VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

PLÁNOVÁNÍ CESTY PRO AUTONOMNÍ VYSAVAČE

THESIS TITLE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN HRANICKÝ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Hranický Jan**
Program: Informační technologie
Název: **Plánování cesty pro autonomní vysavače**
Path Planning for Autonomous Vacuum Cleaners
Kategorie: Umělá inteligence

Zadání:

1. Nastudujte problematiku autonomních vysavačů a jejich senzory a programové vybavení. Nastudujte algoritmy pro plánování cesty. Zaměřte se na algoritmy používané pro kompletní pokrytí daného prostoru (tzv. coverage path planning).
2. Navrhněte program, který umožní vytvořit nebo z externího souboru nahrát tvar oblasti a pro ni vytvořit plán cesty. Počítejte s HW omezeními senzorů reálného vysavače.
3. Navržený program implementujte jako aplikaci s grafickým uživatelským rozhraním. Otestujte jeho funkčnost a navrhněte případná vylepšení.

Literatura:

- Howie Choset et al., Principles of Robot Motion, 2005, ISN 0-262-03327-5.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Algoritmy kompletního pokrytí prostoru plánují takovou cestu skrze daný prostor, že jsou všechny body prostoru navštíveny nejméně jednou. Tato práce se zaměřuje na plánování cesty kompletního pokrytí pro autonomní robotické vysavače. Práce se zabývá aktuální nabídkou robotických vysavačů dostupných na trhu. Především se pak zaměřuje na samotné algoritmy kompletního pokrytí prostoru. V poslední části jsou praktické poznatky z rešerše uplatněny spolu s teoretickými popisy algoritmů při implementaci několika z nich v knihovně vizlib.

Abstract

Complete coverage path planning is a task of finding such path that passes over all points in given area at least once. This thesis applies complete coverage path planning to autonomous vacuum cleaner robots. Thesis summarises current deal of autonomous vacuum cleaner robots. But mainly it focuses on describing complete coverage algorithms. Selected algorithms are implemented using Java vizlib library.

Klíčová slova

plánování cesty, kompletní pokrytí, vizlib, java, vizualizace, autonomní robotický vysavač

Keywords

path planning, complete coverage, vizlib, java, vizualization, autonomous vacuum cleaner

Citace

HRANICKÝ, Jan. *Plánování cesty pro autonomní vysavače*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Plánování cesty pro autonomní vysavače

Prohlášení

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

.....
Jan Hranický
11. května 2021

Poděkování

V první řadě bych chtěl poděkovat svému vedoucímu, panu Ing. Jaroslavu Rozmanovi, Ph.D., za nasměrování k tématu a jeho připomínky.

Dále bych chtěl poděkovat své rodině a nejbližším přátelům, Světlaně, Tereze, Zuzaně a Radkovi, za pomoc při překonání těchto težkých časů.

Obsah

Kapitola 1

Úvod

Kompletní pokrytí prostoru je problém, krerý řeší jak pokrýt každou část prostoru co nejefiktivněji. Autonomní robotické vysavače musí tento problém řešit za běhu (online) v neznámém prostředí.

Cílem práce je navrhnout a vytvořit program, který bude plnit funkci simulátoru pohybu autonomních vysávacích robotů. V simulátoru bude uživatel moci vybrat z několika plánovacích algoritmů. Robot se bude pohybovat v uživateli předdefinovaném prostředí, to uživatel vytvoří v editoru, nebo nahraje ze souboru. Po spuštění bude program simulovat pohyb autonomního vysavače v daném prostoru a plánování jeho kompletního pokrytí.

První část práce se zaměřuje na konkrétní robotické vysavače. Druhá kapitola je věnována rešerši, jejíž účel je zjistit jakými technologiemi současné robotické vysavače disponují. Rešerše se převážně zaměřuje na princip navigace, senzory a programové vybavení jednotlivých robotů. Třetí kapitolu shrnuje poznatky rešerše.

Druhá část práce se zaměřuje na plánovací algoritmy kompletního pokrytí prostoru. Ty jsou popsány v čtvrté kapitole.

Poslední částí práce je implemntace vybraných algoritmů. Pátá kapitola se věnuje představení knihovny, vizlib Jakuba Rusnáka [14], s pomocí které je práce napsaná. Poslední, šestá, kapitola se věnuje vytvořené aplikaci.

Kapitola 2

Robotické vysavače

Tato kapitola je věnována konkrétním firmám, které nabízí modely robotických vysavačů. U každého modelu jsou vypsány funkce, jimiž disponuje. Zároveň u jednotlivých výrobců uvádím dodatečné doplňky, které k robotům nabízí.

Informace o aktuálních technologíích použítych v moderních robotech jsou převzaty z oficiálních stránek jednotlivých výrobců [12], [19], [5], [18]. Seznam jednotlivých výrobců byl převzat z e-shopu [1].

2.1 iRobot

Společnost iRobot [12] je jednou z nejstarších společností na trhu, které nabízejí autonomní robotické vysavače. Nabízí hned několik sérií robotů, kdy se každá série pohybuje v jiné cenové kategorii.

2.1.1 iRobot Home

iRobot Home je mobilní aplikace vyvinutá společností iRobot. Lze s ní spárovat vybrané roboty Roomba. Po spárování aplikace s robotem zjednodušuje aplikace jeho ovládání. iRobot na svých stránkách [12] vyzdvihuje jednotlivé funkce aplikace, mezi které patří tipy a doporučení na plánování úklidu a také samotný plánovač, ve kterém lze naplánovat úklid na konkrétní čas. Robotický vysavač tak sám započne úklid.

2.1.2 Virtuální zeď

Všichni roboti Roomba jsou schopni komunikovat s virtuální zdí, toto zařízení si uživatel může dokoupit a umístit do prostoru, do kterého robot nemá zasahovat.

Virtuální zeď vysílá signál směrem rovně a tvoří tak zeď. Zařízení je možno umístit do dveří a robot tak místnost nevyčistí.

Druhým režimem virtuální zdi je režim Halo. V tomto režimu zařízení vysílá signál kolem sebe v kruhu o průměru cca 60cm.

2.1.3 Roomba® 2v1

Tato série robotů obsahuje pouze jednoho robota, a to Roomba Combo, který je schopen, jak již název napovídá, podlahu nejen zamést ale i vytřít. Liší se tak od ostatních robotů. Cenově vyjde na 7 999 Kč.

Robota je možno spárovat s technologií Smart Home a je ho možno ovládat pomocí asistenta s umělou inteligencí společnosti Amazon Alexy nebo Google Home společnosti Google. Navíc je možno také Roomba Combo spárovat s aplikací iRobot Home, v níž lze navíc nastavit intenzita vody, kterou Roomba Combo při vytírání použije.

Roomba Combo se pohybuje v dvou režimech pohybu, v tzv. přímočarém, nebo reaktivním režimu. Přímočarý režim má za cíl uklízet systematicky v řádcích, zatímco reaktivní režim se postará o úklid celé místnosti.

V balení robota se také nachází dokovací stanice do které se robot automaticky sám po skončení úklidu vrátí, a ve které se automaticky dobije jeho baterie.

Co se týče samotného čištění, má Roomba Combo na boku umístěny dva kartáče, které smetou nečistoty směrem do středu robota, kde jsou poté nasáty do sběrného koše. Vytírání je zajištěno pomocí čisticí podložky na robotově podvozku, ta je namáčena elektrickým čerpadlem vody.

Oficiálně iRobot uvádí, že Roomba Combo disponuje klasickou dvojicí senzorů nárazník a protipádový senzor. Nárazník je navíc patentován technologií Lite Touch ™.

2.1.4 Roomba® série 600

Roomba® sdružuje tři generace robotů, a to Roomba 606, Roomba 698 a Roomba 676.

iRobot uvádí, že roboty série 600 využívají technologii iAdapt® 1.0, kterou zlehka popisuje na jejich stránkách zákaznické podpory [11]. Z poskytnutého popisu to vypadá, že iAdapt® 1.0 je obdobu náhodného pohybu 4.1. Na obrázku 2.1 lze vidět cestu opsanou vysavačem s technologií iAdapt® 1.0, vizuálně připomíná trasu vysavače při algoritmu náhodného pohybu.

Všechny roboty série 600 mají dvojici senzorů nárazník a senzor proti pádu ze schodů.

Obrázek 2.1: Převzato z https://homesupport.irobot.com/euf/assets/images/faqs/roomba/19541/iadapt1.png

Roomba 606

Roomba 606 je první generací série 600, cenově je dostupný za 5 999 Kč. Vysavač nelze spárovat s mobilní aplikací iRobot Home, uživatel musí tak robota ovládat skrz tlačítka na jeho vrchní části.

V balení je také základna robota, do které se po dokončení úklidu robot sám vrátí a dobije. Roomba 606 najde základnu pomocí IR signálu, který základna vysílá. V případě,

že robot nezačal úklid ze základny, nebo signál nemůže najít, skončí a zastaví se na místě, kde úklid skončil.

Roomba 606 má jeden rotující boční kartáč, který smete nečistoty do středu robota. Zároveň má dvojici kartáčů na podvozku, které smetou nečistoty přímo pod robotem. Nečistoty jsou poté nasáty do sběrného koše. Robot navíc disponuje funkcí Dirt detect, která za pomocí akustického senzoru detekuje nečistoty a robot tak intenzivněji vysává. Další funkce, kterou robot disponuje, je funkce Antiangle, která zabrání nasátí kabelů. Kartáče při odporu otočí svou rotaci, robot se uvolní z kabelů a může pokračovat v úklidu.

Roomba 676

Roomba 698 je druhou generací série 600, má všechny funkce jeho předchůdce, navíc ale umožňuje spárovat robota s aplikací iRobot Home. Cenově Roomba 698 vychází na 5 999 Kč.

Roomba 698

Jediný rozdíl, který jsem našel mezi Roomba 698 a starší Roomba 676, je čisticí systém Aero Force, který Roomba 698 nabízí. Cenově robot vychází na 6 999 Kč.

2.1.5 Roomba® E

Série E sdružuje 2 generace robotů Roomba e5 a Roomba e6. Cenově vychází na 7 999 Kč a 8 999 Kč.

Rozdíly mezi předešlou sérií jsou minimální, nebudu tedy rozdělovat roboty do sekcí.

Série E využívá čisticího systému AeroForce. Roomba e6 navíc disponuje funkcí plného koše, při přeplnění sběrného koše se rozsvítí kontrolka a uživatel je tak o přeplněném koši informován.

2.1.6 Roomba® 900

V sérii Roomba 900 je jen jeden robot, Roomba 976. Lze ho pořídit za 9 999 Kč. Velkou změnou od předchozích sérií je navigace iAdapt® 2.0, která využívá kameru umístěnou na vrchní straně robota. Kamera je umístěna pod úhlem a snímá překážky před robotem, ten si s pomocí pořízených snímků v paměti tvoří mapu prostoru, tu ale neukládá a je při každém úklidu tvořena znova.

iRobot opět neuvádí celkový výčet senzorů, nicméně existují články jako [7], ve kterých lidé roboty rozeberou a senzory zkoumají sami. Článek uvádí, že je série 900 kromě klasických senzorů obdařena senzorem, který snímá podlahu. S jeho pomocí si Roooma 976 měří ujetou vzdálenost a mapuje tak prostor okolo sebe.

Roomba 900 sleduje stav své baterie a v případě, že se při úklidu dostane na kritickou hodnotu, vrátí se zpět do stanice dobít. Po dobití opět pokračuje tam, kde přestala. Tato funkce je umožněna díky pokročilejší navigaci.

2.1.7 Roomba® i

Roomba série i obsahuje dva roboty Roomba i7 a Roomba i7+ v cenové kategorii 14 999 Kč a 19 999 Kč.

Obě generace využívají třetí stupeň technologie iAdapt a to iAdapt® 3.0. Od předchozí verze umožňuje iAdapt 3.0 vytvářet perzistentní mapy okolí. To přináší další benefity, jako je výběr tzv. keep out zones v mobilní aplikaci, do kterých robot při úklidu nezasahuje.

Oba modely jsou schopny komunikovat s dokovací stanicí Clean Base®, která robotům umožňuje automaticky vyprázdnit jejich sběrný koš, automatizace úklidu je tak zase o něco vyšší. Rozdíl je ten, že Roomba i7+ obsahuje tuto stanici v základním balení.

2.1.8 Roomba® s

Roomba série s je momentálně nejmodernější sérií nabízenou společností iRobot. Stejně jako její předchůdce série obsahuje dva roboty Rooba s9 a Roomba s9+, cenově vycházejí na 29 999 Kč a 39 999 Kč.

Série s používá stejný typ navigace jako její předchůdce, iAdapt® 3.0. Má však vylepšenou funkci mapování pomocí technologie Imprint®, která se postupně naučí poznávat jednotlivé místnosti. Robot je schopen si v paměti udržet až deset různých map, například různá patra domu.

Novinkou je také fukce Power Boost, robot je schopen rozeznat povrch, po kterém se pohybuje. Jakmile se nachází na koberci, zvýší vysávací výkon a spolehlivěji tak koberec vyčistí.

Roomba s9 a Roomba s9+ se liší pouze v tom, že Roomba s9+ obsahuje stanici Clean Base® již v základním balení.

Série S se také liší ve svém tvaru, který ji umožňuje lepší čistění rohů viz. 2.2.

(a) Nový tvar série s (b) Starý tvar série i

Obrázek 2.2: Obrázky robotických vysavačů jsou převzaty z [12]

2.2 Symbo

Druhým výrobcem autonomních robotických vysavačů, kterého do své rešerše zahrnu, je Symbo [19]. Stejně jako iRobot i Symbo nabízí několik sérií robotů.

2.2.1 Symbo D300

Symbo D300 je první generací robotů, kterou Symbo nabízí. Doporučená cena prodeje Symbo D300 je 2 799 Kč.

Kromě vysávání plní i funkci mopu, avšak pouze za pomocí prachovky umístěné na podvozku robota.

Součástí balení není dobíjecí stanice a robot musí být dobíjen ručně pomocí adaptéru.

Stejně jako první generece robotů Roomba, disponuje Symbo D300 pouze dvojicí senzorů, protipádový senzor a nárazník.

Symbo neuvádí způsob navigace, pomocí kterého se Symbo D300 pohybuje, s největší pravděpodobností to ale bude obdoba náhodného pohybu 4.1.

2.2.2 Symbo D400

Série D400 obsahuje dva roboty, Symbo D410 a Symbo D420, cenově se oba prodávají za 5 999 Kč. Kromě barvy jsem nenašel v modelech rozdíl, dále je proto budu označovat souhrně jako sérii D400.

Série D400 vylepšuje funkci mopování o nádobku na vodu, kterou robot při úklidu použije.

Dalším vylepšením je dobíjecí stanice, do které se robot po úklidu vrátí a sám se dobije.

Symbo uvádí, že je série 400 schopna vykonávat 3 čisticí režimy, neuvádí však jejich popis. Usuzuji, že se nejspíše jedná o obdobu algoritmu náhodného pohybu v podobě pohybu kolem zdi a dalších pohybů, viz 4.1.

2.2.3 Symbo xBOT 5

Další generací robotů Symbo je série XBOT 5, do které patří dva roboti, xBOT 5 a xBOT 5 Pro, cenově jsou roboti dostupní za 7 999 Kč a 8 999Kč.

Série přináší vylepšenou navigaci Smart GYRO. Robot využívá pro pohyb v prostoru senzory a nejspíše si v paměti vytváří nějakou reprezentaci navštíveného prostoru. Smart GYRO navigace přináší hned několik režimů vysávání:

- Automatický režim – Symbo xBOT 5 za pomocí Smart GYRO navigace pokryje co největší prostor.
- Režimy vysávání – Robota lze manuálně přepnout do jednoho z následujících režimů :
  - Edge cleaning – robot vysaje okraje místností a nábytku
  - Local cleaning – robot bude vysávat prostor o rozloze přibližně $2x2$ m
  - Random cleaning – robot nebude využívat Smart GYRO navigaci a bude vysávat náhodně
  - Double cleaning – robot vysaje celý prostor 2x
- Režim MAX – režim, ve kterém Symbo xBOT 5 aktivuje zvýšený sací výkon

Robot Symbo xBOT 5 Pro navíc umožňuje spárování s mobilní aplikací Symbo Home Blue, mobilní aplikace nabizí totožné funkce jako aplikace iRobot Home, nebudu je proto znova vypisovat, viz 2.1.1.

Série xBOT je také schopna nezasahovat do určitých oblastí, ty ale uživatel musí vyznačit magnetickou páskou, kterou je robot schopen rozpoznat.

2.2.4 Symbo LASERBOT

Série LASERBOT je nejnovější sérií, kterou Symbo nabízí. Obsahuje dvě generace robotů, Symbo LASERBOT 650 (10 999 Kč) a Symbo LASERBOT 750 (15 999Kč).

Série LASERBOT disponuje laserovým LiDAR senzorem, ten je využit v Smart LASER navigaci, pomocí které robot systematicky uklízí prostor a zároveň tvoří jeho mapu. Jedná se tedy o SLAM navigaci. Symbo uvádí, že LASERBOT je navíc vybaven gyroskopem a infračervenými senzory.

