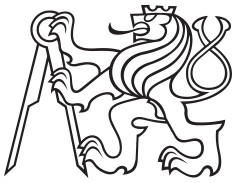
Diplomová práce



České
vysoké
učení technické
v Praze

F2

Fakulta strojní
Ústav Přístrojové a řídicí techniky

Autonomizace robotické platformy

Martin Hasal

Vedoucí: Ing. Cyril Oswald, Ph. D.
Obor: Automatizace a průmyslová informatika
Studijní program: Automatizační a přístrojová technika
Červen 2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hasal** Jméno: **Martin** Osobní číslo: **483974**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídící techniky**

Studijní program: **Automatizační a přístrojová technika**

Specializace: **Automatizace a průmyslová informatika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Autonomizace robotické platformy

Název diplomové práce anglicky:

Autonomization of the robotic platform

Pokyny pro vypracování:

Navrhněte systém autonomizace robotického podvozku pro pohyb v uzavřeném prostoru využitím hloubkové kamery RealSense.
- Proveďte rešerši dostupných senzorů pro orientaci robota v prostoru a srovnejte je s možnostmi hloubkové kamery.
- proveďte rešerši metod autonomizace, plánování a navigace robota vhodných uzavřené prostory.
- Návrhněte řešení autonomizace pohybu dle zadání za použití OS ROS na mikropočítači NVIDIA Jetson NX.
- Navržené SW řešení realizujte a simulačně otestujte
- Implementujte navržené řešení na reálné robotické platformě

Seznam doporučené literatury:

[1] Köseoğlu, M; et.al. Design of an autonomous mobile robot based on ROS, pp. 1-5, doi: 10.1109/IDAP.2017.8090199.
[2] Sariff N. and Buniyamin N. An Overview of Autonomous Mobile Robot Path Planning Algorithms. pp. 183-188, doi: 10.1109/SCORED.2006.4339335.
[3] Sánchez-Ibáñez, J.Ret.al. Path Planning for Autonomous Mobile Robots: A Review. https://doi.org/10.3390/s21237898

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Cyril Oswald, Ph.D. U12110.3

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **28.04.2023** Termín odevzdání diplomové práce: **08.06.2023**

Platnost zadání diplomové práce: _____

Ing. Cyril Oswald, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	doc. Ing. Miroslav Španiel, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____ Datum převzetí zadání

_____ Podpis studenta

Poděkování

Děkuji Ing. Cyrilu Oswaldovi, Ph. D. za věcné rady.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 8. června 2023

Abstrakt

Práce se zabývá autonomizací upraveného šestikolového robota založeného na projektu ExoMy. Řídicí jednotkou je Nvidia Jetson, k řízení je využit The Robot Operating System (ROS). Cílem autonomizace je úspěšná lokalizace robota v prostředí, zadání cílového bodu a nalezení optimální cesty v prostředí s překážkami.

Klíčová slova: RealSense, ROS, autonomní, mobilní, robot

Vedoucí: Ing. Cyril Oswald, Ph. D.

Abstract

The work deals with the autonomization of a modified six-wheeled robot based on the ExoMy project. The control unit is Nvidia Jetson, The Robot Operating System (ROS) is used for control. The goal of autonomization is to successfully localize the robot in the environment, enter the target point and find the optimal path in the environment with obstacles.

Keywords: RealSense, ROS, Autonomous, Mobile, Robot

Title translation: Autonomization of the robotic platform

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

Roboti se obecně používají k tomu, aby sloužili nebo pomáhali lidem v několika oblastech. Od každodenních prací až po průmyslové aplikace. Věda o robotech se v poslední době výrazně rozrostla díky požadavkům lidí a průmyslu. Jednou z hlavních oblastí vědy o robotech jsou mobilní roboti. Mobilní roboti jsou schopni navigovat v prostředí a interagovat s ním prostřednictvím senzorů a aktuátorů. Autonomní navigace je klíčovým prvkem, který umožňuje mobilním robotům samostatně a efektivně provádět úkoly. Jejím hlavním cílem je umožnit robotům pohybovat se a operovat v různých prostředích bez lidského dohledu. To otevírá možnosti pro využití robotů v průmyslu, logistice, zdravotnictví a dalších oblastech. Autonomní navigace se v praxi používá v mnoha situacích. Například autonomní mobilní roboty (AMR) lze využít v rozsáhlých skladech k přepravě zboží nebo k automatizaci logistických procesů. V oblasti zdravotnictví mohou autonomní roboti pomáhat při distribuci léků nebo při monitorování pacientů. Z osobního pohledu mě tato problematika autonomní navigace mobilních robotů fascinuje a motivuje mě se jí věnovat. Vidím v tomto směru velký potenciál pro vytváření robotických systémů, které mohou přinést výrazné přínosy pro společnost. Je třeba zdůraznit, že navrhování a vývoj autonomních mobilních robotů není jednoduchý úkol. Vyžaduje složitý design hardwaru a softwaru, který je schopen zpracovávat a interpretovat data z různých senzorů a akčních členů. Koordinace a souběžná činnost těchto systémů v reálném čase je klíčem k úspěchu autonomní navigace. Proto je důležité udělat si průzkum a porozumět principům a algoritmům, které umožňují mobilním robotům autonomně navigovat v nepředvídatelném prostředí.

Kapitola 2

Cíle práce

V této diplomové práci se věnuji navržením a implementováním autonomního mobilního robota, který dokáže vytvořit mapu vnitřního prostředí a zároveň se v něm pohybovat. Z hardwarového hlediska bych chtěl použít a upravit šestikolovou platformu Exomy navrženou European Space Agency, řídící jednotku Nvidia Jetson Tegra. Každý autonomní mobilní robot potřebuje senzoriku minimálně pro lokalizaci v prostoru, provedu tedy průzkum a výběr vhodných senzorů. K dispozici mám Realsense D455, který bych osobně rád vyzkoušel pro tuto oblast a zjistil zda je hoden využití. Za software chci použít The Robot Operating System (ROS). Cílem tedy bude upravit šestikolovou platformu Exomy, provést výběr vhodné senzoriky a následně spojit hardware a software, aby byl mezi sebou kompatibilní. Poté sestavit program pro úspěšnou lokalizaci robota v prostředí po zadání cílového bodu a nalezení optimální cesty v prostředí s překážkami. Postup návrhu a integrace budou podrobně vysvětleny dále.

Část I

Rešerše

Kapitola 3

Rešerše

3.1 Prostorové vidění

Počítačové prostorové vidění je způsob, pro analýzu a vyhodnocování užitečných informací, dle námi určených parametrů, ze získaných prostorových dat z okolí. Data jsou získávána jedním až několika senzory a jsou reprezentována převážně ve tvaru vizuálním RGB obrazu ve spojení s prostorovou hloubkovou mapou. Hloubková mapa je forma obrazu, kde hodnota každého pixelu představuje vzdálenost mezi senzorem a bodem, který daný pixel reprezentuje ve snímaném prostředí. Obraz s přidanou informací o vzdálenosti jednotlivých pixelů přidává na využití, jako rozpoznávání objektů, tvarů a jejich rozměrů nebo pro vytváření 3D hloubkových map prostorů. Všechny tyto aspekty se v dnešní době využívají v průmyslu pro automatizaci, autonomizaci robotů, vyhodnocování kvality nebo pro další úroveň bezpečnosti automobilů. Je několik způsobů jak získávat data pro prostorové vidění, které budou popsány v podkapitolách.[19][13]

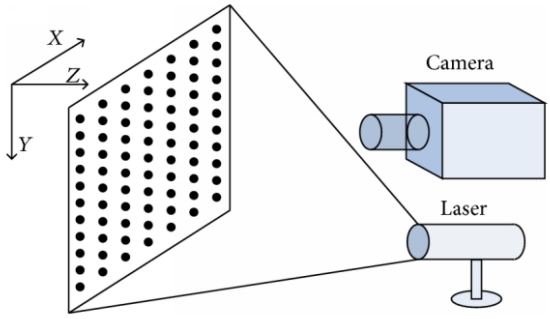
3.1.1 Stereo kamera

Stereo kamera je založená na stejném principu jako naše oči. Stereo kamera má dvě RGB kamery, které mají stejnou ohniskovou vzdálenost a jsou od sebe vzdáleny na danou vzdálenost. Tím má každá jinou perspektivu vidění, máme jako vstupní data dva RGB obrazy, které spojíme pomocí triangulace

a získáme podle použití buď hloubkovou mapu nebo vzdálenost jednotlivých bodů.[28]

3.1.2 IR kamera(strukturované světlo)

Infračervená neboli IR kamera využívá strukturované světlo. Daný senzor se skládá z infračerveného zářiče a jeho přijímače, který je umístěn vedle něho danou vzdálenost. Zářič vyzařuje síť bodů rovnoměrně rozprostřených a podle referenční roviny definovaný. Přijímač tuto síť snímá a podle vzdálenosti a nerovnoměrnosti prostředí jsou jednotlivé body různě vzdáleny od sebe než body referenční roviny. Podle těchto rozdílů se pomocí korelace určí vzdálenost bodů od senzoru a vytvoří se hloubková mapa. Tento typ senzoru má nevýhodu, že nedokáže detekovat průhledné plochy, jako skleněné a další infračerveným světlem průhledné materiály.[35]



Obrázek 3.1: Princip IR kamery[2]

Intel RealSense

Senzor RealSense je od firmy Intel. Vybraný model je D455, který se skládá ze dvou RGB a jedné IR kamery, je tedy schopen vytvořit hloubkovou mapu oběma způsoby. Infračervený zářič je vzdálen od infračerveného přijímače 95mm s čímž dosahuje nepřesnosti měření na vzdálenost 4 metrů na méně než 2%. Tato verze dokáže měřit hloubkovou mapu od minimální vzdálenosti 0,52 m a přesné snímání je mezi 0,6 m až 6 m od senzoru. Rozlišení hloubkové mapy je 1280 x 720 pixelů. Úhel zorného pole je 87° horizontálně a 57° vertikálně. Celkové rozměry tohoto typu senzoru je 124 x 26 x 29 mm o váze 390 g. Je napájen přímo z USB portu a není potřeba externí napájení. Pro zaznamenání změny pohybu je použit 6-ti osý senzor Bosch BMI055, který využívá akcelerometr s rozsahem ±4 g a gyroskop s rozsahem ±1000°/s.[3]

Obrázek 3.2: Kamera Realsense D455

3.1.3 Time of Flight (doba letu)

Time of Flight, ve zkratce ToF je metoda pro měření vzdáleností mezi senzorem a měřeným bodem. Je založena na principu rozdílů časů mezi vyslaným emitovaným signálem a jeho návratem po odrazu od měřeného objektu. Z těchto časů se vypočtou vzdálenosti jednotlivých pixelů, ze kterých následně vznikne hloubková mapa. Je několik signálů, které jsou použity pro tuto metodu. Nejčastěji se použito zvukové nebo světelné vlnění. Světelné vlnění má výhodu toho, že je rychlejší než zvuk, dosáhne větších vzdáleností a senzor má nízkou hmotnost. Použitím například infračerveného světla se zajistí redukce šumů okolí a je jednodušší ho rozlišit od okolního světelného znečištění. Zvukové jsou použity pro kratší vzdálenosti a pro místa, kde se mohou vyskytovat světlem průhledné materiály. ToF senzory využívající světelného vlnění se rozdělují na modulované světlo a pulzní světlo. Modulované světlo je světlo s periodickými časovými změnami intensity s frekvencí 10 až 100 MHz. Pulzní světlo používá jeden dlouhý pulz, pro zjištění vzdálenosti.[41][22][16][9]

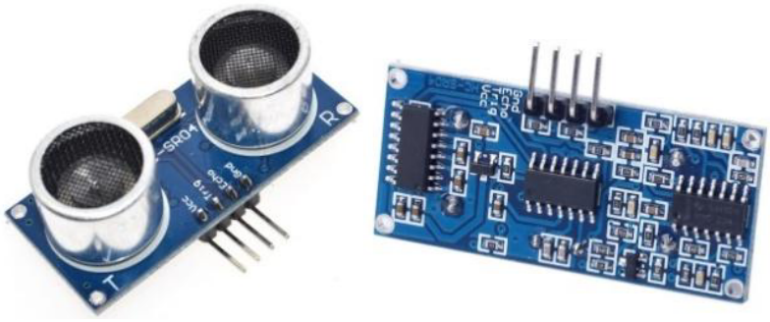
Lidar

Light Detection and Ranging , ve zkratce LIDAR, je v překladu světelná detekce a měření vzdálenosti. Lidar je ToF kamera využívající metodu modulovaného světla. Tento senzor je velice rychlý, přesný a má velký dosah. Je používán převážně do špičkových a komplexních technologií a to z důvodu vysoké pořizovací ceny. Používá se buď amplitudově modulované spojité vlny nebo frekvenčně modulované spojité vlny. Amplitudově modulované spojité vlny modulují intenzitu světla při zachování stejné frekvence. Frekvenčně modulované spojité vlny jsou založeny na změně frekvence vyřazovaného

světla. Tím je bod měřen několika frekvencemi a je potom výsledek přesnější. Podle požadovaných přesností měření je zapotřebí vysokorychlostní vysokofrekvenční elektronika, pro modulaci intenzity světla. U přijímače tento požadavek může být snížen, protože demodulační přijímač dokáže převést vysokofrekvenční signály do základního pásma.[31][38]

Ultrazvuk

Ultrazvuk má podobné charakteristiky šíření v prostředí jako slyšitelný zvuk. To je prostředí mechanických vibrací částic. Ultrazvuk se může šířit v plynných, kapalných a pevných látkách. Pro ultrazvuk je obecně považován zvuk o frekvenci vyšší než 20 kHz. Podle použití lze ultrazvuk rozdělit do dvou skupin: Aktivní ultrazvuk při aplikaci vykazuje fyzikální nebo chemické účinky. Generovaný výstup dosahuje vyšších hodnot. Ultrazvuk se používá k čištění, svařování, vrtání a podobně. Pasivní ultrazvukový výstup je generován při mnohem nižších (obvykle malých) hodnotách. Jeho hlavní oblastí použití je pak měření vzdálenosti, zjišťování vad materiálů, tloušťky materiálů, měření průtoku kapalin a plynů. Rychlost zvuku je závislá na typu prostředí, ve kterém se pohybuje, a na aktuální teplotě prostředí. Ultrazvukové senzory fungují na principu měření doby mezi vysláním obvykle několika velmi krátkých impulsů a přijetím odrazu vysílaného signálu. Základem senzoru je vysílač a přijímač. Mohou se skládat ze dvou typů převodníků: Magnetostrikčního měniče - pracují na nízkých frekvencích a jejich princip je založen na mechanické změně délky magnetického materiálu. Piezoelektrického měniče - pracují na vysokých frekvencích a princip je založen na inverzním piezoelektrickém jevu. Ultrazvukový přijímač je založen na principu přenosu mechanických vln odražených zpět na elektrický signál. Je zvažován ultrazvukový dálkoměr HC-SR04. Ultrazvukový dálkoměr HC-SR04 je znázorněn na obrázku 3.3, kde jsou vysílač a přijímač označeny jako T a R. Dálkoměr generuje zvukové vlny o frekvenci 40 kHz. Zvukové vlny se odrážejí od objektu a vrací se zpět do přijímače, senzor dává informaci o čase, který byl požadován pro šíření zvukových vln od senzoru k objektu a zpět. Takové snímače jsou levné a jsou schopny pracovat v dosahu až několika metrů, i když vznikají problémy s jejich přesností a chováním v hlučných venkovních podmínkách. Měřená vzdálenost je v rozsahu 0,1–0,3 m s nejistotou 1 mm v teplotním rozsahu 0°C až 40°C. Měření vzdáleností do 1 m a v širších teplotních rozsazích jsou možné, ale s vyšší nejistotou.[11][47][21]



Obrázek 3.3: Ultrazvukový dálkoměr HC-SR04 [47]

3.1.4 IMU

Inerciální měřicí jednotky (IMU) jsou zařízení, která se používají k měření rychlosti, orientace a gravitačních sil objektu. IMU se běžně používají v robotice, letectví a aplikacích virtuální reality. IMU se obvykle skládají z kombinace akcelerometrů, gyroskopů a magnetometrů, které měří lineární zrychlení, úhlovou rychlost a magnetická pole. Tyto senzory společně poskytují informaci o pohybu a orientaci objektu. Akcelerometry jsou senzory, které měří lineární zrychlení nebo změny rychlosti v čase. Typicky jsou založeny na technologii mikroelektromechanických systémů (MEMS) a mohou měřit zrychlení ve všech třech rozměrech. Gyroskopy na druhou stranu měří úhlovou rychlost nebo rotační pohyb. Dokážou detekovat změny orientace a jsou také založeny na technologii MEMS. Magnetometry, které měří magnetická pole, jsou také běžně součástí IMU. Pomocí nich můžeme určit směr zemského magnetického pole, které lze použít jako orientační bod pro orientaci. IMU mohou být také integrovány s dalšími senzory a technologiemi, jako je GPS, aby poskytovaly přesnější a robustnější měření. Například kombinace IMU s GPS může poskytnout vysoce přesné informace o poloze a rychlosti pro autonomní vozidla, drony a další aplikace. Kromě tradičních senzorů nalezených v IMU se pro použití v IMU zkoumají také novější technologie, jako jsou optické senzory a senzory magnetické rezonance. Tyto senzory nabízejí jedinečné výhody, jako je vyšší přesnost a nižší spotřeba energie, díky čemuž by mohly být užitečné v aplikacích, kde tradiční IMU nemusí být praktické. Jedním z hlavních problémů s IMU je, že jsou náchylné k chybám a posunu v čase. To může být způsobeno různými faktory, jako je šum senzoru, změny teploty a nesouosost senzoru. K eliminaci těchto chyb se používají pokročilé algoritmy a kalibrační techniky k filtrování a opravě dat ze senzorů. IMU jsou k dispozici v několika výkonnostních stupních. Jsou rozděleny do jedné ze čtyř kategorií na základě specifikací akcelerometru a gyra: • Automobilová třída • Průmyslová třída • Taktická třída • Stupeň navigace Tyto výkonnostní kategorie jsou obvykle definovány na základě stability předpětí snímače za

běhu, protože stabilita předpětí za běhu hraje tak velkou roli při určování výkonu inerciální navigace. [7][17]

3.1.5 Zhodnocení

Z výše popsaných možností, které jsou použitelné pro mou práci jsem vybral dvě možnosti, které nejlépe vyhovují pro daný účel. Hloubkovou kameru RealSense a senzor Lidar, ty budu následně simulačně testovat a na implementaci na reálného robota použiji jen jeden z nich.

3.2 Autonomita

Mobilní roboti jsou schopni pohybovat se v prostředí a interagovat s ním prostřednictvím senzorů a aktuátorů. Mobilní roboty lze klasifikovat do dvou skupin, jako autonomní mobilní roboty (AMR) a autonomně řízená vozidla (AGV). AMR a AGV jsou dva způsoby autonomie těchto robotů. AGV je založeno na vedení a navigaci v předem definovaném prostředí po předem definované cestě. AGV je v průmyslu často používána, ale v poslední době přebírá jeho roli v průmyslu AMR. AGV jsou vhodná pro opakující se úkoly, jako jsou roboti sledující linii, a běžně se navrhují a vyrábějí pro specifické úkoly. Úkol, který bude AGV provádět, musí být důkladně naplánován a všechny detaily musí pro AGV definovat programátor, protože nemůže rozhodovat a nemá rozhodovací mechanismus založený na umělé inteligenci. AGV pracují podle přednastavených systémů a procesů, které mohou provést rychlou změnu jen obtížně. Hlavní nevýhodou AGV je neschopnost provést dynamicky měnící se úkoly. AMR je schopen navigace v nepředvídatelném prostředí. AMR mohou snímat parametry prostředí a vytvářet model prostředí a lokalizovat se v tomto modelu. Toto chování umožňuje AMR vytvořit navigační plán a optimalizovat tento plán pomocí plánovacího algoritmu. Tudíž při postavení překážky do trajektorie cesty AMR je schopen ji identifikovat a objet optimální cestou, na rozdíl od AGV, které se zastaví a nebude pokračovat, dokud nebude překážka odstraněna. AMR může také vytvořit mapu prostředí pomocí dat senzorů a zároveň se lokalizovat v mapě. Toto je známé jako simultánní lokalizace a mapování (SLAM). SLAM umožňuje vytvořit poměrně citlivý navigační plán, který může programátor dynamicky upravovat a vylepšovat. Základy mobilní robotiky tvoří lokomoce, vnímání a navigace.[23]

3.2.1 Úrovně autonomity

Zvyšující se úroveň autonomie by měla být navržena tak, aby stále více zjednodušovala lidské uživatelské prostředí. Počínaje tím, že člověk musí ovládat téměř všechny aspekty robotického systému (úroveň 0), až po tým robotů provádějících specifické úkoly v dynamických a nestrukturovaných prostředích, kteří se přizpůsobují a učí se nad rámec toho, co naprogramoval designér nebo uživatel (úroveň 5). SAE International je organizace se snahou rozvíjet autonomní řízení. Úrovně autonomie SAE jsou inspirovány pozorností operátora a požadavky na zásah. Každá úroveň je také spojena s odpovědností za provádění kontroly, monitorování prostředí, nouzová opatření a autonomní schopnosti jako udržování v jízdním pruhu. Pro obecnou robotiku můžeme podobně přiřadit odpovědnost za sledování provozu robota. Schopnosti jako je doba mezi zásahy, míru nezávislosti a přizpůsobivosti nebo schopnost být připraven na neočekávané události nebo vlastní údržbu.

Úroveň	Popis	Čas mezi intervencí
0	Plně manuální teleoperace	n/a
1	Robot na viditelný dosah (bez ručního ovládání)	5 minut
2	Operátor na pracovišti nebo v dosahu (bez viditelného dosahu)	1 hodina
3	Jeden operátor hlídá více robotů (bez stálého dohledu)	8 hodin
4	Supervizor bez nutnosti být na pracovišti (bez monitoringu)	3 dny
5	Roboti se adaptují a zlepšují provádění a rozšířený provoz	

Tabulka 3.1: Úrovně autonomity

Úroveň autonomie 1: Člověk musí být vždy v zorném poli robota. Například v zemědělském automatizačním systému musí člověk vždy následovat robota, když prochází polem. Jednoduché reaktivní úkoly, jako je udržení robota ve středu řádku nebo postřik při detekci plevele, jsou automatizovány. Široce používaným příkladem autonomních systémů na této úrovni autonomie jsou traktory naváděné GPS. Zde se vyžaduje, aby člověk byl v kabině, aby se postaral o nepředvídané události, ale traktor jede sám po předem naprogramovaných drahách.

