Plánováni pohybu a optimální cesty v komplexním prostředí

Matej Škvarenina

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Matej Škvarenina**
Osobní číslo:	**A19111**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Plánování pohybu a optimální cesty v komplexním prostředí**
Téma práce anglicky:	**Optimal Route and Motion Planning in Complex Environment**

Zásady pro vypracování

1. Zpracujte literární rešerši na téma problému obchodního cestujícího a plánování pohybu.
2. Vytvořte testovací scénáře.
3. Aplikujte vhodné metody na testovacích scénářích a porovnejte je.
4. Zhodnoťte výsledky práce a jejich možné využití.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. REINELT, Gerhard. TSPLIB-A Traveling Salesman Problem Library. ORSA Journal on Computing [online]. 1991, 3(4), 376-384 [cit. 2021-11-30]. ISSN 0899-1499. Dostupné z: doi:10.1287/ijoc.3.4.376
2. REGO, César, Dorabela GAMBOA, Fred GLOVER a Colin OSTERMAN. Traveling salesman problem heuristics: Leading methods, implementations and latest advances. European Journal of Operational Research. 2011, 211(3), 427-441. ISSN 0377-2217. Dostupné z: doi: 10.1016/j.ejor.2010.09.010
3. TIWARI, Ritu, Anupam SHUKLA a Rahul KALA. Intelligent planning for mobile robotics: algorithmic approaches. Hershey, PA: Information Science Reference, 2013. ISBN 9781466620759.
4. LAVALLE, Steven Michael. Planning algorithms. New York: Cambridge University Press, 2006. ISBN 9780521862059.
5. PATNAIK, Srikanta, Xin She YANG a Kazumi NAKAMATSU, ed. Nature-Inspired Computing and Optimization: Theory and Applications. Springer International Publishing AG. Cham: Springer, 2017. ISBN 978-3-319-84522-7.

Vedoucí bakalářské práce: **Ing. Adam Viktorin, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **3. prosince 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalárske práci pracoval samostatně a použitou literaturu jsem citoval. V prípadě publikace výsledků budu uveden ako spoluautor.
- že odevzdaná verze bakalárské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 18.5.2022

Matej Škvarenina v.r.
podpis studenta

ABSTRAKT

Táto bakalárska práca sa zaoberá problematikou plánovania pohybu a optimálnej cesty v komplexnom prostredí. Teoretická časť sa venuje základnej problematike, popisuje koncepty, algoritmy a možné aplikácie v reálnom živote. Praktická časť je zameraná na vytvorenie testovacích scenárov, porovnanie testovacích scenárov v algoritmoch a vvyhodnotiť výsledky, implementovaním testovacích scenárov v mriežke plánovania optimálnej cesty. Testovacie scenáre sú testované v aplikáciách, ktoré sú vytvorené v programovacom jazyku Python.

Klíčová slova: Plánovanie pohybu a optimálnej cesty, Python, Testovacie scenáre

ABSTRACT

This bachelor thesis deals with the issue of motion planning and optimal travel in a complex environment. The theoretical part deals with the basic issues, describes concepts, algorithms and possible applications in real life. The practical part is focused on creating test scenarios, comparing test scenarios in algorithms and evaluating the results, by implementing test scenarios in the optimal path planning grid. Test scenarios are tested in applications that are created in the Python programming language.

Keywords: Motion planning and optimal travel, Python, Testing scenarios

Chcel by som poďakovať svojmu vedúcemu práce pánovi Ing. Adamovi Viktorinovi, Ph.D. za odborné vedenie práce a za všetky rady a konzultácie pri vypracovávaní tejto bakalárskej práce.

Prohlašuji, že odevzdaná verze bakalárské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Vo svojej práci sa zaoberám plánovaním cesty. Je to úzko spojené s robotikou, samotná robotika a jej aplikácie vo svete prinášajú mnoho výhod. Predovšetkým je to schopnost robotov jednať samostatne v rôznych prostrediach. Veľmi známou aplikáciou tohoto typu je napríklad robot Mars Rover, ktorého vlastnosti sa osvedčili v takých podmienkach, ako je prieskum Marsu. Je schopnosť k adaptácií sa miestnému prostrediu niekoľkonásobne vyvážili náklady, ktoré boli vynaložené na riešenie prieskumu cudzieho vesmírného telesa. Úspech tohoto robotického vozítka umožnil vývoj omnoho dokonalejších a sofistikovanejších robotov, ktoré budú použité k ďalšiemu prieskumu slnečnej sústavy. Vlastnosť robotov pracovať v cudzom alebo agresívnom prostredí otvára široké možnosti aplikácie a náhrady robotov za ľudskú prácu. Jednou z hlavných otázok v oblasti robotiky je dnes programovanie robotov. Existuje mnoho spôsobov, ktoré sú tradičné ale aj netradičné. Jedným z netradičných spôsobov je napríklad programovanie robotov v jazyku C# od spoločnosti Microsoft. Tento spôsob využíva spoločnosť ABB. Použitie tohoto vysokoúrovňového objektovo orientovaného programovacieho jazyk môže pre mnoho programátorov predstavovať priehľadnejšie, efektívnejšie a netradičnejšie riešenie pri vývoji robotický aplikácií. Cieľom tejto bakalárskej práce je ukázať rozmanitosť, efektívnosť a možnosti aplikovania robotiky vo svete s využitím Path planningu. Path planning alebo plánovanie cesty a pohybu je jedným z najdôležitejších výzkumných problémov v robotike z pohľadu riadiaceho inžiniera. Mnoho problémov v rozdielných oblastiach je riešených návrhom plánovania trasy. Tento algoritmus, pracuje tak, že automaticky plánuje trasu, ktorú robot prejde z bodu A do bodu B. V dnešnej dobe takmer každý pozná plánovače pohybu, no väčšina ľudí si to neuvedomuje. V minulosti by sme museli prísť s invenčnou analógiou.

I. TEORETICKÁ ČÁST

1 DEFINÍCIA ROBOTIKY

Robotika [1] je priesečníkom vedy, inžinierstva a technológie, ktorá vyrába stroje nazývané roboty, ktoré nahrádzajú (alebo replikujú) ľudské činy. Popkultúra bola vždy fascinovaná robotmi. R2-D2. Optimus Prime. WALL-E. Tieto prehnané humanoidné koncepty robotov sa vyčajne zdajú ako karikatura skutočnej veci, alebo myslia skôr dopredu, ako si uvedomujeme? Roboty získavajú intelektuálne a mechanické schopnosti, ktoré v budúcnosti neodstraňujú možnosť stroja podobného R2-D2. S pokrokom technológie sa mení aj rozsah toho, čo sa považuje za robotiku. V roku 2005 bolo možné nájsť 90% všetkých robotov pri montáži áut v automobilových továrňach. Tieto roboty pozostávajú hlavne z mechanických ramien, ktorých úlohou je zváranie alebo skrutkovanie určitých častí auta. Dnes sme svedkami rozvinutej a rozšírenej definície robotiky, ktorá zahŕňa vývoj, vytváranie a používanie robotov, ktoré skúmajú najtvrdšie podmienky na Zemi, roboty, ktoré pomáhajú pri presadzovaní práva, a dokonca roboty, ktoré pomáhajú takmer vo všetkých aspektoch zdravotnej starostlivosti. Prísľub robotického priemyslu je neprehliadnuteľný, pretože umelá inteligencia a softvér tiež pokračujú v robení pokrokov. Vďaka pokroku v týchto technológiách budú v blízkej budúcnosti roboty aj naďalej inteligentnejšie, felxibilnejšie a energeticky účinnejšie. Naďalej budú tiež hlavným ústredným bodom v inteligentných továrňach, kde budú čeliť náročnejším výzvam a pomôžu zabezpečiť globálne dodávateľské reťazce. Hoci je robotický priemysel relatívne mladý, je naplnený obdivuhodným prísľubom pokroku, o ktorom sci-fi mohla kedysi iba snívať. Od najhlbších hlbín našich oceánov až po tisíce miľ vo vesmíre nájdeme roboty, ktoré budú vykonávať úlohy, o ktorých by ľudia ani nesnívali, že by ich dosiahli sami.

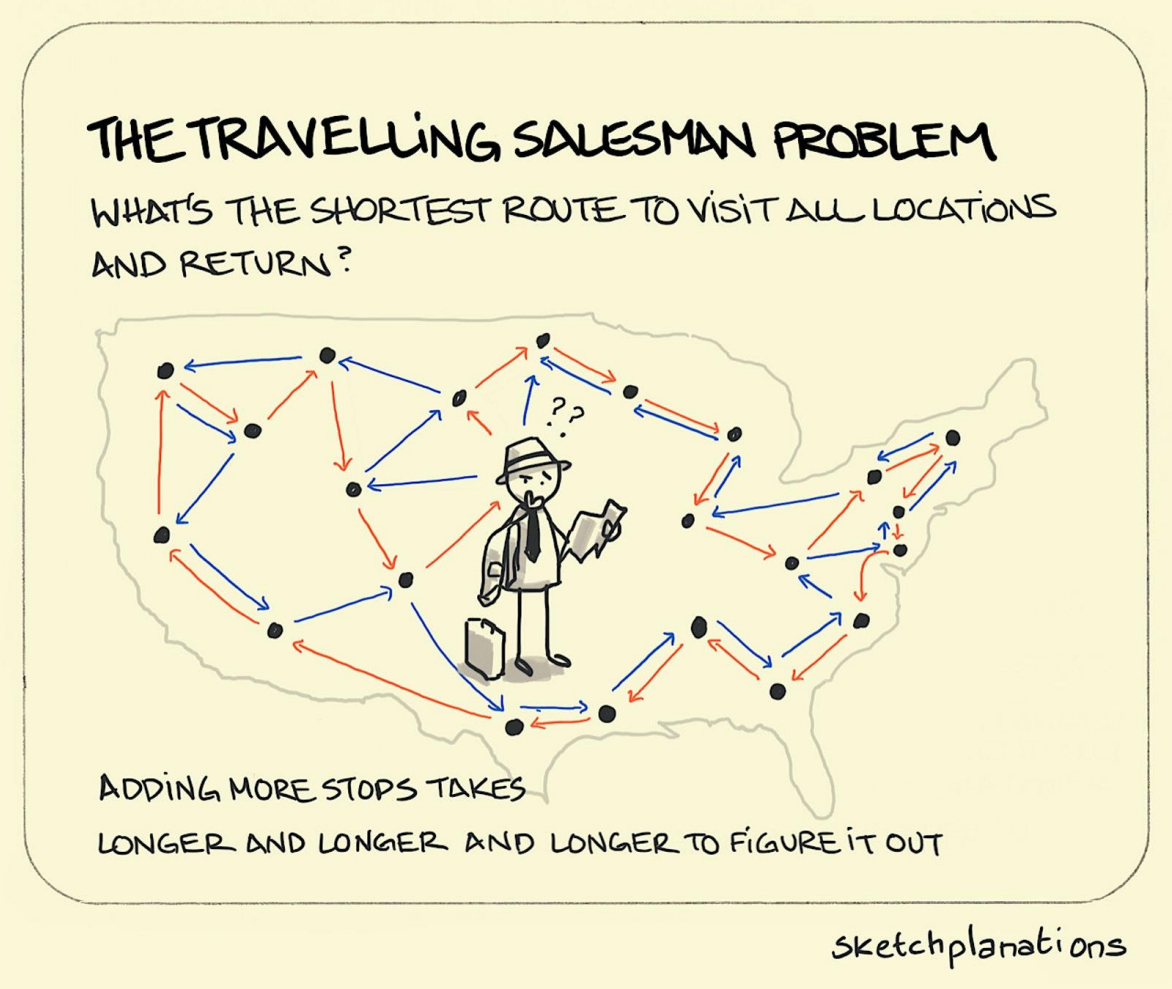
2 PROBLÉM OBCHODNÉHO CESTUJÚCEHO

Problém obchodného cestujúceho [2] je algoritmický problém, ktorého úlohou je nájsť najkratšiu cestu medzi súborom bodov a miest, ktoré je potrebné navštíviť. V definícií sú body určené ako mestá, ktoré môže predajca navštíviť. Jeho Cieľom je udržať cestovné náklady aj prejdenú vzdialenosť čo najnižšie. Obchodný cestujúci ,ktorý sa zameriava na optimalizáciu je často používaný v informatike na nájdenie najefektívnejšieho spôsobu prenosu údajov medzi rôznymi uzlami. Aplikácie zahŕňajú identifikáciu metód optimalizácie siete alebo hardvéru. Prvýkrát ho opísali írsky matematik W.R. Hamilton a britský matematik Thomas Kirkman v roku 1800 vytvorením hry, ktorá bola riešiteľná nájdením Hamiltonovho cyklu, čo je neprekrývajúca sa cesta medzi všetkými uzlami. Tento problém je skúmaný už desaťročia a teoretizuje sa niekoľko riešení. Najjednoduchším riešením je vykúšať všetky možnosti, toto riešenie je najdrahšie a časovo náročné. Mnohé riešenia využívajú heuristiku, ktorá poskytuje výsledky pravdepodobnosti. Výsledky sú však približné a nie vždy optimálne. Medzi ďalšie riešenia patria algoritmy branch and bound, Monte Carlo a Las Vegas. Miesto sústredenia sa na hľadanie najefektívnejšej trasy sa tento problém často zaoberá hľadaním najlacnejšieho riešenia. V probléme obchodného cestujúceho vytvára veľké množstvo premenných problém pri hľadaní najkratšej trasy, čo robí približné, rýchle a lacné riešenia ešte atraktívnejšie.

