VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

TESTOVÁNÍ PLÁNOVÁNÍ CESTY PRO ÚPLNÉ POKRYTÍ PROSTORU

TESTING OF COVERAGE PATH PLANNING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DAMIÁN KRAJŇÁK**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Krajňák Damián**
Program: Informační technologie
Název: **Testování plánování cesty pro úplné pokrytí prostoru**
Testing of Coverage Path Planning
Kategorie: Umělá inteligence

Zadání:

1. Nastudujte algoritmy pro plánování cesty. Zaměřte se na algoritmy používané pro kompletní pokrytí daného prostoru (tzv. coverage path planning).
2. Navrhněte program s grafickým uživatelským rozhraním pro testování těchto algoritmů. Program umožní vytvořit nebo z externího souboru nahrát tvar oblasti a pro ni vytvořit plán cesty. Umožněte vytvořit simulovaného robota (šířka, rychlost pohybu a zrychlení), který oblast projede a výsledky z jednotlivých testovacích běhů zobrazte, případně ukládejte.
3. Navržený program implementujte, dále implementujte více algoritmů a otestujte funkčnost.

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

Cieľom tejto práce je navrhnúť a implementovať program s grafickým užívateľským rozhraním, určený k testovaniu algoritmov pre úplné pokrytie priestoru. Problém som riešil pomocou rozkladu 2D priestoru na bunky, ktoré sú postupne prechádzané. Oblasti jednotlivých buniek sú pokrývané spôsobom, určeným na základe zvoleného algoritmu. Vytvorené riešenie poskytuje návrh optimálnej trasy pre robotov určených k úplnému pokrytiu priestoru, akými sú napríklad autonomné vysávače, čistiace roboty, robotické kosačky a iné.

Abstract

The aim of this thesis is to design and implement a program with graphical user interface, designed for algorithm testing and complete coverage of a given area. Problem was solved by decomposition of a 2D space into cells, which were sequentially covered. The areas of individual cells are covered in a way, that is selected based on the selected algorithm. Created solution providese a draft of an optimal pathway for robots intended for complete coverage of space, which are, for example, autonomous vacuum cleaners, cleaning robots, robotic lawn mowers etc.

Kľúčové slová

plánovanie cesty s úplným pokrytím priestoru, algoritmy pre plánovanie cesty, pohyb robota, bunková dekompozícia, graf susednosti, vizualizácia algoritmov, testovanie algoritmov, vlastnosti robota

Keywords

coverage path planning, path planning algorithms, robot motion, cell decomposition, adjacency graph, visualization of algorithms, testing of algorithms, robot properties

Citácia

KRAJŇÁK, Damián. *Testování plánování cesty pro úplné pokrytí prostoru.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Testování plánování cesty pro úplné pokrytí prostoru

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jaroslava Rozmana, PhD. Uviedol som všetky literárne pramene, publikácie a dalšie zdroje, z ktorých som čerpal.

.....
Damián Krajňák
21. mája 2021

Poďakovanie

Chcel by som poďakovať pánovi Ing. Jaroslavovi Rozmanovi Ph.D., za vedenie a pomoc pri mojej práci.

Obsah

Kapitola 1

Úvod

Plánovanie cesty robota, je jedným z kľúčových aspektov robotiky. Môžeme ho definovať ako výpočtový problém, kde sa snažíme určiť postupnosť validných krokov, ktoré zaručia presun objektu zo štartovacej pozície, do pozície cieľovej. Sú však situácie, kedy od robota vyžadujeme, aby touto trasou zároveň pokryl celý priestor, v ktorom operuje. Takúto cestu označujeme ako cestu s úplným pokrytím priestoru. S robotmi, ktoré využívajú práve tento princíp, sa stretávame relatívne často v každodennom živote. Jedná sa napríklad o robotov určených ku koseniu trávy, čisteniu okien, kombajny, autonómne vysávače, čističe bazénov a mnoho ďalších.

Algoritmov k nájdeniu cesty s úplným pokrytím priestoru existuje viacero. Keďže sa voľba takéhoto algoritmu odzrkadľuje na efektívnosti daného robota, je našou prirodzenou motiváciou zvoliť čo najefektívnejší algoritmus, či už z časového alebo energetického hľadiska. Avšak, označiť niektorý z algoritmov za všeobecne najefektívnejší, by bolo unáhlené. To, ktorý algoritmus je pre konkrétny problém optimálny ovplyvňuje viacero faktorov, ako napríklad tvar, veľkosť a typ robota, spôsob akým sa robot pohybuje, veľkosť a hustota prekážok v jeho pracovnom priestore a mnoho ďalších. Cieľom tejto práce je vytvoriť plánovač, ktorý dokáže tieto skutočnosti zohľadniť, a na základe nich naplánovať pre robota optimálnu či suboptimálnu cestu.

Kapitola 2 popisuje niektoré základné algoritmy vyhľadávania cesty. Kapitola takisto opisuje základné vlastnosti robota a jeho pohybové vzory. Kapitola 3 popisuje bunkovú dekompozíciu – spôsob rozdelenia konfiguračného priestoru robota na geometricky jednoduchšie celky, nazývané bunky. Táto kapitola takisto popisuje samotné spôsoby, akými robot tieto menšie celky pokrýva. V kapitole 4 je popísaný celkový návrh aplikácie, pričom samotné užívateľské rozhranie aplikácie je popísané až v kapitole 5. Kapitola 6 dopodrobna popisuje architektúru aplikácie. Kapitola takisto vysvetľuje spôsob implementácie jednotlivých algoritmov v programe. Kapitola 7 je poslednou kapitolou tejto práce. Detailne popisuje priebeh jednotlivých fáz testovanie spolu s výstupmi týchto testov.

Kapitola 2

Vlastnosti robota

Úplné pokrytie priestoru využíva veľké spektrum robotov, akými sú napríklad roboty určené k čisteniu podláh, okien alebo dien bazénov, no taktiež sa môže jednať o robotické kosačky či iné hospodárske stroje väčších rozmerov.
Keďže každý z týchto robotov pracuje v inom prostredí a disponuje inými vlastnosťami či požiadavkami, je pochopiteľné že tvar a spôsob pohybu každého robota je často úplne odlišný. Plánovač cesty robota musí s touto skutočnosťou počítať, nakoľko nevyhnutnou podmienkou každého plánovača cesty robota je to, aby bol robot schopný naplánovaný pohyb vykonať. V opačnom prípade je takýto plánovač cesty prakticky nepoužiteľný.

Táto práca bude pre jednoduchosť simulovať robota kruhového tvaru, ktorý sa dokáže otáčať okolo vlastnej osi. Tieto vlastnosti sú charakteristické pre robotické vysávače, akými sú napríklad model Roomba i7 (Obr. 2.1a) od spoločnosti iRobot, no taktiež pre mnoho iných robotov, ako napríklad robot TurtleBot 2 (Obr. 2.1b).

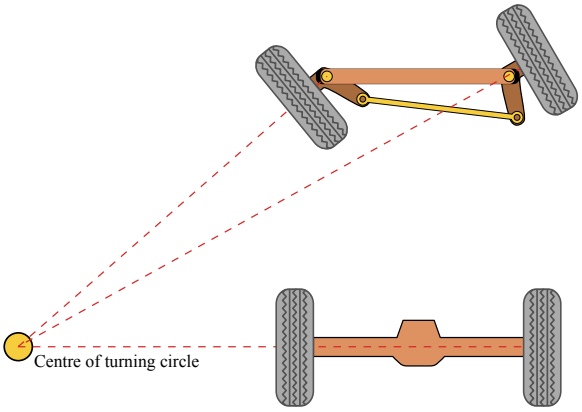
(a) **iRobot Roomba i7**
Obrázok prevzatý z [2].

(b) **TurtleBot 2**
Obrázok prevzatý z [1].

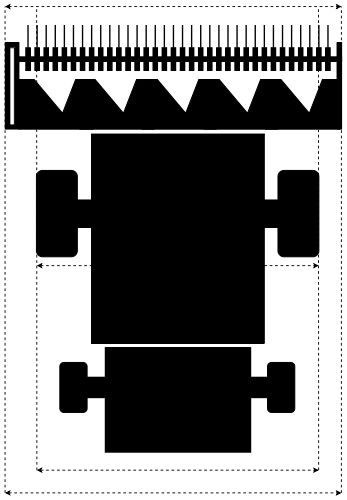
Obr. 2.1

Je však vhodné uviesť aj iné typy robotov či strojov, ktoré pri svojej činnosti vykonávajú úplné pokrytie priestoru. Väčšina robotov má zložitejší tvar ako modely ilustrované na obrázku 2.2, čo často implikuje zložitejší pohyb robota. Najbežnejším príkladom je osobný

automobil, ktorý má polomer otáčania typicky 10,5 m[11], z čoho vyplýva nemožnosť opísať každú cestu (Obr. 2.2a). Zatiaľ čo automobil má typicky jednoduchý tvar, pohyb je zložitejší pri robotoch nepravidelných tvarov. Typicky sa jedná o hospodárske stroje, ktoré v prednej časti obsahujú adaptér, ktorý je širší ako samotná šírka tohoto stroja (Obr. 2.2b). Práve to robí jednoduchý pohyb "tam s spať"(Obr. 3.1) pre tieto stroje zložitejší, nakoľko sa tieto stroje nedokážu otočiť na mieste, no musia si v mieste, kde sa potrebujú otočiť nadbehnúť, prípadne pásy na striedačku vynechávať a pokryť ich cestou späť.



(a) Nákres ilustrujúci polomer otáčania osobného automobilu. Dĺžky prerušovaných čiar smerujúcich z kolies prednej nápravy a končiacich v mieste, kde pretnú os otáčania zadných kolies predstavujú vnútorný a vonkajší polomer otáčania vozidla. Tieto prerušované čiary sú osi otáčania jednotlivých kolies. Keďže sa otáča len jedna náprava, vozidlo nie je schopné otočiť sa okolo vlastnej osi. Obrázok prevzatý z [11].



(b) Obrázok ilustrujúci nepravidelný tvar stroja. Žací adaptér upevnený v prednej časti kombajnu je širší ako samotný stroj, tým pádom je menévrovanie náročnejšie.

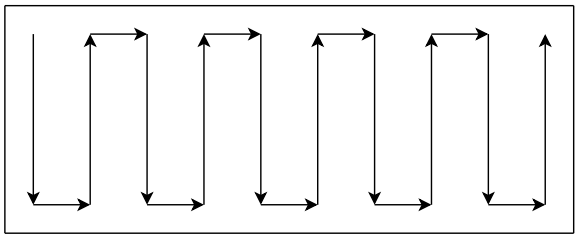
Obr. 2.2

Ako však bolo spomenuté v predošlom odseku, táto práca bude zameraná na plánovanie cesty pre kruhové roboty, s možnosťou otáčania sa okolo vlastnej osi.

