VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

NÁVRH A REALIZACE ELEKTRONIKY A SOFTWARE AUTONOMNÍHO MOBILNÍHO ROBOTU

ELECTRONICS CIRCUIT BOARD AND CONTROL SOFTWARE DESIGN FOR AUTONOMOUS MOBILE ROBOT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN MEINDL

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. STANISLAV VĚCHET, Ph.D.

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Jan Meindl**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**doc. Ing. Stanislav Věchet, Ph.D.**
Akademický rok:	2016/17

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Návrh a realizace elektroniky a software autonomního mobilního robotu

Stručná charakteristika problematiky úkolu:

Podstatou práce je navrhnout a realizovat elektroniku a obslužný software pro autonomní mobilní robot, jehož mechanická konstrukce je k dispozici. Elektronika je představována řadou modulů pro zpracování senzorických dat a řízení kontrolérů motorů. Obslužný software je založen na frameworku ROS (Robotic Operating System). Součástí práce je implementace rozhraní mezi moduly elektroniky a ROSem. Pro ověření funkčnosti navrženého řešení je nutné v rámci práce implementovat také grafické rozhraní a nakonfigurovat navigační rutiny ROSu.

Cíle diplomové práce:

1) Seznamte se s frameworkem ROS
2) Navrhněte a realizujte následující elektronické moduly: modul pro odečítání odometrie, modul pro řízení kontrolérů motorů, modul pro zpracování dat z ultrazvukových senzorů, modul pro zpracování dat z pneumatických nárazníků
3) Implementujte rozhraní mezi výše uvedenými moduly a ROSem
4) Navrhněte a implementujte grafické rozhraní pro základní funkcionalitu robotu
5) Nakonfigurujte a ověřte funkčnost navigačních rutin ROSu

Seznam doporučené literatury:

THRUN Sebastian, BURGARD Wolfram, and FOX Dieter . Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series). Intelligent robotics and autonomous agents. The MIT Press, August 2005.

CHOSET Howie, LYNCH Kevin M., HUTCHINSON Seth, KANTOR George, BURGARD Wolfram, KAVRAKI Lydia E. , and THRUN Sebastian. Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents). The MIT Press, June 2005.

ROS.org. ROS.org | Powering the world's robots. [online]. 2.11.2016 [cit. 2016-11-02]. Dostupné z: http://www.ros.org/.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2016/17

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt
Diplomová práce se zabývá návrhem a realizací vestavěného řídicího systému a navigačního softwaru autonomního mobilního robotu DACEP. Rešeršní část se věnuje výběru senzorického vybavení. V práci je popsán návrh vestavěného řídicího systému a komunikačního rozhraní mezi tímto systémem a nadřazeným počítačem. Součástí práce je vytvoření lokalizačního a navigačního softwaru, ke kterému je použit framework ROS. Důraz je kladen na to, aby tato část byla co nejvíce návodná pro pomoc pří vývoji robotu podobné konstrukce. V rámci diplomové práce bylo vytvořeno grafické rozhraní, které slouží pro diagnostiku robotu a jeho vzdálené řízení.

Summary
The master's thesis deals with the design and realization of embedded control system and software of the autonomous mobile robot DACEP. The research section focuses on the selection of sensory equipment. Moreover, the design of the embedded control system and the communication interface between this system and the master PC is described in detail, followed by the design of localization and navigation software that uses ROS framework. The section is written as instructive as possible for the development of robots of similar construction. Finally the development of a graphical interface for robot diagnostics and remote control is depicted.

Klíčová slova
Řídicí vestavěný systém, autonomni vozidlo, LIDAR, ROS, DACEP, Mbed.

Keywords
Control embedded system, autonomous driving, LIDAR, ROS, DACEP, Mbed.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně dle pokynů vedoucího a s použitím uvedené odborné literatury.

Bc. Jan Meindl

MEINDL, J.*Návrh a realizace elektroniky a software autonomního mobilního robotu*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2017. 76 s. Vedoucí diplomové práce doc. Ing. Stanislav Věchet, Ph.D..

Děkuji svému vedoucímu doc. Ing. Stanislavu Věchetovi, Ph.D. a spolupracovníkům z Bender Robotics s.r.o. za cenné rady při tvorbě této diplomové práce. Děkuji své rodině a přítelkyni za trpělivost a podporu.

Bc. Jan Meindl

Obsah

1. Úvod

Letos je to půl století od doby, kdy spoluzakladatel firmy Intel Gordon Moore publikoval článek, ve kterém předpověděl, že „počet tranzistorů, které mohou být umístěny na integrovaný obvod, se při zachování stejné ceny zhruba každých osmnáct měsíců zdvojnásobí“ [1], tedy že složitost integrovaných obvodů bude růst exponenciálně. Toto pravidlo se ukázalo zcela správné a platné dodnes. Interval zdvojnásobení se jen postupně posunul z osmnácti na dvacet čtyři měsíců spolu s tím, jak se technologie musí mimo jiné čím dál více potýkat s fyzikálními limity křemíku, ze kterého jsou integrované obvody vyráběny.

Díky tomuto stále dostupnějšímu výpočetnímu výkonu se mění zvyklosti a pohlížení na techniku. Ze sálových počítačů se výpočty přesunuly na stolní počítače, z nich na notebooky a nyní částečně na mobilní telefony.

Ovšem sálové počítače zůstaly. V digitálním světě vzniká stále více dat, která musí být někde uložena. Jedná se o internetové stránky, firemní zálohy, hudbu, filmy, mapové podklady, aplikace, množství statistických dat o uživatelích... Velké množství dat se dnes nachází mimo zařízení, které je používá. Dokumenty je možné editovat přímo v internetovém prohlížeči bez nutnosti instalovat příslušné programy, filmy nemusí člověk fyzicky vlastnit, protože je lze za měsíční poplatek pomocí populární služdy Netflix zhlédnout online. Jednotliví uživatelé i firmy často volí k zálohování svých dat služby Amazonu, Googlu či Microsoftu místo stavby vlastního úložiště. To vše přispívá ke vzniku mnoha velkých data center (serveroven), kam jsou všechna tato data ukládána. Tento trend lze dobře demonstrovat na Amazonu, jehož služba Amazon Web Services vytvořila ve druhém fiskálním období roku 2016 zisk 857 milionu dolarů [2] a prokazuje téměř exponenciální růst, jak ukazuje obrázek 1.1. Ostatní společnosti zabývající se cloudovými službami, například Microsoft, IBM a Google prokazují v tomto segmentu také velmi dobré finanční výsledky a prudký růst [3].

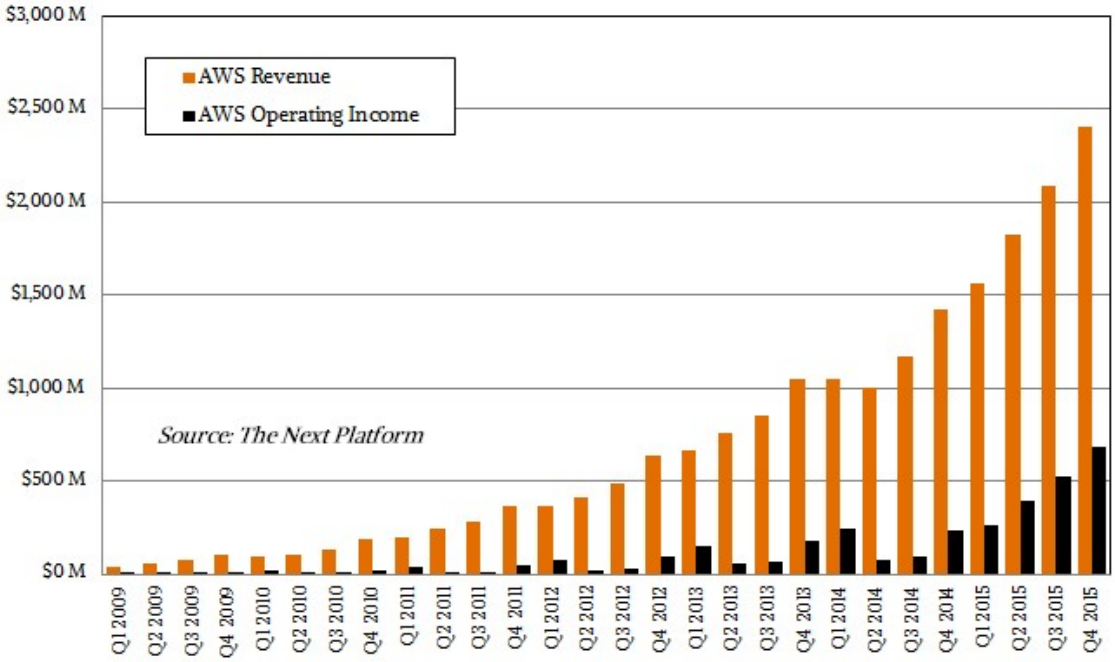
Tím vznikají rozsáhlé prostory data center, které musí být velmi přísně střežené a vyžadují stabilní teplotu a vlhkost. Hardwarové komponenty, především pevné disky, mají omezenou životnost a je potřeba je pravidelně obměňovat. Tyto změny je nutné pečlivě zaznamenávat, aby provozovatel měl přehled o množství a stavu nasazeného vybavení. Pokud by se podařilo některé tyto úkony automatizovat, znamenalo by to výrazné snížení nákladů na personál a možnost provozovat data centera i v odlehlejších nebo hůře přístupných oblastech. To dalo za vznik projektu DACEP (DAta CEnter Patrol robot).

Cílem tohoto projektu je vytvořit platformu určenou pro datová centra, která bude postavena na čtyřkolovém, diferenciálně řízeném podvozku. Nutná je přesná lokalizace a navigace. Požadavkem je schopnost monitorovat stav okolního prostředí pomocí senzorů a možnost vyčítat obsah jednotlivých komponent v racku[1].

Obsahem této práce je integrace elektronických částí tohoto robotu včetně příslušné senzoriky, vytvoření vestavěného řídicího systému, navigačního softwaru a vytvoření grafického obslužného programu pro řízení robotu včetně zobrazení diagnostických údajů.

Práce si klade za hlavní cíl být nápomocna při stavbě robotu podobné konstrukce.

[1] Rack je speciální systém pro montáž a propojování jednotlivých serverových komponent do jedné perfektně uspořádané struktury, kterou lze rychle a jednoduše připojit do elektrické a datové sítě.[4]



Obrázek 1.1: Tržby společnosti Amazon v cloudové oblasti.

2. Formulace problému

Podstatu funkcionality nezbytné pro plnění cílů autonomního mobilního robotu při použití v datovém centru lze formulovat jako plně autonomní pohyb v předem definovaném prostředí. Robot tedy musí být schopný dojet na stanovené místo a tam provést výčet dat ze snímačů. Konkrétně se jedná především o monitorování prostředí (teplota, vlhkost) a zpracování obrazových dat z kamer (identifikace zařízení v racku). Autonomní pohyb vyžaduje součinnost všech subsystémů robotu, tedy mechanického, elektrického, senzorického a řídicího, a to na nízkoúrovňové (komunikace se senzory, řízení motorů), tak vysokoúrovňové (lokalizace a plánování cesty). Mechanická konstrukce robotu, stejně jako rámcový výběr pohonu a snímačů byla v době zadání práce již k dispozici. Podstatou zadání je tedy doplnění příslušných modulů hardwarových i softwarových tak, aby robot získal výše uvedenou funkcionalitu. Dekompozicí problému získáme přehled konkrétních kroků, které tvoří dílčí cíle práce:

- výběr konkrétních typů senzorů
- výběr softwarové platformy
- návrh HW jednotek
- vestavěný řídicí systém
- vysokoúrovňové řízení
- grafické rozhraní pro řízení robotu

3. Rešerše tématu, možná řešení

Tato kapitola se věnuje základním navigačním rutinám robotu a vyhodnocením, jaké z toho plynou požadavky na výběr senzorického vybavení.

3.0.1. Framework ROS

Po domluvě s vedoucím práce bylo rozhodnuto pro navigaci robotu použít framework ROS (Robot Operating System). Jedná se o rozsáhlý framework pro meziprocesovou komunikaci, podobně jako LCM (Lightweight Communications and Marshalling). Projekt ROS získal obrovskou uživatelskou podporu, komunita má přes 10 000 uživatelů, kteří tvoří knihovny, dokumentaci a podílí se na odpovědích začínajícím uživatelům. Postupně bylo vytvořeno přes 3 000 balíčků (knihoven) obsahujících lokalizační a navigační algoritmy a mnoho dalšího. To tvoří rozsáhlý ekosystém, který v mnohém značně ulehčuje vývoj robotického softwaru. Jádro ROSu a velká část knihoven je použitelná pod licencí BSD, která patří k nejsvobodnějším licencím vůbec a umožňuje jakoukoliv část programu změnit a použít pro komereční použití.

3.1. Navigační rutiny autonomního robotu

Pro správný výběr senzorického vybavení je potřeba rozumět základním navigačním rutinám robotu. Tyto budou zběžně popsány v této kapitole.

Lokalizace
Robot musí být schopný lokalizovat se ve známém prostředí. V prostředí ROS je nejběžněji používána knihovna AMCL (Adaptive Monte Carlo Localization). Jedná se o pravděpodobnostní lokalizační systém pro pohyb ve 2D prostředí.

Knihovna vyžaduje data z laserového scanneru, odometrická data, odhadovanou počáteční pozici robotu a rastrovou mapu prostředí. Mapu lze vytvořit buď v grafickém programu, nebo nástrojem gmapping. Tento program je poskytován ROSem a s využitím metody SLAM[1] je schopný vyprodukovat mapu prostředí samotným pohybem robotu v tomto prostředí. Požadavkem jsou dostatečně přesná odometrická data a kvalitní data z laserového scanneru.

Globální plánovač
Ve chvíli, kdy je robot lokalizován a je mu zadán cílový bod v mapě, musí být systém schopný naplánovat cestu do tohoto místa, případně vyhodnotit, že zadané místo je nedostupné. K tomuto je využita mapa prostředí a mapa globálních statických překážek. Ta je vytvářena na základě dat ze senzorů s delším dosahem (2 - 30 metrů), obvykle laserového scanneru nebo kamery.

Lokální plánovač
Lokální plánovač je v ROSu podřízen globálnímu plánovači. Zajišťuje bezpečný pohyb robotu v jeho definovém okolí. Algoritmus plánuje trajektorii robotu na základě naplánované globální trasy, lokálních překážek a odometrických dat. Mapa lokálních překážek na

[1] Simultaneous Localization and Mapping - souběžná lokalizace a mapování

rozdíl od mapy globálních překážek bere v potaz větší množství senzorů včetně těch, které mají kratší dosah (1 cm - 3 metry). Jedná se především o laserový scanner, ultrazvukové snímače vzdálenosti, optické snímače vzdálenosti/přítomnosti, nárazníky... Výstupem z lokálního plánovače je příkaz k pohybu robotu ve formátu rychlost v ose x (m/s), rychlost v ose y (m/s) a rychlost rotace kolem osy z (rad/s).

3.2. Volba senzorického vybavení

Z předchozí kapitoly plynou požadavky na senzorické vybavení robotu. Další možnosti budou rozebrány níže.

3.2.1. LIDAR

Zkratka LIDAR vznikla z anglického termínu "LIght Detection And Ranging", česky se obvykle překládá jako laserový dálkoměr nebo laserový scanner a přesně podle názvu slouží k měření vzdálenosti od senzoru k překážce.
Tento senzor je využíván téměř všemi lokalizačními i navigačními systémy robotu. Z toho plyne důležitost tohoto senzoru a protože se zpravidla jedná o nejdražší senzor na robotu, je jeho výběru potřeba věnovat patřičnou pozornost a zahrnout více parametrů, ze kterých ne všechny jsou patrné na první pohled.

Pojem LIDAR dnes váže velké množství senzorů nejrůznějšího provedení, pracujících na různých principech a diametrálně se lišící cenou. Je nutné předem znát použití, požadované parametry a podle toho vybírat konkrétní model.

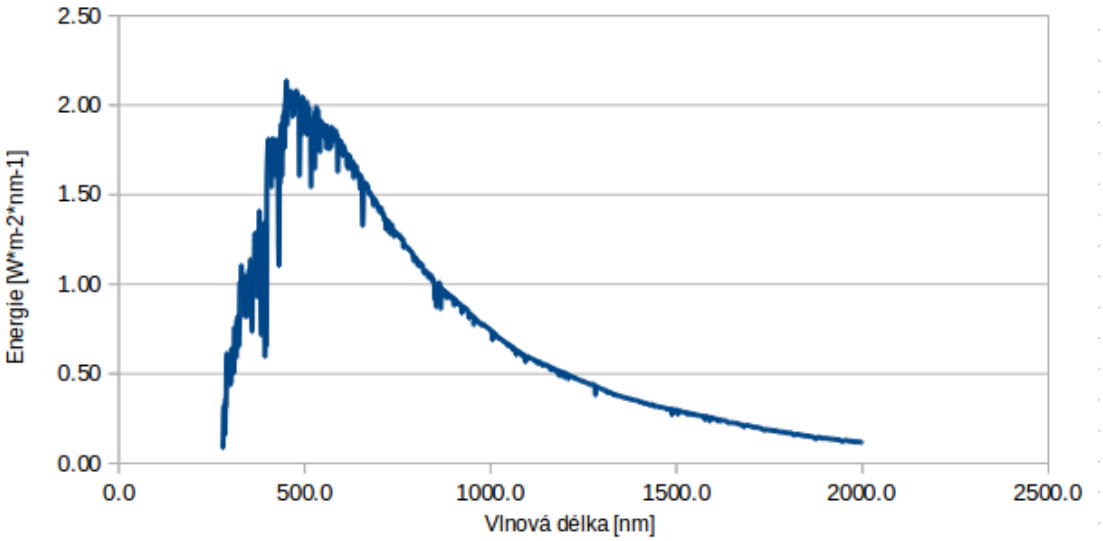
Dělení podle počtu dimenzí
Na trhu lze sehnat 1D senzory, které měří vzdálenost jednoho bodu (obrázek 3.1 a). V mobilní robotice jsou nejvíce zastoupeny 2D senzory, které měří v jedné rovině (obrázek 3.1 b). Úhlové rozlišení mají obvykle mezi 0.2° - 2°. Poskytují rozsah buď celých 360°, nebo jsou omezeny, například na 190° (Sick, řada LMS), respektive 240° (Hokuyo, řada URG). Nakonec se vyrábějí třídimenzionální jednotky zabírající celý 360° horizontální rozsah a například 27° vertikální rozsah (obrázek 3.1 c). Takový senzor dává velmi komplexní informaci o okolí, ovšem za cenu začínající na 8 000 $. V dalším dělení se věnuji již jen 2D senzorům, neboť 1D nemají v mobilní robotice velký význam (poměrově cena/výkon neodpovídají jiným typům senzorů) a 3D jsou mimo finanční možnosti většiny konstruktérů malých mobilních robotů a své uplatnění nachází až v autonomních automobilech

Dělení podle vzorkovací frekvence
Důležitým parametrem je vzorkovací frekvence, tedy jak často lze ze senzoru dostat aktuální data. U vícerozměrových senzorů použitých v mobilní robotice je to navíc důležité z důvodu, aby nedocházelo k příliš výraznému posunu senzoru vůči překážkám během jednoho scanu. Běžné hodnoty bývají 5 - 20 Hz. Jsou jednotky, které mají zvětšený dosah na úkor menší vzorkovací frekvence (například Sweep V1). Obecně platí, že zvyšování vzorkovací frekvence při zachování dosahu výrazně zvyšuje nároky na kvalitu použitých komponent a tím pádem zvyšuje cenu.

Obrázek 3.1: a) 1D GARMIN LIDAR-Lite v3[8] b) 2D LIDAR Hokuyo URG-04LX [9] c) 3D LIDAR Velodyne HDL-64 [10].

Dělení podle prostředí

Většina levnějších senzorů nemá úpravu pro venkovní prostředí, ta s sebou totiž přináší dva problémy. Prvním je požadavek na kvalitnější, alespoň částečně voděodolné pouzdro a uzpůsobení nízkým teplotám. Druhým problémem je značné rušení slunečním zářením. Na obrázku 3.2 (data převzata z [11]) je vidět, že v rozsahu infračerveného záření o vlnové délce 700 až 1000 nm, ve kterém obvykle LIDARy pracují, přichází značné množství energie a laser senzoru musí být schopný vyprodukovat pulz výrazně převyšující hladinu okolního záření.



Obrázek 3.2: Energetické rozložení záření dopadajícího ze Slunce.

Dělení podle dosahu, přesnosti

Zjevným požadavkem je, na jakou vzdálenost musí být schopný senzor snímat. Jde jak o maximální vzdálenost, tak neméně důležitou minimální vzdálenost. Na trhu jsou senzory, které nejsou schopné detekovat překážku, která je blíž než 0,5 metru[12]. Takové omezení může být pro mnohé aplikace nevyhovující a výrobci tento údaj ne vždy uvádí. Maximální vzdálenost začíná u levnějších modelů na 6 metrech. V takových případech se obvykle jedná o senzory pro vnitřní použití. Mnoho senzorů má dosah 20 - 40 metrů. A nejvýkonnější, obvykle s venkovní úpravou, mají dosah 80 - 120 metrů. Přesnost se

pohybuje v rozmezí ± 0.5 cm až ± 3 cm. Přesnost může být v celém rozsahu stejná, nebo může tvořit procento vzdálenosti, záleží na principu.

