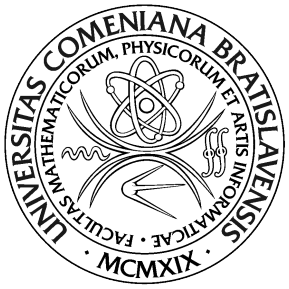
UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



MAPOVANIE PROSTREDIA MOBILNÉHO ROBOTA

Diplomová práca

2021 Bc. Alexander Szendy

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

MAPOVANIE PROSTREDIA MOBILNÉHO ROBOTA

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Mgr. Pavel Petrovič, PhD.

Bratislava, 2021 Bc. Alexander Szendy

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Alexander Szendy
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Mapovanie prostredia mobilného robota
Mobile Robot Mapping its Environment

Anotácia: Cieľom práce je preskúmať algoritmy mapovania a navrhnúť a realizovať systém mapovania prostredia v rozsahu jedného pavilónu pre autonómneho mobilného robota. Mapovací algoritmus využije senzory hĺbkovej kamery Asus Xtion, lidaru, otáčkových senzorov a ultrazvukových senzorov na meranie vzdialenosti a na zostrojenej mape sa robot bude úspešne pohybovať a lokalizovať.

Literatúra: C. Cadena and L. Carlone and H. Carrillo and Y. Latif and D. Scaramuzza and J. Neira and I. Reid and J.J. Leonard, "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age", in IEEE Transactions on Robotics 32 (6) pp 1309-1332, 2016

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 24.10.2018

Dátum schválenia: 24.10.2018 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne len s použitím uvedenej literatúry a za pomoci konzultácií u môjho školiteľa.

. . . . . .

Bratislava, 2021 Bc. Alexander Szendy

Poďakovanie

Touto cestou by som sa chcel v prvom rade poďakovať môjmu školiteľovi Mgr. Pavlovi Petrovičovi, PhD. za jeho cenné rady a usmernenia a ochotu pomôcť, ktoré mi veľmi pomohli pri riešení tejto diplomovej práce. Takisto sa chcem poďakovať všetkým mojím kamarátom a celej mojej rodine za podporu počas môjho štúdia.

Abstrakt

V tejto diplomovej práci sa venujeme autonómnemu 2D mapovaniu neznámeho prostredia mobilného robota s podvozkom typu differential drive. Pracujeme s robotom Mikešom, ktorý bol postavený a upravovaný do rôznych súťaží. Pri riešení využívame laserový senzor SICK TIM571, sledovaciu 3D kameru Intel RealSense T265 a ultrazvukové senzory HC-SR04. Laserový senzor používame na vytváranie mapy a spolu s ultrazvukmi na lokálnu navigáciu. Využívame spôsob sensor fusion pre zjednotenie dát týchto dvoch senzorov. Vďaka obstojnej presnosti kamery T265 nevyužívame žiadne vylepšenia pozície. V práci sme navrhli a úspešne realizovali originálny systém autonómneho mapovania. Overili sme ho na zmapovaní dostupnej časti jedného fakultného pavilónu. Výsledný systém je modulárny a môže byť využívaný v ďalších aplikáciách, demonštráciach, výučbe a výskumných úlohách.

Kľúčové slová: slam, autonómne mapovanie, sensor fusion

Abstract

In this diploma thesis we deal with the problem of autonomous mapping of unknown 2D environment of a mobile robot with differential drive type of chassis. We work with robot Mikeš, which was built and remaked for various robot competitions. We use laser range sensor SICK TIM571, tracking 3D camera Intel RealSense T265 and ultrasonic sensors HC-SR04. The laser sensor is used for map construction, and together with ultrasonic sensors it is also used for local navigation. We use a method of sensor fusion for combining both the ultrasonic and TIM571 data. Thanks to a decent accuracy of tracking camera T265 no improvements for odometry are required. In this work we proposed and realised a unique system of autonomous mapping. We confirmed that solution is feasible by mapping one accessible part of our faculty pavilion. The resulting system is modular and it can be used in other applications, demonstrations, learning and research.

Keywords: slam, autonomous mapping, sensor fusion

Obsah

Úvod

Robotika sa čoraz viac dostáva do popredia aj v bežných domácnostiach a chceli sme sa dozvedieť viac o tom, ako takéto roboty fungujú spolu s výskumom rôznych algoritmov a k nim potrebných senzorov. V posledných rokoch začali byť dostupné robotické spotrebiče aj pre bežných ľudí ako sú napríklad robotické vysávače alebo robotické kosačky.

Zaujímavosťou je aj to, že tieto robotické spotrebiče dosahujú rozmerov podobných ich neautonómnym alternatívam, dokonca dokážu byť menšie vďaka tomu, že nepotrebujú mať rôzne rukoväte a pomôcky potrebné na ovládanie človekom. Roboty nám uľahčujú život a šetria čas každý deň a možno si to niekedy ani neuvedomujeme. Roboty sa využívajú už dlhšiu dobu aj v továrňach a výrobných halách, kde dokážu dokonca pracovať s väčšou presnosťou ako človek.

Roboty pomáhajú tiež v zdravotníctve alebo pri záchrane ľudí. Existujú roboty, ktoré skenujú prostredie a vytvárajú mapy, ktoré možno použiť na zrekonštruovanie pôdorysu budov. V zdravotníctve roboty pomáhajú pri rozvoze liekov pacientom v rámci budovy.

V neposlednom rade prispejeme k zviditeľneniu katedry robotiky na fakulte, ktorá môže robota prezentovať na dni otvorených dverí. Mapovanie je základom množstva systémov. Náš výskum sa bude dať použiť ako základ pre následné využitie robota na vykonávanie rôznych úloh v rámci budovy, ako je napríklad rozvoz pošty. Naša práca sa venuje mapovaniu prostredia

mobilného robota. Cieľom je, aby robot zmapoval dané prostredie a súčasne sa v ňom vedel navigovať a lokalizovať.

V kapitole prehľad problematiky si priblížime začiatky výskumu v danej oblasti vo svete a ich postupné zlepšovanie. Porovnáme si rôzne možné prístupy a možné riešenia našej práce, z ktorých si neskôr zvolíme tie najvhodnejšie. Vyberieme si tiež z už vytvorených riešení a popíšeme si tie, ktoré nám určitým spôsobom pomohli a inšpirovali nás pri vytváraní nášho riešenia.

V ďalšej kapitole si predstavíme nášho robota. Popíšeme si z čoho sa skladá, aké má osadené senzory s ich využiteľnosťou v našej práci. V tejto kapitole bude tiež uvedená fotka nášho robota pre jeho lepšiu predstavu.

V kapitole návrh a implementácia si popíšeme ako sme našu prácu vypracovali. Popíšeme si zvolené postupy a naše konkrétne riešenia v podobe vzorcov pre výpočty a pseudo-kódu našej implementácie.

V predposlednej kapitole si popíšeme výsledky, ktoré sme s našim riešením dosiahli. Popíšeme si jednotlivé verzie, ktoré sme počas práce vylepšovali a odstraňovali prípadné nepresnosti alebo chyby s úpravou rôznych parametrov senzorov, prípadne pridávaním nových senzorov.

Nakoniec si zhrnieme, k čomu sme v našej práci dospeli a či sme splnili ciele našej práce. Predstavíme si tiež možné vylepšenia a pokračovania v nadväznosti na našu prácu.

V zadaní uvádzame použitie hĺbkovej kamery ASUS Xtion, ktorá nám mala pôvodne slúžiť na detekciu prekážok a detekciu známych bodov pri použití uzatvárania cyklu. Namiesto tejto kamery sme však použili iné 3D zariadenie Intel Realsense T265, ktoré má síce trochu odlišnú funkcionalitu, ale tiež pracuje na princípe spracovania 3D obrazu a výrazne zlepšuje možnosti nášho systému.

Kapitola 1

Prehľad problematiky

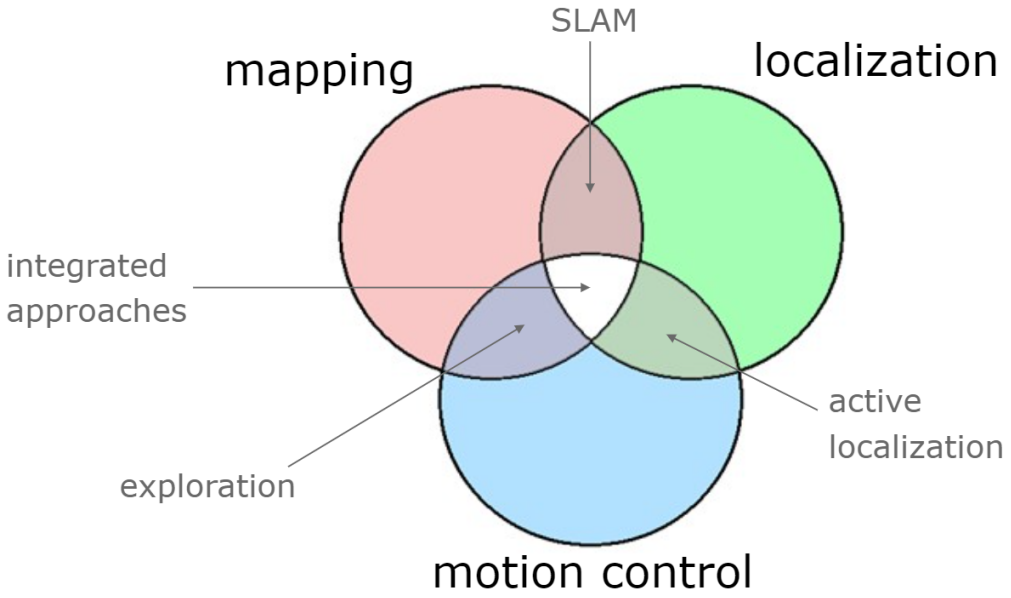
V tejto časti si predstavíme všetky základné pojmy a definície, s ktorými budeme ďalej v práci pracovať. Predstavíme si metódu SLAM, ktorá sa používa na mapovanie a lokalizáciu robota v neznámom prostredí. Povieme si aj v akom stave sa momentálne nachádza výskum v tejto oblasti.

1.1 Prehľad skúmania prostredia

Metóda skúmania prostredia v robotike je kľúčový systém pre úspešné fungovanie autonómnych robotov.Využíva sa na vytváranie máp, 3D skenovanie budov a rovnako aj pri bežnej navigácii robota. Problém skúmania prostredia môžeme rozdeliť na jednotlivé pod problémy popísané v článku *An experiment in integrated exploration*[MWBDW02] a zobrazené na obrázku 1.1. Hlavnými troma časťami sú:

- Mapovanie
- Lokalizácia
- Navigácia

Mapovanie slúži na vytváranie mapy prostredia. Mapa prostredia sa vytvára na základe dát z hĺbkových senzorov, ktoré dané prostredie naskenujú. Lokalizácia možno rozdeliť na lokálnu, vzhľadom na aktuálny stav robota a globálnu, čím sa rozumie jeho pozícia na mape. Stav robota určuje jeho aktuálna parametre ako sú napríklad smer natočenia, uhol natočenia voči svetovým stranám alebo prejdená vzdialenosť. Navigácia slúži na určenie smeru, ktorým sa má robot pohybovať.



Obr. 1.1: Úlohy, ktoré robot rieši pri práci s modelom prostredia. [Bur]

Medzi jednotlivými časťami ešte rozlišujeme prelínajúce sa prístupy popísané v článku [All]:

- Vzájomné porovnávanie viacerých skenov prostredia
- Porovnávanie skenov prostredia s už vytvorenou mapou
- SLAM

Vzájomné porovnávanie skenov slúži na vylepšenie lokalizácie robota. Na základe porovnávania viacerých skenov sa určuje odhadovaná zmena pozície robota. Tento prístup je nezávislý od existujúcej mapy. Využíva sa hlavne pri prvotnom vytváraní mapy prostredia, tiež v situácii, keď aktuálne prostredie nezodpovedá už vytvorenej mape vzhľadom na zmeny v prostredí.

Porovnávanie skenov s mapou sa využíva, podobne ako vzájomné porovnávanie, na odhad pozície robota, ale porovnávaním dát skenovania s vytvorenou mapou. Pri neustále sa opakujúcich podobných objektoch v prostredí sa môže stať, že sa zle určí podobnosť naskenovaných objektov s objektami na mape. Ak sa takýto jav deje dlhodobejšie, môže sa stať, že sa robot zle lokalizuje a neskôr bude mať problémy s návratom k reálnej pozícii.

1.2 SLAM

Súčasná lokalizácia a mapovanie (SLAM) pozostáva zo súčasného vytvárania modelu prostredia (mapy) a odhadovaného stavu robota pohybujúceho sa v ňom. Stav robota sa odhaduje pomocou na ňom nainštalovaných senzorov popísaných v sekcii 1.3 a vytváranie mapy prostredia sa realizuje z nich získaných údajov. Údajmi o stave robota rozumieme v jednoduchších verziách jeho pozíciu a smer, ale aj hodnoty ako sú rýchlosť robota, odchýlky senzorov alebo kalibračné parametre. Mapa, na druhú stranu, je reprezentáciou bodov záujmu, ako sú napríklad pozície orientačných bodov a prekážok, popisujúcich neznáme prostredie, v ktorom sa robot pohybuje.

Mapa prostredia sa používa na rôzne účely, na základe individuálnych potrieb v konkrétnom riešení. Môže byť úlohou robota vytvoriť výstupnú mapu prostredia, s ktorou sa bude ďalej pracovať, alebo sa mapa vytvára pre jeho samotné plnenie úloh, pri ktorých je potrebné, aby sa úspešne navigoval a

lokalizoval. Pre vylepšenie lokalizácie sa môže využiť aj vzájomné porovnávanie mapy, kde je snaha eliminovať lokalizačnú chybu. Takéto porovnávanie je bližšie popísané v sekcii 1.2.4.

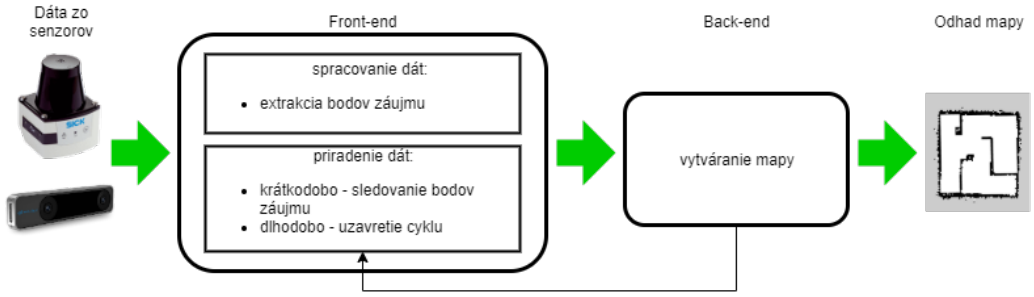
1.2.1 História

V snahe vytvoriť autonómneho robota bol problém súčasnej lokalizácie a mapovania publikovaný niekoľkými vedcami na konferencii IEEE Robotics and Automation v roku 1986 konajúcej sa v San Francisku v článku [DWB06]. Hlavnými z nich boli Smith, Cheeseman a Durrant-Whyte. Dôkladným prehľadom prvých 20 rokov tejto problematiky, tiež nazývaným *klasickým vekom* (1986-2004)[CCC+16], sa zaoberá práve Durrant-Whyte a Bailey v dvoch článkoch [DWB06, BDW06]. V tomto čase bol problém považovaný za problém pravdepodobnostný a štatistický. Prvé riešenia boli založené na filtroch: rozšírený Kalmanov filter, Rao-Blakwellove časticové systémy a odhad maximálnej pravdepodobnosti. V tejto počiatočnej dobe boli používané najmä senzory ako sú radary, lidary a sonary.

Nasledujúce obdobie, nazývané *doba algoritmickej analýzy*(2004-2015) [CCC+16], sa zaoberá základnými vlastnosťami problému SLAM, vrátane pozorovania, konvergencie a dôslednosti. V rámci pozorovania sa začali pre lepšiu orientáciu používať zariadenia snímajúce obraz okolia - kamery. To prinieslo úplne iný pohľad na problém SLAM, keď sa začala používať vizuálna odometria, ktorá je ďaleko presnejšia oproti predtým používaným otáčkovým senzorom prípadne iným málo presným zariadeniam. O rozdiele v presnosti sme sa presvedčili aj pri našom riešení.

1.2.2 Algoritmus

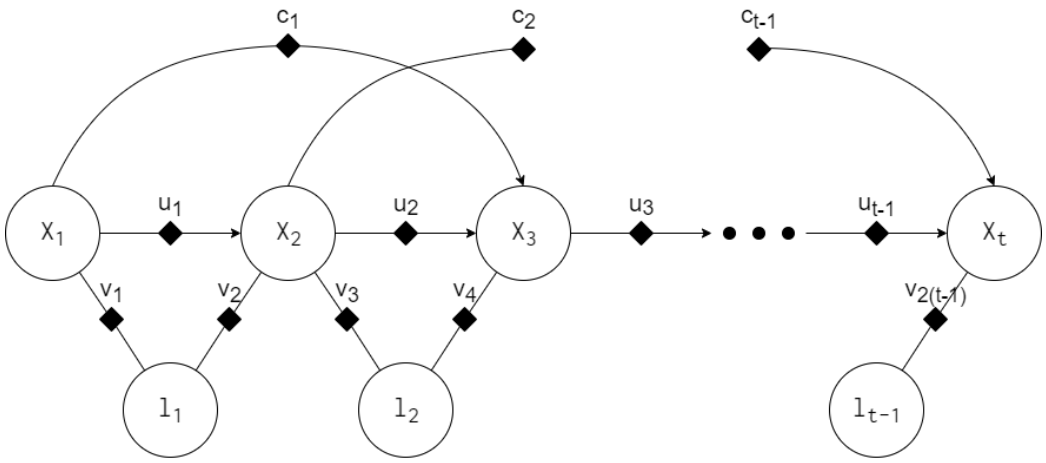
SLAM systém sa skladá z dvoch hlavných komponentov: *front-end* a *back-end*. Front-end pretvára dáta zo senzorov na dáta spracovateľné pre odhadovanie a back-end tieto dáta spracúva a vytvára odhadovanú mapu. Takýto systém je znázornený na obrázku 1.2.