Kapitola 3

Bunková dekompozícia

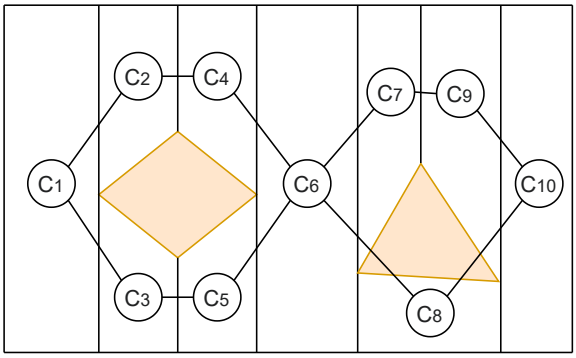
Bunkovou dekompozíciou, nazývame proces, pri ktorom rozdelíme voľný pracovný priestor na menšie časti, nazývané bunky. Jednotlivé bunky sa pritom neprekrývajú a výsledkom ich zlúčenia je opäť pôvodný voľný pracovný priestor [6]. Výhodou rozdelenia priestoru na bunky je fakt že pri vhodne zvolenom spôsobe dekompozície, sú výsledné bunky jednoduchšie na pokrytie. Často je možné priestor jednotlivých buniek pokryť jednoduchým pohybom "tam a späť".



Obr. 3.1: **"ZigZag".** Ilustrácia klasického pohybu "tam a späť".

3.1 Graf susednosti

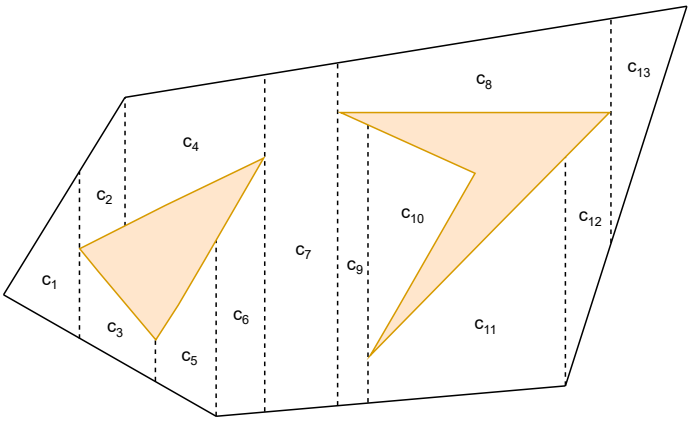
Ako reprezentácia bunkovej dekompozície často slúži graf susednosti. Uzly tohoto grafu predstavujú jednotlivé bunky. Hrany grafu vyjadrujú vzťah susednosti jednotlivých buniek. Dve bunky sú susedné, ak majú nejakú spoločnú hranicu. Po vytvorení grafu susednosti buniek z dekompozície je možné nájsť najkratšiu cestu prechádzajúcu všetkými uzlami grafu. Typicky na to slúžia vyhľadávacie algoritmy do hĺbky [10]. Ak je algoritmus úspešný, komplexný problém pokrytia celého priestoru je tak rozdelený na 2 body – Prejsť priestor podľa tejto sekvencie buniek a následne každú prechádzanú bunku pokryť – napríklad spôsobom ilustrovaným na obrázku 3.1.



Obr. 3.2: **Graf susednosti.** Dekompozícia(Lichobežníková) prostredia a jej zodpovedajúci graf susednosti.

3.2 Lichobežníková dekompozícia

Lichobežníková dekompozícia [5] je jedna z najjednoduchších bunkových dekompozícii. Táto metóda je je určená pre polygonálne priestory v rovine. Metóda takisto vyžaduje, aby všetky prekážky v priestore boli polygóny. Ako napovedá jej názov, bunky vytvorené touto dekompozíciou majú tvar lichobežníka, v špeciálnych prípadoch tvar trojuholníka.
Pre úplné pokrytie priestoru použitím lichobežníkovej dekompozície, je potrebné nájsť cestu grafom susednosti odvodeného od tejto dekompozície tak, aby cesta prešla cez každý uzol grafu aspoň raz. Každú bunku je potom možné pokryť jednoduchých pohybom "tam a späť", ilustrovaných na obrázku 3.1. Tento pohyb v jednotlivých bunkách pozostáva z opakujúcej sa sekvencie priamočiarych pohybov rovnobežných s vertikálnym predĺžením vrchola danej bunky, s rozostupmi vzdialenosti šírky robota. Tieto krátke segmenty, spájajúce dlhé priamočiare segmenty, typicky kopírujú tvar prostredia.



Obr. 3.3: **Lichobežníková dekompozícia.** Ilustrácia lichobežníkovej dekompozície.

Zostrojenie lichobežníkovej dekompozície

Lichobežníkovú dekompozíciu je možné zostrojiť tak, že každému vrcholu vytvoríme tzv. horné vertikálne predĺženie, prípadne dolné vertikálne predĺženie [5]. To dosiahneme pre-

chodom zvislej čiary, nazývanej rez [5], celým priestorom pozdĺž x-ovej osi. Ak rez narazí na vrchol, v danom mieste vytvoríme odpovedajúce vertikálne predĺžením čím vzniká nová bunka. Vrchol tak môže mať spodné vertikálne predĺženie, horné vertikálne predĺženie, prípadne oba súčasne. Vertikálne predĺženie začína na vrchole niektorej z prekážok v pracovnom priestore, prípadne na vrchole ohraničenia pracovného priestoru. Predĺženie končí na mieste, kde pretne niektorú z prekážok alebo hranicu pracovného priestoru.

Udalosť, kedy rez narazí na vrchol delíme na 3 typy. Názvy týchto udalostí sú prevzaté z [5]. Udalosť `IN`, udalosť `OUT` a udalosť `MIDDLE`. K udalosti `IN` dochádza, keď sa jedna bunka uzatvorí a 2 nové bunky sa otvoria. K udalosti `OUT` dochádza, keď sa uzatvoria 2 bunky a otvorí sa jedna nová bunka. Udalosť `MIDDLE` znamená, že sa práve jedna bunka uzatvára a jedna otvára. K tejto udalosti dochádza práve vtedy, ak sa vytvára vertikálne predĺženie iba do jednej strany (hore alebo dole).

Táto metóda je často neefektívna, práve kvôli veľkému množstvu vytvorených buniek udalosťou `MIDDLE`. Tieto bunky je často krát výhodné spojiť do jednej. Práve tento princíp využíva nasledujúci typ bunkovej dekompozície.

x_L = *pozice řezu*
$\{x_1, x_2 \ldots x_n\}$ = *seznam x − souřadnic vrcholů*
$D = (\ldots, c_{i-1}\, c_i, c_{i+1}, \ldots)$ = *seznam buněk*
for $x_L = x_1$ *to* x_n ***do***
    if *vrchol rozděluje buňku* c_i *na dvě*
        then c_i *nahraď* c_d, c_{d+1}
    else if *vrchol slučuje dvě buňky* c_i *a* c_{i+1}
        then c_i *a* c_{i+1} *nahraď za* c_d
    else if *vrchol nahrazuje buňku ci novou buňkou*
        then c_i *nahraď* c_f
    end if
end for

Obr. 3.4: Algoritmus pre vytvorenie Lichobežníkovej dekompozície prebratý z práce [9].

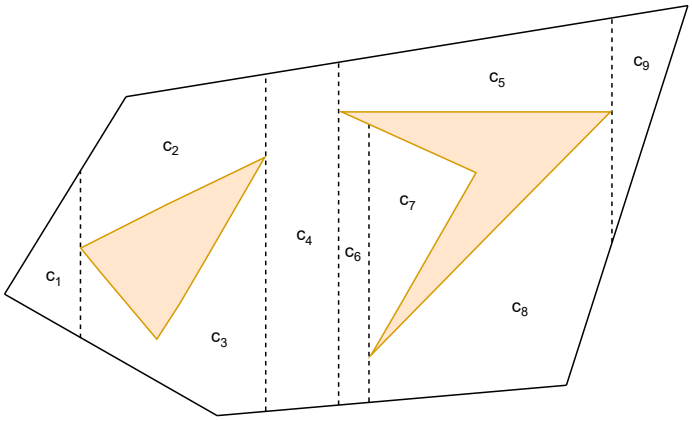
3.3 Bustrofedónová dekompozícia

Názov tejto dekompozície je odvodený z gréckeho *boustrophēdón*, čo v doslovnom preklade znamená "spôsob pohybu vôla", narážajúc na trasu, ktorú vykonáva vôl ťahajúci pluh na poli [4]. Táto metóda je tak pomenovaná z dôvodu že bunky vytvorené touto dekompozíciou je možné pokryť práve týmto jednoduchým pohybovým vzorom, ilustrovaným na obrázku 3.1.

x_L = *pozice řezu*
$\{x_1, x_2 \ldots x_n\}$ = *seznam* x − *souřadnic vrcholů*
$D = (\ldots, c_{i-1}\ c_i, c_{i+1}, \ldots)$ = *seznam buněk*
for $x_L = x_1$ *to* x_n ***do***
    if *vrchol rozděluje buňku* c_i *na dvě*
        then c_i *nahraď* c_d, c_{d+1}
    else if *vrchol slučuje dvě buňky* c_i *a* c_{i+1}
        then c_i *a* c_{i+1} *nahraď za* c_d
    end if
end for

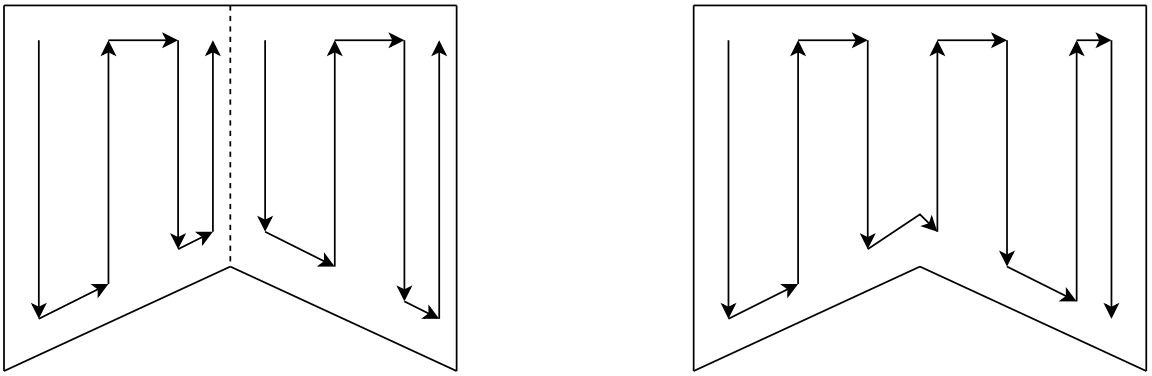
Obr. 3.5: Algoritmus pre vytvorenie Bustrofedónovej dekompozície prebratý z práce [9].