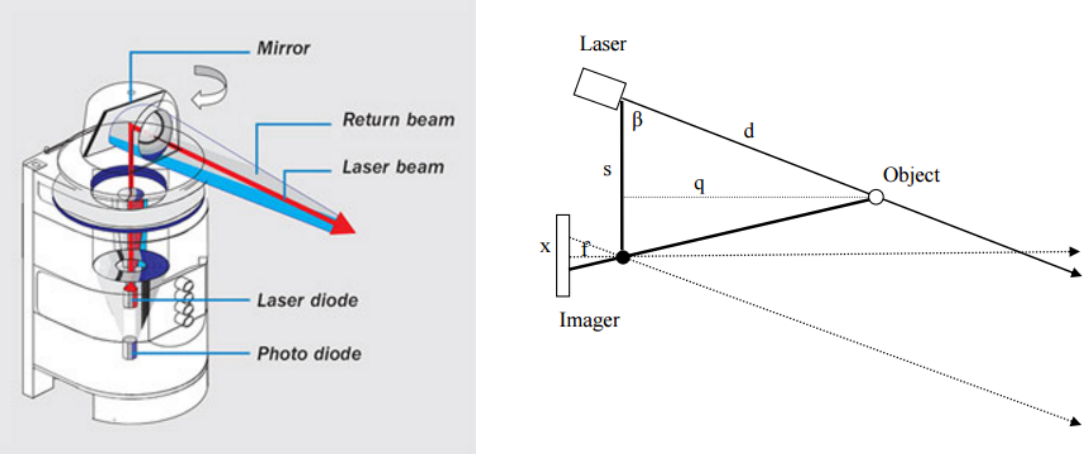
Dělení podle principu měření

Princip měření vzdálenosti může být trojího typu.

Jedním principem je *time of flight*, tedy měření doby mezi vysláním krátkého světelného pulzu a jeho detekcí senzorem po odražení překážkou, jak je znázorněno na obrázku 3.3 a). Klíčovou součástí je rotující zrcátko se známou polohou, které určuje směr světelného pulzu a poskytuje tak druhý rozměr. Výhodou je, že jediná rotující součást je v tomto provedení jen toto pasivní zrcátko. Tím může být zaručena spolehlivost, robustnost a velká vzorkovací frekvence. Nevýhodou je, že světlo za jednu nanosekundu urazí 30 cm. Je vyžadováno tedy naprosto přesné časování v řádu desítek pikosekund, a to klade velké požadavky na kvalitu komponent a zpracování signálu. Tento princip je používán například senzory Sick nebo Velodyne.

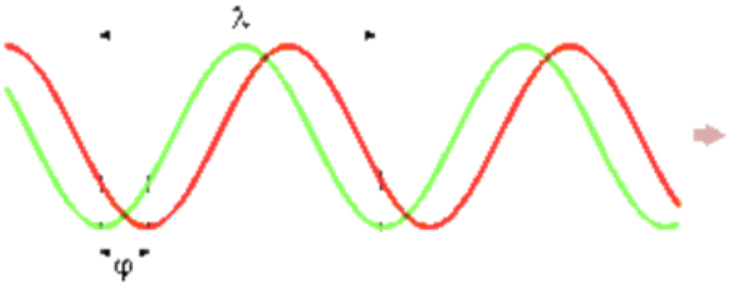
Druhým principem je měření fázového posunu. Senzor pracuje s monochromatickým laserem o přesně stanovené vlnové délce, kdy se měří fázový posun mezi vyslaným paprskem a paprskem odraženým zpět od překážky (viz obrázek 3.4). Tyto mají stejnou periodu, ale rozdílnou fázi. Tento posun je možné převést na vzdálenost, kterou paprsek od laseru ke snímači urazil. Tento princip využívá u svých senzorů japonská firma Hokuyo.

Je evidentní, že v obou případech má výsledek lineární závislost na vzdálenost. Díky tomu mají oba typy senzorů neměnnou přesnost v celém rozsahu měření.



Obrázek 3.3: a) Princip *time of flight* [13] b) princip triangulace [14].

Poslední používanou metodou, která se u 2D senzorů více uplatňuje teprve v posledních letech, je postavená na principu triangulace. Jednotka je složena z laseru a detektoru (digitálního obrazového snímače), kdy poloha těchto dvou části vůči sobě je přesně definovaná (obrázek 3.1 b). Vzdálenost je pomocí triangulace vypočtena z polohy na obrazovém snímači, kam odražený paprsek dopadne. Tato metoda je principiálně méně náročná na použité komponenty, a proto výrazně (řádově) levnější na výrobu, je ale citlivější na sluneční rušení. Druhou nevýhodou je, že celá senzorická část s elektronikou složená z laseru i detektoru rotuje. Tím se zhoršuje spolehlivost a snižuje vzorkovací frekvence (snahou o menší opotřebení kontaktních ploch rotujících datových a napájecích linek). V tomto měření se chyba vypočtené vzdálenosti zvyšuje se vzdáleností.



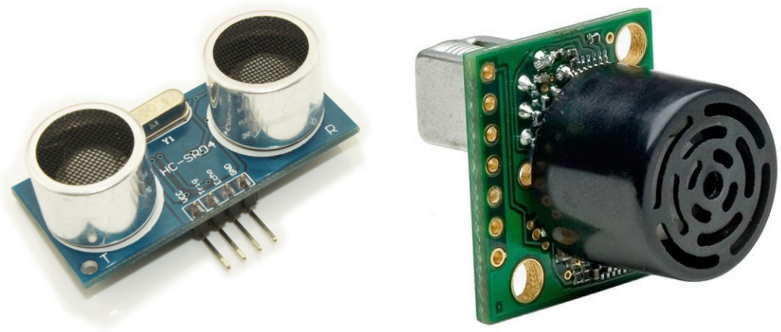
Obrázek 3.4: Princip fázového posunu [15]

3.2.2. SONAR

Sonar (SOund Navigation And Rangin) je v mobilní robotice zařízení schopné detekovat vzdálenost překážky od senzoru. Princip spočívá ve vyslání modulovaných ultrazvukových pulzů (obvykle kolem 40 kHz) a jejich opětovná detekce po odražení od překážky. Z času mezi vysláním a detekcí se počítá vzdálenost překážky. Protože rychlost zvuku je mnohonásobně nižší než rychlost světla, nevyžaduje tato metoda tak přesné komponenty a SONAR proto patří mezi levnější senzor.

Provedení sonaru může být různé. Vysílač i přijímač mohou být vyvedeny v jednom pouzdru (obrázek 3.5 b), nebo je možné mít vyvedeny vysílač a přijímač zvlášť (Obr. 3.5 a). Velký rozdíl je v komunikačních rozhraní - nejčastěji se používá buď I^2C nebo PWM výstup. Další parametry potom jsou rozlišení, vzorkovací frekvence (kvůli nízké rychlosti zvuku nepřesahuje 10 Hz), maximální dosah (3 - 6 m), minimální dosah (1 - 5 cm)… Cena se běžně pohybuje od 2 $ (vysílač a přijímač jsou zvlášť, senzor má PWM výstup) do 50

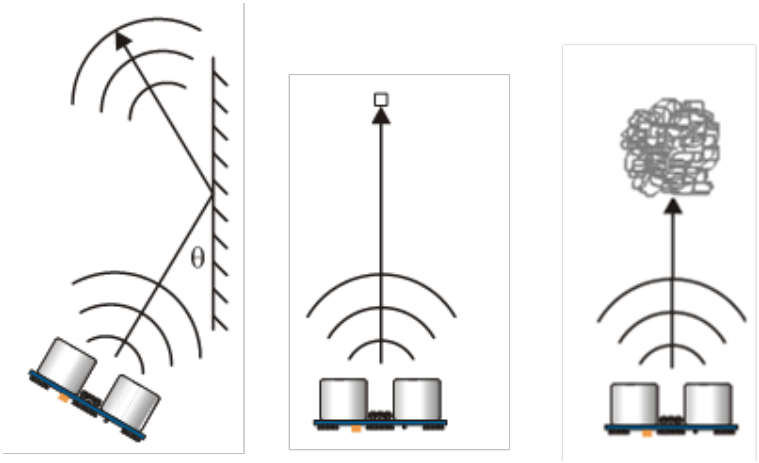
$ (vysílač a přijímač v jednom pouzdře, I^2C rozhraní). Vlastní kategorii tvoří průmyslové senzory a senzory s venkovní úpravou.



Obrázek 3.5: a) Sonar SRF04 b) I2CXL-MaxSonar-EZ4 [16].

Zpracování dat měřených sonarem není zcela triviální. Při měření nastává mnoho případů, kdy změřená hodnota neodpovídá. Na obrázku 3.6 jsou případy, kdy překážka není senzorem detekována. Jedná se tyto případy: snímání hladké plochy pod větším než kritickým úhlem, snímání příliš malé překážky nebo překážky pohlcující ultrazvukový pulz. Dále dochází k falešným detekcím způsobeným například mnohačetným odrazem v rohu nebo detekcí pulzu jiného senzoru. Další problém u mobilního robotu například při jízdě po dlažbě jsou náhodné detekce hran jednotlivých dlaždiček. Naměřenou vzdálenost ovlivňují fyzikální parametry vzduchu (a tedy rychlost šíření zvuku), především teplota a vlhkost.

V robotu DACEP je použito 8 senzorů typu SRF08. Ty slouží lokálnímu plánovači pro detekci překážek bližších než 3 metry.



Obrázek 3.6: Chyby při měření sonarem [17].

3.2.3. Kamera

Ve vnějším prostředí je kamera téměř nezbytná pro lokalizaci. Ve vnitřním prostředí je účelnější mít jako nosný senzor laserový dálkoměr, který dává podstatně snadněji zpracovatelná data. Kamera potom slouží především pro přenos obrazu operátorovi a logování dat. V robotu DACEP boční kamera slouží k vyhledávání a dekódování QR kódů.

Zajímavou alternativou je stereovize. Myšlenka dvou obyčejných kamer uchycených v definované poloze bohužel není realizovatelná, takové řešení nesplňuje požadavky na kvalitu obrazu a především přesnou polohu dvou senzorů vůči sobě a není tedy schopné naplnit očekávání stereovize. Řešením by mohlo být například v levnější stereokameře ZED od firmy Stereo labs (obrázek 3.7) a výpočetního minipočítače Nvidia Jetson TK1, nebo dostatečně výkonné grafické karty. Firma Stereo labs vytvořila pro kameru ZED knihovny pro ROS na zpracování obrazu a lokalizaci. Takové řešení kamery s výpočetní kartou stojí cca 900 $, tedy srovnatelně jako laserový scanner a je použitelné pro lokalizaci i ve venkovním prostředí. Vyžaduje ale podstatně komplikovanější algoritmy pro zpracování a velký výpočetní výkon vyžadující vlastní grafickou kartu.

Obrázek 3.7: Stereo kamera ZED [18].

3.2.4. Nárazníky

Pro případ, že žádný z předchozích senzorů nedetekuje překážku a dojde k nárazu, je pro jeho detekci potřeba použít nějakou formu nárazníku. Nejčastěji se jedná o pevnou obrubu, která v případě nárazu tlačí na mikrospínač.

Společnost Bender Robotics vyvinula pneumatický nárazník tvořený gumovým D profilem, který je připojený na pneumatický tlakový senzor. Při nárazu je D profil stlačen, uvnitř dojde ke zvýšení tlaku a tato změna je velmi spolehlivě detekovatelná tlakovým snímačem.

4. Upřesnění cílů

Z rešeršní studie je patrné, jaké senzorické vybavení robot bude muset obsahovat. Po poradě s vedoucím práce bylo rozhodnuto, že pro vestavěný řídicí systém bude použita platforma Mbed a pro vysokoúrovňové řízení bude využit robotický framework ROS. Ten má značnou uživatelskou podporu a obsahuje mnoho užitečných nástrojů, které mohou být nápomocné při vývoji robotu. Tím práce navazuje na diplomovou práci Petra Tomáše [5], který se zabývá základy ROSu a dále potom Luboše Vence [20] rozebírající více do hloubky problematiku navigace robotu s využitím tohoto frameworku. Práce má za cíl být nápomocna při stavbě robotu podobné konstrukce, kapitoly s vlastní realizací tedy předpokládají znalosti ROSu v rozsahu těchto závěrečných prací.

5. Vestavěný řídicí systém

Vestavěný řídicí systém zprostředkovává komunikaci mezi jednotlivými senzory a nadřazeným počítačem a naopak předává pokyny od počítače motorům a zajišťuje provádění specifických úkolů. Pro vestavěný systém byla vybrána platforma Mbed, která sjednocuje knihovny pro různé ARM procesory. Jednotným způsobem se tak přistupuje na vstupně výstupní piny a jiné periferie a výsledný kód lze zkompilovat na jakýkoliv procesor podporovaný platformou Mbed (do projektu jsou zapojeni například výrobci NXP, Freescale, STM a jiní, vybírat je možné mezi více než 100 vývojovými deskami).

Jedná se o podobný projekt jako Arduino. Ten sjednocuje práci s periferiemi především osmibitových mikrokontrolérů (nejprve Atmel, nyní i jiné). Zatímco projekt Arduino se snaží o co největší zjednodušení při programování levných mikrokontrolérů a cílí tak především na kategorii hobby, Mbed má cílovou skupinu firmy zabývající se vývojem, především potom vývojem IoT [1].

5.1. Schéma zapojení

V následující kapitole bude vysvětleno zjednodušené schéma zapojení dílčích elektronických celků robotu. Schéma zapojení je na obrázku 5.1. Vynechány jsou napájecí obvody a části, které nebyly vytvořeny autorem této diplomové práce. Na schématu jsou vysokoúrovňové systémy podbarveny červeně, vestavěný řídicí systém oranžově. Vstupy (senzory) jsou žluté, výstupy a aktuátory modré. Šedě jsou jiné elektronické celky, bíle sběrnice a komunikační kanály.

PC Master

Pro zprostředkování vysokoúrovňového řízení robotu je instalován stolní počítač umístěný v pasivně chlazeném hliníkovém boxu. Jako operační systém je použito Ubuntu 14.04 a pro navigaci rozsáhlý framework ROS. Vytvořením navigačního softwaru se zabývá kapitola 6. Počítač přes USB hub přijímá data z kamer a laserového scanneru a komunikuje s mikrokontrolérem LPC1768, který dále zprostředkovává nízkoúrovňové řízení.

PC Slave

Pro kontrolu a řízení robotu byl vytvořen vizualizační software, který se přes WiFi může připojit k Master PC. V tomto softwaru je vizualizován robot a jeho prostředí snímané instalovanými senzory, je zobrazen obraz z kamer a termokamery, je možné zadávat robotu nové cíle pohybu, ovládat LED osvětlení, kontrolovat stav systémů (baterie, chybová hlášení…) a další. Grafickým prostředím se zabývá kapitola 7.

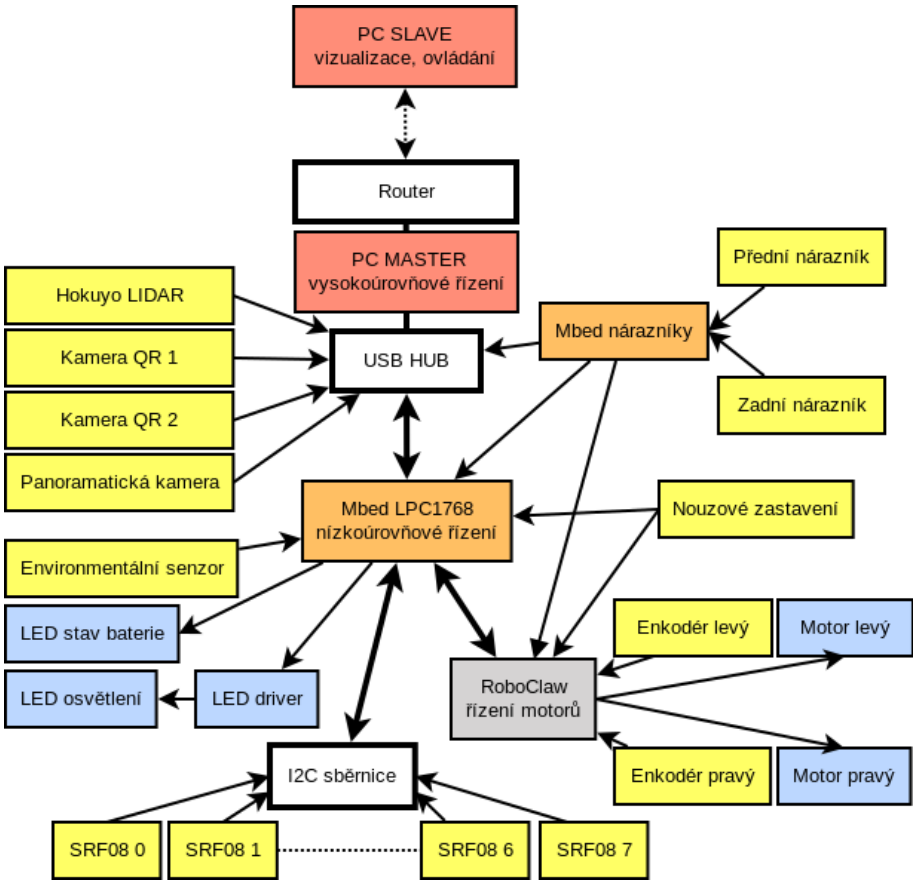
Mbed LPC1768

Veškeré nízkoúrovňové řízení obstarávají dva mikrokontroléry LPC1768 od firmy NXP, přičemž jeden z nich je použit pouze na vyhodnocení dat z náraznků. Tato činnost by nevyžadovala vlastní mikrokontrolér, ale kvůli kritické funkcionalitě bylo přistoupeno k tomuto řešení pro co největší bezpečnost systému.

[1] IoT - Internet of Things (internet věcí)

Řízení motorů

Pro řízení motorů byla vybrána deska Roboclaw od firmy Ion Motion Control. Umožňuje komunikovat přes rozhraní UART a je podporováno mnoho pokročilých funkcích. Mimo jiné je deska schopna měřit vstupní napětí, měřit proudové odběry motorů, chod motorů je možné řídit mnoha různými způsoby, na vstupní piny je možné připojit enkodéry a nouzové tlačítko zapojení. Podrobněji se zapojení tohoto mikrokontroléru věnuje kapitola 5.5.



Obrázek 5.1: Zjednodušené schéma zapojení řídicích prvků robotu.

5.2. Platforma MBED

Vestavěný řídicí systém je vytvořen pod platformou Mbed. To umožňuje rychlý vývoj programu bez instalace vývojového prostředí, protože Mbed obsahuje online vývojové prostředí a kompilátor, pro vývoj aplikace stačí potom internetový prohlížeč. Po vytvoření programu tedy stačí vybrat konkrétní mikrokontrolér, pro který je program určen, na cloudu proběhne kompilace a je stažen binární soubor, který je použit pro naprogramování tohoto mikrokontroléru. To má bonus v podobě platformní nezávislosti (je možné vytvářet program v jakémkoliv operačním systému - stačí webový prohlížeč) a bezpečné umístění programu na vzdáleném úložišti. Nevýhodou je jednoduché vývojové prostředí bez zvýrazňování syntaxe, doplňování a jiných poměrně běžných nástrojů moderních vývojových prostředí. Druhou nevýhodou je, že vývojář je odkázán na kompilaci mimo jeho počítač, která obvykle trvá řádově jednotky sekund, v době většího vytížení (během odpoledních hodin) může dojít k přetížení kompilačního serveru a kompilace potom trvá desítky vteřin až minuty. Pro koho by tyto nevýhody převažovaly výhody, může si exportovat program z on-line prostředí a importovat do vývojového prostředí podporovaného výrobcem daného mikrokontroléru (podporované jsou exporty například pro Keil uVision, IAR systems, GCC a jiné).

5.2.1. Mbed RTOS a jeho nedostatky

Kód pro Mbed se píše v programovacím jazyku C++, přináší s sebou tedy všechny výhody a problémy objektového programování. Přímo Mbed vyvinul svůj real-time operační systém, postavený nad CMSIS-RTOS, který je určen pro ARM mikrokontroléry. Na tomto místě bych rád upozornil na některé problémy, které se v tomto operačním systému vyskytují a v manuálu jsou zmíněny přinejlepším mlhavě. Na začátku tohoto roku vyšla nová verze operačního systému mbed-rtos s označením 5, která některé nedostatky řeší, ale ne vždy a ne zcela uspokojivě.

Velikost stacku
V tomto případě se nejedná o chybu ale obecně (u všech real-time operačních systémů) o věc, na kterou je třeba dávat pozor. Některé knihovny (například knihovna pro čtení z SD karty) jsou poměrně hardwarově náročné a snadno je potom překonána velikost stacku daného procesu. Mbed RTOS má pro procesy velmi snadno použitelné funkce *free_stack()* a *used_stack()*. Důrazně doporučuji tyto funkce využívat ke kontrole, že je ve stacku u všech procesů dostatek volného místa. Přetečení stacku je chyba, kterou je těžké najít.

Priorita procesů
Na některých procesorech špatně funguje nastavování priority jednotlivých vláken. V některých případech se může stát, že vlákno s větší prioritou nedá prostor žádnému jinému vláku, vlákno s nižší prioritou se naopak nemusí nikdy vykonat, nebo (a to je horší případ) se vykonává ve zcela náhodných okamžicích. Do určité míry tyto problémy mohou záviset na nastavené dostatečné velikosti stacku.

Použití knihovny Serial
Na toto je v manuálu upozorněno, ale považuji za důležité to zdůraznit. Při použití real-

-time operačního systému není možné kvůli MUTEXu[2] využívat knihovnu Serial. Je ovšem možné ji nahradit knihovnou RawSerial, která problémy řeší a je určena pro použití s OS. Jen je osekaná o některé funkce, které použití sériové komunikace ulehčují.

