EKONOMICKÁ UNIVERZITA V BRATISLAVE

FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103004/I/2022/421000214406

OPTIMALIZÁCIA NAVIGÁCIE AUTONÓMNEHO ROBOTA

Diplomová práca

2022 **Bc. Marcel Jaroš**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

OPTIMALIZÁCIA NAVIGÁCIE AUTONÓMNEHO ROBOTA

Diplomová práca

Študijný program: Informačný manažment

Študijný odbor: Informačný manažment

Školiace pracovisko: Katedra aplikovanej informatiky

Vedúci práce: doc., Ing. Jaroslav Kultan, PhD.

Bratislava 2022 **Bc. Marcel Jaroš**

Čestné vyhlásenie

Čestne vyhlasujem, že diplomovú prácu som vypracoval samostatne a že som uviedol všetku použitú literatúru.

Dátum: .....

Vlastnoručný podpis študenta

Poďakovanie

Týmto by som chcel poďakovať vedúcemu mojej diplomovej práce, doc., Ing. Jaroslavovi Kultanovi, PhD., za rady, čas a usmernenia pri jej tvorbe, taktiež chcem poďakovať svojej rodine a kamarátom za podporu a chcem poďakovať aj sebe za to, že si stanovujem nové výzvy, ktoré napĺňam a v živote sa tak posúvam tým správnym smerom dopredu.

ABSTRAKT

JAROŠ, Marcel: Optimalizácia navigácie autonómneho robota. Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra aplikovanej informatiky FHI. – Vedúci diplomovej práce: Ing. Jaroslav Kultan, PhD. – Bratislava: FHI EU, 2022, 62 s.

Diplomová práca nadväzuje na moju bakalársku prácu kde som ukázal, že je možné zostrojiť vlastného autonómneho robota, ktorý sa dokáže pomocou svojich senzorov pohybovať po miestnosti a vyhýbať sa prekážkam. Cieľom diplomovej práce je analyzovať existujúce navigačné systémy, ich prostredie a spôsob, ako v ňom hľadajú optimálnu cestu a na základe tejto analýzy potom navrhnúť vlastné metódy prechádzania prostredia. Súčasťou práce sú aj zdrojové kódy, podľa ktorých zariadenie pracuje. Práca je rozdelená do piatich hlavných kapitol. Obsahuje 57 obrázkov, 2 tabuľky a 5 príloh. Po úvode je v prvej kapitole popísaná základná problematika, prehľad a analýza existujúcich zariadení. Následne je uvedený cieľ práce. V tretej kapitole sú popísané nástroje, ktoré slúžia ako podklad na tvorbu vlastných metód. Štvrtá kapitola je zameraná na implementáciu metód a vyhodnotenie ich efektívnosti. Predposledná kapitola je zameraná na diskusiu a možnosti vylepšenia v budúcnosti. V záverečnej kapitole je zoskupená použitá literatúra a zdroje a taktiež zoznam obrázkov a ich zdroje. Výsledkom diplomovej práce je niekoľko rôznych otestovaných metód na optimálne prechádzanie priestoru, ktoré realizuje zariadenie.

Kľúčové slová: autonómny robot, orientácia v priestore, hľadanie cesty, optimalizácia, informačný systém, teória grafov

ABSTRACT

JAROŠ, Marcel: Autonomous robot navigation optimization, University of Economics in Bratislava. Faculty of Economic Informatics; Department of Applied Informatics FHI. – Diploma thesis supervisor: Ing. Jaroslav Kultan, PhD. – Bratislava: FHI EU, 2022, 62 pp.

The diploma thesis follows up on my bachelor's thesis where I showed that it is possible to build an autonomous robot that can use its sensors to move around the room and avoid obstacles. The diploma thesis aims to analyze the existing navigation systems, their environment, and the way they look for the optimal path in it and based on this analysis, to design custom methods for navigating the environment. The work also includes the source codes according to which the device works. The work is divided into five main chapters. It contains 57 images, 2 tables, and 5 appendices. After the introduction, the first chapter describes the basic problems, overview, and analysis of existing devices. Subsequently, the aim of the work is stated. The third chapter describes the tools that serve as a basis for creating custom methods. The fourth chapter focuses on the implementation of methods and evaluation of their effectiveness. The penultimate chapter focuses on the discussion and possibilities for improvement in the future. The final chapter groups the used literature and sources, as well as a list of pictures and their sources. The result of the diploma thesis is several different tested methods for optimal passage of space, which are realized by the device.

Keywords: autonomous robot, spatial orientation, pathfinding, optimization, information system, graph theory

Obsah

Úvod

Moderné technológie zožali za posledné roky rapídny pokrok a sú integrované do bežného ľudského života viac ako kedykoľvek predtým. Veľa z týchto technológií je navrhnutých tak, aby nám pomáhali v rôznych procesoch. Existujú algoritmy, ktoré nám na základe vstupných parametrov pomocou GPS vypočítajú optimálnu cestu do práce, alebo nám na základe nášho nákupného správania ukážu cielené reklamy na produkty, u ktorých je potenciál že si ich kúpime. Takmer každý podnik používa určitý systém na optimalizáciu výrobného procesu, teda optimálnu premenu vstupov na výstupy.

Optimalizácia je akt, proces alebo metodika tvorenia niečoho (napríklad optimalizácia dizajnu, systému alebo rozhodnutia), pričom dbáme na to, aby riešenie bolo čo najdokonalejšie, najfunkčnejšie alebo najefektívnejšie. Optimalizáciu môžeme pozorovať v systémoch zo súkromného života, ale aj v profesionálnom, respektíve podnikovom prostredí.

Táto diplomová práca nadväzuje na moju bakalársku prácu s názvom Autonómna kosačka, kde som ukázal, že je možné zostrojiť vlastného robota, ktorý pomocou senzorov dokáže vnímať vnemy z prostredia. Na základe týchto vnemov ďalej dokáže realizovať kroky na orientáciu a vyhýbanie sa prekážkam. Navigačné metódy v bakalárskej práci boli zamerané najmä na rozoznanie hraníc a prekážok. Jednou z hlavných myšlienok bolo ukázať, že je možné automatizovať rôzne opakujúce sa aktivity. Prirodzeným krokom vpred je optimalizácia týchto autonómnych činností, ktoré robot vykonáva pomocou nejakého algoritmu.

Na trhu existuje viacero zariadení, ktoré pracujú v rôznych prostrediach a majú rôzne ciele a každé z týchto zariadení do istej miery využíva nejaký algoritmus na optimalizáciu svojej práce. Existujú aj rôzne abstraktné systémy, ako napríklad počítačové hry, kde postavy hľadajú najkratšiu cestu ku svojmu cieľu.

Diplomová práca sa bude skladať najmä z analýzy rôznych orientačných systémov a algoritmov, návrhy vlastných riešení a nakoniec implementácia do môjho existujúceho systému s ďalšími vylepšeniami. Táto problematika je úzko spätá s oblasťami ako sieťová analýza a teória grafov, z ktorých taktiež budú čerpané poznatky. Cieľom práce bude zdokonalenie systému autonómnej kosačky, najmä implementácia nových algoritmov a spôsobov navigácie na optimálne prechádzanie prostredia a lepšiu orientáciu.

1 Analýza existujúcich systémov

Diplomová práca je zameraná na optimalizáciu navigácie autonómneho robota na kosenie trávy. Prvým krokom bude analyzovať existujúce systémy, od ich senzorov až po samotné správanie, teda algoritmus ich činnosti. Takéto algoritmy môžeme pozorovať pri robotoch, ktoré operujú v reálnom fyzickom prostredí, ale aj pri rôznych abstraktných systémoch alebo pri počítačových hrách.

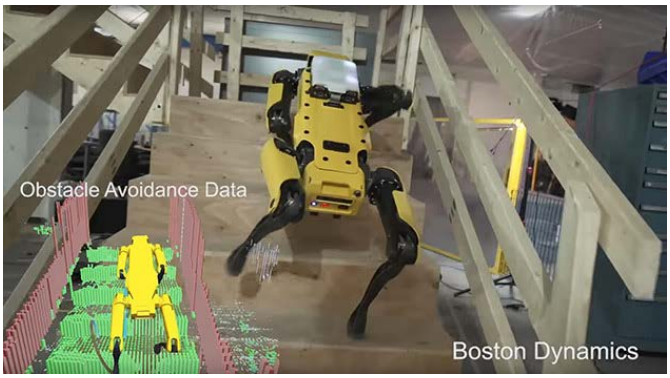
1.1 Navigačné systémy robotov v praxi

SPOT®

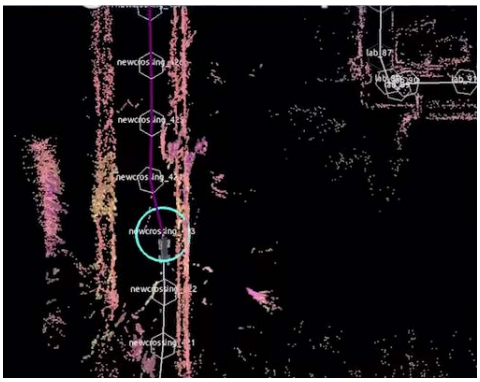
Spot je agilný mobilný robot, navrhnutý spoločnosťou Boston Dynamics, ktorý sa pohybuje v teréne s bezprecedentnou mobilitou, čo umožňuje automatizovať rutinné kontrolné úlohy a zachytávať dáta bezpečne, presne a často. Výsledkom sú bezpečnejšie, efektívnejšie a predvídateľnejšie procesy [2].

Obrázok 1 - *Spot*

Počiatočná kalibrácia spočíva v manuálnej navigácii, kedy si robot zostaví mapu priestoru za pomoci vizuálnych údajov z kamier namontovaných na jeho prednej, zadnej a bočnej strane. Zároveň aj zberá informácie o prekážkach vo svojej blízkosti, tak ako je na obrázkoch 2 a 3. Následne je Spot pripravený na autonómnu navigáciu cez analyzované prostredie pomocou 3D kamier a ďalších senzorov [3].

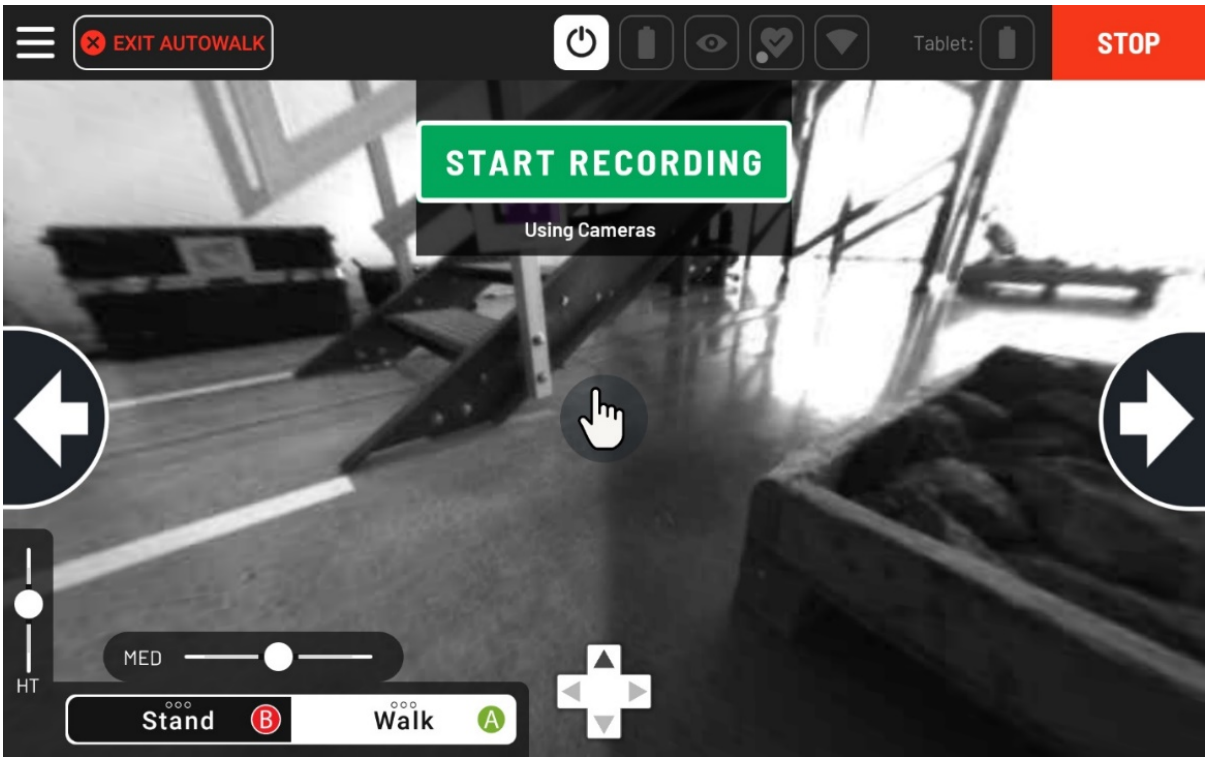


Obrázok 2 - *Spot* sníma prekážky v okolí



Obrázok 3 - *Spot* buduje mapu prostredia

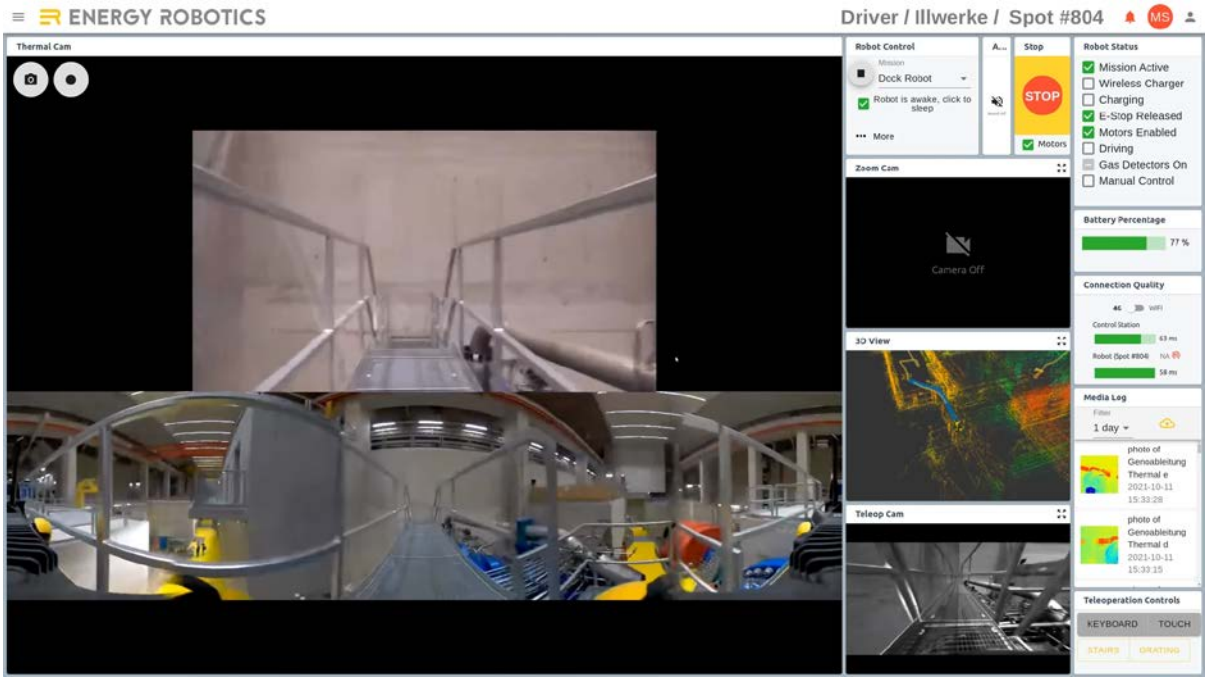
V novších verziách autonómneho robota *Spot* je možné nastaviť a ovládať autonómiu externe pomocou ovládača s aplikáciou Autowalk. V aplikácii je možné manuálne naplánovať cestu a následne robotovi nariadiť rôzne príkazy ako choď na určité miesto, vykonávaj pravidelné hliadky a monitoruj alebo zisti aktuálnu lokáciu na mape. Vývojári môžu tiež zaznamenávať, upravovať a nahrávať mapy do robota [4].



Obrázok 4 - Aplikácia Autowalk

Pre širšie možnosti ovládania je dôležité ponúknuť rozhranie pre počítačový kód. Tak, ako je možné automatizovať proces pomocou programu, tak isto je možné naprogramovať aj autonómneho robota. V kóde môže byť definovaná celá mapa, kde robot operuje a je možné mu nariadiť rôzne príkazy ako choď na určitý bod najkratšou možnou cestou a pomocou termokamery skontroluj zariadenie a ak je niečo mimo normy alebo cesta je blokovaná, pošli správu operátorovi. Ak je všetko v poriadku, robot pokračuje vo svojej misii, navštevuje

všetky miesta a body záujmu a na konci vytvorí report zo získaných dát. Takýto robot je nadmieru užitočný ak prácu vykonáva v prostredí, ktoré je ťažko prístupné alebo nebezpečné pre zdravie človeka. Takéto vysoko prispôsobiteľné naprogramovanie je možné realizovať za pomoci partnerských spoločností, ako je napríklad Energy Robotics, ktorá má aj svoj vlastný Softvér ukázaný na obrázku 5 [5].



Obrázok 5 - Softvér na pokročilé ovládanie robota

Robotické vysávače rady Xiaomi Mi Robot Vacuum Mop

Xiaomi Mi Robot Vacuum Mop 2 používa na orientáciu v priestore systém VSLAM (Visual Simultaneous Localization and Maping) [6]. Vysávač kombinuje základnú sadu kolíznych senzorov s hlavným vizuálnym senzorom, kamerou, ktorá je umiestnená v strede zariadenia smerom dohora. Sleduje tak strop miestnosti a niektoré vysoké prekážky ako napríklad gauče, stoly alebo postele. Optický systém dokáže identifikovať orientačné body ako aj posúdiť vzdialenosť medzi stenami. Nedokáže však pracovať v tmavej miestnosti. VSLAM tiež vypočítava relatívnu polohu vysávača v miestnosti v reálnom čase čo umožňuje robotovi vytvoriť mapu počas čistenia. Robotické vysávače, ktoré fungujú týmto spôsobom sa pohybujú po miestnosti s vyššou účinnosťou a systematicky čistia podlahu v logickom vzore. Nebudú strácať čas vysávaním oblastí miestnosti, o ktorých robot vie, že už prešiel. Vďaka tomu dokážu pokryť rovnakú oblasť za kratší čas a s lepším pokrytím ako robot založený len na fyzických senzoroch [7].

Xiaomi Mi Robot Vacuum Mop 2 Ultra je zatiaľ najinovatívnejší model autonómneho vysávača čínskej značky Xiaomi. Na orientáciu v priestore používa LIDAR technológiu

(Laser Imaging, Detection, And Ranging), ktorá pomocou rotujúcej veže vysiela laserové lúče, ktoré sa odrážajú od povrchov a prekážok. Pre každý laserový lúč vypočíta čas odrazu, a tak zistí vzdialenosť robota od prekážky. Lasery emituje konštantne a do všetkých smerov. Výhodou laseru je, že robot dokáže pracovať aj v tme. Ďalšou technológiu je S-crosss 3D, ktorá na princípe ToF (Time of Flight) rozoznáva trojrozmerné objekty ako káble, oblečenie, hračky alebo schody a efektívne sa im vyhýba [6].

Xiaomi Mi Robot Vacuum Mop 2 Lite využíva systém VINS (Visual-Inertial Navigation System), čo je štandardné riešenie lacnejších modelov [6]. Systém poskytuje odhad pohybu v 3D. Okrem navigácie autonómnych robotov je tento systém implementovaný aj napríklad v leteckej navigácii alebo v rozšírenej realite. Používa pri tom dva komponenty, monokulárnu kameru, ktorá sleduje strop miestnosti a relatívne lacné IMU (Inertial Measurement Unit) komponenty, v prípade tohto modelu je to obyčajný gyroskop, ktorý meria uhol sklonu. Systém má ale aj svoje nevýhody, a to absencia merania priamej vzdialenosti. Štartovací bod teda musí byť inicializovaný na jednoznačnom mieste, ktorý slúži ako „bod odrazu“. Dlhodobo sa presnosť aj tak znižuje v dôsledku vznikania malých odchýlok počas merania [8].

Obrázok 6 - Vysávač s laserovou vežou

Obrázok 7 - Vysávač s kamerou

Ocado Smart Platform

Na obrázku 8 je zobrazený sklad na balenie potravín, navrhnutý spoločnosťou *Ocado*. Ľudia si môžu objednať potraviny online a na miesto chodenia z domu hore-dole do kamennej predajne im tieto roboty zabalia viac ako milión potravín denne. Celý sklad bol navrhnutý čisto pre robotov s myšlienkou, čo najväčšej efektivity. Sklady dosahujú veľkosť až sedem futbalových štadiónov a množina robotov, označovaných ako roj, sa pohybujú rýchlosťou do 14 km/h a prechádzajú popri sebe iba vo vzdialenosti 5 milimetrov. Celý sklad je kovová mriežková konštrukcia s koľajami, kde sa v každej bunke nachádza nejaký produkt. Roj robotov sa pohybuje po x a y osi a pomocou drapáku zbierajú a zoskupujú produkty podľa

zákazníckych objednávok. Efektivita tohto riešenia spočíva aj v analýze objednaných produktov; najobjednávanejšie produkty sú umiestnené v bližšie k baliacemu oddeleniu, kde s robotmi spolupracujú aj ľudia a produkty zabalia do obalov a balíkov pripravených na expedíciu. Každý robot používa kombináciu senzorov a kamier na uskutočnenie svojej úlohy, no samotné plánovanie cesty je uskutočnené externe, centrálnou riadiacou stanicou [9].

Obrázok 8 - Mriežková platforma po ktorej jazdia roboty na zberanie potravín

Hai Robotics

HAI ROBOTICS poskytuje komplexné riešenie automatizácie skladov, ktoré zahŕňa roboty *HAIPICK*, nabíjacie stanice, prispôsobiteľné úložné jednotky, pracovné stanice a softvérovú platformu *HAIQ* [10].

Obrázok 9 - Automatizovaný sklad kde pôsobia vysokozdvižné autonómne roboty

Robot *HAIPICK* manipuluje, autonómne naviguje, aktívne sa vyhýba prekážkam a navštevuje nabíjaciu stanicu. Precízny a stabilný robot nahradzuje opakujúce sa časovo náročné, ťažké manipulačné a skladovacie práce manuálne vykonávané robotníkmi. Jednotlivé roboty cestujú po dopredu definovaných cestách a uličkách, kde hľadajú jednotlivé položky, ktoré následne odovzdajú na niektorom z odberných miest. Platforma *HAIQ* slúži ako mozog celého roja robotov, dokáže presne vypočítať najefektívnejšiu prepravnú cestu a naplánovať triedenie na základe množstva nákladu v sklade, polohy robotov a očakávanej prichádzajúcej a odchádzajúcej premávky, pričom každý robot vykonáva úlohy v reálnom čase. Sleduje aj stav batérií [10].

Okibo

Okibo je plne autonómny robot na lepenie, omietanie a maľovanie stien, ktorý spolupracuje na stavbe spolu s robotníkmi. Pomocou svojho robotického ramena vykonáva prácu kvalitne a za zlomok času stráveného klasickými spôsobmi. Automatizáciou tohto procesu zabraňuje robot vystaveniu ľudských pracovníkov potenciálnym rizikám, ako je práca vo výškach alebo práca s toxickými materiálmi. Jeho konštrukcia je navrhnutá aby dokázal bezpečne prechádzať aj cez hrboľatý a drsný terén staveniska. Na začiatku práce svojimi senzormi naskenuje prostredie a vytvorí 3D model. Potom naplánuje cestu, ktorá môže, ale nemusí byť upravená podľa požiadaviek používateľa a samostatne začne vykonávať prácu. Pomocou 3D modelu sleduje robot svoj pokrok a taktiež ho dokáže porovnať s informačným modelom budovy [11].