2.1 Popis základných variant

Pre konkrétnejšiu predstavu si môžeme predstaviť pár známých možností riešenia [3] tohoto problému. Medzi známe riešenie môžme zaradiť prístup hrubou silou alebo anglicky nazývaný aj Brute Force Approach, ktorý je tiež známy ako naivný prístup, počíta a porovnáva všetky možné permutácie trás alebo ciest, aby určil najkratšie jedinečné riešenie. Ak chcete vyriešiť TSP pomocou tohoto prístupu, tak je potrebné vypočítať celkový počet trás a potom nakresliť a uviesť všetky možné trasy. Najskôr vypočítame vzdialenosť každej trasy a potom vyberieme tú najkratšiu. Tento postup je označovaný za optimálne riešenie. Ako ďalšie známe riešnie je metóda vetvenia a viazania alebo The Branch and Bound Method. Toto je metóda, ktorá rozdeľuje problém, ktorý sa má vyriešiť na niekoľko čiastkových problémov. Je to systém na riešenie série čiastkových problémov, z ktorých každý môže mať niekoľko možných riešení, pričom riešnie zvolené pre jeden problém môže mať vplyv na možné riešenia následných čiastkových problémov. Ak chceme problém obchodného cestujúceho vyriešiť pomocou tejto metódy, tak musíme vybrať počiatočný uzol a potom nastaviť hranicu

na veľmi veľkú hodnotu. Vyberieme najlacnejší oblúk medzi nenavštíveným a aktuálným uzlom a potom pridáme vzdialenosť k aktuálnej vzdialenosti. Tento proces opakujeme pokiaľ je aktuálna vzdialenosť menšia ako hranica. Ak je aktuálna vzdialenosť menšia ako hranica, tak sme hotoví. Ako ďalší krok sčítame vzdialenosť tak, aby sa hranica rovnala aktuálnej vzdialenosti. Tento postup opakujeme, až do momentu, pokiaľ nezakryje všetky oblúky. Pre tretí príklad by som využil metódu najbližšieho suseda alebo aj The Nearest Neighbor Method. Táto metóda je pravdepodobne najjednoduchšia heuristika problému obchodného cestujúceho. Kľúčom k tejto metóde je vždy navštíviť najbližšiu destináciu a potom sa vrátiť na prvé miesto, keď sú navštívené všetky ostatné miesta. Ak chceme vyriešiť problém obchodného cestujúceho pomocou tejto metódy, tak si vyberieme náhodné miesto a potom vyhľadáme najbližšie nenavštívené miesto a ideme tam. Pokiaľ sme navštívili všetky miesta, musíme sa vrátiť na prvé miesto.



Obrázok č.1 Problém obchodného cestujúceho

3 PLÁNOVANIE CESTY A POHYBU

Funkciu plánovania pohybu [4] robota je možné pravdepodobne dobre odhadnúť pri pohľade na samotný názov. Plánovanie pohybu je algoritmus, ktorý automaticky plánuje trasu, ktorú robot prejde, aby sa dostal z bodu A do bodu B. V dnešnej dobe takmer každý pozná plánovače pohybu, no väčšina ľudí si to neuvedomuje. Pred niekoľkými rokmi by sme museli prísť s invenčnou analógiou, aby sme opísali, čo robil plánovač ophybu. Už nie. Minimálne posledné desaťročie ho má takmer každý smartfón s GPS a mapovou aplikáciou. Tieto aplikácie používajú rovnaký typ plánovacích algoritmov ako plánovače pohybu robotov. Prečo využívame plánovače pohybu v priemyselnej robotike? Hlavný dôvod, prečo používame plánovače pohybu, je to, že urýchľujú proces programovania, keď je robot v zložitom prostredí. Namiesto toho, aby sme sami plánovali každý jeden pohyb, môže plánovač pohybu automaticky vytvoriť jednu alebo viacero dobrých trás, po ktorých môže robot kráčať. Zložité prostredie v tomto prípade znamená prostredie, kde je veľa prekážok. Pre mnohé aplikácie robotov je cesta medzi dvoma miestami v jeho pracovnom priestore bez prekážok. Ak áno, jednoduchý príkaz „Joint Move“ je rýchlejší a efektívnejší. Keď sa však v prostredí nachádzajú prekážky, existuje nebezpečenstvo, že by sa s nimi robot mohol zraziť. V tomto prípade je manuálne plánovanie menej efektívne ako použitie plánovača pohybu.

3.1 Predstavenie path planningu

Path planning [5], alebo plánovanie pohybu je jedným z najdôležitejších výzkumných problémov v robotike z pohľadu riadiaceho inžiniera. Mnohé problémy v rôznych oblastiach sa riešia návrhom plánovania trasy. Je využívaný pri navádzaní robota k dosiahnutiu konkrétneho cieľa od veľmi jednoduchého plánovania trajektórie až po výber vhodnej postupnosti akcií. Plánovanie trasy nemôže byť vždy navrhnuté vopred, pretože informácie o globálnom prostredí nie sú vždy dostupné. Navrhnutím správneho algoritmu môže byť plánovanie cesty široko používané v čiastočne neznámych a neznámych štruktúrovaných prostrediach.

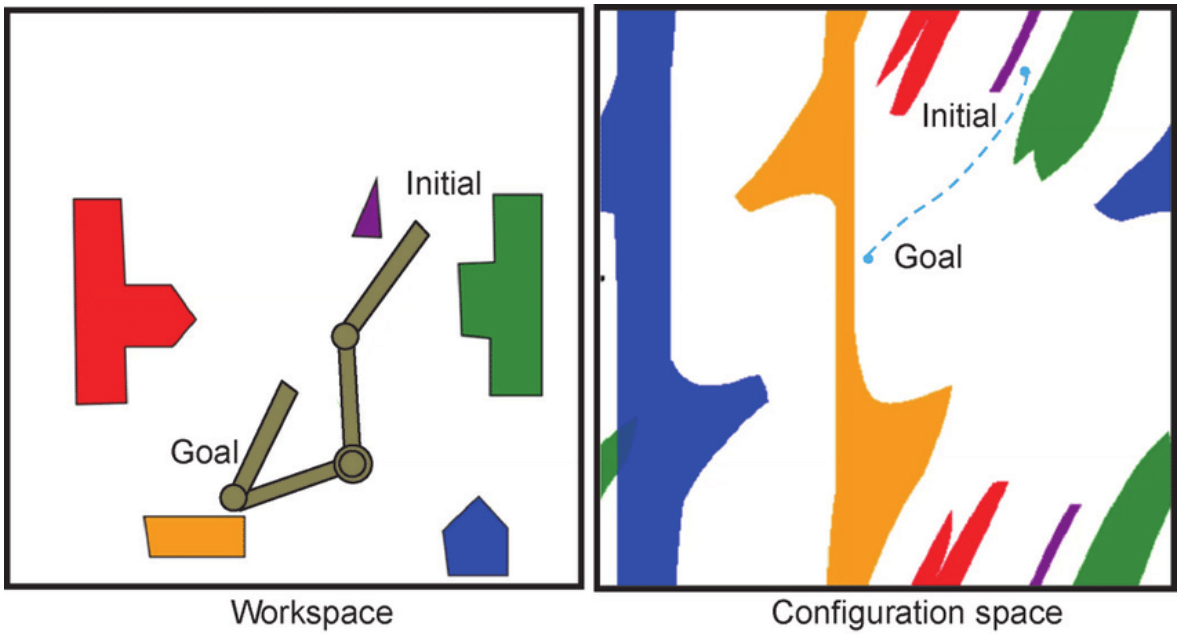
4 KONCEPTY

Aj keď sa na plánovanie cesty robotického pohybu pre rôzne roboty používajú rôzne matematické nástroje, niektoré koncepty [6] sú rovnaké pre všetky scenáre. Jednoduchým problémom plánovania pohybu by bolo naplánovať súvislú dráhu medzi počiatočnou pozíciou A a požadovanou pozíciou B. S tým by boli spojené obmedzenia. Rameno by pri pohybe do požadovanej polohy nemalo narážať na žiadne steny alebo predmety.

4.1 Configuration space [7]

Pre vytvorenie plánu pohybu pre roboty musíme byť schopní špecifikovať polohu robota. Presnejšie povedané, musíme byť schopní poskytnúť špecifikáciu umiestnenia, teda každý bod robota, pretože musíme zabezpečiť, aby sa žiadny jeho bod nezrazil s prekážkou. Toto vyvoláva niekoľko základných otázok:

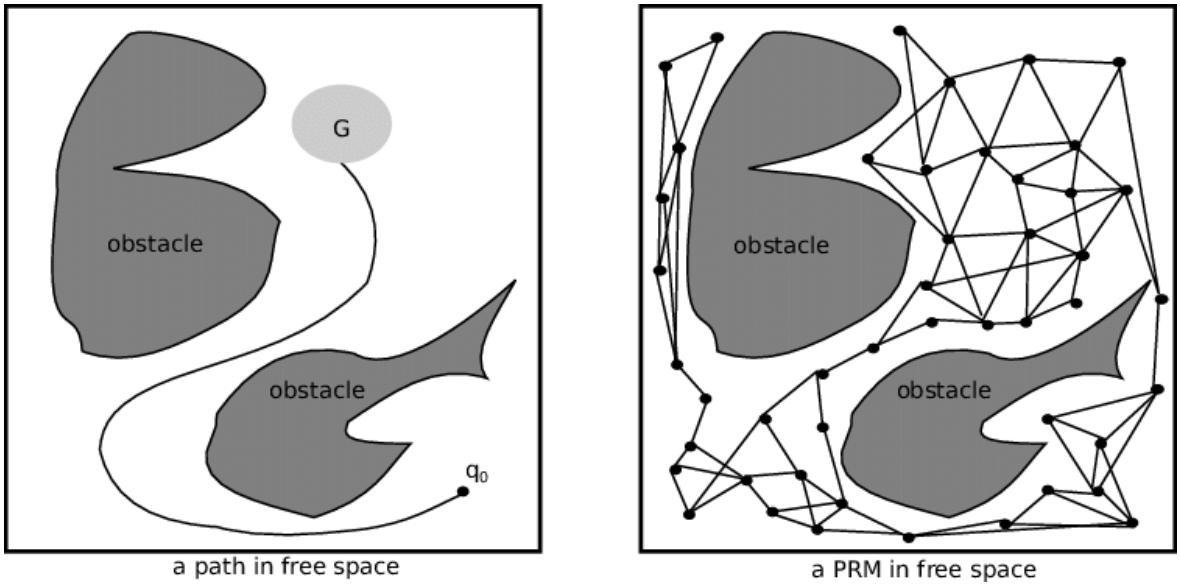
- Koľko informácií je potrebné pre úplnú špecifikáciu každého bodu u robota?
- Ako by mali byť tieto informácie zastúpené?
- Aké sú ich matematické vlastnosti reprezentácie?
- Ako možno brať do úvahy prekážky vo svete robotov pri plánovaní cesty robota?



Obrázok č.2 Rozdiel medzi pracovným priestorom a konfiguračným priestorom [30]

4.2 Free space [8]

Pokiaľ by sme mapovali prekážku z karteziánského priestoru do priestoru C, musíme si byť istý, že každý bod v priestore C, ktorý by viedol ku kolízii, je uložený v našej reprezentácii. Ak vynecháme jeden bod, plánovač pohybu môže vytvoriť plán pohybu, ktorý vedie ku kolízii. Preto vynakládáme veľké množstvo výpočtového času a pamäte, aby sme sa uistili, že naše mapovanie je úplné. Tento problém môžeme prekonať riešením duálneho problému. To znamená, že namiesto určovania oblastí C-priestoru, ktoré vedú ku kolízii s objektom, určujeme oblasti C-priestoru, ktoré sú zaručene bez kolízie. Tieto oblasti nazývame voľné plochy a je oveľa rýchlejšie počítať, ukladať a pracovať s nimi. Tým náš plánovač pohybu stratí prístup k niektorým bodom C-priestoru, ktoré sú bez kolízií, ale nie sú zahrnuté vo vypočítaných voľných priestoroch, ale výhody v čase a pamäti to viac než vynahradia.