Bustrofedónová dekompozícia [4] je veľmi podobná Lichobežníkovej dekompozícii spomínanej vyššie, avšak berie do úvahy iba tie vrcholy, z ktorých je možné vytvoriť horné aj dolné vertikálne predĺženie - vytvára bunky iba pri `IN` a `OUT` udalostiach. Hoc sú takto vytvorené bunky na rozdiel od Lichobežníkovej dekompozície nekonvexné, stále je možné pokryť ich jednoduchým pohybovým vzorom. Nasledujúci obrázok (Obr. 3.6) ilustruje rovnaký priestor ako je na obrázku 3.3, tentokrát však rozdelený Bustrofedónovou dekompozíciou.



Obr. 3.6: **Bustrofedónová dekompozícia.** Priestor z obrázku 3.3, rozdelený Bustrofedónovou dekompozíciou

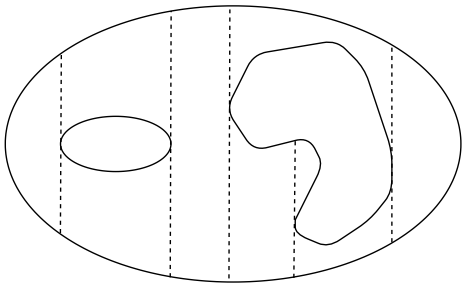
Výhoda tohoto prístupu spočíva v tom, že pokrytie rovnakého priestoru, rozdeleného do menšieho počtu buniek môže byť efektívnejšie, nakoľko robot prechádza väčší priestor kontinuálne, bez potreby častého nastavovania sa do počiatočnej pozície. Tento princíp je ilustrovaný na nasledujúcom obrázku 3.7, kde môžeme vidieť, o koľko sa skrátila cesta, ak priestor nie je zbytočne rozdelený na 2 bunky.



Obr. 3.7: **Lichobežníková a Bustrofedónová dekompozícia.** Na obrázku môžeme vidieť porovnanie rozdelenia toho istého priestoru Lichobežníkovou dekompozíciou (naľavo) a Bustrofedónovou dekompozíciou (napravo).

3.4 Morseova dekompozícia

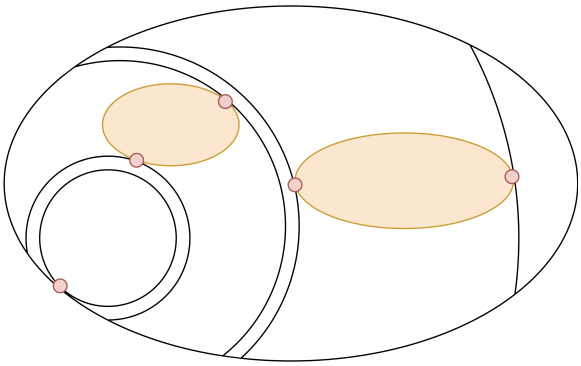
V práci [3] bolo zistené, že Bustrofedónová dekompozícia, je akýsi špecifický prípad viac komplexnej dekompozície, nazývanej Morseova dekompozícia. Tento typ dekompozície využíva kritické body Morseovej funkcie, definovanej v [8]. Výhodou Morseovej dekompozície oproti Lichobežníkovej či Bustrofedónovej dekompozície je fakt, že dokáže pracovať s priestorom obsahujúcim prekážky ktoré nie sú polygonálne. Tvar buniek vzniknutých touto dekompozíciou je variabilný. Závisí od Morseovej funkcie, ktorá bola pri dekompozícii použitá. Najjednoduchším príkladom takejto funkcie je funkcia $h(x, y) = x$ (Obr. 3.8). S použitím tejto funkcie vznikne dekompozícia zhodná z Bustrofedónovou dekompozícioum, avšak s tým rozdielom, že túto dekompozíciu je možné uplatniť aj v priestore s nepolygonálnymi prekážkami.



Obr. 3.8: Morseova dekompozícia s použitím Morseovej funkcie $h(x, y) = x$.

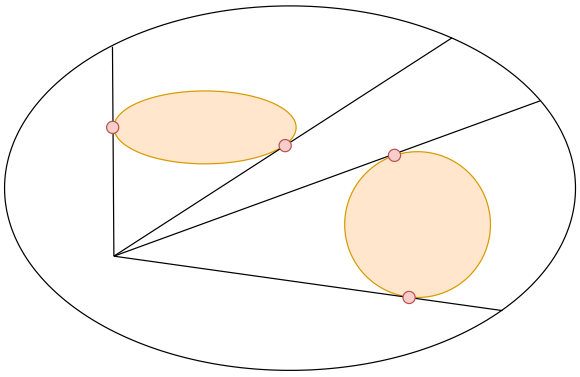
Samotnú dekompozíciu zostrojíme potiahnutím rezu naprieč daným priestorom. Miesto, kde sa mení konektivita tohto rezu, označíme ako kritický bod. Inak povedané je to miesto, kde rez narazí na prekážku, t.j. jeho konektivita sa zväčší, alebo miesto, kde rez dokončí prechod cez prekažku, tj. jeho konektivita sa zmenší. Tvar tohto rezu môže byť rôzny. Odvíja sa to od konkrétnej typu Morseovej funkcie, ktorá bola pri danej Morseovej dekompozícii použitá. Morseova teória [8] nám zaručuje, že topológia bunky ohraničenej takto vytvorenými kritickými bodmi sa nemení (t.j. nemení sa počet bodov na prieniku obrysových kriviek a priamky tvoriacej smer rezu [7]), takže je možné jej priestor pokryť jednoduchým pohybovým vzorom.

Pri použití Morseovej funckie $h(x, y) = \sqrt{x^2 + y^2}$ dosiahneme rezy, ktoré majú tvar sústredných kružníc (Obr. 3.9), z čoho vyplýva aj tvar rozdelenie priestoru na sústredné kružnice. Veľkosť týchto kružníc určujú kritické body, keďže práve to sú miesta, kde sa kružnice dotýkajú prekážok či hranice priestoru. Bunky vytvorené touto Morseovou funkciou zvyčajne nie je možné pokryť jednoduchým pohybovým vzorom "tam a spať", ilustrovaným na obrázku 3.1, keďže tvary buniek nie sú definované jednoduchými čiarovými segmentami. Pri takomto rozdelení priestoru robot zvyčajne vykonáva špirálovitý pohyb spôsobom, že opisuje kružnicu ktorej polomer zväčší o šírku robota práve vtedy, ak sa ocitne na mieste kde kružnicu opisovať začal. V prípade že robot narazí na prekážku, obchádza ju dovtedy, kým sa polomer nezväčší o šírku robota. Následne robot opisuje novú kružnicu s väčším polomerom.



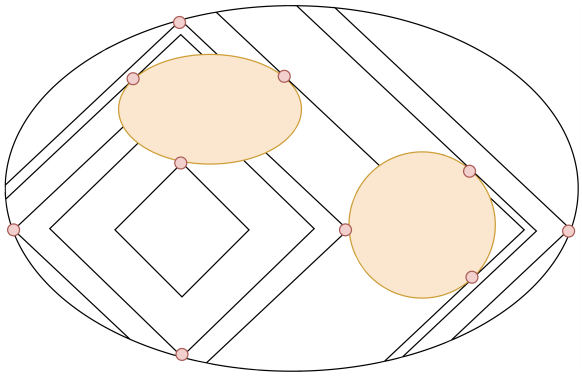
Obr. 3.9: Morseova dekompozícia s použitím Morseovej funkcie $h(x, y) = \sqrt{x^2 + y^2}$.

Použitie Morseovej funkcie $h(x, y) = \tan(\frac{y}{x})$ rozdeľuje priestor podľa čiar, ktoré sú kolmé k sústredným kružniciam z predošlého rozdelenia (Obr. 3.9). Keďže sa všetky tieto čiary sústreďujú do jedného stredu, je pochopiteľné, že ich vzdialenosť od seba bude menšia čím bližšie k tomuto stredu sme. Nakoľko má robot nenulovú šírku, miesto, kde sa tieto čiary stretávajú robot pokryje hustejšie, t.j. s väčším prekrytím. Túto vlastnosť je výhodné použiť práve vtedy, aj je robot informovaný o časti priestoru, ktorú je potrebné pokryť dôkladnejšie či opakovane.



Obr. 3.10: Morseova dekompozícia s použitím Morseovej funkcie $h(x, y) = \tan(\frac{y}{x})$.

Podobnú dekompozíciu ako v predošlom príklade (Obr. 3.9) indukuje Morseova funkcia $h(x, y) = |x| + |y|$. Použitím takejto Morseovej funkcie vznikne rozdelenie na bunky, ktoré svojim tvarom pripomínajú štvorce pootočené o 45°. O takejto dekompozícii môžeme uvažovať ako o aproximácii dekompozícii z predošlého príkladu (Obr. 3.9), ktorá používala sústredné kružnice. Takáto dekompozícia môže byť vhodná pre robotov, pre ktorých je vykonávanie pohybu opisujúceho kružnicu náročné.



Obr. 3.11: Morseova dekompozícia s použitím Morseovej funkcie $h(x, y) = |x| + |y|$.

Kapitola 4

Návrh riešenia

Táto kapitola obsahuje návrh programu, ktorý bude slúžiť k testovaniu rozličných algoritmov plánovania cesty pre úplné pokrytie priestoru uvedených v predošlej kapitole 3. Program nebude pracovať so všetkými algoritmami uvedenými v tejto práci, nakoľko týchto algoritmov je nemalé množstvo a implementácia všetkých týchto algoritmov predstavuje značne komplexnejší problém. Základné algoritmy vyhľadávania cesty pre úplné pokrytie priestoru však v programe implementované sú.

4.1 Návrh aplikácie

Cieľom navrhovanej aplikácie je testovať rôzne algoritmy, používané k nájdeniu cesty s úplným pokrytím priestoru. Konkrétne sa jedná o vizualizáciu týchto algoritmov a ich následné vyhodnotenie. Keďže plánovacie algoritmy tohto typu sú zväčša používané pozemnými robotmi či strojmi, aplikáciu som sa rozhodol navrhnúť tak, aby vykresľovala tieto algoritmy v 2D priestore.

4.2 Použité technológie

Program som sa rozhodol vytvoriť s použitím objektovo orientovaného programovania, konkrétne v programovacom jazyku Java, s použitím aplikačnej platformy JavaFX. K automatizácii zostavovania som použil nástroj Gradle. Grafické užívateľské rozhranie som tvoril s pomocou nástroja JavaFX Scene Builder. Ako formát externého súboru z ktorého sú serializované dáta načítavané som zvolil YAML. Prvky užívateľského rozhrania sú popísané v súbore s formátom FXML.