Obrázok 10 - Autonómny robot na maľovanie steny

Informačný model budovy (Building Information Modeling, BIM), simuluje stavebný projekt vo virtuálnom prostredí. Pomocou technológie BIM sa digitálne vytvorí presný virtuálny model budovy, známy ako informačný model budovy. Po dokončení informačný model budovy obsahuje presnú geometriu a relevantné údaje. Podporuje taktiež procesy obstarávania, výroby a stavebných činností potrebných na realizáciu budovy. Po dokončení je možné tento model použiť na prevádzkové účely a údržbu [12].

Trombia Free

Trombia Free je prvý autonómny elektrický robot na čistenie a umývanie ulíc od fínskej technologickej spoločnosti Trombia Technologies. Spotrebuje len 15 % energie, ktorú vyžadujú bežné cestárske vozidlá a počas čistenia nevytvára emisie. Okrem toho spotrebuje len zlomok množstva vody potrebnej pri konvenčných čistiacich metódach. Je to tiež prvé zariadenie na čistenie ulíc na svete, ktoré je skonštruované tak, aby bolo plne autonómne prevádzkované v moderných inteligentných mestách a priemyselných destináciách. *Trombia Free* je vybavený technológiou strojového videnia a za pomoci LIDAR technológie filtruje vizuálny šum z prostredia a umožňuje presnú a bezpečnú lokalizáciu za každého počasia. [13].

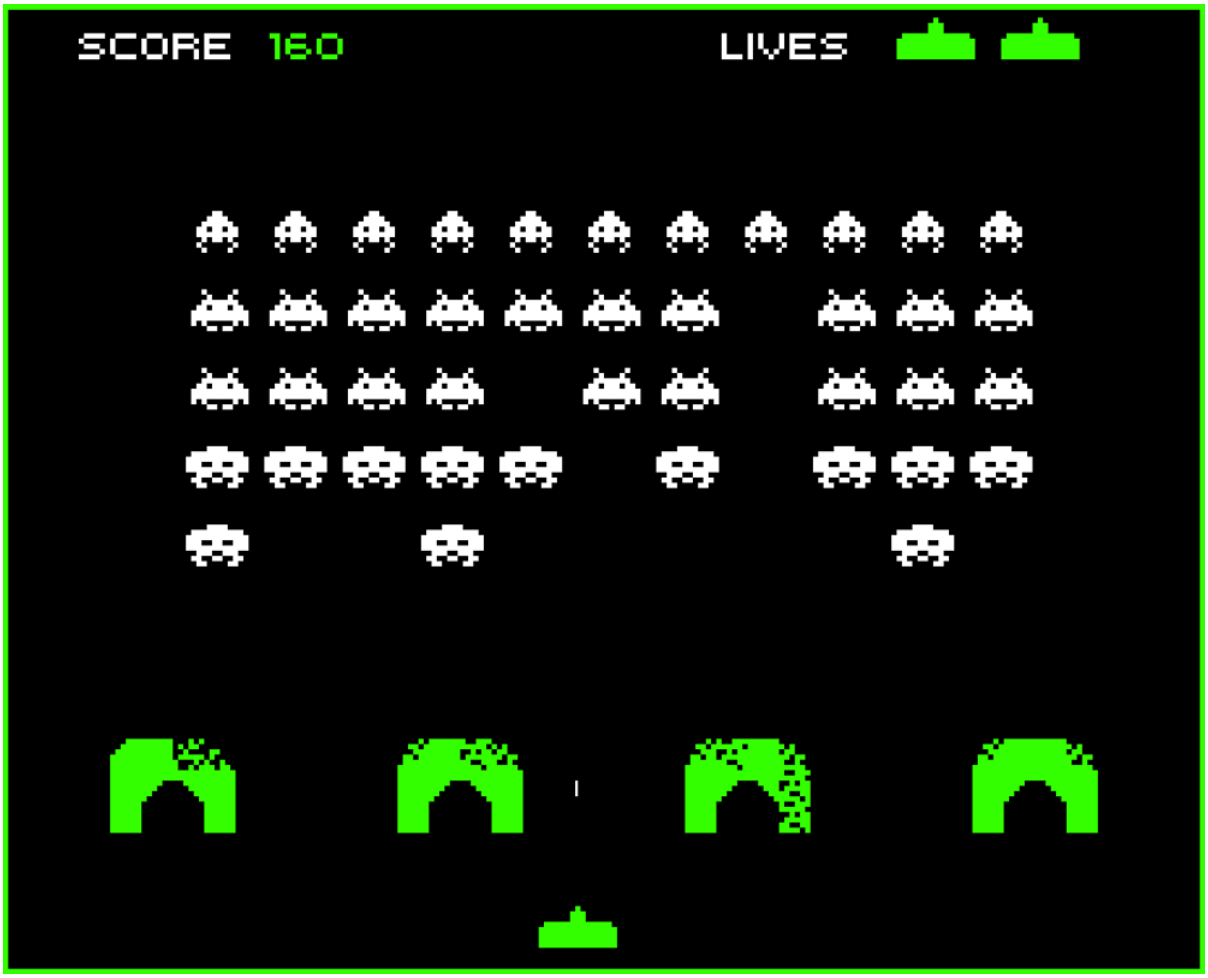
Obrázok 11 - Autonómny robot na čistenie a umývanie ciest

1.2 Pathfinding v počítačových hrách

Pathfinding (vyhľadávanie cesty) je základnou súčasťou mnohých dôležitých systémov v oblastiach GPS, videohier, robotiky, logistiky a simulácie davu a je možné ho implementovať v statickom a dynamickom prostredí a v prostredí, ktoré sa mení v reálnom čase. S nárastom a úspechom digitálnych hier za posledných niekoľko desaťročí sa algoritmy

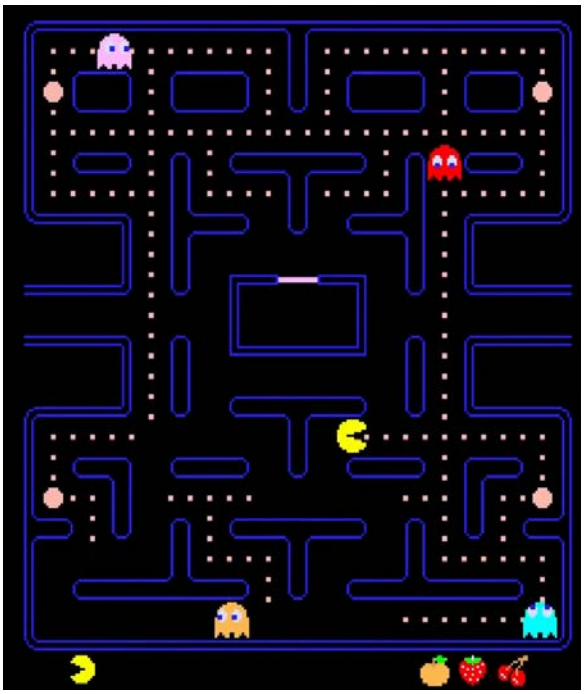
plánovania ciest stali dôležitým aspektom moderného vývoja hier pre všetky typy žánrov. Nehrateľné herné postavy (NPCs, Non Playable Characters) si musia nájsť cestu herným prostredím, aby dosiahli svoj cieľ. Úloha generovať uskutočniteľné a vierohodné cesty časovo a pamäťovo efektívnym spôsobom je však veľkou výzvou v tejto vznikajúcej a zaujímavej oblasti výskumu. Existuje mnoho digitálnych hier, či už s 2D alebo 3D herným priestorom, kde sú tieto techniky implementované. Je možné sa týmito technikami inšpirovať a implementovať ich do algoritmu autonómneho robota, ktorý zdieľa podobné prostredie v reálnom svete [14].

Zo začiatku bol pohyb v hrách triviálny a obmedzený, ako príklad je uvedená hra *Space Invaders* (Taito Corporation, 1978). Tu sa nepriatelia pohybujú vopred určeným a naskriptovaným spôsobom, ktorý nezahŕňa žiadne plánovanie cesty. Priechodný priestor hráča je iba 1D priamy segment v spodnej časti obrazovky. Na obrázku 12 je vidno, že ani hráč, ktorý ovláda hlavnú postavu nemá priestor na naplánovanie komplexnej cesty, musí ale počítať s obmedzením na ľavej a pravej strane obrazovky [15].

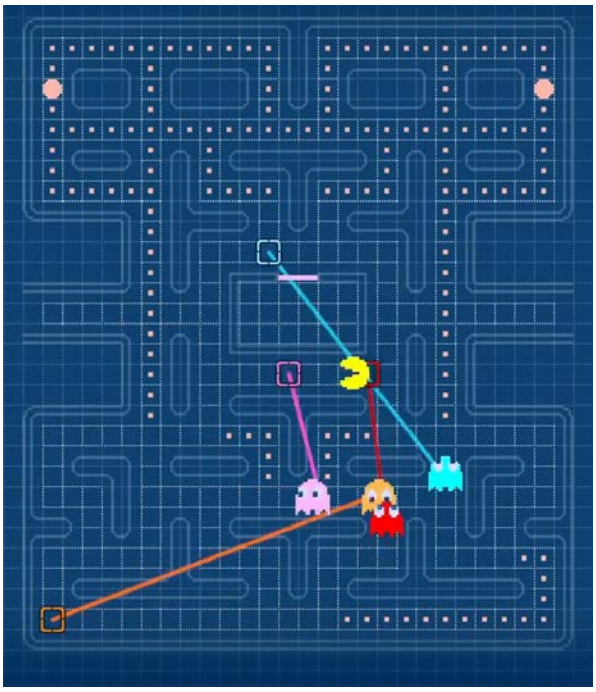


Obrázok 12 - Triviálny pohyb hráča a nepriateľov v *Space Invaders*

Ďalšia hra, ktorá zahŕňa plánovanie cesty je *Pac-Man* (Namco, 1980). Cieľom hráča je ovládať Pac-mana a vyzbierať všetky body na mape, pričom sa musí vyhýbať nepriateľom, duchom, ktorí používajú rôzne stratégie, aby hráča chytili. Postavy sa nachádzajú v 2D hernom svete s uličkami cez ktoré je možné prechádzať tak, ako je zobrazené na obrázku 13.



Obrázok 13 - Mapa v hre *Pac-man*



Obrázok 14 - Mriežková reprezentácia mapy

Herný svet môžeme ďalej rozložiť na jednotlivé dlaždice, ktoré sú súčasťou mriežky a niektoré sú prechodné a spolu tvoria uličky a niektoré sú neprechodné a tvoria spolu steny. Skutočnou revolúciou ale bola umelá inteligencia, ktorá poháňala Pac-manových nepriateľov, duchov. Pohyb duchov funguje na princípe mriežok cez ktoré sa nepriatelia a taktiež postava Pac-mana pohybujú. Červený duch „Blinky“, ktorý je najagresívnejší a najpredvídateľnejší nepriateľ prenasleduje Pac-mana priamo. Červený duch „Pinky“ sa snaží dostať pred Pac-mana prenasledovaním miesta, ktoré je vzdialené štyri mriežky od smeru Pac-mana. Tyrkysový duch „Inky“ prenasleduje špeciálne miesto, ktoré je možné nájsť na priamke, ktorá vychádza z Inkyho, pretne miesto kde je Pac-man a ešte je predĺžená o rovnakú vzdialenosť. Posledný, hanblivý oranžový duch „Clide“ prenasleduje Pac-mana iba do ohraničenia ôsmych mriežok okolo Pac-mana. Keď sa ale dostane do tejto zóny, utečie do najbližšieho rohu. Do týchto konkrétnych bodov sa duchovia dostanú za pomoci algoritmov na hľadanie cesty, ktoré budem rozoberať v neskorších kapitolách. Zároveň sú duchovia v troch rôznych stavoch:

- režim prenasledovania, kedy prenasledujú Pac-mana, ako je popísané vyššie,
- režim rozptýlenia kedy sú duchovia na ústupe a idú do rohu obrazovky z dôvodu, aby hráčovi dali chvíľku na oddýchnutie,

- vystrašený režim, ktorý nastáva v momente kedy Pac-man obdrží schopnosť zjesť duchov, ktorí sa boja a tým pádom utekajú [16].

Tieto 2D herné svety sa skladajú z priamočiarych dlaždíc, ktoré sú buď priechodné, alebo blokované. Jednoduchý mriežkový prístup založený na dlaždiciach je teda dostatočným spôsobom, ako reprezentovať celý priechodný priestor.

Na trhu sú dostupné aj moderné 3D hry, ktoré simulujú otvorený svet. Príkladom takejto hry je *Minecraft* (Mojang, 2009). Tieto typy hier obsahujú vysoko dynamické viacvrstvové 3D prostredia s rôznymi typmi terénu. Pre herné postavy je nielen dôležité naplánovať cestu bez kolízií, ale aj využívať oblasti, kde je možné liezť alebo preskakovať medzery, aby sa dostali do ťažko dostupných oblastí v hernom svete. Reprezentácia takéhoto priestoru a algoritmy na hľadanie cesty sú výrazne zložitejšie ako pri v 2D hrách [15].

Obrázok 15 - V hre *Minecraft* nepriatelia preskakujú prekážky, aby sa dostali k hráčovi

Napriek tomu, že vývoj v tejto oblasti za posledné dve desaťročia zlepšilo presnosť a efektivitu techník hľadania cesty, pathfinding je stále obšírne študovaný v rôznych výskumoch a aplikáciách [14].

Existujú rôzne variácie problému hľadania cesty, ako je napríklad multi agentové vyhľadávanie cesty, dynamické zmeny v prostredí alebo čiastočne pozorovateľné prostredie, externé faktory, ktoré ovplyvňujú robota a iné. Každý z týchto problémov má rôzne aplikácie v rôznych oblastiach [14]. Teória rôznych typov prostredí, bola popísaná v predošlej práci. Poznatky z nej budú využité v ďalšej kapitole.

1.3 Analýza prostredia

Okrem analýzy správania autonómneho robota je nezanedbateľným faktorom aj analýza samotného prostredia, v ktorom svoju činnosť vykonáva. V predošlej práci bolo popísaných niekoľko základných rozdelení prostredia a bola zostavená tabuľka s rôznymi príkladmi prostredí, v ktorých operovali robotický agenti. Tieto poznatky boli aplikované na analyzované roboty z predošlej kapitoly a sú zobrazené v tabuľke 1. Azda najzaujímavejším faktorom je pozorovateľnosť prostredia. Prostredie je skutočne plne pozorovateľné, ak senzory zistia všetky aspekty, ktoré sú relevantné pre výber akcie [1].

Tabuľka 1 - Typ prostredia a typ agenta

Typ agenta/ prostredia	Pozorova-teľnosť	Determini-zmus	Epizodic-kosť	Dynamic-kosť	Diskrét-nosť	Single/multi agent
Spot	Čiastočne pozorovateľné	Stochastické	Epizodické	Dynamické	Kontinuálne	Single
Vysávače Xiaomi	Čiastočne pozorovateľné	Deterministické	Sekvenčné	Dynamické	Diskrétne	Single
Ocado Smart Platform	Pozorovateľné	Stochastické	Epizodické	Statické	Diskrétne	Multi
Okibo	Čiastočne pozorovateľne	Deterministické	Epizodické	Statické	Diskrétne	Single
Hai Robotics	Pozorovateľné	Stochastické	Epizodické	Statické	Diskrétne	Multi
Trombia Free	Čiastočne pozorovateľne	Stochastické	Sekvenčné	Dynamické	Diskrétne	Single

Kamerové systémy vysávačov sa blížia k plnej pozorovateľnosti, no prostredie zostáva stále priveľmi komplexné. S týmto problémom výrazne pomáha počiatočná kalibrácia, ktorej cieľom je zostaviť mapu prostredia. Je to okrem senzorov ďalšia pomôcka pomocou ktorej sú definované prekážky a algoritmus ju vie využiť napríklad na naplánovanie najkratšej cesty z jedného bodu do druhého.

Zaujímavým príkladom je mriežková platforma pre roboty na zbieranie potravín. Toto prostredie bolo špeciálne navrhnuté iba pre roboty, takže všetky externé faktory, ktoré by mohli zasahovať do práce robotov sú eliminované a prostredie je tak plne pozorovateľné. V podobnom mriežkovom prostredí hľadajú cestu aj postavy z hry *Pac-man*. Cesta jedného robota je ovplyvnená pozíciou iných robotov a cesta postáv je ovplyvnená pozíciou Pac-mana.

Úplne na druhom konci spektra je prostredie v ktorom vykonáva prácu *Spot*, ktoré je prispôsobené pre ľudí. *Spot* je tým pádom navrhnutý aby dokázal pracovať medzi ľuďmi, respektíve na miesto ľudí. Sú tu prekážky ako dvere, schody alebo obrubníky cez ktoré sa *Spot* vie dostať. Prostredie sa podobá na 3D prostredie v počítačových hrách, ktoré práve simuluje reálny svet.

Prostredie autonómnych kosačiek má najviac príbuzné prostrediu autonómnych vysávačov. Kosačky nemusia otvárať dvere alebo jazdiť po schodoch no zároveň prostredie nie je jedna veľká platforma bez prekážok. Sú tu pevne dané hranice, v nejakých prípadoch dynamicky meniace sa prostredie a malé prekážky, ktorým sa kosačka musí vyhnúť. Na rozdiel od iných autonómnych zariadení, plánovanie cesty má väčší dôraz na samotné cesty ktorými zariadenie prechádza, na miesto jednotlivých bodov v prostredí. Ideálne by mali tieto zariadenia prejsť všetky cesty, aby bola pokrytá – a tým pádom opracovaná celá plocha. Na podobnom princípe funguje aj umývač a čistič ciest *Trombia Free*, ktorý by mal prejsť každú jednu cestu v meste, ale optimálnou trasou alebo malovač stien *Okibo*, ktorý chce pomaľovať celú stenu.

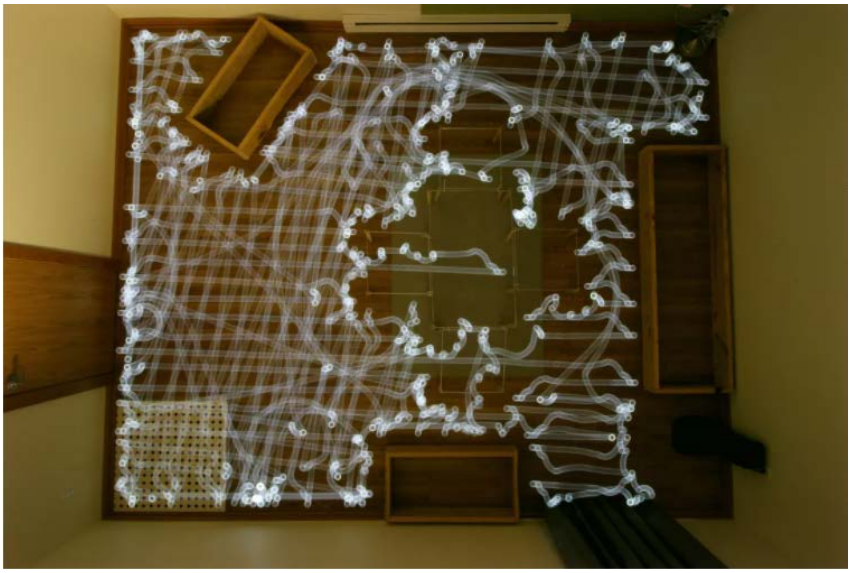
1.4 Analýza navigácie a technických prostriedkov

V mojej predošlej práci som analyzoval niekoľko autonómnych zariadení, predovšetkým autonómne kosačky. Predmetom analýzy boli najmä technické prostriedky ako snímače a senzory, pohonné jednotky a konštrukcia zariadení. V krátkosti boli analyzované aj ich navigačné systémy, ktoré boli zväčša triviálne. Zistil som, že roboty cestujú až pokým nenarazia na nejakú prekážku, či už svojim nárazníkom, alebo zistia jej prítomnosť inými senzormi. Narážajú do vecí a náhodne sa otáčajú po miestnosti [1]. Technológie postupujú rapídne dopredu a v súčasnosti je štandardom, že autonómne roboty obsahujú robustnejšie navigačné systémy. Spôsob navigácie respektíve algoritmus činnosti a typ prostredia v ktorom robot prácu vykonáva, je úzko spätý s technickými prostriedkami a senzormi robotov.

Niektoré lacnejšie alebo staršie systémy, ktoré boli analyzované aj v bakalárskej práci využívajú relatívne jednoduchú technológiu na orientovanie pomocou IMU (Inertial Measurement Unit) modulov. Ako píše M. Soták [17]: „Inerciálna navigácia je navigácia založená na nepretržitom vyhodnocovaní polohy navigovaného objektu s využitím senzorov citlivých na pohyb, tzn. gyroskopov a akcelerometrov, ktoré sú považované za primárne inerciálne senzory, alebo iných senzorov, umiestnených na navigovanom objekte. Pomocou navigačného počítača a údajov zo senzorov je nepretržite určovaná poloha, orientácia, smer a rýchlosť pohybu bez externých zdrojov informácií o pohybe. Aktuálna poloha objektu je

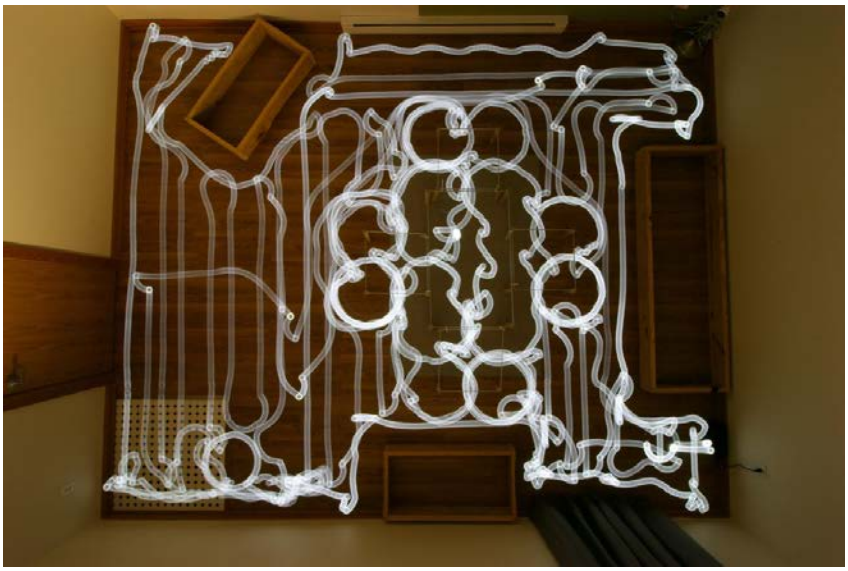
vyhodnocovaná na základe znalosti počiatočnej polohy a následného kontinuálneho merania zrýchlenia a smeru pohybu v referenčnej sústave“.

Väčšina robustnejších autonómnych vysávačov a iných zariadení využíva buď navigačný systém LIDAR alebo VSLAM, ktoré sa za pomoci laserov alebo kamery orientujú v priestore, vytvárajú mapu prostredia a prechádzajú do určitej miery optimalizovanou trasou. VSLAM vyžaduje osvetlenú miestnosť, keďže v tme by kamera nič nevidela. Na obrázku 16 sa nachádza miestnosť, v ktorej vykonal prácu *iRobot Roomba i7 Plus* a ukazuje logickejšiu a dôkladnejšiu navigačnú cestu vďaka svojej optickej VSLAM technológii [7].