Symbo LASERBOT 750 má všechny funkce jeho předchůdce, navíc je ho možno propojit s novou verzí mobilní aplikace Symbo Home Orange. Ta vylepšuje funkce předchozí verze Symbo Home Blue a zároveň funkce nové přidává. Uživatel si může zobrazit mapu vytvořenou robotem jak při úklidu, tak po jeho skončení. Díky SLAM navigaci také uživatel může robotovi nastavit zónové čistění, či naopak zakázané oblasti.

2.3 CleanMate

Třetím výrobcem, jehož roboty ve své rešerši shrnu, je CleanMate [5]. Narozdíl od předchozích výrobců společnost CleanMate nabízí pouze tři generace robotů, každá generace navíc obsahuje pouze jednoho robota.

2.3.1 CleanMate RV500

CleanMate RV500 je prvním robotem společnosti CleanMate. Jeho pořizovací cena je 4 999 Kč.

Narozdíl od prvních generací ostatních výrobců je již CleanMate RV500 vybaven gyroskopickou navigací, prostor tedy uklízí systematicky, nikoli náhodně. CleanMate také uvádí, že je RV500 navíc vybaven patnácti infračervenými senzory, s jejich pomocí s největší pravděpodobností měří vzdálenost od překážek. Nechybí ani klasická dvojice senzorů nárazník a protipádový senzor. CleanMate RV500 podporuje uklízecí módy, konkrétně AUTO, SPOT a EDGE.

Čisticí systém je podobný čisticím systémům ostatních výrobců, robot disponuje dvěma kartáči na jeho bocích a jedním na jeho podvozku. Nečistoty tak smete do jeho středu, kde je vysaje do zásobníku. Stejně jako roboti společnosti Symbo má navíc RV500 funkci mopu. Mop je umístěn na spodní straně robota a je automaticky namáčen vodou v nádobce, systém dávkování je pouze mechanický a robot vodu nedávkuje za pomocí elektrického dávkovače.

Po skončení úklidu se CleanMate RV500 vrátí zpět do nabíjecí stanice, která je součástí balení.

Nechybí ani mobilní aplikace, se kterou RV500 komunikuje prostřednictvím Wi-Fi. V aplikaci může uživatel plánovat úklid dopředu.

2.3.2 CleanMate QQ6SLi

CleanMAte QQ6SLi je robot druhé generace společnosti CleanMate. Vylepšuje funkce a možnosti jeho předchůdce. Je možné sehnat ho za 5 999Kč.

Co se týče navigace, bude nejspíš CleanMate QQ6SLi používat podobný způsob jako jeho předchůdce RV500, avšak používá jiné senzory. Na rozdíl od RV500 je ale vybaven ultrazvukovými senzory.

CleanMate QQ6SLi již není vybaven mopem. Je schopen ale pokrytý prostor dezinfikovat pomocí UV lampy.

Série QQ6SLi je schopna komunikovat se sonickou zdí, kterou uživatel umístí do prostoru a robot jí neprojede.

2.3.3 CleanMate LDS700

Posledním modelem nabízeným společností CleanMate je LDS700, robot je dostupný za 8 999 Kč.

Model má vylepšenou navigaci SLAM, při které využívá laserový systém LiDAR. Při úklidu tvoří mapu pokrytého prostoru, s tou může uživatel dále pracovat v mobilní aplikaci. Stejně jako u ostatních robotů využívajících SLAM navigaci, může uživatel robotovi přikázat zónové čištění, či zakázané oblasti.

LDS700, stejně jako QQ6SLi, disponuje funkcí mopu, který je schopen sám navlhčit pomocí elektrického dávkovače vody.

Díky vylepšené navigaci je LDS700 schopen při nízké úrovni baterie dojet do stanice, dobít se a poté pokračovat v úklidu z předešlého místa.

LDS700 také detekuje povrchy, po kterých se pohybuje, při pohybu po koberci automaticky zvýší sací výkon a koberec tak lépe vyčistí.

2.4 Samsung

Samsung [18] je posledním výrobcem, kterého do rešerše zahrnu. Nabízí čtyři série robotů.

2.4.1 POWERbot™ VR5000

Série POWERbot™ VR5000 je první sérií první generace, obsahuje pouze jednoho robota s označením VR05R5050WK. Cenově se série prodává za 9 990 Kč.

Podobně jako robot první generace společnosti CleanMate 2.3 se i série VR5000 nepohybuje náhodně. Robot disponuje gyroskopem, protipádovým senzorem a nárazníkem. Taktéž série VR5000 podporuje uklízecí módy, do kterých může robota uživatel manuálně přepnout. VR5000 podporuje následující módy :

- Zig-Zag – robot uklízí ve tvarech písmena 'S'
- Edge – úklid okrajů místnosti
- Spot – intenzivnější úklid znečištěného místa
- Auto – náhodný pohyb

Čisticí systém VR5000 tvoří dva kartáče na bocích, které smetou nečistoty směrem dovnitř robota, kde je další kartáč smete a nasaje do sběrného koše. VR5000 navíc disponuje funkcí mopu, který je namáčen z nádobky s vodou.

Série také obsahuje v balení nabíjecí stanici, ale informace zda se po úklidu robot do stanice vrátí, není dostupná.

VR5000 lze spárovat s mobilní aplikací POWERbot-E.

2.4.2 POWERbot™ VR7000M

Série POWERbot™ VR7000M taktéž obsahuje pouze jednoho robota, a to VR10M702CUW, cenově je robot dostupný za 16 490 Kč.

Nová série přináší vylepšenou navigaci, která využívá technologií FullView Sensor™ 2.0 a Visionary Mapping™ +. Robot má pomocí vylepšených senzorů překážek lepší přehled o tom, kde se nachází. Své okolí navíc mapuje pomocí kamery.

Čisticí systém tvoří jeden větší kartáč, který smete nečistoty směrem dovnitř. Robot nemá kartáče na boku, které předešlé sérii pomáhaly zvládat úklid okrajů. Tuto funkci zajišťuje pomocí uzávěrky, která se vyklopí, jakmile se robot pohybuje kolem zdi. Tím je zvýšen sací výkon. VR7000M je schopna rozeznat povrch, po kterém se pohybuje. Na kobercích tak zvýší sací výkon.

Easy Pass™ je další technologií, kterou VR7000M disponuje. Jedná se o vylepšenou funkci koleček, ty se po kontaktu s překážkou zvednou a robot je tak schopen překonat prahy. Také je schopen lépe se dostat na tlustší koberce.

Díky navigační technologii je robot schopen během úklidu sám vyhledat nabíjecí stanici, sám se dobít a pokračovat v úklidu. Nabíjecí stanice je součástí balení.

Nechybí ani mobilní aplikace, která ale umožňuje uživateli pouze vybírat jeden z módů a úklid plánovat. Aplikace neobsahuje mapu vytvořenou robotem, zakázané zóny musí proto uživatel vyznačit fyzicky pomocí zařízení Virtual Guard, které vysílá do daného místa infračervený paprsek, jenž robot rozpozná a nezasahuje do něj.

2.4.3 POWERbot™ VR7200

Série VR7200 obsahuje robota s označením VR10R7220W1/GE, prodává se za 17 990 Kč.

VR7200 disponuje stejnými funkcemi jako předchozí série VR7000M. Navíc ale vylepšuje čisticí systém o kartáč Soft Action, který je podle Samsungu schopen podlahu lépe vyčistit. Zároveň čisticí systém obsahuje drtiče, které kartáč v průběhu úklidu čistí, v kartáči tak nezůstávají vlasy a chlupy.

Druhým vylepšením oproti předchozí generaci je funkce Point Cleaning™. Robotův ovladač má v sobě zabudovaný laser, pomocí kterého může uživatel robotovi ukázat na oblast, kterou má vyčistit.

2.4.4 POWERbot™ VR7200R

VR7200R je poslední generací vysávacích robotů Samsung. Spadá do ní robot s označením VR20R7250WC, který se prodává za 23 990 Kč.

Série pouze vylepšuje funkce předešlých sérií. Navíc RV7200R disponuje senzory podlahy a sací výkon tak narozdíl od předchozích sérií upraví i na tvrdé podlaze.

Kapitola 3

Shrnutí rešerše

V této kapitole je uvedeno shrnutí rešerše z kapitoly 2. Shrnutí je rozděleno do několika částí. V první části jsou shrnuty senzory robotů 3.1, které jsou rozděleny podle způsobu použití. V druhé části 3.2 jsou uvedeny způsoby navigace, které roboti používají a k nim příslušné senzory. V poslední části 3.3 je shrnuto dodatečné vybavení, to je rozděleno na programové vybavení a doplňky, které si uživatel může k robotovi dokoupit.

3.1 Senzory

Senzory, které robot využívá, jsou setříděny do kategorií podle způsobu použití. Senzory, které jsou využívány při navigaci, se liší podle způsobu navigace, a jsou proto popsány u jednotlivých metod v kapitole 3.2. Popis senzorů a jejich využití je převzat z [16],[6].

3.1.1 Senzory pro detekci překážek

Data získaná ze senzorů umožňují robotům orientovat se v prostoru. Autonomní robotické vysavače dostupné na trhu jsou obdařeny řadou senzorů, které se liší podle výrobce robota a taky série, do které robot patří. Každý vysavač ale disponuje následující dvojicí senzorů, které mu umožňují detekovat překážky:

1. Senzor proti pádu ze schodů – za pomocí tohoto senzoru robot detekuje okraj schodů, který při detekci bere okraj jako překážku, a tak ze schodů nespadne.

2. Protinárazové senzory – díky protinárazovému senzoru je robot schopen detekovat okraj překážek a nenarazí do nich. Typicky má robot mechanický nárazník, který se sepne po nárazu s překážkou. Novější modely robotů jsou navíc obdařeny senzory, které měří vzdálenost od překážek. Robot je tak schopný snížit svou rychlost před nárazem do překážky.

3.1.2 Senzory úklidu

Další kategorie senzorů, kterými roboti disponují, jsou senzory úklidu. S jejich pomocí robotické vysvače detekují nečistoty a jsou schopny intenzivněji uklidit znečištěnou oblast.

Obvykle jsou roboti obdařeni senzory, které jsou schopny detekovat samotné nečistoty. Například někteří roboti společnosti iRobot mají pro tento účel ultrazvukové senzory.

Druhým typem senzorů, které jsou v robotech používány, jsou senzory podlahy. Robot je tak schopen detekovat měkké povrchy, například koberce, na kterých automaticky zvýší sací výkon.

3.1.3 Senzor pro komunikaci s dokovací stanicí

Dokovací stanice je místem, kde robot tráví čas, pokud zrovna neuklízí. Stanice také umožňuje automatické dobití robota. Nejmodernější stanice, například stanice robota Roomba i7+ [13], jsou navíc schopny automaticky vysypat robotův sběrný koš.

Robot se stanicí nejčastěji komunikuje za pomocí dvojice infračervených senzorů, jeden detekuje signál vysílaný dokovací stanicí a druhý pro samotný dokovací proces.

3.2 Způsoby navigace

Sekce obsahuje výčet způsobů navigace, které autonomní robotické vysavače používají při svém pohybu. Sekce je rozdělena na tři části, každá popisuje jeden způsob navigace. Způsoby jsou navíc setříděny od nejjednodušího až po ten nejkomplexnější.

3.2.1 Náhodný pohyb

Náhodný pohyb je nejstarším způsobem používaným v robotických vysavačích. Z hlediska senzorů není vůbec náročný, stačí když robot disponuje senzory, které detekují náraz. Tyto senzory jsou již popsány v sekci 3.1.1 a jsou dnes již základem, disponuje jimi každý robot. Náhodný pohyb bývá často doprovázen uklízecími módy. Tyto módy jsou často pohyb kolem zdi a pohyb po spirále. Robot střídá mezi jednotlivými módy a oblast tak lépe pokryje.

3.2.2 Navigace za pomocí senzorů

Druhým způsobem navigace je navigace pomocí dat získaných ze senzorů robota. Robot za pomocí příslušných algoritmů plánuje svou cestu a systematicky pokrývá celý prostor.

Na rozdíl od náhodného pohybu musí být robot schopen rozeznat překážky všude kolem sebe, je proto navíc vybaven sonickým či laserovým prstencem, který mu toto umožňuje. Roboti jsou často také vybaveni gyroskopen, který usnadňuje jeho orientaci v prostoru.

3.2.3 SLAM

Simultaneous **l**ocalization **a**nd **m**apping, dále pouze SLAM, je nejmodernější způsob navigace dostupný na dnešním trhu. Robot si v paměti vytváří přesnou mapu prohledávaného prostoru, pomocí které následně prostor pokryje. Vytváření mapy je nad rámec této práce, budu se tedy věnovat pouze plánovacím algoritmům, které robot použije pro pokrytí již vytvořené mapy.

Pro tvorbu mapy je robot nejčastěji vybaven jedním ze dvou typů senzorů :

LiDAR

Lidar je zkratkou pro **Li**ght **D**etection **A**nd **R**anging, jde o metoda měření vzdálenosti na základě výpočtu doby šíření pulsu laserového paprsku odraženého od snímaného objektu [20]. Robotický vysavač využívající tuto technologii disponuje laserovým snímačem, který se otáčí v úhlu 360° a mapuje tak robotovo okolí. LiDAR používají roboti společnosti Symbo 2.2.

VSLAM

VSLAM je zkratkou pro **V**isual **S**imultaneous **L**ocalization **A**nd **M**apping. Robot má na sobě umístěnou kameru, která snímá strop místnosti. Kamerový systém pořizuje snímky stropu a také objektů, které jsou v jeho zorném poli. Z těchto snímků je v robotově paměti následně vytvořena mapa prostoru. Tento způsob implementace mapování využívají roboti společnosti iRobot 2.1.

Existují i hybridní systémy jako například robot Deebot OZMO 960 firmy Ecovacs [8], který disponuje laserovým LiDAR systémem jakožto hlavním typem navigace a zároveň má přední kamerou, kterou pomocí technologie AIVI™ identifikuje předměty ve své cestě. Je proto schopen vyhnout se lehkým předmětům jako ponožky, kabely, či dokonce exkrementům domácích mazlíčků. Jedná se o poměrně novou technologii uvedenou na trh v roce 2019.

3.3 Vybavení

V této sekci je popsáno vybavení, kterým autonomní robotické vysavače disponují a které vylepšuje uživatelský komfort.

3.3.1 Mobilní aplikace

Mobilní aplikace umožňuje uživatelům robota nejen ovládat, ale také plánovat úklid dopředu.

Starší modely robotů disponují aplikací, která je s robotem spárovaná skrze technologii bluetooth, uživatel proto musí být v blízkosti robota. Uživatel je schopen robotův pohyb ovládat manuálně, za pomocí směrových instrukcí. Většinou také může robotovi nakázat, aby své okolí vyčistil důkladně za pomocí předdefinovaného pohybu. Takovéto aplikace nejsou moc praktické, protože uživatel musí sledovat, co právě robot dělá, a ztrácí se tak robotova autonomnost.

Nynější modely robotů jsou již schopny s aplikací komunikovat skrze bezdrátové připojení Wi-Fi. Uživatel tak má možnost robota ovládat i mimo domov. Takovéto aplikace zpravidla obsahují plánovač úklidu, ve kterém uživatel může plánovat úklid dopředu. Uživatel tak může spustit úklid, i když není doma.

Autonomní robotické vysavače s technologií SLAM navíc mapují své okolí. Mapu prostoru má uživatel k dispozici v mobilní aplikaci, kde s ní může dále pracovat. Typicky z ní může vybírat plochy, které má robot vyčistit důkladněji, nebo naopak vybrat oblasti, do kterých robot při úklidu nebude zasahovat.

3.3.2 Vybavení pro vymezení robotova úklidu

Před příchodem navigace typu SLAM 3.2.3 musel uživatel vymezit plochu, do které nemá robot zasahovat, manuálně. Nejčastěji tuto funkci firmy implementovaly pomocí magnetické pásky nebo sonické zdi, kterou je robot schopen detekovat. Robot tak detekuje zakázanou zónu, do které při úklidu nezasahuje.

Kapitola 4

Plánovací algoritmy

Tato kapitola bude věnována jednotlivým plánovacím algoritmům, které jsou v autonomních vysavačích používány.

Přesné názvy a popisy algoritmů, které jsou používané v dnešních autonomních vysavačích, jsou firemní tajemství. Nicméně z popisu robotů, jejich senzorů a způsobu práce lze předpokládat, že autonomní vysavače pro plánování cesty používají algoritmy popsané v této kapitole, nebo jejich obdobu.

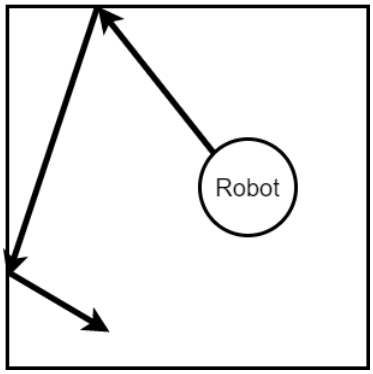
Kapitola je rozdělena do několika sekcí. Každá sekce sdružuje jeden či více plánovacích algoritmů. V první sekci je popsán nejjednoduší plánovací algoritmus vysavačů – náhodný pohyb 4.1. Druhá sekce obsahuje algoritmy, které provádějí přibližný rozklad na buňky 4.2. Třetí sekce se věnuje přesnému rozkladu na buňky 4.3. Čtvrtá sekce se věnuje rozkladu řezem 4.5, pátá sekce je věnována topologickému plánování 4.6 a poslední sekce se věnuje boustrophedon A^* algorithmu 4.7.