Obr. 1.2: Na obrázku je znázornená schéma typického systému SLAM. Back-end poskytuje spätnú väzbu front-endovej časti na detekciu uzavretého cyklu a jeho overovanie.

Back-End

Vychádzajúc zo štandardnej formulácie SLAM [LM97] s optimalizačnými úpravami [KJR+12] sa dá SLAM definovať ako *maximum a posteriori* (MAP) odhad založený na Bayesovej vete. MAP odhad sa dá chápať ako najvhodnejší odhad s použitím rôznych aj odhadovaných dát [BD19]. Spájaním odhadov pozícií s pozíciami orientačných bodov z dát obrázkov sa dosahuje väčšia presnosť výslednej mapy. Pri formulácii SLAM problému MAP odhadom sa často používa *faktorový graf* [KFL01] pre dobré znázornenie vzájomných závislostí medzi jednotlivými premennými. Takýto graf môžme vidieť na obrázku 1.3 znázorňujúci jednoduchý problém SLAM. Na grafe sú znázornené premenné reprezentujúce pozície robota, pozície orientačných bodov a faktory pôsobiace medzi týmito premennými v podobe obmedzení.



Obr. 1.3: SLAM znázornený ako graf faktorov. Premenná x predstavuje postupnosť pozícií robota v určitom čase t. Premenná l predstavuje pozície orientačných bodov. Čierne kosoštvorce znázorňujú faktory vplývajúce na jednotlivé premenné: u označuje faktory prislúchajúce obmedzeniam odometrie, v označuje faktory prislúchajúce ku pozorovaniam z kamery, c označujú uzavretia cyklov.

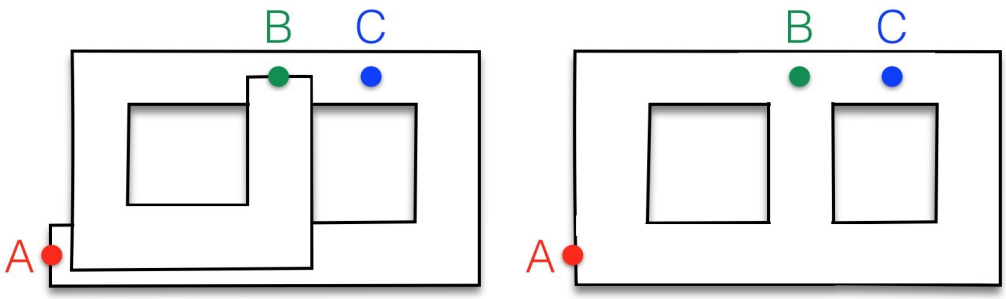
Front-End

Front-end pracuje s nespracovanými dátami získanými priamo zo senzorov. Pri praktických aplikáciach v robotike môže byť náročné napísať analytickú funkciu stavu z dát z meraní senzorov, aká je potrebná pre MAP odhad. Napríklad pre neupravené dáta zo senzorov, v prípade obrázka je zložité vyjadriť intenzitu každého pixelu ako funkciu stavu. Pri jednoduchších senzoroch ako sú napríklad lasery s jedným lúčom nastáva rovnaký problém. Problém je spojený s tým, že nie je možné zostrojiť úplne všeobecný obraz prostredia, ale aj v prípade zostrojenia takéhoto obrazu by bolo zložité napísať analytickú funkciu spájajúcu merania s parametrami danej reprezentácie obrazu.

Preto sa pred back-endom zvykne používať modul (front-end), ktorý spracúva relevantné informácie z dát zo senzorov. Napríklad vo vizuálne založe-

nom SLAM systéme, modul získava pozície pixelov niektorých jednoznačne odlíšiteľných bodov v prostredí, ktoré sa následne jednoduchšie modelujú v back-ende. Vo front-ende sa tiež priraďuje každé meranie ku zodpovedajúcemu orientačnému bodu v prostredí. Toto *priradenie dát* je znázornené na obrázku 1.2. Modul priradenia dát teda priraďuje každému meraniu z_k podmnožinu neznámych premenných X_k: $z_k = h_k(X_k) + \epsilon_k$. Nakoniec môže front-end poskytnúť prvotný odhad pre premenné v nelineárnej optimalizácii.

Modul priradenia dát, ako je zobrazený na obrázku 1.2 typického SLAM systému, pozostáva z dvoch častí: krátkodobá časť a dlhodobá časť. Krátkodobé priradenie dát je zodpovedné za priraďovanie zodpovedajúcich bodov záujmu v po sebe idúcich meraniach senzorov. Krátkodobé priradenie má teda napríklad za úlohu v dvoch po sebe idúcich meraniach rozpoznať ten istý bod v priestore. Na druhú stranu dlhodobé priradenie (uzavretie cyklu) má za úlohu priraďovať nové merania k starším orientačným bodom. Preto ako je znázornené na obrázku 1.2, back-end spätne poskytuje dáta front-endovej časti, pre podporu zisťovania a overovanie uzatvárania cyklov.



Obr. 1.4: Vľavo: mapa vytvorená z odometrie. Mapa sa na spodnej časti chodby prekrýva, pričom je vytvorená zo začínajúceho bodu A do koncového bodu B. V tejto odometrickej mape sú body B a C od seba vzdialené, aj keď sa v skutočnosti nachádzajú vedľa seba.
Napravo: mapa vytvorená metódou SLAM. S využitím metódy uzatvárania cyklov je odhadovaná mapa upravená podľa skutočnej topológie prostredia a spája v tomto prípade predtým vzdialené body B a C. [CCC+16]

1.2.3 Reprezentácia modelu mapy

Mapové reprezentácie SLAM modelov môžeme rozlíšiť podľa toho, aké informácie využívajú na tvorbu mapy a tiež ako tieto informácie na mape zobrazujú. Mapy môžeme rozdeliť na metrické a sémantické.

1.2.3.1 Metrické modely

Metrická reprezentácia je štruktúra zachytáva geometriu prostredia. Výber štruktúry pre reprezentáciu úzko súvisí s potrebami jej využitia. Každá z možností poskytuje iné vlastnosti a teda zvažujeme dlhodobú navigáciu, interakcie s prostredím a interakcie s človekom.

2D geometria

V 2D prostredí je geometrické mapovanie oveľa jednoduchšie s dvomi dominantnými technikami: mapy založené na pozícii orientačných bodov, alebo bodov záujmu a mapy obsadenosti (grid mapy). Prvá z nich modeluje prostredie ako množinu orientačných bodov a druhá rozdeľuje prostredie na bunky, v ktorých zisťuje pravdepodobnosť obsadenosti. Štandard definujúci metrické reprezentácie 2D prostredia je popísaný v článku [73015], ktorý popisuje tieto dve hlavné metódy reprezentácie a topologické mapy. O topologických mapách a grid mapách tiež čerpáme informácie z článku [TB96a].

Grid mapy

Pri použití grid mapy sa rozdeľuje priestor na rovnako veľké časti mriežky [Mor89]. Každá bunka mriežky môže napríklad obsahovať informáciu o tom, či je dané miesto v skutočnom prostredí prázdne alebo sa v ňom nachádza nejaká prekážka. Tieto mapy sú jednoduché na zostrojenie a udržiavanie aj pre veľké priestory [TB96b]. Keďže štruktúra mapy priamo zodpovedá štruktúre

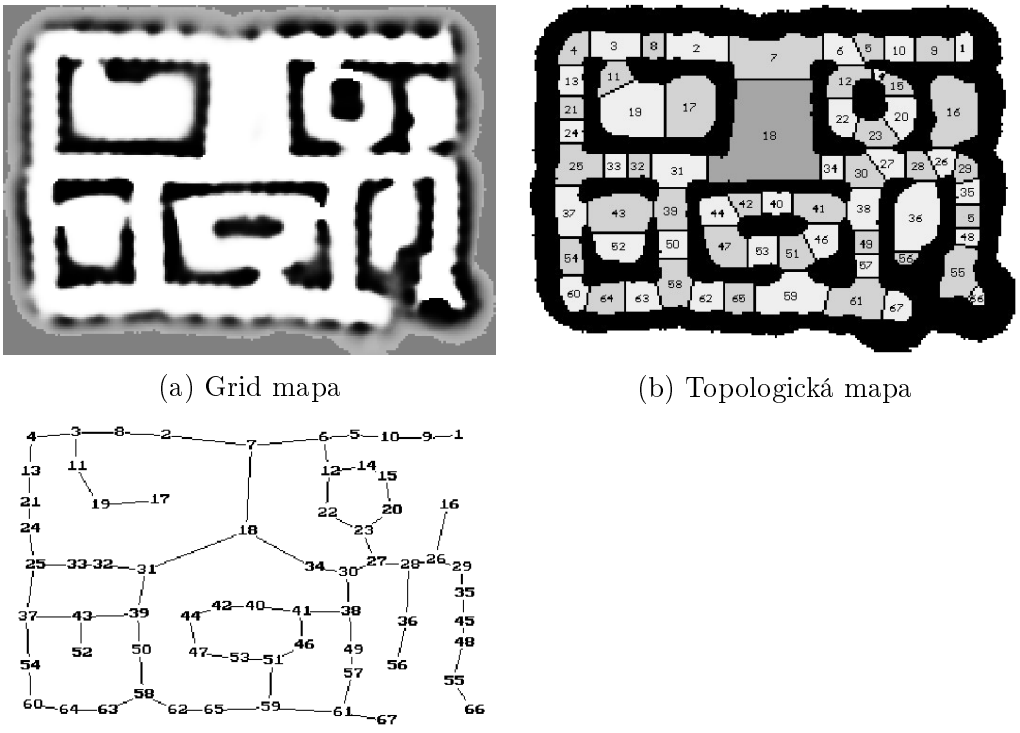
prostredia, pozícia a orientácia robota v zostrojenej mape sa dá jednoducho odhadnúť podľa skutočnej pozície robota v prostredí. Vďaka tomu v prípade rôznych pozícií vyzerajúcich z pohľadu senzorov rovnako (podobné tvary prostredia) sú jednoducho rozpoznateľné a odlíšené. Na druhú stranu kvôli potrebe zachytenia každého detailu v prostredí musí byť rozlíšenie dostatočne veľké, preto spotrebujú dosť pamäte a tiež sa zvýši časová náročnosť na jednotlivé výpočty. Vzhľadom k tomu, že sa získava pozícia priamo zo senzorov, môže sa stať, že po čase nebude zodpovedať skutočnej pozícii, preto je nutná neustála kontrola a kompenzácia tohto problému. Príklad gridmapy je znázornený na obrázku 1.5a.

Topologické mapy

V topologických mapách [PK94] sa priestor reprezentuje pomocou grafu. Vrcholy grafu predstavujú rôzne situácie, miesta alebo orientačné body. V prípade, že existuje priama cesta medzi nimi, sú spojené oblúkom. Pozícia robota v prostredí sa určuje porovnávaním s modelom orientačných bodov alebo rôznymi vizuálnymi vlastnosťami. V prípade, že robot prechádza podobným priestorom, môže mať problémy s detekciou, či sa jedná o už prejdenú časť, alebo ide len o novú, ale podobnú časť. Topologické mapy sú často dosť presné a nepotrebujú oproti grid mapám toľko miesta, keďže členitosť grafu je priamo úmerná členitosti prostredia. Vďaka grafovej štruktúre sa tiež dá jednoducho realizovať plánovanie trasy. Topologická mapa a k nej príslušný graf je vyobrazená na obrázkoch 1.5b a 1.5c.

Grid mapa		Topologická mapa	
+	jednoduchá na zostrojenie a údržbu	+	umožňuje efektívne plánovanie
+	rozpoznávanie už navštívených miest	+	nepotrebuje presnú lokalizáciu
-	neefektívne plánovanie trasy trasy	-	náročnejšie vytvorenie a údržba vo väčších prostrediach
-	potrebné presné zisťovanie pozície	-	rozpoznávanie rovnakého prostredia (citlivé na uhol pohľadu)
-	rozlíšenie nezávisí od členitosti prostredia		

Tabuľka 1.1: Porovnanie grid mapy a topologickej mapy



(a) Grid mapa

(b) Topologická mapa

(c) Topologický graf

Obr. 1.5: Na obrázku je znázornená *(a)* Grid-mapa, *(b)* vytvorená topologická mapa z obrázku *(a)*, *(c)* grafová reprezentácia danej topologickej mapy. [TB96a]

3D geometria

Modelovať 3D geometriu je o dosť zložitejšie. 3D modelovanie mapy je spojené hlavne s počítačovým videním a počítačovou grafikou. Jednou z metód je *riedka reprezentácia orientačných bodov*, ktorá reprezentuje prostredie ako množinu 3D objektov prislúchajúcich k diskriminačným vlastnostiam v prostredí (líniám, rohom) [MAMT15]. Ďalšou je *nízkoúrovňová nespracovaná reprezentácia* [NLD11], ktorá je narozdiel od tej predchádzajúcej náročná na pamäť, keďže sa snaží produkovať mapu s vysokým rozlíšením. Tieto mapy sú vhodné pre využitie v prípade vyhýbania sa prekážkam alebo vizualizáciu a renderovanie. Ďalším príkladom je *reprezentácia ohraničenia a rozdelenia priestoru* [LS15], v ktorom sa explicitne zobrazujú povrchy alebo hranice a časti. Tieto reprezentácie sú vhodné na pohyb a plánovanie trasy a tiež na vyhýbanie sa prekážkam.

1.2.3.2 Sémantické modely

Sémantické mapy pozostávajú z priraďovania sémantických konceptov ku geometrickým objektom v okolí robota. Môže sa zdať, že je to podobný spôsob topologickému mapovaniu, avšak je úplne odlišný. V topologickom mapovaní sa vytvára graf prostredia, v ktorom vrcholy predstavujú rozdielne miesta a hrany dostupnosť medzi nimi. Kým topologické mapovanie je závislé na určovaní miesta na základe predchádzajúcich skenovaní (bez ohľadu na typ miestnosti), sémantické modely rozdeľujú prostredie do sémantických označení. Sémantické modely súvisia s úlohou robota v danom prostredí. Roztrieďujú jednotlivé časti prostredia do skupín podľa súvislostí. Na vytváranie sémantických máp sa využíva sémantická segmentácia s využitím modelov hlbokých neurónových sietí popísaná v článku [HX20].

Sémantický model je vytvorený definovaním nasledujúcich hľadísk:

- Úroveň/detail konceptov: napríklad pre úlohu robota prejsť z miestnosti A do B stačia kategórie- miestnosti, chodby, dvere; pre úlohu zobrať šálku čaju by boli potrebné ďalšie kategórie- stôl, šálka, pohár.

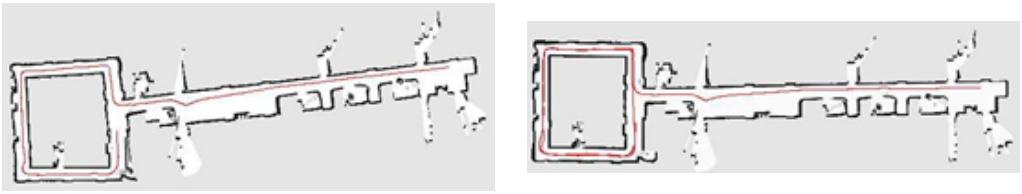
- Organizácia konceptov: sémantické koncepty sa vzájomne nevylučujú, ba naopak jeden objekt môže mať nekonečne veľa vlastností. Napríklad stôl a stolička majú rovnakú vlastnosť "pohybovateľnosť". Kým oba predmety sú kusy nábytku, obsahujú rovnakú vlastnosť pohybu ale každý s odlišnou použiteľnosťou. Preto je potrebné dôkladné navrhnutie celého systému, aby boli ošetrené takéto dvoj alebo viac-konceptuálne prípady.

Sémantické modely sa v robotike začali využívať pomerne nedávno, oproti metrickým, ktoré sa používali v podstate od vzniku problému mapovania. Jedným z prvých využití boli práce [MTJ+07, CGKM07].

1.2.4 Uzatváranie slučky

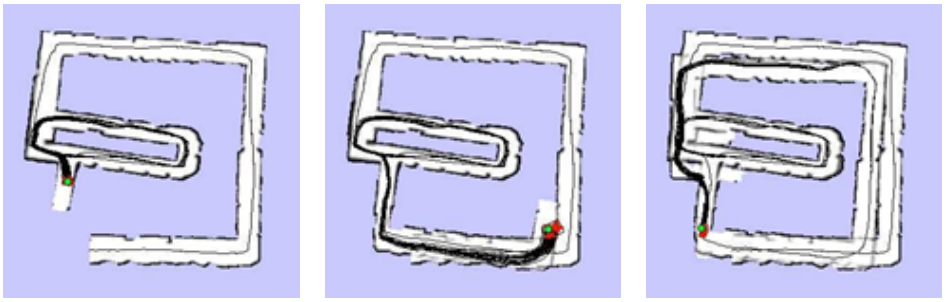
Metóda uzatvárania slučiek je potrebná pre správne fungovanie celého SLAM systému. Má za úlohu opraviť polohu robota, ktorá sa mohla po čase odchýliť od skutočnej pozície v prostredí, v závislosti od presnosti merania senzorov. Túto metódu je nutné využívať najmä pri využití grid mapového mapovania, keďže v tomto mapovaní sa využívajú priame neupravené dáta z senzorov pozícia sa po čase môže ľahko odchýliť, následkom čoho sa celé mapovanie stáva nepresné.

