITMY PLÁNOVÁNÍ CESTY

Pro plánování cesty mobilního robotu existuje množství metod. Dle [4] se tyto metody většinou sestávají ze dvou dílčích částí. První část je takzvané *předzpracování (preprocessing)*, kdy se popíše prostor pomocí grafu nebo funkce. Po ní následuje *dotazovací část (query phase)*, kdy již probíhá samotné hledání cesty mezi dvěma body.

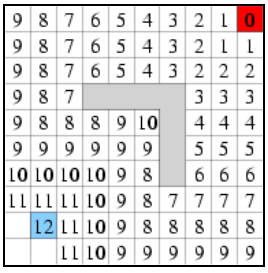
3.1 Informované hledání cesty

Při informovaném hledání cesty má robot k dispozici kompletní znalost okolního prostředí – jak informace o překážkách, tak informace o poloze startu a cíle. Robot nejprve na základě mapy vypočítá kompletní trasu, po které se posléze začne pohybovat směrem k cíli. Jsou možné i přístupy, kdy robot vypočte část trasy, přesune se o kus blíže cíli a zde započne nový výpočet – do cíle tak dospěje po více krocích.

3.1.1 Metody rastrové a dekompoziční

Plánování na mřížce – jedná se o aproximativní metodu, která daný prostor převádí na diskrétní tvar. Přesnost a výpočetní náročnost je odvislá od velikosti buněk diskrétního modelu. Aproximativní rozklad prostředí můžeme vidět na Obr. 2. Dle [5] je možné aplikovat na problematiku plánování na mřížce řadu algoritmů jako jsou:

- *potenciálová pole* – každé buňce v mapě je přiřazena hodnota, která se použije pro rozhodování, kterým směrem se vydat. Cíli se přiřazuje zpravidla hodnota co nejnižší a startu naopak co nejvyšší. Hodnoty se přiřazují například pomocí algoritmu *záplavového vyplňování (flood fill),* viz Obr. 3. Při pohybu mřížkou se pak postupuje v sestupném smyslu od nejvyšších hodnot po nejnižší, přičemž buňky v bezprostředním okolí překážek mají hodnotu vyšší než jejich vzdálenější okolí. Tak se robot vyhne překážkám.



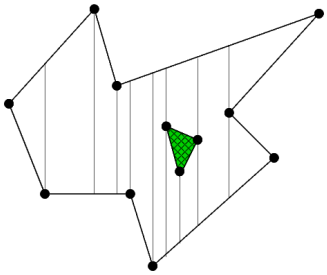
Obr. 3 Mřížka s potenciálovým polem po záplavovém vyplňování [5].

- *grafový přístup – procházení do šířky (breath-first search)* – algoritmus dělí buňky (případně vrcholy) na tři kategorie: *nenavštívené (unvisited)* – nemají přiřazenu vzdálenost, *živé (alive)* – již navštívené buňky, ale s nenavštívenými sousedními buňkami a *mrtvé (dead)* – navštívené buňky se všemi sousedními také navštívenými. Typická implementace spočívá v uchovávání fronty ještě nezpracovaných buněk nebo vrcholů. Z fronty se vždy vybere jeden prvek a spolu s ním všechny prvky, do kterých vede hrana a jsou nenavštívené.

- *Dijkstrův algoritmus* (a jeho modifikace A*, B*) - používá se pro nalezení nejkratší cesty na mřížce. Jedinou modifikací proti předchozímu řešení je nahrazení obyčejné fronty *prioritní frontou,* která je tříděna dle vzdálenosti buněk (vrcholů) od cíle. Algoritmy A* [ei sta:] a B* [bi: sta:] přinášejí další přístupy k třídění fronty nezpracovaných prvků (navíc kromě vzdálenosti od cíle zohledňují také součet vzdálenosti s ohledem na vzdálenost

od startu nebo odhad vzdálenosti od startu). Algoritmy jsou tak agresivnější a rychleji konvergují k řešení, ovšem to již není vždy nejkratší.

- *pseudo-Voronoi diagramy* – ne vždy je v praxi potřeba cesty nejkratší. Někdy je požadavkem například dodržovat odstup od překážek. V kombinaci s procházením do šířky z každé překážky je možné definovat *Voronoi hrany* tam, kde se setkají vlny z různých překážek. Mnohem typičtější aplikací Voronoi diagramů než je mřížka je prostředí složené z polygonů. Více o Voronoi diagramech proto naleznete v kapitole *3.1.2 Metody mapy cest*.



Obr. 4 Lichoběžníková dekompozice [6].

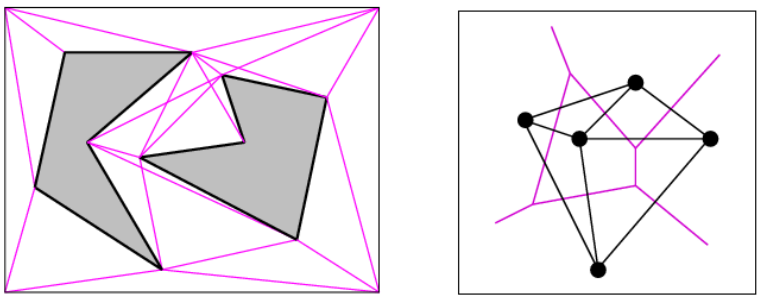
Lichoběžníková dekompozice – dekomponuje prostředí do jednodušších útvarů – nepřekrývajících se buněk. Ty představují lichoběžníky nebo trojúhelníky. Z každého vrcholu mapy jsou (pokud je to možné) vedeny dva vertikální paprsky, dokud nenarazí na překážku. Ukázka takového rozkladu je na Obr. 4. Výsledná cesta se pak hledá z topologického grafu extrahovaného ze seznamu lichoběžníků, které jsou mezi počátečním a cílovým lichoběžníkem.

3.1.2 Metody mapy cest (roadmaps)

Tyto metody ze spojité reprezentace prostoru vytvářejí mapu cest – graf představující volný prostor. Hranami grafu jsou tedy průchodné cesty, po kterých se robot může volně pohybovat. Pro nalezení výsledné trasy se pak používají algoritmy hledání cesty grafem. Dle [6] mezi metody mapy cest můžeme řadit například:

- *graf viditelnosti* – jednoduchá struktura, která umožňuje nalézt nejkratší cestu mezi dvěma body. Za vrcholy grafu se vezmou vrcholy překážek (polygonů) a pozice startu a cíle. Dva vrcholy jsou spojeny hranou, pokud na sebe „vidí“ = jejich spojnice neprochází překážkami, viz Obr. 5 vlevo. Rychlost algoritmu pak samozřejmě závisí na počtu hran.

- *Voronoi diagram* – představuje planární graf, geometrickou strukturu v rovině tvořenou body se stejnou vzdáleností od překážek. Spojnice těchto bodů vymezují uzavřené nebo otevřené oblasti nazvané *Voronoiovy buňky*, po jejichž hranách (na Obr. 5 vpravo vyznačeny fialovou bavou) se robot může pohybovat s jistotou, že si udržuje maximální vzdálenost k překážkám. Proces tvorby těchto buněk nazýváme *Voronoi teselací* [7].



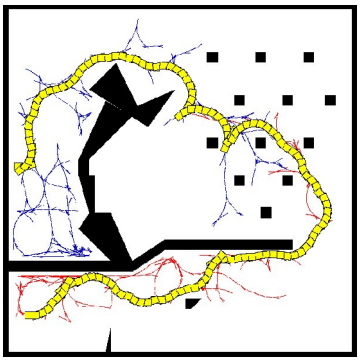
Obr. 5 Graf viditelnosti (vlevo) a Voronoi diagram s jednou uzavřenou a čtyřmi otevřenými oblastmi (vpravo) [6].

3.1.3 Pravděpodobnostní plánování

Tato oblast dle [4] představuje plánovací strategii založenou na formalizaci pomocí konfiguračního prostoru C_{space}, u kterého nás zajímá počet parametrů jednoznačně definujících pozici/konfiguraci robotu. Jedná-li se o rameno, počet parametrů je shodný s počtem kloubů. U všesměrového mobilního robotu postačí tři parametry (x, y a směr, pakliže omezení vlivem natočení nápravy kol jaké je možné např. u osobního automobilu neuvažujeme).

Volný konfigurační prostor C_{free} je pak prostorem všech konfigurací, které nekolidují s překážkami. Podmínkou úspěšného plánovaní je požadavek, aby startovní a cílová konfigurace náležely volnému konfiguračnímu prostoru. Problém plánování zde představuje hledání křivky ve volném konfiguračním prostoru, která umožní provést sekvenci dovolených pohybů.

V nejjednodušší formě *pravděpodobnostní plánovač* pracuje ve dvou fázích, nejprve vygeneruje graf cest (roadmap) a pak hledá cestu grafem. Graf cest je pravděpodobnostní, protože vzniká z náhodně vybíraných konfigurací, u kerých je ověřeno, že náleží C_{free} a lze je spojit s ostatními přímou cestou rovněž náležící C_{free}. Toto přidávání hran vzniklých z vyhovujících náhodných konfigurací plánovač opakuje dokud nenapojí hranu vedoucí do cíle. Pro urychlení konvergence k cíli je vhodné vždy po určitém počtu iterací vyzkoušet napojit přímo konfiguraci cílovou.



Obr. 6 RRT – plánování cesty automobilu se zohledněním jeho manévrovacích možností [4].