Úroveň autonomie 2: V této úrovni jsou operátoři vzdálenými supervisory. Nemusí robota sledovat, robot může být mimo zorný úhel, ale člověk musí stále zůstat na místě a sledovat robota pro případ zásahu. Tato schopnost je výchozím bodem pro vysoce hodnotné aplikace v mnoha průmyslových odvětvích. Například na úrovni 2 může být zemědělský robot schopen navigovat po předepsané cestě, vyhýbat se většině překážek, a jen jednou za čas narazí. Cílová doba mezi zásahy se prodlužuje na přibližně hodinu. Na této úrovni

autonomie může být člověk schopen dělat jiné úkony, ale pravděpodobně bude mít pod dohledem pouze jednoho nebo dva roboty, kteří běží autonomně.

Úroveň autonomie 3: V mnoha odvětvích představuje autonomie úrovně 3 inflexní bod, kde se rozsáhlá nasazení stávají docela atraktivní. Robotický tým 3. úrovně je dostatečně schopný řešit hraniční případy po dobu několika dní, takže jeden člověk může sledovat několik robotů. Pro použití stejného příkladu na zemědělských strojích, zde se většina zemědělských systémů založených na více robotech začíná rozšiřovat. Na místě, ale může být stále potřeba člověk, aby vyměnil baterie, provedl opravy nebo zachránil uvízlého robota.

Úroveň autonomie 4: Na této úrovni lze autonomní roboty skutečně nasadit ve velkém měřítku, aniž by je omezovaly náklady na pracovní sílu. Autonomní robotické týmy 4. úrovně jsou schopny si samy poradit s mnoha hraničními případy a stanou se dostatečně autonomními, takže člověk nemusí být na místě. Mají také dostatečnou automatizovanou podpůrnou infrastrukturu. Roboti jsou schopni najít své základnové stanice, získat novou baterii, provést drobné opravy a dostat se z obtížných případů (s občasnou pomocí vzdáleného člověka). Tato úroveň autonomie potřebuje nejen kvalitní software robota, ale také místní infrastrukturu k automatizaci a obvykle spolehlivé spojení se vzdálenými uživateli.

Úroveň autonomie 5: Roboti se začínají učit ze svých zkušeností, aby zlepšili provoz nad rámec toho, co naprogramoval konstruktér. Učí se jeden od druhého, na místě a od týmů robotů z jiných míst. Učí se předvídat, jak události ovlivňují jejich schopnosti, a proaktivně plánovat.[5][4]

3.2.2 Předpoklady

Řídicí systém pro netriviální roboty, tj. robotický systém s určitou mírou autonomie a složitosti musí splňovat některé specifikace chování a požadavky na design.

Reaktivita vůči prostředí. Mobilní robot by měl být reaktivní na náhlé změny prostředí a schopné zohledňovat vnější události s časovými limity, které jsou kompatibilní se správným, efektivním a bezpečným prováděním jeho úkolů.

Inteligentní chování. To vyžaduje, aby byly učiněny různé kompromisy založené na pravidlech zdravého rozumu, aby se projevilo inteligentní chování.

Reakce robota na vnější podněty se musí řídit cíli jeho hlavního úkolu.

Integrace více senzorů. Omezená přesnost, spolehlivost a použitelnost jednotlivých senzorů musí být kompenzována integrací několika komplementárních senzorů.

Řešení více cílů. V případě mobilních robotů jsou situace vyžadující protichůdné souběžné akce nevyhnutelné. Kontrolní systém by měl poskytovat prostředky k naplnění těchto mnoha cílů.

Robustnost, robot musí zvládnout zpracovat nedokonalé vstupy, neočekávané události a náhlé poruchy. Spolehlivost, schopnost fungovat bez poruch nebo snížení výkonu po určitou dobu.

Modularita, řídicí systém autonomních vozidel by měl být rozdělen do menších subsystémů nebo modulů, které lze samostatně a postupně navrhovat, implementovat, ladit a udržovat.

Flexibilita, experimentální robotika vyžaduje během implementační fáze neustálé změny v designu. Proto jsou vyžadovány flexibilní řídicí struktury, které umožňují, aby se návrh řídil úspěchem nebo selháním jednotlivých prvků.

Rozšiřitelnost, návrh, stavba a testování jednotlivých komponent robota vyžaduje dlouhou dobu. Proto je žádoucí rozšiřitelná architektura, aby bylo možné systém budovat postupně.

Adaptabilita, vzhledem k tomu, že stav světa se mění velmi rychle a nepředvídatelně, musí být řídicí systém adaptabilní, aby mohl hladce a rychle přepínat mezi různými strategiemi řízení.

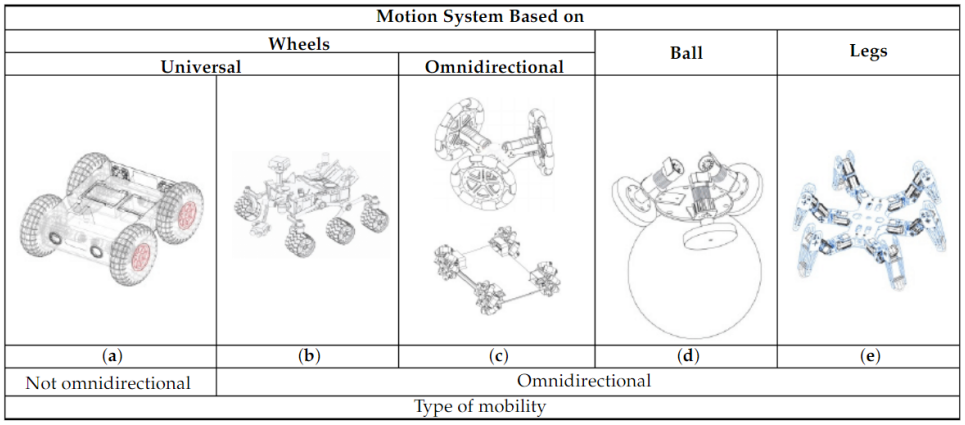
Globální uvažování. Od globálního rozhodovacího činitele na vysoké úrovni, odpovědného za pochopení celkové situace, se vyžaduje, aby rozpoznal chyby způsobené nesprávnou interpretací senzorických dat a spojil dílčí dostupné informace.[27]

3.2.3 Pohybový systém (Locomotion)

Pohybový systém je důležitým aspektem mobilního robota, který se nespoléhá pouze na médium, ve kterém se robot se pohybuje, ale také na jiných faktorech, jako je schopnost manévrování, ovladatelnost, terénní podmínky, výkonnost, stabilita, a podobně. V rámci návrhu robota je zařazen podle stacionarity a mobility: na pozemní, vodní nebo vzdušný. Mobilní roboti zvláště autonomní jsou velmi žádaní kvůli jejich schopnosti vykonávat úkoly, které se mohou zdát obtížné pro lidi. Příklady takto navržených pozemních mobilních robotů jsou kolová, chodící nebo hybridní. Nohy, kolečka, a kloubová těla jsou hlavními způsoby, jak se mobilní robot pohybuje. Koloví roboti jsou jednoduší a vhodní pro měkkou i tvrdou zem, zatímco nohy a kloubové těla vyžadují určitou míru volnosti a tedy větší mechanickou složitost. Výhody kola jsou efektivnost a jednoduchost. Použití koleček je jednodušší a levnější na stavbu, design a programování než ostatní varianty. Ovládání je méně složité a způsobují minimální poškození povrchu, kde se pohybují. Další výhodou je, že nemají problémy se stabilitou a vyvážením kvůli jeho stálému kontaktu se zemí. Jejich nedostatkem je, že nejsou vhodná pro navigaci přes překážky, jako je kamenitý terén a nerovný povrch. Navrhnout a vyvinout pohybový systém, musí být specifické na daný typ terénu. Typy terénu, se kterými se může roboto setkat jsou: nerovný a rovný terén, schodiště nahoru a dolu a nepřekonatelný terén. Další faktor ke zvážení při navrhování mobilního robota je stabilita. Stabilita není obvykle velký problém u robotů na kolech, protože jsou navrženy tak, aby všechna kola byla vždy v kontaktu se zemí. Použití čtyř kol je stabilnější než tři, dva a jedno, protože těžiště bývá umístěno ve středu prostoru kol.[8]

3.2.4 Trakční pohony

Mobilní roboty lze klasifikovat podle pohybového systému nebo podle typu mobility. Pohybový systém může být založen na kolech, drahách, kolech ve tvaru koule nebo nohou. Typy mobility mohou být klasifikovány jako všesměrová (neboli holonomická) nebo nevšesměrová (neholomická). Holonomický mobilní roboti mají tu výhodu, že mohou měnit směr pohybu, aniž by museli provádět mezikroky rotace a jsou schopni se od daného startu pohybovat ve všech směrech bodu při současném otáčení.



Obrázek 3.4: Typy pohybových systémů mobility[18]

V současné době jsou mobilní roboti obvykle založeni na kolech, aby bylo dosaženo mobility. Použití kol je na tvrdém a hladkém povrchu energeticky účinnější než roboti s nohama. Nejoblíbenější koloví mobilní roboti používají diferenciální pohon, to jsou dvě nezávislá pevná hnací kola se dvěma stupni volnosti (DOF) namísto tří DOF (x, y, uhel0). Tyto roboti, jako například domácí vysavače, mají pouze dva pohony, vyžadují méně místa k otáčení kolem libovolného bodu a to také umožňuje tři stupně volnosti, ale jejich omezení je v tom, že nemohou provádět holonomní pohyby, jako jsou pohyby do stran.

K překonání tohoto omezení používají jiní mobilní roboti všesměrový pohybový systém, jako například mobilní roboti vybavení řiditelnými a koordinovanými hnacími koly a všesměrové mobilní roboti založení na kolech, roboti s kulovými koly nebo roboti s nohami. Tato zařízení nabízejí zajímavé funkce při provozu ve stísněných prostorách. Umožňují jak rotaci, tak i pohyb do stran, ale ne současně. Toto omezení lze překonat použitím holonomního všesměrového pohybového systému, který se může kdykoli pohybovat všemi směry bez změny směru kola, protože může dosáhnout 3-DOF pohybu ve 2-rozměrné rovině. Hlavním omezením jejich pohybu je prokluz kola.

Holonomičtí roboti jsou založeni na použití tří nebo čtyř všesměrových kol, která se skládají z několika pasivních válečků nebo kuliček, jejichž osy jsou tečné k obvodu kola a volně se otáčejí. Tříkoloví všesměroví mobilní roboti mohou mít tři nezávislé pohony a mohou dosáhnout dvou nezávislých translačních a jednoho rotačního stupně volnosti, celkem tedy 3 stupňů volnosti na rovném povrchu, manévrování a navigace ve stísněných prostorách. Kvůli jejich vysokému těžišti však mají problémy se stabilitou, když se pohybují po rampě kvůli trojúhelníkové kontaktní ploše se zemí. Tento problém se stabilitou lze překonat použitím čtyř kol s 4-DOF.

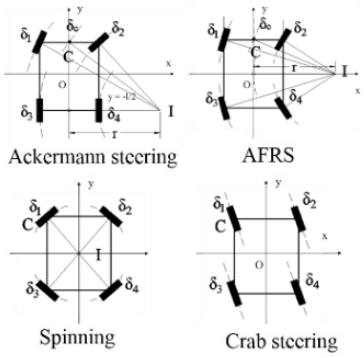
Robot s koly ve tvaru koule může běžet v libovolném směru, ale ne po nerovném terénu nebo schodech. Všesměroví mobilní roboti založení na kolech mohou mít univerzální nebo všesměrová kola. Robot s nohama se může pohybovat jakýmkoli směrem a může se pohybovat po jakémkoli typu povrchu, avšak mechanismus robotů s nohama je velmi složitý a má omezení rychlosti.

Obecně jsou principy fungování všesměrových mobilních robotů založeny na kinematických modelech. Je k dispozici mnoho konstrukcí holonomních mobilních robotů. Existuje několik typů všesměrových kol, ale u všech je princip funkce založen na zajištění trakce ve směru kolmém k ose motoru a použití vnitřních pasivních válečků, které se mohou posouvat ve směru osy motoru. Tato vnitřní pasivní kola, kuličky nebo válečky jsou umístěny po obvodu nebo hlavních kolech. Konstrukce kola se tedy skládá z více pasivních válečků (nebo vnitřních pasivních kol), jejichž osy jsou umístěny tečně k obvodu hlavního kola. Tato konstrukce se nemůže vyhnout nespojitým stopám a způsobuje nepravidelný kontakt s povrchy kvůli mezerám mezi po sobě jdoucími válečky nebo koly, které způsobují vibrace v robotu.

Je to tedy nevýhodou těchto kol, že generují horizontální vibrace kvůli parazitním točivým momentům, které jsou generovány skutečností, že se kontaktní bod pohybuje podél linie rovnoběžné s hřídelí kola. Koncepce dvojitého kola, je řešením založeným na dvou překrývajících se paralelních kolech. Kontakt mezi montážním kolem a zemí je nepřetržitý. Tato konstrukce generuje horizontální vibrace způsobené mezerami mezi rotujícími vnitřními koly. [18][1]

3.2.5 Systémy řízení

Je několik režimů řízení a mezi ty nejčastěji používané a základní režimy řízení patří, diferenciální řízení, Ackermann řízení, aktivní přední a zadní řízení (AFRS), krabí řízení, a otočné řízení (jak je znázorněno na obrázku 3.5). [44]



Obrázek 3.5: Systémy řízení[44]

Diferenciální řízení (taky skid steering)

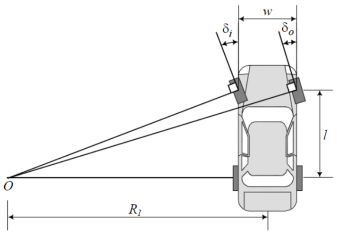
Diferenciální řízení je na principu rozdílu mezi rychlostmi kol na levé straně a rychlostmi kol na pravé straně. Rozdíl v rychlosti způsobí, že se vozidlo otočí. Tato metoda se používá pro vozidla, které nemají schopnost vertikálně natáčet kola. Je velký rozdíl použít tuto metodu na vozidle s pohonem všech kol ve srovnání s vozidlem 2WD. Diferenciální řízení všech kol se často používá u vozidel, která by měla být velmi dobře ovladatelná v těžkém terénu. Vozidla využívající tuto metodu řízení se může otočit na místě. Převodovka je také konstrukčně jednoduchá, protože kola na jedné straně se točí stejnou rychlostí a poloha a orientace kol jsou pevné. Metoda má také některé nevýhody. V zatáčce se vozidlo také otáčí kolem svislé osy středem vozidla, což znamená, že kola musí klouzat do stran. To spotřebovává velké množství energie k otáčení vozidla. Metoda je také nevhodná pro počítání ujeté vzdálenosti z důvodu nadměrného prokluzu, který činí údaje z počítadla kilometrů nepoužitelnými.[25]

Ackerman

Pro vnitřní a vnější kolo vozidla v zatáčkách v ustáleném stavu existuje kinematický vztah zvaný Ackermanova podmínka a je definována rovnicí

$$\cot \delta_0 - \cot \delta_i = \frac{w}{l} \tag{3.1}$$

, kde δ_0 je úhel natočení vnějšího kola, δ_i je úhel natočení vnitřního kola, w je rozchod a l je rozvor kol vozidla. Tato podmínka znamená, že vnitřní kolo má větší úhel natočení než vnější kolo, protože je blíže středu rotace. Získá se tím lepší manévrovací schopnosti.[29][40]



Obrázek 3.6: Řízení typu Ackerman[29]

Aktivní přední a zadní řízení (AFRS)

Aktivní ovládání předních a zadních kol umožňuje natáčet přední i zadní kola. Je to strategie, která spočívá v tom, že zadní kola jsou řízena buď mimo fázi vzhledem k předním kolům při nízké rychlosti a velký úhel řízení nebo řízené ve fázi při vysoké rychlosti. S touto kontrolní strategií, lze vyvinout zlepšení manévrovatelnosti. Fázové řízení zadních kol zvyšuje rychlost zatáčení a stabilitu vozidla. [12]

Bod otáčení

Bod otáčení je metoda řízení, kde se pro zatočení natočí všechny 4 kola kolmo ke středu vozidla. Tím se docílí, že se vozidlo otočí okolo své osy. Je to tedy na podobném principu, jako diferenciální řízení. Výhodou oproti diferenciálnímu řízení je, že se eliminuje prokluz kol. Nevýhodou je, že pro otočení se musí vozidlo zastavit.[25]

Crabbing

Crabbing je založen na principu pohybu kraba v přírodě, ten se umí pohybovat ve všech směrech bez rotace těla. Tento princip se dá použít i na kolové vozidla a vozidlo se pohybuje všemi směry do stran bez rotace jeho samotného. To se dá využit například k paralelnímu parkování ve stísněných prostorách.[42]

3.2.6 Systémy kol

Konvenční kolo

Jedním z příkladů neholonomního systému je kolový robot s diferenciálním pohonem s konvenčními koly. Konvenční kolo je široce používáno ve všech oblastech strojírenství díky své jednoduchosti a funkčnosti, která je omezena na zajištění otáčení kola dopředu a dozadu. U robota s diferenciálním pohonem umožňují konvenční kola rotaci robota, když jsou pro jeho konvenční kola aplikovány různé rychlosti otáčení a/nebo směr otáčení.[34]

Kolo s dvěma osamy rotace (steering wheel)

Na první pohled může kolo vypadat stejně jako konvenční kolo, ale má jinou mechanickou strukturu. Pod pojmem řízení se rozumí nejen kolo, ale určitý mechanismus řízení, který umožňuje běžnému kolu otáčet se kolem své svislé osy. Pro tento účel mechanismus používá motor řízení k ovládání směru pohybu (tj. rotaci kolem svislé osy) kola a hnací motor k zajištění pohybu vpřed a vzad. [34]

Univerzální Omni Wheel

Konfiguraci lze optimálně sestavit pomocí tří nebo čtyř kol. Tři univerzální kola mohou být namontována na trojúhelníkovou plošinu s jejich osami nakloněnými o 120 stupňů vůči sobě. Pro čtyřkolové provedení jsou typické dvě konfigurace:

1. Kola jsou umístěna symetricky po stranách čtvercové mobilní plošiny s úhlem 90 stupňů mezi koly.

2. Kola jsou umístěna symetricky v rozích čtverce základna mobilní plošiny a jejich osy jsou vůči sobě nakloněny o 90 stupňů.[34]

3.2.7 Percepce (vnímání)

Vnímání je velmi důležité, aby autonomní mobilní robot získal informace ze svého okolí, vnímal předměty kolem sebe, nebo jeho relativní polohu. Vnímání je imperativním aspektem mobilních robotů. Aby toho bylo dosaženo, informace jsou vnímány použitím senzorů a dalších souvisejících zařízení. Senzory umožňují autonomně provádět lokalizaci robotů. Používají se také pro sběr dat, identifikaci objektů, mapování a reprezentace. Senzory používané v oblasti dat jsou rozdělena do dvou hlavních hledisek: Proprioceptivní a exteroceptivní senzory a aktivní a pasivní senzory. Proprioceptivní senzory měří hodnoty uvnitř systému robota, např. stav baterie, poloha kola, úhel kloubu, rychlost atd. Těmito snímači mohou být enkodéry, potenciometry, gyroskopy, kompasy atd. Exteroceptivní senzory jsou zvyklé extrahovat informace z prostředí nebo objektů. Sonarové senzory, infračervené (IR) senzory, ultrazvukové vzdálenostní senzory jsou některé příklady exteroceptivních senzorů. Aktivní senzory vydávají svou vlastní energii do okolí a poté měří reakci prostředí. Obvykle dosahují skvělého výkonu díky jejich schopnosti zvládat interakce s prostředím. Příklady aktivních senzorů jsou sonarové senzory, radary atd. Zatímco pasivní senzory dostávají energii k pozorování jako je fotoaparát, teplotní čidla, dotyková čidla atd. [8]

3.2.8 Lokalizace

Lokalizace je dalším základním problémem, se kterým se setkáváme u mobilních robotů, která také vyžaduje pozornost. Náročnou částí lokalizace je odhad polohy a orientace robota, o kterém lze tyto informace získat ze senzorů a dalších systémů. K vyřešení problému lokalizace by tedy měla být navržena dobrá technika, která se vypořádá s chybami, faktory snižujícími kvalitu, nesprávným měřením a odhady. Techniky jsou rozděleny do dvou kategorií na relativní a absolutní lokalizaci.

1) RELATIVNÍ LOKALIZAČNÍ TECHNIKY
Tato metoda odhaduje polohu a orientaci mobilního robota integrací informací produkovaných různými senzory prostřednictvím kombinace informací prezentovaných různými senzory, obvykle kodérem nebo inerciálními senzory. Integrace začíná od výchozí pozice a průběžně se aktualizuje v čase. Samotné relativní polohování lze použít pouze po krátkou dobu.

2) ABSOLUTNÍ LOKALIZAČNÍ TECHNIKY
Tato metoda umožňuje mobilnímu robotu hledat svou polohu přímo z prostředí

mobilního systému. Jejich četné metody obvykle závisejí na navigačních majících, aktivních nebo pasivních orientačních bodech, mapových shodách nebo satelitních signálech, jako je Global Positioning System (GPS). Pro absolutní lokalizaci, je růst chyb zmírněn, když jsou dostupná měření. Poloha robota je určena externě a jeho přesnost je obvykle nezávislá na čase a místě. Jinými slovy, není integrace šumu dat a proto nedochází k žádné agregaci chyb s časem nebo ujetou vzdáleností. Omezení spočívá v tom, že nelze sledovat robota na krátké vzdálenosti.[8][33][37]

Lokální a globální

Autonomní navigace mobilního robota je problémem v oblasti robotiky. Existují dva způsoby, jak lze problém s navigací kategorizovat na lokální a globální navigaci. Lokální a globální navigace se liší co do vzdálenosti, měřítek, vyhýbání se překážkám a schopnosti dodržet cílový stav. Pro lokální navigaci se pro určení směru navigace používá mřížka obsazenosti mapy a pro globální navigaci se používá orientační přístup založený na topologické mapě. Ty mají kompaktní reprezentaci prostředí a nezávisí na geometrické přesnosti. Omezení tohoto přístupu spočívá v tom, že jsou sníženy šumem generovaným snímači. Navigační systémy mobilních robotů závisí na úrovni reprezentace prostředí. Pro přesné určení polohy a orientace mobilního robota je nezbytné, aby prostředí bylo modelováno v jednoduché a srozumitelné struktuře. Tři hlavní techniky pro reprezentaci prostředí jsou: geometrické, topologické a sémantické.