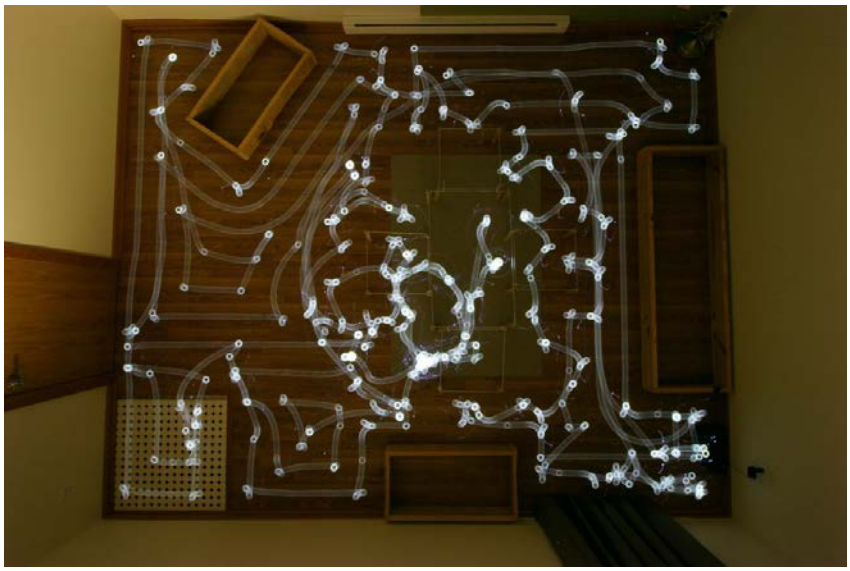
Obrázok 16 - Cesta robota s VSLAM technológiou

LIDAR zvykne byť o mieru presnejší vďaka mape prostredia, ktorú vytvára počas práce a pomocou laserov majú zariadenia jasnejší prehľad o prekážkach na úrovni podlahy. Oba systémy ponúkajú možnosť zakázanej zóny. Po vytvorení mapy je pri oboch systémoch možné v aplikácii nastaviť oblasti do ktorých robot nevstúpi. Aj v pomenovaní, aj v cenovom rozdiely modelov *Xiaomi* robotov je implikované, že systém LIDAR je pokročilejší, avšak aj tento systém má svoje nevýhody. Roboty na báze laserov majú problém s „falošnými“ bariérami, ako sú záclony a závesy, o ktorých si myslia že sú reálne prekážky, zatiaľ čo iné roboty by cez tieto prekážky bezproblémovo prešli [18]. LIDAR (Laser Imaging, Detection, And Ranging) je podobný technológii RADAR (Radio Detection And Ranging) a SONAR (Sound Navigation And Ranging), LIDAR ale využíva laser, zatiaľ čo RADAR využíva na vypočítanie času odrazu elektromagnetické vlny a SONAR zvukové vlny. Na obrázku 17 je zobrazená cesta autonómneho vysávača *Neato Botvac D6*, ktorý pomocou systému LIDAR systematicky cestu optimalizoval, aby bola práca vykonaná v čo najkratšom čase [7].



Obrázok 17 - Cesta robota s LIDAR technológiou

Riešením daných problémov by mohlo byť zlúčenie týchto dvoch navigačných systémov. Pokúsil sa o to *Electrolux Pure i9*, ktorý ale prekvapivo vynechal oblasti v testovacej miestnosti, ako je ukázané na obrázku 18 [7].



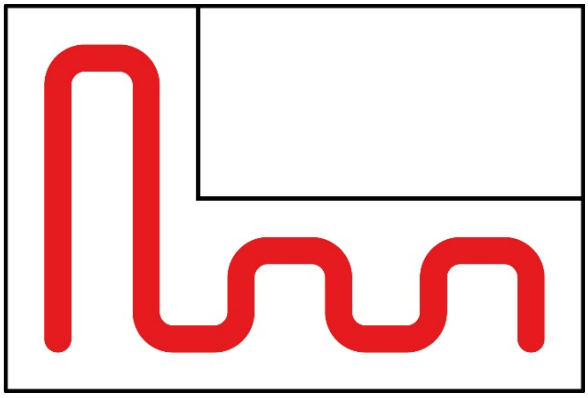
Obrázok 18 - Cesta robota s hybridným systémom navigácie

Ďalšou úrovňou je kombinácia tohto hybridného systému s umelou inteligenciou, ktorú využíva autonómny robot *Ecovacs Deebot Ozmo T8 AIVI* (Artificial Intelligence and Visual Interpretation). Disponuje aj laserovou vežou, aj prednou kamerou ktorá pomocou umelej inteligencie rozoznáva aj tie najmenšie prekážky, ako napríklad ponožky, hračky alebo káble, ktoré by obyčajný senzor nezachytil [19]. Tým pádom kamera spárená s umelou inteligenciou bude budúcnosťou autonómnych robotov, pretože kamery dokážu prijať najviac informácií z prostredia spolu s výkonnými procesormi, ktoré tieto dáta dokážu užitočne interpretovať.

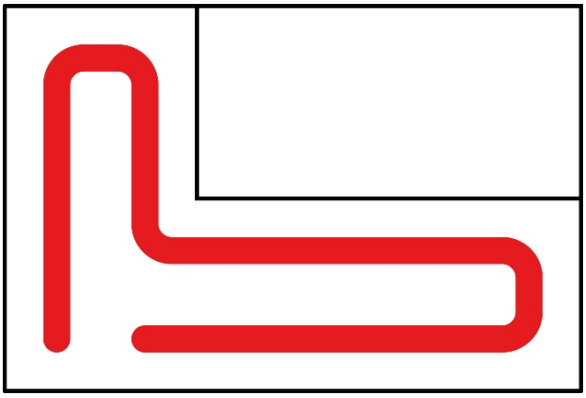
Prejdené cesty troch rozdielnych zariadení taktiež ukazujú, že podľa toho v akej časti prostredia sa nachádzajú, vykonávajú rozdielne spôsoby prejdenia plochy. Pri podlhovastých častiach miestnosti prechádzajú hore-dole pozdĺž dlhšej hrany alebo špirálovým spôsobom. V komplexnejších častiach izby sa snažia prejsť okolo každej prekážky a zachytiť čo najväčšiu plochu.

Tieto navigačné systémy pôsobili v prostredí single agent čo znamená, že prácu vykonával iba jeden robot. Súčasťou analýzy boli aj roboty, ktoré patrili do robotického roja. Toto prostredie označujeme ako multi agent, teda viacero robotov pracovalo spolu na dosiahnutie určitého cieľa [1]. Jednotlivé zberné boxy na platforme *Ocado* a vysokozdvižné prepravné roboty *Haipick* majú svoje vlastné senzory, no ich algoritmus, respektíve ich programové správanie je centralizované v riadiacej jednotke, ktorá každý jeden robot monitoruje, plánuje mu cestu a určuje iné úlohy. Aj keď sa v praxi stretávame s kosačkami, ktoré pracujú samostatne na jednom malom pozemku, tieto systémy by mohli byť inšpiráciou pre veľké robotické projekty v rámci väčšieho pozemku, kde by mohlo spolupracovať viacero robotických kosačiek alebo robotov na hnojenie, zavlažovanie, zber vypestovaných surovín a rýľovanie.

Autonómny robot *Okibo* na maľovanie stien má jedno výrazné obmedzenie; prácu vykonáva epizodicky, vždy len v dosahu svojho robotického ramena. Následne sa posunie na ďalšiu oblasť steny, kde pokračuje v maľovaní, výhradne vo vzore z hora dole a opačne, tak ako je znázornené na obrázku 19. V sekvenčnom prostredí, v ktorom sa nachádzajú autonómne vysávače a kosačky (súčasné rozhodnutie má vplyv na všetky budúce rozhodnutia), dbáme na celkovú optimalizáciu trasy, v ktorej uprednostňujeme priame jednotkové cesty a trestáme časté otáčanie, tak ako je znázornené na obrázku 20 (červená čiara znázorňuje cestu, nie opracovanú plochu).



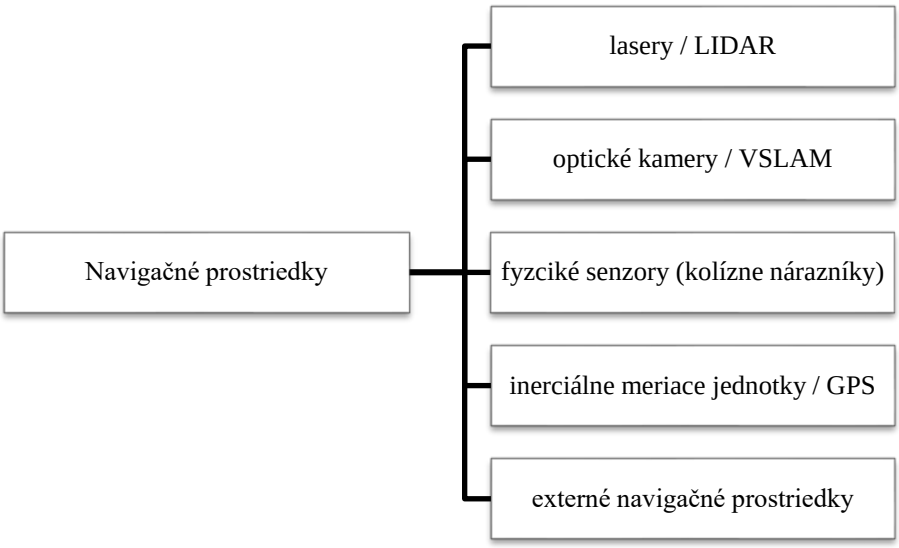
Obrázok 19 - Časté a neefektívne otáčanie



Obrázok 20 - Systematická a efektívnejšia cesta

Silnou stránkou robota *Spot* je, že okrem svojich bežných navigačných senzorov môže byť rozšírený o nové moduly priamo od výrobcu. Jedným z takýchto modulov *Spot Arm;* robotická ruka, ktorá uchopí, zdvíha, prenáša, umiestňuje a ťahá rôzne položky, točí ventily, preklopí páky, otvára dvere a má zabudované led svetlo s kamerou. Ďalší z modulov je *Spot Cam+IR;* rotačná kamera s 360° zorným poľom pre úplnú prehľadnosť situácie pri diaľkovom ovládaní robota. Taktiež obsahuje termálnu kameru, mikrofón a reproduktory. Pomocou modulu *Spot Eap* môže využiť technológiu LIDAR s ktorou dokáže zostrojiť väčšiu a presnejšiu mapu prostredia. V neposlednom rade modul *Spot Core* s prídavným procesorom umožňuje vývojárom implementovať ich vlastný softvér na riadenie robota pomocou nástroja SDK (Software Development Kit) [20]. Jeden z týchto softvérov bol krátko popísaný v kapitole 1.1.

Podobne ako kosačky a vysávače, čistič ulíc je orientovaný na cesty (hrany) a nie miesta (vrcholy), je teda viac menej jedno, ktoré vrcholy navštívi a v akom poradí, dôležité je aby prešiel všetkými cestami. Činnosť robota *Trombia Free* pripomína problém čínskeho poštára, ktorý taktiež chce prejsť všetky cesty v meste. Avšak, zatiaľ čo čínsky poštár chce každou cestou prejsť ideálne jeden krát, čistiť ulíc musí cez nejaké ulice prejsť viac krát, z dôvodu viacprúdových ciest. Na obrázku 21 sú zhrnuté navigačné prostriedky.

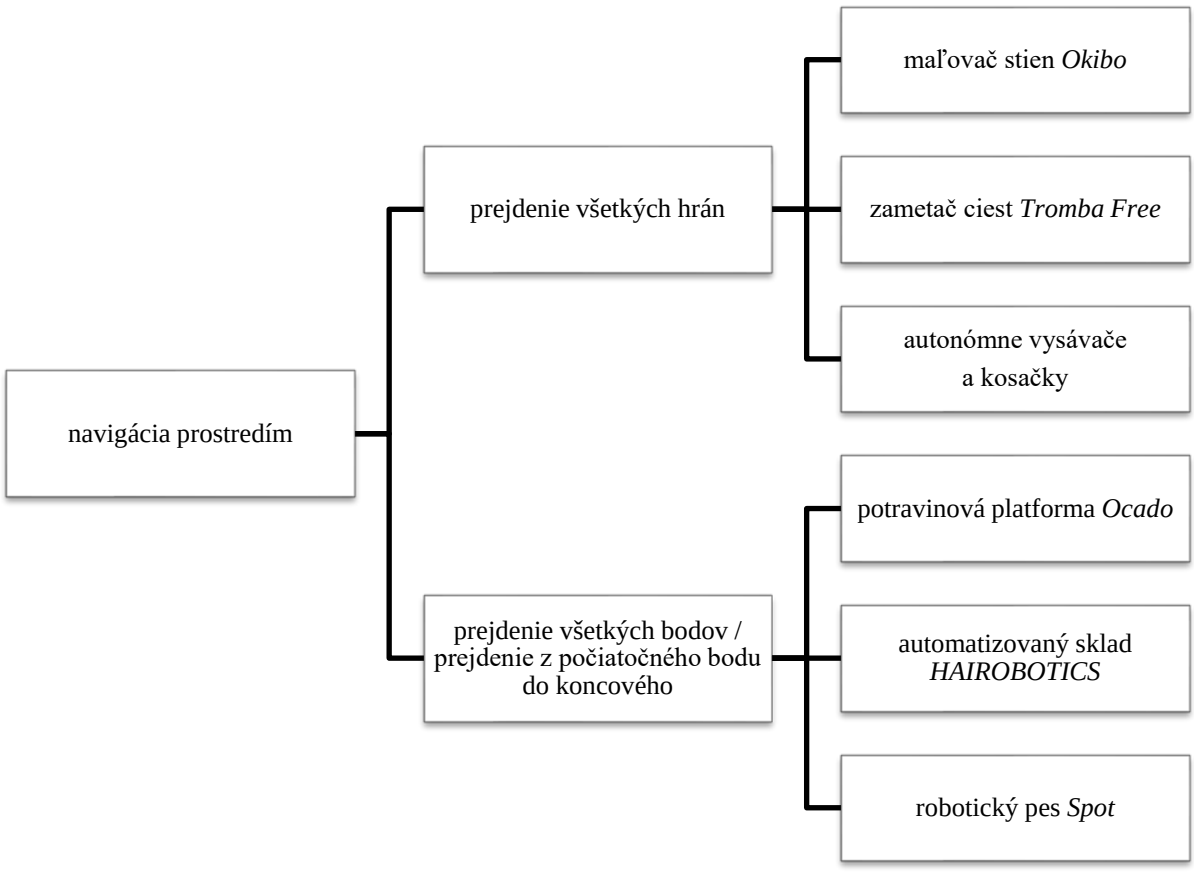


Obrázok 21 - Navigačné prostriedky

1.5 Zhodnotenie analýzy

Každé jedno z analyzovaných zariadení je svojimi technickými prostriedkami, konštrukciou a senzormi prispôsobené prostrediu v ktorom vykonáva prácu, pričom hľadá optimalizovanú trasu, aby naplnilo svoj cieľ. V rámci svojho prostredia majú skoro všetky systémy určité body záujmu, ktoré navštevujú. Sú to nabíjacie stanice, výdajné miesta alebo

vo všeobecnosti určité definované miesta, kde vykonávajú časť svojej práce. Celkové prostredie môže byť neznáme; počas prvého spustenia robot oskenuje celé prostredie a zostrojí mapu, ktorú si uloží do svojej pamäte a využije ju pri neskorších, opakovaných spusteniach. Počiatočná kalibrácia vo forme špeciálneho postupu pri prvom spustení výrazne pomôže algoritmu na optimálne prechádzanie prostredia. Algoritmy používané počas prvotnej kalibrácie sú komplexné a väčšinou sú použité iba raz za celú životnosť zariadenia. Prostredie môže byť aj dopredu známe, uložené v pamäti robota, alebo v centralizovanej riadiacej stanici s ktorou robot komunikuje. V tomto prípade je optimálna navigácia možná instantne podľa rôznych požiadaviek a cieľov. Komplexnosť prostredí sa od systému k systému mení a taktiež to platí aj pre reprezentáciu tohto prostredia v pamäti. Pri niektorých systémoch je očividné, že mriežkový prístup založený z priechodných alebo nepriechodných dlaždíc resp. orientovaných alebo neorientovaných ciest je postačujúci, pri iných systémoch je reprezentácia výrazne zložitejšia. Charakter činnosti robotov taktiež určuje, akou metódou sú prechádzané prostredia. Roboty buď navštevujú všetky body v priestore alebo sa chcú dostať z jedného bodu do iného bodu alebo chcú prejsť všetky cesty v rámci prostredia optimalizovanou trasou. Toto rozdelenie je ilustrované na obrázku 22.



Obrázok 22 - Rozdelenie typov prechádzania prostredím

2 Cieľ a metodika práce

Cieľom práce je navrhnúť, implementovať a otestovať rôzne navigačné algoritmy a spôsoby orientácie pre autonómnu kosačku na základe analýzy existujúcich navigačných systémov v priestore a algoritmov na nájdenie najkratšej alebo optimálnej cesty v grafe. Na dosiahnutie daného cieľa je potrebné vyriešiť nasledujúce úlohy:

1. ***Analyzovať*** existujúce algoritmy, ktoré sú používané v rôznych systémoch na nájdenie najkratšej cesty z počiatočného bodu do koncového, respektíve analyzovať spôsoby navigácie za účelom dosiahnutia zvoleného cieľa. Taktiež treba analyzovať poznatky z teórie grafov a algoritmy, ktoré umožňujú prejsť cez všetky hrany v grafe optimálnou cestou.
2. ***Navrhnúť*** niekoľko pokročilých spôsobov orientácie na základe analýzy. Primárnou inšpiráciou budú práve analyzované algoritmy, ktoré autonómiu posunú na vyššiu úroveň. Budú navrhnuté aj iné techniky, ktoré budú vyžadovať rôzne stupne prípravy zariadenia a interaktivity od užívateľa pred prácou zariadenia. Súčasťou návrhu budú aj spôsoby, ako je možné reprezentovať prostredie, ako ho uložiť v pamäti a ako v rámci neho uskutočniť algoritmus.
3. ***Implementovať*** algoritmy do systému automatického robota. Každý algoritmus je postupnosť krokov, ktorá sa dá vyjadriť aj slovne, ale aj pomocou nejakého programovacieho jazyka. Robot je riadený programovateľným mikropočítačom Arduino, kde v rámci vývojového prostredia bude napísaný kód v jazyku podobnom jazyku C/C++. Implementáciu bude možné zrealizovať pomocou rôznych popísaných programových nástrojov.
4. ***Otestovať*** každý algoritmus v kontrolovanom priestore. Výstupom bude porovnávacia tabuľka, kde budú ohodnotené rôzne kritériá efektivity daného spôsobu orientácie.
5. ***Vyhodnotiť*** algoritmy na orientovanie na základe intenzity splnenia kritérií. Výstupom bude slovné zhodnotenie a rebríček zoradený od najúspešnejších spôsobov orientácie.

Výsledkom budú algoritmy testované na robotovi v kontrolovanom priestore, navrhnuté na základe poznatkov z teórie grafov a na základe analýzy existujúcich navigačných systémov. Bude poskytnutý zdrojový kód, ktorý môže slúžiť ako inšpirácia na aplikovanie do iných robotických systémov.

3 Metódy a nástroje spracovania

V nasledujúcich kapitolách sú uvedené metódy, ktorými je možné optimalizovať hľadanie cesty pomocou poznatkov získaných z analýzy existujúcich zariadení a z teórie grafov.

Hľadanie cesty sa vo všeobecnosti skladá z dvoch hlavných krokov, ktorými sú generovanie grafu a realizácia algoritmu na hľadanie cesty. Generovanie grafu môže prebiehať rôznymi technikami počas počiatočnej kalibrácie, kedy zariadenie generuje mapu prostredia a v nej nachádza a vyberá rôzne body záujmu, ktoré sú v grafe reprezentované vrcholmi. Medzi tieto techniky patrí napríklad skenovanie prostredia pomocou laserov alebo manuálne prechádzanie pomocou ovládača. Existuje možnosť aj dopredu definovať prostredie čo zaberie čas navyše, ale počiatočná kalibrácia je v tomto prípade nepotrebná.

Budú popísané algoritmy na prejdenie všetkých hrán, ktoré sú pre kosačku viac intuitívne, keďže chceme prejsť celé prostredie, respektíve cestu ktorá prostredie celé pokryje. Takisto budú popísané aj algoritmy na prejdenie všetkých vrcholov a algoritmy na nájdenie najkratšej cesty, ktoré môžu byť využité ak sa napríklad zariadenie potrebuje dostať z hocijakého bodu do nabíjacej stanice, alebo navštíviť miesto, kde bola v minulosti nejaká prekážka. Po objasnení problematiky teórie grafov bude popísané, ako je možné grafy reprezentovať tak, aby s nimi bolo možné pracovať v počítačovom kóde. Ak je problém príliš zložitý, algoritmus niekedy môže byť sprevádzaný heuristickými funkciami. Heuristika je technika na vyriešenie problému rýchlejšie ako klasické metódy alebo na nájdenie približného riešenia, keď to klasické metódy nedokážu. Ide o kompromis, pretože optimálnosť, úplnosť alebo presnosť často vymieňame za rýchlosť.

Počas definovania prostredia je potrebné tieto informácie vhodným spôsobom uchovať v pamäti počítača. Jednou z metód reprezentácie grafu v teórii grafov je matica susednosti, ktorú môžeme v pamäti uložiť ako obyčajné dvojrozmerné pole. Táto metóda ale nemusí byť vždy optimálna, preto budú popísané aj iné metódy špecifické pre počítačový kód ako listy, spájané zoznamy a iné.

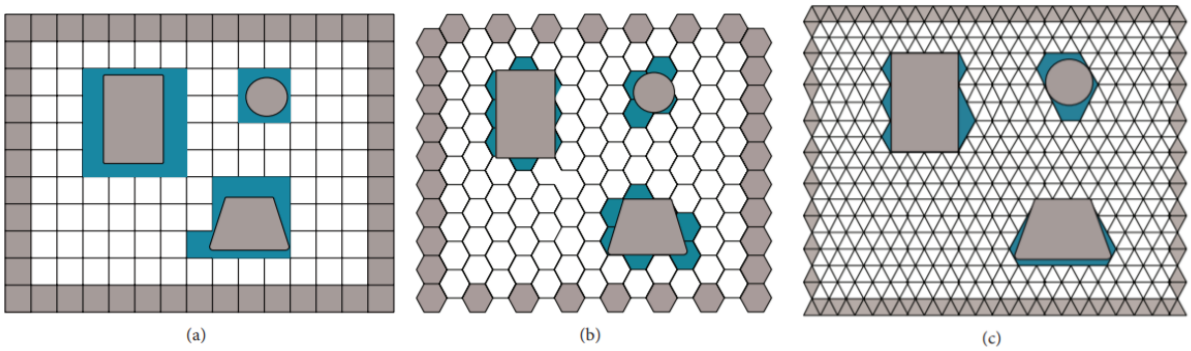
Ďalej budú popísané programové nástroje, ktoré môžu byť použité na tvorbu počítačového kódu a technické prostriedky, ktoré môžu byť spolu s algoritmami použité na realizáciu vybranej navigácie v prostredí.