4.3 Vstupy aplikácie

Vstupy aplikácie bude užívateľovi umožnené zadať interaktívne priamo v užívateľskom rozhraní, no aplikácia bude takisto schopná tieto vstupy načítať z externého súboru. Týmito vstupmi sú:

- **Prostredie, v ktorom robot hľadá cestu** – Toto prostredie bude môcť užívateľ zadať buď priamo v aplikácii pridávaním okrajových hrán prostredia pomocou myši, prípadne bude prostredie načítané z externého súboru, kde je popísané pomocou jeho vrcholov.

- **Vlastnosti robota** – Týmito vlastnosťami sú šírka robota, maximálna rýchlosť a zrýchlenie. Tieto parametre sú zadávané v bežne používaných metrických jednotkách (cm, km/h, m/s^2), ktoré sa neskôr konvertujú na jednotky s ktorými bude pracovať samotný program.
- **Voľba algoritmu** – Typ algoritmu si užívateľ zvolí spomedzi množiny algoritmov ktoré mu budú ponúknuté v užívateľskom rozhraní.

Tieto tri parametre programu bude možné zvoliť jedine pred spustením samotnej simulácie pokrývania priestoru robotom. Aplikácia však bude podporovať ešte jeden parameter, ktorý bude možné meniť za behu simulácie. Týmto parametrom je:

- **Rýchlosť simulácie** – Užívateľ bude pomocou tohoto parametru schopný meniť rýchlosť simulácie v reálnom čase.

4.4 Výstupy aplikácie

Keďže navrhovaná aplikácia je vo svojej podstate aplikácia určená k testovaniu, výstupy predstavujú jej neodmysliteľnú časť. Výstupom aplikácie bude vyhodnotenie práve skončeného behu pokrývania priestoru. Toto vyhodnotenie bude zobrazené užívateľovi po skončení simulácie prostredníctvom vyskakovacieho okna v užívateľskom rozhraní. Medzi tieto výstupy patria:

- **Doba trvania** – Časový údaj reprezentujúci dobu, počas ktorej robot pokryl daný priestor.
- **Obsah plochy prostredia** – Plocha prostredia vyjadrená v m^2.
- **Percento pokrytia** – Percentuálne vyjadrenie veľkosti plochy, ktorú sa robotovi podarilo pokryť.
- **Prejdená vzdialenosť** – Tento údaj predstavuje vzdialenosť, ktorú robot počas daného behu urazil.

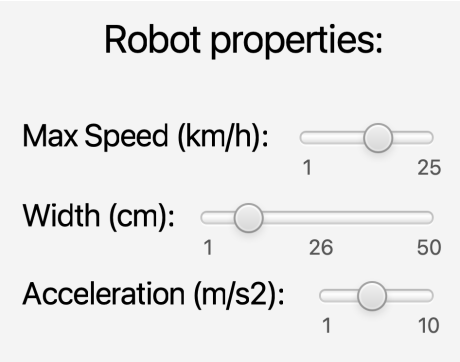
Kapitola 5

Užívateľské rozhranie

Táto kapitola detailne popisuje jednotlivé prvky grafického užívateľského rozhrania. Kapitola takisto slúži ako návod k obsluhe aplikácie.

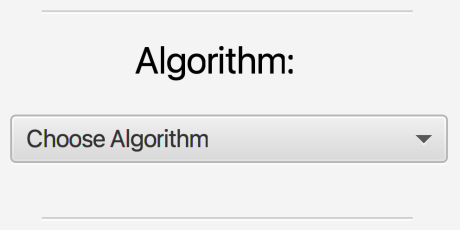
5.1 Postranný ovládací panel

Približne štvrtinu celkového užívateľského rozhrania zaberá postranný ovládací panel. Hlavným účelom tohoto panelu je zadávanie vstupov aplikácie. Táto sekcia rozoberá jednotlivé prvky tohoto panela a takisto vysvetľuje ich účel.



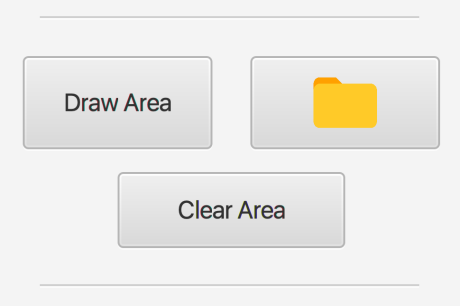
Obr. 5.1: Časť postranného ovládacieho panelu, slúžiaca k nastaveniu parametrov robota.

Obrázok vyššie (Obr. 5.1) znázorňuje hornú časť postranného ovládacieho panelu, ktorá slúži k nastaveniu parametrov robota. Ako už bolo popísané v kapitole 4, týmito parametrami sú maximálna rýchlosť robota(v km/h), šírka robota(priemer v cm) a zrýchlenie robota(v m/s^2). Každý z týchto vstupov sa v užívateľskom rozhraní nastaví posúvaním slidera. Rozsah hodnôt každého slidera je limitovaný. Minimálna hodnota každého z týchto parametrov je nastavená na 1. Maximálna hodnota je zvolená tak, aby užívateľ nebol schopný nastaviť niektorý z parametrov robota na prehnane veľkú hodnotu, čím by simulácia neodpovedala realite. Zmena každého z týchto parametrov sa aplikuje po pustení posuvníka.



Obr. 5.2: Časť postranného ovládacieho panelu, slúžiaca k zvoleniu algoritmu.

Obrázok vyššie (Obr. 5.2) znázorňuje časť postranného ovládacieho panelu, ktorá slúži k zvoleniu algoritmu, ktorý bude robot pri pokrývaní priestoru využívať. Algoritmus je možné zvoliť po rozkliknutí dropdown menu, kedy sa užívateľovi následne zobrazí ponuka dostupných algoritmov. Po kliknutí na niektorý z algoritmov sa zmena aplikuje.



Obr. 5.3: Časť postranného ovládacieho panelu, slúžiaca k zvoleniu vytvoreniu a následnému uloženie prostredia.

Na obrázku (5.3) je možné vidieť časť postranného panelu, ktorá je zodpovedná za vytvorenie či načítanie prostredia, v ktorom bude robot pracovať. Táto časť je takisto zodpovedná za zmazanie aktuálneho prostredia z hlavnej scény. Nachádzajú sa tu tri tlačidlá. Tlačidlo Draw Area slúži k spusteniu interaktívnej tvorby prostredia. Tlačidlo s ikonou priečinku načíta prostredie z externého súboru. Pomocou tlačidla Clear Area užívateľ zmaže aktuálne prostredie vykreslené v hlavnej scéne aplikácie.

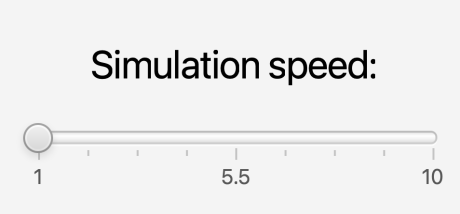
Obr. 5.4: Ilustrácia vytvárania prostredia pridávaním hrán klikaním myši.

Na obrázku (5.4) je znázornený spôsob tvorby prostredia. Užívateľ klikaním ľavého tlačidla myši pridáva jednotlivé hrany prostredia. Miesto kam práve užívateľ klikne predstavuje koncový bod predošlej hrany a začiatočný bod nasledujúcej hrany. Keďže aplikácia pracuje iba s pravými uhlami, rozhranie užívateľa v tomto smere čiastočne limituje. Užívateľ je schopný pridať iba takú hranu, ktorá s predošlou zviera pravý uhol. Uplatniť algoritmy

používané v tejto práci na prostredie s ľubovoľnými uhlami predstavuje výrazne zložitejší problém. Zároveň však táto podmienka nijako nelimituje drvivú väčšinu prostredí, ktoré samé o sebe obsahujú výlučne pravé uhly. Po pridaní poslednej hrany užívateľ ukončí tvorbu prostredia stlačením pravého tlačidla myši. V prípade že užívateľom zadané prostredie nie je uzatvorené, program automaticky poupraví dĺžku niektorých hrán, aby uzatvorené bolo.

Obr. 5.5: Tlačidlá na spustenie a pozastavenie simulácie.

Tlačidlá zobrazené na obrázku 5.5 sú umiestnené na postrannom ovládacom panely priamo pod nastavovaním prostredia aplikácie. Nachádzajú sa tú dve tlačidlá – tlačidlo Start simulation a tlačilo Stop simulation . Stlačením tlačidla Start simulation sa spustí simulácia. Takisto sa zablokujú všetky ostatné vstupy aplikácie, s výnimkou tlačidla Stop simulation . Toto tlačidlo simuláciu pozastaví a opäť sprístupní užívateľovi meniť vstupné parametre aplikácie. Tlačidlo takisto resetuje všetky meradlá - časomiera, prejdená vzdialenosť a percento pokrytia.

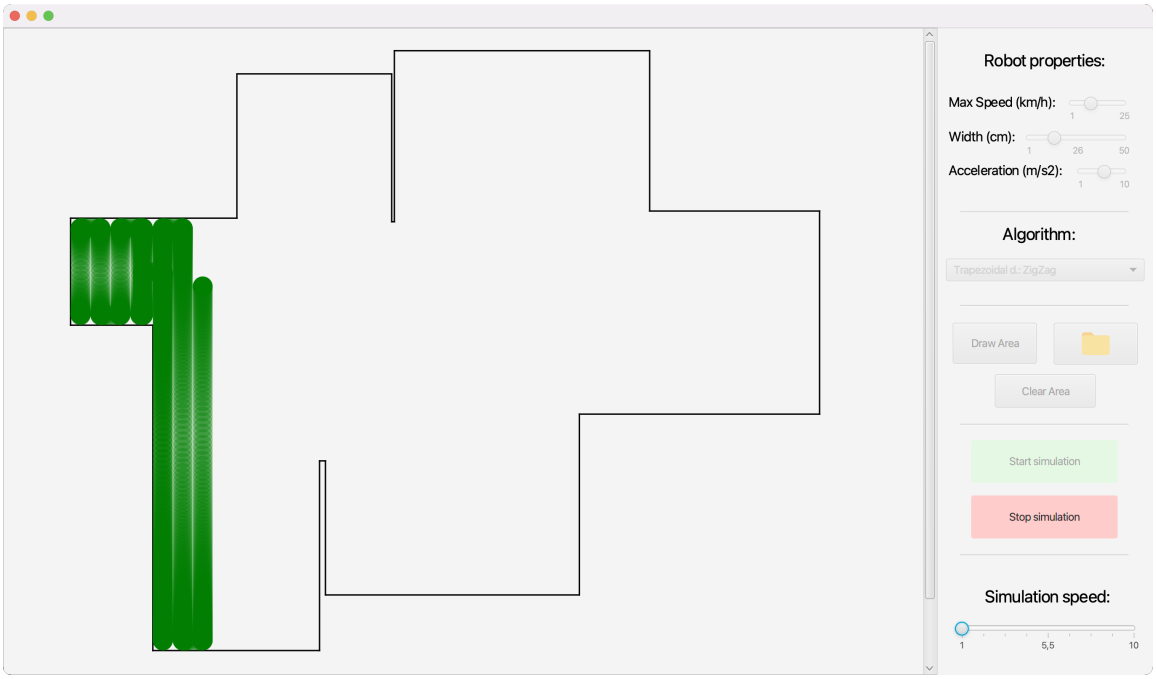


Obr. 5.6: Slider, umožňujúci modulovať rýchlosť simulácie počas jej behu.