1) GEOMETRICKÉ
Používá se k popisu prostředí robota pomocí parametrizace primitivních geometrických objektů, jako jsou křivky, čáry a body. Geometrické znázornění prostředí je bližší světu senzorů a aktuátorů a je nejlepší pro provádění lokální navigace. Pro ukázku zde máme dvě metody. Metoda Principal Components Analysis (PCA) – Bayesovské metody se zobrazením mřížkové mapy ke kompresi obrázků a snížení výpočetních zdrojů. PCA byl také použit ke snížení rozměrů a modelování parametru prostředí tím, že se pixely obrazu považovaly za soubor dat vektorů. Markovova lokalizační metoda, která poskytuje přesnost a multimodalitu reprezentující rozdělení pravděpodobnosti různého druhu, ale vyžaduje značné zpracování pro aktualizaci, a proto je nepraktická pro velké prostředí.

2) TOPOLOGICKÉ
Je definována referencí prvků prostředí a podle zřetelných vztahů mezi nimi. Konvenční metodou pro modelování prostředí robota je diskretizace modelu prostředí pomocí topologické reprezentace stavu věrohodnosti, kde každá

pravděpodobná pozice mobilního robota je připojena k uzlu v topologické mapě. Navrhovaný přístup využívá vizuální prvky extrahované z dvojice stereo obrazů jako orientační body. Zatímco nové orientační body jsou sloučeny do mapy a přechodné orientační body jsou z mapy časem odstraněny. Topologická reprezentace prostředí využívá k modelování prostředí grafy a používá se ve velkých navigačních úlohách.

3) SÉMANTICKÁ

Současný vývoj v robotice spočívá v odklonu od reprezentačních modelů, které jsou nejblíže hardwaru robota, jako jsou geometrické modely, k modelům bližším lidskému uvažování, se kterými bude robot interagovat. Navrhuje se spojit model s tím, jak roboti reprezentují prostředí a jak to dělají lidé. Roboty, které jsou vybaveny sémantickými modely prostředí, kde působí, mají větší rozhodovací autonomii a stávají se robustnějšími a efektivnějšími. Sémantické informace představují lepší řešení pro interakci s lidmi, jsou nejabstraktnějším modelem reprezentace a přidává do mapové reprezentace pojmy, jako jsou pomůcky nebo významy prvků prostředí. Sémantická navigace je považována za navigační systém, který považuje sémantické informace za model, který zahrnuje konceptuální a fyzickou reprezentaci objektů, míst, užitečnost objektů a sémantické vztahy mezi objekty a místy. Tento model umožňuje robotovi spravovat prostředí a pokládat dotazy na prostředí, aby bylo možné provádět plány pro navigační úkoly. Environmentální model vyžaduje vylepšenou reprezentaci, aby umožnil úspěšný výsledek, lepší přesnost a také snížil výpočetní náklady. Aby to převládlo, musí být prostředí dobře reprezentováno, musí být přijata jednoduchá technika a musí být začleněna do reprezentace prostředí robota. Bezpečná a efektivní navigace mobilním robotem vyžaduje efektivní techniku plánování cesty, protože kvalita generované cesty extrémně ovlivňuje robotické aplikace. V prostředí s několika překážkami se hledání cesty bez kolize s překážkami z počátečního bodu do konečného bodu stává problémem, protože krátkost a jednoduchost trasy jsou důležitými kritérii ovlivňujícími optimalitu vybraných tras. Vzhledem k délce ujeté cesty robotem, spotřebou energie a dobou jeho výkonu je nejvhodnější algoritmus ten, který najde nejkratší možnou cestu. V zásadě existují dva typy prostředí: statické a dynamické. Zatímco dynamické prostředí se dělí na globální a lokální plánování cest. Globální navigační strategie se zabývá zcela známým prostředím, zatímco lokální navigační strategie se zabývá neznámým a částečně známým prostředím. [8][33][37]

Kalmanova filtrace

Základem je Kalmanův filtr a metoda částicového filtru na přístupu Bayesova filtru. Každý vzorkovací čas metody předpovídá stav odhadu pomocí interních informací, jako je rychlost otáčení měřená IMU. Kalmanův filtr má určitá

omezení na model systému a nejistoty. Obvykle to může nastat, když se stav vyvíjí podle lineárního přechodu rovnic a měří lineárně souvisící se stavy. Kromě toho nejistota v přechodném stavu a měření má být Gaussovská. Naopak, částicový filtr se může vypořádat s libovolným přechodem stavu a měřením. Také nejistota v přechodovém stavu a měření nemusí být Gaussovská. Kalmanův filtr poskytuje jeden výsledek odhadu a jeho nejistota je popsána chybovou kovariancí. Částicový filtr poskytuje částice rozdělení jako výsledek pro odhad a nejistotu rozdělení.[20]

Rozšířený Kalmanův filtr (EKF)

Rozšířený Kalmanův filtr (EKF) je jedním z nejpopulárnějších, ovladatelných, pravděpodobnostních algoritmů, který rozšířil své aplikace do lokalizace i mapování. Vzhledem k jeho nízkým výpočetním nákladům a snadné implementaci. Algoritmus EKF používá předpověď a aktualizaci k výpočtu kombinované pravděpodobnosti odhadu stavu pro robota a vytváří mapu informace. Algoritmus EKF je založen na prvním řádu Taylorovy řady pro rozšíření modelu nelineárního systému a pozorovacího modelu. Model je linearizován a zpracován klasickým kalmanovým filtrem. Existují zjevné vady. Za prvé, EKF nemůže odhadnout Gaussův náhodný vektor nelineárního přenosu střední hodnoty a rozptylu přesně, proto tedy není přesnost filtru vysoká. Za druhé, je potřeba EKF, aby vyřešil jakobiánskou matici stavu a rovnice pozorovatele. Některé systémy nemají analytické řešení jakobiánskou maticí a to omezuje jeho aplikaci. Za třetí, kdy je systém silně nelineární nebo počáteční chyba odhadu je velká, je obtížné dosáhnout požadovaných výsledků. Když se používá k odhadu pozice robota, má tendenci rychle konvergovat k nekonzistentním hodnotám a není schopen zvládnout nelineární systémy, tam kde je stupeň nelinearity příliš vysoký. Hlavní nevýhodou této techniky je to, že potřebuje další výpočetní čas, aby bylo možné iterativně určit nejoptimálnější způsob kombinace střední hodnoty a kovariancí za účelem dosažení konzistentních odhadovaných výsledků. V minulosti běžný přístup k překonání nekonzistence zahrnovaly heuristické ladění založené na přidání umělého šumu do systému, který kompenzuje účinky vzájemných závislostí a linearizace.[26][24]

Monte Carlo lokalizace

Navigace mobilních robotů má základní předpoklad efektivní lokalizace v okolním prostředí. Problém lokalizace je definován jako nalezení pozice robota, kterému byla poskytnuta mapa prostředí a dat ze senzorů. Pozice robota

zahrnuje jeho polohu a orientaci vzhledem k lokálnímu nebo globálnímu koordinačnímu systému. V robotickém navigačním systému robot používá vizuální senzor a odometrii a poté je zkombinuje s Monte Carlo Algoritmem lokalizace (MCL), který může dosáhnout vyšší přesnosti lokalizace ve 2D mapě. Proto tato uvedená metoda může nahradit vizuální lokalizaci, může se vyhnout procesu extrakce a spárování obrazových prvků a snižuje časovou náročnost. Mezitím multi fúze informací senzoru mezi vizuálním senzorem a odometrií může opravit kumulativní chyby systému. MCL ve 2D mapě nám může umožnit, aby robot pózoval v různém čase. Podle výpočtu vztahů mezi těmito pozicemi můžeme získat matici přechodu pozice. Tyto přechodové matice slouží pro registraci 3D mračna bodů (point cloudu). Potom získáme vnitřní 3D mapu.

Předběžným požadavkem na algoritmus MCL je známá mapa prostředí, ve kterém se robot nachází. Předdefinovaná sada částic se používá k popisu možné polohy robota. Vizuální senzory se běžně používají v MCL aplikaci, protože prostřednictvím měření vzdálenosti mezi stěnou a překážkou, je možné určit pozici robota. Můžeme říci, že MCL je členem skupiny statistických a pravděpodobnostních algoritmů. Vzhledem k nízkému výpočetnímu úsilí potřebné k provádění lokalizačních úloh, se používá v mnoha aplikacích, kde je potřeba nízká doba odhadu a je požadována vysoká rychlost aktualizace. Nicméně MCL je vysoce ovlivněna počtem bodů měření, které senzor může poskytnout. Čím větší prostředí, tím více bodů měření je potřeba dosáhnout pro dostatečně přesnou lokalizaci.

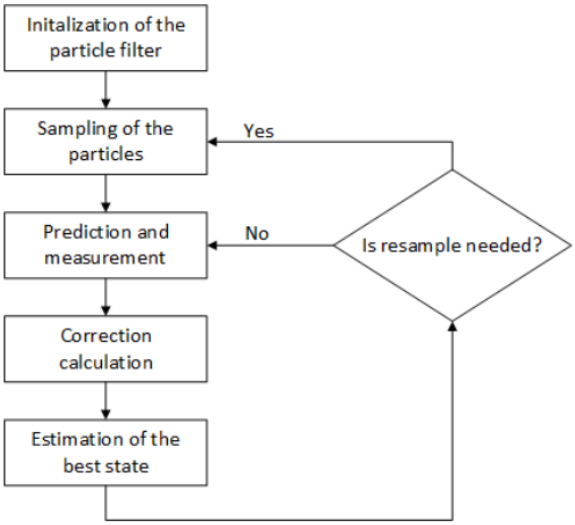
Lokalizace Monte Carlo lokalizuje mobilní roboty podle spojování pravděpodobnostních a mobilních informací o pohybu robota. Rozdělení pravděpodobnosti robota po celém pozičním prostoru je odvozeno od Bayesova teorému. Lokalizace Monte Carlo využívá částicový filtr reprezentující posteriorní přesvědčení. Sada vážených vzorků, které se také nazývají částice, se používají k vytvoření distribuce a toto rozložení částic ve stavovém prostoru je charakterizované rozdělením pravděpodobnosti polohových stavů. Čím hustší částice tím vyšší pravděpodobnost a čím řidší částice tím nižší pravděpodobnost. Dokud číslo částic je dostatečně velká, distribuce částic může charakterizovat jakékoli rozdělení pravděpodobnosti, umožňující globální lokalizaci, která má být provedena. Nevýhoda metody MCL je požadavek určitého počtu částic k dosažení a spolehlivé výsledky lokalizace. Navíc počet požadovaných částic roste spolu s velikostí prostředí.

Existují také modifikace tohoto algoritmu, jako je Adaptivní lokalizace Monte Carlo (AMCL), která je modifikací metody MCL využívající Kullback-Leibler Distance (KLD) vzorkování pro zkrácení doby provádění lokalizačního algoritmu. Technické implementace metody AMCL často využívají k získání požadovaných mapových dat vizuální ze senzorů a odometrie založené na

datech pocházející z kodérů kol pro výpočet pohybu robota. Takové implementace však omezují potenciální aplikace AMCL v rozsahu kolových robotů.[36][46][32]

Částicový filtr

Částicový filtr je variantou bayesovského filtru, který používá diskrétní částice k rekurzivnímu odhadu posterioru rozložením stavů, jak je znázorněno na obrázku 3.7. Můžeme rozlišit dva stupně částicového filtru na predikci a korekci. Během fáze predikce filtr využívá model systému a předchozí stav pro určení aktuálního stavu pro každý z částic. V poslední fázi měření ze senzorů se používají ke korekci stavových odhadů získaných během předpovědi.[32]



Obrázek 3.7: Částicový filtr[32]

Simultánní lokalizace a mapování

SLAM je zkratka pro simultánní lokalizaci a mapování, což znamená algoritmus, který odhaduje svou vlastní polohu pomocí senzoru prostředí a provádí mapování současně. SLAM je považován za důležitou techniku pro autolokalizaci robotů, zejména v oblastech kde postrádáme globální pozici informace, jako jsou tunely a vnitřní scény. Skládá se ze součtu různých algoritmů. Mezi různými algoritmy se nejzákladnější algoritmus SLAM nazývá odometrie. Dále máme algoritmus založený na vizuálních datech jménem V-SLAM. Hlavní problémy spojené s viděním založeným na V-SLAM jsou extrakce a párování řady

vizuálních struktur z obrazových sekvencí s časovými vztahy a jak tyto rysy využít k odhadu pozice kamery. K řešení těchto problémů systémy V-SLAM obecně obsahují sadu společných bloků, včetně sledování funkcí, vytváření map a detekce uzavření smyčky pro korekci posunu chyb. Podrobnosti o implementaci těchto modulů se liší podle mnoha faktorů, včetně použité typy vizuálních senzorů, využití funkcí, a optimalizačních metod.
Pro V-SLAM se používá mnoho typů kamer, včetně monokulárních kamer, stereo kamer a RGB-depth (RGB-D) kamer. SLAM pomocí pouze monokulárního fotoaparátu není schopen odhadnout globální měřítko přímo a musí se spoléhat na další senzory, např jako inerciální měřicí jednotky (IMU) nebo doplňkové priory, jako jsou tvarové priory, aby se překonala nejednoznačnost měřítka.
Pro srovnání, RGB-D SLAM může získat informace o hloubce a odhadnout globální měřítko přímo, ale je extrémně citlivý na světlo, což omezuje jeho použití ve většině venkovních scén. Stereo kamera SLAM dokáže odhadnout hloubku a globální měřítko přímo na základě délky základní linie mezi levou a pravou kamerou. Přesnost odhadu hloubky v modelech stereo kamer silně závisí na délce základní linie, která limituje jejich aplikace v přenosných mobilních zařízeních.
Na rozdíl od tří konvenčních typů fotoaparátů uvedených výše, kamery založené na událostech jsou biologicky inspirované. Události jsou časově označené změny jasu nezávislých pixelů. Kamery založené na událostech mohou přímo snímat události asynchronně, což vede k nižší latenci a vyššímu dynamickému rozsahu než u běžných fotoaparátů. Proto, kamery založené na událostech lze použít k řešení obtížných úkolů jako je rychlé a dynamické vyhýbání se překážkám. Podle množství použitých informací o vlastnostech pro shodu lze vizuální prvky rozdělit na dvě úrovně. Nízkoúrovňové funkce, jako jsou záplaty pixelů, body nebo čáry a funkce na vysoké úrovni, jako je sémantické označení objektů. Různé funkce popisují scény z různých perspektiv. Nízkoúrovňové prvky se zaměřují na místní detaily, jako jsou textury nebo geometrická primitiva objektů a scény. Funkce na vysoké úrovni integrují detaily do sémantických štítků, které se více shodují s člověkem a jeho pochopením světa. Z hlediska optimalizace může být SLAM rozdělen do dvou tříd. SLAM založený na filtrech a SLAM založený na rámcích. První z nich třídí minulé pózy a shrnuje informace získané v průběhu času a rozděluje pravděpodobnosti. Naproti tomu druhý vybírá pouze malý počet minulých snímků a použije svazek nastavení (metoda BA) k těmto rámům. I když se mnohokrát prokázalo ohledně rámcové metody SLAM, že metoda BA je pro V-SLAM efektivnější, u dynamických SLAMů se stále vyplatí filtrační metody na základě jejich přirozených předností z hlediska manipulace statistické informace, které jsou důležité pro odhad hloubky, kombinace senzorů, určování dynamických vlastností a robustní správu map. Problém V-SLAM lze elegantně řešit ve statických nebo přibližně statických texturovaných scénách. V takových případech je dostatek funkcí na pozadí pro odhad. Nicméně ve složitějších reálných prostředích jako jsou přeplněné chodby v obchodních centrech přináší klasický SLAM špatné výsledky, protože neumí správně zacházet

s dynamickými funkcemi. Existují dva způsoby řešení tohoto problému. První je vyřazení dynamických prvků, což je známé jako robustní SLAM problém. Druhou metodou je integrace SLAM a sledování více objektů (MOT), což je známé jako problém SLAMMOT.[43][30][15]

GMapping (Grid Mapping)

Po zpracování mapovacím algoritmem je vytvořena 2D mapa. Algoritmus GMapping je laserový algoritmus SLAM pro mapování mřížky. GMapping je pravděpodobně nejpoužívanější algoritmus SLAM. Hlavní myšlenkou je použití Rao–Blackwellizovaných částicových filtrů (RBPF) k predikci funkce stavového přechodu. GMapping algoritmus je také známý jako algoritmus RBPF SLAM, pojmenovaný po použití Rao–Blackwellized částicových filtrů. To je pak nazváno GMapping (G pro mřížku) kvůli použití map mřížky. Z dat získaných ze senzoru nebo odometrických dat se vytvoří částice s pravděpodobností, která představuje možnou polohu robota.[45][14]

3.2.9 Odometrie

Přesná lokalizace vozidla je základní výzvou v mobilních robotických aplikacích. Aby robot dosáhl autonomní navigace, musí si udržovat znalosti o své poloze v průběhu času. Systém odometrie poskytuje lokálně přesný odhad polohy a rychlosti robota na základě jeho pohybu. Informace o odometrii lze získat z různých zdrojů, jako jsou IMU, LIDAR, RADAR, GPS a enkodéry kol. Nicméně, každá technika má své slabiny. Přestože odometrie kol je nejjednodušší dostupnou technikou pro odhad polohy, trpí odchylkou polohy v důsledku prokluzování kol. IMU se v průběhu času posouvá kvůli šumu. Ačkoli je GPS nejběžnějším řešením lokalizace, protože dokáže poskytnout absolutní polohu bez hromadění chyb, je efektivní pouze v místech s volným výhledem na oblohu. Navíc jej nelze používat v uzavřených prostorách a ve stísněných prostorách. Komerční GPS odhaduje polohu s chybami v řádu metrů. Tato chyba je považována za příliš velkou pro přesné aplikace, které vyžadují přesnost v centimetrech, jako je autonomní parkování. Diferenciální GPS a kinematická GPS v reálném čase mohou poskytnout polohu s centimetrovou přesností, ale tyto techniky jsou drahé. Odometrie je spojená s transformací a využívá systém odometrie robota k publikování informací o lokalizaci. Informace jsou nepřetržité, ale v průběhu času nebo vzdálenosti se stávají méně přesné v závislosti na modalitách snímače a posunu. Navzdory tomu může robot tyto informace stále používat k navigaci ve svém bezprostředním okolí např. vyhýbání se překážkám. Pro získání konzistentně

přesných informací o odometrii v průběhu času poskytuje mapa s globálně přesnou informací, které se používají ke korekci odometrie. Protože každý senzor má své nevýhody, používá se pro zpřesnění odhadu jejich kombinace, aby se vzájemně eliminovali negativní vlastnosti.
Vizuální odometrie (VO) je technika používaná k lokalizaci robota pomocí pouhé série snímků získaných z jednoho nebo více kamer připojených k robotu. Snímky obsahují dostatečné množství smysluplných informací (barva, textura, tvar atd.) pro odhad pohybu kamery.[6][10]

Vizuální odometrie

Vizuální lokalizace je hlavním úkolem pro autonomní vozidla, aby byla schopna sledovat své cesty a správně detekovat a vyhýbat se překážkám. Odometrie založená na vidění je jednou z robustních technologií používané pro lokalizaci vozidel, proto vizuální odometrie a simultánní lokalizace a mapování (SLAM) hraje důležitou roli v oblasti vizuální navigace. Většina běžně používaných vizuálních senzorů zahrnuje monokulární kamery, stereo kamery a RGB-D kamery, kde RGB-D kamery jsou omezeny ve vnitřní aplikace, kvůli jejich základní technologii strukturovaného světla.
Vizuální odomerie je proces odhadu pozice agenta (např. vozidla, člověka a robota), který zahrnuje použití pouhé série snímků získaných z jedné nebo více kamer. Tento přístup je bezkontaktní metodou pro efektivní polohování mobilních robotů.
Vizuální odometrie je levná a alternativní odometrická technika, která je přesnější než konvenční techniky, jako je GPS a odometrie kol. Tato metoda se vyznačuje dobrou rovnováhou mezi cenou, spolehlivostí a složitost implementace. Použití spotřebitelského fotoaparátu místo drahých senzorů nebo systémů, jako je GPS, laserová lokalizace systémů (LIDAR), je přímou a levnou metodou odhadu polohy. Obrázky ukládají velké množství smysluplných informací, které jsou dostatečné pro odhad změny polohy. U vizuální odometrie není ovlivněn prokluz kol v nerovném terénu nebo jiných nepříznivých podmínkách. Dále funguje vizuální odometrie efektivně v prostředích s nedostupným GPS. Lokálního drift pro vizuální odometrii je menší než rychlost driftu kolových kodérů.Vizuální odometrii lze integrovat s GPS a IMU pro maximální přesnost. Ve srovnání s použitím jiných senzorů, použití kamer pro lokalizaci robotů má výhody snížení nákladů, umožňující jednoduchou integraci dat o pohybu do jiných algoritmů založených na vidění, jako je detekce překážek, chodců a jízdních pruhů a bez nutnosti kalibrace mezi senzory. Fotoaparáty jsou malé, levné, lehké, s nízkou spotřebou propracované a všestranné. Lze je tedy také použít v jakémkoli dopravním prostředku (pozemní, podvodní, vzdušné) a pro další robotické úkoly (např. detekce a rozpoznávání objektů).
Přestože lokalizace vnitřního robota je snadno implementována, lokalizace

robota ve venkovním prostředí zůstává náročným problémem. Mnoho faktorů (např. terény obvykle nejsou rovné, přímé sluneční světlo, stíny a dynamické změny v prostředí způsobené větrem a slunečním zářením) ztěžují lokalizaci ve venkovním prostředí. Hlavní výzvy v systémech vizuální odometrie se týkají především výpočetních nákladů, světelných a zobrazovacích podmínek. Aby vizuální odometrie fungovala efektivně, dostatečné osvětlení a statická scéna s dostatkem struktur v prostředí by měla být přítomna, aby bylo možné extrahovat zdánlivý pohyb. Stíny ze statických nebo dynamických objektů, nebo že samotné vozidlo může narušit výpočet přemístění pixelů a tím způsobit chyby odhadu nové pozice.
Vizuální odometrie má širokou škálu aplikací a byla účinně aplikována v několika oblastech. Mezi její aplikační domény patří robotika a automobilový průmysl. Vizuální odometrie se uplatňuje v mnoha typech mobilních robotických systémů, jako jsou pozemní, podvodní, vzdušní a vesmírní roboti. Při průzkumu vesmíru se například vizuální odometrie používá k odhadu pohybu NASA Mars roveru. Používá se také v bezpilotních vzdušných dopravních prostředcích (UAV). Autonomní vzlet, přistání a navigace z bodu do bodu. Navíc vizuální odometrie hraje významnou roli v autonomních podvodních vozidlech a systémech kontroly korálových útesů. Vzhledem k tomu, že signál GPS degraduje nebo se stane nedostupným v podvodním prostředí se podvodní vozidla nemohou spoléhat na GPS pro odhad pozice; proto je vizuální odometrie považována za nákladově efektivní řešení pro podvodní lokalizační systémy. V automobilovém průmyslu hraje vizuální odometrie také velkou roli. Používá se v mnoha ovladačích asistenčních systémů, jako jsou asistované brzdové systémy založené na vidění. Vizuální odometrie se považuje za nákladově efektivní řešení ve srovnání se systémy LIDAR. V robotice pozemních vozidel je efektivní využití vizuálních senzorů pro navigaci a detekci překážek hlavním cílem. Vizuální odometrie se využívá v případech, kdy signál GPS je nedostupný, je příliš těžký na přenášení (na malém letecké vozidlo), nebo nedostatečně přesné za nízkou cenu (v zemědělských aplikacích). Používá se v zemědělských polních robotech k odhadu pozice robota vzhledem k plodinám. [10][15]

3.2.10 Plánování a navigace

Problematika plánování cesty nebo hledání cesty je dobře známá v robotice a hraje důležitou roli pro navigaci autonomními mobilními roboty. Navigace, což je proces nebo činnost k plánování a nasměrování trasy nebo cesty je úkol, který autonomní robot musí provést správně, aby se mohl bezpečně přesunout z jednoho místa na druhé, aniž by se ztratil nebo nesrazil s jiným objektem. Tři obecné problémy navigace jsou lokalizace, plánování cesty a řízení pohybu. Mezi těmito třemi problémy, lze tvrdit, že plánování trasy je jeden z nejdůležitějších v procesu navigace. Plánování cesty umožňuje výběr

a identifikaci vhodné cesty, kterou má robot procházet v oblasti pracovního prostoru. Výzkum plánování cesty autonomního mobilního robota přitahoval pozornost od 70. let 20. století. Za posledních několik let a v poslední době se výzkum v této oblasti zvýšil kvůli použití mobilních robotů v různých aplikacích. Tito roboti tedy musí fungovat s nejistotami v různých oblastech. Přesné plánování trasy umožňuje autonomním mobilním robotům následovat nebo sledovat optimální cestu z výchozí pozice do cílové pozice bez kolize do překážky v oblasti pracovního prostoru.