Príklad takejto odchýlky môžeme vidieť na obrázku 1.6 z článku [SHBG05]. V článku autori zdôrazňujú, že pre prípad správnej lokalizácie je potrebné znovu-navštívenie už prejdeného miesta, pre potrebu re-lokalizácie a opravenia odchýlky v pozícii.



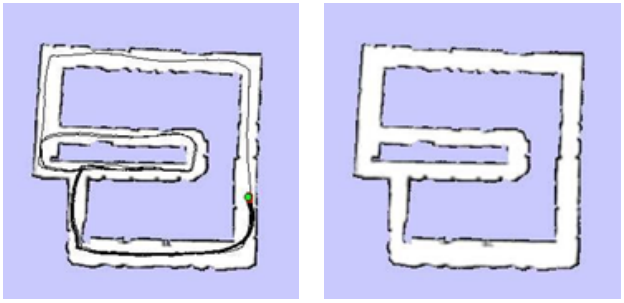
Obr. 1.6: Príklad použitia metódy uzavretia cyklu.
Vľavo je zobrazenie mapy vytvorenej s použitím metódy uzavretia cyklov bez znovu-navštívenia už prejdenej časti, čo viedlo k 7° chybe vo vytváraní mapy horizontálnej chodby.
Napravo je zobrazená mapa s algoritmom, kde robot prešiel už predtým prejdenou časťou, čím sa jeho pozícia upravila a chodba vpravo už je vychýlená len o 1°. [SHBG05]

Nutnou podmienkou je v prípade detekcie cyklov pamätanie si už navštívených miest a orientačných bodov k nim prislúchajúcim. Pri väčších prostrediach môže byť takýto systém príliš náročný na pamäť, preto sú často nutné optimalizácie. S čím väčšími dátami sa pracuje, tým je celý systém pomalší a pre prípad aktívneho mapovania to nie je vhodné. Napríklad jednou z optimalizácií môže byť redukcia orientačných bodov na čo najmenší počet, avšak v dostatočne veľkom pokrytí prostredia. Pri prílišnej optimalizácii môže nastať problém znázornený na obrázku 1.7.



Obr. 1.7: Príklad neúspešného pokusu o uzavretie cyklu.
Na obrázku vľavo vidno cyklus, v ktorom robot sa robot "zacyklil"pri snahe uzatvoriť cyklus a následne po niekoľkých prejdeniach cyklu pokračoval ďalej, kde už ale na druhom obrázku vidno, že sa nedokázal správne lokalizovať. Napravo je zobrazená mapa po neúspešnom pokuse o uzavretie cyklu. [SHBG05]

V prípade, že sa robot zacyklí, nemusí to znamenať, že nedokáže detegovať orientačný bod vôbec. Môže sa stať, že orientačný bod síce deteguje, ale nelokalizuje sa s dostatočnou presnosťou a algoritmus to môže vyhodnotiť ako neúspešnú lokalizáciu a pokúša sa o uzavretie cyklu, až kým sa to nepodarí. Avšak čím väčšiu vzdialenosť v prostredí robot prejde, o to viac prvotných údajov o prostredí stráca. V takomto prípade je možné nastaviť premennú obmedzenia, ktorá určí toleranciu v nepresnosti lokalizácie pri uzatváraní cyklov. Na obrázku 1.8 vidno ako vyzerá úspešná detekcia uzavretia cyklu z obrázku 1.7.



Obr. 1.8: Príklad úspešného mapovania s detekciou uzavretia cyklu. [SHBG05]

1.2.5 Plánovanie trasy

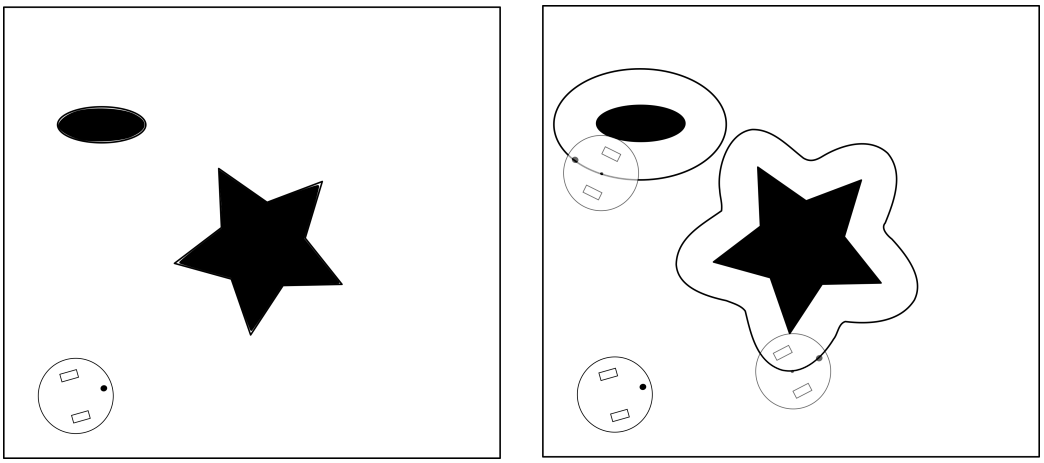
Plánovanie trasy je dôležitou súčasťou autonómneho mobilného robota. Používa sa pri navigácii, kde robotovi slúži na nájdenie najkratšej, alebo optimálnej trasy medzi dvoma bodmi. Na optimálnu trasu sa môžme pozrieť z rôznych uhlov pohľadu, záleží na potrebách konkrétneho robota. Môžeme rozlišovať medzi trasami, ktoré minimalizujú potreby zatáčania alebo otáčania a s tým spojenú potrebu brzdenia alebo pri vyhýbaní sa určitým úsekom v prostredí. S algoritmami na hľadanie najkratšej trasy sa môžme stretnúť nielen v robotike, ale aj pri trasovaní siete, vo video hrách alebo sekvenovanie

DNA [CHHR22].

Pre plánovanie trasy je dôležité poznať mapu prostredia a pozíciu robota v ňom. Existujú rôzne reprezentácie mapy prostredia, najčastejšie sú používané gridmapy alebo topologické mapy. V gridmapách je reprezentácia prostredia prevažne realizovaná pomocou mriežky, rozdelujúc prostredie na štvorčeky rozlíšenia napríklad 1cm x 1cm prislúchajúc skutočnej veľkosti prostredia, v ktorých je informácia, či sa na danom mieste nachádza prekážka alebo voľný priestor 1.2.3.1. V topologických mapách sa prostredie reprezentuje pomocou grafu. Vrcholy grafu predstavujú časti mapy a hrany ich spojenia, teda či sa robot vie dostať z jednej časti na druhú 1.2.3.1.

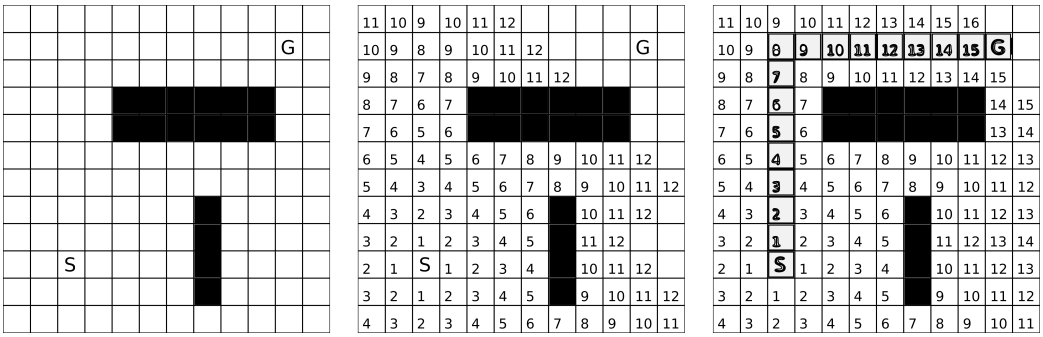
Problémom plánovania najlepšej trasy ako sa dostať z jedného bodu do druhého pomocou spájaných grafov sa zaoberajú rôzne oblasti, najčastejšie na internete, kde sa hľadá optimálna trasa pre dátový paket. Najlepšia trasa znamená najmenšia akumulovaná hodnota presunu medzi vrcholmi, čo predstavuje prejdenú fyzickú vzdialenosť v robotike alebo odozvu na internete.

Jednou z komplikácií pri hľadaní cesty môže byť veľkosť robota, kedy by nájdenou trasou nemusel prejsť. Na zamedzenie tomuto problému sa napríklad využíva zväčšenie prekážok o polomer najdlhšej časti robota, takže robot z pohľadu algoritmu zostane o veľkosti súradnice x a y. Takáto reprezentácia sa nazýva *konfiguračný priestor* [CHHR22].



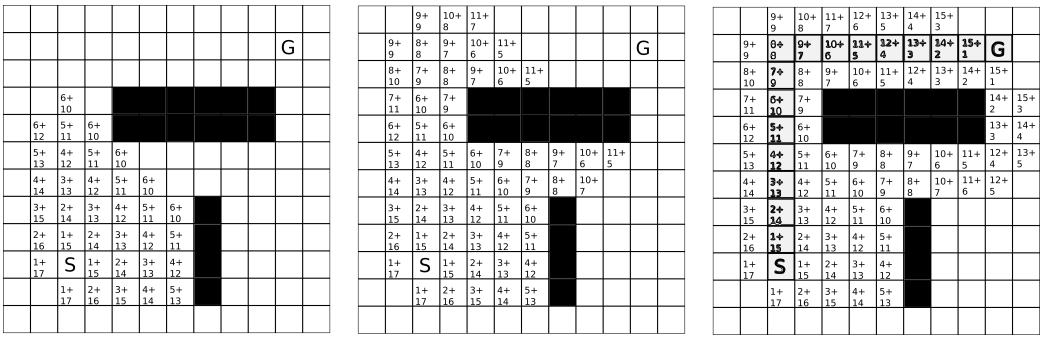
Obr. 1.9: Rozšírenie veľkostí prekážok o polomer robota.[CHHR22]

Prvým a najjednoduchším algoritmom je *Dijkstrov algoritmus* [D+59]. Začínajúc v počiatočnom vrchole cesty algoritmus vyberá všetkých jeho priamych susedov s určením ich ceny potrebnej dostať sa k nim. Následne z týchto susedných vrcholov vyberá práve ten, ktorý má najnižšiu hodnotu hrany a pokračuje jeho susednými vrcholmi a tiež ich označí hodnotou, ak už bol vrchol označený z predošlej iterácie, tak len v prípade, že je menšia ako tá pôvodná. Po skontrolovaní všetkých susedných vrcholov algoritmus prejde na ďalší vrchol s najmenšou hodnotou. Keď sa algoritmus dostane na koniec, k cieľovému vrcholu, cyklus skončí a vyberie sa trasa hranami s najnižšou hodnotou. Príklad Dijkstrovho algoritmu si môžeme pozrieť na obrázku 1.10.



Obr. 1.10: Hľadanie najkratšej trasy z bodu S do bodu G s laterálnou(nie diagonálnou) možnosťou pohybu robota a s cenou jedna na bunku s využitím Dijkstrovho algoritmu.[CHHR22]

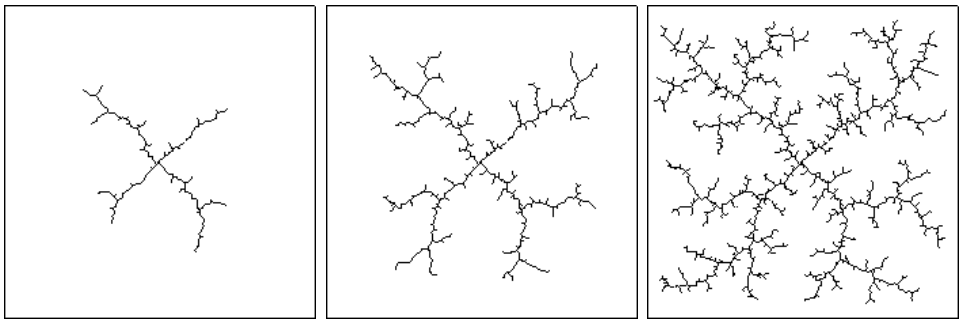
Vzhľadom k povahe Djikstrovho algoritmu sa vykonávajú niektoré výpočty v niektorých prípadoch zbytočne. Algoritmus navštevuje aj vrcholy, ktoré idú opačným smerom od cieľovej pozície. Z vonkajšieho pohľadu na prostredie sa dajú vidieť pozície prekážok a približný smer k cieľu. Túto vedomosť môžeme využiť v *heuristickej funkcii* [KL15]. V heuristike môžeme zohľadniť napríklad vzdialenosť vrcholov od cieľa a uprednostniť tie, ktorých odhadovaná vzdialenosť je menšia. Vzdialenosť medzi aktuálnym vrcholom a cieľovým vrcholom sa dá vypočítať rôznymi spôsobmi, používa sa napríklad *Euklidovská vzdialenosť* alebo *Manhattanská vzdialenosť*. Rozšírením Djikstrovho algoritmu a pridaním k prejdenej vzdialenosti aj odhadovanú vzdialenosť od cieľa vytvoríme algoritmus nazývaný $A*$[KL15]. Vzhľadom na prostredie, by mal $A*$ nájsť trasu rýchlejšie, oproti Djikstrovmu algoritmu. Ukážku ako algoritmus postupuje od začiatočného bodu len smerom ku koncovému bodu je možné pozrieť si na obrázku 1.11.



Obr. 1.11: Hľadanie najkratšej trasy z bodu S do bodu G s laterálnou možnosťou pohybu robota a s cenou jedna na bunku použitím A* algoritmu. Podobne ako Dijkstrov algoritmus vyberá iba bunky s najnižšou cenou, ale berie do úvahy aj odhad vzdialenosti bunky od cieľa.[CHHR22]

Rozšírením A* algoritmu, ktoré by vyriešilo problém drahého znovuplánovania celej trasy od začiatku v prípade, že sa robotovi v ceste objaví prekážka, je *D** algoritmus [Ste93]. Oproti *A**, *D** začína hľadať trasu od koncového bodu a je schopný zmeniť dĺžku častí trasy, ktoré obsahujú prekážku. To umožňuje D* zmeniť plán trasy okolo prekážky so zachovaním väčšiny pôvodnej už naplánovanej trasy.

A* a D* algoritmy by vo veľkom priestore mohli byť náročné na výpočet. Pre väčšie mapy môžeme použiť napríklad *RRT* algoritmus [L+98], ktorý je založený na princípe vytvárania náhodného stromu. Body sú náhodne generované a spájané s najbližším dostupným vrcholom, pričom pri vytváraní vrcholu sa musí zakaždým skontrolovať, či neleží v prekážke. Pospájaním týchto vrcholov sa strom vytvára a rozširuje, čo môžeme vidieť na obrázku 1.12. Cieľom algoritmu je vygenerovanie vrcholu v oblasti cieľového bodu alebo dosiahnutím určitého limitu. Tento algoritmus nie je zostrojený na hľadanie najoptimálnejšej trasy.



Obr. 1.12: Znázornenie priebehu vytvárania trasy pomocou RRT algoritmu [L+98].

1.3 Prehľad senzorov

Senzory sú najdôležitejšou súčasťou lokalizácie a mapovania, keďže poskytujú dáta celému systému. Na presnosti nameraných dát závisí správnosť následnej lokalizácie a mapovania prostredia. Čím viac senzorov sa v systéme použije, o to lepšie a presnejšie bude celý systém fungovať. Rozlišujeme senzory používané na lokalizáciu alebo mapovanie, niektoré sa však dajú použiť pre oba spôsoby. Prehľad senzorov na zisťovanie polohy je popísaný v článku [BEFW97]. Senzory na zisťovanie polohy môžeme rozdeliť do dvoch kategórií podľa závislosti na predchádzajúcich meraniach a to na senzory, ktoré zisťujú relatívnu alebo absolútnu pozíciu.

1.3.1 Senzory na meranie relatívnej pozície

Do tejto kategórie patria senzory, ktoré sú závislé na predošlých meraniach. Pozícia sa vypočítava na základe predchádzajúceho merania a prejdenej vzdialenosti, respektíve posunu od poslednej pozície. Tento spôsob lokalizácie sa tiež nazýva *odometria*. Odometria poskytuje dobrú krátkodobú presnosť, avšak v závislosti na predchádzajúcich meraniach sa nameraná chyba kumuluje

a teda po dlhšej prejdenej trase už môže byť priveľká. Od jednoduchosti riešenia a presnosti lokalizácie teda závisí presnosť zariadenia. Do tejto kategórie patria otáčkomery, merače zrýchlenia, gyroskopy a sledovacie kamery.

1.3.2 Senzory na meranie absolútnej pozície

Senzory v tejto kategórii pracujú len s aktuálnymi dátami. Patria sem zariadenia ako sú napríklad magnetický kompas alebo GPS senzor. Pomocou kompasu zisťujeme smer, ktorým je robot otočený. Nevýhodou kompasu je, že magnetické pole Zeme môže byť často rušené v blízkosti rozvodných sietí alebo železných konštrukcií, preto sa často nedá využiť v interiéri. GPS senzor je závislý od priamej viditeľnosti satelitov, preto je jeho využiteľnosť v interiéri rovnako ako pre kompas.