RTOS Timer
Veliké nepříjemnosti se mohou stávat při použití RtosTimer. Ten vykonává určitou funkci s definovanou periodou. Nikde ovšem není k dohledání, že se v důsledku jedná jen o nadstavbu obyčejného timeru a **celý vykonávaný kód se tedy vykonává v přerušení**. To má dvojí problém. V tu chvíli nejsou vykonávána ostatní vlákna a když jsou ve funkci prováděny složitější matematické operace a dojde k přerušení například sériovou komunikací, která má vyšší prioritu, může dojít k narušení výsledku. Program potom funguje korektně a čas od času vrátí naprosto nesmyslnou hodnotu, nebo může dokonce dojít k zamrznutí mikrokontroléru. V takovém případě, není-li implementován nezávislý watchdog, robot pokračuje v posledním odeslaném příkazu (například pohybu plnou rychlostí vpřed).

5.3. Obsluha sériové komunikace

Komunikace mezi počítačem a vestavěným řídicím systémem musí být zcela spolehlivá, stabilní a musí být dimenzována na přenos alespoň tisíce zpráv za sekundu.

5.3.1. Komunikační protokol

Pro potřeby předávání zpráv byl vytvořen protokol sestávající z hlavičky, těla zprávy a kontrolního součtu, jak je znázorněno na obrázku 5.2, vysvětlení jednotlivých bytů se nachází v tabulce pod ním 5.1. V tabulce 5.2 je výpis podporovaných typových hodnot. Message ID hodnoty jsou vypsány v příloze B.

Obrázek 5.2: Komunikační protokol vytvořený pro robot DACEP.

hodnota	typ proměnné
Start byte	Start byte o fixní hodnotě 0xF0
Message length	Délka těla zprávy (bytů Data[])
Data type	Formát dat v těle zprávy
Message ID	Jedinečný identifikátor typu zprávy
Data	Tělo zprávy
Control Checksum	Kontrolní součet

Tabulka 5.1: Komunikační protokol

[2]MUTual EXclusion - algoritmus zabraňující vykonávání dvou a více kódů nad jedním sdíleným prostředkem. V tomto případě sériovou linkou.

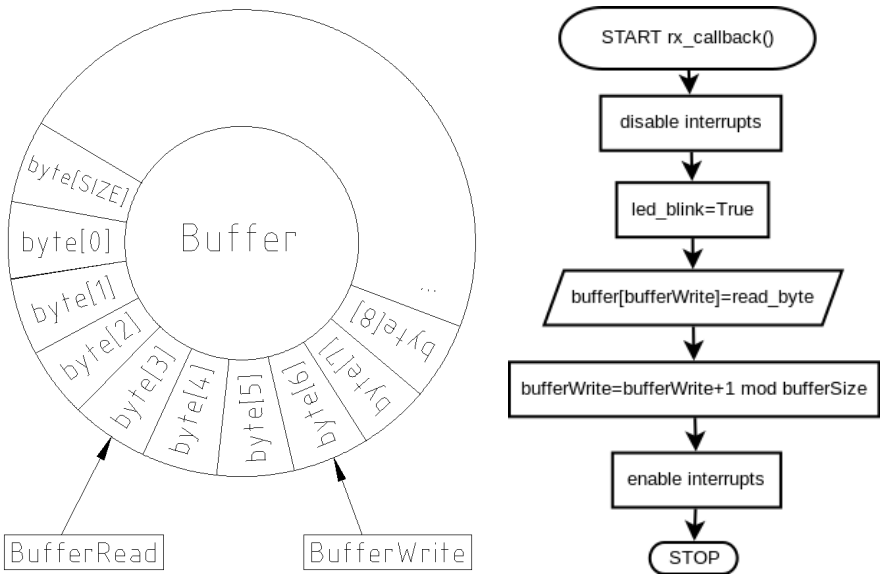
hodnota	typ proměnné
000	zpráva neobsahuje data
001	unsigned int8
002	signed int8
010	unsigned int16
011	signed int16
012	unsigned int32
013	signed int32
014	unsigned int64
015	signed int64
020	float32
021	float64
030	string utf-8
050	double int32

Tabulka 5.2: Protokolem podporované formáty dat.

5.3.2. Příjem zprávy

Uložení do kruhového zásobníku - callback funkce příjmu dat

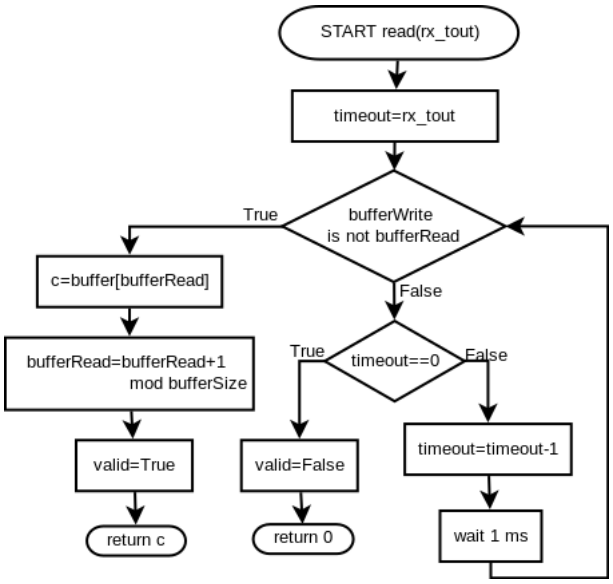
Příjem zpráv je řešen asynchronně vektorem přerušení. Přijaté byty jsou ukládány do kruhového zásobníku o fixní velikosti, viz obrázek 5.3 a). Pro uchování informace, na jakém místě zásobníku se nachází nezpracovaná data, slouží dvě proměnné *bufferRead* a *bufferWrite*. Ve chvíli, kdy je přijat po sériové lince nový byte, tento je zapsán na místo označené proměnnou *bufferWrite*. Proměnná *bufferWrite* je o jedna zvětšena, pokud dosáhla maximální hodnoty zásobníku, je přepsána na nulu. K tomu je nejjednodušší použít operace modulo. Diagram tohoto procesu je na obrázku 5.3 b). Když je byte ze zásobníku zpracován dalším procesem, je proměnná *bufferRead* zvětšena o jedna a když přesáhne velikost zásobníku, je přepsána na nulu stejným systémem jako *bufferWrite*.



Obrázek 5.3: a) Kruhový zásobník b) Funkce pro příjem dat.

Vyčtení ze zásobníku - funkce read

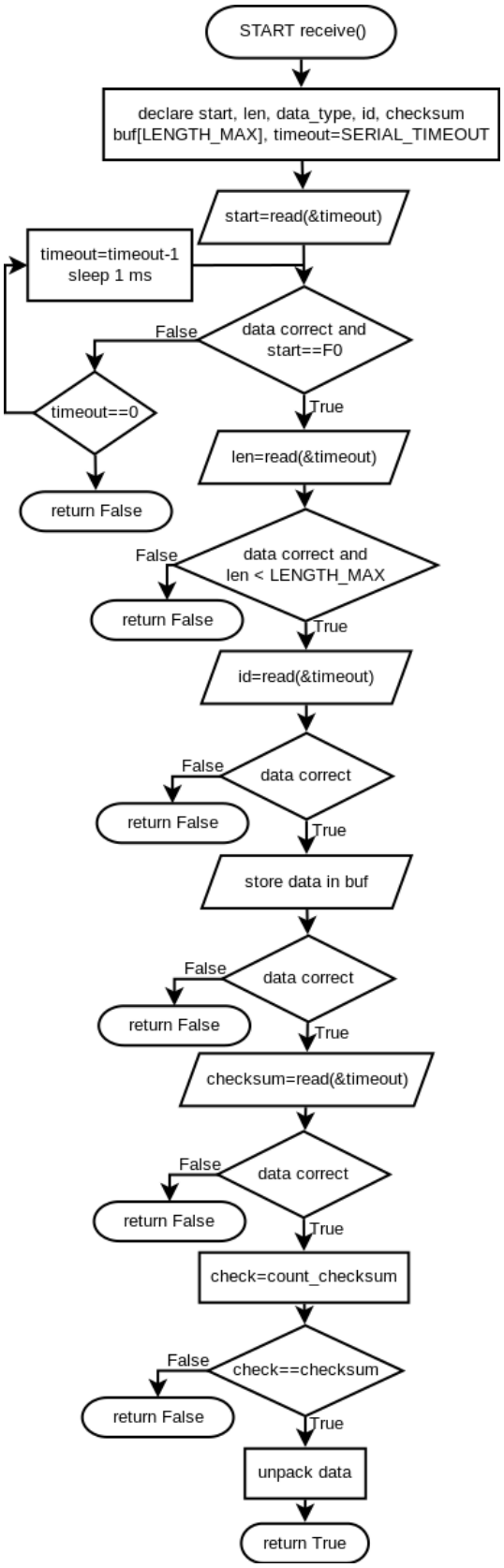
Ve druhém vlákně probíhá zpracování přijatých dat. Výhoda tohoto přístupu je, že příjem dat nezatěžuje procesor a neovlivňuje jiné procesy, data tak mohou být nejprve přijata a zpracována až v definovanou dobu. Při správném použití se vykonávání programu nezastaví na operaci čekání na vyžádaná data. Diagram zpracování dat je na obrázku 5.4. Prvním krokem je kontrola, zda se v zásobníku nachází nezpracovaná data. Toho lze docílit kontrolou, že *bufferWrite* není stejný jako *bufferRead*. Pokud tato podmínka není splněna, čeká se 1 ms a zkouší se to znovu, dokud není dosaženo časového limitu definovaného v milisekundách. Pokud jsou v zásobníku nová data, vyčtou se do proměnné, posune se proměnná bufferRead, jak je popsáno výše, poznamená se do proměnné, že příjem byl úspěšný a data jsou poslána jako návratová hodnota funkce.



Obrázek 5.4: Funkce čtení dat ze zásobníku.

Příjem a rozbalení celé zprávy

Přijetí celé zprávy je zobrazeno na obrázku 5.5. Postupně je načtena hlavička a zkontrolovány její náležitosti. Zpráva začíná čekáním na *start byte*. V případě, že nenastala chyba v přenosu, jedná se o první přijatý byte, je potřeba mít ale ochranný mechanismus právě pro případ chyb přenosu. Následuje údaj o velikosti zprávy, která nesmí přesáhnout definovanou hodnotu. Pokud by se tak stalo, zpráva je zahozena a je vyčištěn zásobník (proměnné *bufferRead* a *bufferWrite* jsou jednoduše vynulovány) a to z důvodu, aby se dlouze neprováděl příjem zprávy, o které je předem známo, že není validní. Následuje uložení typu zprávy a identifikátoru pro další zpracování. Následně je přečteno tělo zprávy a uloženo do zásobníku buf. V závěru se přečte kontrolní součet a porovná se součtem vytvořeným přijatými daty (jde o součet od druhého přijatého bytu, tedy délky zprávy). V budoucnosti bude tento součet nahrazen alespoň 16-ti bitovým CRC kontrolním součtem, který je spolehlivější. Pokud v kterémkoliv kroku nastala chyba, příjem je ukončen chybovou návratovou hodnotou. Pokud příjem proběhl v pořádku, je navrácena hodnota *True*.

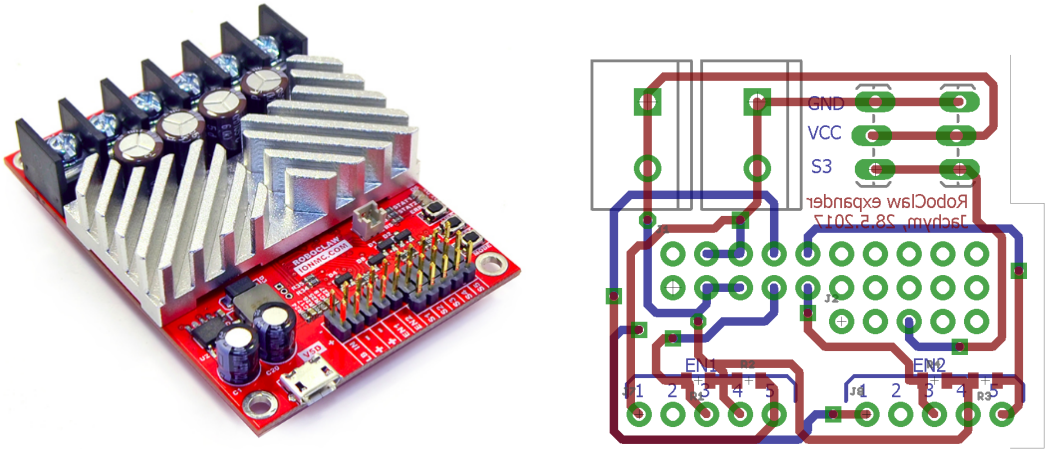


Obrázek 5.5: Funkce zpracování dat.

5.4. Modul pro odečítání odometrie

Jako kontrolér motorů byl vybrán RoboClaw od firmy Ion Motion Control (obrázek 5.6 a). Tento kontrolér má provozní napětí mezi 6 V až 34 V. Trvalý proud je maximálně 2x15 A, špičkový 2x30 A. Kontrolér umožňuje několik komunikačních rozhraní, celá funkcionalita je umožněna pouze při připojení přes UART. Kontrolér snímá úroveň vstupního napětí, proud, který je pouštěn do motorů, podporuje připojení enkodérů. Komunikace je paketová s kontrolním součtem. Umožňuje velké množství nastavení a speciálních příkazů, jako spuštění motorů s rozběhovou rampou.

Na vstupně výstupní piny kontroléru byla vytvořena rozšiřující deska (obrázek 5.6 b), na které jsou vývody pro připojení k řídicímu mikrokontroléru, konektory ke snadnému připojení enkodérů a nouzového tlačítka zastavení.

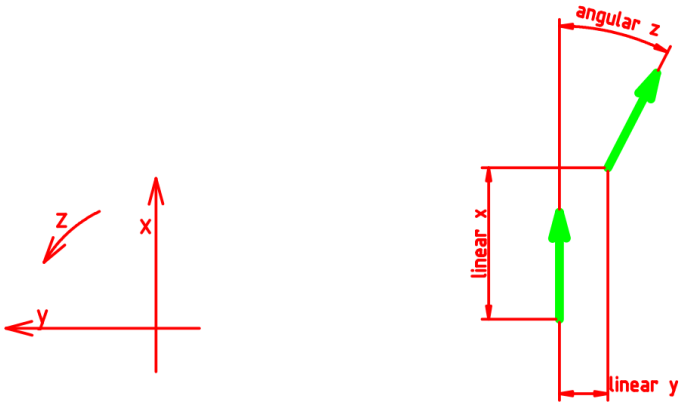


Obrázek 5.6: a) Kontrolér motorů RoboClaw [19] b) Navržená deska pro vyvedení konektorů.

5.4.1. Odometrický model

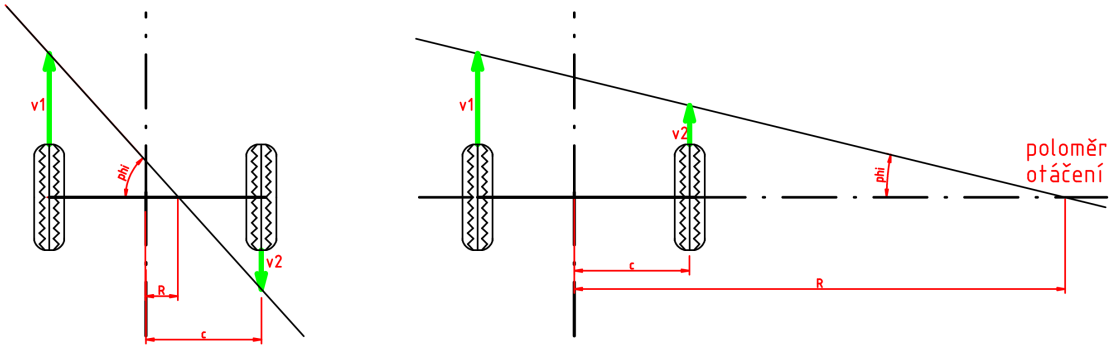
Velkou výhodou diferenciálního podvozku oproti například čtyřkolovému ackermanovu podvozku je možnost otočit se na místě. S tím souvisí podstatně jednodušší vztahy pro výpočet poloměru zatáčení a následného dopočtu odometrických dat. Odometroie z pohledu mobilní robotiky je transformace dat z enkodérů na změnu pozice, jak je vidět z obrázku 5.7 b). Zeleně je vyznačena změna pozice a orientace robotu, červeně jsou data lineárního posunu v ose x, y a úhlového posunu v ose z. Znaménková konvence v ROSu je na obrázku 5.7 a). Stejným způsobem (linear_x, linear_y, angular_z) jsou předávány i požadavky na pohyb robotu.

Do výpočtu odometrie vstupují, jak bylo řečeno, data z enkodérů, tedy jak moc se od minulého měření otočilo levé a pravé kolo. Z výpočtu je potřeba získat lineární posun robotu v ose x, v ose y a úhlové otočení v ose z. Tato data jsou následně zderivována a rychlost se posílá plánovači ROSu. K výpočtu pomůže obrázek 5.8, kde jsou schematicky znázorněna dvě kola, zeleně potom vektory jejich rychlosti. Je zřejmé, že pokud mají levé i pravé kolo stejnou rychlost, jede robot rovně vpřed. Pokud se každé točí stejně rychle, ale na opačnou stranu, robot se otáčí na místě kolem středu kol. Pokud mají kola stejný



Obrázek 5.7: a) Znaménková konvence odometrie b) Odometrie pohybujícího se robotu.

směr a rozdílnou rychlost, robot jede vpřed (respektive vzad) po kružnici s poloměrem R[7].



Obrázek 5.8: Dvě ukázky výpočtu poloměru rotace u diferenciálního podvozku.

Stanovme počátek souřadného systému uprostřed osy robotu. Pro rychlosti kol musí platit tyto rovnice:

$$v_1 = \varphi(R - c) \tag{5.1}$$

$$v_2 = \varphi(R + c) \tag{5.2}$$

Z těchto rovnic plyne vzdálenost středu rotace R:

$$R = c * \frac{v_1 + v_2}{v_2 - v_1} \tag{5.3}$$

Úhel otočení φ je vypočítán rovnicí 5.4,

$$\varphi = \frac{v_2 - v_1}{2 * c} \tag{5.4}$$

posun v ose x a y potom je:

$$v_x = \frac{v_1 + v_2}{2} * \cos\varphi \tag{5.5}$$

$$v_y = \frac{v_1 + v_2}{2} * \sin\varphi \tag{5.6}$$

Opačné vztahy jsou potřeba, když počítač vypočítá ideální trajektorii a předává systému pod sebou příkazy v podobě žádané rychlosti x, y a rychlost otáčení z a systém z těchto údajů musí dopočítat rychlost pro levé a pravé kolo:

$$R = \frac{v_x}{\sin z} \tag{5.7}$$

$$v_1 = z * (R - c) \tag{5.8}$$

$$v_2 = z * (R + c) \tag{5.9}$$

Všechny tyto výše uváděné vztahy jsou linearizované a je možné je s dostatečnou přesností použít jen pro φ menší než $\pi/10$!

5.5. Modul pro řízení motorů

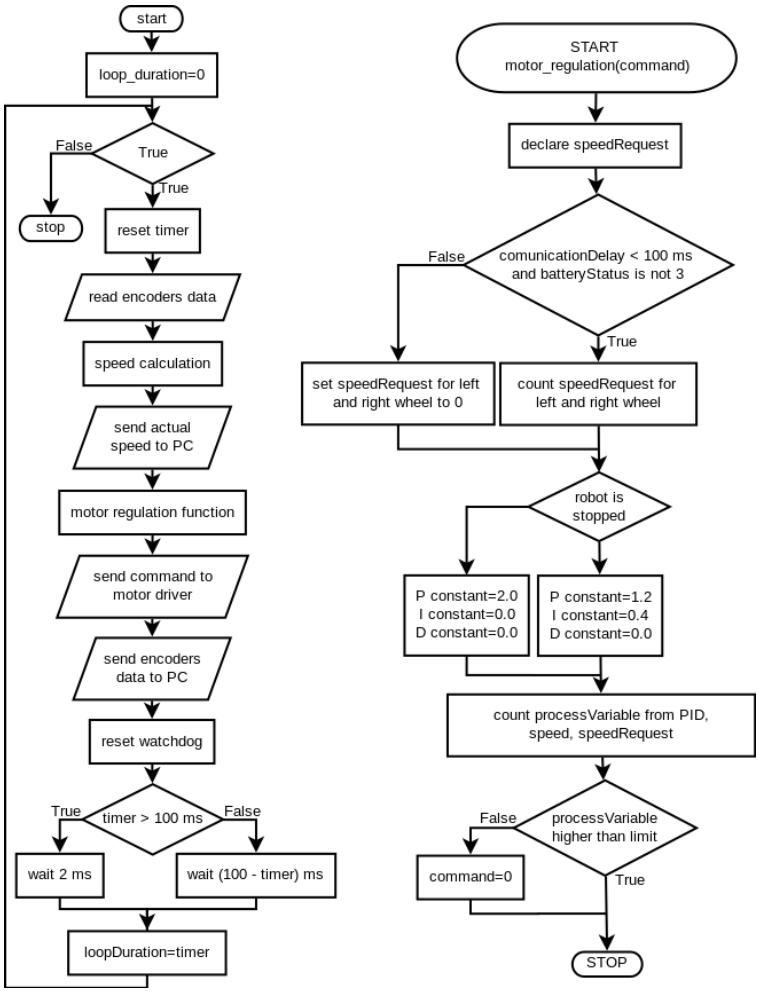
K řízení motorů je použit stejný kontrolér jako pro odečet odometrie, tedy RoboClaw od Ion Motion Control. Řízení motorů probíhá ve vlastním vlákně s periodou 100 ms a je na něj navázán nezávislý watchdog - pokud by se výpočet pozdržel, je z bezpečnostního důvodu resetován celý mikrokontrolér. Vlákno řízení motorů je na obrázku 5.9 a). Vlákno probíhá v nekonečné smyčce. Na začátku každého běhu je resetován časovač. Následuje vyčtení stavu enkodérů. Z těchto dat je vypočítáná aktuální rychlost, která vstupuje do PID regulátoru (porovnání žádané/skutečné hodnoty). Rychlost se pro kontrolní účely posílá i do nadřazeného počítače. Následuje řídicí funkce pro výpočet akčního zásahu pro motory. Tato hodnota je předána kontroléru motorů a data z enkodérů jsou opět poslána do počítače. Na konci smyčky se resetuje watchdog a funkce čeká příslušnou dobu, aby perioda tvořila žádaných 100 ms. Závěrem je délka trvání smyčky uložena do proměnné.