Výhoda metody náhodných vzorků je patrná při mnohodimenzionálním konfiguračním prostoru, kdy potřebujeme plánovat např. rameno manipulátoru s 5 stupni volnosti a krokovým rozlišením 1 úhlový stupeň. Při rozsahu pohybu kloubů 120° bychom potřebovali $120^5 = 24883200000$ bitů pro uložení všech vzorků. Ukládat pouze náhodné vzorky které prošly procedurou testující příslušnost k C_{free} a testem na nepřítomnost kolize s překážkou je mnohem únosnější.

Obecným problémem pravděpodobnostního plánovače bývá velký počet hran a s tímto faktem související kostrbatost cesty vedoucí k cíli. Proto je vhodné dále aplikovat algoritmy vyhlazování výsledné cesty.

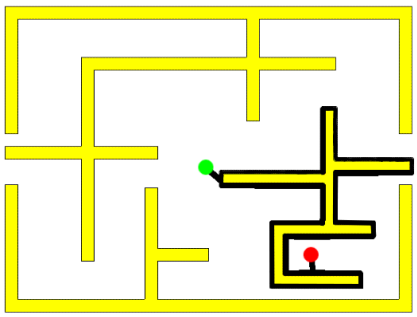
Jak uvádí [8]: „Popis C_{free} pomocí jediného stromu má však za následek, že i relativně blízké pozice mohou být od sebe velmi vzdáleny, vedeme-li cestu přes pravděpodobnostní mapu cest. Problém řeší *RRT ("Rychle rostoucí nahodné stromy")*, kdy pokaždé vzniká strom nový s ohledem na pozici startu a cíle.“ Tomuto algoritmu se budeme věnovat dále v samostatné (čtvrté) kapitole.

3.2 Neinformované hledání cesty

Při neinformovaném hledání cesty robot nemá k dispozici kompletní znalost okolního prostředí. Informace o překážkách zcela chybí, v extrémních případech mohou chybět i informace o pozici startu a cíle. Robot tedy řeší obtížnou úlohu mapování a lokalizace současně, kdy se musí plně spolehnout na poznávání prostoru pomocí vlastních senzorů. Podle [9] mezi nejstarší a nejznámější metody pracující v neznámém prostředí patří Bug algoritmy, z nichž si můžeme uvést:

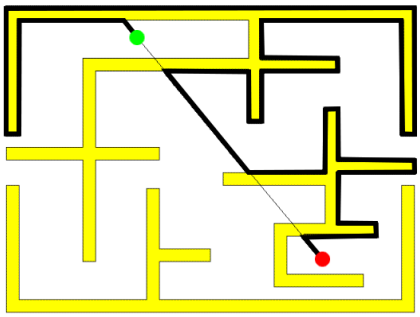
- *algoritmus Bug1* – na vstupu algoritmu je robot s dotykovým senzorem reprezentovaný bodem v rovině. Výstupem pak je cesta k cíli nebo řešení, že taková cesta neexistuje. Základní myšlenkou je, že se robot vydá směrem k cíli dokud nenarazí na překážku. V případě kolize

robot těsně obchází překážku, dokud opět nedorazí do bodu kolize. Poté nalezne nejlepší pozici pro další pohyb do cílového místa a z této pozice opouští bezprostřední okolí překážky a dále směřuje směrem k cíli.



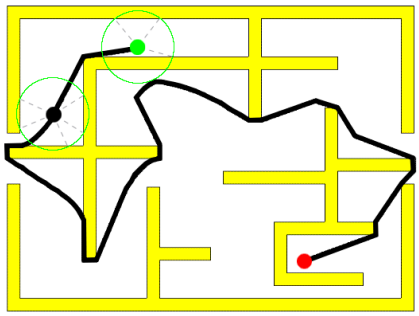
Obr. 7 Algoritmus Bug1 [11].

- *algoritmus Bug2* – je vylepšením předchozího algoritmu. Také u něj předpokládáme, že robot je bodem v prostoru a je vybaven dotykovým senzorem. Obdobně jako *Bug1* využívá dvou typů chování robotu: pohyb prostorem směrem k cíli a pohyb kolem překážky. Během pohybu k cíli se však robot pohybuje podél úsečky *m* vedené od startovní k cílové pozici. V případě kolize robot objíždí překážku na libovolnou stranu dokud nedosáhne takového bodu na úsečce *m*, který je blíž cíli než bod kolize.



Obr. 8 Algoritmus Bug2 [11].

- *algoritmus Tangent Bug* – (někdy uváděn také jako *VisBug,* viz [11]) vznikl vylepšením algoritmu *Bug2*, kdy robot opět uvažujeme jako bod, tentokrát ale se senzorem s konečným nebo nekonečným dosahem a rozsahem 360°. Tento senzor je schopen detekovat přítomnost překážek kolem robotu. Díky této schopnosti, která se projevuje zkrácením trasy, robot objíždí překážky jen o tolik, kolik je v dané situaci nutné. Proto je tento algoritmus v praxi nejpoužitelnější z uvedených Bug algoritmů.

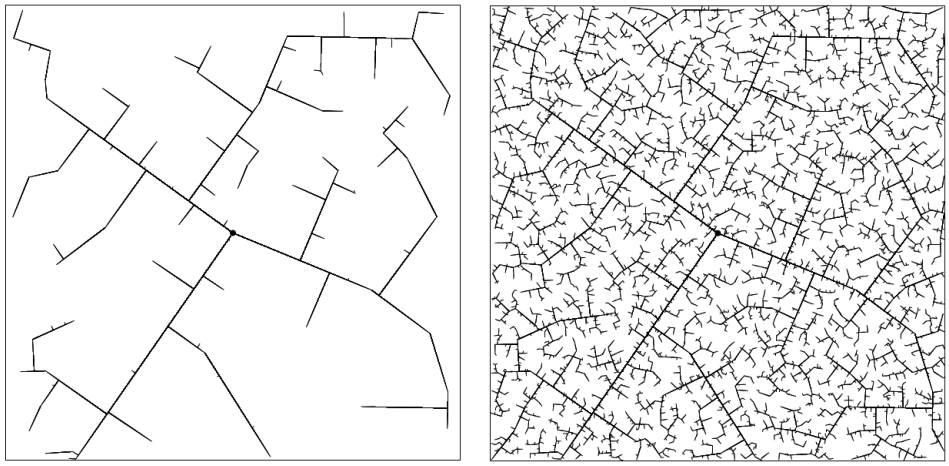


Obr. 9 Algoritmus Tangent Bug se znázorněným dosahem senzoru na dvou vybraných místech [11].

4 ALGORITMUS RRT

Rapidly-exploring Random Tree (RRT, rychle rostoucí náhodný strom) je datová struktura a algoritmus vyvinutý Stevenem M. LaValle a Jamesem Kuffnerem určený pro efektivní vyhledávání v nekonvexních vysokodimenzionálních prostorech. Ve své nejjednodušší podobě je konstruován postupem, kdy každý náhodný vzorek polohy v prostoru je připojen k nejbližšímu vzorku, který je součástí stromu. [12]

Algoritmus přírůstkovým způsobem vytváří stromovou strukturu a velmi rychle prorůstá prostředím. S postupujícími iteracemi se prudce zkracuje očekávaná vzdálenost mezi náhodným vzorkem a nejbližším bodem na stromu. RRT je velmi vhodný pro nasazení na problémy plánování cest, které zahrnují složité překážky a kinematická omezení robotu s mnoha stupni volnosti [13].



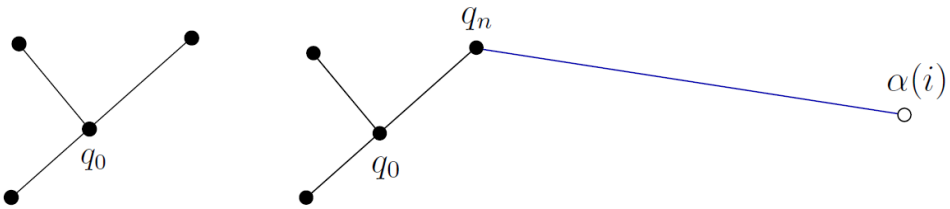
Obr. 10 RRT po 45 iteracích (vlevo) a 2345 iteracích (vpravo) [12].

RRT je možno použít ke tvorbě otevřených trajektorií v nelineárních systémech. Algoritmus se běžně považuje za metodu typu Monte Carlo vyhledávající ve velkých Voronoi diagramech. Některé z variant RRT lze považovat za stochastické fraktály. O algoritmu RRT a plánování pohybu pomocí této metody velice podrobně informuje [4], kde je demonstrován potenciál algoritmu v početném množství modifikací a aplikací.

Obvykle samotný RRT algoritmus k nalezení optimálního řešení nestačí. Bývá často začleněn jako dílčí součást do plánovacích algoritmů šitých na míru daným úlohám a prostředím. Je např. velmi žádoucí aplikovat vhodné *algoritmy pro vyhlazení trajektorie* k vylepšení trajektorie nalezené pomocí Rychle rostoucích náhodných stromů nebo jiná vylepšení napomáhající konvergenci řešiče k cíli (např. optimalizací výběru náhodného vzorku – vektoru vzhledem k poloze cíle za pomocí vektorové algebry).

4.1 Základní podoba algoritmu

Jak uvádí [13] základní algoritmus pro konstrukci RRT bez uvažování překážek vyžaduje pouze hustou množinu stavového prostoru. Cílem je dostat se co nejblíže ke každé konfiguraci v prostoru, začínajíc v počáteční konfiguraci. Algoritmus iterativně připojuje náhodné vzorky $\alpha(i)$ a nové hrany grafu, které jsou spojnicí náhodného vzorku $\alpha(i)$ a nejbližšího bodu na stromu G, kterým je na Obr. 11 vertex q_n. Obr. 12 popisuje exploraci algoritmu v pseudokódu dle [4]. Metoda NEAREST slouží k nalezení nejbližší konfigurace vzhledem k náhodnému vzorku na stromu.

