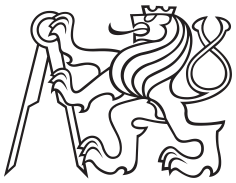
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra měření

LIDAR senzor a jeho aplikace v mobilním robotu

Pavel Doubrava

Vedoucí: Ing. Vojtěch Petrucha, Ph.D.
Obor: Kybernetika a Robotika
Květen 2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Doubrava** Jméno: **Pavel** Osobní číslo: **465959**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Kybernetika a robotika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

LIDAR senzor a jeho aplikace v mobilním robotu

Název diplomové práce anglicky:

LIDAR sensor and its application in a mobile robot

Pokyny pro vypracování:

Navrhněte konstrukci rotačního laserového dálkoměru umožňujícího detekovat volný prostor v úhlu 360° v blízkém okolí mobilního robota.
Konstrukci realizujte, otestujte a začleňte do systému robota.
Zprovozněte řídicí elektroniku robota a implementujte libovolnou demonstrační funkci využívající LIDAR.

Seznam doporučené literatury:

[1] Santiago Royo and Maria Ballesta-Garcia: "An Overview of Lidar Imaging Systems for Autonomous Vehicles," Appl. Sci. 2019, 9, 4093; doi:10.3390/app9194093
[2] Paul F. McManamon: "LiDAR technologies and systems," SPIE Press 2019, ISBN:1510625399, 9781510625396
[3] Ahmet Bindal: "Electronics for Embedded Systems", Springer International Publishing, Switzerland 2017

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Vojtěch Petrucha, Ph.D., 13138

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

Datum zadání diplomové práce: **13.01.2021** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce:
do konce letního semestru 2021/2022

Ing. Vojtěch Petrucha, Ph.D.	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce		podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. _____ Podpis studenta

Datum převzetí zadání

Poděkování

Chtěl bych poděkovat panu Ing. Vojtěchu Petruchovi, Ph.D. za věcné připomínky, odborné vedení, vstřícnost a trpělivost při konzultacích. Dále bych chtěl poděkovat své rodině a přátelům za jejich podporu.

Prohlášení

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

V Praze, 15. května 2021

Abstrakt

Cílem předložené diplomové práce je návrh a realizace senzorové jednotky umožňující mapovat okolní prostředí pomocí rotujících lidar senzorů vzdálenosti.

Je popsáno dokončení konstrukce, začlenění senzorové jednotky a zprovoznění elektroniky robotické platformy na základě open-source projektu SAWPPY.

Je zde popsán celý proces vývoje senzorové jednotky a návrh softwaru robotické platformy.

Elektronika platformy je složena z několika specializovaných modulů řízených mikroprocesory z rodiny STM32 a hlavní řídicí jednotky Raspberry pi 4, na které je vytvořena řídicí aplikace v prostředí ROS.

Následně bylo provedeno otestování a hodnocení funkcí robota.

Klíčová slova: STM, Lidar, ROS, SAWPPY, rover, VL53L1x

Vedoucí: Ing. Vojtěch Petrucha, Ph.D.

Abstract

The submitted master thesis aims to design and implement a sensor unit enabling the mapping of the surrounding environment using rotating LIDAR distance sensors.

The construction is described, the sensor unit is integrated, and the electronic modules of the robotic platform are put into operation on the basis of the open-source project SAWPPY.

The whole process of sensor unit development and software design of the robotic platform is described here.

Electronics of the platform consists of several specialized modules controlled by microcontroller from the STM32 family and the Raspberry pi 4 central control unit, which runs a control application in the ROS environment.

Subsequently, the functions of the robot were tested and evaluated.

Keywords: STM, Lidar, ROS, SAWPPY, rover, VL53L1x

Title translation: LIDAR sensor and its application in a mobile robot

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod a motivace

Dobývání vesmíru bylo odjakživa touhou lidstva. Z důvodu vesmírné radiace a dalších nepříznivých vlivů, neslučitelných s lidským životem jsou pro první průzkumy vesmírných těles používány různé sondy a vozítka.

Pro prozkoumávání těles jsou navrhovány specializované výzkumné sondy umožňující odběr a analýzu vzorků z povrchu. Při návrhu konstrukce sondy je kladen důraz na robustnost rámu a schopnost pohybovat se v členitém terénu. Sonda musí být vybavena dostatečnou senzorovou základnou pro určení optimální trajektorie neznámým terénem, aby nedošlo k jejímu poškození nebo uvíznutí.

Dálkové ovládání sond v reálném čase z důvodu doby letu řídicího signálu mezi ovládací stanicí a sondou není možné (odezva se může pohybovat i v řádu hodin). Sondy se proto musí dokázat samy rozhodovat a orientovat v neznámém prostředí. Jednou z aktuálních používaných metod pro zisk 3D mapy prostředí jsou senzory na principu doby letu světla (tzv. lidary). Tato technologie je aplikována i v autonomních automobilech.

V rámci předmětu Projekt v týmu ve druhém semestru magisterského studia jsme s kolegy vytvářeli klon robotické platformy open-source projektu SAWPPY the rover, který je inspirován sondou Curiosity mapující povrch Marsu.

Cílem našeho projektu bylo vytvořit robotickou platformu určenou pro prezentaci fakulty (Noc Vědců, Den otevřených dveří, Dětská univerzita, atd.). Z důvodu podcenění náročnosti projektu a nástupem opatření spjatých s nemocí covid-19 nebyl tento projekt dokončen.

V rámci mé diplomové práce jsem se zaměřil na vytvoření senzorové jednotky, která by umožnila mapovat okolní prostředí pomocí rotujících lidar senzorů vzdálenosti VL53l1x od firmy ST Microelectronics. Jednotka dále umožňuje určit natočení robota v prostoru (zisk Eulerových úhlů).

Další částí zadání mé diplomové práce bylo dokončení robotické platformy. Zde bylo potřeba oživit všechny řídicí desky plošných spojů (DPS), následně pro ně implementovat software a otestovat funkci celého zařízení.

Kapitola 2
Teoretický rozbor

Nejdříve bych chtěl popsat technologie a postupy, které jsem v mé diplomové práci použil. Kapitola má sloužit k vysvětlení základů. Podrobné vysvětlení je dostupné v citovaných zdrojích. Popisovanými tématy jsou:

- **Sběrnice CAN bus** používaná pro komunikaci jednotlivých modulů robota
- **Lidar senzory** vzdálenosti
- **Fúze dat** pro AHRS (Attitude and Heading Reference Systems) a s tím spojená **kalibrace senzorů**

2.1 CAN bus

CAN bus je rozšířená sběrnice, která původně vznikla pro vnitřní komunikační síť senzorů a jednotek v automobilech. Z této oblasti se CAN bus následně rozšířil do dalších odvětví jako je například průmyslová automatizace, řízení výtahů, modelářství a automatizace budov.

2.1.1 Historie

Komunikační protokol byl publikován na konferenci Society of Automotive Engineers roku 1986. Roku 1991 byl realizován první automobil využívající sběrnice založené na CANu.

Firma Bosh publikovala několik verzí specifikace CAN, poslední verzí je CAN 2.0 vydaný roku 1991. Tato specifikace má dvě části: část A (CAN 2.0A) definující standardní 11-bitový identifikátor, část B definující rozšířený 29-bitový identifikátor zpráv. Oba standardy jsou volně dostupné na webových stránkách firmy Bosch [2].

Od roku 1993 ISO vydala několik standardů zahrnujících a dále rozšiřujících původní specifikace od firmy Bosch.

Firma Bosch nadále pokračuje s rozšiřováním specifikací této sběrnice. V roce 2012 vydala specifikaci CAN FD 1.0 (Flexible Data-rate), která upravuje strukturu rámce, čímž umožňuje přenos většího objemu dat a volitelně změnu přenosové rychlosti po odeslání hlavičky. Uzly pracující podle specifikace

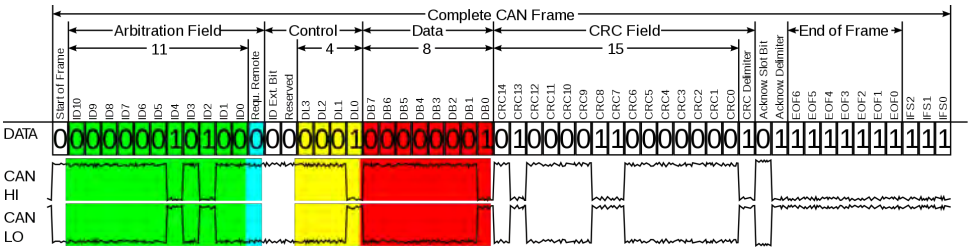
CAN FD mohou pracovat na stejné sběrnici s uzly podle specifikace CAN 2.0. [9]

2.1.2 Popis

Sběrnice funguje na principu producent/konzument. Jeden uzel vysílá zprávu (rámec) s pevně daným identifikátorem, který určuje data zprávy. Všechny ostatní uzly mohou danou zprávu přijmout, pokud daný obsah potřebují. Každý uzel musí vysílat rámce s unikátními ID. Dva uzly nemohou posílat rámce se stejným ID. Pokud by dva uzly posílaly zprávu se stejným ID, nefungoval by používaný algoritmus přístupu k fyzickému médiu.

Hardwarová vrstva protokolu je diferenciální dvoulinka (CAN_L, CAN_H) s charakteristickou impedancí 120 Ω (viz obrázek 2.3). Zde poté rozlišujeme dva stavy: recesivní a dominantní. Bez zásahu vysílačů je na sběrnici recesivní úroveň (logická 1), která je reprezentována stejnou napěťovou úrovní na obou vodičích. Toho je docíleno pomocí rezistorů spojujících vodiče. Dominantní úrovně (logická 0) je docíleno pomocí zásahu libovolného uzlu. Na sběrnici je tato úroveň reprezentována rozdílným napětím mezi CAN_L a CAN_H (snížení napětí na CAN_L a zvýšení na CAN_H).

Pro high speed CAN, rychlosti do 1 Mbit/s je použita diferenciální dvoulinka, která je na dvou bodech nejdále od sebe propojena rezistorem 120 Ω. Jestliže jakýkoliv uzel vysílá dominantní úroveň nominální napětí na signálech je $V_{CANH} = 3{,}5\,\mathrm{V}$ a $V_{CANL} = 1{,}5\,\mathrm{V}$. Nominální diferenciální napětí pro dominantní úroveň činí 2 V. Při návratu vysílacího uzlu do recesivní úrovně se sběrnice pomocí rezistorů propojující dvoulinku vrátí do klidového stavu. Maximální diferenciální napětí pro recesivní úroveň činí 0,5 V a minimální diferenciální napětí pro dominantní úroveň je 1 V.



Obrázek 2.1: CAN bus: Ukázka rámce [1]

Podoba zprávy je přesně daná. Na obrázku 2.1 je ukázka popsaného rámce pro specifikaci CAN 2.0A. V tabulce 2.1 jsou poté popsány jednotlivé bity. Jestliže navržený systém potřebuje více identifikátorů zpráv než umožňuje standartní 11-bitové ID zprávy (CAN 2.0A), je možné použít rozšířené 29-bitové ID (extended ID/extID) specifikace CAN 2.0B.

Přístup ke komunikační sběrnici CAN bus je řízen protokolem CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). Uzel může začít vysílat jen tehdy, když na sběrnici neprobíhá žádný přenos. Uzel musí sledovat stav sběrnice při vysílání zprávy, jestliže logická úroveň na sběrnici neodpovídá jím vysílanému bitu, uzel musí neprodleně přerušit vysílání. Tímto pravidlem

Název	Délka (bits)	Popis
Start-of-frame	1	Značí začátek rámce (0)
Identifier (green)	11	Unikátní identifikátor, který také určuje prioritu zprávy.
Remote transmission request (RTR) (blue)	1	Dominantní (0) pro datové rámce a recesivní (1) pro remote request rámce
Identifier extension bit (IDE)	1	Dominantní (0) pro rámce se standartním 11-bitovým ID
Reserved bit (r0)	1	Reservovaný bit musí být dominantní (0)
Data length code (DLC) (yellow)	4	Délka dat v bytech [0-8]
Data field (red)	0–64 (0-8 bytes)	Data (délka je určena předchozím řádkem)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Musí být recesivní (1)
ACK slot	1	Odesilatel posílá recesivní (1), přijímač může potvrdit přijmutí pomocí přepsaní bitu na dominantní úroveň (0)
ACK delimiter	1	Musí být recesivní (1)
End-of-frame (EOF)	7	Musí být recesivní (1)

Tabulka 2.1: CAN bus: Popis bitů rámce se standartním ID [1]

je docíleno prioritizace zpráv podle jejího ID. Vždy je odvysílána zpráva s menším ID (dominantní úroveň přepíše recesivní), viz složení zprávy v tabulce 2.1. Praktická ukázka principu přístupu k fyzickému médiu je demonstrována v tabulce 2.2. Uzel A se snaží odeslat zprávu s ID 0x2ff a uzel B 0x1ff. [2].

	Start bit	Bity ID										
		10	9	8	7	6	5	4	3	2	1	0
Uzel A	0	0	1	odesílání přerušeno								
Uzel B	0	0	0	1	1	1	1	1	1	1	1	1
CAN bus	0	0	0	1	1	1	1	1	1	1	1	1

Tabulka 2.2: CAN bus: Demonstrace CSMA/CA

2.1.3 Časování

Všechny uzly na sběrnici musí používat stejnou přenosovou rychlost, která není na sběrnici přenášena pomocí vlastního signálu. Nepřesnosti ve výrobě, časový/teplotní drift oscilátorů, prostředí diferenciální dvoulinky a další vlivy způsobují, že přenosová rychlost nebude přesně ta nominální. Každý uzel má vlastní generátor hodinového signálu, je tedy potřeba zaručit, jak synchronizaci začátků rámců (fáze), tak i variaci v přenosové rychlosti.

Synchronizace probíhá při každé změně z recesivní na dominantní úroveň, počínaje prvním bitem *start-of-frame* (viz tabulka 2.1).

K docílení synchronizace je každý bit dále rozdělen na tzv. časová kvanta, která jsou podle standartu rozdělena do 4 fází: [10]

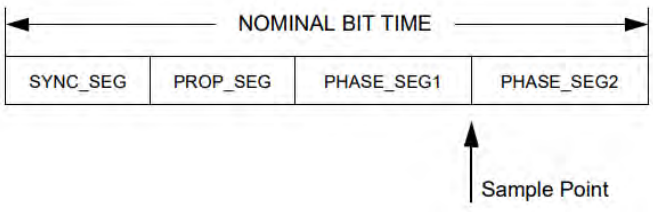
- **Synchronizace** - V této části by se měly synchronizovat jednotlivé uzly. Zde by se měla nacházet hrana, pokud dochází k změně logické úrovně signálu.
- **Propagace** - Část určena ke kompenzaci vlivu zpoždění na sběrnici.
- **Fáze 1** a **Fáze 2** - Části určené pro kompenzaci variace přenosové rychlosti jednotlivých uzlů.

K okamžiku interpretace logické úrovně bitu dochází mezi fázemi 1 a 2. [2].

Hodnota časového kvanta je určena hodinovým signálem budícího CAN driver. CAN driver obsahuje prescaler, který dokáže délku časového kvanta prodloužit.

Počet časových kvant může být v průběhu přenosu upravován na základě předchozích přenosů. K fázi 1 může být přidáno časové kvantum a od fáze 2 může být odebráno, pro kompenzaci rozdílů přenosových rychlostí jednotlivých uzlů.

Na obrázku 2.2 je toto dělení znázorněno.



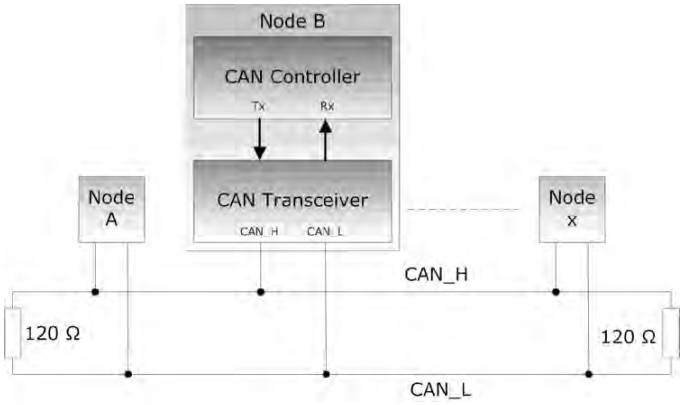
Obrázek 2.2: CAN bus: Rozdělení bitu na časové kvanta [2]

2.1.4 Struktura uzlu

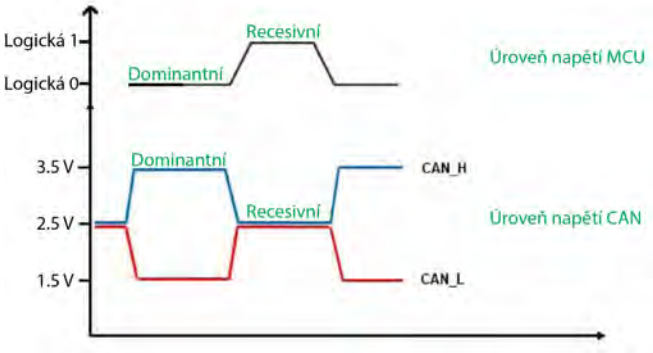
Každý uzel se skládá z těchto částí (viz. obrázek 2.3):

- **Řídící jednotka** - Například mikroprocesor nebo FPGA.
- **CAN controller** - Integrovaný obvod, zajišťující správu přístupu k fyzickému médiu. Jednotka spravuje přenos a příjem zpráv, na základě požadavků od řídicího prvku. Většina současných mikrokontrolerů (např. rodina STM32) má již CAN controller integrovaný v MCU.
- **CAN transceiver** - Integrovaný obvod podle ISO 11898-2/3, překládající signály (Rx, Tx) od CAN controlleru na signály diferenciální dvojlinky (CAN_H + CAN_L) a naopak. Obsahuje také ochranné obvody, aby nebyly poškozeny řídicí obvody.

Překlad signálů CAN controlleru na diferenciální je znázorněn na obrázku 2.4. [1]



Obrázek 2.3: CAN bus: Připojení uzlů na sběrnici [1]



Obrázek 2.4: CAN bus: Překlad signálů CAN controlleru na diferenciální [3]

2.2 Lidary

Senzory typu Light Detection and Ranging zkráceně Lidar jsou senzory měření vzdálenosti na základě doby letu světelného paprsku (tzv. Time-of-Flight, zkráceně ToF).

Senzory obvykle využívají infračervenou část spektra. Senzory jsou často používány pro mapování prostoru. Výsledkem měření je tzv point-cloud (mračno bodů), který tvoří množina dvojic měřeného směru a naměřené vzdálenosti. Pomocí mračna bodů můžeme následně například vygenerovat 3D model předmětů (budova, auto ...). Dalším využitím je mapování neznámého prostředí a orientace v něm. Další kapitoly byly vypracovány v souladu s článkem [11].

2.2.1 Principy měření vzdálenosti

Pro měření vzdálenosti se mohou použít tři přístupy:

- Pulsní
- Amplitudová modulace nosné vlny (AMCW)

- Frekvenční modulace nosné vlny (FMCW)

Jejich srovnání naleznete v tabulce 2.3.

Pulsní přístup je nejjednodušší. Vzdálenost objektu d je určena pomocí času letu pulzu t a rychlosti světla v mediu c (viz rovnice 2.1). Metoda dosahuje přesností v řádu cm. Jejími hlavními výhodami jsou jednoduchost návrhu elektroniky, odolnost proti silnému slunečnímu pozadí. Hlavními nevýhodami jsou dosah měření kvůli nízkému SNR při delších vzdálenostech v souvislosti s omezením maximálního vysílaného výkonu z důvodu bezpečnosti (poranění očí) a měření času s přesností na setiny nanosekund pro přesnost v řádu cm.

$$d[\mathrm{m}] = \frac{c[\mathrm{m\,s^{-1}}]}{2} \cdot t[\mathrm{s}], \tag{2.1}$$

Při použití amplitudové modulace průběžné vlny je doba letu určena pomocí fázového posunu vysílaného a přijímaného signálu. Senzory AMCW nabízejí podobné rozlišení jako pulsní, umožňují průběžné měření, ale jsou více citlivé na hodnotu SNR (při pulzní metodě může mít vysílaný pulz větší intenzitu), z toho důvodu je složité senzory AMCW použít ve venkovních aplikacích.

Při použití frekvenční modulace průběžné vlny je doba letu určena pomocí tzv. beat frekvence, která je získána pomocí zpracování smíšeného vysílaného a přijímaného signálu. Tato frekvence je proporcionální k době letu. Návrh senzorů tohoto typu umožňuje větší měřenou vzdálenost a přesnost při použití stejného vysílacího výkonu, než předchozí dvě kategorie z důvodu vyšší odolnosti vůči rušení. Pro měření vzdálenosti musí být použit koherentní paprsek. Pomocí zpracování signálu FFT na základě Doplerova jevu je možné získat i rychlost pohybu měřeného objektu. Nevýhodou těchto senzorů je složitost jejich elektroniky (koherentní paprsek, zpracování signálu). Koherentní systém senzoru musí mít stabilní pracovní podmínky (teplotní drift, linearita elektroniky), aby byl spolehlivý. To je významné pro aplikace, vyžadující robustnost a opakovatelnost měření po dobu několika let.

	Pulsní	**AMCW**	**FMCW**
Měřená veličina	Intensita pulzů	Fáze AM signálu	Doplerův frekvenční posun, beat frekvence
Způsob měření	Přímé	Nepřímé	Nepřímé
Světelný paprsek	Nekoherentní	Nekoherentní	Koherentní
Použití	Indoor/Outdoor	Indoor	Indoor/Outdoor
Výhody	Jednoduchost návrhu	Komerčně zavedeno	Současné měření vzdálenosti a rychlosti
Omezení	Nízke SNR příchozího pulzu	Omezný rozsah určen periodou modulace	Stabilita koherentního signálu
Rozlíšení	1 cm	1 cm	0.1 cm

Tabulka 2.3: Lidar: Srovnání principů měření vzdálenosti.

2.2.2 Principy mapování prostředí

Bylo představeno několik strategií, jak mapovat prostředí pomocí lidarů. Tyto strategie zisku obrazu můžeme rozdělit do tří kategorií: skenery, detektorové pole a kombinové přístupy. Porovnání jednotlivých popisovaných metod je v tabulce 2.4.

	Princip funkce	**Hlavní výhody**	**Hlavní nevýhody**
Mechanické skenery	rotující zrcadla nebo hranoly	Horizontální zorný úhel 360°	Pohyblivé části, velikost
MEMS Skenery	MEMS mikrozrcadla	Kompaktní	Ovládání výkonu paprsku, linearita
OPA	Fázové pole zdrojů	Bez mechanických prvků	Jen krátké a střední vzdálenosti
Flash	Pulsní "flood"iluminace	Vysoký frame rate	Omezený dosah, oslnitelné
AMCW pole	CMOS	Komerční sensory	Jen indoor aplikace

Tabulka 2.4: Lidar: Srovnání principů mapování prostředí.

Skenery

Generalizovaně řečeno kategorie skenerů používá nějakou skenující metodu, která mění směr měřícího paprsku senzoru, za účelem vygenerování mračna bodů.

V současné době je používáno několik skenujících metod. To zahnuje i galvanometrická zrcadla nebo Risley hranoly. V běžných aplikacích nalezneme tři metody:

- **Mechanické skenery** - Používají rotující zrcadla a galvanometricky nebo piezoelektricky nastavitelná zrcadla a hranoly. Většinou používají pulsní zdroje. Systémy tohoto typu mohou měřit vzdálenosti více jak 1 km a dosahují obnovovací frekvence do 100 Hz. Nevýhodami těchto senzorů jsou citlivost vůči otřesům a vibracím, vysoká spotřeba, složitá rozšiřitelnost, rozměrnost, opotřebení pohyblivých části a vyšší cena. Senzory vyrábí například firma Velodyne.

- **Micro electromechanical systems (MEMS)** - Ke směřování paprsku používají drobná zrcadla (řádově jednotky mm) ovládané pomocí elektromagnetických nebo piezoelektrických aktuátorů, často doplněných o rozšiřující optiku. Senzory typu MEMS nahrazují mechanický hardware předchozí metody pomocí mikroelektromechanického ekvivalentu. Tím ale dochází ke zmenšení zorného úhlu systému z důvodu absence rotačního pohybu. Tato nevýhoda může být odstraněna použitím systému s

více senzory. Mezi výhody senzoru se řadí nízká spotřeba, cena a malá velikost, oproti předchozímu typu.

- **Optical phased Arrays** - Je nová metoda založená na směřování paprsku pomocí interference několika zdrojů světla. Metoda funguje na stejné principu jako mikrovlnová fázová pole. Zařízení nepoužívá žádné pohyblivé části, to přináší vysokou mechanickou odolnost vůči nárazům a vibracím. Senzory jsou schopny skenovat prostor frekvencemi přes 100 kHz. Je možné je integrovat do malého čipu. Omezujícím faktorem v současné době je možný vysílaný výkon, který neumožňuje použití senzoru pro větší vzdálenosti. Senzory pro měření velkých vzdáleností jsou stále ve vývoji, na trhu jsou nabízený senzory pro krátkou a střední vzdálenost. Kombinace technologií OPA a FMCW má do budoucna vysoký potenciál, co se týče spolehlivosti a nízké ceny.

Detektorová pole

Kvůli malé popularitě senzorů s pohyblivými částmi a možnosti měřit vždy vzdálenost jen v jednom směru, byly navrženy alternativní metody, které kombinují specializované strategie osvitu a pole přijímačů. Vysílací elementy osvítí celou scénu a lineární pole/matice detektorů paralelně přijímá signály rozdělené do samostatných úhlových sektorů. Tato metoda umožňuje získat všechny měřené pozice při jednom osvitu. Zdroje osvícení scény můžeme dělit na pulsní (flash imagers) a průběžné (AMCW nebo FMCW maticové pole senzorů). S výjimkou metody FMCW, kde koherentní přístup umožňuje dlouhý dosah, jsou senzory (používající pulzní nebo AMCW metodu měření) krátkého až středního dosahu.

- **Flash imagers** - Současné senzory tohoto typu používané v automotive průmyslu mají dosah 20 - 150 m. Jejich velkým omezením je regulace vysílaného výkonu pro ochranu očí. Jednou z jejich největších nevýhod je rozdílná reflexe jednotlivých povrchů, která může způsobit oslnění senzoru a tím znehodnotit naměřená data (odrazky, reflexní pásky atd.). Mezi hlavní výhody se řadí vysoká vzorkovací rychlost a z toho pramenící vysoká odolnost vůči vibracím a pohybům.

- **AMCW maticové pole** - Druhou kategorií jsou senzory používající metodu AMCW. Tyto senzory používají pro příjem klasické CMOS detektory s dostatečně vysokou snímací frekvencí. Z toho pramení nízká cena a velikost. Senzory měří krátké vzdálenosti (pár centimetrů až jednotky metrů). Vzhledem k použité technologii jsou senzory určeny pro vnitřní aplikace. Nalezneme je například v robotice, počítačovém vidění nebo v kinect senzoru platformy XBOX.

Kombinované přístupy

Kombinované přístupy kombinují předchozí dvě kategorie. Kombinují například rotační přístup s vertikálním polem pro paralelizaci měření a zvýšení

obnovovací frekvence.

2.3 Fúze dat z inerciální jednotky (9 DoF)

Přesné určení orientace je v mnoha odvětvích klíčové (např. robotika, letectví, kosmonautika nebo navigace). K určení těchto dat existuje mnoho přístupů. Systémy založené na inerciálních jednotkách mají velkou výhodu v nulových nárocích na okolní infrastrukturu a z toho pramenící plné soběstačnosti. Inerciální jednotka (dále jen IMU) se skládá z tří-osého akcelerometru a gyroskopu. Ty umožní jednotce mapovat rotační a translační pohyby. Inerciální jednotky můžou být doplněny o magnetometr. V ten okamžik se bavíme o tzv. Magnetic, Angular rate and Gravity (MARG) systémech. Samostatná inerciální jednotka je schopná určit polohu vzhledem k vektoru Zemské gravitace, což je v mnoha aplikacích dostatečné. Systémy založené na senzorech MARG určují směr vzhledem k vektoru gravitace a magnetického pole. Systémy zpracovávající data z MARG za účelem zisku orientace se běžně nazývají zkratkou AHRS (Attitude and Heading Reference Systems).

Gyroskop měří úhlovou rychlost, která může být integrována v závislosti na čase za účelem zisku orientace senzoru. Při použití samostatného gyroskopu je nutné určit počáteční podmínky orientace. Přesné gyroskopy jsou velmi nákladné a objemné pro aplikaci v menších systémech (např. civilní a modelářské drony). Ve většině aplikací se proto používají méně přesné senzory (nelinearita, časový drift, teplotní drift atd.) typu MEMS (Micro Eletrical Mechanical System). Použití takového nepřesného gyroskopu samostatně za účelem zisku orientace by vedlo k integrování chyby, která znemožňuje použití senzoru samostatně. Zde přichází na řadu magnetometr a akcelerometr, které mohou poskytnout absolutní referenci. Tyto senzory, ale podléhají vysokému rušení (akcelerometr měří gravitační zrychlení Země ale i zrychlení objektu, magnetometr je ovlivněn okolními magnetickými zdroji). Úlohou filtračního algoritmu je provést optimální fúzi dat z akcelerometru, magnetometru a gyroskopu za účelem zisku odhadu orientace.

Metody fúze dat můžeme rozdělit do dvou hlavních kategorií. Metody používající principy Kalmanovy filtrace a alternativní metody.

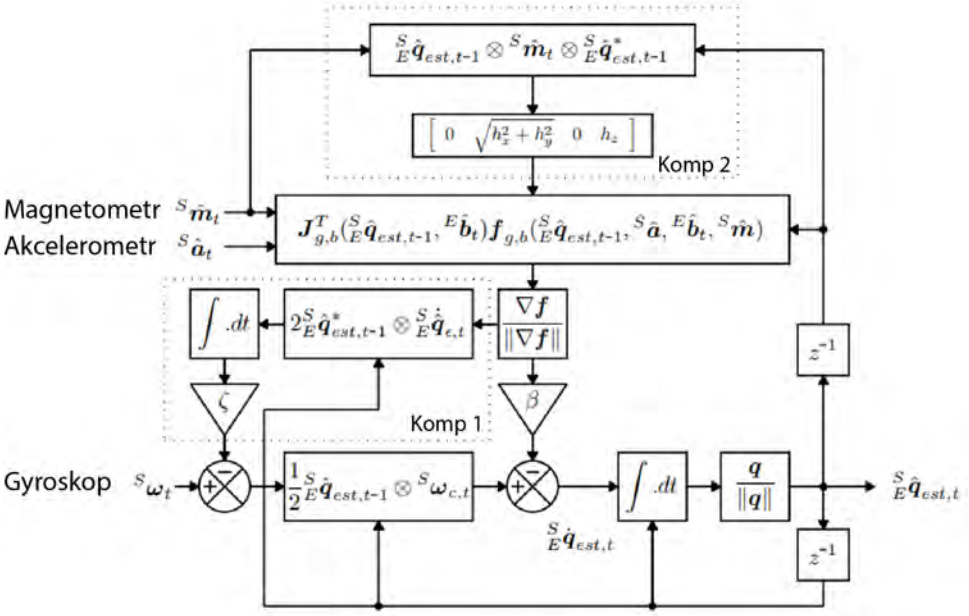
Kalmanova filtrace je základem pro mnohé komerčně používané filtrační algoritmy. Používají ji například senzory firem xSens, Micro-strain a Intersense. Rozšířenost řešení na bázi Kalmanovy filtrace je potvrzením její funkčnosti a efektivity. Pro moji aplikaci tyto metody mají jednu významnou nevýhodu a tou je výpočetní náročnost. Obnovovací frekvence algoritmu musí být výrazně vyšší než je rychlost pohybu. Například pro mapování lidského pohybu se doporučují frekvence od 512 Hz do 25 kHz[12]. (Magnetometr, který jsem použil je schopný dodávat data maximálně s frekvencí 300 Hz.)

Tato omezení inspirovala mnoho vědců pro vývoj alternativních algoritmů fúze dat. Nejdříve vznikly algoritmy používající tzv. "fuzzy processing"nebo pevné filtry používající data z akcelerometru a magnetometru při pomalých pohybech a gyroskop při rychlých. V současné době jsou nejvíce rozšířeny dva algoritmy nazvané podle jejich tvůrců: Mahonny a Madgwick. Oba algoritmy

jsou výrazně méně výpočetně náročné než metody s Kalmanovou filtrací. Na základě článku [12] srovnávajícího tyto metody, jsem se rozhodl použít metodu Madgwick, která dosahuje přesnějších výsledků za cenu nepatrného zvýšení výpočetní náročnosti oproti filtru Mahonny.

Použitá metoda využívá algoritmu gradient-descent (klesání po gradientu) umožňující ji provozovat na nižších frekvencích. Dále kompenzuje vlivy magnetického zkreslení a offsety gyroskopu. Použitý algoritmus je publikován v článku [4].

Blokový diagram principu funkce kompletního Madgwickova filtru je uveden na obrázku 2.5. Algoritmus pro reprezentaci polohy používá quaterniony.



Obrázek 2.5: Fúze dat: Blokový diagram Madgwickova orientačního algoritmu pro senzory typu MARG. [4]

2.3.1 Gyroskop

Data z gyroskopu ω jsou integrována za účelem zisku odhadu nové polohy senzoru. Orientace souřadnicového systému Země (E) vzhledem k systému senzoru (S) je odhadována vztahem:

$$ {}^{S}\omega = \begin{bmatrix} 0 & \omega_x & \omega_y & \omega_z \end{bmatrix} \tag{2.2} $$

$$ {}^{S}_{E}\dot{\mathbf{q}}_{\omega,t} = \frac{1}{2} {}^{S}_{E}\hat{\mathbf{q}}_{est,t-1} \otimes {}^{S}\omega_t \tag{2.3} $$

$$ {}^{S}_{E}\mathbf{q}_{\omega,t} = {}^{S}_{E}\hat{\mathbf{q}}_{est,t-1} + {}^{S}_{E}\dot{\mathbf{q}}_{\omega,t}\Delta t, \tag{2.4} $$

kde ${}^{S}_{E}\mathbf{q}_{\omega,t}$ je odhad orientace na základě dat z gyroskopu, ${}^{S}_{E}\hat{\mathbf{q}}_{est,t-1}$ je výsledný odhad orientace z předchozí iterace algoritmu a Δt je perioda vzorkování.

2.3.2 Magnetometr a akcelerometr

Senzory nejsou schopny samostatně poskytnout dostatek informace pro jednoznačné určení orientace. Orientaci určenou libovolným senzorem je možné rotovat kolem měřeného vektoru. Například pokud měřený vektor má stejný směr jako osa z senzoru, nemohu přesně určit směr os x a y. Mohu systém senzoru rotovat kolem osy z bez změny výsledku měření.

Pro některé aplikace může být akceptovatelné nalezení jen dvou Eulerových úhlů ze tří. Při použití quaternionů je potřeba nalézt kompletní řešení. Toho může být docíleno pomocí optimalizační úlohy, kde je orientace senzoru ${}^{S}_{E}\mathbf{q}$ určována pomocí předdefinovaného referenčního směru pole v souřadnicovém systému Země ${}^{E}\mathbf{d}$ a směrem pole naměřeným v souřadnicovém systému senzoru ${}^{S}\hat{\mathbf{s}}$. Dostáváme obecně následující optimalizační úlohu:

$$\min_{{}^{S}_{E}\hat{\mathbf{q}}\in\Re^4} \mathbf{f}\left({}^{S}_{E}\hat{\mathbf{q}}, {}^{E}\mathbf{d}, {}^{S}\hat{\mathbf{s}}\right) \tag{2.5}$$

Pro řešení optimalizační úlohy autor zvolil metodu klesání po gradientu, z důvodu její jednoduché implementace a nízké výpočetní náročnosti. Jedná se o iterační metodu podle následujícího vzorce.

$$\mathbf{q_{k+1}} = \mathbf{q}_k - \mu\frac{\Delta\mathbf{f}}{\|\Delta\mathbf{f}\|}, \quad k = 0, 1, 2...n \tag{2.6}$$

$$\Delta\mathbf{f} = \mathbf{J}^T\mathbf{f}, \tag{2.7}$$

kde $\mathbf{J}$ je jakobián funkce $\mathbf{f}$ a μ konstanta.

Po dosazení měřených veličin za obecné vektory dostáváme funkci $\mathbf{f}_g$ závislou na naměřeném zrychlení ${}^{S}\hat{\mathbf{a}}$ a funkci $\mathbf{f}_b$ závislou na normalizovaném naměřeném vektoru magnetického pole ${}^{S}\hat{\mathbf{m}}$ a vektorem ${}^{E}\mathbf{b}$ vyjadřující poměr složek vektoru magnetického pole mezi vertikální a horizontální složkou. K funkcím náleží jakobiány $\mathbf{J}_g$ a $\mathbf{J}_b$.

$$\mathbf{f}_g\left({}^{S}_{E}\hat{\mathbf{q}}, {}^{S}\hat{\mathbf{a}}\right) \tag{2.8}$$

$$\mathbf{f}_b\left({}^{S}_{E}\hat{\mathbf{q}}, {}^{E}\mathbf{b}, {}^{S}\hat{\mathbf{m}}\right) \tag{2.9}$$

Jak již bylo zmíněno, měření gravitace nebo magnetického pole samostatně neumožňuje jednoznačně určit orientaci. Ale můžeme zkombinovat obě měření, abychom získali jednoznačnou pozici.

Konvenční přístup k výpočtu řešení vyžaduje výpočet několika iterací pro každý naměřený vzorek s dalšími dílčími výpočty (druhá derivace pro určení optimálního μ). Za předpokladu, že konvergenční rychlost je vyšší (nebo stejná) než rychlost změn orientace, můžeme pro každý vzorek vypočítat jen jednu iteraci algoritmu.

$${}^{S}_{E}\mathbf{q}_{\Delta,t} = {}^{S}_{E}\hat{\mathbf{q}}_{est,t-1} - \frac{\Delta\mathbf{f}}{\|\Delta\mathbf{f}\|} \tag{2.10}$$

$$\mathbf{f} = \begin{bmatrix}\mathbf{f}_g \\ \mathbf{f}_b\end{bmatrix} \tag{2.11}$$

2.3.3 Fúze odhadů

Odhad orientace je proveden pomocí fúze výpočtů orientace z gyroskopu a orientace magnetometru s akcelerometrem. Podrobné odvození vztahu je uvedeno ve článku [4]. Výsledný odhad orientace je určen předchozím stavem ${}^S_E\hat{\mathbf{q}}_{est,t-1}$ a integrací hodnot naměřených gyroskopem ${}^S_E\dot{\mathbf{q}}_{\omega,t}$ upravených o hodnoty získané z magnetometru a akcelerometru ${}^S_E\dot{\mathbf{q}}_{\epsilon,t}$. Konstanta β určuje věrohodnost dat z magnetometru a akcelerometru.

$$ {}^S_E\mathbf{q}_{est,t} = {}^S_E\hat{\mathbf{q}}_{est,t-1} + {}^S_E\dot{\mathbf{q}}_{est,t}\Delta t \tag{2.12} $$

$$ {}^S_E\dot{\mathbf{q}}_{est,t} = {}^S_E\dot{\mathbf{q}}_{\omega,t} - \beta {}^S_E\dot{\mathbf{q}}_{\epsilon,t} \tag{2.13} $$

$$ {}^S_E\dot{\mathbf{q}}_{\epsilon,t} = \frac{\Delta\mathbf{f}}{\|\Delta\mathbf{f}\|}. \tag{2.14} $$

2.3.4 Kompenzace

Algoritmus implementuje aktivní potlačení magnetického zkreslení, které může vznikat například přítomností feromagnetických předmětů v okolí magnetického senzoru. Při velikosti naměřeného vektoru magnetického pole mimo povolený rozsah jsou použita jen data z akcelerometru a gyroskopu.

Dále algoritmus implementuje potlačení driftu offsetu, který může vznikat vlivem změny teploty senzoru nebo v průběhu času funkce senzoru. Jestliže je senzor nějakou dobu bez pohybu, je provedena nová kalibrace offsetu gyroskopu, potlačující nově vzniklé offsety gyroskopu.

2.3.5 Kalibrace senzorů

Ke správnému fungování AHRS algoritmů je potřeba přesně zkalibrovat použité senzory. U gyroskopu je potřeba odstranit offset. U magnetometru kalibrovat zisk a offset, tuto kalibraci je potřeba provést až po umístění senzoru na jeho pracovní pozici, kvůli odstranění pasivního zkreslení magnetického pole vzniklého vlivem okolních součástek.

Pro kalibraci offsetu gyroskopu postačuje zprůměrovat dostatek měření při nulovém pohybu senzoru.

Kalibrace magnetometru

Kalibrace magnetometrů můžeme rozdělit do dvou kategorií.

- Vektorové
- Skalární

U vektorových kalibrací známe vektor magnetického pole, který porovnáváme s naměřenými hodnotami kalibrovaného senzoru. Vývoji zařízení, na generování libovolného magnetického vektoru s aktivním potlačením okolního rušení, určeného právě pro tento typ kalibrace, jsem se věnoval v bakalářské práci (viz [13]).

Při skalární kalibraci známe jen intenzitu magnetického pole. Kvalita vektorové kalibrace je lepší než skalární, ale vyžaduje kvalitnější vybavení. Mezi jedno z hlavních omezení skalárních metod se řadí absence detekce vzájemné polohy souřadnicového systému senzoru a referenčního souřadnicového systému.

Metody skalární kalibrace jsou dostatečné pro zisk 9 "intristických" parametrů senzoru.

Pro kalibraci mého magnetometru jsem použil skalární metody, Z důvodu dostatečnosti tohoto typu kalibrace a absence generátoru magnetického pole. Kalibrace byla prováděna v domácích podmínkách

Vztah mezi naměřeným vektorem $\mathbf{F}$[bit] a reálným vektorem magnetického pole měřeného senzorem $\mathbf{B_s}$ můžeme popsat následujícím vztahem:

$$\mathbf{F} = \mathbf{SPB_s} + \mathbf{b}, \tag{2.15}$$

kde

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{2.16}$$

je vektor offsetů,

$$\mathbf{S} = \begin{bmatrix} S_1 & 0 & 0 \\ 0 & S_2 & 0 \\ 0 & 0 & S_3 \end{bmatrix} \tag{2.17}$$

diagonální matice citlivostí jednotlivých os a

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ -\sin u_1 & \cos u_1 & 0 \\ \sin u_1 & \sin u_3 & \sqrt{1 - \sin^2 u_2 - sin^2 u_3} \end{bmatrix} \tag{2.18}$$

je matice neortogonalit, která měřený vektor transformuje z ortogonálních souřadnic do neortogonálních souřadnic naměřeného vektoru. Úhly neortogonality jsou nadefinovány na obrázku 2.6.

Dále je možné zahrnout vzájemnou polohu souřadnicového systému reference a senzoru, která se v následujících vztazích pokrátí a tím dokážeme, nemožnost určení rotace referenčního a senzorového systému pomocí použité skalární metody.