3.1 Reprezentácia prostredia

Generovanie grafov z terénu prostredia je jeden zo základných problémov pri robotických systémoch a pri video hrách. Existuje niekoľko techník a prístupov. Klasický prístup je reprezentovanie prostredia mriežkovým grafom alebo grafom pozostávajúcim z hrán a vrcholov, kde je potom objekt (autonómny robot alebo herná postava) vykonáva vyhľadávanie cesty pomocou algoritmu. Novším prístupom k reprezentácii priechodného priestoru je použitie navigačnej siete.

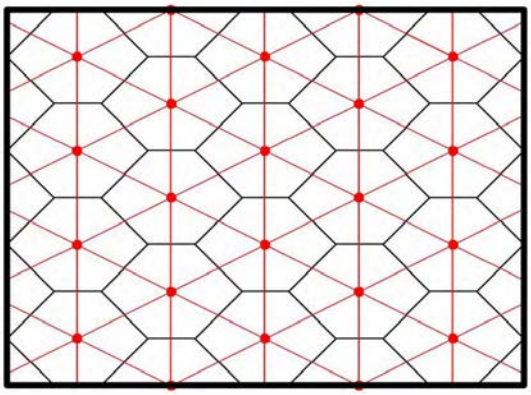
Mriežkový graf

Mriežkový prístup možno opísať ako systematické rozdelenie prostredia na bunky určitého tvaru a veľkosti. Graf je relatívne jednoduché implementovať, preto sa stal populárnou možnosťou reprezentácie priestoru. Každá bunka môže mať tvar trojuholníka, štvorca alebo iného konvexného mnohouholníka a predstavuje malú oblasť priechodného priestoru bez prekážok. Nevýhodou však je že mriežky nemusia pokryť celý priestor, kvôli zložitosti terénu, znázornené modrou farbou na obrázku 23 [14, 15].



Obrázok 23 - Rôzne typy buniek v mriežkovom grafe

Pohyb v rámci hľadania cesty môže byť uskutočnený na hranách alebo po samotných dlaždiciach, kedy je používaný duálny graf mriežkového grafu, kde vrchol predstavuje stred bunky, ktorý je s ostatnými bunkami prepojený hranami [15]:

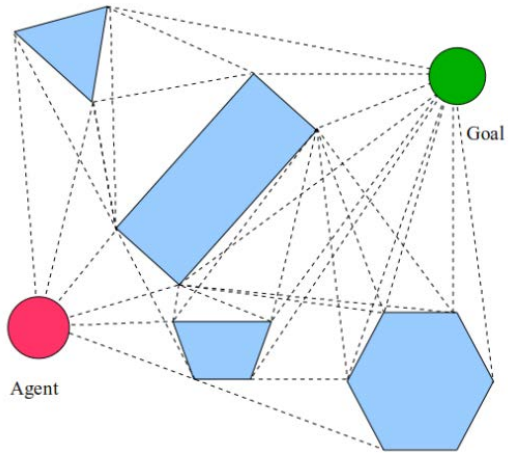


Obrázok 24 - Duálny graf mriežkového grafu

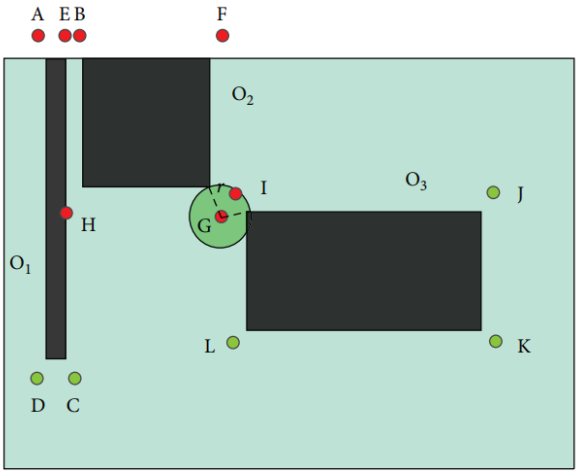
Skeletonizačné techniky extrahujú kostru z kontinuálneho prostredia. Táto kostra zachytáva najvýraznejšiu topológiu (najdôležitejšie body a prekážky, ktoré formujú prostredie) priestoru. Vo všeobecnosti môžu skeletonizačné techniky produkovať dva typy sieťových grafov a to grafy viditeľnosti (Visibility graph) alebo graf trasových bodov (Waypoint graph) [14].

Waypoint graf

Vrcholy (Waypoints) predstavujú prístupné miesta v ktorých sa môže objekt nachádzať, a sú spojené hranami, ktoré reprezentujú rovnú cestu cez ktorú môže objekt prechádzať bez toho aby narazil do nejakej prekážky. Zvyčajne sú tieto grafy využívané vo videohrách a majú tendenciu byť manuálne vybrané a pevne zakódované. Existujú ale aj techniky na automatickú generáciu. Zvyčajne je Waypoint graf tvorený z Visibility grafu (graf viditeľnosti), kde sú hľadané hrany prekážok. Na obrázku 25 je vidieť modré objekty, ktoré sú prekážky a čiary medzi nimi sú kandidáti na priechodné cesty [15, 21].



Obrázok 25 - Graf viditeľnosti

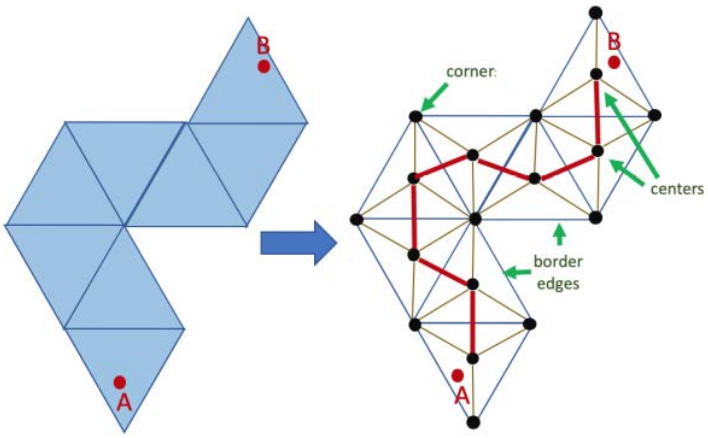


Obrázok 26 - Červené nevhodné a zelené vhodné body

Tvorenie grafu môže prebiehať v týchto krokoch: Pre každú prekážku je zvolených niekoľko kandidátov na body. Pre každý roh prekážky pozostávajúci z dvoch susedných hrán je v uhlovej osi rohu definovaný kandidát na bod so vzdialenosťou *r* od spoločného rohu dvoch hrán prekážky. Niekedy sú body definované priamo na hrane alebo rohu prekážky, no toto môže spôsobovať problémy pre robotov z reálneho prostredia, ktorý majú určitú šírku svojej konštrukcie, napríklad pri okrúhlych vysávačoch s polomerom *r* by bolo definovanie takýchto trasových bodov nevhodné, lebo by do prekážok narážali, preto s týmto prístupom prekážky „nafukujeme“ o vzdialenosť *r*. Kandidáti na Waypointy sú zobrazené na obrázku 26. Sú neplatné, ak sa nachádzajú mimo regiónu (A, B, E a F), vo vnútri prekážok (H) alebo ak kruh so stredom v danom bode a s polomerom *r* pretína prekážky (G, I) [22].

Navigačná sieť

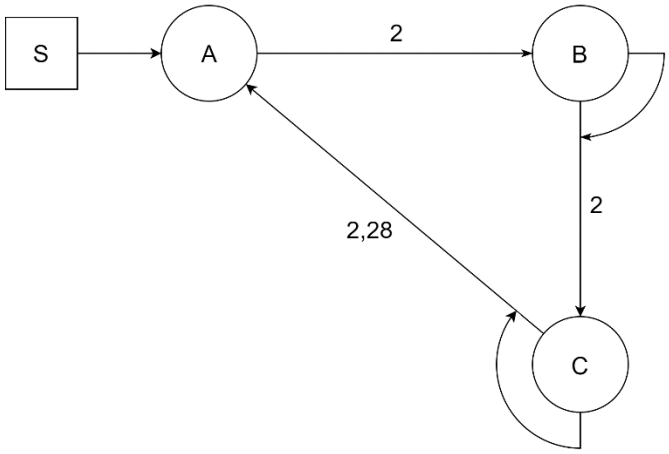
Na rovnom 2D teréne, ako je napríklad izba v dome, možno väčšinu prekážok registrovať napríklad pomocou laserov. Navigácia v neupravenom alebo nerovnomernom teréne je oveľa zložitejšia. Roboty operujúce vonku, pri prieskume neznámeho prostredia, sa musia vyrovnať s takýmto nerovným terénom kde sú špecifické typy prekážok a musia byť schopné nájsť cestu k svojmu cieľu. Medzi takéto prekážky môžu patriť jamy alebo ostré a nebezpečné predmety [23]. Sieťové grafy a grafy viditeľnosti vyzerajú veľmi podobne, ale v skutočnosti to tak nie je. Navigačná sieť je reprezentácia, ktorá v prostredí pokrýva priechodné povrchy pomocou 2D alebo 3D konvexných polygónov. Konvexné polygóny zaručujú, že objekt môže voľne prechádzať z akéhokoľvek miesta v rámci polygónu na akékoľvek iné miesto s rovnakým polygónom, tak ako je znázornené na obrázku 27. Jednou z výhod takéhoto znázornenia je, že dokáže rovnako dobre reprezentovať vnútorné prostredie aj rozsiahle vonkajšie plochy, ale ako aj automatická, aj manuálna generácia navigačnej siete je pomerne zložitá [24].



Obrázok 27 - Navigačná sieť

Vektory

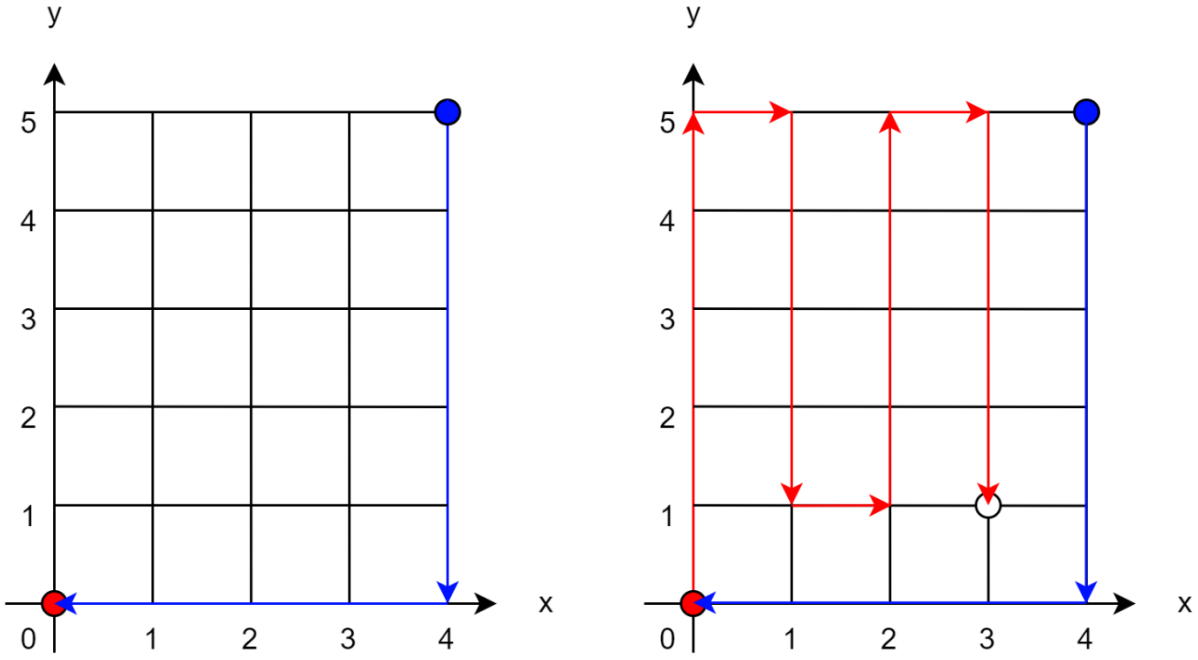
Vektor možno definovať ako geometrický objekt, ktorý je daný dĺžkou, smerom a súradnicami. V prípade grafov je súradnica bod a dĺžka a smer sú hrany medzi nimi. Ak sa robot nachádza v určitom bode a chce sa dostať do iného, musí sa otočiť do smeru vektora a prejsť danú vzdialenosť. Otáčanie je možné monitorovať inerciálnym modulom, v tomto prípade gyroskopom, ktorý pomocou gravitačného zrýchlenia zaznamenáva zmeny pri rotácií okolo osi. Pri danej rýchlosti potom robot vykoná presun určitý počet časových jednotiek na dosiahnutie želanej vzdialenosti. Relatívne k počiatočnému bodu sa robot pomocou gyroskopu otočí do ďalšieho smeru a pokračuje do nasledujúceho bodu. Nevýhodou je, že vektory je potrebné manuálne vybrať a definovať ale riešenie je technicky nenáročné.



Obrázok 28 - Vektorový graf

Súradnicový prístup

Inšpirovaný karteziánskym súradnicovým systémom a rozšírením vektorového a mriežkového prístupu, súradnicový prístup môže reprezentovať prostredie, ktoré bolo kalibrované prejdením dvoch alebo viac hrán, napríklad z modrého počiatočného bodu do červeného bodu, ako je ilustrované na obrázku 29. Dve hrany postačia na definovanie štvorcovej alebo obdĺžnikovej plochy, ktorá je následne prejdená. Červená cesta je potom vykonaná algoritmom, špirálovo alebo pohybom hore dole, po ose *y* so vzdialenosťou *y – 1* z kalibračnej cesty, otáčajúc sa v cykle, *x – 1* krát. Ak uvažujeme že zariadenie je široké presne o jednotku vzdialenosti na súradnici, v konečnom bielom bode prejde celú plochu.

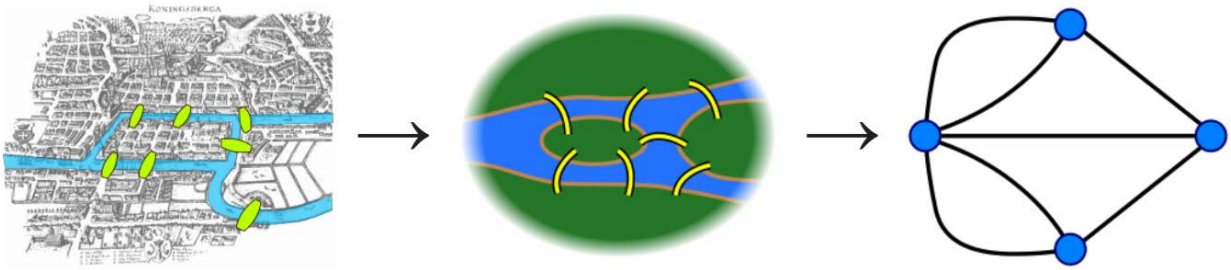


Obrázok 29 - Súradnicový prístup reprezentácie prostredia

Ďalším krokom v procese hľadania cesty je samotný vyhľadávací algoritmus. Hlavný problém prestavuje nájsť optimálnu trasu nejakým efektívnym spôsobom. Vývojári hier a robotickí inžinieri používajú rôzne techniky. Značný zlomok poznatkov však čerpajú z teórie grafov.

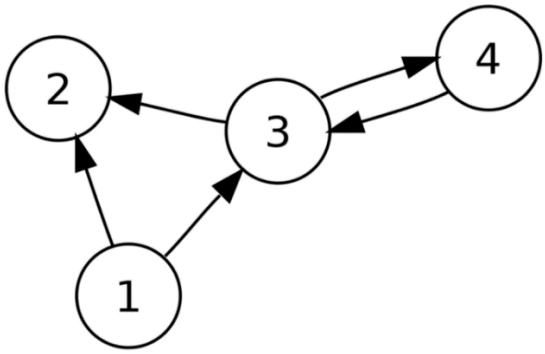
3.2 Teória sieťových grafov a algoritmy na hľadanie optimálnej cesty

Základnú myšlienku grafov prvýkrát predstavil v 18. storočí švajčiarsky matematik Leonhard Euler, jeden z najvýznamnejších matematikov všetkých čias. Jeho práca o slávnom probléme „Sedem mostov z Königsbergu“ sa bežne uvádza ako pôvod teórie grafov. Problém spočítal v tom, že treba nájsť cestu, ktorou by sa dalo prejsť celým mestom s tým, že každý most prejdeme len raz. Eulerovi bolo jasné, že najdôležitejšie prvky sú miesta návštevy (jednotlivé ostrovy) a cesty medzi nimi (mosty). Abstrakciou mapy mesta vznikol prvý graf, ktorý mal iba relevantné údaje potrebné na vyriešenie problému. Nakoniec sa ale ukázalo, že problém riešenie nemá [25]. Vysvetlenie prečo, bude v ďalšej kapitole.

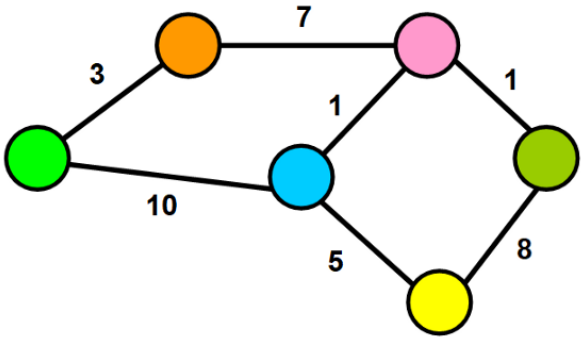


Obrázok 30 - Abstrakcia mapy až po vytvorenie prvého grafu

Ako píšu autori I. Brezina a P. Gežík [26]: „Najjednoduchšie možno graf charakterizovať ako objekt, ktorý vznikne pospájaním vrcholov (reprezentované bodmi) čiarami. Je to teda útvar, ktorý tvorí množina bodov (vrcholov) a množina ich spojníc (hrán).“ Grafy sú teda zbierkou uzlov – reprezentujúcich určitú triedu objektov, ako sú ľudia, podnikové rady, proteíny alebo destinácie na zemeguli – a hrán, ktoré slúžia na znázornenie spojení, ako sú priateľstvá, cesty, mosty alebo interakcie molekulárnych väzieb [27]. Stupeň vrcholu udáva, koľko hrán z neho vychádza. Graf môže byť ohodnotený – jeho hrany majú váhu, ktorá môže reprezentovať intenzitu, cenu, vzdialenosť, priepustnosť, a iné. S ohodnoteným grafom sa bežne stretávame pri optimalizácii cesty. Taktiež poznáme aj hranovo orientované grafy – jeho hrany sú dané začiatočným a koncovým vrcholom, v grafe označené šípkou. Opakom je neorientovaný graf, cez ktorého hrany môžeme prechádzať ľubovoľným smerom [26].



Obrázok 31 - Hranovo orientovaný graf



Obrázok 32 - Ohodnotený graf

Grafy sa dajú použiť na modelovanie mnohých typov vzťahov a procesov vo fyzikálnych, biologických, sociálnych a informačných systémoch a má široké spektrum užitočných aplikácií ako napríklad:

- hľadanie komunít a nových priateľov v sociálnych sieťach,
- monitorovanie možného šírenia vírusu medzi ľuďmi, ktorí boli spolu v kontakte,
- štúdium molekúl a atómov v chémii,
- sekvenovanie DNA,
- bezpečnosť počítačovej siete,
- nájdenie najkratšej cesty [25].

Vrcholy sú jednotlivé body, ktoré môžu byť navštívené a hrany reprezentujú cesty medzi nimi. Podľa toho akú cestu a aký ciel v grafe máme, existuje niekoľko rôznych algoritmov, ktoré sú popísané v nasledujúcich podkapitolách.

Než budú popísané rôzne algoritmy na hľadanie cesty a metódy uchovania grafu v počítačovej pamäti, treba podotknúť, že pri počítačových systémoch je dôležitým faktorom časová zložitosť a pamäťová zložitosť. Časová zložitosť algoritmu kvantifikuje množstvo času potrebného na uskutočnenie algoritmu v závislosti od meniacej sa dĺžky vstupu. Podobne priestorová zložitosť algoritmu kvantifikuje množstvo priestoru alebo pamäte, ktorú algoritmus potrebuje na riešenie úlohy v závislosti od meniacej sa dĺžky vstupu [26]. Časová a pamäťová zložitosť závisí okrem samotného algoritmu aj od iných vecí, ako je hardvér, operačný systém, procesory atď. Je pri tom využitá Big-O notácia, ktorá predstavuje hornú hranicu týchto zložitostí [28]. Prirodzeným cieľom každého systému je minimalizovať resp. optimalizovať časovú aj pamäťovú zložitosť.

3.2.1 Metódy na prejdenie všetkých hrán

Eulerovská cesta je sled v grafe, ktorá začína v začiatočnom vrchole, prejde cez všetky hrany a skončí v konečnom vrchole. Podmienkou je aby všetky hrany mali párny stupeň, ale začiatočný a koncový bod musia mať nepárne. Logika za touto podmienkou je, že keď jednou hranou vstúpime do vrcholu, musíme ho ďalšou, inou hranou opustiť. Do začiatočného (koncového) bodu nevstupujeme (nevystupujeme) žiadnou hranou, preto musia mať nepárny uhol. Eulerovský cyklus je uzatvorený sled obsahujúci všetky prvky grafu. Začína a končí v tom istom vrchole. Podmienkou je aby všetky hrany mali párny stupeň. Postup:

- Z ľubovoľného začiatočného vrcholu *V* vyberať hrany, až po opätovné navštívenie vrcholu *V*. Vznikne uzatvorený cyklus ale všetky hrany nemuseli byť navštívené,
- Ak má vrchov *V* hranu, ktorá ešte nebola navštívená, opakovať krok a vyberať hrany až po opätovné navštívenie vrcholu *V*.
- V prípade hľadania Eulerovskej cesty musí byť začiatočný vrchol nepárneho stupňa [26].

Fleuryho algoritmus

Fleuryho algoritmus, ktorý využíva *DFS (Depth First Search)* heuristiku je algoritmus na nájdenie Eulerovského cyklu v neorientovanom grafe (ktorý obsahuje Eulerovskú cestu alebo cyklus) v exponenciálnej časovej náročnosti $O(e^2)$, kde e je počet hrán. Heuristická funkcia alebo heuristika je skratka na vyriešenie problému, keď preň neexistujú presné riešenia alebo čas na získanie riešenia je príliš dlhý. Cieľom je nájsť rýchlejšie riešenie, aj keď nie je optimálne. Inými slovami, pri použití heuristiky vymieňame presnosť za rýchlosť riešenia a tým pádom je suboptimálne. *DFS* začína v zvolenom uzle a skúma cestu čo najďalej ako je to možné, bez tvorenia cyklov, až po kým sa nedostane do uzlu z ktorého nevedie cesta do nenavštíveného uzlu. Následne sa vráti (backtracking) a preskúma všetky ostatné uzly rovnakým princípom. Podobná heuristika s názvom *BFS (Breadth First Seach)* začína v zvolenom uzle a prechádza cez všetky jeho susedné uzly (priamo pripojené k počiatočnému uzlu). Tento krok je opakovaný pre každú úroveň. V prípade hranovo orientovaného grafu je možné použiť elegantný a efektívny Hierholzerov algoritmus, ktorý nájde cestu v lineárnom čase $O(v+e)$, kde v je počet vrcholov [29].