Plánovací algoritmy cest můžeme klasifikačně rozdělit na dva druhy rozlišení: podle toho, zda prostředí je dynamické nebo ne, online a offline plánovače cest nebo lokální a globální. Online je obvykle spojena s lokálním a offline s globálním plánovačem. Problémem je, že existují algoritmy, které lze vzít v úvahu pro obě kategorie. Například reaktivní výpočetní algoritmus zvaný Dynamický přístup k oknu (Dynamic Window Approach - DWA) se obvykle používá pro lokální plánování, ale lze jej použít i pro globální plánování. S ohledem na nejednoznačnost rozdělení použiji rozdělení obecné klasifikace pomocí čtyř tříd: Reactive Computing, Soft Computing, C-Space Search a Optimal Control.

Ideální plánovač cest musí být schopen zvládnout nejistoty v modelu daného světa. K minimalizování dopadu předmětů pro robota a jeho hledání optimální dráhy v minimálním čase. Robotická reprezentace světa v konfiguračním prostoru (C-space) a implementace algoritmu jsou dva hlavní komponenty pro globální plánování cesty a tyto dvě složky spolu souvisí a mají velký vliv navzájem v procesu určit optimální cestu pro robota procházet pracovním prostorem v optimálním čase. Algoritmy plánování cesty jsou obvykle založeny na reprezentace konfiguračního prostoru, jako jsou Voronoi diagram, pravidelné mřížky, zobecněné kužely a vrcholový graf, kde je C-space plný s datovými strukturami, které ukazují polohu a orientaci objektů a robota v oblasti pracovního prostoru včetně oblasti volného prostoru a zakázané oblasti s překážkami nebo bludišti. Za účelem zjednodušení problému plánování cesty je potřeba zajistit, aby robot běžel hladce a přitom se vyhýbal překážkám v nepřehledném prostředí. C-space musí být v souladu s použitým algoritmem. Proto je důležitý výběr algoritmu pro daný problém. Ve výpočetní technice datové struktury výrazně ovlivňují výpočetní náročnost a efektivitu implementace algoritmu. Existuje mnoho typů algoritmů k vyhledávání a manipulovat s datovou strukturou používanou k ukládání map prostředí.

Závěrem lze říci, že výzvy, kterým mobilní robot čelí, je třeba řešit, aby byl zajištěn efektivní výkon. Navigace je jedním z nejdůležitějších aspektů, které je třeba vzít v úvahu, pokud jde o mobilního robota, protože vyžaduje plánovací algoritmy a příslušné informace o poloze robota. To bude navigovat robota jeho předem definovanou cestou. Stejně jako je důležitá navigace, je

důležité i plánování trajektorie. To určí cestu, kterou musí robot sledovat, aby dosáhl svého cíle. Proto musí být trasa naplánována odpovídajícím způsobem, aby se zabránilo kolizi a překážkám. Pro vyhýbání se překážkám se uvažují různé algoritmy v závislosti na cíli, kterého má být dosaženo. Nakonec musí robot znát svou polohu a směr za čas. V tomto ohledu je pro shromažďování přesných informací vyžadována účinná lokalizační technika a spolehlivé senzory.[8][33][37]

3.2.11 Plánovací algoritmy

Algoritmy plánování cest založené na reaktivním výpočtu

Tato kategorie zahrnuje algoritmy plánování cesty, kde prostředí, obvykle mapa rozlišující mezi překážkovými a nepřekážkovými regiony, pouze ukazuje umístění a tvar existujících překážek. Algoritmy reaktivního počítání se obvykle používají jako lokální plánovače cest (pokrývající okolí robota s dynamickým přeplánováním) kvůli jejich schopnosti rychle zpracovat nové informace (např. ve formě nově objevených překážek), které často pocházejí z omezených palubních senzorů.

Lokální algoritmy obvykle plánují další cestu nebo manévr, aby se vyhnuly blízkým překážkám, přičemž se řídí globálním plánem vytvořeným jiným algoritmem. Tyto algoritmy mohou vypočítat lokální minimální cesty, ale také dokonce způsobit, že robot narazí, takže je tomu třeba věnovat zvláštní pozornost. Existují dvě podkategorie algoritmů reaktivního počítání: metody reaktivního manévru, kde přítomnost překážek určuje okamžitý další manévr robota, a metody lokální optimalizace, kde se stávající dráha upravuje podle přítomnosti překážek.

Reaktivní manévr

Algoritmy zde uvedené spoléhají na definování toho, jak robot v každém okamžiku reaguje na přítomnost překážek. Tato reakce může být definována podle formulace, která řeší umístění existujících překážek. Společným rysem různých formulačních přístupů jsou nízké výpočetní požadavky potřebné k vyvolání reakce, obvykle ve formě řízení nebo příkazu rychlosti. Protože tato formulace postrádá globální informace, tyto techniky se běžně používají jako lokální plánovače. Dotyčná formulace může být založena na použití polí k řešení lokalizace překážek, detekci hranic překážek k jejich obcházení nebo

vytvoření příkazu rychlosti po vyhodnocení dostupného volného prostoru nebo rychlosti pohybujících se překážek. Použití metod pole zahrnuje algoritmy zvané jako Umělé potenciální pole (APF) a Histogram vektorových polí (VFH).

V APF může být pohyb robota výsledkem součtu virtuálních sil, které vytvářejí vnější prvky, jako jsou překážky. Tímto způsobem se robot dostane dále od těchto překážek a vyhýbá se srážce s nimi, protože síly z nich jsou odpudivé. Přitažlivá síla vytvořená cílovou pozicí přiměje robota jít k ní. Hlavní nevýhodou této strategie je však to, že je náchylná k tomu, aby robot uvízl v místních minimálních bodech. K překonání této situace lze dosáhnout kombinací APF s genetickými metodami (Evoluční algoritmy).

Řešení inspirované elektrostatickými potenciálovými poli, ve kterém namísto použití součtu virtuálních sil vede robota tzv. skalární potenciální pole. VFH, vytvoří polární histogram pro vyhodnocení hustoty překážek kolem robota, přičemž vybere úhel řízení s nejnižší hustotou překážek.

Algoritmy Bug, Bug1 a Bug2, přimějí robota obejít jakoukoli překážku, která se na jeho cestě objeví, dokud nedosáhne cíle. Hlavní rozdíl mezi nimi je v tom, že Bug1 umožňuje robotovi řídit celou hranici jakékoli překážky, zatímco Bug2 ji může řídit pouze částečně. Mají minimální optimalitu ve prospěch jednoduchosti implementace a velmi minimálních výpočtů. Lze je použít na robotech vybavených pouze senzory, které právě detekují překážky v jejich bezprostřední blízkosti. Tímto způsobem tito roboti buď jedou k cíli, nebo jedou podél hranic překážek, které najdou.

Následující přístupy jsou zaměřeny na vytvoření příkazu rychlosti pro robota. Metody Velocity Obstacles (Rychlostní překážky) vypočítávají bezpečnou trajektorii s ohledem na vektory rychlosti jak robotického agenta, tak jakékoli jiné pohyblivé překážky. Tento výpočet vyhodnocuje kužel kolize.

Dynamický přístup k oknu (DWA) je algoritmus, který hledá v prostoru rychlosti příkaz rychlosti pro sledování kruhové trajektorie bez kolize, vymezené přípustnými hodnotami rychlosti a časovým oknem. Toto řešení nemusí být globálně optimální, ale spíše lokálně optimální. DWA lze použít i pro roboty pohybující se vysokou rychlostí.

Lokální optimalizace

Tyto algoritmy obvykle vycházejí z již existující cesty a podle toho upravují existující překážky. Zde je prioritou omezit výpočetní využití na minimum

na úkor ztráty optimality nebo dokonce úplnosti. Existují různé možnosti úpravy dráhy, od výběru rychlostních profilů v rámci rychlostního prostoru až po protažení a prodloužení dráhy působením umělých sil. [8][33][37]

Algoritmy plánování cesty založené na soft-computing

Tento druh algoritmu nemá v úmyslu najít přesné optimální řešení, ale spíše přiblížit se, tolerovat určitý rozsah nepřesnosti. Obecně platí, že tyto algoritmy vyžadují vyladění určitých parametrů uživatelem, aby správně fungovaly podle vlastností prostředí. Dokážou si poradit i s dynamickým prostředím a jsou vhodné pro problémy zahrnující velké množství proměnných a vysoký stupeň volnosti. Obecně však vyžadují vysoký počet výpočetních zdrojů. Tento přehled se řídí klasifikací, která rozlišuje mezi evolučními, fuzzy kontrolami a metodami strojového učení. Evoluční používá techniky inspirované biologií a přírodou. Začínají systémem tvořeným jednotlivci, který se v čase mění, vyvíjí. K výrobě regulátorů používají fuzzy pravidla a neuronové sítě. Tyto ovladače jsou velmi užitečné pro navigaci v původně neznámých scénářích a obecně vytvářejí cesty podle překážek, které robot na své cestě detekuje. Abychom to shrnuli, algoritmy Soft Computing umožňují vyladění řady opakujících se prvků, buď jednotlivců založených na přírodě, fuzzy pravidel nebo umělých neuronů, za účelem generování cesty.

Evoluční počítání

Evoluční algoritmy jsou většinou inspirované přírodou. Tyto algoritmy generují cestu, která je výsledkem vývoje populace. Tato populace se skládá z inteligentních jedinců, jejichž jednání je modelováno podle chování vyskytujícího se v přírodě. Tyto akce mohou zahrnovat úpravu sebe sama anebo interakci s jinými jednotlivci. V některých případech tyto operace implikují pohyb jednotlivců ve volném prostoru prostředí. Po provedení série těchto operací algoritmy aproximují optimální řešení. Výsledná cesta a čas konvergence závisí na politice chování přiřazené k jednotlivci, povaha scénáře a hodnoty přiřazené uživatelem k určitým konfigurovatelným parametrům. Příkladem konfigurovatelných parametrů je například počet jedinců, kteří zaplňují problém plánování cesty. Evoluční algoritmy zahrnují genetické metody a rojový optimizátor (Swarm Optimizer). První metoda využívá chromozomové modely, zatímco druhá modeluje chování živých bytostí. Jak již bylo zmíněno, genetické algoritmy pracují s jedinci modelovanými podle chromozomů. Tyto jedinci obsahují geny, stejně jako chromozomy, ve formě binárních čísel. Tato čísla kódují řešení, což je soubor trasových bodů tvořících cestu v konkrétním případě řešení problému při plánování cesty. Jinými slovy, každý chromozom v populaci představuje cestu.

Genetický algoritmus začíná náhodnou sadou chromozomů. Tato sada se vyvíjí pomocí tří procesů a to reprodukce, křížení a mutace. Reprodukce vytváří nové chromozomy kopírováním těch nejlepších. Odstraňuje i ty nejhorší. Křížení je proces, při kterém si chromozomy vyměňují své geny. Mutace zavádí náhodné změny v genech, aby stimulovaly prozkoumávání vyhledávacího prostoru a vyhýbaly se lokálním minimům. V důsledku neustálého opakování těchto procesů algoritmus konverguje.

Na rozdíl od algoritmů založených na genetických metodách, rojový optimizátor používá agenty, které se pohybují ve volném prostoru. Tito jedinci jsou ve většině případů modelováni podle zvířat. Po sérii iterací vytvoří pohyb těchto jedinců směrem k cíli, který nakonec konverguje k výsledné cestě. Částicový rojový optimizátor (The Particle Swarm Optimization - PSO), algoritmus vyniká svou jednoduchostí. Je inspirován chováním určitých skupin zvířat, jako jsou hejna ryb. Vytváří řadu částic, které se v průběhu času přemisťují, dokud se algoritmus nekonverguje. Tyto algoritmy hledají nejlepší pozice a komunikují spolu s ohledem na své předchozí zkušenosti. Dalším známým algoritmem je optimalizátor mravenčích kolonií (ACO), který, jak název napovídá, simuluje chování mravenců. Tento hmyz se pohybuje a zanechává při hledání potravy stopu feromonů. Tuto stopu mohou sledovat i ostatní mravenci. Ta místa, která obsahují více feromonů, tvoří průjezdní body nejlépe nalezené cesty. Zde je nejlepší cesta ta nejkratší. Podle stejné strategie se mohou virtuální mravenci pohybovat po mřížce a zanechávat více nebo méně feromonů podle jejich stavu týkajícího se cíle. Aby nedošlo k pádu do lokálního minima, některé práce kombinují tuto metodu s heuristickými funkcemi.[8][33][37]

Algoritmy plánování cest založené na C-Space-Search

Algoritmy v této kategorii považují pracovní prostor plánovače cest za prostor všech stavů nebo konfigurací dosažitelných robotem. Z tohoto důvodu většina děl v této kategorii označuje tento pracovní prostor jako C-Space (konfigurační prostor). Hlavní myšlenkou těchto algoritmů je použití diskrétní sady vzorků, které jsou součástí tohoto C-prostoru. Jinými slovy, C-Space je diskretizovaný. Tato sada vzorků zahrnuje počáteční a cílový stav, nebo alespoň vzorky relativně jim blízké. Tímto způsobem algoritmy provádějí vyhledávací operaci a navštěvují vzorky z této sady. V určitém okamžiku algoritmus najde a vrátí určitou podmnožinu vzorků spojujících počáteční a cílový stav představující výslednou cestu. To znamená, že vygenerovaná cesta silně závisí na tom, jak jsou tyto vzorky rozptýleny, jak jsou propojeny a jak jsou navštěvovány. Ve skutečnosti se kvůli této závislosti v některých přístupech následné zpracování provádí za účelem vyhlazení tvaru výsledné cesty.

Kategorie C-Space Search je rozdělena do dvou skupin algoritmů podle toho, jak diskretizují C-Space. Algoritmy vyhledávání grafů to dělají pomocí již z existujícího grafu. Každý z uzlů v tomto grafu představuje vzorek C-Space a je připojen k dalším blízkým uzlům. S ohledem na Sampling-Based algoritmy se zaměřují na vytváření anebo modifikaci vzorků v C-prostoru iterativním způsobem.

Vyhledávání grafů

Jak již bylo zmíněno, C-Space lze diskretizovat ve formě grafu. Algoritmy vyhledávání grafů plně nebo částečně navštěvují tento graf, dokud nenajdou cestu spojující počáteční a cílový stav. První algoritmy vytvořené v této kategorii vracejí cesty, jejichž trasové body jsou umístěny nad sousedními vzorky. Jinými slovy, spojení mezi po sobě jdoucími trasovými body cesty jsou shodné s okraji grafu. Tyto cesty následně závisí na tom, jak je graf strukturován. Existují různé grafové struktury ve formě dekompozice buněk a cestovní mapy. K vyřešení tohoto problému s omezením na okraje grafu byl vytvořen jiný druh algoritmu na vyhledávání grafů, algoritmus libovolného úhlu (Any-angle). Důvodem použití tohoto názvu je, že cesty vytvořené plánovači s omezením hrany používají pouze určité hodnoty orientace. Například v pravidelné mřížce s osmi sousedstvími, mohou mít cesty s omezením okraje pouze orientaci s 0, ±45, ±90, ±135 a 180 stupni. Algoritmy libovolného úhlu vytvářejí cesty, které nejsou omezeny na tyto orientace, protože jejich trasové body nemusí být nutně umístěny v sousedních uzlech. Nejznámějším a základním plánovačem cest s omezeným okrajem je Dijkstrův algoritmus. Jako počáteční krok tento algoritmus vezme jeden uzel, buď začátek, nebo cíl. Poté pokračuje v šíření informací svým sousedům. Může to být buď hodnota nákladů potřebných k dosažení od začátku, nebo náklady, které zbývají k dosažení cíle. Algoritmy opakovaně navštěvují sousedy již navštívených uzlů. Informace o výši nákladů se neustále šíří a algoritmy přidělují každému navštívenému uzlu nadřazený uzel. Pokud to prostředí dovolí, tedy žádné překážky neizolují cíl ani start, algoritmus nakonec navštíví oba. V tomto okamžiku je cesta načtena zpětným sledováním nadřazených uzlů. Jinými slovy, cesta začíná od posledního navštíveného uzlu a vede zpět přes nadřazené uzly. Skutečnost, že je inkrementální, znamená, že tento algoritmus recykluje předchozí výpočty, kdykoli dojde ke změnám v nákladech přiřazených uzlům sítě. Tím se zabrání tomu, aby algoritmus provedl zcela nový výpočet od začátku. Toto snížení výpočtů umožňuje rychlé přeplánování v případech, kdy se robot například na své cestě setká s novými překážkami. Vylepšená verze nazvaná Soustředěné D* (Focused D*) dokázala dále zkrátit dobu výpočtu D*. Protože algoritmy A* a D*, včetně jejich verzí, využívají heuristické funkce, výsledné cesty mohou být suboptimální. Pokud jde o algoritmy Any-angle, jedním z prvních byl Pole-D*. Jedná se o dobře známý algoritmus především díky jeho použití na NASA roverech použitých na MArsu. D* se jedná o inkrementální algoritmus, takže v následujících provedeních recykluje předchozí výpočty.

Na základě vzorkování

Algoritmy plánování cesty založené na vzorkování vytvářejí vzorky C-prostoru jeden po druhém podle různých zásad. Později získávají cestu z vytvořených vzorků po splnění určité podmínky nebo souboru podmínek, jako je například dosažení časového limitu. Tento druh algoritmu je asymptoticky optimální. To znamená, že mohou vytvářet více a více vzorků a pokoušet se najít lepší řešení, jak plyne čas. Obecně se tyto algoritmy obvykle používají pro vyhledávání ve vysoko rozměrných prostorech. Počet vzorků však může být relativně velký, aby se přiblížil globálnímu optimálnímu řešení, což vyžaduje použití velkých paměťových zdrojů pro uložení všech vzorků. Pokud jsou uvažovány pouze dva body (počáteční pozice a cíl), je algoritmus algoritmem s jedním dotazem, zatímco pokud je vybráno více bodů pro stejné prostředí, je algoritmus kategorizován jako více dotazový. Pokud jde o jeden dotaz, jedním z nejznámějších je algoritmus Rychle náhodný strom (Rapidly Random Tree - RRT), což je také speciální případ Rychle deterministický strom (Rapidly Deterministic Tree - RDT). Tento algoritmus emuluje rostoucí strom v tom smyslu, že od výchozího bodu jsou vzorky dynamicky vytvářeny, jako by to byly větve. Když je jeden ze vzorků blíže k cíli, než je určitá vzdálenost, lze cestu získat zpětným sledováním, dokud nedosáhnete výchozího bodu. Jak již bylo zmíněno, stále lze provést více iterací pro nalezení lepších cest. Vylepšená verze s názvem Informovaný RRT* zlepšila výkonnost RRT* vymezením elipsy ohraničující počáteční a cílovou pozici, když je nalezena proveditelná cesta. Další iterace ke zlepšení této cesty se provádějí v rámci této elipsy, místo aby algoritmus prozkoumal další možnosti, které pravděpodobně neovlivní výsledek.

Pokud jde o algoritmy založené na vzorkování s více dotazy, nejznámější je metoda pravděpodobnostního plánu (Probabilistic Roadmap Method - PRM). Tento algoritmus začíná sérií vzorků, které jsou již rozptýleny po C-prostoru. Odtud se vytvoří nové vzorky, které vytvoří nový strom z každého z těchto počátečních vzorků. Poté se k načtení cesty pomocí grafu vytvořeného PRM použije metoda grafický vyhledávač (Graph Search), jako je A*. [8][33][37]

Algoritmy plánování cesty založené na optimálním řízení

Základní linií algoritmů založených na Kontrolním přístupu (Control Approach) je vytvoření řídicí funkce, která přenese robota z počátečního stavu v C-prostoru do cíle. Jak název napovídá, problém plánování cesty je zde řešen pomocí optimálního řídicího přístupu. Hlavní rozdíl oproti metodám Soft Computation spočívá v tom, že neexistují žádné konfigurovatelné parametry, problém zde musí být plně uzavřen. Zde jsou dvě různé podkategorie. Prv-

ním z nich je řešení PDE, algoritmy řeší parciální derivační rovnici (PDE) na mřížce na základě principu dynamického programování (DPP). Druhá podkategorie je numerická optimalizace, zahrnuje algoritmy, které obecně optimalizují již existující cestu s ohledem na kino-dynamická omezení robota, aby byla proveditelná.