Na obrázku vyššie (Obr. 5.6) je zachytená posledná časť postranného ovládacieho panelu. Nachádza sa v jeho spodnej časti. Obsahuje slider, ktorým užívateľ nastaví koeficient zrýchlenia simulácie. Tento vstup predstavuje jediný parameter aplikácie, ktorý je možné meniť priamo za behu simulácie. Zmena rýchlosti simulácie sa aplikuje po pustení posúvača.

5.2 Hlavná scéna

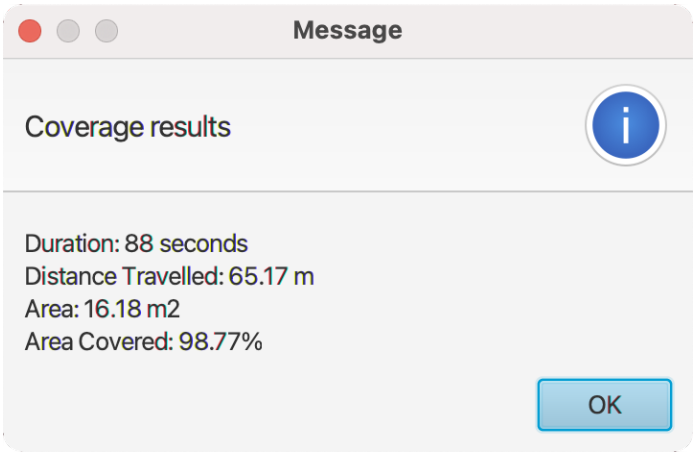
Táto sekcia bližšie popisuje hlavnú scénu aplikácie. Táto scéna predstavuje priestor pre samotnú vizualizáciu plánovacích algoritmov pre úplné pokrytie priestoru. Takisto slúži k tvorbe prostredia, v prípade že sa užívateľ rozhodol pre tento spôsob (druhým spôsobom je načítanie prostredia z externého súboru).



Obr. 5.7: Kompletné grafické užívateľské rozhranie, zachytené za behu simulácie.

Snímka obrazovky vyššie (Obr. 5.7) reprezentuje kompletné grafické užívateľské rozhranie aplikácie. V pravej časti sa nachádza postranný ovládací panel, popísaný v sekcii 5.1. Časť rozhrania ktorou sa zaoberá táto sekcia je však hlavná scéna. Ako už bolo spomenuté vyššie, hlavným účelom tejto scény je vizualizácia simulovaných algoritmov. Scéna obsahuje robota a prostredie, v ktorom robot operuje. Obsah scény si dokáže užívateľ prispôsobovať dvoma spôsobmi:

- **Približovanie** – Užívateľ je schopný obsah scény približovať a oddaľovať pomocou kolieska na myši, prípadne tomu odpovedajúcemu gestu na touchpade, v prípade laptopu.
- **Posúvanie** – Pre posúvanie obsahu scény má užívateľ tieto možnosti:
  - **Pomocou postranných posúvačov** – Klasický spôsob posúvania pomocou posúvačov v dolnej a pravej časti scény.
  - **"Drag and Drop"** – Užívateľ môže obsah scény posúvať takisto metódou "Drag and Drop", kedy sa spolu s podržaním ľavého tlačidla myši obsah scény posúva pohybom myši.



Obr. 5.8: Vyskakovacie okno zobrazujúce výsledky behu simulácie.

Obrázok 5.8 ilustruje vyskakovacie okno, ktoré sa užívateľovi vrámci hlavnej scény objaví po skončení simulácie. Okno informuje užívateľa o výsledkoch daného behu. Informácia zahŕňa výstupy aplikácie popísané v sekcii 4.4 – doba trvania simulácie, prejdená vzdialenosť, obsah priestoru a percentuálne vyjadrená miera pokrytia plochy priestoru.

Kapitola 6

Implementácia

Program pre testovanie plánovania cesty s úplným pokrytím priestoru bol implementovaný podľa návrhu uvedenom v kapitole 4. K implementácii slúžil programovací jazyk Java s použitím technológie JavaFX. Keďže sa jedná o objektovo orientovaný programovací jazyk, program bol taktiež implementovaný na základe tejto paradigmy.

6.1 Popis jednotlivých tried

Nasledujúca sekcia popisuje účel jednotlivých tried a takisto detailne popisuje väčšinu metód každej triedy (vynechané sú generické metódy ako napríklad getter, setter, konštruktor či metóda `toString`, nakoľko ich princíp je v tomto programe triviálny a nie je pre túto prácu dôležité ho popisovať).

Trieda `Main`

Trieda `Main` vytvára a následne vykresľuje hlavnú scénu užívateľského rozhrania. Takisto načítava a vykresľuje jednotlivé elementy grafického užévateľského rozhrania zo súboru vo formáte FXML. Trieda obsahuje jedinú metódu:

- `void start(Stage primaryStage)`: Metóda je zodpovedná za vykonanie vyššie popísaných udalostí.

Trieda `MainController`

Táto trieda riadi celkový chod programu. Je zodpovedná za obsluhu udalostí užívateľského rozhrania. Trieda takisto riadi simulačný čas programu a taktiež sú v triede implementované všetky algoritmy k nájdeniu cesty s úplným pokrytím priestoru. Trieda obsahuje tieto metódy:

- `void onComboClicked()`: Metóda sa aktivuje po kliknutí na dropdown menu, pomocou ktorého užívateľ volí algoritmus pokrývania priestoru. Metóda toto dropdown menu naplní dostupnými algoritmami.
- `void onStartSimulation()`: Metóda najprv vykreslí na hlavnú scénu priestor ktorý užívateľ vytvoril, prípadne načítal z externého súboru. Následne metóda pomocou algoritmov vytvorí robotovi cestu, po ktorej má ísť. Táto metóda taktiež mení stav väčšiny vstupov užívateľského rozhrania na neprístupné (nemožné ich meniť). Spôsob

implementácie algoritmov ktoré táto metóda používa bude bližšie popísaný v neskoršej sekcii tejto kapitoly.

- `void onStopSimulation()`: Táto metóda zastaví simulácia a opäť sprístupní vstupy v užívateľskom rozhraní (umožní ich meniť).
- `void onMaxSpeed()`: Pri zmene parametra "maximálna rýchlosť"v užívateľskom rozhraní uloží novú hodnotu tohto parametra do príslušnej premennej.
- `void onWidth()`: Pri zmene parametra "šírka robota"v užívateľskom rozhraní uloží novú hodnotu tohto parametra do príslušnej premennej.
- `void onAcceleration()`: Pri zmene parametra äkcelerácia"v užívateľskom rozhraní uloží novú hodnotu tohto parametra do príslušnej premennej.
- `void onCreateArea()`: Zmení stav väčšiny vstupov v užívateľskom rozhraní na neprístupné (nemožno ich meniť). Metóda taktiež nastaví globálnu premennú `creatingArea` na hodnotu `true`, čo indikuje interaktívne vytváranie prostredia pomocou myši.
- `void onReadArea()`: Metóda načíta prostredie z externého súboru a toto prostredie následne vykreslí.
- `void onClearArea()`: Táto metóda zmaže aktuálne vytvorené či načítané prostredie z hlavnej scény.
- `void onClick(MouseEvent event)`: Ak užívateľ práve vytvára prostredie pomocou klikania tlačidla myši, táto metóda pridá do vytváraného priestoru vrchol, podľa miesta kam užívateľ klikol. V prípade kliknutia pravého tlačidla myši sa uloží dotvorené prostredie (mierne sa poupraví, aby bolo prostredie uzatvorené a hrany sa nepretínali).
- `Line onMoved(MouseEvent event)`: Metóda je zodpovedná za vizualizáciu práve ťahanej čiary. Začiatok vizualizovanej čiary je v mieste posledne pridaného vrcholu a koniec tejto čiary je variabilný – záleží na aktuálnej pozícii kurzora vrámci hlavnej scény.
- `void onTimeScaleChange()`: Metóda načíta hodnotu zadanú na vstupe (zodpovednú za zmenu rýchlosti simulácie) vrámci užívateľského rozhrania. Následne sa aplikuje zmena rýchlosti simulácie.
- `void onZoom(ScrollEvent event)`: Metóda obsluhuje udalosť točenia kolieska myši nad hlavnou scénou užívateľského rozhrania. Podľa smeru točenia kolieska obsah scény priblíži či oddiali.
- `void setElements(List<Drawable> elements)`: Táto metóda je zodpovedná za vykreslenie zoznamu elementov typu `Drawable`, zadaného ako argument tejto metódy.
- `void stopTime()`: Metóda pozastaví časovač simulácie.
- `void startTime(float scale)`: Táto metóda spustí časovač simulácie. Argument `scale` pritom umožňuje rýchlosť tohto časovača modulovať. Čím je hodnota `scale` väčšia, tým rýchlejšie beží simulačný čas a opačne.

Trieda `Coordinate`

Trieda `Coordinate` slúži k reprezentácii miesta pomocou súradnicového systému. Každý objekt tejto triedy disponuje X-súradnicou a Y-súradnicou. Táto trieda okrem generických metód neobsahuje žiadne podstatné metódy.

Trieda `Data`

Trieda `Data` reprezentuje štruktúru, do ktorej sú načítavané dáta z externého súboru. Objekt typu `Data` obsahuje zoznam hrán typu `Edge`, zoznam súradníc typu `Coordinate` a objekt samotného robota typu `Robot`. Táto trieda okrem generických taktiež neobsahuje žiadne podstatné metódy.

Trieda `Edge`

Táto trieda slúži k reprezentácii hrany prostredia. Obsahuje atribúty `Coordinate start`, `Coordinate stop` a názov hrany typu `String`. Trieda obsahuje metódu:

- **`List<Shape> getGUI()`**: Metóda vytvorí objekt type `Line` s parametrami odpovedajúcimi atribútom objektu typu `Edge` (začiatok a koniec).