Problém čínskeho poštára

Problémom čínskeho poštára (CPP, Chinese Postman Problem) je nájsť najkratšiu obežnú cestu, ktorá pokrýva všetky jednotlivé cesty v cestnej sieti. Uplatnenie je možné v orientovaných, neorientovaných a aj v zmiešaných grafoch. V zmiešaných sieťach je možné

prejsť niektorými cestami obojsmerne a inými len jedným určeným smerom. CPP je možné aplikovať na smerovanie vozidiel na doručovanie pošty alebo na zber domáceho odpadu, zametače ulíc, snehové pluhy, školské autobusy a veľa iných. V meste je orientovaná cestná sieť rozdelená na dva smery, prípadne sa niekde môžu vyskytnúť jednosmerné cesty. Abstrakciou takejto siete by bol orientovaný alebo zmiešaný graf. Mestské cesty môžu byť aj viacprúdové, čo by bolo v grafe znázornené násobnou hranou medzi vrcholmi [30].

Nie každý graf má Eulerovskú cestu alebo cyklus a navštívenie každej jednej hrany práve jeden krát nie je možné. Preto niekedy pri CPP využívame Eulerizáciu. Eulerizácia je proces pridávania hrán do grafu na vytvorenie Eulerovho sledu v grafe. Na Eulerizáciu grafu sa hrany duplikujú, aby sa spojili dvojice vrcholov s nepárnym stupňom. Spojenie dvoch vrcholov nepárneho stupňa zvyšuje stupeň každého z nich a dáva im párny stupeň. Ak sú hrany ohodnotené, hrany duplikujeme tak, aby suma cien nových hrán bola čo najmenšia. Postup Eulerizácie grafu čínskeho poštára:

- Nájsť všetky nepárne vrcholy,
- Nájsť všetky možné páry nepárnych vrcholov,
- Pre každý pár nájsť hrany, ktoré spájajú vrcholy s minimálnou cenou,
- Nájsť páry tak, aby bol súčet váh minimalizovaný,
- Do pôvodného grafu pridať hrany, ktoré boli nájdené v kroku 4,
- Dĺžka optimálnej trasy čínskeho poštára je súčet všetkých hrán pripočítaných k pridaným hranám [32].

3.2.2 Metódy na prejdenie všetkých bodov

Primov algoritmus (1957), známy ako aj Jarníkov algoritmus, a Kruskalov algoritmus (1956) na báze Borůvkovho algoritmu (1926), sú greedy algoritmy, ktoré sa používajú na nájdenie minimálnej kostry grafu. Minimálna kostra grafu je podmnožina daného grafu, ktorá zahŕňa všetky jeho vrcholy a súčet hrán je minimalizovaný [26]. Nižšie sú popísané princípy fungovania:

Primov algoritmus

- Vybrať ľubovoľný počiatočný vrchol,
- Vybrať hranu s najnižším ohodnotením, ktorá patrí do množiny hrán vychádzajúcich z počiatočného vrcholu,
- Prejsť všetky vrcholy v kostre a pridávať hrany v kostre nezaradených hrán s najnižším ohodnotením, tak aby nevznikla kružnica [26].

Kruskalov algoritmus

- Zoradiť všetky hrany od najnižšieho ohodnotenie po najvyššie,
- Vybrať hranu s najnižšou hmotnosťou a pridať do kostry tak aby nevznikla kružnica,
- Pokračovať v pridávaní hrán, kým sa v kostre nenachádzajú všetky vrcholy [26].

3.2.3 Metódy na prejdenie z počiatočného vrcholu do koncového

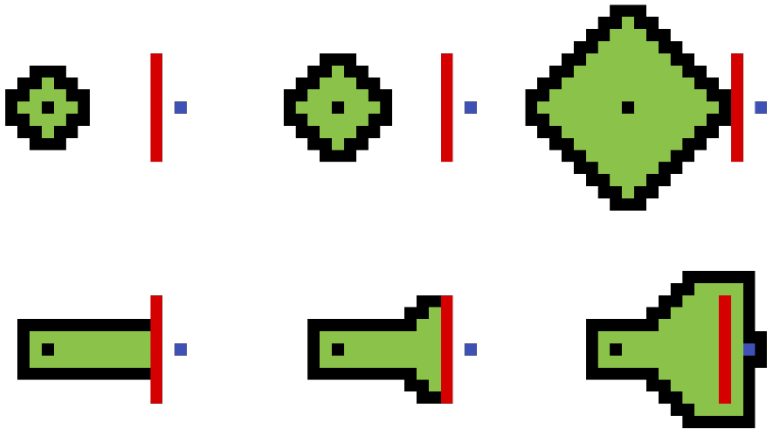
Dijkstrov algoritmus

Dijkstrov algoritmus bol navrhnutý holandským matematikom Edsgerom Wyberom Dijkstrom v roku 1959. Primárne sa využíva na hľadanie najkratšej cesty v hranovo-ohodnotenom orientovanom grafe z počiatočného bodu do koncového bodu. Graf sa skladá z uzlov, respektíve vrcholov a hrán, ktoré sú ohodnotené a vyjadrujú cenu, trvanie alebo dĺžku cesty. Časová zložitosť je $O(v^2)$. Princíp algoritmu:

- Dijkstrov algoritmus začína vo zvolenom počiatočnom uzle a 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 ak nájde kratšiu cestu, aktualizuje tieto hodnoty,
- 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 zdrojový uzol so všetkými ostatnými uzlami po najkratšej možnej ceste na dosiahnutie každého uzla,
- Podmienkou je, aby hrany neboli ohodnotené záporným číslom [32].

A*

A* je jedným z najznámejších vyhľadávacích algoritmov v oblasti hier a robotiky. Je to vylepšenie Dijkstrovho algoritmu, pretože ako prvý algoritmus používal heuristickú funkciu *BFS (Best First Search)* na hľadanie cesty v ohodnotenom grafe. Na miesto skúmania všetkých možných stavov sú skúmané len tie, ktoré sa zdajú že ponúkajú lepšie riešenie. Heuristika v A* algoritme odhaduje cestu z uzlov do cieľa sčítaním ceny cesty a reálnej vzdialenosti do cieľa (vo všeobecnosti sa chceme blížiť smerom k cieľu, nie od neho, obetujeme tak ale preskúmanie potenciálnej cesty, ktorá síce ide okľukou ale môže byť lacnejšia). Časová zložitosť je $O(e)$. Na obrázku 33 je v prvom riadku ilustrované hľadanie cesty z počiatočného čierneho bodu do modrého bodu cez červenú prekážku bez heuristiky a v druhom riadku hľadanie cesty s heuristikou, zelené dlaždice sú preskúmané uzly [33, 34].



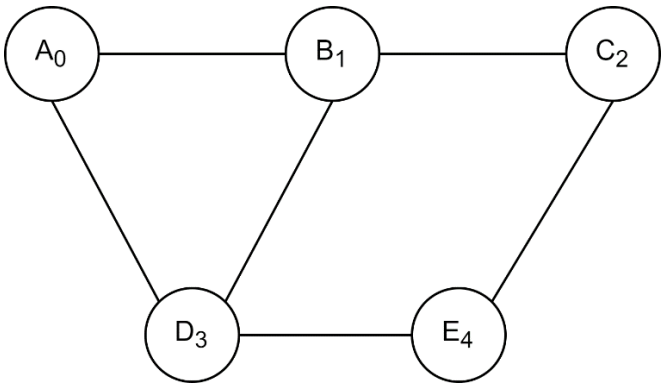
Obrázok 33 - Hľadanie najkratšej cesty bez heuristiky a s heuristikou

3.3 Uchovanie grafu v počítačovej pamäti

Matica susednosti je v matematike a informatike používaná na reprezentáciu grafov. Je to 2D matica, ktorá sa používa na mapovanie asociácii medzi hranami grafu. Ak má graf *G* počet vrcholov *v*, potom matica susednosti tohto grafu je *v*v* a každý záznam matice predstavuje počet hrán od jedného vrcholu k druhému. Graf *G* je potom definovaný maticou susednosti *M* [a_{ij}], kde:

a_{ij} = 1 {ak existuje cesta z v_i do v_j}

a_{ij} = 0 {inak} [35]



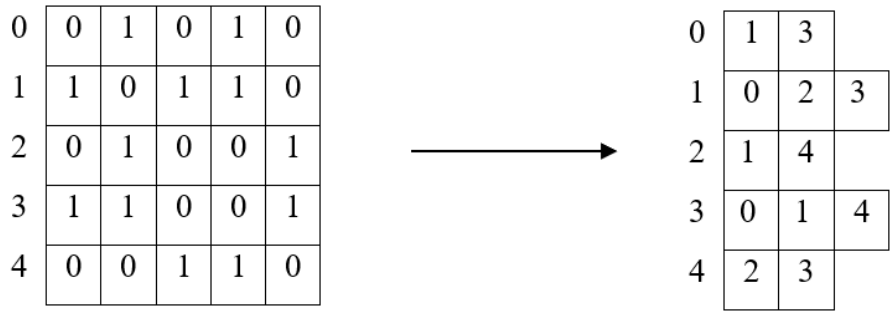
Obrázok 34 - Graf *G*

	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	0
C	0	1	0	0	1
D	1	1	0	0	1
E	0	0	1	1	0

Obrázok 35 - Matica susednosti *M* grafu *G*

Matica susednosti je efektívna v rámci časovej náročnosti operácií. Čas nájdenia existujúcej cesty medzi vrcholmi je O(1). Čas nájdenia susedných vrcholov je O(*v*), kde *v* je počet vrcholov. Avšak matica susednosti nie je veľmi efektívna v rámci spotreby pamäte. Konkrétne to je O(v^2), kde *v* je počet vrcholov. Často sa stáva, že existujúcich asociácií (hrán) je oveľa menej ako všetkých možných asociácií. Všetky tieto neexistujúce asociácie sú v matici uložené ako 0. Ako príklad je možné uviesť sociálnu sieť, ktorú používa 1 miliarda ľudí. Každý užívateľ by mal teda v matici susednosti 1,000,000,000 záznamov. Ak by mal tento užívateľ 1000 priateľov, tieto záznamy by boli reprezentované číslom 1, ostatných

999,999,000 záznamov by bolo reprezentovaných číslom 0, čo je veľmi nepraktické. Pre väčšinu grafov platí: $e < v^2$ kde e je počet hrán a v je počet vrcholov. Ak by boli tieto redundantné nuly vymazané a jednotky nahradené konkrétnymi spojenými vrcholmi, z matice *M* na obrázku 36 by sme dostali niečo takéto [36]:



Obrázok 36 - Dvojrozmerný zoznam polí, ktoré reprezentujú hrany medzi vrcholmi

Graf je definovaný ako zoznam párov vrcholov, medzi ktorými je hrana. Výhodou takejto reprezentácie neorientovaného grafu je, že jednotlivé páry môžu byť v ľubovoľnom poradí. Hrana medzi vrcholmi A-B je tá istá hrana ako B-A. Napríklad na indexe 2 má pole hodnoty 1 a 4. Znamená to, že vrcholy 1, 2 a 4 sú prepojené hranou. Takáto štruktúra je šetrnejšia k spotrebe pamäti, konkrétne O(*e*). V rámci časovej efektivity nájdenia existujúcej cesty alebo nájdenia susedného vrcholu je O(*v*). Pri implementácii v programovacom jazyku C/C++ je možné využiť pole pointerov, ktoré ukazujú na polia rôznej veľkosti alebo využiť spájaný zoznam, kde je možné uchovať aj hodnotu hrany [36]. V počítačovom kóde by rôzne reprezentácie vyzerali takto:

```cpp
#include <iostream>
using namespace std;

int main()
{
    int M[4][4];

    int *A[5];
    A[0] = int[2];
    A[1] = int[3];
    A[2] = int[2];
    A[3] = int[2];
    A[4] = int[2];

    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    int weight;
    struct Node* next;
};

int main()
{
    struct Node *N[5];

    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

struct Edge {
    int startEdge;
    int endEdge;
    int weight;
};

int main()
{
    struct Edge V[5];

    return 0;
}
```

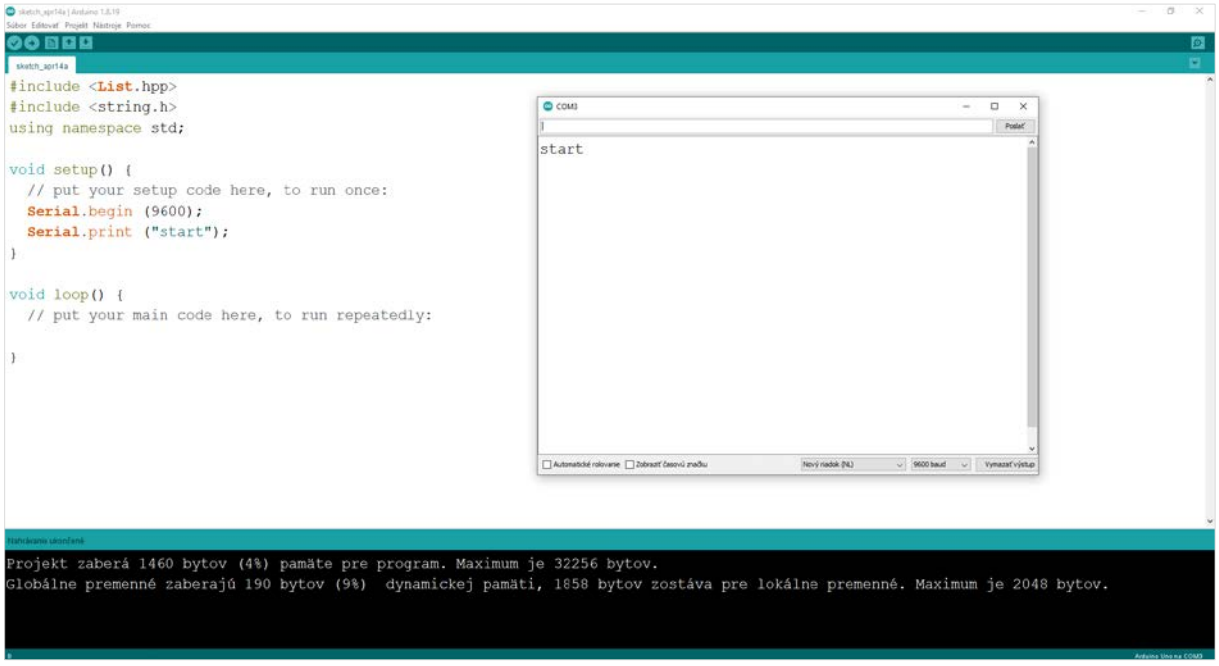
Obrázok 37 - Graf uložený ako matica, spájaný zoznam a ako trieda

V rámci jazdenia a otáčania, pri najtriviálnejšej implementácii, sú používané štyri funkcie na pohyb rovno, doľava, doprava a dozadu, ktoré môžu byť reprezentované char znakom „r“, „l“, „p“ a „z“, ktoré sú za sebou ukladané do dynamického poľa alebo listu. Pre zopakovanie funkcií je pole alebo list čítané a podľa písmena volané funkcie na pohyb. Časové a pamäťové zložitosti sú pri tejto implementácii lineárne O(1).

3.4 Programovacie nástroje

Arduino IDE

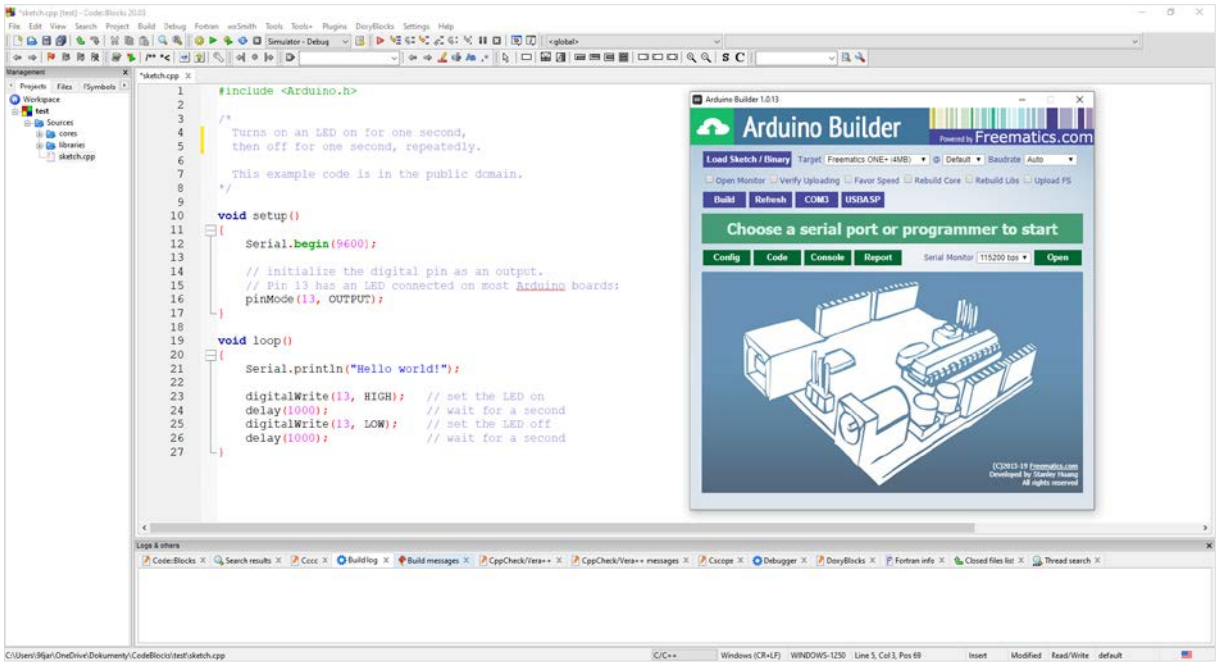
V bakalárskej práci bolo popísané vývojové prostredie *Arduino IDE*, kde je možné kód písať, kompilovať a rovno nahrať na mikropočítač. Monitor sériového portu slúži na čítanie výstupu [1]. V rámci uľahčenia tvorby kódu a rozšírenia funkcionality, je možné používať knižnice. Knižnice predstavujú súbor definovaných metód a funkcií a môžu byť vo všeobecnosti používané vo vlastných programoch podľa potreby. V *Arduino IDE* je možné zahrnúť knižnicu navigáciou cez možnosti na lište Projekt > Zahrnúť knižnice > Spravovať knižnice. Vyskočí okno, kde je vo vyhľadávacom poli možné podľa názvu, kľúčových slov alebo popisu nájsť vyžadovanú knižnicu. Je tiež možné vybrať možnosť Pridať .ZIP knižnicu uloženú v počítači. Dajú sa stiahnuť z oficiálnych stránok Arduino alebo z rôznych fór alebo z GitHub. Na stránkach sú dostupné aj oficiálne dokumentácie, ktoré vysvetľujú ako s knižnicami pracovať.



Obrázok 38 - *Arduino IDE* s monitorom sériového portu

Code::Blocks Arduino IDE

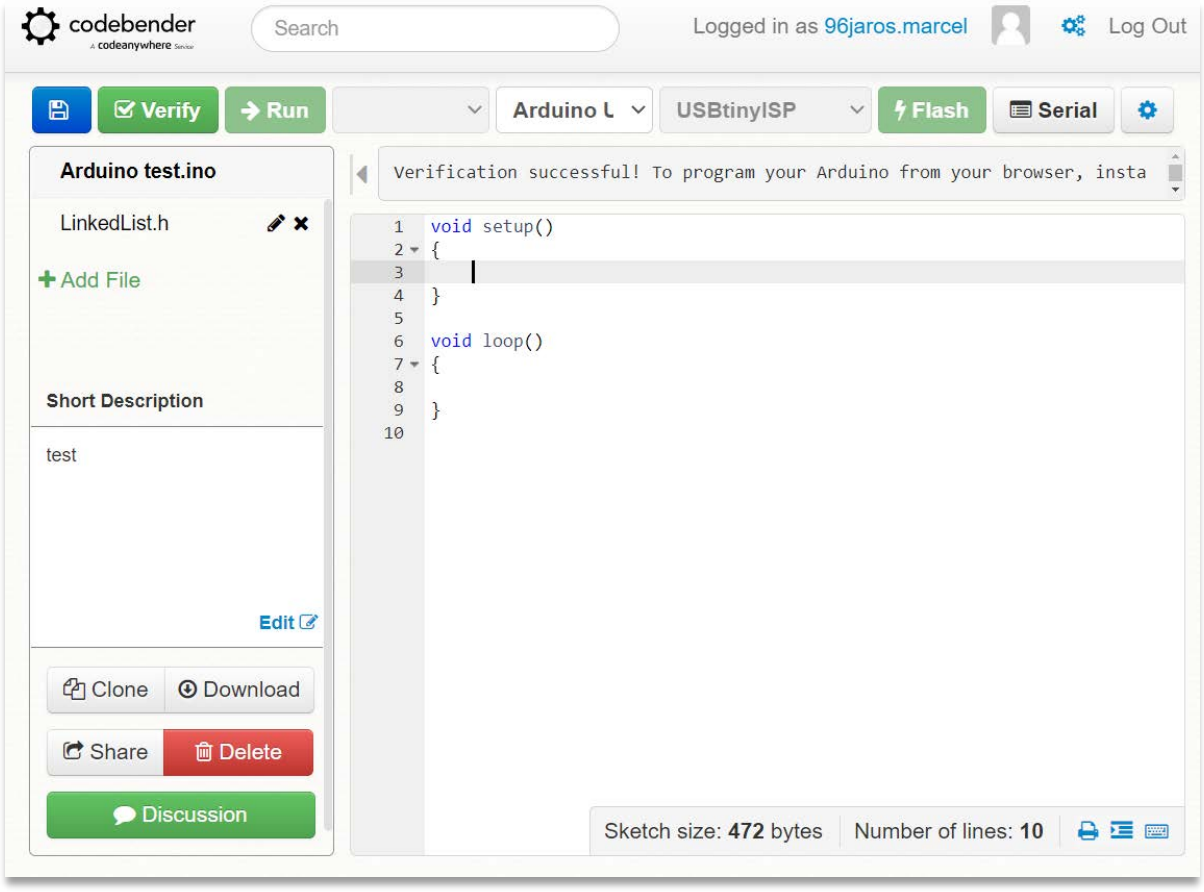
Program je tiež možné písať v open source vývojovom prostredí *Code::Blocks* resp. v jeho upravenej distribúcii *Code::Blocks Arduino IDE*. Disponuje nástrojom *Arduino Builder* cez ktorý je možné kód nahrať. Ponúka možnosti moderného IDE, ako skladanie kódu, automatické dokončovanie, pokročilá navigácia a zvýraznenie príkazu v celom kóde a iné. Obsahuje všetky základné knižnice Arduino, ale ponúka aj pridanie vlastných knižníc, ktoré sú uložené v počítači, navigáciou cez Project > Build options > Linker settings > Add.



Obrázok 39 - *Code::Blocks Arduino IDE* a *Arduino Builder*

Codebender

Ďalšou možnosťou je webová aplikácia *Codebender*. Po inštalácii pluginu je možné kód písať priamo v prehliadači.



Obrázok 40 - Web aplikácia *Codebender* v prehliadači s cloudovými možnosťami

Taktiež je na stránke zabudovaná možnosť kompilovať kód a nahrávať ho na mikropočítač a sledovať výstup zo sériového portu. Najväčšou výhodou aplikácie je, že ponúka ukladanie na cloud; kód je tak dostupný hocikde, stačí sa prihlásiť do svojho účtu. Je možné prehľadávať a použiť kódy tvorené od iných používateľov alebo takisto zdieľať vlastný projekt a viesť o ňom s používateľmi diskusiu. Knižnice je jednoducho možné pridávať cez ikonu + Add File a výberom súboru z počítača.