PDE-Solving-Based

Optimální přístup k řízení je zde založen na principu dynamického programování. Protože se jedná o parciální derivační rovnici (PDE), tato podkategorie se nazývá PDE Solving. Lze to považovat za nalezení numerického řešení problému výpočtu šíření vlny přes mřížku. Každému z uzlů mřížky je přiřazena hodnota doby příchodu vlny. Způsob, jakým se vlna šíří, bude záviset na tom, jak je formulována rovnice Hamilton–Jacobi-Bellmanovy (HJB), včetně nákladové funkce. Hlavní nevýhodou tohoto druhu algoritmu je, že se s ním obecně neumí vypořádat omezení ve formě diskontinuit. Ta není pouze statická, ale také zohledňuje nákladovou funkci a vrací pouze skalární hodnotu podle pozice na mapě. To znamená, že vlna se šíří na uzlu rychlostí, která závisí pouze na přiřazené skalární hodnotě. Tímto způsobem jsou charakteristické směry shodné s gradientem funkce celkových nákladů, a proto lze cestu získat jednoduše pomocí metody Gradientního sestupu (Gradient Descent). V průběhu let byla navržena celá řada metod pro výpočet řešení této formulace problému s nízkými výpočetními požadavky, a proto se nazývají rychlé metody. Jednou z nejznámějších je metoda rychlého pochodu (FMM). Tento algoritmus sleduje stejnou strategii jako algoritmus Dijkstra. Dijkstraův algoritmus je algoritmus pro nalezení nejkratších cest v grafu s nezápornými hranami. Začíná se výchozím uzlem a postupně rozšiřuje hranice známých nejkratších cest. V každém kroku vybere nejbližší dosud neprozkoumaný uzel a upraví jeho odhadovanou vzdálenost od výchozího uzlu na základě přilehlých hran. Algoritmus pokračuje, dokud nejsou prozkoumány všechny uzly nebo je dosaženo cílového uzlu. Výsledkem je nejkratší cesta z výchozího uzlu do všech ostatních uzlů v grafu. Na rozdíl od Dijkstra, FMM přiřazuje hodnotu částky nákladů každému uzlu řešením algoritmu Eikonal. Výsledná cesta je hladká, souvislá a optimální. Hodnoty nákladů, které používá Eikonal, určují rychlost šíření vypočítané vlny. Pro hladké vyhýbání se překážkám počítá Fast Marching Square (FMS) dvakrát FMM a jako první vytváří odpudivé pole obklopující překážky. Pro práci s obecnějšími výrazy Hamilton–Jacobi-Bellmanovy (HJB) rovnice je třeba použít jiné druhy metod. FMM poskytuje suboptimální výsledky, je-li použit se směrově závislými náklady (anizotropie). Tento druh nákladů znamená, že vlna se šíří odlišně v závislosti na jejím směru vzhledem k tomu, jak jsou uzlu přiřazeny vektorové náklady. Existují konkrétní situace, ve kterých FMM produkuje přesné výsledky pod určitou úrovní anizotropie, jako je například nákladová funkce formulovaná tak, že se mění většinou ve směrech rovnoběžných s referenčními osami.

Globální optimalizace

Tato podkategorie obsahuje algoritmy plánování cesty, které optimalizují existující předběžnou proveditelnou cestu. Na rozdíl od lokálních optimalizačních metod se globální optimalizační metody dělají výslednou cestu globálně optimální výměnou za cenu investování většího výpočetního zatížení. První krok používá metody založené na vzorkování, jako je RRT nebo PRM. Druhý krok spočívá v použití technik optimalizace gradientu k aproximaci optimálního řešení z této proveditelné cesty. [8][33][37]

Část II

Navrh a realizace

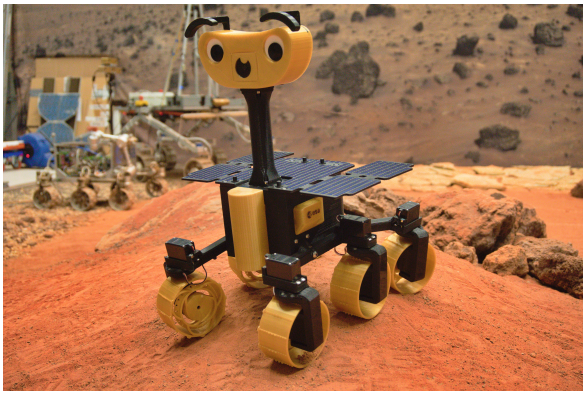
Kapitola 4

Návrh a realizace

V této části se budu zabývat popisem, návrhem, realizací a řešením problematiky, která v průběhu nastala. Z hardwarového hlediska chci použít a upravit šestikolovou platformu Exomy navrženou European Space Agency. Za řídicí jednotku použiji Nvidia Jetson Tegra a jako hlavní senzor jsem vybral Lidar a RealSense D455 a jeden z nich po simulaci následně implementuji na reálného robota. Za software chci použít The Robot Operating System (ROS). Cílem tedy bude upravit šestikolovou platformu Exomy, spojit hardware a software, tak aby byl mezi sebou kompatibilní. Následně sestavit program pro úspěšnou lokalizaci robota v prostředí, a po zadání cílového bodu, nalezení optimální cesty v prostředí s překážkami a dosažení jej.

4.1 Návrh a úprava robota

Zde se budu zabývat návrhem a popisem zvolené robotické platformy. Abych nevytvářel naprosto od nuly, rozhodl jsem se upravit už existující platformu a to projekt EXOMY od European Space Agency, ten je vidět na obrázku 4.1.



Obrázek 4.1: Původní projekt Exomy[39]

Cílem bylo provést úpravy, aby konstrukce byla co nejuniverzálnější a mohla být využita i na jiné budoucí práce. Každé kolo této šestikolové platformy má dva servo motory. Jedno na pohyb a druhé na natáčení daného kola, jedná se tedy o kolo se dvěma osami rotace. Robot bude tedy schopen pohybovat se několika systémy řízení, jako je differenciální řízení, řízení Ackerman, aktivní přední a zadní řízení, bod otáčení nebo crabbing. Řídíci jednotkou je Raspberry PI, která je umístěna uvnitř s baterií a drivem pro motory. Tuto platformu jsem zamýšlel upravit tak, aby byla co nejvíce univerzální a dala se použít i pro jiné budoucí práce. Jako první jsem se rozhodl vyndat raspberry pi a umístit ho zvenčí na horní víko. Jedním z důvodů je, že pokud bude potřeba použití jiné řídící jednotky, která bude mít jiné rozměry, bude zapotřebí vytištění pouze nového víka, které se vytiskne zhruba za hodinu. Na rozdíl od 7-8 hodin tisknutí celé základny na nejvyšší výšku vrstvy tisku. Tím jsem získal ve vnitřním prostoru volný prostor, kterého se mohu zbavit a snížit tak základnu. Na obrázku 4.2a A. je navržená a aplikovaná úprava. V původní verzi byli ze zadní strany základny umístěny dva nevhodně velké vypínače, které jsem musel díky snížení základny přemístit na víko a vyměnit za klasické páčkové vypínače. Primárním senzorem, který chci použít je Lidar nebo RealSense D455 a protože jsem netušil kolik a jaké další senzory v kombinaci použiji, rozhodl jsem se vytvořit stohovatelné podlaží. Pro univerzálnost jsem vytiskl desku bez první a poslední vrstvy, tudíž je vytištěna mřížka vnitřku. Mohu na ní tedy připevnit cokoliv a kamkoliv. Určitě to není tak robustní jako když bych to vytiskl normálně, ale to mohu provést, až budu mít jasno jaké senzory použiji a kam je umístím. V zadní straně desky jsem umístil otvor pro kabeláž. Upravil jsem přední kryt baterie a vytiskl ho průsvitným materiálem, aby šel vidět stav napětí baterie, protože u krytu je umístěn voltmetr. Na závěr jsem navrhl a zhotovil držák pro senzor RealSense, který je možno naklápět a mít například možnost být stále ve stejné poloze, při jízdě na nerovném povrchu. Mým prvním výsledkem byla realizovaná úprava, která by se dala vzhledově popsat jako šestikolová obdoba robotické platformy turtlebot.

Po této úpravě se naskytla možnost použít Nvidia Jetson Tegra, což je výkonnější řídicí jednotka než Raspberry PI. Zároveň se projevily určité nedostatky v tuhosti navržené konstrukce a přibyly optimalizační nápady. Bylo vytištěno nové tělo základny s odvětrávacími průduchy, horní víko bylo uzpůsobeno jednotce Nvidia Jetson Tegra a bylo přidáno pouze jedno podlaží se zadním držákem pro použití kamery RealSense D455. Vpředu je stále možnost přidat senzor Lidar. Finální úprava je na Obrázku 4.2b.

(a) : Prvotní úprava

(b) : Finální úprava

Obrázek 4.2: Úprava kontrukce projektu Exomy

4.2 Softwarová příprava

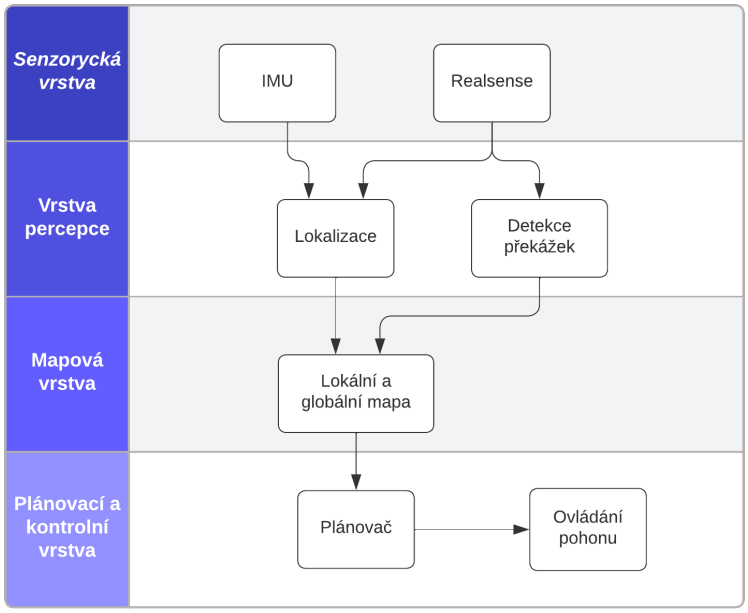
Má práce obsahovala mnoho problémů s kompatibilitou, jak hardware se softwarem, tak i softwary mezi sebou i balíčky či knihovnami v ROSu. První problémy se objevily hned na začátku. Protože všechny osobní zařízení mám s operačním systémem Windows, chtěl jsem simulaci provést v tomto operačním systému. Jako první jsem tedy vyzkoušel nahrát do operačního systému Windows ROS. To skončilo neúspěchem. V této části pokud mluvím o neúspěchu, jedná se především o chybové hlášení při instalaci ROSu nebo nějakého balíčku pro ROS, pro které jsem nenašel řešení. Zkusil tedy jinou metodu a to stáhnout Linuxový terminál pro Windows a do něj nahrát ROS. Tento pokus také skončil neúspěchem. Následně jsem zkusil WSL, to je Windowsový subsystém pro Linux, který jsem měl nahraný ve Visual Studio Code. S ním jsem nedocílil také žádného úspěchu. Zbývaly mi 2 možnosti. Vytvořit virtuální počítač a na něm mít Linux nebo udělat čistou instalaci operačního systému Linux. Rozhodl jsem se nejdříve vyzkoušet virtuální počítač. První úspěch, po nahrání Ubuntu 22.04 jsem měl funkční i ROS1. Po dalším pokračování jsem

zjistil určitou nekompatibilitu v některých knihovnách a byl jsem nucen udělat tento proces znovu, ale pro Ubuntu 20.04. Teď jsem byl konečně schopen začít řešit simulaci. Systém jako celek byl i přes dobrou výkonnost mého počítače nedostatečně rychlý a musel jsem nakonec přejít na čistou instalaci Ubuntu 20.04 bez virtuálního počítače s verzí ROS1 Noetic. Trvalo mi mnoho hodin, než jsem dal dohromady pouze správnou kombinaci verzí, aby tyto softwary byly mezi sebou kompatibilní a plně funkční. Nyní jsem připraven začít řešit simulaci.

4.3 Simulace

4.3.1 Konceptuální model

Autonomní řízení musí být robustní vůči chybám, musí být spolehlivé a to vše v reálném čase. Proto jako první jsem si udělal konceptuální model, abych měl základ, od kterého začít. Na obrázku 4.3 je znázorněna má představa softwarové architektury. Ta vychází z paralelního řetězového zpracování. Vstupní data jsou analyzována a postupně přeměněna v potřebné informace, až nakonec jsou přeměněna ve výstupní informace a konkrétní pohybové úkony. Paralelní zpracování datových toků minimalizuje dobu zpracování.



Obrázek 4.3: Konceptuální model autonomního robota

Koncept se skládá ze čtyř vrstev:

- Senzorická vrstva zahrnuje moduly k jednotlivým senzorům, které provádí nízkoúrovňové předzpracování dat.
- Vrstva percepce provádí vysokoúrovňové zpracování dat. Hlavní moduly jsou modul lokalizace, ten je dvojího typu, lokální a globální lokalizace a modul detekující překážky.
- Mapová vrstva se skládá z lokální mapy tvořená z lokální lokalizace a globální mapa tvořená z lokálních map a z globální lokalizace.
- Plánovací a kontrolní vrstva se skládá z plánovače, který plánuje navigaci a řízení robota pro lokální i globální plán.

4.3.2 Robot Operating System

Pro navigaci jsem zvolil použít rozhraní ROS (Robot Operating System). Jedná se o meziprocesorovou komunikaci a má velkou uživatelskou podporu s velkou komunitou, která tvoří knihovny a dokumenty, pro lepší porozumění a aplikaci pro mé účely. Je vytvořeno mnoho knihoven obsahujících lokalizaci, navigační algoritmy a mnoho dalšího užitečného, pro mou práci. Jako celek představuje ekosystém, který značně usnadňuje a urychluje vývoj robotů. Samotný vnitřek ROSu je velká část knihoven s licencí BSD, která umožňuje jakoukoliv část programu změnit a použít, jak pro soukromé, tak i pro komerční účely.

Hlavní ROS struktura se skládá Nodů (uzlů) a Topiců (téma). Každý Node v ROSu by měl být zodpovědný za jediný modulární účel, jako například ovládání motorů nebo publikování dat ze senzoru. Každý Node může posílat a přijímat data od ostatních Nodů přes Topicy. Topic funguje jako sběrnice pro Nody pro výměnu informací. Node může posílat data k libovolnému počtu Topiců a současné přijímat libovolný počet Topiců. Topic je jedním z hlavních způsobů, jak se data přesouvají mezi Nody, a tedy i mezi různými částmi systému. Celý robotický systém je složen z několika Nodů pospojovaných mezi sebou.

Hlavní struktura se skládá z Nodů, Topiců a Servisů.

- V zásadě Nody jsou procesy, které vykonávají úlohy. Samotné Nody jsou ve skutečnosti softwarové procesy, ale se schopností komunikovat s

ostatními Nody v systému. Myšlenkou návrhu ROS je, že každý Node je nezávislý a interaguje s ostatními Nody pomocí komunikační schopnosti ROS. Jednou ze silných stránek ROS je, že konkrétní úkol, jako je ovládání kolového mobilního robota, lze rozdělit na řadu jednodušších úkolů. Úkoly mohou zahrnovat vnímání prostředí pomocí kamery nebo laserového skeneru, tvorbu map, plánování trasy, sledování stavu baterie robota a ovládání motorů poháněcích kola robota. Každá z těchto akcí se může skládat z Nodu ROS nebo řady Nodů k provedení konkrétních úkolů. Node může nezávisle spouštět kód k provedení svého úkolu, ale může také komunikovat s jinými Nody odesíláním nebo přijímáním zpráv. Zprávy mohou obsahovat data, příkazy nebo jiné informace nezbytné pro aplikaci.

- Některé Nody poskytují informace pro jiné Nody. O takovém Nodu se říká, že zveřejňuje informace, které mohou přijímat jiné Nody. Informace v ROS se nazývá Topic. Nody, které přenášejí data, zveřejňují název Topicu a typ zprávy, která má být odeslána. Skutečná data zveřejňuje Node. Node se může přihlásit k odběru Topicu a odeslané zprávy na tento Topic přijímá Node, který se o ně přihlašuje. Node může publikovat i odebírat jeden nebo více Topiců.
- Servisy (Služby) jsou dalším způsobem komunikace pro Nody v grafu ROS. Servisy jsou založeny na modelu volání a odpovědi oproti modelu vysílání a odebírání Topiců. Zatímco Topicy umožňují Nodům přihlásit se k odběru datových proudů a získávat nepřetržité aktualizace, servisy poskytují data pouze tehdy, když jsou konkrétně volána klientem.

4.3.3 Grafické prostředky ROSu

ROS obsahuje grafické nástroje vytvořené pomocí softwarového nástroje Qt a jsou nazvány rqt. Tyto nástroje mohu použít při navrhování a ladění programu robota. Jiné grafické nástroje mohou být použity pro simulaci jako je Gazebo či ovládání robotu přes Rviz. Pro lepší pochopení popíši grafické prostředky, které jsem k mé práci použil.

Program Rviz

Rviz je 3D vizualizační nástroj vyvinutý jako součást frameworku Robot Operating System (ROS). Jedná se o software s otevřeným zdrojovým kódem, který poskytuje platformu pro vizualizaci dat a modelů robotů v reálném čase. Je navržen tak, aby podporoval širokou škálu robotických platforem a senzorů,

včetně kamer, laserových skenerů a 3D senzorů. Poskytuje uživatelsky přívětivé rozhraní pro vizualizaci a analýzu dat robotů, což vývojářům umožňuje rychle a snadno pochopit, jak se jejich roboti chovají. RVIZ také poskytuje bohatou sadu vizualizačních nástrojů, včetně interaktivních značek, trajektorií a mřížkových map.

Program Gazebo

Gazebo je simulační prostředí pro roboty vyvinuté jako součást v rámci Robot Operating System (ROS). Jedná se o open-source software, který poskytuje platformu pro testování a vyhodnocování chování a algoritmů robotů ve virtuálním prostředí. Je navržen tak, aby podporoval širokou škálu robotických platforem, včetně mobilních robotů, humanoidních robotů a vzdušných vozidel. Jednou z klíčových vlastností Gazebo je jeho schopnost simulovat senzory a akční členy. To umožňuje vývojářům testovat a vyhodnocovat různé senzory a akční členy bez potřeby fyzických prototypů. Gazebo také poskytuje fyzikální engine, který simuluje dynamiku robota a jeho prostředí, což umožňuje vývojářům testovat výkon jejich algoritmů za různých podmínek. Gazebo je úzce integrován s ROS, což umožňuje vývojářům snadno připojit své modely robotů k jinému softwaru s podporou ROS. Tato integrace usnadňuje testování a hodnocení různých algoritmů a řídicích strategií v simulovaném prostředí před jejich nasazením na fyzického robota.

Program Rqt_graph

Rqt_graph je nástroj grafického uživatelského rozhraní pro vizualizaci Nodů a Topiců, které jsou v danou chvíli pro systémem ROS spuštěny. Jedná se o software, který poskytuje platformu pro vizualizaci spojení mezi Nody a Topicy, která publikují a naslouchají. Rqt_graph je navržen tak, aby pomohl vývojářům porozumět chování jejich systému ROS tím, že poskytuje grafické znázornění toku dat mezi různými Nody. Umožňuje vývojářům snadno vizualizovat vztahy mezi různými Nody a Topicy a identifikovat potenciální problémy. Jednou z klíčových vlastností rqt_graph je jeho schopnost dynamicky aktualizovat graf při přidávání nebo odebírání Nodů a Topiců ze systému. Díky tomu mohou vývojáři snadno sledovat chování svého systému v reálném čase a rychle identifikovat jakékoli problémy, které mohou nastat. Rqt_graph také poskytuje řadu možností přizpůsobení, včetně možnosti filtrovat graf podle názvu Nodu nebo Topicu, zvýraznit konkrétní Nody nebo Topica a upravit rozložení grafu tak, aby lépe vyhovovalo preferencím uživatele. Rqt_graph

je součástí sady nástrojů rqt (ROS Qt), což je sada nástrojů GUI (grafické uživatelské rozhraní), které jsou vytvořeny pomocí rámce Qt.

Program Rqt_tf_tree

Rqt_tf_tree je nástroj grafického uživatelského rozhraní pro vizualizaci transformací v systému ROS. Poskytuje platformu pro vizualizaci hierarchie transformací souřadnicových systémů (fyzickou provázanost), které se používají k popisu polohy a orientace objektů v prostředí robota. Umožňuje vývojářům snadno vizualizovat vztahy rodič-potomek mezi různými souřadnicovými systémy a identifikovat potenciální problémy nebo nekonzistence v jejich transformačním stromu.

4.3.4 Popis programu

- Spuštění senzoru RealSense (launch)
- Spuštění programu Exomy (launch)
  - robot node
  - joint command node
  - robot state publisher node
  - motors node
  - visual odometry node
  - slam node
- Spuštění navigace (launch)

Seznam hlavních balíků

- RealSense
- Exomy
- RTAB-map
- Nav2
- depthimage_to_laserscan

URDF model robotu

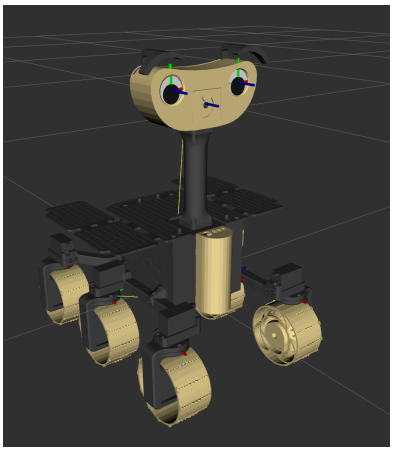
Pro získání modelu robotu a definování jednotlivých částí, tvaru a fyzikálních vlastností slouží soubor URDF (Unified Robot Description Format - Jednotný formát popisu robota), který využívá XML formát reprezentace. Soubor se skládá z elementů, základním elementem je <robot>. Tento element je povinný atribut a slouží jako název celé sestavy. Pod ním jsou složeny další elementy. V mém případě jsou použity primárně <link> a <joint>. Link popisuje tělo objektu a v něm se dají nastavit kolizní vlastnosti, vzhled a moment setrvačnosti. Joint popisuje vazbu mezi dvěma objekty (linky). Je možné zde nastavit typ vazby jako fixní, rotace a posuv. Každý link musí obsahovat přiřazení, který objekt je rodič (nadřazený) a který je potomek (podřazený rodiči). Rodič a potomek stanovuje souřadný systém potomka a je odvozený od souřadného systému rodiče. Jsou zde rozdíli URDF souboru mezi simulací a při použití na reálném robotu. Při použití v simulaci Gazebo, je zde přidán plagin, který simuluje senzory, ten při použití na reálnem robotu není potřeba.