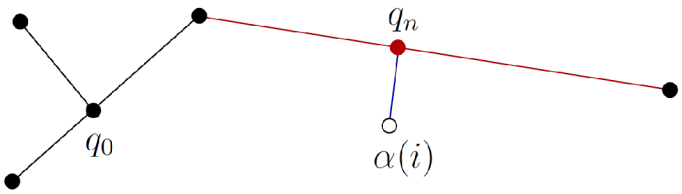


Obr. 11 Každou novou hranu představuje spojnice mezi vzorkem α(i) a nejbližší konfigurací [4].

```
SIMPLE_RRT (q0)
  1.  G.init (q0);
  2.  for i = 1 to k do
  3.      G.add_vertex (α(i));
  4.      qn ← NEAREST (S(G, α(i));
  5.      G.add_edge (qn, α(i));
```

Obr. 12 Nejjednodušší podoba RRT algoritmu bez uvažování překážek v pseudokódu.

V případech, kdy nejbližší konfigurace (nejbližší bod na stromu) q_n leží někde na větvi stromu a ne na jeho okrajovém uzlu, dojde k rozdělení větvě na dvě a přidání dvou nových uzlů q_n a $\alpha(i)$ spolu s novou hranou (větví) mezi q_n a $\alpha(i)$, jak je patrné z Obr. 13. [4]



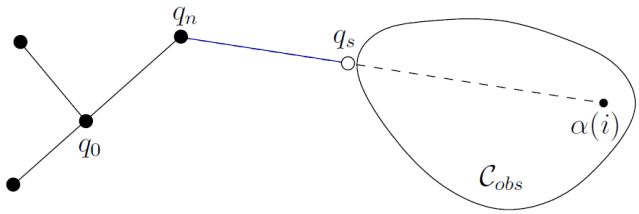
Obr. 13 Připojení náhodného a nového vzorku, rozdělení hrany za situace, že nejbližší konfigurace neleží na okrajovém uzlu [4].

4.2 Jedno-stromové prohledávání

Poměrně efektivní plánovač můžeme sestavit na užitím algoritmu uvedeném v pseudokódu na Obr. 14. Tento postup již zohledňuje přítomnost překážek, jak můžeme vidět na řádcích 4 a 5. Metoda STOPPING-CONFIGURATION vrací bod q_s, který vznikne ořezáním spojnice vzorku a nejbližší konfigurace na stromu podle překážky. Postup je patrný z Obr. 15. Aby byl hledací algoritmus opravdu účinný, je potřeba místo náhodného vzorku $\alpha(i)$ vždy po několika iteracích použít přímo koncovou konfiguraci q_g. Počet iterací, po kterých se vždy zkouší napojit do cíle, může být dán pevným parametrem nebo náhodným faktorem (každá iterace může testovat náhodu s vhodnou pravděpodobností.)

```
SIMPLE_RRT (q0)
  1.  G.init (q0);
  2.  for i = 1 to k do
  3.      qn ← NEAREST (S,α(i));
  4.      qs ← STOPPING-CONFIGURATION (qn, α(i));
  5.      if qs ≠ qn then
  6.          G.add_vertex ( qs);
  7.          G.add_edge (qn, qs);
```

Obr. 14 RRT algoritmus uvažující překážky.



Obr. 15 V případě překážky putuje vzorek α(i) až k hranici překážky (bod q_s), o tuto akci se stará algoritmus detekující kolize [4].

4.3 Balancované dvou-stromové prohledávání

Mnohem lepšího výkonu můžeme dle [4] dosáhnout použitím dvou RRT: jednoho, který prozkoumává prostor z počáteční konfigurace q_i, a druhého, který mu jde naproti z koncové konfigurace q_g. Pro dvou-stromové prohledávání musíme zajistit, aby se oba stromy potkaly a přitom si zachovaly svou rychle rostoucí podstatu. Toho nejlépe dosáhneme tak, že oboustranné prohledávání *balancujeme*.

```
RDT_BALANCED_BIDIRECTIONAL (qI,qG)
1.    Ta.init (qI); Tb.init (qG);
2.    for i = 1 to K do
3.       qn ← NEAREST (Sa,α(i));
4.       qs ← STOPPING-CONFIGURATION (qn, α(i));
5.       if qs ≠ qn then
6.          Ta.add_vertex ( qs);
7.          Ta.add_edge (qn, qs);
8.          q'n ← NEAREST (Sb,qs);
9.          q's ← STOPPING-CONFIGURATION (q'n, qs);
10.         if q's ≠ q'n then
11.            Tb.add_vertex (q'n);
12.            Tb.add_edge (q'n, q's);
13.         if q's = qs then return SOLUTION;
14.      if |Tb| > |Ta| then SWAP (Ta,Tb);
15.   return FAILURE
```

Obr. 16 Balancovaný dvou-stromový algoritmus RRT v pseudokódu.

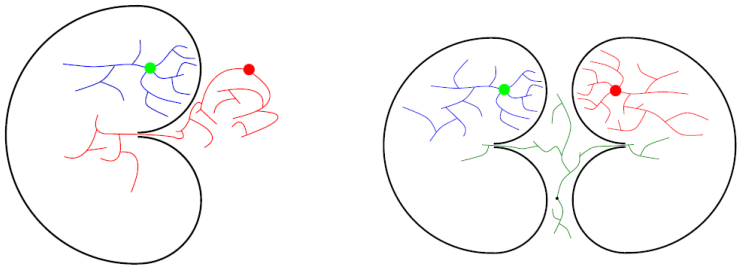
Graf *G*, jak jej známe z předchozích ukázek na Obr. 12 a 14, je dekomponován na dva stromy, které na ukázce v pseudokódu na Obr. 16 představují T_a a T_b. Jeden strom vychází z q_i a druhý z q_g. Vždy po několika iteracích jsou oba stromy navzájem zaměněny. Je proto třeba pamatovat na fakt, že strom T_a nemusí být vždy ten, který startoval z konfigurace q_i.V každé iteraci strom T_a roste stejným způsobem jako v příkladě na Obr. 14. Pokud je přidán nový vertex q_s k T_a , je bezprostředně po tom (řádek 10-12 v ukázce pseudokódu na Obr. 16) vykonán požadavek rozšířit strom T_b. Raději než nová náhodná konfigurace *α(i)* je v tomto případě použit přímo nový vertex q_s, který byl právě připojen ke stromu T_a. Tento postup zajistí, že T_a roste směrem k T_b. Na řádku 13 je pak testováno, zda se oba stromy skutečně potkaly a v případě úspěchu je nalezeno řešení.

Řádek 14 představuje zmíněné balancování. Pokud má jeden ze stromů problémy s expandováním, je třeba zaměřit na něj více energie. Proto se nový vzorek hledá vždy pro menší strom. Pojem menší strom můžeme chápat ve smyslu stromu s menším počtem vertexů nebo s kratší délkou všech segmentů stromu.

4.3.1 Více-stromové prohledávání?

Dvou-stromové vyhledávání přináší výhody jak v podobě výkonnostního rozdílu na jednoduchých mapách, tak například v podobě snazšího úniku z různých pastí, které algoritmu dokáže prostředí klást pod nohy. Nejjednodušší past si dle [4] nebo [14] můžeme představit třeba v podobě komůrky s tenkým a dlouhým otvorem ven, kterým bude tunel otočený směrem dovnitř komůrky (tzv. *Bug trap*, viz Obr. 17 vlevo), podobně jako past na myši. V případě, že počáteční konfigurace je uvnitř pasti a konečná venku, oboustranné pronikání v podobě dvou stromů přináší nespornou výhodu.