Výpočet akčního zásahu je zobrazen na obrázku 5.9 b). Motory jsou řízeny pomocí PID regulátoru, kdy D složka je nastavena na nulu. V prvním kroku se kontroluje, zda nebyl překročen časový limit pro komunikaci s počítačem. Pokud ano, je vyhodnocen potenciální problém s komunikací a robot je zastaven, dokud není spojení obnoveno. Toho je dosaženo tak, že žádaná rychlost pro levé a pravé kolo je v takovém případě nastavena na nulu. V opačném případě se podle rovnic 5.8 a 5.9 podle příkazu z počítače dopočítává žádaná rychlost pro levé a pravé kolo. Následuje kontrola, jestli robot není zastaven (ať už přirozeně, nebo pomocí tlačítka nouzového zastavení). Pokud rychlost robotu je blízká nule, jsou přenastaveny parametry PID regulátoru tak, že I je vynulována a P zvětšena. Důvod je dvojí: bylo experimentálně zjištěno, že s tímto nastavením kvůli šnekové převodovce, která má vysoký koeficient statického tření (a tím je soustava značně nelineární), je rozjezd robotu plynulejší. Druhý důvod je zmáčknuté tlačítko nouzového zastavení. V takovém případě je velmi nežádoucí, aby I složka dále narůstala (robot stojí, ale příkaz k pohybu zůstává, protože nouzové tlačítko není navázáno na program pracující na počítači). Podle žádané rychlosti a aktuální rychlosti je PID regulátorem dopočítáván akční zásah pro motory. Pokud je tato hodnota menší než limit, je oříznuta na nulu. Důvodem je zvýšení životnosti DC motorů, kterým se při velmi malé rychlosti opalují kartáče.

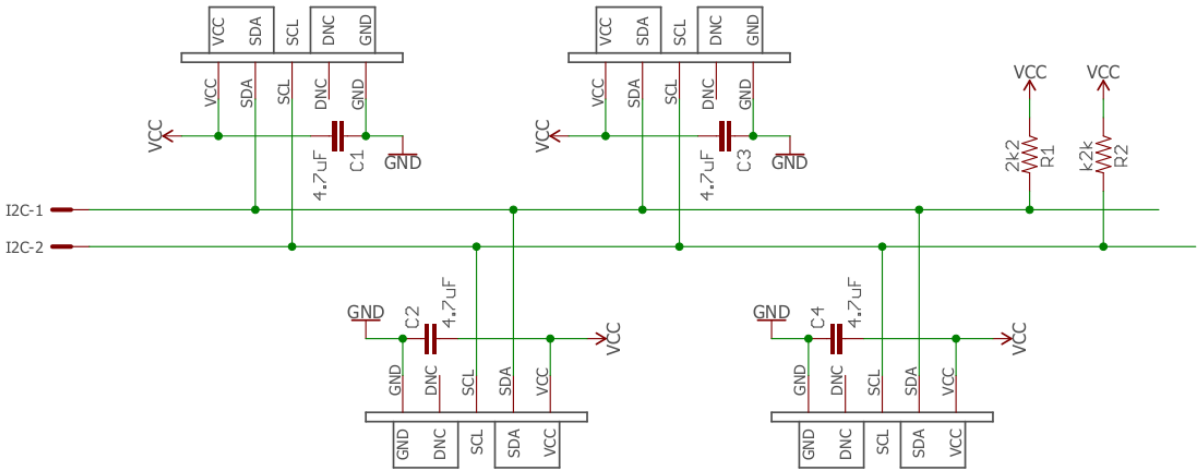
5.6. Modul pro zpracování dat z ultrazvukových senzorů

Jako ultrazvukové senzory byly vybrány SRF08, komunikující přes I^2C rozhraní. Sběrnice I^2C musí mít linku SDA i SCL pull-up rezistorem připojenou na napájecí napětí. Hodnota rezistorů se může lišit podle délky sběrnice a rychlosti komunikace, v případě robotu DACEP vyhovují rezistory o hodnotě 2k2. Senzory mají velmi malý odběr kolem 10 mA, ve chvíli přijetí příkazu k měření je ovšem proud na 3 μs 275 mA. Na robotu je 8 senzorů a příkaz k měření je odesílnám všem senzorům současně. Proto bylo nutné ke každému senzoru přidat elektrolytický kondenzátor pro pokrytí těchto proudových špiček.

Díky I^2C rozhraní může data z ultrazvukových senzorů zpracovávat přímo řídicí mikrokontrolér Mbed. Každých 100 ms je vyslána všem senzorům žádost o naměření nové hodnoty, počká se 75 ms na dokončení měření a z každého senzoru je postupně vyčtena aktuální hodnota. Schéma zapojení je na obrázku 5.10.



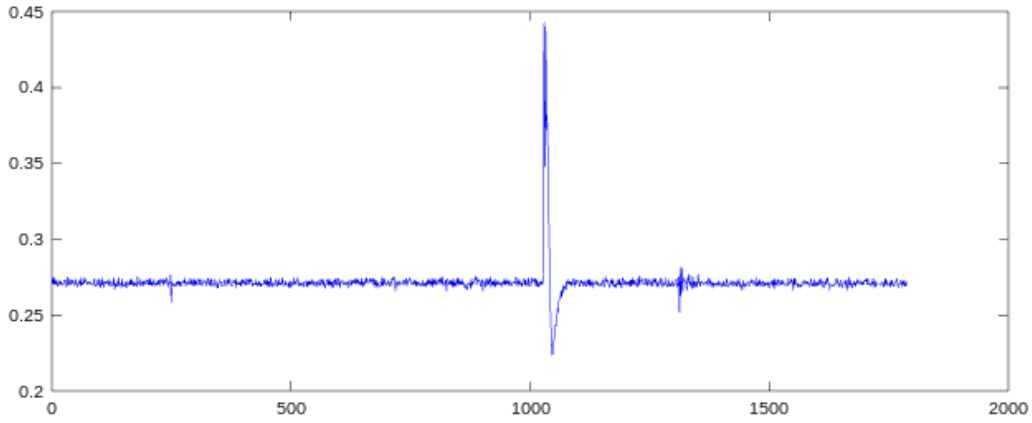
Obrázek 5.9: a) Vlákno řízení motorů b) Regulační funkce.



Obrázek 5.10: Zapojení senzorů SRF08.

5.7. Modul pro zpracování dat z pneumatických nárazníků

Nárazníky jsou důležitý senzor pro případ, že ostatní senzory a metody selhaly. Obvykle bývají mechanické a detekce je realizována kontaktním spínačem. Firma Bender Robotics vyvinula pneumatický nárazník, který je tvořený gumovým profilem s průřezem D, který je na koncích uzavřený a utěsněný. Z tohoto profilu je vyvedena silikonová hadička, která tlak přivádí na diferenciální tlakový senzor. Ten udržuje plovoucí průměr snímaného tlaku a zaznamenává prudké změny této hodnoty. Průběh lehkého nárazu je zaznamenán na obrázku 5.11. Měřena je analogová hodnota v rozsahu $< 0, 1 >$ a časový krok je 10 ms.



Obrázek 5.11: Průběh změny tlaku v nárazníku během nárazu.

6. Vysokoúrovňové systémy

V této kapitole bude popsána vysokoúrovňová část řízení robotu. Jedná se o software, který je spuštěn na počítači, kterým je robot vybaven. Tento software na základě dat ze senzorů musí být schopný v reálném čase vyhodnotit svou polohu, naplánovat cestu do zadaného cíle, vyhýbat se překážkám, zabraňovat kolizi a plnit zadané úlohy, například měřit a ukládat teplotu vzduchu nebo snímat QR kódy.

Na tomto místě je potřeba zdůraznit, že práce svým zaměřením navazuje na bakalářskou práci Luboše Vence [20], který v kapitolách 5 a 6 velmi zdařile uvádí čtenáře do problematiky frameworku ROS. Čtenář by měl mít znalosti alespoň v rozsahu této práce.

6.1. Schéma řídicího softwaru

Na obrázku 6.1 je zjednodušené schéma systémů robotu, pro rychlejší orientaci barevně zvýrazněné. Žlutě jsou celky vytvářeny zcela v rámci této diplomové práce. Šedě jsou celky, které jsou poskytovány knihovnou ROS, ale je potřeba jejich konfigurace, která bude popsána v kapitolách níže. Bíle jsou celky poskytnuté ROSem bez nutnosti do nich jakkoliv zasahovat. Modře jsou senzorické vstupy do systému, jedná se o informace posílané po sériové lince z Mbedu (odometrická data, data ze sonarů, napětí baterie…) a data z laserového dálkoměru. Zeleně je výstup systému a to příkaz, jak se má robot rychle pohybovat. Hranatý rámeček znamená *Topic*, oválný značí *Node*.

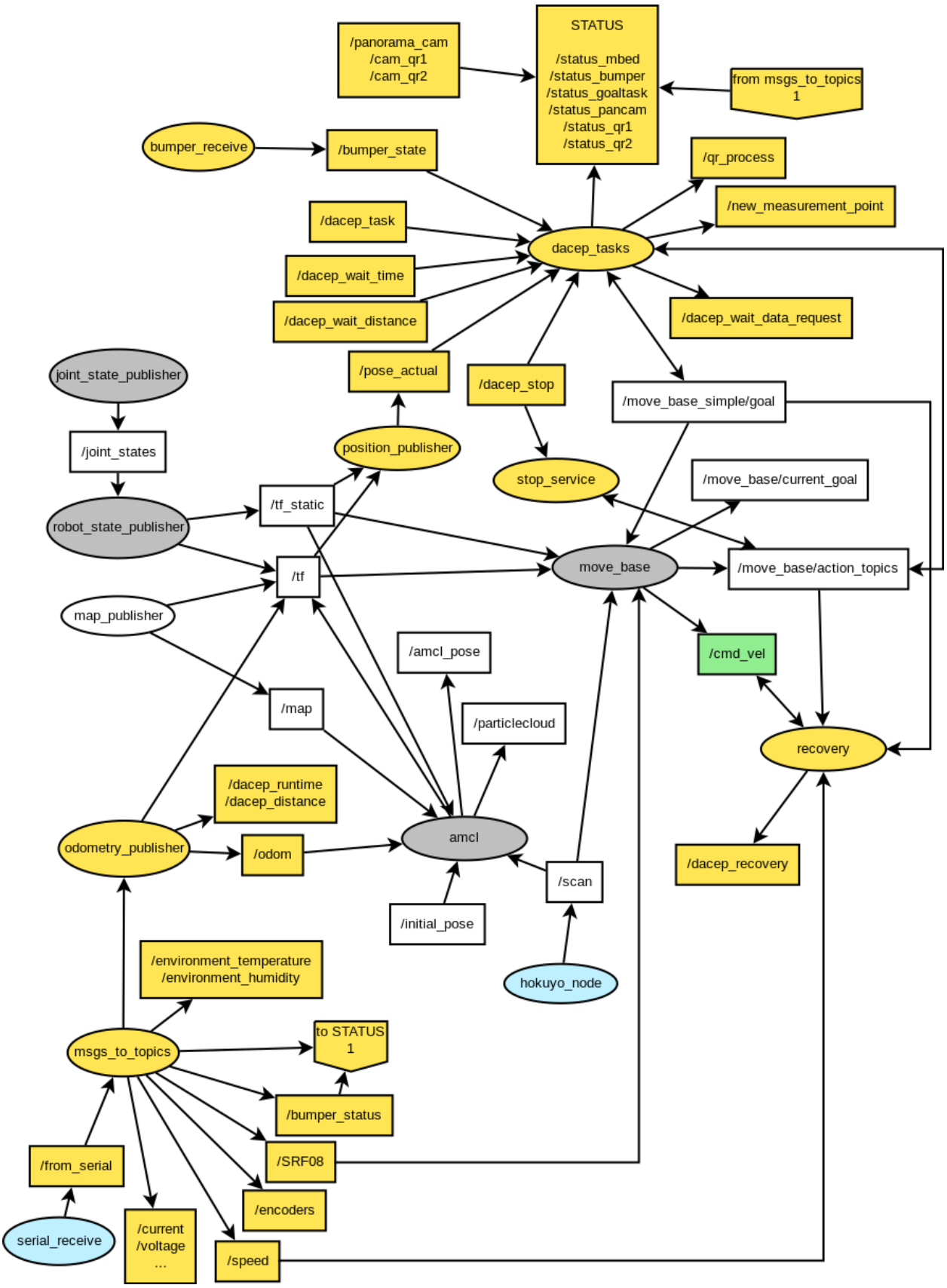
Bude následovat krátké shrnutí, k čemu je který *Node* určen. Podrobnější popis se nachází v dalších kapitolách.

Node joint_state_publisher
Node, který vezme textový soubor ve formátu URDF[1], ve kterém je robot fyzikálně definován včetně pohyblivých částí, a vytvoří příslušné */joint_states Topic*, které slouží k fyzikálnímu popisu jednotlivých pohyblivých uložení (rozsah pohybu, možné zrychlení, další omezení…). O vytvoření příslušného modelu pojednává kapitola 6.4.

Node robot_state_publisher
Z fyzikálního popisu robotu ve formátu URDF a *Topiců /joint_states* je vytvořen model robotu a TF funkce (vztah souřadných systémů jednotlivých objektů mezi sebou). O tom, co to je TF a proč je v ROSu více souřadných systémů, pojednává kapitola 6.3.

Node amcl
Pravděpodobností lokalizace na principu particle filtru. Z odometrie, mapy a laserových dat určuje pravděpodobnou pozici robotu /amcl_pose a /particlecloud. Vyžaduje zadat počáteční polohu robotu. Podrobněji se knihovně AMCL věnuje kapitola 6.5.

[1] Unified Robot Description Format



Obrázek 6.1: Zjednodušené schéma navigačního softwaru robotu.

Node move_base
Komplexní knihovna zahrnující globální plánovač, lokální plánovač a pomocné programy. Z pozice robotu v mapě a známých překážek plánuje nejméně problematickou cestu, zásadním výstupem je */cmd_vel* (command velocity), tedy příkaz k pohybu robotu ve formátu lineární rychlost x a y a rychlost rotace v ose z. Knihovně *move_base* je věnována kapitola 6.6.

Node serial_receive
Obsluha sériové komunikace. Odesílá data a předává přijatá data. V další verzi byl sloučen s *Nodem /msgs_to_topics*.

Node msgs_to_topics
Přebírá přijaté zprávy a podle typu a ID je třídí do jednotlivých *Topiců*.

Node odometry_publisher
Z dat enkodérů dopočítává odometrii a rychlost.

Node map_publisher
Z rastrové mapy vytváří mapu globálních překážek pro lokalizaci AMCL a globální plánovač.

Node position_publisher
Vytváří pozici robotu ve formátu lépe použitelného kvaternionu pro další použití. Pomocný *Node* pro /dacep_tasks. Stejného výsledku by šlo dosáhnout použitím knihovny *topic_tools transform*.

Node stop_service
Node, který zruší všechny vykonávané úkoly a robotu vydá příkaz k zastavení. Pomocný *Node* pro /dacep_tasks.

Node hokuyo_node
Node pro zpracování dat laserového dálkoměru Hokuyo.

Node dacep_tasks
Objemný *Node* navázaný na grafické prostředí, který zadává robotu příkazy (ve stylu dojeď na místo XY, zapni světlo a vyčti QR kód). Možnostem grafického prostředí a tohoto *Nodu* je věnována celá kapitola.

Node recovery
V experimentálních testech se robot občas dostával do místa, ve kterém se na dlouhou dobu zacyklil (postupná jízda dopředu-dozadu-dopředu-dozadu-...). Byl proto vytvořen *Node*, který tento stav detekuje a v takové chvíli přepne robot do definované sekvence úkolů, která pomáhá z tohoto místa vyjet.

Node status
Kontrola stavu všech systémů a signalizace případných chyb a problémů.

6.2. Grafické prostředky ROSu

Framework ROS obsahuje mnoho grafických nástrojů napsaných ve frameworku Qt, nazvaných *rqt*. Tyto nástroje mohou být používány samostatně při vývoji a ladění softwaru robotu, nebo mohou být použity jako plugin do grafického prostředí pro ovládání robotu. Protože mnoho začínajících tvůrců o těchto prostředcích neví nebo namejí úplné informace, bude těm subjektivně důležitějším věnována jedna kapitola. Seznam všech rqt nástrojů lze najít na stránkách ROS wiki [21].

Spustit rqt nástroj lze několika způsoby. Jednou možností je pomocí příkazu *rqt* spustit okno, do kterého se přes menu vkládají jednotlivé pluginy. Jinou možností je spustit plugin jako samostatný program přes standardní ROS volání:

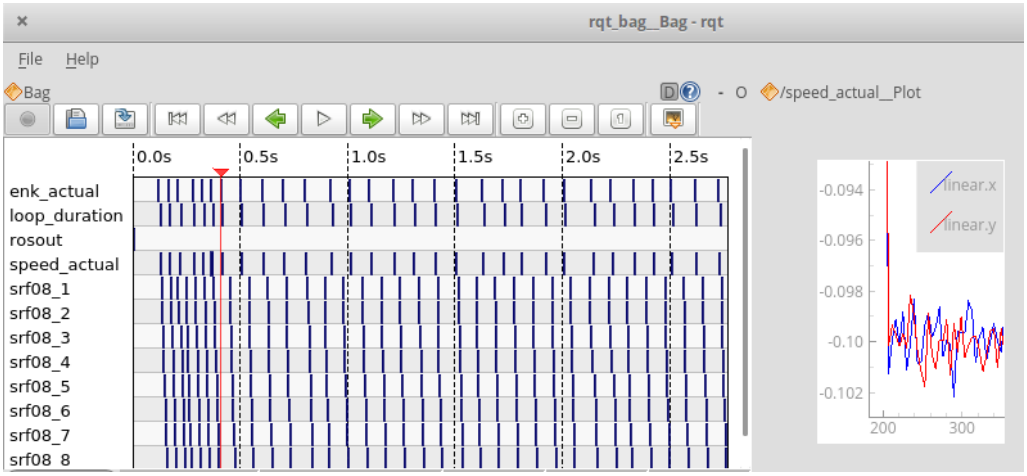
```
1  rosrun rqt_nazevProgramu rqt_nazevProgramu
```

Program rviz

První položka mírně vybočuje ze seznamu. Nejedná se jenom o samostatný plugin, ale o komplexní program pro ovládání a sledování robotu. Jedná se o 3D projekční plochu, do které je možné zanést mapu, model robotu, zobrazení dat ze senzorů. Program může zobrazovat streamovaný obraz. Pro rviz je možné psát vlastní pluginy, které jsou navázány na ROS. Této funkce bylo využito při tvorbě vlastního GUI pro robot DACEP. O tom pojednává samostatná kapitola.

Program rqt_bag

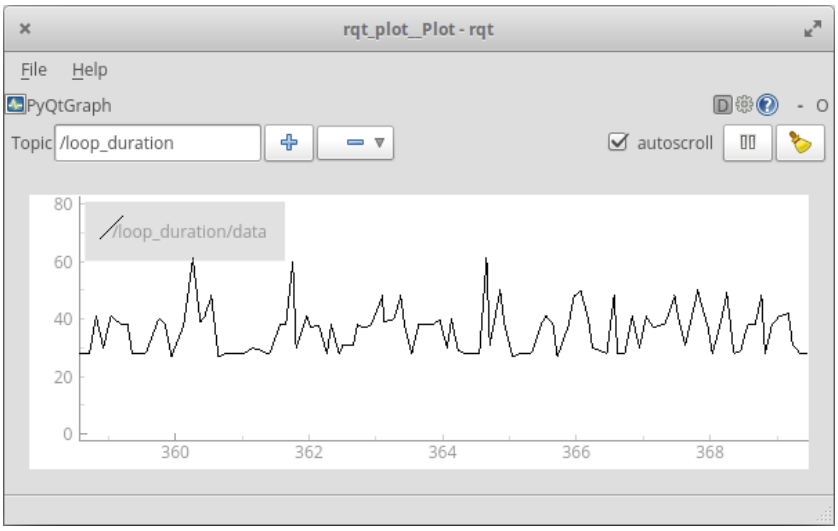
ROS přináší poměrně mocný nástroj pro ukládání dat nazvaný *rosbag*. Ten je možné používat v terminálu (spouštět například spolu s roslaunch pro automatické ukládání potřebných dat pro případnou diagnostiku), má ale také grafický nástroj *rqt_bag*. Ten má dvojí funkcionalitu. Buď slouží pro ukládání dat, kdy rqt_bag umožňuje vybrat, které z *Topiců* se mají ukládat a kam, druhou možností je nahrát dříve uložená data. Rqt_bag potom umožňuje je procházet, zobrazovat jejich hodnoty v konkrétní okamžik, vykreslovat z dat grafy, jak je ukázáno na obrázku 6.2. Umožňuje publikovat uložená data a tím simulovat situaci, která byla v době pořízení dat. Grafické rozhraní je poměrně nestabilní a především při časových posunech dojde k error stavu následovaného pádem aplikace. Svému účelu ovšem obvykle poslouží. Veškerá funkcionalita je popsána detailně na wiki stránkách ROSu [22].