TF Topic

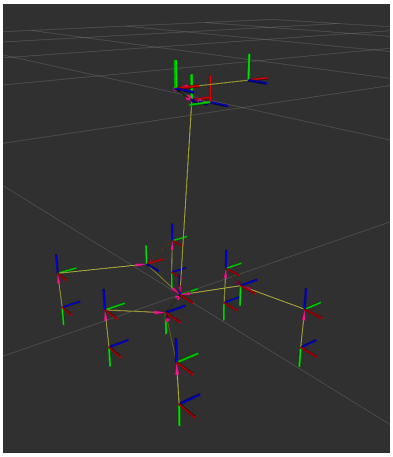
Každý objekt má svůj vlastní souřadný systém, který je relativní k souřadnému systému jiného objektu. ROS sám o sobě nemá žádný absolutní bod, ke kterému by se jednotlivé objekty vázaly, proto pozice každého objektu je odkázána na souřadný systém jiného objektu a jako celek tvoří hierarchický strom. Tento strom je reprezentován transformační funkcí (TF) posílána do Topicu /tf, který určuje vztahy souřadných systému jednotlivých objektů. V praxi a i v mém případě je tomu tak, že mapa je stanovena jako fixní a tvoří počátek souřadného systému. Na tu je připojena odometrie, tedy odhadovaná pozice robotu. Na odometrii je připojen samotný robot, kde se daný strom skládá od základny až po jednotlivá kola a jejich natočení.

```
^Cthezerdas@thezerdasVBox:~$ ros2 topic echo /tf_static
transforms:
- header:
    stamp:
      sec: 0
      nanosec: 0
    frame_id: base_link
  child_frame_id: base_footprint
  transform:
    translation:
      x: 0.0
      y: 0.0
      z: 0.0
    rotation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
```

Obrázek 4.4: Příklad výpisu Topicu tf



(a) : model robotu



(b) : TF modelu robotu

Obrázek 4.5: Porovnání modelu a TF modelu

Popis jednotlivých Nodů

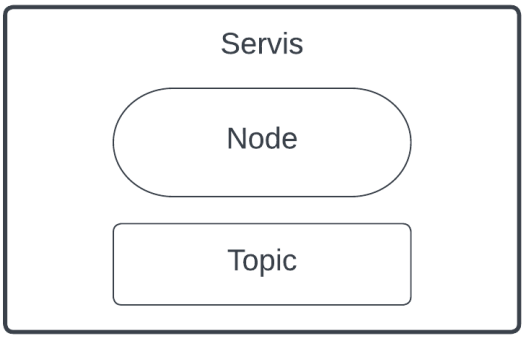
- **Robot node modified** - Node, který posílá příkazy rychlostí a natočení kol. Původní Node se řídil, podle Nodu vysílající informace z joysticku. Tento Node je upravený, aby přijímal informace z klávesnice a tudíž byl kompatibilně ovladatelný pro navigační algoritmy.
- **Joint_command_node** - Node, který vezme soubor URDF s upraveným modelem Exomy a vytvoří příslušný Topic /joint_states, který slouží k fyzikálnímu popisu jednotlivých pohyblivých částí.
- **robot_state_publisher** - Node, který vezme soubor URDF s upraveným modelem Exomy a z Topicu /joint_states vytvoří model robotu a

souřadné systémy jednotlivých objektů mezi sebou (TF funkce).

- **motors node** - Node, který ovládá jednotlivé motory.
- **rviz node** - Node, který spustí předem nastavený zobrazovací rozhraní Rviz.
- **gazebo node** - Node, který spustí přednastavené simulační prostředí v softwaru Gazebo.
- **spawn_rover** - Node, který přidá do simulace model robota.
- **depthimage_to_laserscan node** - Node, který převede hloubkovou mapu prostředí na data ve tvaru 2D bodů, jako pro snímaní senzorem Lidar v nastavené výšce.
- **RealSense node** - Node, který zpracovává data ze senzoru RealSense.
- **visual_odometry node** - Node, který z dat posílaných od RealSense Nodu vytváří odometrii.
- **slam node** - Node, který čte data posílané od RealSense Nodu a Nodu visual_odometry a vytváří z těchto dat simultánní lokalizaci a mapu.
- **nav node** - Node, který plánuje cestu a naviguje robota v prostředí.

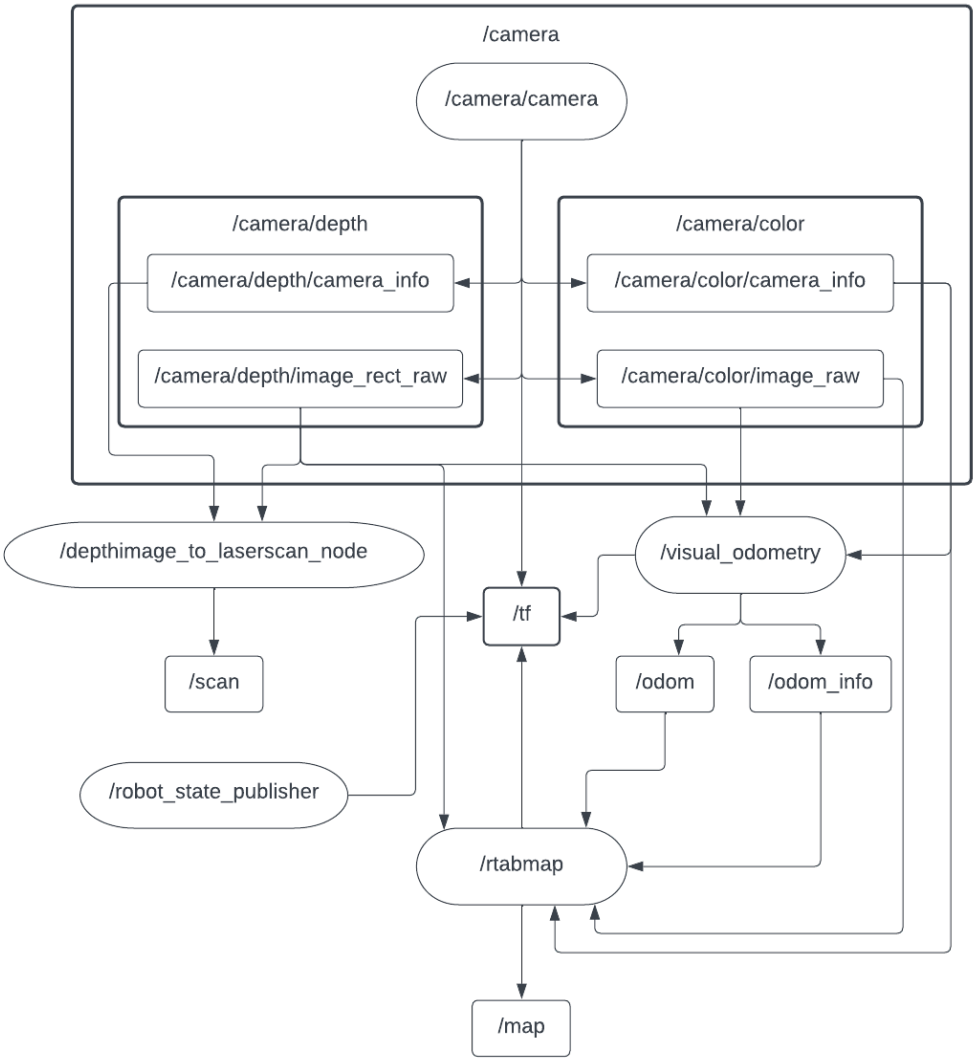
Schéma řídicího softwaru

Celé schéma řídicího softwaru je rozsáhlé a spletité, pokusím se rozdělit na části a popsat. Celý robotický systém je složen z několika nodů pospojovaných mezi sebou Topicy nebo Servisy. Na obrázku 4.6 je znázorněná grafická vizualizace těchto jednotlivých částí.



Obrázek 4.6: Grafická vizualizace Servisu, Nodu a Topicu

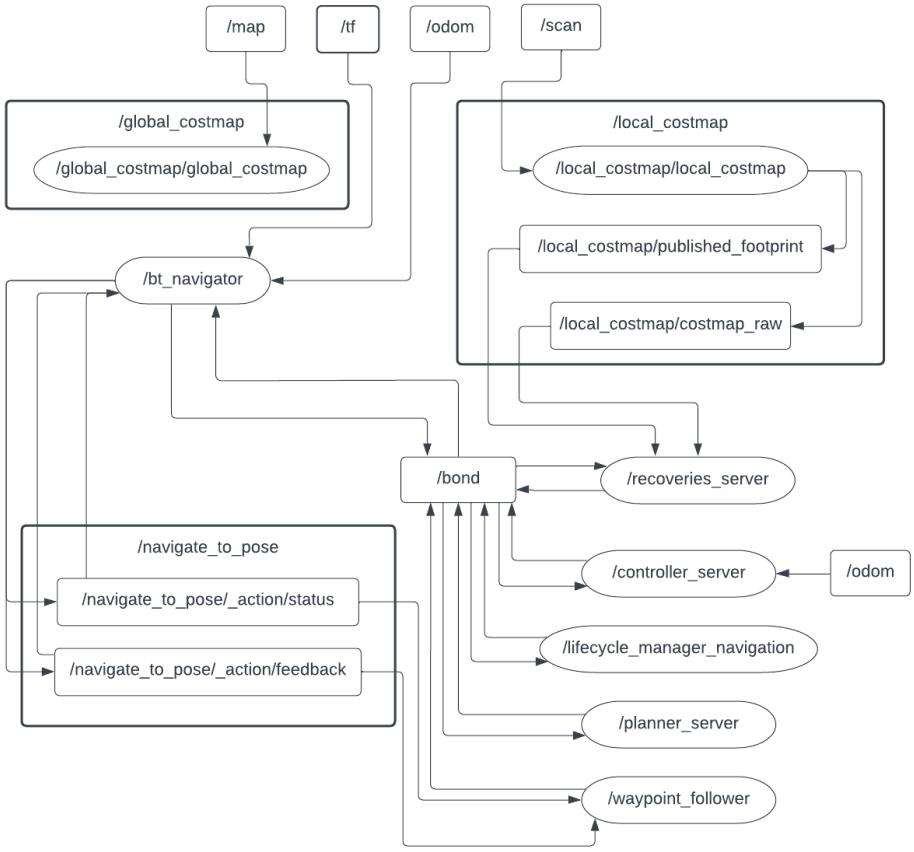
Na obrázku 4.7 je znázorněn začátek schématu řídicího softwaru. Začíná senzorickou vrstvou, kde se načtou a zpracují data ze senzoru RealSense pomocí nodu /camera a vysílá tyto data do Topiců. Z těchto Topiců si je čte Node /depthimage_to_laserscan a Node /visual_odometry. Node /depthimage_to_laserscan převádí data do formy 2D lidarových dat a vysílá je do Topicu scan. Node /visual_odometry vytváří odometrii a vysílá ji do Topiců /odom a /odom_info. Z Topiců odometrie a senzoru si Node rtabmap vytváří mapu a lokalizaci. Mapu vysílá do Topicu /map a lokalizaci do Topicu /tf. Do Topicu /tf posílají data také Nody /camera, /robot_state_publisher a /visual_odometry a čtou tyto transformační data všichni, co potřebují informaci o poloze robota.



Obrázek 4.7: Schéma první části řídicího softwaru

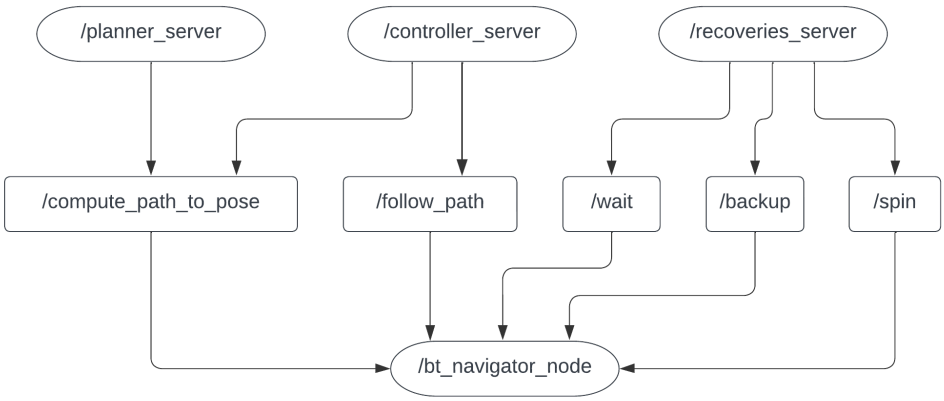
V tuto chvíli nám software ze senzoru vytváří odometrii, mapu a lokalizuje se v prostředí. Následně na obrázku 4.8 je znázorněná část navigace. Navigační část si nejdříve z mapy vygeneruje globální hodnotovou mapu a z Topicu /scan vytvoří lokální mapu. Zároveň si Node /bt_navigator vezme data z Topiců /tf a /odom a oboustranně si vyměňuje informace s /navigate_to_pose. Node /bt_navigator implementuje rozhraní úlohy navigace na pozici. Jedná se o implementaci navigace na bázi stromu chování, která má umožnit flexibilitu v navigační úloze a poskytnout způsob, jak snadno specifikovat složité chování robota, včetně obnovy. Node /navigate_to_pose naviguje z výcho-

zího bodu do cíle zadaného bodu ve volném prostoru. Dále je zde spuštěn Node /lifecycle_manager_navigation (Spravovaný životní cyklus nodů), který umožňuje větší kontrolu nad stavy systému ROS. Zajišťuje, aby při spuštění softwaru všechny komponenty byly správně vytvořeny, než umožní kterékoli komponentě začít cokoliv vykonávat. Umožní také restartování nebo výměnu Nodů online. Správce životního cyklu také obsahuje připojení /bond ke každému serveru životního cyklu. To znamená, že pokud se server zhroutí nebo se ukončí, správce životního cyklu bude neustále kontrolovat. To funguje jako hlídací pes během běhu, který doplňuje přechody správce životního cyklu nahoru a dolů z aktivních stavů. Topic /bond zde propojuje Nody /bt_navigator, /recoveries_server, /controller_server, /lifecycle_manager_navigator, /planner_server a /waypoint_follower. Node /recoveries_server provádí obnovu při selhání nodů. Planner Server zpracovává požadavky plánovače. Controller Server zpracovává požadavky kontroleru. Node /waypoint_follower implementuje způsob sledování trasového bodu pomocí serveru /navigate_to_pose. Vezme sadu uspořádaných navigačních bodů, které je třeba sledovat, a pokusí se k nim navigovat v pořadí. Pokud není trasový bod dosažitelný, určí, zda pokračovat k dalšímu bodu nebo zastavit.



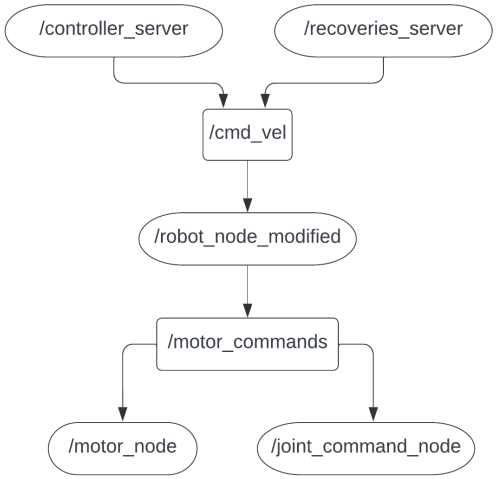
Obrázek 4.8: Schéma druhé části řídicího softwaru

Na obrázku 4.9 je druhá část navigace. Zde se počítá cesta do cílového bodu, posílají se informace o správnosti sledování cesty a při poruše se přes /recoveries_server provede obnovení nebo při identifikaci zapadnutí robota se pokusí o vyproštění.



Obrázek 4.9: Schéma třetí části řídicího softwaru

Obrázek 4.10, Nody /recoveries_server a /controller_server jsou Nody, které vysílají požadavky na řízení robota přes Topic /cmd_vel. Tato informace se následně zpracuje a projde přes Node /robot_node_modified do Topicu /motor_commands a vyšlou se do robota přes Topicy /motor_node a /joint_command_node. Takto jsou řízeny kola a jejich natočení.



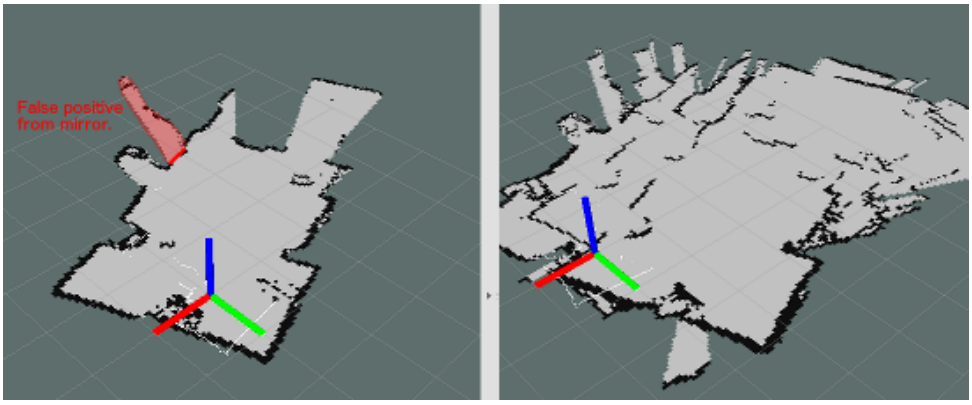
Obrázek 4.10: Schéma čtvrté části řídicího softwaru

4.3.5 Postup

Při navrhování v simulaci, jsem se kvůli popularitě a podle mého aktuálně jedním z nejlépe provedeným návodům inspiroval TurtleBot3. TurtleBot je malý, modulární, programovatelný mobilní robot, založený na ROSu, který se používá pro výuku a navrhování. Díky velké popularitě, skvělým návodům a velké komunitní podpoře jsem byl schopen pochopit fungování a strukturu ROSu pro vývoj a testování mé vlastní práce a stala se ideální inspirací pro začátek.

Po nainstalování všech potřebných komponent jsem po několika testech spustil simulaci s turtlebot burger a byl jsem schopen ho ovládat přes klávesnici. Při dalších pokusech se mi povedlo vytvořit mapu prostoru s překážkami, pomocí nástroje gmapping, který je součástí softwarového balíčku TurtleBot3. Po vytvoření mapy jsem spustil algoritmus pro plánování cesty, který umožňuje navigování v prostředí. Po zadání cílového bodu, robot začal plánovat a navigovat k tomuto cíli. V tomto základu je několik obtíží, které jsem chtěl eliminovat. Nejprve jsem chtěl vyřešit, aby se vše pouštělo co nejméně příkazy a nemuselo být potřeba mít otevřeno mnoho terminálů. To se dá uskutečnit launch (spouštěcím) souborem. Další problematikou je, že po každém prvotním spuštění je nutno ukázat na mapě, kde se robot nachází. Tento problém jsem vyřešil vytvořením Nodu, který tuto prvotní pozici pošle robotu sám.

Než budu dále pokračovat je zde problematika simultánní lokalizace a mapování, kterou musím brát na vědomí. Hromadění lokalizačních chyb způsobených šumem v měření v odometrii a vizuálních senzorech. Hromadění takových typů chyb obvykle způsobuje velké odchylky od skutečné polohy mapy nebo překážek na mapě. Jinými slovy při aktualizaci mapy, se nová mapa obtiskne posunutá na předchozí. Tomu chceme zabránit snížením chyb v měření polohy robota, protože se v průběhu času chyby hromadí a dělá to mapu plnou falešných překážek a tudíž nepoužitelnou.



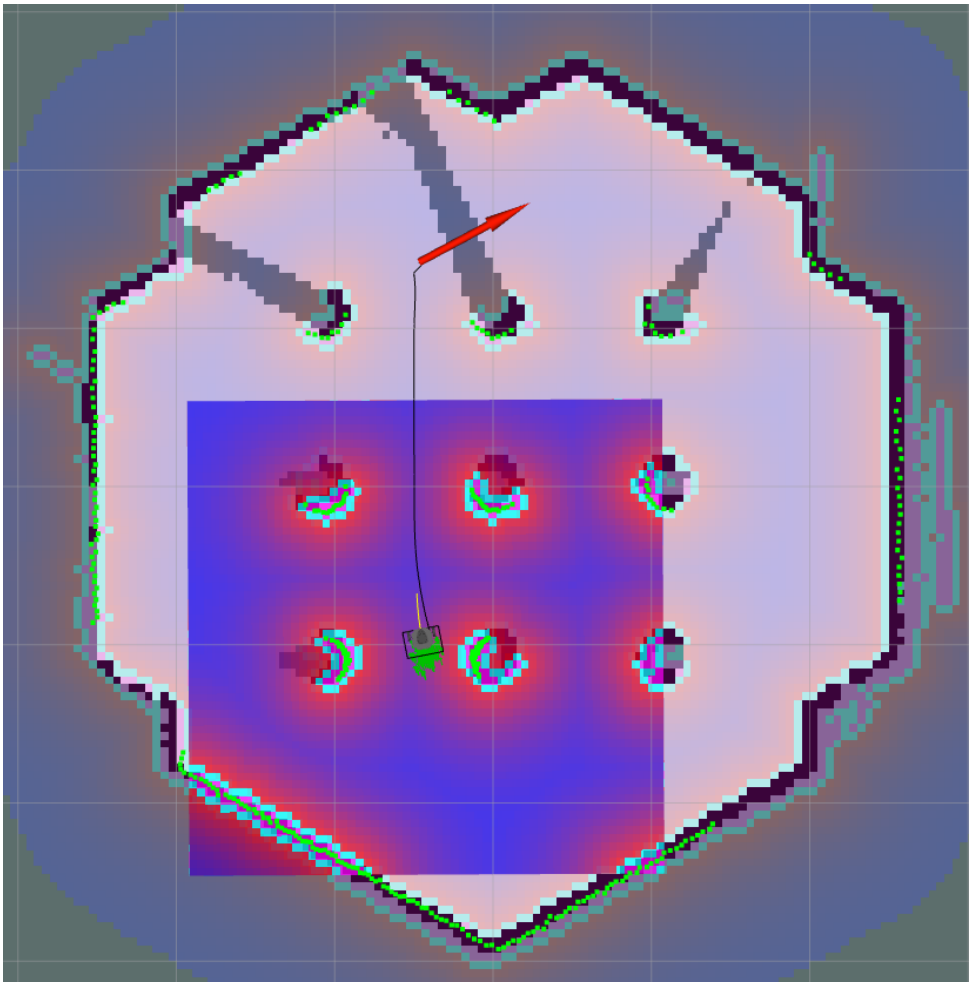
Obrázek 4.11: Levá strana představuje správně složenou mapu. Pravá strana naznačuje danou problematiku.