4.1 Náhodný pohyb

Náhodný pohyb je nejstarším algoritmem používaným v robotických vysavačích. V dnešní době se již příliš nepoužívá.

Pro implementaci náhodného pohybu využívám algoritmů z článku [10].

Robotické vysavače řízené tímto algoritmem nemají žádnou umělou inteligenci a jak již název napovídá, pohybují se čistě náhodně. Robotický vysavač se pohybuje směrem rovně, dokud nenarazí na překážku, kterou detekuje senzory. Poté se otočí do náhodného směru a pokračuje v pohybu do té doby, než opět narazí na překážku. Po nějakém čase robot takto pokryje celý prostor. Pohyb vysavače je naznačen na obrázku 4.1



Obrázek 4.1: Vizualizace pohybu robota při implementaci algoritmu náhodného pohybu

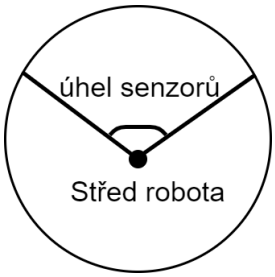
Nevýhodou algoritmu je to, že nelze s jistotou určit, zda robot opravdu pokryl celý prostor. Také se může stát, že se robot dostane do prostoru, ve kterém se zacyklí a nebude schopen z něj vyjet. Tato situace může nastat například v rohu místnosti. Z tohoto důvodu autoři článku počítají dobu, po kterou se má robot po nárazu s překážkou otáčet, pomocí následující rovnice :

$$t = [\frac{R\emptyset}{720 * rN} + Random(0, \frac{R\emptyset}{720 * rN})]sekund \tag{4.1}$$

Význam jednotlivých proměnných je následující :

- R – vzdálenost kol robota od jeho středu
- r – poloměr kol robota
- N – počet otáček kol za sekundu
- $\emptyset$ – úhel, který snímají senzory nárazu

Rozsah senzorů robota je také vizualizován na obrázku 4.2



Obrázek 4.2: Vizualizace rozsahu senzorů robota

Rovnice 4.1 se používá v algoritmu při otočení o náhodný čas. Pseudokód náhodného pohybu je na algoritmu 1.

S postupem času se k tomuto algoritmu přidali ostatní algoritmy, pomocí nichž se robot pohybuje v několika módech, ty mezi sebou střídá. Kromě náhodného pohybu je to pohyb po spirále, pohyb kolem zdi a pohyb, který v literatuře [10] autoři označují jako pohyb ve tvaru písmena 'S'.

Algoritmus 1: Algoritmus náhodného pohybu

```
for Dobu úklidu do
    Pohybuj se směrem vpřed
    if Byla detekována kolize then
        if Kolize na pravém senzoru then
            Couvni
            Otáčej se doleva po náhodnou dobu
        end if
        else
            Couvni
            Otáčej se doprava po náhodnou dobu
        end if
    end if
end for
```

4.1.1 Pohyb po spirále

Při provádění tohoto algoritmu robotický vysavač svým pohybem opisuje tvar spirály. S jeho pomocí je schopen pokrýt velkou část prostoru místnosti. Pohyb po spirále je možno simulovat tak, že se jedno kolečko robota otáčí na plný výkon, zatímco druhé kolečko svůj výkon postupně zvyšuje. Robotický vysavač se po spirále pohybuje do té doby, než narazí na překážku. Pseudokód pohybu po spirále je na algoritmu 2, v němž robotický vysavač opisuje spirálu, která se zvětšuje v proti směru hodinových ručiček.

Algoritmus 2: Algoritmus pohybu po spirále

```
while Nedošlo ke kolizi do
    Navyš otáčky levého kola
end while
```

4.1.2 Pohyb kolem zdi

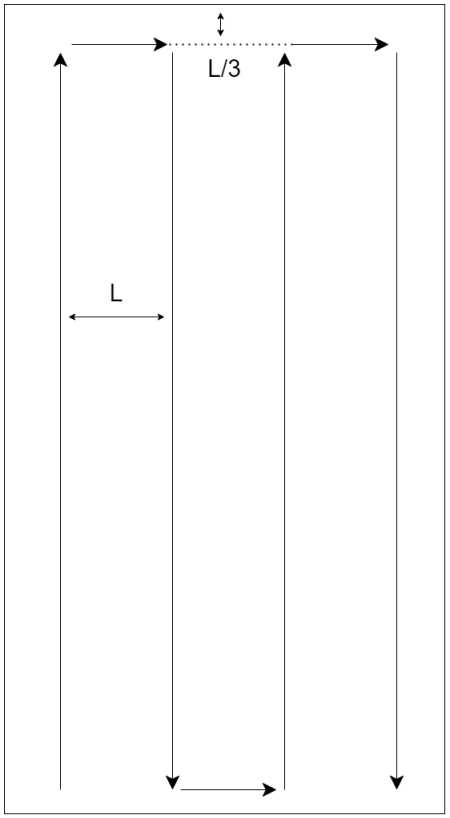
Cílem tohoto algoritmu je provést úklid místnosti okolo stěn a uklidit tak nečistoty v jejich okolí. Pseudokód pohybu kolem zdi je na algoritmu 3.

Algoritmus 3: Algoritmus pohybu kolem zdi

```
for Dobu úklidu do
    Pohybuj se směrem vpřed
    if Byla detekována kolize then
        while Je detekována kolize do
            Otáčej se doprava
        end while
        Pohybuj se směrem vpřed po dobu T
        while Není detekována kolize do
            Otáčej se doleva
        end while
    end if
end for
```

4.1.3 Pohyb ve tvaru písmena 'S'

Provedením tohoto algoritmu pokryje robot danou oblast pohyby tam a zpět, které vizuálně připomínají tvar písmene 'S'. Tyto pohyby se v literatuře [3] označují jako *boustrophedon* pohyby, toto označení pochází z řečtiny a označoval se tak tvar pohybu volů, kteří orali pole. Při popisu algoritmu ale vycházím z pojmů názvu zavedeném v článku [10]. Tvar pohybů, které robotický vysavač při vykonávání tohoto algoritmu opisuje, jsou vizualizovány na obrázku 4.3. L na obrázku je průměr robota.



Obrázek 4.3: Vizualizace tvaru pohybů robota

Aby se robot pohyboval ve tvaru písmene 'S', po nárazu s překážkou vykoná tyto pohyby:

1. Couvej
2. Otoč se o 90° doleva/doprava
3. Popojeď dopředu
4. Otoč se o 90° doleva/doprava

Pro výpočet doby, po kterou má robot pohyby vykonávat, se opět používají proměnné r – poloměr kol robota a N – počet otáček kol za sekundu. Rovnice pro výpočet časů

jednotlivých pohybů vypadají následovně:

$$\begin{aligned} \text{Doba couvání} &= \frac{L}{6\pi r N} sekund \\ \text{Doba otočky} &= \frac{R}{4rN} sekund \\ \text{Doba pohybu vpřed} &= \frac{L}{2\pi r N} sekund \end{aligned} \tag{4.2}$$

Robot se pohybuje směrem vpřed, dokud nenarazí na překážku, poté vykoná sérii pohybů uvedených výše. Pseudokód, který využívá těchto rovnic, je na algoritmu 4.

Algoritmus 4: Algoritmus pohybu ve tvaru písmena 'S'

```
1  Cnt = 0
2  for Dobu úklidu do
3      Pohybuj se směrem vpřed
4      if Byla detekována kolize then
5          Couvni
6          if Cnt je liché čislo then
7              Otoč se o 90°doprava
8              Vykonej pohyb směrem vpřed
9              Otoč se o 90°doleva
10         end if
11         else
12             Otoč se o 90°doleva
13             Vykonej pohyb směrem vpřed
14             Otoč se o 90°doprava
15         end if
16         Cnt ++
17     end if
18 end for
```

4.2 Přibližný rozklad na buňky

Plánovací algoritmy této kategorie rozdělí prohledávaný prostor do mřížky stejně velkých tvarů. Nejčastěji se prostor rozděluje do čtverců velikosti průměru robota, ale lze využít i trojúhelníky či jiné tvary.

Každá buňka mřížky uchovává informace o svých sousedech a zároveň informaci, zda byla prozkoumána.

Nevýhodou rozdělení prostoru do mřížky je exponenciální náročnost na paměť, která se zvětšuje se zvětšujícím se prostředím. Ideálním prostředím pro rozdělení na mřížku jsou uzavřené prostory. Při špatně zvolené velikosti jedné buňky mřížky dochází k tomu, že robotický vysavač nevyčistí okolí překážek. Tento problém lze vyřešit zmenšením velikosti buňky.

4.2.1 Algoritmus záplavového vyplňování

V literatuře [23] je představen algoritmus pro kompletní pokrytí prostoru rozděleného do mřížky. Jednotlivým buňkám mřížky jsou nejprve pomocí algoritmu záplavového vyplňování přiřazeny specifické hodnoty.

Algoritmus záplavového vyplňování, který přiřadí buňkám hodnoty pracuje podle algoritmu 5.

Algoritmus 5: Záplavové vyplňovaní

```
1  C_Goal = Náhodná buňka z pole
2  C_Goal.value = 0
3  C_Curr = C_Goal
4  C_Last
5  while Nejsou ohodnoceny všechny buňky do
6      foreach Neohodnoceného souseda buňky C_Curr do
7          Soused.value = C_Curr + 1
8          C_Last = Soused
9      end foreach
10     C_Curr = C_Last
11 end while
```

Pole ohodnocené algoritmem záplavového vyplňování je na obrázku 4.4.

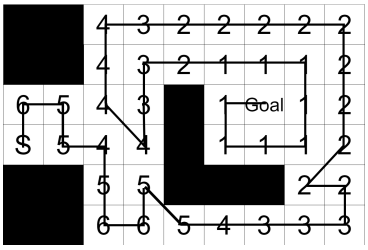
Obrázek 4.4: Ohodnocené pole algoritmem záplavového vyplňování

Pro kompletní pokrytí oblasti je cesta plánována přes buňky s nejvyšším ohodnocením až po ty s nejnižším. V případě, že prozkoumaná buňka má dvě sousední se stejně velkým nižším ohodnocením, než je ohodnocení akutální buňky, je následující buňka vybrána

náhodně. Robot plánuje cestu podle algoritmu 6. Naplánovaná cesta je vizualizována na obrázku 4.5

Algoritmus 6: Plánovaní cesty v mřížce ohodnocené záplavovým vyplňováním

```
1  C_Start = Aktuální buňka
2  Nastav všem buňkám příznak visited na hodnotu false
3  while do
4      Najdi nenavštívenou sousední buňku s nejvyšší hodnotou value
5      if Buňka nebyla nalezena then
6          Nastav příznak aktuální buňky na hodnotu true
7          Pokud jsi v cíli ukonči úklid
8      end if
9      if Hodnota value nalezené buňky <= hodnota value aktuální buňky then
10         Nastav příznak aktuální buňky na hodnotu true
11         Pokud jsi v cíli ukonči úklid
12     end if
13     Pokračuj do sousední buňky
14 end while
```



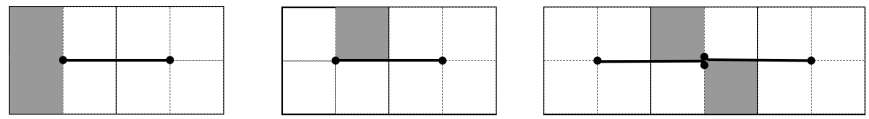
Obrázek 4.5: Cesta vytvořená v mřížce ohodnocené algoritmem záplavového vyplňování

4.2.2 Spiral-STC algoritmus

Tento algoritmus byl poprvé představen v [9]. S jeho pomocí je autonomní robotický vysavač schopen kompletně pokrýt předem neznámý prostor.

Prohledávaný prostor je rozdělen do tzv. mega buňek velikosti $2D$, kde D je průměr robota. Každá mega buňka je navíc rozdělena na čtyři menší buňky velikosti D. Mega buňky jsou uloženy v grafu, uzly grafu jsou středy mega buňek a hrany grafu propojují středy sousedních mega buňek. Algoritmus postupně vytváří kostru tohoto grafu a s její pomocí plánuje cestu kompletního pokrytí.

Algoritmus také počítá s výskytem překážek. Nová mega buňka je do grafu přidána v případě, že obsahuje alespoň jednu buňku velikosti D, ve které se nenachází žádná překážka. Přidáním překážek do pole vznikají tři nové typy hran, viz obrázek 4.6. Oboustranná hrana, která má volné buňky na obou stranách. Jednostranná hrana, která má volnou buňku jen na jedné z jejích stran. Poslední typ hrany se nazývá dvojitý uzel, nastane v případě, že mega buňka má dvě volné buňky, které jsou umístěny na diagonále. V tomto případě se vytvoří dva uzly megabuňky, kdy každý z nich je napojen na jednu hranu.



Obrázek 4.6: Nové typy hran při přidání překážek do pole

V následujícím pseudokódu využívá algoritmus označení mega buňek jako *stará* a *nová*. Mega buňka je *nová*, jestliže každá její podbuňka nebyla pokryta. V opačném případě je označena jako *stará*. Pseudokód algoritmu je následující :

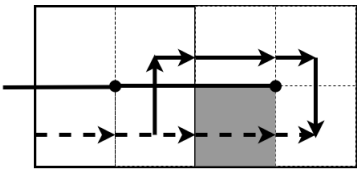
Algoritmus 7: SPIRAL-STC

```
1  STC2(Null,S) // S = startovní buňka
2  STC2(w, x).
3  Označ aktuální mega buňku x jako starou
4  while x má nové volné nebo z částí okupované sousední buňky do
5      Vyber první sousední mega buňku ve směru proti hodinovým ručičkám, začni
         od rodičovské mega buňky w, vybranou megabuňku označ jako y
6      Vytvoř hranu kostry grafu z x do y
7      Přesuň se do buňky megabuňky y podél hrany kostry grafu, podle typu hrany
         (viz. níže)
8      STC2(x, y)
9  end while
10 if x != S then
11     Přesuň se zpět do buňky megabuňky w podél hrany kostry grafu, podle typu
         hrany (viz. níže)
12     Return
13 end if
```

V případě, že jsou v grafu jednostranné hrany, musí robot vykročit z trasy a vyhnout se tak překážce. Nechť α je označení pro starou cestu, kterou by vykonal v případě, že by byla

buňka bez překážek. Modifikovaná trasa, β, je získána posunutím originální cesty α mimo buňku s překážkou tak, že všechny body cesty β jsou vzdáleny $D/2$ od buňky s překážkou.

Tento úhybný manévr je vizualizován na obrázku 4.7



Obrázek 4.7: Čárkovanou čarou je znázorněna originální cesta α. Plnou čarou je znázorněna modifikovaná cesta β

4.3 Přesný rozklad na buňky

Anglicky se tato metoda v literatuře [3] označuje jako *Exact cell decomposition*. Metody tohoto typu rozdělí požadovaný prostor do množiny jednoduchých tvarů, které označujeme jako buňky. Rozložení buněk je typicky uloženo v grafu sousednosti, ve kterém uzly odpovídají jednotlivým buňkám. Dvě buňky grafu jsou propojeny hranou v případě, že spolu sousedí.

Přesný rozklad na buňky lze jednoduše využít pro kompletní pokrytí daného prostoru, každou buňku lze totiž pokrýt pomocí předdefinovaných pohybů. Jakmile robot takto projde všechny buňky grafu sousednosti, docílí kompletního pokrytí daného prostoru.

4.3.1 Lichoběžníková dekompozice

Lichoběžníková dekompozice [3] pracuje v polygonálním 2D poli, ve kterém se nachází polygonální překážky. Daným prostorem se po jednotlivých vrcholech posouvá řez. Při kontaktu vrcholu a řezu vznikají či se uzavírají nové buňky.

Mohou pak nastat 3 typy událostí:

- **IN** – je událost, při které řez narazí na překážku. Uzavře se aktuální buňka a otevřou se dvě nové, viz obrázek
- **OUT** – je událost, při které se naopak dvě buňky uzavřou a jedna nová se vytvoří, viz obrázek
- **MIDDLE** – je událost, při které se jedna buňka uzavře a jedna otevře, viz obrázek

Pseudokód algoritmu lichoběžníkové dekompozice je na algoritmu 8. Algoritmus pracuje se setříděným seznamem Y-nových souřadnic vrcholů a dále seznamem D, do kterého ukládá jednotlivé buňky.

Z finální dekompozice lze vidět, že některé buňky jsou vytvořeny zbytečně a mohly by být spojeny v jednu větší buňku. Pokrytí většího množství menších buněk je méně efektivní, viz obrázek.

4.3.2 Boustrophedon dekompozice

Boustrophedon dekompozice [3] modifikuje lichoběžníkovou dekompozici a řeší problém malých přebytečných buněk.

