VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VOZÍTKO PRO MAPOVÁNÍ MALÝCH NEBO ŠPATNĚ DOSTUPNÝCH PROSTOR

VEHICLE FOR SMALL AND REMOTE SPACE MAPPING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Pavel Koupý
AUTHOR

VEDOUCÍ PRÁCE — Ing. Vítězslav Beran, Ph.D.
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20988/2017/xkoupy00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Koupý Pavel**
Obor: Informační technologie
Téma: **Vozítko pro mapování malých nebo špatně dostupných prostor**
Vehicle for Small and Remote Space Mapping
Kategorie: Modelování a simulace

Pokyny:

1. Nastudujte si problematiku MonoSLAM a seznamte se s frameworkem ROS a dalších prerekvizit na Raspberry PI.
2. Postavte vhodný podvozek a navrhněte řídicí, lokalizační a mapovací systém s jednoduchým rozhraním pro ovládání pomocí Raspberry PI. Navrhněte mód pro autonomní objevování a mapování neznámého prostoru a jeho vizualizaci.
3. S pomocí existujících knihoven a nástrojů řešení realizujte.
4. Otestujte řešení autonomního módu provedením experimentů na různě složitých překážkách. Výsledky diskutujte.
5. Vytvořte plakát a krátké demonstrační video reprezentující Vaše řešení.

Literatura:

- Sebastian Thrun, Wolfram Burgard, and Dieter Fox. 2005. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press.
- Michael Montemerlo, Sebastian Thrun. *FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*. Springer Tracts in Advanced Robotics - Svazek 27.
- Dále dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a částečně bod 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Beran Vítězslav, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
L.S. 612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Práce popisuje výrobu robotického vozítka pro vzdálené mapování malých prostor. Výroba zahrnuje návrh mechanické konstrukce, elektroniky a implementaci rozhraní poskytujícího sadu nástrojů jako je vzdálené ovládání, snímání kamery nebo autonomní prohledávání místností.

Abstract

This work describes process of making a robotic vehicle for remote mapping of small indoor spaces and areas. Such process involves design and construction, selection and connection of used electronic parts and implementation of interface witch wil provide set of tools as remote control, camera stream or autonomous space search.

Klíčová slova

Autonomní vozítko, Robotika, Arduino, Raspberry PI, ROS, lokalizace a mapování, vzdálené ovládání

Keywords

Autonomous vehicle, Robotics, Arduino, Raspberry PI, ROS, localization and mapping, remote control.

Citace

Pavel Koupý: Vozítko pro mapování malých nebo špatně dostupných prostor, bakalářská práce, Brno, FIT VUT v Brně, 2018

Vozítko pro mapování malých nebo špatně dostupných prostor

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vítězslava Berana, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Pavel Koupý
17.5.2018

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Vítězslavu Beranovi, Ph. D. za odborné vedení této práce. Také bych chtěl poděkovat všem těm, kteří mi poskytli své názory, nápady nebo případné korekce gramatiky a pravopisu při psání této práce.

Obsah

1 Úvod....2
2 Konstrukce a elektronika....4
2.1 Vozítka a další platformy....4
2.2 Konstrukce....6
2.3 Elektronika....9
3 Software....13
3.1 ROS a operační systém....13
3.2 Řízení podvozku....15
3.2.1 Uživatelské rozhraní....16
3.2.2 Testy systému a spouštění....18
3.2.3 Manuální režim ovládání....19
3.3 Autonomní řízení....19
3.3.1 Subsumpční architektura....19
3.3.2 Senzory....21
3.3.3 Implementace....22
3.3.4 Testování....26
3.3.5 Nedostatky....26
3.4 Tvorba mapy a lokalizace....27
3.4.1 Navigace....27
3.4.2 SLAM....27
3.4.3 Implementace....28
3.4.4 Testování....30
4 Závěr....33
Literatura....34
Přílohy....35
A – Náklady....36

1 Úvod

Dnes jsou autonomní vozítka a roboti předmětem zájmu, jejich návrh zahrnuje více pohledů na problematiku realizace takovéhoto robota či vozítka. Těmito pohledy jsou kromě samotné implementace softwarové části, také stavba a obvodové zapojení. Tyto části bývají při vývoji softwaru na osobních počítačích obvykle programátorovi skryty. Existuje samozřejmě mnoho stavebnic a napůl hotových řešení, které by projekt velmi usnadnily, ale záměrem práce je komplexní dílo a tedy projekt zahrnuje stavbu vozítka od tisku konstrukčních části kol a podvozku, po implementaci uživatelského rozhraní a dalších softwarových částí.

Základem bylo vytvořit platformu, která ponese výpočetní jednotku, na které poběží lokalizace a mapování vozítka v prostoru, řízení pohonu, přenos videa kamery pro dálkové ovládání a další nezbytné nástroje či senzory. Základem tak bude univerzální platforma – vozítko, které bude schopné nezávislého prozkoumávání vnitřních prostor a bude obohaceno o lokalizaci v prostoru a výstup z kamery, které umožní zmapování a orientaci v daném prostoru.

Důležitým článkem je tak i operátor vozítka, který musí být schopen vozítko uvést do chodu skrze rozhraní, které mu bude poskytovat možnost spustit automatické procházení místnosti nebo případně dálkové ovládání vozítka za pomoci kamery a senzorů. Jak již bylo zmíněno, vozítko bude univerzální platformou, a tak bude možné jej využít pro další výzkum. Cílem je s využitím nových dostupných technologií navrhnout nízko-nákladové řešení.

Před samotným návrhem řešení, chci věnovat pár vět hlubší specifikaci a analýze tématu práce. Vzhledem k volbě vlastního zadání, které od počátku mělo za cíl postavit slušnou platformu pro další výzkum a využití, které zatím nemá jasné obrysy, ale v podstatě jde o vyzkoušení celého procesu stavby robota. Vozítko jako hlavní senzor používá jedinou kameru a případně další nízko nákladové senzory, jako ultrazvukové senzory vzdálenosti, dotykové(tlačítkové) nárazníky pro zpřesnění či další účely. Dále bych chtěl zpřesnit tu část názvu zadání, která se týká určení cíle vozítka a to část: „malých nebo špatně dostupných prostor“. Tato část názvu práce potřebuje přinejmenším vysvětlení a případně definici, aby bylo možné zhodnotit, zda-li je výsledná konstrukce a implementace v rámci zadání. Pro mne nezbytný krok, protože při volbě zadání mi tato část byla zřejmá jen v hrubých obrysech, protože s takhle pokročilou technologii pro lokalizaci a mapování jsem se při práci s robotickými hračkami a stavebnicemi nesetkal. Myšlenka je tedy vytvořit vozítko, které používá čtyři kola v kombinaci s čtyřkolovým diferenciálním pohonem, které dokáže vytvářet mapu svého okolí, a taktéž bude schopno se v této mapě lokalizovat.

Při návrhu takového systému je vhodné začít dekompozicí na podproblémy. Pro samotnou implementaci vyšších cílů, jako je lokalizace a mapování nebo případně autonomní řízení, je nutné

nejprve vytvořit platformu, na které bude možné tyto části testovat a ladit. Práci jsem rozvrhl na tři větší celky :

- konstrukce vozítka,
- výběr elektroniky včetně obvodové zapojení,
- implementace žádané funkcionality v podobě uzlů pro ROS.

Text práce tak popisuje, jak dosáhnout realizace výše zmíněného. Kapitoly jsou děleny teoretický podklad, návrh a vlastní implementaci. Kapitola 2 je věnována konstrukci, obvodovému zapojení. Kapitola 3 se zabývá návrhem softwaru a dělí se na tři hlavní celky. Prvním je řízení podvozku, které zahrnuje i manuální ovládání. Druhý celek je autonomní řízení a poslední je tvorba mapy a navigace.

Pro demonstraci a ověření vytyčených cílů jsem provedl několik experimentů, které jsou zdokumentovány na konci každé větší části implementace. Tyto experimenty by měli demonstrovat úspěšnost implementace a konfigurace jednotlivých součástí systému. Jako prostředí pro tyto pokusy jsem vybral jeden z pokojů v domě. V pokoji jsem vytvořil experimentální podmínky, za použití falešných zdí a překážek z krabic. Krabice jsou různých velikostí a to umožní otestovat nejen ultrazvukový senzor, ale i senzor doteku. Testy, které jsem provedl, jsou zaměřeny spíše na subjektivní míru funkcionality, tedy jestli implementovaná součást funguje očekávaným způsobem. U každého pokusu je uveden subjektivní názor na úspěšnost řešení, případně další komentáře k řešené problematice. Hlavními body, které se v těchto pokusech snažím otestovat jsou :

- Autonomní režim
- Technologie pro současnou lokalizaci a mapování
- Manuální ovládání

Výstup z experimentů je možné zhlédnout v demo videu, které je součástí odevzdaných souborů.

2 Konstrukce a elektronika

Prvním krokem tak byla konstrukce. Zde vzhledem k účelu vozítka bylo zapotřebí dohledat plány Mecanum[1] kol a případně podvozku. V tomto bodě jsem se obrátil na komunitu lidí používajících webovou platformu Thingiverse.com, kteří se ve svém volném čase zabývají návrhem a stavbou různých hraček, robotů, dronů a dalších pomůcek každodenního života. Ke svému tvoření používají 3D tiskárny, co by nejlevnějšího prototypovacího nástroje, který lze v dnešní době pořídit. Na této platformě existuje nepřeberné množství volně dostupných modelů a na nehotových řešení, které jsou přímo určené pro další vývoj. Zde jsem našel 3D model zvolených kol a rám podvozku.

2.1 Vozítka a další platformy

Na trhu je k dispozici několik velmi zajímavých robotů a robotických vozítek, ale v rámci tohoto projektu jsem se rozhodl pro z mého hlediska nejlevnější řešení. Tím bylo většinu nosných částí včetně kol vytisknout pomocí 3D tiskárny. Toto řešení v kombinaci s použitými elektrickými a elektromechanickými součástkami, tak bylo v konečném součtu levnější, než dostupná komerční řešení. Nicméně tato komerční řešení posloužila jako námět pro konkrétní model, který byl nakonec zvolen a tedy následně zmíním několik takových platforem. Komerční řešení lze rozdělit zhruba do tří oblastí a to servisní specializované platformy, tele-operační platformy a stavebnice pro děti. Z těchto nabízených řešení jsou první dvě kategorie obvykle vysoce nákladné, a jejich účel je také obvykle specializován na určitý druh práce. Vhodným příkladem mohou být řešení od firmy *Fetch Robotics, Inc.*, která vyrábí robotické platformy pro sklady (viz. Obrázek 1[2] (b)). Dalšími zajímavými zástupci prvních dvou kategorií jsou např. PR2 (c) a Sanbot (a). Tato řešení zapadají jak do kategorie servisních, tak do kategorie tele-operačních robotů, protože tyto platformy jsou velmi pokročilé a tedy nabízí kromě automatizovaných úkonů také možnost vzdáleného ovládání a některé modely i technologii pro tzv. virtuální přítomnost. Tyto platformy používají pro automatizované úkoly obvykle technologie pro současnou lokalizaci a mapovaní, která jim umožňuje si vytvořit mapu neznámého prostoru a zároveň jim poskytuje možnost plánovat nejkratší trasu díky faktu, že robot zná svou pozici v tomto prostoru.

1 https://www.thingiverse.com/thing:1358552

2 (a) http://www.sanbot.com/, (b) https://fetchrobotics.com/, (c) http://www.willowgarage.com/pages/pr2/overview

(a) Sanbot (b) Fetch (c) PR2

Obrázek 1: Příklady robotů

Třetí kategorie, tedy dětské a poloprofesionální stavebnice (viz. Obázek 2[3]), jsou dostupnější řešením, ale jejich kvalita je přímo závislá na jejich ceně. Tyto stavebnice byly taktéž rozhodující faktorem pro výběr konečného modelu vozítka. V závislosti na těchto řešeních, jsem hledal alternativu, která by poskytla prvky ze všech těchto tří kategorií. Těmi nejjednoduššími jsou *DRC mark 1*, které používá tříkolový podvozek s diferenciálně řízenými předními koly a všesměrovým zadním kolem, dalším je čtyřkolová stavebnice *J-bot office*, kde jsou řízena všechna čtyři kola. Vhodnějším čtyřkolovou konstrukcí je robotické vozítko s manipulačním ramenem *Kuka YouBot*, které využívá speciální typ kol.