Všetky tieto možnosti sú multiplatformové; aplikácie sa dajú používať na Windows, Linux a aj Mac OS X. *Arduino IDE* je jednoduché a vhodné pre začiatočníkov, poskytuje všetky nástroje na písanie jednoduchých programov. *Code::Blocks Arduino IDE* poskytuje pokročilé možnosti vývojového prostredia a je vhodné pre náročnejších programátorov alebo pre veľké projekty. Aplikácia *Codebender* je vďaka cloudu vhodná pre decentralizovanú tvorbu projektu alebo pre programátorov, ktorí sa chcú angažovať v komunite.

3.5 Technické prostriedky

V bakalárskej práci bolo osobitne popísaných viacero modulov a súčiastok [1]. V tejto kapitole budú rozšírené možnosti aplikácie týchto súčiastok alebo nové metódy spracovania, ktoré kombinujú viacero modulov za účelom presnejšej navigácie.

Replikácia LIDAR

LIDAR pozostáva z vysielača, ktorý osvetľuje cieľ laserovým lúčom, a prijímača schopného detegovať odrazený lúč. Senzory prijímača vypočítavajú vzdialenosť na základe času potrebného na to, aby lúč dosiahol cieľ a vrátil sa. Vzorec na vypočítanie vzdialenosti:

$$d = c * i / 2; \quad (1)$$

kde c je rýchlosť svetla,

i je jednotka času odkedy lúč vyšiel z prijímača a bol detegovaný senzorom (/2 pretože tú istú vzdialenosť lúč prešiel dva krát).

Druhou časťou je servo motor, čo je špecifický typ motora u ktorého je na rozdiel u bežného motora možné nastavenie presnej polohy natočenia okolo svojej osi. Postup pre zistenie všetkých prekážok v okolí robota: Pre každý stupeň uhla od 0 po 360 v ktorej je servo motor zmeraj a vypíš vzdialenosť z laser modulu vypočítanú z hore uvedeného vzorca (1).

GSP a gyroskop

Zemepisná dĺžka a šírka sú nástroje, ktoré presne určujú akékoľvek miesto na Zemi, respektíve každé z týchto miest má v súradniciach absolútnu hodnotu, ktorá sa nemení. Ak sú nám v opačnom prípade GPS súradnice prístupné, vieme tieto miesta identifikovať. Zemepisná šírka a dĺžka sú rozdelené do stupňov, minút, sekúnd a smerov, počnúc

zemepisnou šírkou. Napríklad oblasť so súradnicami označenými ako N 48° 7' 18.491" E 17° 6' 24.458" by sa čítala ako 48 stupňov, 7 minút, 18,491 sekúnd severne a 17 stupňov, 6 minút, 24.458 sekúnd východne. GPS je zvyčajne presné na 2-5 metrov. Samotné GPS teda nie je vhodné ako jediná technológia na orientovanie pre robota na kosenie trávy v záhrade, kde aj meter hore dole môže znamenať neželané pokosenie záhona kvietkov. Je treba asistencia iných orientačných senzorov, ako bolo naznačené v bakalárskej práci.

Gyroskop modul kontinuálne meria zmenu pozície objektu na základe gravitačného zrýchlenia. Takéto namerané údaje môžu slúžiť ako reprezentácia už prejdenej cesty od nejakého počiatočného bodu. Na rozdiel od daných GPS súradníc, takáto navigácia je relatívna k počiatočnému bodu, ktorý musí ostať rovnaký. Údaje môžu slúžiť ako informácia že zariadenie na danej polohe už bolo alebo ako spiatočná cesta do počiatočného bodu. Ak je objekt na pozícii X100 a posunie sa na X180, rozdielom aktuálnej a predošlej hodnoty je možné získať informácie o zmene pohybu. Zariadenie sa na osi X pohlo o 180 – 100 = 80 jednotiek. Merania gyroskopu však nie sú vždy úplne presné, takéto merania musia byť uskutočnené veľa krát v priebehu každej sekundy. Zmeny je možné monitorovať pseudokódom:

```
newX = getGyroData();
diff = newX – nowX;
nowX = newX;
```

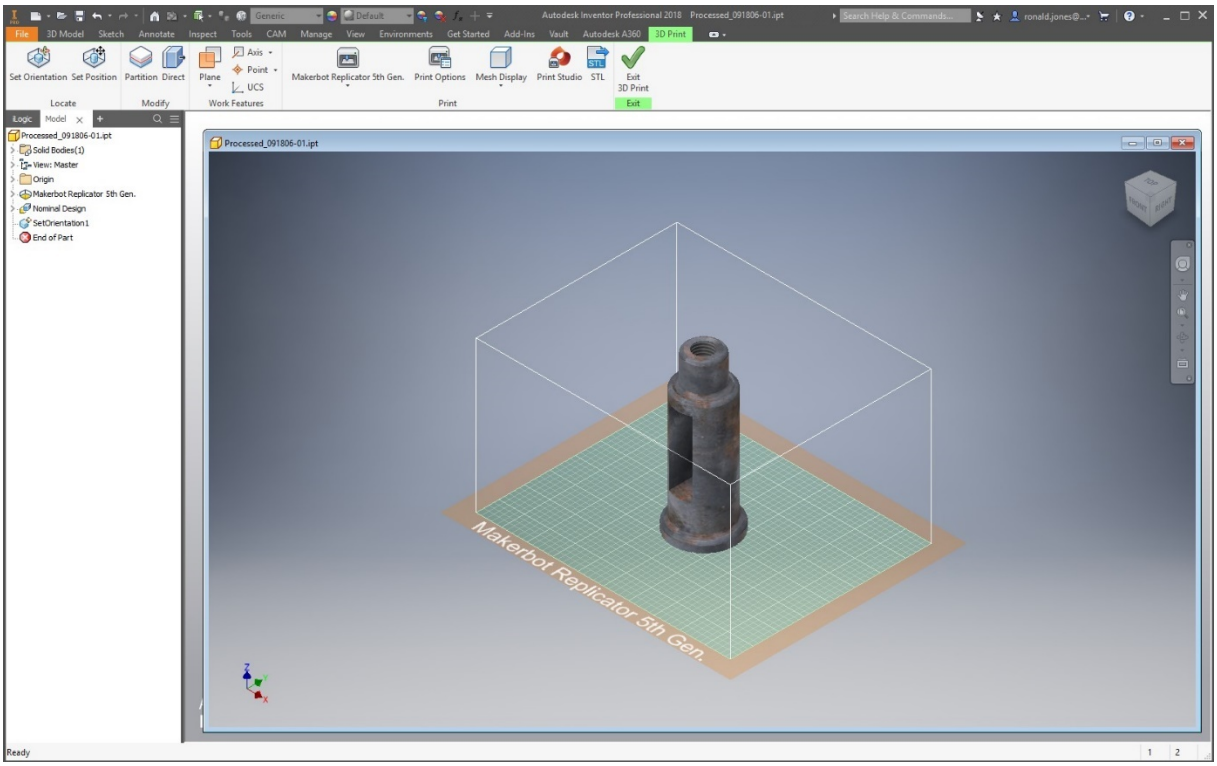
3D tlač

V bakalárskej práci boli popísané aj rôzne dizajny a materiály, ktoré mohli byť použité na konštrukciu robota. Ďalšia možnosť je 3D tlač, čo je relatívne nová technológia vytvárania trojrozmerného objektu. Tlačiareň číta súbor z počítača, ktorý určuje, ako vrstviť materiál. Aplikácie sú naozaj široké, od tlačenia rôznych bežných súčiastok, hračiek, bytov, protetických končatín a dokonca aj vesmírnych rakiet a ich motorov. Zaujímavé príklady:

Pri štarte vesmírnej rakety dosahuje vnútro spaľovacej komory teploty, ktoré by roztavili hocijaký kov. Aby sa ale raketa neroztavila, komora je chladená komplexným systémom trubičiek, cez ktoré preteká tekutý dusík. Tradične na takýchto motoroch býva viac ako tisíc trubičiek, čo je pracovne, finančne a časovo extrémne náročné. 3D tlač umožňuje výrobu spaľovacej komory ako jeden kus, za veľmi malý pracovný, finančný a časový zlomok ako pri tradičnej metóde [36]. Ak dieťa stratí končatinu, poisťovňa väčšinou nepreplá náklady na protetickú robotickú končatinu kvôli tomu, že dieťa rastie a musela by sa meniť každého pol roka. 3D tlač umožňuje lacnú reprodukciu a redesign a tak výrazne zvyšuje dostupnosť týchto produktov [37].

Výhodou sú nízke náklady, možnosť produkcie komplexnejších návrhov, zníženie počtu súčiastok, menší ekologický dopad na životné prostredie a relatívne ľahká a lacná zmena dizajnu počas produkcie.

Postup 3D tlače začína pri 3D modelovacom softvéri a tvorbe modelu. Hotový návrh je exportovaný ako *.stl súbor a následnej je nahraný do „krájacieho“ programu. Medzi komerčne dostupné programy patria napríklad *Simplify3D* alebo *Cura*. Tu sa nastaví ako sa bude materiál vrstviť, akou cestou pôjde, a iné. Je možné použiť robustnejšie programy kde je integrovaná podpora pre 3D tlač, ako napríklad *Autodesk Inventor*. 3D súbory je možné zakúpiť alebo stiahnuť od iného autora. V poslednom kroku je súbor nahraný do 3D tlačiarne, ktorá pomocou materiálu vytvorí reálny 3D objekt.



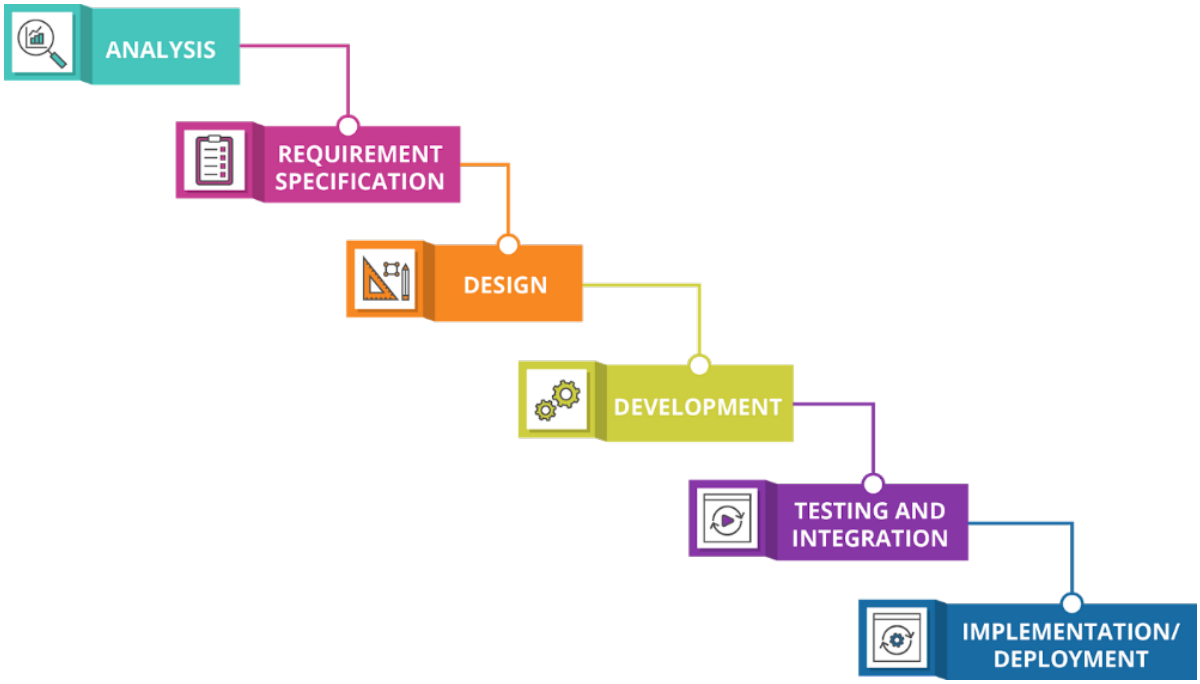
Obrázok 41 - *Autodesk Inventor* na tvorbu 3D modelov s integrovanou podporou procesu 3D tlače

3.6 Metódy tvorby informačných systémov

Cyklus tvorenia informačného systému, tak isto zvaný ako SDLC (Software Development Life Cycle), je zložitý proces tvorby nejakého systému a je vhodné postupovať podľa určitého overeného postupu alebo metodiky. Existujú rôzne modely vývoja informačných systémov, ktoré určujú postupy, pravidlá, nadväznosť fáz, uľahčujú plánovanie a riadenie projektu. Taktiež sa označujú ako SDPM (Software Development Process Models) a každý z tých modelov má svoje špecifické kroky, aby zabezpečil úspech v procese vývoja softvéru. Medzi základné modely patria:

Vodopádový model

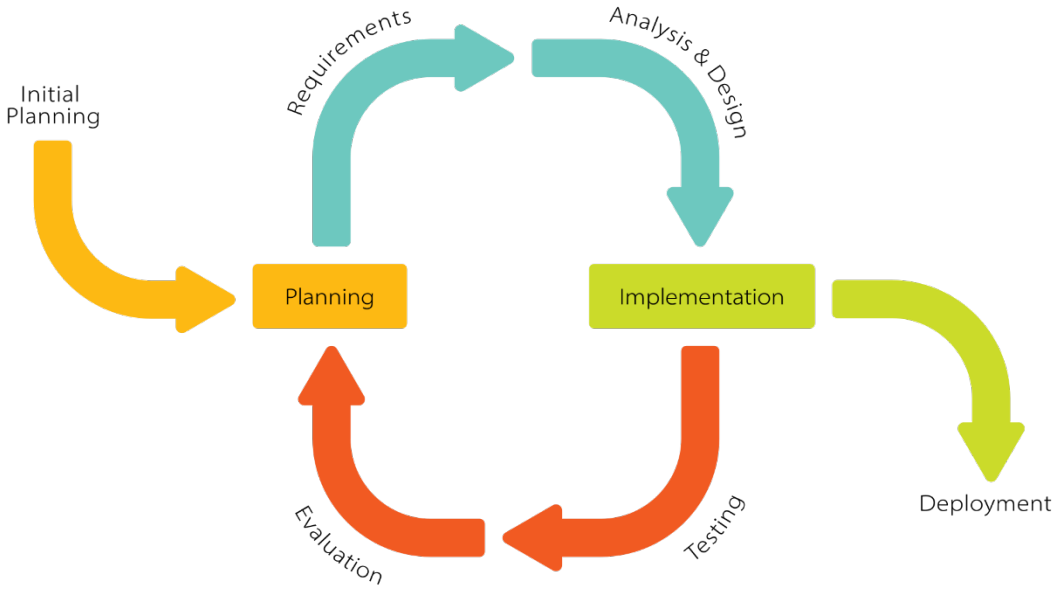
Vodopádový model, taktiež nazývaný ako Sekvenčný model alebo Klasický model je najstarší model a vyžaduje, aby každý krok bol postupne ukončený a zdokumentovaný pred začatím ďalšieho kroku. Je jednoduché sa ním riadiť a je vhodný pre menej flexibilné tímy. Nadväznosť krokov je intuitívna a logická. Avšak veľkou nevýhodou je, že veľa energie sa investuje do pochopenia a eliminácie rizika prostredníctvom snahy predvídať konečný stav produktu s predstihom. Takéto rozsiahle plánovanie môže pre nejaké projekty trvať aj roky bez znalosti, ako sa technológie a trh zmenia počas doby vývoja. Vracanie sa k predošlým krokom je veľmi nákladné, hlavne ak zistíme vady v predposlednom kroku, pri testovaní. Z týchto dôvodov, je vhodnejší pre kratšie projekty a produkty, ktoré sú novšie iterácie existujúcich produktov [39].



Obrázok 42 - Vodopádový model

Iteratívny model

Na rozdiel od postupného a lineárneho Vodopádového modelu je Iteratívny model súbor cyklických procesov, kedy sa po počiatočnej plánovacej fáze malá hŕstka etáp opakuje znova a znova, pričom každý cyklus prináša malé zlepšenie softvéru. Systém je vyvíjaný prostredníctvom opakovaných cyklov (iterácií) a v menších častiach súčasne (inkrementálne). Výhodou je flexibilita, možnosť rýchlej implementácie nových zlepšení a funkcionalít a v prípade vážnej chyby jednoduchý „roll back“ do nedávnej verzie, ak je systém nefunkčný. Ak je plán nekompletný, môže to spôsobiť veľa iterácií navyše. Taktiež počas celého vývoja je potreba spolupráce klienta alebo prípadné testovanie každej novej verzie [40].



Obrázok 43 - Iteratívny model

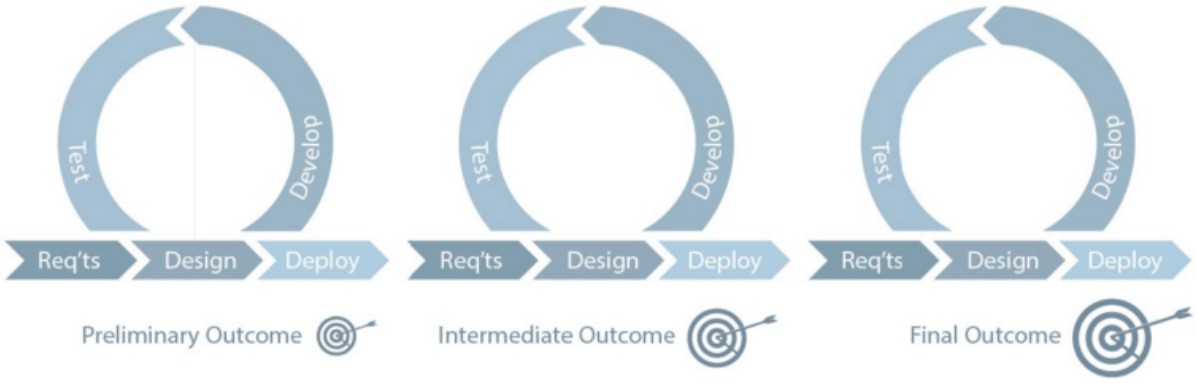
Špirálový model

Špirálový model kombinuje Iteratívny model procesu vývoja s prvkami Vodopádového modelu. Svojou flexibilitou je vhodný pre veľké a komplexné projekty. Po každej iterácii cez špirálu je systém o mieru kompletnejší a dokonalejší a je možné ho otestovať pomocou prototypu vytvoreného na základe danej iterácie. Prototyp umožňuje riadiť neznáme riziká po spustení projektu a taktiež umožňuje získať spätnú väzbu od zákazníka. Je relatívne drahý a komplexný a protokoly riadenia musia byť pevne dodržiavané. Taktiež počet opakovaných fáz je neznámy, takže time management je zložitý [41].

Obrázok 44 - Špirálový model

Agilný model

V súčasnosti je agilný model populárny pri softvérovom vývoji, zameriava sa na adaptívne plánovanie, samoorganizáciu a krátke dodacie lehoty. Je flexibilný, rýchly a jeho cieľom je neustále zlepšovanie kvality. Agilné metódy rozdeľujú úlohy na menšie iterácie a časti priamo nezahŕňajú dlhodobé plánovanie. Zrieka sa risku, že po mesiacoch a rokoch sa môže celý proces zrútiť kvôli malej chybe v začiatočných fázach [42].



Obrázok 45 - Agilný model

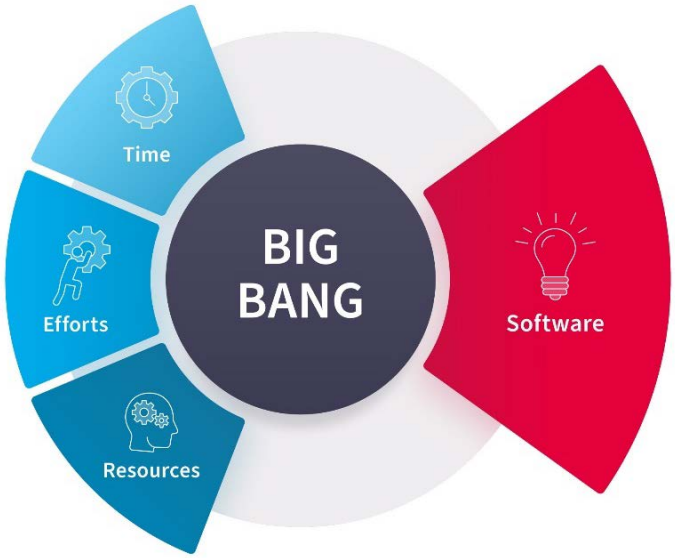
V rámci agilného vývoja rozoznávame rôzne agilné frameworky, ktoré možno definovať ako prístup alebo súbor techník a usmernení používaných na implementáciu agilnosti a udržiavanie jej hodnôt [43]. Medzi najznámejšie frameworky patria:

- Scrum – najrozšírenejší, tímovo orientovaný framework, ktorý využíva jasne definované tímové roly a zodpovednosti jednotlivých členov na implementáciu responzívneho štýlu agilného projektového manažmentu. Medzi scrum roly patria development tím (dizajnéri, programátori, testeri atď.), product owner, ktorý má víziu produktu a usmerňuje development tím a definuje priority a scrum master, ktorý pomáha product ownerovi definovať aktuálny ciel alebo úlohy, odkomunikovať ich development tímu a celkovo zabezpečuje aby Scrum fungoval správane. Scrum sa zameriava na optimalizáciu procesu vývoja [43, 44].
- Kanban – z japonského „kanban“, v preklade nástenka alebo tabuľa, je metóda založená na vizualizácii a manažovaní jednotlivých pracovných tokov a úloh. Každý člen dokáže prehľadne prezerať jednotlivé procesy a ich stav. Kanban sa zameriava na vizualizáciu procesu vývoja [43, 45].
- XP (Extreme Programming) – cieľom frameworku je rýchly vývoj kvalitného softvéru so schopnosťou prispôsobiť sa meniacim požiadavkám zákazníkov. XP sa zameriava na dodanie pridanej hodnoty [43, 46].

- FDD (Feature Driven Development) je metóda orientovaná na zákazníka, iteratívna a inkrementálna, s cieľom často a efektívne poskytovať hmatateľné softvérové výsledky [43, 48].
- Crystal – vhodné pre menšie tímy, ktoré uprednostňujú menej dokumentácie a reportingu, spoľah je na jednotlivých členov tímu, ktorí dokážu robiť kľúčové rozhodnutia aby bol projekt úspešný [43].
- DSDM - jej cieľom je pravidelné poskytovanie pridanej hodnoty a jasná komunikácia so zainteresovanými stranami. Sústreďuje sa na dodanie cieľov projektu včas a v rámci rozpočtu [43].

Big Bang Model

Na rozdiel od takmer všetkých ostatných populárnych modelov je model Big Bang jedinečný v tom, že nevyžaduje prakticky žiadne plánovanie, organizáciu a osvedčené postupy. Namiesto toho je projekt jednoducho spustený, bez formálnej štruktúry, rozvoja alebo organizácie. Vývoj prebieha podľa požiadaviek a cieľov, ktoré sú známe a aktuálne, ale vo všeobecnosti nie sú jasne definované a sú formulované počas vývoja. Vo všeobecnosti nie je braný ohľad na budúce požiadavky, systém je tvorený spontánne, podľa toho ako to vývojárov baví. Absencia plánovania je vhodná pre malé, nízkorizikové projekty a pre začínajúcich vývojárov, ktorý sa chcú rovno vrhnúť na písanie kódu. Samotný model je veľmi riskantný a aj keď má miesto v určitých projektoch, bez plánovania, formálneho vedenia, nedodržiavania štandardov sa môže objaviť masívny problém, ktorý môže spôsobiť celkové zlyhanie aktuálneho riešenia [48].