Obrázek 6.2: Ukázka programu rqt_bag.

Program rqt_plot

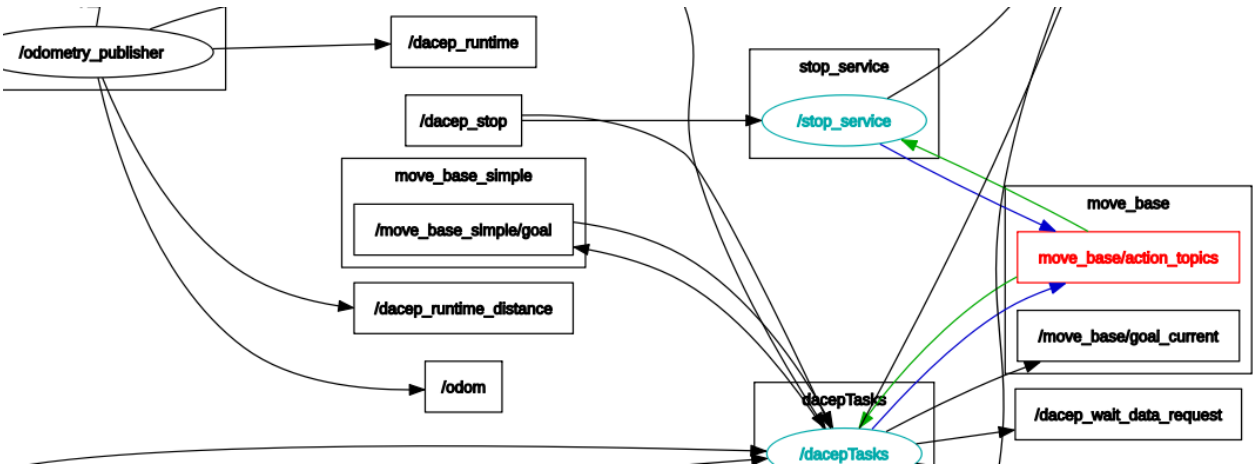
ROS umožňuje vykreslovat matematická data v reálném čase do plotu (viz obr. 6.3) a tím získat mnohem lepší přehled, co se v robotu odehrává. Nástroj je velmi nápomocný také při kalibraci a konfiguraci různých senzorů.



Obrázek 6.3: Ukázka programu rqt_plot.

Program rqt_graph

Komplexní zobrazení všech *Nodů* a *Topiců*, které jsou v danou chvíli pod ROSem spuštěny, umožňuje nástroj *rqt_graph*. Objekty je možné sdružovat do logicky souvisejících celků. Velmi dobře funguje zvýrazňování objektů - při najetí myši na objekt se barevně odliší vstupní a výstupní *Topic* a *Nody*, které s ním souvisejí. Výřez z tohoto programu je na obr. 6.4.



Obrázek 6.4: Ukázka z programu rqt_graph.

Program rqt_logger_level

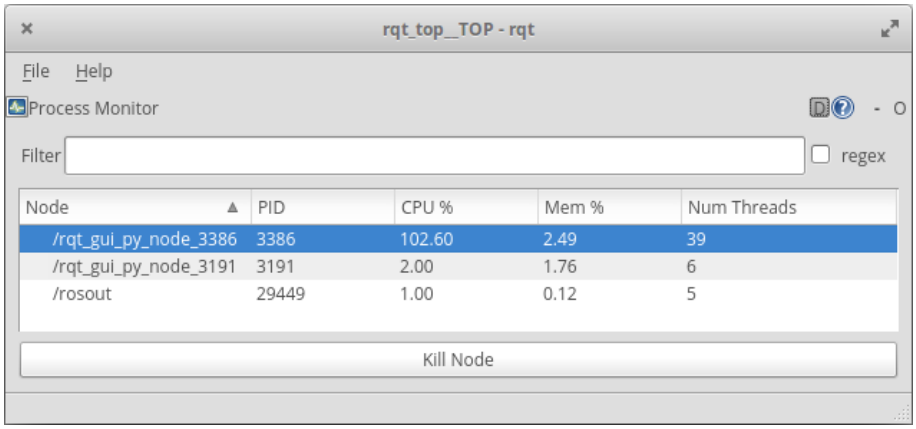
ROS má propracovaný systém logování dat. Stupeň logování je DEBUG, INFO, WARN, ERROR a FATAL. U každého *Nodu* lze volit, na jakém stupni bude logování probíhat. Pro rychlou změnu tohoto nastavení přímo za běhu programu slouží program *rqt_logger_level*. Za běhu programu lze tedy změnit loggovani jakéhokoliv *Nodu* například ze stupně WARNING na INFO nebo DEBUG. To může pomoci například při zjišťování příčiny nečeka-

ného problému, při kterém nastane potřeba detailnějších dat.

Program rqt_publisher
Pokud je potřeba zasáhnout do některého *Topicu* a poslat jednorázově vlastní data (například z důvodu testování), je možné použít program **rqt_publisher**. V něm je možné vybrat *Topic* a nastavit, jak často se mají posílat zvolená data.

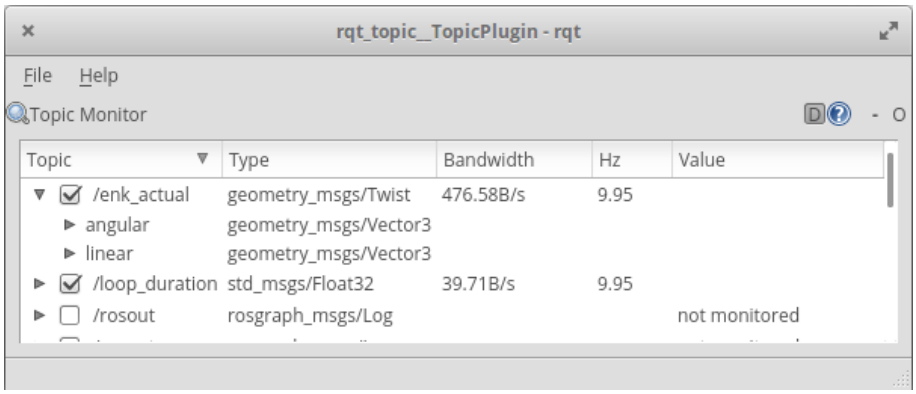
Program rqt_reconfigure
Knihovny, které vyžadují konfiguraci (například *move_base*) je možné za běhu konfigurovat programem *rqt_reconfigure* a najít tím vyhovující parametry.

Program rqt_top
ROS obsahuje systémový monitor pro sledování, jak který spuštěný proces vytěžuje hardwarové prostředky počítače (obr. 6.5).



Obrázek 6.5: Ukázka programu rqt_top.

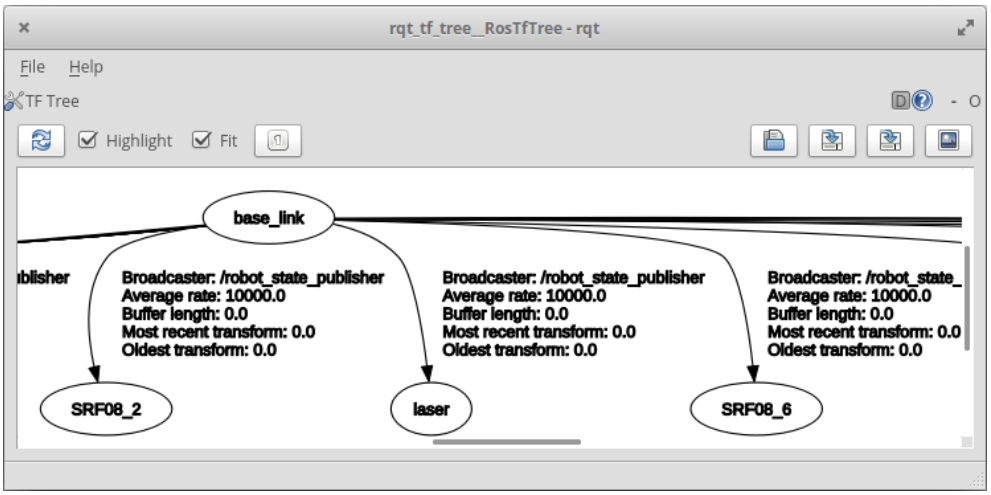
Program rqt_topic
Jeden z diagnostických nástrojů je *rqt_topic* pro zevrubnou analýzu dat mezi *Nody*. Tento nástroj umožňuje nahlížet do jednotlivých *Topiců* a zobrazit data, která se uvnitř nachází. Nástroj obsahuje sledování, jaký je datový tok vybraným *Topicem* a s jakou frekvencí jsou vytvářena nová data (obr. 6.6). Stejné informace poskytuje terminálový příkaz *rostopic bw* a *rostopic hz*, zde jsou tyto informace v přívětivější podobě a především nástroj je možné použít jako plugin do vlastního programu.



Obrázek 6.6: Ukázka programu rqt_topic.

Program rqt_tf_tree

Pro grafické zobrazení transformací TF byla vytvořena velmi užitečná grafická aplikace *rqt_tf_tree*, která zobrazuje fyzickou provázanost (závoslosti souřadných systémů) jednotlivých objektů. Aplikace je zobrazená na obr. 6.7.



Obrázek 6.7: Ukázka programu rqt_tf_tree.

Program rqt_console

ROS má nástroj pro pohodlné procházení loggovaných zpráv *rqt_console*.

Program rqt_runtime_monitor

Pro zobrazení *diagnosticsArray* zpráv[2] je určena aplikace *rqt_runtime_monitor*.

Pro další studium doporučuji tyto stránky s dokumentací:

základní rqt nástroje
http://wiki.ros.org/rqt_common_plugins

rqt nástroje pro operace s robotem
http://wiki.ros.org/rqt_robot_plugins

ROS common messages
http://wiki.ros.org/common_msgs

ROS diagnostic messages
http://wiki.ros.org/diagnostic_msgs

[2]diagnosticsArray je standardizovaný typ zprávy, která nese informaci o stavu jednotlivých komponent v ROSu.

6.3. TF Topic

Důležité pro pochopení celé této kapitoly je uvědomit si, jakým způsobem v ROSu funguje pozicování jednotlivých objektů a k čemu slouží *topic /tf*. Základem je, že každý objekt má svůj souřadný systém, který je relativní k souřadnému systému jiného objektu. ROS sám o sobě nemá žádný absolutní bod, ke kterému by vše vázal, pozice každého objektu je odkázána na souřadný systém jiného objektu a tyto závislosti tvoří hierarchický strom. Tento strom je reprezentován TF (transformační) funkcí, která určuje vztahy souřadných systémů jednotlivých objektů.

Z toho automaticky plyne, proč je */tf Topic* tak důležitý. Ten v systému slouží k reprezentaci vztahů jednotlivých souřadných systémů. Vždy, když se změní pozice dvou na sobě závisejících souřadných systémů, je tato změna uveřejněna právě v *Topicu /tf*. V praxi to funguje tak, že mapa je stanovena jako fixní (je stanoveno, že osy mapy tvoří počátek souřadného systému). Z přijímaných dat z enkodérů kol je vypočítávána odometrie, tedy odhadovaná pozice robotu. Tu využívá *Node amcl*, který zpřesňuje odhad a kompenzuje vznikající nepřesnost pomocí porovnávání snímků laserového dálkoměru s mapou a tuto pozici posílá do *Topicu /tf* jako transformaci osy robotu na souřadný systém mapy. Na souřadný systém robotu potom může být navázáno například manipulační rameno robotu, pokud jím robot disponuje. Kdykoliv zpětně mohou být dopočítány například souřadnice konce manipulačního ramene v mapě pomocí transformací mapa → robot → kloub ramena → konec ramena.

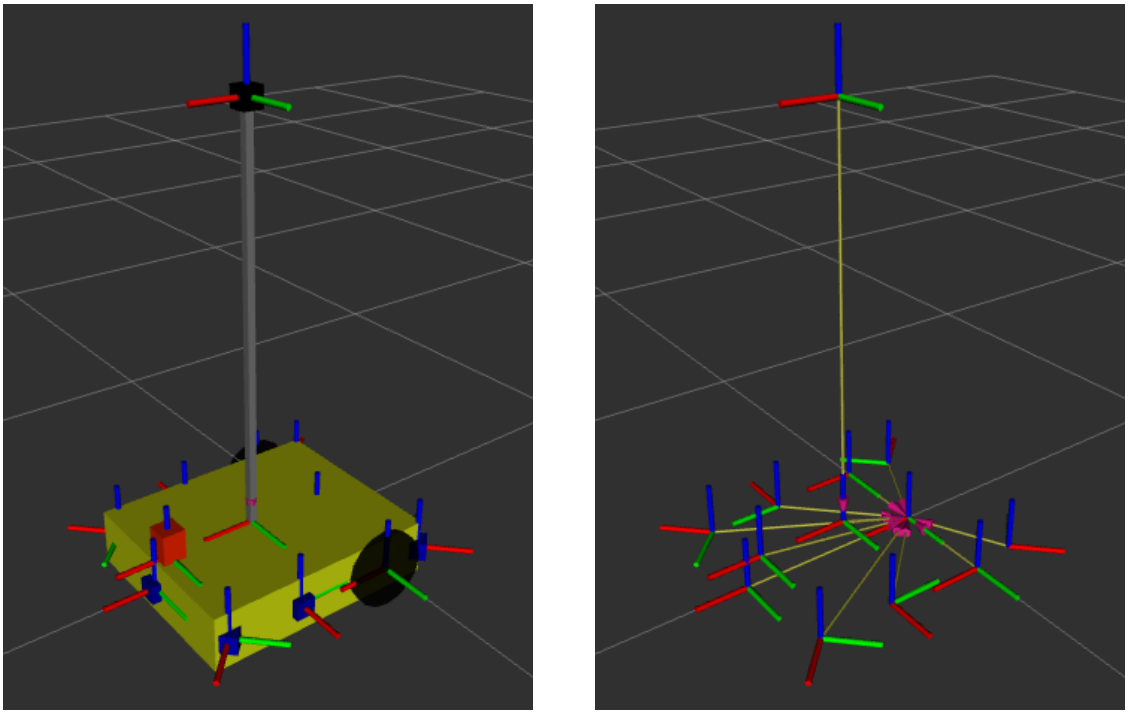
```
jan@jachym-thinkpad:~$ rostopic echo /tf_static
transforms:
  -
    header:
      seq: 0
      stamp:
        secs: 1494877761
        nsecs: 213603795
      frame_id: base_link
    child_frame_id: SRF08_4
    transform:
      translation:
        x: 0.49
        y: 0.0
        z: 0.0
      rotation:
        x: 0.0
        y: 0.0
        z: 0.0
        w: 1.0
  -
    header:
      seq: 0
      stamp:
        secs: 1494877761
        nsecs: 213613207
      frame_id: base_link
    child_frame_id: SRF08_1
    transform:
      translation:
```

Obrázek 6.8: Výpis Topicu "/tf".

Topic /tf je reprezentován datovým typem *tf2_msgs/TFMessage*. Tato zpráva obsahuje seznam jednotlivých transformací, v hlavičce je mimo jiné *frame_id* objektu, na který je vytvořena závislost, ve zprávě se potom nachází *child_frame_id* neboli frame_id transfor-

mavaného objektu. A nakonec následuje samotná hodnota transformace (posun v metrech a rotace reprezentovaná kvaterniony). Příklad části *Topicu /tf* je na obrázku 6.8. Je vidět, že sonar SRF08_4 je od osy těla robotu posunut v ose x o 49 cm a není otočen.

Na obrázku 6.9 a) je pro ilustraci zjednodušený model robotu vizualizovaný v programu rviz, na obrázku 6.9 b) je ten samý pohled, jen model je neviditelný. Je patrné, že každý objekt má svůj souřadný systém a každý je navázán svou polohou na střed osy robotu (závislost/transformace objektů je znázorněna žlutofialovými šipkami).

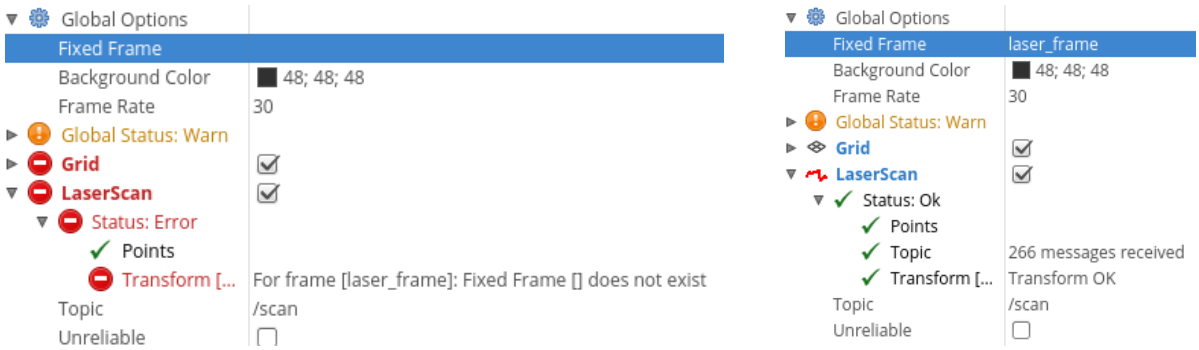


Obrázek 6.9: a) model robotu b) souřadné systémy jednotlivých objektů.

6.3.1. Chyba fixed_frame[ ] does not exist

Častá začátečnická chyba je, když se zapojí samotný laserový dálkoměr a uživatel si chce vizualizovat data v programu rviz. V tom by pochopitelně neměl být žádný problém. Vše zapojí dobře a přesto se setká jen s chybovým stavem, že fixed_frame neexistuje (obrázek 6.10 a). Rviz totiž nemá informaci o vztahu mezi (neexistující) mapou a laserovými daty. Ve chvíli, kdy se jako fixed frame zvolí frame_id LIDARu (v mém případě /laser_frame), je vše v pořádku (obrázek 6.10 b) - střed laserových dat je zvolen jako počátek souřadného systému.

Pozor, nezadává se název *Topicu*, ale frame_id, to jsou dvě odlišné věci. Frame_id je součást hlavičky *Topicu*, tedy identifikátor *Topicu* používaný právě TF funkcí. Tento údaj lze získat například z terminálu příkazem *rostopic echo /nazev_topicu*, jak je na obr. 6.11, nebo přes grafickou aplikaci *rqt_topic*. Grafickým nástrojům ROSu se věnovala předchozí kapitola.



Obrázek 6.10: a) Špatně a b) dobře nastavený program Rviz.

```
jan@jachym-thinkpad:~$ rostopic echo /scan
header:
  seq: 12
  stamp:
    secs: 1494536632
    nsecs:  73954496
  frame_id: laser_frame
angle_min: -3.12413907051
angle_max: 3.14159274101
angle_increment: 0.0174532923847
time_increment: 4.69398514724e-07
scan_time: 0.000168514074176
range_min: 0.15000000596
range_max: 8.0
ranges: [inf, 2.3580000400543213, 2.357000111
1501465, 2.3399999141693115, 2.430000066757
```

Obrázek 6.11: Výpis topicu "/scan".

6.4. Parametry tvaru robotu, soubor URDF

Pro definování částí robotu, jejich tvaru, fyzikálních vlastností a jejich vzájemné poloze slouží soubor URDF[3], využívající XML formát reprezentace. API slibuje kompletní zpětnou kompatibilitu pro všechny budoucí verze. Protože se jedná o XML, upozorňuji, že elementy jsou párové a musí být uzavřeny.

Ve výpisu 6.1 je ukázka, jak může nejjednodušší URDF soubor vypadat. V příloze C jsou bez popisu přehledně vypsány běžně používané elementy pro vytvoření URDF modelu. XML začíná kořenovým elementem <*robot*>, jako povinný atribut je název celé sestavy *name*.

Pod ním jsou možné elementy: <*link*>, <*sensor*>, <*joint*> a <*transmission*>. Při jednoduchém návrhu si uživatel vystačí s <*link*> a <*joint*>[4], proto se budu věnovat pouze těmto dvěma.

[3]Unified Robot Description Format

[4]Element <*sensor*> byl definován, ale nikdy později nenašel uplatnění, <*transmission*> určuje vztah mezi aktuátorem a vazbou, umožňuje tedy definovat například fyzikální vlastnosti převodovky a řízeného objektu a tím umožňuje lepší (prediktivní) řízení. Více informací o balíku řízení na [25].

Pro prohloubení znalostí se odkazuji na oficiální wiki [23] a dva vyčerpávající tutoriály na Wiki ROSu:

Tutoriál pro vytvoření URDF modelu
Building a Visual Robot Model with URDF from Scratch.

Tutorial, jak přidat fyzikální vlastnsti do URDF modelu
Adding Physical and Collision Properties to a URDF Model.