Algoritmus 8: Lichoběžníková dekompozice

$y_L \leftarrow$ souřadnice vrcholu, ve kterém se nachází aktuální řez
$\{y_1, \cdots, y_n\} \leftarrow$ setříděný seznam všech vrcholů prostoru
$D = (\dots, c_{i-2}, c_{i-1}, c_i, c_{i+1}, c_{i+2}, \dots)$
for $y_L = y_1$ *až* y_n **do**
    if *Událost* = *IN* **then**
        $(c_i) \leftarrow (c_d, c_{d+1}$
        $D = (\dots, c_{i-2}, c_{i-1}, c_d, c_{d+1}, c_{i+1}, c_{i+2}, \dots)$
    end if
    if *Událost* = *OUT* **then**
        $(c_i, c_{i+1}) \leftarrow (c_e)$
        $D = (\dots, c_{i-2}, c_{i-1}, c_e, c_{i+2}, \dots)$
    end if
    if *Událost* = *MIDDLE* **then**
        $(c_i) \leftarrow (c_f)$
        $D = (\dots, c_{i-2}, c_{i-1}, c_f, c_{i+1}, c_{i+2}, \dots)$
    end if
end for

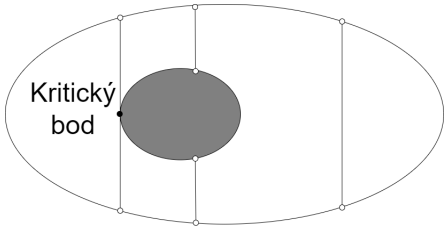
Jednotlivými vrcholy se opět prokládá řez, avšak mohou nastat pouze dvě události *IN* a *OUT*. Obě události jsou stejné jako při lichoběžníkové dekompozici. Událost *MIDDLE* při Boustrophedon dekompozici nenastává a nevytvářejí se proto nadbytečné buňky.

4.3.3 Morseova dekompozice

Morseova dekompozice [3] je rozkladová metoda, která využívá pro rozklad řezy. Je to další modifikace, tentokrát Boustrophedon dekompozice. Řez je posouván v prostoru po malých krocích, nikoli po vrcholech. Morseova dekompozice je tak schopna vytvořit dekompozici prostředí, s nepolygonálními překážkami.

Řez je definován jako podmnožina nadplochy prostoru a označuje se jako Q_λ, kde λ je parametr, pomocí kterého lze řez v prostoru posouvat. λ nabývá kladných hodnot ve směru úklidu a naopak záporných ve směru opačném. Řez umístěný do daného prostoru, Q_{free}, se označuje jako $Q_{free\lambda}$. Se zvětšujícím se parametrem λ se řez posouvá v prostoru a protíná (nebo přestává protínat) překážky, řez se takto rozdělí na více částí (nebo spojí zase v jednu). V prvním bodě, ve kterém se setká řez s překážkou, se mění konektivita výsledné dekompozice. Konektivita je vlastnost, která udává, že z počátečního bodu q_{start} existuje cesta do koncového bodu q_{end}. Tyto body označujeme jako kritické body viz. obrázek 4.8. Řezy, které obsahují kritické body, označujeme jako kritické řezy. Z druhého řezu v obrázku 4.8 lze vidět, že řez může být rozdělen na několik částí. Sjednocením všech částí jednoho řezu dostaneme zpátky řez $Qfree\lambda$. Poslední pojem, který pro definici morseovy dekompozice potřebujeme, je množina všech kritických řezů, I^*.

Definition 4.3.1 (Morseova dekompozice)**.** Morseova dekompozice je přesný rozklad na buňky. Buňky Morseovy dekompozice jsou propojené prvky množiny $Q_{free} \setminus I^*$.



Obrázek 4.8: První řez je kritický řez, druhý protíná překážku a je rozdělen na dva a třetí je neprotíná žádnou překážku

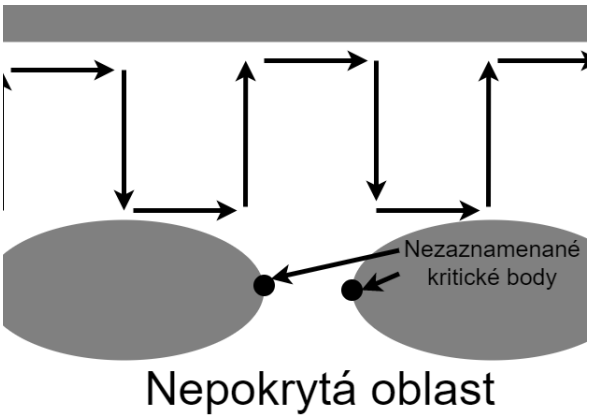
4.4 Prohledávání prostoru pomocí senzorů

V literatuře [3], [2] se tento algoritmus označuje pouze jako *Senzor-based coverage*. Algoritmus implementuje algoritmus morseovy dekompozice, který upravuje tak, aby pracoval v neznámém prostředí. Robot tak pokrývá neznámý prostor a zároveň si v paměti vytváří *Reeb* graf.

Reeb graf je duální vůči grafu sousednosti vytvořeným morseovou dekompozicí. Uzly tohoto grafu představují kritické body prostoru a hrany mezi nimi značí jednotlivé buňky. Pro úspěšné pokrytí daného prostoru musí být robot schopen identifikovat pomocí svých senzorů kritický bod a také být schopen všechny kritické body najít.

Robot identifikuje kritické body za pomocí svých senzorů, v literatuře [3] robot disponuje sonarovým prstenem. Má-li robot k dispozici tento typ senzorů, pak kritický bod lze najít za pomocí normál povrchu. Normála kritického bodu je rovnoběžná se směrem úklidu.

Druhým problémem, se kterým se robot musí poprat, je nalezení všech kritických bodů oblasti. Kdyby algoritmus tento problém neřešil, mohlo by se stát, že by robot při úklidu nenarazil na jeden či více kritických bodů a nepokryl by tak celý prostor. Tato situace je ilustrována na obrázku 4.9, na kterém robot nezaznamená dva kritické body v horní části buňky, protože zrovna uklízel spodní část buňky. Ve zdrojové literatuře [2] je proto zaveden tzv. *Cycle algorithm*, pomocí kterého robot najde všechny kritické body a pokryje tak celý prostor. Vývojový diagram algoritmu se nachází na obrázku 4.10



Obrázek 4.9: Vizualizace případu, ve kterém robot nezaznamená kritické body

Na začátku *Cycle* algoritmu se robot nachází v bodu S_i. Bod S_i se nachází přímo na hranici volného prostoru nebo v jeho blízkém okolí. Směr úklidu, $\delta h(x)$, je značen jako směr vpřed. Horní hranice buňky se nazývá "strop" a spodní hranice "země". Pohyb robota směrem od *stropu* k *zemi* je označován jako pohyb ve směru kruhu. Naopak pohyb ve směru od *země* ke *stropu* se nazývá pohyb proti směru kruhu. Robot hledá kritické body v následujících fázích :

1. 1. fáze – *Forwad phase*

   Robot se pohybuje ve směru kruhu do té doby, než narazí na překážku. Poté se pohybuje okolo překážky vpřed. *Forward phase* končí, jakmile robot narazí na kritický bod, nebo jakmile robot urazí vzdálenost rovnu průměru robota. Robot urazí vzdálenost ve směru úklidu w_f, pro kterou platí, že je menší nebo rovna průměru robota.

2. 2. fáze – *Reverse phase*

   Robot se pohybuje ve směru proti směru kruhu, jakmile narazí na překážku, pohybuje se okolo ní v opačném směru úklidu. V případě, že při pohybu okolo překážky narazí robot na kritický bod, který leží mezi řezy CS_{λ_i} a $CS_{\lambda_{i+1}}$, se robot vrátí k pohybu proti směru kruhu. Tyto pohyby se střídají do té doby, než robot narazí na kritický bod, který se nachází nalevo od počátečního bodu S_i, nebo pokud urazí vzdálenost v opačném směru úklidu, která je rovna průměru robota. Může se stát, že na konci 2. fáze se robot dostane do počátečního bodu S_i, pokud se tak stane, algoritmus skončí. V opačném případě robot přechází do 3. fáze algoritmu

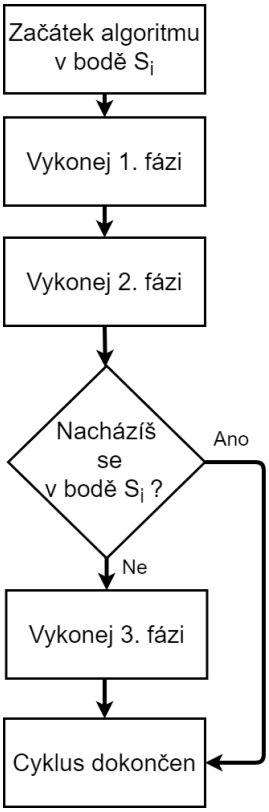
3. 3. fáze – *Closing phase*

   Robot se pohybuje ve směru kruhu, pokud narazí na překážku, pohybuje se kolem ní ve směru úklidu. Tyto dvě fáze se střídají do té doby, dokud robot nedorazí do počátečního bodu S_i, nebo do buňky, které bod S_i náleží.

Cycle algoritmus je hlavní částí algoritmu pro kompletní pokrytí prostoru. V průběhu si robot v paměti ukládá objevené kritické body. Poslední kritický bodC_{p_f}, který robot objevil, se stává uzavírajícím bodem a společně s počátečním bodem S_i jsou v *Reeb* grafu propojeny hranou. V případě, že při vykonávání *cycle* algoritmu objeví robot více kritických bodů, vytvoří se v *reeb* grafu odpovídající počet nových buňek.

Pokud po skončení algoritmu robot objevil nový kritický bod C_{p_f}, který hraničí s jednou či více sousedními buňkami, vydává se robot prozkoumat jednu z nich. V případě, že kritický bod C_{p_f} nehraničí s žádnou buňkou, robotův plánovač provede prohledávání do hloubky *reeb* grafu a vybere tak jiný kritický bod s doposud nepokrytou buňkou. Pokud takový uzel neexistuje, znamená to, že robot pokryl celý prostor.

K pohybu k neprozkoumanému kritickému bodu na grafu robot využívá upravené verze Bug2 algoritmu [17]. V upravené verzi algoritmu je cesta od robota k cílovému kritickému bodu rozdělena na několik částí, každé části odpovídá jedna buňka. V každé buňce se robot pohybuje podél jedné z jejích hraničních překážek, *stropu* nebo *země*. Mezi buňkami se robot pohybuje po jednotlivých řezech viz obrázek 4.11.



Obrázek 4.10: Vývojový diagram *Cycle* algoritmu

4.5 Dekompozice řezem

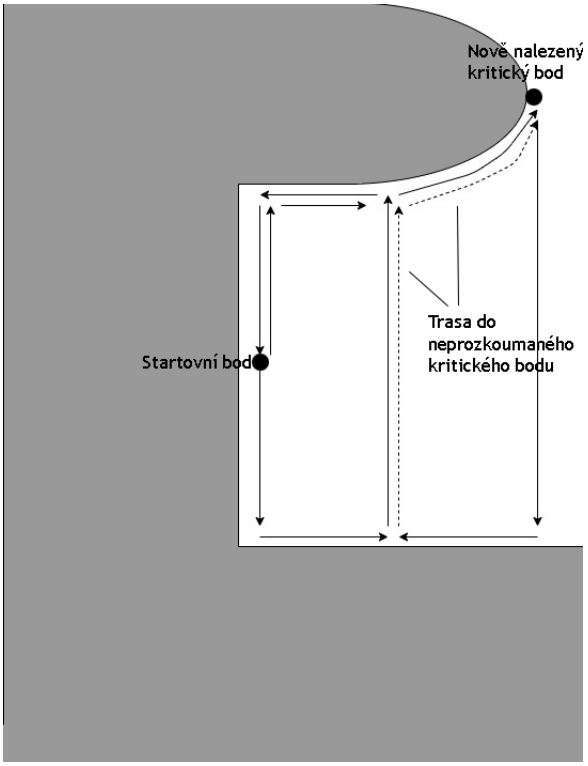
4.5.1 Dekompozice řezem I

Dekompozice řezem je rozkladová metoda poprvé představena v [22]. Vychází z Boustrophedon dekompozice 4.3.2 a rozšiřuje koncept událostí *IN* a *OUT*.

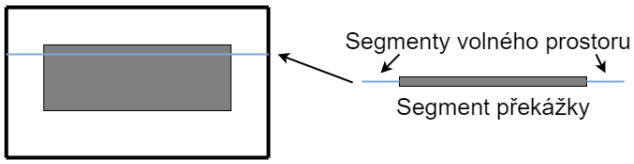
Výsledkem je prostor rozdělený do množiny buněk, které lze snadno pokrýt boustrophedon pohyby. Dekompozice je tvořená řezy. Řez je postupně posouván v prostoru směrem odshora dolů, lze si ho představit jako paprsek protínající prostor a v něm umístěné překážky směrem zleva doprava. Podle topologie prostoru rozděluje řez prostor na několik částí, které jsou označovány jako tzv. segmenty. Umístění řezu do prostoru a rozdělení na segmenty je na obrázku 4.12.

Existují dva typy segmentů. Segmenty volného prostoru, ty vznikají v místě, kde řez nic neprotl. Opakem jsou potom segmenty překážek, které vznikají v místech, kde řez koliduje s překážkou. Hranice nové buňky je tvořena, jakmile řez narazí na místo, ve kterém se liší počet segmentů aktuálního a předchozího řezu. Existuje pět různých událostí, které mohou nastat:

1. *Objevení překážky* – Segment volného prostoru předešlého řezu je rozdělen na dva nové v aktuálním řezu. Událost je totožná s událostí IN Boustrophedon dekompozice.
2. *Objevení volného prostoru* – Segment překážky je rozdělen na dva nové segmenty.
3. *Zánik překážky* – Segment překážky z předešlého řezu v aktuálním řezu zmizí. Událost je totožná s událostí OUT Boustrophedon dekompozice.



Obrázek 4.11: Vizualizace plánování cesty k nově objevenému kritickému bodu

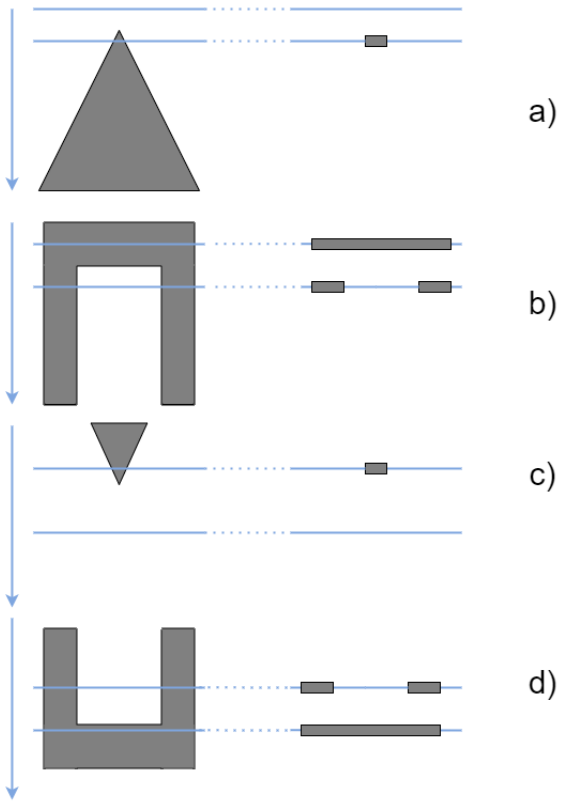


Obrázek 4.12: Řez umístěn do prostoru s popisem jeho segmentů

4. *Zánik volného prostoru* – Segment volného prostoru z předešlého řezu v aktuálním řezu zmizí.

Všechny typy událostí jsou zároveň na obrázku 4.13

Algoritmus pracuje se seznamem D aktivních buněk. Buňky jsou děleny na buňky překážek a volné buňky. Princip je podobný jako u lichoběžníkové dekompozice 4.3.1, výsledná dekompozice je tvořena stavy, ve kterých se seznam D nacházel v jednotlivých krocích. Algoritmus dekompozice řezem je na algoritmu 9



Obrázek 4.13: Události dekompozice řezem I : a) Objevení překážky b) Objevení volného prostoru c) Zánik překážky d) Zánik volného prostoru

Algoritmus 9: Dekompozice řezem I

$c \in \{$volné buňky, buňky překážek$\}$
forall *Řezy* **do**
    Posuň řez směrem dolů o vzdálenost Δx
    $D_{t-1} = (\ldots, c_{i-2}, c_{i-1}, c_i, c_{i+1}, c_{i+2}, \ldots)$
    forall *segmenty v* D_{t-1} **do**
        if *Nový segment v* c_i **then**
            $(c_i) \leftarrow (c_{e-1}, c_e, c_{e+1})$
            $D_t = (\ldots, c_{i-2}, c_{i-1}, c_{e-1}, c_e, c_{e+1}, c_{i+1}, c_{i+2}, \ldots)$
        end if
        if *Zánik segmentu* c_i **then**
            $(c_{i-1}, c_i, c_{i+1}) \leftarrow (c_d)$ $D_t = (\ldots, c_{i-2}, c_d, c_{i+2}, \ldots)$
        end if
    end forall
end forall

4.5.2 Dekompozice řezem II

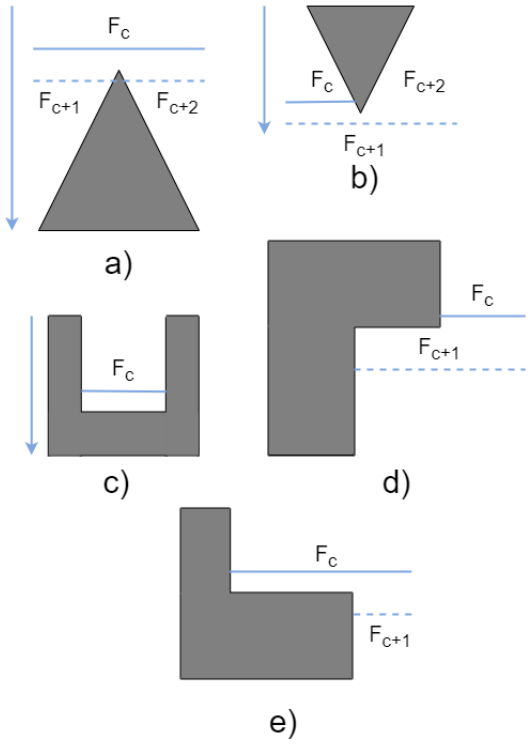
Dekompozice řezem II [22] modifikuje první variantu tak, aby ji bylo možné implementovat v robotech. Robot se nemůže pohybovat uvnitř překážek, řez je proto limitován jen na

buňku, ve které se robot aktuálně nachází. Zároveň není robot schopen pokrýt všechny překážky pohybem odshora dolů, například konkávní překážky ve tvaru písmena U. Události algoritmu jsou proto modifikovány, aby tyto problémy vyřešily.