Dosadíme vztah:

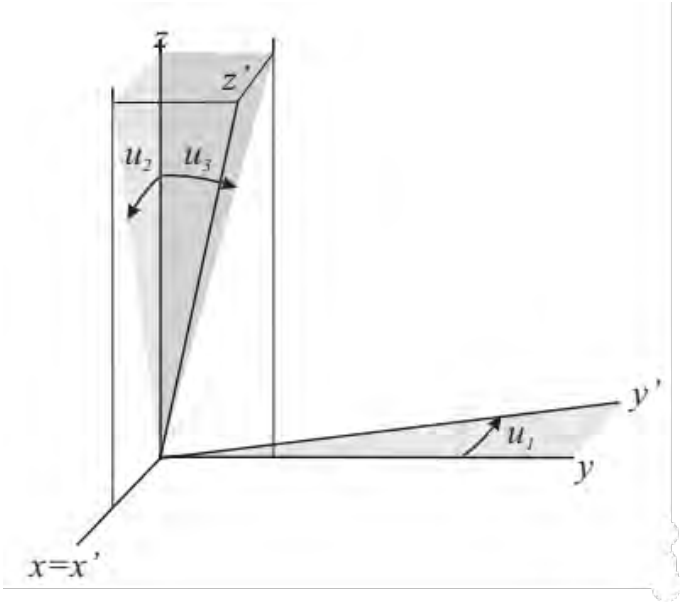
$$\mathbf{B_s} = \mathbf{RB_{reff}}, \tag{2.19}$$

kde $\mathbf{R}$ je rotační matice, otáčející systém postupně o tři Eulerovy úhly α, β a γ.

$$\mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.20}$$

Z rovnice 2.15 můžeme vyjádřit reálnou hodnotu v souřadnicovém systému senzoru a po dosazení rovnice 2.19 v referenčním.

$$\mathbf{B_s} = \mathbf{P}^{-1}\mathbf{S}^{-1}\left(\mathbf{F} - \mathbf{b}\right) \tag{2.21}$$



Obrázek 2.6: Kalibrace senzorů: Definice úhlů neortogonality [5]

$$\mathbf{B_{reff}} = \mathbf{R}^{-1}\mathbf{P}^{-1}\mathbf{S}^{-1}\left(\mathbf{F}-\mathbf{b}\right) \tag{2.22}$$

Skalární metoda kalibrace porovnává jen velikost vektoru magnetického pole. Potřebuji ji tedy vyjádřit:

$$B_{reff}(\mathbf{F},\mathbf{m}) = \|\mathbf{B_{reff}}(\mathbf{F},\mathbf{m})\| \tag{2.23}$$

$$\|\mathbf{B_{reff}}(\mathbf{F},\mathbf{m})\| = \sqrt{\mathbf{B_{reff}}^{T}\mathbf{B_{reff}}} \tag{2.24}$$

$$= \sqrt{\left(\mathbf{F}-\mathbf{b}\right)\left(\mathbf{S}^{-1}\right)^{T}\left(\mathbf{P}^{-1}\right)^{T}\left(\mathbf{R}^{-1}\right)^{T}\mathbf{R}^{-1}\mathbf{P}^{-1}\mathbf{S}^{-1}\left(\mathbf{F}-\mathbf{b}\right)}, \tag{2.25}$$

kde parametr $\mathbf{m} = \begin{bmatrix} b_1 & b_2 & b_3 & S_1 & S_2 & S_3 & u_1 & u_2 & u_3 \end{bmatrix}$.

Matice $\mathbf{S}$ je diagonální, tedy platí $\mathbf{S} = \mathbf{S}^T$. Pro rotační matice platí $\mathbf{R}^{-1} = \mathbf{R}^T$ a $\mathbf{R}\mathbf{R}^T = \mathbf{E}$, kde $\mathbf{E}$ je jednotková matice. Po dosazení těchto vztahů do 2.25 dostáváme výsledek:

$$\|\mathbf{B_{reff}}(\mathbf{F},\mathbf{m})\| = \sqrt{\left(\mathbf{F}-\mathbf{b}\right)\mathbf{S}^{-1}\left(\mathbf{P}^{-1}\right)^{T}\mathbf{P}^{-1}\mathbf{S}^{-1}\left(\mathbf{F}-\mathbf{b}\right)} \tag{2.26}$$

Z výsledného vztahu nám zmizí závislost na vzájemné poloze souřadnicových systémů. To dokazuje větu v úvodním odstavci.

Pro získání parametrů je použita metoda nejmenších čtverců. Minimalizujeme následující funkci:

$$\chi^2 = \sum_{j=0}^{n}\left(\frac{B_{reff}(\mathbf{F_j},\mathbf{m}) - B}{\sigma_B}\right), \tag{2.27}$$

kde B je měřená velikost vektoru magnetického pole, n je celkový počet měření a σ_B je chyba dat. Jelikož je stejná pro všechna data můžeme pro zjednodušení

výpočtu minimalizovat jen čitatel zlomku $\delta d = B_{reff}(\mathbf{F_i}, \mathbf{m}) - B$, bez vlivu na výsledné parametry.

Výsledná závislost není lineární. Pro vyřešení optimalizační úlohy zvolíme Gaussovu iterační metodu.

$$\delta d_i = \mathbf{G}_i \cdot \delta \mathbf{m}_i \tag{2.28}$$

$$\mathbf{G}_i = \left.\frac{\partial d(\mathbf{F}, \mathbf{m})}{\partial \mathbf{m}}\right|_{m=m_i} \tag{2.29}$$

i-tá iterace Gaussova iteračního algoritmu bude vypadat následovně:

$$\delta \mathbf{d}_i = \begin{bmatrix} B_{reff}(\mathbf{F_0}, \mathbf{m_i}) - B \\ B_{reff}(\mathbf{F_1}, \mathbf{m_i}) - B \\ \vdots \\ B_{reff}(\mathbf{F_n}, \mathbf{m_i}) - B \end{bmatrix} \tag{2.30}$$

$$\mathbf{G}_i = \begin{bmatrix} \left.\frac{\partial d(\mathbf{F}, \mathbf{m})}{\partial \mathbf{m}}\right|_{m=m_i, \mathbf{F}=\mathbf{F}_0} \\ \left.\frac{\partial d(\mathbf{F}, \mathbf{m})}{\partial \mathbf{m}}\right|_{m=m_i, \mathbf{F}=\mathbf{F}_1} \\ \vdots \\ \left.\frac{\partial d(\mathbf{F}, \mathbf{m})}{\partial \mathbf{m}}\right|_{m=m_i, \mathbf{F}=\mathbf{F}_n} \end{bmatrix} \tag{2.31}$$

$$\delta \mathbf{m}_i = \left(\mathbf{G}_i^T \cdot \mathbf{W} \cdot \mathbf{G}_i\right)^{-1} \left(\mathbf{G}_i^T \cdot \mathbf{W} \cdot \delta \mathbf{d}_i\right) \tag{2.32}$$

$$\mathbf{m}_{i+1} = \mathbf{m}_i + \delta \mathbf{m}_i, \tag{2.33}$$

kde $\mathbf{W}$ je matice Huberových vah [14]. Postup kalibrace byl převzat z [5].

Kapitola 3

Konstrukce platformy

Konstrukce robotické platformy je postavena podle open source platformy Sawppy the Rover. Tato platforma napodobuje kinematiku reálných sond Curiosity a Persenvance a byla vytvořena jako levnější verze projektu JPL Open Source Rover. Cílem vývoje platformy Sawppy bylo vytvořit funkční hobby platformu za přijatelnou cenu, napodobující kinematiku vozítek používaných na Marsu. Na obrázku 3.1 je zachycena hotová robotická platforma.

Obrázek 3.1: Konstrukce: Hotová robotická platforma.

Design kinematiky robotické platformy umožňuje překonávat složitý terén. Všechna kola se mohou pohybovat nahoru a dolů nezávisle na ostatních. Tím je docílena větší stabilita a vlastnost překonávat členité prostředí. Uložení hlavního rámu pro elektroniku je navrženo tak, aby se udržovalo co možná nejvíce ve vodorovné pozici při jakékoliv konfiguraci kol.

Konstrukce robota se skládá z hliníkových čtvercových profilů o hraně

15 mm a 3D tištěných dílů, které tvoří spojky hliníkových profilů a uchycení ostatních dílů.

Platforma pro zatáčení využívá natáčení všech krajních kol za pomoci chytrých servomotorů LewanSoul LX-16A. Všechny servomotory jsou napojeny na stejnou jedno-vodičovou sběrnici a jsou rozlišeny pomocí přidělených identifikátorů (ID). Kola originální verze jsou hnána stejnými servomotory jako jsou použity na natáčení kol. Mnou upravená verze konstrukce robota používá klasické stejnosměrné motory pro zvýšení výkonu (viz následující kapitola 3.1).

3.1 Úpravy konstrukce

Původní design robota používá pro pohon kol servomotory LewanSoul LX-16A. Toto řešení přináší jednoduchou ovladatelnost a kabeláž robota. Navíc k servomotorům je možné dokoupit modul pro jejich přímé připojení do USB konektoru řídicí jednotky Raspberry Pi. Tyto servomotory umožňují jen velmi pomalý pohyb.

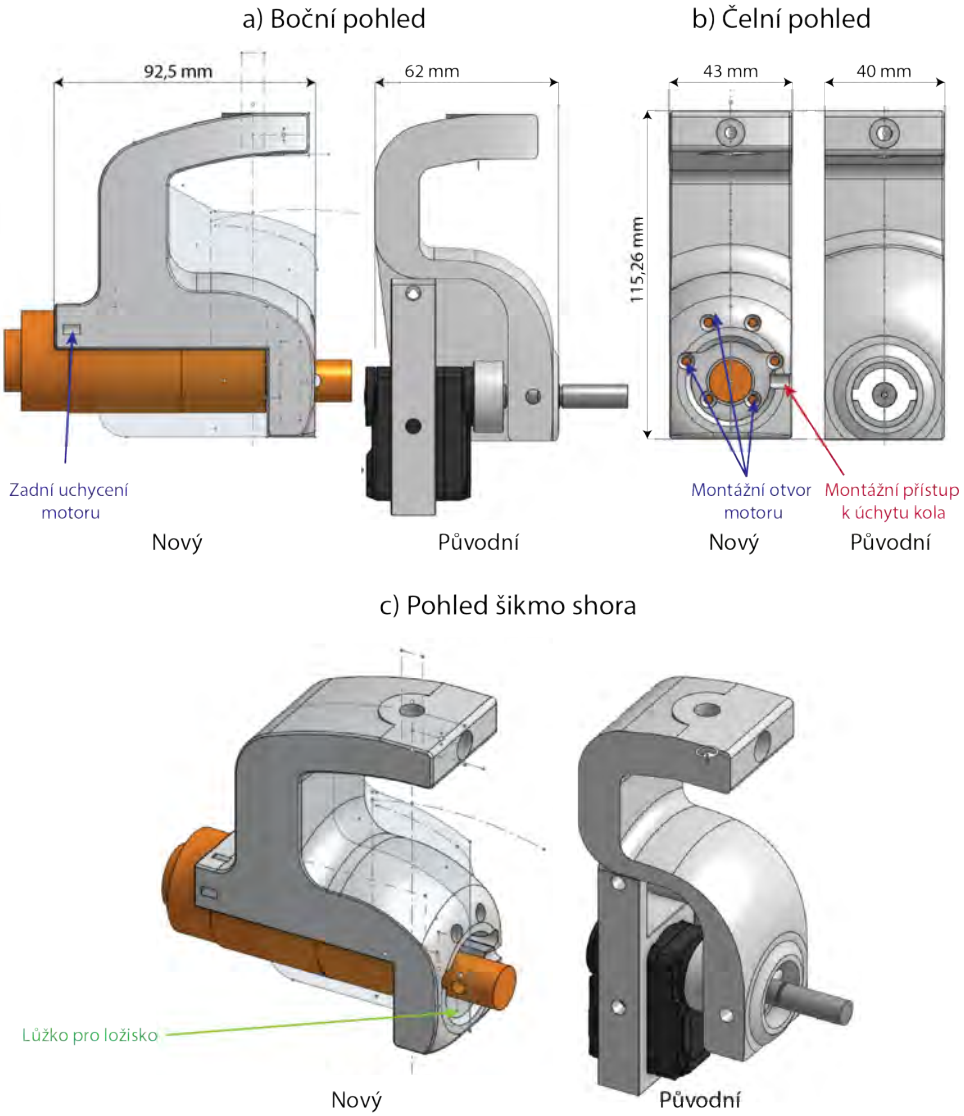
Jelikož tyto servomotory umožňují jen velmi pomalý pohyb, rozhodli jsme se je v rámci projektu v týmu (PVT) nahradit klasickými stejnosměrnými motory, pro vylepšení pohybových vlastností platformy. Další nevýhodou původního designu jsou 3D tištěná kola. Jelikož část kol dotýkající se povrchu je plastová, nedosahují kvalitní přilnavosti a na hladkém povrchu jednoduše prokluzují. Z toho důvodu jsme zvolili klasická kola s gumovou pneumatikou o podobném vnějším průměru.

Pro použití nových součástek jsem musel upravit jednotlivé části modelu robota, který je volně dostupný v online 3D modelovacím programu Onshape.

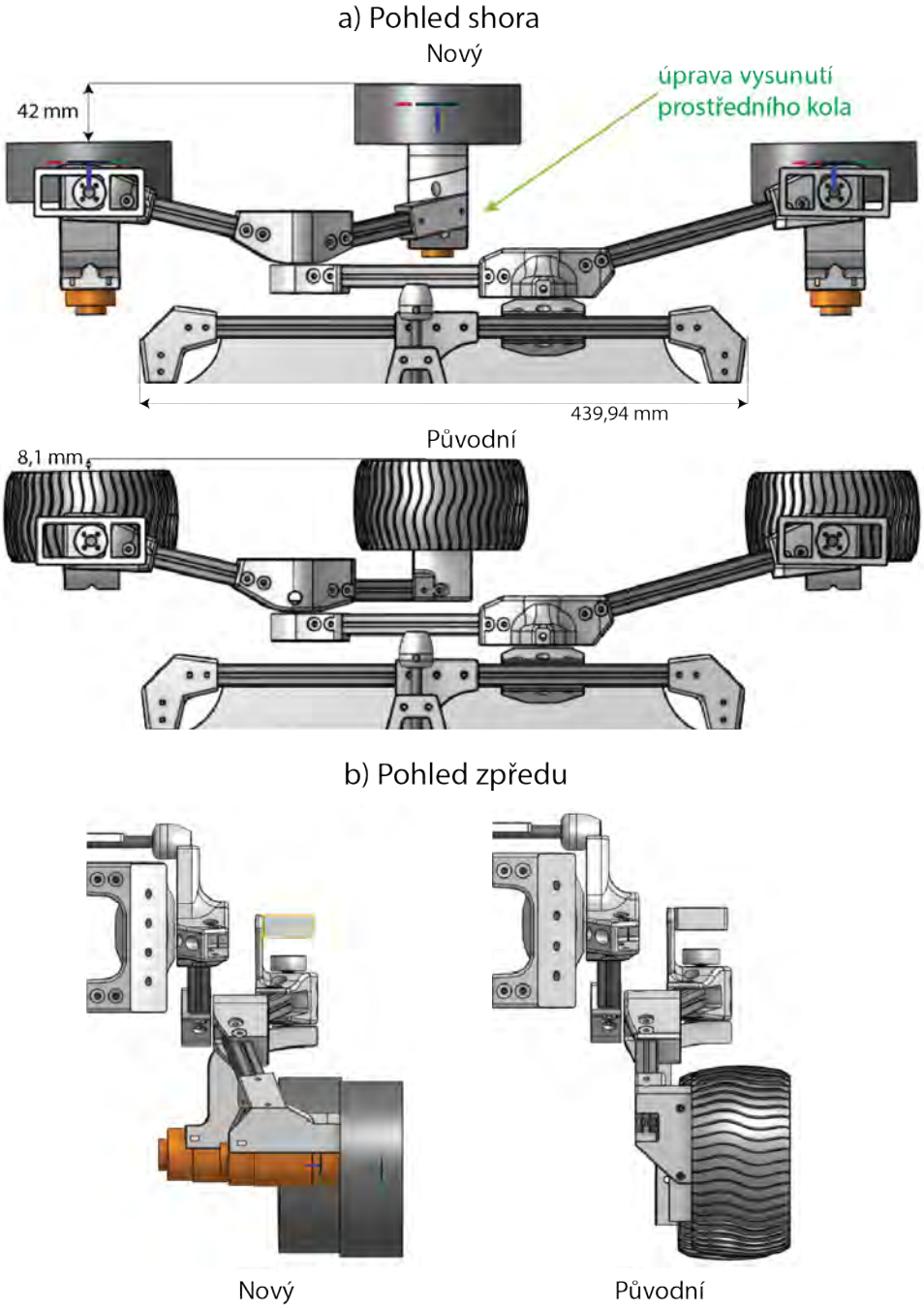
Původní a nově navrhované aktuátory mají jiný tvar. Spodní část modelu uložení kol jsem musel protáhnout a mírně rozšířit za hlavní rameno pro vytvoření lůžka na uložení motoru. Motor má montážní otvory jen v čele, pro vyztužení připojení motoru jsem přidal otvory na rychloupínací svorku v koncové části motoru. Další komplikací byla větší tloušťka pneumatiky. (Navrhovaná pneumatika má menší vnitřní průměr.) Musel jsem snížit výšku spodní části kloubu bez omezení její pevnosti.

Při použití nových kol bylo nutné předělat i systém jejich uchycení. Pro uchycení kol jsem použil vysoustružené spojky, které se připevňují přímo na osu motoru. Aby motory nenesly váhu celého robota je tento úchyt z poloviny vsazen do ložiska. Porovnání původního a upraveného kloubu je na obrázku 3.2.

Z důvodu prodloužení kloubů nesoucích aktuátory s koly směrem do konstrukce robota, bylo potřeba upravit další geometrii robota. Pro zachování funkčnosti designu robota jsem musel upravil úhel odsazení prostředního kola tak, aby při rotačních pohybech jednotlivých kloubů nedocházelo ke kolizi mezi jejich rameny. (viz obrázek 3.3)



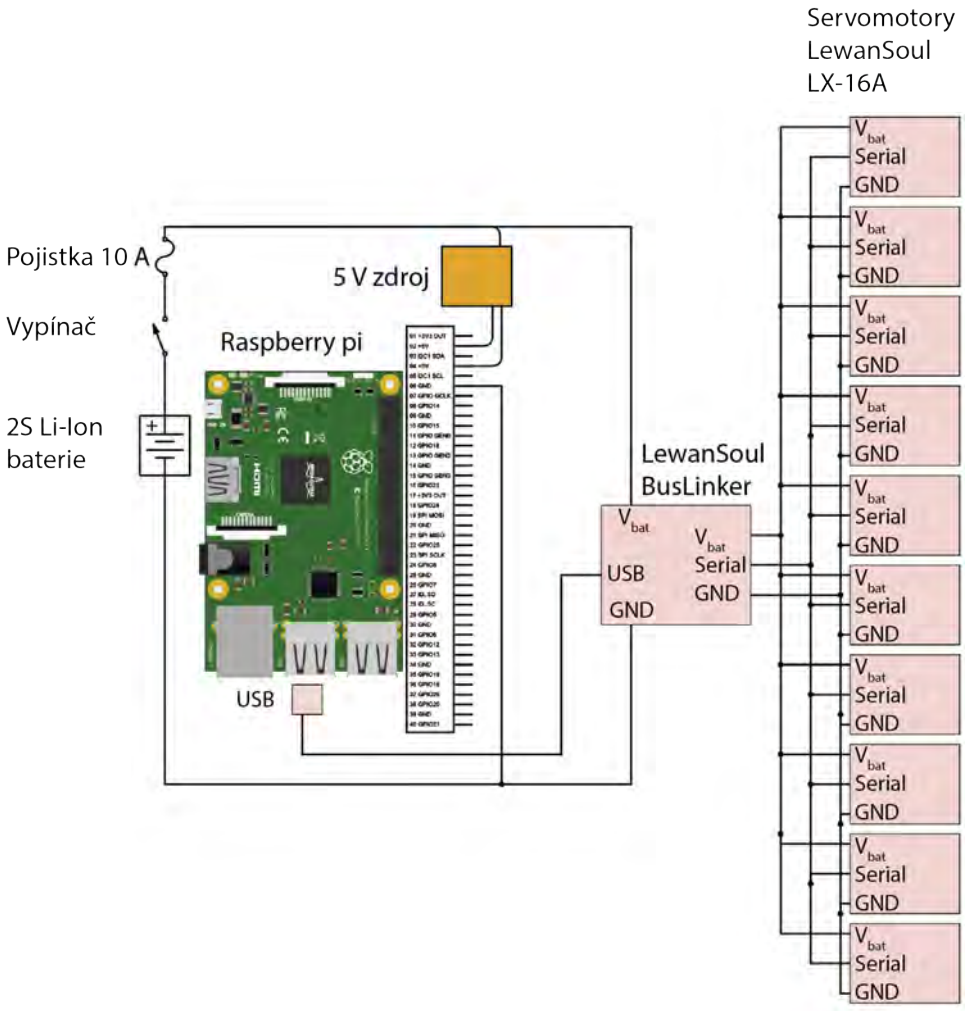
Obrázek 3.2: Popis robota: Ukázka úpravy kloubu pro uchycení kola.



Obrázek 3.3: Popis robota: Úprava vysunutí prostředního kola.

3.2 Elektronika

Originální návrh elektroniky robota je minimalistický. Skládá se z řídicí jednotky Raspberry Pi , baterie, modulu pro připojení servomotorů a 10 servomotorů LewanSoul LX-16A (viz schéma na obrázku 3.4).



Obrázek 3.4: Popis robota: Originální návrh elektroniky. [6]

V rámci PVT jsme se rozhodli přepracovat původní minimalistickou elektroniku. Řídící jednotku Raspberry Pi jsme doplnili o tři specializované moduly. Pro komunikaci mezi moduly jsem na základě mých předchozích zkušeností navrhl sběrnici CAN bus. Toto řešení nabízí jednoduchou rozšiřitelnost projektu při přidání nového modulu, stačí dodefinovat nové zprávy s unikátními identifikátory a upravit jen ty ostatní moduly, které mají interagovat s nově přidaným modulem.

Raspberry Pi nemá periferii CAN bus. Je potřeba připojit externí CAN controller. Dalším požadavkem byla schopnost řídit robota pomocí radiového

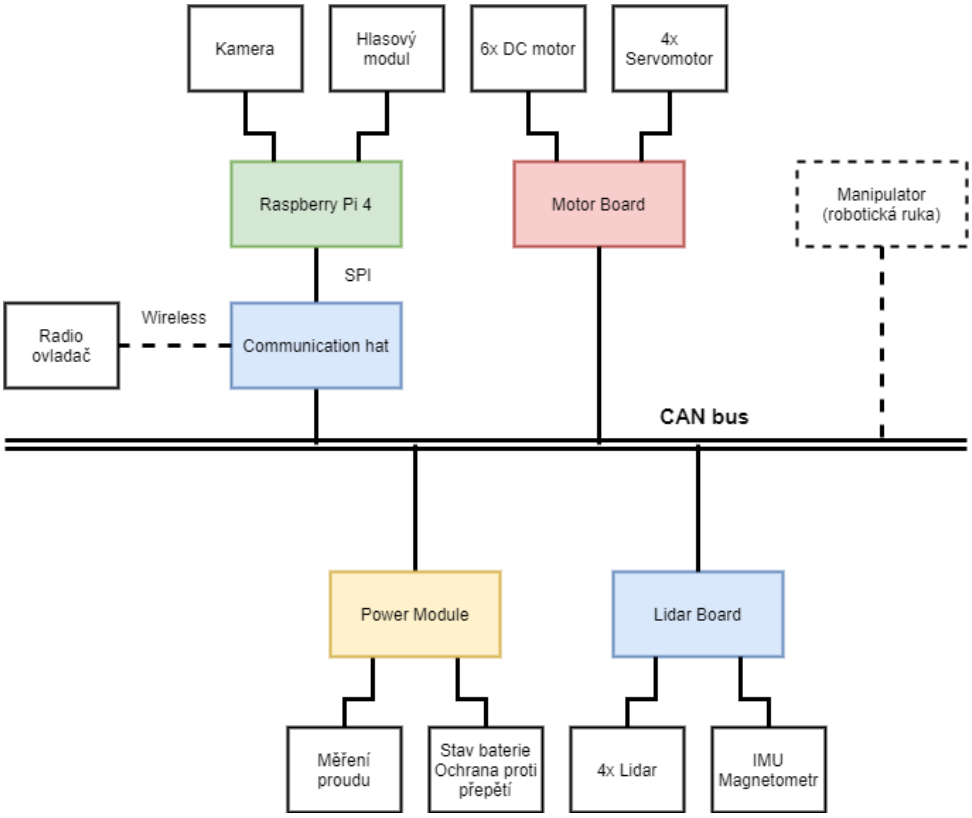
ovladače. Pro rozšíření funkcionality řídicí jednotky o tyto požadavky jsem navrhl tzv. hat (desku plošných spojů připojenou na GPIO konektor Raspberry Pi)

Nově navržená řídicí elektronika robota se skládá z následujících jednotek. (Moduly jsme si rovnoměrně rozdělili. Jejich původní rozdělení mezi členy týmu je uvedeno v závorkách.):

- Raspberry Pi 4 (Maksym Shcherban)
- Lidar (Pavel Doubrava)
- Motor Board (Jan Gärtner)
- Power Board (Miroslav Tržil)
- Raspberry Pi Communication hat (Pavel Doubrava)

Řídící elektronika je doplněna: 3S4P Li-ion baterií o kapacitě 19 200 mA h, šesti DC motory a čtyřmi servomotory LewanSoul LX-16A. Zapojení nově navržené elektroniky je uvedeno na obrázku 3.5. V rámci diplomové práce jsem se seznámil s návrhem desek plošných spojů mých kolegů, následně jsem desky oživil a vytvořil software pro všechny moduly. Při oživování a programování některých DPS bylo nutné upravit některá zapojení obvodů (viz. kapitoly věnující se jednotlivým modulům).

V následujících kapitolách bude přiblížena práce na jednotlivých modulech.



Obrázek 3.5: Popis robota: Finální návrh elektroniky

Kapitola 4

Lidar

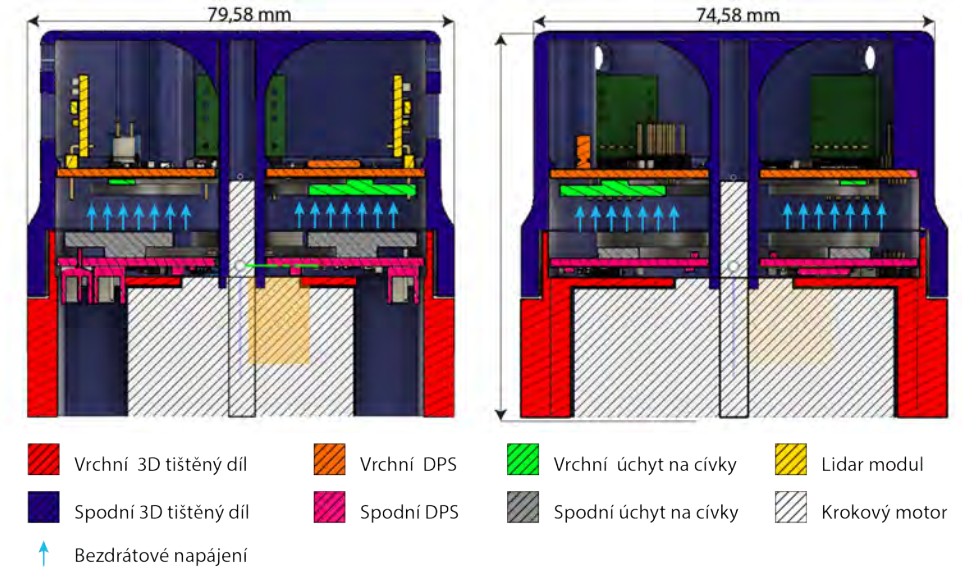
Lidarový modul je hlavní senzorovou jednotkou robotické platformy. Slouží k získávání informací o okolí robota a jeho orientaci v něm. Jednotka umožňuje detekovat volný prostor v úhlu 360°. Data získaná senzory na modulu můžou být dále použita například v algoritmech SLAM (Simultaneous localization and mapping), které umožňují lokalizaci a prozkoumávání neznámých prostředí (tvorba mapy). Jeden z těchto algoritmů je použit v hlavní řídicí jednotce (Raspberry Pi 4) viz kapitola 8.3.5. Jednotka dále implementuje fúzi dat z inerciální jednotky a magnetometru poskytující informace o natočení robota v prostoru. Tato informace je používána jako základní odometrie robota.

Výslednou podobu modulu, uchyceného na konstrukci robota můžete vidět na obrázku 4.1.

Obrázek 4.1: Lidar: Snímek kompletního senzoru.

4.1 Návrh zařízení

Lidar je složen ze dvou desek plošných spojů (DPS/PCB) a dílů vytištěných na 3D tiskárně. Na obrázku 4.2 jsou dva řezy 3D modelem lidarového modulu. Řezy vzájemně svírají 45°. Vrchní část jednotky je připevněna na rotor krokového motoru a rotuje oproti spodní. Na obrázku 4.3 je uveden náhled do dutiny senzoru mezi DPS. Je zde vyznačeno bezdrátové napájení pomocí cívek a sériová komunikace uskutečněná za pomoci optických prvků.



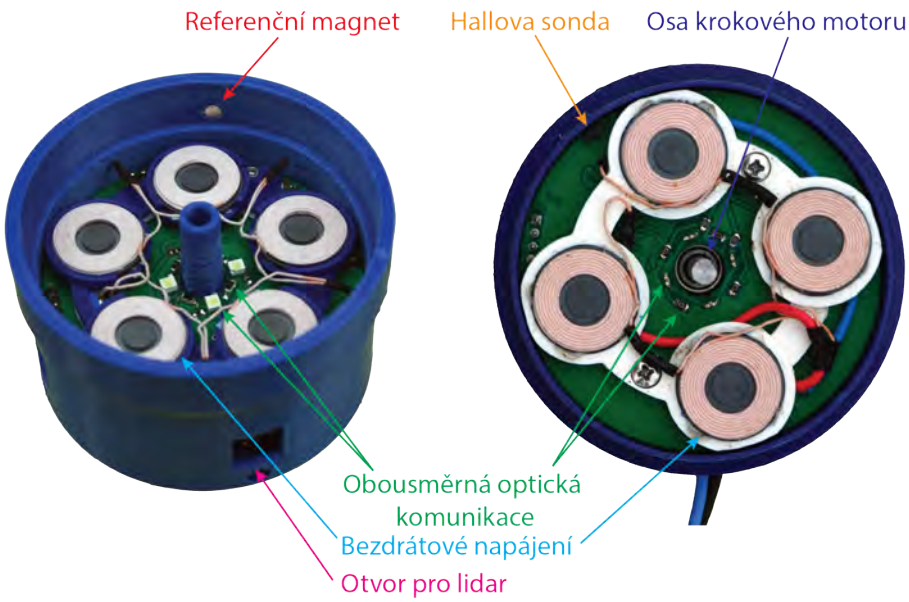
Obrázek 4.2: Lidar: Řez modelem Lidaru

Obě DPS jsou řízeny mikroprocesory STM32F103RE. Tento procesor disponuje dostatečným výpočetním výkonem a všemi potřebnými periferiemi. Stejným procesorem je vybaven čínský modul Bluepill. V čase než byly vyhotoveny a osazeny DPS, jsem mohl části firmware implementovat již na zmíněném modulu (knihovna pro ovládání CAN perferie, sběr dat ze senzorů).

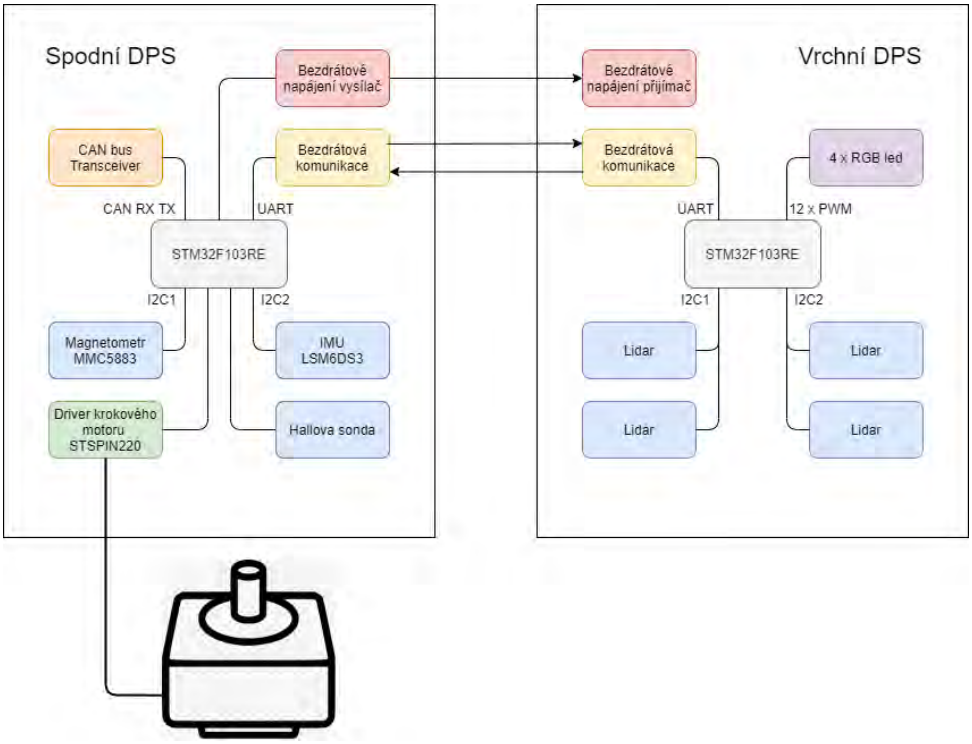
Na obrázku 4.4 je uvedeno blokové schéma modulu. Hlavním řídicím prvkem zařízení je procesor na spodní DPS. Tento mikroprocesor je připojen na sběrnici robota, ze které dostává pokyny a publikuje zde svoje data. Je zde také umístěna inerciální měřící jednotka LSM6DS3 se šesti stupni volnosti, Hallova sonda, pro kalibraci pozice vrchní DPS, driver krokového motoru STSPIN220 a konektor pro magnetometr MMC5883, který musí být, kvůli magnetickému rušení od krokového motoru a bezdrátovému napájení, umístěn mimo tělo zařízení a komunikuje se zbytkem jednotky pomocí I2C.

Komunikace mezi DPS modulu je zajištěna přenosem UARTu pomocí LED a fotodiod. Pro komunikaci směrem od mastera (spodní DPS) jsem použil IR část spektra a v opačném směru (od slave/vrchní) jsem použil modrou část spektra. Použil jsem multiplex na základě vlnové délky použitého světla.

K vrchní DPS jsou kolmo připájeny čtyři desky plošných spojů s lidary VL53L1x, navržené pro předmět LPE vyučovaný Katedrou měření ČVUT.



Obrázek 4.3: Lidar: Vnitřní uspořádání modulu.



Obrázek 4.4: Lidar: Blokové schéma zapojení hardwaru

Tyto DPS zajišťují uchycení použitého senzoru a základní zapojení (blokovací kondenzátory, pull-up resistory atd.). Nad každým senzorem vzdálenosti se nachází RGB led pro základní signalizaci.

4.2 Hardware

Návrh elektronické části hardwaru probíhal v programu KiCad. Pro lepší orientaci v projektu jsem použil hierarchické listy. Tato funkcionalita umožňuje zapouzdřit části obvodu do vlastního bloku se vstupy a výstupy.

Tělo modulu jsem navrhoval v 3D modelovacím programu Fusion 360. Skládá se z těchto komponent (viz obrázek 4.2):

- **Základna** se spodní DPS.
- **Vrchní část lidaru** nasazená na krokový motor, v němž je uchycena vrchní DPS.
- **Uchycení** na konstrukci robota.
- **Uchycení cívek** k deskám plošných spojů.

4.2.1 Základní zapojení mikroprocesoru

Pro základní zapojení mikroprocesoru jsem použil vlastní hierarchický list. Ten jsem následně použil u obou DPS. Základní zapojení mikroprocesoru bylo převzato z příslušné aplikační poznámky od výrobce MCU [15]. Kvůli použití sběrnice CAN bus a sériové linky, jsem pro zpřesnění frekvence hodinového signálu použil na všech DPS krystal. Pro usnadnění programování a základní signalizaci jsem na desku umístil čtyři LED. Tři jsou kontrolovány pomocí procesoru, čtvrtá signalizuje napájení mikroprocesoru. Zapojení je přiloženo v příloze B.

4.2.2 Senzorová část spodní DPS

Senzorová část spodní desky se skládá z Hallovy sondy, inerciální měřící jednotky a magnetometru. Krokový motor neumožňuje určit absolutní polohu natočení. Pro daný účel jsem použil kombinaci Hallovy sondy a magnetu, která umožní kalibraci natočení vrchní DPS. V překryvu vrchního a spodního 3D tištěného dílu je do pláště vrchního dílu vložen magnet. Tento magnet je následně detekován pomocí Hallovy sondy umístěné na spodní desce. Viz obrázek 4.3.

Inerciální měřící jednotka (IMU) i magnetometr používají komunikační rozhraní I2C. Pro zvýšení datové prostupnosti jsem připojil každý senzor na vlastní komunikační sběrnici. Sběrnice I2C potřebuje ke správné funkci na datovém a hodinovém signálu pull-up rezistory. Na použitém modulu s magnetometrem jsou již tyto rezistory osazeny, tedy nejsou potřeba na mnou navrhované desce. Na sběrnici k IMU jsem tyto rezistory přidal.

4.2.3 Ovládání krokového motoru

Další částí na spodní desce je driver krokového motoru. Hlavním kritériem při volbě driveru bylo napájecí napětí. Hledal jsem driver, který mohl být napájen 5 V. Do konečného výběru se dostal driver MP6500 a STSPIN220. Oba dva drivery podporují 3,3 V logiku. Vybral jsem driver STSPIN220 z důvodu možnosti jemnějšího mikrokrokování a menšího pouzdra (ušetření místa na DPS). Schéma zapojení jsem převzal z datasheetu driveru [7].

Při návrhu jsem musel dopočítat hodnotu snímacích rezistorů R_{sence}, pomocí kterých je regulován proud vinutími krokového motoru. Zakoupený krokový motor má odpor vinutí $R_{wind} = 5{,}4\,\Omega$. V datasheetu driveru je uvedeno, že maximální napětí na snímacích rezistorech $U_{R_{sence}max} = 0,5V$. Tyto rezistory jsou zapojeny do série s vinutím krokového motoru, tedy součet napětí na vinutí U_{wind} a na snímacím odporu $U_{R_{sence}}$ může být maximálně napájecí napětí $U_{cc} = 5V$. Z tohoto předpokladu mohu určit maximální dosažitelný proud tekoucí vinutím a podle toho zvolit požadovaný snímací odpor:

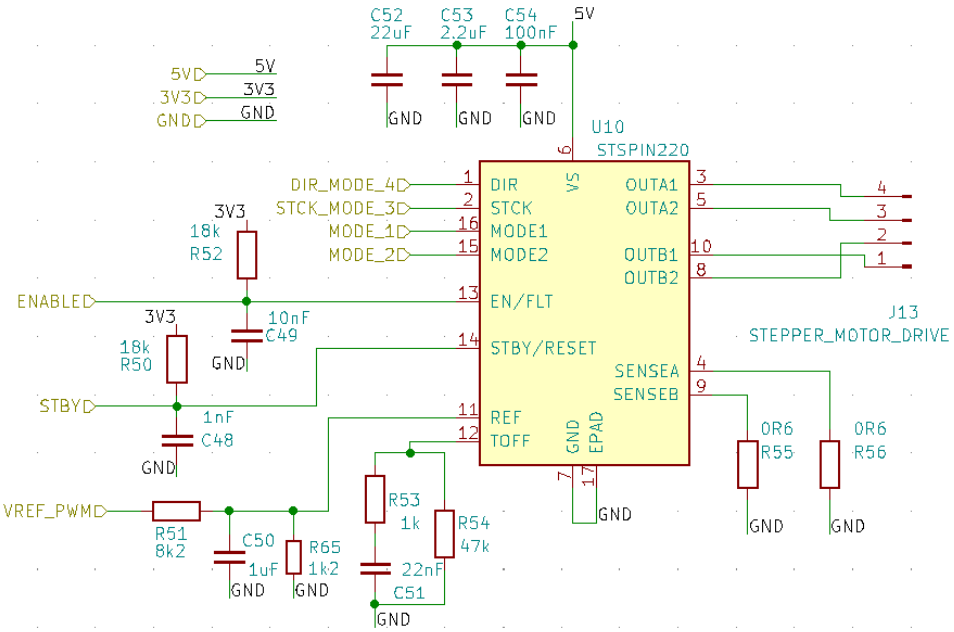
$$I_{max} = \frac{U_{cc} - U_{R_{sence}max}}{R_{wind}} = \frac{5 - 0,5}{5,4} = 0,8\overline{3}A \tag{4.1}$$

$$R_{sence} = \frac{U_{R_{sence}max}}{I_{max}} = \frac{0,5}{0,8\overline{3}} = 0,6\Omega \tag{4.2}$$

Bylo by možné dosáhnout i mírně vyšších hodnot proudu vinutím, ale bylo by to na úkor rozlišení nastavení proudu vinutím. Proud vinutími je nastavován pomocí napětí na pinu 11 (V_{ref}). Regulátor uvnitř integrovaného obvodu driveru reguluje proud vinutím tak, aby špičková hodnota napětí na snímacích rezistorech odpovídala napětí V_{ref}.

Pro zvýšení rozlišení nastavovaného proudu jsem za dolní propust PWM, nastavující V_{ref}, přidal zátěž, která funguje jako dělič napětí. Rozsah $0 - 3{,}3$ V převádí přibližně na $0 - 0{,}5$ V.

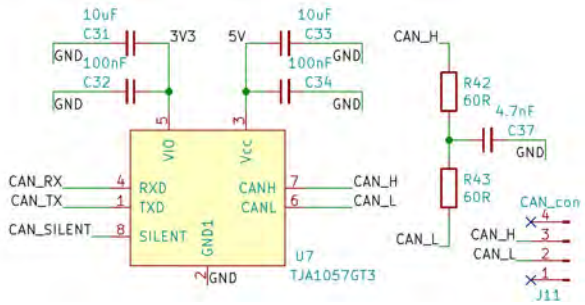
Výsledné zapojení je uvedeno na obrázku 4.5



Obrázek 4.5: Lidar: Schéma zapojení driveru krokového motoru

4.2.4 CAN bus

Hlavní komunikační sběrnící robota je CAN bus. Mikroprocesor nemůže být připojen přímo na tuto sběrnici. Je zde potřeba použít mezičlánek tzv. CAN transceiver, který zajišťuje obousměrný překlad signálu (viz kapitola 2.1.4. Schéma zapojení CAN transceiveru je na obrázku 4.6.



Obrázek 4.6: Lidar: Schéma zapojení CAN transceiveru

4.2.5 Bezdrátové napájení

Pro bezdrátové napájení byly použity obvody od firmy Semtech. Používám přenos elektrické energie pomocí střídavého magnetického pole. Zapojení obvodů pro příjem a vysílání energie byla převzata z datasheetů použitých integrovaných obvodů [16], [17], [18].

Vysílací obvod se skládá ze dvou integrovaných obvodů (řídicí obvod TS80002 a spínací obvod TS51231), zpětné vazby a cívek. Zpětná vazba

umožňuje regulovat přenesený výkon. Regulace přeneseného výkonu je uskutečněna volbou frekvence generovaného střídavého signálu budící cívky.

Přijímací obvod se skládá z cívek, usměrňovače a spínaného step-down zdroje. Usměrňovač v sobě má zabudovanou ochranu proti přepětí na vstupu z cívek. Maximální hodnota napětí je nastavena pomocí Zenerovy diody na 12 V. K usměrnění je použita dvojice Schottkyho diod a mosfetů pro minimalizaci ztrát. Usměrněný signál je pomocí soustavy blokovacích kondenzátorů vyhlazen. K návrhu soustavy v datasheetu jsem přidal kondenzátor 470 µF, pro poskytnutí energie při dočasném poklesu výkonu dodávaného bezdrátovým napájením, způsobeným například méně příznivou vzájemnou konfigurací cívek.