(a) DRC mark 1 (b) Jbot office (c) Kuka YouBot

Obrázek 2: Robotické stavebnice

3 (a) https://www.robotshop.com/letsmakerobots/daddys-robot-car-drc-mark-1 , (b) https://www.jameco.com/jameco/workshop/JamecoBuilds/jbotrobot.html, (c) http://www.youbot-store.com/

Tento typ kol tzv. *Mecanum* kola jsou zajímavým řešením, svoje využití najdou zejména ve skladech a posterech, kde není místo pro manévrování, které vyžadují klasická kola. Tento druh kol umožňuje vozítku jezdit ve všech směrech, bez nutnosti otáčení základnou, taktéž je možné se otáčet na místě. Tato technologie se stala zajímavou alternativou k obyčejným kolům a v rámci zaměření práce na malé a špatně dostupné prostory se zdála velmi vhodná. Tento typ kol je použit na obrázku 2 (c). Jednotlivé výše zmíněné prvky jako jsou kola a druhy robotických platforem jsem chtěl skloubit do jednoho celku, který nebude nákladný a bude snadno dále rozšiřitelný. Překvapivě nebylo nutné si takovou konstrukci navrhnout, stačilo pouze několik hodin průzkumu na internetu, a dostavil se výsledek. Tím byla volně dostupná tisknutelná stavebnice, kterou jsem nakonec použil pro realizaci této práce.

2.2 Konstrukce

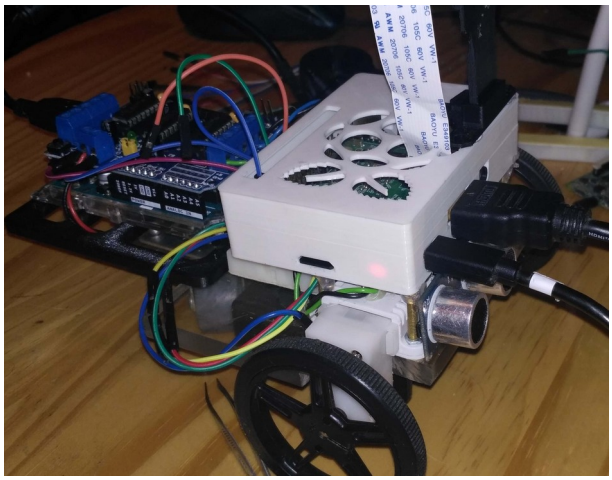
Pro tisk plastových součástí je použito mateiálu PLA[4], který se taví při 215 stupních Celsia, a jednotlivé prvky byly tisknuty s rozlišením 0.20 mm. Části, u kterých nebylo nutné tak jemné rozlišení je použito rozlišení 0.35 mm, které významně zkracuje dobu tisku a pokud není nutný hladký povrch, tak se toto rozlišení jeví jako to nejvhodnější.

Použité modely jsou volně dostupné od komunity Thingiverse.com, kde lidé poskytují své modely volně pod licencí „CC BY-SA 3.0“[5]. Tedy volně šiřitelné, podmíněné pouze uvedením autora a v případě úprav povinnost sdílet takto upravený soubor. Soubor nebyl nijak upravován a není tedy nutné znovu sdílet dle uvedené licence.

Pro zkoušení různých částí jsem zprvu vytvořil jednodušší prototyp a využil i částí vytisknuté pro finální verzi vozítka. Na tomto prototypu jsem si mohl vyzkoušet funkcionalitu elektronických součástek.

4 https://cs.wikipedia.org/wiki/Vl%C3%A1kno_PLA

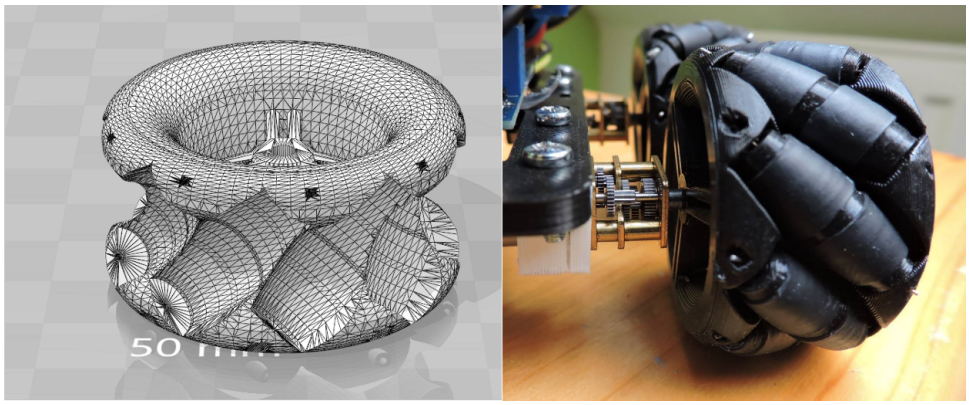
5 https://creativecommons.org/licenses/by-sa/3.0/cz/



Obrázek 3: Prototyp

Podvozek u finální verze vozítka je čtyř motorový diferenciální podvozek , který využívá stejnosměrných modelářských motorů[6]. Tyto motory jsou osazeny kovovou převodovkou s poměrem 1:100 a jsou dimenzovány na napětí 9v.

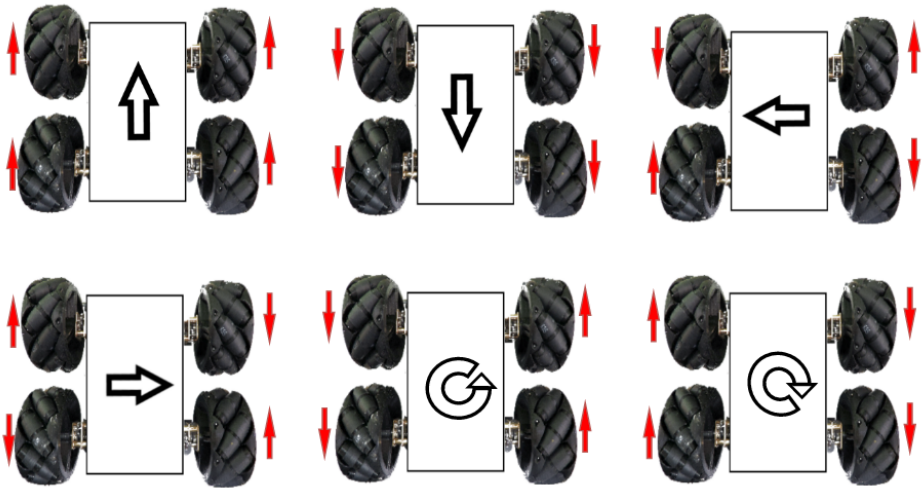
Kola jsou složena z nosného rámu kola a devět válců, které jsou pomocí kovových špendlíků připevněny na rám kola. Každý z těchto válců je pro lepší trakci obalen teplem smršťující bužírkou. Kola s motory jsou ke konstrukci připevněna dvojicí šroubů a plastovou objímkou.



Obrázek 4: Detail kol

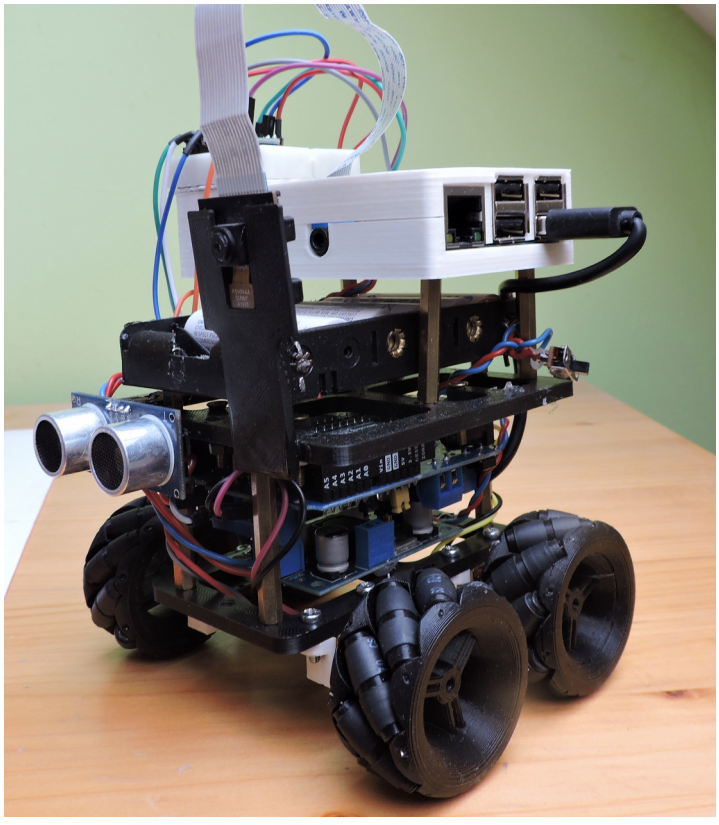
Řízení podvozku je díky použitým kolům odlišné od běžného čtyř motorového diferenciálního podvozku. Obrázek níže abstrahuje výpočty výsledného směru pohybu vozítka (značeno černou šipkou) a směr točení jednotlivých kol (zobrazeno červenými šipkami).

6 https://www.pololu.com/file/0J1487/pololu-micro-metal-gearmotors.pdf



Obrázek 5: Pohon a jeho řízení

Jednotlivé rámy konstrukce, obal výpočetní jednotky a držák baterií jsou k sobě připevněny distančními sloupky různých velikostí. Pomocí těchto sloupků je dále možné rozšiřovat základní rám a ten je tak velmi variabilní a univerzální. Samotná konstrukce tak do budoucna umožňuje připevnění dalších senzorů či aktivních prvků. Avšak vzhledem k množství elektroniky, které je použité a usazení baterií v nevhodné výšce, nemá vozítko příliš dobré těžiště. Maximální možný náklon, který je vozítko schopno zvládnout je 35 stupňů.



Obrázek 6: finální konstrukce

2.3 Elektronika

Z hardwarového hlediska bylo zapotřebí zvolit vhodné komponenty a při volbě byla zohledněna kritéria jako náročnost použití, cena, dostupné knihovny a dostupnost komponentů v tuzemských obchodech.

Jako výpočetní jednotku jsem zvolil **Raspberry Pi** v3.0[7], protože jsem chtěl implementovat SLAM[8] na Raspberry Pi bez nutnosti provádět výpočty na dalším zařízení. To by vyžadovalo stroj s linuxovým operačním systémem a rozcházení ROS frameworku pod tímto systémem. Také jsem chtěl zjistit, jestli je možné takový systém implementovat na zvoleném jednodeskovém počítači s ARM procesorem a omezenou pamětí. Další částí v tomto celku bylo řízení motorů. Zde jsem chtěl oddělit řízení jednotlivých motorů od Raspberry Pi, protože to není navrženo na vyšší odběry, napěťové a proudové výkyvy, ke kterým může při řízení stejnosměrných motorů, tedy hlavně při roztáčení, docházet. Pro tento účel jsem použil vývojovou desku **Arduino UNO**[9], která používá mikrokontrolér ATmega328P. K tomuto mikrokontroléru bude dále připojena rozšiřující deska, které umožní jednoduché ovládání až 4 stejnosměrných motorů. Arduino bude připojeno k Raspberry Pi pomocí sériového rozhraní USB kabelem, tato možnost se jevila jako nejjednodušší, protože umožňuje i napájení přes Raspberry Pi při ladění, kdy je Raspberry Pi připojeno k siťovému adaptéru. Další potřebné součástky jako motory, kamera, baterie a další nezbytnosti jsou popsány dále.