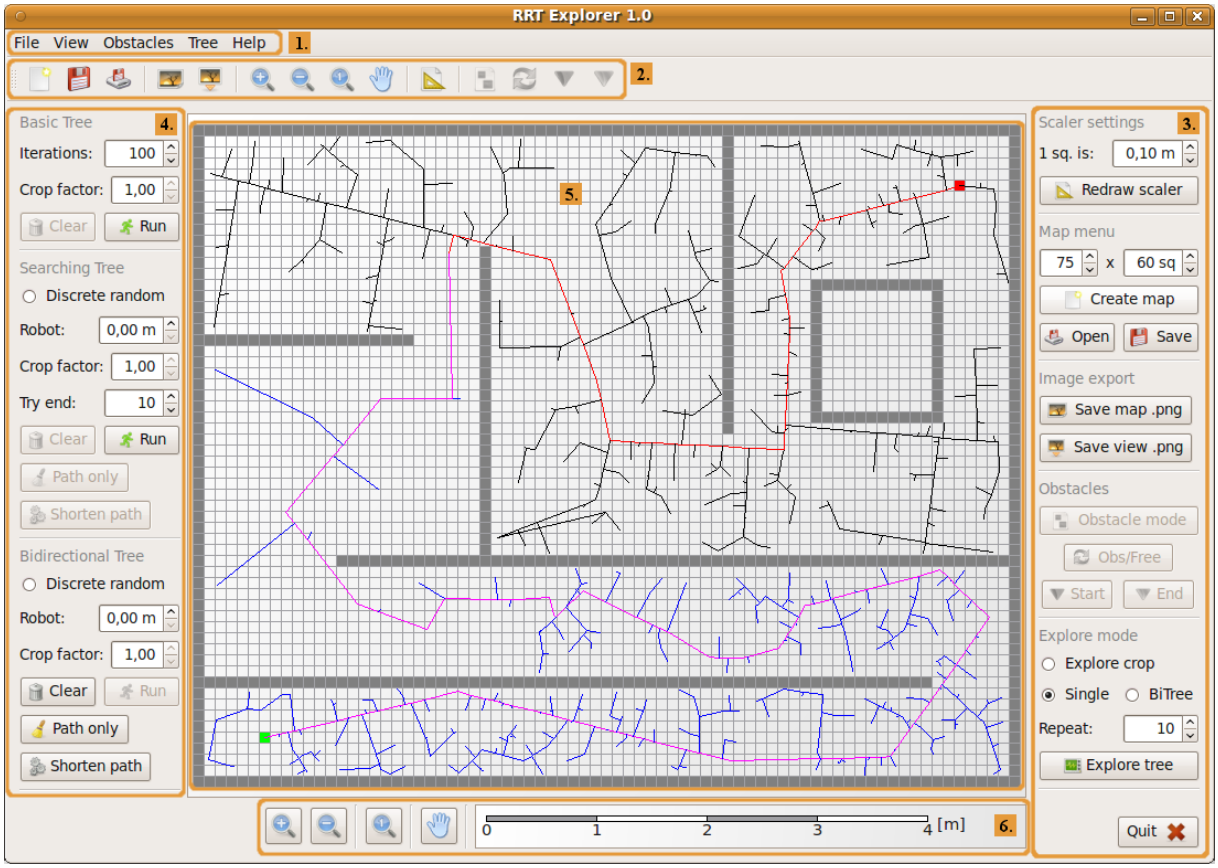
Nabízí se proto otázka plánování metodou více než dvou RRT – např. pro situaci, kdy výše zmíněnou past Bug trap zdvojíme – start i cíl budou každý v jedné komůrce (podobně jako můžeme vidět na Obr. 17 vpravo). Popis podmínek případného balancování vícenásobných stromů stromů nebo metod rozhodování, kdy a kde takové stromy při hledání dynamicky tvořit, jsou však již nad rámec této práce.



Obr. 17 Past Bug trap (vlevo) a zdvojená varianta pasti (vpravo). Startovní pozice znázorněna zeleně, cíl červeně. [14].

5 FUNKCIONALITA APLIKACE

Následující kapitola se bude věnovat popisu funkcionality a ovládání linuxové aplikace *RRT Explorer 1.0* pro prostředí Ubuntu, která je výstupem praktické části této bakalářské práce. Program slouží k měření vlastností algoritmu RRT v jeho jedno i dvou-stromové variantě pomocí parametrizovatelného řešiče a speciálního dávkového módu na daných mapách prostředí, které umožňuje pohodlně editovat. Grafické uživatelské rozhraní, nápověda i dokumentace ve zdrojových kódech jsou v anglickém jazyce, neboť angličtina je jazykem počítačů. Na Obr. 18 jsou popsány jednotlivé části hlavního okna:



Obr. 18 RRT Explorer 1.0 – hlavní okno aplikace.

1. Hlavní menu
2. Lišta akcí
3. Pravé funkční menu
4. Levé funkční menu
5. Scéna
6. Měřítko a tlačítka zoomu

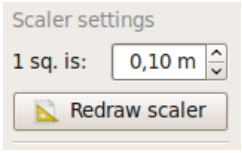
5.1 Editor mapy

Prvním krokem ke zkoumání algoritmu RRT pomocí aplikace *RRT Explorer* je vytvoření nové mapy prostředí nebo otevření již existujícího souboru. Mapa je v aplikaci představována sítí čtverců, kde tmavošedé čtverce jsou překážky. Přípona mapových souborů je **.rrtm*. Tvorbu mapy je vhodné začít volbou měřítka.

5.1.1 Měřítko

Měřítko se nastavuje v menu *Scaler settings* (Obr. 19), které se nachází v pravém funkčním menu zcela nahoře (viz Obr. 18). Hodnota měřítka zde představuje velikost jednoho čtverečku mapy v reálném prostoru. Jednotkou jsou metry a lze ji nastavit v rozmezí 0,01 – 10m.

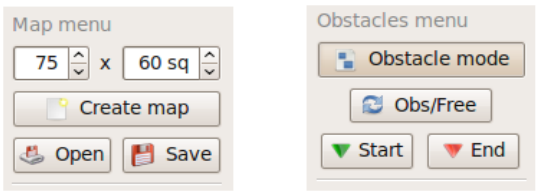
Grafické znázornění měřítka je situováno do spodní části hlavního okna a je možné ho kdykoli překreslit dle aktuálního nastavení stisknutím tlačítka *Redraw scaler*.



Obr. 19 Scaler settings – nastaveni měřítka.

5.1.2 Nová mapa a editace překážek

Pro vytvoření nové mapy je třeba v Map menu (Obr. 19) nastavit rozměry mapy (počet čtverců v osách x a y, minimálně *10 x 10*). Po stisknutí tlačítka *Create map* se na scénu vykreslí prázdná čtverečkovaná síť.

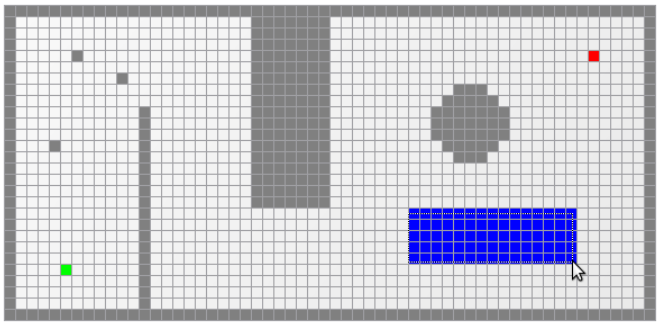


Obr. 20 Map menu – vytváření, načítání a ukládáni mapy.
Obstacles menu – editace překážek.

K editaci slouží *Obstacle mode*, do kterého vstoupíme stisknutím stejnojmenného tlačítka v menu *Obstacles* (viz Obr. 20 vpravo). Poté je možné na hlavní scéně:

- označovat čtverečky *klikáním* na ně
- označovat více čtverečků najednou *klikáním se současně stisklou klávesou Ctrl*
- označovat obdélníkové plochy *stisknutím pravého tlačítka myši a tažením*

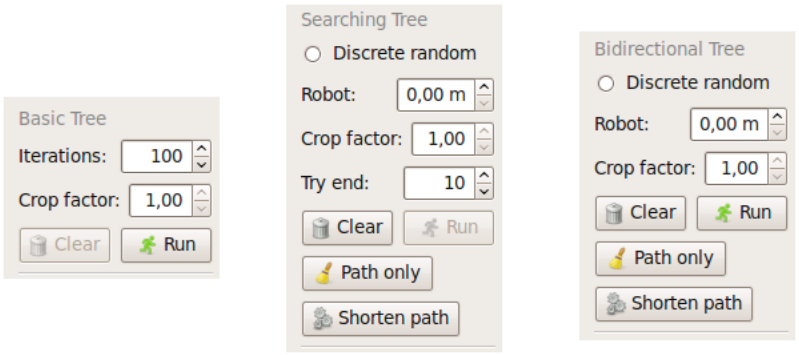
Tlačítko *Obs/Free* mění vybrané čtverečky z volných na překážky a naopak. Tlačítka *Start* a *End* slouží k definování startovní/cílové pozice robotu. Tato akce může být provedena pouze v případě právě jednoho čtverečku ve výběru. *Start* a *End* fungují obdobně jako tlačítko *Obs/Free* – revertují start a cíl zpět na volná políčka. Ukázka práce v editoru je na Obr. 21.



Obr. 21 Ukázka práce v editoru – obdélníkový výběr.

5.2 Řešič stromů

Aplikace nabízí algoritmus RRT ve třech provedeních: základní variantu, která neuvažuje kolizi s překážkami (pouze pro demostraci), dále pak RRT umožňující jedno-stromové hledání a RRT ve dvou-stromové balancované variantě. Budou popsány blíže v následujících kapitolách. Nastavení parametrů řešiče se provádí z levého funkčního menu.

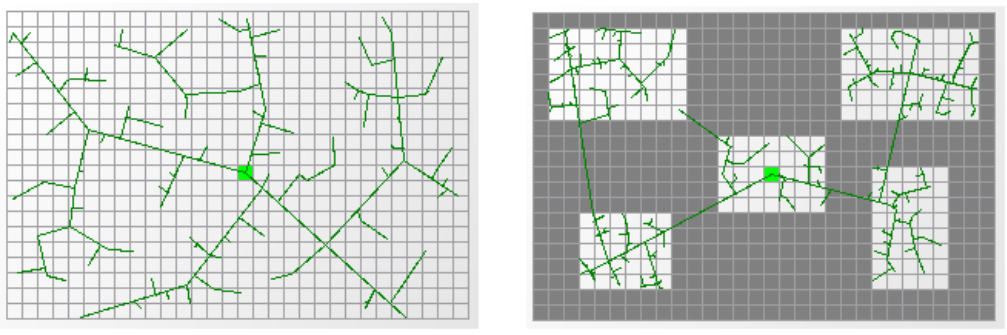


Obr. 22 Nastavení řešiče z levého funkčního menu.