Obrázok 46 - Big bang model

4 Výsledky práce

Kapitola je zameraná na implementáciu a testovanie rôznych spôsobov optimálneho prechádzania priestoru robotom. V rámci tvorby informačného systému robota budú popísané aj dané algoritmy, tvorené v programovacom jazyku C++, ktoré riadia činnosť robota a sú uložené v pamäti mikropočítača.

4.1 Tvorba informačného systému robota

Tvorba tejto diplomovej práce a informačného systému robota najviac čerpala praktiky a postupy z vodopádového modelu popísaného v predošlej kapitole. Avšak v rámci krokov návrh – testovanie boli prítomné prvky iteratívneho vývoja. Dôležitá časť procesu tvorenia je učenie sa praxou a reflektovanie. V rámci testovania treba reflektovať na časti projektu, ktoré fungujú a ako ich je možné vylepšiť. Takýmto štýlom je zdokonaľovaný samotný projekt ale aj schopnosti a zručnosti tvorcu. Diplomová práca sa charakterom podobá na moju bakalársku prácu, v zásade ide o novšiu iteráciu respektíve vylepšenie už existujúceho systému. Aj keď je práca tvorená počas obdobia trvajúceho viac ako jeden rok, stále ide o relatívne malý projekt. Konkrétne ide o tieto kroky:

- *Plánovanie* – abstraktný krok, počas ktorého bola formovaná vízia konečného produktu, jednotlivé kroky na uskutočnenie a časový harmonogram.
- *Analýza, zber požiadaviek* - pozostávala najmä z analýzy existujúcich systémov, ktoré položili základ požiadaviek pre tvorbu vlastného systému.
- *Dizajn, návrh systému* – prvá fáza je študovaná a na jej základe je navrhnutých niekoľko možných riešení. Problémy sú rozdelené na menšie časti a pre každý je popísaných pár možností. Návrh pomáha definovať technické a systémové požiadavky.
- *Implementácia* – na základe návrhu sú tvorené prvé časti kódu a funkcie. Každé z týchto funkcií sú osobitne testované.
- *Integrácia a testovanie* – vytvorené funkcie sú integrované do systému ako celok a sú testované podľa definovaných hodnotiacich kritérií. Testovanie poskytuje nové poznatky, informácie a podnety na vylepšenie systému a nové návrhy.
- *Nasadenie* – v prípade diplomovej práce odovzdanie.

Treba poznamenať, že modely vývoja sú bežnejšie v tímových projektoch, zatiaľ čo diplomová práca je tvorená len jedným autorom, ktorý v tvorbe informačného systému

zastupuje ako aj programátora alebo testera, tak aj manažéra alebo vlastníka produktu, ktorý má víziu končeného projektu.

Plán, overené praktiky a postupy sú dôležité pri tvorbe hocijakého projektu, dôležitým prvkom je ale aj produktivita. Produktivita je vykonanie čo najväčšieho pokroku za najkratší možný čas. Jedným z konceptov, ktorým sa v práci venujem je optimalizácia. Tak ako sa dá optimalizovať cesta robota, dá sa optimalizovať aj práca človeka na určitom projekte. Produktivitu je možné definovať rovnicou:

$$d=v*T=D*Y*L*n*t; \quad (2)$$

kde *d* je pokrok,

v je rýchlosť,

T je čas,

D je smer,

Y je dĺžka pokroku,

L je rytmus,

n je počet šprintov,

t je trvanie šprintov.

Pokrok je merateľný výstup činnosti (pri písaní je to počet strán, pri testovaní počet otestovaných funkcionalít, atď.). Dá sa zvýšiť časom *T* alebo rýchlosťou vykonania činnosti *v*. Čas *T* sa dá rozložiť na počet šprintov *n* a trvanie šprintov *t*. Šprint je v tomto kontexte ohraničené obdobie, počas ktorého činnosť vykonávame s intenzívnym sústredením a vedomou participáciou, ktoré tlačíme na maximálnu hranicu. Počet a trvanie šprintov chceme maximalizovať ale musíme dbať na udržateľnú konzistenciu. Ak sa jeden deň príliš vyťažíme, druhý deň pokrok klesá a tým aj konzistencia alebo dokonca môže nastať aj syndróm vyhorenia. Dôležitou časťou sú prestávky medzi šprintami; ak človek pracuje nepretržite niekoľko hodín, produktivita výrazne klesá. Prestávky predstavujú aj formu odmeny a zresetujú sústredenie pred novým šprintom. Rýchlosť vykonania činnosti *v* sa dá rozložiť na dva hlavné komponenty; smer *D* a pokrok *YL*. Než začneme šprintovať, potrebujeme vedieť smer. Smer môžeme definovať ako aj cieľ nejakej práce alebo projektu. Správny smer je často kritickejší ako intenzita šprintu. Pokrok je konkrétna akcia, ktorou sa efektívne posunieme bližšie k cieľu, dá sa rozdeliť na dĺžku pokroku *L* a rytmus *Y*. Dĺžka pokroku je definovaná schopnosťou človeka vykonávať danú aktivitu. Pri nových aktivitách je dĺžka pokroku malá, ale ako človek trénuje, dĺžka sa zvyšuje. Dĺžka pokroku sa dá zvýšiť aj imitovaním práce profesionálov v danej aktivite. Rytmus je schopnosť kontinuálne pracovať bez rozptylov

z okolia. Rozptyly môžu byť napríklad pípajúci telefón, neporiadok na pracovnom stole, iný ľudia atď. [49]. Optimalizovaná práca teda pozostáva zo šprintov s prestávkami, počas ktorých pracujeme zručne, kontinuálne a sústredene, tým správnym smerom k cieľu.

4.2 Návrh riešenia

Na základe analýzy z prvej kapitoly a rôznych metód spracovania z tretej kapitoly sú navrhnuté konkrétne spôsoby ako možno implementovať a zrealizovať hľadanie cesty, pričom by robot mal naplniť tieto predpoklady:

- Prejsť celú plochu,
- Nájsť najkratšiu cestu,
- Vyhýbať sa prekážkam,
- Neprechádzať už prejdenú cestu,
- Otáčanie zaberá čas a energiu, rovné čiary sú efektívnejšie.

Ako je popísané v kapitole 3.1, prvým krokom je tvorba grafu, ktorý v mojom prípade bude reprezentovať abstrakciu prostredia, v ktorom robot bude pracovať. Prostredie, respektíve graf môže byť dopredu známy alebo neznámy. Ak je prostredie neznáme, robot musí vykonať špeciálny krok pri prvom spustení aby prostredie zmapoval. Tento krok môžeme nazvať aj ako počiatočná kalibrácia, ktorú vykonávali niektoré z analyzovaných zariadení. Môže byť dosiahnutá manuálne aj autonómne. Jedným z riešení je použitie ovládača, respektíve analógovej páčky na prejdenie terénu, ktorého hrany sa postupne budú ukladať do pamäte robota ako časové jednotky alebo jednotky vzdialenosti od počiatočného bodu. Graf prostredia môže byť aj známy, teda dopredu definovaný a uložený v počítačovej pamäti. V tomto prípade je možné informácie o pozícii samotných bodov ukladať relatívne alebo absolútne. Absolútne vo forme GPS súradníc (pretože sú pre každý bod pevne dané) alebo relatívne, vzdialenosťou od počiatočného bodu. Bolo však skonštatované, že GPS je pre takto malé zariadenie až príliš nepresné, preto volím relatívne ukladanie bodov, pomocou vektorov. Nasleduje samotný algoritmus činnosti, pomocou ktorého by malo zariadenie optimálnym spôsobom prejsť celé prostredie. Prvé dva navrhnuté spôsoby budú určené pre neznáme prostredie a budú vyžadovať počiatočnú kalibráciu, ktorú vykoná užívateľ pomocou analógovej páčky. Ďalšie dva spôsoby budú čerpať poznatky z teórie grafov a v dopredu známom grafe vyhľadajú optimálnu cestu. Osobitnou skupinou bude spôsob orientovania z bakalárskej práce, ktorý v tomto prípade bude slúžiť len ako porovnanie. Tieto pomenované

spôsoby hľadania optimálnej cesty budú implementované a následne v ďalšej kapitole porovnané:

- *CopyCat* – pomocou analógovej páčky prejdeme cestu cez prostredie, robot si túto cestu bude ukladať do pamäte a po stlačení tlačidla ju zopakuje.
- *BorderLine* – pomocou analógovej páčky prejdeme dve hranice prostredia, ktoré reprezentujú dve hrany obdĺžnika a zariadenie pomocou algoritmu prejde prostredie tak, aby minimalizovalo otáčanie; bude prechádzať hore dole pozdĺž dlhšej hrany.
- *Fleuryho algoritmus* – v pamäti mikropočítača bude uložený sieťový graf, ktorý obsahuje Eulerovský sled, algoritmus tento cyklus alebo cestu nájde a zariadenie túto cestu zrealizuje v reálnom prostredí.
- *Primov algoritmus* – v pamäti mikropočítača bude uložený ohodnotený sieťový graf v ktorom algoritmus nájde minimálnu kostru, ktorá bude reprezentovať cestu, ktorú zariadenie zrealizuje v reálnom prostredí.

Oddelenú kategóriu tvorí metóda *Random* – takto robot funguje pôvodne, neorientovaná navigácia, iba jazdí hore dole a vyhýba sa prekážkam.

4.3 Realizácia

Na základe piatich konkrétnych návrhov z predchádzajúcej kapitoly budú tvorené a implementované programy do zariadenia v jazyku C++ vo vývojovom prostredí *Arudino IDE*. Každý z daných programov je kompilovateľný a funguje na mojom zariadení. Niektoré z nich sú primitívnejšie, niektoré sú pokročilé a využívajú aj heuristiky. V nasledujúcich riadkoch je vysvetlená implementácia a činnosť algoritmov.

CopyCat

Z názvu je očividné, že robot bude fungovať tak, že bude replikovať cestu, ktorú navrhne používateľ, podobne ako napríklad analyzované zariadenie *Spot*. Je teda potrebné čítať vstup z ovládacieho prvku, konkrétne z analógovej páčky. Páčka obsahuje dva potenciometre, ktoré podľa polohy pre obidve súradnice, či už v *x*-ovej alebo *y*-ovej, posielajú výstup od 0 po 1023. Tieto hodnoty je možné mapovať na rozmedzie prípustné pre udávanie rýchlosti motorov, od 0 po 254. Pre smer dopredu, v neutrálnej polohe. Páčka posiela hodnoty na *x*-ovej súradnici 511, pri postupnom posúvaní dopredu posiala hodnoty do 1023.

```
poz = Map(hodnota, z_min, z_max, na_min, na_max); //vysvetlenie
poz = Map(getData(), 511, 1023, 0, 254);
```

Moja súčiastka je však lacná a nekvalitná a namerané hodnoty sú nepresné, preto nameranie určitej hodnoty beriem ako obyčajný pohyb dopredu v jednotnej rýchlosti.

Zaznamenaný pohyb ukladá ako čísla do listu pomocou metódy .addLast(), ktorý je možné používať vďaka zahrnutej knižnici. Po stlačení tlačidla na analógovej páčke je list čítaný od začiatku po jeho dĺžku pomocou metód .getSize() a .getValue() a na základe hodnôt sú volané funkcie na pohyb, ktoré sú prebrané prvotného riešenia v bakalárskej práci (návrh *Random*).

```
#include <List.hpp>
List<int> smer;
```

Metódy sú odlišné ako pri klasickej C++ knižnici, bolo potrebné si naštudovať dokumentáciu. Napríklad .addLast() v Arduino <List.hpp> knižnici je ekvivalent .push_back() v C++ <list> knižnici.

```
if((xP>=250) && (xP<=630)) {
  stoj();
  if(yP<100) {
    otocDoprava();
    int pom = 1;
    smer.addLast(pom);
  }
  if(yP>650) {
    otocDolava();
    int pom = 2;
    smer.addLast(pom);
  }
}
if(xP<500 && xP>0) {
  rychlost = 255; //255
  Serial.print("Rychlost dopredu: ");
  Serial.println(rychlost);
  chodDopredu();
  int pom = 0;
  smer.addLast(pom);
}
```

Obrázok 47 - Na základe čítaného vstupu sú volané funkcie na pohyb

```
if (!SW_state) {
  for (int i = 0; i < smer.getSize(); i++) {
    Serial.print(smer.getValue(i));
    if(smer.getValue(i) == 0) {
      chodDopredu();
      delay(50);
    } else if (smer.getValue(i) == 1) {
      otocDoprava();
      delay(50);
    } else if (smer.getValue(i) == 2) {
      otocDolava();
      delay(50);
    }
  }

smer.clear();
}
```

Obrázok 48 - Po stlačení tlačidla je čítaný list kde sú uložené inštrukcie na pohyb a sú vykonávané

BorderLine

BorderLine metóda je založená na súradnicovom prístupe, ktorý je popísaný v kapitole 3.1. Prebieha v dvoch krokoch, prvou je počiatočná kalibrácia kedy prejdeme priamu rovnú cestu, ktorá reprezentuje prvú hranu obdĺžnikovej oblasti, ktorá bude celá opracovaná. Stlačíme tlačidlo na analógovej páčke a zariadenie sa otočí doprava o 90 stupňov. Prejdeme ďalšiu hranu. Znova stlačíme tlačidlo a algoritmus dokončí prácu tak, že bude v cykloch prechádzať označenú plochu hore dole pozdĺž dlhšej hrane. V premennej *x* je údaj, koľko krát bola volaná funkcia na pohyb dopredu, ak je to viac ako *y*, zariadenie bude prechádzať cestu v čase *del* v cykloch hore dole pozdĺž tejto dlhšej hrany. 100 časových jednotiek v tomto prípade reprezentuje za aký čas je prejdená vzdialenosť šírky zariadenia. Posielaním tohto údaju do funkcií na pohyb minimalizujeme prechádzanie cez už prejdenú cestu. Ak je druhá

nameraná hrana dlhšia, zariadenie sa otočí dvakrát doprava, prejde cestu a potom sa pri párnej iterácii otočí doľava, pri nepárnej doprava.

```
void klik()
{
    if(x>y) {
        int del = (x - 100) * 100;
        otocDopravaDvakrat(100);
        for(int i = 0; i < (y/100);i++) {
            if(i % 2 == 0) {
                otocDolavaDvakrat(100);
            } else {
                otocDopravaDvakrat(100);
            }
            chodDopredu(del);
        }
    } else if (y>x) {
```

Obrázok 49 - Cyklicky vykonávaná cesta hore dole *y*-krát pozdĺž *x*

Fleuryho algoritmus

Predpokladom je, že v pamäti je uložený graf.

```
//trieda ktorá reprezentuje graf
class Graph {
    int V; //počet vrcholov
    List<int>* adj; //dynamický list
public:
    //konštruktor a deštruktor
    Graph(int V)
    {
        this->V = V;
        adj = new List<int>[V];
    }
    Graph() { delete[] adj; }

    //funkcia na ukladanie vrcholov
    void addEdge(int u, int v)
    {
        //Serial.println("pridavam hrany");
        adj[u].addLast(v);
        adj[v].addLast(u);
        //Serial.println(adj[u].getValue(0));
    }
```

Obrázok 50 - Trieda ktorá reprezentuje graf

```
Graph g2(5);
g2.addEdge(0, 1);
g2.addEdge(1, 4);
g2.addEdge(2, 4);
g2.addEdge(1, 2);
g2.addEdge(1, 3);
g2.addEdge(0, 3);
g2.printEulerPath(0);
```

Obrázok 51 - Pridanie hrán do grafu

Hneď na začiatku je volaná funkcia setPriority(), ktorá prioritné hrany posunie na začiatok listu metódou .addFirst(). Pomáha pri tom matica priorít, kde číslo 1 značí to, že cesty medzi týmito vrcholmi sú priame. Týmto spôsobom minimalizujeme nadmerné otáčanie a je podporovaná priama, rovná cesta medzi vrcholmi.

```
void Graph::setPriority() {
  //pre každý prvok z 2D listu kde je uložený graf
  for(int i = 0; i < V; i++) {
    int s = adj[i].getSize();
    for(int j = 0; j < s; j++) {
      int pom = adj[i].getValue(j);
      //najdi prioritu pre daný vrchol z matice priorit
      int priorita = costM[i][pom];
      //ak je priorita 1, odstráň vrchol a daj ho na začiatok
      if(priorita == 1) {
        rmvEdge(i, j);
        addPriorityEdge(i, j);
      }
    }
  }
}
```

Obrázok 52 - Funkcia setPriority()

Prvotný bod je manuálne zvolený a poslaný ako parameter *u* pri volaní funkcie printEulerPath().

```
void Graph::printEulerPath(int u)
{
    //usporiadaj podla priorit
    setPriority();
    //pre každý vrchol v ktorý s u tvorí hranu
    for(int i = 0; i < adj[u].getSize(); i++) {
        int v = adj[u].getValue(i);
        //kontrola či je hrana validna
        if (isValidNextEdge(u, v)) {
          Serial.print(u); Serial.print(" - "); Serial.println(v);
            rmvEdge(u, v);
            printEulerPath(v);
        }
    }
}
```

Obrázok 53 - Funkcia printEulerPath()

Vo funkcii isValidNextEdge() zisťujeme či hrana nie je mostom. Zistíme počet susedných vrcholov *u* a ak existuje len jeden susedný vrchol, vyberieme hranu, ktorá k vrcholu vedie ako časť cesty. Ak existuje viac ako jeden susedný vrchol, považujeme ich za susedné len vtedy, ak hrana *u* – *v* nie je mostom (most v tomto prípade reprezentuje hranu medzi prejdenou cestou a zbytkom grafu, ktorý by bol odstrihnutý a Eulerovský sled by bol nekompletný).

```
bool Graph::isValidNextEdge(int u, int v) {
    //u-v je validná ak:
    //1) existuje iba jedna cesta z u
    int count = 0;
    if (adj[u].getSize() == 1)
      return true;
    //ak je viacero ciest, zistit ktorá nie je most
    //2.a) spočítaj počet vrcholov ktoré je možné navštívit z u
    bool visited[V];
    memset(visited, false, V);
    int count1 = DFSCount(u, visited);
    //2.b) odstráň u-v a znovu spočítaj počet vrcholov
    rmvEdge(u, v);
    memset(visited, false, V);
    int count2 = DFSCount(u, visited);
    //2.c) pridaj hranu späť
    if(costM[u][v] == 1)
      addPriorityEdge(u, v);
    else
      addEdge(u, v);
    //ak count1 je väčší, u-v je most
    if(count1 > count2) {
      return false;
    } else {
      return true;
    }
}
```

Obrázok 54 – Funkcia isValidnextEdge()

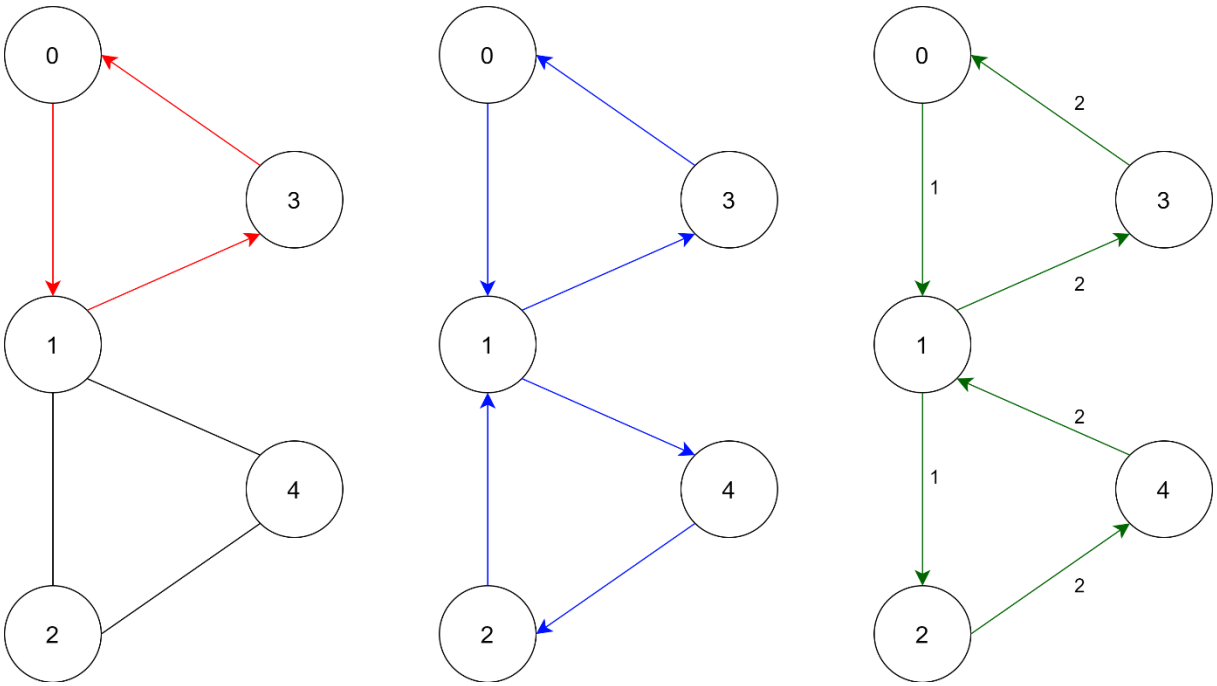
Pomocou heuristickej metódy spočítame počet vrcholov ku ktorým je možné sa dostať od aktuálneho vrcholu. Rekurzívna funkcia DFSCount() vráti počet vrcholov prístupných z *u*. Následne daný most *u* – *v* odstránime. Ak sa zníži počet nájdených vrcholov, potom okraj *u* – *v* je most (táto hrana, most, by zneprístupnil časť grafu, kde sú neprejdené hrany).

```
int Graph::DFSCount(int v, bool visited[]) {
    //Serial.print("zacinam dfs, start node: "); Serial.println(v);
    //označ aktuálny vrchol ako navštívený
    visited[v] = true;
    int count = 1;
    /*for(int j = 0; j < V; j++)
      Serial.print(visited[j]);*/
    //rekurzívne volanie pre všetky hrany z tohto vrcholu
    for (int i = 0; i < adj[v].getSize(); i++)
        if (!visited[adj[v].getValue(i)])
            count += DFSCount(adj[v].getValue(i), visited);

    return count;
}
```

Obrázok 55 - Funkcia DFSCount()

Výsledkom je sekvencia vrcholov cez ktoré zariadenie prejde a vykoná tak Eulerovský sled pričom si vyberá priame cesty na miesto nadmerného otáčania. Zariadenie cestu prechádza pomocou vektorov, volaním funkcií na pohyb s parametrami, aby sa dostalo do želaného vrcholu. Na obrázku 56 je znázornené ako by algoritmus fungoval bez heuristiky, červenou cestou by prešiel z vrcholu 0 do 1 a vybral by si most 1 – 3, ktorý oddeľuje zbytok grafu a vrátil by sa do počiatočného vrcholu. S heuristikou prejde modrú cestu 0 – 1 – 4 – 2 – 1 – 3 – 0. Táto cesta je Eulerovský cyklus ale zariadenie sa otočilo päť krát. Zelenú cestu prejde s prioritami a otočí sa štyri krát (jedna otočka navyše môže byť takmer zanedbateľná ale v rámci optimalizácie veľkých grafov reprezentujúcich veľké prostredia, je takýto prístup prívetivý). Inšpiroval som sa programom na internete [29].