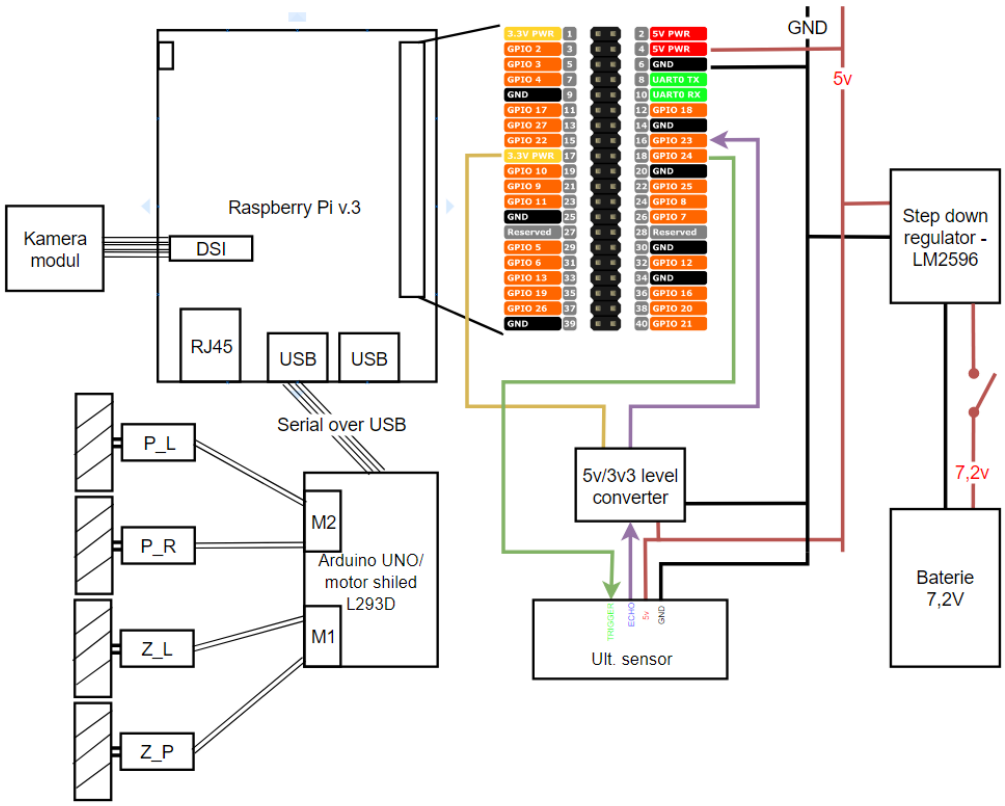
V případě hlavní řídící jednotky, na které poběží ROS framework a většina řídícího softwaru, se nabízely možnosti použít FPGA v kombinaci s mikroprocesorem[10]. Druhou možností je použití nějakého, dnes běžné dostupného mikropočítače s ARM jádrem, kterých je na trhu velké množství a jsou jednodušší na použití než varianta s FPGA. Pro tento projekt je, jak již bylo zmíněno, využito Raspberry PI v3.0, které by mělo poskytnout dostatečný výpočetní výkon pro běh ROS frameworku a zároveň poskytuje i integrovaný WiFi adaptér, který je použit pro bezdrátovou komunikaci rozhraní a vozítka.

7 http://docs-europe.electrocomponents.com/webdocs/14ba/0900766b814ba5fd.pdf

8 Simultaneous localization and mapping

9 https://store.arduino.cc/usa/arduino-uno-rev3

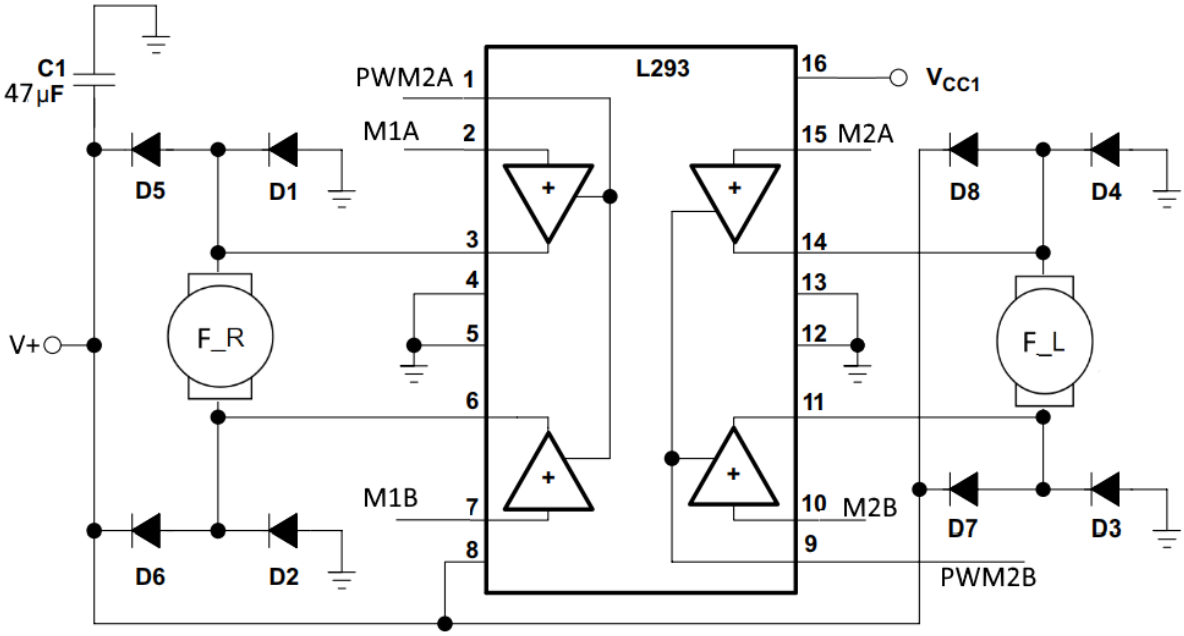
10 http://merlin.fit.vutbr.cz/FITkit/



Obrázek 7: Obvodové zapojení

Motory jsou řízeny za pomoci shieldu[11] pro Arduino, který se sestává z dvojce obvodů L293D[12] a posuvného registru 74HC595N. Skrze obvod L293D je možné řídit vždy dvojici motorů, pomocí PWM[13] signálu. Obvod 74HC595N slouží pro výběr směru otáčení, převádí sériová data z desky Arduina na paralelní data, kde dvojice jednotlivých výstupních bitů (dále specifikováno) slouží pro řízení jednotlivých motorů[14]. Na obrázku níže je zapojení předních motorů, kde každý pin motoru(3,6; 13,14) je připojen na výstupy obvodu L293D. Logickou hodnotu na pinech, s přivedenými signály M1A/B a M2A/B, určuje směr otáčení motorů. Na signály PWM2A a PWM2B je přiveden PWM signál, jehož střídou je určována rychlost motorů, tedy výsledné napětí na svorkách motorů.

11 Obvodu, který svými vstupně-výstupními piny přesně pasuje na vývodu nosiče obvodu, zde Arduino UNO.
12 Obvod skládající se z 4 polovičních H-můstků, které v párech tvoří plné H-můstky.
13 Pulzně šířková modulace
14 V obrázku 8 signály M1A/B, M2A/B a obdobně u druhého obvodu M3A/B ,M4A/B



Obrázek 8: Zapojení stejnosměrných motorů

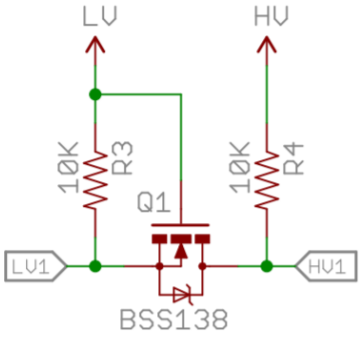
V obrázku jsou dále vyznačeny diody D1 až D8, které slouží jako ochranné. Ty chrání elektroniku před výkyvy napětí na motorech. Kondenzátor C1 v obvodu slouží pro vyrovnávaní rozdílů referenčního napětí 5v. VCC1 je totožné s V+ tedy také 5v. Je tomu tak z důvodu napájení z USB.

Napájení, které je řešeno tužkovými bateriemi je potřeba snížit na úroven, která vyhovuje Raspberry Pi (5v).Z toho důvodu je použit **Step-down regulátor**, který transformuje napětí šesti AAA NiMH baterií s napětím 1,2v. Celkové napětí baterií (7,2v) transformuje na 5v. Regulátor používá obvod LM2596, který má maximální vstupní napětí až 46V. Obvod také disponuje trimrem, jehož regulací je možné plynule s krokem 0,1V nastavit výstupní napětí. Maximální výstupní proud je až 3A, takže by měl poskytovat i zálohu, protože dle dokumentace k Raspberry PI je doporučený 2,5A zdroj. Obsahuje také ochranné členy proti zkratu, přehřátí a obrácení polarity.

Obrázek 9: Step-down regulátor

Několik slov k elektronice senzorů. **Dotykový senzor** je založen na styku dvou částí a to drátu nárazníku a distančních sloupků. Do drátu je přivedeno napětí 3,3V a sloupky jsou připojeny na pin Raspberry Pi, který je nastaven jako vstupní. Při kontaktu je tedy na tomto pinu zaznamenána logická „1“.

Ultrazvukový senzor HC-SR04 používá 5V logiku, ale Raspberry PI operuje na logických hodnotách 3,3V, a tak není vhodné výstupní signál ze senzoru připojit přímo na programovatelný pin Raspberry Pi. Je tedy použit **měnič logické úrovně**. Unipolární tranzistor BSS138 umožňuje obousměrnou komunikaci díky pull-up rezistorům, ale to nebylo pro tuto konkrétní aplikaci zapotřebí. Senzor má datové signály Trigger a Echo. Trigger může být připojen přímo na I/O pin Raspberry Pi, protože se jedná o vstupní signál senzoru, který zahájí vyslání ultrazvukových vln. Senzoru nevadí nižší napětí tohoto řídícího signálu . Echo je výstupním signálem senzoru v logice TTL a tedy je nutné ho převést pomocí výše zmíněného měniče na 3,3V logiku. Měnič používá referenční napětí na vstupech LV (3,3V) a HV (5V) k vystavení logické hodnoty na vstup/výstup LV1 a HV1. Dle číslování pinů desky je signál Trigger připojen na pin 16 a Echo na pin 18. Toto číslování je jedním ze tří způsobů, které Raspberry Pi a knihovny pro práci s GPIO používají. Pro použitou knihovnu WiringPi bude použito číslování zavedené tvůrcem této knihovny. Tato skutečnost bude dále popsána v softwarové části.



Obrázek 10: Měnič logické úrovně

Modul kamery[15] je připojen přímo pomocí plochého DSI kabelu a není zde z hlediska zapojení žádný problém, protože kamera je navržena přímo pro Raspberry PI. Kamera má rozlišení5 Megapixelů a senzor je schopen rozlišení 2592x1944 pixelů. Takové hodnoty však nejsou zapotřebí a i z hlediska přenosu videa bezdrátovými technologiemi je vhodné zvolit rozumnější rozlišení. Zvolené rozlišení bude dále diskutováno.

15 https://www.raspberrypi.org/documentation/hardware/camera/README.md

3 Software

První částí byl tisk jednotlivých komponentů. Poté následovalo osazení elektroniky a pohonu. Na tyto kroky navazuje testování a zkouška pohonu a jednotlivých senzorů. Tato kapitola se věnuje softwarové implementaci a instalaci jednotlivých ROS frameworku a potřebných balíčků. Jednotlivé podkapitoly představují jednotlivé podcíle zadání.

Poslední částí je implementace požadovaných softwarových součástí, jak již bylo naznačeno vozítko bude postaveno na platformě ROS[16]. Tato platforma umožňuje jednoduché oddělení implementovaných softwarových částí a integraci již existujících nástrojů. Zde jsou zajímavé především uzly pro autonomní řízení, které bude zastřešovat všechny ostatní spuštěné uzly a na základě subsumpční architektury bude volit svoje další akce.

U konkrétní implementace prostředí ROS se nabízí dvě možnosti a to klient-server implementace podobně jako v projektu [2]. Druhým způsobem, který jsem zvolil v této práci je implementovat SLAM na Raspberry PI a informace zasílat pomocí SSH protokolu formou grafického rozhraní s případnou vizualizací zpracované mapy a dalšími ovládacími prvky. Toto řešení je pouze pokusné, protože nejsem schopen dopředu říci, jestli na takovou aplikaci bude Raspberry Pi stačit výpočetní výkon.

ROS umožňuje vzdálenou komunikaci s jádrem nez nutnosti hostovat ROS framework pomocí balíčku Rosbridge. Vytváří server, ke kterému lze přistupovat pomocí websocketu. Zásahy v linuxovém operačním systému pomocí ROS systému, však již tak triviální nejsou. Kvůli nutnosti zvýšeného oprávnění superusera, které nejsou zcela bezpečné a mohou vyústit v nepoužitelnost systému, nebo v případě Raspberry Pi až k poškození souborového systému a nemožnosti korektně spustit operační systém. Dalším vodítkem na rozhraní realizované pomocí SSH a né za pomoci ROS prostředků, byl fakt, že kromě Raspberry Pi nemám k dispozici počítač s Linuxovým operačním systémem. Disponuji pouze strojem s operačním systémem Windows, pro který neexistuje plně funkční implementace ROS knihoven.

3.1 ROS a operační systém

Robotická platformy se skládají z různých hardwarových součástí, které je třeba pokrýt softwarovým rozhraním. Příkladem mohou být senzory, řízení motorů a případně další čistě abstraktní funkce, jako autonomní řízení. Zde přichází v úvahu vlastní implementace založená na stavových automatech, ale i tento jednoduchý způsob by zahrnoval definice komunikačních protokolů a adresace jednotlivých prvků. Všechny tyto prostředky poskytuje **Robotic Operation System** (ROS) a to velmi jednoduše

16 http://www.ros.org/

pomocí ROS node (uzly). Uzly zastřešují jednotlivé hardwarové prvky, jako ultrazvukový senzor, senzor doteku a dalších softwarových částí. V dalším odstavci je popsaná současná lokalizace a mapování pomocí balíčku *ORB-SLAM2* nebo autonomní řízení, které je částí mé vlastní implementace. ROS poskytuje také další nástroje pro vizualizaci mapy, různé komunikační uzly, kde příkladem může být uzel pro sériové rozhraní **ROSSerial**, který je použit v tomto projektu pro komunikaci s Arduinem.