1.3.3 Senzory na mapovanie

Mapovacie senzory sa využívajú na vytváranie máp pomocou senzorov na meranie vzdialenosti od prekážok. Meranie vzdialenosti od prekážok sa môže realizovať pomocou ultrazvukového senzora (sonaru), radaru alebo laserového senzora (lidaru) fungujúceho na rovnakom princípe- vysielania vĺn a času, po ktorom sa vlny odrážajú od prekážok. Problém pri lidare môže nastať pri odraze laseru od priehľadných prípadne príliš lesklých povrchov, kedy laser buď prejde prekážkou, alebo sa neodrazí. Tiež sa môže stať, že tmavé alebo čierne objekty môžu lúče úplne pohltiť a neodraziť, vtedy to vyzerá, že žiadna prekážka sa v danom smere nenachádza. Používa sa tiež svetlo vyžarujúca hĺbková kamera alebo kamery s detekciou zmeny obrazu. Kamery s detekciou zmeny obrazu vysielajú iba lokálne zmeny pixelov spôsobené pohybom a nie sú náročné na prenos dát. Všetky tieto senzory fungujú len do určitej vzdialenosti.

1.3.4 Sensor fusion

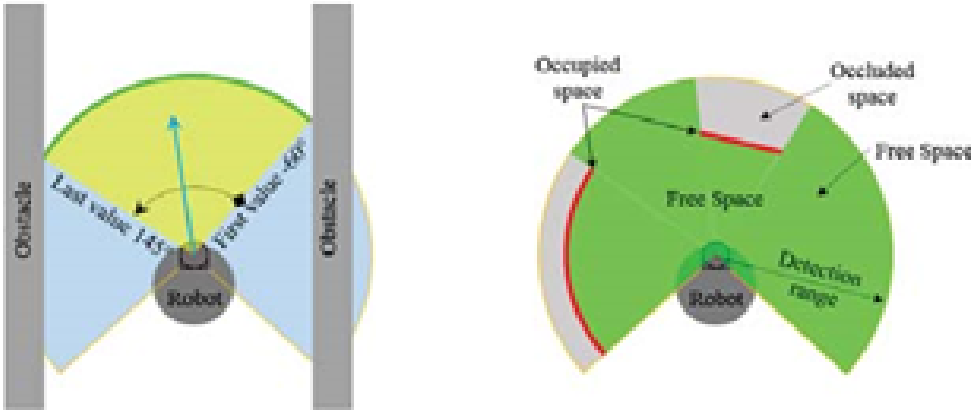
Senzory pracujú na rôznych princípoch a s využitím rôznych vlastností prostredia. Niektoré senzory snímajú zvukové, iné svetelné a ďalšie laserové vlny. Každý z týchto senzorov má nejaké výhody a nevýhody. Laserový alebo svetelný senzor napríklad zle sníma odrazenie vĺn od skla alebo priehľadných materiálov, pričom ultrazvukový senzor úspešne zachytí zvukovú vlnu po odrazení, no na druhú stranu má nevýhody napríklad v uhle snímania. Pre zjednotenie takýchto rôznych, aj nepresných, meraní sa využíva zlúčenie meraní senzorov na základe rôznych kritérií.

1.4 Predchádzajúce riešenia

V tejto sekcii si predstavíme podobné riešenia, ktoré nás inšpirovali pri vytváraní našej práce.

1.4.1 Navigácia mobilného robota s použitím 2D lidaru

Táto práca [CSV18] sa zaoberá navigáciou robota v chodbe pomocou 2D lidaru. V práci je predstavený spôsob navigácie v chodbe pomocou oblúkov, kde oblúk predstavuje priestor v určitej vzdialenosti, v ktorom sa nenachádza žiadna prekážka. Na vyhýbanie sa prekážkam sa využíva ultrazvukový a infra senzor. Pre vyhladenie dát z laserového senzora využívajú gaussov filter.



Obr. 1.13: Znázornenie oblúkov [CSV18].

1.4.2 Micromouse

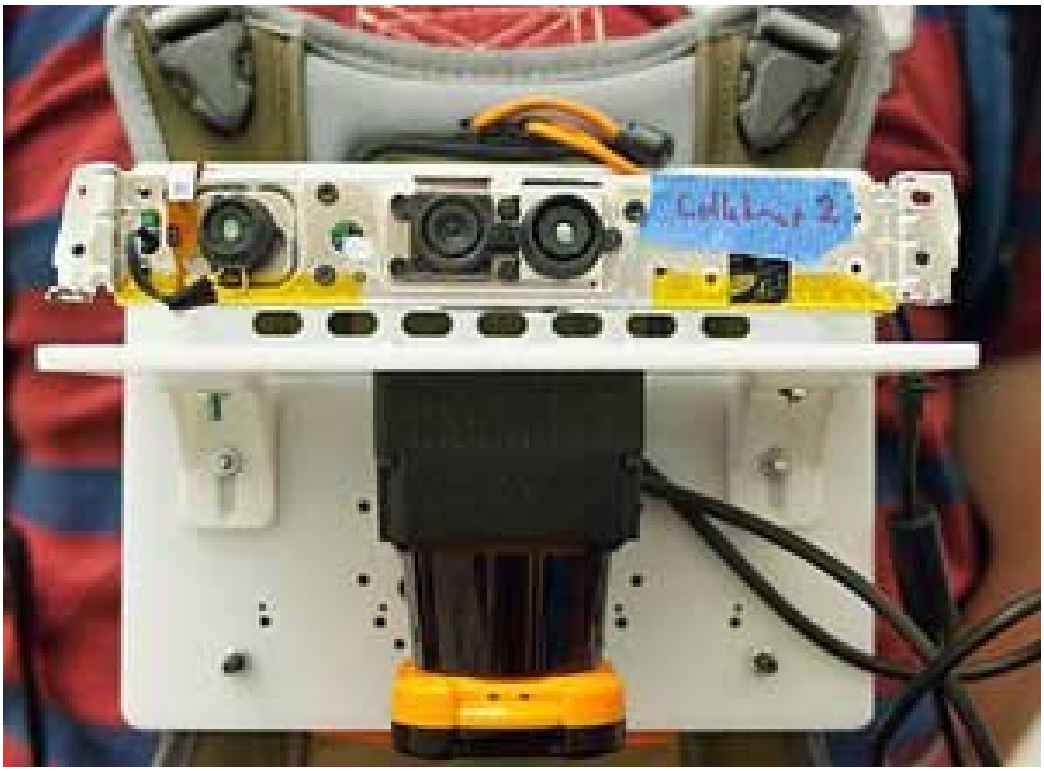
Micromouse je súťaž, v ktorej malý robot rieši bludisko veľkosti 16x16 buniek mriežky. Pointou súťaže je, aby sa robot zo začiatočnej pozície dostal do stredu bludiska sám, bez pomoci a čo najkratšou a najrýchlejšou trasou. Robot musí počas cesty vedieť, kde sa nachádza, objavovať steny a mapovať bludisko s detekciou možného príchodu do cieľa, ktorý je vždy v strede bludiska. Robot po príchode do cieľa začne ďalej objavovať bludisko a hľadať ďaľšie možné kratšie trasy od začiatočného bodu. Ďalej počas cesty môže robot prísť do križovatky, to je miesto kde sa dá ísť viacerými smermi. Robot si takúto križovatku zapamätá a následne sa rozhodne na základe naprogramovania, ktorým smerom pôjde, pričom sa vždy snaží ísť bližšie ku stredu. Jedným z možných riešení je využitie Flood-fill algoritmu popísanom v článku [ZPW14]. Je to algoritmus, ktorý prehľadáva všetky bunky nachádzajúce sa v prostredí. Na riešenie sú používané tiež Dijkstra alebo A* algoritmy.

Úloha v súťaži Micromouse je podobná - ide rovnako o mapovanie neznámeho prostredia, hoci v Micromouse je určený jasný cieľ. Pre potreby mapovania pavilónu je však užitočné práve rozhodovanie sa o tom, do ktorej

križovatky s dosiaľ nepreskúmanou odbočkou by sa robot mal vrátiť, tak, aby mapovanie celého pavilónu napredovalo čo najefektívnejšie.

1.4.3 Mapy vytvarané pre hasičov pomocou laserového senzora

V práci [Off] je popísané riešenie, ktoré má za úlohu pomáhať hasičom pri preskúmavaní budov. Cieľom je zmapovať budovy pomocou robota osadeného na výstroji, ktorý mapuje prostredie kadiaľ prešiel. Táto mapa má následne pomôcť pri rekonštrukcii budov v prípade ich zrútenia a následne pri hľadaní tiel. Na skenovanie prostredia použili laserový senzor. Jeden z problémov, ktorý riešili bolo, že pri osadení robota na výstroj hasiča nemajú dáta z otáčkových senzorov kolies robota, a tak museli pre spoľahlivejší chod osadiť rôzne ďalšie senzory ako sú senzor zrýchlenia, gyroskop a kameru. Pri ľudskom pohybe museli tiež filtrovať trasenie spôsobené chôdzou, hlavne na schodoch. Pre zaznamenanie zmeny výšok, v prípade zmien poschodia používajú barometer, ktorý sníma zmeny tlaku. Namiesto toho, aby hasiči museli nosiť so sebou robota ako výstroj, by autonómne mapovanie mohlo zvýšiť ich bezpečnosť a zlepšiť efektívnosť zásahu.



Obr. 1.14: Robot používaný na mapovanie budov [Off].

1.4.4 ROS

ROS alebo Robot Operating System [Sta] je open-source framework využívaný na písanie softvéru pre robota. Je to súbor nástrojov, knižníc a postupov s cieľom uľahčiť vytvorenie robota na rôzne účely. Je napísaný v jazykoch Python a C++. Má množstvo knižníc použiteľných pre rôzne platformy s rôznymi zameraniami.

Kapitola 2

Robot Mikeš

V nasledujúcej kapitole si popíšeme robota, s ktorým pracujeme. Predstavíme si súčiastky, ktoré využívame pre naše potreby mapovania ale aj tie, ktoré síce na robotovi osadené sú, ale rozhodli sme sa ich nevyužiť kvôli ich nepresnostiam v meraniach v našom prostredí.

Konštrukcia robota je z dreva a hliníka. Skladá sa z troch podlaží. Drevo tvorí podlahu jednotlivých podlaží a hliníkové profily ich spájajú. Na jednotlivých podlažiach a na hliníkovej konštrukcii sú namontované senzory a komponenty, ktoré sa používajú, alebo boli už v minulosti použité na rôznych projektoch a súťažiach. Mikeš sa zúčastnil v roku 2016 na súťaži *SICK Robot Day*, na ktorú bol pôvodne zostrojený. Neskôr, v roku 2018 sa znovu zúčastnil súťaže, odkiaľ už si odniesol cenu, ktorou je laserový senzor SICK TIM571. Prípravou robota na súťaž v roku 2018 sa zaoberá diplomová práca Dušana Matejku [Mat18], ktorej zameraním bolo mapovanie už známeho prostredia. Robot bol tiež využitý vo výučbe, kde témou bolo naprogramovanie Monte-Carlo lokalizácie a robot sa lokalizoval v priestore pomocou RFID tagou rozmiestnených po miestnosti [Kub].

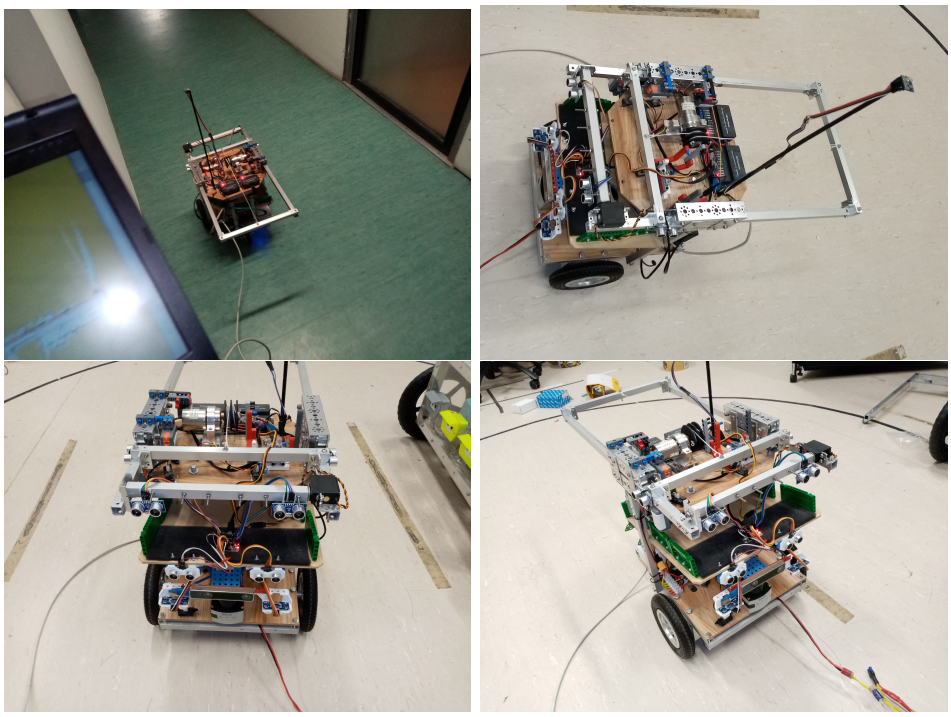
Riadiacou jednotkou robota je Raspberry PI 4 so štvorjadrovým 64-

bitovým ARM procesorom s frekvenciou 1.5 GHz a s veľkosťou operačnej pamäte 1 GB. Výstupnými portami zariadenia sú dva Micro HDMI porty na prenos obrazu v 4K rozlíšení, dva porty USB 3.0 a dva porty USB 2.0 a LAN port. Napájanie je riešené pomocou USB-C konektora. Využíva sa externá pamäť v podobe Micro-SD karty, kde je nainštalovaný operačný systém Raspbian 10 s verziou kernelu 4.19.75-v7l+.

Ďalšou súčasťou robota je Aruino NANO, ktoré nám slúži na komunikáciu medzi zariadeniami a riadiacou jednotkou. Arduino NANO komunikuje so zariadeniami pomocou analógových a digitálnych vstupov a výstupov. Na programovanie sa používa prostredie Arduino NANO IDE, ktoré priamo komunikuje so zariadením a zapisuje naň kód písaný v jazyku C++. Vývojové prostredie je jednoduché na orientáciu a má presne definované a jednoducho zapamätateľné funkcie pre každú potrebnú úlohu. V našom systéme máme osadené dva tieto Arduino Nano kontrolery. Jedno komunikuje s podvozkom a teda riadi pohyb. Ďalšie využívame na zber dát z ôsmich ultrazvukových senzorov.

Používame Ethernetový switch na rozšírenie LAN konektorov. Raspberry disponuje len jedným LAN konektorom. Na komunikáciu s lidarom sa používa LAN kábel a preto ich potrebujeme rozšíriť. Cez LAN konektor tiež komunikujeme pre efektívne testovanie a ladenie, pretože WiFi v našich priestoroch je nespoľahlivá a nemá dostatočné pokrytie v krajových úsekoch. Táto komunikácia je znázornená na obrázku 2.1.

Na napájanie robota používame 4000 mAh LiIon batériu, ktorá má výdrž približne dve hodiny na jedno nabitie pri aktívnom fungovaní robota. Nachádza sa pod spodným poschodím.

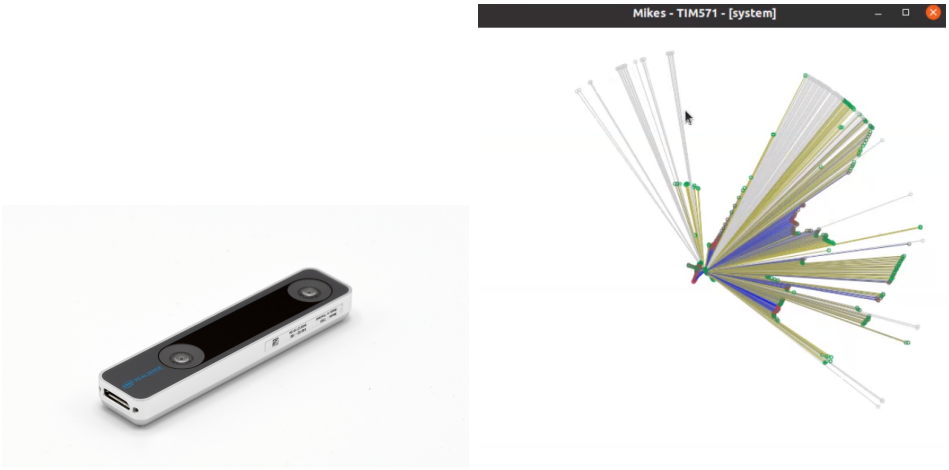


Obr. 2.1: Vľavo hore názorná ukážka sledovanie stavu mapovania počas testovania pomocou pripojenia cez LAN kábel, na ostatných obrázkoch je robot fotený z rôznych uhlov, pre lepšiu predstavu celej zostavy.