Také pozice robota na mapách s nahromaděnou chybou se velmi liší od skutečné pozice robota. To nás přivádí k dalšímu problému, kterým je skluz polohy robota na mapě oproti skutečnosti. To znamená pokud robot projede nějakou cestu a vrátí se do původní pozice, pozice na mapě bude posunuta. Chyby v odhadu, jako jsou tyto, jsou nevyhnutelné. Existují algoritmy, které tyto problémy jsou schopny vyřešit. Nejrozšířenější je rozšířený Kalmanův filtr (EKF) a Monte Carlo lokalizace (MCL), který lokalizuje mobilní roboty podle spojování pravděpodobnostních a mobilních informací o pohybu robota. V této fázi jsem jako řešení těchto problémů použil balíček slam_toolbox a vytvořil program, který spouští algoritmus adaptivní Monte Carlo lokalizaci. V pozdější fázi realizace jsem, ale přešel na balíček RTAB-map, který obsahuje všechny potřebné algoritmy na eliminaci těchto problémů. Důvod změny je popsán v kapitole *Aplikace simulace na reálného robota*.

Následný problém je, že metoda SLAM používá složité algoritmy, které mají vysoké výpočetní náklady se zpracování obrazu, mračna bodů. Proto je podstatné mít dostatečně výkonný hardware, který samozřejmě musí být kompaktní a nízkoenergetický. Abych získal SLAM přesný, je nutné provádět zpracování obrazu, zpracování mračna bodů a opravu mapy (skluz) při vysokých frekvencích. Bude to tedy chtít, aby všechny tyto procesy běžely paralelně. Proto je pro tuto problematiku nutno využit Nvidia Jetson, který je dostatečně výkonný.

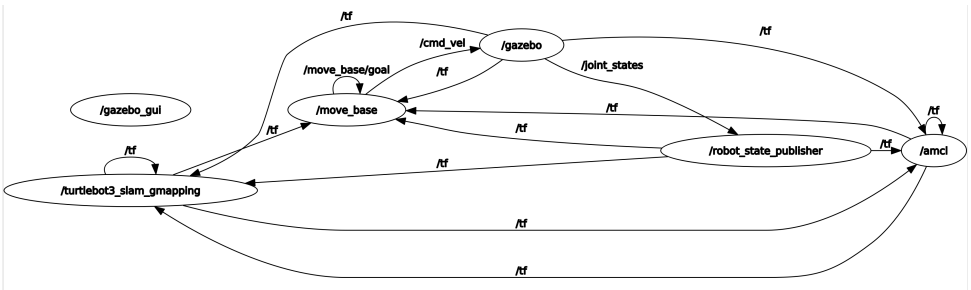
Dále jsem tedy vytvořil spouštěcí program s algoritmem AMCL (Adaptivní Monte Carlo Lokalizace), který mi umožní určovat polohu robota v reálném čase s vysokou přesností. Na závěr chci, aby se robot pohyboval v neznámém prostředí a vytvářel si sám mapu v průběhu navigace a nebyla tedy potřeba předem vytvořená mapa. Toho jsem docílil změněním globální hodnotové mapy ze statické na dynamickou a dal mu prázdnou mapu místo předem

vytvořené. Simulace je znázorněna na obrázku 4.12.



Obrázek 4.12: Simulace s ROS1

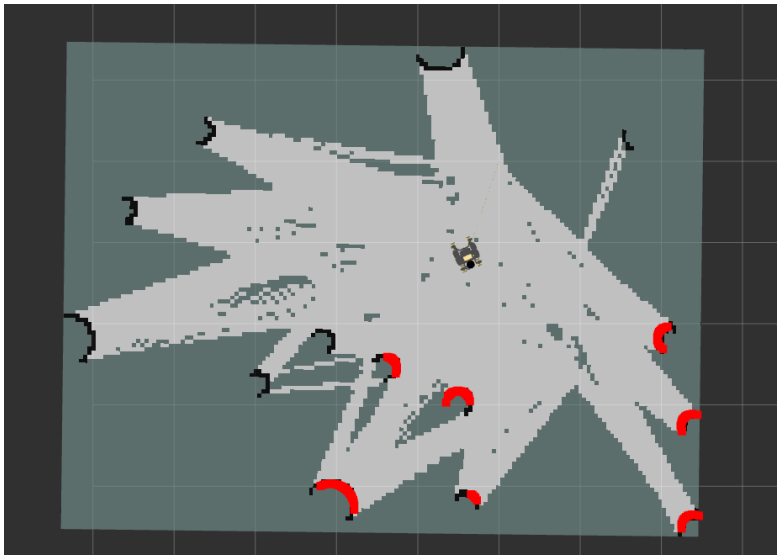
Výsledné schéma je vidět na obrázku 4.13, kde je grafická vizualizace Nodů a jejich propojení mezi nimi. Toto zobrazení neobsahuje Topicy, protože by toto zobrazení bylo příliš veliké a na stránku nepřehledné.



Obrázek 4.13: Schéma výsledné simulace

V tuto chvíli nadešel čas, abych vyměnil model TurtleBot, za model Exomy a otestoval můj program na tomto modelu. Po hlubším průzkumu, jsem došel ke zjištění, že model Exomy je vytvořen pouze pro ROS2. ROS2 je novější verzí ROS a snaží se řešit limity a nespolehlivost, které měla předchozí verze. ROS2 používá novou architekturu, která umožňuje lepší propojování modulů a komponent v systému. Vzhledem k nedostatečné znalosti tvorby komplexnějších modelů v tuto chvíli a zjištění, že ROS2 je pro mne lépe pochopitelný, lépe ovladatelný a obsahuje více balíčků, pro tvorbu autonomního mobilního robotu, jsem se rozhodl přejít z verze ROS1 Noetic na verzi ROS2 Foxy.

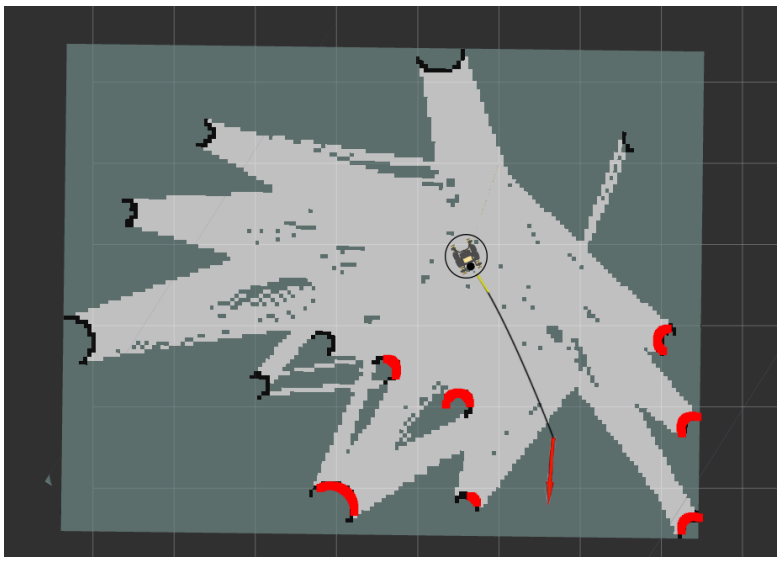
Nová verze ROS má jinou architekturu a tedy i jiný způsob psaní kódu. Nemohu přímo použít kód, který jsem již vytvořil v ROS1. Začal jsem tedy vytvářet od začátku v ROS2 rovnou pro nově získaný model Exomy. Nejdříve jsem začal upravovat URDF soubor, který představuje model Exomy, tak aby souhlasil s reálným upraveným modelem. Pro prvotní jednoduchost jsem použil místo senzoru RealSense senzor Lidar. Hlavní důvod bylo větší množství dostupných informací a návodů k použití s ROS. Základní balíček Exomy je připraven pro mobilní zařízení s virtuálním joystickem. Upravil jsem tento balíček tak, aby byl robot ovládán pomocí klávesnice. Tím jsem zároveň docílil toho, že je zároveň schopen přijímat příkazy od navigačních balíků, kterým by předtím nerozuměl. Následně jsem přidal navigační balíček Nav2. Z něj jsem nejdříve použil asynchronní slam node (simultánní lokalizace a mapování). Po doladění konfigurace a nastavení automatické určování prvotní polohy jsem provedl úspěšný test viz. obrázek 4.14.



Obrázek 4.14: Simulace s ROS2

Po úspěšném provedení testu jsem rozšířil program o navigační část z navigačního balíčku Nav2. Zde se znovu musela nastavit správná konfigu-

race, tak aby se robot pohyboval správnou rychlostí a našel nejoptimálnější trasu adekvátní k počtu výpočtů možných tras. Po správném nastavení a úspěšném testu přišel čas, abych odebral senzor Lidar a zakomponoval senzor RealSense. Pro realizaci bylo potřeba upravit URDF model (primárně gazebo plugin) a přidat RealSense balíček. Po průzkumu, jak propojit hloubkovou kameru RealSense jsem došel k závěru, že nejideálnější bude použít balíček depth_to_laserscan, který si z hloubkové mapy vezme pás bodů v nastavené výšce a šířce a vytvoří z nich mapu 2D bodů, stejně jako ze senzoru Lidar. Tím jsem dosáhl kompatibility s navigačním balíčkem Nav2. Na obrázku 4.15 je znázorněna simulace s naplánovanou cestou.



Obrázek 4.15: Finální simulace s ROS2

4.4 Aplikace simulace na reálného robota

Po úspěšném vyladění robota v simulaci nastává čas na implementaci do skutečného prostředí na reálného robota. Pokud bude robot úspěšně implementován, bude navigace v uzavřených prostorách a připravena k využití pro další práce a různé úkoly v uzavřených prostorách.

Docker je softwarová platforma, která vám umožňuje rychle vytvářet, testovat a nasazovat aplikace. Docker balí software do standardizovaných jednotek nazývaných kontejnery, které mají vše, co software potřebuje ke spuštění, včetně knihoven, systémových nástrojů a kódu. Pomocí Dockeru je možno rychle nasadit a škálovat aplikace do jakéhokoli prostředí a vědět, že váš kód poběží. Spuštění Dockeru poskytuje vývojářům a správcům vysoce spolehlivý

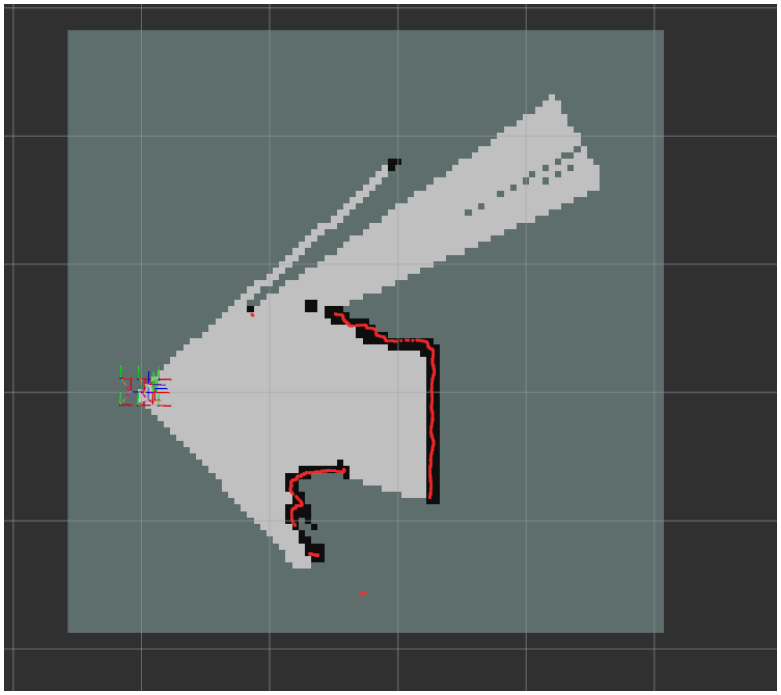
a levný způsob vytváření, dodávání a spouštění distribuovaných aplikací v jakémkoli měřítku.

Docker funguje tak, že poskytuje standardní způsob, jak spustit váš kód. Docker je operační systém pro kontejnery. Podobně jako virtuální stroj virtualizuje (odstraňuje nutnost přímé správy) serverový hardware, kontejnery virtualizují operační systém serveru. Docker je nainstalován na každém serveru a poskytuje jednoduché příkazy, které můžete použít k sestavení, spuštění nebo zastavení kontejnerů.

Z důvodů kompatibility byl na Jetson Tegra nahrán Ubuntu 16.04 a protože je to starší verze, bylo možné na ni nahrát pouze starší verzi ROSu a to verzi Dashing. Pokusil jsem se tedy nejdříve provést implementaci na tuto kombinaci. Po nastavení konfigurace jednotlivých kol upravené šestikolové platformy Exomy, jsem byl schopen rozpohybovat robota. Další častí úkolu bylo nainstalovat balíček RealSense. Ten se nepodařilo nainstalovat z důvodu nekompatibility a proto jsem vytvořil Docker (kontejner) s ROS Foxy. V Dockeru se mi podařilo nahrát a nainstalovat všechny potřebné balíčky a knihovny ze simulace. Konfiguraci kol robotu jsem měl hotovou a test v Dockeru proběhl úspěšně. Dále jsem odebral z URDF modelu Gazebo plugin. Simulaci na reálném robotu nepotřebuji a mohla by způsobovat nechtěné chybové hlášení programu.

Abych mohl robota lokalizovat v prostředí je potřeba mít odometrii. Tu jsem původně v simulaci získával z informace o natočení kol ve spojení s vizuální odometrií (vizuální senzor Lidar nebo RealSense). Informace o natočení kol v reálném světě je nedostatečně spolehlivá, z důvodu prokluzu kol. Proto jsem se rozhodl použít primárně vizuální odometrii a pro dodatečné zlepšení lokalizace použít IMU (inerciální měřící jednotku), které má v sobě RealSense zabudováno. Potřeboval jsem tedy získat balík, který vytvoří z dat RealSensu vizuální odometrii a balík, který spojí odometrie, pro lepší a robustnější lokalizaci. Pro vizuální odometrii jsem zvolil rf2o_laser_odometry, který publikuje odhady odometrie z Lidaru a poté vypočítá základní odometrii robota pomocí transformací. Tento balík mohu použít i na svůj senzor RealSense, protože jsem schopen převést hloubkovou mapu na formát ve stylu laseru ze senzoru Lidar. K němu jsem potřeboval balíček, který spojí IMU a vizuální odometrii. Ten jsem nemohl nikde získat, tak abych byl schopen ho zprovoznit v mojí kombinaci. Největší nadějí byl balíček, který využívá metody rozšířeného Kalmanova filtru a je hodně používán. Mě osobně se ho nepodařilo zprovoznit, ale při hledání řešení jsem našel balíček rtabmap. Je schopen vytvořit vlastní vizuální odometrii přímo z hloubkové mapy a zároveň ji spojit s IMU. Tím jsem získal odometrii, která skončila úspěšným testem a je připravena pro použití k lokalizaci.

Když mám odometrii, mohu řešit lokalizaci, konkrétně chci získat SLAM (simultánní lokalizaci a mapování). Měl jsem v úmyslu použít balíček Nav2, který toto umí a zároveň je schopen navigace, ale při implementaci jsem narazil na překážky, které jsem nebyl schopen vyřešit. Zároveň jsem při hledání řešení, řádně prostudoval balíček rtabmap a zjistil, že je také schopen vytvořit SLAM. Výsledný test je vidět na obrázku 4.16.



Obrázek 4.16: Mapa z robotu

Na obrázku 4.16 je mapa, kde šedě je označena zem, černě překážka a červeně jsou označeny aktuální data ze senzoru.

Dále je zde problematika simultánní lokalizace a mapování, kterou musím brát na vědomí. Mapování dynamického prostředí je proces získávání informací o okolním prostředí a jeho stavu v reálném čase. Tato problematika se často vyskytuje v oblastech jako autonomní řízení, počítačové vidění a dalších aplikacích, které vyžadují přesné znalosti o okolním prostředí pro správné rozhodování a akce. Existuje několik významných problémů spojených s mapováním dynamického prostředí jako je dynamika prostředí, detekce a identifikace objektů, sledování objektů, aktualizace map, plánování a reakce.

- Dynamika prostředí: Prostředí, ve kterém se pohybujeme, je často dynamické a neustále se mění. Objekty se pohybují, překážky se objevují a

mizí. Mapovací algoritmy musí být schopny detekovat a sledovat tyto změny v reálném čase a aktualizovat mapy.

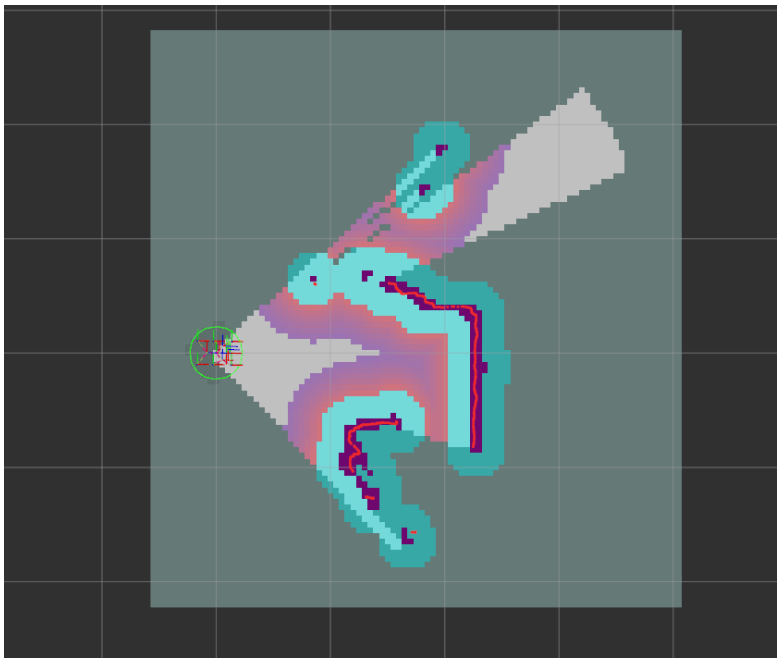
- Detekce a identifikace objektů: Pro úspěšné mapování je nezbytné detekovat a identifikovat objekty v prostředí. To může zahrnovat vozidla, pracovníky, překážky a další prvky, které mohou ovlivnit plánování a navigaci. Detekce objektů v dynamickém prostředí je obtížná úloha kvůli proměnlivým podmínkám osvětlení, různorodosti objektů a překážek a dalším faktorům.
- Sledování objektů: Proces sledování objektů v čase je klíčovým prvkem při mapování dynamického prostředí. Sledování umožňuje identifikovat trajektorie objektů, jejich rychlosti a předpovědět jejich budoucí pohyb. Sledování je náročné kvůli nejistotě ve vstupních datech (např. šum, nedostatek informací) a také kvůli potřebě zachovat konzistenci sledovaných objektů mezi snímky.
- Aktualizace map: Vzhledem k dynamice prostředí je důležité aktualizovat mapy v reálném čase, aby byly stále relevantní a přesné. To vyžaduje správnou integraci nových informací získaných ze senzorů (např. kamery, lidary, radaru) do stávajících map a také správné rozlišení mezi novými a stávajícími objekty.
- Plánování a reakce: Na základě dynamického mapování musí systém rozhodovat a plánovat akce v reálném čase. Musí brát v úvahu překážky, ostatní účastníky provozu, cíle a další faktory. Efektivní plánování v dynamickém prostředí vyžaduje rychlé reakce na změny a schopnost přizpůsobit se novým podmínkám.

Z těchto problémů je pro mě důležitá dynamika prostředí a aktualizace map. Ty jsou pro mě vyřešeny v balíčku Rtab-map. Problémy s mapováním dynamického prostředí zahrnují pokročilé algoritmy pro detekci objektů, sledování, online mapování, průzkum prostředí, plánování a reakce.

Problém, který jsem řešil u SLAM se vyskytl i při nastavování konfigurace navigace. Způsoboval ho balíček depthimage_to_laserscan, který není již potřeba, protože odometrie je vytvářena přímo z hloubkové mapy a mapa prostoru nyní také. Pro navigaci sice byl potřeba také, ale to jsem vyřešil upravením konfigurace SLAM nodu, kde jsem přidal generování scanu. Při provádění testu, který je vidět na obrázku 4.17, jsem získal globální hodnotovou mapu. Zbarvené místa označují překážky.

Lokální a globální hodnotová mapa
Hodnotové mapy slouží k ukládání informací o překážkách v prostoru. Používám dvě hodnotové mapy k ukládání informací o překážkách v prostoru. Jedna

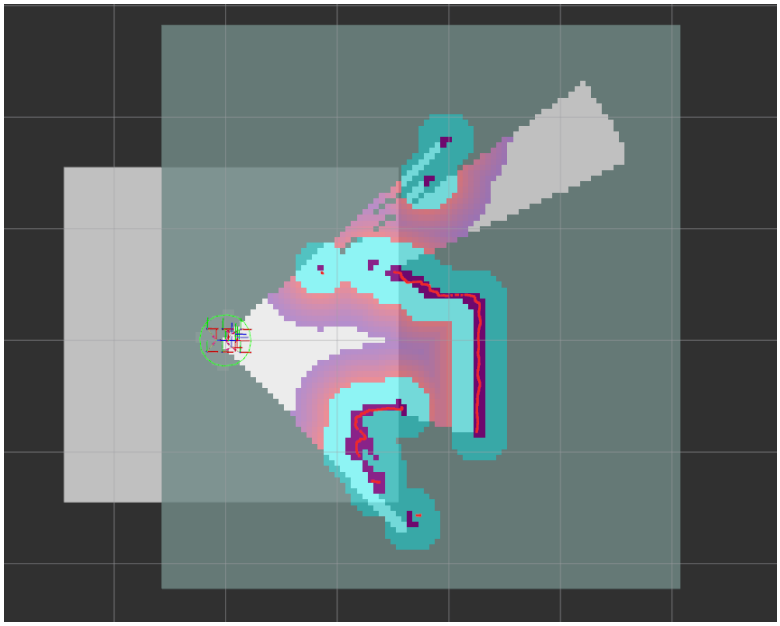
hodnotová mapa (globální) se používá pro globální plánování, což znamená vytváření dlouhodobých plánů pro celé prostředí, a druhá lokální hodnotová mapa se používá pro lokální plánování a vyhýbání se překážkám. Existuje mnoho různých způsobů, jak nakonfigurovat dvě hodnotové mapy. Já jsem je uspořádal tak, že globální hodnotová mapa je nakonfigurována tak, aby používala mapu, zatímco místní plánovač pracuje v odometrickém rámci. Abych to měl správně, potřebuji nasměrovat hodnotové mapy na senzor, u kterých by měly odebírat aktualizace. U globální hodnotové mapy je důležité nastavení parametru "static_map"na hodnotu true. To znamená, že pro navigaci použijete externí zdroj mapy. Tato mapa může pocházet ze SLAMu nebo může pocházet ze zdroje, jako je map_server.