5.2.1 Basic tree – základní strom

Tento strom slouží pouze k prvnímu seznámení a demonstraci algoritmu RRT. Pro své spuštění vyžaduje pouze startovní bod, protože neprovádí hledání. V případě překážek sice vybírá náhodné vzorky z volného prostoru, ale neuvažuje detekci překážek při napojování vzorku. Počet iterací lze zvolit v rozmezí 10 – 10 000, viz hodnota *Iterations* (Obr. 22 zcela vlevo). Algoritmus se spouští tlačítkem *Run*, vykreslený strom se ze scény smaže tlačítkem *Clear*.

Crop factor udává hodnotu z intervalu *<0,1; 1>*, kterou je násobena délka každé připojované větve a větev takto zkrácena. Nastavíme-li např. hodnotu *0,5*, každá větev bude zkrácena na polovinu své délky.



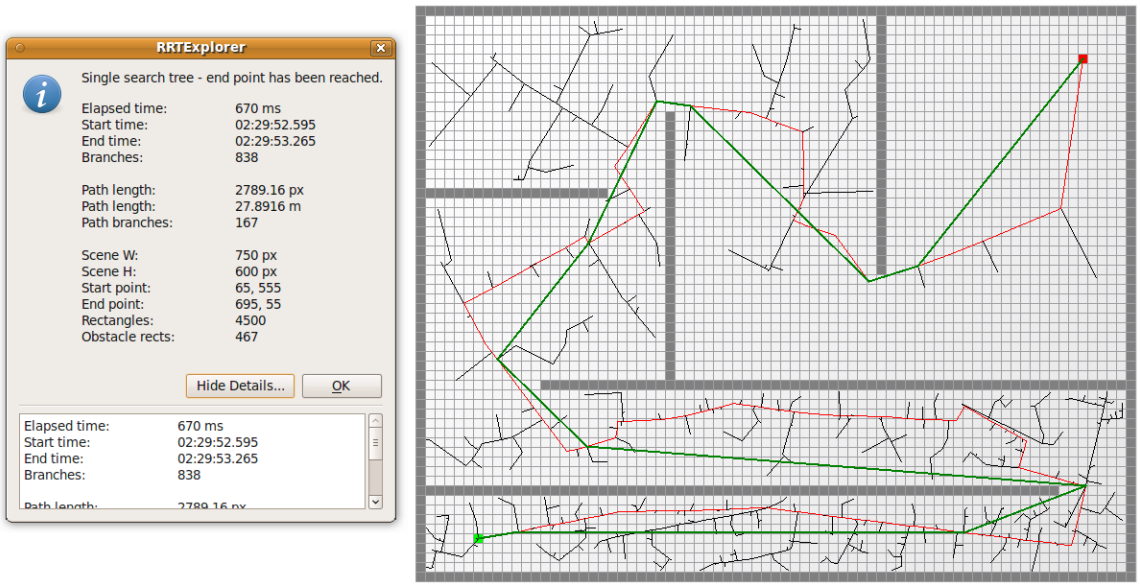
Obr. 23 Basic tree exploruje prostorem.

5.2.2 Searching tree – jedno-stromové hledání

Tento strom představuje jedno-stromové hledání, které je popsáno v kapitole 4.2. Pro své spuštění proto vyžaduje definován start a cíl. Parametry, které je možné v řešiči nastavit (viz Obr. 22 uprostřed) představují:

Discrete random – pokud je zaškrtnuto, vybírá pouze vzorky ze středů volných čtverců.
Robot – velikost robotu (poloměr případně největší rozměr od středu) v metrech.
Crop factor – násobič větví z intervalu *<0,1; 1>*.
Try end – po kolika iteracích je vždy testováno připojení do cílové pozice.

Po nastavení požadovaných hodnot můžeme stisknout tlačítko *Run* a strom se začne vykreslovat do scény. Jakmile hledání dosáhne cílového bodu, objeví se dialogové okno (Obr. 24 vlevo) s informacemi o právě spočítaném stromu (doba trvání v milisekundách, čas startu a konce výpočtu, počet větví, délka cesty v px a metrech, počet větví tvořících cestu a další hodnoty jako rozměry scény apod.). Pomocí tlačítka *Show details* je možné vyvolat nabídku, odkud lze všechny hodnoty zkopírovat do schránky.

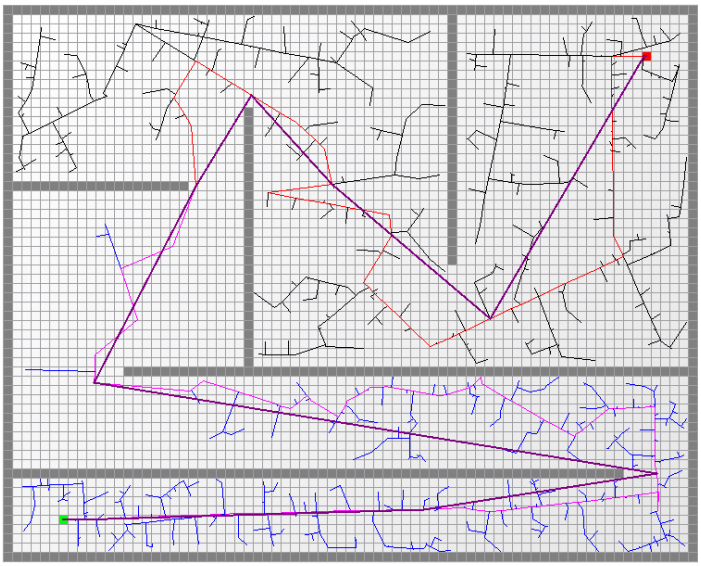


Obr. 24 Searching tree se zkrácenou cestou a dialogem informujícím o výpočtu.

Tlačítko *Shorten path* aplikuje algorimus zkrácení cesty a informuje uživatele dialogem o předchozí a nové cestě – délkách a počtu větví. Tlačítko *Path only* přepíná pohled zobrazující celý strom nebo pouze cestu.

5.2.3 Bidirectional tree – dvou-stromové hledání

Tato verze algoritmu představuje dvou-stromové balancované prohledávání, jak je popsáno v kapitole 4.3. Nastavení řešiče a význam hodnot je obdobný jako u *Searching tree* s jedním rozdílem – dvou-stromová verze nepotřebuje parametr *Try end*. Ukázka menu *Bidirectional tree* je na Obr. 22 zcela vpravo. Výpočet ve finále opět informuje o právě vykresleném stromu.



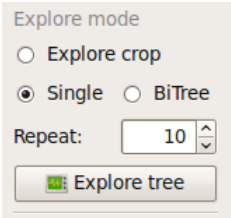
Obr. 25 Bidirectional tree – ukázka algoritmu se zkrácenou výslednou cestou.

5.3 Mód měření

Tato funkce ke svému chodu vyžaduje balíčky *python2.6*, *python-numpy* a *python-matplotlib*, které lze získat zadáním příkazů:

```
sudo apt-get install python2.6
sudo apt-get install python-numpy
sudo apt-get install python-matplotlib
```

Mód měření (*Explore mode*) je dávkový mód pro efektivní měření výkonu *Searching tree* a *Bidirectional tree* v cyklech, kdy jsou výsledky ukládány do souborů a pomocí Pythonu zobrazovány grafem a obohacovány o další statistické údaje. Obecně se sestává ze dvou režimů – *jednoduchého měření* a *měření Crop factoru*. Nastavení se provádí v menu *Explore mode*, které je situováno do pravého dolního rohu aplikace a blíže zobrazeno na Obr. 26. Oba režimy jsou popsány v následujících podkapitolách.



Obr. 26 Menu explore mode.

5.3.1 Jednoduché měření

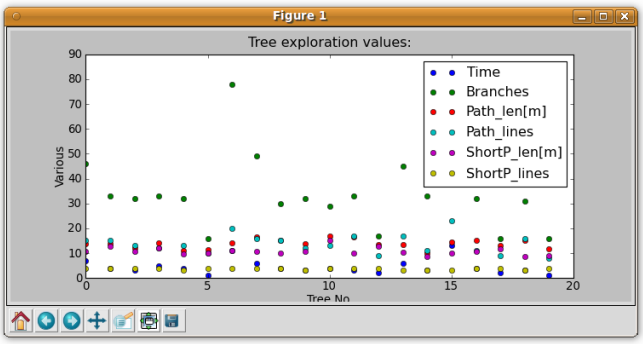
Tento mód je aktivován tlačíkem *Explore tree*, pokud není zaškrtnuta volba *Explore crop*. Hodnota *Repeat* udává počet měření, přepínač *Single/BiTree* přepíná mezi měřením *Searching tree* a *Bidirectional tree* s jejich nastavením z příslušného levého funkčního menu. Po stisku tlačítka je uživatel dotázán, aby vybral existující adresář, ve kterém bude vytvořena složka s artefakty měření pojmenovaná:

ExpRRT_[Single/BiTree]_[datum]_[čas]_[počet iterací]it

Výstupem jednoduchého měření jsou:

- *soubor *.tsv* (tab separated values – tabulátorem oddělené hodnoty) – surový log měření s význačnými hodnotami (čas, počet větví, délka cesty v metrech, počet větví cesty, délka zkrácené cesty v metrech a počet větví zkrácené cesty) pro každý strom.
- *soubor *_result.tsr* – textový soubor s mediány, aritmetickými průměry a směrodatnými odchylkami všech výše uvedených hodnot.
- *interaktivní graf* – v podobě popup okna matplotlib, zobrazuje hodnoty *.tsv souboru. Grafem se dá zoomovat, panovat, ukládat z něj screenshoty.
- *soubor *.png* – který je screenshotem grafu.
- *soubor *.rrtm* – uložená mapa, na které probíhalo měření.

kde * představuje název složky *ExpRRT_[Single/BiTree]_[datum]_[čas]_[počet iterací]it.*



Obr. 27 Interaktivní graf jednoduchého měření.