Existuje pět typů událostí, které mohou nastat:

1. *Split* – Segment volného prostoru minulého řezu je rozdělen na dva nové segmenty, v důsledku objevení překážky viz obrázek.
2. *Merge* – Segment volného prostoru aktuálního řezu sousedí s novým prostorem, událost nastává při překonání překážky, viz obrázek.
3. *End* – Předchozí segment volného prostoru byl posledním v aktuální buňce, událost nastává ve slepé uličce, viz obrázek.
4. *Lenghten* – Aktuální segment volného prostoru sousedí se segmentem překážky a zároveň sousedí se segmentem volného prostoru předchozího řezu. Událost nastává při prodloužení řezu, viz obrázek.
5. *Shorten* – Přesný opak události *Lenghten*, zkracuje řez, viz obrázek.

Vizualizace všech událostí je na obrázku 4.14, modrá šipka naznačuje směr pohybu robota, neznamená to však, že událost nastane pouze v tomto směru. Všechny události mohou nastat i při směru opačném.



Obrázek 4.14: Události dekompozice řezem II : a) Split b) Merge c) End d) Lenghten e) Shorten

Algoritmus dekompozice řezem 10 pracuje s dvěma seznamy O (open) a F (finish). Seznam open, O, udržuje všechny objevené buňky, zatímco seznam finish, F, uchovává

všechny navštívené buňky. Algoritmus pracuje dokud není seznam open prázdný. Jakmile je seznam open prázdný, všechny buňky byly navštíveny a pokryty.

Algoritmus 10: Dekompozice řezem II

```
1  O ← Počáteční buňka
2  F ← ∅
3  while O ≠ ∅ do
4      f_c ← f ∈ O
5      Přesuň se na jednu z hranic buňky f_c
6      repeat
7          Pohybuj se směrem k druhé hranici buňky o vzdálenost Δx
8          if Nastala událost then
9              F ← F + f_c
10             O ← O − f_c
11             if událost = Split nebo Merge then
12                 O ← O + f_{c+1}, f_{c+2} if f_{c+1}, f_{c+2} ∉ (O ∪ F)
13             end if
14             if událost = Lenghten nebo Shorten then
15                 O ← O + f_{c+1} if f_{c+1} ∉ (O ∪ F)
16             end if
17         end if
18     until Nenastane událost;
19 end while
```

4.6 Algoritmus topologického pokrytí

Algoritmus topologického [22] pokrytí vytváří dekompozici řezem 4.5, zatímco pokrývá neznámé prostředí. Robot si při pokrývání prostoru v paměti vytváří topologickou mapu prostoru – G, která využívá událostí z dekompozice řezem pro tvorbu uzlů.

Algoritmus topologického pokrytí implementuje dekompozici řezem jako konečný automat se třemi stavy. Pseudokód implementace konečného automatu je na algoritmech 11,12,13.

- Následuj hranici – Výchozí stav, ve kterém se robot při startu algoritmu nachází. Robot následuje hranici překážky, jakmile prozkoumá celou hranici, přesune se do stavu Cestuj. Viz algoritmus 11
- Cestuj – Robot prohledá topologickou mapu a vybere prozatím nepokrytou buňku, do které se přesune. Jakmile dorazí do buňky, nastane stav Pokryj. Viz algoritmus 12
- Pokryj – Ve stavu pokryj vykonává robot boustrophedon pohyby. Viz algoritmus 13

4.6.1 Tvorba grafu

Jak již bylo zmíněno, robot si při vykonávání algoritmu vytváří v paměti topologickou mapu prostoru. Vznikne tak graf, který odpovídá dekompozici řezem. Zároveň je topologickou ma-

Algoritmus 11: Následuj hranici

```
1  while do
2      Pohybuj se směrem vpřed po hranici
3      if jsi narazil na orientační bod then
4          Aktualizuj G
5      end if
6      if se nacházíš na konci buňky then
7          Aktualizuj G
8          if je hranice plně pokryta then
9              Stav ← Cestuj
10         end if
11         else
12             Otoč se o 180◦
13         end if
14     end if
15 end while
```

Algoritmus 12: Cestuj

```
1  T(n) ← Prohledej G
2  if T(n) = ∅ then
3      Ukonči algorimus
4  end if
5  while T(n) ≠ ∅ do
6      Přesouvej se do T(0)
7      if se nacházíš v T(0) then
8          T(n) ← T(n) − T(0)
9      end if
10 end while
11 Stav ← Pokryj
```

Algoritmus 13: Pokryj

```
1  repeat
2      Vykonávej boustrophedon pohyby
3  until Nenarazíš na orientační bod;
4  Aktualizuj G
5  Stav ← Následuj hranici
```

pou prostoru, protože každý uzel odpovídá nějáké části prostoru, například rohu místnosti či stěně.

4.6.2 Uzly grafu

Uzly grafu nesou informaci o jejich typu a o hranách, které jim náleží. Existují čtyři typy uzlů : volný prostor, překážka, neprozkoumaný prostor a spoj.

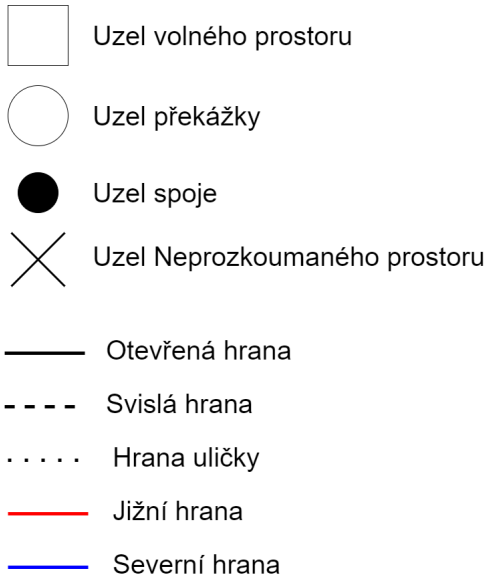
- Uzel volného prostoru – je z dvou přilehlých hran obklopen překážkou. Ačkoliv je to neintuitivní označení, je tento uzel v literatuře [22] označen jako uzel volného prostoru, protože je detekován při zmizení segmentu volného prostoru.
- Uzel překážky – jsou detekovány poblíž překážek
- Uzel neprozkoumaného prostoru – značí v grafu doposud neprozkoumanou část prostoru
- Uzel spoje – je využit pro propojení uzlů překážek a uzlů volného prostoru

4.6.3 Hrany grafu

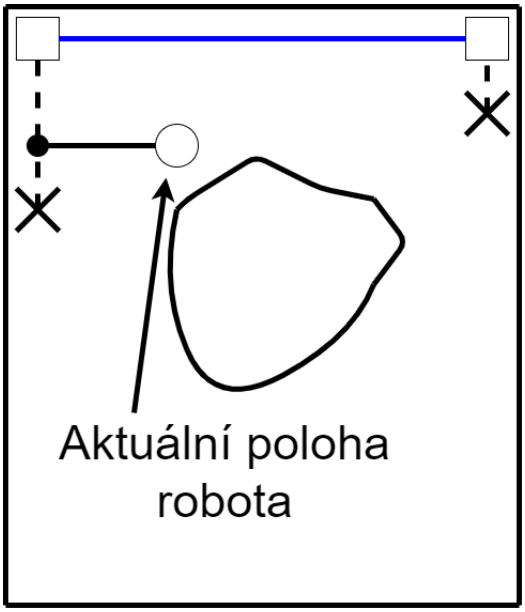
Uzly grafu jsou propojeny jedním z pěti typů hran. Každá hrana nese informaci o svém typu a také uchovává přibližnou vzdálenost dvou uzlů, které propojuje.

- Otevřená hrana – podobně jako uzel volného prostoru je tato hrana neintuitivní, označuje vodorovnou hranu, která okolo sebe nemá překážky.
- Svislá hrana – svislou hranou jsou označeny levé a pravé hranice jednotlivých buněk
- Severní hrana – vodorovná hrana, nad kterou je umístěna překážka
- Jižní hrana – vodorovná hrana, pod kterou je umístěna překážka
- Hrana uličky – hrana, která má překážky na obou svých stranách

Označení jednotlivých uzlů a hran se nachází na obrázku 4.15. Na obrázku 4.16 je z části vytvořená topologická mapy prostoru.



Obrázek 4.15: Uzly a hrany grafu



Obrázek 4.16: Částečně vytvořená topologická mapa prostoru

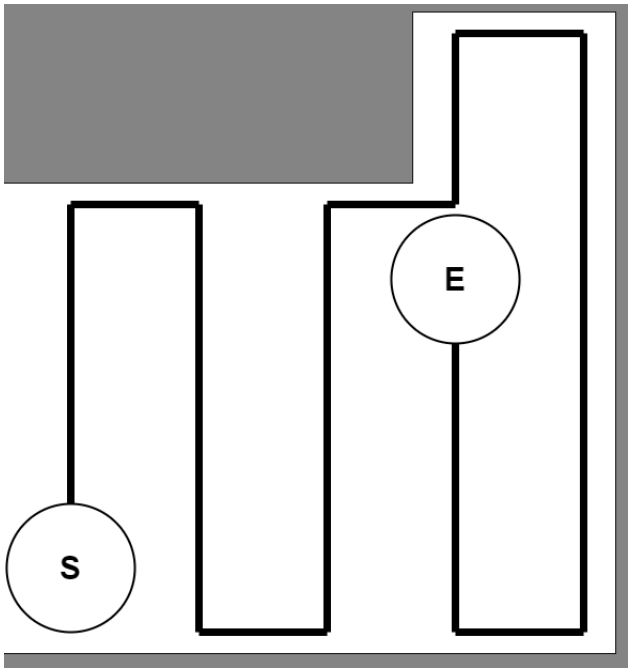
4.7 Boustrophedon A^*

Jak již z názvu vyplývá, algoritmus Boustrophedon A^*, dále pouze BA^*, využívá ke kompletnímu pokrytí prostoru boustophedon pohyby. Robot tyto pohyby vykonává do té doby, než narazí na kritický bod. V průběhu vykonávání boustrophedon pohybu si robot vytváří tzv. dlaždicový model prostoru. Jedna dlaždice je čtverec o straně odpovídající robotově průměru. Poté, co robot narazí na kritický bod, využije vytvořený dlaždicový model k nalezení nového startovacího bodu, ze kterého začne vykonávat další cyklus boustrophedon pohybů. Pro plánování cesty z kritického bodu do nového startovacího bodu se využívá A^* algoritmus. Odtud druhá část názvu. Algoritmus je představen v článku [4]

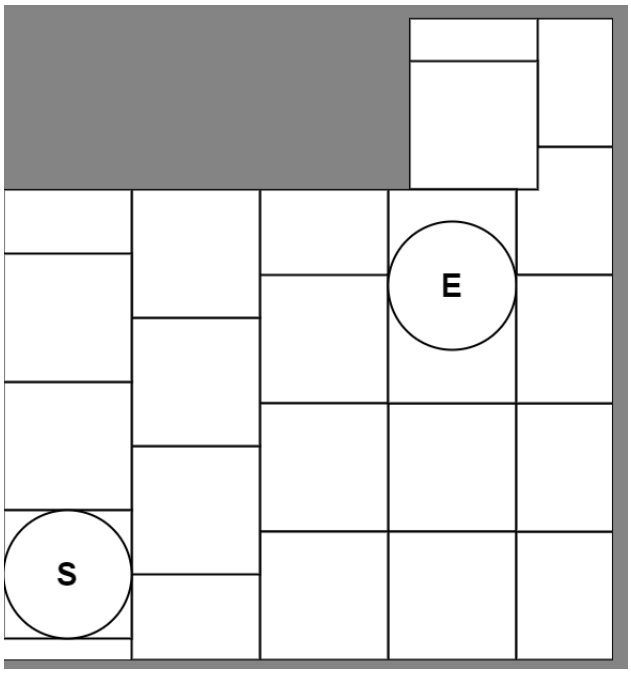
4.7.1 Boustrophedon pohyb

Robot pokrývá neznámý prostor pomocí boustrophedon pohybů. Vytváří tak postupně dlaždicový model prostoru. Narozdíl od boustrophedon pohybů v předešlých algoritmech se robot na každé dlaždici rozhoduje, kterým směrem se vydá. Z okolí aktuální dlaždice si vybere toho souseda, který není zablokovaný, tzn. nejedná se o překážku, a ani o již pokrytou dlaždici. V případě, že aktuální dlaždice žádného takového souseda nemá, robot narazil na kritický bod.

Robot tedy k výběru směru nevyužívá pouze dat ze sensorů, ale také již vytvořeného dlaždicového modelu. Směr dalšího pohybu je plánován prioritně v následujícím pořadí: sever, jih, východ, západ. V případě, že je severní směr zablokován, je kontrolován jižní směr. Jestliže je zablokován i ten, kontroluje se východní směr a v případě, že je zablokován i ten, kontroluje se nakonec směr západní. Jeden boustrophedon pohyb je na obrázku 4.17. 'S' značí startovní pozici, ze které začal robot boustrophedon pohyb vykonávat a 'E' je kritický bod, ve kterém jsou všechny směry jeho okolí zablokovány. Vytvořený dlaždicový model odpovídající pohybu na obrázku 4.17 je na obrázku 4.18.



Obrázek 4.17: Část jednoho boustrophedon pohybu



Obrázek 4.18: Dlaždicový model vytvořený vykonáním pohybu na obrázku 4.17

4.7.2 Nalezení nového startovacího bodu

V drtivé většině případů nestačí ke kompletnímu pokrytí prostoru pouze jeden boustrophedon pohyb. Robot musí proto být schopen najít startovní bod nového boustrophedon pohybu.

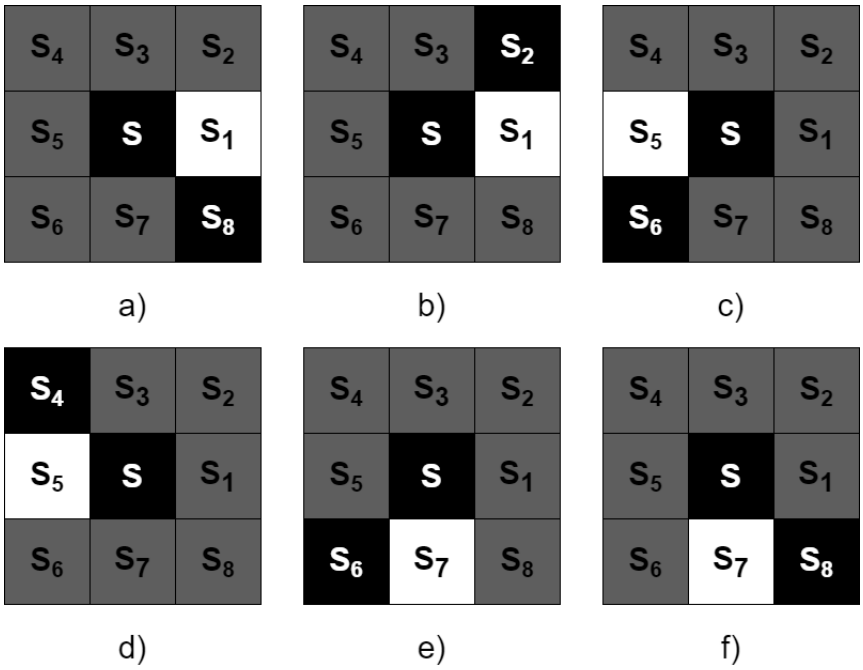
Algoritmus 14: BOUSTROPHEDON POHYB

Input: Poloha robota a dlaždicový model M
Output: Aktualizovaná poloha robota a aktualizovaný dlaždicový model M

1 Najdi první neblokovaný směr v pořadí Sever, Jih, Východ, Západ
2 **if** *Všechny směry jsou zablokované* **then**
3 Kritický bod nalezen
4 Ukonči pohyb
5 **end if**
6 Pohybuj se nalezeným směrem o vzdálenost průměru robota
7 Vytvoř novou dlaždici s = (x,y,2r), tzn. dlaždici o velikosti robotova průměru na jeho aktuální pozici
8 Přidej nově vygenerovanou dlaždici do modelu M a vrať se na krok 1.