Trieda `Path`

Trieda `Path` reprezentuje cestu, ktorú robot nasleduje. Táto cesta je reprezentovaná zoznamom súradníc typu `Coordinate`. Trieda obsahuje metódy:

- **`double getDistanceBetweenCoordinates(Coordinate a, Coordinate b)`**: Métóda vypočíta vzdialenosť medzi dvoma miestami, reprezentovanými typom `Coordinate`.
- **`Coordinate getCoordinateByDistance(double distance)`**: Metóda na základe dĺžky prejdenej časti cesty robota, vypočíta súradnice, kde sa momentálne nachádza.
- **`double getPathSize()`**: Metóda vypočíta dĺžku cestu `Path`.

Trieda `Robot`

Trieda `Robot` slúži k reprezentácii robota samotného. Obsasuje atribúty, ktorými sú šírka, rýchlosť, maximálna rýchlosť, akcelerácia, cesta, pozícia a prejdená vzdialenosť. Trieda `Robot` disponuje týmito metódami:

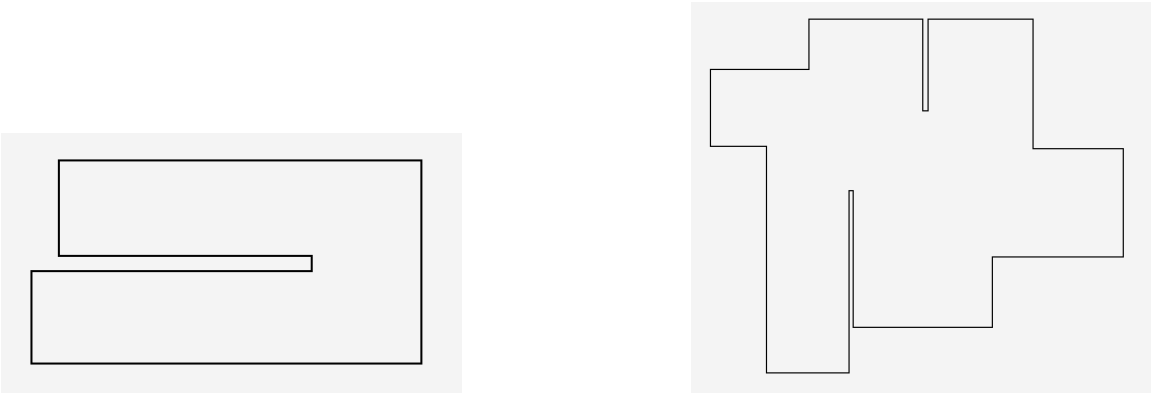
- **`void moveGui(Coordinate coordinate)`**: Táto metóda presunie robota vrámci hlavnej scény užívateľského rozhrania na pozíciu zadanú ako argument tejto metódy (`Coordinate coordinate`).
- **`void setGui()`**: Táto metóda je zodpovedná za vytvorenie elementu vrámci grafického užívateľského rozhrania, ktorý bude robota reprezentovať. Týmto elementom je objekt typu `Circle`.
- **`void update(LocalTime time)`**: Vrámci tejto metódy sa vypočítava aktuálna pozícia robota. Volá sa tu takisto metóda `moveGUI()` zodpovedná za posunutie robota. V tejto metóde sa taktiež aplikuje akcelerácia robota.

6.2 Implementácia algoritmov

Vstupom každého z algoritmov implementovaných v programe je množina vrcholov. Tieto vrcholy spolu s hranami ktoré ich spájajú tvoria prostredie pre robota. Algoritmus, ktorý program využíva k vytvoreniu cesty pre robota by sa dal slovne popísať nasledujúcimi krokmi:

- **1. Vytvoriť zoznam *X*-súradníc vrcholov prostredia**
- **2. Na každú vodorovnú hranu prostredia pridať všetky *X*-súradnice zo zoznamu z kroku č. 1, ktoré sa sa nachádzajú medzi počiatočnou a koncovou *X*-súradnicou hrany**
- **3. Zvoliť dva vrcholy, ktoré sa nachádzajú najviac vľavo.**
- **4. K zvoleným ľavým dvom vrcholom pridať ďalšie 2 vrcholy tak, aby vznikol obdĺžnik.**
- **5. Zvolený obdĺžnik tvorený štvoricou vrcholov pretvoriť na konfiguračný priestor robota – z každého kraja sa uberie polovica šírky robota.**
- **6. Pokrytie aktuálneho obdĺžnika.**
- **7. Prechod do miesta, kde sa obdĺžnik spája so zvyškom priestoru. Stále vrámci daného štvorca.**
- **8. Posun doprava o šírku robota.**
- **9. Odstránenie štyroch vrcholov predošlého obdĺžnika zo zoznamu všetkých vrcholov.**
- **10. Spať na krok č. 3.; Opakovať pokiaľ nie je zoznam vrcholov prostredia prázdny.**

Tento algoritmus predstavuje podstatnú časť fungovania programu. Užívateľ je schopný tento algoritmus modulovať pomocou dropdown menu (Obr. 5.2) v užívateľskom rozhraní. Táto zmena sa prejaví v kroku č. 6 v algoritme popísanom vyššie. To akým spôsobom budú jednotlivé bunky (obdĺžniky) pokrývané záleží od algoritmu zvoleného užívateľom. Algoritmus však má istú limitáciu tvaru prostredia, nad ktorým je uplatnený. Prostredie musí byť rozložiteľné na obdĺžniky, ktorých rozsah na X-súradnici sa neprelína. V praxi to znamená, že výsledné prostredie nesmie mať 2 obdĺžniky nad sebou a medzi sebou voľný priestor.



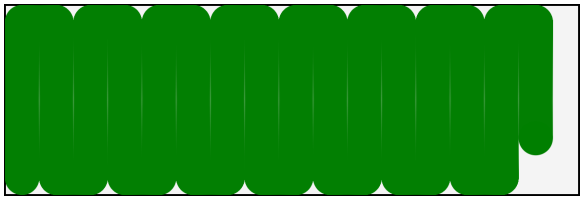
Obr. 6.1: Obrázok vľavo znázorňuje aplikáciou nepodporovaný typ prostredia – nie je rozložiteľné obdĺžniky tak, aby žiadne dva neboli nad sebou. Obrázok napravo znázorňuje príklad prostredia, ktoré aplikácia podporuje.

V programe sú implementované dva algoritmy, ktorými robot pokrýva jednotlivé bunky prostredia:

- **Špirálovitý pohyb** (Obr. 6.2): Tento algoritmus sa dá slovne popísať spôsobom:
  - 1. Choď rovno, pokiaľ nenarazíš na okraj prostredia, prípadne svoju vlastnú stopu.
  - 2. Otoč sa okolo vlastnej osi o 90° doprava.
  - 3. Kroky č. 1 a 2 opakuj, pokiaľ sa po otočení okolo vlastnej je kam pohnúť.

Obr. 6.2: Špirálovitý pohyb. Obrázok je snímka obrazovky priamo za behu aplikácie.

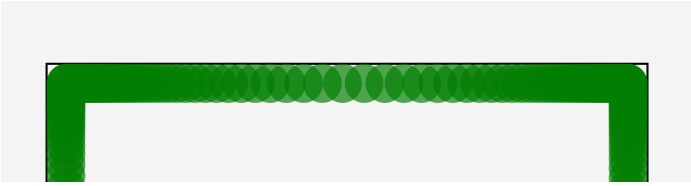
- **Pohyb "tam a späť"** (Obr. 6.3): Slovný algoritmus popisujúce tento spôsob pohybu vyzerá nasledovne:
  - 1. Choď rovno, pokiaľ nenarazíš na okraj prostredia.
  - 2. Otoč sa okolo vlastnej osi o 90° doprava.
  - 3. Posuň sa dopredu o šírku robota.
  - 4. Otoč sa okolo svojej vlastnej osi o 90° doprava.
  - 5. Opakuj kroky č. 1–4; v každej druhej iterácii však zmeň smer otáčania na opačný; opakuj, pokiaľ sa je kam posúvať.



Obr. 6.3: Pohyb "tam a späť". Obrázok je snímka obrazovky priamo za behu aplikácie.

6.3 Pohyb robota

To ako sa samotný robot vrámci simulácie pohybuje predstavuje kľúčovú časť aplikácie. Pohyb robota som implementoval pomocou zmeny jeho pozície vrámci súradnicového systému v pomerne rýchlej frekvencii. Pri štandardnej rýchlosti simulácie k tejto zmene jeho pozície dôjde každých 30 milisekúnd. Pri zrýchlenej simulácii k tomu dochádza častejšie. Robot za sebou zanecháva stopu, ktorá znázorňuje trasu ktorú prešiel. Táto stopa je znázornená sekvenciou kruhov, ktoré napodobňujú súvislý pás. Priehľadnosť týchto kruhov je nastavená na hodnotu `0.2` – to spôsobuje že v miestach kde robot spomalil zanechával stopu pomalšie, čo má za následok tmavší odtieň stopy.

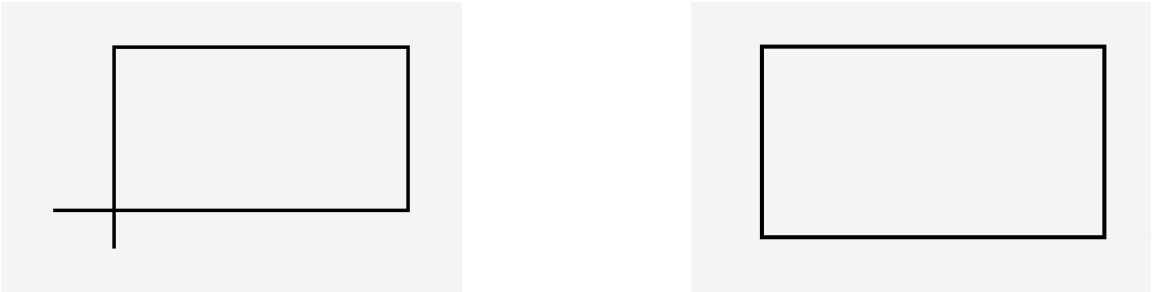


Obr. 6.4: Tmavšia stopa znázorňuje miesto, kde šiel robot pomalšie.

Obrázok 6.4 znázorňuje princíp popísaný vyššie. Takisto je na obrázku 6.4 vidieť vplyv rýchlosti na hustotu vykreslených segmentov stopy, nakoľko je v aplikácii simulovaná taktiež akcelerácia robota.