Obrázok č.3 Rozdiel cesty vo voľnom priestore [31] a pravdepodobnou cestou

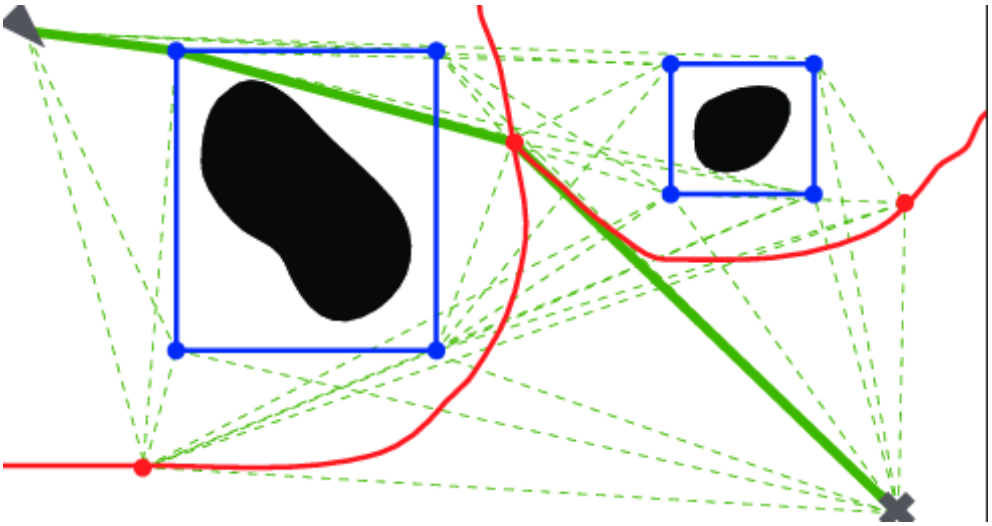
4.3 Target space [9]

Je to požadovaná poloha, ktorú má robot dosiahnuť. V závislosti od prípadu použitia môže byť zohľadnené iba koncové umiestnenie. Pozícia spojení robota môže byť ľubovolná, pokiaľ je umiestnenie platné. Robot musí prejsť z aktuálního priestoru do cieľového priestoru cez voľný priestor.

4.4 Obstacle space [10]

Je definovaný ako oblast, kde nie je povolený robot. Mohlo by to byť spôsobené prekážkami, alebo preto, že je to za okrajom konfiguračného priestoru, alebo zakázané z dôvodu iných obmedzení v konfigurácií robota.

$C_{obs} = C - C_{free}$.



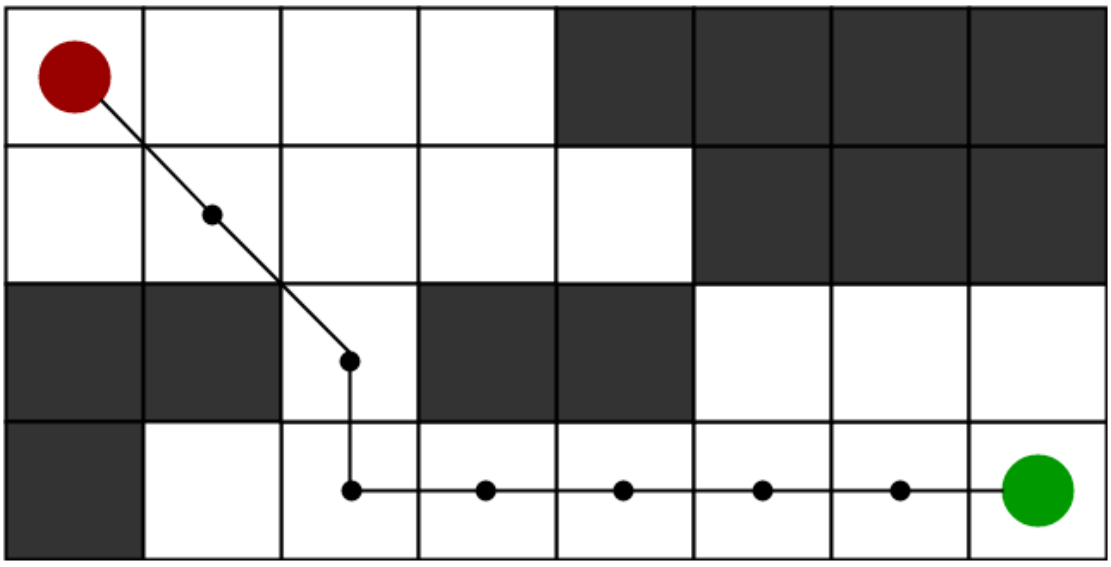
Obrázok č.4 Priestor s prekážkami [32]

5 ALGORITMY

Algoritmy [11] plánovania pohybu sú ústredným prvkom riadenia mobilných robotov, keďže prispievajú k ich vytváraniu schopnosti autonómneho správania. Navyše takáto trieda algoritmov je kritická, pretože zlyhanie môže spôsobiť prerušenie misie alebo môže spôsobiť veľké škody, ako je ľudská strata. Validácia takýchto algoritmov je potom povinná s cieľom zvýšiť dôveru koncových používateľov. Tieto algoritmy tiež podliehajú obmedzeniam, napr spotreba paliva. V dôsledku toho je výpočet bezpečnej cesty zvyčajne nedostačujúce hľadanie pre minimalizáciu nákladov.

5.1 Grid-based vyhľadávanie [12]

Hľadanie cesty v počítačových hrách môže byť koncepčne jednoduché, ale pre mnohých v herných doménach je ťažké ich robiť dobre. Obmedzenia v reálnom čase obmedzujú zdroje – čas aj priestor, ktoré možno použiť na hľadanie cesty. Jedno riešenie je znížiť zrnitosť mriežky, čo má za následok menší priestor na vyhľadávanie. Toto poskytuje hrubšie zobrazenie, ktoré je často pre používateľa rozpoznateľné. Ďalším riešením je podvádzať a nechať postavy pohybovať sa nereálnymi spôsobmi. Samozrejme, tretie riešnie je získať rýchlejší procesor.

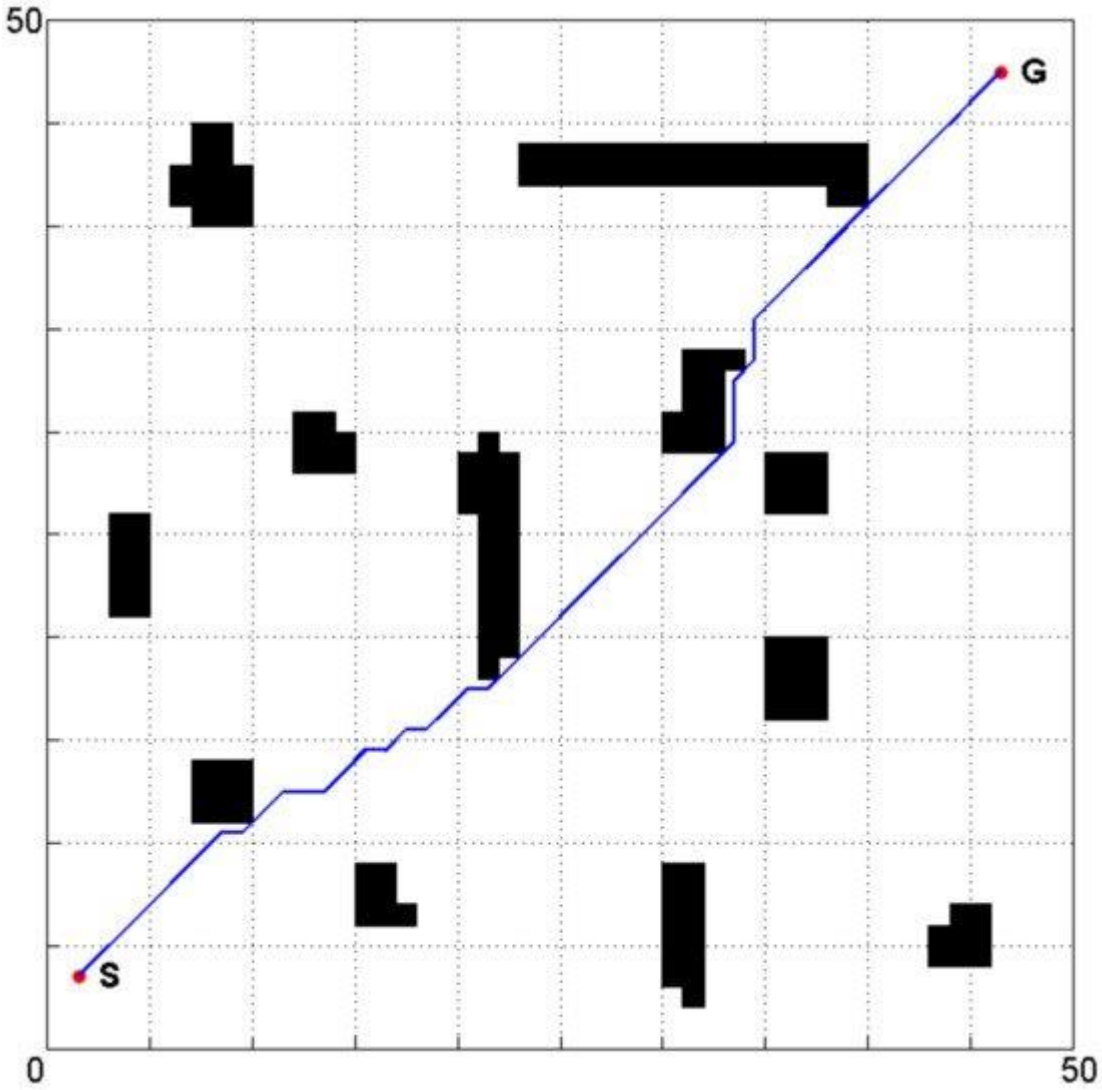


Obrázok č.5 Príklad Grid-based vyhľadávania [33]

5.2 Geometrický algoritmus [13]

Problém plánovania cesty v 2D sa týka plánovania cesty bez kolízie pre mobilného robota medzi rovinnou sadou nesúvislých polygonálných prekážok. Kvalita cesty sa môže merať z hľadiska jej dĺžky, hladkosti, vzdialenosť od prekážok alebo kombináciou týchto a iných faktorov. Používame termín „optimálny“ pre cesty, ktoré spĺňajú viac ako jedno kvalitatívne kritérium. Prístupy k path planning problému možno rozdeliť do troch základných kategórií:

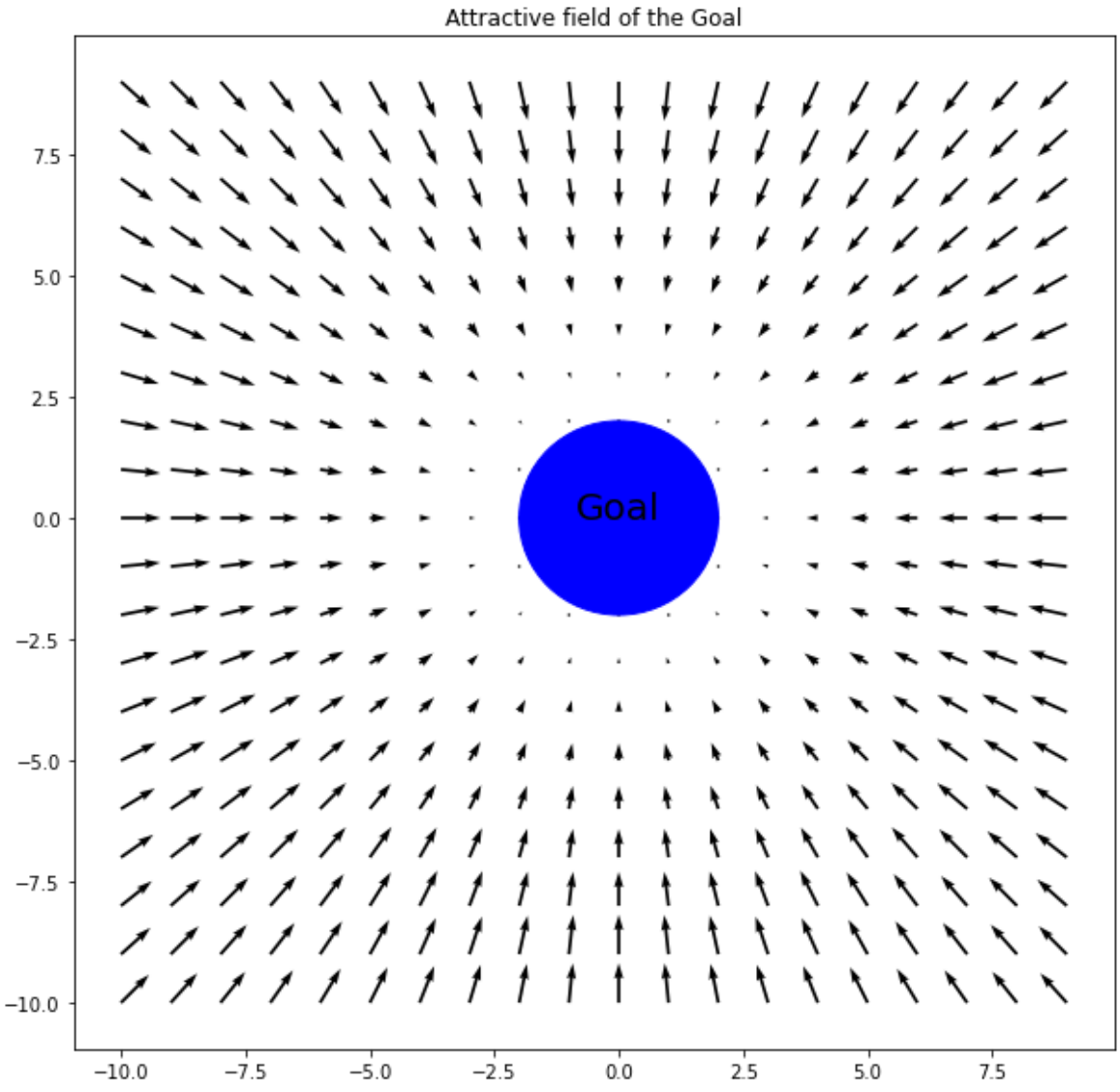
- Metóda cestovnej mapy
- Metóda bunkovej dekompozície
- Potenciálne pole