Spínaný zdroj (TS51221) je přímo určen k použití při bezdrátovém přenosu energie a vyznačuje se vysokou efektivitou a vyšší spínací frekvencí (1 MHz). Tato frekvence zjednodušuje filtraci střídavé složky a zmenšuje nároky na velikost plošného spoje. Vybral jsem verzi obvodu s přizpůsobitelným napětím na výstupu.

Napětí na výstupu je určeno napěťovým děličem zapojeným mezi zem, výstup a zpětnovazební pin integrovaného obvodu. Napětí jsem volil, co nejmenší možné, v závislosti na dalších použitých obvodech. Na vrchní DPS jsem použil Modré LED s dopředným napětím 2,9 V. Čím větší zvolím napájecí napětí vrchní DPS, tím méně závislé bude nastavení proudu LED na přesnosti hodnoty rezistorů omezujících proud. Vybraný lineární regulátor pro napětí 3,3 V má ztrátové napětí 350 mV, tedy minimální hodnota vstupního napětí je $U_{min} = 3{,}65\,\mathrm{V}$. Při zohlednění obou faktorů a řad hodnot rezistorů jsem zvolil hodnotu napětí $U_{out} = 4{,}41\,\mathrm{V}$.

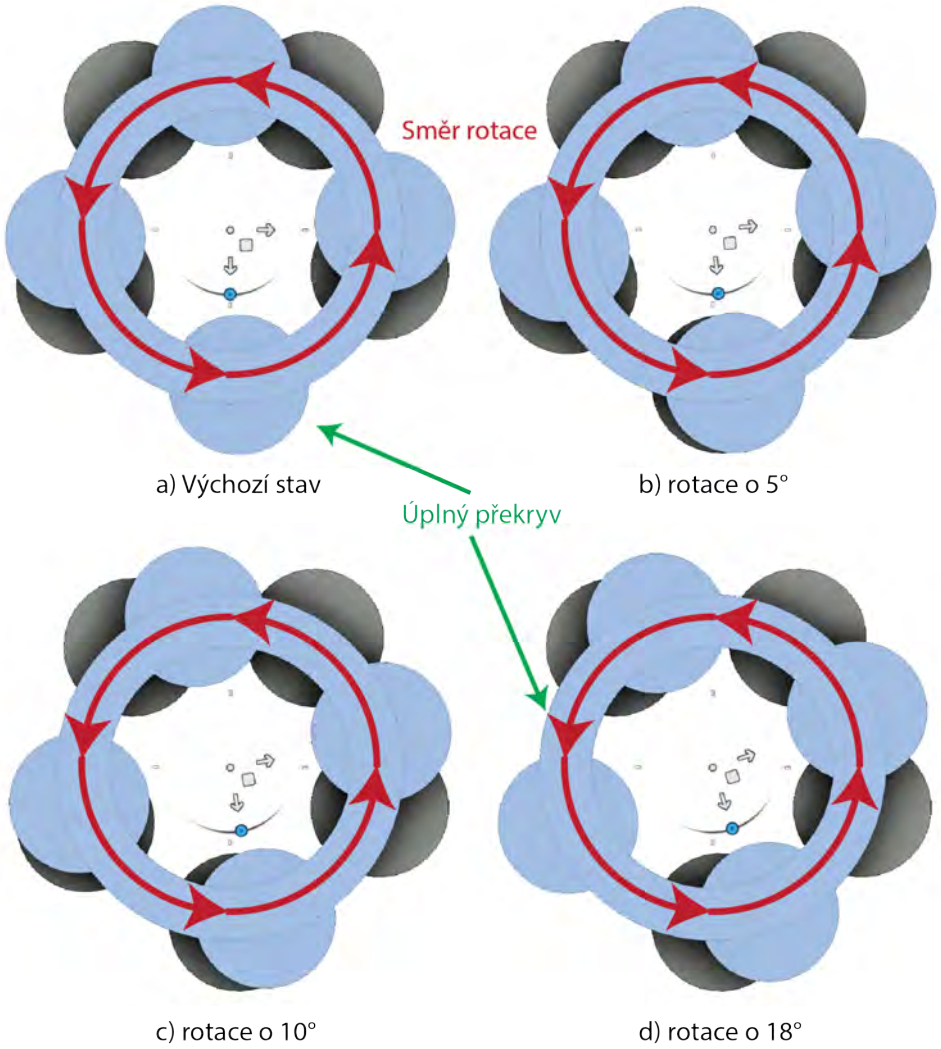
$$U_{out} = 0,9 \cdot \left(1 + \frac{R_{24}}{R_{25}}\right) = 0,9 \cdot \left(1 + \frac{39\,\mathrm{k\Omega}}{10\,\mathrm{k\Omega}}\right) = 4{,}41\,\mathrm{V} \tag{4.3}$$

Oživení obvodu

Nejdříve bylo potřeba nalézt vhodnou konfiguraci cívek na vrchní a spodní straně tak, aby byl zaručen co největší překryv při jakémkoliv vzájemném natočení vrchní a spodní DPS. V programu Fusion360 jsem vymodeloval několik variant a následně vybral tu nejlepší. Na přijímací stranně jsem použil 5 cívek a na vysílací 4, tato konfigurace nabízí největší dosažitelnou rovnoměrnost překryvu přijímacích a vysílacích cívek závislosti na jejich vzájemném natočení. Při této konfiguraci dochází každých 18° k úplnému překryvu dvojice cívek.

Screenshoty výsledné konfigurace jsou uvedeny na obrázku 4.7.

Při oživování obvodu jsem zjistil, že řídicí obvod TS80002 potřebuje nahrát firmware. Výrobce poskytuje podrobný návod, jak toho docílit, ale od vytvoření tohoto manuálu výrobce aktualizoval svoje webové stránky a firmwary zde již nejsou k dispozici. Opakovaně jsem se snažil kontaktovat support, bohužel neúspěšně. Rozhodl jsem se tedy nahradit řídicí IC procesorem na desce. Nejdříve jsem otestoval, jestli tento postup bude možný. Přeřezal jsem



Obrázek 4.7: Lidar: Vývoj překryvu cívek bezdrátového napájení

řídicí PWM signál z TS80002 a napojil jej na generátor signálů. Obvod budící cívky podle očekávání začal generovat výstupní signál.

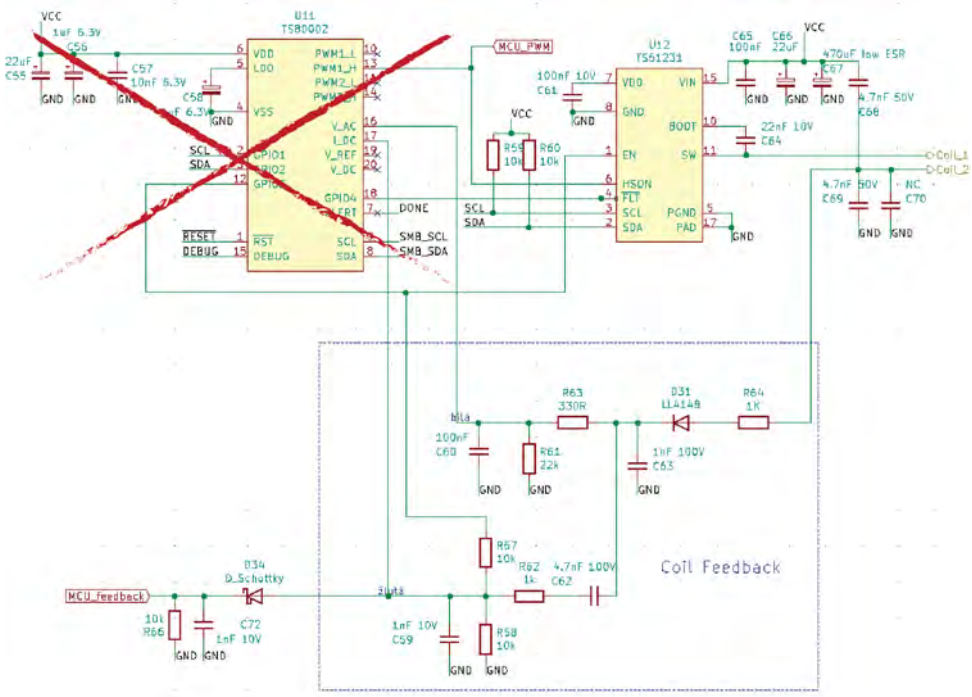
Další částí oživování bylo určit správné zapojení cívek. Přenos energie mezi cívkami lze modelovat jako transformátor s nízkým vazebním koeficientem, závislým na vzájemném natočení DPS a vzdálenosti cívek. Tedy platí zde stejný vliv poměru závitů na primárním a sekundárním vinutí.

Při zapojení cívek v sérii na vysílacím a přijímacím zařízení nebylo napětí za usměrněním dostatečně velké. Výstup step-down regulátoru vrchní DPS vypadával. Z toho důvodu jsem vyzkoušel zvětšit transformační poměr. Přepojil jsem tedy spodní konfiguraci cívek ze sériového na paralelní zapojení dvou cívek v sérii. Po této upravě jsem na výstupu usměrňovače již dosáhl dostatečného napětí.

Při těchto pokusech jsem odhalil pro naši aplikaci nebezpečnou teplotní

závislost rezonanční frekvence cívek, kterou bylo potřeba potlačit. K potlačení jsem využil zpětnovazební signál spodního obvodu. Tento signál nese užitečnou hodnotu jako velikost rozkmitu signálu. Pro jednodušší zpracování signálu mikroprocesorem jsem tuto hodnotu převedl na stejnosměrnou složku. Tato konverze využívá detektoru špiček.

Nakonec jsem vše připojil na piny řídicího MCU. Provedené úpravy jsou naznačeny na schématu na obrázku 4.8.



Obrázek 4.8: Lidar: Schéma úprav bezdrátového napájení

4.2.6 Bezdrátová komunikace mezi DPS

Pro komunikaci mezi deskami jsem použil UART. Kvůli vzájemnému pohybu desek přenáším UART bezdrátově pomocí světla. Na obou deskách je téměř totožný obvod. Obvody se liší jen velikostí zesílení operačních zesilovačů a konfigurací LED založených na jejich dopředném napětí.

Protože UART má v klidovém stavu hodnotu logické jedničky, rozhodl jsem se signál negovat z důvodu zmenšení doby svícení LED a z toho plynoucí spotřeby. Tedy LED svítí, pokud je na sběrnici logická nula. Vysílací část se skládá ze dvou tranzistorů. Ty zajišťují negaci a zesílení signálu z procesoru.

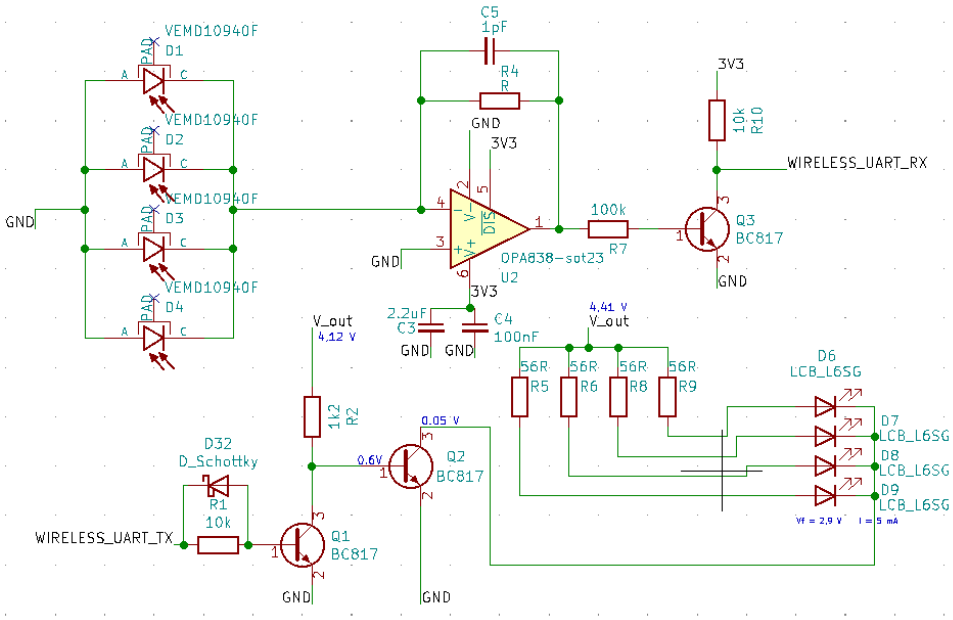
Přijímací část se skládá ze 4 paralelně zapojených fotodiod použitých jako zdroj proudu. Vygenerovaný proud je následně převeden na napětí pomocí I-U převodníku. Signál je následně pomocí tranzistoru negován na původní podobu.

Oživení obvodů

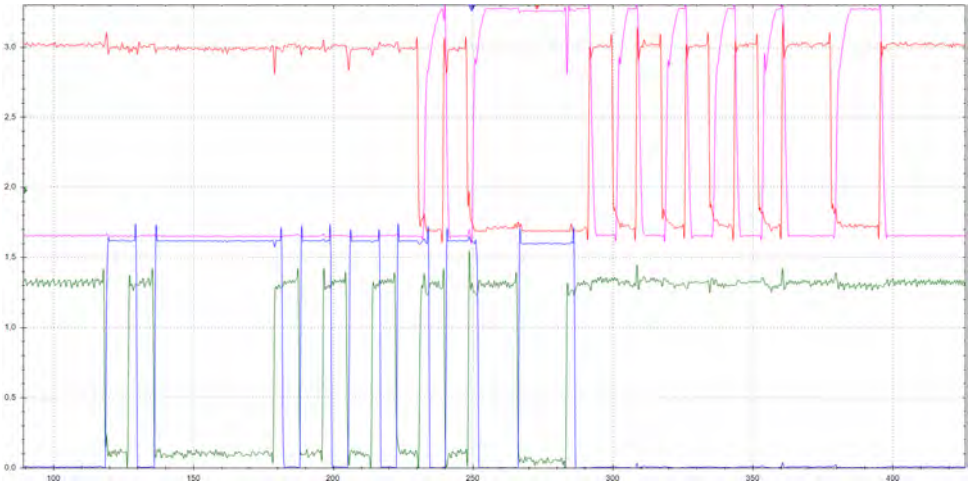
Při prvotním testování funkčností obvodů jsem objevil několik komplikací. Na obrázku 4.9 je schéma zapojení obvodu a na obrázku 4.10 je snímek z osciloskopu demonstrující jednotlivé problémy v přenosu. Na vrchní části obrázku je komunikace směrem od spodní desky k vrchní. Na spodní části obrázku je komunikace opačným směrem. Modré a fialové jsou výstupy I-U převodníků U2, U6. Zelené a červené signály jsou měřeny na bázích tranzistorů Q1 a Q6.

Fotodiody v použitém zapojení měly velkou dobu odezvy, což se projevovalo pomalým zánikem generovaného proudu po vypnutí zdroje světla. Zjistil jsem, že fotodiody jsou přesaturovány. Snížil jsem tedy intenzitu osvětlení fotodiod pomocí snížení proudu LED. Dále jsem určil zesílení I-U převodníku.

Pro urychlení vypnutí tranzistorů Q1 a Q6 při vypnutí jsem k bázovým odporům přidal paralelně Schottkyho diody.



Obrázek 4.9: Lidar: Schéma zapojení bezdrátového přenosu Uartu



Obrázek 4.10: Lidar: Záznam sériové komunikace mezi deskami

4.2.7 Lidary

K vrchní desce jsou připojeny DPS se senzory VL53l1x. Opět jsem použil pro větší prostupnost dat dvě sběrnice I2C. Senzorové moduly jsou rozmístěny tak, aby směry, ve kterých senzory měří vzdálenost, tvořily kříž. Viz obrázek 4.11 část b).

Tyto senzory se vyznačují nízkou cenou. Senzor nabízí kompletní řešení pro měření vzdálenosti, není jej potřeba doplňovat o další elektroniku. Senzor je schopen měřit vzdálenosti do čtyř metrů a vzorkovací frekvence může být maximálně 50 Hz. Senzor komunikuje po sběrnici I2C.

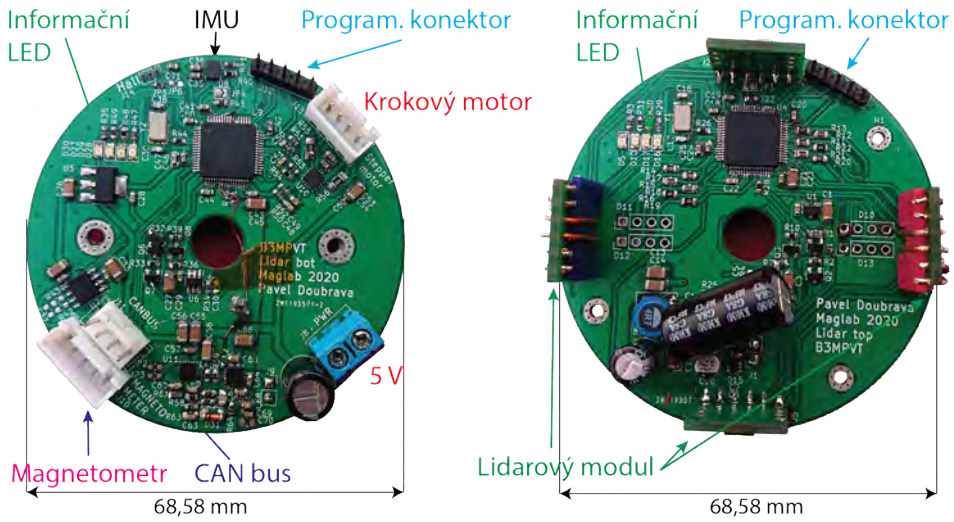
4.2.8 Desky plošných spojů

Vytvořil jsem dvě desky plošných spojů tvarem připomínající CD. Tento tvar byl zvolen kvůli rotaci vrchní desky. Většina součástek je umístěna na jedné straně tištěných spojů, na druhé straně je uchycení pro cívky bezdrátového napájení a okolo vnitřního otvoru LED a fotodiody optické sériové komunikace. Celá tato část je kryta obalem pro minimalizaci vniknutí okolního světla.

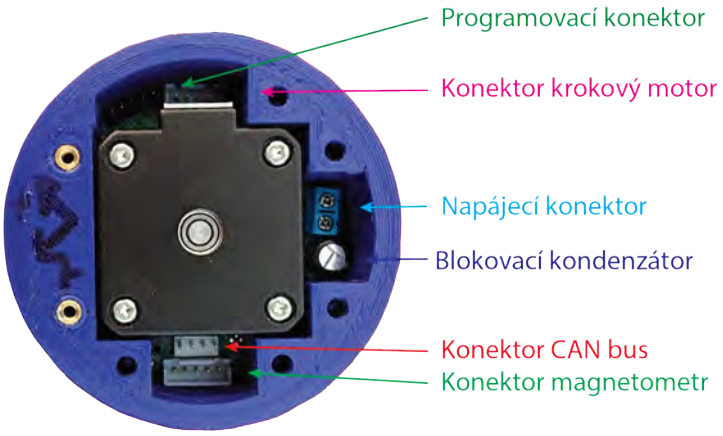
Pozice montážních otvorů spodní DPS jsou určeny pro připevnění desky ke krokovému motoru. Poloha konektorů poté tak, aby se nacházely kolem stěn krokového motoru (viz obrázek 4.12).

Pod krystalem procesoru je oddělená rozlitá zem pro minimalizaci rušivých vlivů. Tato zem je poté připojena jedním místem ke zbytku a nejkratší možnou cestou do procesoru.

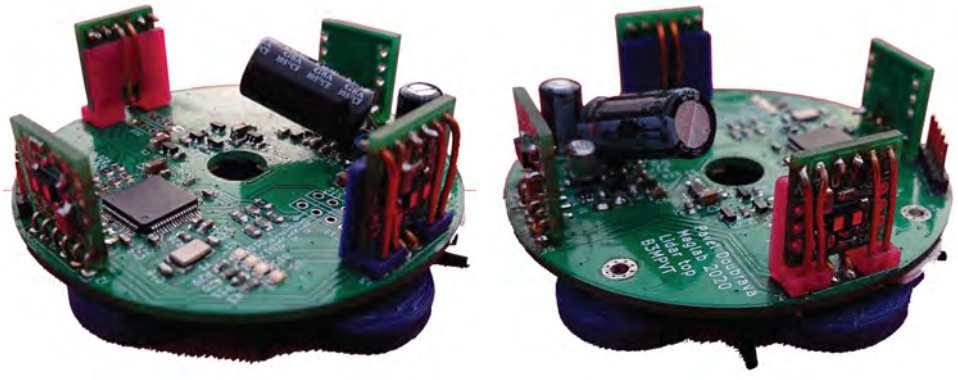
Desky plošných spojů jsem osadil. Osazené DPS s úpravami zdokumentovanými v následujících kapitolách jsou uvedeny na obrázku 4.11. Na obrázku 4.13 jsou detaily na připojení lidarových modulků k vrchní DPS.



Obrázek 4.11: Lidar: Osazené desky plošných spojů.



Obrázek 4.12: Lidar: Detail na rozvržení konektorů kolem stěn krokového motoru



Obrázek 4.13: Lidar: Detaily připojení DPS VL53L1x k vrchní DPS

4.3 Software

Při vývoji softwaru jsem použil HAL knihovny poskytované výrobcem. Vývoj softwaru probíhal ve vývojovém prostředí poskytovaném výrobcem STM32CubeIDE, které nabízí interaktivní nastavení hardwaru MCU a následné vygenerování kódu využívajícího již dříve zmíněné HAL knihovny.

Pro každý funkční celek kódu jsem vytvořil nový soubor (knihovna/library), Tyto knihovny jsem následně použil i při vývoji SW pro další MCU. Například *CAN_routines* pro ovládání periferie CAN nebo *uart_lib* pro bezdrátovou komunikaci mezi deskami.

4.3.1 Spodní DPS

Spodní mikroprocesor je hlavním řídicím prvkem celé jednotky. Návrh programu je shrnut na procesním diagramu na obrázku 4.14.

Nejdříve probíhá inicializace potřebných periferií a funkcí. Postupně proběhnou následující procesy:

- Inicializace periferií vygenerovaná pomocí Device configuration tool (STM32CUBEMX).
- Nastavení filtrů periferie CAN bus tak, aby jednotka přijímala jen zprávy určené pro ni.
- Inicializace knihovny driveru krokového motoru a kalibrace polohy.
- Inicializace inerciální měřící jednotky a kalibrace gyroskopů.
- Inicializace magnetometru.

Po inicializaci probíhá nekonečná smyčka. Zde jsou měřena, zpracovávána a odesílána data ze senzorů a vyhodnocovány příchozí zprávy ze sběrnice robota.

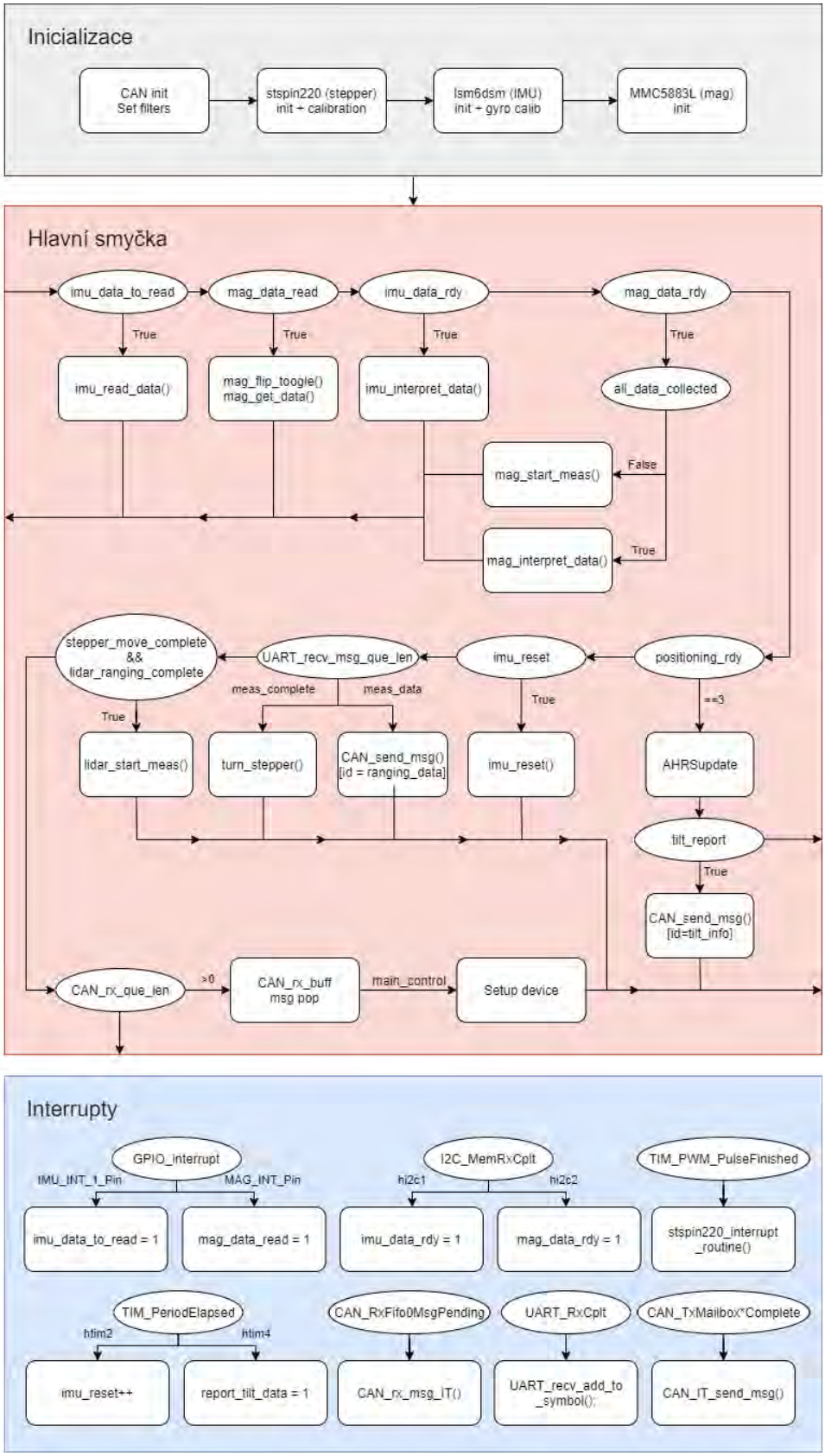
V následujících odstavcích jsou části softwaru rozděleny na jednotlivé funkční celky.

CAN bus

HAL knihovny poskytují jen funkce pro ovládání hardwaru periferie. Procesor disponuje třemi mailboxy pro odesílání zpráv a třemi pro příjem. Vytvořil jsem knihovnu *CAN_routines*, která velikost těchto mailboxu softwarově zvětšuje a celkově zjednodušuje práci s periferií.

Moje knihovna implementuje tyto uživatelem používané funkce:

- ***CAN_filter_config_user()*** - Inicializace filtrů ID rámců
- ***CAN_send_msg()*** - Funkce pro odeslání CAN rámce
- ***CAN_IT_send_msg()*** - Funkce definující interrupt handler ***HAL_CAN_TxMailbox*CompleteCallback()***



Obrázek 4.14: Lidar: Procesní diagram programu spodního MCU

- ***CAN_rx_buff_len()***
- ***CAN_rx_buff_pop()***
- ***CAN_rx_msg_IT()*** - Funkce definující interrupt handler ***HAL_CAN_RxFifo*MsgPendingCallback()***
- ***count_counter_settings()*** - Doplňková funkce umožnující spočítat hodnoty registrů časovače pro nastavení konkretní periody
- ***set_reporting_interval()*** - Doplňková funkce nastavující časovač pro generování interruptu s konkretní periodou

Knihovna uvnitř používá dva buffery, jeden pro příjem a druhý pro odesílání zpráv. V případě, kdy jsou všechny odesílací mailboxy plné, je zpráva uložena do bufferu. Po odeslání každé zprávy je vygenerován interrupt a zavolán interrupt handler *HAL_CAN_TxMailbox*CompleteCallback()*, ten případně doplní HW mailbox o zprávu z bufferu pomocí funkce *CAN_IT_send_msg()*.

V případě, že periferie příjme zprávu, která splňuje nastavené filtry, je vygenerován interrupt a v příslušném interrupt handleru je použita funkce *CAN_rx_msg_IT()*, pomocí které je zpráva přesunuta do bufferu. Zpracování zpráv následně probíhá pomocí funkcí *CAN_rx_buff_len()* a *CAN_rx_buff_pop()*.

Pro použití knihovny je nejdříve potřeba nastavit periferii CAN použitého MCU. CAN bus chci provozovat při přenosové rychlosti $f_{CAN} = 1\,\text{Mbit/s}$. Vstupní hodiny mají frekvenci $f_{clk} = 72\,\text{MHz}$. Pro nastavení přenosové rychlosti sběrnice potřebuji nastavit hodnotu prescaleru (PSC) a počet časových kvant (n_{tq}) tak, aby platil následující vztah:

$$f_{CAN} = \frac{f_{clk}}{(PSC + 1) \cdot n_{tq}} = \frac{72\,\text{MHz}}{(4 + 1) \cdot 9}. \tag{4.4}$$

Časové kvanta je dále potřeba rozdělit na 3 části (viz kapitola 2.1.3 a [19] (kapitola 24.7.7 Bit timings)):

- **Synchronization segment (SYNC_SEG)** - Zde dochází k detekci hrany. Fáze má fixovanou délku jednoho časového kvanta.
- **Bit segment 1 (BS1)** - Určuje pozici vzorkovacího bodu a spojuje propagační fázi s fází 1 standardu CAN. Délka je nastavitelná v rozsahu 1 - 16 časových kvant. Může být podle standardu automaticky prodloužena pro kompenzaci pozitivního fázového driftu.
- **Bit segment 2 (BS2)** - Reprezentuje fázi 2 standardu. Délka je nastavitelná v rozsahu 1 - 8 časových kvant. Může být automaticky zkrácena pro kompenzaci negativního fázového driftu.

Podle předchozích informací potřebuji osm časových kvant rozdělit mezi BS1 a BS2. Není přesně dané, jak se mají časová kvanta rozdělit. Na základě

principu funkce (od BS1 se mohou časová kvanta automaticky odebírat a k BS2 přidávat) se používá $BS1 > BS2$. Zvolil jsem následující konfiguraci:

$$\begin{aligned} PSC &= 4 \\ n_{tq} = SYNC\_SEG + BS1 + BS2 &= 9 \\ n_{SYNC\_SEG} &= 1 \\ n_{BS1} &= 5 \\ n_{BS2} &= 3 \end{aligned} \tag{4.5}$$

Následně je potřeba aktivovat Tx a Rx interrupty periferie CAN v nastavení NVIC (Nested Vector Interrupt Control).

Po vygenerování kódu je potřeba přidat do sekce *USER CODE 4* souboru main.c následující interrupt handlery:

```
1  /* USER CODE BEGIN 4 */
2  void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *
       hcan) {
3      CAN_rx_msg_IT(hcan, CAN_RX_FIFO0);
4  }
5  void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *
       hcan) {
6      CAN_rx_msg_IT(hcan, CAN_RX_FIFO1);
7  }
8
9  void HAL_CAN_TxMailbox0CompleteCallback(CAN_HandleTypeDef *
       hcan) {
10     CAN_IT_send_msg(hcan);
11 }
12 void HAL_CAN_TxMailbox1CompleteCallback(CAN_HandleTypeDef *
       hcan) {
13     CAN_IT_send_msg(hcan);
14 }
15 void HAL_CAN_TxMailbox2CompleteCallback(CAN_HandleTypeDef *
       hcan) {
16     CAN_IT_send_msg(hcan);
17 }
18 /* USER CODE END 4 */
```

Ovládání krokového motoru

Pro ovládání krokového motoru jsem vytvořil knihovnu **stspin220_driver**. Driver je řízen celkově 7 signály:

- ***MODE 1*** - Nastavení krokování
- ***MODE 2*** - Nastavení krokování
- ***STCK/MODE 3*** - Krokovací hodinový signál / nastavení krokování
- ***DIR/MODE 4*** - Směr otáčení / nastavení krokování

- ***ENABLE/FAULT*** - Aktivace výstupu, vynucená logická nula při detekci chyby
- ***STBY/RESET*** - Aktivace módu s nízkou spotřebou
- ***VREF_PWM*** - Nastavení proudu motorem

Proud krokovým motorem řídím pomocí střídy příslušného signálu PWM.

Při každé náběžné hraně signálu *STCK* driver posune rotor motoru o jeden mikrokrok/krok směrem určeným signálem *DIR*. Maximální frekvence hodinové signálu *STCK* je 1 MHz.

Při náběžné hraně signálu *STBY* je navzorkována úroveň všech signálů *MODE 1/2/3/4* a podle navzorkovaných hodnot je určeno mikrokrokování. Poté již signály *STCK/MODE 3* a *DIR/MODE 4* slouží k ovládání otáčení krokového motoru.

Knihovna krokového motoru implementuje tyto funkce:

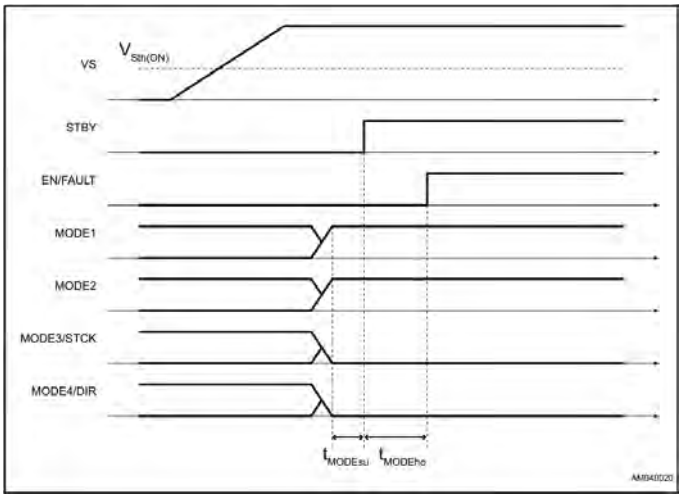
- ***stspin220_init()***
- ***stspin220_set_microstepping()***
- ***stspin220_enable()***
- ***stspin220_stby()***
- ***stspin220_fullstep()***
- ***stspin220_set_number_of_positions()***
- ***stspin220_next_position()***
- ***stspin220_set_position()***
- ***__stspin220_interrupt_routine()***

Nejdříve je potřeba provést inicializaci driveru. Toho docílíme pomocí funkcí *stspin220_init()* a *stspin220_set_microstepping()*, které implementují postup zachycený na obrázku 4.15.

Pro generování signálů *STCK* a *VREF_PWM* jsem použil dva kanály časovače 1. Frekvenci časovače jsem nastavil tak, aby rychlost otáčení senzoru byla optimální.

Jednou ze základních implementovaných funkcí je přesné otočení o daný počet mikrokroků. K tomu potřebuji počítat náběžné hrany řídicího signálu *STCK* a při dosažení požadovaného počtu mikrokroků deaktivovat hodinový signál *STCK*. Tuto část jsem vyřešil pomocí aktivace interruptů pro periferii časovače. Vytvořil jsem interrupt handler, který při každém zavolání zmenší počet mikrokroků, o kolik se rotor má otočit. Až toto číslo dosáhne nuly, je hodinový signál (CH1 TIM1) přímo v interrupt handleru deaktivován.

Moje aplikace vyžaduje, abych jedno otočení krokového motoru rovnoměrně rozdělil na předem daný počet pozic a opakovaně jsem motor do těchto pozic natáčel. (Měřil v nich vzdálenost pomocí lidarů na vrchní



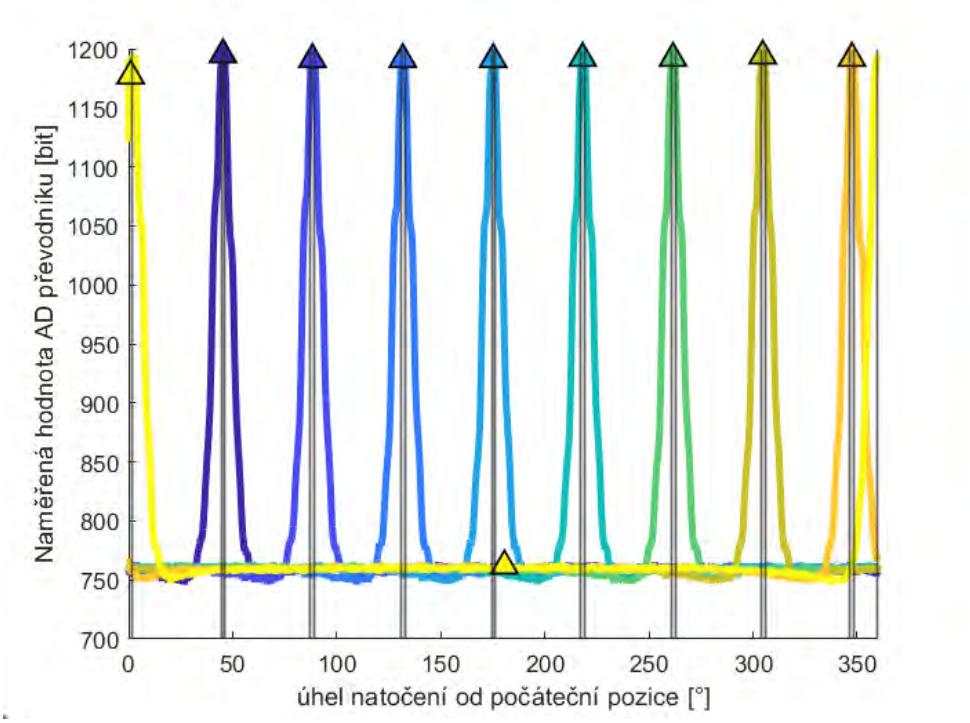
Obrázek 4.15: Lidar: Inicializační sekvence driveru krokového motoru. [7]

DPS.) Implementoval jsem tedy funkci *stspin220_set_number_of_positions()*, která rozdělí celkový počet mikrokroků do požadovaného počtu pozic. Pozice je následně možné měnit pomocí funkcí *stspin220_next_position()* a *stspin220_set_position()*. Při implementaci jsem musel ošetřit nedělitelnost celkového počtu mikrokroků zvoleným počtem pozic a zbytek po dělení rozdělit mezi jednotlivé pozice.

Rotor krokového motoru směruje lidary na vrchní DPS. Při zapnutí lidarové jednotky je potřeba zkalibrovat natočení vrchní části modulu. U krokového motoru nejsme schopni zjisti absolutní polohu rotoru. Proto jsem do pláště vrchního dílu přidal magnet, který detekuji pomocí analogové Hallovy sondy, umístěné na spodní DPS (viz obrázek 4.3).

Při kalibraci uměle zvětším počet pozic na 3200 pro zvýšení přesnosti určení pozice magnetu (úhlové rozlišení 6,75′). Následně pro každou polohu změřím velikost magnetického pole. Použitý magnet nedává ostré maximum (je moc široký). V naměřených datech je vrchol s ostrými hranami a plochou mezi nimi. Z důvodu plochého maxima nehledám jen absolutní maximum, ale interval, ve kterém jsou naměřené hodnoty magnetické indukce, v dovoleném rozsahu od maximální naměřené hodnoty. Následně vypočítám střed intervalu, označím jej jako referenční pozici, otočím se do něj a vrátím původní nastavení počtu pozic.

Na obrázku 4.16 lze vidět ukázku několika kalibračních měření začínajících s jinou počáteční pozicí. Svislými čárami je zde vyznačen detekovaný interval a trojúhelníkem detekovaná referenční pozice. Lze zde pozorovat špatné určení referenční pozice (žlutá křivka) při vrcholu rozděleném na okraje pole. Původní algoritmus s tímto scénářem nepočítal. Opravený algoritmus již detekuje všechny pozice správně.



Obrázek 4.16: Lidar: Kalibrace natočení vrchní části modulu za pomocí měření magnetického pole magnetu Hallovou sondou.

Inerciální jednotka

Použitá inerciální jednotka měří absolutní hodnotu zrychlení a úhlovou rychlost. IC je připojen k MCU pomocí sběrnice I2C. Výrobce již poskytuje knihovnu implementující práci se zařízením.

Knihovna umožňuje použití periferie DMA a z toho plynoucí neblokující čtení a zápis dat. Další výhodou je její jednoduché použití s jakýmkoliv zařízením disponujícím sběrnicí I2C, stačí jen naimplementovat funkce komunikace po sběrnici a přidat je do handleru knihovny.

Nejdříve jsem musel naimplementovat tyto funkce, dále jsem pro zpřehlednění kódu v souboru *main.c* tuto knihovnu zapouzdřil do vlastní knihovny *imu_control*, která shlukuje funkce původní knihovny na následující:

- ***imu_init()***
- ***imu_calibrate()***
- ***imu_read_data()***
- ***imu_interpret_data()***

Funkce *imu_init()* provádí nastavení zařízení. Inerciální jednotka je nastavena tak, aby sama vzorkovala data a generovala interrupty při dostupnosti nových dat. Funkce provede postupně následující operace:

1. Detekce senzoru

2. Restart IC
3. Nastavení rozsahu měřených veličin ($g_{max} = 2\,\mathrm{g}$, $\omega_{max} = 250\,\mathrm{deg/s}$)
4. Aktivace a nastavení FIFO pro naměřené hodnoty.
5. Aktivace interruptů
6. Nastavení frekvence vzorkování měřených veličin. ($f_{vz} = 416\,\mathrm{Hz}$)

Pro zpřesnění naměřených dat úhlové rychlosti je nejdříve provedena kalibrace offsetu(imu_calibrate()). Kalibraci provádím opakovaným měřením úhlové rychlosti (senzor musí být v klidu). Naměřená data jsou zprůměrována a následně odečítána od každé naměřené hodnoty (*imu_interpret_data()*).

Po detekci interruptu od IC je použita funkce *imu_read_data()*. Tato funkce aktivuje čtení surových dat ze senzoru za pomocí DMA. Dokončení čtení dat je signalizováno dalším interruptem. Následuje použití funkce *imu_interpret_data()*, která odečte offset gyroskopů a interpretuje vyčtená data. Získám hodnoty v jednotkách $\mathrm{m\,deg/s}$ a g, které jsou dále zpracovány v AHRS algoritmu.

Kvůli zvolenému módu čtení dat jsem při debugování kódu narazil na problém. Při zaplnění bufferu a vyčtení jen potřebného počtu vzorků již není senzorem vygenerován interrupt *data_rdy*. Nejdříve je potřeba vyprázdnit jeho frontu, aby zařízení fungovalo správně. Přidal jsem tedy časovač, který po určitém časovém intervalu bez čtení dat ze senzoru vyčte všechna data a tím opět nastartuje klasický režim čtení dat. K tomuto jevu nedojde při normálním běhu aplikace. Dochází k němu při pozastavování aplikace (při debugování).

Magnetometr

Jak již bylo zmíněno, magnetometr nemůže být umístěn přímo na navrhovanou DPS kvůli blízkému krokovému motoru. Z tohoto důvodu jsem použil desku s magnetometrem MMC5883 používanou v předmětu Laboratoře průmyslové elektroniky.

Pro ovládání senzoru jsem použil knihovnu používanou v tomto předmětu. Knihovnu jsem musel upravit a dokončit, protože je určena pro programovací prostředí mbed a části knihovny nejsou implementovány z edukačních důvodů (studenti je musí naimplementovat sami). Do knihovny jsem dále přidal funkcionalitu neblokujícího čtení pomocí DMA nebo interruptů.