2.1 Orientácia v priestore

Odometria nám hovorí o smere a pohybe robota. Robot disponuje dvoma motormi, ktoré poháňajú dve predné kolesá. Každé koleso má svoj otáčkomer, ktorý sníma jeho otáčanie a podľa toho sa dá vypočítať aj smer, ktorým sa robot otáča. Má obmedzenú presnosť kvôli tomu, že si nevie poradiť s prešmykovaním kolies na klzkejšom povrchu, alebo s miernymi nerovnosťami na podklade, ktoré môžu viesť k chybe.Nezohľadňuje tiež konštrukčné nepresnosti robota, ktoré spôsobujú, že sa chyba po dlhšom pohybe akumuluje. Ďaľším zariadením, ktoré nám slúži na snímanie pohybu robota je sledovacia kamera *T265* od Intelu. Výrobca udáva len 1 % odchýlku na jedno uzavretie

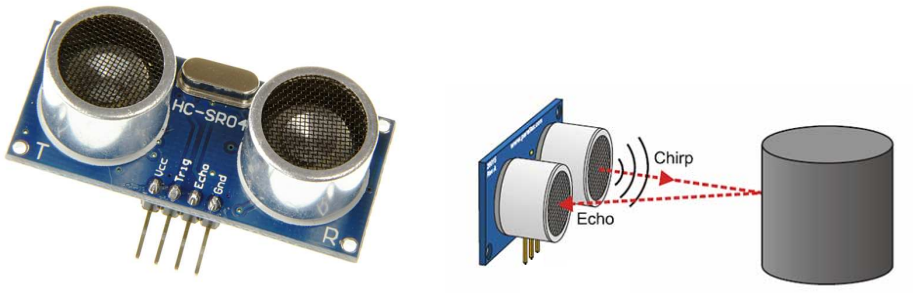
cyklu pri ideálnych podmienkach, pričom ideálnymi podmienkami by mali byť: dostatok svetla a málo pohybujúcich sa objektov v zornom poli, čo sme si aj pri testovaní overili, že pri nedostatku svetla už jeho presnosť lokalizácie klesala. Zariadenie disponuje dvoma fish-eye šošovkami, ktoré dokážu snímať šírku priestoru až 163° a na základe tohto obrazu zariadenie rozpoznáva svoju polohu. Aj napriek tomu, že kamera sníma obraz, nie je vhodná na snímanie objektov a prekážok v priestore. Nasledujú Kompas a GPS lokátor, ktorých dáta žiaľ vďaka nášmu prostrediu nie sú pre nás akceptovateľné. GPS lokátor v interiéri nefunguje príliš spoľahlivo a Kompas s radiátormi na chodbách, plechovými rámami skríň, dverí a učební v chodbách pavilónu má príliš veľké rušenie.



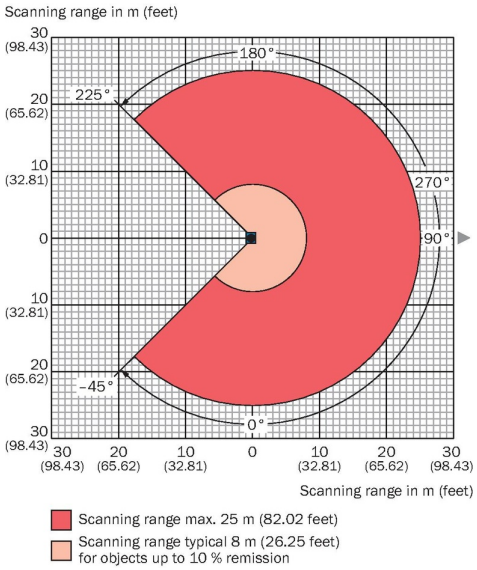
Obr. 2.2: Vľavo sledovacia kamera T265 s dvoma fish-eye šošovkami. Zariadenie nie je príliš veľké, no dokáže robiť všetky výpočty samostatne, takže neuberá výkon samotnému robotovi, vpravo zobrazenie lúčov senzora TIM571. Farba lúčov zodpovedá kvalite odrazu, čo je ovplyvnené povrchovými vlastnosťami prekážky. Lúče, ktoré sa vôbec neodrazia sú znázornené sivou farbou a predstavujú tiež určitý typ užitočnej informácie - v danom smere sa nenachádza žiadna viditeľná prekážka v dosahu senzora. To sa prejavovalo v 40m dlhých chodbách pavilónu nepretržite. Farebné znázornenie priesečníkov s prekážkou podobne zodpovedá sile odrazu - červené body sú najlepšie viditeľné, zelené slabšie a sivé nevidno. Silu odrazu (RSSI) poskytuje senzor v každom meraní.

2.2 Snímanie okolitého priestoru

Na snímanie priestoru využívame dva typy senzorov. Jedným z nich je laserový skener *TIM571*, ktorého uhol snímania je 270 stupňov, takže dokáže snímať takmer celé okolie robota, avšak len v jednej rovine. Základom jeho fungovania je laser, ktorého odraz od nejakého predmetu sa sníma. Jeho presnosť teda závisí od odrazivosti materiálu daného objektu. Napríklad sklo nie je vhodným na odrážanie laserových lúčov, pretože väčšina z nich sklom prejde, alebo sa odrazí a nasníma vzdialenosť objektu po odraze. Na takýto povrch vieme využiť ultrazvukové senzory. Na robotovi máme osadených osem ultrazvukových senzorov v rôznych smeroch a výškach pre čo najlepšie snímanie prekážok a následné ošetrenie kolízií. Ultrazvukový senzor *HC-SR04* osadený na robotovi sníma do vzdialenosti až štyroch metrov pod uhlom 15 stupňov. Na spojenie dát oboch senzorov, aby dávali čo najrealistickejšie dáta vzhľadom na vlastnosti okolitých objektov, využívame techniku zlúčenia senzorov popísanú v predošlej kapitole. 1.3.4



Obr. 2.3: Ultrazvukový senzor HC-SR04, znázornenie princípu odrazu zvuku. Jedna sonda zvukovú vlnu vysiela, druhá sníma jej návrat a na základe času trvania sa určí vzdialenosť od objektu.



Obr. 2.4: TIM571

Obr. 2.5: Laserový senzor TIM571, vyobrazenie vzdialenosti a rozptylu snímania priestoru. Senzor dokáže snímať objekty až do 25 metrov

2.2.1 Upgrade

Raspberry PI 4

Na osadenie a fungovanie T265 bol potrebný port USB 3.0. V pôvodnej zostave bolo Raspberry PI 3, ktoré obsahuje len porty USB 2.0 a tak sme ho nahradili Raspberry PI 4, kedy sme tiež v rámci práce museli všetko nakonfigurovať, aby správne fungovali napríklad obe Arduina, ktoré komunikujú cez USB porty.

Ultrazvukové senzory

Počas našej práce sme na robota tiež nainštalovali 8 nových ultrazvukových senzorov. Na predné spodné dva senzory sme tiež na niekoľko pokusov

vyrábali a prerábali ich držiaky, aby sme dostávali čo najlepšie hodnoty pri narazení na vyvýšenu alebo zníženú prekážku, ako sú napríklad schody, ktoré nemusia byť zachytené ani vo výške laserového senzora ale robot cez tieto vyvýšené miesta nedokáže prejsť. Pri osádzaní ultrazvukových senzorov sme tiež narazili na problém, kedy sme potrebovali rozlíšiť tieto zariadenia medzi sebou, žial sa v ničom nelíšili. Vyriešili sme to tým, že sme mali dostupné ešte jedno Aruino NANO, ktoré už sme rozlíšili pomocou výrobného čísla a mohli sme mu v systéme priradiť port.

2.3 Riadiaca architektúra robota

Riadiaca architektúra robota vychádza z myšlienok Behavior-Based Robotics [AA+98], čiže ju tvorí množina paralelne bežiacich správaní - alebo modulov, pričom každý z nich má svoju samostatnú úlohu alebo sadu zodpovedností, môže pristupovať k senzorom, alebo riadiť motory. O nízkoúrovňovú obsluhu motorov a senzorov sa stará jednočipový mikropočítač Arduino Nano, ale v rámci riadiacej architektúry mu zodpovedá modul "base module". Moduly komunikujú posieľaním správ - každý modul produkuje dátové pakety v definovanej štruktúre. Jednotlivé moduly sa môžu prihlásiť na odber určitých typov správ, alebo si aktuálne platné dáta osobitne vyžiadať. K vytváraným modulom ich autori typicky píšu testy, ktorými ich čiastkovú funkcionalitu vedia otestovať a preto je možné prípadnú porochu niektorého komponentu rýchlo a prakticky diagnostikovať. Systém má centrálnu podporu logovania, kde sa ukladá množstvo debugovacích výpisov, ktoré sa ukladajú vždy do nového súboru, pričom posledný záznam je jednoduché vyvolať a prezrieť jediným príkazom. Keďže moduly sú výpočtovými vláknami v rámci toho istého procesu, v prípade potreby môžu rozsiahlejšie dáta jednoducho zdieľať

namiesto kopírovania, čím sa dosahuje vysoká efektivita. Systém monitoruje bežiace moduly a pri ukončení behu programu všetky korektne povypína. Systém je rozdelený do dvoch repozitárov: common - ktorý osahuje knižnicu modulov, z ktorých je možné stavať architektúru pre každú špecifickú aplikáciu a generic - v ktorom sa vytvára (podľa šablóny) samostatný projekt pre každú jednu aplikáciu zvlášť. Systém má jednotný systém konfigurácie, v ktorom je jednoduché moduly aktivovať alebo deaktivovať. Na rozdiel od komplikovaných systémov ako je ROS je tento framework jednoducho udržiavateľný, netrpí nekompatibilitou pri každom väčšom upgrade systému a má len minimum nevyhnutných závislostí na iné knižnice. Na vizualizáciu používa rýchlu grafickú 2D knižnicu Cairo.

Kapitola 3

Návrh riešenia a implementácia

V tejto kapitole sa budeme venovať návrhu a implementácii nášho problému. Popíšeme si jednotlivé moduly, z ktorých sa náš systém skladá.

Prehľad

Kód je open-source zavesený na github [RaDoAIa][RaDoAIb]. Je napísaný v jazyku *C*, kvôli jeho rýchlosti, a je rozdelený do dvoch častí, ktorého rozdelenie bolo súčasťou diplomovej práce Dušana Matejku.[Mat18]

- *mikes-common*
- *mikes-generic*

Repozitár $mikes - common$ sa považuje za jadro Mikeša a skladá sa zväčša z vzájomne nezávislých modulov. Nachádzajú sa tam základné moduly na komunikáciu so senzormi, modulmi pohybu, logovacím systémom. Repozitár $mikes - generic$ slúži pre jednotlivé projekty, ktoré využívajú moduly z repozitára $mikes - common$.

Jednotlivé moduly sa ešte delia na aktívne a pasívne na základe toho, či na nich beží aktívne vlákno. Aktívne moduly sú také, ktoré napríklad neustále

čakajú na nové údaje zo senzorov, ktoré následne spracúvajú a na základe týchto spracovaných dát musia aktívne reagovať. Pasívne moduly slúžia na podporu aktívnych, kde sú uložené funkcie na výpočty, prípadne rôzne dátové štruktúry. K pasívnym patria tiež moduly na vykresľovanie aktuálnych dát zo senzorov, prípadne zobrazenia aktuálneho stavu systému robota.

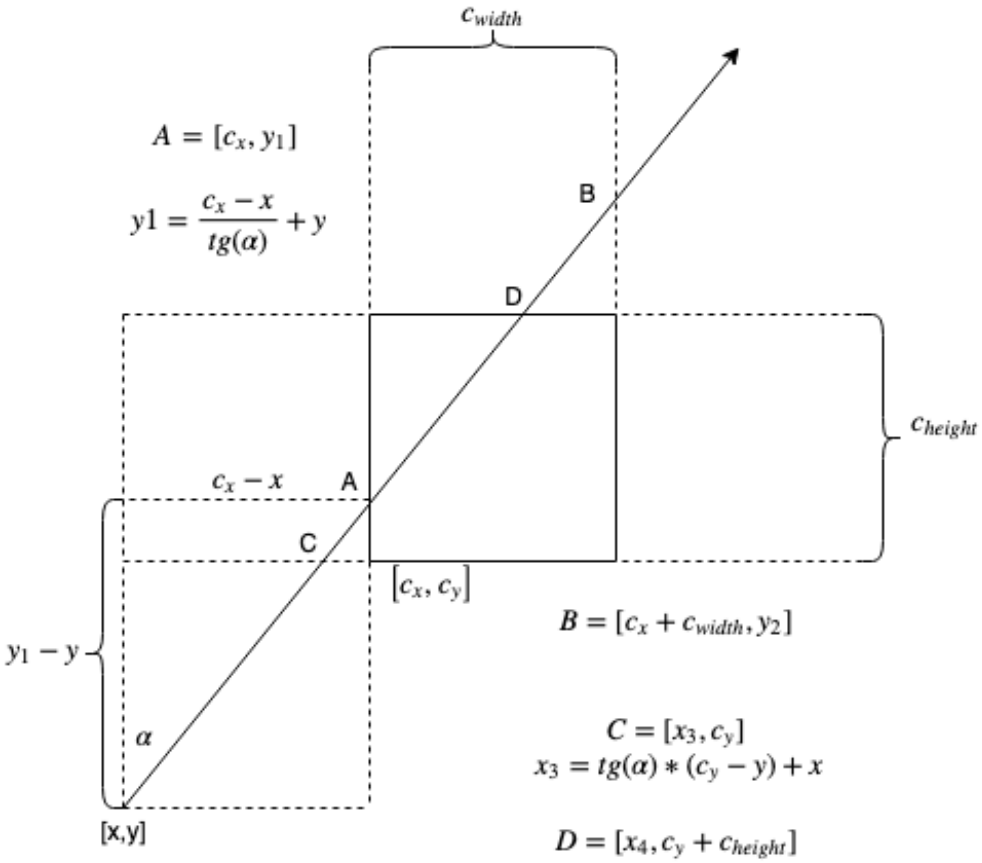
V našom riešení sme sa venovali hlavne modulom:

- Aktívne moduly
  - gridmapping
  - mapping_navig
  - t265

Tieto moduly si môžeme predstaviť ako tie z konceptu mapovania na obrázku 1.1. Gridmapping predstavuje mapovanie, mapping_navig predstavuje navigáciu a t265 lokalizáciu.

3.1 Reprezentácia mapy

Mapu sme sa rozhodli reprezentovať pomocou gridmapy. V gridmape je priestor rozdelený na rovnako veľké štvorčeky alebo bunky, ktoré prislúchajú nejakej skutočnej veľkosti. Do gridmapy si ukladáme hodnoty zo senzorov, ktoré snímajú okolité prostredie. Z laserového senzoru dostávame hodnoty vzdialeností jednotlivých lúčov od prekážky, od ktorej sa odrazia. Vieme teda, že v danom smere sa prekážka pravdepodobne nachádza až vo vzdialenosti, ktorú senzor vráti. Priestor medzi robotom a prekážkou môžeme v našej mape označiť ako prázdny a tiež si označíme bunku, kde sa potenciálne nachádza prekážka ako obsadenú. V nasledujúcich riadkoch si popíšeme jednotlivé potrebné výpočty zobrazené na obrázku 3.1



Obr. 3.1: Výpočet prevádzaný na dátach z laserového senzora využívaný pri vypĺňaní našej gridmapy.

Algoritmus spočíva v tom, že zisťujeme, či lúč končí v bunke, alebo cez ňu prechádza. V prípade, že cez bunku lúč len prechádza, pretína ju v dvoch bodoch a keď v nej končí, bude mať iba jeden vstupný bod. Poznáme pozíciu začiatku lúča (pozíciu robota) bod $[x, y]$ a smer lúča, uhol α a pozíciu rohu bunky $[c_x, c_y]$ a dĺžku lúča d. Poznáme tiež veľkosť jednej bunky, šírku označenú ako c_{width} a výšku c_{height}. Chýbajúce súradnice bodov

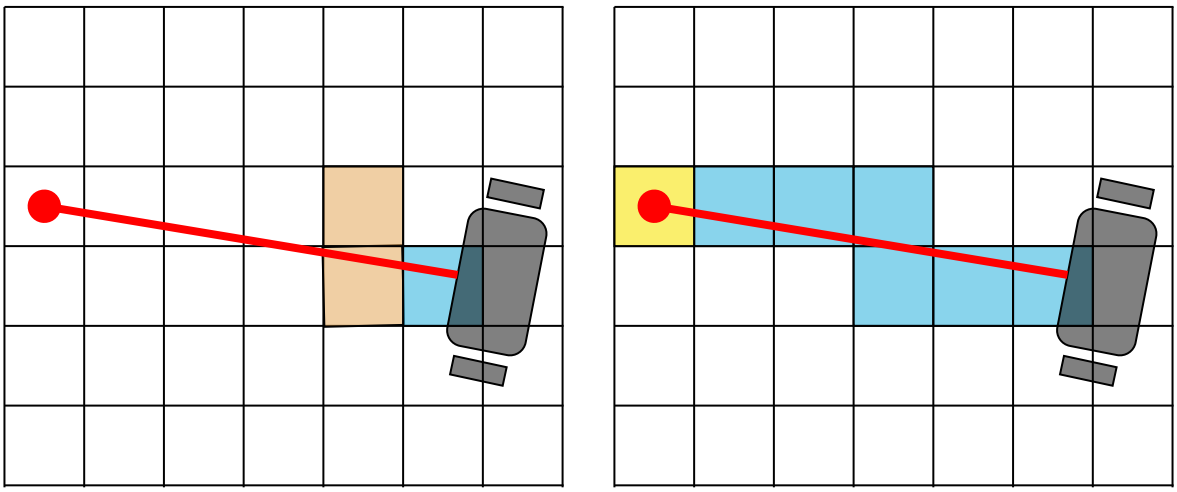
- $A = [c_x, y_1]$
- $C = [x_3, c_y]$
- $B = [c_x + c_{width}, y_2]$
- $D = [x_4, c_y + c_{height}]$

vypočítame nasledovne:

$$y_1 = \frac{c_x - x}{\tan(\alpha)} + y$$
$$x_3 = \tan(\alpha)(c_y - y) + x \tag{3.1}$$

Podobne vypočítame aj y_2 a x_4 s pripočítaním veľkosti bunky k ich prislúchajúcim súradniciam. Následne pomocou pytagorovej vety spočítame vzdialenosti týchto bodov od robota. Ak je vzdialenosť bodu väčšia ako dĺžka lúča, môžeme bunku označiť ako obsadenú.