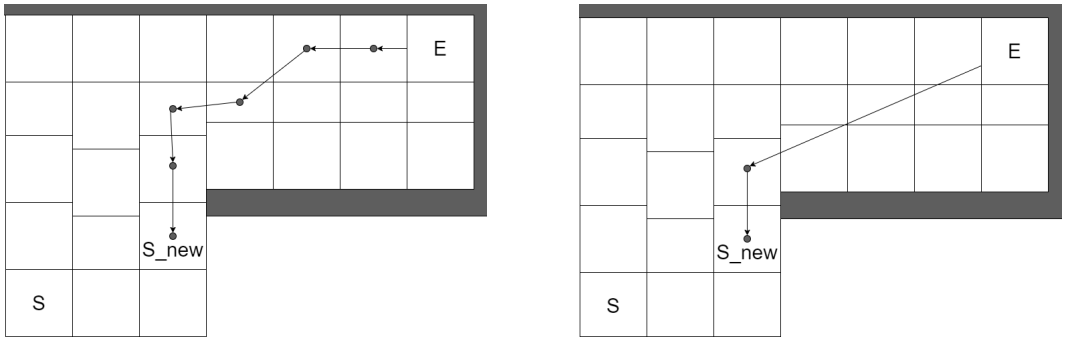
Nový startovní bod může být jakkákoliv dlaždice, která má alespoň jednoho volného souseda. Vybírat mezi těmito dlaždicemi náhodně však není ideální, protože špatně zvolený startovní bod může rozdělit prostor do částí, které nelze pokrýt jedním boustrophedon pohybem. To způsobí delší dobu úklidu.

V článku [4] proto autoři zavádějí skupinu pravidel pro výběr ideálního startovního bodu. Při výběru pracují s dlaždicí S a jejím rozšířeným okolím osmi sousedů. Při splnění jedné z podmínek na obrázku 4.19 je dlaždice zařazena mezi kandidáty na novou startovní pozici. Dlaždice označeny černou barvou značí, že jsou zablokovány, tzn. ze všech čtyřech stran jsou obklopeny dlaždicí nebo překážkou. Bíla barva naopak znamená, že je dlaždice volná, tzn. aspoň jeden její soused je volný prostor. Šedá barva pak znamená, že se na stav dlaždice nebere ohled, může být jak volná, tak zablokovaná.



Obrázek 4.19: Případy, ve kterých je dlaždice S vybrána jako kandidátní startovní pozice

Pro ohodnocení dvojice dlaždic je také v článku [4] zavedena funkce :



(a) Cesta nalezena algoritmem A^* (b) Výsledná vyhlazená cesta

Obrázek 4.20: Hledání cesty do nového startovního bodu

$$b(s_i, s_j) = \begin{cases} 1, \text{jestliže}(s_i\text{je volná})\text{a}(s_j\text{je zablokovaná}) \\ 0, \text{jinak} \end{cases}$$

Tato funkce ohodnotí dvojici dlaždic číslem 1 nebo 0. Pro vyhodnocení, zda je dlaždice S kandidátem na novou startovní pozici, se využije součet hodnot vracených funkcí b().

$$\mu(s) = b(s_1, s_8) + b(s_1, s_2) + b(s_5, s_6) + b(s_5, s_4) + b(s_7 + s_6) + b(s_7, s_8)$$

Indexy dlaždic odpovídají rozmístění na obrázku 4.19. V případě, že $\mu(s) \geq 1$, je dlaždice S zařazena jako kandidátní.

V případě, že se v dlaždicovém modelu M nenachází žádná dlaždice splňující $\mu(s) \geq 1$, je algoritmus ukončen, protože robot kompletně pokryl prostor. V opačném případě je ze seznamu vybrána taková dlaždice, která je aktuálně robotovi nejblíže. Pro výpočet vzdálenosti je možno použít manhattanovskou, či euklidovskou vzdálenost.

4.7.3 Plánování cesty k novému startovnímu bodu

Poté, co je vybrán nový bod, ze kterého robot bude vykonávat boustrophedon pohyb, musí k němu být naplánována cesta. K tomuto je využito A^* algoritmu, viz algoritmus 15.

Cesta nalezená algoritmem A^* však často není ideální a je kostrbatá. Pro její vyhlazení je proto využito algoritmu 16. Příklad cesty nalezené algoritmem A^* a příklad této cesty vyhlazené algoritmem 16 je na obrázku 4.20

Algoritmus 15: A^* ALGORITMUS

Input: Aktuální dlaždice s_s a cílová dlaždice s_g
Output: Cesta z dlaždice s_s do dlaždice s_g

```
1  OPEN = prioritní fronta obsahující dlaždici s_s
2  CLOSED = prázdná množina
3  while Dlaždice s nejmenším ohodnocením fronty OPEN není s_g do
4      CURRENT = Dlaždice s nejmenším ohodnocením fronty OPEN
5      vyjmi CURRENT z OPEN
6      přidej CURRENT do CLOSED
7      for sousedy N dlaždice CURRENT do
8          f = g(CURRENT) + h(CURRENT,N)
9          if N je v OPEN a f < g(N) then
10             vyjmi N z OPEN
11         end if
12         if N je v CLOSED a f < g(N) then
13             vyjmi N z CLOSED
14         end if
15         if N není v OPEN a N není v CLOSED then
16             nastav hodnotu g(N) na hodnotu f
17             hodnotu f(N) nastav na g(N) + h(N)
18             rodiče N nastav na dlaždici CURRENT
19             přidej N do OPEN
20         end if
21     end for
22 end while
```

Algoritmus 16: ALGORITMUS VYHLAZENÍ CESTY NALEZENÉ ALGORITMEM A^*

Input: Cesta $P = [s_1, ..., s_n]$ nalezená algoritmem A^*
Output: Vyhlazená cesta $\hat{P} = [\hat{s_1}, ..., \hat{s_k}]$

1 Nastav $k = 1$ a přidej s_k do $\hat{P}$
2 Najdi nejvzdálenější dlaždici $s_i (i = n, n-1, ..., k+1)$, která je v přímé viditelnosti z dlaždice s_k
3 Přidej dlaždici s_i do $\hat{P}$
4 Inkrementuj k
5 Jestliže $s_k = s_n$ vrať cestu $\hat{P}$, v opačném případě se vrať na krok 2

Kapitola 5

Knihovna vizlib

Tato kapitola bude věnována knihovně vizlib, ve které byly vybrané algoritmy pro plánování cest vizualizovány.

První část kapitoly tvoří popis důležitých zobrazení, které se při tvorbě algoritmů využívají, u každého zobrazení je navíc kus demonstračního kódu.

V druhé části je potom shrnuto, jak vytvořit samotný algoritmus a princip třídy `UndoableCommand`.

5.1 Práce s knihovnou

Knihovna vizlib je knihovna napsána v jazyce Java, Jakubem Rusnákem [14], jako výstuip jeho diplomové práce. V této bakalářské práci byla využita pro vizualizaci vybraných algoritmů kompletního pokrytí prostoru. Knihovna implementuje návrhový vzor MVC (Model-View-Controller) [21]. Výhodou použití tohoto návrhového vzoru jsou zobrazení, která knihovna nabízí. Zobrazení mají předdefinované chování a programátor tak upraví pouze jejich obsah a může používat jejich funkce.

V následujících sekcích budou shrnuta nejdůležitější zobrazení pro tvorbu algoritmu. Při vytváření zobrazení je důležité zavolat konstruktora předka pomocí příkazu `super()`.

5.1.1 MainPanel

Hlavní panel je základem celé aplikace. V konstruktoru třídy `MainPanel` se inicializují všechna zobrazení, která budou použita při vizualizaci algoritmu. Pro použitá zobrazení je také vhodné nastavit rozložení. Ta je možno rozložit do rozhrání `Frame`, které je nastavuje automaticky. Další možností je vytvoření vlastního rámce, či použití přímo metod třídy `MainPanel`.

Vytvoření hlavního panelu lze vidět na výpisu 5.1.

```
public class RandomAlgMainPanel extends AlgClassMainPanel {
    private final RandomConsole console;
    private final RandomConfigurationSpace space;
    private final RandomToolBar toolBar;
    private final RandomParameters parameters;
    private final RandomCode code;

    public RandomAlgMainPanel() {
        super(new RandomAlg());
        console = new RandomConsole();
```

```
        space = new RandomConfigurationSpace();
        code = new RandomCode();
        toolBar = new RandomToolBar(space);
        parameters = new RandomParameters(space);
        new Frame1(this, toolBar, space, parameters, code, console);
    }
}
```

Výpis 5.1: Vytvoření hlavního panelu a registrace příslušných zobrazení

5.1.2 CodeView

Zobrazení `CodeView` slouží pro vizualizaci pseudokódu právě vykonávaného algoritmu. Lze v něm nastavit zvýraznění klíčových slov. Navíc lze také zvýraznit celý řádek s právě vykonávanou instrukcí. Vytvoření vlastního `CodeView` lze vidět na výpisu 5.2.

```
public class RandomCode extends CodeView {

    public static final String NAME = "Pseudocode";
    private String chosenAlg = "";

    public RandomCode() {
        super(NAME, null);
    }
}
```

Výpis 5.2: Vytvoření CodeView

5.1.3 ConsoleView

Třída `ConsoleView`, jak již název napovídá, simuluje funkci konzole. Do té je možno zobrazit dodatečné informace ve formě textu. Tato funkcionalita se hodí obzvlášť při krokování. Výpis 5.3 ukazuje vytvoření vlastního `ConsoleView` lze vidět na výpise 5.3.

```
public class RandomConsole extends ConsoleView {

    public final static String NAME = "Console";

    public RandomConsole() {
        super(NAME, null);
    }
}
```

Výpis 5.3: Vytvoření ConsoleView

5.1.4 ParameterView

Toto zobrazení umožňuje zpracovávat uživatelem zadané parametry, ty se mohou využít při samotném běhu algoritmu nebo mohou měnit uživatelské rozhrání. Uživatel parametry zadá do vstpuních kompotent definovaných programátorem.

Na výpise 5.4 lze vidět vytvoření `ParameterView` obsahující čtyři parametry. Tři vstupy pro číselnou hodnotu a jeden combobox.

```
public class RandomParameters extends ParameterView {

    public static final String CLEANEDSPACE = "Pocet narazu do prekazek";
    public static final String ANIMATIONSTEP = "Pocet vykreslenych kroku";
    public static final String NUMOFCYCLES = "Pocet cyklu";
    public static final String ALGORITHMCOMBOBOX = "Vyber algoritmu";

    private final static String NAME = "Parameters";

    public RandomParameters(ConfigurationSpaceView sv) {
        super(NAME, null);

        Properties.INSTANCE.setProperty(CLEANEDSPACE, 100);
        IntegerField percentage = new IntegerField(CLEANEDSPACE, 4);
        add(percentage);
        Properties.INSTANCE.setProperty(ANIMATIONSTEP, 20);
        IntegerField stepSize = new IntegerField(ANIMATIONSTEP, 4);
        add(stepSize);
        Properties.INSTANCE.setProperty(NUMOFCYCLES, 2);
        IntegerField numOfCycles = new IntegerField(NUMOFCYCLES, 4);
        add(numOfCycles);
        ComboBox algorithmComboBox = new ComboBox(ALGORITHMCOMBOBOX);
        algorithmComboBox.addElement("Nahodny pohyb");
        algorithmComboBox.addElement("Pohyb okolo zdi");
        algorithmComboBox.addElement("Boustrophedon pohyby");
        algorithmComboBox.addElement("Pohyb po spirale");
        algorithmComboBox.addElement("Stridat vsechny pohyby");
        add(algorithmComboBox);
}
```

Výpis 5.4: `ParameterView` obsahující čtyři parametry

5.1.5 ToolbarView

`ToolbarView` umožňuje zobrazit ovládací prvky aplikace. Kromě jejich definice je také nutno namapovat jednotlivé ovládací prvky pro všechny stavy, ve kterých se aplikace nachází. Příklad vytvoření `ToolBarView` obsahující řídící prvky aplikace je na výpise 5.5.

```
public class RandomToolBar extends ToolbarView {

    final static String NAME = "ToolBar";

    public RandomToolBar(ConfigurationSpaceView mv) {
        super(NAME, null);
        // Buttons for controlling simulation
        SimulationButton sim = new SimulationButton();
        EditButton edit = new EditButton();
        PlayPauseContinueRestartButton play = new PlayPauseContinueRestartButton();
        LargeStepBackButton largeStepBack = new LargeStepBackButton();
        StepBackButton stepBack = new StepBackButton();
        StepButton step = new StepButton();
        LargeStepButton largeStep = new LargeStepButton();
        StopButton stop = new StopButton();
```

```
        // Buttons for adding object to map
        JToggleButton start = new FirstGenRobotButton(mv);
        JToggleButton obstacle = new ObstacleButton(mv);
        JToggleButton circularObstacle = new CircularObstacleButton(mv);
        JToggleButton remove = new RemoveButton(mv);
        NoneSelectedButtonGroup group = new NoneSelectedButtonGroup();
        group.add(start);
        group.add(obstacle);
        group.add(circularObstacle);
        group.add(remove);
        // Slider for controlling speed of simulation
        Properties.INSTANCE.setProperty(AlgClass.DELAY_KEY,
            AlgClass.DELAY_DEFAULT);
        Slider speed = new Slider(AlgClass.DELAY_KEY, 3);
        speed.getSlider().setMaximum(AlgClass.MAX_SPEED);
        // Setting content of toolbar for different states of application
        addComponentsForState(State.INIT, edit, sim);
        addComponentsForState(State.EDIT, sim, start, obstacle, circularObstacle,
            remove);
        addComponentsForState(State.READY, edit, largeStepBack, stepBack, step,
            largeStep, stop, play, speed);
        addComponentsForState(State.RUNNING, edit, largeStepBack, stepBack, step,
            largeStep, stop, play, speed);
        addComponentsForState(State.PAUSED, edit, largeStepBack, stepBack, step,
            largeStep, stop, play, speed);
        addComponentsForState(State.STEP, edit, largeStepBack, stepBack, step,
            largeStep, stop, play, speed);
        addComponentsForState(State.STEP_BACK, edit, largeStepBack, stepBack,
            step, largeStep, stop, play, speed);
        addComponentsForState(State.LARGE_STEP, edit, largeStepBack, stepBack,
            step, largeStep, stop, play, speed);
        addComponentsForState(State.LARGE_STEP_BACK, edit, largeStepBack,
            stepBack, step, largeStep, stop, play, speed);
        addComponentsForState(State.FINISHED, edit, largeStepBack, stepBack, step,
            largeStep, stop, play, speed);
    }
}
```

Výpis 5.5: Vytvoření `ToolbarView` obsahující řídící prvky aplikace

5.1.6 Zobrazení běhu algoritmu

V této sekci budou sdruženy zobrazení, ve kterých lze vizualizovat samotný průběh algoritmu. Na rozdíl od již zmíněných zobrazeních se tato často mění algoritmus od algoritmu, v knihovně vizlib je jich proto hned několik. V případě, že na daný algoritmus není vhodné žádné z předdefinovaných zobrazení, není problém vytvořit nové.

DrawView

`DrawView` umožňuje vykreslovat různé tvary, které jsou instancí třídy `PaintableShape`. Zaznamenává pohyb myši a uživatel tak může tyto tvary měnit.

MapView

Zobrazení `MapView` obsahuje kromě skupiny překážek také start a cíl. Uživatel stejně jako v `DrawView` může pomocí myši všechny tvary přesouvat. Překážky může navíc měnit, přidávat i mazat. Jakýkoliv tvar mapy jde zvýraznit, dodatečně k němu může být přidán i popisek. Tato funkce opět pomáhá při vizualizaci algoritmu.

GridMapView

`GridMapView` je mapa zobrazena na mřížce čtvercových buněk. Každá buňka má svůj typ, uživatel tak může do mřížky přidat start a cíl.

5.2 Vytvoření algoritmu

Třída `AlgClass` umožňuje tvorbu samotného algoritmu. Právě v této třídě probíhají výpočty daného algoritmu. Pro vizualizaci kroků se využívá třídy `UndoableCommand`, která má dvě metody `Redo()` a `Undo()`. Metoda `Redo()` spustí příkaz a metoda `Undo()` naopak zvrátí jeho průběh. Pro správnou funkci krokování je nutno definovat, co mají obě metody dělat. Aby byl algoritmus vizualizován, je tedy nutné během výpočtů vytvářet instance třídy `UndoableCommand` a ty přidávat do seznamu instrukcí pomocí metody `addCommand()`. Vytvoření nového příkazu a jeho následné přidání lze vidět na výpisu 5.6.

```
final Point finalAfterDirection = new
    Point(afterDirection.getX(),afterDirection.getY());
final Point finalBeforeDirection = new
    Point(beforeDirection.getX(),beforeDirection.getY());
addCommand(new UndoableCommand() {
    @Override
    public void redo() {
        space.getRobot().setDirection(finalAfterDirection.getX(),
          finalAfterDirection.getY());
    }

    @Override
    public void undo() {
        space.getRobot().setDirection(finalBeforeDirection.getX(),
          finalBeforeDirection.getY());
    }
});
```

Výpis 5.6: Vytvoření příkazu `UndoableCommand` a jeho přidání do seznamu příkazů

Běh algoritmu bude simulován po zavolání metody `simulate()`, ta začne simulovat algoritmus pomocí seznamu příkazů `UndoableCommand`. Uživatel má tak možnost algoritmus krokovat pomocí ovládacích prvků.

Kapitola 6

Aplikace pro vizualizaci algoritmů kompletního pokrytí prostoru

Tato kapitola bude věnována praktické části této bakalářské práce, a to implementaci vybraných algoritmů v knihovně vizlib.