Výsledná knihovna implementuje tyto high-level funkce:

- ***MMC5883L_init()***
- ***MMC5883L_flip_set()***
- ***MMC5883L_flip_reset()***
- ***MMC5883L_flip_toogle()***

- ***MMC5883L_start_measurement_temp()***
- ***MMC5883L_start_measurement_mag()***
- ***MMC5883L_get_XYZ_RAW_data_DMA()***
- ***MMC5883L_get_XYZ_RAW_data_blocking()***
- ***MMC5883L_measure_XYZ_RAW()***
- ***MMC5883L_interpret_data()***

Upravil jsem všechny funkce tak, aby měly první argument typu *MMC5883L_handle_t*. Tato struktura nese všechny proměnné používané knihovnou, jako jsou například adresa pro handler používané periferie I2C, stav magnetizace senzoru, pointery na funkce pro zápis a čtení dat z I2C. Tento přístup je někdy nazýván jako objektové programování v jazyce C.

Funkce *init()* provádí inicializaci zařízení, nastavuje nejvyšší vzorkovací rychlost a aktivuje interrupt "data ready". Při volání funkcí na čtení dat jsou data uložena v struktuře handleru. Naměřená data lze získat pomocí funkce *MMC5883L_interpret_data()*.

Pro správný běh knihovny je potřeba aktivovat GPIO interrupty s detekcí vzestupné hrany na pinu MCU připojeném k interrupt pinu magnetometru. Aktivaci interruptu je následně nutné potvrdit v nastavení NVIC. Poté jen stačí přidat do vygenerovaného kódu interrupt handler a při detekci interruptu přečíst data ze senzoru.

Použitý senzor k měření magnetického pole používá jev anisotropní magnetoresistance. Senzory pracující na základě tohoto jevu jsou známy svým offsetem. Pro jeho odstranění je potřeba měnit magnetizaci senzoru (tzv stavy set a reset). Naměřené hodnotě s opačnou magnetizací se změní znaménko ale offset zůstane stejný. Postup jak odstranit offset je uveden v následujících rovnicích:

$$B_{set} = B_{real} + B_{offset} \tag{4.6}$$

$$B_{reset} = -B_{real} + B_{offset} \tag{4.7}$$

$$B_{real} = \frac{B_{set} - B_{reset}}{2} = \frac{B_{real} + B_{offset} - (-B_{real} + B_{offset})}{2} = B_{real}. \tag{4.8}$$

Po implementaci flipování jsem udělal skalární kalibraci senzoru (viz kapitola 2.3.5. Kalibraci jsem prováděl v domácích podmínkách, zvolil jsem tedy jako referenci vektor magnetického pole Země. Absolutní hodnota velikosti vektoru pro naši aplikaci není důležitá, stačí mi jen její odhad. Naměřené hodnoty budou použity pro určení směru magnetického severu. Potřebuji, aby jednotlivé osy měly stejný zisk a neměly offset. V dalších výpočtech se zohledňuje jen vzájemná velikost jednotlivých složek vektoru. Velikost vektoru jsem zvolil $B = 49\,000\,\mathrm{nT}$.

Nejdříve bylo potřeba nasbírat data. V knihovně magnetometru jsem přidal nové makro *CALIBRATE_MAG*. Pomocí konstrukce klíčových slov

b_x [LSB]	1572
b_y [LSB]	-200
b_z [LSB]	564
S_x [-]	$0,911201 \cdot 24,414062$
S_y [-]	$0,938749 \cdot 24,414062$
S_z [-]	$0,888292 \cdot 24,414062$
u_1 [°]	$-1,16145$
u_2 [°]	$0,30964$
u_3 [°]	$2,58632$

Tabulka 4.1: Lidar: Výsledky kalibrace magnetometru

#ifdef, *#else* a *#endif* jsem upravil funkci interpretující data tak, aby při nadefinování makra *CALIBRATE_MAG* jen provedla průměr ze vzorků s magnetizací SET a RESET. Do hlavní smyčky programu jsem přidal kód, který posílá tyto surová data hned po interpretaci přes sběrnici robota (opět jen při definici makra). Výhodou této realizace je nekompilování kalibračního kódu při zakomentování makra *CALIBRATE_MAG*. Zprávě s daty jsem přiřadil vlastni ID.

V Raspberry Pi jsem upravil uzel zpracovávající zprávy ze sběrnice. Pro data jsem vytvořil vlastní topic a přidal další ROS uzel, který tyto zprávy ukládá do souboru.

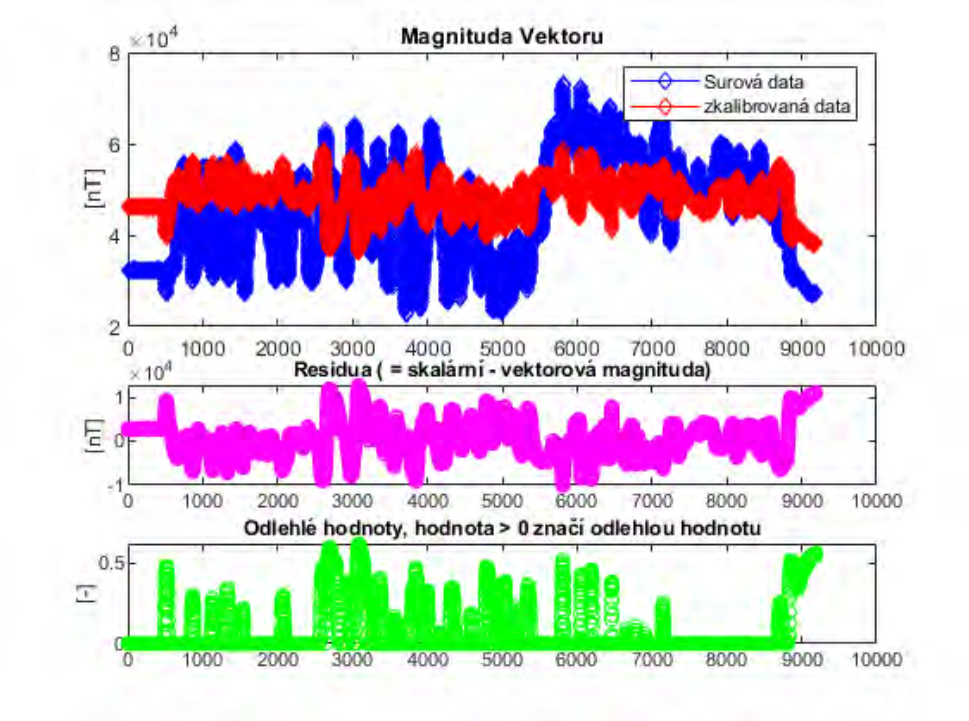
Při sbírání dat jsem se snažil magnetometr natočit do co nejvíce poloh a před započetím měření jsem se snažil vyčistit okolí senzoru od magnetických předmětů.

Data jsem následně zpracoval v MATLABu pomocí scriptu ze článku [5]. Postup kalibrace je vysvětlen v kapitole 2.3.5. Data jsem před použitím kalibračního algoritmu přenásobil konstatním ziskem senzoru (24.414062), na základě jeho měřeného rozsahu a rozlišení.

Výsledek kalibrace je na obrázku 4.17. Výsledné kalibrační konstanty následně v tabulce 4.1, kde b_i jsou offsety, S_i zisk a u_i - úhly neortogonality.

Chybu v ortogonalitě jsem zanebdal a v MCU implementoval jen korekci offsetu a zesílení. Pro další ušetření výpočetního výkonu jsem velikost offsetu přepočítal na počet bitů a odečítám jej před převedení čísla z typu *int* na *float*, následně již aplikuji jen zisk. Následuje výsledná funkce pro interpretaci dat. První 3 hodnoty v poli *mag_data.u16bit* odpovídají magnetizaci SET a další tři RESET.

```
1  int32_t MMC5883L_interpret_data(MMC5883L_handle_t *handle,
       float *result)
2  {
3  #ifdef CALIBRATE_MAG
4      for (int i = 0; i < 3; i++)
5      {
6          result[i] = (float)(handle->mag_data.u16bit[i] -
               handle->mag_data.u16bit[i + 3]);
7      }
8  #else
```



Obrázek 4.17: Lidar: Výsledky kalibrace magnetometru

```
9       for (int i = 0; i < 3; i++)
10      {
11          result[i] = (float)(handle->mag_data.u16bit[i] -
                handle->mag_data.u16bit[i + 3] - bit_offsets[i])
                * gains[i];
12      }
13  #endif
14      handle->data_offset = 0;
15      return 1;
16  }
```

Attitude and Heading Reference System

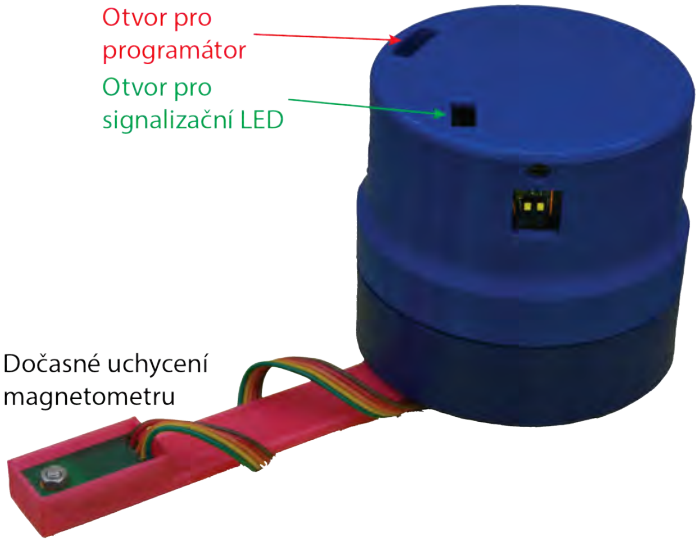
Pro fůzi dat z inerciální jednotky a magnetometru jsem použil již naimplementovanou knihovnu využívající Madgwickův algoritmus (viz. teoretický rozbor 2.3).

Algoritmus je provozován na frekvenci 208Hz. Funkce aktualizace parametrů je volána ihned, jak jsou dostupná nová data z akcelerometru a magnetometru.

Při vývoji jsem zkusil několik knihoven implementujících Madgwickův algoritmus. První použitá knihovna nebyla dostatečně zdokumentována, musel jsem postupem pokus omyl určit v jakých jednotkách knihovna přijímá data.

Jedním z dalších faktorů je stejná orientace os jednotlivých senzorů. Protože magnetometr je umístěn mimo tělo zařízení, vytvořil jsem při vývoji dočasné uchycovací rameno, které drží magnetometr ve stejné orientaci jako inerciální

jednotku (viz. obrázek 4.18).



Obrázek 4.18: Lidar: Připevnění magnetometru při vývoji modulu před připevněním modulu ke konstrukci robota.

Dalším důležitým faktorem při zprovoznění AHRS byla kalibrace magnetometru. Při prvních testech mě Eulerův úhel *yaw* postupně konvergoval k jedné hodnotě. Při kalibraci offsetů a zisků magnetometru se tato chyba odstranila. Algoritmus je velmi citlivý na okolní feromagnetické předměty. Například pokud senzor přiblížím ke klávesnici, změní se úhel *yaw* až o 10°. Velikost použitého upevňovacího ramene je nedostatečná pro úplné potlačení vlivu krokového motoru na určení orientace, V průběhu otáčení krokového motoru má úhel *yaw* rozkmit více jak 5°.

Po montáži jednotky na robota jsem musel zopakovat kalibrační postup, kvůli rozdílné pasivní distrozi magnetického pole zapříčiněné jiným okolním prostředím.

Při tvorbě textu diplomové práce jsem narazil na aktualizovanou knihovnu implementující algoritmus přímo od tvůrce. Tato knihovna přidává funkcionalitu tzv. on-the-fly (za běhu algoritmu) kalibrace offsetu gyroskopu. Rozhodl jsem se tedy nahradit používanou knihovnu nově objevenou.

4.3.2 Komunikace mezi deskami

Při testování komunikace jsem zjistil, že mnou navržený design a použité součástky nejsou schopny dosáhnout plánované přenosové rychlosti. Aby rychlost přenosu dat mezi deskami nezpomalovala funkcionalitu zařízení, použil jsem neblokujících funkcí využívající interrupty. Chtěl jsem nejdříve použít periferii DMA, ale již mám DMA kanály, které používá UART využity u periferie I2C. Oba dva přístupy umožňují paralelizovat přenos dat.

Pro přenos dat stačí předat funkci ukazatel na pole s přenášenými daty a jejich délku. Pro příjem ukazatel na pole kam se mají data uložit a délku

přijímané zprávy, následně již vše probíhá na HW úrovni mimo výpočetní vlákno aplikace. Při důležité události (dokončení přenosu, chyba parity atd.) je vygenerován příslušný interrupt.

Pro komunikaci mezi deskami jsem vytvořil vlastní knihovnu, která používá stejný koncept jako knihovna CAN bus. Jsou zde opět použity dva buffery (jeden pro Tx a druhý pro Rx). Knihovna implementuje následující funkce:

- ***UART_recv_buffer_pop()***
- ***UART_recv_msg_que_len()***
- ***UART_recv_add_to_symbol()*** - Funkce volaná v interrupt handleru, zpracuje příchozí znak.
- ***UART_deserialize_msg()*** - Funkce deserializující přijatou zprávu do struktury.
- ***UART_TX_set_instance()*** - Inicializace, předání handleru používané periferie UART
- ***UART_TX_Send()***
- ***UART_TX_Send_from_IT()*** - Funkce volaná v interrupt handleru. Pošle další zprávu, jestliže Tx buffer není prázdný.

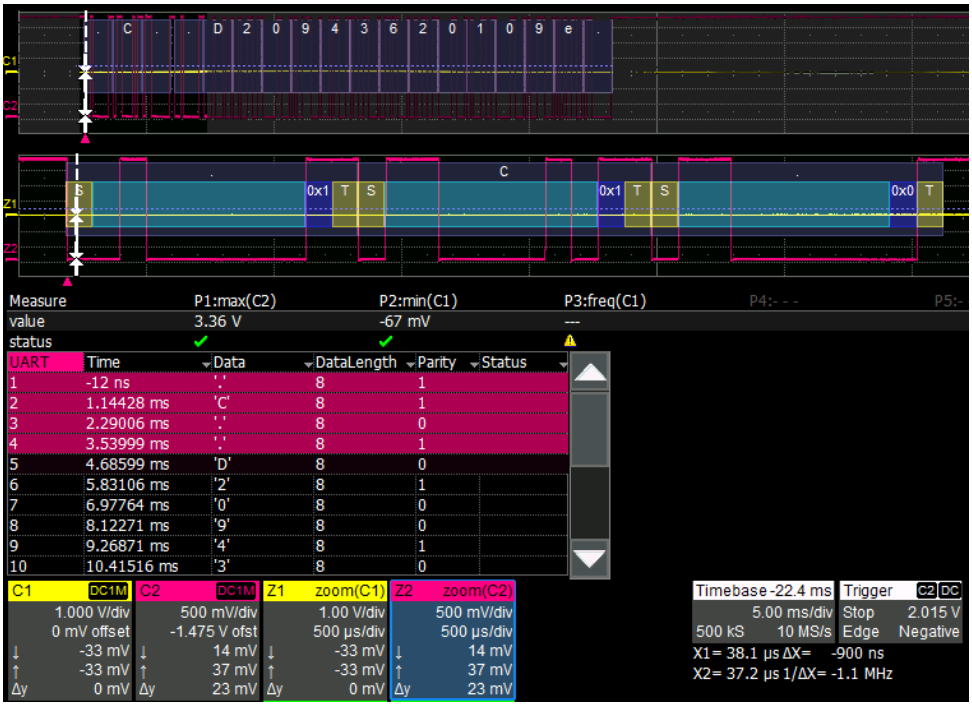
Při použití sériové linky bez arbitračních signálů (RTS, CTS) je složité přenášet data v binární formě. Může zde docházet k desynchronizaci příjemce a odesílatele a tím ke špatné interpretaci dat. Z tohoto důvodu jsem se rozhodl přenášet data v human-readable formě za pomocí ASCII znaků. Pro označení začátku zprávy jsem použil znak *STX* (start of text) - '\0x02' a pro označení konce zprávy znak *ETX* (End of text) - '\0x03'. První užitečný byte v rámci (po arbitračním znaku *STX*) označuje typ zprávy, jestliže daná zpráva nese data, následují data.

Zde jsou všechny používané zprávy:

- ***START_MEASUREMENT*** - [STX 0x02]S[ETX 0x03]
- ***MEASUREMENT_COMPLETE*** - [STX 0x02]C[ETX 0x03]
- ***MEASUREMENT_DATA*** - [STX 0x02]D%3x%3x%3x%3x[ETX 0x03]

Zprávu MEASUREMENT_COMPLETE jsem použil, pro zrychlení odezvy mezi koncem jednoho měření a začátkem dalšího, protože přenos celé zprávy MEASUREMENT_DATA výrazně zpomaloval framerate senzoru. Z tohoto důvodu přenos dat probíhá hned po odeslání zprávy MEASUREMENT_COMPLETE, většina zprávy je odesílána v době, kdy již probíhá měření další polohy. Ale spodní procesor může dříve zareagovat na dokončené měření.

Na obrázku 4.19 je snímek z osciloskopu s přeloženou komunikací směrem od vrchní desky. Pro lepší čitelnost jsou hodnoty znaků zobrazeny ve formátu



Obrázek 4.19: Lidar: Snímek z osciloskopu zachycující komunikaci mezi deskami modulu.

ASCII. Znaky STX a ETX jsou symbolizovány tečkou. Jejich přesná hodnota lze odečíst v přiblížení v prostřední části obrázku.

Rozdílem mezi knihovnou pro CAN a knihovnou pro UART komunikaci mezi deskami je příjem zpráv. Periferie CAN přijme zprávu jako jeden celek nezávisle na její délce. Tuto funkcionalitu jsem musel u této knihovny doplnit sám. Příchozí data přijímám a zpracovávám po jednom bytu. Detekuji arbitrační znaky a podle nich dělím příchozí data do jednotlivých zpráv, které ukládám do přijímacího bufferu. Ostatní části knihovny se chovají stejně.

Pro použití knihovny je potřeba aktivovat Rx a Tx interrupty v nastavení periferie a opět do sekce *USER CODE 4* souboru main.c přidat následující interrupt handlery:

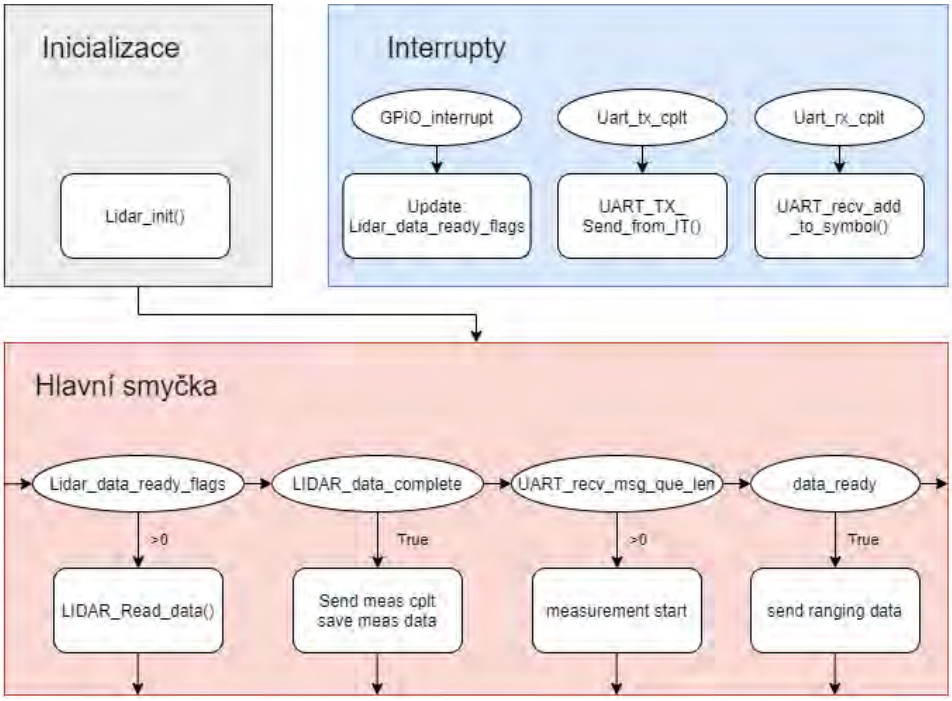
```
1  /* USER CODE BEGIN 4 */
2  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
3      UART_recv_add_to_symbol(Uart_recv_symb);
4      HAL_UART_Receive_IT(&huart1, (uint8_t*) &Uart_recv_symb,
           1);
5  }
6  void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart){
7      UART_TX_Send_from_IT();
8  }
9  void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart){
10     if(huart->ErrorCode == HAL_UART_ERROR_PE){
11         HAL_GPIO_WritePin(LED_R_GPIO_Port, LED_R_Pin,
               GPIO_PIN_SET);
```

```
12          }
13  }
14  /* USER CODE END 4 */
```

Poslední interrupt handler slouží jen jako upozornění na chybu parity.

4.3.3 Horní DPS

Jediným úkolem vrchního mikroprocesoru je měřit vzdálenost pomocí 4 připojených lidarů VL53L1x a posílat naměřená data spodnímu procesoru. Čtyři použité lidary jsou rozděleny rovnoměrně na dvě sběrnice I2C. Shrnutí návrhu programu je na procesním diagramu na obrázku 4.20



Obrázek 4.20: Lidar: Procesní diagram programu vrchního MCU.

Nejdříve program provede inicializaci potřebných periferií:

- **Inicializace periferií** vygenerovaná pomocí Device configuration tool (STM32CUBEMX)
- **Inicializace lidarů**

Po inicializaci probíhá nekonečná smyčka. Zde jsou měřena a odesílána data z lidarů.

Lidary

Výrobce použitých senzorů vzdálenosti nabízí knihovnu pro práci se senzory. Pro zpřehlednění kódu v hlavním souboru main.c jsem vytvořil knihovnu

lidar.h/.c, která implementuje ovládání všech čtyř použitých lidarů s využitím knihovny výrobce.

Moje knihovna implementuje následující funkce pro přehledné ovládání všech čtyř senzorů:

- ***LIDAR_init()***
- ***LIDAR_IRQ_handler()*** - zprcování GPIO interruptů
- ***LIDAR_Read_data()***
- ***LIDAR_start_measurement()***
- ***LIDAR_recover()***
- ***LIDAR_data_complete()***
- ***LIDAR_clear_rdy()***
- ***LIDAR_get_measured_values()***

Inicializační funkce postupně aktivuje lidary - nastaví jim unikátní I2C adresu a provede dále popsanou inicializaci. Senzory mají konfigurovatelnou dobu měření jednoho vzorku (20 - 1000 ms), pro měření krátkých vzdáleností (do 136 cm) postačuje použít dobu odběru jednoho vzorku 20 ms. Při měření středních (do 290 cm) a dlouhých vzdáleností (do 400 cm) je potřeba nastavit minimálně 33 ms. Se zvyšující se dobou odběru jednoho vzorku se zvyšuje přesnost a dosah senzoru. Pro dosáhnutí dosahu senzoru 4 m je potřeba zvolit dobu odběru vzorku minimálně 140 ms. V mé aplikaci potřebuji zvolit, co možná nejkratší dobu odběru vzorku z důvodu, co možná nejvyšší obnovovací frekvence obrazu vzdáleností okolí. Zvolil jsem tedy hodnotu 33 ms, která je nejmenší možná pro měření středních a dlouhých vzdáleností. Další nastavovanou konstantou je minimální časový rozestup mezi dvěma začátky měření. Ten musí být alespoň o 4 ms větší než doba jednoho měření. Zvolil jsem tedy hodnotu 37 ms. Dalším nastavením je volba měřeného rozsahu mezi krátkou, střední a dlouhou vzdáleností. Zde jsem zvolil dlouhou vzdálenost. Senzor má konfigurovatelný zorný úhel. Ten je možné měnit pomocí tzv. Region of interest (ROI), který určuje jaká část matice detektorů příchozího pulzu má být použita. ROI jsem nadefinoval tak, aby senzor měl vodorovně co možná nejužší zorný úhel, pro zvýšení rozlišení detekce překážek v okolním prostoru. Ve vertikálním směru jsem senzoru nastavil nejširší možný rozsah. Následně jsem zkontroloval, že měření nejsou ovlivněna odrazem signálu od podlahy a od libovolné části konstrukce robota.

Pro signalizaci událostí z jednotlivých senzorů jsem volil bitový přístup. Každý senzor má v logických proměnných přidělen jeden bit, kterým signalizuje danou událost. Například *data_ready* nebo *data_readed*.

Další funkcí je interrupt handler (*LIDAR_IRQ_handler()*), který interpretuje čísla pinů GPIO interruptů na vlajky *Lidar_data_ready_flags*.

Pokud je nějaký bit proměnné *Lidar_data_ready_flags* roven 1, značí to připravená data na vyčtení z příslušného senzoru. Toho může být docíleno pomocí *LIDAR_Read_data()*.

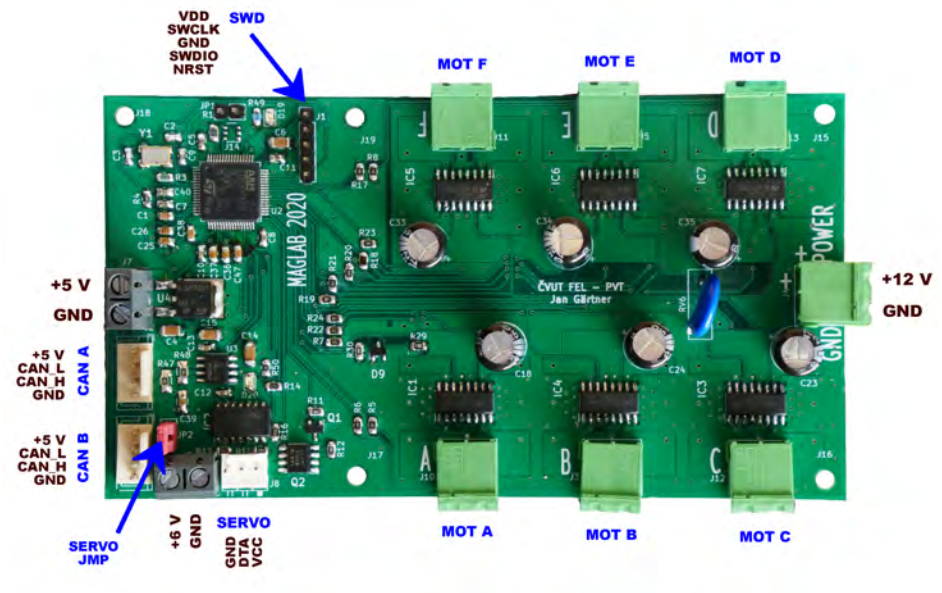
LIDAR_data_complete() udává stav vyčtení dat ze všech senzorů. Vrací *true*, pokud byla vyčtena data ze všech senzorů. Pro vyresetování stavu je potřeba zavolat funkci *LIDAR_clear_rdy()*

Ostatní funkce fungují intuitivně podle jejich názvů.

Kapitola 5

Motor Board

Jednotka řízení motorů je další z modulů připojených na sběrnici CAN bus. Jejím údělem je přijímat zprávy o požadovaném pohybu robota a ovládat motory k danému výsledku. Na obrázku 5.1 je výsledný modul s popsanými konektory. Návrh desky plošných spojů prováděl kolega Jan Gärtner v rámci předmětu Projekt v týmu.



Obrázek 5.1: Motor Board: Foto modulu

5.1 Hardware

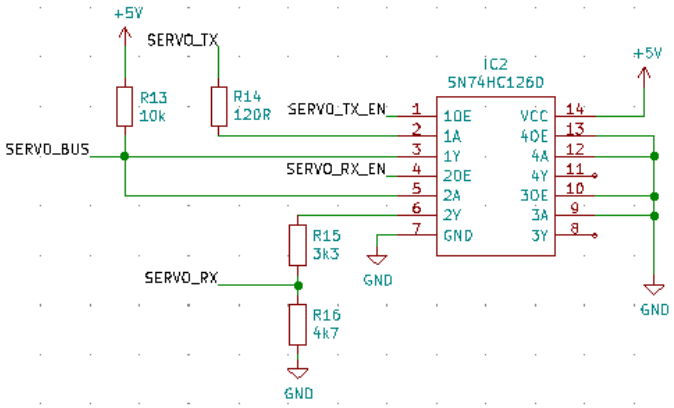
Modul má celkem tří napájecí vstupy. První je 5 V pro napájení MCU a CAN transceiveru. DC motory jsou napájeny ze separátního napájecího vstupu. Vstupní napájení se může pohybovat v rozmezí 4 – 14 V. V mé aplikaci je tento vstup napojen přímo na hlavní baterii robota o nominálním napětí 12 V. Třetím napájecím vstupem je napájení servomotorů. Tento napájecí vstup lze nahradit propojením s 5 V větví MCU pomocí jumperu JP2.

Použité servomotory je možné napájet napětím v rozsahu $5 - 8{,}4\,\mathrm{V}$. Se zvyšujícím se napájecím napětím roste jejich točivý moment separátní napájecí vstup umožňuje použít vyšší napájecí napětí, nebude-li $5\,\mathrm{V}$ dostatečné.

Rover používá chytré servomotory Lewansoul LX-16A, které jsou ovládány pomocí half-duplex sériové linky namísto PWM. Oproti klasickým servomotorům nabízejí například možnost sledovat teplotu a nastavit při jaké teplotě se mají deaktivovat, aby nedošlo k jejich poškození. Na jedné sběrnici může být připojeno až 253 servomotorů, ty jsou pak identifikovány pomocí nastavitelných identifikátorů. To nám přináší výrazné zjednodušení kabeláže robota.

Servomotory mohou pracovat ve dvou módech: *motor control* a *position control*. V prvním zmíněném módu se servomotor chová jako klasický DC motorek, můžeme zde regulovat rychlost a směr otáčení. V druhém módu je možné nastavit pozici v rozsahu 0 - 240 stupňů.

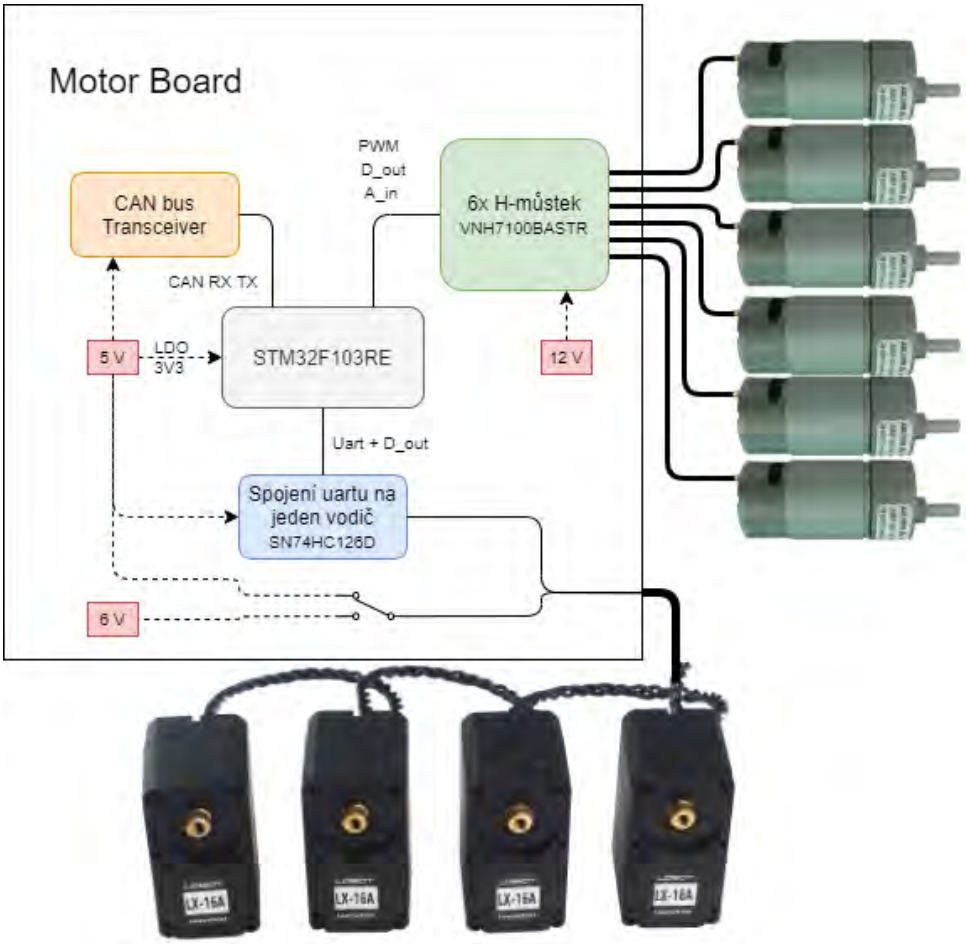
Použitý mikrokontroler v základu nepodporuje half-duplex sériovou linku. Mezi periferií UARTu MCU a signálem k servomotorům je potřeba obvod, který je uveden na obrázku 5.2. Ten přepojuje signál *SERVO_BUS* mezi *SERVO_TX* a *SERVO_RX* pinem MCU pomocí digitálních signálů *SERVO_TX_EN* a *SERVO_RX_EN*, tedy plní funkci multiplexeru. Serva mají $5\,\mathrm{V}$ logiku, pro snížení napětí signálu *SERVO_RX* byl použit napěťový dělič *R15*, *R16*. Signál *SERVO_TX* není potřeba upravovat.



Obrázek 5.2: Motor Board: Schéma zapojení obvodu pro ovládání servomotorů

Ovládání DC motorů je zajištěno pomocí H-můstků. Zvolené H-můstky mají analogový výstup pro sledování proudu tekoucího zátěží (signál *CS_**). Vzhledem ke konstrukci robota je tato hodnota velmi zásadní. Dovoluje nám detekovat nadzvednuté kolo robota a příslušně upravit výkon na ostatních kolech tak, aby se platforma srovnala.

Na obrázku 5.3 je diagram shrnující zapojení hardwaru.



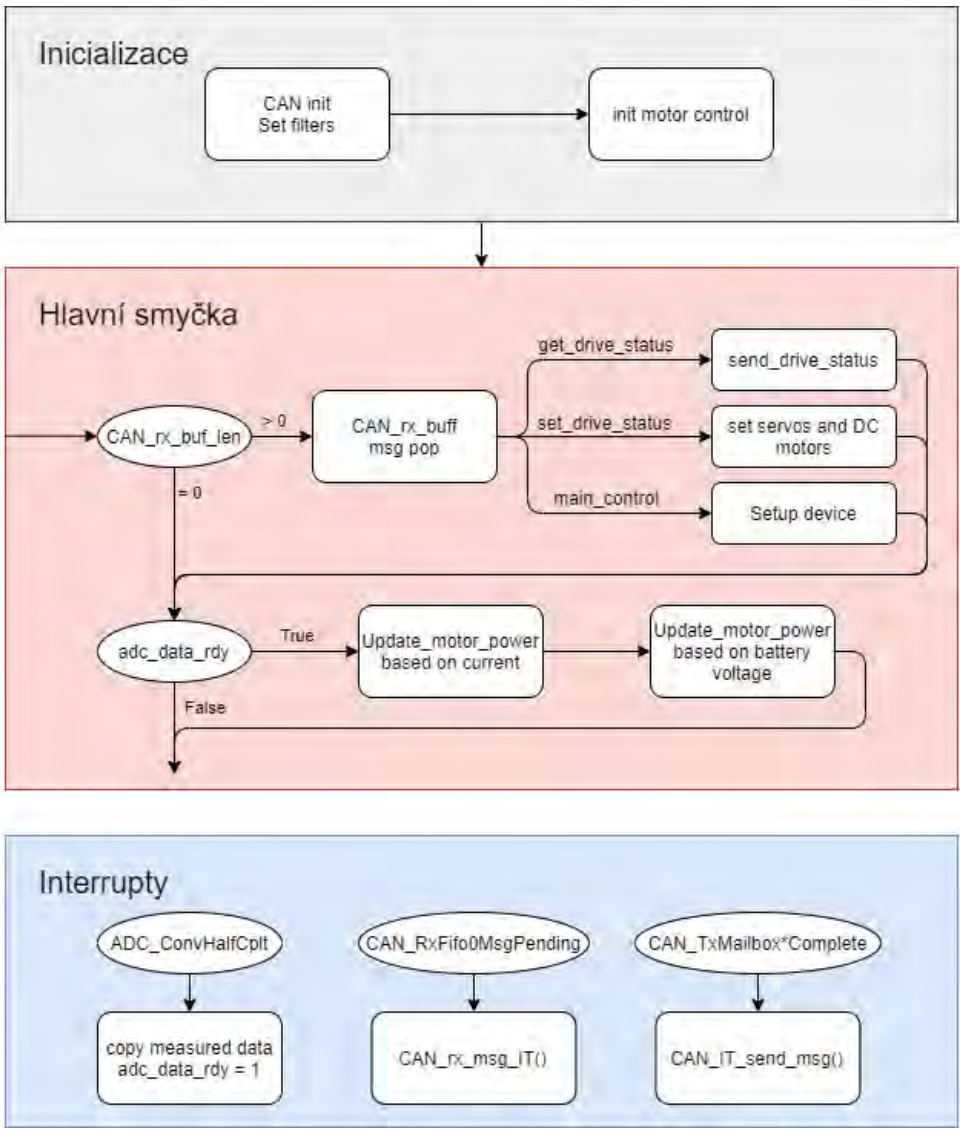
Obrázek 5.3: Motor Board: Diagram zapojení hardwaru

5.2 Software

Modul pracuje jako *slave*, nastavuje natočení servomotorů a výkon motorů podle příchozích zpráv. Procesní diagram programu je na obrázku 5.4.

Nejdříve probíhá inicializace potřebných periferií a knihoven. Postupně proběhnou následující procesy:

- **Inicializace periferií** vygenerovaná pomocí Device configuration tool (STM32CUBEMX).
- Nastavení filtrů periferie CAN bus tak, aby jednotka přijímala jen zprávy určené pro ni.
- **Aktivace kanálů PWM** pro řízení H-můstků.
- **Kalibrace servomotorů**
- **Aktivace AD převodníku**.



Obrázek 5.4: Motor Board: Procesní diagram programu

Po inicializaci probíhá nekonečná smyčka. Zde jsou vykonávány dva procesy: provedení pokynů dle příchozích zpráv ze sběrnice CAN a aktualizace výkonů motorů na základě dat z AD převodníku.

5.2.1 AD převodník

Použitý procesor disponuje třemi dvanácti bitovými AD převodníky. Před každým je umístěn multiplexer, který dovoluje napojení více signálů k jednomu převodníku s omezením, že v jeden okamžik je možné měřit jen jeden kanál.

Pro moji aplikaci jsem použil periferii ADC1, kterou používám pro zisk informace o tekoucím proudu jednotlivými H-můstky a velikosti budícího napájení. Měřím tedy celkem 7 signálů.

Data získaná z měření budou použita pro upravení výkonu jednotlivých

motorů. Pro tuto aplikaci jsem odhadl, že obnovovací frekvence $f_{vz} = 100\,\text{Hz}$ bude dostatečná. Dobu měření jednoho vzorku jsem volil, co možná největší.

Je potřeba nastavit dobu měření jednoho vzorku. To se nastavuje pomocí frekvence vstupních hodin a počtu jejich period. Vstupní hodinový signál nejdříve prochází děličkou, která jej umožňuje zpomalit.

Vstupní hodinový signál má frekvenci $f_{clock} = 72\,\text{MHz}$. Maximální hodnota hodinového signálu AD převodníku je $f_{adc,clock,max} = 14\,\text{MHz}$. Pro splnění této podmínky můžeme použít jen dělení šesti a osmi z nabízených možností děličky.

Počet vzorkovacích cyklů (sampling_time) je možné zvolit z několika předem daných hodnot. Při zohlednění počtu odebíraných vzorků a požadované frekvence vzorkování signálu, můžeme použít největší počet cyklů. I tak zde bude velká rezerva.

$$f_{adc,clock} = \frac{f_{clock}}{8} = 9\,\text{MHz} \tag{5.1}$$

$$f_{vz,real} = \frac{f_{adc,clock}}{n_{samples} * sampling\_time} = \frac{9\,\text{MHz}}{7 \cdot 239,5} \approx 5{,}4\,\text{kHz} \tag{5.2}$$

V konfiguraci převodníku jsem nastavil, aby se po instrukci na započetí měření postupně automaticky změřila napětí na všech sedmi měřených kanálech (6 x CS_* + budící napětí můstků).

Pro docílení mé zvolené vzorkovací frekvence jsem použil spouštění měření za pomoci události generované časovačem. Časovač a AD převodník jsou nastaveny tak, aby při každém dokončení periody časovače započalo nové měření AD převodníku.

Aby vzorkování AD převodníku co nejméně zatěžovalo procesor a probíhalo co nejvíce na hardwarové úrovni, použil jsem periferii DMA.

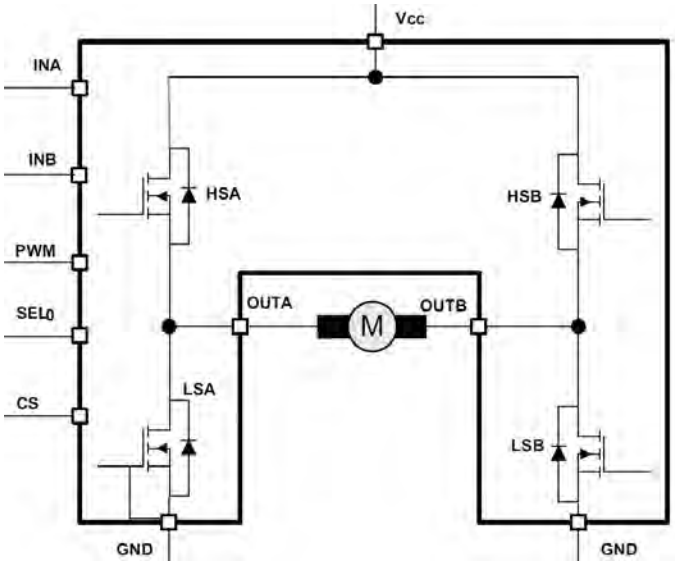
Při nasbírání poloviny a dokončení sběru všech dat je vygenerován příslušný DMA interrupt. V příslušném interrupt handleru jsou data z pole používaného DMA překopírována do druhého pole, kvůli zamezení nechtěnému přepisu dat v průběhu práce s nimi. Následně je pomocí proměnné hlavní smyčce signalizována připravenost nových dat z AD převodníku na zpracování.

5.2.2 Řízení motorů

Jednotka disponuje 6 H-můstky. Každý H-můstek je buzen vlastním signálem PWM. Ostatní řídicí signály (INA, INB, SEL0) jsou spojeny do trojic vždy pro jednu stranu robota (kanál A+B+C a D+E+F).

Použitý H-můstek je možné provozovat do frekvence PWM 20 kHz. Zvolil jsem nejvyšší možnou frekvenci z důvodu eliminaci nepříjemného zvuku při nižších frekvencích.

Nastavení operačního módu H-můstků probíhá pomocí pinů IN_A, IN_B a SEL_0. V tabulce 5.1 je uveden přehled nastavení těchto módů. Sloupec CS označuje stav pinu pro monitorování proudu můstkem. Sloupce HSA, LSA, HSB a LSB označují stav sepnutí jednotlivých vnitřních tranzistorů můstku (viz zapojení na obrázku 5.5). HSDA a ASDB označuje připojení k signálu drain příslušných mosfet tranzistorů.