5.3.2 Měření Crop factoru

Mód je aktivován tlačíkem *Explore tree*, pokud je zaškrtnuta volba *Explore crop* v menu Explore mode (Obr. 26). Slouží k efektivnímu měření nastavení optimálního Crop factoru.

V tomto módu se hodnota *Crop factor* v levém funkčím menu příslušných stromů chová jinak, než jsme byli doposud zvyklí, viz příklad: nastavíme-li *Crop factor 0,1*, strom bude měřen *Repeat* krát pro každý *Crop factor* od *0,1* do *1,0* jdoucí po kroku *0,1*. Metoda pracuje jako algoritmus na Obr. 28.

```
for (i = CropFactor; i <= 0.1; i += CropFactor)
  for (j = 0; j < Repat; j++)
  compute_tree_save_values();
```

Obr. 28 Postup metody měření Crop factoru

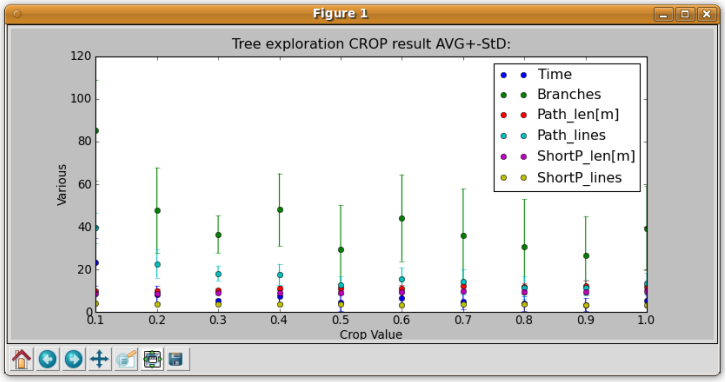
Výstupem měření Crop factoru je složka pojmenovaná:

ExpRRT_[Single/BiTree]CROP_[datum]_[čas]_[počet iterací]it

která obsahuje:

- *soubory *.tsv* – surové logy naměřených hodnot, pro každou hodnotu *Crop factoru* jeden.
- *soubor *_AvgStD.tsr* – textový soubor s aritmetickými průměry a směrodatnými odchylkami ze všech významných hodnot pro každý crop.
- *soubor *_Medians.tsr* – textový soubor s mediány z významných hodnot pro každý crop.
- *škálovatelný graf* – v podobě interaktivního okna, zobrazuje hodnoty **_AvgStD.tsr* souboru.
- *soubor *.png* – který je screenshotem grafu.
- *soubor *.rrtm* – uložená mapa, na které probíhalo měření.

kde * představuje název složky s artefakty měření.



Obr. 29 Interaktivní graf měření Crop factoru.

5.4 Další funkce

V následující podkapitole budou popsány další funkce, které nabízí aplikace *RRT Explorer*.

5.4.1 Zoomování a panning

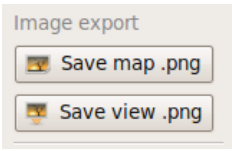
Ve spodní části hlavního okna můžeme najít tlačítka (Obr. 30), která umožňují zoomovat (přibližovat nebo oddalovat) a posouvat scénu (panning).

Obr. 30 Zoomování a panning.

Nechybí ani tlačítko, které vrátí zoom do původního stavu (1:1).

5.4.2 Export obrázků

Je možný z menu *Image export* (Obr. 31). Tlačítko *Save map .png* uloží obrázek celé mapy, tlačítko *Save view .png* pak ukládá pouze aktuální výřez scény.



Obr. 31 Menu Image export – výstup png obrázků.

5.4.3 Lišta akcí a klávesové zkratky

Obr. 32 Lišta s nejčastěji používanými akcemi.

Dosud jsme si ukázali pouze ovládání z tlačítkových menu. Pro větší komfort uživatele však aplikace nabízí další způsoby ovládání. Všechny akce z tlačítek je možné nalézt v hlavním menu, dále lze vybrané často používané akce nalézt na liště akcí (Obr. 32). Lištu akcí je možné přesouvat a dokovat v jiných částech okna, stejně jako jednotlivé položky hlavního menu. To může být užitečné při práci na velkých mapách na více monitorech.

Většina akcí je asociována s klávesovými zkratkami:

Ctrl + N	Nová mapa.
Ctrl + O	Otevřít mapu.
Ctrl + M	Uloží obrázek .png celé mapy.
Ctrl + W	Uloží .png aktuálního výhledu scény.
Ctrl + Q	Ukončí aplikaci.
Ctrl + R	Překreslí měřítko.
Ctrl + P	Přepíná mód panningu.
Ctrl + B	Přepíná mód překážek.
Ctrl + E	Spustí mód měření.
Num +	Zoom dovnitř.
Num -	Zoom ven.
*Num **	Restore zoom 1:1.
B	Překážka/volné políčko.
[F5, F6, F7]	Pusť [Basic, Searching, Bidirectional] tree.
[F9, F10, F11]	Smaž [Basic, Searching, Bidirectional] tree.
F1	Nápověda.

6 POPIS IMPLEMENTACE PROBLÉMU

Kapitola stručně seznamuje se zvolenými prostředky a metodami implementace aplikace RRT Explorer. Popisuje použitý framework, programovací jazyky, prostředí operačního systému a implementaci vybraných funkcí softwaru.

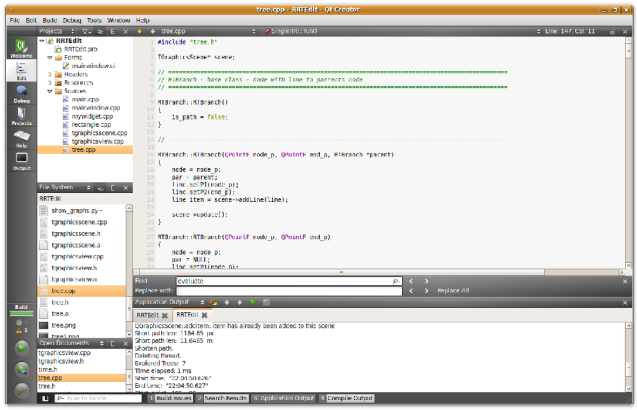
6.1 Operační systém Ubuntu 9.10

Aplikace byla vytvořena v populární linuxové distribuci Ubuntu, konkrétně ve verzi 9.10 „Karimc Koala“. Distribuce byla vybrána s ohledem na snadnou instalaci, širokou podporu komunitou Ubuntu a celkovou uživatelskou přívětivost systému. Systém používám na svém PC spolu s dual-bootem platformy Windows. Proto bylo Ubuntu pochopitelnou volbou.

Obr. 33 Logo linuxové distribuce Ubuntu.

6.2 Aplikační framework Qt 4.6

Qt je jedna z nejpopulárnějších multiplatformních knihoven pro vytváření programů s kvalitním grafickým uživatelským rozhraním. Aktuální je verze 4.6. Jedná se o knihovnu programovacího jazyka C++ (i když existuje i pro jiné jazyky jako je Python, Ruby a mnoho dalších). Podporuje mimo jiné správu vláken, přístup k souborům, zpracování XML a jiné užitečné funkce [15].



Obr. 34 Hlavní okno Qt Creatoru.

Tento framework jsem vybral zejména díky jeho částečné předchozí znalosti a zkušenostmi s ním. Oceňuji jak integrované vývojové prostředí *Qt Creator* (na Obr. 34), ve kterém aplikace Tree Explorer vznikala a které jsem si pro jeho jednoduchost a střídmost v designu velmi oblíbil, tak širokou podporu v přehledné dokumentaci, příkladech a internetových fórech.

6.3 C++

Kromě části měřícího módu je celá aplikace napsaná v jazyce C++. Tento jazyk je natolik rozšířený, oblíbený a univerzální v oblastech použití, že jeho volbu při implementaci problému snad není třeba komentovat. Cílem autora bylo mj. zdokonalit své schopnosti programování v tomto jazyce.

6.4 Python

Ke zpracování a zobrazování výstupních logů měření byl vytvořen skript v jazyce Python (pro počítání mediánů, průměrů, směrodatných odchylek a zobrazení grafického výstupu v podobě interaktivního grafu). Při implementaci bylo použito modulů NumPy (počítání) a Matplotlib (zobrazování výsleků). Jazyk je díky své základně matematických knihoven pro tuto úlohu velmi vhodný.

6.5 Implementace vybraných funkcí

V této podkapitole následuje stručný popis nejzásadnějších případně jinak zajímavých míst implementace dané aplikace.

6.5.1 Reprezentace RRT algoritmu

Vytvořená knihovna RRT implementuje vzorky – větve stromu pomocí třídy *RTBranch* (která velmi zjednodušeně představuje bod a přímku a také ukazatel na svého rodiče). Přímka představuje hranu grafu, bod uzel. Samotný strom (třída *RRTree*) je v podstatě kolekcí těchto větví umožňující provádět nad stromem různé metody, které využívá konkrétní vlákno příslušné varianty stromu *(Simple, Searching, Bidirectional tree).*

Vláknový přístup umožňuje vidět, jak strom roste větev po větvi.