Definování fyzikálních vlastností najde plné uplatnění při použití knihovny *MoveIt*, která pracuje s 3D prostorem a nabízí mnohem víc možností plánování a řízení. Při použití základní knihovny *Move_base* je důležité jen definování vzhledu pro pozdější vizualizaci, plánovač pracuje pouze se zjednodušeným 2D půdorysem robotu, který je definován na jiném místě.

Výpis 6.1: Ukázka vytvoření modelu

```
<?xml version="1.0"?>
<robot name="Dacep">
<!-- define base link object -->
    <link name="base_link">
        <visual>
            <geometry>
                <box size="0.62 0.51 0.15"/>
            </geometry>
            <origin rpy="0 0 0" xyz="0.18 0 0.0"/>
            <material name="base">
                <color rgba="0.88 0.93 0.059 1.0"/>
            </material>
        </visual>
    </link>

<!-- define laser object -->
    <link name="laser">
        <visual>
            <geometry>
                <box size="0.06 0.06 0.08"/>
            </geometry>
            <origin rpy="0 0 0" xyz="0 0 0.04"/>
            <material name="base" />
        </visual>
    </link>

<!-- define joint between base and laser-->
    <joint name="base_to_laser" type="fixed">
        <parent link="base_link"/>
        <child link="laser"/>
        <origin xyz="0.45 0 0.075"/>
    </joint>
</robot>
```

6.4.1. Element link

Element *link* popisuje tělo objektu, obsahuje volitelné elementy <*visual*>, <*collision*> a <*inertial*>. Ty definují parametry: vzhled, kolizní vlastnosti a moment setrvačnosti. Oficiální dokumentace je na stránkách ROSu [24].

<**link name="jmeno"** >

<**visual name="jmeno"** >

Popis tvaru tělesa. Atribut *name* je nepovinný. Pokud existuje jednou pojmenovaný a definovaný element, u dalších výskytů tohoto elementu tohoto jména se atributy uvnitř <*visual*> zkopírují. Tento princip platí u všech elementů s atributem name (ukázka ve výpisu 6.1, kde se na řádku 22 přebírá materiál.

<**origin xyz="x y z" rpy="r p y"** />

Posune objekt o určenou vzdálenost od osy. Když je například osa robotu posunuta dozadu, musí to být provedeno posunem objektu vpřed od osy (osa je vždy v počátku a určuje se umístění objektu). XYZ určuje lineární posun, RPY rotaci ve třech osách.

<**geometry** > (Povinný)

Určuje tvar objektu. Může se jednat o elementární objekt, nebo je podporován 3D objekt ve formátu Collada.

<**box size="x y z"** />
<**cylinder length="value" radius="value"** />
<**sphere radius="value"** />
<**mesh filename="/slozka/soubor.DAE"** />

<**material name="jmeno"** >

Specifikuje vzhled (barvu a průhlednost) objektu.

<**color rgba="R G B A"** />

RGBA definuje barvu ve formátu RGB (červená zelená modrá) v rozsahu [0,1] a průhlednost v rozsahu [0,1].

<**texture filename="/slozka/soubor.DAE"** />

<**collision name="jmeno"** >

Specifikuje kolizní tvar objektu. Ten bývá běžně stejný jako <*geometry*>, ale pro lepší optimalizaci je možné tvar zjednoduštit.

<**origin xyz="x y z" rpy="r p y"** />
<**geometry** >

<**inertial** >

Specifikace momentů setrvačnosti objektu. Ty jsou důležité pro simulaci. Pokud je potřeba jen vizuální model robotu, není nutné se tímto elementem zabývat. Buď je možné dopočítat hodnoty ručně podle známých vztahů, nebo je možné je získat z 3D programů, například programu MeshLab. Element <*origin*> určuje polohu těžiště od počátku souřadného systému objektu (v tom je rozdíl oproti <*origin*>u elementu <*visual*>. <*Mass*> určuje váhu objektu [kg], <*inertia*> momenty setrvačnosti z 3x3 matice setrvačných momentů.

<**origin xyz="x y z" rpy="r p y"** />
<**mass value="váha"** /> (povinné)
<**inertia ixx="xx" ixy="xy" ixz="xz" iyy="yy" iyz="yz" izz="zz"** />
(povinné)

6.4.2. Element joint

Element *joint* popisuje vazbu mezi dvěma objekty ve formátu *link*. Atribut *name* je unikátní pojmenování vazby. *Type* určuje typ vazby. Ta může mít mnoho podob, nejjednodušíš je fixní (*fixed*). Pro ostatní možnosti čerpejte z manuálu [26]. Jsou povinné elementy <*parent*> a <*child*>, určující rodiče a potomek, které stanovují, že souřadný systém potomka je odvozený od souřadného systému rodiče. Identifikace je pomocí atributu *name*. Element <*origin*> potom stanovuje posun souřadného systému mezi rodičem a potomkem. Ostatní parametry jsou určeny jen pro pohyblivé spoje, nebudou popisovány a odkazuji se na specifikaci uvedenou v odkazu výše.

<**joint name="jmeno" type="typ"** >

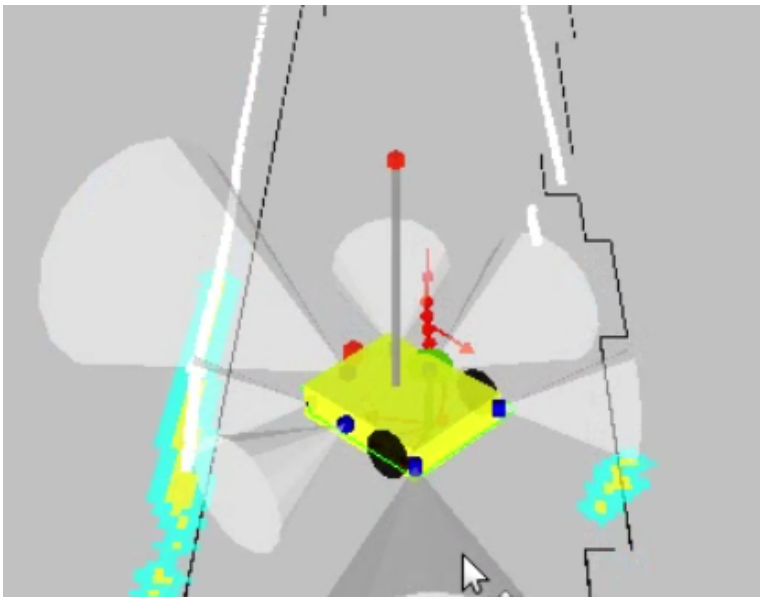
<**origin xyz="x y z" rpy="r p y"** />
<**parent link="name"** />
<**child link="name"** />

Výpis 6.2: Načtění modelu v roslaunch

```
1  <launch>
2      <!-- Read robot model -->
3      <arg name="model"
           default="/home/UZIVATEL/catkin_ws/src/model/urdf/JMENO.urdf" />
4      <param name="robot_description" textfile="$(arg model)" />
5
6      <!-- Run joint publisher -->
7      <node name="joint_state_publisher" pkg="joint_state_publisher"
           type="joint_state_publisher">
8          <param name="rate" value="10" />
9      </node>
10
11     <!-- Run robot state publisher -->
12     <node name="robot_state_publisher" pkg="robot_state_publisher"
           type="state_publisher">
13         <param name="publish_frequency" value="10.0" />
14     </node>
15 </launch>
```

Ukázka načtení URDF modelu v souboru *roslaunch* je ve výpisu 6.2.

Na obrázku 6.12 je ukázka, jak může vypadat definovaný zjednodušený model robotu. Žlutě je znázorněné tělo, modře sonary, červeně kamera a LIDAR. Černě kola, šedě tyč kamery. Na obrázku je vidět šedá mapa s černými liniemi zdí, žlutozeleně jsou znázorněny lokální překážky, bíle jsou data z LIDARu, světle šedé válce znázorňují data ze sonarů, červené šipky značí odometrii na posledním ujetém metru.



Obrázek 6.12: Ukázka modelu robotu.

6.5. Lokalizační knihovna AMCL

Knihovna AMCL (Adaptive Monte Carlo Localization) slouží k lokalizaci robotu za použití particle filtru. Adaptivita spočívá v nekonstantním mmnožství vzorků particle filtru. Knihovna je napsána podle publikace Propabilistic Robotics[6]. Konfigurační parametry jsou v této publikaci velmi obsáhle vysvětleny.

Vstupem pro Amcl je *Topic /scan*, */map*, odometrická data (podle *frame_id* specifikovaného v konfiguračním souboru) a */tf*. *Topic /map* předává mapu okolí, */scan* data z laserového scanneru, odometrie je potřeba k dopočtení přibližné polohy. Knihovna AMCL musí mít informaci o počáteční poloze robotu. Výstupem je */amcl_pose*, tedy nejpravděpodobnější poloha robotu a */particlecloud*, definující seznam vzorků reprezentujících rozdělení pravděpodobnosti. Do *Topicu /tf* je předávána nejpravděpodobnější poloha robotu.

Konfigurace používaná v robotu DACEP je v příloze D.

6.6. Navigační knihovna Move Base

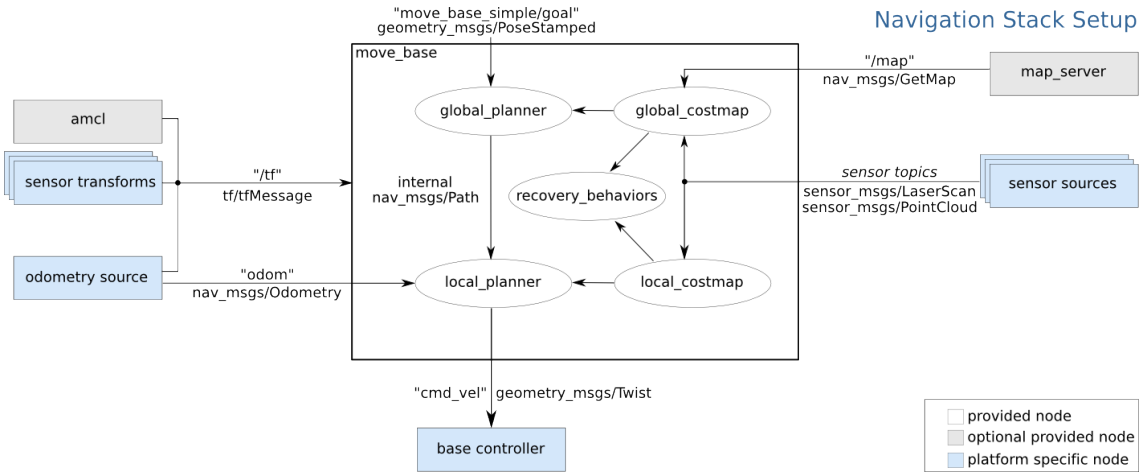
K navigaci slouží v ROSu knihovna *navigation_stack*. Schéma navigačního jádra nazvaného *move_base* je zobrazeno na obrázku 6.13. *Global_planner*, *local_planner* a *recovery_behaviors* jsou programy ve formě pluginů, které jsou upravitelné nebo zcela nahraditelné za dodržení navrženého rozhraní. Do *move base* vstupuje definování požadovaného cíle */move_base_simple/goal*, kam se má robot dostat, a vystupuje příkaz k rychlosti robotu */cmd_vel*. Když se robot dostane do situace, ze které není schopný naplánovat cestu, přechází do režimu čištění mapy (*recovery_behavior*) ve snaze zbavit se falešně zaznamenaných překážek a také překážek, které již nejsou platné. Dále *move base* pracuje s těmito vstupy:

Topic /tf
Vztahy souřadnicových systémů, a tedy i poloha robotu v mapě (pomocí Amcl) a poloha jednotlivých senzorů na robotu. To je důležité k přiřazení směrového údaje k datům z jednotlivých senzorů. Hodnota senzoru bývá typicky vzdálenost. Překážkou v mapě se stává až po přepočtu, kde se nachází senzor, který tento údaj naměřil a jakým směrem je orientován.

Topic odometrie
Pro lokální plánovač může být výhodnější pracovat přímo s odometrickými daty (která jsou spojitá na rozdíl od Amcl, kde dochází ke skokům pozice).

Topic /map
Rastrová mapa známého prostředí. Bývá poskytována na žádost jako service.

Topic senzorických dat
Naměřená data senzorů, která po spojení s */tf* vytváří mapu známých překážek.



Obrázek 6.13: Navigační jádro frameworku ROS.

Globální plánovač vytváří podle zadaných parametrů možnou cestu vedoucí do zadaného bodu za využití mapy globálních překážek, lokální plánovač potom vyhledává nejbezpečnější cestu v definovaném okolí robotu podle mnoha hledisek, mezi která patří

respektování globální cesty, zabraňování kolizi, plynulost jízdy. Lokální plánovač určuje, jakým směrem a jakou rychlostí má robot jet.

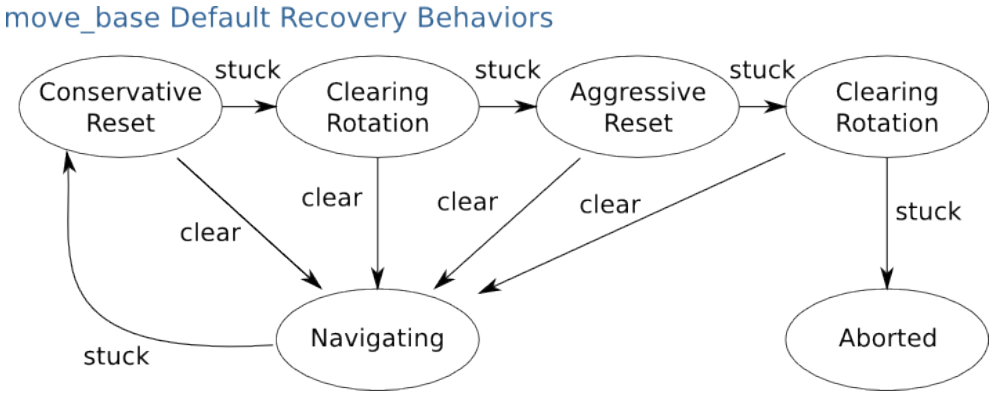
Ukázka spuštění *Nodu move base* v robotu DACEP a konfigurační soubory pro globální a lokální navigaci se nachází v příloze E.

Globální navigace je v ROSu řešena metodou potenciálového pole. Tato metoda je rychlá a pokud existuje řešení, touto metodou je nalezeno, ale obsahuje mnoho kompromisů, především, že se robot v prostorách typu chodba nepohybuje středem, ale po kraji. To je u robotu určeného do uliček datového centra problém. Bylo vyzkoušeno použít existující knihovnu využívající metodu na bázi zobecněného voronoiova diagramu, ale tato je nestabilní a hardwarově mimořádně náročná. Vhodné řešení by bylo vzít stávající metodu potenciálového pole a rozšířit ji podle práce doc. Věcheta [27].

Lokální navigace funguje bezproblémově. Základní knihovnu *base_local_planner* je ale možné nahradit knihovnou *teb_local_planner*, která poskytuje ještě podstateně více konfiguračních možností a podporuje jako jedna z mála i ackermanovu geometrii čtyřkolového podvozku (automobilový podvozek).

6.6.1. Recovery behavior

Robot pro hledání cesty využívá dvě mapy: mapu lokálních překážek a mapu globálních překážek. Tyto jsou využívány lokálním, respektive globálním plánovačem. Mapa lokálních překážek bývá v počátku prázdná, mapa globálních překážek přebírá zadanou mapu známého prostředí. Robot si při svém pohybu do těchto map zanáší informace o nalezených překážkách, přičemž senzory, které se berou v potaz při vytváření globální i lokální mapy jsou definovány a mohou se lišit. V robotu DACEP globální mapa využívá jen laserový dálkoměr, lokální potom i ultrazvuky a nárazníky.



Obrázek 6.14: Režimy knihovny *move base*.

Chování robotu je v *move base* definováno vzorcem, který je vykreslen na obrázku 6.14. Robot se většinu doby nachází ve stavu *Navigating*, tedy řízení je ponecháno zvoleným navigačním programům. Když uvízne v místě, na kterém setrvá definovanou dobu, přechází do režimu *Conservative Reset*. V základním nastavení tento režim využívá program *clear_costmap_recovery*, kdy se mapa známých překážek v okolí robotu o definovaném poloměru nahradí zadanou známou mapou z *Topicu /map*. Rizika u tohoto jednoduchého způsobu jsou zjevná. V těsné blízkosti robotu se potom mohou nacházet překážky nedetekovatelné žádným senzorem (například kvůli příliš blízké poloze). Pokud toto řešení

nebylo postačující, přechází robot do stavu *Clearing Rotation*. To spočívá v otočce o 360° kolem své osy a hledání možné cesty z uzavřeného místa, na kterém se robot nachází. Pokud nemá robot osu kol v geometrickém těžišti, jedná se o velmi hazardní krok, který doporučuji v parametrech zakázat. Pokud ani tato metoda nebyla účinná, následuje stav *Aggresive Reset*. Ten je v základu nastaven stejně jako *Conservative Reset*, ale jako všechny metody zde popisované, může být změněn. Následuje ještě jeden pokus o *Clearing Rotation* a následně zrušení zadaného cíle oznámené v příslušném *Topicu*.

Žádná z uvedených metod nepřinesla při testech robotu DACEP odpovídající výsledky. Z důvodu časového tlaku bylo přistoupeno k tvorbě *Nodu*, který celý systém obchází. Jedná se o *Node recovery*. Ten na základě rychlosti a stanoveného cíle detekuje uvíznutí robotu na jednom místě a v takovém případě je řízení nuceně přepnuto na krátkou sekvenci, kdy robot jede vzad. Do budoucna by bylo lepší tento program napsat jako příslušný plugin do *recovery behavior* se sofistikovanější únikovou sekvencí.

6.7. Node dacep_tasks

Pro potřeby robotu DACEP byl vytvořen software určený k plnění navržených úkolů, mezi které patří systematické měření teploty a vlhkosti vzduchu v zadaném úseku a detekce QR kódu v obraze. Hlavní schéma tohoto softwaru je na obrázku 6.15. Oválně jsou ohraničeny *Nody*, hranaté jsou *Topicy*. Žlutě jsou *Nody* vytvořené v rámci diplomové práce potřebné pro povelování robotu. Červeně je ROS *Service*, na který může být napojen další program závislý na prováděných činnostech robotu. Zeleně jsou data navigačního programu. Modře jsou vstupní data ze senzorů.

Node rviz_dacep_plugin a map_marker

Tyto *Nody* jsou spuštěny na vzdáleném počítači, který slouží pro vizualizaci a povelování. Možnosti pluginu *rviz_dacep_plugin* jsou popsány v následující kapitole. *Node map_marker* slouží k zanášení interaktivních značek do mapy vizualizačního programu. Tyto značkou mohou být buď aktivní, které umožňují kliknutím na vybranou značku plnit robotu předdefinovanou sekvenci úkolů (například jet na zadané místo a v definovaných intervalech měřit teplotu). Případně mohou být pasivní, kdy není možné na ně kliknout myší. Tyto značky slouží k zaznamenávání neměřené teploty a vlhkosti v mapě.

Node dacep_tasks

Node je spuštěný přímo na robotu s slouží k přijímání rozkazů ze vzdáleného počítače. Komunikuje s ROS *servicem*, na který může být navázán jiný program závislý na aktuálním stavu robotu. Když je robotu vydán pokyn k měření teploty, jsou vypočítány body ležící mezi aktuální a cílovou pozicí robotu, na kterých má být změřena teplota. Těmito body je naplněn zásobník a robot postupně na všech souřadnicích zastaví, počká definovanou dobu na ustálení měření a změřené hodnoty zanese do mapy pomocí *Nodu map_marker*.

Topic /dacep_stop

Příkaz k okamžitému zastavení robotu a zrušení všech činností.

Topic /dacep_task
Identifikace prováděnné činnosti. Hodnota 0 znamená normální činnost a navigaci, 1 je systematické měření teploty, 2 znamená detekci QR kódu.

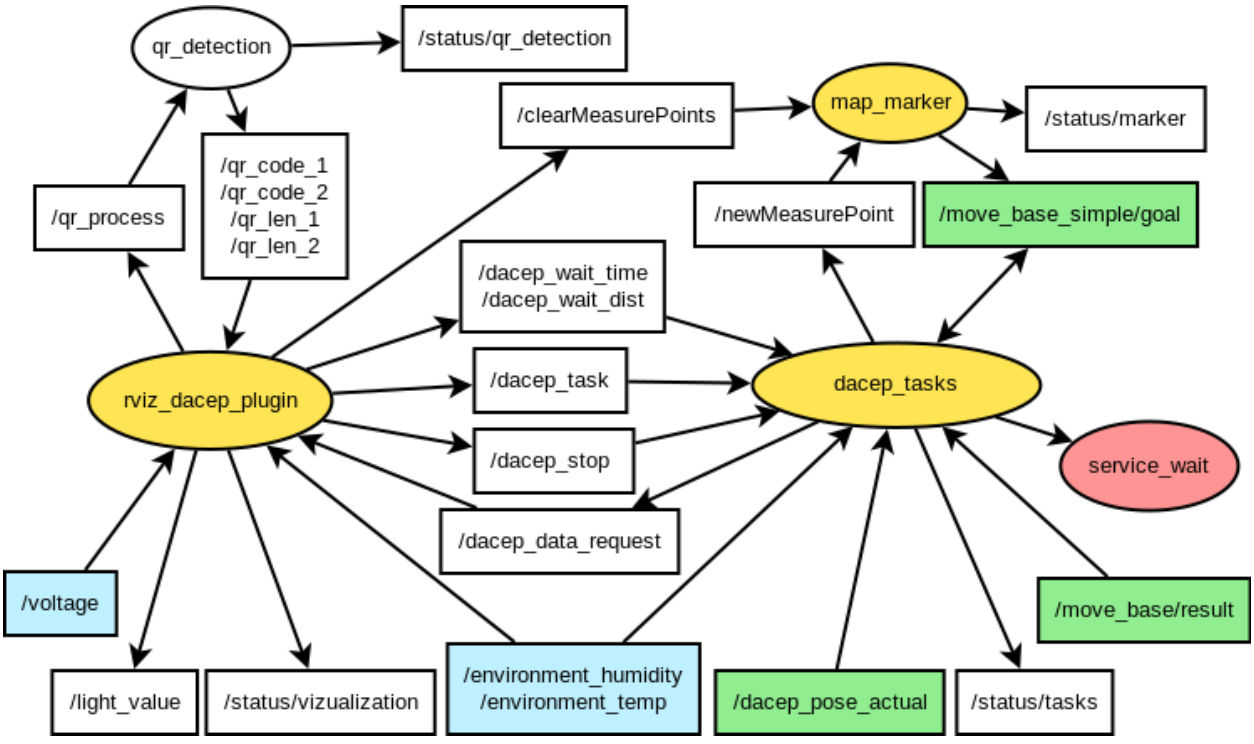
Topic /dacep_wait_time a dacep_wait_dist
Konfigurace pro měření teploty. Hodnoty definují, jaká má být při systematickém měření teploty vzdálenost mezi dvěma měřeními a jak dlouho má robot čekat na ustálení měření.