6.4 Tvorba prostredia

Prostredie, ktorého priestor robot pokrýva je možné zadať dvoma spôsobmi. Prvým spôsobom je načítanie z externého súboru, stlačením tlačidla s ikonou priečinka (Obr. 5.3). Druhou možnosťou je prostredie vytvoriť ťahaním čiar priamo v hlavnej scéne aplikácie. Po stlačení ľavého tlačidla myši na ľubovoľnom mieste plochy určenej k vykresľovaniu, užívateľ pridá prvý vrchol prostredia. Následne užívateľ pomocou myši ťahá čiary a tým prostredie dotvára. Aplikácia avšak pracuje výlučne s pravými uhlami, preto užívateľovi nie je umožnené vytvárať prostredie s uhlami inými ako 90°. Hrana môže byť buď vodorovná alebo vertikálna. To, aby hrana ktorú užívateľ práve ťahá, vyhovovala týmto podmienkam zaručuje automatické presmerovanie hrany do vodorovnej alebo vertikálnej polohy. To, či sa práve ťahaná hrana zobrazuje ako vodorovná alebo vertikálna sa vypočíta podľa smernice práve ťahanej čiary. Limitáciou tvorby prostredia je nutnosť skončiť v rohu, v ktorom užívateľ začal. Po stlačení pravého tlačidla myši, ktorým užívateľ ukončí tvorbu prostredia, sa vytvorené prostredie automaticky poupraví tak, aby bolo uzatvorené a jednotlivé hrany sa nepretínali. (Obr. 6.5).



Obr. 6.5: Po stlačení pravého tlačidla myši sa vytvorené prostredie naľavo automaticky transformuje na prostredie vpravo.

Výpočet obsahu pokrytého priestoru

Keďže aplikácia pracuje s kruhovým robotom a priestorom s pravými uhlami, nie je možné pokryť celý priestor. Nepokrytý priestor som algoritmicky počítal ako súčet obsahov nepokrytých trojuholníkov v rohoch priestoru a nepokrytých trojuholníkov vzniknutým typom pohybu 6.3 či 6.2.

Kapitola 7

Testovanie

Testovanie výsledného programu prebiehalo v dvoch fázach. Prvou fázou boli systematické testy, zamerané na funkčnosť aplikácie, no taktiež na presnosť a pravdivosť nameraných hodnôt. Druhá fáza testovania predstavuje užívateľský prieskum. Táto fáza testovania je viacej zameraná na hodnotenie prehľadnosti a intuitívnosti grafického užívateľského rozhrania.

7.1 Funkčnosť a presnosť aplikácie

Táto sekcia popisuje testy programu, zamerané na funkčnosť a presnosť jednotlivých nameraných výstupov aplikácie.

Meranie času

Tieto testy boli zamerané na overenie správneho fungovania merania času simulácie. Testy overujú, či nameraný čas simulácie odpovedá reálnej dobe trvania simulácie. Ďalej tieto testy overujú, či je pri rovnakých vstupoch aplikácie nameraná rovnaká doba trvania pokrytia priestoru, v prípade že rýchlosť simulácie je za behu zrýchľovaná a spomaľovaná. Posledným aspektom ktorý tento druh testov overuje je správne meranie času simulácie pri rôznych rýchlostiach robota. Nameraný čas vrámci aplikácie je uvádzaný v celých sekundách, nakoľko tento výstup program zaokrúhľuje.

	Nameraný čas	Reálna doba
Beh1	45s	45s
Beh2	1m 14s	1m 13s
Beh3	14s	14s
Beh4	2m 48s	2m 46s
Beh5	1m 57s	1m 57s

Obr. 7.1: Porovnanie nameraného času simulácie aplikáciou a stopkami.

Tabuľka 7.1 porovnáva nameraný čas trvania pokrytia priestoru v 5 rôznych behov v 5 rôznych prostrediach. Stĺpec "Nameraný čas"predstavuje čas, ktorý nameral časovač vrámci aplikácie. Stĺpec "Reálna doba"predstavuje nameraný čas mimo aplikácie stopkami. V ideálnom prípade by tieto dva stĺpce mali mať identické hodnoty, avšak, ako je vidieť v tabuľke 7.1, pri dlhších vzdialenostiach vzniká istá časová odchýlka.

	Rýchlosť simulácie	Nameraný čas	Reálna doba
Beh1	1x	1m 39s	1m 38s
Beh2	2x	1m 39s	1m 38s
Beh3	3x	1m 39s	1m 38s
Beh4	5x	1m 39s	1m 38s
Beh5	10x	1m 39s	1m 38s

Obr. 7.2: Vplyv rýchlosti simulácie na dobu trvania pokrytia priestoru.

Tabuľka 7.2 obsahuje výsledky testov, ktorých účelom bolo overiť vplyv zmeny rýchlosti simulačného času, na dobu pokrytia priestoru. Keďže rýchlosť simulácie nemá vplyv na výslednú dobu pokrytia priestoru, experimenty zachytené v tabuľke 7.2 dokazujú, že tento aspekt funguje v poriadku.

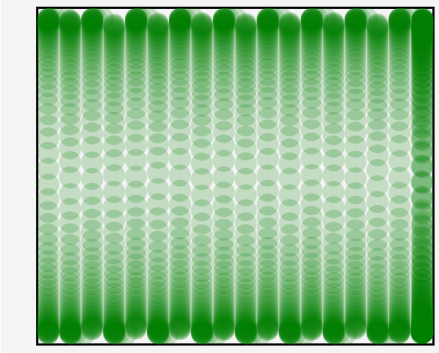
	Max. rýchlosť robota	Nameraný čas
Beh1	1km/h	2m 9s
Beh2	3km/h	49s
Beh3	10km/h	17s
Beh4	15km/h	12s
Beh5	25km/h	6s

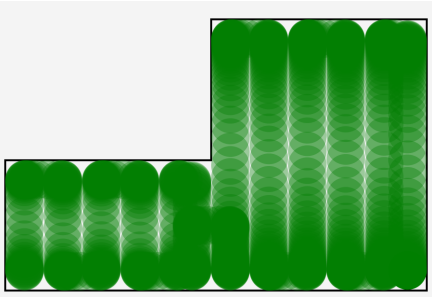
Obr. 7.3: Vplyv maximálnej rýchlosti robota na dobu trvania pokrytia priestoru.

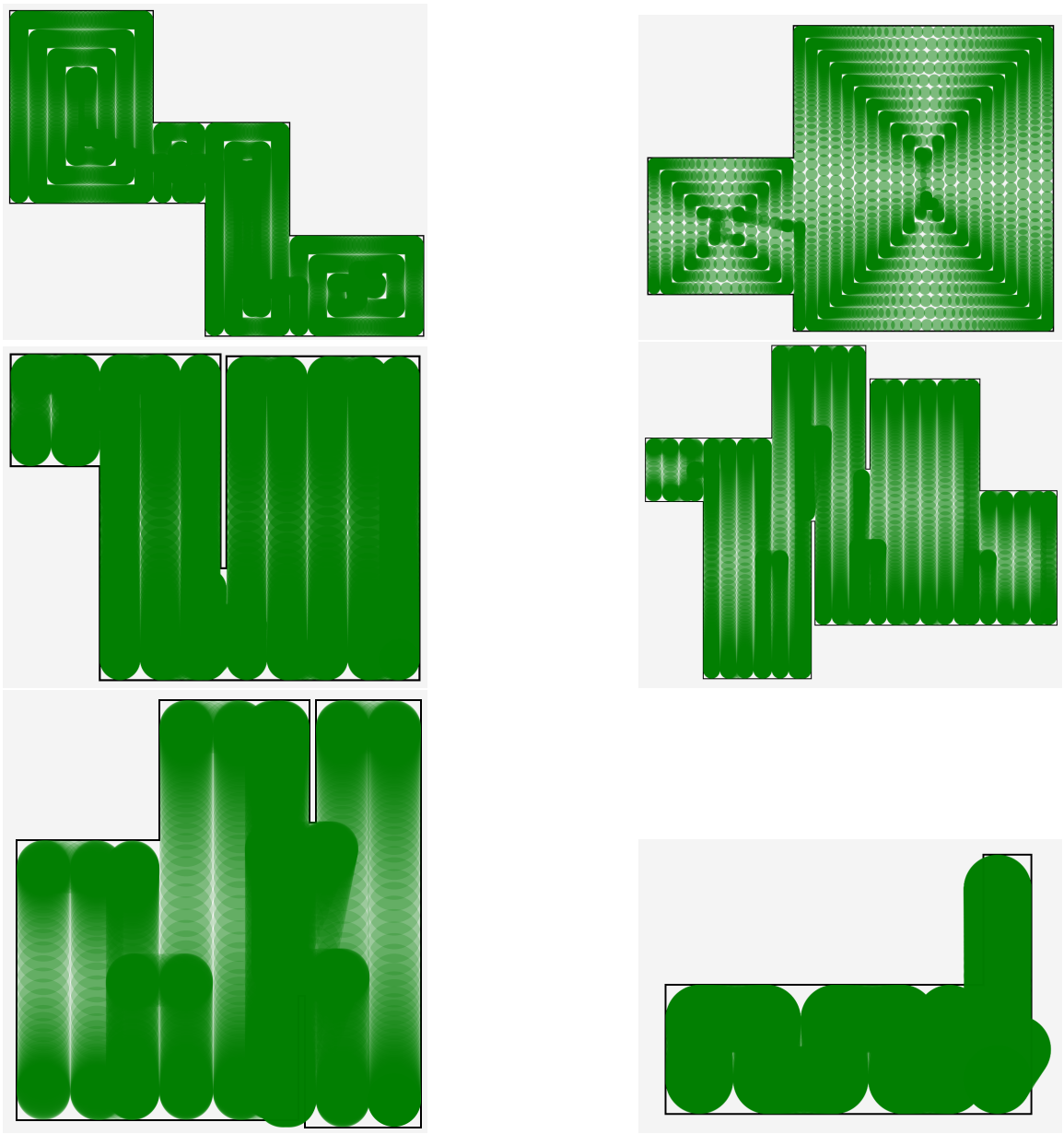
Poslednú sadu testov z oblasti merania času boli testy zachytené v tabuľke 7.3. Účelom týchto testov bolo overiť vplyv maximálnej rýchlosti na dobu trvania pokrytia priestoru. Sada obsahuje takisto 5 testov s rovnako zadanými vstupmi. Z výsledkov tejto sady testov vidieť značný vplyv maximálnej rýchlosti robota na dobu trvania pokrytia daného priestoru.

Funkčnosť aplikácie

Testovanie funkčnosti aplikácie pozostávali s testovaním algoritmov pokrývania priestoru v rôznych prostrediach, s rôzne zadanými vstupmi. Účelom týchto testov bolo zistiť, či algoritmus pracuje správne aj v zložitejších prostrediach. Výsledky týchto testov ilustruje nasledujúca sada obrázkov







Obr. 7.4: Obrázky ilustrujú jednotlivé behy programu, zakaždým v inom prostredí a s inými vstupnými parametrami.