V gridmape sme si zvolili mierku 1pixel : 10cm, teda 1pixel predstavuje 10cm v skutočnosti, čo je pre naše testovanie v rámci jedného pavilónu postačujúce. Mapu sme si rozdelili do dvoch dvojrozmerných polí: $grid_{empty}$ a $grid_{occupied}$. V $grid_{empty}$ ukladáme počet, koľkokrát sa na jednotlivých súradniciach neobjavila žiadna prekážka a v $grid_{occupied}$ zas koľkokrát senzor prekážku zaznamenal. Princíp je popísaný v predošlej kapitole v časti 3.1.

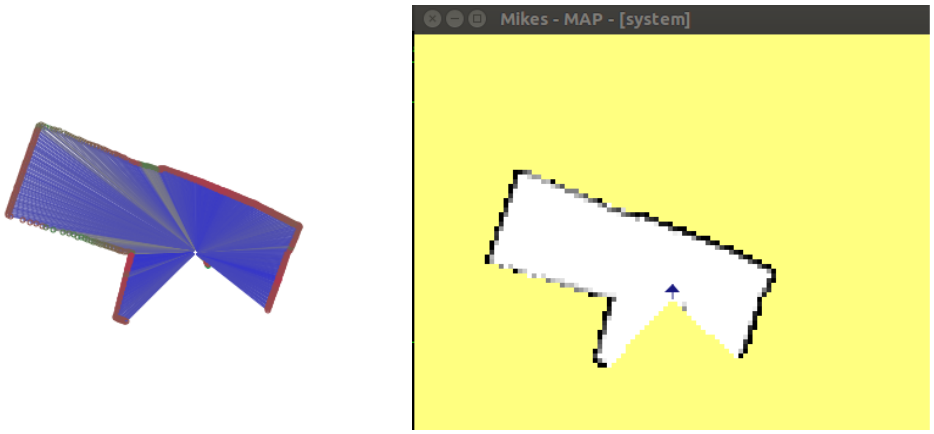


Obr. 3.2: Názorná ukážka laseru smerujúceho k prekážke. Modré polia sú miesta, ktorými laser preletel. V žltom poli laser končí a tak ho môžeme označiť ako obsadené.

V smere laseru kontrolujeme dve políčka, susediace k tomu predošlému,

či cez ne laser prechádza. Ak laser cez políčko prechádza, zvýšime hodnotu v $grid_{empty}$ na prislúchajúcej pozícii o jedna. Pokračujeme, kým neprídeme na koniec lasera, kde zvýšime hodnotu v poli $grid_{occupied}$ na danej pozícii tiež o jedna. Pre určenie pravdepodobnosti výskytu prekážky na pozícii $[x, y]$ vydelíme hodnotu z poľa $grid_{occupied}$ súčtom hodnôt políčok z oboch polí. Čím vyššia je výsledná hodnota, tým väčšia je pravdepodobnosť výskytu prekážky na danej pozícii v mape.

Na vizualizáciu mapy využívame k nej prislúchajúci modul. Pri vykresľovaní v tomto module obidve polia navzájom spriemerujeme a na základe priemeru sa im pridelí intenzita farby. Jeden pixel na obrázku zobrazuje charakter jednej pozície v našej gridmape.



Obr. 3.3: Vľavo: vizualizácia dát laserového senzora, Vpravo: Vyobrazenie mapy - biela určuje prázdny priestor, čierna prekážku, žltá nepreskúmané územie a šípka znázorňuje pozíciu a smer robota.

3.2 Ultrazvuky

Pre účely testovania sme navrhli a zostrojili dve verzie získavania dát z ultrazvukov.

V prvej verzii pristupujeme k ultrazvukom postupne, vo vopred zvolenom

poradí. Vyšleme signál najskôr z jedného ultrazvuku a čakáme určitý čas, kým sa zvuk neodrazí späť k senzoru. Po uplynutí času alebo po zaznamenaní odrazu pokračujeme ďalším senzorom v poradí.

V druhej verzii vysielame všetky lúče naraz, čakáme a kontrolujeme všetky senzory a keď na niektorý príde spätná odozva, tento senzor už nekontrolujeme a zaznamenáme si čas návratu na daný senzor. V tomto postupe môže byť potenciálne množstvo vzájomného rušenia ultrazvukov, keďže vysielame všetky lúče naraz, môže sa stať, že si ultrazvuky budú vzájomne zachytávať zvukové vlny. Predpokladáme však, že takéto vzájomné interferencie budú buď ohlasovať s väčšou pravdepodobnosťou prekážku v ich danom smere, prípadne zachytia odrazenú vlnu, ktorá už prešla väčšiu vzdialenosť a takéto merania odfiltrujeme.

Jednou z ďalších vecí, ktorú chceme odfiltrovať sú práve náhodné odrazy vĺn z iných senzorov alebo odrazy z predošlého cyklu merania. Zatiaľ čo predošlé možnosti prístupu k dátam robíme priamo na Arduine, do ktorého sú senzory zapojené, filtrovanie riešime už v kóde robota.

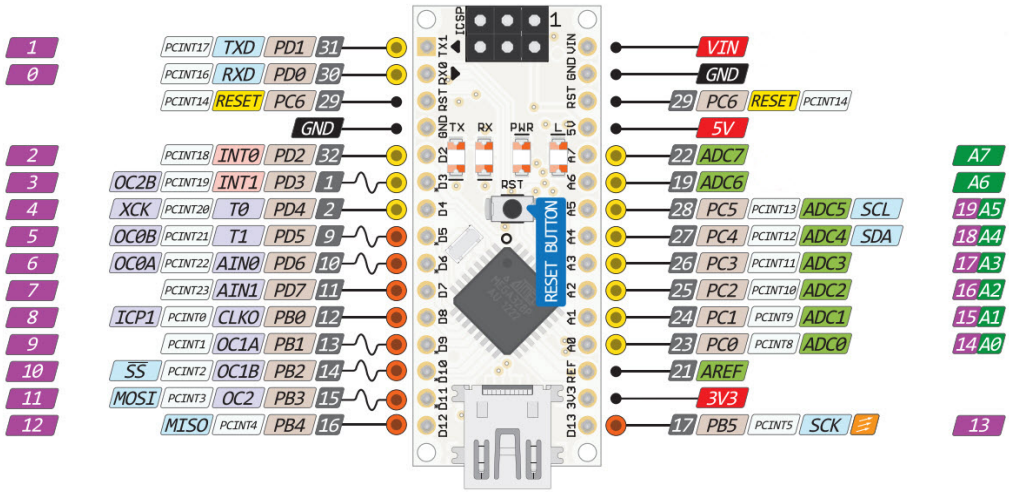
V algoritme si prejdeme každý senzor po jednom, kedy mu povieme, aby vyslal signál a následne čakáme kým sa signál vráti v určitom časovom limite. Vrátime si vypočítanú vzdialenosť v centimetroch. Experimentovali sme s dvoma hodnotami WaitTime - 20 ms a 40 ms, prvá zodpovedá odrazom zo vzdialenosti 3.3m a druhá zo vzdialenosti 6.6m, aby nasledujúci ultrazvuk ovplyvnil predchádzajúce merania, musel by teda prísť odraz zo vzdialenosti vyše 6.5m.

Podobne postupujeme aj v prípade paralelného prístupu, kde vyšleme signál na začiatku všetkým naraz a následne v cykle prechádzame jednotlivé porty a kontrolujeme jeho návrat. Pre zefektívnenie času medzi jednotlivými úkonmi, v snahe aby boli jednotlivé signály vyslané a prijaté v čo najkratšom

Algorithm 3.1 ultrasonic

```
1: function MEASURE(index)
2:     trigger[index];
3:     t := time();
4:     while !echo[index] do
5:         wait for echo on port
6:         if t > waitTime then return −1;
7:         end if
8:     end while
9:     return (time() − t)/58;
10: end function
1: function ROUNDTRIP()
2:     t := time();
3:     for i = 0 to 8 do
4:         while time() < t + waitTime ∗ i do
5:             wait
6:         end while
7:         m[i] := measure[i];
8:     end for
9: end function
```

časovom odstupe, nepristupujeme k portom cez indexy, čo by viedlo k ďalšiemu prechádzaniu poľa, ale dá sa s nimi komunikovať priamo. Ku každému portu prislúcha index v poli, v ktorom sa pri volaní cez index hľada príslušný port. Priamo s portami pracujeme pomocou logických operácií, v ktorých zisťujeme a nastavujeme ich stav. Schému si môžeme pozrieť na obrázku 3.4, kde indexu 15 prislúcha port $PC1$. Pripojenie senzorov na jednotlivé piny je popísané v zdrojovom kóde pre Arduino.



Obr. 3.4: Nízkoúrovňové označenie portov a ich pinov pre zodpovedajúce čísla pinov Arduino sme určili zo schémy rozloženia pinov jednočipového počíča Arduino Nano.

V testovacích podmienkach v laboratóriu fungovali ultrazvuky 100% spoľahlivo v obidvoch verziách programu, avšak v chodbách pavilónu sa pri všetkých našich snahách vyskytovali nežiadúce odrazy. Bolo to spôsobené napríklad tým, že steny chodieb nie sú v skutočnosti steny, ale skrine, ktoré sú ohraničené zvislými železnými rámami, ktoré takto tvoria kovové priečky s odrazivou plochou niekoľkých centimetrov. Podobne sú z kovu vytvorené vyčnievajúce rúčky na dverách. Je známe, že zvuk sa od kovových predmetov odráža veľmi zvučne a tak tieto odrazy prichádzali v nevhodných časoch aj s

viacmetrovej vzdialenosti, z akej by senzory bežne prekážku nezosnímali. Do tejto zbierky náročných prvkov prostredia je možné zaradiť aj drevené mreže pred radiátormi, ktoré boli rovnako kolmé na smer pohybu robota. Hlavný zámer teda bol pomocou ultrazvukov vyriešiť problémy s priesvitnými stenami pavilónu, ktoré tvoria okná a tiež s priesvitnými dverami, ktoré sú takmer všetky sklenené. Problém s dverami bol tým pomerne úspešne odstránený - pri jazdách bez ultrazvukov robot stále narážal do sklenených dverí a po namontovaní ultrazvukov sa pohyboval prakticky priamo - akonáhle sa k dverám priblížil dostatočne blízko, aby ich bočné ultrazvuky zachytili. Problém s priesvitnými oknami bol vyriešený iba čiastočne, pretože sa nachádzali v širších častiach chodby, za inými zložitými objektami, s ktorými spoľahlivým zachytením už ultrazvuky mali ťažkosti. Možným riešením do budúcnosti by teda zrejme bolo až spracovanie obrazu, alebo 3D hĺbkovej mapy.

3.3 Sensor fusion

Sensor fusion alebo zlúčenie senzorov sme použili pre lepšiu orientáciu robota v prostredí. Vďaka tomuto modulu by mal mapovať prostredie s úspešným vyhýbaním sa okolitým prekážkam. Spojili sme dáta z laserového senzora a ultrazvukových senzorov. Laserový senzor je umiestnený v strede v prednej časi robota. Ultrazvukové senzory sme umiestnili tak, aby dokázali čo najlepšie pokryť prednú, najdôležitejšiu stranu, ale aj bočné strany. Dopredu sme umiestnili šesť týchto senzorov, na boky po jednom senzore. Pre lepšiu predstavu si môžeme robota pozrieť na fotografii 2.1 v predošlej sekcii. V nasledujúcich riadkoch si popíšeme, ako takéto zlučovanie prevádzame.

Rozmiestnené ultrazvukové senzory máme na presných určených pozíciach, aby sme vedeli presne sčítavať a zlučovať dáta z oboch typov senzo-

rov. Poznáme teda relatívnu pozíciu ultrazvukových senzorov ku laserovému senzoru. Na obrázku 3.5 je vizualizácia potrebných výpočtov jednotlivých premenných. Náš laserový senzor sa v našom prípade nachádza na pozícii $O[0,0]$ a ultrazvukové senzory na pozíciach $[X,Y]$, ktorých hodnoty máme presne dané. D je výstupná hodnota z ultrazvukového senzora, teda vzdialenosť od prekážky a maximálny uhol, v ktorom je ultrazvuk schopný detegovať prekážku je 30°. W je šírka oblúka vo vzdialenosti prekážky od senzora. Keďže ultrazvukový senzor neudáva šírku prekážky, musíme brať celú šírku oblúka ako prekážku. Šírku vieme vypočítať pomocou kosínusovej vety $c^2 = \sqrt{(a^2 + b^2 - 2abcos(\alpha))}$, keďže naše $a = b = D$, môžeme vzorec upraviť nasledovne:

$$W = \sqrt{2D^2 - 2D^2 \cos(30°)}$$

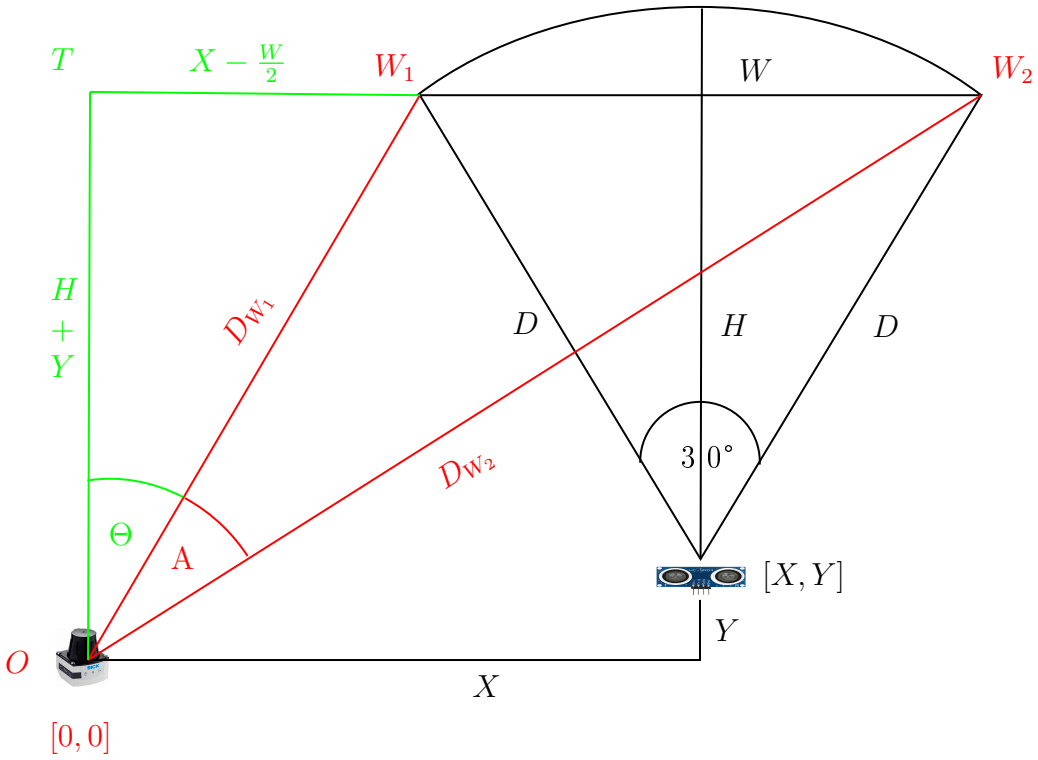
Ďalšími premennými, ktoré potrebujeme vypočítať sú dĺžky lúčov smerujúcich z laserového senzora smerom k šírke pokrytia ultrazvukového senzora D_{W_1} a D_{W_2}. Tie vypočítame určením vzdialeností bodov W_1 a W_2 od laserového senzora na pozícii $O[0,0]$. Na to ešte potrebujeme zistiť vzdialenosť šírky od ultrazvukového senzora - premennú H. H vypočítame pomocou pytagorovej vety z pravouhlého trojuholníka so stranami $\frac{W}{2}DH$.

$$H = \sqrt{D^2 - \frac{W}{2}^2}$$

$$W_1 = [X - \frac{W}{2}, Y + H], D_{W_1} = \sqrt{((X - \frac{W}{2}) - 0)^2 + ((Y + H) - 0)^2}$$

$$W_2 = [X + \frac{W}{2}, Y + H], D_{W_2} = \sqrt{((X + \frac{W}{2}) - 0)^2 + ((Y + H) - 0)^2}$$

Keď poznáme dĺžky oboch lúčov, vieme vypočítať uhol A z trojuholníka so



Obr. 3.5: sensor fusion

stranami D_{W_1} D_{W_2} W.

$$\measuredangle W_1OW_2 = \mathrm{A} = \cos(\frac{D_{W_1}^2 + D_{W_2}^2 - W^2}{2D_{W_1}D_{W_2}})^{-1}$$

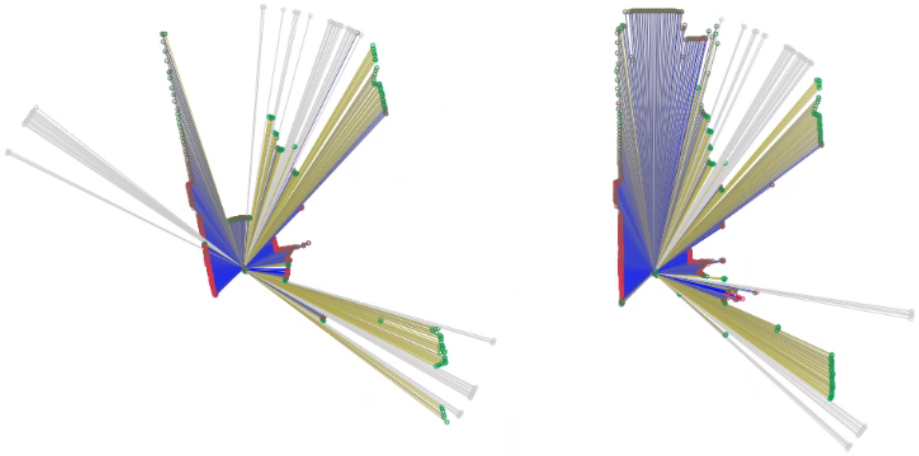
Podobne vieme vypočítať aj uhol Θ, ktorý zviera najbližší lúč s azimutom.

$$\measuredangle TOW_1 = \Theta = \cos(\frac{(Y+H)^2 + 2D_{W_1}^2 - (x - \frac{W}{2})^2}{2D_{W_1}(Y+H)})^{-1}$$

Keď k uhlu Θ pripočítame smer natočenia ultrazvukového senzora zistíme prvý uhol laserového senzora zasahujúci do časti, ktorú sníma ultrazvukový senzor. Na jeden stupeň prislúchajú tri laserové lúče. Nazvime si ich ako množinu R, pričom $\|R\| = 3\|\mathrm{A}\|$. R_i je lúč z množiny R na indexe i. Každému R_i priradíme prislúchajúcu vzdialenosť od nameranej prekážky z ultrazvu-

kového senzora. Prislúchajúcu vzdialenosť vypočítame pripočítaním rozdielu vzdialenosti prvého a posledného lúča vydeleného počtom lúčov a vynásobeného aktuálnym indexom, ku vzdialenosti prvého lúča. Tento výpočet je znázornený v rovnici 3.2.