Obrázek 4.17: Globální hodnotová mapa z robotu

Navigační balíček po vygenerování hodnotových map čeká na zadání cílového bodu, aby mohl vytvořit plán navigace. Při zadání cílového bodu v softwaru Rviz se při simulaci bod nastavil a navigační balíček začal s navigací, ale teď se nic nestalo. Po nějakém čase průzkumu jsem došel k zjištění, že je nevhodně nastavená konfigurace softwaru Rviz a informaci o cílovém bodu posílá na jiný Topic než je poslouchán od Navigačního balíku. Po opravení konfigurace jsem byl schopen posílat požadovaný cílový bod. Po zadání cílového bodu se robot nerozpohyboval a objevilo nové chybové hlášení. Nejdříve jsem upravil konfiguraci ovládacího prvku robotu v navigačním balíčku na správné rychlostní hodnoty. Přestože tato úprava konfigurace byla zapotřebí, nevyřešila danou chybu. Následně bylo zjištěno, že chyba byla způsobována nekompatibilitou mezi ROS Foxy a Nav2. Jediné řešení bylo přejít na novější

verzi ROS Galactic. Získáním nové verze ROS a úspěšným nainstalováním všech balíčku a knihoven, které jsem doposud použil, jsem eliminoval toto chybové hlášení. V následujícím testu se objevilo jiné chybové hlášení, že není schopen naplánovat cestu navigace, která vy splňovala všechna kritéria. Po snaze upravit konfiguraci, aby některá z cest splnila kritéria, jsem po hlubším prozkoumání vyzkoušel zpětně zjistit co všechno mi funguje a zjistil jsem, že dostávám prázdnou lokální hodnotovou mapu.



Obrázek 4.18: Mapa s prázdnou lokální hodnotovou mapou

Lokální hodnotová mapa se generuje z dat ze senzorů, zde ji vytváří navigační balíček Nav2 z Topicu /scan, dat ve formě laserových (Lidarových) dat. Tyto data má vytvářet balík rtabmap. Ten po zjištění sice vytváří tuto mapu, ale vytváří ji prázdnou. Vrátil jsem zpátky tedy balíček depthimage_to_laserscan, který mi do Topicu /scan potřebné data nahraje. I když tento balík v předchozí implementaci způsoboval problémy, teď se problémy neprojevily, protože nyní byla použita jiná kombinace balíku (jiný SLAM), než tomu bylo předtím. Dostávám data do Topicu /scan, ale lokální hodnotová mapa se stále negeneruje a je pouze prázdná. V pozdějším prozkoumání se došlo k zjištění, že z laserových dat se počítala jak mapa, tak i globální a lokální hodnotová mapa. Tento přístup se ukázal být příčinou a po převedení globální i lokální hodnotové mapy na statické mapy. Dále nastal problém s plánovačem cest. Plánovač, který hledá možné cesty nenašel jedinou, která by splňovala všechny kritéria. Tato příčina byla způsobována špatnou konverzí mezi navigačním balíčkem a přepočtem řídicí části na PWM pro motory. U navigačního balíčku je totiž úhel natočení dán požadovanou rychlostí rotace kolem osy Z, ale u řídicí části rover_node tento úhel bere jako úhel natočení

joysticku, na který byl projekt Exomy vytvořen. Bylo tedy zapotřebí upravit Rover_node následovně.

Nejdříve bylo zapotřebí získat maximální možnou rychlost robota. Tu jsem získal z následujících rovnic:

$$s = 2 \cdot \pi \cdot r_k = 2 \cdot \pi \cdot 0.05 = 0.3142m \tag{4.1}$$

kde s je délka dráhy kola za jedno otočení a r_k je poloměr kola, rychlost motoru je 0.2s/60° a potom maximální rychlost je:

$$v_{max} = \frac{s}{0.2 \cdot \frac{360}{60}} = 0.26\frac{m}{s} \tag{4.2}$$

Následně je potřeba znát rychlost na kruhové dráze, která je dána vztahem

$$v = r\acute{a}dius \cdot \omega \tag{4.3}$$

kde v je rychlost, *rádius* je poloměr oblouku a ω je úhlová rychlost. Z této rovnice získám rádius (poloměr oblouku)

$$r\acute{a}dius = \frac{v_x}{\omega} \tag{4.4}$$

Pro řízení typu Ackermann potřebuji úhel natočení kol, které je dáno vztahem

$$\Theta = \arctan(\frac{r_k}{r\acute{a}dius}) \tag{4.5}$$

Rychlost natočeného kola je dána vztahem

$$v = \frac{v_x}{R} \tag{4.6}$$

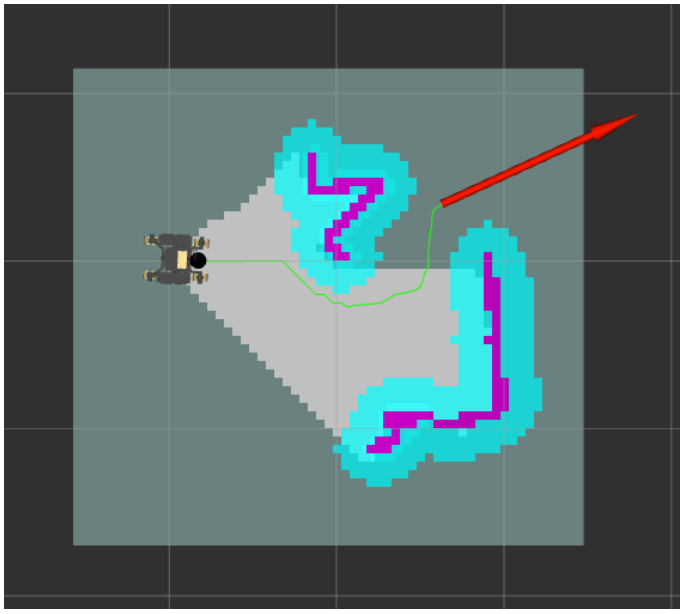
kde R je poloměr oblouku vnitřního nebo vnějšího kola. Ten získám vztahem

$$R = r\acute{a}dius \pm \frac{1}{2} \cdot \omega \tag{4.7}$$

kde pro vnitřní kolo se úhlová rychlost ω odečítá a pro vnější kolo přičítá. Dále potřebuji vyjádřit rychlost v procentech a tu získám následovně

$$v = \frac{v_x}{v_{max}} \cdot 100[\%] \tag{4.8}$$

Výsledkem a výstupem z rover_node je úhel natočení a rychlost v procentech pro PWM jednotlivých kol. Po této úpravě navigační balíček je schopen najít cestu do cílového bodu, která splňuje všechny podmínky a výsledek je na obrázku 4.19.



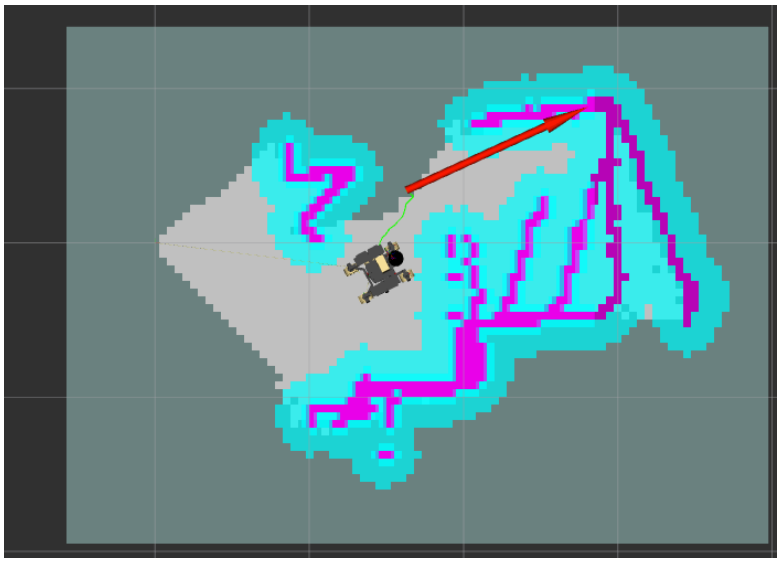
Obrázek 4.19: Naplánovaná cesta pro robota

Další problematikou je, že robot nebyl schopen dojet do cílového bodu. Důvodem bylo, že plánovač se snaží jet pomalu, ale motory se při tak pomalých rychlostech nerozhýbou. Při zadání minimální povolené lineární rychlosti do konfigurace se tento problém vyřešil, ale způsobil jiný. Teď není schopen se otočit na místě, protože má nastavenou nejnižší možnou hodnotu lineární rychlosti větší než 0.

Obrázek 4.20: Robot se pohybuje po naplánované cestě k cíli

I když robot není schopen otočení na místě, je schopen dojet do cílového

bodu.



Obrázek 4.21: Robot

Po těchto všech úpravách je navigační balíček schopen najít cestu do cílového bodu, která splňuje všechny podmínky a dojede do zadaného bodu. Dále tu jsou další problémy, které by bylo dobré eliminovat a pomohlo by to optimalizovat kvalitu navigace. Osobně z časových důvodů, jsem nebyl schopen je uskutečnit.

Robot by byl schopen jet maximální rychlostí 0,26m/s, ale musí být nastaveno omezení rychlosti, protože při příliš rychlých pohybech není schopna odometrie stíhat zaznamenávat všechny změny a ztratí svou polohu na mapě. Tento problém by se dal vyřešit přidáním IMU a spojit tyto dvě odometrie.

Kapitola 5

Závěr

V této diplomové práci jsem se zabýval návrhem a realizací autonomního mobilního robota, který po zadání cílového bodu má najít cestu skrz překážky v uzavřených prostorách a dostat se co cílového bodu.

Provedl jsem rešerši aktuálně dostupných senzorů vhodných pro orientaci robota v prostoru a jejich principy snímání. Z daných možností jsem skončil u výběru senzoru Lidar a RealSense, které jsem následně simulačně testoval. Provedl jsem rešerši metod autonomizace, plánování a navigace vhodných pro využití a pro pohyb v uzavřeném prostoru. V této rešerši jsem popsal úrovně autonomity, pohybové systémy, percepci, způsoby lokalizace, odometrie, plánování, navigaci a popsal jsem některé plánovací algoritmy. Dále jsem navrhl a realizoval úpravu robotického šestikolové platformy Exomy tak, aby vyhovovala požadavkům na umístění zvoleného hardwaru.

Navrhl jsem řešení autonomizace pohybu za použití OS ROS pro mikropočítač Nvidia Jetson. I přes velké problémy s kompatibilitou jsem navržené softwarové řešení realizoval a úspěšně simulačně ověřil. V průběhu realizace jsem testoval použití senzoru Lidar, které je více užívané a aplikované pro tyto úlohy. V porovnání Lidaru se senzorem RealSense má každý z nich své výhody a nevýhody. RealSense má omezený zorný úhel na rozdíl od Lidaru, který ho má 360°. Na druhou stranu RealSense snímá větší pás prostoru, tudíž je možnost vytvářet 3D mapu a je lépe cenově dostupný, kdežto Lidar snímá pouze úzký pruh v dané výšce. Závěrem nejsem proto schopen označit jeden z nich, jako jednoznačně lepší pro tuto úlohu a každý musí usoudit sám, podle jeho požadavků, co vybrat. Můj osobní výběr pro implementaci na reálného robota byl senzor RealSense.

Následně jsem toto navržené řešení implementoval na upravenou šestikolovou robotickou platformu. V průběhu implementace nastaly četné problémy s kompatibilitou, jak mezi hardwarem a softwarem, tak i mezi jednotlivými softwary a jejich balíčky a knihovny. Když si vezmeme k porovnání robot Turtlebot. Ten má kombinaci hardwaru a softwarového frameworku před připraven pro snadné použití a dokumentaci, které mu pomáhají rychle začít. Na rozdíl od Turtlebotu má kombinace řešení je flexibilnější. Vlastní kombinace hardwaru a softwaru mi umožňuje navrhnout a přizpůsobit robota podle vlastních požadavků. Další výhodou je, že Nvidia Jetson je výkonnější výpočetní modul, který mi poskytuje vysoký výkon pro zpracování dat a algoritmů v reálném čase. Na druhou stranu, jsou zde četné problémy s kompatibilitou dílčích částí, které vývojáři Turtlebot vyřešili za nás.

Abych celou implementaci na reálného robota shrnul. Robot je schopen pohybu, umí vytvořit odometrii, mapu prostředí a na základě toho následně se lokalizovat v tomto prostředí v reálném čase. V navigační části vygeneruje lokální i globální hodnotovou mapu a po nastavení správných přepočtů mezi navigačním balíčkem a výsledným pohybem robota je plánovač schopen najít cestu do zadaného cílového bodu. Po upravení rychlostních nastavení je robot schopen navigace a dosažení cíle.

Robot při příliš rychlých pohybech ztrácí polohu, kvůli vizuální odometrii, která nestíhá tyto rychlé pohyby dostatečně rychle zpracovat. Dále by bylo proto dobré přidat IMU, které by zpřesnilo odometrii a následnou lokalizaci.

Literatura

[1] Autonomous Mobile Robot Mechanical Design — researchportal.vub.be. https://researchportal.vub.be/nl/studentTheses/autonomous-mobile-robot-mechanical-design. [Accessed 15-Apr-2023].

[2] Depth Measurement Based on Infrared Coded Structured Light — hindawi.com. https://www.hindawi.com/journals/js/2014/852621/. [Accessed 16-Apr-2023].

[3] Introducing the intel® RealSense™ depth camera d455.

[4] Levels of autonomy for field robots.

[5] SAE international releases updated visual chart for its "levels of driving automation" standard for self-driving vehicles.

[6] Setting up odometry — navigation 2 1.0.0 documentation.

[7] What is an inertial measurement unit?

[8] Mary B. Alatise and Gerhard P. Hancke. A review on challenges of autonomous mobile robot and sensor fusion methods. 8:39830–39846. Conference Name: IEEE Access.

[9] Amad-ud-Din, I. A. Halin, and S. B. Shafie. A review on solid state time of flight TOF range image sensors. In *2009 IEEE Student Conference on Research and Development (SCOReD)*, pages 246–249.

[10] Mohammad O. A. Aqel, Mohammad H. Marhaban, M. Iqbal Saripan, and Napsiah Bt. Ismail. Review of visual odometry: types, approaches, challenges, and applications. 5(1):1897.

[11] A. Carullo and M. Parvis. An ultrasonic sensor for distance measurement in automotive applications. 1(2):143.

[12] Sheng Chang, Hong-guo Xu, and Hong-fei Liu. Simulation on steering stability of 4ws tractor semi-trailer. In *2009 International Conference on Measuring Technology and Mechatronics Automation*, volume 2, pages 355–358. ISSN: 2157-1481.

[13] E. DANDIL and K. K. ÇEVİK. Computer vision based distance measurement system using stereo camera view. In *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pages 1–4.

[14] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. 23(1):34–46. Conference Name: IEEE Transactions on Robotics.

[15] Shuang Guo, Jifeng Guo, and Chengchao Bai. Semi-direct visual odometry based on monocular depth estimation. In *2019 IEEE International Conference on Unmanned Systems (ICUS)*, pages 720–724.

[16] Y. S. Huang, Y. P. Huang, M. S. Young, and Ke-Nung Huang. The assessment system of human visual spectral sensitivity curve by frequency modulated light. In *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pages 263–265. ISSN: 1558-4615.

[17] Maged Ismail and Ezzeldin Abdelkawy. A hybrid error modeling for MEMS IMU in integrated GPS/INS navigation system. 16(1):6.

[18] Javier Javier Moreno, E. Clotet, Ruben Lupiañez, Marcel Tresanchez, Dani Martinez, Tomàs Pallejà, Jordi Casanovas, and Jordi Palacín. Design, implementation and validation of the three-wheel holonomic motion system of the assistant personal robot (APR). 16:1658.

[19] Tao Jiang. *Stereo Vision for Facet Type Cameras*. Logos Verlag Berlin GmbH. Google-Books-ID: kXIYDQAAQBAJ.

[20] Nak Yong Ko and Tae Gyun Kim. Comparison of kalman filter and particle filter used for localization of an underwater vehicle. In *2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 350–352.

[21] L. Koval, J. Vaňuš, and P. Bilík. Distance measuring by ultrasonic sensor. 49(25):153–158.

[22] M. Kurc, K. Wegner, and M. Domański. Transformation of depth maps produced by ToF cameras. In *2014 International Conference on Signals and Electronic Systems (ICSES)*, pages 1–4.

[23] Murat Köseoğlu, Orkan Murat Çelik, and Ömer Pektaş. Design of an autonomous mobile robot based on ROS. In *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, pages 1–5.

[24] Xun Li, Yingbin Feng, Ronghui Huang, Xin Zhang, Shungui Liu, and Jingwen Ai. The application of square-root cubature kalman filter in SLAM for underwater robot. In *2017 Chinese Automation Congress (CAC)*, pages 2183–2187.

[25] Tommy Ertbølle Madsen and Hans Lavdal Jakobsen. Master thesis project january 31st 200.

[26] Ruslan Masinjila and Pierre Payeur. Consistent multirobot localization using heuristically tuned extended kalman filter. In *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, pages 297–303.

[27] Adelardo A. D. Medeiros. A survey of control architectures for autonomous mobile robots. 4:35–43. Publisher: Sociedade Brasileira de Computação.

[28] Schoenbein Miriam. *Omnidirectional Stereo Vision for Autonomous Vehicles*. KIT Scientific Publishing. Google-Books-ID: c4PwCAAAQBAJ.

[29] D Moldovanu, A Csato, and N Bagameri. Study regarding the implementation of an ackerman steering geometry in MATLAB. 568(1):012092.

[30] Jongsik Moon and Byung-Yoon Lee. 2d lidar enhanced direct sparse odometry for scale recovery. In *2020 20th International Conference on Control, Automation and Systems (ICCAS)*, pages 453–456. ISSN: 2642-3901.

[31] J. P. Queralta, F. Yuhong, L. Salomaa, L. Qingqing, T. N. Gia, Z. Zou, H. Tenhunen, and T. Westerlund. FPGA-based architecture for a low-cost 3d lidar design and implementation from multiple rotating 2d lidars with ROS. In *2019 IEEE SENSORS*, pages 1–4. ISSN: 2168-9229.

[32] Sławomir Romaniuk, Adam Wolniakowski, Adam Pawłowski, and Cezary Kownacki. Adaptation of ultra wide band positioning system for adaptive monte carlo localization. In *2022 26th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 238–243.

[33] N. Sariff and N. Buniyamin. An overview of autonomous mobile robot path planning algorithms. In *2006 4th Student Conference on Research and Development*, pages 183–188.

[34] Ksenia Shabalina, Artur Sagitov, and Evgeni Magid. Comparative analysis of mobile robot wheels design. In *2018 11th International Conference on Developments in eSystems Engineering (DeSE)*, pages 175–179. ISSN: 2161-1351.

[35] C. Shi, J. Feng, S. Tang, and Z. Song. A robust feature detection method for an infrared single-shot structured light system. In *2018*

IEEE International Conference on Real-time Computing and Robotics (RCAR), pages 693–697.

[36] Yefeng Sun. A comparative study on the monte carlo localization and the odometry localization. In *2022 IEEE International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA)*, pages 1074–1077.

[37] José Ricardo Sánchez-Ibáñez, Carlos J. Pérez-del Pulgar, and Alfonso García-Cerezo. Path planning for autonomous mobile robots: A review. 21(23):7898. Number: 23 Publisher: Multidisciplinary Digital Publishing Institute.

[38] R. Torun, M. M. Bayer, I. U. Zaman, J. E. Velazco, and O. Boyraz. Realization of multitone continuous wave lidar. 11(4):1–10. Conference Name: IEEE Photonics Journal.

[39] Miro Voellmy and Maximilian Ehrhardt. Exomy: A low cost 3d printed rover. 10 2020.

[40] Alejandro J. Weinstein and Kevin L. Moore. Pose estimation of ackerman steering vehicles for outdoors autonomous navigation. In *2010 IEEE International Conference on Industrial Technology*, pages 579–584.

[41] F. Wermke and B. Meffert. Interference model of two time-of-flight cameras. In *2019 IEEE SENSORS*, pages 1–4. ISSN: 2168-9229.

[42] Junjun Xu, Bo Liu, Kangjie Li, Yixiong Feng, Hao Zheng, and Yicong Gao. Design and structure analysis of multi-legged bionic soft robot. In *2020 International Conference on Advanced Mechatronic Systems (ICAMechS)*, pages 180–185. ISSN: 2325-0690.

[43] Zewen Xu, Zheng Rong, and Yihong Wu. A survey: which features are required for dynamic visual simultaneous localization and mapping? 4(1):20.

[44] Yunxiang Ye, Long He, and Qin Zhang. Steering control strategies for a four-wheel-independent-steering bin managing robot. 49(16):39–44.

[45] Xuexi Zhang, Jiajun Lai, Dongliang Xu, Huaijun Li, and Minyue Fu. 2d lidar-based SLAM and path planning for indoor rescue using mobile robots. 2020:1–14.

[46] Lei Zhao, Zhun Fan, Wenji Li, Honghui Xie, and Yang Xiao. 3d indoor map building with monte carlo localization in 2d map. In *2016 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII)*, pages 236–240.

[47] V A Zhmud, N O Kondratiev, K A Kuznetsov, V G Trubin, and L V Dimitrov. Application of ultrasonic sensor for measuring distances in robotics. 1015:032189.