Obrázek 5.5: Motor Board: Schéma zapojení h-můstku. [8]

Na řádku číslo 1 je konfigurace brzdění s monitorováním proudu a na řádku 6 je brzdění bez monitorování proudu. Konfigurace na řádcích 2,3,4 a 5 jsou pro nás nejdůležitější. Tyto konfigurace nastavují tok proudu zátěží. Na řádku 7 je odpojení motoru od řídicích obvodů. Na posledním je přepnutí modulu do režimu standby.

	Pin status					Mosfet status				
#	IN_a	IN_b	SEL_0	PWM	CS	HSA	LSA	HSB	LSB	Stav motoru
1	1	1	1	x	Monitorování proudu HSDA	On	Off	On	Off	Brzdění
			0		Monitorování proudu HSDB					
2	1	0	1	1	Monitorování proudu HSDA	On	Off	Off	On	Pohyb vpřed
				0					Off	
3	1	0	0	1	Hi-Z	On	Off	Off	On	Pohyb vpřed
				0					Off	
4	0	1	1	1	Hi-Z	Off	On	On	Off	Pohyb vzad
				0			Off			
5	0	1	0	1	Monitorování proudu HSDB	Off	On	On	Off	Pohyb vzad
				0			Off			
6	0	0	1	1	Hi-Z	Off	On	Off	On	Brzdění
			0							
7	0	0	1	0	x	Off	Off	Off	Off	Volný pohyb
			0							

Tabulka 5.1: Motor Board: Konfigurace h-můstku

Zpětná vazba regulace výkonu

Jak již bylo zmíněno použité můstky poskytují zpětnou vazbu o velikosti protékajícího proudu. Tuto hodnotu měřím pomocí AD převodníku (viz kapitola 5.2.1) a následně jsou tyto hodnoty zpracovány v hlavní smyčce programu.

Pro aktualizaci výkonu se v hlavní smyčce volá funkce *motor_update_load()*. Tato funkce nejdříve vypočítá průměrnou hodnotu tekoucího proudu můstky. Následně je pro každý motor zvlášť proveden výpočet regulačního zásahu. V současné aplikaci je použit proporcionální regulátor, který se ukázal jako dostatečný.

Hodnoty výkonů jsou následně upraveny o koeficient získaný na základě napětí baterie. Tento koeficient slouží k tomu, aby se co nejvíce potlačil vliv nabití baterie na pohyb robota.

Pokud je výsledná hodnota výkonu motoru větší než je maximální povolená, jsou hodnoty přeškálovány.

5.2.3 Ovládání servomotorů

Serva jsou ovládána pomocí sériové linky přenášené po jednom signálovém vodiči (viz kapitola. 5.1).

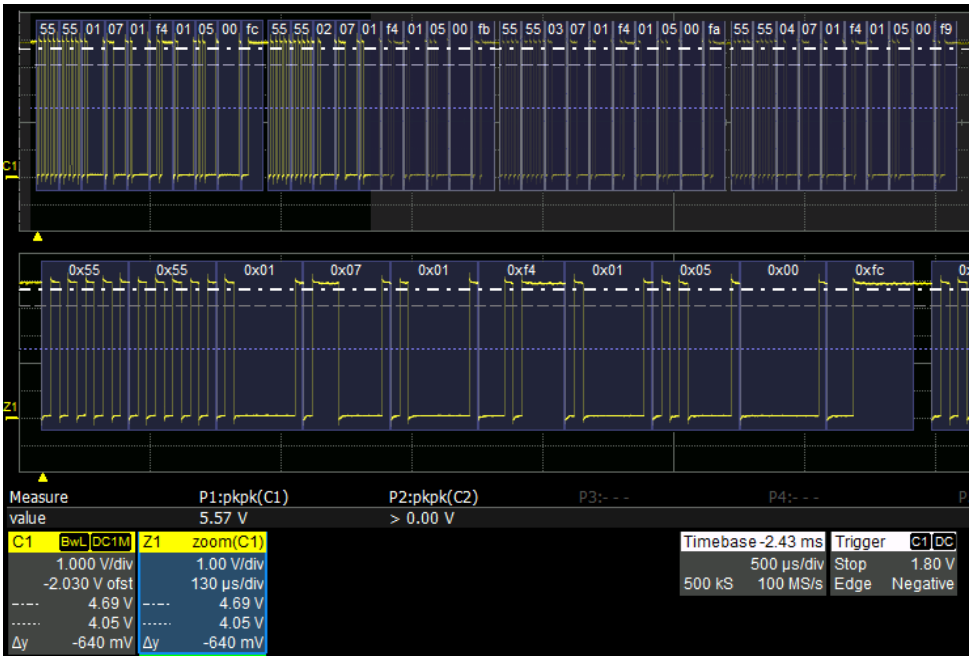
Složení zprávy je uvedeno v tabulce 5.2. Nejdříve je odeslána synchronizační hlavička ve formě dvou bytů 0x55. Následuje unikátní identifikátor servomotoru, délka a typ instrukce, proměnný počet parametrů v závislosti na typu instrukce. Zpráva je zakončena kontrolní sumou všech bytů po synchronizační hlavičce. Více informací o komunikačním protokolu lze nalézt v [20].

Header	ID number	Data Length	Command	Parameter	Checksum
0x55 0x55	ID	Length	Cmd	Prm 1... Prm N	Checksum

Tabulka 5.2: Motor Board: Struktura rámce komunikace se servem

Pro ovládání serv byla vytvořena knihovna, která implementuje instrukce čtení a zápisu. Ty jsou následně rozšířeny na funkce implementující jednotlivé používané příkazy (set_offset, set_mode, set_max_temperature atd.), které jsou použity v hlavní smyčce.

Na obrázku 5.6 je ukázka komunikace nastavení úhlu jednotlivých serv. Lze zde pozorovat měnící se ID hned po dvojici bytů 0x55, 0x55, následuje délka dat, příkaz, úhel natočení serva v rozsahu 0 - 1000 bitů. Zpráva je zakončena kontrolní sumou.



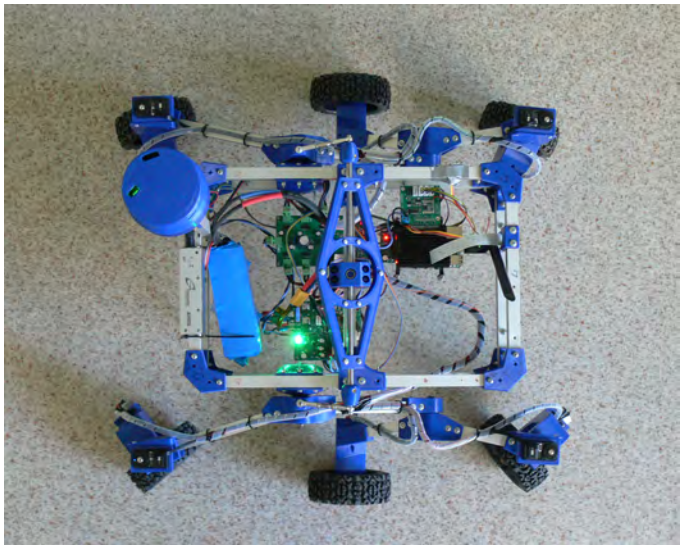
Obrázek 5.6: Motor Board: Ukázka komunikace se servomotory.

5.2.4 Komunikace

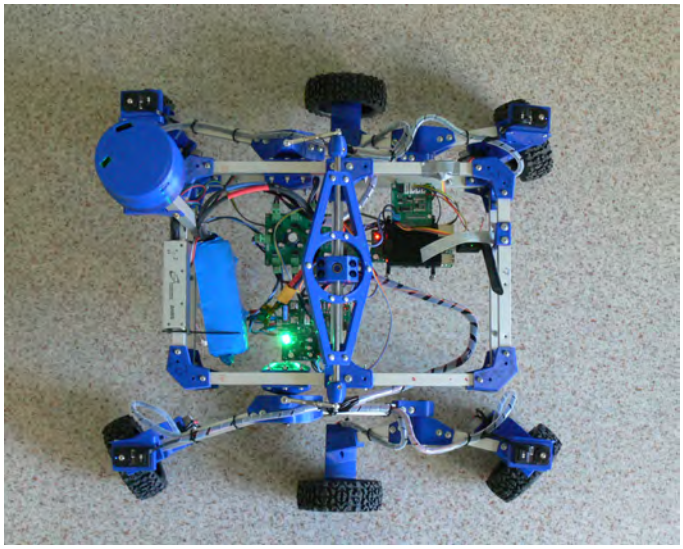
Pro komunikaci je použita stejná knihovna jako na lidarovém modulu (2.1). V hlavní smyčce opět probíhá vyhodnocení příchozích zpráv. Motor Board celkově podporuje tři příchozí zprávy:

- ***MotorBoard_Main_Controll*** - nastavení hlavní funkcionality. Aktivace servomotorů a regulace výkonu motorů
- ***MotorBoard_set_drive_status*** - nastavení výkonu motorů a natočení servomotorů
- ***MotorBoard_get_drive_status*** - požadavek na odeslání výkonu motorů a natočení servomotorů.

Díky rozmístění kol robota je možné aby se robot otočil na místě. Natáčení servomotorů z tohoto důvodu podporuje dva různé režimy. V klasickém režimu se kola na levé i pravé straně natáčí stejným směrem. Robot potom zatáčí podobným způsobem jako automobil. Ve druhém režimu se kola natočí proti sobě do tvaru sudu a robot se může otočit na místě. Fotky robota s natočenými koly v obou režimech jsou na obrázcích 5.7 a 5.8.



Obrázek 5.7: Motor Board: Natočení kol při klasickém režimu

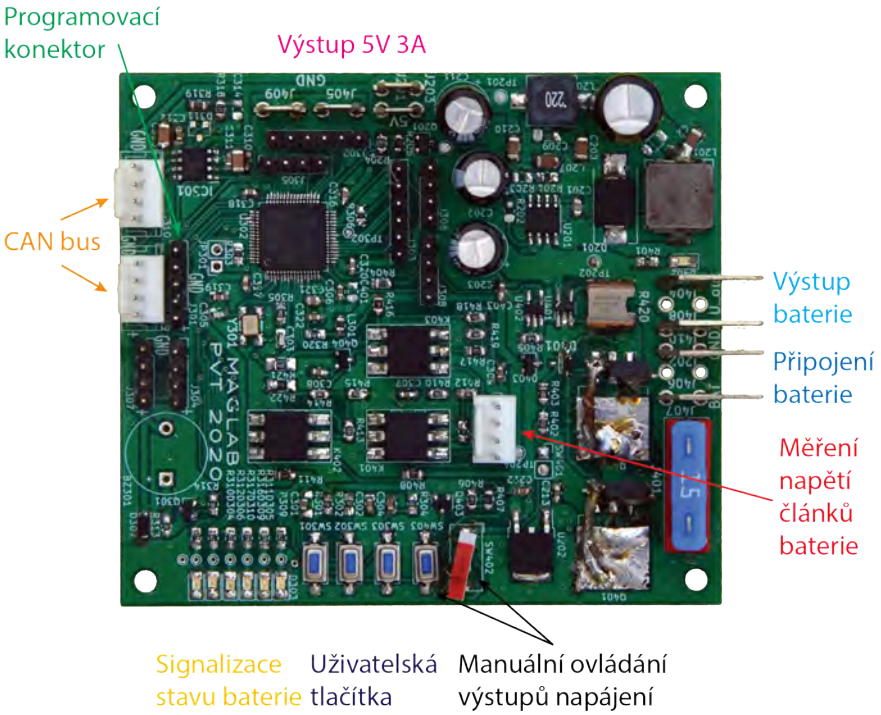


Obrázek 5.8: Motor Board: Natočení kol při režimu otočení na místě

Kapitola 6

Power Board

Další v pořadí je napájecí modul. Tato jednotka zajišťuje sledování stavu baterie a vytvoření potřebných napěťových úrovní pro všechny ostatní moduly. Návrh desky plošných spojů prováděl kolega Miroslav Tržil v rámci předmětu Projekt v týmu. Na obrázku 6.1 je zachycen výsledný modul.



Obrázek 6.1: Power Board: Výsledný modul

6.1 Hardware

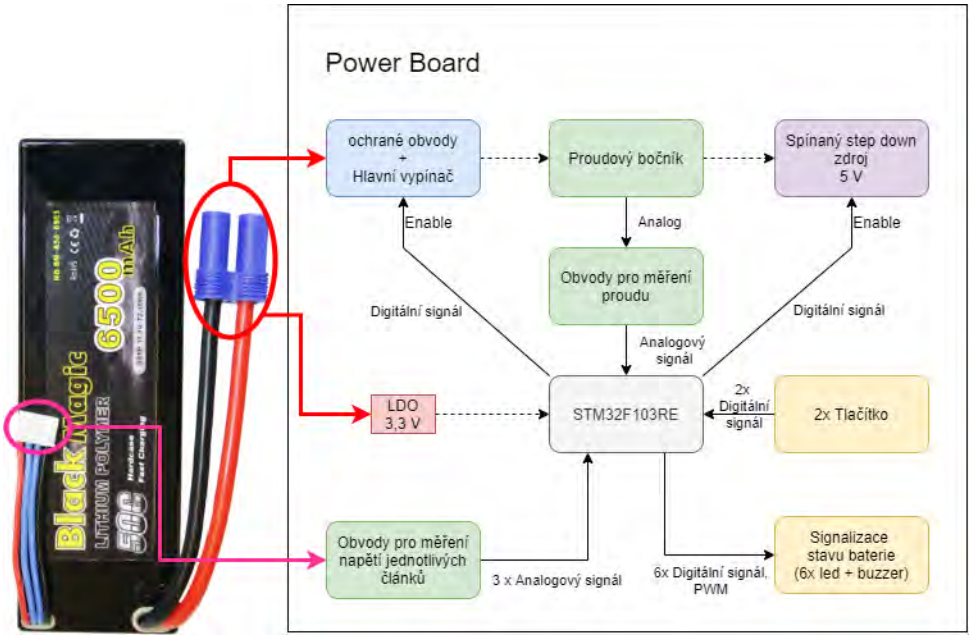
Řídící jednotkou celé desky je opět MCU STM32F103RE. Mikroprocesor je napájen pomocí lineárního regulátoru napětí 3,3 V připojeného přímo na vstupní napájení před ochranné prvky. K procesoru jsou připojeny LED diody

pro signalizaci stavu nabití baterie a buzzer pro signalizaci důležitých stavů. Dále jsou k MCU připojena 4 tlačítka (reset MCU, 2x softwarově definové tlačítko a deaktivace výstupu).

Další částí desky je spínaný zdroj napětí 5 V. Tento zdroj je schopný dodávat až 3 A. Tato hodnota se následně ukázala jako nedostatečná. Pro napájení Raspberry Pi jsem použil samostatný step-down zdroj, který jsem napojil na ovládaný výstup bateriového napětí z popisované jednotky.

Modul je vybaven proudovým bočníkem a obvody pro monitorování stavu jednotlivých článků baterie. Pro zachování životnosti baterie, je potřeba sledovat napětí jednotlivých článku a při detekci kritické hodnoty signalizovat ostatním modulům blížící se odpojení a po určitém časovém intervalu baterii odpojit. K modulu je možné připojit bezpečnostní vypínací tlačítko, které okamžitě odpojí elektroniku robota od baterie. Dále je zde umístěn hlavní vypínač napájení robota.

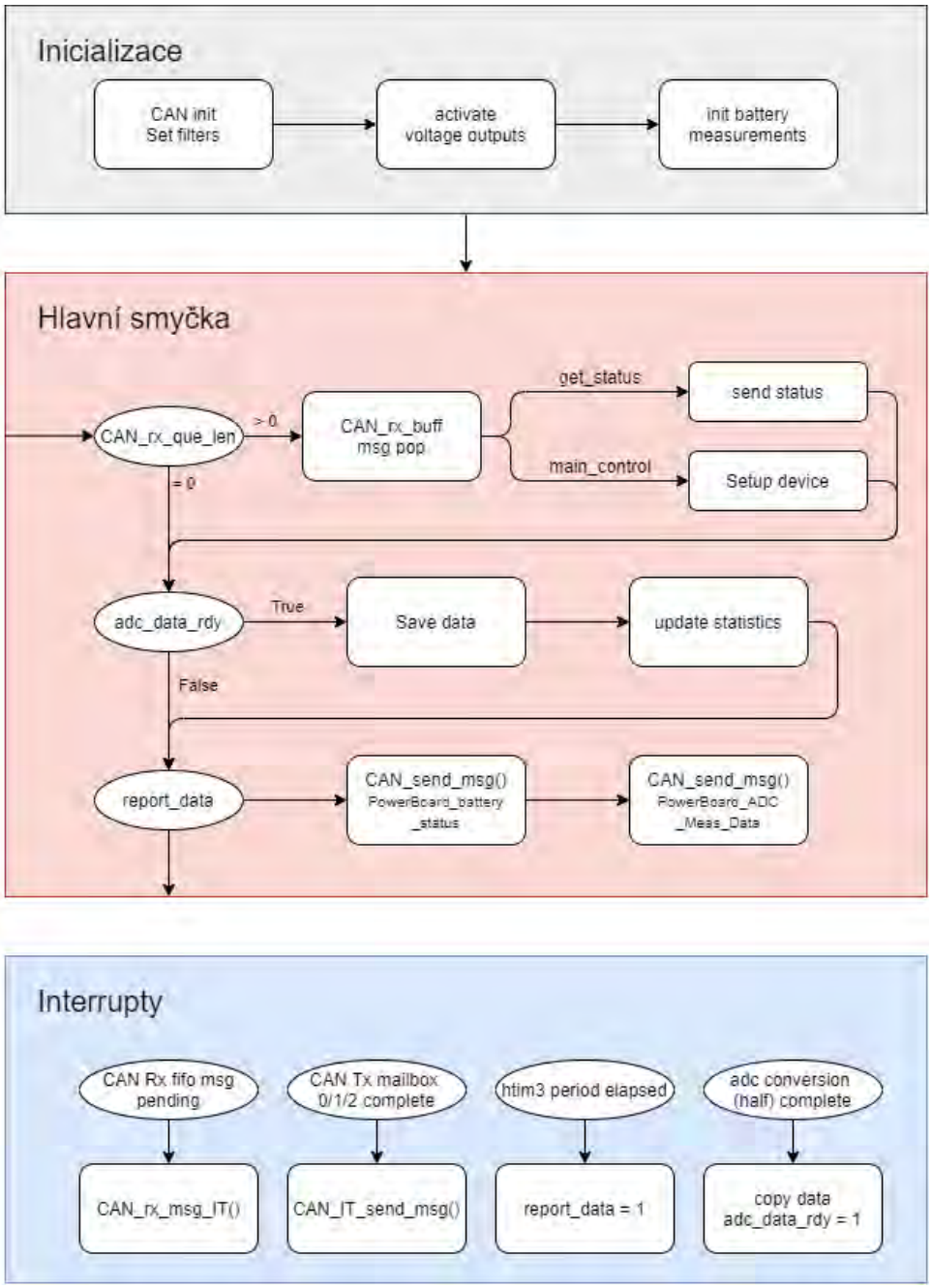
Blokový diagram zapojení hardwaru je na obrázku 6.2.



Obrázek 6.2: Power Board: Blokové schéma zapojení hardwaru

6.2 Software

Modul měří aktuální stav baterie, proudový odběr, naměřená data zpracovává a odhaduje stav nabití baterie. Data následně publikuje na sběrnici robota. Ve vytvořené aplikaci jsou použity knihovny z předchozích modulů. Procesní diagram programu je uveden na obrázku 6.3.



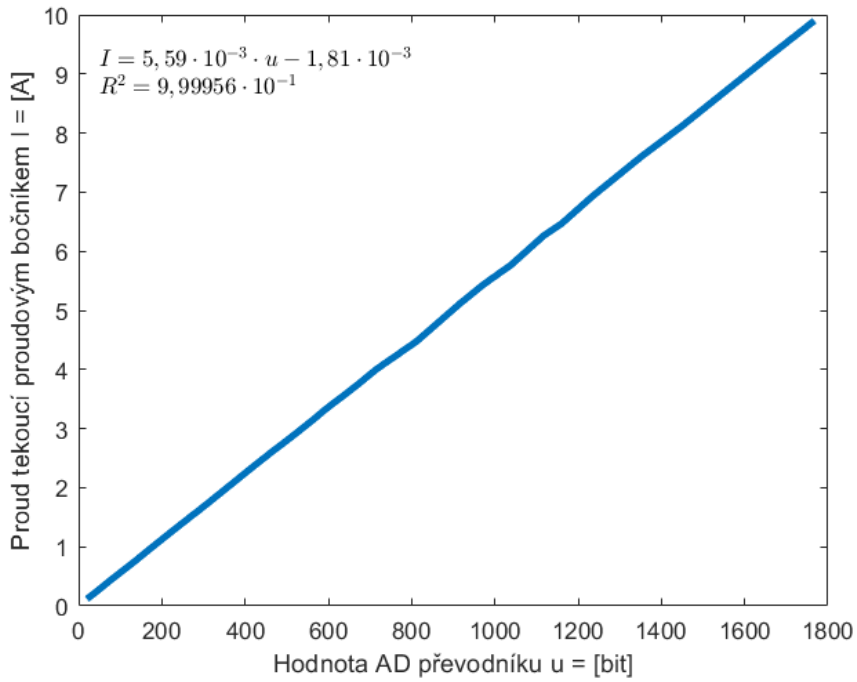
Obrázek 6.3: Power Board: Procesní diagram programu

6.2.1 AD převodník

Pomocí AD převodníku jsou měřeny výstupy signálů z obvodu proudového bočníku a sledování napětí článků baterie.

Použil jsem stejný postup nastavení převodníku jako u Motor Boardu viz. kapitola 5.2.1. Nastavení se liší jen v počtu použitých kanálů. Na tomto modulu měřím jen čtyři signály.

Při osazování obvodu pro proudový bočník kolega upravil hodnoty osazovaných pasivních součástek a nezanesl nové hodnoty do schématu. K měření osazených součástek existují tzv. in-circuit metody, ale ty vyžadují vybavení, které jsem neměl dostupné. Jednodušší bylo změřit závislost naměřených hodnot AD převodníkem na proudu tekoucího bočníkem. Pro nastavení proudu jsem použil výkonové drátové potenciometry. Připojil jsem vždy jezdec a jeden z krajních konců. Následně jsem pomocí pohybu jezdce nastavoval odpor a tím reguloval proud tekoucí proudovým bočníkem. Hodnoty proudu tekoucího obvodem jsem měřil pomocí laboratorního proudového bočníku HP 34330A voltmetrem Agilent 34401A. Výsledná převodní charakteristika i s dopočtenou závislostí je na obrázku 6.4. Naměřený proud je následně integrován za účelem zisku spotřebované kapacity baterie.



Obrázek 6.4: Power Board: Závislost proudu tekoucího proudovým bočníkem na hodnotě naměřené AD převodníkem

Pro zisk napětí jednotlivých článků jsou použity děliče napětí vůči zemi. Zjednodušené zapojení bez ochranných prvků je uvedeno na obrázku 6.5. Měřená napětí U_{sense_2} a U_{sense_3} jsou závislá i na napětí ostatních článků. Pro přesné určení napětí jednotlivých článků je potřeba tyto křížové závislosti

odstranit. Viz následující rovnice.

$$U_{sense\_1} = U_{BT1}\frac{R_2}{R_1 + R_2} \tag{6.1}$$

$$U_{sense\_2} = (U_{BT1} + U_{BT2})\frac{R_4}{R_3 + R_4} \tag{6.2}$$

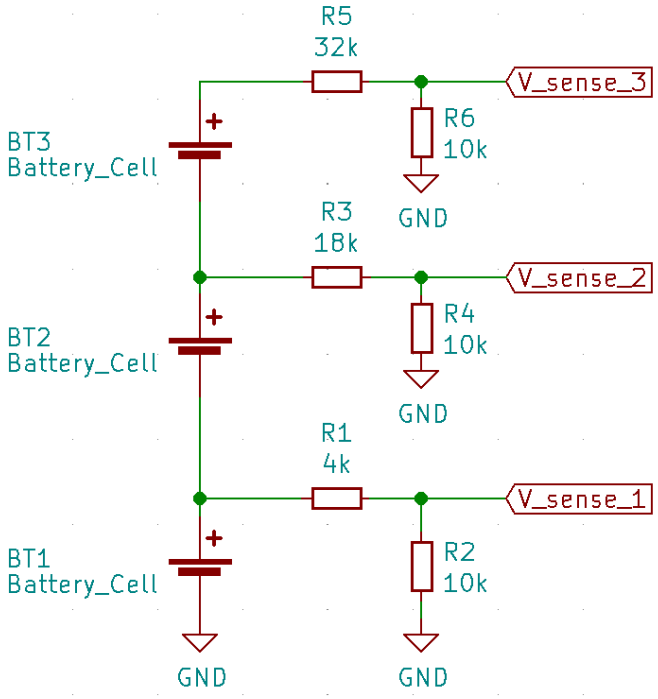
$$U_{sense\_3} = (U_{BT1} + U_{BT2} + U_{BT3})\frac{R_6}{R_5 + R_6} \tag{6.3}$$

$$U_{BT1} = U_{sense\_1}\frac{R_1 + R_2}{R_2} \tag{6.4}$$

$$U_{BT2} = U_{sense\_2}\frac{R_3 + R_4}{R_4} - U_{BT1} \tag{6.5}$$

$$U_{BT3} = U_{sense\_3}\frac{R_5 + R_6}{R_6} - U_{BT1} - U_{BT2} \tag{6.6}$$

Z rovnic lze odvodit, že měření napětí článku BT3 má třetinovou přesnost oproti článku BT1.



Obrázek 6.5: Power Board: Schéma zapojení kanálů AD převodníku pro měření napětí článků baterie.

6.2.2 Určení stavu nabití baterie

Jako hlavní ukazatel pro odhad stavu baterie jsem použil integraci odebíraného proudu. Pro použití této metody je potřeba znát nominální hodnotu kapacity

baterie. V zařízení je "hardcodováná" hodnota pro klasickou používanou baterii. Při použití baterie o jiné kapacitě je možné upravit tuto hodnotu pomocí CAN rámce.

Aby tato metoda byla funkční je potřeba si uchovat spotřebovanou kapacitu i při odpojení napájení a opětovném zapnutí zařízení. K tomu jsem zvolil periodické ukládání spotřebované kapacity do paměti Flash MCU. Výrobce MCU garantuje životnost paměti pro 10 000 zápisů. Použitý MCU se řadí do rodiny tzv High-density-devices, to jsou procesory s největší možnou velikosti Flash paměti pro danou řadu. Můj program nepoužívá ani 20% dostupné 512 kB paměti. To mi umožňuje rozprostřít ukládaná data do většího prostoru pro snížení opotřebení paměti. Data jsem rozprostřel na 3 stránky Flash paměti (dohromady 6 kB). Adresy stránek jsem stanovil na poslední 3 dostupné stránky Flash paměti, protože flash paměť je od začátku plněna zkompilovaným zdrojovým kódem vytvořené aplikace. Každou používanou stránku nejdříve označím unikátní 16 bytovou hlavičkou, kterou lze interpretovat jako textový řetězec ASCII znaků "batterrytrackmem". Jestliže na začátku stránky není nalezena tato hlavička, stránka je smazána a nainicializována (je zapsána hlavička).

Každý záznam se skládá ze 4 čísel typu *uin32_t*. Postupně to jsou *sequence_number*, *mah_drawn*, *battery_voltage*, *batterry_capacity* . Hodnota *battery_voltage* je potřebná pro detekci nabití baterie. Jestliže při zapnutí zařízení je naměřeno napětí o 0,5 V vyšší než udává poslední záznam, je detekováno nabití baterie a hodnota spotřebované kapacity je vynulována. Hodnotu spotřebované kapacity lze manuálně vynulovat pomocí tlačítka SW_302.

Záznamy do flash pamětí jsou ukládány při každých spotřebovaných 50 mAh.

Při zapnutí zařízení je otestováno, jestli jsou nainicializovány nějaké stránky v paměti FLASH. Jestliže program nenalezne nainicializované stránky, je provedena inicializace a systém použije výchozí hodnoty. Jestliže je paměť nainicializována, program vyhledá (pomocí hodnoty *sequence_number*) nejnovější záznam.

Dalším ochranným prvkem je kontrola, jestli napětí jednotlivých článků nekleslo pod minimální povolenou úroveň (3 V).

Pro signalizaci zbývající kapacity baterie jsem použil LED diody osazené na desce. Na DPS je pro tuto úlohu celkem 6 LED diod, dvě zelené, dvě oranžové a dvě červené. Některé piny MCU, na které jsou připojeny LED, nejsou konfigurovatelné jako výstupy PWM. Pro signalizaci jsem zvolil stálý svit určitého počtu LED diod, případně blikání poslední z řady rozsvícených LED. Tím mohu signalizovat celkem dvanáct úrovní zbývající kapacity baterie. Pro blikání LED jsem použil časovač s interrupty.

Buzzer na desce je použit pro přerušovanou signalizaci kapacity baterie menší než 10 % nebo napětí libovolného článku menší než 3,4 V.

6.2.3 Uživatelské rozhraní

Pro vykonávání jednotlivých úloh s jinou obnovovací frekvencí jsem použil časovače. Jeden časovač je použit pro vzorkování AD převodníku, další pro

vzorkování tlačítek a poslední pro blikání s LED.

Deska obsahuje 6 informačních LED diod a 2 programovatelná tlačítka. Tlačítka vzorkuji s frekvencí 50 Hz. Pro měření doby stisku tlačítka používám počítání vzorků, při kterém je stisknuté. Při jeho uvolnění počet vzorků vynuluji. Následně implementuji reakce na základě délce stisku tlačítka.

Tím mám ošetřen i přechodový jev tlačítka, při kterém se může rychle měnit hodnota logického vstupu.

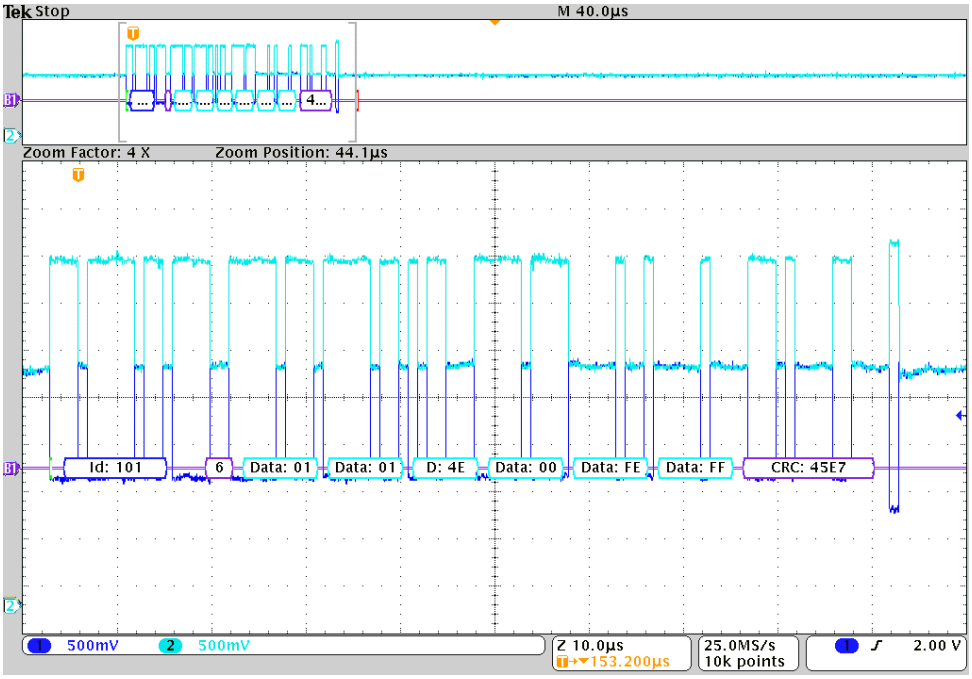
Kapitola 7

CAN bus

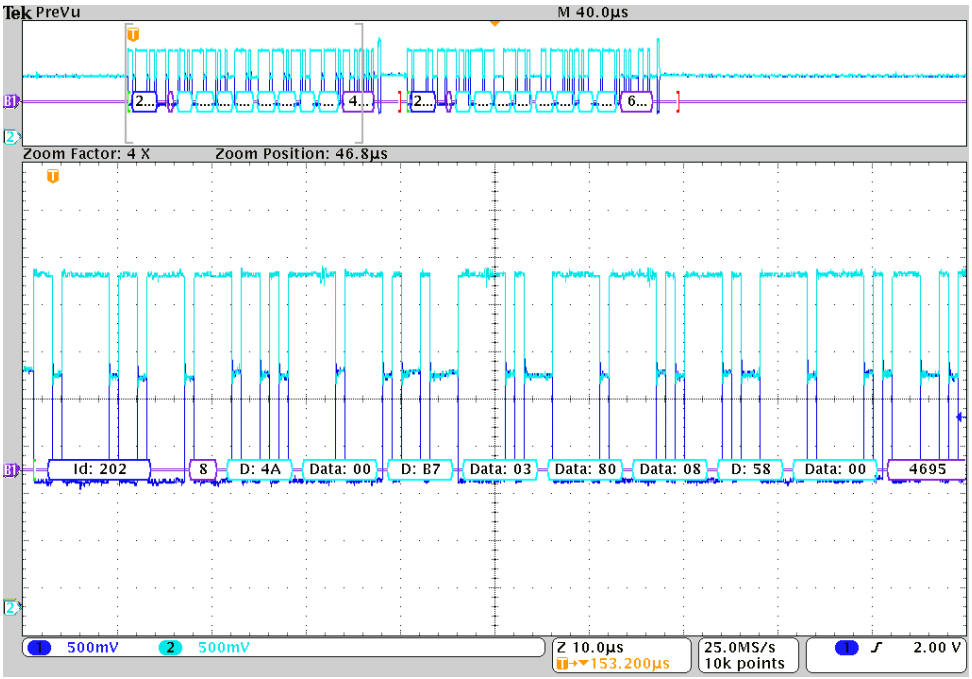
Jak již bylo zmíněno pro komunikaci mezi moduly jsem použil sběrnici CAN bus. Tato sběrnice mi usnadnila vytvoření komunikačního protokolu. Komunikační protokol této sběrnice již v sobě implementuje přesné rozlišení dat pomocí identifikátorů a prioritu jednotlivých zpráv. Sběrnice je určena pro systémy s deterministickou odezvou.

Pro rozdělení dat do jednotlivých zpráv jsem si vytvořil tabulku, která je uvedena v příloze C. Při přiřazovaní adresního prostoru ID zpráv jsem se řídil jejich prioritou. Nejvyšší prioritu (nejnižší ID rámců) jsem přiřadil Motor Boardu (*0x100 - 0x1FF*), následují rámce lidarového modulu (*0x200-0x2FF*) a jako poslední zprávy Power Boardu (*0x300-0x3FF*) .

Na obrázcích 7.1 a 7.2 jsou zachyceny CAN rámce komunikace mezi moduly pomocí osciloskopu. Na obrázku 7.1 je zachycen rámec nastavující natočení kol a výkon motorů. První dva byty nesou informaci o módu zatáčení a nastavování výkonu DC motorů. Další dvě dvojice nastavují výkon DC motorů a úhel natočení kol. Na obrázku 7.2 jsou přenášena data z měření lidaru, data jsou přenášena pomocí dvou za sebou jdoucích zpráv, každá nese informaci o poloze natočení vrchní části modulu, celkový počet poloh a měření ze dvou senzorů.



Obrázek 7.1: CAN bus: Ukázka rámce *MotorBoard_set_drive_status*



Obrázek 7.2: CAN bus: Ukázka rámce *LidarBoard_ranging_data*

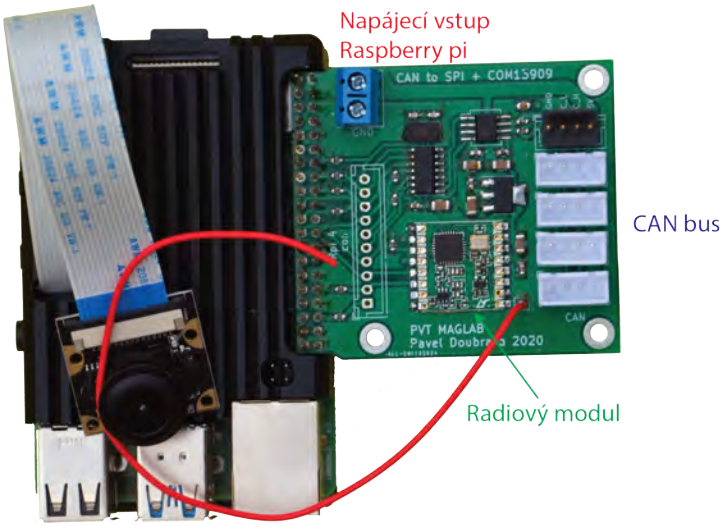
Kapitola 8

Raspberry Pi 4

Raspberry Pi 4 je řídicí jednotkou celé platformy. Na Raspberry Pi jsem nainstaloval operační systém Raspberry Pi OS s grafickým rozhraním, bez doporučených aplikací. Rapsberry pi OS je operační systém založený na distribuci Debian operačního systému Linux.

Pro vývoj řídicí aplikace jsem zvolil Robot Operating System (ROS). ROS je open-source sada sofwarových knihoven a nástrojů, které zjednodušují návrh aplikací robotů. ROS nabízí knihovny od základních driverů až po state-of-the-art algoritmy. Tyto knihovny jsou doplněny o vývojářské nástroje.

Na obrázku 8.1 je použitý počítač Raspberry Pi 4 uzavřený do pasivního chladiče s vytvořeným modulem "Communication Hat".



Obrázek 8.1: Raspberry Pi: Raspberry Pi 4 s modulem Communication hat

8.1 Instalace

Pro Raspberry Pi OS není distribuovám instalační balíček ROS. Musel jsem nainstalovat tento software ze zdrojových kódů. Zvolil jsem verzi ROS 1

Melodic, protože některé používané balíčky nejsou oficiálně vydané pro novější verzi Noetic.

Po instalaci základního souboru balíčků ROSu jsem postupně v průběhu vývoje aplikace doinstalovával další potřebné balíčky. Všechny doinstalované balíčky bylo nutné instalovat ze zdrojových kódů. Repozitáře těchto balíčku jsem přidal jako submoduly do repozitáře s řídící aplikací v Raspberry Pi a přidal je do workspace této aplikace. Po instalaci ROS stačí naklonovat tento repozitář, spustit sestavení workspace a všechny potřebné baličky se sestaví společně s vyvíjenou aplikací.

Proces instalace je popsán v souboru README.md v gitlab repozitáři softwaru Raspberry pi *rover_rpi_sw*, který je dostpuný na adrese `https://gitlab.fel.cvut.cz/doubrpa1/rover_rpi_sw`

8.2 Communication hat

Na trhu existuje mnoho rozšiřujících modulů pro Raspberry Pi (tzv. hat), Tyto moduly nabízí velmi různorodé funkcionality od příjmu televizního signálu přes vlastní LoRa gateway, moduly vhodné pro chytrou domácnost, pokročilé zvukové karty pro domácí kino, až po ovládání motorů.

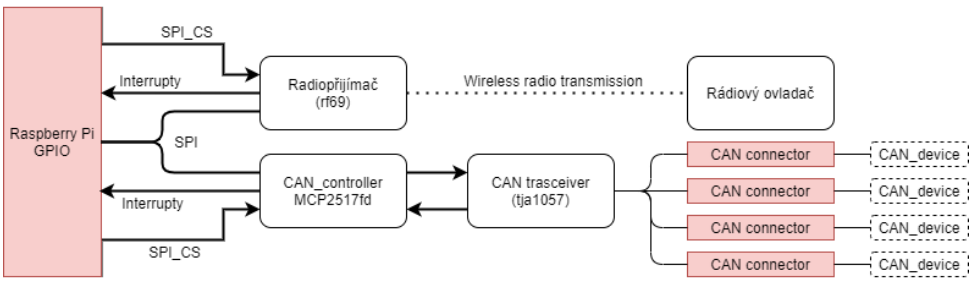
Pro naši aplikaci jsme potřebovali připojit radiový přijímač, rozšířit funkcionalitu GPIO pinů o periferii CAN bus a přidat svorkovnici pro napájení.

GPIO piny Raspberry Pi 4 v základu nepodporují sběrnici CAN bus. Tuto část jsem vyřešil pomocí externího CAN kontroleru MCP2517fd. Obvod tvoří rozhraní mezi sběrnicí CAN bus a SPI.

Vedoucí projektu požadoval robustní řešení řízení robota pro prostředí, ve kterém nebude možné použít hlavního webového rozhraní (nespolehlivá WiFi). Pro tuto funkcionalitu byl využit návrh ovladače pro předchozí verzi roveru. Tento ovladač používá pro přenos dat integrovaný obvod RF69. S obvodem lze komunikovat pomocí sběrnice SPI.

Vytvořil jsem vlastní "hat"pro Raspberry Pi, který umožňuje komunikaci po sběrnici CAN bus (MCP2517FD), komunikaci s radiovým ovladačem (RF69) a externí napájení Raspberry Pi ze svorkovnice na modulu.

Diagram zapojení je uveden na obrázku 8.2.



Obrázek 8.2: Raspberry Pi: Diagram zapojení hardwaru Communication hat.

Na platformě robota je dostatek místa pro elektroniku. Použil jsem tedy netradiční vysunutí DPS mimo tělo Raspberry Pi pro lepší cirkulaci vzduchu.

Modul je zachycen v rámci fotografie Raspberry Pi na obrázku 8.1. Schéma zapojení je uvedeno v příloze B.2.

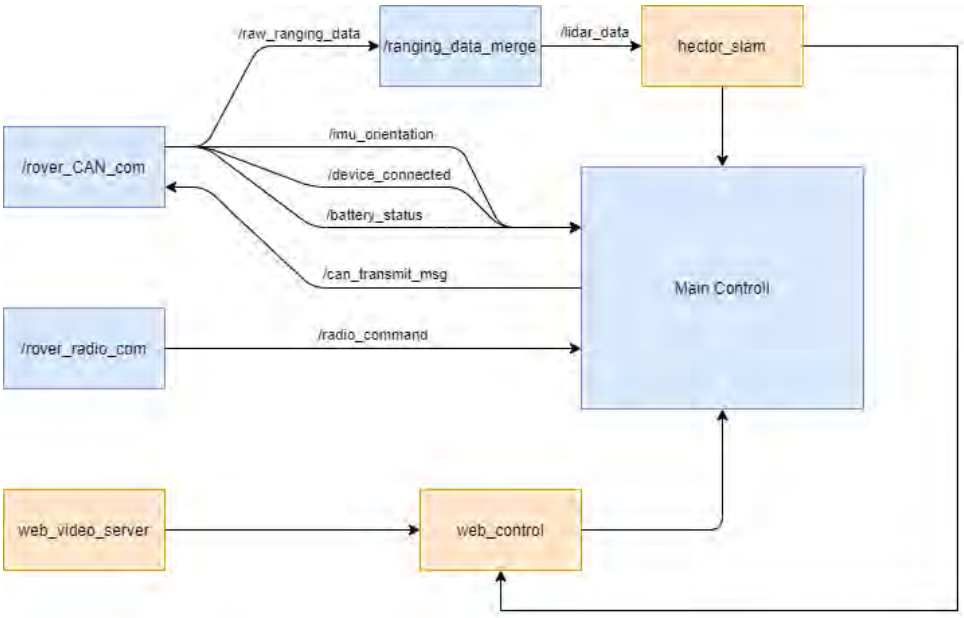
8.3 ROS

Pro návrh řídicí aplikace byl zvolen ROS. Program vyvíjený za pomocí ROSu se dělí do uzlů, kdy každý uzel vykonává specifický úkol. Uzly mezi sebou komunikují prostřednictvím zpráv a "služeb"(service). Zprávy jsou jednosměrná komunikace, kdy jeden uzel publikuje a ostatní mohou přijímat. Naopak služba je obousměrná komunikace, uzel vyšle požadavek a přidělený uzel mu pošle odpověď. Může se například jednat o nějaké zpracování dat.