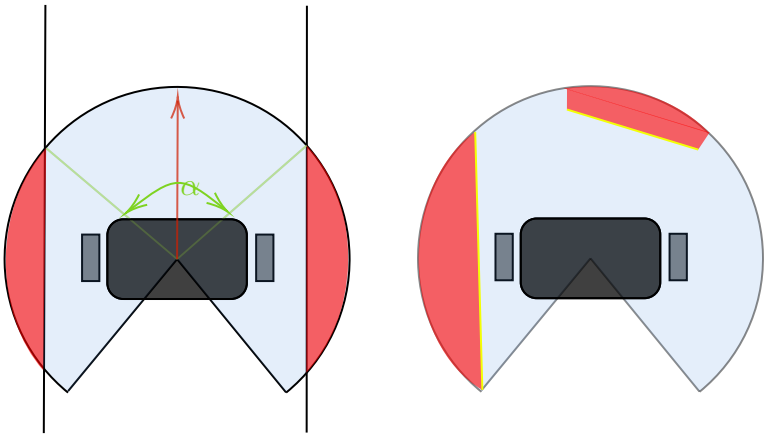
$$\forall R_i = \|D_{W_1}\| + \frac{\|D_{W_2}\| - \|D_{W_1}\|}{\|R\|} i, R_i \in R \tag{3.2}$$



Obr. 3.6: Naľavo je zobrazenie aktuálneho stavu okolia s integrovaním metódy sensor-fusion, Napravo je zobrazenie lúčov zo senzora TIM571.

3.4 Lokálna navigácia

Lokálnou navigáciou rozumieme navigáciu robota podľa dát získaných v danom momente. Táto navigácia spočíva v hľadaní oblúkov v určitej vzdialenosti od robota. Oblúk je prázdny priestor vo zvolenej vzdialenosti od robota tvorený z lúčov laserového senzora. Lúče, ktoré sú dlhšie ako naša zvolená vzdialenosť v postupnosti za sebou môžeme nazvať šírka alebo uhol oblúka. Lúče, ktoré sú v menšej vzdialenosti môžeme považovať ako prekážku v danom uhle a smere lúča. Pre pohyb sa rozhodujeme medzi nájdenými oblúkmi



Obr. 3.7: Ukážka oblúkov pre lokálnu navigáciu. Červená farba znázorňuje prekážku, bledomodrá prázdny priesto a zelenou je znázornený uhol oblúka.

v rôznych smeroch a veľkostiach. V našom prípade volíme stred najvhodnejšieho lúča pre smer pohybu. Ďalšou z možností bolo zvoliť najdlhší z lúčov v danom oblúku ako smer pohybu, čo by ale bolo optimálne pre riešenie, kde záleží na efektivite pohybu. V našom prípade volíme stred práve preto, lebo predpokladáme, že stred chodby je stredom oblúka, čo môžeme vidieť na obrázku 3.7 vľavo. Stred chodby chceme voliť preto, lebo zo stredu je najlepšie vidieť väčšinu prekážok v okolí robota. Po pridaní metódy *sensorfusion* do riešenia môžeme princíp oblúkov zakomponovať tiež v podobe určenia vzdialenosti od prekážky na základe ultrazvukového senzora a laserového senzora. Dáta z oboch zariadení porovnáme a zvolíme vždy tú kratšiu vzdialenosť, ktorú jeden zo senzorov udáva. Princíp navigácie po aplikovaní metódy sensor fusion je veľmi podobný ako len so samotným laserovým senzorom. Keď si predstavíme dáta zo senzorov transformované na lúče, môžeme rovnako kontrolovať prekážky na podobnom princípe.

3.5 Plánovanie trasy v už známom prostredí

Najvhodnejší algoritmus na hľadanie trasy pre nás bol A*, popísaný v algoritme 3.3. Tento algoritmus si vyberá bunky, ktoré prehľadáva na základe vypočítanej heuristickej funkcie h, počtu krokov, teda prejdených buniek od začiatku g a hodnotou f vypočítanou súčtom hodnôt g a h. My sme si zvolili heuristickú funkciu vypočítať z Euklidovskej vzdialenosti bunky od cieľa a zároveň, aby sa trasa tiahla stredom chodby, sme zvolili pridať výpočet vzdialenosti bunky od steny a túto vzdialenosť pripočítať ku Euklidovskej.

struct *cell* {
parent;
f, g, h; }
struct *node* {
pos;
f;
node **next*; }

Pre výpočet trasy sme si zvolili dátovú štruktúru *front* nazvanú v našom prípade štruktúrou *node*. Front je lineárna dátová štruktúra s výberom prvkov v rovnakom poradí, v akom boli vložené. Funguje teda na princípe $FIFO$ (first in first out) - prvý dnu, prvý von. Máme k nemu dve jednoduché funkcie *enqueue*(p), pričom p je prvok, ktorý táto funkcia do frontu vloží na koniec a *dequeue*() vyberie prvok zo začiatku. Pre správne fungovanie potrebujeme dve premenné odkazujúce na prvý a posledný prvok frontu. Jednotlivé prvky v našom fronte sú prehľadávané bunky v danom poradí.

Ďalšou štruktúrou, ktorú sme si definovali je štruktúra *cell*, ktorá nám reprezentuje jednotlivé bunky mapy a ukladá si vypočítané hodnoty f,g,h a susednú bunku, z ktorej na ňu prešiel.

Algorithm 3.2 A* tracePath

```
function TRACEPATH(cells, goal)
    path = {};
    while cells[goal].parent not goal do
        path.add(cells[goal]);
        goal := cells[goal].parent;
    end while
    return path.reverse();
end function
```

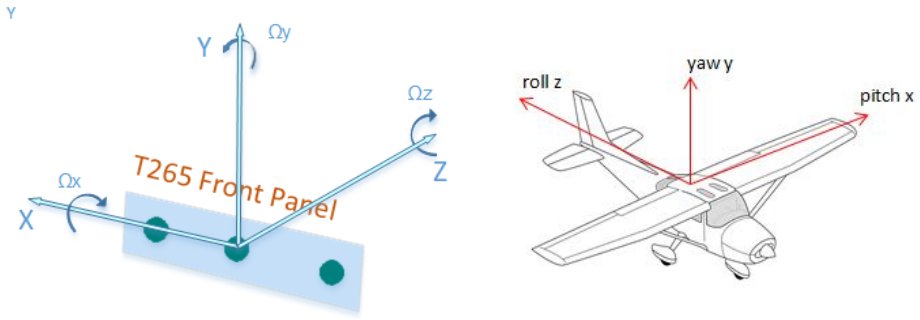
V algortime si definujeme pole *visited*, ktoré predstavujú už navštívené bunky, ktoré už viac nebudeme chcieť navštevovať a pole *cells* štruktúry *cell* o rozmeroch gridmapy. V prvom cykle prejdeme okolitými ôsmimi bunkami od štartovacej bunky. V nich si ohodnotíme jednotlivé premenné f,g,h a následne zvolíme ten s najmenšou hodnotou f. Toto opakujeme až kým neprejdeme všetkými bunkami, alebo sa nedostaneme do cieľa. Keď nájdeme cieľ, vstúpime do funkcie *tracePath*, kde zrekonštruujeme danú trasu od cieľa po začiatok. Následne pole prevrátime, aby sme mali trasu v poradí od začiatočnej bunky po cieľovú.

3.6 Odometria

Pre zisťovanie odometrie využívame dáta z kamery Intel T265. Na komunikáciu s týmto zariadením sme si vytvorili modul T265. T265 nám poskytuje dáta o pohybe pri ich zmenách. Na výpočet smeru zmeny sme využili kvaternión rotácie, ktorý si vieme tiež vypýtať z už zabudovanej funkcie kamery. Súradnicovú sústavu kamery a Eulerove uhly na príslušných obrázkoch 3.8 sme museli zlúčiť do rovnakej reprezentácie a vytvoriť si vzorec na výpočet smeru otáčania.

Algorithm 3.3 findPathInGridmap algoritmus

```
1: function FINDPATHINGRIDMAP(gridmap, start, goal)
2:     cells := {} * mapWidth{} * mapHeight;
3:     visited := {};
4:     nodes := queue of struct node;
5:     cells[start].parent := start;
6:     nodes.enqueue(start, f = 0.0);
7:     while not nodes.isempty() do
8:         x, y := nodes.front().pos;
9:         visited.add([x, y]);
10:        for i := x - 1 to x + 1 do
11:            for j:= y - 1 to y + 1 do
12:                if [i, j]! = [x, y] and cellValid(i, j) then
13:                    if [i, j] = goal then
14:                        cells[i, j].parent := [x, y];
15:                        return tracePath(cells, goal);
16:                    else if [i, j] not in visited then
17:                        g := cells[x, y].g + 1;
18:                        h := calcH();
19:                        f := g + h;
20:                        if cells[x,y].f > f then
21:                            nodes.enqueue(f, [i, j])
22:                            cells[i, j].{f, g, h, parent} := f,g,h,[x,y];
23:                        end if
24:                    end if
25:                end if
26:            end for
27:        end for
28:        nodes.dequeue();
29:    end while
30:    return {};
31: end function
```



Obr. 3.8: Vľavo súradnicová sústava T265, vpravo štrandard yaw, pitch, roll.

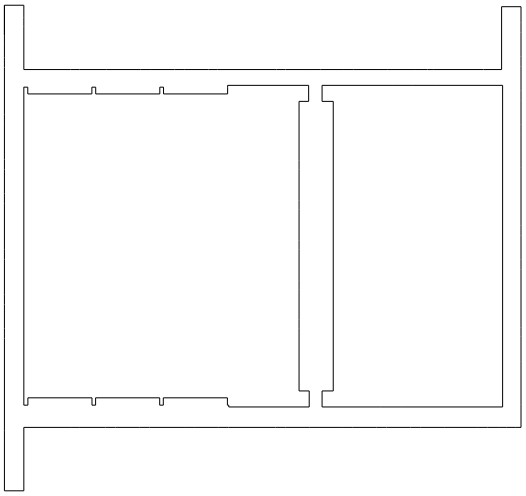
Máme kvaternión $q = w, x, y, z$ a z neho sme vytvorili vzorec na zistenie smeru *yaw* z:

$$smer = \arctan \frac{2wy - 2xz}{x^2 + y^2 - w^2 - z^2} \tag{3.3}$$

Kapitola 4

Výsledky

Počas práce sme testovali naše riešenie v pavilóne Informatiky vyobrazenom na obrázku 4.1.

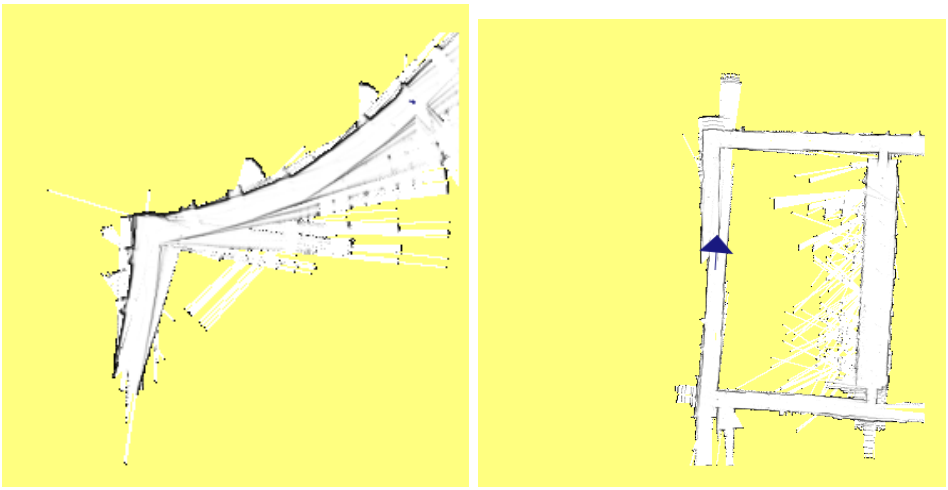


Obr. 4.1: Ručne (meracím pásmom) vytvorená mapa pavilónu Informatiky na FMFI UK.[Mad14]

Prvým riešením, ktoré zobrazovalo nejaké výsledky je zobrazené na obrázku 4.2 vľavo. Tam môžeme vidieť ako sa mapa zvlnila a keď začal robot v ľavej časti a išiel smerom doprava, je vidno, že sa mapa ťahá doľava. To bolo spôsobené odometriou a ľavým kolieskom. Zistili sme, že ľavé koliesko má

tendenciu ísť pomalšie, aj keď je mu dodávaný rovnaký prúd ako pravému a preto, keď sme aj brali odometriu z kolies, boli výsledky nedostačujúce.

Rozhodli sme sa preto pridať nové zariadenie, kameru Intel T265, ktorá dávala dostatočne presné hodnoty odometrie. Na obrázku 4.2 vpravo môžme vidieť zmapovanie jedného cyklu prostredia, kde je vidno, že sa robot trochu odchýlil, čo bolo spôsobené pravdepodobne jednak obmedzeným osvetlením v niektorých miestach mapy, jednak monotónnosťou chodby s veľkými bielymi plochami, ktoré v určitých výhľadoch z robota môžu obsahovať menej oporných vizuálnych bodov a jednak samotnou obmedzenou presnosťou zariadenia RealSense. Takéto cykly by sa dali vyriešiť implementáciou algoritmov na ich uzatváranie.



Obr. 4.2: Vľavo prvá mapa vytvorená ešte za pomoci otáčkových senzorov používaných ako odometriu, vpravo zmapovanie jednej slučky.

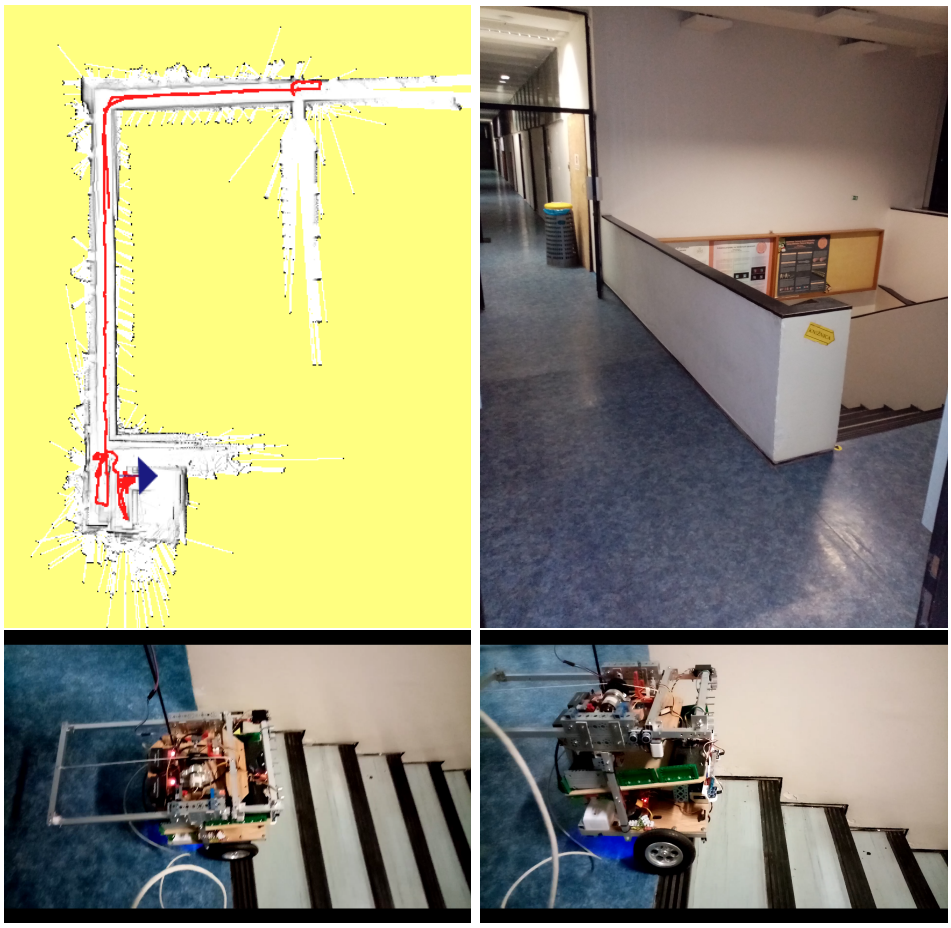
Jednou z možností by bolo rozdeliť si mapu na segmenty, po ceste sledovať a pamätať si body záujmu a následne po uzavretí cyklu, respektíve po navštívení miesta, kde už nejaké body záujmu poznáme, skontrolujeme pozíciu a vzdialenosť daných bodov od robota a následne postupne upravíme jednotlivé segmenty mapy v určitom pomere k prejdenej vzdialenosti

a zapamätáme si posun robota pre ďalšie správne mapovanie. Body záujmu môžeme určiť niektorou štandardnou metódou vyhľadávania význačných bodov v obraze, alebo aj porovnaním nájdených rohov - priesečníkov stien v lokálnom okolí robota s vytvorenou mapou, ktoré už boli naprogramované v predchádzajúcej práci a teda sú súčasťou už existujúcich modulov robota.