6.5.2 Algoritmy nalezení a zkracování výsledné cesty

Výsledná cesta je nalezena jednoduchým způsobem – postupným vracením ukazatele na rodičovskou větěv, kdy postupujeme odzadu a jako první se dotazujeme na rodiče větve, která dorazila do cíle.

Zkracovací algoritmus pak prochází kolekci přímek (větví) výsledné cesty. Postupuje směrem ze startovního uzlu po větvích cesty a postupně spojuje start s uzlem každé další větve, dokud se mu to daří bez protnutí překážky. Při první kolizi spojované přímky s překážkou se zastaví, vrátí o přímku na cestě zpět – k poslední funkční spojnici, tu uloží do kolekce přímek nové cesty a uzel vezme coby nový startovní bod. Tento postup se opakuje, dokud algoritmus nedosáhne cíle.

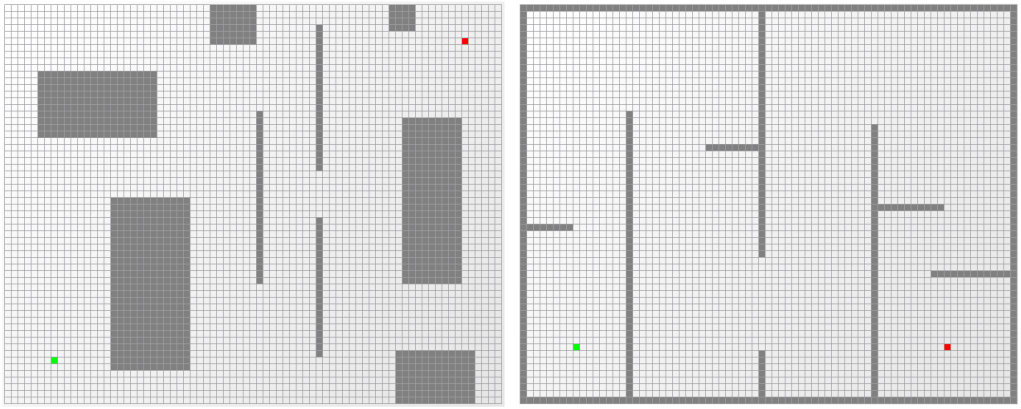
Ač je tento algoritmus poměrně jednoduchý, zejména pro mapy typu bludiště případně cesty s mnoha větvemi, je poměrně efektivní. Nespornou výhodu zjednodušení cesty na několik málo přímek (vzhledem k původní cestě) je výsledný menší nárok na manévrování robotu – nemusí tak často zatáčet.

6.5.3 Implemetace módu měření

Mód pro dávková měření je implementován jako vlákno, které spouští vlákna jednotlivých měřených stromů a čeká na jejich doběhnutí. Tak bylo možno dosáhnout plynulého chodu měření bez nepříjemného zamrzání GUI nebo jiných komponent.

7 VZOROVÁ MĚŘENÍ

Následující kapitola prezentuje výsledky vzorových měření provedených pomocí dávkového módu aplikace *RRT Explorer*. Srovnává jedno-stromovou verzi RRT algoritmu s dvou-stromovou a přednáší výsledky optimalizace *Crop factoru*. Měření byla provedena na dvou vzorových mapách, jedné typu otevřeného prostoru s několika překážkami a druhé typu bludiště. Obě mapy jsou velikosti 75 krát 60 čtverečků. V praxi optimální nastavení řešiče závisí na hustotě překážek mapy.



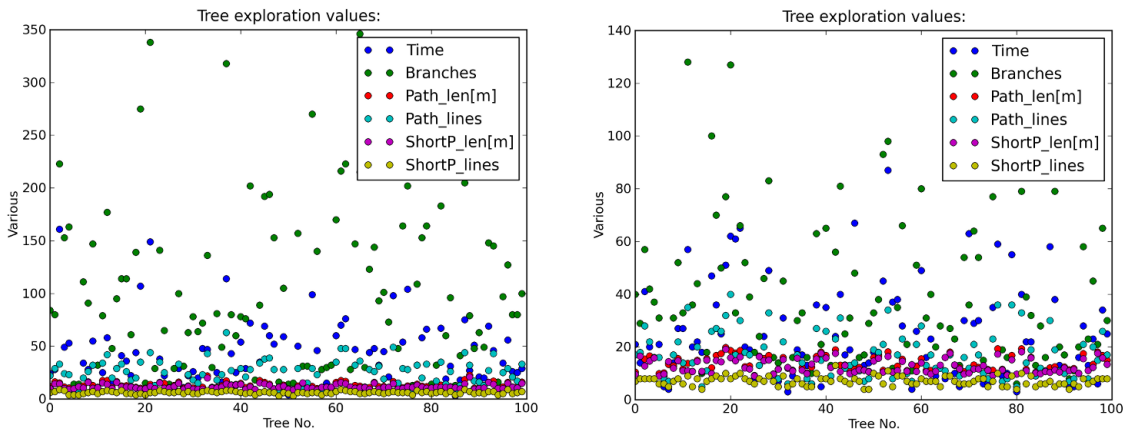
Obr. 35 Vzorové mapy měření – otevřený prostor (vlevo) a bludiště (vpravo).

7.1 Vzorové měření Single vs. Bidirectional

Na obou mapách časově vítězí dle předpokládání *Bidirectional tree* – RRT s dvou-stromovým prohledáváním. V případě délky cesty je jen těsně vítězem *Single tree*.

Srovnání průměrných hodnot s odchylkami pro mapu typu *otevřený prostor:*

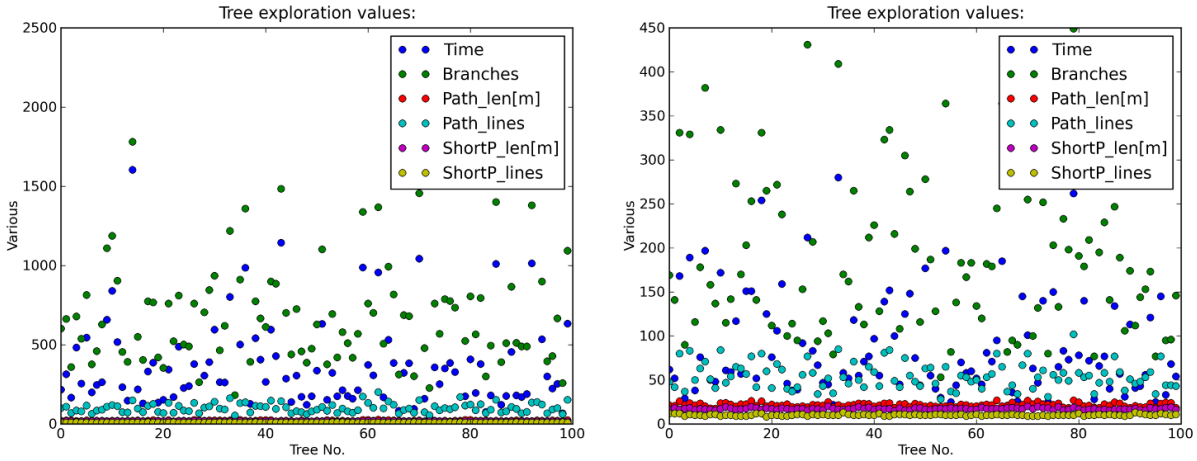
Algoritmus	**Čas [ms]**	**Počet větví**	**Délka cesty [m]**	**Větví cesty**
Single RRT	39,55 ± 35,11	103,52 ± 76,33	12,82 ± 2,67	23,54 ± 11,86
BidirectionalRRT	19,00 ± 17,68	40,85 ± 28,87	13,57 ± 2,86	16,84 ± 8,32



Obr. 36 Mapa typu otevřený prostor – Single Tree vlevo, Bidirectional tree vpravo.

Následuje stejné srovnání průměrných hodnot a odchylek pro mapu typu *bludiště:*

Algoritmus	**Čas [ms]**	**Počet větví**	**Délka cesty [m]**	**Větví cesty**
Single RRT	393,84 ± 356,18	712,09 ± 76,33	21,07 ± 2,67	101,84 ± 29,66
BidirectionalRRT	87,75 ± 56,18	188,93 ± 28,87	21,60 ± 2,13	54,58 ± 14,81

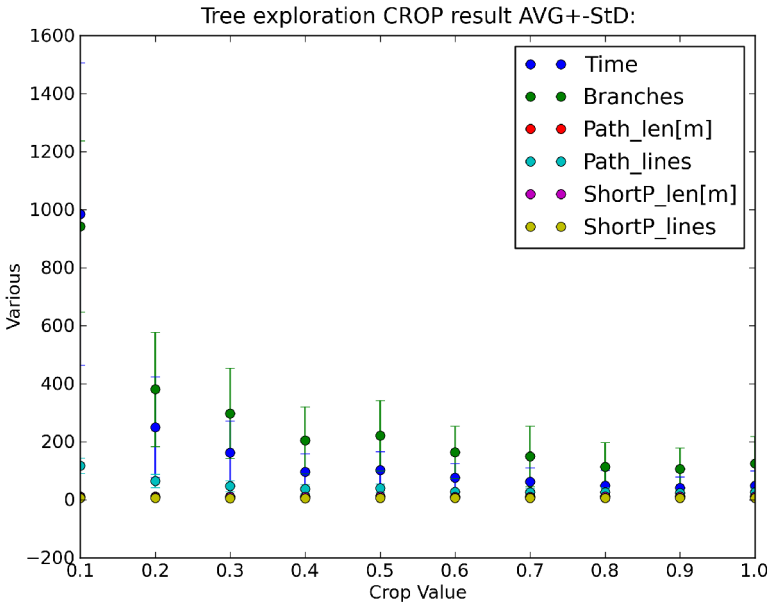


Obr. 37 Mapa typu bludiště – Single Tree vlevo, Bidirectional tree vpravo.