Obrázok 56 - Verzie Fleuryho algoritmu zľava doprava bez heuristiky, s heuristikou a s prioritami

Primov algoritmus

Predpokladom je uložená matica susednosti G[][]v pamäti mikropočítača. Pole Selected[] obsahuje zoznam vrcholov, ktoré boli navštívené. Postupne prechádzame susednými vrcholmi zvoleného vrcholu a vyberieme suseda s hranou, ktorá má najmenšiu hmotnosť pričom kontrolujeme či vrchol, ktorý tvorí hranu nie je v Selected[], predchádzame tak tvoreniu cyklu. Inšpiroval som sa programom na internete [50].

```
//zvol vrchol 0 a nastav na true
selected[0] = true;
//vypíš hranu a jej cenu
while (no_edge < V - 1) { //číslo hrany, musí ich byť menej než všetky vrcholy - 1
  //pre každý vrchol v sete najdi susedné vrcholy,
  //vypočítaj vzdialenosť od prvotného vrcholu
  //ak je vrchol zvolený, vylúč ho a
  //najdi ďalší najbližší vrchol od prvotného vrcholu

  int min = INF; // velké číslo reprezentujúce nekonečno
  x = 0;
  y = 0;

  for (int i = 0; i < V; i++) {
    if (selected[i]) { //pole pre zvolené vrcholy, pôvodne naplnené false
      for (int j = 0; j < V; j++) {
        if (!selected[j] && G[i][j]) {  // nezvolený a existuje
          if (min > G[i][j]) { //najdi ten najlacnejší
            min = G[i][j];
            x = i;
            y = j;
          }
        }
      }
    }
  }
  //vypíš zvolenú hranu, jej cenu, označ ako zvolený a iteruj počet hrán
  Serial.print(x); Serial.print(" - "); Serial.print(y); Serial.println(G[x][y]);
  selected[y] = true;
  no_edge++;
}
```

Obrázok 57 - časť celej funkcie na hľadanie minimálnej kostry grafu pomocou Primovho algoritmu

Random

Random bol dopodrobna popísaný v bakalárskej práci [1]. Treba poznamenať že tento spôsob je viac orientovaný na vyhýbanie sa prekážkam a nie na optimálne prechádzanie cesty, ako je cieľom tejto práce. *Random* ale uvádzam aj tak, pretože môže byť zaujímavé ho porovnať s novými metódami.

4.4 Vyhodnotenie

Navrhnuté spôsoby orientácie a hľadania cesty sú otestované v kontrolovanom priestore a na základe hodnotiacich kritérií posúdené z rôznych aspektov. Budem hodnotiť hodnotou od 0 po 1, relatívne k ostatným spôsobom (ak nejaký spôsob orientácie perfektne spĺňa dané kritérium, dostane 1, ostatné spôsoby sú ohodnotené tomto základe). Hodnotiace kritéria sú:

- Interaktivita používateľa pri prvotnom spustení - ako výrazne musí zasahovať užívateľ keď prvý krát spustí robota (definovanie grafu alebo hraníc),
- Interaktivita používateľa pri opakovaných spusteniach – ako výrazne musí užívateľ zasahovať po prvotnej kalibrácii (po prvom spustení môžu byť dáta uložené v pamäti),
- Časová efektívnosť – koľko trvá opracovanie plochy,
- Množstvo senzorov použitých na orientovanie – viac senzorov môže zvýšiť presnosť orientácie, ale zároveň a komplexnosť a cenu,
- Úplnosť prejdenia plochy – pri sieťových grafoch to bude 100% ale pri *Random*, prípadne iných technikách to 100% nemusí byť,
- Pamäťová zložitosť – koľko pamäte zaberá kód a premenné (zobrazuje sa pri kompilácii).

Tabuľka 2 - Hodnotenie kritérií pre spôsoby navigácie

Orientácia/ kritériá	1. Interakcia	n. Interakcia	Časová efektívnosť	Senzory/ moduly	Úplnosť	Pamäťová zložitosť	Výsle-dok
CopyCat	0,5	1	0,5	0,6	1	0,74	4,34
Border-Line	0,7	0	0,8	0,6	1	1	4,10
Fleuryho algoritmus	0	1	1	1	1	0,74	4,74
Primov algoritmus	0	1	0,2	1	0	1	3,2
Random	1	1	0	0	0	0	2

5 Diskusia

Optimalizácia cesty a najmä dynamická tvorba grafu a reprezentácia prostredia je komplexným problémom. V predchádzajúcej kapitole boli otestované a vyhodnotené viaceré možné spôsoby navigácie prostredím. Z navrhnutých spôsobov uchovania grafu v pamäti som vybral dynamický list. Najlepšie dopadla metóda hľadania cesty pomocou *Fleuryho algoritmu*. Jedinou nevýhodou je, že graf s Eulerovským sledom musí byť dopredu definovaný v pamäti mikropočítača. S mojim vylepšením, ktoré pridáva prioritu hranám potom zariadenie prejde optimálnym spôsobom cez všetky hrany grafu, pričom uprednostňuje rovnú, priamu cestu. Cesta bola uskutočnená manuálne pomocou vektorov. Na druhom mieste skončila metóda *CopyCat,* ktorou výhoda je, že užívateľ pri prvom spustení navrhne svoju cestu, ktorá môže byť suboptimálna, ale sám si prejde všetky miesta tak, ako chce aby robot išiel. Cesta je uložená v pamäti a pri každom spustení je táto cesta zopakovaná bez ďalšej interakcie. *BorderLine* metóda skončila na treťom mieste, má však vysoký potenciál. Je možné na tejto myšlienke stavať, užívateľ by mohol prejsť celé hranice prostredia, akokoľvek komplexné a vylepšený algoritmus by v rámci tejto plochy našiel optimálnu cestu. *Primov Algoritmus* nie je najvhodnejší na prechádzanie všetkých ciest, do všetkých bodov sa síce dostaneme, no musíme sa vracať zo slepých uličiek, aby sme prešli všetky vetvy minimálnej kostry grafu. Aplikácia pre robota, ktorý chce prejsť celú plochu je nevhodná. Podľa hodnotenia najhoršie dopadla metóda *Random* a to z toho dôvodu, že prvotne bola navrhnutá len na vyhýbanie sa prekážkam, nie na optimálne prechádzanie cesty. Má aj najhoršie skóre v oblasti počtu modulov, lebo používa tri ultrazvukové moduly a dva nárazníky, ostatné metódy nepoužívajú tieto metódy, alebo len analógovú páčku. Výhodou tejto metódy však je, že zariadenie netreba kalibrovať v priestore, po zapnutí jednoducho začne pracovať bez ďalšej potrebnej interakcie od používateľa. Nastavovanie a definovanie hraníc je náročné, takže v rámci tohto kritéria, kde všetky ostatné metódy strácajú body, *Random* vyhráva.

Cena komerčných autonómnych kosačiek alebo vysávačov, ktorá môže byť viac ako desaťnásobná oproti mojim nákladom na postavenie robota, je opodstatnená. Na ich tvorbu navrhujú šikovní inžinieri robustné algoritmy a využívajú precízne súčiastky. Mapa prostredia je v súčasnosti týmito zariadeniami tvorená optickou kamerou, ktorá za pomoci umelej inteligencie rozoznáva prekážky a kombináciou rôznych metód prechádza prostredie optimálnym spôsobom. V mojej bakalárskej a diplomovej práci som však ukázal, že takéto zariadenie si môže postaviť aj bežný človek a naprogramovať ho tak, aby prostredím prechádzalo optimálnou cestou.

Záver

V diplomovej práci som navrhol a otestoval niekoľko algoritmov, pomocou ktorých zariadenie naviguje cez prostredie optimálnym spôsobom. Pred tým, než boli tieto algoritmy tvorené, bolo potreba analyzovať existujúce systémy a roboty, ktoré využívajú svoje technické prostriedky a určitým spôsobom optimálne prechádzajú prostredie a tak napĺňajú svoje ciele. Boli analyzované rôzne zariadenia, ktoré prechádzajú celú plochu, alebo navštevujú len určité body v rámci pracovného prostredia. Analyzované boli aj abstraktné systémy ako hry, kde postavy tiež hľadajú optimálnu cestu, aby naplnili svoj cieľ.

Po stanovení cieľa, boli v tretej kapitole rozpísané rôzne metódy a nástroje, ktoré mohli byť použité pri tvorbe informačného systému robota a na naplnenie definovaných cieľov v tejto diplomovej práci. V prvej časti bolo popísané, ako je možné reprezentovať prostredie pomocou rôznych typov grafov. Dôležitou časťou bola analýza poznatkov z teórie grafov, teda aké rôzne algoritmy hľadania cesty existujú a ako fungujú. Ďalej boli navrhnuté spôsoby ukladania grafov v počítačovej pamäti. V kapitole sú tiež naznačené metódy tvorby informačného systému a prostriedky na tvorbu a kompiláciu programu a aj ďalšie rôzne technické prostriedky na vylepšenie robota.

Vo štvrtej, najdôležitejšej kapitole, som popísal, ako som tvoril informačný systém zariadenia. Keďže sa venujem optimalizácií, v rámci vývoja informačného systému robota som venoval pár riadkov tomu, ako optimalizovať svoj čas, spôsob a prístup pri práci alebo tvorbe nejakého projektu. Na základe analýzy existujúcich systémov a poznatkov z teórie grafov som navrhol niekoľko spôsobov optimálnej navigácie v prostredí. Niektoré návrhy boli môj vlastný nápad, niektoré boli inšpirované existujúcimi algoritmami, ktoré sú prístupné na internete. Tieto algoritmy som upravil, aby bola možná kompilácia na mikropočítači a pri jednom som pridal vlastné vylepšenie. Všetky sú vo všeobecnosti popísané, dôležité časti programov sú znázornené fotografiou z obrazovky a bližšie vysvetlené. Súčasťou bolo hodnotenie na základe kritérií.

Na základe hodnotenia podľa definovaných kritérií, som v kapitole diskusia slovne zhodnotil, ako výkonné boli jednotlivé implementované spôsoby navigácie. Podobne ako bakalárska práca slúžila ako podklad na vylepšenie systému, ktoré som uskutočnil v tejto práci, daná problematika je stále dostatočne obšírna, komplexná a zaujímavá, že aj táto práca môže slúžiť ako podklad pre ďalšie vylepšenia systému, prípadne ako príručka na stavbu svojho vlastného zariadenia, ktoré má navigovať cez prostredie optimalizovaným spôsobom.

Výsledok práce predstavuje informačný systém v podobe autonómneho zariadenia, ktoré využíva rôzne algoritmy na optimálne prechádzanie priestoru. Časťou práce sú aj zdrojové kódy v prílohe, podľa ktorých zariadenie funguje a je možné ich nahrať na mikropočítač. Celá diplomová práca odzrkadľuje tvorbu informačného systému, od analýzy, návrhu a popisu tvorby.

Použitá literatúra

[1] M. Jaroš, “Autonómny robot na kosenie trávy: bakalárska práca”, *Ekonomická Univerzita v Bratislave*, 2020. [Prezerané 17.11.2020].

[2] “Spot® - The Agile Mobile Robot,” *Boston Dynamics.* [Online]. Dostupné na: https://www.bostondynamics.com/products/spot. [Prezerané: 22.1.2021].

[3] L. Cao, “How Does Spot Work? The Robot That Compares Design to Reality at the Construction Site,“ *ArchDaily*, 14.1.2021. [Online]. Dostupné na: https://www.archdaily.com/954784/how-does-spot-r-work-the-robot-that-compares-design-to-reality-at-the-construction-site. [Prezerané: 4.2.2021].

[4] “Getting started with Autowalk,“ *Boston Dynamics*, 23.2.2022. [Online]. Dostupné na: https://support.bostondynamics.com/s/article/Getting-Started-with-Autowalk. [Prezerané: 5.3.2022].

[5] Tom Scott, “An actual, real-world use for robot dogs,“ *YouTube*, 1.11.2021. [Online]. Dostupné na: https://youtu.be/PkW9wx7Kbws. [Prezerané 4.12.2021].

[6] M. Urban, “Xiaomi Predstavilo 4 nové Robotické Vysávače Série vacuum mop 2,” *Techbyte*, 1.11.2021. [Online]. Dostupné na: https://www.techbyte.sk/2022/01/xiaomi-predstavilo-4-nove-roboticke-vysavace-serie-vacuum-mop-2/. [Prezerané: 14.1.2022].

[7] B. Bennett, “This is why your Roomba's random patterns actually make perfect sense,“ *Cnet*, 28.7.2021. [Online]. Dostupné na: https://www.cnet.com/home/kitchen-and-household/this-is-why-your-roombas-random-patterns-actually-make-perfect-sense/. [Prezerané: 20.2.2022].

[8] T. Qin, P. Li, a S. Shen, “Vins-mono: A robust and versatile monocular visual-inertial state estimator,” *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018. DOI: 10.1109/TRO.2018.2853729. [Prezerané: 20.2.2022].

[9] Tom Scott, “How many robots does it take to run a grocery store?, “ *YouTube*, 5.7.2021. [Online]. Dostupné na: https://youtu.be/ssZ_8cqfBlE. [Prezerané: 5.7.2021].

[10] “Cross Border E-commerce Warehouse,“ *Hai Robotics*, 30.11.2021. [Online]. Dostupné na: https://www.hairobotics.com/en/case_study/Case_StudyDetails?id=2c9268d87ba0e8d7017d4fe53a5902cd. [Prezerané: 4.2.2022].

[11] O. Mitchell, “Inside Israel’s growing construction robotics ecosystem,“ *Therobotreport*, 7.12.2020. [Online]. Dostupné na: https://www.therobotreport.com/israels-growing-construction-robotics-ecosystem. [Prezerané: 8.2.2022].

[12] S. Azhar, “Building Information Modeling (BIM): Trends, benefits, risks, and challenges for the AEC Industry,” *Leadership and Management in Engineering*, vol. 11, no. 3, pp. 241–252, Jun. 2011. DOI: https://doi.org/10.1061/(ASCE)LM.1943-5630.0000127. [Prezerané: 8.2.2022].
[13] T. Stone, “WORLD FIRST: High-power electric, autonomous street sweeper launched,“ *iVTinternational*, 7.10.2020. [Online]. Dostupné na: https://www.ivtinternational.com/news/autonomous-vehicles/world-first-high-power-electric-autonomous-street-sweeper-launched.html. [Prezerané: 20.3.2022].
[14] Z. Abd Algfoor, M. S. Sunar, and H. Kolivand, “A comprehensive study on pathfinding techniques for Robotics and video games,” *International Journal of Computer Games Technology*, vol. 2015, pp. 1–11, 2015. DOI: https://doi.org/10.1155/2015/736138. [Prezerané: 18.11.2020].
[15] D. Russell a J. M. Laffey, “Navigating Through Virtual Worlds: From Single Characters to Large Crowds,” *Handbook of Research on gaming trends in P-12 education*, Hershey, Pennsylvania (701 E. Chocolate Avenue, Hershey, PA 17033, USA): IGI Global, 2016, pp. 527–554. DOI: 10.4018/978-1-4666-9629-7. [Prezrané: 18.11.2020].
[16] Game Maker's Toolkit, “What Pac-Man Brought to Game Design | Design Icons,“ *YouTube*, 16.10.2019. [Online]. Dostupné na: https://youtu.be/S4RHbnBkyh0. [Prezerané: 19.11.2020].
[17] M. Soták, “INERCIÁLNY NAVIGAČNÝ SYSTÉM V SIMULINKU“, *Akadémia ozbrojených síl gen. M. R. Štefánika*, [Online]. Dostupné na: http://dsp.vscht.cz/konference_matlab/matlab09/prispevky/093_sotak.pdf. [Prezerané: 21.3.2022].
[18] Vacuum Wars, “Lidar vs Vslam (cameras vs lasers) For Robot Vacuums - Which One is Best?,“ *YouTube*, 17.12.2019. [Online]. Dostupné na: https://youtu.be/5O8VmDiab3w. [Prezerané: 20.1.2022].
[19] A. Moscaritolo, “Ecovacs Deebot Ozmo T8 AIVI Review,“ *pcmag*, 2.10.2020. [Online]. Dostupné na: https://www.pcmag.com/reviews/ecovacs-deebot-ozmo-t8-aivi. [Prezerané: 20.1.2022].
[20] “Spot® Payloads,“ *Boston Dynamics*. [Online]. Dostupné na: https://www.bostondynamics.com/payloads. [Prezerané: 24.2.2022].
[21] A. Felder, D. Van Buskirk, and C. Bobda, “Automatic generation of waypoint graphs from distributed ceiling-mounted smart cameras for decentralized multi-robot indoor

navigation," *Proceedings of the 13th International Conference on Distributed Smart Cameras*, 2019. DOI:10.1145/3349801.3349814. [Prezerané: 1.3.2022].
[22] W. Zhu, D. Jia, H. Wan, T. Yang, C. Hu, K. Qin, and X. Cui, "Waypoint Graph Based Fast Pathfinding in Dynamic Environment," *International Journal of Distributed Sensor Networks*, vol. 11, no. 8, p. 238727, 2015. DOI: https://doi.org/10.1155/2015/736138. [Prezerané: 1.3.2022].
[23] S. Pütz, T. Wiemann, J. Sprickerhof, and J. Hertzberg, "3D navigation mesh generation for path planning in uneven terrain," *IFAC-PapersOnLine*, vol. 49, no. 15, pp. 212–217, 2016. DOI: https://doi.org/10.1016/j.ifacol.2016.07.734. [Prezerané: 6.3.2022].
[24] X. Cui a H. Shi, "Direction oriented pathfinding in video games," *International Journal of Artificial Intelligence & Applications*, vol. 2, no. 4, pp. 1–11, 2011. DOI: 10.5121/ijaia.2011.2401. [Prezerané: 7.3.2022].
[25] V. Flovik, "What is Graph Theory, and why should you care?," *Towards Data Science*, 12.8.2020. [Online]. Dostupné na: https://towardsdatascience.com/what-is-graph-theory-and-why-should-you-care-28d6a715a5c2. [Prezerané: 10.3.2022].
[26] I. Brezina a P. Gežík, "Teória grafov pre ekonómov," Bratislava: Vydavateľstvo Letra Edu, 2018. [Prezerané: 10.3.2022].
[27] "Graphs, " *Brilliant.* [Online]. Dostupné na: https://brilliant.org/wiki/graphs/. [Prezerané: 12.3.2022]
[28] A. Sharma, "Time and Space Complexity," *Hackerearth*, [Online]. Dostuné na: https://www.hackerearth.com/practice/basic-programming/complexity-analysis/time-and-space-complexity/tutorial/. [Prezerané: 15.4.2022].
[29] "Fleury's Algorithm for printing Eulerian Path or Circuit," *GeeksForGeeks*, 4.2.2022, [Online]. Dostuné na: https://www.geeksforgeeks.org/fleurys-algorithm-for-printing-eulerian-path/. [Prezerané: 16.4.2022].
[30] W. L. Pearn a J. B. Chou, "Improved solutions for the Chinese postman problem on Mixed Networks," *Computers & Operations Research*, vol. 26, no. 8, pp. 819–827, 1999. DOI: https://doi.org/10.1016/S0305-0548(98)00092-6. [Prezerané: 17.3.2022].
[31] D. Lippman, " Eulerization and the Chinese Postman Problem," *Libretexts*, 5.8.2021. [Online]. Dostupné na: https://math.libretexts.org/@go/page/34208. [Prezerané: 17.3.2022].
[32] E. C. Navone, "Dijkstra's Shortest Path Algorithm - A Detailed and Visual Introduction," *FreeCodeCamp*, 28.9.2020. [Online]. Dostupné na: https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/. [Prezerané: 7.1.2022].

[33] “A* Search Algorithm,“ *GeeksForGeeks*, 13.4.2022, [Online]. Dostuné na: https://www.geeksforgeeks.org/a-search-algorithm/. [Prezerané: 16.3.2022].
[34] Computerphile, “A* (A Star) Search Algorithm – Computerphile,“ *YouTube*, 15. 2. 2017. [Online]. Dostupné na: https://youtu.be/ySN5Wnu88nE. [Prezerané: 8.1.2022].
[35] “What is an adjacency matrix?,“ *javaTpoint.* [Online]. Dostupné na: https://www.javatpoint.com/what-is-an-adjacency-matrix. [Prezerané: 11.3.2022].
[36] mycodeschool, “Graph Representation part 03 - Adjacency List,“ 24. 10. 2016, *YouTube*, [Online]. Dostupné na: https://youtu.be/k1wraWzqtvQ. [Prezerané: 16.4.2022].
[37] Veritasium, “ The Genius of 3D Printed Rockets,“ *YouTube*, 12. 8. 2021. [Online]. Dostupné na: https://youtu.be/kz165f1g8-E. [Prezerané: 19.4.2022].
[38] Veritasium, “An Affordable 3D-Printed Arm,“ *YouTube*, 26. 2. 2015. [Online]. Dostupné na: https://youtu.be/AcLh-aSUdx0. [Prezerané: 19.4.2022].
[39] J. Jones a S. Waddell, “The Cascading Costs of Waterfall,“ *Medium*, 23.4.2019. [Online]. Dostupné na: https://medium.com/@joneswaddell/the-cascading-costs-of-waterfall-5c3b1b8beaec. [Prezerané: 3.4.2022].
[40] “Iterative Model: What Is It And When Should You Use It?,“ *Airbrake*, 15.12.2016. [Online]. Dostupné na: https://airbrake.io/blog/sdlc/iterative-model. [Prezerané: 3.4.2022].
[41] “spiral model,“ *TechTarget*, 31.8.2019. [Online]. Dostupné: https://www.techtarget.com/searchsoftwarequality/definition/spiral-model. [Prezerané: 3.4.2022]
[42] A. Altvater, “What is Agile Methodology? How It Works, Best Practices, Tools,“ *Stackify*, 17.9.2017. [Online]. Dostupné na: https://stackify.com/agile-methodology/. [Prezerané: 3.4.2022].
[43] “Agile Frameworks: Your Helpful, 3-Minute Guide,“ *professional development*, 29.2.2022. [Online]. Dostupné na: https://www.professionaldevelopment.ie/what-are-agile-frameworks. [Prezerané: 3.4.2022].
[44] D. West, “Scrum roles and the truth about job titles in scrum,“ *Atlassian.* [Online]. Dostupné na: https://www.atlassian.com/agile/scrum/roles. [Prezerané: 3.4.2022].
[45] “What Is Kanban? Explained for Beginners,“ *Kanbanize.* [Online]. Dostupné na: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban. [Prezerané: 3.4.2022].
[46] “Extreme Programming: Values, Principles, and Practices,“ *altexsoft*, 8.1.2021. [Online]. Dostupné na: https://www.altexsoft.com/blog/business/extreme-programming-values-principles-and-practices/. [Prezerané: 3.4.2022].

[47] R. Lynn, “What is FDD in Agile?“ *planview*. [Online]. Dostupné na: https://www.planview.com/resources/articles/fdd-agile/. [Prezerané: 3.4.2022].
[48] “SDLC - Big Bang Model,“ *tutorialspoint.* [Online]. Dostupné na: https://www.tutorialspoint.com/sdlc/sdlc_bigbang_model.htm. [Prezerané: 3.4.2022].
[49] Freedom in Thought, “How To Be 10x More Productive | The Ultimate Guide to Productivity,“ *YouTube*, 23.11.2020. [Online]. Dostupné na: https://youtu.be/lbtte7iTS9g. [Prezerané: 15.4.2020].
[50] “Prim's Algorithm,“ *Programiz.* [Online]. Dostupné na: https://www.programiz.com/dsa/prim-algorithm. [Prezerané: 17.4.2022].

Zoznam obrázkov

Zoznam tabuliek

Prílohy