Všechny tyto uzly mezi sebou komunikují pomocí tzv. **topics**, tedy komunikačních kanálů, do kterých mohou zasílat data., to je možné skrze **Publishery**, nebo z nich tyto informace vyčítat pomocí **Subscribe** funkcionality. Funkce publikace a vyčítání příchozích informací je základním principem komunikace mezi jednotlivými uzly. Komunikace probíhá pomocí zpráv, kde je možné využít předem definované zprávy. Tyto zprávy mohou být typu řetězce literálů, čísel různých typů a případně předdefinovaných struktur. Je však také možné zadefinovat vlastní typ zprávy, a tím si tak zjednodušit interpretaci jednotlivých informací, které zpráva obsahuje. Pokud je typ zprávy triviální, tedy například naměřená vzdálenost, obvykle stačí použít předdefinované zprávy jako je číslo s pohyblivou desetinnou čárkou, nebo některý celočíselný typ.

Pro vytvoření uzlu je použita ROS šablona, dle které je nutné postupovat, aby mohl uzel správně komunikovat s ostatními uzly . Tato šablona je volná a jedná se spíše jen o ukázku, jak uzly koncipovat. Obsahuje pouze několik základních nutností jako je zadefinování a inicializace node objektu, který poskytuje rozhraní pro přihlášení do jednotlivých topics. Dále šablona definuje vytvoření *Publisher* a *Subscriber* objektů, které definují jméno topicu a typ zprávy, který je možné posílat. A poslední částí je programová smyčka, která zajišťuje opakování zadefinovaného chování uzlu a jeho ukončení v případě ukončení jádra.

Uzly, které poskytuje ROS nebo jsou volně dostupné, obvykle obsahují spouštěcí konfigurační soubory tzv. *launch* soubory. Ty jsou součástí jednotlivých balíčků a umožňují jednodušší spuštění jednotlivých uzlů, protože počáteční konfigurace může být manuálně zdlouhavá a náchylná na chyby.

Pro samotnou implementaci je možné zvolit několik programovacích jazyků a to C++, Python apod.. Zvolil jsem C++, pod ROS je k dispozici **ROSCPP** knihovna, která umožňuje interagovat s ROS jádrem.

Softwarová část projektu zahrnuje zprovoznění ROS frameworku a dalších potřebných uzlů pod tímto frameworkem, případně nástrojů, které tyto uzly používají. Některé uzly jsou již k dispozici, takže jejich zavedení do systému spočívalo jen v ověření funkčnosti na operačním systému Raspberry Pi, kde nemusí být přímo dostupné všechny potřebné prerekvizity, které dále popsané ROS uzly a nástroje potřebují ke své funkci. Zvolená verze ROS frameworku je pro tuto práci ROS Kinetic[17].

17 http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi

Operační systémy pro Raspberry Pi jsou distribuovány v podobě obrazu a jejich zavedení na SD kartu je nenáročnou operací[18]. Zvolený operační systém je Ubuntu 16.04.4 LTS (Xenial Xerus) 32-bit s grafickým prostředím MATE. Na tomto nebo starších verzích toho systému, je podporována většina nástrojů pro ROS a jedná se tak nejpřímočařejší volbu. Pro jednodušší konfiguraci je v systému na Ethernetovém rozhraní nastavena statická IP adresa, která umožní konfiguraci bezdrátového rozhraní, které pak slouží pro bezdrátové ovládání. Zajímavým úkolem bylo na tomto systému přeložit nezbytné nástroje, a samotný balíček pro ORB-SLAM2[19], protože Raspberry Pi má k dispozici pouze 1 GB operační paměti a ta nebyla dostačující pro běh systému, přeložení a slinkování nezbytných balíčků bylo tedy nutné povolit swapování paměti na SD kartu. K přeložení byl nutný swapovací soubor o velikost 2 GB.

3.2 Řízení podvozku

Ze softwarového hlediska bylo možné pro řízení motorů zvolit různá řešení, ale díky sériovému rozhraní, je možné řešit tuto část jednoduše posíláním příkazů stavovému automatu, který je bude v nekonečné smyčce s určenou periodou provádět. Tímto způsobem je provedena konečná implementace. S tím rozdílem, že Arduino je naprogramováno jako jeden z uzlů v ROS frameworku a je možné k němu přistupovat jako běžnému uzlu, které tvoří architekturu vozítka a může tak vyžívat role *publisher* či *subscriber*. Ze strany Raspberry Pi je nutné nastavit sériové rozhraní k Arduinu pomocí uzlu ROSSerial a poté je již možné zasílat příkazy a zprávy pomocí topicu.

Pro samotné řízení slouží stavový automat uvnitř uzlu, který běží na Arduinu a pro generování PWM a řídících je zde použita knihovna[20], dodávaná výrobcem rozšiřující desky s čipy L293D. Tato knihovna umožňuje řízení nejen stejnosměrných motorů, ale také serv a krokových motorů. Knihovna zajišťuje inicializaci mikrokontroléru a správné nastavení PWM výstupu pro jednotlivé motory. Samotné řízení probíhá také pomocí sériové komunikace. Arduino při invokaci metody run() poskytované knihovnou, v určité periodě, posílá na vstup řídícího posuvného registru na rozšiřujícím modulu datové slovo, které je poté vodiči dovedeno na řídící vstupy dvojice L293D čipů, kde slouží pro řízení směru otáčení.

Z pohledu rozhraní knihovny stačí k ovládání příkaz *run()* a *setSpeed()*. Příkaz *run()* bere jako parametr identifikaci motoru a směr otáčení, nebo případně zastavení motoru. Příkaz *setSpeed()* nastavuje střídu PWM signálu a tedy i rychlost otáčení.

Tento uzel má pouze jeden objekt pro komunikaci a to Subscribe objekt s názvem topicu *motors_ctrl*, který slouží pro řízení motorů.

18 https://www.raspberrypi.org/documentation/installation/installing-images/

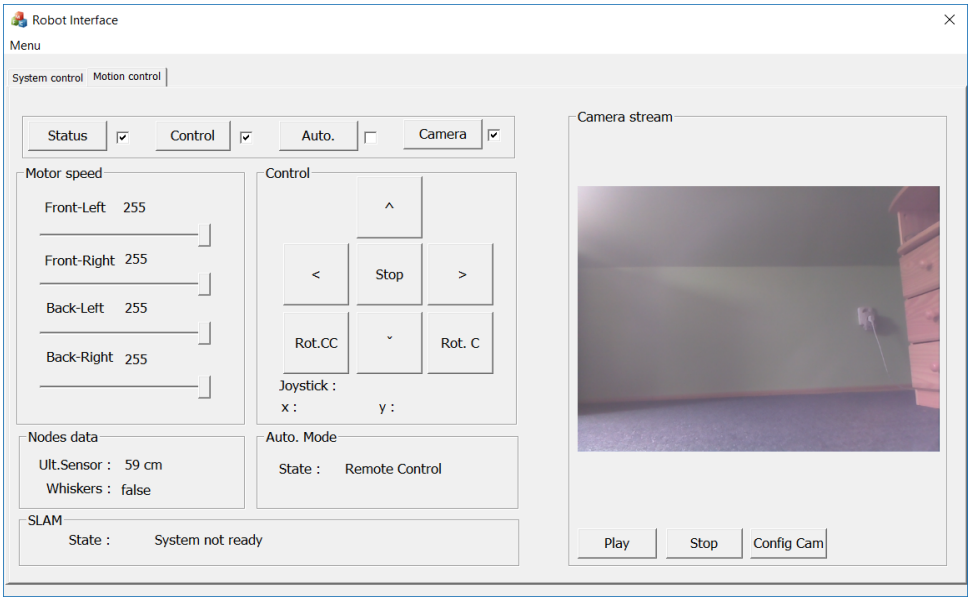
19 https://github.com/raulmur/ORB_SLAM2

20 https://github.com/adafruit/Adafruit-Motor-Shield-library

3.2.1 Uživatelské rozhraní

Rozhraní pro vzdálené ovládání je vytvořeno pomocí knihoven libSSH[21] a MFC[22] ,které zajišťují připojení k vozítku a grafické prvky. Rozhraní má dvě hlavní obrazovky, a to obrazovku pro ovládání vozítka s přístupem k videu z kamery. Aplikace je koncipována pomocí jednotlivých obrazovek, mezi kterými je možno pomocí komponenty *Tab Control*. Každý tab (dále „obrazovka“) je řešen vlastní třídou. V hlavním dialogu je připraven vektor, ve kterém jsou vytvořeny instance tříd obrazovek a dle jejich indexu jsou, pomocí metodu čekající na kliknutí, přepínány jednotlivé obrazovky.

Konfigurační soubory jsou načteny v instanci hlavního dialogu a poté odkazem předány obrazovkám.



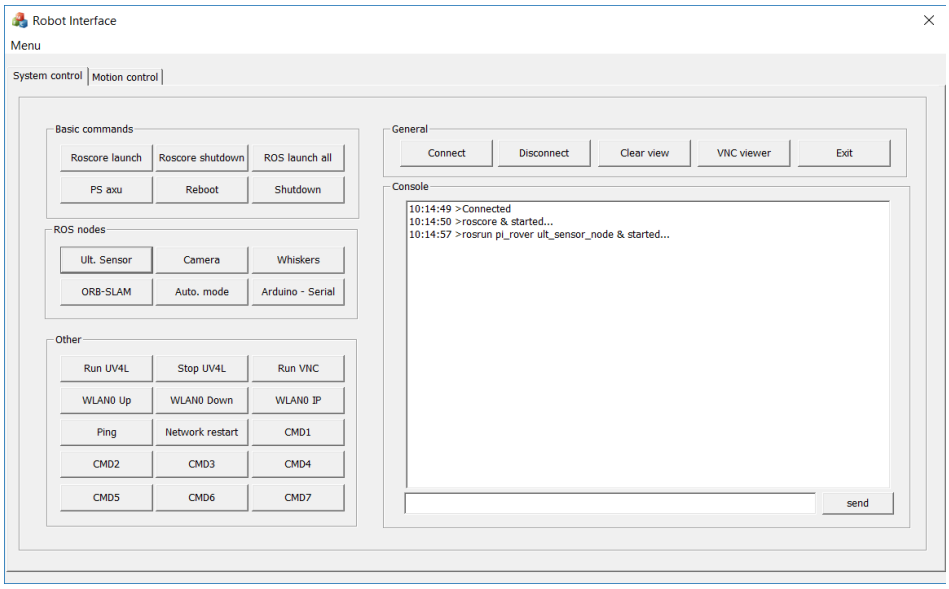
Obrázek 11: Rozhraní pro vzdálené ovládání

Obrazovka pro vzdálené řízení je implementována ve třídě *MotionControlTab*. Na této obrazovce je možné měnit rychlost motorů, interagovat s autonomním systémem řízení a vyčítat data z dalších senzorů připevněných na vozítku. Manuální ovládání vozítka je možné pomocí tlačítek, umístěných v obrazovce nebo pomocí joysticku. Ovládání pomocí tlačítek je řešeno jako neustálý pohyb, aby nebylo nutné tlačítka držet. Snímání pohybů joysticku je řešeno pomocí snímaní *raw input* událostí. Tento přístup zastřešuje rozhraní pro velké množství periferií (joystick, touchscreen, apod.), které jsou souhrnně označovány HID[23]. Programově tak umožňuje abstrahovat specifika periferních vstupních zařízení na generické objekty typu tlačítko nebo pohyb po osách.