Topic /dacep_data_request
Když je spuštěn *Node dacep_tasks*, tímto *Topicem* jsou vyžádány údaje *dacep_wait_time* a *dacep_wait_dist*.

Topic /qr_process
Příkaz, že má být spuštěn program pro detekci QR kódu v obrazu.

Topic /qr_code, qr_len
Detekované QR kódy jsou pro další zpracování, případně zobrazení přenášemi těmito *Topicy*.

Topic /clearMeasurePoints
Příkaz k vymazání dat z předchozího měření teploty (všech hodnot v mapě). K tomuto kroku je přistoupeno při vypnutí systematického měření teploty a opětovném zapnutí z důvodu udržení přehlednosti dat zanesených v mapě.



Obrázek 6.15: Struktura softwaru pro zadávání úkolů robotu.

Topic /newMeasurePoint
Zanesení nového údaje s teplotou do mapy.

Topic /move_base/result
Stav navigačního softwaru. Odlišuje se například stav "navigace", "plánování cesty"nebo chybové stavy "na zadanou pozici nebyla nalezena cesta, navigace přerušena".

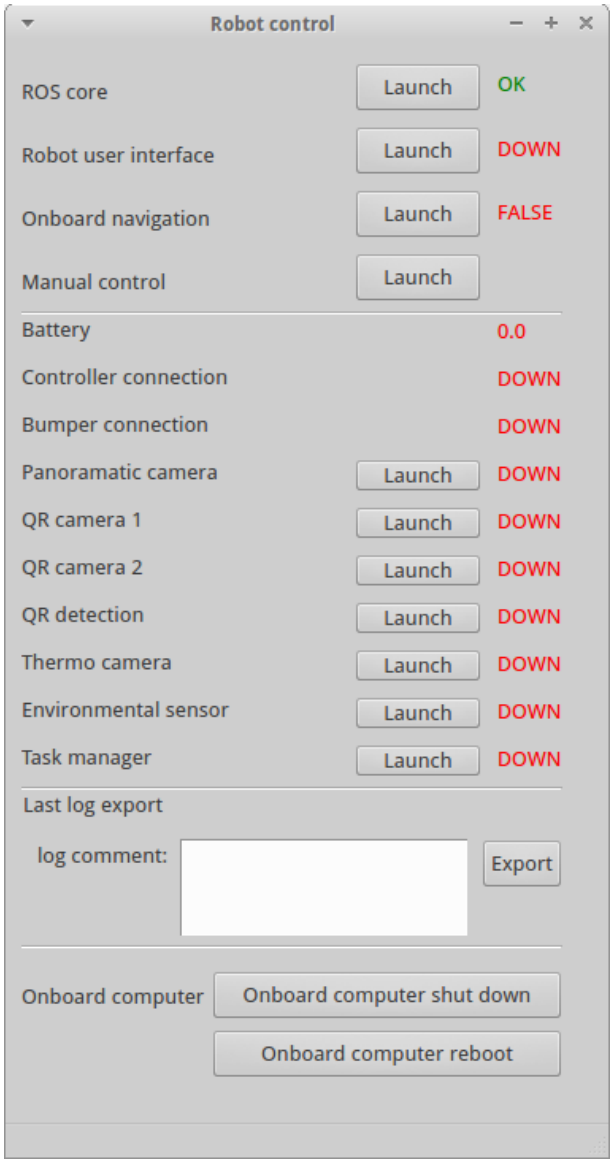
Topic /light_value
Zadání hodnoty osvětlení pro řízení LED panelů kolem kamer.

Topic /status/...
Informace pro diagnostiku o stavu daného systému.

7. Grafické uživatelské rozhraní

Řídit robot musí být schopný kdokoliv s technickým vzděláním po úvodním zaškolení. Z toho plyne požadavek na srozumitelné grafické prostředí.

K tomutom účelu byl v jazyku python s využitím grafického frameworku *Qt* vytvořen program *Robot Control* zobrazený na obrázku 7.1. Spuštění tohoto programu je možné až po zapnutí robotu a připojení na WiFi síť robotu. Program během zapnutí vytváří virtuální terminál, kam vypisuje podrobnější informace. Dále se pokouší pomocí SSH připojit na počítač instalovaný na robotu. Když je úspěšný, skript v *shellu* vymaže nejstarší uložená data z posledních jízd, aby nebyla překročena kapacita disku a následně spustí *Roscore*. Úspěšné připojení na vzdálený počítač robotu a spuštěný *Roscore* je signalizován zeleným *OK* v příslušné kolonce programu. Uživatel má potom možnost spustit grafické prostředí pro ovládání robotu, které je na obrázku 7.2. Dále je možné na robotu spustit celý systém autonomní navigace, případě manuální řízení robotu pomocí klávesnice.

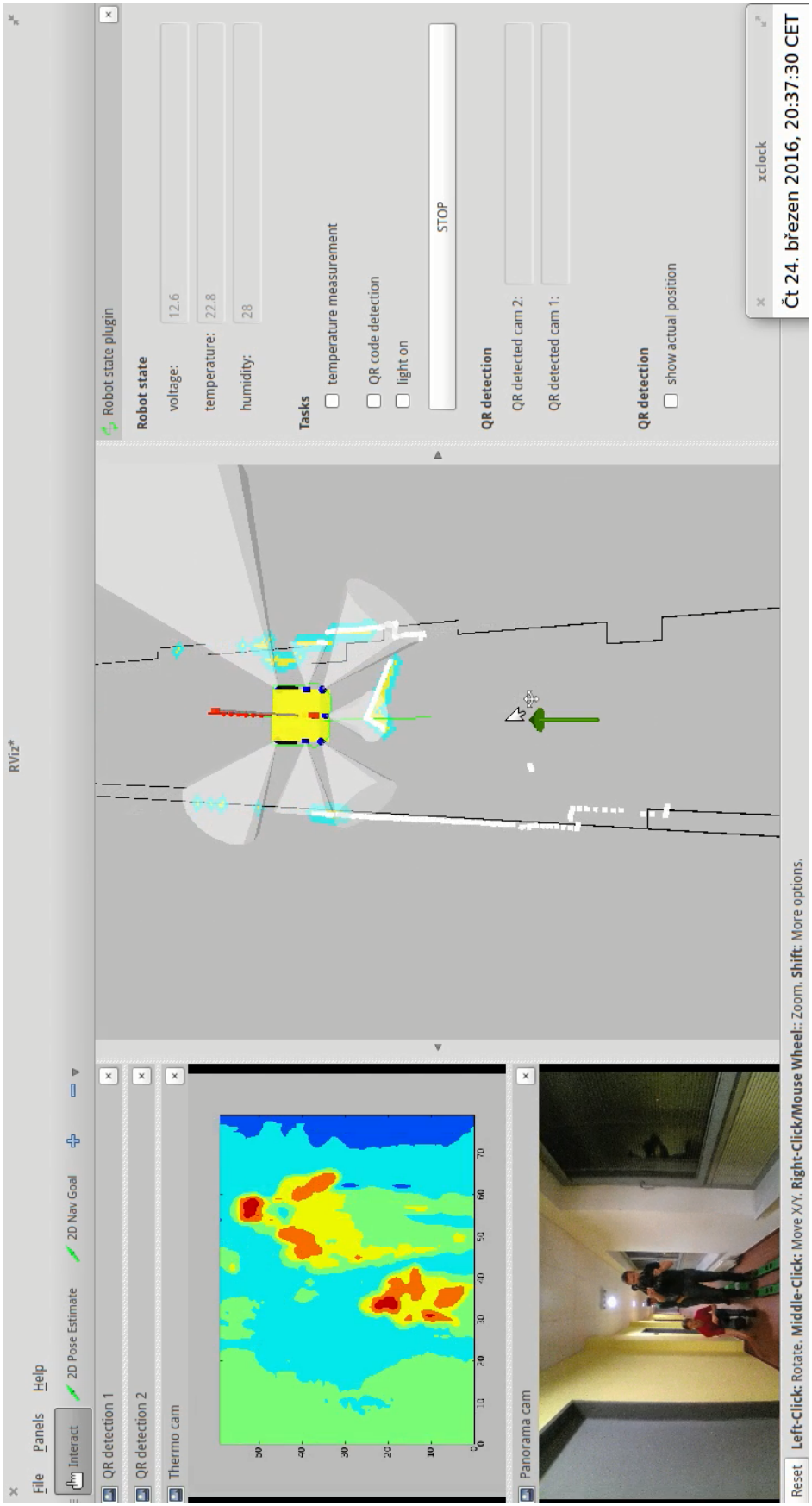


Obrázek 7.1: Výchozí ovládání grafického rozhraní.

Program po připojení a zapnutí navigace kontroluje stav jednotlivých komponent, tedy stav baterie, připojené periferie a spuštěné subsystémy. Všechny červené značky *DOWN* by se měly změnit na zelené *OK*. Pokud se tak nestane, je potřeba zkontrolovat fyzické přípojení problémových komponent a jejich následný restart.

V případě chyby umožnuje program komplexní údaje z poslední jízdy exportovat do příslušné složky i s komentářem pro případnou pozdější analýzu problému (k logování dat je použit nástroj *rosbag*).

Nakonec je možné počítač na robotu na dálku softwarově vypnout nebo restartovat v případě přetrvávajících problémů.



Obrázek 7.2: Ukázka grafického programu pro řízení robotu.

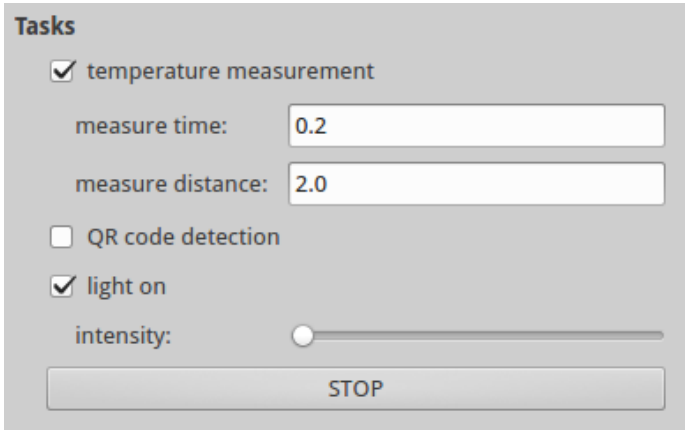
Grafické prostředí pro obsluhu robotu je tvořeno programem *rviz*, pro který byl vytvořen v jazyku C++ plugin, obsluhující periferie robotu (na obrázku 7.2 panel vpravo). Takto upravený program rviz umožňuje stanovit robotu počáteční pozici ve známé mapě pro případ, že se robot při startu nachází mimo dokovací stanici. Robotu je možné zadat v mapě cíl, na který má dojet. Je možné vytvořit interaktivní značky, na které jsou navázány konkrétní úkoly (tedy například jedním klikem do mapy poslat robot na určené místo v mapě s danou orientací a například vyčíst QR kódy na daném místě).

Robotu je možné jako úlohu zadat měření teploty a vlhkosti zaškrtnutím volby *temperature measurement*. Po zaškrtnutí je zobrazena volba, po jaké vzdálenosti má robot pro měření zastavovat a jakou dobu se má čekat na ustálení měření (obrázek 7.3). Změřená teplota a vlkost je zanesena značkou do mapy a případně uložena do databáze. Robot potom jede po zadané trase a automaticky v žádaném intervalu zastavuje a provádí měření.

Robot pomocí dvou kamer umístěných na boku je schopný detekovat QR kódy. Obraz kamery s vyznačenými QR kódy je možné zobrazit v záložce QR detection 1, respektive QR detection 2. Detekované údaje spolu s pozicí robotu je možné opět zanést do databáze, případně s nimi jinak pracovat. Program pro detekci QR kódů v obraze byl vytvářen mimo tuto diplomovou práci.

Robot ke svému pohybu nepotřebuje osvětlení - lokalizaci a navigaci zabezpečují laserový dálkoměr spolu s ultrazvuky. Robot se tedy běžně může pohybovat ve tmavých prostorách a jediné, k čemu osvětlení potřebuje je vyčítání QR kódů pomocí kamery. Pro tento účel je kolem kamer několik LED osvětlovacích panelů, které je možné sofwarově ovládat pomocí jezdce pod volbou *light on*.

Tlačítko *STOP* ruší všechny naplánované činnosti robotu a vydává příkaz k zastavení.



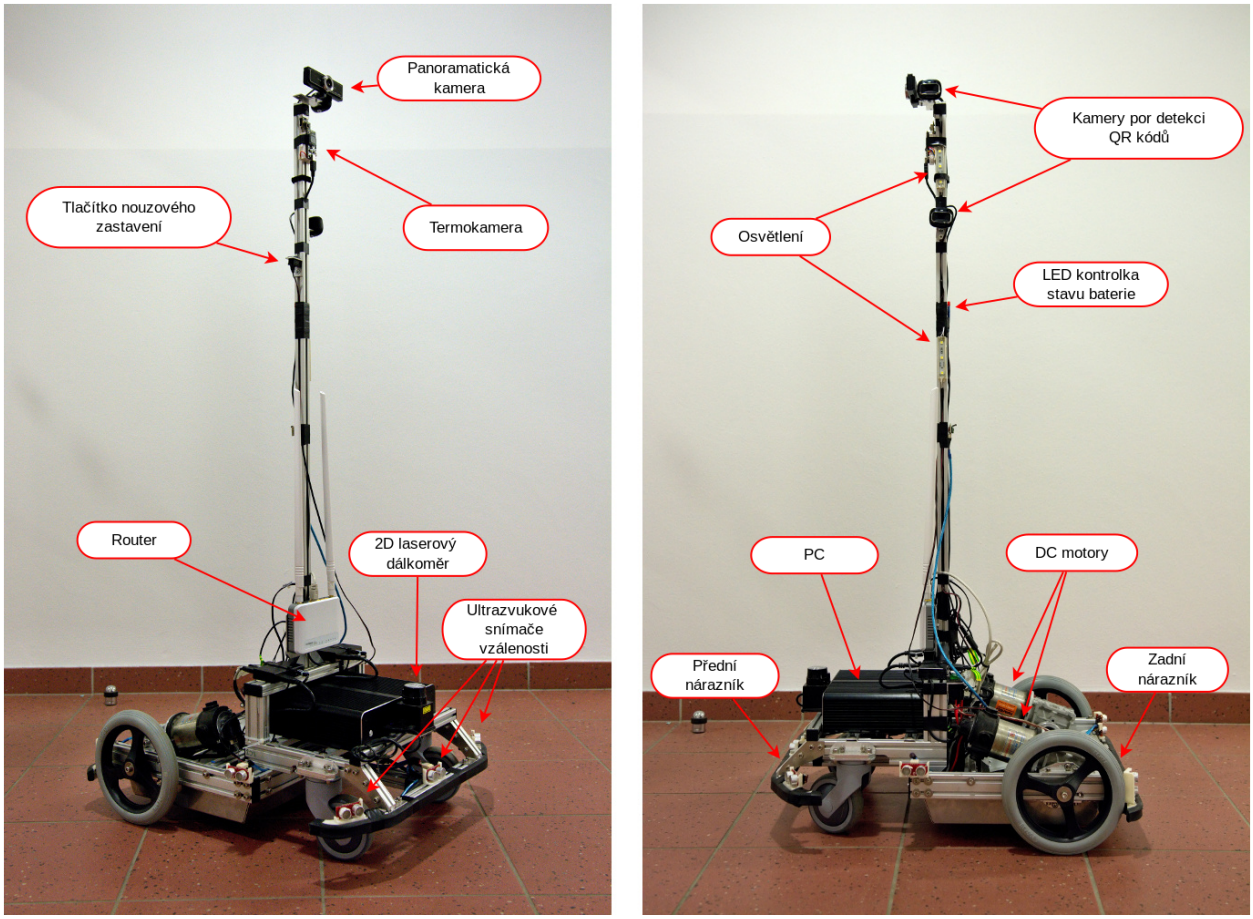
Obrázek 7.3: Vyjížděcí nabídka některých možností v grafickém programu.

Popis vývoje grafických programů a jejich navázání na framework ROS přesahuje rozsah této diplomové práce, zdrojové kódy s komentáři jsou přiloženy v elektronické podobě k diplomové práci.

8. Ověření výsledků

Na obrázku 8.1 je robot DACEP ve fázi prototypu. Testy probíhaly v prostorách Jihomoravského inovativního centra. Nosný test byla autonomní jízda po chodbě cca 20 metrů s otočkou. Toto testování bylo nastaveno a následně do něho nebylo zasahováno, robot tedy celou dobu jezdil autonomně s naprostou spolehlivostí.

Doba jízdy ve výsledku byla cca 10 hodin a ujetá vzálenost dosáhla 7,4 km. Tyto parametry překonaly původní očekávání. Je evidentní, že robot je připraven s rezervou na osmihodinový provoz. Protože motory mají nominální výkon cca 200 W a ostatní systémy necelých 50 W, doba provozu robotu při poloviční ujeté vzdálenosti by měla přesáhnout 16 hodin.



Obrázek 8.1: Popis robotu DACEP.

8.1. Další vývoj

Spolehlivost autonomního robotu DACEP odpovídá stádiu prototypu. Díky grafickému prostředí, které umožňuje využívat kompletní funkcionalitu robotu jej zvládně obluhovat technik po úvodním zaškolení. Grafické prostředí neumožňuje provádět servisní úkony na robotu a jeho konfiguraci včetně výběru aktuální mapy nebo vytváření nových interaktivních značek. Pro komereční využití projektu by bylo potřeba tuto funkcianalitu zařadit.

Prostor pro další vývoj je u globálního plánovače, který pro hledání cesty využívá metodu potenciálového pole. To způsobuje v prostorách typu chodba jízdu robotu po jejím kraji místo uprostřed. Žádná z vyzkoušených knihoven ROSu nevyhovovala.

Při spouštění softwaru robotu, kdy jsou zapínány všechny subsystémy, dochází ve více než 5% případů k chybám, nejčastěji kvůli nesprávné identifikaci všech připojených periferií, především USB zařízení. Pro obnovení činnosti je potřeba některá zařízení restartovat a opět spustit běh programu. To je činnost, kterou by bylo možné automatizovat. Řešením by mohl být nadřazený mikrokontrolér s paralelním přístupem ke všem zařízením s možností diagnostiky stavu a možností restartu vybraných zařízení i běhu programu na PC. Toto řešení je pracné a časově náročné, ale technicky realizovatelné. Vyrobit takovou diagnostickou jednotku je plánováno pro další vývojovou verzi robotu.

Robot je potřeba manuálně nabíjet. Pro průmyslové použití by byla potřeba dokovací stanice se schopností automatického nabíjení.

9. Závěr

V rámci diplomové práce byl popsán vývoj vestavěného řídicího systému a navigačního i obslužného softwaru autonomního mobilního robotu DACEP. Autor se podílel na výběru konkrétní senzorické výbavy. Úkolem bylo toto vybavení zapojit do fungujícího celku spolu s motory, jejich kontrolérem a vestavěným řídicím systémem. Senzorické vybavení zahrnuje ultrazvukové senzory vzdálenosti, pneumatické tlakoměry pro nárazníky, enkodéry motorů, tlačítko nouzového zastavení, senzor teploty a vlhkosti a jiné.

Vestavěný řídicí systém je postavený na platformě Mbed. Jeho činnost je korektní a během 150 hodin testování nebyly zaznamenány problémy nebo nestabilita. Vestavěný řídicí systém a nadřazený počítač komunikují po sběrnici USB přes navržené rozhraní, které je zabezpečeno proti chybám přenosu. Je přenášeno přibližně 300 zpráv za vteřinu, rozhraní je dimenzované na 2000 zpráv za vteřinu. Jednou z funkcí řídicího systému je regulace rychlosti kol, která je realizována pomocí PSD regulátoru.

Pro účely lokalizace a navigace byl použit framework ROS. Bylo zapotřebí nakonfigurovat lokalizační a navigační knihovny a splnit všechny potřebné závislosti potřebné pro jejich korektní běh. V experimentálním ověření byl robot schopný s více než 90% úspěšností bez dalšího zásahu dojet na zadané místo ve známé mapě, kdy jedním z úkonů byl průjezd otevřenými dveřmi. Robot za celou dobu testování neměl v autonomním režimu kolizi se statickou překážkou.