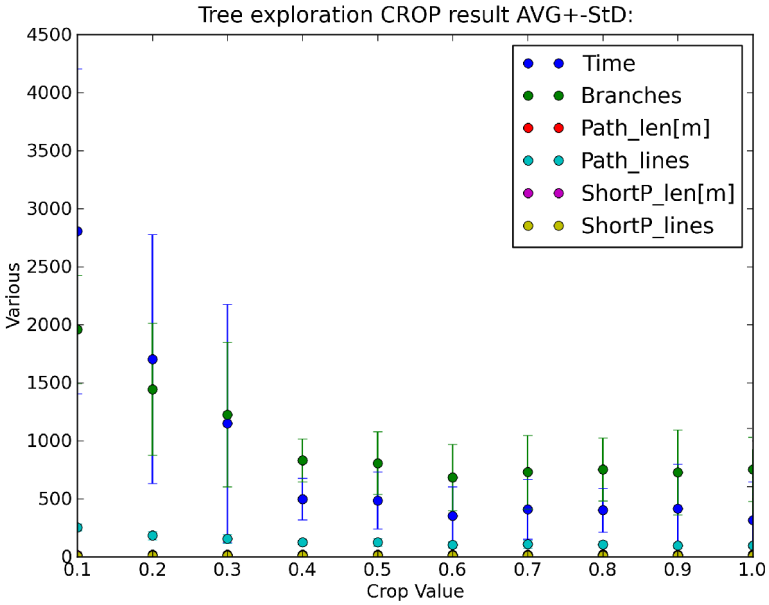
Všechna měření byla opakována 100x.

7.2 Vzorové měření Crop factoru

Ve vzorovém měření si dále můžeme ukázat příklad optimalizace *Crop factoru* pro *Single RRT* a *Crop factor* od *0,1* do *1,0* jdoucí po kroku *0,1*. Výsledky si můžete prohlédnou na Obr. 38 a Obr. 39. Měření byla opakována *100x* pro každý crop.



Obr. 38 Ideální hodnota Crop factoru pro vzorovou mapu typu otevřený prostor a algoritmus Single RRT se nachází kolem hodnoty 0,9.



Obr. 39 Ideální hodnota Crop factoru pro vzorovou mapu typu bludiště a algoritmus Single RRT se nachází kolem hodnoty 0.6.

Všechny hodnoty kromě počtu iterací a nastavení Crop factoru byly ponechány v defaultních hodnotách. Podrobnější měření s rozsáhlejšími závěry jsou již mimo rozsah této práce.

8 ZÁVĚR

Tato bakalářská práce se zabývá plánováním cesty všesměrového mobilního robotu pomocí algoritmu RRT – Rychle rostoucích náhodných stromů.

V první části práce přednáší čtenáři základní pojmy z oblasti, přibližuje problematiku plánování cesty a reprezentace prostředí. Následuje popis a rozdělení základních algoritmů plánování cest. Zde jsou představeny z oblasti informovaného hledání cesty metody rastrové a dekompoziční, metody map cest nebo pravděpodobnostní. Z oblasti neinformovaného hledání cesty popisuje Bug algoritmy. Dále práce podrobně informuje o samotném algoritmu RRT, o variantách jeho provedení (jedno-, dvou- a případně i více stromové struktuře), možnostech a příkladech využití.

Druhá část se zabývá popisem funkcionality, ovládání a implementace linuxové aplikace RRT Explorer 1.0, která je hlavním výstupem práce. Cílem bylo vytvořit program s kvalitním grafickým uživatelským rozhraním, který umožní zkoumání algoritmů RRT pomocí parametrizovatelného řešiče. Součástí programu je také pohodlný editor map „světa“ a dávkový režim pro provádění statistických měření. Aplikace dále nabízí zahrnutí opravných algoritmů pro zefektivnění navržené trajektorie a široké možnosti výstupu – jak v podobě obrázků, tak v podobě textových souborů a neposlední řadě také interaktivních grafů. Součástí programu je handbook – nápověda v podobě krátké e-dokumentace.

Implementovaná knihovna pro RRT je obecně připravena řešit úlohy, kde na vstupu jsou volný prostor a překážky popsány pomocí přímek a start s cílem definovány pomocí bodů. Proto nabízí mnohem univerzálnější přístup, než jak jej předkládá aplikace RRT Explorer. Potenciál do budoucna můžeme vidět v implementaci lepšího editoru světa (editovat překážky jako polygony), zahrnutí stupňů volnosti složitého robotu nebo převodu problematiky do světa 3D.

Na konci druhé části jsou k nahlédnutí výsledky vzorových měření, které potvrzují, že dvou-stromové vyhledávání je efektivnější než jedno-stromové a ukazují příklad optimalizace Crop factoru pro danou mapu.

V neposlední řadě bylo osobním cílem autora se díky této bakalářské práci zdokonalit v jazyce C++ a detailně se seznámit s frameworkem Qt.

SEZNAM POUŽITÉ LITERATURY

[1] Přispěvatelé Wikipedie, *Robot* [online], Wikipedie: Otevřená encyklopedie, c2010, Datum poslední revize 16. 05. 2010, 01:19 UTC, [citováno 22. 05. 2010] <http://cs.wikipedia.org/w/index.php?title=Robot&oldid=5347744>.

[2] ZHUANG Hui-zhong, DU Shu-xin, WU Tie-jun. On-line real-time path planning of mobile robots in dynamic uncertain environment, *Journal of Zhejiang University - Science A* , 2006, roč. 7, č. 4, s. 516-524. ISSN 1862-1775.

[3] KOLOMAZNÍK, J. *Implementace A* algoritmu na konkrétní problém orientace v prostoru budov* [online]. 2005, Datum poslední revize 28. 11. 2005, [citováno 22. 5. 2010]. <http://nlp.fi.muni.cz/uui/referaty2005/kolomaznik_jan/Implementace%20A%20Star%20-%20text.pdf>.

[4] LAVALLE, Steven M. *Planning Algorithms*. Cambridge University Press, 2006. 1007 s. ISBN 0-521-86205-1.

[5] WINKLER, Z. *Plánování na mřížce* (Robotika.cz > Průvodce) [online], 2003. Datum poslední revize 12. 3. 2003 , [citováno 23. 5. 2010]. <http://robotika.cz/guide/gridplan/cs>.

[6] DLOUHÝ, M. *Exaktní plánování* (Robotika.cz > Průvodce) [online], 2003. Datum poslední revize 7. 11. 2003, [citováno 23. 5. 2010]. <http://robotika.cz/guide/exactplan/cs>.

[7] Přispěvatelé Wikipedie, *Voronoi diagram* [online], Wikipedia: The Free Encyclopedia, c2008, Datum poslední revize 3. 06. 2008, 07:54 UTC, [citováno 23. 05. 2010] <http://cs.wikipedia.org/w/index.php?title=Voronoi_diagram&oldid=2647314>

[8] DLOUHÝ, M. *Pravděpodobnostní plánování* (Robotika.cz > Průvodce) [online], 2003. Datum poslední revize 5. 12. 2003, [citováno 23. 5. 2010]. <http://robotika.cz/guide/probplan/cs>.

[9] CHOSET, H. M.; HUTCHINSON, S.; LYNCH, K. M.; aj. *Principles of Robot Motion*. The MIT Press, 2005. 150 s. ISBN 0-262-03327-5

[10] ASSAF, M. *Robot navigation project*. (Center for Intelligent System: Computer Science Departament) [online], [citováno 23.5.2010]. <http://isl.cs.technion.ac.il/index.php/undergraduate/done/robot-navigation-project>.

[11] DLOUHÝ, M. *Bug algoritmy* (Robotika.cz > Průvodce) [online]. 2003. Datum poslední revize 7. 11. 2003, [citováno 23. 5. 2010]. <http://robotika.cz/guide/bug-alg/cs>.

[12] Přispěvatelé Wikipedie, *Radpidly-exploring random tree* [online], Wikipedia: The Free Encyclopedia, c2010, Datum poslední revize 12. 2. 2010, 02:09 UTC, [citováno 23. 05. 2010] <http://cs.wikipedia.org/w/index.php?title=Robot&oldid=5347744>.

[13] LAVALLE, Steven M., *Rapidly-exploring random trees: A new tool for path planning*. Computer Science, 1998. Dept, Iowa State University, Tech. Rep. TR: 98–11.

[14] YERSHOVA, A. Dynamic-Domain RRTs: Efficient Exploration by Controlling the Sampling Domain. *Proceedings IEEE International Conference on Robotics and Automation*, 2005.

[15] Přispěvatelé Wikipedie, *Qt* [online], Wikipedie: Otevřená encyklopedie, c2006, Datum poslední revize 2. 08. 2006, 16:06 UTC, [citováno 28. 05. 2010] <http://cs.wikipedia.org/w/index.php?title=Qt&oldid=655026>.

Obsah CD

BPLukasKnispel.odt – zdrojový soubor dokumentu.
BPLukasKnispel.pdf – dokument ve formátu pro prohlížení a tisk.
RRTExplorer.mpeg – demonstrační video aplikace.
RRTExplorer.zip – archiv se samotným programem RRT Explorer 1.0.
Sources.zip – zdrojové kódy programu (zaheslovaný archiv).