Zprávy a služby jsou rozlišovány pomocí témat (topic).

8.3.1 Struktura uzlů

Před vývojem aplikace jsem si nejdříve vytvořil návrh struktury aplikace a její dataflow diagram. Popsal jsem si jednotlivé témata zpráv a datové typy zpráv. Výsledný návrh dataflow diagramu je na obrázku 8.3. Uzly, které jsou na obrázku vyznačeny modře, jsem implementoval já. Uzly vyznačené žlutě byly staženy v rámci volitelných knihoven pro ROS.



Obrázek 8.3: Raspberry Pi: Dataflow diagram

V následujících kapitolách postupně popíši jednotlivé uzly.

8.3.2 Rover_radio_com

Jako první jsem vytvořil uzel pro příjem dat z rádiového ovladače. Přijímané zprávy jsou v uzlu publikovány na téma */radio_command*. Uzel je napsán v

jazyce C++.

Pro jednoduchou přenositelnost kódu na další platformy jsem využil dědičnosti. Nejdříve jsem nadefinoval třídu *SPI_Template*. Tato třída jen určuje všechny používané metody pro ovládání periferie SPI programem. Každý kdo chce použít můj kód pro jinou platformu, musí jen vytvořit vlastního potomka třídy *SPI_Template* a předefinovat původní funkce.

Třída definuje metody:

- ***init()***
- ***close_dev()***
- ***prepare_transmition()***
- ***read_write()***
- ***read()***
- ***write()***

Následně jsem vytvořil třídu *SPI_Rpi*, která je potomkem třídy *SPI_Template*. Třída *SPI_Rpi* implementuje práci s periferií SPI Raspberry Pi 4.

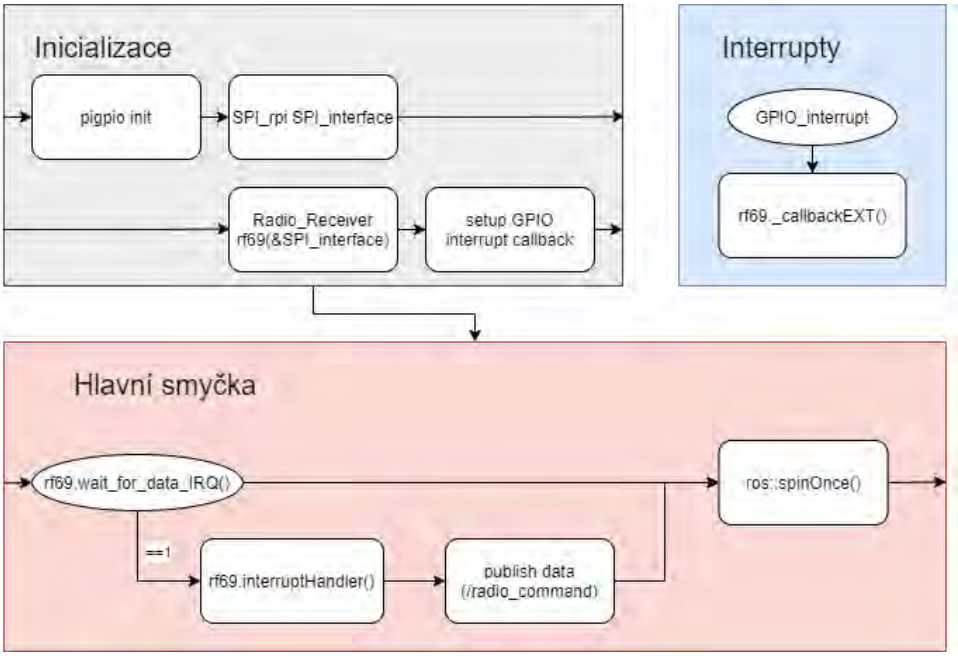
Při prvotní rešerši softwarových nástrojů pro Raspberry Pi 4 v roce 2019 jsem nenašel žádnou dokončenou knihovnu pro ovládání GPIO pinů Raspberry. Všechny byly ve vývoji. Z tohoto důvodu jsem se pro ovládání periferie SPI rozhodl použít knihovnu zabudovanou přímo v operačním systému *<linux/spi/spidev.h>*.

Kód pro práci s přijímačem (MCP2517fd) byl již vytvořen pro HAL knihovny rodiny MCU STM32. Pro použití v mém programu bylo nutné upravit funkce pracující s hardwarem. Kód napsaný pro mikrokontroler jsem přepracoval do třídy *Radio_receiver*. Třída ve svém konstruktoru přijímá jako parametr instanci třídy *SPI_Template*, tedy můžeme jí předat místo této třídy libovolného potomka, v našem případě třídu *SPI_Rpi*.

Pro otestování funkčnosti přepsaného kódu jsem nejdříve použil opakované dotazování, jestli jsou data připravena. Po úspěšném ověření jsem začal hledat možnosti, jak na Raspberry Pi nastavit GPIO interrupty.

Zjistil jsem, že knihovna *pigpio* již podporuje Raspberry Pi 4 (při původní rešerši byla ve vývoji). Původní kód pro práci s SPI jsem již nepřepisoval, zmíněnou knihovnu jsem použil na práci s interrupty. Výsledná třída implementuje následující veřejné metody:

- ***init()***
- ***receiveBegin()***
- ***interruptHandler()***
- ***wait_for_data_IRQ()***
- ***__callbackExt()***



Obrázek 8.4: Raspberry Pi : Procesní diagram uzlu Rover_radio_com

Na obrázku 8.4 je procesní diagram uzlu.

V hlavním vlákně uzlu nejdříve proběhne inicializace knihovny pigpio, a třídy *SPI_Rpi*. Metoda *_callbackExt()* je předána knihovně pigpio jako interrupt handler. Tento interrupt handler používá mutex a podmíněnou proměnnou (conditional variable) pro signalizaci interruptu hlavnímu vláknu uzlu.

V nekonečné smyčce je následně použita funkce *wait_for_data_IRQ()*. Která čeká na signál podmíněné proměnné z *_callbackExt()*. Po obdržení signálu je z IC vyčtena přijatá zpráva. Zpráva je interpretována a publikována na příslušný topic (*/radio_command*). Pro publikaci dat jsem nadefinoval vlastní strukturu zprávy, protože publikovaná data neodpovídají žádné předefinované zprávě.

8.3.3 Rover_CAN_com

Tento uzel je zaměřen na komunikaci po sběrnici CAN bus. Zpracovává příchozí rámce a interpretuje je do zpráv. Uzel odebírá zprávy z tématu */CAN_transmit_msg* a posílá je na sběrnici CAN.

Komunikace s MCP2517fd

Při zprovozňování komunikace mezi Raspberry Pi a MCP2517fd jsem nejdříve chtěl použít knihovnu publikovanou výrobcem obvodu a ukázkových kódů jak ji používat. Nejdříve jsem implementoval vrstvu ovládající SPI (za pomocí třídy *SPI_Rpi*). V ukázkových kódech je několik ladících funkcí, které

umožňují otestovat zápis a čtení do jednotlivých částí paměti zařízení. Nebyl jsem schopen docílit zápisu do zařízení.

Po několika dnech čtení datasheetu a testování různých úprav kódu jsem našel jiné řešení připojení integrovaného obvodu využívající tzv. DT overlays. Jádro Linuxu Raspberry Pi používá tzv. Device tree (DT) pro popsání přítomného hardwaru. Zařízení lze dále konfigurovat pomocí tzv. DT parametrů.

Jestliže používané zařízení je v základním balíčku DT overlays, kterým Raspberry Pi OS disponuje, přidání dalšího zařízení probíhá pomocí úpravy souboru */boot/config.txt*. Do souboru je potřeba přidat řádky definující další používaný DT overlay. Pro aktivaci použitého CAN kontroleru jsem přidal následující řádek, který jádru linuxu popisuje, na jaké piny je modul připojen a jaký používá oscilátor.

```
1 dtoverlay=mcp251xfd,spi0-0,interrupt=18,oscillator
    =40000000
```

Zmíněnou metodu využívá například CAN hat od firmy Seedstudio. Ověření připojení správné inicializace připojeného hardwaru pro restartování Raspberry Pi jsem provedl pomocí příkazu *dmesg*.

```
1 pi@sawppyrover:~ $ dmesg | grep spi
2 [6.417652] spi_master spi0: will run message pump with
    realtime priority
3 [6.433356] mcp251xfd spi0.0 can0: MCP2517FD rev0.0 (-
    RX_INT +MAB_NO_WARN +CRC_REG +CRC_RX +CRC_TX +ECC -HD
    c:40.00MHz m:20.00MHz r:17.00MHz e:16.66MHz)
    successfully initialized.
```

Zařízení se připojilo jako další síť. Ověřit připojení a zjistit její název můžeme pomocí příkazu *ifconfig -a*. Následuje zkoumaná část výpisu po provedení příkazu.

```
1 can0: flags=193<UP,RUNNING,NOARP>  mtu 16
2     unspec
          00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          txqueuelen 10  (UNSPEC)
3     RX packets 26848  bytes 180310 (176.0 KiB)
4     RX errors 0  dropped 0  overruns 0  frame 0
5     TX packets 4861  bytes 29166 (28.4 KiB)
6     TX errors 0  dropped 0 overruns 0  carrier 0
          collisions 0
7     device interrupt 66
```

Po každém restartu zařízení je potřeba aktivovat periferii a nastavit parametry komunikace (např. rychlost přenosu) pomocí příkazu *ip link set <name> up type can <parameters>*

Pro prvotní otestování správného přenosu dat jsem použil balíček *can-utils* implementující debugovací programy *cangen* a *candump* a převodník "USB to CAN Analyzer" od firmy Seeedstudio. Tento levný převodník umožňuje mapovat pohyb po sběrnici i posílat data.

Použití přístupu pomocí DT overlays přineslo mnoho výhod oproti původnímu návrhu. Implementace DT overlay umožňuje více aplikacím používat stejné zařízení CAN. Princip funkce sběrnice CAN je rozšířen i mezi tyto aplikace, tedy každá aplikace obdrží všechny zprávy přijaté pomocí hardwaru MCP2517fd, ale i zprávy posílané jinou aplikací na Raspberry Pi.

Další výhodou je možnost vytvoření virtuální sběrnice CAN. Při vývoji jsem nemusel mít připojeny ostatní moduly a testované zprávy jsem si mohl poslat pomocí terminálu nebo testovací aplikace.

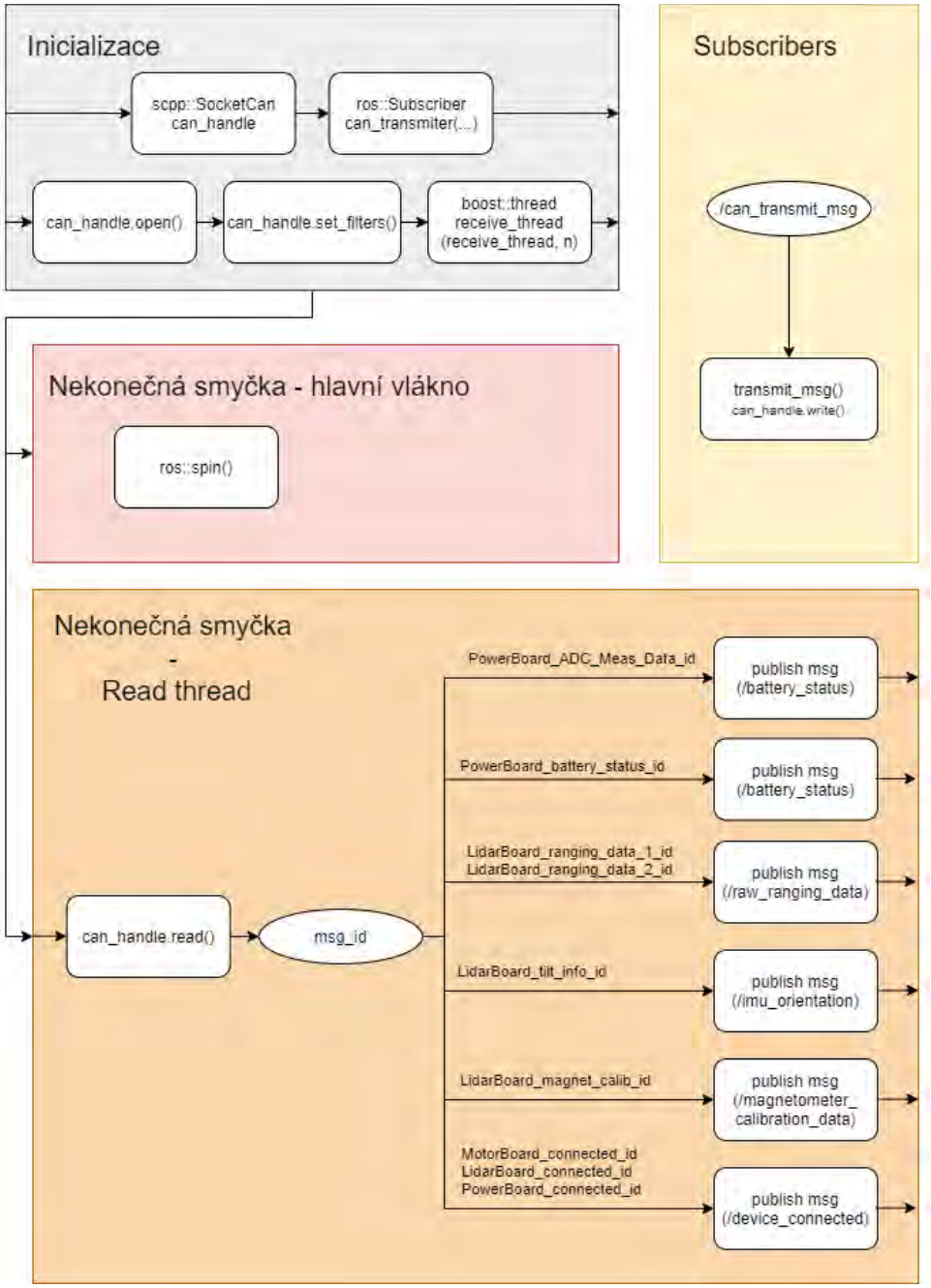
Pro použití zařízení v programu napsaného v jazyce C jsem použil knihovny linux, které jsem uzavřel do vlastní třídy *socketCAN*. Třída implementuje následující veřejné metody:

- ***open()***
- ***set_filters()***
- ***write()***
- ***read()***
- ***close()***

Implementace uzlu

Uzel Rover_CAN_com používá dvě vlákna jedno na příjem zpráv a druhé na odesílání. Pro čtení zpráv jsem použil blokující přístup ve vlastním vlákně. V hlavním vlákně probíhá *ros::spin()*. V uzlu implementuji subscribera přeposílajícího zprávy z tématu */can_transmit_msg* na CAN.

Na obrázku 8.5 je procesní diagram uzlu.



Obrázek 8.5: Raspberry Pi : Procesní diagram uzlu Rover_CAN_com

8.3.4 Ranging_data_merge

Uzel přijímá zprávy z tématu */raw_ranging_data* a vysílá zprávy na téma */lidar_data*. Uzel je implementován v jazyce python.

Data měření vzdálenosti z lidarového modulu přicházejí po dvojicích. Tento uzel naměřená data spojuje do standardizovaných zpráv typu *sensor_msgs LaserScan* poskytující měření z celého rovinného úhlu.

8.3.5 Hector_slam

Při výběru knihovny pro uzel zajišťující slam (simultaneous localisation and mapping - simultánní lokalizace a mapování) jsem vybíral na základě článku [21]. Článek porovnává tři knihovny: Gmapping, Google cartographer a Hector SLAM. První dvě pro prvotní určení polohy používají odometrii. Poslední metoda používá jen data z lidaru. DC motory platformy neumožňují měřit délku ujeté vzdálenosti (nemají enkodéry). I přes výrazně horší výsledky knihovny Hector SLAM ji musíme použít.

Pro nastavení spouštěného uzlu hector_slam jsem nastavil parametry popsané v dokumentaci.

Přesný popis algoritmu je uveden v článku [22].

Zpřesnění algoritmu můžeme dosáhnout předáním informace o natočení robota v prostoru. Pro použití dat o natočení poskytované AHRS algoritmem jsem nejdříve musel nadefinovat transformace mezi jednotlivými souřadnicovými systémy. (viz. dokumentace [23].)

Tato informace funguje jako částečná odometrie, chybí zde odhad informace o uražené vzdálenosti.

8.3.6 Web_control

Tento uzel je implementovaný v rámci knihovny *rosbridge_suite*. Uzel rozšiřuje funkcionalitu ROSu o JSON rozhraní. Pomocí tohoto uzlu je možné připojit k mému programu webové rozhraní vyvíjené kolegou Maximem Scherbanem.

Toto webové rozhraní umožňuje zobrazit stream videa z webkamery Raspberry Pi , naměřená data z lidaru a interaktivní radiový ovladač, který může pracovat ve dvou módech.

Při zvolení módu ovládání robota *Ovladač* je na stránce vykreslen stav radiového ovladače. Při zvolení módu *Webové rozhraní* se stane ovladač interaktivním a můžeme robota ovládat stejně jako přes radiový ovladač.

8.3.7 Web_video_server

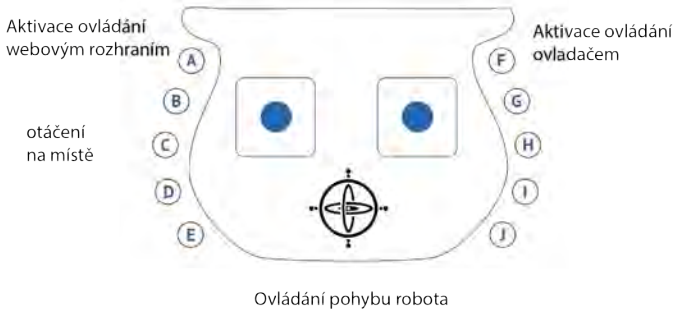
Tento uzel je implementovaný v rámci knihovny *web_video_server*. Uzel zajišťuje přenos videa pro webové rozhraní.

8.3.8 Main_control

Je hlavní ovládací uzel, který vyhodnocuje zprávy z ostatních uzlů a podle toho řídí robota. Jednou z hlavních povinností tohoto uzlu je řídit pohyb

robota. Robota je možné řídit ze dvou zdrojů: radiového ovladače a webového rozhraní. Oba dva zdroje používají unifikovanou podobu zpráv. K přepínání mezi řídicími vstupy je použit tzv. rosparam */rover_control_source*, který může nabývat hodnot textových řetězců "remote_control"a "radio_control".

V průběhu času jsem doplňoval k tlačítkům ovladače jednotlivé funkcionality. Výsledné přiřazení funkcionalit k ovládacím prvkům je uvedeno na obrázku



Obrázek 8.6: Raspberry Pi : Přiřazení funkcionalit robota k ovládacím prvkům ovladače

Kapitola 9

Stavba robotické platformy SAWPPY

Robota jsem začal stavět přes letní prázdniny po ukončení druhého semestru magisterského studia.

9.1 Stavba konstrukce robota

Nejdříve jsem na 3D tiskárně vytisknul všechny plastové díly konstrukce. Tisk dílu robota trval přibližně 94 hodin čistého času. Po vytištění všech dílů hlavní konstrukce, jsem začal pracovat na jejím sestavení. Nejdříve jsem zpracoval zakoupené hliníkové profily. Ty bylo nutné naformátovat na správnou délku a udělat do nich montážní otvory na šroubky.

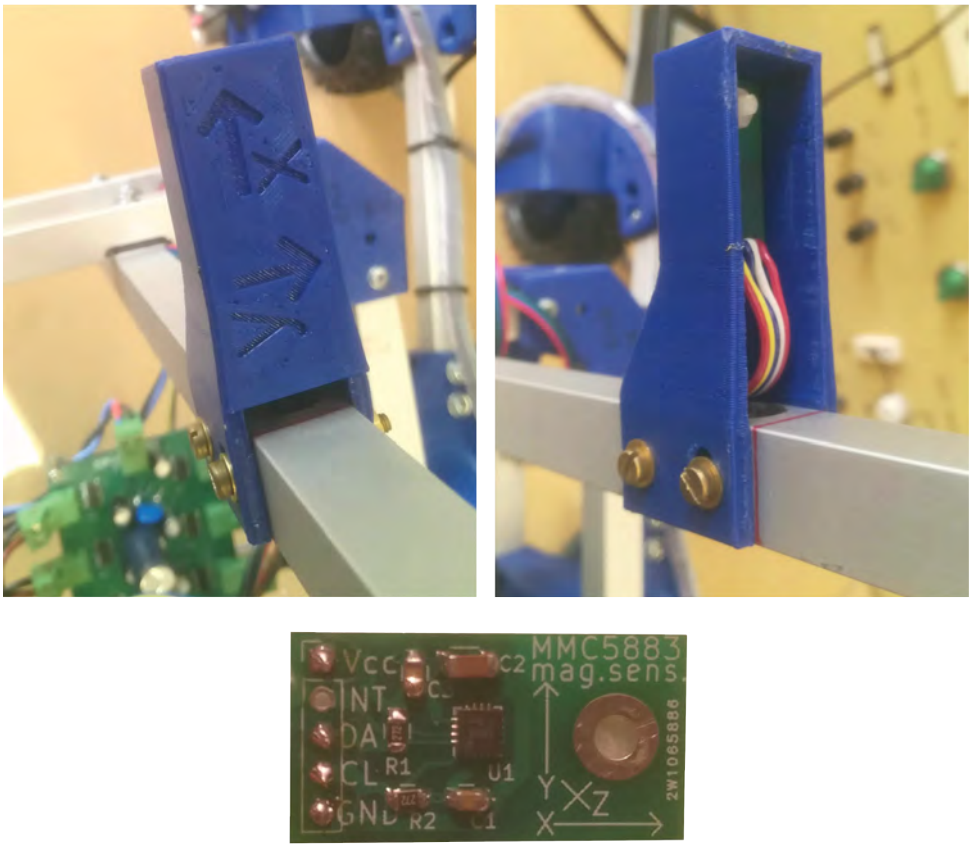
Původní návrh platformy používá profily tvaru X od firmy Misumy. V České Republice jsem nenašel profily tohoto tvaru o požadované hraně 15 mm, použil jsem tedy klasické čtvercové profily. Tyto profily jsou levnější, ale stále dostatečně pevné.

Po sešroubování konstrukce robota následovalo připevnění elektroniky.

9.2 Připevnění a zapojení elektroniky

Lidarový modul jsem vysunul asi 30 cm nad centrální část platformy kvůli zamýšlenému robotickému ramenu, které ve složeném stavu zasahuje do výšky 20 cm nad centrální část. Pro umístění Lidarového modulu nad platformu jsem použil hliníkový profil připevněný k zadním profilům centrální části platformy. Pro připevnění modulu k profilu jsem vymodeloval a vytisknul plastový díl. Do dílu jsem zabudoval kompenzaci pasivního naklonění centrální části robota ve směru jízdy (viz obrázek 9.1). Všechny dráty lidarového modulu, jsem protáhl profilem držícím senzor. Kabeláž vedoucí od spodní desky plošných spojů jsem doplnil o prodloužení programovacího konektoru, kvůli zjednodušení přehrávání firmwaru spodní DPS. (Bylo by potřeba pokaždé rozšroubovat senzor.) Doprostřed profilu držícího senzor jsem pomocí vytištěného plastového dílu připevnil magnetometr (viz. obrázek 9.2). Tento díl disponuje stejným mechanismem nastavení sklonu pro minimalizaci úhlu mezi měřícími osami inerciální jednotky a magnetometru.

Obrázek 9.1: Stavba robota: Detail uchycení Lidaru.



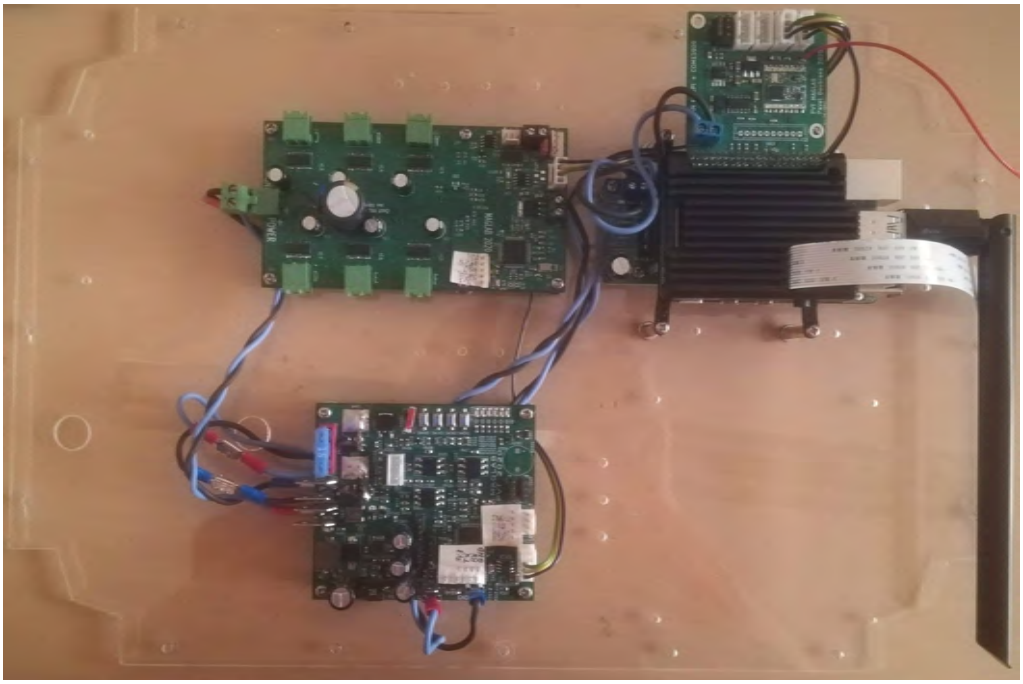
Obrázek 9.2: Stavba robota: Uchycení magnetometru.

Ostatní moduly elektroniky robota jsem připevnil na plexisklo přišroubované ke spodnímu rámu centrální části platformy. Desky jsou připevněny pomocí distančních sloupků. Pro Rasberry Pi jsem použil výrazně vyšší sloupky kvůli bezproblémovému přístupu ke konektorům (všechny konektory jsou umístěny po obvodu modulu). Power Board a Motor Board jsem umístil hned vedle baterie pro minimalizaci délek propojovacích napájecích vodičů a s tím spojenými ztrátami.

Použité napájecí vodiče mají modrou a černou barvu, pro jejich jednoznačné označení (5 /12 V a GND) jsem použil krimpovací konektory s modrým a červeným límcem.

Zjistil jsem, že Communication hat má u CAN bus konektoru prohozené vodiče GND a 5V. Pro eliminaci špatného zapojení jsem použil rozdílné nekompatibilní konektory (Molex a JST-XH).

Detail upevnění elektroniky na plexisklový podklad centrální části robotické platformy je na obrázku 9.3.



Obrázek 9.3: Stavba robota: Upevnění elektroniky na plexisklový podklad centrální části.

Kapitola 10
Testování

Po dokončení připevňování elektroniky započala fáze testování. Kontroloval jsem pohybové vlastnosti robota a na základě provedených testů upravoval řídicí algoritmy motorů a servomotorů. Kontroloval jsem teploty jednotlivých integrovaných obvodů. Dále jsem testoval vytvořenou lidarovou jednotku.

10.1 Lidar

U lidarové jednotky jsem prováděl testování lidarových senzorů a jejich schopnosti mapovat okolní prostředí. Dále jsem porovnával hodnoty poskytované AHRS algoritmem s referenčním přesným senzorem Innalabs AHRS M3.

10.1.1 Vzdálenost

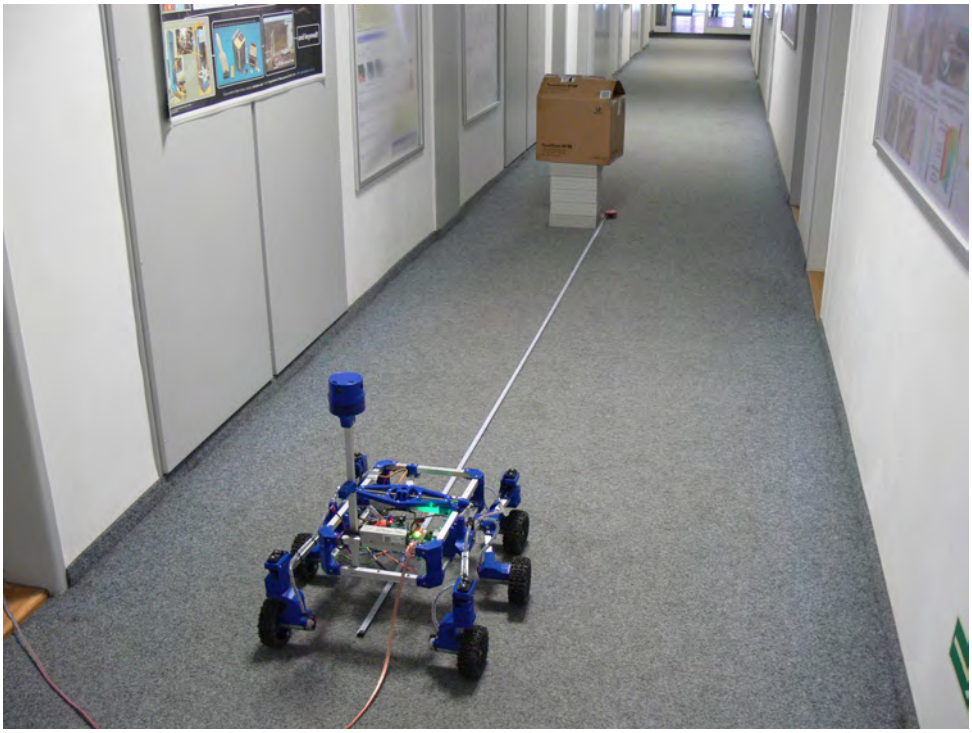
Nejdříve jsem provedl měření závislosti vzdálenosti naměřené senzorem VL53L1x na vzdálenosti překážky před senzorem. Deaktivoval jsem otáčecí mechanismus lidaru, jeden senzor jsem natočil směrem na měřící soustavu. Od paty senzoru jsem natáhl metr a postupně jsem po 20 cm oddaloval překážku od robota až do vzdálenosti 4 m. Fotografie měřící soustavy jsou na obrázcích 10.1 a 10.2. Pro záznam naměřených dat jsem vytvořil další uzel do aplikace běžící v Raspberry Pi, který po zmáčknutí tlačítka zaznamenal 50 následujících měření vzdálenosti jednotlivých senzorů a uložil průměrnou, minimální a maximální hodnotu. Porovnání naměřených vzdáleností senzoru (s jejich rozptylem) vůči reálným vzdálenostem překážky jsou v tabulce 10.1 a grafu na obrázku 10.3.

Z grafu lze vypozorovat, že senzor přestává při použité době odběru jednoho vzorku (33 ms) fungovat přibližně na dvou metrech. V lineární oblasti lze pozorovat zvýšení rozptylu měřené vzdálenosti se zvyšující se měřenou vzdáleností.

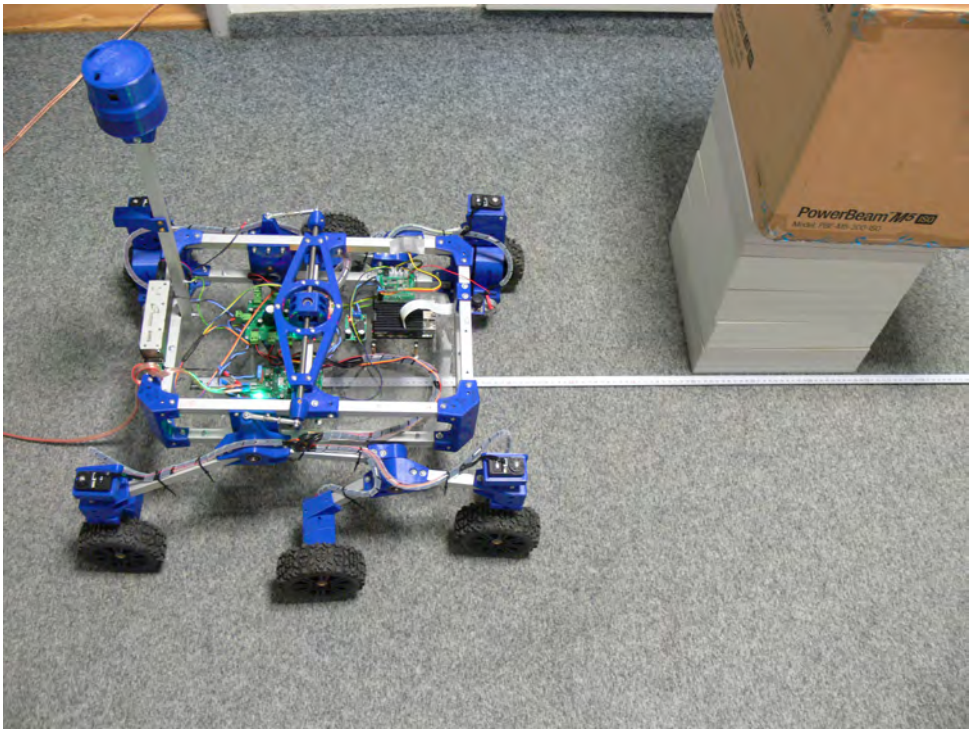
Po proložení lineární části (do 2 m) naměřených dat lineární křivkou mi vyšel následující trend.

$$l_{meas} = p \cdot l_{real} + q = 1,26 \cdot l_{real} - 0,20, \tag{10.1}$$

kde p je zisk a q offset. Na základě tohoto zjištění jsem doimplementoval tyto kalibrační konstanty do firmwaru lidarového modulu.



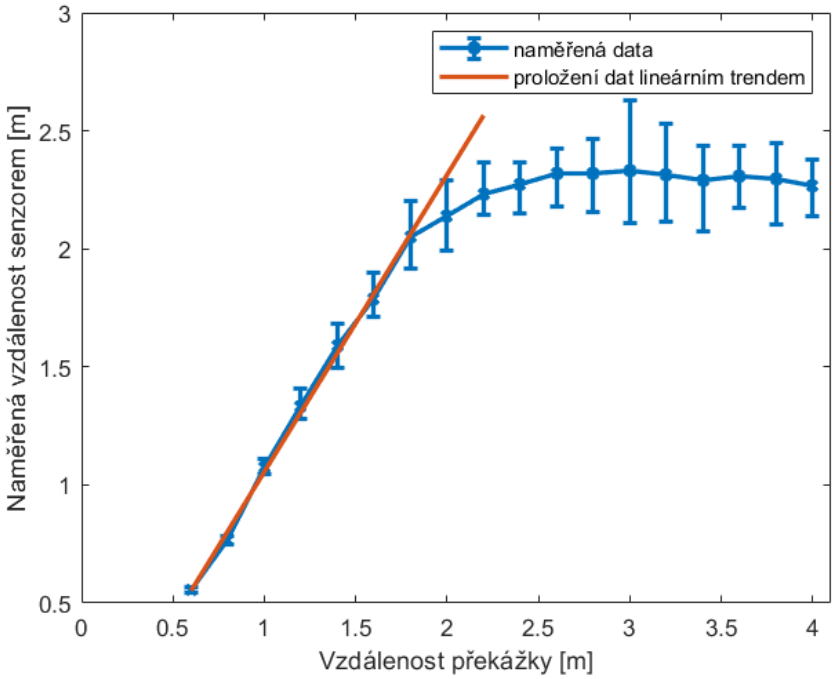
Obrázek 10.1: Testování: Soustava pro měření závislosti měřené vzdálenosti senzorem na skutečné vzdálenosti překážky - vzdálenost 4 m.



Obrázek 10.2: Testování: Soustava pro měření závislosti měřené vzdálenosti senzorem na skutečné vzdálenosti překážky - vzdálenost 0,8 m.

Vzdálenost překážky [m]	Měření Lidarem [m]		
	Průměr	Min	Max
0,6	0,55460	0,54100	0,56600
0,8	0,76434	0,74800	0,78300
1,0	1,07774	1,04700	1,11000
1,2	1,33024	1,28000	1,40900
1,4	1,58656	1,49800	1,68500
1,6	1,78858	1,71000	1,89900
1,8	2,05072	1,91500	2,20100
2,0	2,13870	1,99500	2,28900
2,2	2,22964	2,14500	2,36900
2,4	2,27086	2,15300	2,36400
2,6	2,32050	2,17900	2,42300
2,8	2,31864	2,15700	2,46400
3,0	2,33260	2,10800	2,62900
3,2	2,31542	2,11700	2,53100
3,4	2,28858	2,07200	2,43500
3,6	2,30648	2,17200	2,43500
3,8	2,29384	2,10500	2,44600
4,0	2,26424	2,13600	2,37600

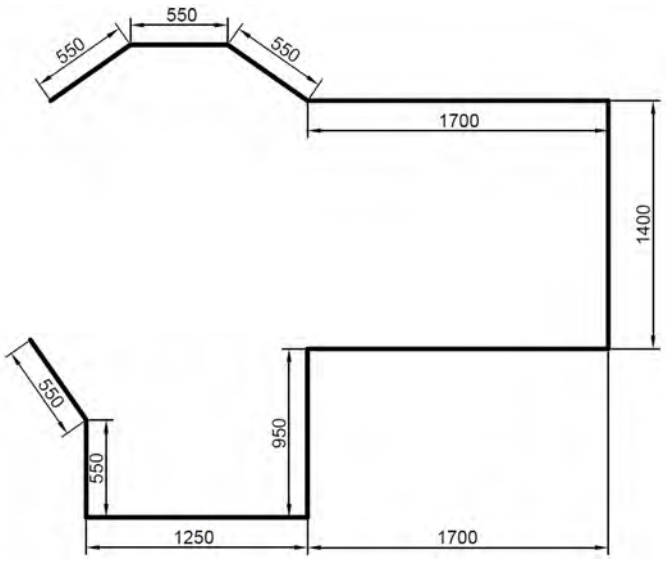
Tabulka 10.1: Testování: Naměřená data ze statického testu lidaru.



Obrázek 10.3: Testování: Graf závislosti naměřené hodnoty lidarem na reálné vzdálenosti.

Jako další jsem testoval schopnost robota mapovat okolní prostor. Přesunul jsem robota do prázdné místnosti a pomocí rozmístění lavic položených na boční stranu jsem měnil půdorys detekovaný robotem. Celkově jsem vytvořil dvě konfigurace. Pro demonstraci výsledků jsem použil data z druhého měření. Náčrt rozmístění lavic je na obrázku 10.4. Čtyři nasnímání prostoru lidarem (dále jen snímky) vykreslené do prostoru pomocí aplikace v Raspberry Pi jsou zachyceny na obrázku 10.5, a pozice při jednotlivých měřeních je zachycena na obrázku 10.6. Hrana jedné buňky mřížky na vizualizaci snímků je 0,5 m.

Jestliže se robot nepohybuje, provedené měření vzdáleností odpovídá vzdálenostem jednotlivých překážek v prostředí okolo robota (viz měření A na obrázcích 10.5 a 10.6). Při pohybu robota vzniká na naměřeném snímku deformace vlivem odběru jednotlivých měření v jiný časový interval. Tento jev lze dobře pozorovat na měření B na obrázku 10.5. Vrchní vodorovná část by měla být tvořena jednou souvislou úsečkou, místo toho je zde výrazný skok způsobený tím, že nejvzdálenější vzorek byl změřen jedním senzorem na začátku daného snímku a nejbližší na konci snímku druhým senzorem. Robot se mezi těmito měřeními posunul odhadem o 35 cm. K nejvýraznější deformaci dochází při otáčení robota z důvodu principu funkce senzoru. Tento jev lze pozorovat u měření D na obrázku 10.5.



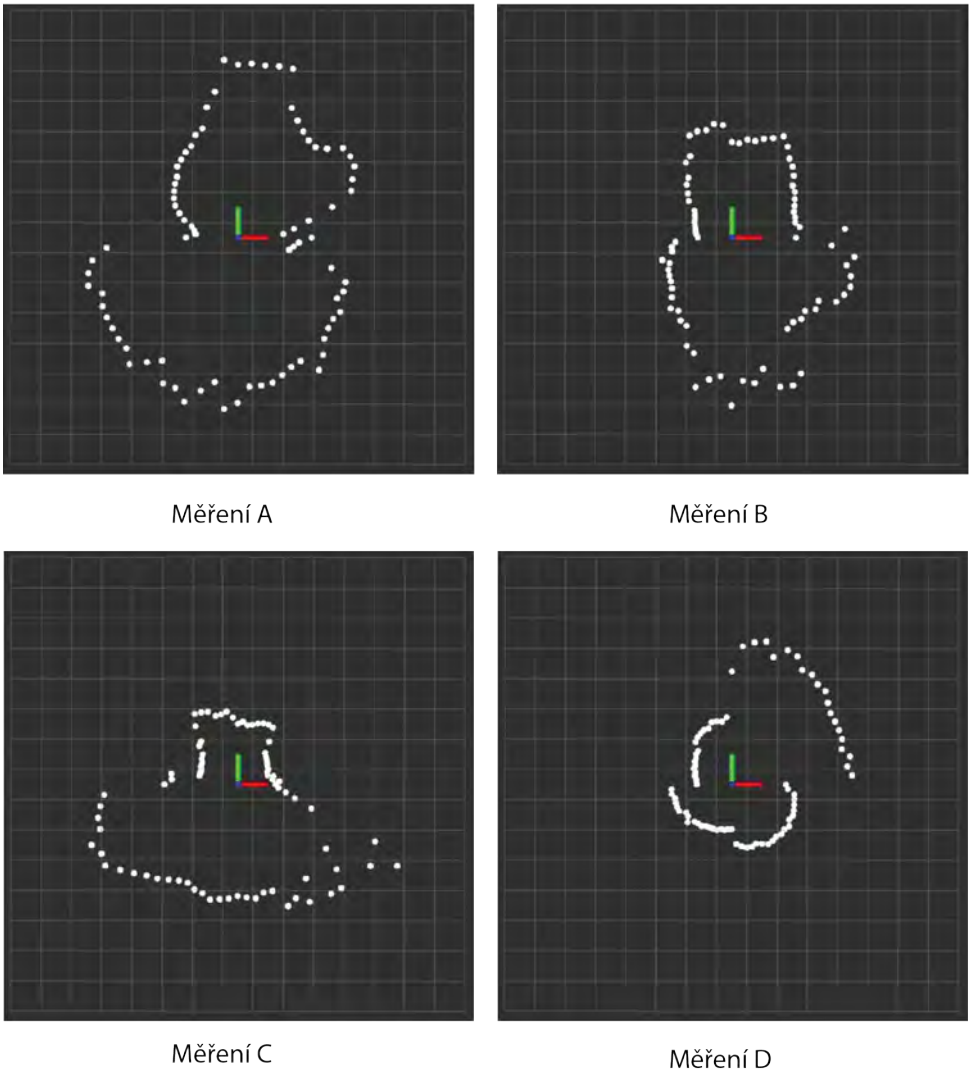
Obrázek 10.4: Testování: Náčrt rozmístění lavic při testování schopnosti Lidaru mapovat okolní prostředí.(Vzdálenosti uvedeny v mm)

Dále jsem porovnával teoretickou obnovovací frekvenci senzoru s reálnou. Teoretickou obnovovací frekvenci určíme pomocí následujících vztahů.

$$t_{turn} = \frac{n_{steps} \cdot n_{microsteps\cdot}}{f_{step}} = \frac{200 \cdot 64}{2500\,\mathrm{Hz}} = 5,12\mathrm{s} \tag{10.2}$$

$$t_{meas} = n_{measurements\_per\_turn} \cdot t_{one\_meas} = 80 \cdot 0.04 = 3,2\mathrm{s} \tag{10.3}$$

$$t_{frame} = \frac{t_{turn} + t_{meas}}{n_{sensor}} = \frac{5,12 + 3,2}{4} = 2{,}08\,\mathrm{s} \tag{10.4}$$



Obrázek 10.5: Testování: Mapování prostoru pomocí lidaru. Vykreslená data pomoci řídicí aplikace.