Obrázok č.6 Geometrický algoritmus [34]

5.3 Artificial potential fields

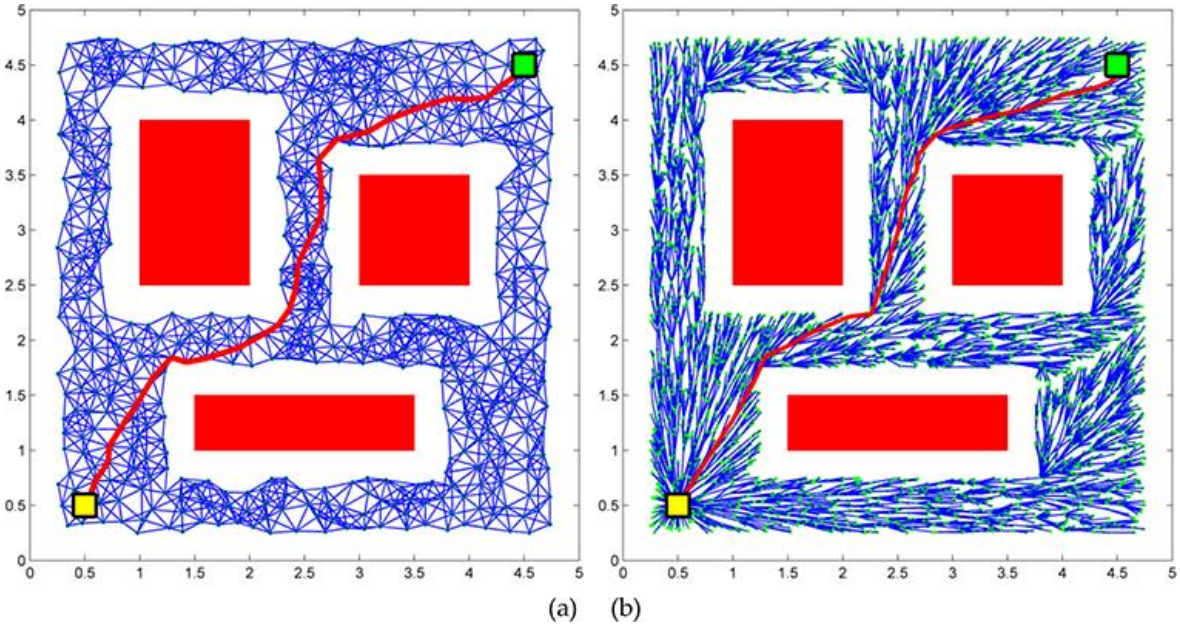
Myšlienka potenciálneho poľa [14] je prevzatá z prírody. Napríklad nabitá častica prechádzajúca magnetickým poľom alebo malá gulička kotuľajúca sa v kopci. Ide o to, že v závislosti od sily poľa alebo sklonu kopca môže častica alebo guľa v tomto príklade doraziť k zdroju poľa, magnetu alebo údoliu. V robotike môžeme simulovať rovnaký efekt vytvorením umelého potenciálneho poľa, ktoré pritiahne robota k cieľu. Navrhnutím adekvátního potenciálneho poľa môžeme prinútiť robota vykazovať jednoduché správanie. Predpokladajme napríklad, že v prostredí nie je žiadna prekážka a že tento cieľ by mal hľadať robot. Pri konvenčnom plánovaní je potrebné vypočítať relatívnu polohu robota k cieľu a potom použiť vhodné sily, ktoré budú hnať robota k cieľu.



Obrázok č.7 Potencionálna pole cieľa [35]

5.4 Sampling-based algoritmus

Pravdepodobne najvplyvnejšie algoritmy plánovania pohybu založené na vzorkovaní [15] k dnešnemu dňu zahŕňajú pravdepodobnostné cestovné mapy a rýchlo skúmajúce náhodné stromy. Aj keď myšlienka spájania náhodne vzorkovaných bodov zo stavového priestoru je ohnutá v oboch prístupoch, tieto dva algoritmy sa líšia v spôsobe, akým vytvárajú graf spájajúci tieto body.



Obrázok č.8 Ukážka Sampling-based algoritmu [36]

6 APLIKÁCIE

6.1 Robotická navigácia

Navigácia autonómnych mobilných robotov [16] je nazývaná aj schopnost umelého agenta pohybovať sa smerom k určenému trasovému bodu hladko a bez kolízie. Pritiahla veľké množstvo pozornosti a výskumu, čo viedlo k stovkám priblížení v priebehu niekoľkých desaťročí. Autonómna navigácia leží na priesečníku mnohých rôznych základných oblastí výskumu v robotike. Presun robota z jednej pozície do druhej si vyžaduje techniky vnímania, odhadu stavu (napr. Lokalizácia, mapovanie a reprezentácia sveta), plánovanie cesty a plánovanie riadenia pohybu. Aj keď bolo navrhnutých mnoho sofistikovaných autonómnych navigačných systémov, zvyčajne sa riadia klasickou hierarchickou plánovacou paradigmou.

6.2 Automatizácia

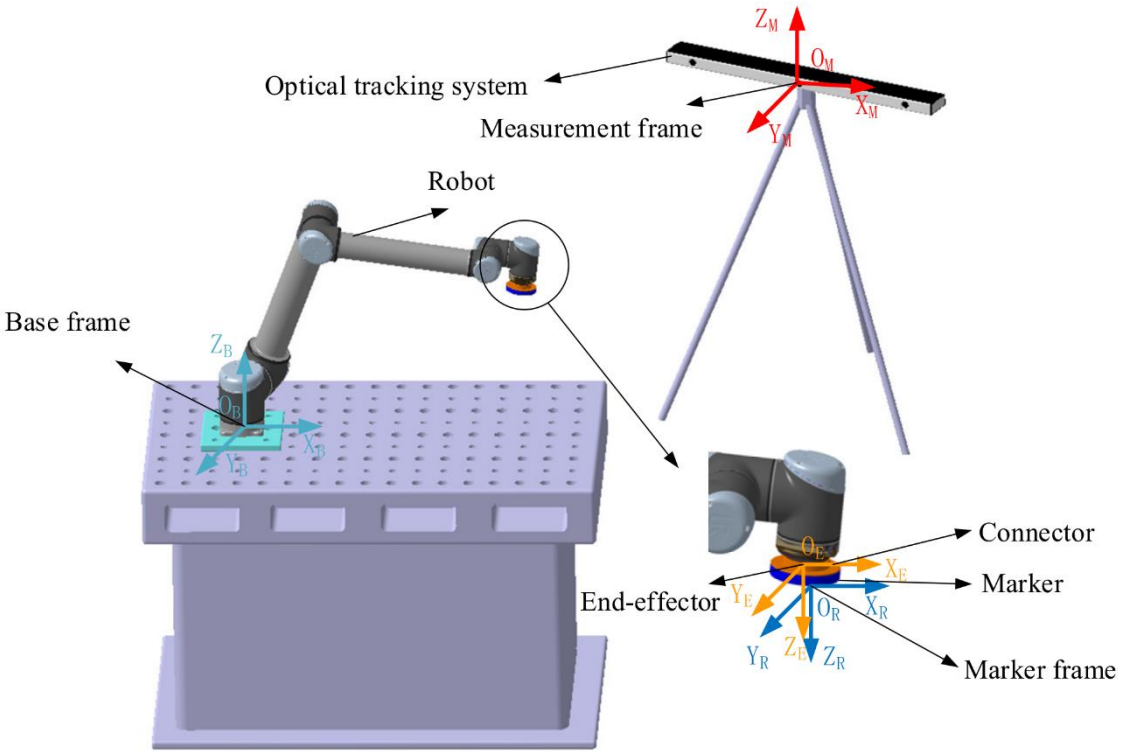
Spoločnosť Realtime Robotics vyvinula ovládač v reálnom čase, ktorý vám umožňuje rýchlo nasadiť pracovnú bunku s viacerými robotmi, ktorá maximalizuje kolektívny výkon. Všetky roboty sa pohybujú rýchlo, aby vykonávali simulované úlohy, len zriedka spomaľujú alebo zastavujú, aby sa vyhli vzájomným kolíziám [17].

6.3 Driverless car

Problém autonómnej mestskej jazdy [18] dostal značnú pozornost od DARPA Urban Challenge (DUC) v roku 2007. Plánovanie pohybu v mestskom prostredí však zostáva náročné z rôznych dôvodov. Zatiaľ čo štruktúru prostredia ako aj pravidlá cesty zvyčajne umožňujú plánovačom urobiť niekoľko trajektórií, tak zjednodušením predpokladov môže byť autonómne vozidlo stále nespoľahlivé a môže sa dostať do zložitých situácií v dôsledku neočakávaného správania inými účastníkmi premávky. Okrem toho sa môže ocitnúť v oblastiach, ktorým chýba jasne rozpoznatelná štruktúra, teda čo si vyžaduje použitie výkonnejších plánovacích algoritmov než tie, ktoré sa bežne používajú pri jazde na ceste.

6.4 Robotic operation

V posledných rokoch boli konvenčné sériové roboty [19] s manipulátormi úspešne aplikované a dodatočne vyvinuté v presných automatizačních procesoch pre rôzne odvetvia. Pre odvetvie lekárstva [20], najmä pri minimálne invazívnych chirurgických zákrokoch, vyvolala pokrokový záujem o výskum v dôsledku zlepšenia presnosti a ich nižších nákladov týkajúcich sa špecializovaných chirurgických robotov. Napríklad bolo zavedené nové robotické zavádzanie kontroly a presnosti chirurgického nástroja s cieľom znížiť traumu pacientov.



Obrázok č.9 Robot [37]

II. PRAKTICKÁ ČASŤ

7 POUŽITÉ TECHNOLÓGIE

Python [20] sa v posledních rokoch stal jedným z najpopulárnejších programovacích jazykov na svete. Používa sa vo všetkom, od strojového učenia po vytváranie webových stránok a testovanie softvéru. Môžu ho používať vývojári aj nevývojári. Python je jeden z najpopulárnejších programovacích jazykov na svete, vytvoril všetko od odporúčacieho algoritmu Netflix až po softvér, ktorý ovláda samojazdiace auta. Python je univerzálny jazyk, čo znamená, že je navrhnutý na použitie v rôznych aplikáciách vrátane vedy o údajoch vývoja softvéru a webu, automatizácie a všeobecné vykonávanie vecí. Je často využívaný na vytváranie webových stránok, automatizáciu úloh a analýzu údajov. Je univerzálny jazyk, čo znamená, že ho je možné použiť na vytváranie rôznych programov a nie je špecializovaný na špecifické problémy. Táto všestrannosť, spolu s jeho prívetivosťou pre začiatočníkov, z neho urobili jeden z napoužívanejších programovacích jazykov šúčasnosti. Python pôvodne navrhol [21] Guido van Rossum v roku 1991 a vyvinula ho Python Software Foundation. Bol vyvinutý hlavne pre dôraz na čitateľnosť kódu a jeho syntax umožňuje programátorom vyjadrovať koncepty v menšom počte riadku kódov. Koncom 80. rokov minulého storočia sa mala písať história. V tom čase sa začalo pracovať na Pythone. Čoskoro potom začal Guido Van Rossum v decembri 1989 vykonávať prácu založenú na aplikácií v Centre Wiskunde & Informatica (CWI), ktoré sa nachádza v Holandsku. Začalo to najskôr ako hobby project, pretože hľadal zaujímavý project, ktorý by ho zamestnal počas Vianoc. Programovací jazyk, o ktorom sa hovorí, že uspel, je programovací jazyk ABC, ktorý mal prepojenie s operačným systémom Amoeba a mal funkciu spracovania výnimiek.

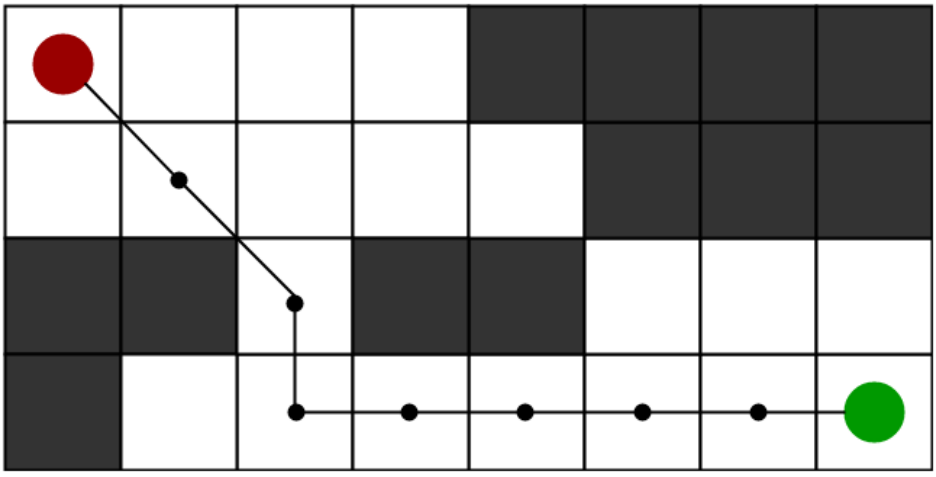
7.1 Algoritmy

Pre vytvorenie praktickej časti som využil 4 rôzne algoritmy. Využité algoritmy som potom spracoval v programovacom jazyku Python na vytvorených scenároch.