Výstupem této práce je implementace algoritmů Náhodného pohybu 4.1 a algoritmu BoustrophedonA^* 4.7 a algoritmu topologického pokrytí 4.6. Algoritmy simulují pohyb autonomního uklízecího robota s omezením reálných senzorů.

Knihovna vizlib neumožňuje implementovat několik algoritmů v jednom programu, každý algoritmus je proto samostatně spustitelná aplikace.

6.1 Implementace

6.1.1 Využitá zobrazení

Pro implementaci všech algoritmů bylo využito rozhrání `Frame1`, které je předdefinované v knihovně vizlib. Toto rozhraní přijímá zobrazení pět libovolných zobrazení `View`. V práci byla využita tato zobrazení `Toolbarview`, `ConfigurationSpaceView`, `ParameterView`, `CodeView`, `ConsoleView`. Všechna zmíněná zobrazení s výjimkou `ConfigurationSpaceView` jsou popsána v kapitole 5.

ConfigurationSpaceView

Toto zobrazení bylo vytvořeno specificky pro vizualizaci algoritmů kompletního pokrytí daného prostoru. Jedná se o modifikovanou verzi zobrazení `MapView` 5.1.6. Stejně jako `MapView` obsahuje seznam překážek. Kromě startu a cíle však obsahuje instanci třídy `Robot`, ta reprezentuje robota, který algoritmus vykonává.

6.1.2 Třída Robot

Třída `Robot` je bázovou třídou pro všechny roboty. Poskytuje metody pro pohyb robota:

- `move()` – robot se pohne ve směru `direction` rychlostí speed
- `rotate(int angle)` – otočí robota ve směru o daný úhel `angle`, kladná čísla otočí robota ve směru hodinových ručiček, záporná ve směru opačném
- `setSpeed(double speed)` – nastaví rychlost robota na hodnotu `speed`

Mnoho robotů je vybaveno mechanickým nárazníkem, který se sepne až po nárazu s překážkou. Je proto nutné, aby bylo možno roboto poslat jakýmkoliv směrem, bez kontroly zda náhodou nekoliduje s překážkou. Za tímto účelem také třída `Robot` implementuje jednoduchou rezoluci kolizí.

Každý z algoritmů pracuje se svým robotem, který dědí od bázové třídy `Robot`.

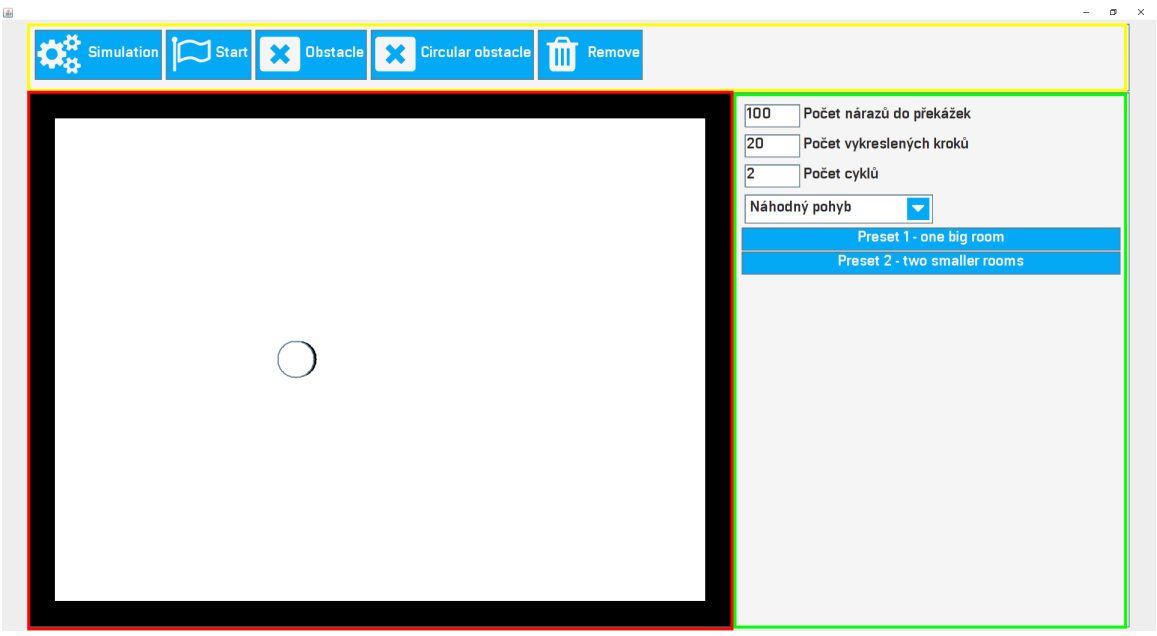
6.1.3 Třída Animator

Knihovna vizlib simuluje běh algoritmu pomocí seznamu instrukcí, třída `Animator` byla vytvořena za účelem vykreslování robotova pohybu. V průběhu výpočtu algoritmu průběžně přidává příkazy `UndoableCommand` obsahující robotovu aktuální polohu. Tímto způsobem je vytvořena animace.

6.2 Aplikace

Jak již bylo zmíněno výše, každý z algoritmů je samostatně spustitelná aplikace. Nicméně všechny tři aplikace se ovládají stejným způsobem.

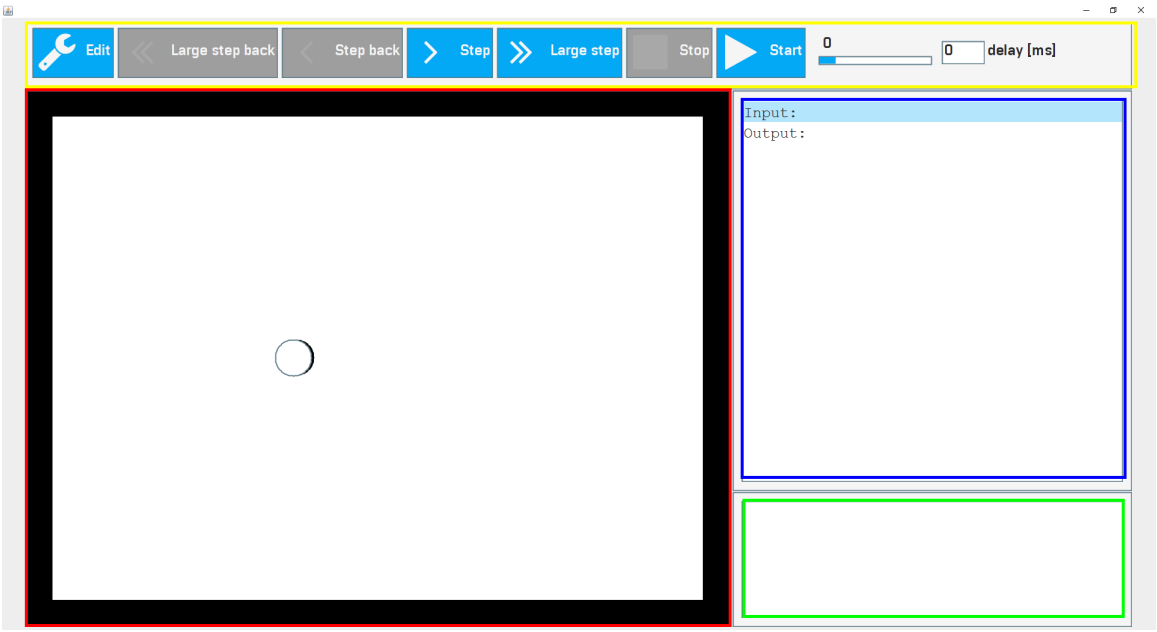
Na obrázku 6.1 lze vidět aplikaci v editovacím módu. Červenou barvou je označeno zobrazení `ConfigurationSpace` a v něm robot. Napravo od `ConfigurationSpaceView` je zelenou barvou označeno zobrazení `ParameterView`, do kterého lze nejen zadávat parametry algoritmu, ale také zvolit jedno z předpřipravených prostředí. Posledním zobrazením na obrázku 6.1 je `ToolbarView`, to v editovacím módu aplikace umožňuje tlačítkem Start přemístit robota a také přidat nové překážky. Tlačítko Obstacle přidá polygonální překážku, zatímco tlačítko Circular obstacle přidá kruhovou překážku. Tlačítko remove pak umožní odebrat z `ConfigurationSpaceView` jak překážku, tak robota.



Obrázek 6.1: Aplikace v editovacím režimu

Po vytvoření požadovaného prostředí a stisknutím tlačítka Simulation je aplikace přepnuta do Simulačního režimu, viz obrázek 6.2. `ConfigurationSpaceView` je opět označeno

červenou barvou, v tomto režimu již nejde editovat. `ToolbarView`, označeno žlutou, také zůstává, avšak mění se jeho stav. Mizí tlačítka pro editaci `ConfigurationSpaceView` a jsou nahrazena za tlačítka pro řízení simulace, mezi které patří i posuvník delay. Ten umožňuje nastavit rychlost provádění jednotlivých kroků algoritmu. Parametry `ParameterView` jsou zpracovány a zobrazení je nahrazeno za dvě nová, modře označené `CodeView` a zeleně označené `ConsoleView`.



Obrázek 6.2: Aplikace v simulačním režimu

Tyto ovládací prvky jsou stejné ve všech aplikacích.

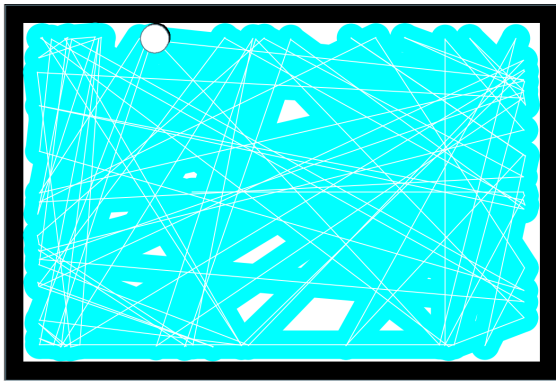
6.2.1 Algoritmus náhodného pohybu

Algoritmus náhodného pohybu simuluje pohyb robota, který disponuje pouze mechanickým nárazníkem (černý okraj robota). Algoritmus přijímá následující parametry:

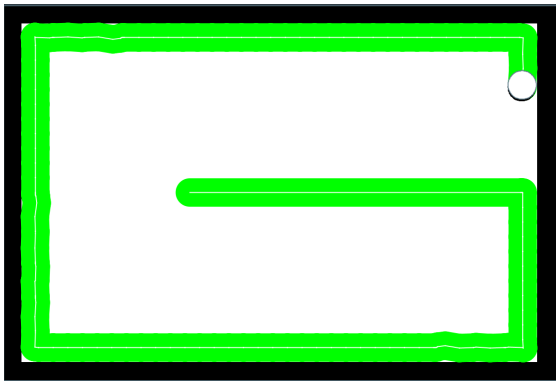
- Počet nárazů do překážek – celočíselná hodnota určující počet překážek, po kterém robot ukončí vybraný mód. Tento parameter neplatí v režimu spirály, ten se ukončí po prvním nárazu do překážky.
- Počet vykreslených kroků – celočíselná hodnota v rozmezí $(1, x)$. Hodnota 1 vykreslí všechny kroky, se zvyšující se hodnotou jsou některé kroky nevykreslovány.
- Počet cyklů – Celočíselná hodnota, parametr je použit v případě výběru módu, ve kterém robot střídá všechny módy.
- Algoritmus – Výběr požadovaného módu z comboboxu.

Po nastavení a spuštění algoritmu robot započne úklid. Prostor, který robot pokryje, je v aplikaci naznačen barevně a liší se podle módu, ve kterém tuto oblast pokryl. Jednotlivým módům náleží následující barvy:

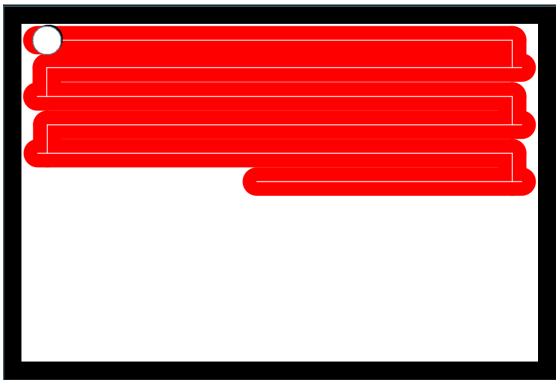
- Náhodný pohyb – světle modrá, viz 6.3a



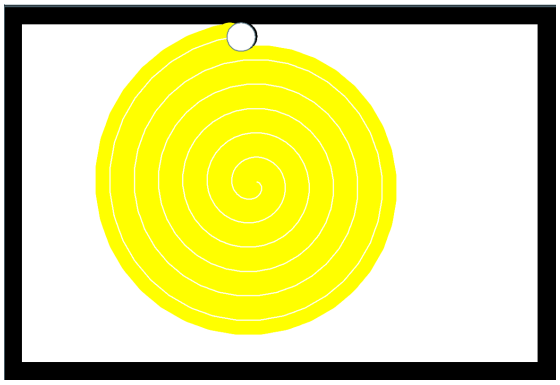
(a) Náhodný pohyb



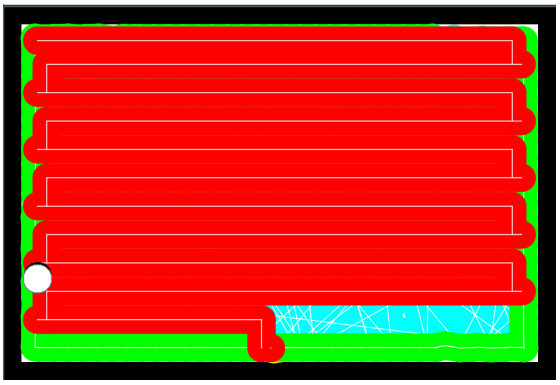
(b) Pohyb okolo zdi



(c) Boustrophedon pohyby



(d) Pohyb po spirále



(e) Všechny pohyby

Obrázek 6.3: Barvy jednotlivých módů

- Pohyb kolem zdi – zelená, viz 6.3b
- Boustrophedon pohyby – červená, viz 6.3c
- Pohyb po spirále – žlutá, viz 6.3d

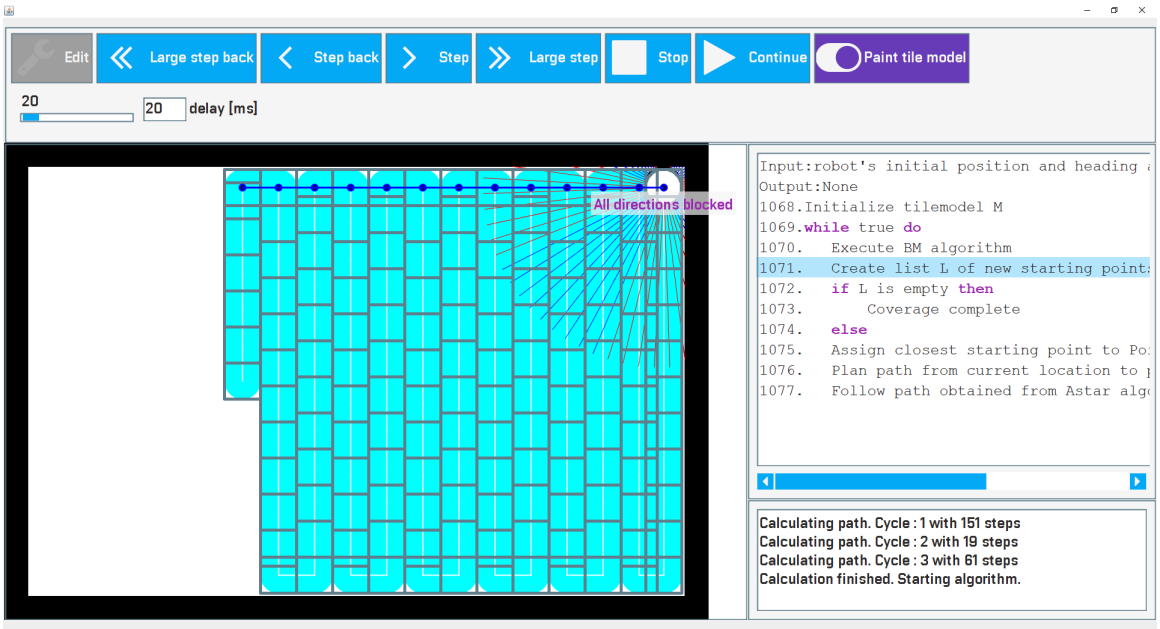
Výsledek po jednom cyklu všech módů je na obrázku 6.3e

6.2.2 Boustrophedon A^* algoritmus

Druhým implementovaným algoritmem je algoritmus boustrophedon A^* 4.7.

Simulovaný robot disponuje sonarovým prstencem, ten je implementován třídou `SonarRingSensor`. Sonarový prstenec je implementován jako seznam úseček s určitým rozestupem, které jsou po celém obvodu robota.

Jak již bylo zmíněno v kapitole 5, knihovna vizlib zajišťuje krokování pomocí instancí třídy `UndoableCommand`, ta obsahuje příkazy, které se mají provést. Před začátkem simulace musí být proto všechny výpočty dokončeny. Tento princip v kombinaci s tím, že je algoritmus BA^* časově náročný, a reálný robot ho provádí za běhu, způsobuje prodlevu po spuštění algoritmu. Prodleva může způsobovat dojmem, že robot počítá plán cesty z mapy, avšak není tomu tak.