Reálnou obnovovací frekvenci jsem určil pomocí časových razítek (timestamp) u zkompletovaných zpráv z lidarového modulu ve vytvořeném programu Raspberry Pi . Průměrná obnovovací frekvence je 2,48 s. Zvýšení oproti teoretické hodnotě je způsobeno nezapočítáním komunikace mezi deskami Lidarového modulu. Při započtení doby přenosu 6 bytů při každém měření jsou již teoretické a naměřené hodnoty shodné.

Obrázek 10.6: Testování: Mapování prostoru pomocí Lidaru. Zachycení pozice při měření na obrázku 10.5.

10.1.2 AHRS

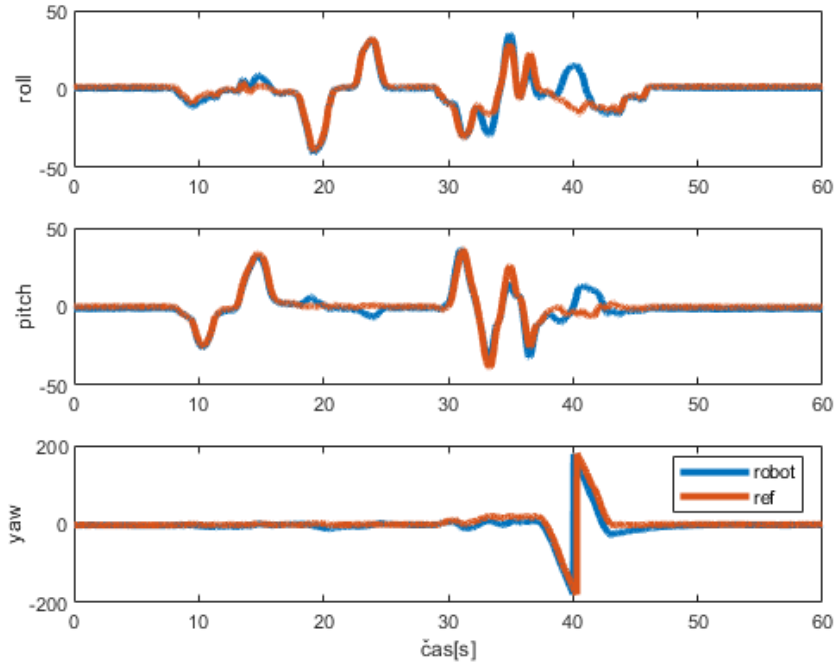
Pro otestování AHRS jsem použil srovnání navrženého systému s přesnou měřící jednotkou Innalabs ARHS M3. Referenční jednotku jsem připevnil tak, aby osy obou systémů ukazovaly stejným směrem. Připevnění jednotky je zdokumentováno na obrázku 10.7. Provedl jsem dva testy. Nejdříve jsem robota postupně naklonil ve všech osách systému. Srovnání výsledků je uvedeno na obrázku 10.8.

Obrázek 10.7: Testování: Připevnění referenční AHRS jednotky.

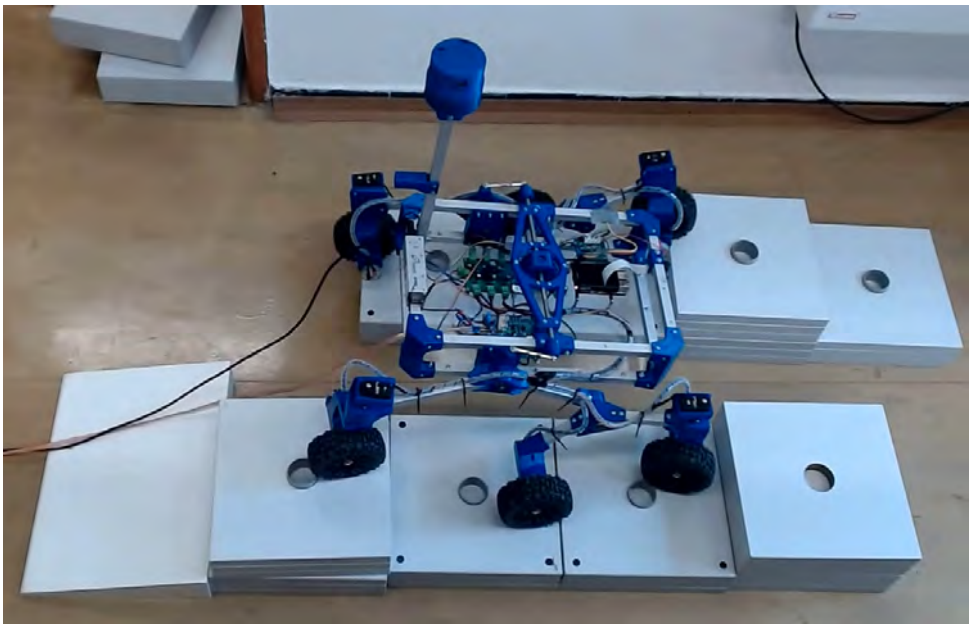
Lze zde pozorovat, že trend u referenčního i vytvořeného senzoru je podobný. Naměřená data se liší v offsetu osy yaw. To je zapříčiněno kalibračními algoritmy referenčního senzoru.

Pro druhý test jsem postavil umělý členitý terén, přes který jsem následně s robotem přejel. Na obrázku 10.9 je vyfocen robot na vytvořeném terénu. Z porovnání naměřených hodnot na obrázku 10.10 jsou vidět velké rozdíly. Tyto rozdíly mohou být způsobeny špatným umístěním senzoru vzhledem k centru rotace robota (IMU je umístěna ve spodní části Lidaru). Na datech z robota lze pozorovat velký rozkmit určené polohy při dopadu kola z vyšší kostky na nižší. To je způsobeno nedokonalou filtrací dat z akcelerometru u zvoleného algoritmu.

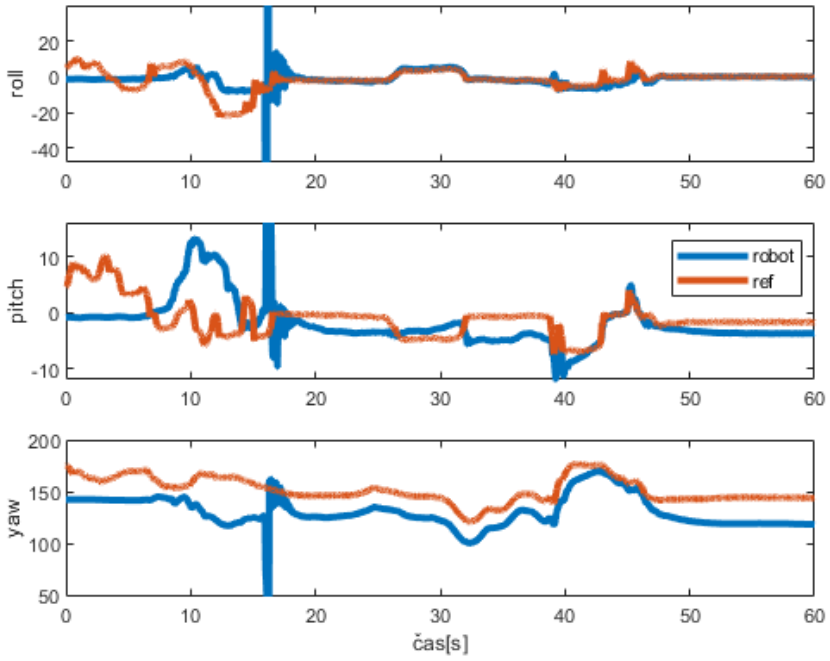
Nejdříve jsem zkontroloval naměřená data ze senzorů na robotu a v referečním systému. Výsledný směr vektoru získaného pomocí dat z akcelerometru i magnetometru vycházel velmi podobně. Při dalším průzkumu jsem zjistil, že při naklonění robota v ose roll o 90° detekuje algoritmus špatnou pozici. Chyba by mohla nastávat z důvodu odchylky v orientaci os magnetometru a inerciální měřící jednotky. Další možností je chyba v použité knihovně.



Obrázek 10.8: Testování: Porovnání výsledků AHRS robota a referenčního při naklonění okolo jednotlivých os.



Obrázek 10.9: Testování: Robot na členitém terénu při testování AHRS.



Obrázek 10.10: Testování: Porovnání výsledků AHRS robota a referenčního při přejezdu členitého terénu.

10.2 Motor Board

U motorové jednotky jsem testoval funkcionalitu řídicích algoritmů motorů.

10.2.1 Elektronický diferenciál

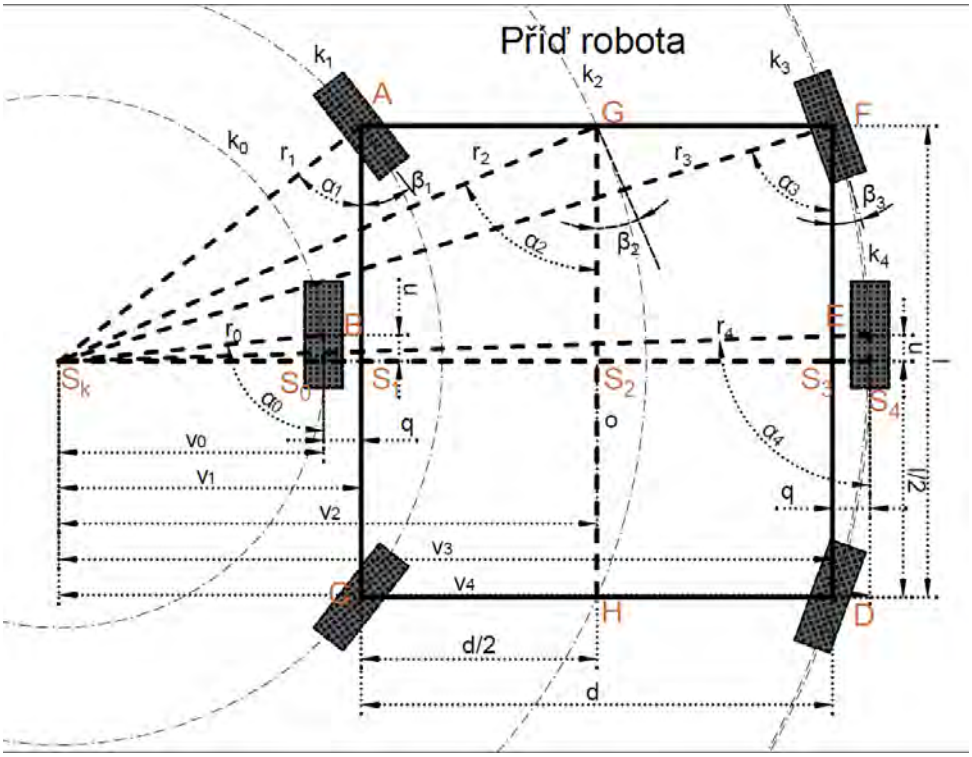
Při prvním testu pohyblivosti platformy jsem zjistil, že použitý zjednodušený přístup k zatáčení robota je nedostatečný. Pro zatáčení jsem používal stejný úhel natočení kol na levé i pravé straně a stejný výkon pro všechny motory.

Navrhl jsem systém elektronického diferenciálu, který vypočítává úhly natočení kol tak, aby se kola pohybovala po kružnicích se stejným středem. Systém na základě poměru jejich obvodů upravuje výkony jednotlivých motorů tak, aby se kola opisující větší kružnici pohybovala rychleji než ta, která opisují menší.

Referenční kružnici k_2 jsem zvolil ve středu mezi kružnicemi k_1 a k_3, které opisují natočitelná kola. Vstupní hodnotou algoritmu pro nastavení kol jsem zvolil úhel β_2 určený tečnou kružnice k_2 v bodě G/H a osou o_1 , která se nachází ve středu mezi úsečkami AC a DF (viz nákres na obrázku 10.11).

Pomocí hodnoty β_2 a vzdáleností jednotlivých kol l, d, q, u musím určit úhly natočení kol levé (β_1) a pravé (β_3) strany a poměry velikostí obvodů kružnic určené poloměry r_0, r_1, r_2, r_3, r_4.

Z nákresu na obrázku 10.11 lze vypozorovat několik vhodných pravoúh-



Obrázek 10.11: Testování: Nákres nastavení kol pro odvození vztahů elektronického diferenciálu.

lých trojúhelníků, pro vyjádření potřebných vztahů. Jedná se o trojúhelníky $\triangle S_k S_0 B$, $\triangle S_k S_1 A$, $\triangle S_k S_2 G$, $\triangle S_k S_3 F$ a $\triangle S_k S_4 E$. Nejdříve vypočítám úhel α_2 s využitím pravého úhlu mezi tečnou kružnice a jejím poloměrem:

$$\alpha_2 = 90° - \beta_2. \tag{10.5}$$

Následně již mohu určit výšky trojúhelníků v_0 - v_4 s pomocí definice funkce tangens:

$$v_2 = \frac{l}{2} \operatorname{tg} \alpha_2 \tag{10.6}$$

$$v_0 = v_2 - \frac{d}{2} - q \tag{10.7}$$

$$v_1 = v_2 - \frac{d}{2} \tag{10.8}$$

$$v_3 = v_2 + \frac{d}{2} \tag{10.9}$$

$$v_4 = v_2 + \frac{d}{2} + q \tag{10.10}$$

Opětovným použitím definice funkce tangens můžeme vyjádřit úhly α_1 a α_3:

$$\alpha_1 = \operatorname{arctg} \frac{2v_1}{l} = \operatorname{arctg} \left(\frac{2v_2 - d}{l} \right) = \operatorname{arctg} \left(\operatorname{tg} \alpha_2 - \frac{d}{l} \right) \tag{10.11}$$

$$\alpha_3 = \operatorname{arctg} \frac{2v_3}{l} = \operatorname{arctg} \left(\frac{2v_2 + d}{l} \right) = \operatorname{arctg} \left(\operatorname{tg} \alpha_2 + \frac{d}{l} \right) \tag{10.12}$$

Pro výpočet poměrů obvodů kružnic (k_0,k_1,k_3,k_4), po kterých se pohybují kola robota vůči referenční kružnici k_2, stačí porovnávat jejich poloměry, zbytek vzorce pro výpočet obvodu se vykrátí. Poloměry r_0 - r_4 kružnic k_0 - k_4 můžeme dopočítat pomocí Pythagorovy věty:

$$r_i = \sqrt{v_i^2 + \left(\frac{l}{2}\right)^2} \qquad i \in \{1,\ 2,\ 3\} \tag{10.13}$$

$$r_j = \sqrt{v_j^2 + u^2} \qquad j \in \{0,\ 4\}. \tag{10.14}$$

Následně již stačí vyjádřit poměry pro upravení výkonů jednotlivých kol:

$$p_B = sign(v_0)\frac{r_0}{r_2} \tag{10.15}$$

$$p_{AC} = sign(v_1)\frac{r_1}{r_2} \tag{10.16}$$

$$p_{DF} = sign(v_2)\frac{r_3}{r_2} \tag{10.17}$$

$$p_E = sign(v_3)\frac{r_4}{r_2}. \tag{10.18}$$

Spodním indexem je značeno přiřazení poměru k jednotlivým kolům. Kola jsou značena body v jejich středu (proti směru hodinových ručiček od levého předního kola - A, B, C, D, E, F).

Při použití Pythagorovy věty přijdeme o znaménka. Vzdálenosti l a u jsou vždy kladné, ale výšky v_0, v_1, v_3 a v_4 mohou nabývat záporných hodnot. Tento jev nám značí, že střed kružnic opisovaných koly S_k je blíže ke středu robota (S_2) než příslušné kolo, proto se kolo musí otáčet opačným směrem. K jevu například dojde, jestliže bod S_k je blíže bodu S_2 než S_0 nebo S_1.

Při návrhu algoritmu pro úpravu výkonu jednotlivých kol jsem provedl zjednodušení, že závislost úhlové rychlosti kol na dodávaném výkonu motorům je lineární. Při tomto zjednodušení mohu vynásobit referenční hodnotu výkonu motorů příslušnými poměry.

Po vyjádření jednotlivých vztahů jsem je zpracoval do firmwaru jednotky. Vytvořil jsem malou knihovnu nazvanou *eletric_diff*, poskytující potřebné hodnoty.

Jelikož tato funkce bude často volána se stejnými hodnotami, rozhodl jsem se pro zvýšení responsibility a efektivity firmwaru vytvořit tzv. lookup tabulku s předpočítanými hodnotami β_1, β_3, p_B, p_{AC}, p_{DF}, p_E, pro všechny možné příchozí hodnoty úhlu β_2. (Celkem 501 hodnot. Úhel ±45° je mapován na ±250 hodnot.). Před implementací tabulkového přístupu jsem zkontroloval, jestli použitý mikrokontroler disponuje dostatečným množstvím volné paměti RAM, protože velikost vytvořené tabulky bude necelé 3 kB.

10.2.2 Omezení maximální akcelerace

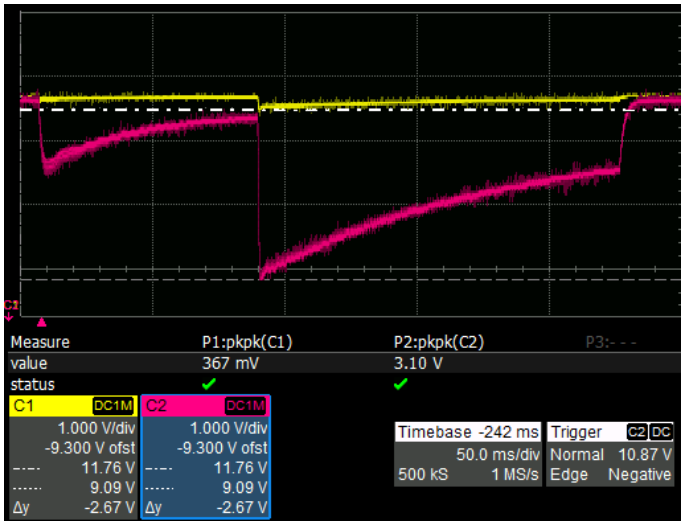
Robota jsem většinu času testování napájel pomocí napájecí zdroje TEOLLNER TEO 8871 a dlouhé výkonové dvoulinky kvůli minimalizaci opotřebení

baterií. Na začátku testování docházelo k restartům řídicí elektroniky při velké skokové změně výkonu dodávaného motorům. Největší problém to činilo u hlavní řídicí jednotky Raspberry Pi , u které to mohlo mít za následek nutnou reinstalaci operačního systému z důvodu poškození souborového systému na SD kartě.

Důvod poklesu napájecího napětí robota byl úbytek na napájecí dvojlince. Pro další testování jsem použil osciloskop. Jednu sondu jsem připojil přímo na výstup zdroje a druhou na vstupní konektor nápájení robota. Nejdříve jsem orientačně určil odpor pomocí úbytku napětí na kabelu a dodávaným proudem zobrazeným na zdroji při konstantním stoprocentním výkonu motorů bez zatížení.

$$R_{kabel} = \frac{\Delta U}{I} = \frac{0,610}{5,7} = 107\,\mathrm{m\Omega} \tag{10.19}$$

Dále jsem na osciloskopu zachytil pokles napětí způsobený proudovou špičkou při skokové změně výkonu z 0 na 100 %. Průběh je zachycen na obrázku 10.12, červeně je uvedeno napětí na napájecím vstupu robota a žlutě na výstupu zdroje. Z obrázku lze pozorovat pokles napětí z 12 V na 9 V. Použitý step-down zdroj pro Raspberry Pi potřebuje pro svou funkci minimálně 8,6 V a špatně potlačuje takto velké výkyvy vstupního napětí. Z obrázku jsem určil velikost proudové špičky za pomocí orientační hodnoty odporu dvojlinky. Maximální hodnota odebíraného proudu přesahovala 25 A.

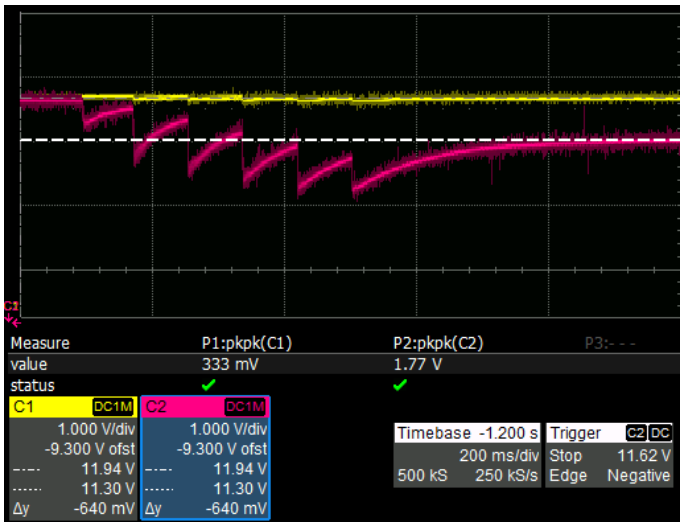


Obrázek 10.12: Testování: Skoková změna výkonu z nuly na plný výkon bez implementace plynulého rozjezdu.

Nejdříve jsem se snažil tyto poklesy napájecího napětí řešit pomocí velkých blokovacích kondenzátorů. Zkusil jsem napájecí zdroj Raspberry Pi oddělit od zbytku obvodů Schottkyho diodou a za ní umístit blokovací kondenzátor. Další kondenzátor jsem umístil na napájecí vstup Raspberry Pi a další na napájecí vstup motorů. Pokles napětí měl dlouhé trvání na použité hodnoty kondenzátorů a řešení nepomohlo.

Rozhodl jsem se tedy proudové špičky omezit pomocí softwaru. Omezil

jsem maximální okamžitou změnu výkonu při zrychlování na 15% celkového výkonu. To mi umožnilo snížit maximální proudovou špičku o 10 A. Robot při tomto omezení dokáže zrychlit na svoji maximální rychlost do 400 ms. Při zpomalování není potřeba změnu výkonu omezovat. Dalším omezením bylo zamezení skokové změny směru otáčení kol. Na obrázku 10.13 lze pozorovat postupné maximální zrychlení z 0 % na 100% výkonu motorů.



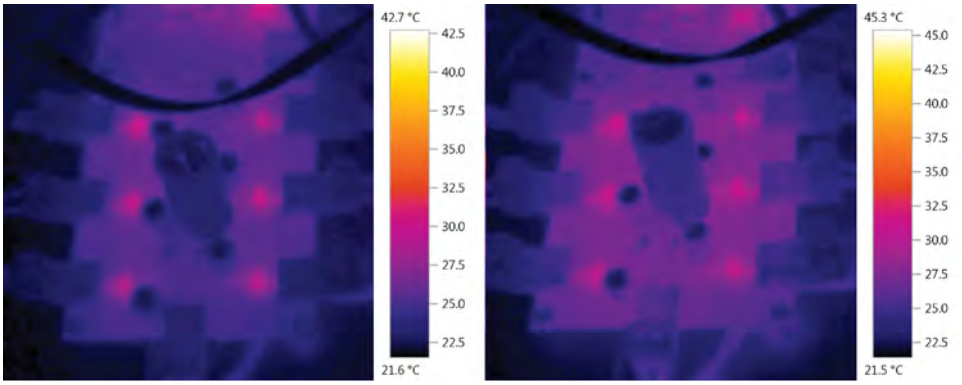
Obrázek 10.13: Testování: Omezení maximální změny výkonu. Postupné zrychlení z nuly na plný výkon.

10.3 Kontrola teploty součástek

Další částí, co jsem při testování kontroloval, byly teploty součástek na jednotlivých tištěných spojích. Při jejich přehřívání by docházelo ke zkrácení jejich životnosti. Pro měření jsem použil termokameru Testo 875-1.

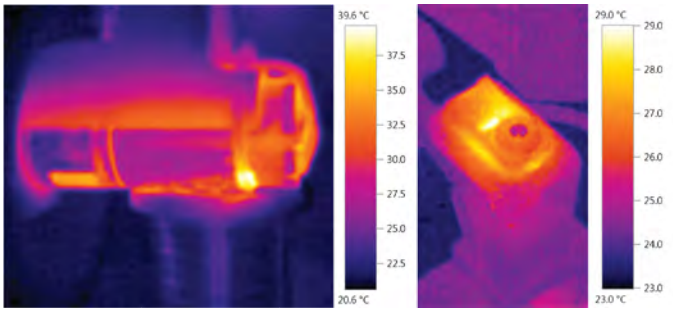
Nejdříve jsem testoval teploty součástek při aktivaci motorů na Motor Boardu. Motory hnacích kol jsem podložil tak, aby se kola nedotýkala podložky a mohla se volně točit. Následně jsem aktivoval napájení robota a spustil kola robota na maximální výkon. Teplotu součástek na Motor Boardu jsem změřil po jedné, pěti a deseti minutách. Za tuto dobu se teploty driverů motorů ohřály přibližně o 4 °C na 28 °C a u ostatních součástek jsem nezaznamenal viditelné ohřátí. Fotografie z měření teploty jednotky pro ovládání motorů termokamerou jsou uvedeny na obrázku 10.14. Následně jsem zkontroloval teplotu motorů a servomotorů (viz obrázek 10.15). Teplota pláště motorů vzrostla za deset minut přibližně na 35 °C. Serva zůstala na pokojové teplotě. Motory při klasickém používání pravděpodobně nebudou běžet deset minut na plný výkon. Tedy naměřená hodnota je v pořádku.

Dále jsem provedl měření teploty Power Boardu (viz obrázek 10.16). Zde jsem kontroloval teplotu spínaného zdroje a dalších výkonových částí. Robot byl před měřením teploty napájen více jak 5 hodin. Všechny moduly, které mají být připojeny na 5 V spínaný zdroj, byly aktivní. Na pohledu termokamerou



a) po 1 minutě b) po 10 minutách

Obrázek 10.14: Testování: Kontrola teploty součástek na motorovém modulu.

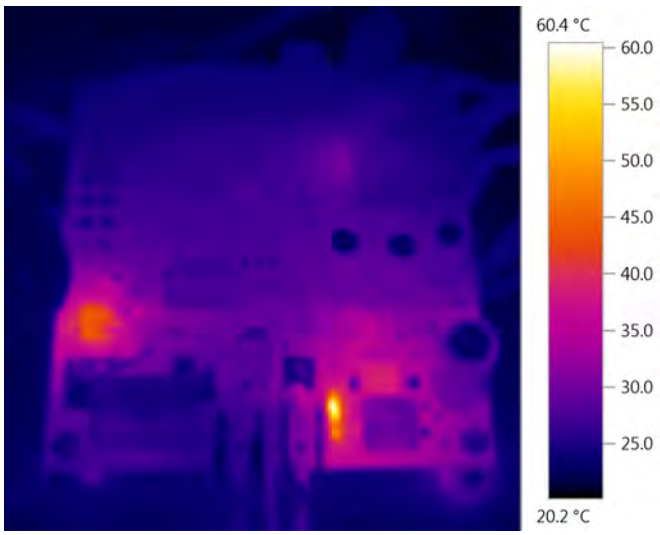


Obrázek 10.15: Testování: Kontrola teploty servomotorů a DC motorů.

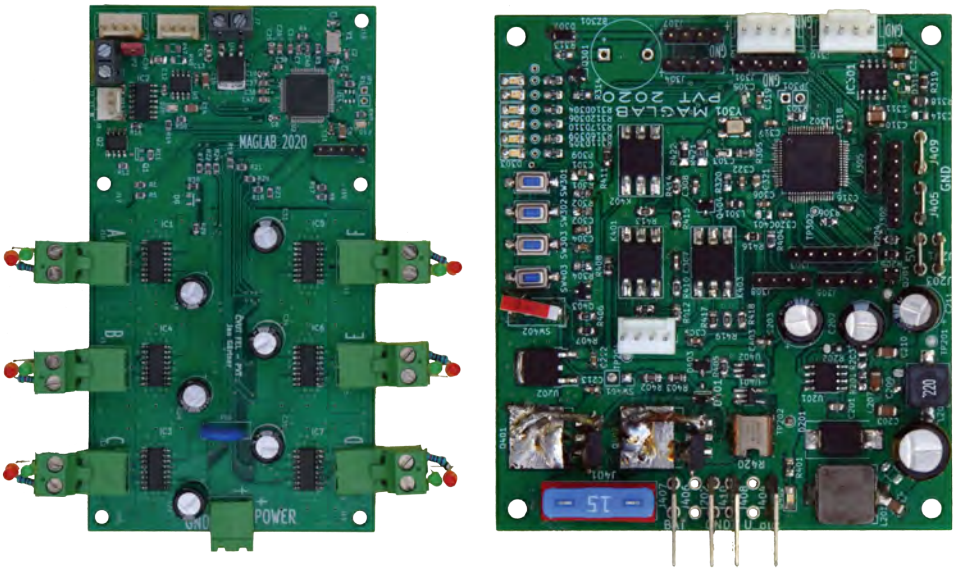
jsem nedetekoval žádné výrazné zvýšení teploty této části. Detekoval jsem zvýšenou teplotu LED značící aktivní napájecí výstup (přibližně 55 °C) a lineárního regulátoru napětí 3V3 (přibližně 45 °C). První zvýšenou teplotu jsem potlačil snížením proudu tekoucího LED pomocí zvýšení hodnoty odporu sériově zapojeného rezistoru. Teplotu lineárního regulátoru nelze jednoduše snížit, je zapříčiněna připojením regulátoru na vstupní napájení 12V, aby mohl připojený MCU fungovat i při deaktivaci napájecích výstupů.

Na obrázku 10.17 jsou uvedeny testované tištěné spoje pro srovnání se snímky z termovize.

Teplota chladiče Raspberry Pi 4 po pěti hodinách nepřetržité funkce při zatížení očekávaném za provozu robota byla přibližně 35 °C. Teplota procesoru se po celou dobu pohybovala pod 48 °C.



Obrázek 10.16: Testování: Kontrola teploty součástek na napájecím modulu.



Motorový modul (Motor Board) Napájecí modul (Power Board)

Obrázek 10.17: Testování: Fotografie modulů pro lepší orientaci v snímcích termovize.

Kapitola 11

Závěr

Navrhl, vytvořil a zprovoznil jsem senzor pro mapování okolí za použití čtyř levných lidarových senzorů VL53L1x od firmy STmicroeletronics. Součástí senzoru je AHRS jednotka umožňující získání natočení robota v prostoru.

Tento senzor umožňuje spolehlivě zmapovat okolní prostředí do vzdálenosti 2 metrů. Počet měřených pozic na jedno otočení a z toho pramenící rozlišení je volitelné pomocí řídicího rámce poslaného po sběrnici robota. Při základním nastavení rozlišení (80 pozic na jedno otočení) je senzor schopen vytvořit snímek okolního prostředí za 2,08 s.

Senzor má problém při mapování prostředí za pohybu robota z důvodu relativně pomalého vzorkování okolního prostředí. To je zapříčiněno zvoleným typem senzorů vzdálenosti, které vyžadují minimálně 20 ms na provedení jednoho měření. Z tohoto důvodu není možné vrchní částí lidaru plynule otáčet, ale musel jsem zvolit přístup s natáčením senzoru do předem připravených pozic.

Pro budoucí verzi lidarového modulu doporučuji použít jiné senzory vzdálenosti, například systém založený na integrovaném obvodu OPT3101 TFmini lidar, který umožňuje měřit vzdálenosti 0 - 12 m se vzorkovací frekvencí až 4 kHz.

Na diplomové práci jsem začal pracovat již v rámci předmětu Projekt v týmu v druhém semestru magisterského studia. S mými kolegy jsme se rozhodli postavit robotickou platformu SAWPPY s vlastním elektronickým hardwarem. Kvůli podcenění časové náročnosti celého projektu a distanční výuce z důvodu nemoci COVID-19 jsme za vyhrazený půlrok stihli pouze navrhnout všechny potřebné desky plošných spojů. DPS byly přes zkouškové období vyrobeny a následně osazeny osobami zodpovědnými za jejich vývoj.

Následně jsem již na daném projektu pokračoval samostatně. Upravil jsem původní návrh konstrukce robota tak, aby místo původních servomotorů používal pro pohon stejnosměrné motory a následně všechny plastové díly vytiskl na 3D tiskárně.

Při návštěvách fakulty přes prázdniny jsem sestavil mechanickou část platformy. Při opětovném zavedení distanční výuky na začátku třetího semestru jsem si odvezl domů všechny řídicí desky a postupně jsem se seznámil s prací mých kolegů, oživil a naprogramoval všechny moduly.

Při návrhu softwaru jsem vytvořil několik knihoven, které jsem sdílel napříč

firmwary pro jednotlivé moduly, pro urychlení a usnadnění vývoje. Jednalo se například o knihovny implementující pokročilé funkce časovačů a ovládání periferie CAN bus.

Na desce pro ovládání motorů jsem implementoval funkci elektronického diferenciálu, který dopočítává úhly natočení jednotlivých kol a jejich výkony tak, aby se při zatáčení pohybovala po soustředných kružnicích.

Obrázek 11.1: Závěr: Fotografie výsledné platformy.

Na Power Boardu jsem implementoval funkce pro sledování a kontrolu stavu nabití baterie. Implementovaný algoritmus využívá data o napětí jednotlivých článků a o odebíraném proudu z baterie robota. Okamžitá hodnota odebíraného proudu je integrována za účelem výpočtu odebrané kapacity baterie. Pro uložení stavu nabití při restartování desky je použito zapisování do flash paměti procesoru. Použitý algoritmus implementuje rozpoznání dobití baterie mezi dvěma zapnutími pomocí sledování napětí na jednotlivých článcích.

Na Raspberry Pi jsem nainstaloval systém Raspberry Pi OS. Ze zdrojových kódů jsem nainstaloval ROS. Dále jsem pomocí tzv. DT overlays aktivoval CAN kontroler MCP2517fd. Vytvořil jsem balíček pro ROS, který implementuje uzel pro komunikaci po sběrnici CAN bus za pomocí linuxových knihoven socketCAN. Uzel zprávy interpretuje a publikuje na odpovídající téma. Další uzel zajišťuje příjem dat z radiového ovladače a jejich publikaci pro další uzly. Data publikovaná těmito uzly jsou následně zpracovávána dalšími uzly

za účelem řízení robota. Jsou zde vytvořeny uzly pro spojení dat z lidarů, připojení webového rozhraní a pro lokalizaci na základě algoritmů SLAM. Dalším důležitým uzlem je *Main Control*, který zajišťuje zpracování signálů z jednotlivých ovládacích rozhraní a následně řídí pohyb robota.

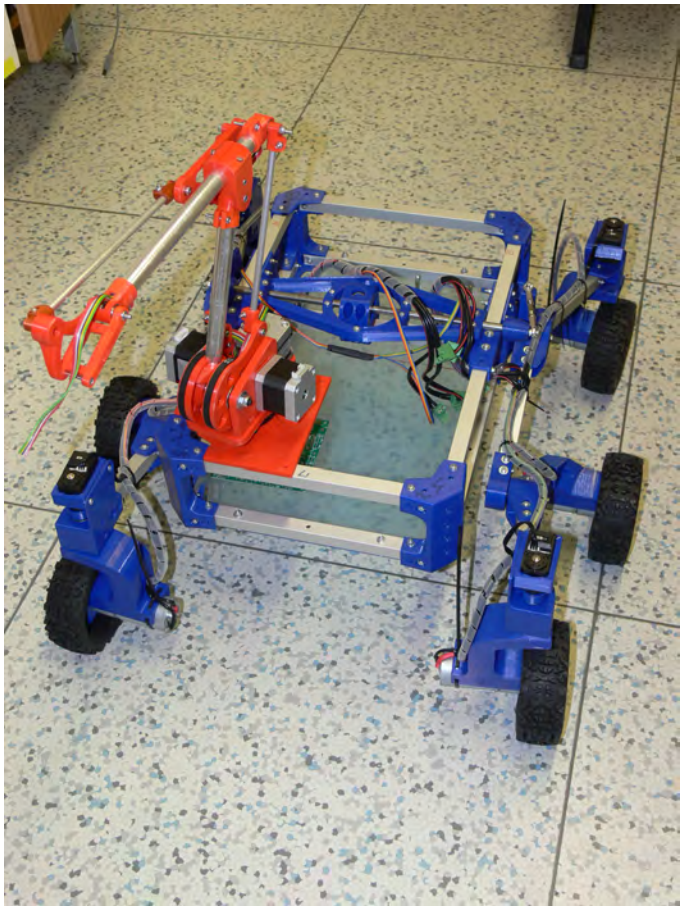
Robota je možné ovládat ze dvou rozhraní. Prvním z nich je radiový ovladač. Druhým je webové rozhraní. Robot na své IP adrese hostuje webovou stránku, na které je mimo ovládání robota možné pozorovat informace o nabití baterie, vykreslení dat z lidarového modulu nebo přenos videa z kamery na Raspberry Pi.

V rámci dalších prací na robotické platformě je v plánu naučit robota mluvit. Webové rozhraní a ROS balíček je již na tuto funkcionalitu připraven. Zbývá vytvořit hardware pro zesílení audiovýstupu Raspberry Pi a celou funkcionalitu otestovat.

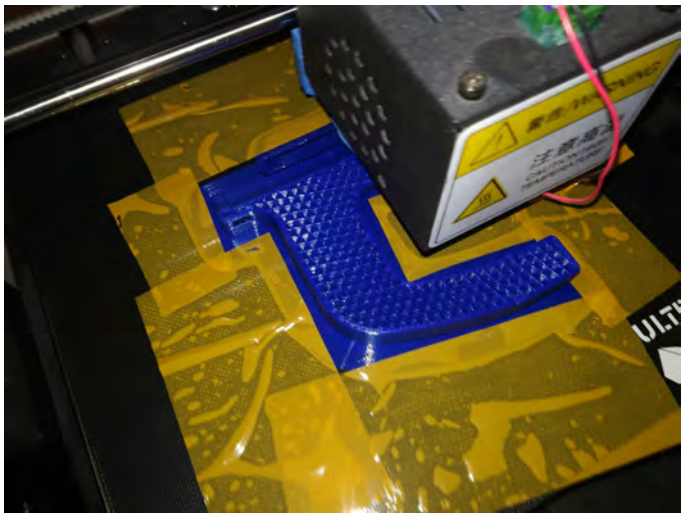
V rámci bakalářské práce kolegy vzniká robotická ruka, která bude připevněna na přední část centrální platformy. V rámci dokončovacích prací je potřeba začlenit robotickou ruku do řídicí aplikace v Raspberry Pi. Ruku lze pozorovat na obrázku 11.2.

Při vývoji robotické platformy jsem používal gitlab repozitář dostupný na adrese: `https://gitlab.fel.cvut.cz/doubrpa1/pvt-rover`

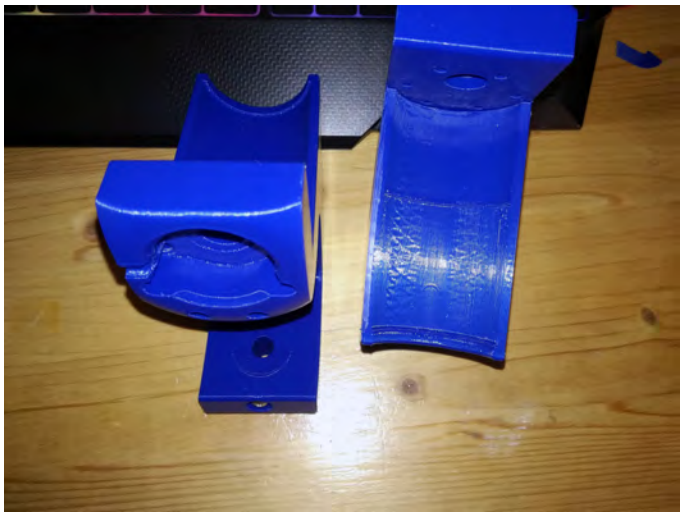
Zadání diplomové práce bylo splněno v plném rozsahu. Na následujících obrázcích je vybraná fotodokumentace projektu.



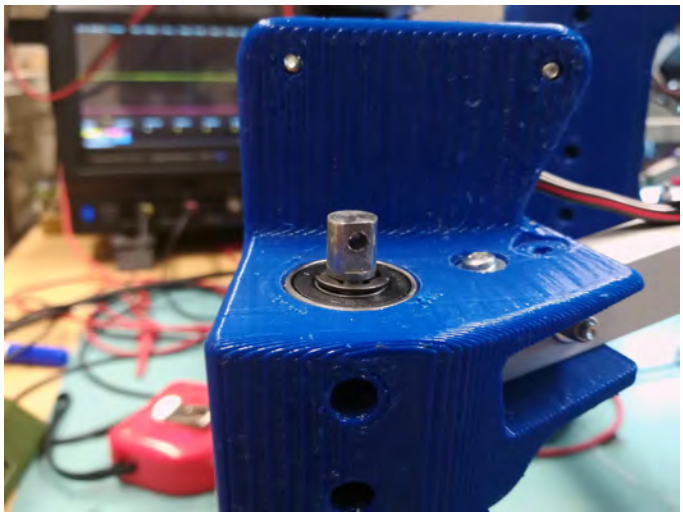
Obrázek 11.2: Závěr: Dokončená konstrukce robota s dokončenou kabeláží motorů a konstrukcí robotické ruky



Obrázek 11.3: Závěr: Tisk kloubu pro uchycení motoru s kolem.

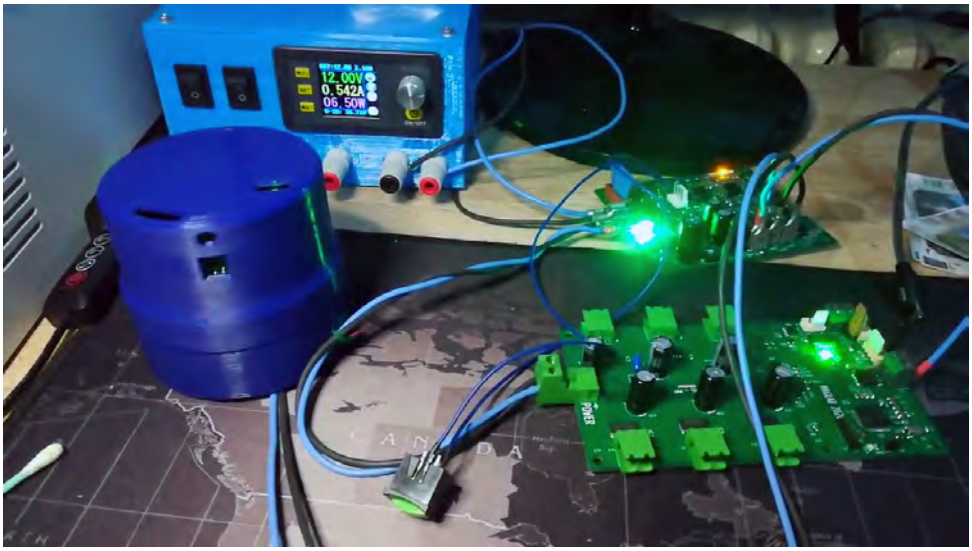


Obrázek 11.4: Závěr: Vytištěné klouby pro uchycení motoru s kolem



Obrázek 11.5: Závěr: Detail na osu pro natáčení kol.

Obrázek 11.6: Závěr: Testování přenosu UARTu. Je zde pozorovatelný problém se zpožděním sestupné hrany, který bylo potřeba potlačit.