7.1.1 A*

Tento algoritmus [22] je vyhľadávací algoritmus, ktorý hľadá najkratšiu cestu medzi počiatočným a konečným stavom. Používa sa v rôznych aplikáciách, ako sú mapy. V mapách sa na výpočet najkratšej vzdialenosti medzi zdrojom, ktorý je označovaný ako počiatočný stav a cieľom, ktorý je stav koncový používa algoritmus A*. Na vysvetlenie fungovania tohoto algoritmu si predstavme štvorcovú sieť s mnohými prekážkami, kde dostaneme počiatočnú bunku a cieľovú bunku. Pokiaľ je to možné, tak sa chceme dostať do cieľovej bunky čo najrýchlejšie. Tu na záchranu prichádza A* [23] search algoritmus. Tento algoritmus robí to, že v každom kroku vyberie uzol podľa hodnoty „f“, čo je parameter rovný súčtu dvoch ďalších parametrov – „g“ a „h“. V každom kroku vyberie uzol alebo bunku s nejnižším „f“ a spracuje tento uzol/bunku. Nižšie definujeme „g“ a „h“ čo najjednoduchšie, preto g = cena pohybu na presun z počiatočného bodu na dané políčko na mriežke do konečného cieľa. Toto sa často označuje ako heuristika, čo nie je nič iné ako druh inteligentného odhadu. Skutočnú vzdialenosť nepoznáme, kým nenájdeme cestu, pretože v ceste môžu byť rôzne veci (steny, voda, atď.) Existuje mnoho spôsob, ako vypočítať toto „h“. Pre vypočítanie „h“ môžeme napríklad urobiť 2 kroky

- vypočítať presnú hodnotu h, ale toto riešenie je časovo náročné.
- Aproximovať hodnotu h pomocou heuristiky, toto riešenie je menej časovo náročné.

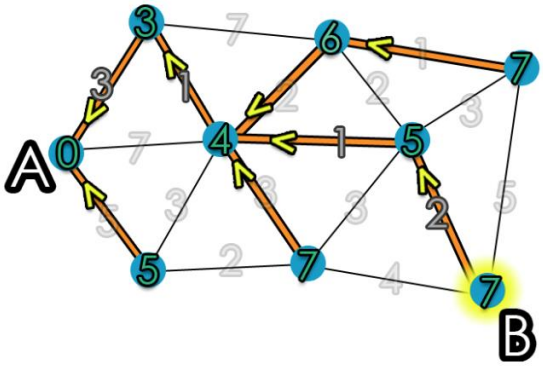


Obrázok č.10 Cesta A* [38] algoritmu v priestore

7.1.2 Dijkstra

Jedným z algoritmov, ktorý sa zaoberá hľadaním najkratšej cesty od počiatočného uzla k cieľovému uzlu v grafe je algoritmus Dijkstra [24]. Tento algoritmus tvorí strom najkratších ciest z počiatočného vrcholu, zdroja a do všetkých ostatných bodov v grafe. Algoritmus Dijkstra bol publikovaný v roku 1959 a pomenovaný podľa svojho tvorcu, holandského počítačového vedca Edsgera Dijkstra. Dijkstrov algoritmus je možné použiť na vážený graf. Graf môže byť orientovaný alebo neorientovaný. Jednou z podmienok používania algoritmu je, že graf musí mať na každej hrane nezápornú váhu. Základom funkčnosti Dijkstrovho algoritmu je to, že akákoľvek podcesta B-> D najkratšej cesty A-> D medzi vrcholmi A a D je zároveň najkratšou cestou medzi vrcholmi B a D. Dijkstra použil túto vlastnosť v opačnom smere, tj. preceňujeme vzdialenosť každého vrcholu od počiatočného vrcholu. Potom navštívíme každý uzol a susedné uzly, aby sme našli najkratšiu podcestu k týmto susedom. Algoritmus používa chamtivý prístup v tom zmysle, že nájdeme ďalšie najlepšie riešnie v nádeji, že konečný výsledok bude nejlepším riešením celého problému. Základy Dijkstrovho algoritmu [25]:

- Dijkstrov algoritmus začína na uzle, ktorý si vyberieme, tento uzol sa nazýva aj zdrojový uzol. Po tomto kroku sa analyzuje graf, aby našiel najkratšiu cestu medzi týmto uzlom a všetkými ostatnými uzlami v grafe.
- Algoritmus sleduje aktuálne známu najkratšiu vzdialenosť od každého uzla k zdrojovému uzlu a aktualizuje tieto hodnoty ,ak nájde kratšiu cestu.
- Keď algoritmus nájde najkratšiu cestu medzi zdrojovým uzlom a iným uzlom, tento uzol sa označí ako “navštívený” a pridá sa k ceste.
- Proces pokračuje, kým sa do cesty nepridajú všetky uzly v grafe. Týmto spôsobom máme cestu, ktorá spája zdrojvý uzol so všetkými ostatnými uzlami po najkratšej možnej ceste na dosiahnutie každého uzla.



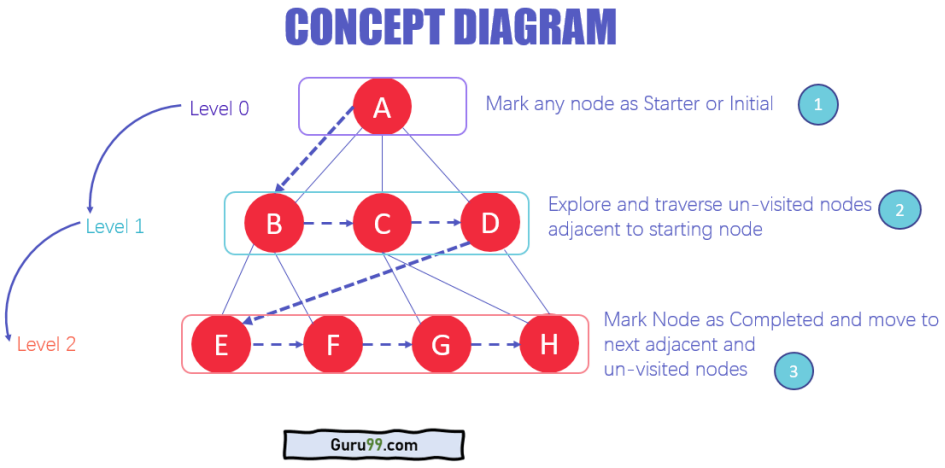
Obrázok č.11 Práca algoritmu Dijkstra [39]

7.1.3 BFS – Breadth First Search

BFS [26] algoritmus je algoritmus prechodu grafu, ktorý začína prechádzať grafom od koreňového uzla a skúma všetky susedné uzly. Potom vyberie najbližší uzol a preskúma všetky nepreskúmané uzly, Pri použití BFS na prechod možno za koreňový uzol považovať ktorýkoľvek uzol v grafe. Existuje mnoho spôsobov, ako prechádzať grafom, ale spomedzi nich je BFS najbežnejšie používaným prístupom. Ide o rekurzívny algoritmus na vyhľadávanie vrcholov stromovej alebo grafovej dátovej štruktúry. BFS rozdeľuje vrchol grafu do dvoch kategórií – navštívené a nenavštívené. Vyberie jeden uzol v grafe a potom navštívi všetky uzly, ktoré susedia s vybraným uzlom.

Aplikácie BFS algoritmu:

- BFS je možné použiť k nájdeniu susedných miest z daného zdroja
- V sieti typu peer-to-peer je možné použiť algoritmus BFS ako metódou prechodu na nájdenie všetkých susedných uzlov. Väčšina torrent klientov, ako sú BitTorrent, uTorrent atď., využíva tento process na nájdenie "seedov" a "peers" v sieti.
- BFS je možné použiť vo webových prehliadačoch na vytváranie indexov webových stránok. Je to jeden z hlavných algoritmov, ktoré je možné použiť na indexovanie webových stránok. Začne prechádzať zo zdrojovej stránky a nasleduje odkazy spojené so stránkou. Tu sa každá webová stránke považuje za uzol v grafe.
- BFS sa používa na určenie najkratšej cesty
- Používame ho v Cheneyho technike na duplikovanie zberu odpadu
- Môže byť využitý vo ford-Fulkersonovej metóde na výpočet maximálneho prietoku v prietokovej sieti

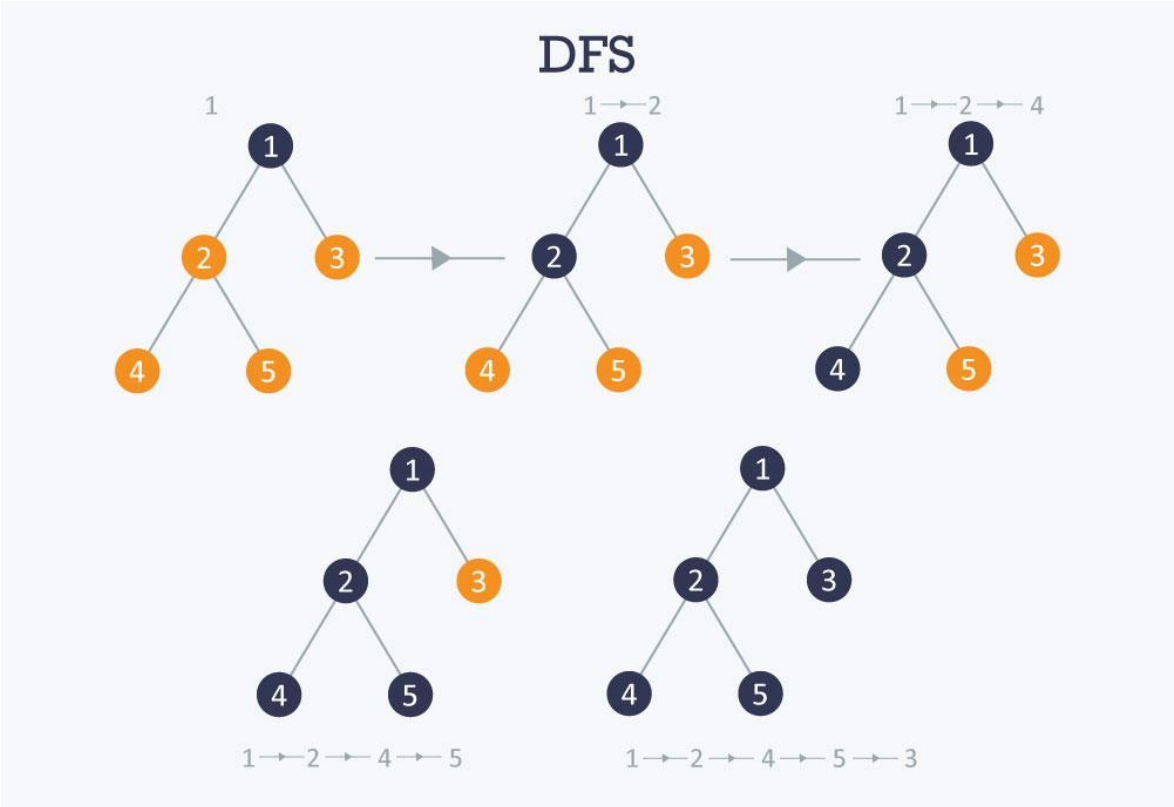


Obrázok č.12 Koncepčný diagram BFS algoritmu [40]

7.1.4 DFS – Depth First Search

DFS [27] algoritmus je rekurzívny algoritmus, ktorý využíva myšlienku spätného sledovania. Zahŕňa vyčerpávajúce prehľadávanie všetkých uzlov postupovaním dopredu, ak je to možné, inak spätným sledovaním. Slovo backtrack pri tomto algoritme znamená, že keď sa pohybujete vpred a na aktuálnej ceste už nie sú žiadne uzly, pohybujete sa späť po tej istej ceste, aby ste našli uzly, ktoré chcete prejsť. Všetky uzly budú navštívené na aktuálnej ceste, kým sa neprejdú všetky nenavštívené uzly, potom sa vyberie ďalšia cesta. Táto rekurzívna povaha DFS môže byť implementovaná cez zásobníky. Základná myšlienka je následovná:

- Je vybraný počiatočný uzol a vložené všetky susedné uzly do zásobníka.
- Vyberieme uzol zo zásobníka a vyberieme ďalší uzol, ktorý chceme navštíviť, a vložíme všetky jeho susedné uzly do zásobníka.
- Tento postup opakujeme pokiaľ nebude zásobník prázdny.
- Musíme sa však uistiť, že navštívené uzly sú označené.
- To nám zabráni navštíviť rovnaký uzol viac ako raz. Ak neoznačíme uzly, ktoré sú navštívené a rovnaký uzol navštívíme viackrát, môžeme sa dostať do nekonečnej slučky.