Byl vytvořen grafický software, pomocí kterého je umožněno využívat kompletní funkcionalitu robotu. Technik je po úvodním zaškolení schopný provádět základní diagnostické úkony na robotu, zadávat cíl v mapě, na který se robot má přesunout, zadávat úkoly, jako je měření teploty nebo detekce QR kódu z obrazu. Do mapy je možné vkládat interaktivní značky, na které může být navázána složitější funkcionalita, například příkaz "jednou za 10 minut jeď trasu zadanou kontrolními body a ukládej teplotu v dané oblasti do databáze". Přes tento software je možné kontrolovat pozici robotu v mapě a sledovat živý obraz z kamer. Robot může být vzdáleně ovládán i manuálně.

Velmi dobře dopadly testy výdrže baterií. Se stávající baterií je robot schopný ujet 7,4 km s časem 10 hodin. To jsou dostačující parametry pro plánované použití. Doba nabíjení je 2,5 hodiny.

Projekt DACEP odpovídá stavu prototypu. Robot je schopný plně autonomního pohybu, pro komereční použití by bylo potřeba zvýšit spolehlivost robotu jak po stránce softwaru, tak hardwaru a bylo by nutné rozšířit možnosti grafického prostředí například o servisní úkony. Počítá se s dalším vývojem projektu.

Všechny cíle diplomové práce byly splněny a cíl vytvořit grafické prostředí a jeho provázání s funkcionalitou robotu byl výrazně překročen.

Literatura

[1] MOORE, Gordon. Cramming more components onto integrated circuits. [online]. 19.5.1965. [2017-01-11]. Dostupné z http://web.eng.fiu.edu/npala/eee6397ex/gordon_moore_1965_article.pdf

[2] STATT, Nick. Amazon, once a big spender, is now a profit machine. [online]. 28.7.2016. [2017-01-12]. Dostupné z https://www.theverge.com/2016/7/28/12313526/amazon-q2-2016-earnings-report-aws-cloud-profit

[3] WEINBERGER, Matt. The cloud wars explained: Why nobody can catch up with Amazon. [online]. 7.12.2015. [2017-01-12]. Dostupné z http://www.businessinsider.com/why-amazon-is-so-hard-to-topple-in-the-cloud--and-where-everybody-else-falls-2015-10

[4] Rack. [online]. 2015. [2017-01-12]. Dostupné z http://www.sprava-site.eu/rack/

[5] TOMÁŠ, P. Návrh a realizace senzorického systému pro mobilní robot s využitím frameworku ROS. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2014. 85 s. Vedoucí diplomové práce Ing. Stanislav Věchet, Ph.D..

[6] THRUN, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series). Intelligent robotics and autonomous agents. The MIT Press, August 2005. ISBN 9780262201629.

[7] DUDEK, Gregory and JENKIN, Michael. Computational principles of mobile robotics. Cambridge: Cambridge university press. 391 s. 2nd ed. 2010. ISBN 978-521-69212-0.

[8] GARMIN. LIDAR-Lite v3. [online]. 2017-05-01 [cit. 2017-05-01]. Dostupné z: https://buy.garmin.com/en-US/US/p/557294

[9] GARMIN. LIDAR-Lite v3. [online]. 2017-05-01 [cit. 2017-05-01]. Dostupné z: http://www.robotshop.com/en/hokuyo-urg-04lx-ug01-scanning-laser-rangefinder.html

[10] VELODYNE LIDAR. HDL-64E. [online]. 2017-05-01 [cit. 2017-05-01]. Dostupné z: http://velodynelidar.com/hdl-64e.html

[11] PVEDUCATION. Standard Solar Spectra. [online]. 2017-05-01 [cit. 2017-05-01]. Dostupné z: http://pveducation.org/pvcdrom/appendices/standard-solar-spectra

[12] RobotShop. Sweep V1 360° Laser Scanner. [online]. 2017 [cit. 2017-05-01]. Dostupné z: http://www.robotshop.com/en/sweep-v1-360-laser-scanner.html

[13] Robotics online. Robotics industry insight. [online]. 2015 [cit. 2017-05-01]. Dostupné z: https://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/Intelligent-Robots-A-Feast-for-the-Senses/content_id/5530

[14] KONOLIGE, Kurt et al. A Low-Cost Laser Distance Sensor. 2008 IEEE International Conference on Robotics and Automation. 13 June 2008. ISSN 1050-4729.

[15] Localize with a Hokuyo laser range finder. [online]. 2016 [cit. 2017-05-01]. Dostupné z: https://www.generationrobots.com/en/content/52-localize-with-a-hokuyo-laser-range-finder

[16] Ultrasonic Range Finders. [online]. 2017 [cit. 2017-05-11]. Dostupné z: http://www.robotshop.com/en/ultrasonic-range-finders.html

[17] Build and Test the Ping Sensor Circuit. [online]. 2017 [cit. 2017-05-11]. Dostupné z: http://learn.parallax.com/tutorials/robot/activitybot/activitybot/navigate-ultrasound/build-and-test-ping-sensor-circuit

[18] 3D Camera Survey. [online]. 2017 [cit. 2017-05-11]. Dostupné z: http://rosindustrial.org/news/2016/1/13/3d-camera-survey

[19] ION Motion Control. RoboClaw 2x15A Motor Controller. [online]. 2017-05-07 [cit. 2017-05-07]. Dostupné z: http://www.ionmc.com/RoboClaw-2x15A-Motor-Controller_p_10.html

[20] VENC, L. Návrh a realizace navigačního systému pro mobilní robot Bender II. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2015. 47 s. Vedoucí bakalářské práce Stanislav Věchet.

[21] SAITO, Isaac. Rqt_common_plugins. ROS Wiki. [online]. 2017-05-07 [cit. 2017-05-07]. Dostupné z:http://wiki.ros.org/rqt_common_plugins

[22] WILSON, Andy. Rqt_bag, Package summary. ROS Wiki. [online]. 2017-05-07 [cit. 2017-05-07]. Dostupné z:http://wiki.ros.org/rqt_bag?distro=kinetic

[23] COLEMAN, Davet. XML Specifications. ROS Wiki. [online]. 2017-05-08 [cit. 2017-05-078]. Dostupné z:http://wiki.ros.org/urdf/XML

[24] HOORN. <Link> element. ROS Wiki. [online]. 2017-05-08 [cit. 2017-05-078]. Dostupné z:http://wiki.ros.org/urdf/XML/link

[25] LAMPRIANIDIS, Nick. Ros_control. ROS Wiki. [online]. 2017-05-08 [cit. 2017-05-078]. Dostupné z:http://wiki.ros.org/ros_control#Hardware_Interfaces

[26] HOORN. <Joint> element. ROS Wiki. [online]. 2017-05-08 [cit. 2017-05-078]. Dostupné z:http://wiki.ros.org/urdf/XML/joint

[27] Hybrid Navigation Method for Dynamic Indoor Environment Based on Mixed Potential Fields. Recent Technological and Scientific Advances. 09.10.2013. p.575-582, ISBN 978-3-319-02293-2

10. Seznam použitých zkratek a symbolů

DACEP	DAtaCEnter Patrol robot. Autonomní platforma vyvíjená společností Bender Robotics pro automatizaci datových center
IDE	Integrated Development Environment - vývojové prostředí
IoT	Internet of Things - internet věcí
MUTEX	MUTual EXclusion - prostředek zabraňující dvěma programům využívat naráz jeden sdílený prostředek
ROS	Robot operating system - rozsáhlý framework pro meziprocesovou komunikaci obsahující mnoho navigačních a lokalizačních metod
SLAM	Simultaneous Localization and Mapping - souběžná lokalizace a mapování. Metoda pro lokalizaci v neznámém prostředí
URDF	Unified Robot Description Format - formát pro fyzikální definování robotu

11. Seznam příloh

Příloha A

Obsah CD

Přiložené CD obsahuje elektronickou verzi bakalářské práce a ukázky zdrojových kódů v následující adresářové struktuře:

- */src* - ukázky zdrojových kódů
- */* - domácí adresář obsahující bakalářskou práci v elektronické podobě

Příloha B

Data přenášená po seriové komunikaci

messages PC → Mbed

ID	akce	data_type	popis
001	reset distance	char 001	
005	x-speed	float 003	m/s
006	y-speed	float 003	m/s
007	rotate	float 003	rad/s
100	set P parameter	float 003	
101	set I parameter	float 003	
102	set D parameter	float 003	
110	set lightning	float 003	0.0... off, 1.0... full power

messages Mbed → PC

ID	akce	data_type	popis
004	radius	float 003	[m]
005	x-speed	float 003	[m/s]
006	y-speed	float 003	[m/s]
007	z-speed	float 003	[rad/s]
040	x-speed goal	float 003	[m/s]
041	z-speed goal	float 003	[rad/s]
050	enk[0] speed	float 003	count in second
051	enk[1] speed	float 003	count in second
052	enk[0] distance	float 003	count since reset
053	enk[1] distance	float 003	count since reset
059	battery status	int 002	1... OK 2... battery low 3... battery critical
060	battery voltage	float 003	[V]
061	motor 1 current	int 002	[A*100]
062	motor 2 current	int 002	[A*100]
100	P parameter	float 003	
101	I parameter	float 003	
102	D parameter	float 003	
150	front bumper value	float 003	from bumper mbed
151	rear bumper value	float 003	from bumper mbed
152	bumper status	char 001	from bumper mbed 0: OK 1: front detect 2: front detect
155	temperature	float 003	[°C]
156	humidity	float 003	[%]
190	regulation loop duration	int32 022	[ms]
191	system start	int32 022	[100: device was reseted]
201	sonar 1 range	float 003	[m]
...	...	...	...
206	sonar 6 range	float 003	[m]

Příloha C

URDF elementy pro model robotu

```
<?xml version="1.0"?>
<robot name="nazevRobotu">
    <link name="jmeno">
        <visual name="jmeno">
            <origin xyz="x y z" rpy="r p y" />
            <geometry>
                <box size="x y z" />
                <cylinder length="value" radius="value" />
                <sphere radius="value" />
                <mesh filename="/slozka/soubor.DAE" />
            </geometry>
            <material name="jmeno" >
                <color rgba="R G B A" />
                <texture filename="/slozka/soubor.DAE" />
            </material>
        </visual>
        <collision name="jmeno">
            <origin xyz="x y z" rpy="r p y" />
            <geometry>
                <box size="x y z" />
                <cylinder length="value" radius="value" />
                <sphere radius="value" />
                <mesh filename="/slozka/soubor.DAE" />
            </geometry>
        <inertial>
            <origin xyz="x y z" rpy="r p y" />
            <mass value="vaha" />
            <inertia ixx="xx" ixy="xy" ixz="xz" iyy="yy" iyz="yz"
                izz="zz" />
        </inertial>
    </link>
    <joint name="jmeno" type="typ">
        <origin xyz="x y z" rpy="r p y" />
        <parent link="name" />
        <child link="name" />
    </joint>
</robot>
```

Příloha D

Konfigurace AMCL knihovny

```
<!-- Spusteni knihovny AMCL s povolenym textovym vystupem na terminal. -->
<node pkg="amcl" type="amcl" name="amcl" output="screen">
    <!-- Zmena topicu laser na topic scan. -->
    <remap from="laser" to="scan" />

    <!-- OBECNE PARAMETRY PARTICLE FILTRU -->
    <!-- Nastaveni, jak casto jsou data predavana pro ucely vizualizace. -->
    <param name="gui_publish_rate" value="5.0"/>

    <!-- Nastaveni vychozi pozice v mape. Prikladem muze byt umisteni dokovaci stanice. -->
    <!-- Pozice x a y [m], natoceni [rad] -->
    <param name="initial_pose_x" value="5.88462686539"/>
    <param name="initial_pose_y" value="6.2906126976"/>
    <param name="initial_pose_a" value="1.53"/>

    <!-- Minialni a maximalni pocet castic filtru. -->
    <param name="min_particles" value="100"/>
    <param name="max_particles" value="5000"/>

    <!-- Minimalni vzdalenost nebo rotace mezi dvema zasahy filtru. -->
    <param name="update_min_d" value="0.25"/>
    <param name="update_min_a" value="0.3"/>

    <!-- Parametry KLD samplingu (Kullback-Leibler divergence). -->
    <param name="kld_err" value="0.01"/>
    <param name="kld_z" value="0.99"/>

    <!-- Casova prodleva, po ktere je transformace oznacena za platnou -->
    <param name="transform_tolerance" value="0.2" />

    <!-- Pocet zasahu filtru pred zmenou poctu vzorku filtru -->
    <param name="resample_interval" value="2"/>

    <!-- PARAMETRY LASEROVEHO SCANNERU -->
    <!-- Minimalni a maximalni vzdalenost laseru, ktera se bere v potaz. Hodnota -1.0 prebira nastaveni rozsahu definovane v topicu laseru. -->
    <param name="laser_min_range" value="-1.0" />
    <param name="laser_max_beams" value="-1.0" />

    <!-- Pocet bodu laseroveho dalkomeru, ktere vstupuji do vypoctu. Vice znamena lepsi lokalizaci a vetsi vypocetni narocnost. -->
    <param name="laser_max_beams" value="30" />

    <!-- PARAMETRY ODOMETRIE -->
    <!-- Frame_id (pozor, ne topic!) odometrie, zakladny robotu a mapy -->
    <param name="odom_frame_id" value="odom"/>
    <param name="base_frame_id" type="str" value="base_link" />
```

```
46      <param name="global_frame_id" type="str" value="map" />
47
48      <!-- Volba typu modelu robotu mezi diferencialne rizenym a omni
            (vsesmerovym). V modelech byla nalezena chyba a byly vytvoreny
            nove nazvane diff-corrected a omni-corrected. -->
49      <param name="odom_model_type" value="diff-corrected"/>
50      <!-- Predopokladany sum rotacni slozky odometrie na zaklade
            rotacni slozky pohybu. -->
51      <param name="odom_alpha1" value="0.1" />
52      <!-- Predopokladany sum rotacni slozky odometrie na zaklade
            translacni slozky pohybu. -->
53      <param name="odom_alpha2" value="0.1"/>
54      <!-- Predopokladany sum translacni slozky odometrie na zaklade
            translacni slozky pohybu. -->
55      <param name="odom_alpha3" value="0.1"/>
56      <!-- Predopokladany sum translacni slozky odometrie na zaklade
            rotacni slozky pohybu. -->
57      <param name="odom_alpha4" value="0.1"/>
58
59      <!-- Parametry laseru, jejichz ucel zustava skryt -->
60      <!-- Mixture weight for the z_hit part of the model.
61      <param name="laser_z_hit" value="0.9" />
62      <!-- Mixture weight for the z_short part of the model. -->
63      <param name="laser_z_short" value="0.1" />
64      <!-- Mixture weight for the z_max part of the model. -->
65      <param name="laser_z_max" value="0.05" />
66      <!-- Mixture weight for the z_rand part of the model. -->
67      <param name="laser_z_rand" value="0.1" />
68      <!-- Standard deviation for Gaussian model used in z_hit part of
            the model. -->
69      <param name="laser_sigma_hit" value="0.2" />
70      <!-- Exponential decay parameter for z_short part of model. -->
71      <param name="laser_lambda_short" value="0.1"/>
72
73      <!-- Mapa je prijata z topicu /map misto volani servicu. -->
74      <param name="use_map_topic" value="true" />
75  </node>
```

Příloha E

Konfigurace move_base knihovny

```
<node pkg="move_base" type="move_base" name="move_base">
    <!-- Volba lokalniho a globalniho planovace -->
    <param name="base_global_planner" value="navfn/NavfnROS" />
    <param name="base_local_planner"
        value="dwa_local_planner/DWAPlannerROS" />

    <!-- Nouzove algoritmy robotu pri uviznuti -->
    <param name="recovery_behaviors" value="[{name:
        conservative_reset, type:
        clear_costmap_recovery/ClearCostmapRecovery}, {name:
        rotate_recovery, type: rotate_recovery/RotateRecovery}, {name:
        aggressive_reset, type:
        clear_costmap_recovery/ClearCostmapRecovery}]" />

    <!-- Zakazani cisteni mapy metodou otocenim na miste -->
    <param name="clearing_rotation_allowed" value="false" />

    <!-- Vzdalenost, v jake jsou vymazany prekazky z mapy behem faze
        conservative reset -->
    <param name="conservative_reset_dist" value="2.0" />

    <param name="controller_frequency" value="5.0" />
    <param name="controller_frequency" value="5.0" />
    <param name="controller_frequency" value="5.0" />
    <param name="controller_frequency" value="5.0" />
    <param name="controller_frequency" value="5.0" />

    <!-- Parametr, jak casto jsou posilana prikazy rychlosti
        (/cmd_vel) -->
    <param name="controller_frequency" value="5.0" />

    <!-- Doba, po ktere je pristoupeno k metodam resicim uviznuti -->
    <param name="planner_frequency" value="4.0" />

    <!-- Nacteni parametru navigace -->
    <rosparam file="$(find navigace)/config/local_costmap_params.yaml"
        command="load" />
    <rosparam file="$(find
        navigace)/config/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find navigace)/config/dwa_local_planner.yaml"
        command="load" />
</node>
```

Konfigurace souboru global_costmap_params.yaml

```
global_costmap:
    global_frame: /map
    robot_base_frame: base_link
    update_frequency: 0.3
    transform_tolerance: 0.5
    publish_frequency: 1.0
    static_map: true
    inf_is_valid: true
    inflation_radius: 40.0
    cost_scaling_factor: 15.0
    footprint_padding: 0.10
    robot_radius: 0.5
    footprint: [[-0.135,-0.25],[0.35, -0.25], [0.505,-0.15], [0.505,
        0.15], [0.35, 0.25], [-0.135, 0.25]]
    obstacle_range: 1
    raytrace_range: 2.5
    max_obstacle_height: 0.6
    min_obstacle_height: 0.08
    observation_sources: laser_scan_sensor
    laser_scan_sensor: {sensor_frame: laser, data_type: LaserScan, topic:
        /scan, marking: false, clearing: true}
```

Konfigurace souboru local_costmap_params.yaml

```
local_costmap:
    global_frame: /odom
    robot_base_frame: /base_link
    update_frequency: 8.0
    transform_tolerance: 0.5
    publish_frequency: 8.0
    static_map: false
    rolling_window: true
    width: 4
    height: 4
    resolution: 0.035
    transform_tolerance: 0.5
    inf_is_valid: true
    obstacle_range: 1.0
    raytrace_range: 1.0
    footprint: [[-0.135,-0.25],[0.35, -0.25], [0.505,-0.15], [0.505, 0.15], [0.35, 0.25], [-0.135, 0.25]]
    footprint_padding: 0.05
    robot_radius: 1.0
    inflation_radius: 0.04
    cost_scaling_factor: 26.0
    max_obstacle_height: 0.6
    min_obstacle_height: 0.08
    observation_sources: laser_scan_sensor sonar1 sonar2 sonar3 sonar4 sonar5 sonar6 sonar7 sonar8
    laser_scan_sensor: {sensor_frame: laser, data_type: LaserScan, topic: /scan, marking: true, clearing: true}
    sonar1: {sensor_frame: SRF08_1, data_type: LaserScan, topic: /SRF08_1, marking: true, clearing: true}
    sonar2: {sensor_frame: SRF08_2, data_type: LaserScan, topic: /SRF08_2, marking: true, clearing: true}
    sonar3: {sensor_frame: SRF08_3, data_type: LaserScan, topic: /SRF08_3, marking: true, clearing: true}
    sonar4: {sensor_frame: SRF08_4, data_type: LaserScan, topic: /SRF08_4, marking: true, clearing: true}
    sonar5: {sensor_frame: SRF08_5, data_type: LaserScan, topic: /SRF08_5, marking: true, clearing: true}
    sonar6: {sensor_frame: SRF08_6, data_type: LaserScan, topic: /SRF08_6, marking: true, clearing: true}
    sonar7: {sensor_frame: SRF08_7, data_type: LaserScan, topic: /SRF08_7, marking: true, clearing: true}
    sonar8: {sensor_frame: SRF08_8, data_type: LaserScan, topic: /SRF08_8, marking: true, clearing: true}
```

Konfigurace souboru dwa_local_planner.yaml

```
visualize_potential: true
DWAPlannerROS:
    #Robot Config Params
    acc_lim_theta: 0.6
    acc_lim_x: 0.5
    acc_lim_y: 0.0
    acc_limit_trans: 0.6
    max_vel_x: 0.5
    min_vel_x: -0.15
    max_vel_y: 0.0
    min_vel_y: 0.0
    max_trans_vel: 0.5
    min_trans_vel: 0.07
    max_rot_vel: 0.4
    min_rot_vel: 0.00
    #Forward Simulation Parameters
    sim_time: 3.0
    sim_granularity: 0.03
    vx_samples: 10
    vy_samples: 0
    vtheta_samples: 20
    penalize_negative_x: true
    #Trajectory Scoring Parameters
    goal_distance_bias: 24.0
    path_distance_bias: 20.0
    occdist_scale: 1.9
    stop_time_buffer: 0.2
    forward_point_distance: 0.15
    scaling_speed: 0.25
    max_scaling_factor: 0.2
    #Goal Tolerance Parameters
    xy_goal_tolerance: 0.15
    yaw_goal_tolerance: 0.07
    #Oscillation Prevention Param
    oscillation_reset_dist: 0.05
    #Global Plan PArameters
    prune_plan: false
```