Ako je možné vidieť na spodných dvoch obrázkoch 7.4, program v istým prípadoch zlyháva. Zlyhanie dokáže zapríčiniť prostredie, ktorého dekompozícia na bunky pozostáva z extrémne tenkých segmentov. Druhá udalosť, ktorá zapríčiňuje nesprávnosť pokrytia je príliš veľká šírka robota (širšia ako najtenší segment dekompozície).

7.2 Prieskum užívateľov

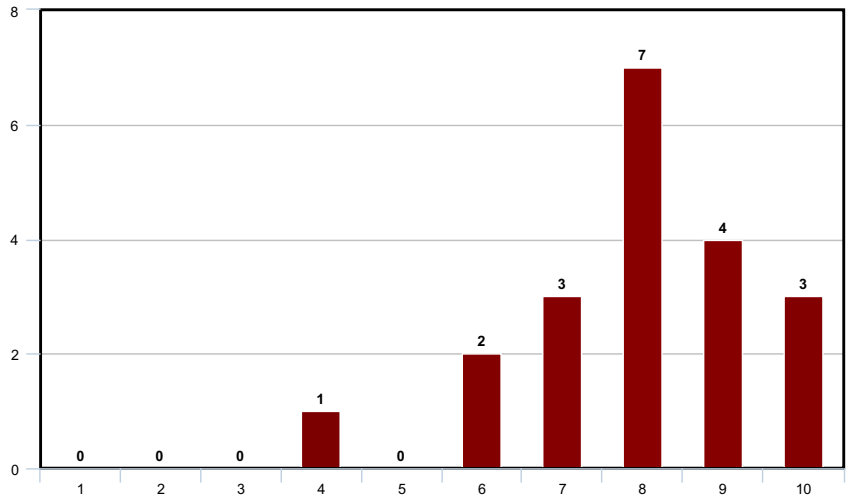
Táto fáza testovania bola vykonávaná formou dotazníkov. Respondenti tohto prieskumu boli 20 študenti VUT v Brne, vo veku 19-24 rokov. Tento prieskum pozostával z troch častí. Každú časť mohol účastník bodovo ohodnotiť v rozmedzí 1–10 bodov (10 bodov predstavuje najvyššie skóre). Pred samotným bodovaním strávil respondent nejaký čas skúšaním aplikácie.

Intuitívnosť grafického užívateľského rozhrania

Prvou časťou prieskumu užívateľov bolo ohodnotiť intuitívnosť grafického užívateľského rozhrania. Táto časť testu prebiehala tak, že respondentom boli zadávané jednoduché úlohy, ako napríklad:

- zmeniť šírku robota
- načítať prostredie z externého súboru
- spustiť simuláciu
- zrýchliť simuláciu

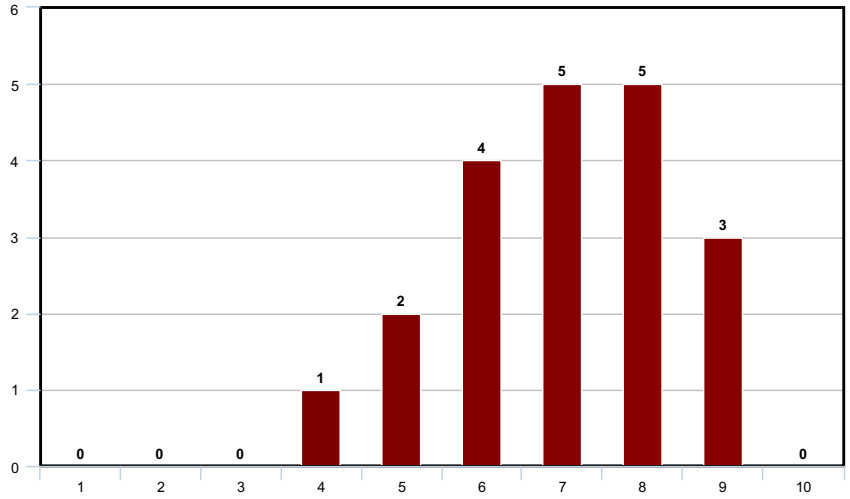
Po vykonaní týchto pokynov boli respondenti vyzvaný k ohodnoteniu, nakoľko jednoduché pre nich bolo zorientovať sa v aplikácii, hoc ju používajú po prvý krát.



Obr. 7.5: Ohodnotenie intuitívnosti grafického užívateľského rozhrania.

Pohodlie interaktívneho vytvárania prostredia

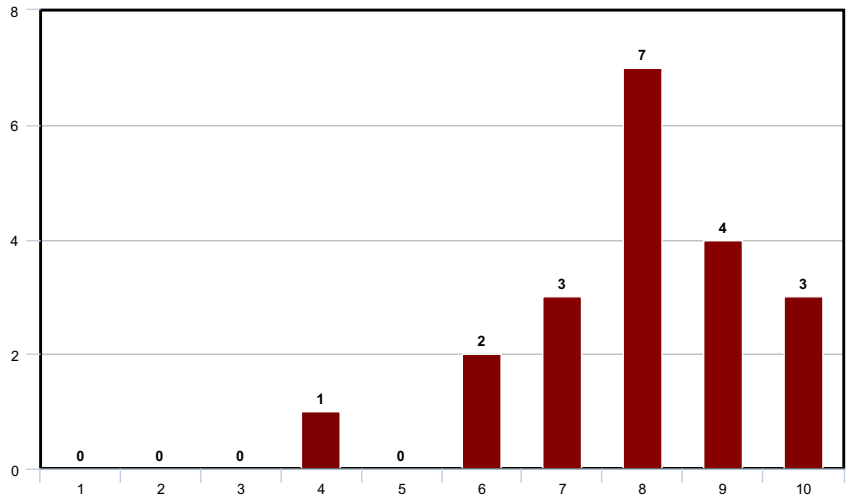
Druhou časťou užívateľského prieskumu bolo ohodnotiť pohodlie pri vytváraní prostredia. Respondenti boli vyzvaní k samostatnému vytvoreniu ľubovoľného prostredia v aplikácii. Žiaden z respondentov nebol informovaný o nemožnosti natiahnuť inú čiaru ako vodorovnú či zvislú. Respondenti túto skutočnosť objavili až počas samotného ťahania čiar. Po dokončení tvorby prostredia boli vyzvaní k ohodnoteniu tohto aspektu aplikácie.



Obr. 7.6: Ohodnotenie pohodlia pri tvorbe prostredia.

Celkový dojem

Treťou a zároveň poslednou časťou užívateľského prieskumu bolo bodovo ohodnotiť celkový dojem z aplikácie.



Obr. 7.7: Ohodnotenie celkového dojmu z aplikácie.

Kapitola 8

Záver

Cieľom tejto práce bolo naštudovať problematiku plánovania cesty so zameraním na úplné pokrytie priestoru a následne navrhnúť a implementovať program, ktorý užívateľovi umožní testovať a vyhodnocovať rôzne prístupy a algoritmy pre plánovanie takejto cesty. Výsledný program užívateľovi umožňuje vytvoriť, alebo z externého súboru nahrať prostredie, v ktorom bude robot operovať. Po zadaní ďalších nevyhnutných vstupov užívateľom, ako sú napríklad šírka robota, rýchlosť robota, zrýchlenie robota a zopár ďalších, program užívateľovi vizualizuje zvolený spôsob pokrytia priestoru spolu s informáciami o efektívnosti konkrétnej cesty.

Robotovi je možné nastaviť rýchlosť, šírku či zrýchlenie a to v metrických jednotkách, ktoré sa v programe prepočítavajú aby proporčne odpovedali vizualizácii. Užívateľ je schopný jednoducho klikaním myši a ťahaním čiar vytvoriť prostredie. Aplikácia v tejto fáze dáva užívateľovi na výber 2 algoritmy. Limitáciou tohto programu je určite nemožnosť vytvoriť ľubovoľné prostredie. Prostredie ktoré užívateľ vytvorí musí disponovať výhradne pravými uhlami.Keďže výsledný program pracuje s algoritmami, algoritmizácia je oblasť v ktorej som sa vďaka tejto práci najviac posnul. Implementácia algoritmov použitých v práci pre mňa predstavovala výzvu, avšak za pomoci vhodnej literatúry to bolo zvládnuteľné.

Výsledný program má rozhodne priestor na pokračovanie či rozšírenie. Do aplikácie je v budúcnosti možné implementovať viacej algoritmov, čím by sa jej použiteľnosť rozhodne rozšírila. Ďalšou oblasťou, ktorá má potenciál na rozšírenie, je rozhodne tvar a topológia priestoru, v ktorom robot operuje. Tento priestor by v budúcej verzii programu nemusel byť nutne pravouhlý, prípadne by mohol obsahovať prekážky.

Literatúra

[1] *TurtleBot2*. 2016. Dostupné z: https://www.turtlebot.com/assets/images/turtlebot_2_lg.png.

[2] *IRobot Roomba i7*. 2021. Dostupné z: https://www.irobot.cz/app/uploads/2019/10/i7_charcoal_photo_studio_frontstanding_hero-1400x1400-c-default.png.

[3] Acar, E. U., Choset, H., Rizzi, A. A., Atkar, P. N. a Hull, D. Morse decompositions for coverage tasks. *The international journal of robotics research*. SAGE Publications Sage UK: London, England. 2002, zv. 21, č. 4, s. 331–344.

[4] Choset, H. a Pignon, P. Coverage path planning: The boustrophedon cellular decomposition. In: Springer. *Field and service robotics*. 1998, s. 203–209.

[5] Choset, H. M., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W. et al. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[6] Galceran, E. a Carreras, M. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*. 2013, zv. 61, č. 12, s. 1258 – 1276. DOI: https://doi.org/10.1016/j.robot.2013.09.004. ISSN 0921-8890. Dostupné z: http://www.sciencedirect.com/science/article/pii/S092188901300167X.

[7] Jurišica, L. a Duchoň, F. *Bunková dekompozícia prostredia v mobilnej robotike*. December 2010. Dostupné z: https://www.atpjournal.sk/buxus/docs/casopisy/atp_2010/pdf/Bunkova_dekmpozicia_prostredia_v_mobilnej_robotike.pdf.

[8] Milnor, J. *Morse theory.(AM-51)*. Princeton university press, 2016.

[9] Sojka, S. *Implementace robotického vysavače*. Brno, CZ, 2011. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/12352/.

[10] Tarjan, R. Depth-first search and linear graph algorithms. *SIAM journal on computing*. SIAM. 1972, zv. 1, č. 2, s. 146–160.

[11] User:Bromskloss. *File:Ackermann.svg*, 28. November 2006. Dostupné z: https://commons.wikimedia.org/wiki/File:Ackermann.svg.

Príloha A

Obsah SD karty