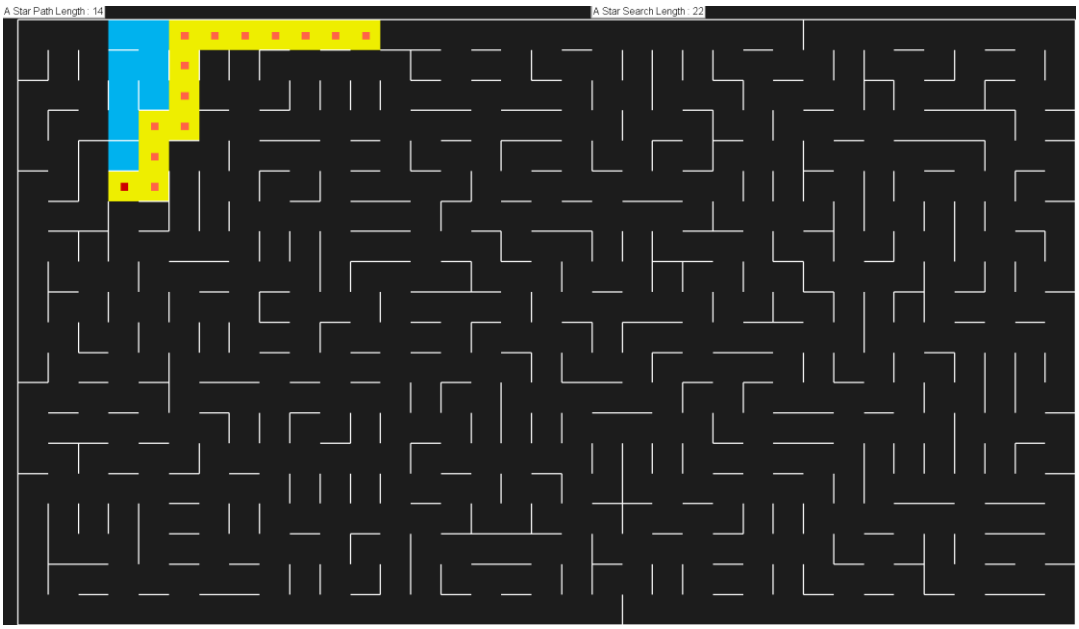


Obrázok č.13 Práca algoritmu DFS [41]

7.2 Testovacie scenáre

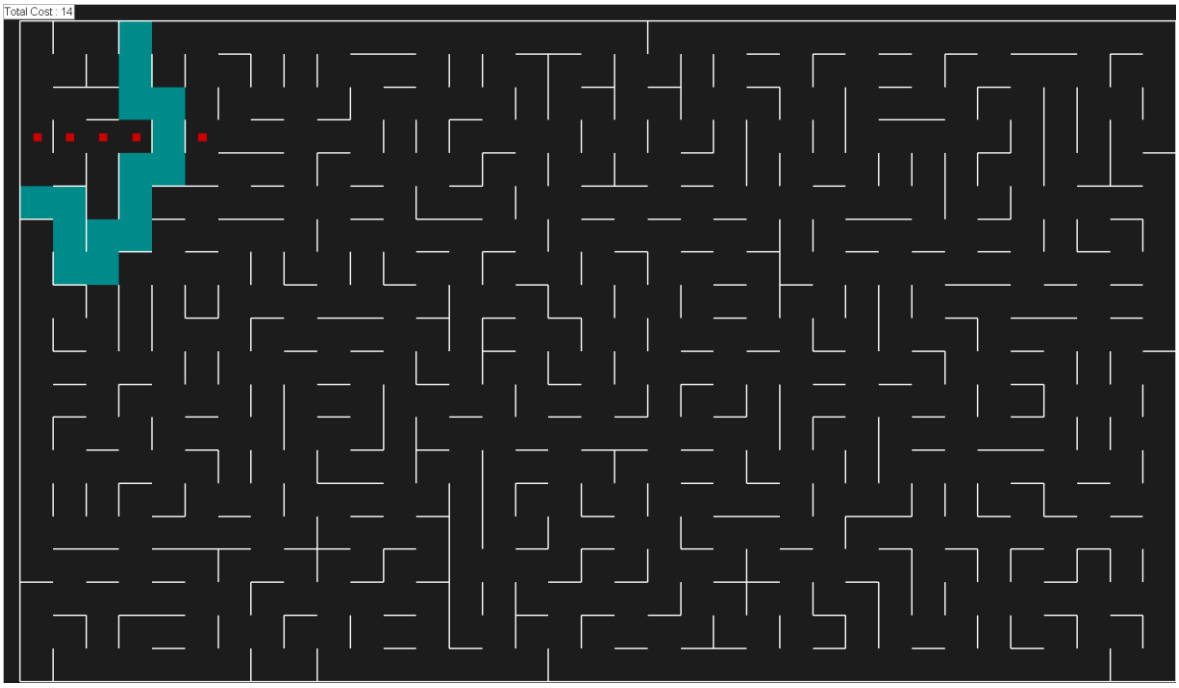
Pre otestovanie algoritmov A*, Dijkstra, BFS a DFS som si vytvoril testovacie scenáre, na ktorých som potom tieto algoritmy otestoval a vyhodnotil výsledky. Ako prvý testovací scenár som si zvolil testovací scenár v bludisku [29]. Na tomto scenári som potom porovnal algoritmy A*, Dijkstra a algoritmy BFS, DFS.

7.2.1 Bludisko A*



Obrázok č.14 Algoritmus A* v bludisku

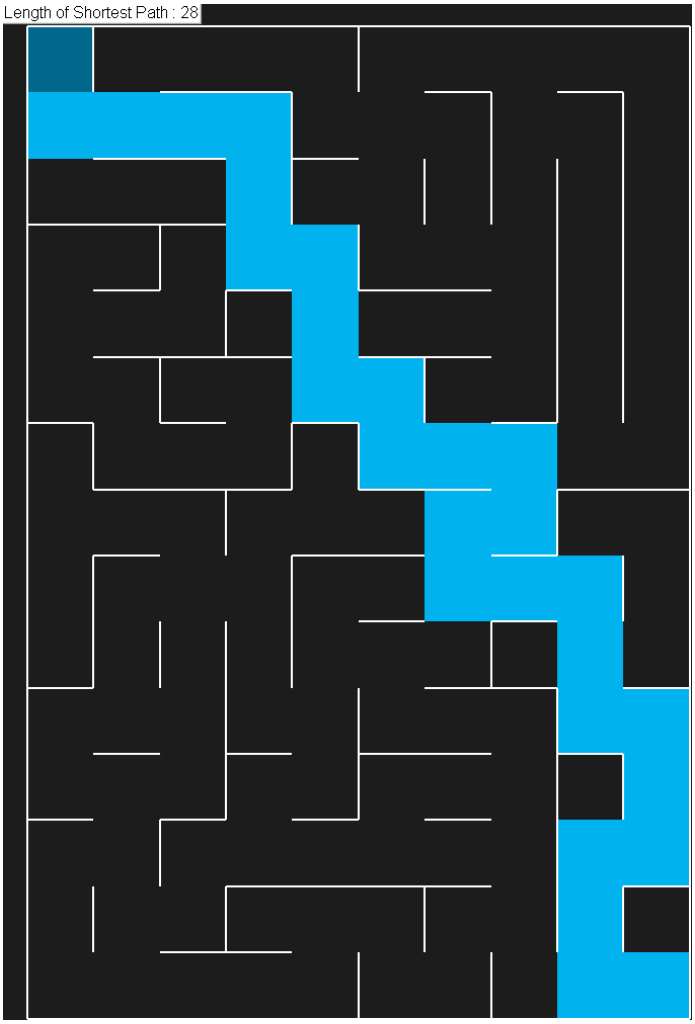
7.2.2 Bludisko Dijkstra



Obrázok č.15 Algoritmus Dijkstra v bludisku

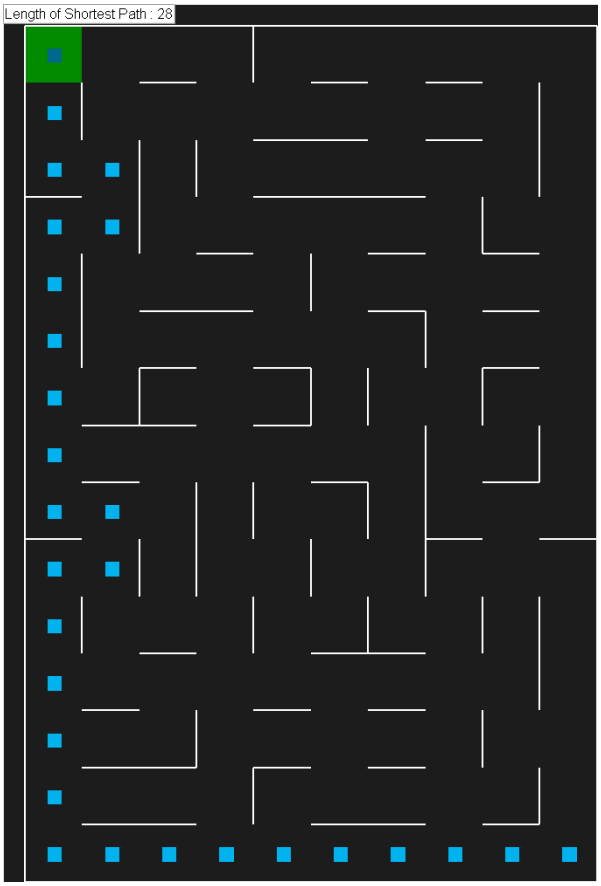
Pri vytváraní bludiska [29] som si stanovil základný parameter rozmer. Rozmer som nastavil na hodnotu 20x35. Časová náročnosť na vyriešenie bludiska tejto veľkosti nie je veľká a vďaka rýchlo pracujúcim algoritmom sa pohybuje v rámci sekúnd. Samotná časová náročnosť je ovplyvňovaná veľkosťou bludiska a zvolením štartového a cieľového bodu v bludisku, v tomto prípade nebolo bludisko veľkých rozmerov a zvolené body ďaleko od seba, takže algoritmy si s vyriešením tohoto problému poradili veľmi rýchlo.

7.2.3 Bludisko BFS



Obrázok č.16 Algoritmus BFS v bludisku

7.2.4 Bludisko DFS



Obrázok č.17 Algoritmus DFS v bludisku

V tomto prípade boli rozmery bludiska [29] nastavené tiež na hodnotu 20x35. Časová náročnosť k vyriešeniu tohoto bludiska pomocou algoritmov BFS a DFS bola väčšia ako v predchádzajúcich testoch. Po časovej náročnosti som zistil, že algoritmy BFS a DFS sú pomalšie a ako algoritmy Dijkstra a A*.

7.2.5 A* vs Dijkstra

Pre porovnanie algoritmov v bludisku som vytvoril testovací scenár, ktorý je zameraný na algoritmy A* a Dijkstra, jeho úlohou bolo zistiť, ktorý algoritmus sa dostane zo zdrojového bodu do cieľového bodu. Základnou podmienkou, ktorú som stanovil je rovnaký rozmer bludiska. Pre vyriešenie bludiska a stanovenie správneho algoritmu som vyskúšal algoritmy desaťkrát a výsledky som vyhodnotil pomocou priemeru.

Tabulka č.1Výsledky algoritmov A* a Dijkstra

Algoritmus	**A***	**Dijkstra**
	Dĺžka cesty	**Dĺžka cesty**
Výsledok1	16	14
Výsledok2	14	16
Výsledok3	14	14
Výsledok4	14	12
Výsledok5	14	16
Výsledok6	14	10
Výsledok7	14	10
Výsledok8	16	20
Výsledok9	14	14
Výsledok10	14	14
Priemer	14,4	14

Podľa tabuľky je vidieť, že priemerná dĺžka cesty pri algoritme A* je 14,4 kroku, kdežto v porovnaní s algoritmom Dijkstra, ktorého priemerná dĺžka cesty je 14 krokov to znamená, že v priemere algoritmus A* potrebuje spraviť väčší počet krokov k dosiahnutiu požadovaného cieľa. V tomto momente je hodnota 0,4 zanedbateľná, ale môže nastať situácia, v ktorej budeme potrebovať otestovať viac prípadov a tým pádom sa môže veľkým spôsobom navýšiť priemerná dĺžka trasy od štartu k cieľu. Z tohoto vyplýva, že algoritmus Dijkstra je na základe jeho vlastností rýchlejší a praktickejší k dosiahnutiu požadovaného cieľa. Počas testovania nastala situácia kedy pri menšom počte testov bola dĺžka cesty oboch algoritmov identická, čo znamená, že pri jednoduchej a krátkej trase je využitie algoritmu na preferencii samotného programátora.

7.2.6 BFS vs DFS

V testovacom scenári, ktorý porovnáva algoritmy pri prechode bludiskom som sa rozhodol porovnať algoritmy BFS a DFS, tieto algoritmy som porovnal rovnako algoritmy A* a Dijkstra 10krát pre dosiahnutie priemerného výsledku. Pri testovaní algoritmov som nastavil rozmery bludiska na 15x10, s týmto rozmerom pracovali oba algoritmy.