Obr. 4.3: Pohľad na finálnu verziu nastavenia spodných ultrazvukov spolu s obrázkom držiakov vytvorených pre 3D tlač.

Do ďalšej verzie sme pridali vykresľovanie trajektórie robota, pre lepšiu predstavu, ako robot mapoval a pre lepšie spätné zistenie, čo mu robilo

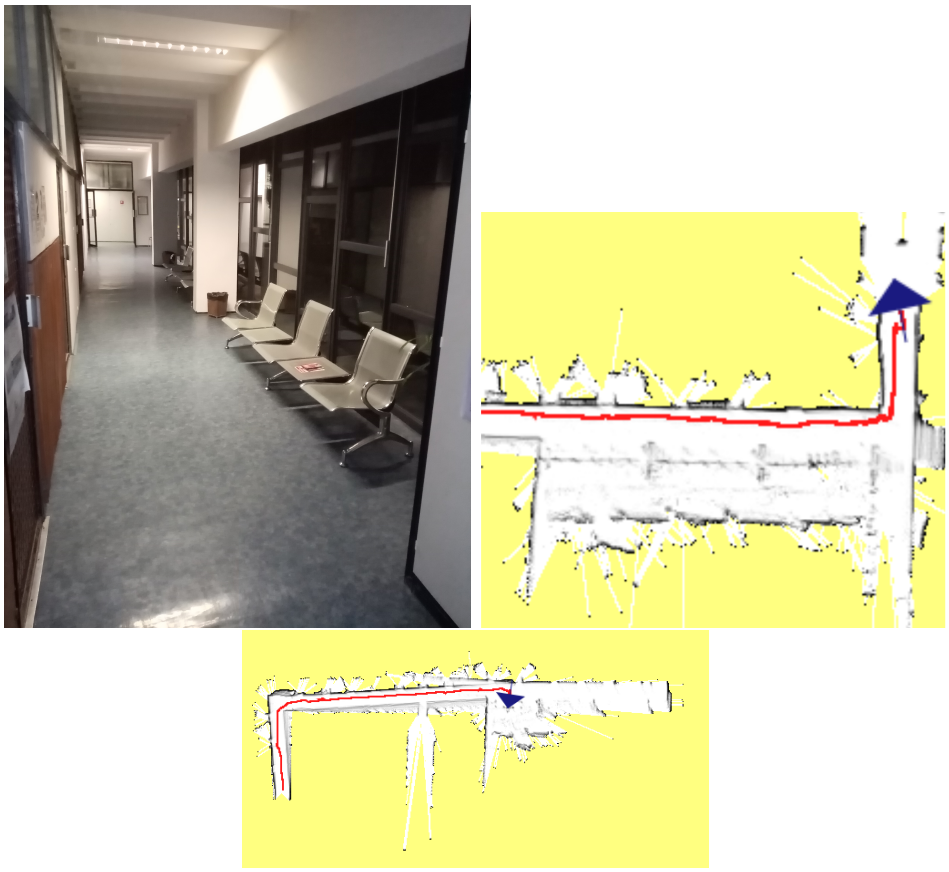
problémy. Pridali sme tiež ultrazvuky, ktoré sledujú priestor pred a z oboch bokov robota a jeho spodok v prednej časti kvôli možným kolíziám na prahoch prípadne pádom zo schodov. Dva senzory zobrazené na obrázku 4.3, ktoré sme umiestnili na sledovanie prednej spodnej časti sme dlhšiu dobu nastavovali, tlačili na 3D tlačiarni a následne prispôsobovali. Pre najlepšie výsledky meraní sme ich umiestnili takmer kolmo dole, pretože viac boli natočené dopredu, tým viac sa zvuk odrážal smerom preč od robota a senzory ho nezachytili. Dosiahli sme to, že aktuálne v bežnej situácii vracajú stabilné hodnoty vzdialenosti od zeme 20cm. Keďže sú senzory nasmerované tesne pred robota, potrebovali sme ešte otestovať, či robot stihne zareagovať pri nejakej prekážke. Test bol úspešný a robot stihol zastaviť. Výsledok je zobrazený na obrázku 4.4.



Obr. 4.4: Vľavo hore mapa vytvorená po ceste smerom ku schodom, schody sa na nej nachádzajú v spodnej časti viac vpravo; nasleduje fotografia pohľadu na schody a spodné dva obrázky zobrazujú kritickú situáciu robota, ako stihol zastaviť na hrane.

Ďalším problémom, na ktorý sme narazili boli okná na celú výšku bočnej steny chodby v pavilóne, za ktorou sa nachádza átrium. Pri testoch v noci sa laserový infračervený lúč lidaru od okna dvakrát odrazil a vytváral tak fantómovú stenu za oknom. Naopak cez deň bolo IR svetelné pozadie na oveľa vyššej hodnote a to zrejme spôsobovalo, že laserový lúč sa lepšie odrážal od predmetov v átriu za oknom, alebo vôbec. Toto bol zároveň jeden z hlavných dôvodov, prečo sme robota vybavili sadou ultrazvukových senzorov, avšak

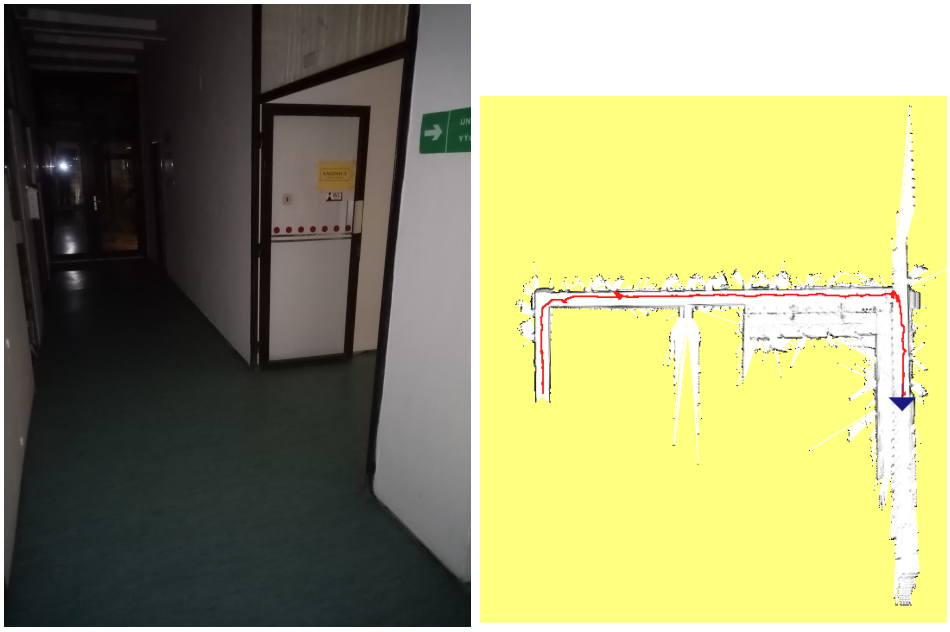
situácia v chodbe - kovové, drevené i čalúnené lavice, veľké odpadkové koše, stoly, stĺpy a výklenky - to všetko spolu je pre ultrazvukové senzory priveľmi komplikovaná scenéria na to, aby spoľahlivo naviedli robota do pokračovania chodby, keď sa práve za oknom javí perspektívnejší smer pre pokračovanie mapovania. Na problém sme narazili náhodou pri poslednom testovaní. Vždy predtým sme testy prevádzali v noci, keď bola vonku tma a nikdy sa nestalo, že by sa robot vybral smerom k oknám. Síce aj v noci robot videl ďalej, ako bolo samotné okno, videl v ňom zrkadlovo odraz chodby, po ktorej išiel a nemal tendenciu ísť smerom k oknám. Na obrázku 4.5 je vidno, ako robot úspešne držal smer jazdy v noci v chodbe so sklenenými oknami, ale cez deň mu to robilo problémy.



Obr. 4.5: Vľavo je zobrazenie priestoru, kde robot smeroval cez deň von, vpravo mapa vytvorená s odrazom chodby od skla aj so zaznamenaním hraníc skla, v spodnej časti je zobrazenie, ako sa robot snaží smerovať von, ale ultrazvukové senzory ho tam nevpustia.

Posledným problémom, ktorý žiaľ komplikoval mapovanie bolo padanie modulu senzoru T265 z dôvodu neaktualizovania obrazu priamo v senzore, keď bola v priestore tma. Snažili sme sa priestor osvetliť LED lampou z mobilného telefónu, ale v mieste, kde boli vypálené žiarivky na chodbe to nepomáhalo. Svietiace žiarivky boli od seba vzdialené viac ako 10 metrov a T265 má zabudovaný časovač 15 sekúnd, po uplynutí ktorého ak nepríde ďalšia použiteľná snímka preruší vykonávanie celého programu. Test do tmavého priestoru sme skúšali previesť ešte dva krát z rôznej strany, avšak vždy prog-

ram skončil kvôli chybe senzoru. Intenzitu svetla rovnako aj mapu vytvorenú po tmavú časť si môžeme pozrieť na obrázku 4.6.



Obr. 4.6: Vľavo je fotografia tmavej chodby, napravo mapa so skončením v danej chodbe po viacerých pokusoch.

Vytvorené mapy dokumentujú, že robot sa takmer počas celej jazdy dokázal pohybovať autonómne bez nášho zásahu, úspešne vyberať zákruty, aj nástrahy v podobe priesvitných dverí kancelárií - cez ktoré presvitali lúče lidaru - ako vidieť napr. na vytvorenej mape na obrázku 4.5. V niektorých prípadoch boli tieto efekty natoľko rušivé, že situáciu zvládol len s ťažkosťami, ale predsa len ju zvládol - ako vidno na červenej trajektórii za prvou zákrutou na obr. 4.6. Druhý zauzlený červený bod na tej istej mape však už v tejto jazde nezvládol (bolo to pred plechovými dverami počítačových hál) a museli sme mu pomôcť manuálnym zásahom. Zároveň to však dokumentuje obrovskú výhodu vizuálnej odometrie v porovnaní s otáčkovými senzormi, lebo aj po manuálnom natočení, prípadne posunutí robota je poloha odometrie presná a mapovanie môže bez akýchkoľvek problémov presne pokračovať.

Záver

Cieľom práce bolo navrhnúť a vytvoriť systém na lokalizáciu a mapovanie neznámeho priestoru. Takýto systém s jednotlivými modulmi sme úspešne vytvorili. Prišli sme aj na niekoľko nedostatkov, ktoré sme sa snažili odstrániť. Niektoré sa však nedali odstrániť vzhľadom na povahu zvoleného algoritmu a tiež povahu systému senzorov.

Výsledkom je teda úspešné autonómne zmapovanie prostredia s hodnovernými výstupnými mapami, ktoré boli však rozdelené na viaceré časti kvôli problémovému úseku, kde k dokončeniu bránil problém so senzorom RealSense a tmavé prostredie.

Na lokalizáciu a mapovanie využívame senzor Intel T265, ktorý nám udáva polohu robota. Vytváranie mapy je realizované transformovaním dát z laserového senzora do našej mapovej reprezentácie- gridmapy. Na lokálnu navigáciu používame údaje z laserového a ultrazvukových senzorov, ktorých dáta upravujeme a spájame metódou sensor fusion a s tými následne manipulujeme. Pre smer pohybu v priestore hľadáme v týchto dátach takzvané oblúky, na základe ktorých sa následne robot rozhoduje, ktorým smerom sa vydá. Na vyhľadávanie trasy pre návrat robota k ešte nenavštívenému miestu využívame upravený algoritmus A*, s využitím heuristickej funkcie vypočítanej zo vzdialenosti bodu od cieľa a zároveň vzdialenosti od steny, aby sa robot držal v strede chodby pre efektívne mapovanie.

Námetom na ďalšiu prácu a zlepšenie systému by bolo zakomponovanie uzavretia cyklov pre presnejšie vytváranie mapy. Problém so zvládaním krízových situácií v častiach s vysokými oknami by mohlo vyriešiť spracovanie obrazu, alebo 3D hĺbková mapa.

Literatúra

[73015] Ieee standard for robot map data representation for navigation. *1873-2015 IEEE Standard for Robot Map Data Representation for Navigation*, pages 1–54, Oct 2015.

[AA+98] Ronald C Arkin, Ronald C Arkin, et al. *Behavior-based robotics*. MIT press, 1998.

[All] Spencer Allen. Self-driving robot navigation methodology explained.

[BD19] Robert Bassett and Julio Deride. Maximum a posteriori estimators as a limit of bayes estimators. *Mathematical Programming*, 174(1-2):129–144, 2019.

[BDW06] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117, 2006.

[BEFW97] Johann Borenstein, Hobart R Everett, Liqiang Feng, and David Wehe. Mobile robot positioning: Sensors and techniques. *Journal of robotic systems*, 14(4):231–249, 1997.

[Bur] Wolfram Burgard. Efficient approaches to mapping with raoblackwellized particle filters.

[CCC+16] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J.J. Leonard. Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[CGKM07] Robert Oliver Castle, Darren J Gawley, Georg Klein, and David W Murray. Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4102–4107. IEEE, 2007.

[CHHR22] Nikolaus Correll, Bradley Hayes, Christoffer Heckman, and Alessandro Roncone. *Introduction to Autonomous Robots: Mechanisms, Sensors, Actuators, and Algorithms*. MIT Press, Cambridge, MA, 1st edition, 2022.

[CSV18] György Csaba, László Somlyai, and Zoltán Vámossy. Mobil robot navigation using 2d lidar. In *2018 IEEE 16th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 000143–000148, 2018.

[D+59] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[DWB06] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[HX20] Shuangquan Han and Zhihong Xi. Dynamic scene semantics

slam based on semantic segmentation. *IEEE Access*, 8:43563–43570, 2020.

[KFL01] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[KJR+12] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

[KL15] Soonkyum Kim and Maxim Likhachev. Path planning for a tethered robot using multi-heuristic a* with topology-based heuristics. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4656–4663, 2015.

[Kub] Tomáš Kubla. Monte carlo localization with robot mikes.

[L+98] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998.

[LM97] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[LS15] Yan Lu and Dezhen Song. Visual navigation using heterogeneous landmarks and unsupervised geometric constraints. *IEEE Transactions on Robotics*, 31(3):736–749, 2015.

[Mad14] Marína Madová. Robot poštár, 2014.

[MAMT15] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[Mat18] Dušan Matejka. Lokalizačný systém pre mobilného robota, 2018.

[Mor89] Hans P. Moravec. Sensor fusion in certainty grids for mobile robots. In *Sensor devices and systems for robotics*, pages 253–276. Springer, 1989.

[MTJ+07] Oscar Martinez Mozos, Rudolph Triebel, Patric Jensfelt, Axel Rottmann, and Wolfram Burgard. Supervised semantic labeling of places using information extracted from sensor data. *Robotics and Autonomous Systems*, 55(5):391–402, 2007.

[MWBDW02] A.A. Makarenko, S.B. Williams, F. Bourgault, and H.F. Durrant-Whyte. An experiment in integrated exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 534–539 vol.1, 2002.

[NLD11] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011.

[Off] MIT News Office. Laser rangefinder maps buildings for firefighters.

[PK94] David Pierce and Benjamin Kuipers. Learning to explore and build maps. In *AAAI*, volume 94, pages 1264–1271, 1994.

[RaDoAIa] Comenius University Robotics at Department of Applied Informatics. mikes-common.

[RaDoAIb] Comenius University Robotics at Department of Applied Informatics. mikes-generic.

[SHBG05] Cyrill Stachniss, Dirk Hähnel, Wolfram Burgard, and Giorgio Grisetti. On actively closing loops in grid-based fastslam. *Advanced Robotics*, 19(10):1059–1079, 2005.

[Sta] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.

[Ste93] Anthony Stentz. Optimal and efficient path planning for unknown and dynamic environments. *INTERNATIONAL JOURNAL OF ROBOTICS AND AUTOMATION*, 10:89–100, 1993.

[TB96a] Sebastian Thrun and Arno Bücken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the National Conference on Artificial Intelligence*, pages 944–951, 1996.

[TB96b] Sebastian Thrun and Arno Bücken. Learning maps for indoor mobile robot navigation. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1996.

[ZPW14] Hao Zhang, Lian Peh, and Ying Wang. Micromouse solve maze based on flood-fill algorithm. *Applied Mechanics and Materials*, 513-517:4227–4230, 02 2014.

Zoznam obrázkov