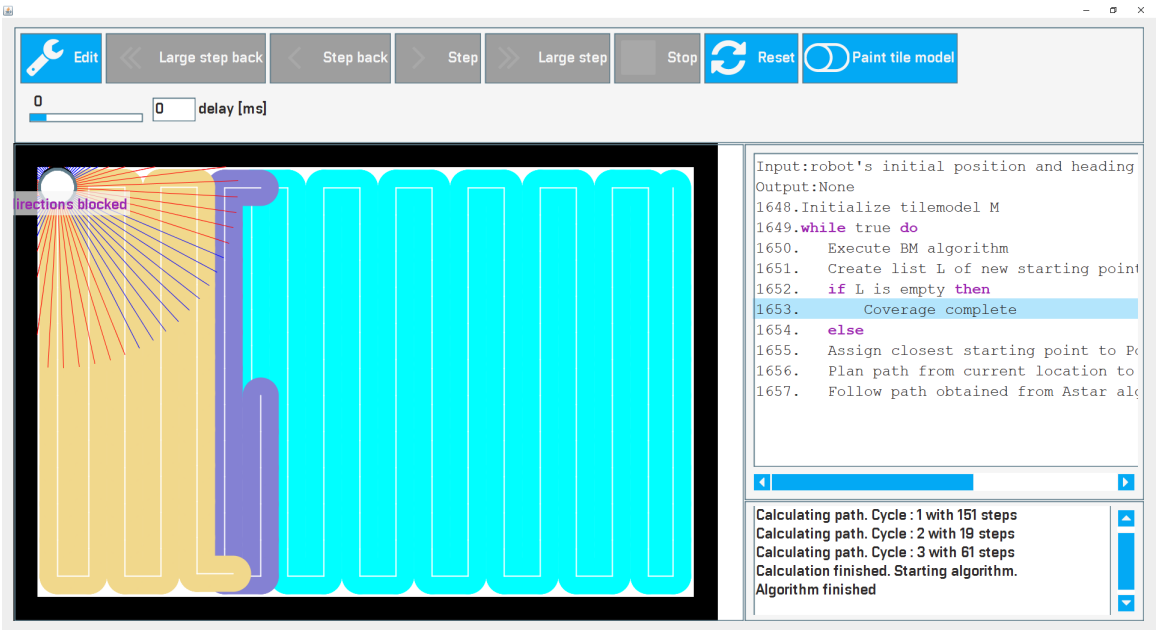


Obrázek 6.4: Vykreslení dlaždicového modelu

Kromě počtu vykreslených kroků nepřijímá algoritmus žádné parametery. Stejně jako u algoritmu náhodného pohybu, začne robot po spuštění uklízet daný prostor z jeho aktuální pozice. V ovládacích prvcích aplikace je možno vykreslit dlaždicový model robota, viz obrázek 6.4. V aplikaci je také zobrazena cesta nalezena algoritmem A^*, po které robot dorazí do nového startovního bodu, viz 6.4. Jednotlivé cykly boustrophedon pohybů jsou od sebe barevně odlišeny, viz obrázek 6.5.

6.2.3 Algoritmus topologického pokrytí

Posledním algoritmem, o jehož implementaci jsem se v bakalářské práci pokusil, je algoritmus topologického pokrytí 4.6. Kvůli nedostatku infromací k několika problémům, které algoritmus musí řešit se mi jej bohužel nepodařilo dokončit. Algoritmus je v rozpracovaném stavu. Zvládne detekovat překážky s pomocí senzorů. Robot v tomto algoritmu disponuje nejen mechanickým nárazníkem (opět černý okraj robota), ale i sonarovým prstencem, který detekuje překážky v úhlu 360° v okolí robota.



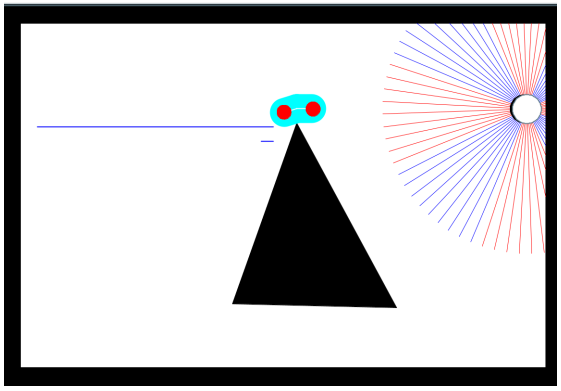
Obrázek 6.5: Zbarvení jednotlivých cyklů boustrophedon pohybů

Uživatel může zvolit vertikální a horizontální směr, kterým se robot začne pohybovat. Ten tímto směrem začne vykonávat boustrophedon pohyby a jakmile detekuje událost, prozkoumá hranici okolo této události a v paměti si vytvoří hrany grafu, při několikanásobném spuštění je však neumí na sebe korektně napojit. Stejně jako u algoritmu 6.2.2 je důležité, aby byl prostor, který robot pokrývá uzavřený. Pokud není prostor ohraničen ze všech stran, může se tak stát, že robot vyjede z prostoru ven a začne pokrývat nekonečný prostor, čímž se zacyklí.

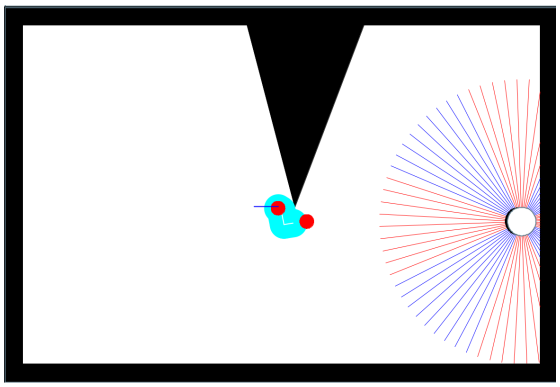
Detekce událostí dekompozice řezem, kterou algoritmus topologického pokrytí vytváří, je na obrázku 6.6, červená kolečka značí uzly grafu, modré čáry značí doposud pokrytou oblast, před detekcí události.

Podle algoritmu 4.6 by dále měl robot vybrat nejbližší neprozkoumanou buňku a v ní pokračovat v pokrývání prostoru. Problémy, které se mi nepodařilo vyřešit, jsou správná detekce již objevených bodů. Při pokrývání prostoru může robot na již objevenou událost, a tedy překážku, narazit z několika stran. V [22] je bohužel tato situace popsána pouze tak, že proběhne metrická, popřípadě topologická kontrola, zda byl již bod objeven.

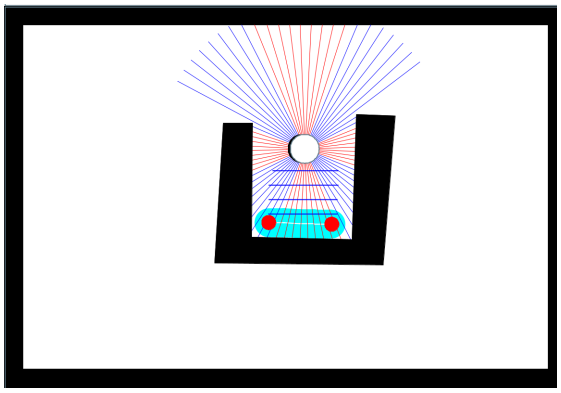
Další problémy, které by se po detekci již objevených bodů musely vyřešit, jsou správná aktualizace topologického grafu a detekce cyklů v něm.



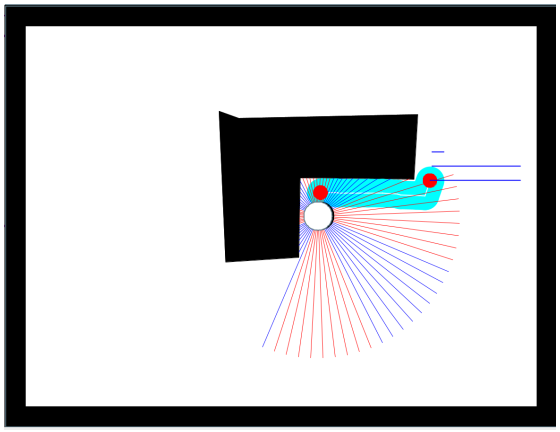
(a) Detekce události split



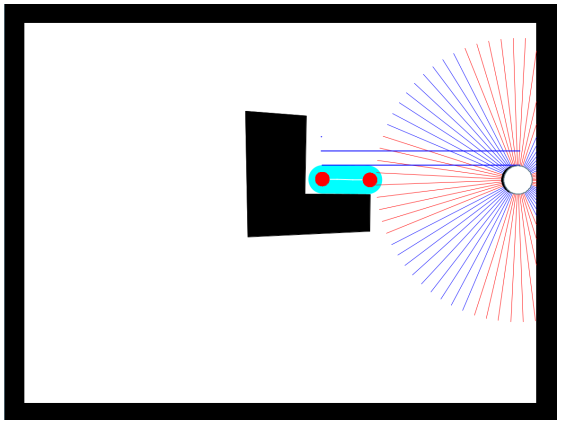
(b) Detekce události merge



(c) Detekce události end



(d) Detekce události lenghten



(e) Detekce události shorten

Obrázek 6.6: Jednotlivé události dekompozice řezem 4.5.2 detekované simulovaným robotem

Kapitola 7

Závěr

V rámci této práce byla vytvořena řešerše aktuální nabídky autonomních robotických vysavačů. Dále jsou v této práci obsaženy algoritmy kompletního pokrytí prostoru, jsou vybrány ty algoritmy, jejichž obdoby se v reálných robotických vysavačích nejspíše používají. Reálné algoritmy jsou však firemním tajemstvím, o jejich použití v reálných robotech můžu tedy pouze spekulovat.

Tři z představených algoritmů jsou, za pomocí knihovny vizlib, implementovány. Každý algoritmus je samostatně spustitelná aplikace, ve které je simulován pohyb robota s omezením reálných senzorů. Z poznatků z rešerše vyplává, že většina robotů disponuje mechanickým nárazníkem a senzorem, který umožní robotovi snímat překážky z dálky. V implementovaných algoritmech mají proto všichni roboti nárazník. Roboti, kteří vykonávají boustrophedon A^* a algoritmus topologického pokrytí mají navíc sonarový prstenec.

Algoritmus topologického pokrytí je v rozpracovaném stavu, navazující práce by tedy tento algoritmus doimplementovala. Mezi další vhodná rozšíření bych zařadil statistiky o pokrytém prostoru a také editor samotného robota, ve kterém by měl uživatel možnost změnit vlastnosti a rozmístění senzorů robota.

Literatura

[1] *Robotický vysavač*. [Online; navštíveno 5.12.2020]. Dostupné z: https://www.roboticky-vysavac.cz/.

[2] Acar, E. U. a Choset, H. Sensor-based Coverage of Unknown Environments: Incremental Construction of Morse Decompositions. *The International Journal of Robotics Research* [online]. 2002, sv. 21, č. 4, s. 345–366, [cit. 2021.10.05]. DOI: 10.1177/027836402320556368. Dostupné z: https://doi.org/10.1177/027836402320556368.

[3] Choset, H. *Principles of robot motion : theory, algorithms and implementations*. Massachusetts: MIT Press, 2005. Intelligent robotics and autonomous agents.

[4] Chung TaeChoong, V. H. H. a Dang Viet-Hung a Laskar Md Nasir Uddin a. BA*: an online complete coverage algorithm for cleaning robots. *Applied Intelligence* [online]. Září 2013, sv. 39, č. 2, s. 217–235, [cit. 10.5.2021.]. DOI: 10.1007/s10489-012-0406-4. ISSN 1573-7497. Dostupné z: https://doi.org/10.1007/s10489-012-0406-4.

[5] CleanMate. *CleanMate robotický vysavač*. [Online; navštíveno 4.5.2021]. Dostupné z: https://www.cleanmate.cz/.

[6] Cohen, D. A., Ozick, D., Vu, C., Lynch, J. a Mass, P. R. *Autonomous robot auto-docking and energy management systems and methods*. Červenec 2005.

[7] Detwiler, B. *Cracking Open the Roomba 980 robot vacuum - TechRepublic*. [Online; navštíveno 21.1.2021]. Dostupné z: https://www.techrepublic.com/blog/cracking-open/cracking-open-the-roomba-980-robot-vacuum/.

[8] Ecovacs. *Deebot ozmo 960*. [Online; navštíveno 3.1.2021]. Dostupné z: https://www.ecovacs.com/us/deebot-robotic-vacuum-cleaner/DEEBOT-OZMO-960.

[9] Gabriely, E. Spiral-STC: an on-line coverage algorithm of grid environments by a mobile robot. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation* [online]. IEEE, 2002, sv. 1, s. 954–960 [cit. 10.5.2021]. DOI: 10.1109/ROBOT.2002.1013479. Dostupné z: https://ieeexplore.ieee.org/document/1014684.

[10] Hasan, K. M., Abdullah Al Nahid a Reza, K. J. Path planning algorithm development for autonomous vacuum cleaner robots. [online]. Květen 2014, s. 1–6, [cit. 2021.10.05]. DOI: 10.1109/ICIEV.2014.6850799. Dostupné z: https://ieeexplore.ieee.org/document/6850799.

[11] iRobot. *How Does my Robot Navigate? | iRobot Customer Care.* [Online; navštíveno 20.1.2021]. Dostupné z: https://homesupport.irobot.com/app/answers/detail/a_id/19541/~/how-does-my-robot-navigate%3F.

[12] iRobot. *IRobot- specialista na robotiku.* [Online; navštíveno 16.12.2020]. Dostupné z: https://www.irobot.cz/.

[13] iRobot. *Roomba i7+.* [Online; navštíveno 4.1.2021]. Dostupné z: https://www.irobot.cz/produkt/roomba-i7-silver-stanice/.

[14] Jakub, R. *Vizualizace algoritmů pro plánování cesty.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/106120.

[15] JetBrains. *IntelliJ IDEA.* [Online; navštíveno 9.5.2021]. Dostupné z: https://www.jetbrains.com/idea/.

[16] Jones, J. L., Mack, N. E., Nugent, D. M. a Sandin, P. E. *Autonomous floor-cleaning robot.* Duben 2005.

[17] Lumelsky, V. J. a Stepanov, A. A. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica* [online]. 2. vyd., verze 1.0. Leden 1987, sv. 2, č. 1, s. 403–430, [cit. 2021.10.05]. DOI: 10.1007/BF01840369. ISSN 1432-0541. Dostupné z: https://doi.org/10.1007/BF01840369.

[18] Samsung. *Samsung Česká republika | Mobilní Telefony | Domácí Spotřebiče | TV.* [Online; navštíveno 4.5.2021]. Dostupné z: https://www.samsung.com/cz/.

[19] Symbo. *Robotický vysavač do každé domácnosti.* [Online; navštíveno 26.1.2021]. Dostupné z: https://www.symbo.eu/cs/.

[20] Wikipedia. *Lidar.* [Online; navštíveno 3.1.2021]. Dostupné z: https://cs.wikipedia.org/wiki/Lidar#P%C5%99%C3%ADklady_vyu%C5%BEit%C3%AD.

[21] Wikipedia. *Model–view–controller.* [Online; navštíveno 4.5.2021]. Dostupné z: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

[22] Wong, S. a MacDonald, B. A topological coverage algorithm for mobile robots. *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)* [online]. Srpen 2003, sv. 2, s. 1685–1690, [cit. 10.5.2021]. Dostupné z: https://ieeexplore.ieee.org/abstract/document/1248886.

[23] Zelinsky, A., Jarvis, R., Byrne, J. a Yuta, S. Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot. In: *In Proceedings of International Conference on Advanced Robotics* [online]. 1993, s. 533–538 [cit. 10.5.2021]. Dostupné z: https://www.semanticscholar.org/paper/Planning-Paths-of-Complete-Coverage-of-an-by-a-Zelinsky-Jarvis/98ce590b9e6ecd60a2ba59aa9d162a08c681f1c8.

Příloha A

Obsah přiloženého paměťového média

- `text prace` – složka obsahující vygenerované pdf s textem práce a zdrojové soubory latexu, společně s Makefilem pro jejich přeložení
- `completeCoverageAlgorithms` – složka se zdrojovými soubory všech tří aplikací
- `executables` – složka obsahující spustitelné soubory .jar jednotlivých aplikací

Příloha B

Přeložení aplikace

Aplikace vyžaduje minimálně Java 8. Novější verze by měly mít zpětnou kompatibilitu.

Přeložení aplikace je možné pomocí vývojového prostředí IntelliJ [15]. V tomto případě lze importovat obsah paměťového média složky `completeCoverageAlgorithms` jako nový projekt nebo lze složku přímo jako projekt otevřít. Projekt vyžaduje knihovnu vizlib [14], ta je součástí zdrojových souborů programu. Po otevření projektu je nutno vybrat, ze které třídy má byt spuštěna funkce `main()`.

Umístění funkcí `main()` jednotlivých algoritmů je následující:

- algoritmus náhodného pohybu – completeCoverageAlgorithms\src\main\randomAlg \RandomAlgFrame.java
- boustrophedon A^* – completeCoverageAlgorithms\src\main\bAstar \BAstarFrame.java
- algoritmus topologického pokrytí – completeCoverageAlgorithms\src\main \topologicalCoverageAlg\TopologicalCoverageFrame.java