Tabuľka č.2 Výsledky algoritmov BFS a DFS

Algoritmus	BFS	DFS
	Dĺžka cesty	Dĺžka cesty
Výsledok1	26	30
Výsledok2	26	24
Výsledok3	26	28
Výsledok4	26	26
Výsledok5	28	28
Výsledok6	26	26
Výsledok7	26	24
Výsledok8	30	24
Výsledok9	24	26
Výsledok10	26	34
Priemer	26,4	27

Z tabulky sú znova viditeľné minimálne rozdiely v dĺžke cesty medzi obomi algoritmami, rozdieľ medzi nimi je znovu minimálny na úrovni 0,6, čo nám ukazuje, že rovnako ako tohoto algoritmoch A* a Dijkstra je tento rozdieľ zanedbateľný. Algoritmus BFS potreboval spraviť 26,4 kroku na dosiahnutie cieľa, pričom algoritmus DFS potreboval 27 krokov. V tomto prípade nenastala situácia, kedy tohoto menšom počte testovaní daného scenáru vyšli výsledky rovnakej hodnoty. Výsledky oboch algoritmov sa od seba oddiaľovali a stále vyhrával algoritmus BFS. Z tohoto teda vieme posúdiť to, že algoritmus BFS je rýchlejší a efektívnejší v riešení bludiska.

7.2.7 Dijkstra vs BFS

Pre určenie najlepšieho algoritmu na vytvorenie bludiska som porovnal algoritmy, ktoré vyhrali predchádzajúce porovnanie. Porovnal som medzi sebou algoritmus Dijkstra a BFS, ktoré mali rovnaké podmienky. Každý algoritmus som otestoval 5krát a výsledky som potom pomocou priemeru vyhodnotil.

Tabuľka č.3 Výsledky algoritmov BFS a Dijkstra

Algoritmus	BFS	Dijkstra
	Dĺžka cesty	Dĺžka cesty
Výsledok1	56	59
Výsledok2	68	63
Výsledok3	60	59
Výsledok4	66	57
Výsledok5	66	59
Priemer	63,2	59,4

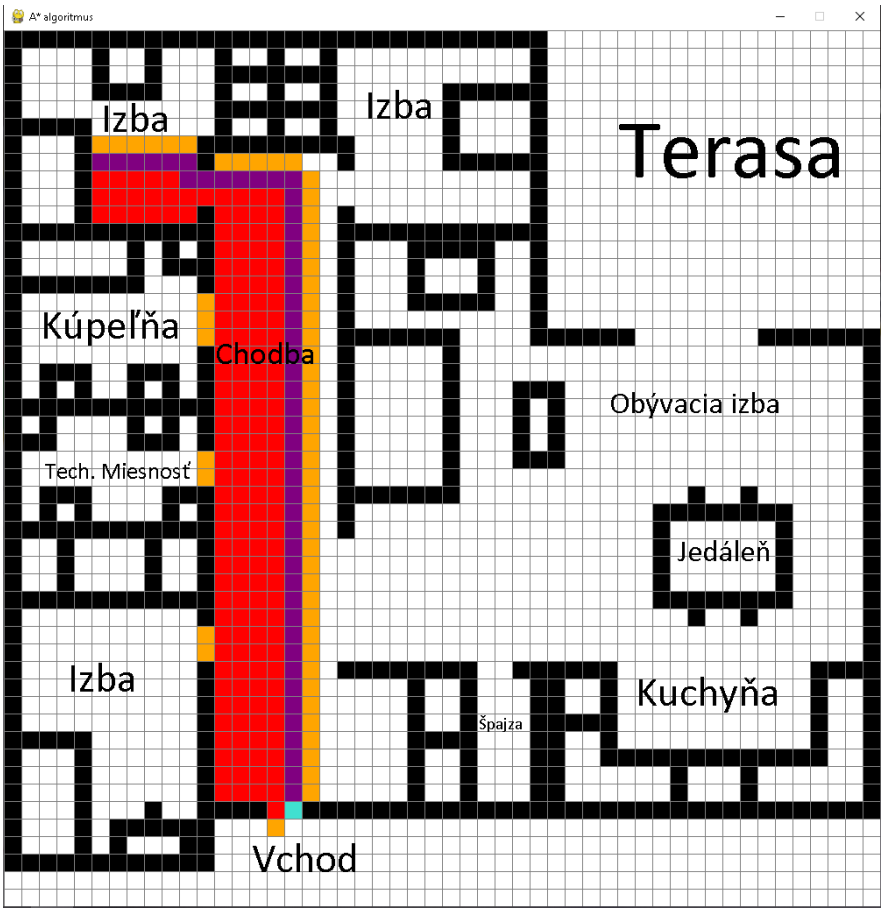
Na tejto tabuľke je vidieť, markantnosť rozdieľu pri hľadaní najkratšej cesty. Algoritmus Dijkstra v porovnaní s minulými testami prekonal algoritmus BFS. Už z 5 testov je vidieť, že algoritmus BFS potrebuje o dosť väčší počet krokov k dosiahnutiu výsledku ako algoritmus Dijkstra. Po piatich testovacích kolách je rozdiel v počte krokov 3,8, čo je v porovnaní s rozdielom z minulých testov veľký rozdiel. Výsledok, ktorý vyplýva zo všetkých výsledkov je následovný – pri porovnaní A* a Dijkstra algoritmov je v menšom počte testovaní výsledok bez rozdielu, ale v momente, kedy chcem otestovať väčší počet rôznych scenárov, tak algoritmus Dijkstra vyjde ako algoritmus, ktorý je efektívnejší a rýchlejší v hľadaní najkratšej cesty. Čo sa rozdielu medzi algoritmami BFS a DFS ,tak z výsledkov vidíme väčší rozdiel v počte krokov ako u algoritmov A* a Dijkstra. Tento rozdiel v počte krokov sa pomaly znižuje počtom testovaní, ktoré sú vytvárané. To znamená, že pokiaľ využívame veľa testovacích možností, tak by sme sa dopracovali k rovnakému počtu krokov u oboch algoritmov. V tomto prípade som tak ale neurobil, takže vyhral algoritmus BFS. Pre finálne porovnanie som si vybral algoritmy, ktoré vyhrali obe porovnania. Z tohto porovnania je vidieť jednoznačný rozdiel v počte krokov medzi algoritmami. Zo všetkých algoritmov, ktoré mali rovnaké podmienky teda vyhral algoritmus Dijkstra a je najlepšia možnosť k hľadaniu cieľového bodu v bludisku.

8 MRIEŽKA PLÁNOVANIA OPTIMÁLNEJ CESTY

Pomocou Mriežky plánovania optimálnej cesty [28] som vytvoril tri testovacie scenáre pre algoritmus A*.

8.1.1 Scenár byt

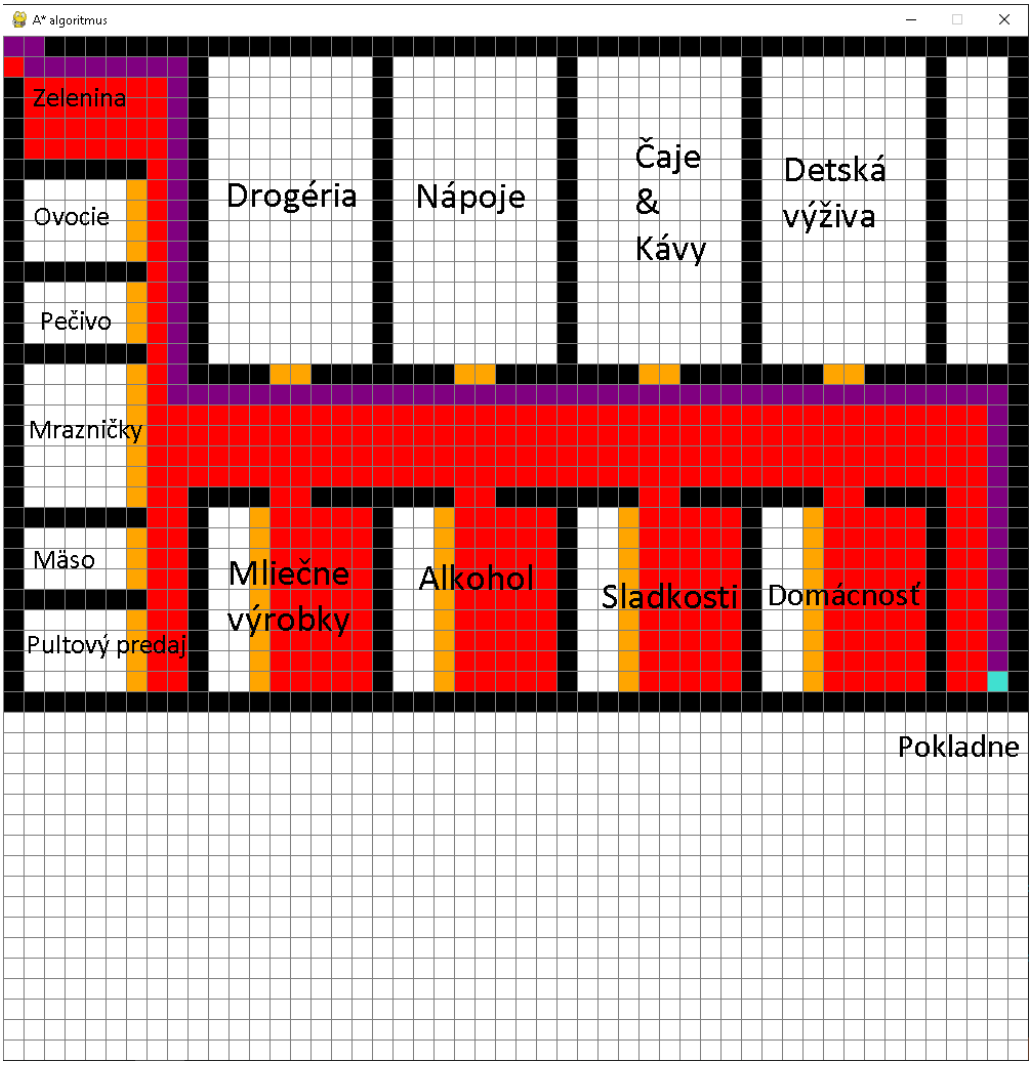
Úlohou tohoto scenáru je otestovanie správnej funkčnosti algoritmu A* pri navigácií v byte, kedy je jeho úlohou dostať sa použitím najkratšej cesty z izby bytu k východu. Byt bol navhrnutý tak, aby bol čo najviac priestranný a algoritmus mal teda viac možností preskúmať rôzne cesty. Časová náročnosť tejto úlohy je veľmi nízka, pretože práca algoritmu je dostatočne rýchla a je schopný cieľovú trasu nájsť pomerne rýchlo.



Obrázok č.18 Najkratšia cesta v byte

8.1.2 Scenár obchod

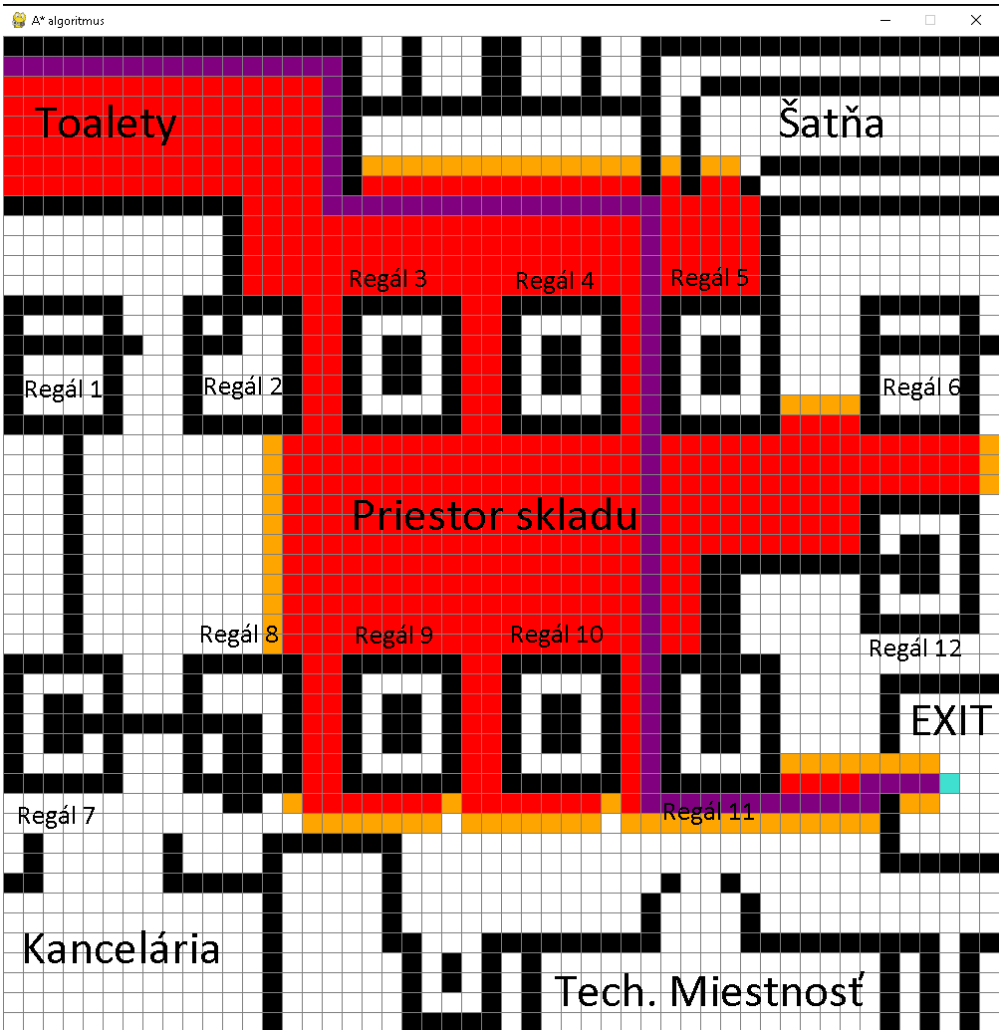
Tento scenár mal za úlohu otestovať algoritmus A* v navigácií v obchode. Typický príklad, navigácie v obchode je dostať robota zo vstupu k východu z obchodu v čo najkratšom čase. Jeho úlohou je splniť všetky podmienky, ktoré má stanovené, pričom sa musí vyhýbať stenám a možným prekážkam. V mojom prípade som pomocou mriežky nakreslil pôdorys obchodu, ktorý predstavuje trasu robota od sekcie so zeleninou, pretože jeho úlohou bolo spraviť nákup v tejto časti obchodu. Po vykonaní tohoto úkonu musel nájsť najrýchlejšiu cestu k východu, ktorý som umiestnil do protilahlého rohu obchodu. Po ceste v obchode mal rôzne uličky, ktoré mohol navštíviť, ale ani jedna z nich nebola správnym riešením. Pri hľadaní najrýchlejšieho možného riešenia bolo potrebné vypočítať dané trasy a zvoliť tú správnu. To sa v tomto prípade podarilo a algoritmus túto trasu aj našiel.