21 https://www.libssh.org/

22 https://cs.wikipedia.org/wiki/Microsoft_Foundation_Class_Library

23 https://en.wikipedia.org/wiki/Human_interface_device



Obrázek 12: Rozhraní pro ovládání OS a spuštění systému vozítka

Obnovu stavu zobrazovaných dat a zasálaní příkazů v obrazovce pro vzdálené ovládání mají na starost samostatná vlákna. Každé z těchto vláken otevře nový SSH kanál a zažádá o spuštění shellu[24]. Pro obnovu dat ze senzorů a systému je spouštěno vlákno s funkcí *WatchStatusRun()*. Do spuštěného shellu je zaslán příkaz pro špuštění ROS uzlu, který z topiců *ult_sensor*, *whiskers_sensor*, *automode_status* a *orb_slam* posílá data na standartní výstup. Tento výstup je pak odchytáván z otevřeného SSH kanálu uživatelským rozhraním a dál zpracováván, protože většina údajů není v čitelné formě. Tím vytváří rozhraní mezi ROS frameworkem a uživatelským rozhraním pro čtení Vzdálené ovládání a zasílání dalších příkazů ROS uzlům obstarává vlákno , kterému je přidělena funkce *ControlUpdate()*. Funkce obdobně do shellu na vyhrazeném kanále posílá příkazy, které zpracovávány ROS uzlem , který tyto údaje přímá skrze standardní vstup a poté je zasílá na jednotlivé uzly. Obě vlákna jsou spuštěna tlačítky z uživatelského rozhraní a jsou řízena atomickými hodnotami, které jsou vláknům předány referencí. Parametry zasílaných příkazů jsou v případě směru jízdy atomickou globální proměnnou. Přístup není chráněn mutexem nebo jiným způsobem, protože do proměnné zapisuje pouze jedno vlákno a pouze jedno vlákno data čte, takže atomicita operací by bezpečnostně měla dostačovat.

Přenos obrazu kamery je možný za pomoci knihovny libVLC[25] a příslušného volně dostupného *wrapperu*[26], který je volně dostupný a zvládá většinu dnes známých formátů. Video je přenášeno ve formátu MJPEG, který je určen hlavně pro CCTV kamery, tím je ideálním formátem pro tuto aplikaci. Stream kamery je na straně Raspberry Pi implementován pomocí nástroje UV4L, který

24 kozole

25 https://www.videolan.org/vlc/libvlc.html

26 https://www.codeproject.com/Articles/38952/VLCWrapper-A-Little-C-wrapper-Around-libvlc

vytvoří webový server dostupný na portu 8080, kde je též možná konfigurace obrazu jako rozlišení, natočení, formát apod.. Také je možné zapnout detekci obličejů, která je implementována pomocí OpenCV knihoven, ale tato funkcionalita není příliš optimalizována a značně zpomaluje přenos videa. Při zapnutí SLAM technologie je stream kamery násilně ukončen, protože o stream videa se stará jiný nástroj, než jaký je použit při zachytávání videa pro ORB-SLAM2 a docházelo by ke kolizím. Tento formát, který uzel vyžaduje je raw, tedy formát bez komprese. Datová náročnost přenosu takového formátu by pro bezdrátové rozhraní Raspberry Pi byla neúnosná, protože není vyhrazeno pouze pro přenos videa, ale obousměrnou komunikaci a ovládání.

Pro nastavení základních parametrů slouží konfigurační soubor. Další částí této aplikace je systémové ovládání (*SystemControlTab*), které slouží pro spouštění ROS uzlů, a je také možné ovládat celý systém. Vzhledem k faktu, že rozhraní je implementováno pomocí SSH protokolu, konkrétně třídou *SSHClient*, najdeme na této obrazovce i jednoduchou konzoly. Na tuto konzoly je směrován výstup z jednotlivých příkazů. Je možné posílat, kromě pevně nastavených příkazů pomocí tlačítek i vlastní příkazy. Příkazy, které jsou namapovány na tlačítka v uživatelském rozhraní je možné upravovat v konfiguračním INI souboru, který je načítán při spuštění aplikace. Při jeho změně je tedy nutné aplikaci restartovat, aby se konfigurační soubor mohl načíst znovu. Pro čtení INI formátu je použita hlavičková knihovna INIH[27]. Z této kontrolní obrazovky je možné plně ovládat operační systém vozítka. Lze tedy vozítko vypnout, restartovat, zjišťovat různé parametry systému a nastavení rozhraní. Tato aplikace též umožňuje spustit server pro vzdálenou plochu. Vzdálená správa plochy pomocí přenášení obrazu systému běžícím na Raspberry Pi je zatím jediným způsobem, jak monitorovat výstup z SLAM uzlu, protože není implementován přenos 3D mapy, kterou vozítko vytváří, do aplikace rozhraní.

Komunikace přes SSH protokol probíhá pomocí kanálů, které jsou v programu vytvořeny účelově, zmíněno výše. Popřípadě slouží obecnému účelu jako u obrazovky pro ovládání systému, kde vzhledem spuštění většiny příkazů na pozadí není spuštěný shell na určeném kanálu blokován výstupem těchto spuštěných programů.

3.2.2 Testy systému a spouštění

V této kapitole jde o ověření funkčnosti systému jako celku. Pomocí uživatelského rozhraní bude zapnut ROS a všechny potřebné uzly. Sledované veličiny, zde budou hlavně teplota procesoru, vytížení procesoru a využitá paměť. Tento pokus je tak základním ověřením funkce systému jako celku.

Při experimentálním spuštění dosahovaly hodnoty výše uvedených veličin následujících hodnot:

27 (INI Not Invented Here) - https://github.com/jtilly/inih

Veličiny / zatížení	IDLE	Plné spuštění
Teplota CPU	~ 35°C	~ 58°C
Frekvence jader CPU	0,6 Ghz	1.2 Ghz
Vytížení jader CPU	~ 5 %	~ 85 %
Využití paměti (noswap)	200 Mb	750 Mb

Z hlediska výpočetní síly se ukázalo, že Raspberry Pi postačuje, ale pohybuje se v hraničních hodnotách a je třeba chlazení pomocí větráčku a pasivních chladičů.

3.2.3 Manuální režim ovládání

Tento pokus se zaměřuje hlavně na manuální ovládání vozítka na dálku s použitím uživatelského rozhraní bez přímého oční kontroly vozítka, pouze s pomocí kamery. Tento pokus má ukázat především kvalitu ovládání pomocí joysticku, klávesnice nebo pomocí tlačítek uživatelského rozhraní. Dále bude ověřena faktická správnost dat přenášených systémem do uživatelského rozhraní a jejich zpoždění.

Vozítko bude umístěno do testovacího prostoru. Pomocí uživatelského rozhraní budou spuštěny ROS uzly, které jsou nezbytné pro manuální ovládání. Jsou jimi uzly senzorů, sériová komunikace s Arduinem.

Rozhraní fungovalo celkem úspěšné. Ovládání je dle mého názoru snadné, protože funkcionálně poskytuje stejné možnosti jako klasický terminál. Nadstavbou jsou zde různá ulehčení v podobě před chystaných příkazů a jednoduchého ovládání.

3.3 Autonomní řízení

Před samotnou implementací chci dále uvést několik teoretických poznatků, ze kterých jsem čerpal při návrhu. Jedná se o teoretický podklad autonomního řízení a měření vzdálenosti pomocí ultrazvukového senzoru.

3.3.1 Subsumpční architektura

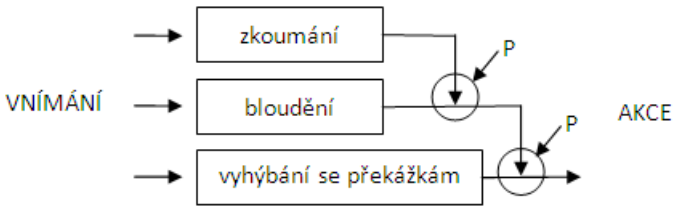
Z hlediska implementace automatického prozkoumávání, se jako zajímavá možnost jevila subsumpční architektura [5], která umožňuje komplexní chování vozítka, spoluprací nebo inhibicí jednotlivých modulů, které obstarávají žádané cíle. Konkrétní cíle (vyhnutí se překážce, náhodné procházení) zajišťují odpovídající odezvu na aktuální prostorovou situaci vozítka, a vytváří tak tzv.

reaktivní agenty. Tento typ agentů si netvoří vlastní symbolickou reprezentaci okolí, a není nutné žádné centrální plánování, ale pouze reaguje na změny ve reálném prostředí, kterým se pohybují.

V obvyklých případech se při tomto druhu problému, tedy implementaci autonomního chování robota, rozloží tento problém na sérii funkcionálních jednotek či stupňů, které obstarávají jednotlivé funkce jako vytváření mapy, plánování směru jízdy apod. Tyto jednotky mají na vstupu data ze senzorů a na výstupu je příkaz k akci robota, jako pohyb vpřed, vzad apod. Tyto prvky jsou na sobě závislé a informace putuje od jedné operační jednotky ke druhé. Podoba přenášené informace je závislá na výstupu každé této jednotky a není tak nutné, aby každý člen měl k dispozici např. informace ze senzoru vzdálenosti. Tuto informaci obdrží pouze první člen a poté ji transformuje na zprávu pro další členy, které však jsou závislé pouze na informacích z předchozích jednotek. Tento přístup je typickým příkladem přístupu k implementaci autonomních robotů. Nicméně tento pohled zanáší do systému chyby, které se množí s každým dalším stupněm, kterým informace prochází, protože žádná z operačních jednotek neoperuje s přesnými daty.

Opačným přístupem je již zmíněná subsumpční architektura, kterou navrhl Rodney Brooks []. Tento přístup je založen na úrovních, které problém rozkládají na jednotlivé cíle robota v závisloti na podnětech z okolí. Tyto úrovně spolupracují pomocí inhibice svých vstupů a výstupů, jinak řečeno tyto úrovně pracují nezávisle na sobě. Všechny moduly mohou posílat příkazy k pohybu robota, ale v závislosti na situaci v prostoru, ve kterém se pohybují, mají některé obvykle nižší úrovně přednost, a umožňují tak reakci například na zaseknutí se o předmět nebo na vzdálenost od zdí apod.. V tomto modelu, vyšší úrovně znamenají komplexnější chování. Tyto vyšší úrovně obvykle potlačují chování nižších úrovní až do chvíle, kdy je nutné reagovat na primitivnější podněty, jako náraz do objektu. Vyšší úrovně obstarávají chování typu průzkum a náhodné procházení. Takovýto model pak ve výslednici vykazuje komplexní chování a umožňuje vozítku dlouhodobě fungovat bez zásahu operátora.

Tyto jednotlivé stupně jsou obvykle popsány konečnými stavovými automaty, které reagují na podměty z fyzického okolí (data ze senzorů). Na základě těchto vstupů a předchozího stavu se volí výstupy. A v konečném důsledku akce robota. Pro popis obecně chování reaktivních agentů existuje formální definice, kterou si dále dovolím zmínit.



Obrázek 13: Příklad architektury reaktivního agenta

Tato definice není kanonická, ale je sdílena více autory a je poměrně srozumitelná. Mějme omezenou množinu jevů, se kterou dokáže agent pracovat, značíme **P.** Tyto jevy vychází z množiny stavů prostředí **E**. (Z prostředí je vybrán jev a vzniká vjem)

$$vjem : E \rightarrow P$$

Přijetí vjemu vede ke **změně stavu agenta**. Tato změna je určena vjemem a stavem jeho přijetí. (Kombinace stavu agenta a aktuálního vjemu vede na nový stav)

$$změna\ stavu : P \times I \rightarrow I$$

Na základě svého stavu a aktuálního vjemu z prostředí **volí agent akci.** (kombinace stavu agenta a vjemu vede na příslušnou akci)

$$akce : P \times I \rightarrow A$$

Provedená akce agenta má následně vliv na okolní prostředí. (kombinace akce a stavu vede na nový stav)

$$vliv\ na\ prostředí : A \times E \rightarrow E$$

Tato formální definice obecně charakterizuje model reaktivního agenta a odpovídá tak šestici {P, A, I, vjem, změna stavu, akce}. Speciálním případem je čistě reaktivní agent, který si nezachovává vnitřní stavy a rozhoduje se pouze na základě informací z prostředí. To omezuje výsledný model pouze na čtveřici {P, A, vjem, akce}.

Této definice jsem se při konkrétním návrh snažil držet. Výsledná architektura a implementace zahrnuje pouze rysy subsumpční architektury a reaktivních agentů obecně a není striktně dodržována. V některých rysech ji porušuje, a to v rámci simulace implementace tohoto modelu v Robotické Operačním Systému (ROS).