Obrázok č.19 Najkratšia cesta v obchode

8.1.3 Scenár sklad

Úlohou posledného scenáru bol test navigácie robota v sklade, kde je veľa objektov, ktoré by mu mohli prekážať alebo môžu sťažiť jeho orientáciu. Scenár navigácie v sklade je najťažší pri navigácií v tomto prostredí. Úlohou robota bolo nájsť najrýchlejšiu a najefektívnejšiu cestu od vchodových dverí smerom až do miestnosti s toaletami a sprchami pre zamestnancov. V tomto prípade mu v ceste stálo mnoho prekážok. Orientácia v sklade je časovo najnáročnejšia úloha pre robota, pretože musel zvážiť všetky možné cesty, ktoré by sa dali použiť a z nich vybrať tu najrýchlejšiu.



Obrázok č.20 Najrýchlejšia cesta v sklade

8.1.4 Vyhodnotenie výsledkov

Každý algoritmus bol schopný nájsť najrýchlejšiu cestu v prostredí, do ktorého bol postavený. Pri použití robota v komplexnom prostredí bolo potrebné vyskúšať viac možností a viac navštívených časti. Na základe tohoto sa zvýšila aj časová náročnosť. Využívanie tejto navigácie v časovej tiesni odporúčam v prostrediach, ktoré sú priestranné a nie sú komplexné. Pokiaľ neexistuje žiadna časová tieseň, tak využitie navigácie robota pomocou algoritmu A* je dobrým riešením.

Tabuľka č.4 Vyhodnotenie časovej náročnosti

Scenáre	**Byt**	**Sklad**	**Obchod**
Časová náročnosť	A	C	B

Tabuľka č.5 Legenda

Popis náročností	
Nenáročné	A
Náročne	B
Veľmi náročné	C

ZÁVĚR

Cieľom tejto bakalárskej práce bolo vyhovtoviť rešerš k path planningu, jeho možných konceptov, algoritmov a aplikácií vo svete. Na základe získaných informácií bola vyhotovená praktická časť, ktorá je zameraná na praktickú funkčnosť algoritmov a ich využitie. Dôležitým aspektom pri návrhu a implementácií bolo stanoviť si postup, ktorý bolo potrebné dodržať. Bakalárska práca je rozdelená do 9 kapitol.

V úvode bakalárskej práce som priblížil robotiku, čo je to robotika a známe využitie robotiky. Popisuje základy problému obchodného cestujúceho a plánovania pohybu.

V ďalšej časti práce som popísal najznámejšie koncepty plánovania cesty robotického pohybu. V tejto časti som si vybral k popísaniu Configuration space, Free space, Target space a Obstacle space.

Tretia časť práce je zameraná na algoritmy, ktoré sa používajú pri plánovaní cesty, algoritmov je v dnešnej dobe mnoho, preto som si vybral Grid-based vyhľadávanie, Geometrický algoritmus, Artificial potential fields, Sampling-based algoritmus.

V poslednej teoretickej časti som potom pracoval s rôznými aplikáciami robotického pohybu v praxi. Existuje mnoho možností, kde sa dá robotika využiť, ale vybral som možnosti robotickej navigácie, automatizácie, možnosť šoférovania auta bez vodiča, teda driverless car a robotické operácie v medicíne.

Ôsma kapitola je zameraná na praktickú časť práce, kde som popísal použité technológie, v tomto prípade Python. Detailnejšie som popísal algoritmy, ktoré boli v praktickej časti použité. Pre prácu som si vybral algoritmy A*, Dijkstra, BFS a DFS. K práci s nimi som si vytvoril testovacie scenáre, ktoré nám pomohli v porovnaní práce algoritmov. Algoritmy som porovnal a výsledky vyhodnotil, v porovnaní algoritmov A* a Dijkstra som dosiahol výsledku, ktorý dokázal efektívnejšiu prácu Dijkstrovho algoritmu. V porovnaní algoritmov BFS a DFS nám výsledky ukazujú lepšiu prácu algoritmu BFS. Na záver som porovnal algoritmy Dijkstra a BFS, výsledné hodnoty ukazujú v prospech algoritmu Dijkstra.

Deviata a posledná kapitola je zameraná na porovnanie algoritmu A* v Mriežke plánovania optimálnej cesty. V mriežke som využil tri testovacie scenáre, na ktorých som algoritmus otestoval. Testovací scenár bytu fungoval správne, časová náročnosť k vyriešeniu nebola veľká. Pri testovaní pohybu robota v obchode všetko fungovalo správne, ale bolo potreba

viac času k vyriešeniu tohto scenára. Posledný scenár bol zameraný na pohyb robota v sklade, tento pohyb si vyžadoval veľa času, ale robot sa úspešne dostal z bodu A do bodu B.

SEZNAM POUŽITÉ LITERATURY

[1] *Builtin* [online]. 2022 [cit. 2022-05-18]. Dostupné z: https://builtin.com/robotics

[2] *Techtarget* [online]. 2020 [cit. 2022-05-17]. Dostupné z: https://www.techtarget.com/whatis/definition/traveling-salesman-problem

[3] *Blog.routific* [online]. 2020 [cit. 2022-05-17]. Dostupné z: https://blog.routific.com/travelling-salesman-problem

[4] *Robodk* [online]. 2019 [cit. 2022-05-17]. Dostupné z: https://robodk.com/blog/robot-motion-planning-made-easy/

[5] *ScienceDirect* [online]. 2021 [cit. 2022-05-17]. Dostupné z: https://www.sciencedirect.com/topics/engineering/path-planning

[6] *Control Automation* [online]. 2021 [cit. 2022-05-17]. Dostupné z: https://control.com/technical-articles/common-control-concepts-used-in-robot-motion-planning/

[7] CHOSET, Howie, Kevin M. LYNCH, Seth HUTCHINSON, George A. KANTOR, Wolfram BURGARD, Lydia E. KAVRAKI a Sebastian THRUN. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. 1.vydanie, Cambridge, Mass: MIT Press, 2005. ISBN 9780262255912.

[8] *Ieeexplore.ieee* [online]. 2007 [cit. 2022-05-17]. Dostupné z: https://ieeexplore.ieee.org/document/4209865

[9] *Control Automation* [online]. 2021 [cit. 2022-05-17]. Dostupné z: https://control.com/technical-articles/common-control-concepts-used-in-robot-motion-planning/

[10] *Control Automation* [online]. 2021 [cit. 2022-05-17]. Dostupné z: https://control.com/technical-articles/common-control-concepts-used-in-robot-motion-planning/

[11] *ScienceDirect* [online]. 2018 [cit. 2022-05-17]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2405896318311662

[12] YAP, Peter. *Grid-Based Path-Finding*. 1.vydanie, Berlín: Springer Berlin / Heidelberg, 2002. ISBN 978-3-540-43724-6.

[13] BHATTACHARYA, Priyadarshi a Marina L. GAVRILOVA. Geometric algorithms for clearance based optimal path computation. *ResearchGate* [online]. 2007 [cit. 2022-05-17]. Dostupné z: https://www.researchgate.net/publication/221589305_Geometric_algorithms_for_clearance_based_optimal_path_computation

[14] *Medium* [online]. 2021 [cit. 2022-05-17]. Dostupné z: https://medium.com/nerd-for-tech/local-path-planning-using-virtual-potential-field-in-python-ec0998f490af

[15] KARAMAN, Sertac a Emilio FRAZZOLI. Sampling-based algoritmus. *Ieeexplore* [online]. 2012 [cit. 2022-05-17]. Dostupné z: https://ieeexplore.ieee.org/document/6315419

[16] *SpringerLink* [online]. 2022 [cit. 2022-05-18]. Dostupné z: https://link.springer.com/article/10.1007/s10514-022-10039-8

[17] *Realtime Robotics* [online]. 2020 [cit. 2022-05-18]. Dostupné z: https://rtr.ai/transforming-industrial-automation-with-the-rapid-motion-planning-pt-1/

[18] FASSBENDER, Dennis a Hans-Joachim WUENSCHE. *Motion planning for autonomous vehicles in highly constrained urban environments* [online]. 2016 [cit. 2022-05-18]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7759692

[19] SU, Hang, Yingbai HU, Jiehao LI, et al. *Improving Motion Planning for Surgical Robot with Active Constraints* [online]. 2021 [cit. 2022-05-18]. Dostupné z: https://ieeexplore.ieee.org/document/9341302

[20] *Coursera* [online]. 2022 [cit. 2022-05-17]. Dostupné z: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python

[21] *GeeksforGeeks* [online]. 2022 [cit. 2022-05-17]. Dostupné z: https://www.geeksforgeeks.org/history-of-python/

[22] *Educative* [online]. 2022 [cit. 2022-05-17]. Dostupné z: https://www.educative.io/edpresso/what-is-the-a-star-algorithm

[23] *GeeksforGeeks* [online]. 2022 [cit. 2022-05-17]. Dostupné z: https://www.geeksforgeeks.org/a-search-algorithm/

[24] *Brilliant* [online]. 2016 [cit. 2022-05-17]. Dostupné z: https://brilliant.org/wiki/dijkstras-short-path-finder/

[25] *FreeCodeCamp* [online]. 2020 [cit. 2022-05-17]. Dostupné z: https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/

[26] *JavaTpoint* [online]. 2021 [cit. 2022-05-17]. Dostupné z: https://www.javatpoint.com/breadth-first-search-algorithm

[27] *Hackerearth* [online]. 2022 [cit. 2022-05-17]. Dostupné z: https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/

[28] Tech With Tim, (16. 7 2020). A* Pathfinding Visualization Tutorial – Python A* Path Finding Tutorial [Video]. YouTube. https://www.youtube.com/watch?v=JtiK0DOeI4A&ab_channel=TechWithTim

[29] Man1986, (2022). pyamaze [počítačový software]. https://github.com/MAN1986/pyamaze

[30] *ResearchGate* [online]. 2015 [cit. 2022-05-18]. Dostupné z: https://www.researchgate.net/figure/Motion-planning-in-workspaces-and-in-configuration-spaces-The-left-figure-shows_fig2_281389394

[31] *ResearchGate* [online]. 2016 [cit. 2022-05-18]. Dostupné z: https://www.researchgate.net/figure/The-goal-of-motion-planning-is-to-find-a-path-in-free-space-left-A-PRM-is-a-graph_fig1_304532889

[32] *ResearchGate* [online]. 2009 [cit. 2022-05-18]. Dostupné z: https://www.researchgate.net/figure/Schema-of-the-path-planning-approach-Shown-are-C-Space-obstacles-black-the-respective_fig4_221072885

[33] *GeeksforGeeks* [online]. 2009 [cit. 2022-05-18]. Dostupné z: https://www.geeksforgeeks.org/a-search-algorithm/

[34] *MDPI* [online]. Bazilej: 2018 [cit. 2022-05-18]. Dostupné z: https://www.mdpi.com/1999-4893/11/4/44/htm

[35] *Medium* [online]. 2021 [cit. 2022-05-17]. Dostupné z: https://medium.com/nerd-for-tech/local-path-planning-using-virtual-potential-field-in-python-ec0998f490af

[36] *Intechopen* [online]. 2016 [cit. 2022-05-18]. Dostupné z: https://www.intechopen.com/chapters/51781

[37] *Mdpi* [online]. 2020 [cit. 2022-05-18]. Dostupné z: https://www.mdpi.com/1424-8220/20/21/6341/htm

[38] *GeeksforGeeks* [online]. 2009 [cit. 2022-05-18]. Dostupné z: https://www.geeksforgeeks.org/a-search-algorithm/

[39] *Brilliant* [online]. 2016 [cit. 2022-05-17]. Dostupné z: https://brilliant.org/wiki/dijkstras-short-path-finder/

[40] *Guru99* [online]. 2022 [cit. 2022-05-18]. Dostupné z: https://www.guru99.com/breadth-first-search-bfs-graph-example.html

[41] *Hackerearth* [online]. 2022 [cit. 2022-05-17]. Dostupné z: https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

BFS Breadth First Search

DFS Depth First Search

DUC DARPA Urban Challenge

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PRÍLOH

PRÍLOHA P I: SCREENSHOTY TESTOV

- Testy.zip