3.3.2 Senzory

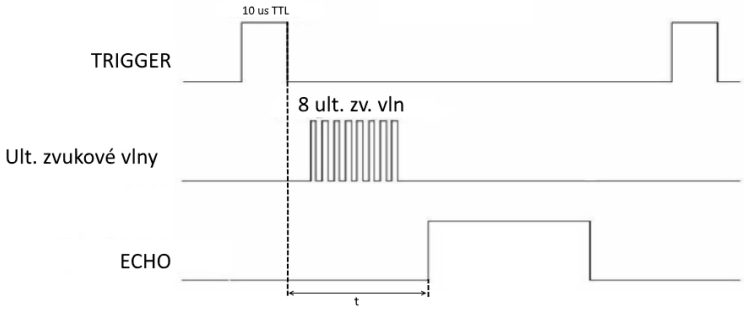
Funkce autonomního řízení využívá dotykový drátový nárazník a senzor pro měření vzdálenosti, které byly z hlediska zapojení již zmíněny. Jsou jimi senzor doteku a ultrazvukový senzor vzdálenosti, které jsou připojeny přes GPIO[28] k Raspberry Pi. Oba senzory obsluhuje samostatný uzel. Z hlediska implementace je zajímavější pouze výpočet vzdálenosti u ultrazvukového senzoru. Knihovna WiringPi[29] umožňuje práci se vstupně-výstupními piny. Tato knihovna zapisuje přímo do

28 https://www.raspberrypi.org/documentation/usage/gpio/

29 http://wiringpi.com/

řídících registrů. Obsahuje různé mapy pro identifikaci jednotlivých pinů. Pro další odkazování na čísla pinů je vybráno mapování dle fyzického očíslování pinů na desce.

Nyní jen krátce zmíním jak je vypočtena vzdálenost u ultrazvukového senzoru.



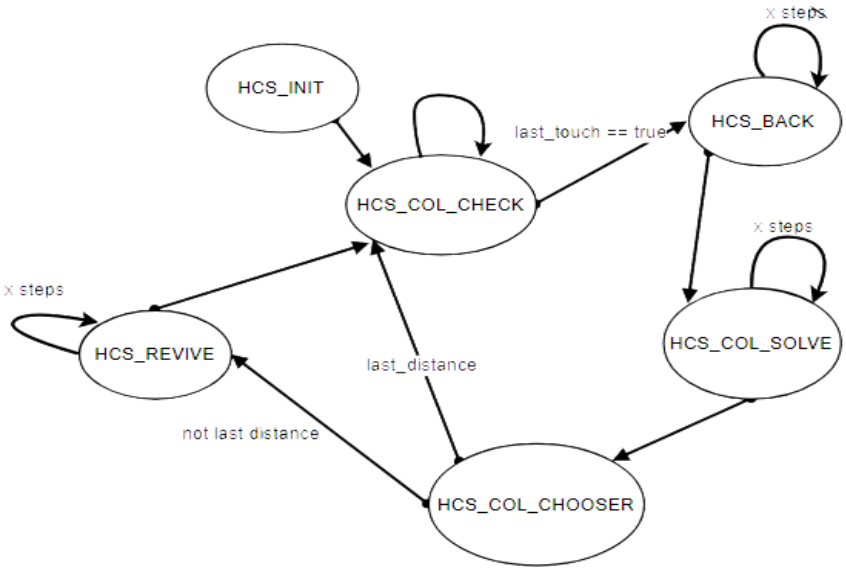
Obrázek 14: Časový diagram ult. senzoru

Vzorec pro výpočet vzdálenosti se odvíjí od rychlosti zvuku, která se liší v závislosti na teplotě. Při teplotě 25 stupňů Celsia v suchém vzduchu, je rychlost zvuku[30] dle vzorce odvozeného ze vzorce pro rychlost zvuku v ideálním plynu 346,1 m/s. Hodnotu jsem zaokrouhlil na 346 m/s. Změříme tedy čas od odeslání signálu Trigger po zachycení odražených vln a změnu signálu Echo z log. „0“ na log. „1“. Tuto hodnotu změříme s přesností na mikrosekundy, z časového diagramu se jedná hodnotu t.

3.3.3 Implementace

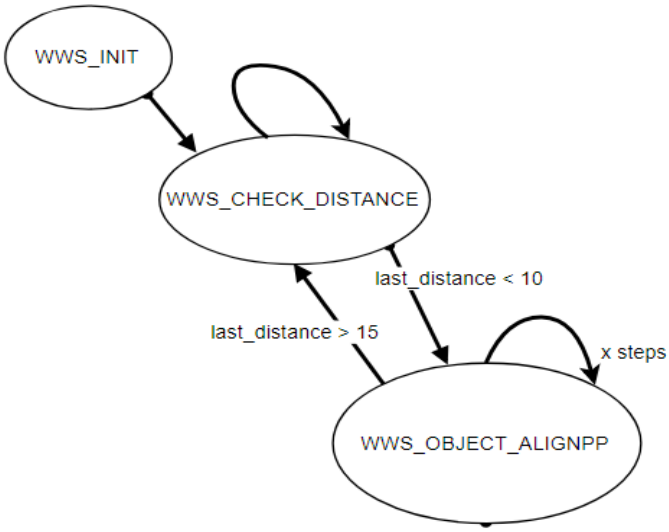
Autonomní režim je podobný subsumpční architektuře, která je zmíněná na začátku kapitoly. Jedná se tedy o sadu konečných stavových automatů, které si pamatují svůj stav a mají jeden nebo více vstupních parametrů. Stav těchto automatů závisí na vstupních datech a aktuálním stavu. Tato funkce je zabalena do samostatného ROS uzlu, přičemž vstup a výstup je řešen pomocí topiců. Tento uzel pro svůj běh vyžaduje spuštění uzlů senzorů a uzlu pro řízení motorů. Uzel je taktéž schopen přijímat jednoduché příkazy, které řídí stav autonomního prozkoumávání. Všechny automaty jsou realizovány jednotlivými funkcemi, které jsou volány z hlavní ROS smyčky s globálními parametry, které představují jejich vstupy a globální proměnné, určujících inhibici především jejich výstupů. Také je možné vozítko ovládat vzdáleně za pomocí jednoduchých příkazů. Tyto příkazy mohou být kontrolní, ale existují i příkazy, které nevyvolávají odezvu na řízení vozítka.

30 https://cs.wikipedia.org/wiki/Rychlost_zvuku



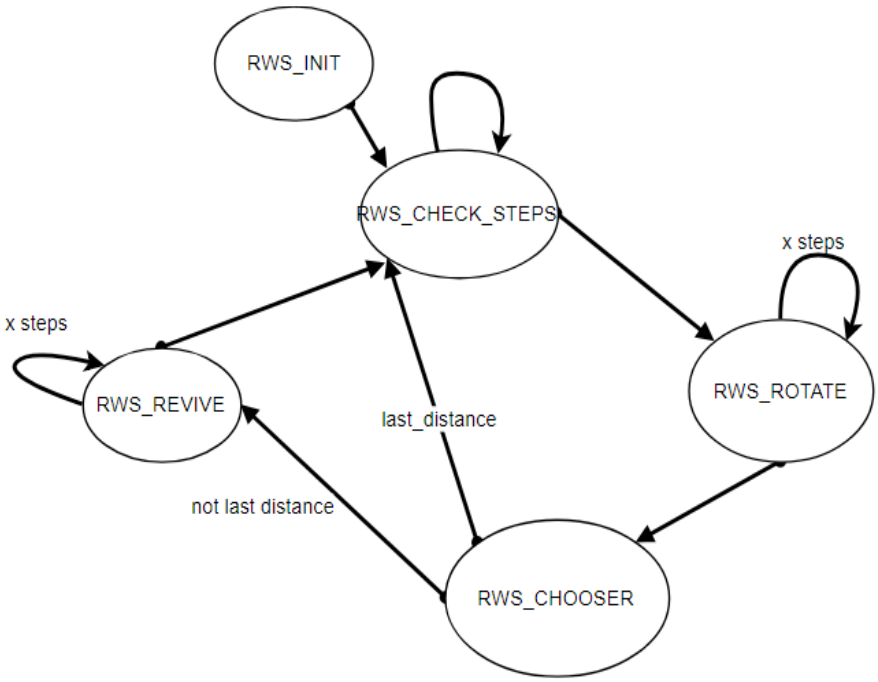
Obrázek 15: přechody stavů 1.úrovně

Autonomní režim má několik úrovní řízení. Na nejnižší úrovní se jedná o vyhýbání se nárazu. Jako vstupní informace je využito senzoru , kterým je drátový nárazník s dvěma místy dotyku na přední části vozítka. Pokud vozítko narazí do nějaké překážky, je iniciována série akcí, která má na starost dostat vozítko z kolize. Pomocí měření vzdálenosti následně určí nový směr jízdy. Vozítko ze stavu HCS_INIT přejde při prvním volání do stavu HCS_COL_CHECK, kde probíhá kontrola, jestli vozítko nenarazilo do nějakého objektu. Informace o nárazu jsou odečítány z proměnné *last_touch*. Je-li detekován náraz, vozítko přejde do stavu HCS_BACK. To umožní vozítku o udělat pár kroků zpět, aby se dostalo z kolize. Následovně vozítko samovolně přejde do stavu HCS_COL_SOLVE. V tomto stavu se vozítko otočí o 270 stupňů a ve třetinách této otočky zaznamená vzdálenosti z ultrazvukového senzoru. Po ukončení této činnosti přejde do stavu HCS_COL_CHOOSER, kde jsou naměřené hodnoty porovnány a je vybrána největší vzdálenost. Dle vzdálenosti je vybrán počet kroků k nasměrování tím směrem, kde vozítko detekovalo největší vzdálenost. Pokud je poslední měření tím největším naměřeným, vozítko je již nasměrováno správně a přechází zpět do stavu kontroly nárazu.



Obrázek 16: přechody stavů 2.úrovně

Druhou úrovní je jízda okolo místnosti, která je realizována na základě dat z ultrazvukového senzoru vzdálenosti. Není-li zaznamenán náraz v nižší vrstvě, je kontrolována vzdálenost z ultrazvukového senzoru. Pokud je tato vzdálenost menší než 10 cm, automat přejde do stavu, ve kterém se vozítko otáčí dokud není vzdálenost od objektu alespoň 15 cm. Automat ze stavu WWS_INIT prvním voláním se dostane do stavu kontroly vzdálenosti WWS_CHECK_DISTANCE. V tomto stavu je kontrolována vzdálenost od překážky pomocí ultrazvukového senzoru vzdálenosti. Pokud je vzdálenost pod již zmíněnou hranici, vozítko přejde do stavu WWS_OBJECT_ALIGNPP. Tento stav upravuje směr jízdy, dokud vzdálenost měřená senzorem není nad určenou hranicí. Poté se vrací do stavu kontroly vzdálenosti. Tento automat i přes svou jednoduchost je tím nejpoužívanějším, protože ostatní dva automaty jsou využity spíše v krajních případech. Jedná se tak o hlavní komponentu, která umožní vozítku vyhýbat se většině srážek.



Obrázek 17: přechody stavů 3.úrovně

Posledním třetím stupněm je náhodné procházení po místnosti. To je implementováno tak, že vozítko se otočí kolem své osy. V průběhu náhodně vybere čtyři místa, na kterých změří vzdálenost a poté se dle této hodnoty rozhodne , jakým směrem se vydat.

Z hlediska implementace je tento automat podobný prvnímu automatu. Ze stavu RWS_INIT se automat dostane do stavu RWS_CHECK_STEPS, kde jsou počítány kroky vpřed. Pokud je tento počet větší než zvolený práh 200 kroků, přejde se do stavu RWS_ROTATE. Tento stav upravuje směr řízení, konkrétně na rotaci. V náhodných směrech získá 4 hodnoty vzdálenosti. Poté přejde do RWS_CHOOSER, kde obdobně jako u prvního automatu je zvolena největší vzdálenost . Pokud je nutná oprava směru, přejde se do stavu RWS_REVIVE, jinak vozítko pokračuje rovně.

Tyto tři stupně se navzájem ovlivňují svým stavem. Je-li například první úroveň ve stavu kolize (na vstupu dotykového senzoru byl zaznamenán náraz do objektu), jsou pozastaveny funkce vyšších úrovní do doby, než první úroveň zahlásí, že kolize byla vyřešena. Pokud je kolize vyřešena, vrací se kontrola druhé nebo třetí úrovni. Priorita mezi druhou a třetí úrovní je určena historickými záznamy kroků. Pokud vozítko jede příliš dlouho, pouze rovně, je po určitém množství kroků zahájeno měření vzdáleností při 360 stupňové otočce.

Tyto stavy jsou propojeny implementačně sekvenčním vykonáním a případnou změnou směru jízdy. Tedy v hlavní smyčce autonomního modu je nastaven krok vpřed. Proměnná uchovávající směr jízdy je profiltrována těmito automaty a výsledná hodnota je poslána na vstup řídicího automatu , který běží na Arduinu.

Autonomní prohledávání lze vypnout a přejít na manuální ovládání vozítka. V tomto případě jsou vypnuty všechny vrstvy autonomního řízení. Data ze senzorů jsou v tomto případě přenášena pouze do uživatelského rozhraní.

3.3.4 Testování

Tento experiment má za cíl, otestovat pouze autonomní řízení, bez spuštěné lokalizace a mapování. Dává tak možnost posoudit pouze systém řízení na základě senzorů dotyku a vzdálenosti.

Vozítko bude umístěno na náhodnou pozici. Budeme sledovat, jestli bude schopné rozumného prohledání místnosti v tom smyslu, že nebude zaseknuté v jednom místě, nebo ve slepé uličce. Ověření probíhá na subjektivní úrovni a je možné, že výsledek, by jiný řešitel zhodnotil odlišně.

Nicméně můj závěr je ve výsledku celkem pozitivní. Vozítko bylo schopno smysluplně projet zadaným prostorem. Nastaly různé komplikace typu zabloudění a převrácení vozítka, ale přesto si vozítko vedlo lépe, než jsem zprvu očekával. Systém i přes svou jednoduchost dokázal projíždět mezi překážkami. Jediný vážnější problém byl v počtu kroků, které je nutné udělat pro kompletní otočku, protože kola nejsou zcela dokonalá. To ve výsledku znamenalo, různý počet kroků, pro různé povrchy. Hodnotu jsem nakonec zprůměroval, protože přesnost není tou nejdůležitější vlastností tohoto systému, a tak není nutné udělat přesně celou otáčku kolem své osy.

3.3.5 Nedostatky

Systém je schopný vyhnout se velkému množství záludností (například slepá ulička). I přesto jsou situace, kdy systém nepostačuje k nalezení vhodného východiska. Jsou to různé extrémy, jako kabely a další nízko profilové předměty, které nezachytí žádné čidlo. Velkým nedostatkem, který vznikl špatným odhadem náročnosti, je nepřítomnost automatu, který by řídil mapování a lokalizaci. Specificky jde o část inicializace systému, a jeho rekonvalescenci, v případě ztráty kontextu. Tento problém se již v případě manuálního ovládání ukázal jako celkem složitý. Při ztrátě kontextu je nutné vozítko dostat do pozice, kde může kontext znovu obnovit. To se i při manuálním ovládání vždy nedařilo dle očekávání. Samotná lokalizace je pak možná i v autonomním režimu, protože už není vytvářena žádná mapa. Pokud je tedy dobře vytvořena mapa s pomocí manuálního ovládání, tak obvykle stačí vozítko otočit na místě, případně udělat pár kroků zpět.

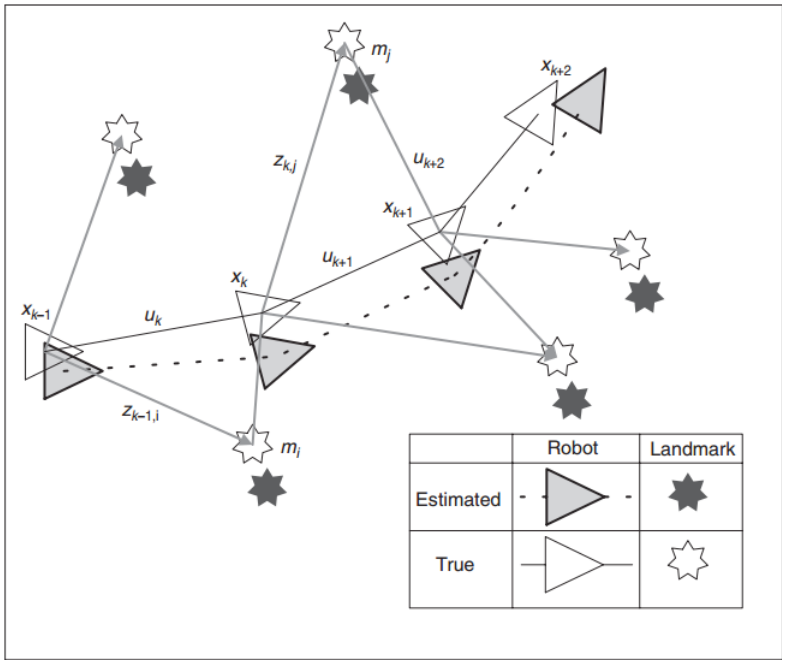
3.4 Tvorba mapy a lokalizace

3.4.1 Navigace

Pro úspěšnou navigaci vozítka, ať už autonomní nebo manuální, vzniká potřeba znát alespoň přibližnou pozici vozítka v prozkoumávaném prostoru. Tento problém zahrnuje též vytváření mapy a pro jeho řešení existuje velké množství algoritmů. Ty umožňují tento problém s různou úspěšností řešit. Jedna skupina algoritmů se označuje SLAM algoritmy. V této práci je cílem použití jedné kamery. Řešení tohoto problému obvykle vyžaduje data z různých měřících senzorů, jako je LIDAR, nebo stereoskopické případně RGB-D kamery. Tyto jsou jak nákladné, tak asociace dat z těchto senzorů pro řešení SLAM problému mohou být výpočetně náročné. Chtěl jsem tak využít možnosti použít pouze jednu kameru, s jejíž pomocí zmapuji daný prostor pro případný zásah operátora a umožním mu sledovat vozítko na vytvořené mapě. V tomto odstavci se tak budu věnovat různým možnostem, jak je tento problém řešen v obecné rovině a poté konkrétněji, jak je tento problém řešen v systémech s jednou kamerou. Nakonec v tomto kontextu zmíním technologii, kterou jsem vybral pro implementaci v této práci.

3.4.2 SLAM

Problém současné lokalizace a mapování (SLAM), který tyto algoritmy [4] řeší spočívá ve zmapování neznámého prostoru a určení pozice vozítka. Vozítko prozkoumáváním tohoto prostoru inkrementálně vytváří mapu prostředí a určuje pozici. Nalezením algoritmu, který řeší takto postavený problém, umožňuje vozítku další úroveň autonomnosti případně rozsáhlejší množství poskytovaných informací o prostředí, kterým vozítko projíždí .



Obrázek 18: Obecný popis SLAM problému

Jednoduše lze popsat tento problém jako zpřesňování předpokládané a skutečné pozice vyčtené z relativního posunu orientačních bodů $\mathbf{m_i}$ z měření prováděných senzory na vozítku v místech $\mathbf{x_k}$. Z obrázku 15[31] tak vychází model popsaný následovně:

- $\mathbf{X_k}$, vektor popisující souřadnice a orientaci vozítka.
- $\mathbf{m_i}$, vektor popisující souřadnice orientačního bodu a ty zůstávají statické.
- $\mathbf{z_{k,i}}$, pozorovaní orientačního bodu $\mathbf{m_i}$ v bodě $\mathbf{X_k}$.
- $\mathbf{u_k}$, kontrolní vektor, který je aplikován v bodě $\mathbf{X_{k-1}}$, aby se vozítko dostalo do bodu $\mathbf{X_k}$.

Výpočty jsou založeny na výpočtech hustot pravděpodobnosti s použitím podmíněné pravděpodobnosti. Dále s pomocí uchovávání předchozích pohybů a míst, kde jsme prováděli měření, můžeme snižovat nejistotu v lokaci vozítka vůči orientačním bodům. Výpočet takových to odhadů polohy probíhá vždy z předchozí pozice

Z praktického hlediska vyžaduje řešení problému zvolení vhodné reprezentace pohybového a pozorovacího modelu. Doposud nejvhodnější reprezentací je stavový popis systému s přidaným gaussovským šumem. Tato reprezentace vede na řešení pomocí rozšířených Kálmánových filtrů (EKF). Další alternativou je často využívaný FastSLAM algoritmus.

Je-li jako hlavní senzor pro řešení vizuálního SLAM problému je využita kamera nebo kombinace kamery s dalšími senzory. Kamery, které jsou k řešení vizuálního SLAM problému obvykle použity jsou RGB-D nebo stereoskopická kamera. V tomto projektu je použita pouze obyčejná jednočo4ková kamera. S použitím jedné kamery vyvstávají nové problémy. Zejména je potřeba vyřešit inicializaci a určení vzdálenosti orientačních bodů od kamery. Pro řešení monokulárního SLAMu se taktéž využívá rozšířených kálmánových filtrů.

3.4.3 Implementace

Tento balíček vyžadoval především překlad a instalaci potřebných aplikací – Pangolin[32], OpenCV[33], Eigen3[34] a DBoW2[35]. Pangolin slouží k vizualizaci mapy.Pro jeho spuštění na vzdáleném počítači pomocí technologie X11 bylo nutné ve zdrojovém kódu upravit nastavení framebufferu, jinak aplikace padala[36].

31 https://blog.acolyer.org/2015/11/05/simultaneous-localization-and-mapping-part-i-history-of-the-slam-problem/

32 https://github.com/stevenlovegrove/Pangolin

33 https://opencv.org/

34 http://eigen.tuxfamily.org/index.php?title=Main_Page

35 https://github.com/dorian3d/DBoW2

36 https://github.com/stevenlovegrove/Pangolin/issues/194

Dalším problémem řešeným především v této části pří překladu příkladů pro ROS a samotných zdrojových kódů, byl nedostatek paměti. To lze řešit swapem, jak již bylo zmíněno, nebo také snížením použitých jader CPU při překladu. V *Makefile* souboru pro překlad zdrojových kodů je tak na Raspberry Pi vhodné přepnout nativní možnost *-j4* na *-j2*.

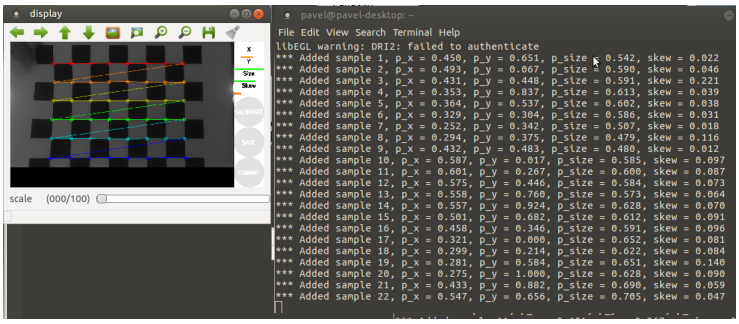
Další malou úpravou bylo přepsání původního příkladu pro monokulární kameru v ROS frameworku. Tato úprava spočívala v přidání *publisher* funkce, která umožňuje ostatním uzlům číst stav mapování. Stav je dostupný přes instanci ORB_SLAM2*::System. B*ylo třeba přepsat hlavní smyčku a přidat publikaci tohoto stavu na topic *orb_slam*.

Pro samotné spuštění balíku ORB-SLAM2 [6] je nutná kalibrace kamery, a zapsání této konfigurace do souboru, který je uzlu předán při spuštění. Druhým parametrem je slovník pro *Bag of Words* reprezentaci, který je vytvořen z velkého množství vzorků. Jedná se o jeden rozsáhlý generický slovník, jehož funkčnost byla ověřena vývojářem. Prováděním experimentů jak ve vnitřním, tak ve venkovním prostředí. Dále byla potvrzena i komunitou používající tento balíček pro řešení SLAM problému.

Pro správnou funkci toho balíku (správný odhad vzdáleností a korekci zkreslení) je nutné předat při spuštění matici pro korekci zkreslení, a matici popisující vztah jednotek (pixelů) používaných kamerou a jednotek skutečného světa (např. milimetry) .

Parametry zkreslení a převodu je možné získat kalibračním nástrojem pro ROS **camera_calibration**. Ten pro kalibraci vyžaduje šachovnici o určeném počtu polí a známé velikosti jednoho pole. Pomocí geometrických výpočtů určuje zmíněné matice. Takto získaná konfigurace je uložena do spouštěcího konfiguračního souboru kamery. Tyto údaje je nutné přepsat do konfiguračního souboru pro ORB-SLAM2, aby byl uzel schopen správně interpretovat vzdálenosti a schopen vyvažovat zkreslení. Rozlišení kamery bylo zvoleno 640x480 a to jak z důvodu výpočetní náročnosti, tak z důvodu přenosu videa do uživatelského rozhraní.

Uzel pro kameru *raspicam_node,* který je použit, publikuje video s kompresí. To není kompatibilní s ORB-SLAM2 aplikací. Uzel přijímá pouze raw video bez úprav a komprese. Pro zajištění správného formátu je použit nástroj, který poskytuje přímo ROS. Nástroj **image_transport** umožňuje transformovat video odstraněním komprese do raw formátu, a zaslat je na nový topic, zde *image_raw.*

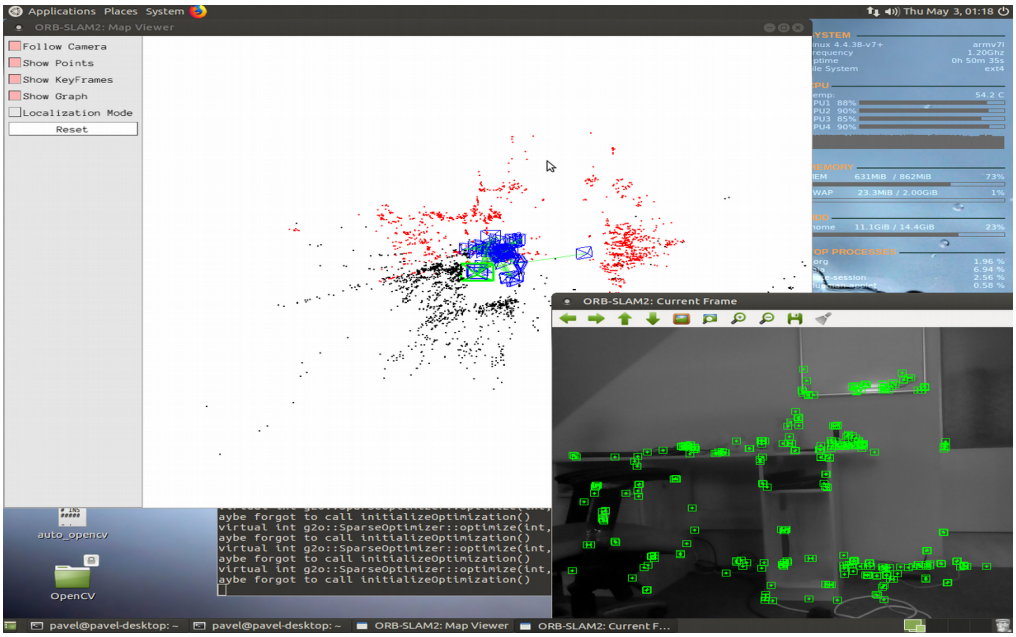


Obrázek 19: Kalibrace kamery

3.4.4 Testování

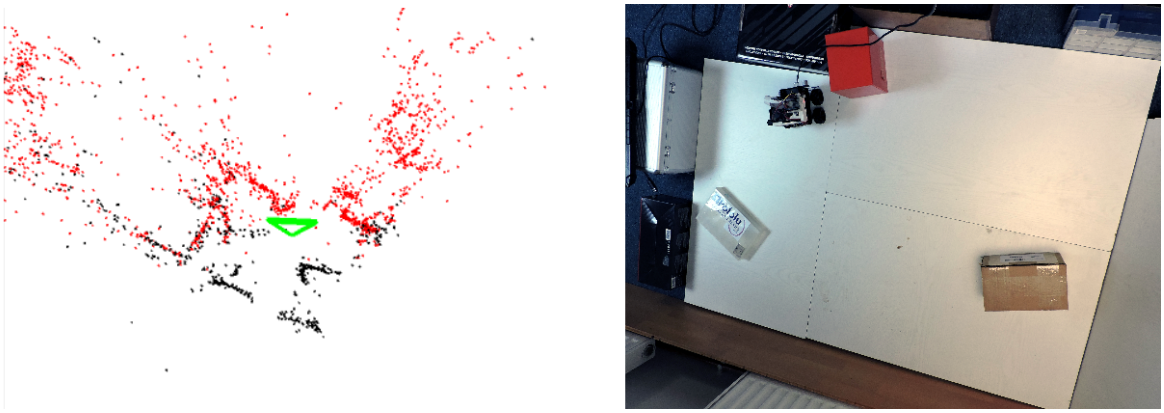
Experiment si klade za cíl otestovat především lokalizaci a mapování. Jedná se o celkový test, který umožní říci jak se vozítko chová v případě, že jsou spuštěny všechny ROS uzly.

Vozítko bude umístěno na náhodné místo. Pomocí uživatelského rozhraní budou spuštěny všechny uzly, které jsou zapotřebí: ORB-SLAM2, senzory včetně kamery, sériová komunikace s Arduinem. První pokus proběhl statickým otáčením na místě. Uzel se podařilo zinicializovat, a následně pomocí otáčení vozítka na místě, se podařilo vytvořit přibližnou podobu prostředí mého pokoje. Nicméně takto provedený pokus je zatížen množstvím chyb , protože kamera se otáčí na místě a neumožňuje zpřesnění vzdálenosti mezi jednotlivými pozorovanými body.



Obrázek 20: ORB-SLAM2 - statický test (rotace kolem y osy vozítka)

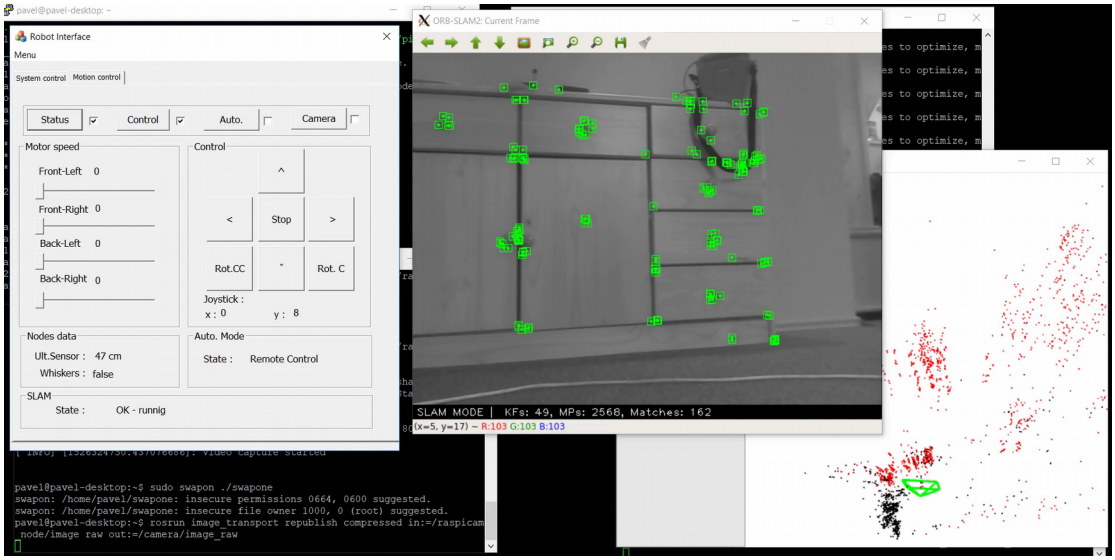
Dalším pokusem bylo menší prostředí na obrázku 21. V takovém prostředí nemělo vozítko problém s inicializací a výsledná vizualizace je velmi přesná. V tomto prostředí nebyl problém, pouze občasné ztráty kontextu, v případě rychlého otočení vozítka.



Obrázek 21: příklad ORB-SLAM2

Poslední pokus byl proveden na celé místnosti. Inicializace neproběhla úplně bez problémů, protože se ukázal vliv světla na inicializační proces a malé množství objektů v pokoji. Po inicializaci uzlu se ukázal další problém. Tímto problémem bylo umístění a směrování kamery. Kamera svírala úhel zhruba 90 stupňů se zemí a viděla tak velkou část země před sebou. To nebyl problém v případě menšího experimentu, kde podkladem byly bíle lakované desky. Nicméně v místnosti je koberec s chaotickým modrým vzorem. Tento vzor byl pro mapovací uzel velmi atraktivní a velké množství bodů bylo hledáno v oblasti, kde kamera zabírá koberec. To samozřejmě silně zkresluje výslednou mapu a výsledek je zcela nepoužitelný. Tento problém jsem částečně vyřešil umístěním kamery na nejvyšší možný bod konstrukce a mírným záklonem směrem od země.

I přes tyto úpravy nebyl výsledek příliš dobrý v místnostech s malým množstvím předmětů, případně s podobnými objekty a prostorovým rozmístěním. Mapování bylo úspěšné v menších částech takových to prostor. V ostatních částech prostoru se často stávalo, že použitá technologie mapovala nové body na již známé části. Tím obvykle znehodnotila celé mapování a po delší době ztrácelo mapování často kontext.



Obrázek 22: Manuální řízení - ORB-SLAM2

4 Závěr

Cílem práce bylo navrhnout a zrealizovat vozítko pro mapování malých a špatně dostupných prostor, které bude poskytovat, jak manuální , tak autonomní ovládání.

Řešení jsem rozdělil na několik dílčích částí: konstrukce a obvodové zapojení, manuální ovládání, autonomní režim, vytváření mapy a lokalizace.

Výsledné řešení se skládá z Raspberry Pi, vývojové desky Arduino a dalších komponentů, které poskytly prostředí pro implementaci softwarové části projektu. Raspberry Pi se i přes počáteční obavy (ohledně přehřívání nebo nedostatku výkonu) ukázalo jako funkční a dostačující pro tuto aplikaci. Arduino splnilo plně svoji funkci. Rozšiřující deska pro řízení motorů fungovala dle očekávání, ale na trhu je již novější verze. Tato verze by byla lepší především z prostorového hlediska, protože použitá deska je osazena starým typem součástek. Tyto součástky jsou prostorově náročnější a vozítko musí být vyšší. To zhoršuje stabilitu a zvyšuje těžiště. Regulátor napětí je postačující, ale pro dlouhodobé použití by vyžadoval pasivní chlazení.

Softwarová část řešení zahrnovala implementaci některých uzlů pro ROS framework a grafické uživatelské rozhraní. Uzly pro senzory byly triviální záležitostí a vyžadovaly pouze jednoduchou aritmetiku a zvládnutí principu použití GPIO na Raspberry Pi. Uzel pro autonomní režim, který umožňuje manuální a autonomní ovládání. Nedostatkem v této části může být absence automatu, který by byl schopen ošetřit stavy uzlu pro mapování a lokalizaci. Informace o stavu mapování je ve zjednodušeném stavu dostupná, ale je využita pouze jako stavová informace v uživatelském rozhraní. Vhodným doplněním by byl automat, schopný inicializovat mapovací uzel a v případě ztráty kontextu převzít řízení a zotavit se ze ztráty.

Uživatelské rozhraní je dostačující z hlediska všech potřeb vzdáleného ovládání. Nicméně použitá implementace obalující knihovnu pro práci s videem, se v případě špatného signálu bezdrátové sítě občas zasekla, a zastavila tak vykonávání celého programu. Rozhraní realizované pomocí SSH se ukázalo (v případě do implementování rozhraní na straně vozítka) jako takřka rovnocenné Rosbridge rozhraní.

Jednotlivé části byly testovány postupně dle implementace. Testy provedené na jednotlivých částech jsou dokumentovány na konci jednotlivých celků. Použité technologie a postupy , které byly v práci navrženy s různou mírou úspěšnosti, splnily vytyčené cíle. Použitá technologie mapování a lokalizace, by pravděpodobně lépe fungovala s lepšími senzory i v prostředí s horšími světelnými podmínkami.

Literatura

[1] DAVISON, Andrew J. SLAM with single camera. Dostupné z : https://www.doc.ic.ac.uk/~ajd/Publications/davison_cml2002.pdf

[2] GAUTHAM P., JACOB G. Real-time ROSberryPi SLAM Robot. Dostupné z : https://courses.cit.cornell.edu/ece6930/ECE6930_Spring16_Final_MEng_Reports/SLAM/Real-time%20ROSberryPi%20SLAM%20Robot.pdf

[3] DAVISON, Andrew J. MonoSLAM. Dostupné z : https://www.doc.ic.ac.uk/~ajd/Publications/davison_etal_pami2007.pdf

[4] NEDUCHAL Petr. Návrh a testování metod vizuální simultánní lokalizace a mapování. Dostupné z : https://otik.uk.zcu.cz/bitstream/11025/7432/1/dp_prace.pdf

[5] BROOKS, Rodney A. A robust layered control systém for a mobile robot. Dostupné z : http://www.dtic.mil/dtic/tr/fulltext/u2/a160833.pdf

[6] MUR-ARTAL, Raúl, MONTIEL, J. M. M. and TARDOS, Juan D., ORB-SLAM: a Versatile and Accurate Monocular SLAM System . Dostupné z : http://webdiis.unizar.es/~raulmur/MurMontielTardosTRO15.pdf

Přílohy

A – Náklady

Jedním z vedlejších cílů bylo omezit náklady na stavbu vozítka použitím již zakoupených komponentů, případně koupí levných variant konkrétní elektroniky. Pro zajímavost připojuji tabulku s výslednou cenou.

Komponenta	**Cena v Kč**
Raspberry Pi v3.0	869
Arduino UNO rev.3	600
Shield pro ovládání mot.	160
Step-down reg.	100
Kamera	270
Ultrazvukový senzor	45
Měnič log. úrovní	30
Kabeláž a spoj. mateiál	50
3D tisk (materiál)	100
Motory	240
Celková cena :	**2464,- Kč**