Obrázek 11.7: Závěr: Zapojení modulů při vývoji jejich firmwaru v domácím prostředí.

Obrázek 11.8: Závěr: Testování bezdrátového napájení.

Příloha A

Literatura

[1] "A brief introduction to controller area network," navštíveno: 2021-04-02. [Online]. Available: https://copperhilltech.com/a-brief-introduction-to-controller-area-network

[2] *CAN Specification version 2.0*, Robert Bosch GmbH, září 1991. [Online]. Available: https://web.archive.org/web/20170621031900/http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf

[3] "Can protocol," navštíveno: 2021-04-22. [Online]. Available: https://piembsystech.com/can-protocol/

[4] S. O. Madgwick, "An efficient orientation filter for inertial and inertial/-magnetic sensor arrays," *ESA - Special Publication*, únor 2010.

[5] N. Olsen, T. Risbo, P. Brauer, J. Merayo, and F. Primdahl, "In-flight calibration methods used for the Ørsted mission," *ESA - Special Publication*, únor 1970.

[6] R. Cheng, "Sawppy the rover," 2021. [Online]. Available: https://github.com/Roger-random/Sawppy_Rover

[7] *STSPIN220 - Low voltage stepper motor driver*, STMicroelectronics, prosinec 2020, rev. 5. [Online]. Available: https://www.st.com/resource/en/datasheet/stspin220.pdf

[8] *Automotive fully integrated H-bridge motor driver*, STMicroelectronics, květen 2016, rev. 6. [Online]. Available: https://www.st.com/resource/en/datasheet/vnh7100as.pdf

[9] "History of can technology," navštíveno: 2021-04-02. [Online]. Available: https://www.can-cia.org/can-knowledge/can/can-history/

[10] *Understanding Microchip's CAN Module Bit Timing*, Microchip, 2001. [Online]. Available: http://ww1.microchip.com/downloads/en/AppNotes/00754.pdf

[11] S. Royo and M. Ballesta-Garcia, "An overview of lidar imaging systems for autonomous vehicles," *Applied Sciences*, vol. 9, no. 19, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/19/4093

[12] S. A. Ludwig, K. D. Burnham, A. R. Jiménez, and P. A. Touma, "Comparison of attitude and heading reference systems using foot mounted MIMU sensor data: basic, Madgwick, and Mahony," in *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems 2018*, H. Sohn, Ed., vol. 10598, International Society for Optics and Photonics. SPIE, 2018, pp. 644 – 650. [Online]. Available: https://doi.org/10.1117/12.2296568

[13] P. Doubrava, "Systém pro generování libovolného magnetického vektoru magnetického pole s potlačením okolního rušení," Bachelor's thesis, ČVUT, květen 2019. [Online]. Available: https://dspace.cvut.cz/handle/10467/82753

[14] R. V. Hogg, "Statistical robustness: One view of its use in applications today," *The American Statistician*, vol. 33, no. 3, pp. 401–411, 1979. [Online]. Available: https://doi.org/10.1080/00031305.1979.10482673

[15] *Getting started with STM32F10xxx hardware development*, STMicroelectronics, listopad 2011, rev 7. [Online]. Available: https://bit.ly/2RFKrSH

[16] *TS51221 High Efficiency Regulator IC for Wireless Power Receiver Applications*, Semtech, duben 2015, rev. 1.1.

[17] *TS51231 High Efficiency Transmitter Driver for Wireless Power Systems*, Semtech, srpen 2018, rev. 2.0.

[18] *TS80002 High-Efficiency Transmit Controller for Wireless Power Systems*, Semtech, únor 2019, rev. 1.2.

[19] *Reference manual - STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs*, STMicroelectronics, únor 2021, rev. 21. [Online]. Available: https://bit.ly/3afyOIr

[20] *LewanSoul Bus Servo Communication Protocol*, LewanSoul. [Online]. Available: https://www.dropbox.com/sh/b3v81sb9nwir16q/AABZRndzrcVjE1-Tbv-JmsAva/LX-16A%20Bus%20Servo/LewanSoul%20Bus%20Servo%20Communication%20Protocol.pdf?dl=01

[21] R. Yagfarov, M. Ivanou, and I. Afanasyev, "Map comparison of lidar-based 2d slam algorithms using precise ground truth," listopad 2018.

[22] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Proc.*

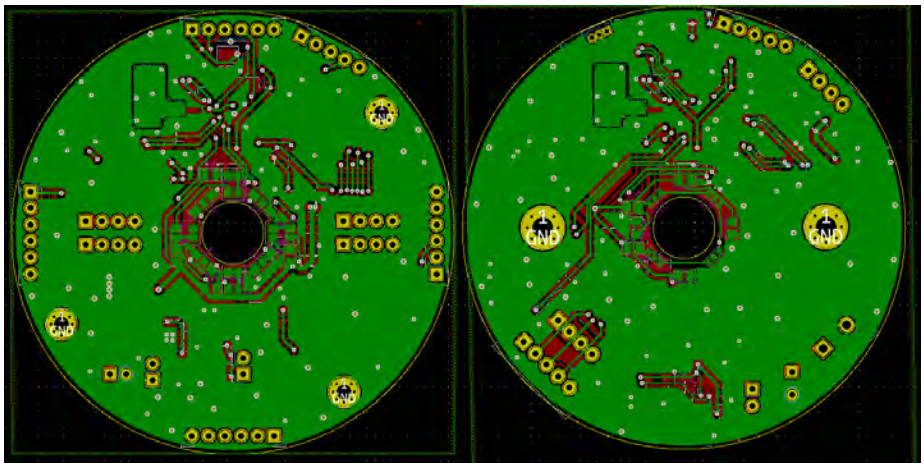
IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR). IEEE, listopad 2011.

[23] *How to set up hector_slam for your robot*, Stefan Kohlbrecher, srpen 2012. [Online]. Available: http://wiki.ros.org/hector_slam/Tutorials/SettingUpForYourRobot

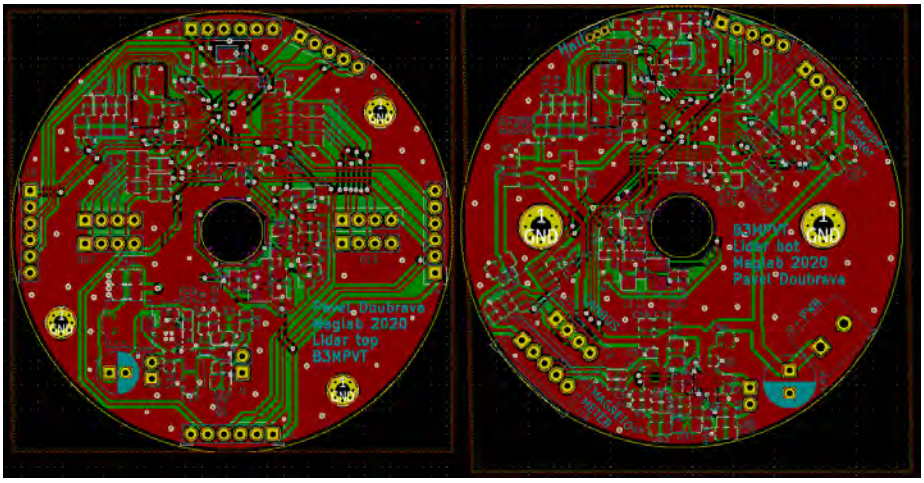
Příloha B

Schémata

B.1 Lidar



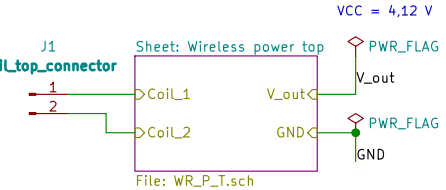
(a) : Spodní vrstva mědi



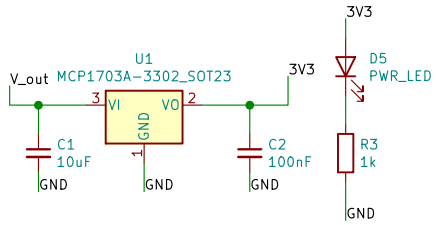
(b) : Vrchní vrstva mědi

Obrázek B.1: Screenshoty návrhu tištěných spojů lidarového modulu

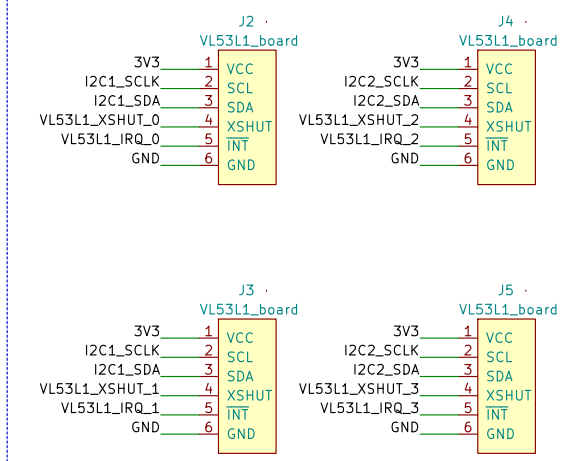
Bezdrátové napájení (přijímač)



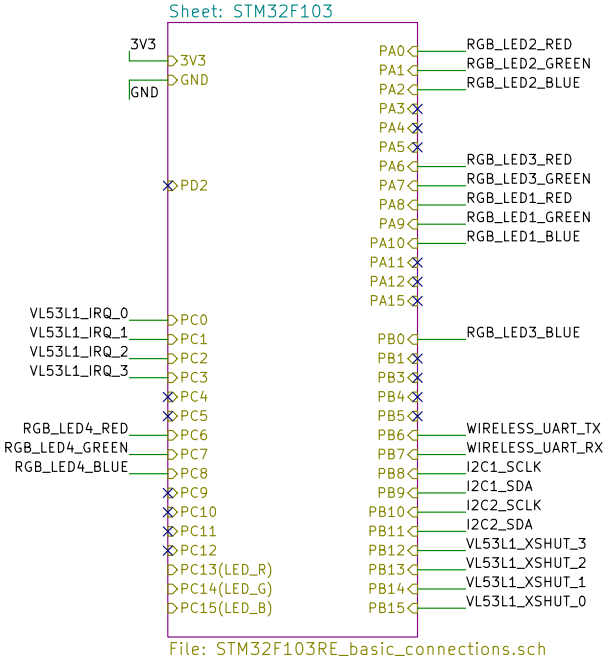
Regulátor napětí 3V3



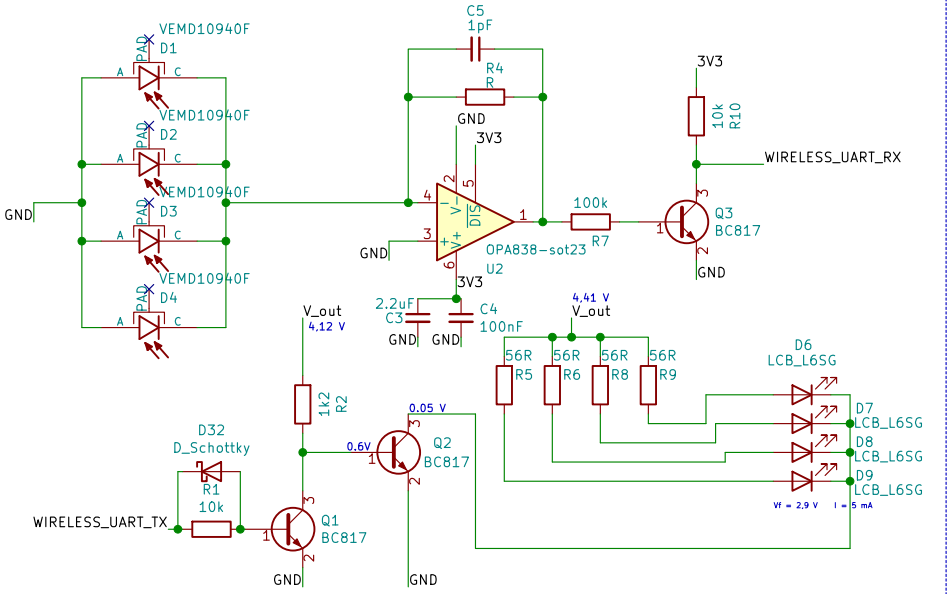
Konektory pro lidary



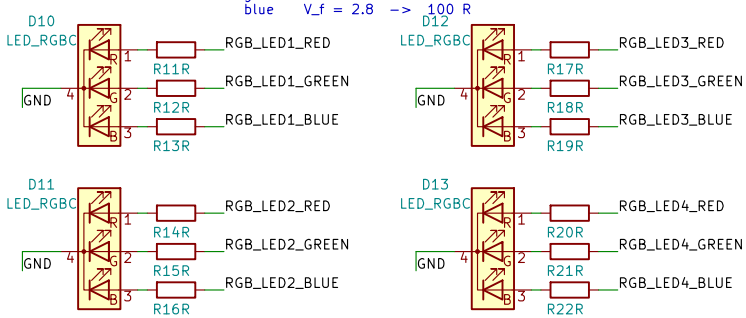
Procesor



Optický obousměrný přenos Uartu (vrchní část)



Světelné efekty 4x RGB led



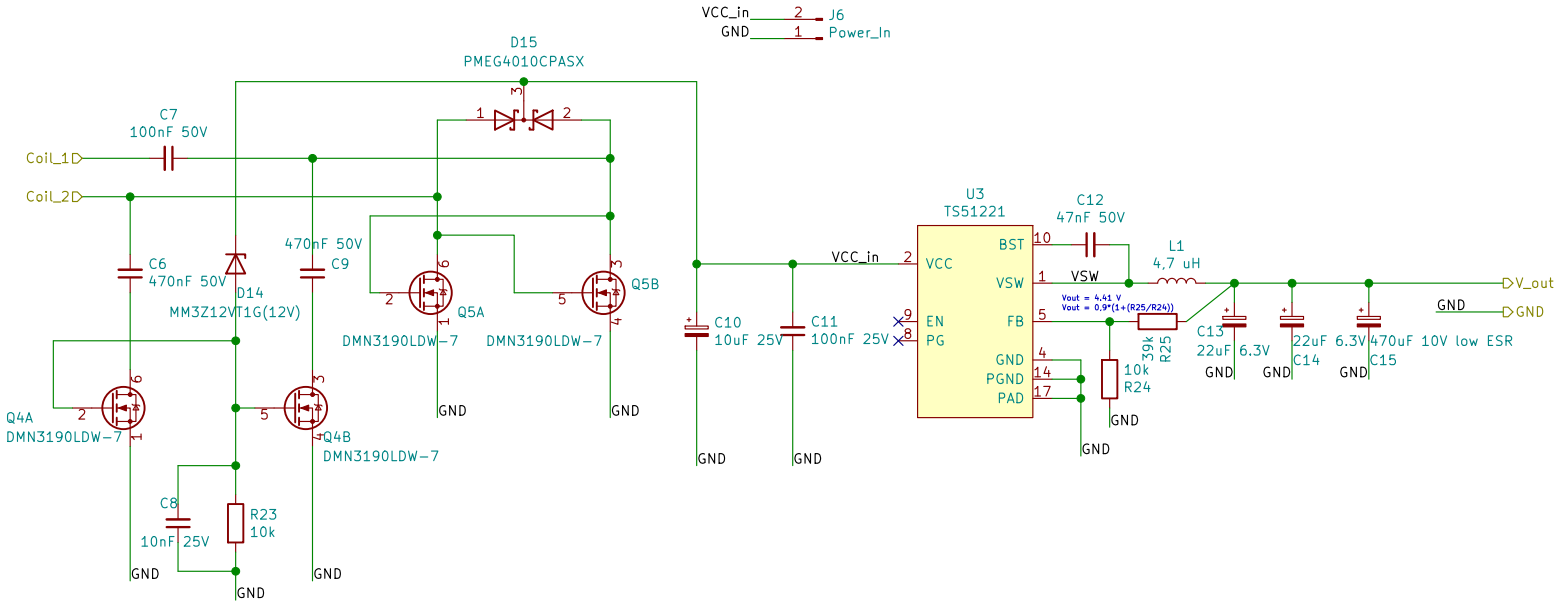
H1 H2 H3
MountingHole_Pad
GND GND GND

Pavel Doubrava

Sheet: /Lidar_top/
File: Lidar_top.sch

Title: Lidar top – Main

Size: A4	Date: 2020-10-20	**Rev: 5**
KiCad E.D.A. kicad (5.1.9)-1		Id: 2/8

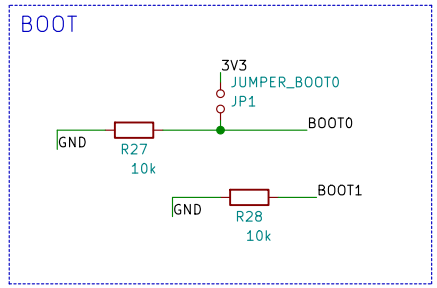


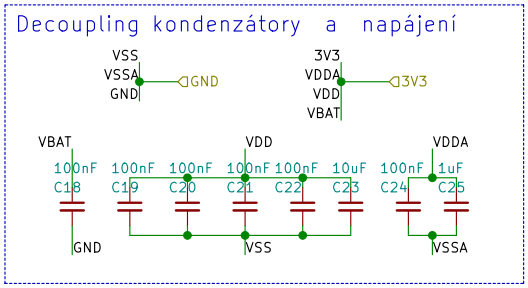
Maglab
Pavel Doubrava

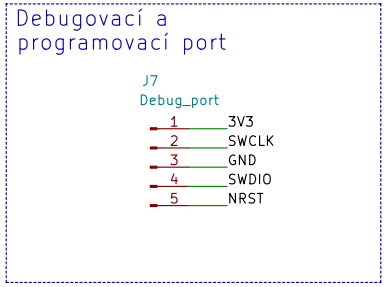
Sheet: /Lidar_top/Wireless power top/
File: WR_P_T.sch

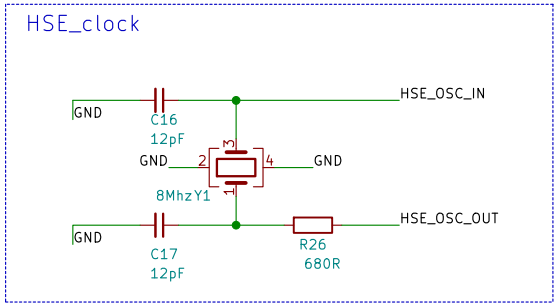
Title: Lidar top – Wireless power receiver

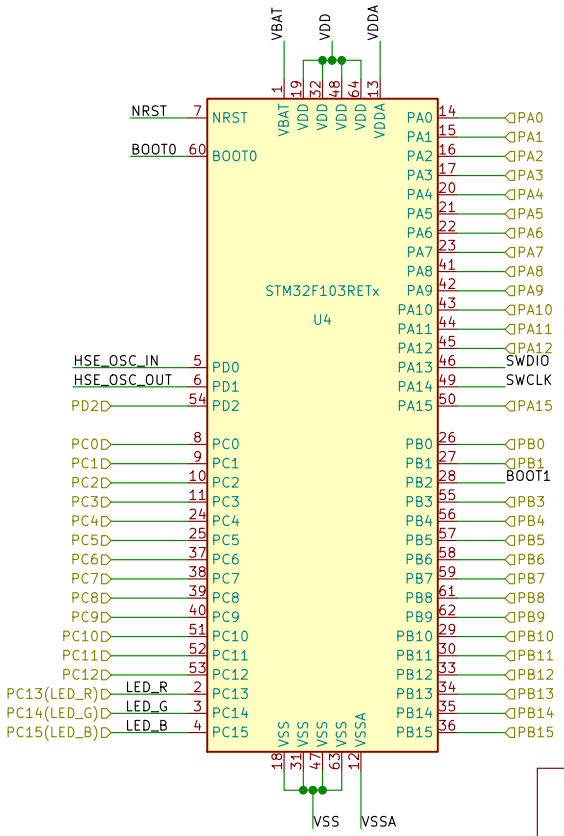
Size: A4	Date: 2020-10-20	**Rev: 5**
KiCad E.D.A. kicad (5.1.9)-1		Id: 3/8

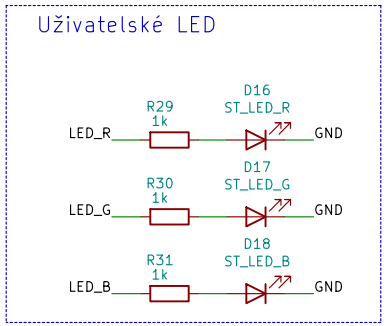










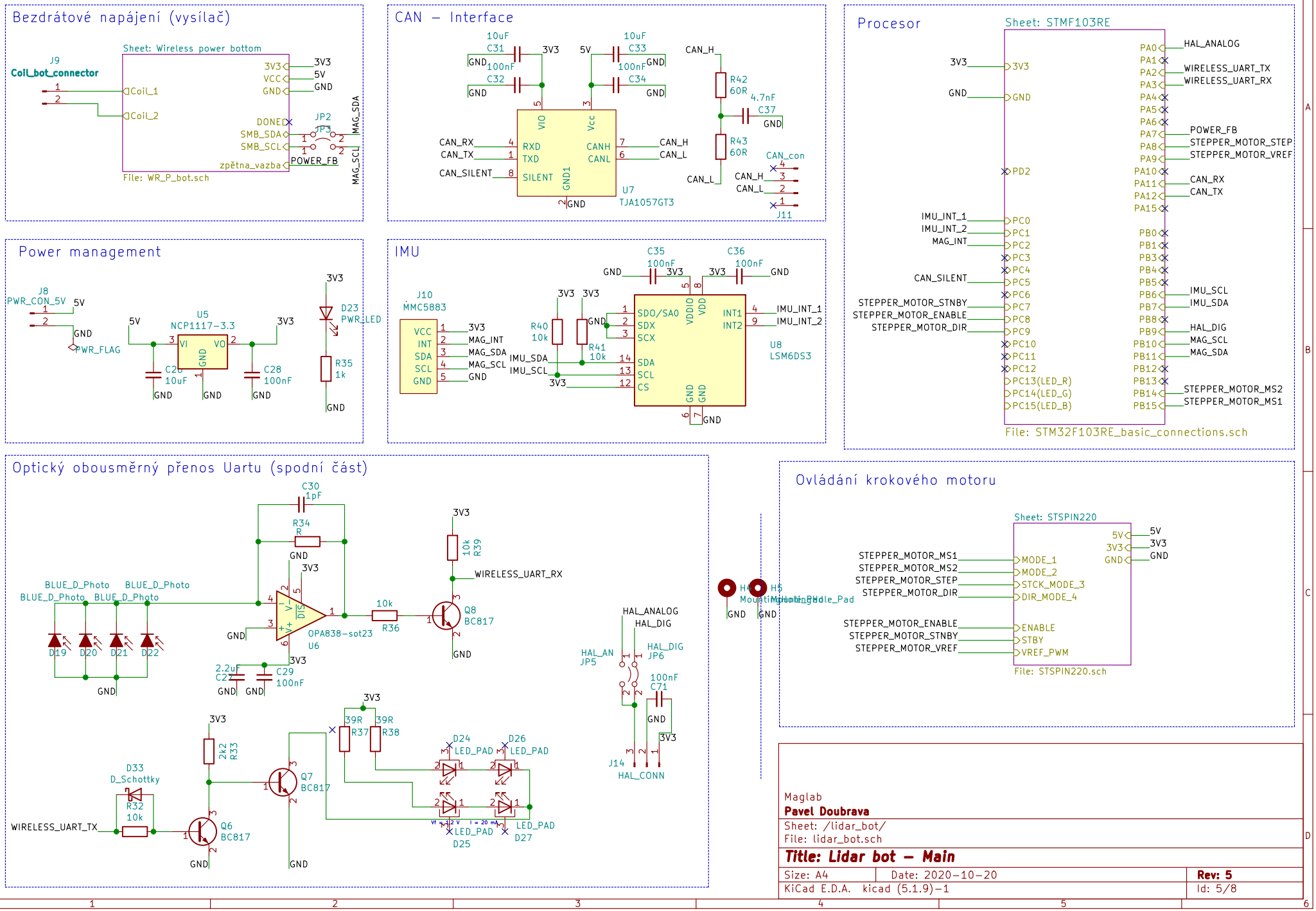


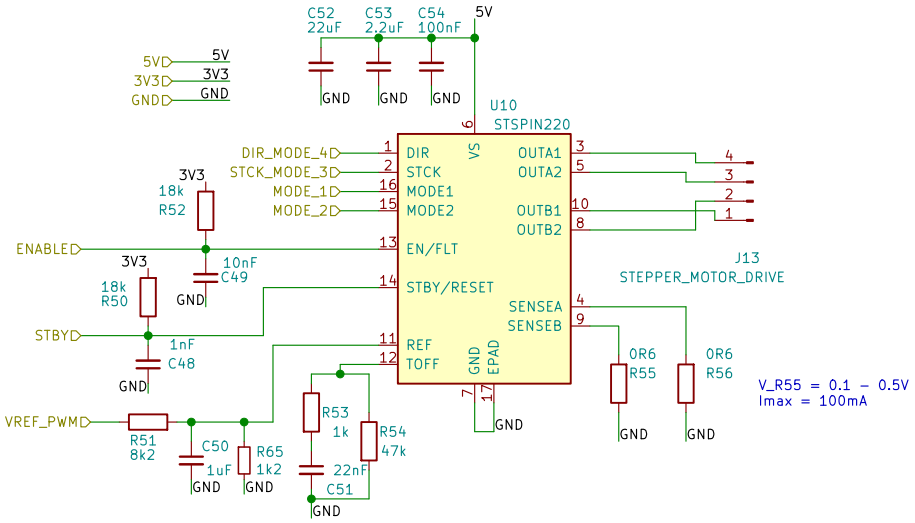
Yellow If = 7 mA Uf = 2 V
Red If = 5 mA Uf = 1,8 V
Green If = 2 mA Uf = 2,5 V
Blue If = 5 mA Uf = 2,7 V

Maglab
Pavel Doubrava
Sheet: /Lidar_top/STM32F103/
File: STM32F103RE_basic_connections.sch

Title: MCU basic connections

Size: A4	Date: 2020-10-20	**Rev: 5**
KiCad E.D.A. kicad (5.1.9)-1		Id: 4/8


Bezdrátové napájení (vysílač)
Sheet: Wireless power bottom
J9
Coil_bot_connector
Coil_1
Coil_2
3V3
VCC
GND
DONE
SMB_SDA
SMB_SCL
zpětna_vazba
POWER_FB
MAG_SDA
MAG_SCL
JP2
JP3
File: WR_P_bot.sch
CAN – Interface
C31 10uF
C32 100nF
C33 10uF
C34 100nF
R42 60R
R43 60R
C37 4.7nF
CAN_H
CAN_L
CAN_RX
CAN_TX
CAN_SILENT
VIO
Vcc
RXD
TXD
SILENT
CANH
CANL
GND1
U7
TJA1057GT3
CAN_con
J11
Procesor
Sheet: STMF103RE
PA0 HAL_ANALOG
PA2 WIRELESS_UART_TX
PA3 WIRELESS_UART_RX
PA7 POWER_FB
PA8 STEPPER_MOTOR_STEP
PA9 STEPPER_MOTOR_VREF
PA10 CAN_RX
PA11 CAN_TX
PD2
PC0 IMU_INT_1
PC1 IMU_INT_2
PC2 MAG_INT
PC5 CAN_SILENT
PC6 STEPPER_MOTOR_STNBY
PC7 STEPPER_MOTOR_ENABLE
PC8 STEPPER_MOTOR_DIR
PB6 IMU_SCL
PB7 IMU_SDA
PB9 HAL_DIG
PB10 MAG_SCL
PB11 MAG_SDA
PC13(LED_R)
PC14(LED_G)
PC15(LED_B)
PB14 STEPPER_MOTOR_MS2
PB15 STEPPER_MOTOR_MS1
File: STM32F103RE_basic_connections.sch
Power management
J8
PWR_CON_5V
PWR_FLAG
U5
NCP1117-3.3
C26 10uF
C28 100nF
D23 PWR_LED
R35 1k
IMU
J10
MMC5883
VCC
INT
SDA
SCL
MAG_INT
MAG_SDA
MAG_SCL
C35 100nF
C36 100nF
R40 10k
R41 10k
SDO/SA0
SDX
SCX
SDA
SCL
CS
VDDIO
VDD
INT1
INT2
IMU_INT_1
IMU_INT_2
U8
LSM6DS3
Optický obousměrný přenos Uartu (spodní část)
C30 1pF
R34
BLUE_D_Photo
D19
D20
D21
D22
OPA838-sot23
U6
R36 10k
Q8 BC817
WIRELESS_UART_RX
C29 100nF
C27 2.2uF
R39 10k
HAL_ANALOG
HAL_DIG
HAL_AN JP5
HAL_DIG JP6
C71 100nF
J14 HAL_CONN
D33 D_Schottky
R32 10k
R33 2k2
R37 39R
R38 39R
Q6 BC817
Q7 BC817
D24 LED_PAD
D25 LED_PAD
D26 LED_PAD
D27 LED_PAD
Ovládání krokového motoru
Sheet: STSPIN220
STEPPER_MOTOR_MS1
STEPPER_MOTOR_MS2
STEPPER_MOTOR_STEP
STEPPER_MOTOR_DIR
STEPPER_MOTOR_ENABLE
STEPPER_MOTOR_STNBY
STEPPER_MOTOR_VREF
MODE_1
MODE_2
STCK_MODE_3
DIR_MODE_4
ENABLE
STBY
VREF_PWM
File: STSPIN220.sch
H4 H5
Maglab
Pavel Doubrava
Sheet: /lidar_bot/
File: lidar_bot.sch
Title: Lidar bot – Main
Size: A4
Date: 2020-10-20
Rev: 5
KiCad E.D.A. kicad (5.1.9)-1
Id: 5/8



Maglab

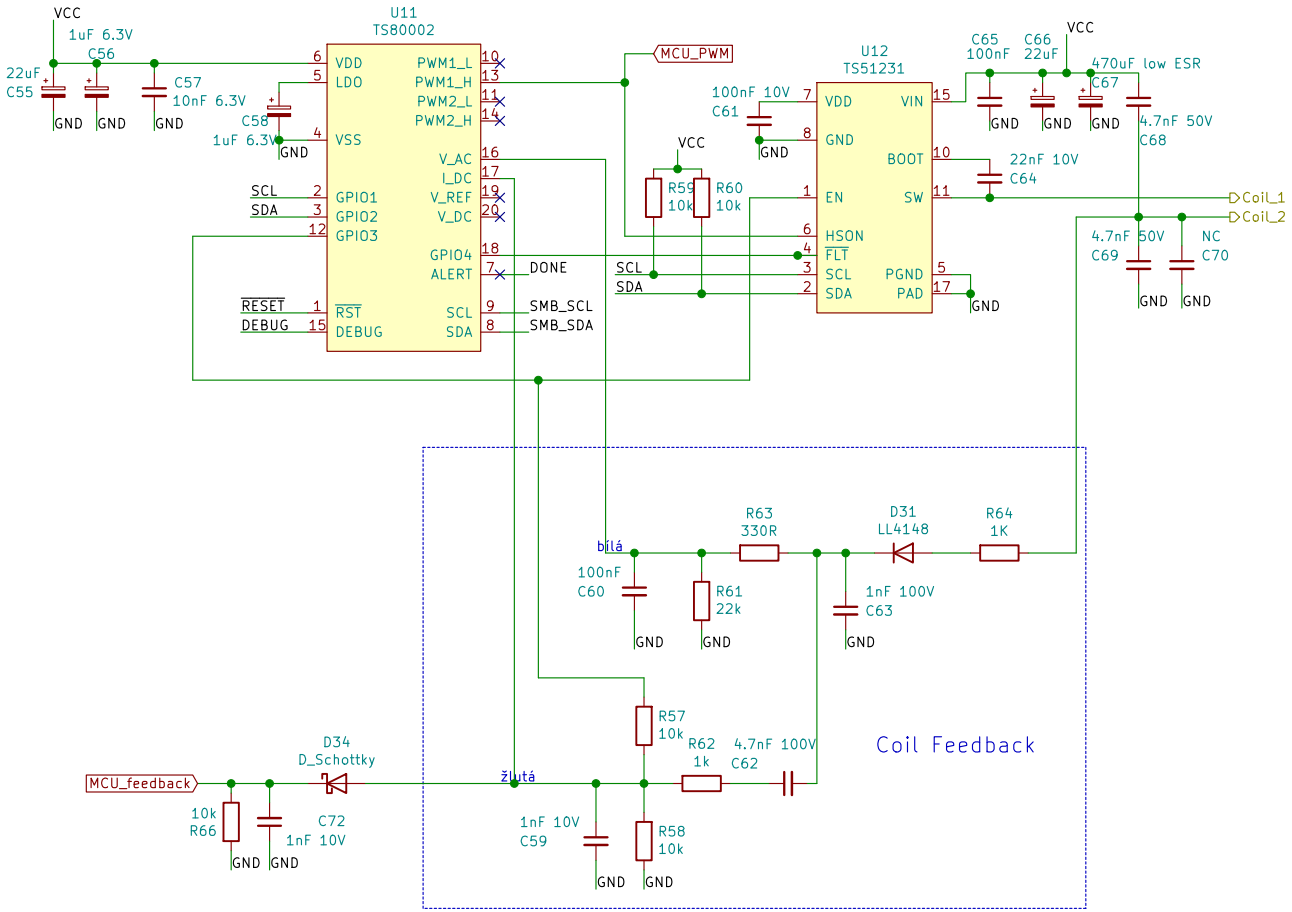
Pavel Doubrava

Sheet: /lidar_bot/STSPIN220/

File: STSPIN220.sch

Title: Lidar bot – Stepper motor

Size: A4	Date: 2020-10-20	**Rev: 5**
KiCad E.D.A. kicad (5.1.9)-1		Id: 6/8



RESET
DEBUG

3V3
VCC
GND

DONE
SMB_SDA
SMB_SCL

zpětna_vazba

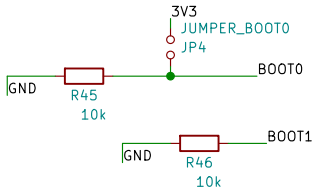
Maglab
Pavel Doubrava

Sheet: /lidar_bot/Wireless power bottom/
File: WR_P_bot.sch

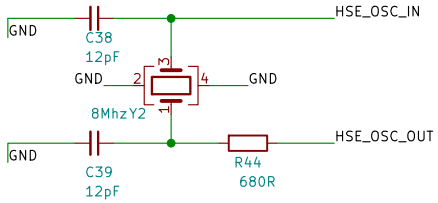
Title: Lidar top – Wireless power transmitter

Size: A4	Date: 2020-10-20	**Rev: 5**
KiCad E.D.A. kicad (5.1.9)-1		Id: 7/8

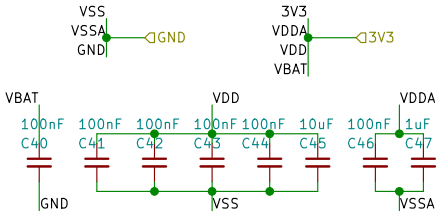
BOOT

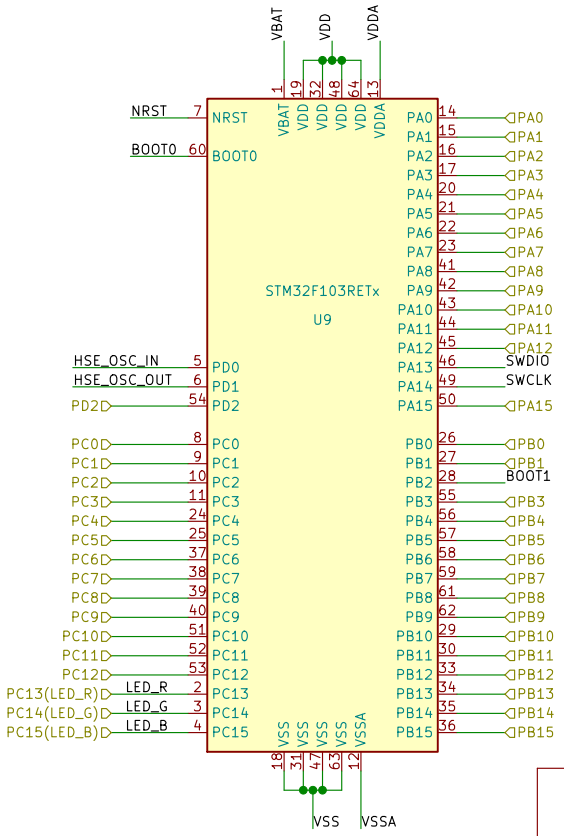


HSE_clock

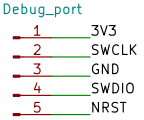


Decoupling kondenzátory a napájení

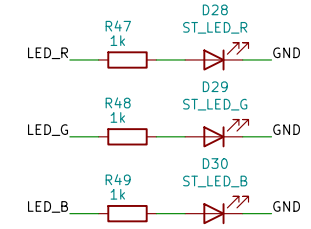




Debugovací a programovací port



Uživatelské LED



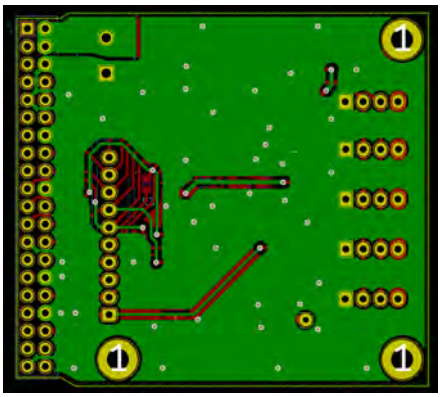
Yellow If = 7 mA Uf = 2 V
Red If = 5 mA Uf = 1,8 V
Green If = 2 mA Uf = 2,5 V
Blue If = 5 mA Uf = 2,7 V

Maglab
Pavel Doubrava
Sheet: /lidar_bot/STMF103RE/
File: STM32F103RE_basic_connections.sch

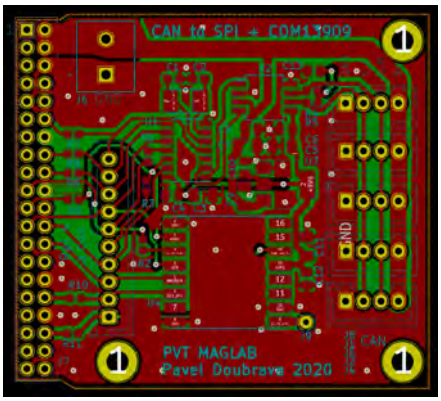
Title: MCU basic connections

Size: A4	Date: 2020-10-20	**Rev: 5**
KiCad E.D.A. kicad (5.1.9)-1		Id: 8/8

B.2 Communication hat

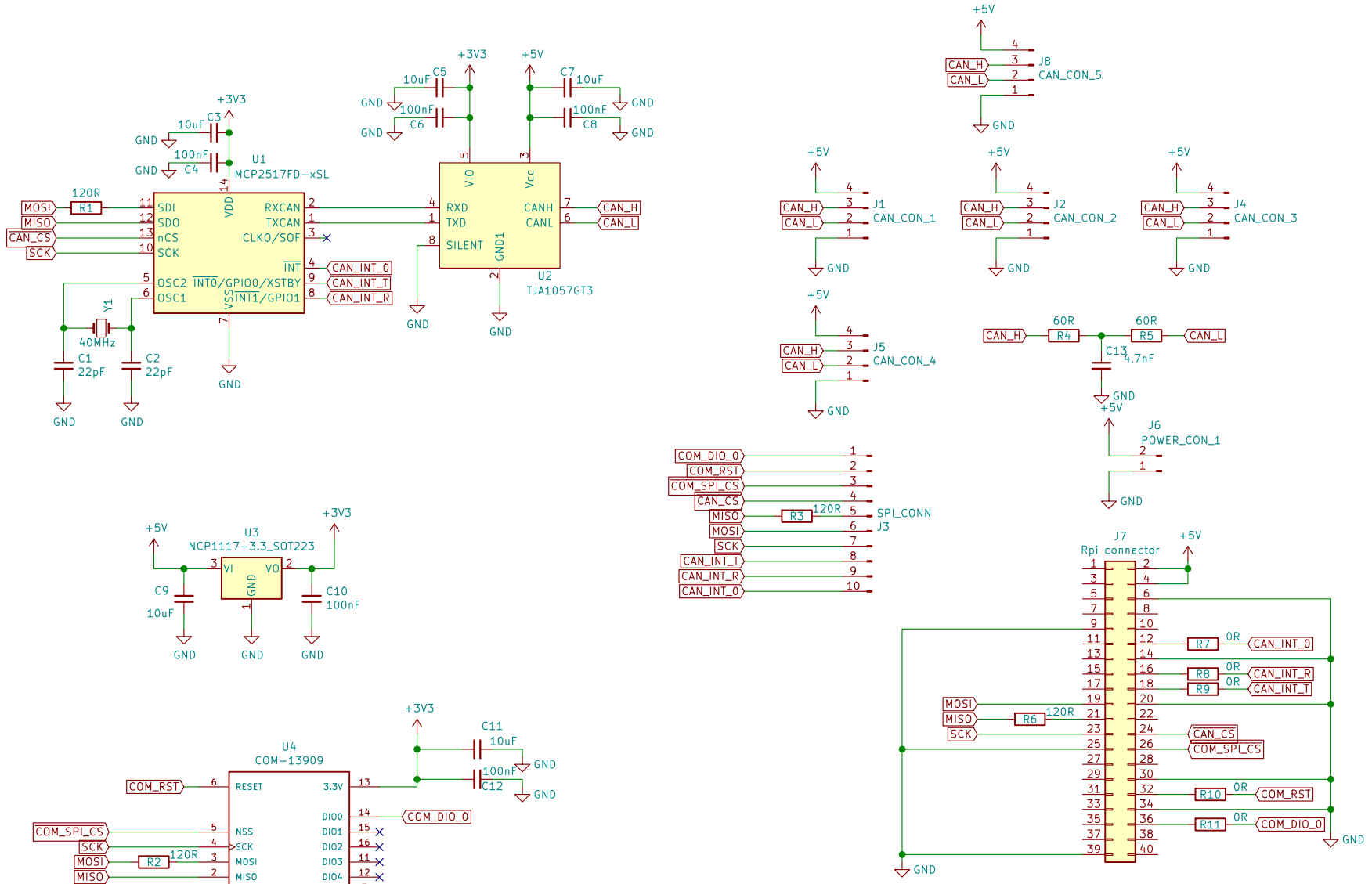


(a) : Spodní vrstva mědi



(b) : Vrchní vrstva mědi

Obrázek B.2: Screenshoty návrhu tištěného spoje modulu communication hat



Maglab
Pavel Doubrava

Sheet: /
File: CAN_to_SPI.sch

Title: Communication hat

Size: A4	Date: 2020-04-20	**Rev: 4**
KiCad E.D.A. kicad (5.1.9)-1		Id: 1/1

Příloha C

CAN bus přehled rámců

Přehled CAN zpráv

Barevné značení komponent			Lidar	L	
			Motor Board	M	
			Power board	P	
CAN_ID [hex]	Binární hodnota identifikátoru	Komp.	Název rámce	Směr	Velikost dat [byte]
0x300	0011 0000 0000	P	PowerBoard_Main_Controll	In	4
0x301	0011 0000 0001	P	PowerBoard_ADC_Meas_Data	Out	6
0x303	0011 0000 0011	P	PowerBoard_battery_status	Out	6
0x302	0011 0000 0010	P	PowerBoard_get_status	In	2
0x304	0011 0000 0100	P	PowerBoard_main_setup	Out	3
0x202	0010 0000 0010	L	LidarBoard_ranging_data_1	Out	8
0x203	0010 0000 0011	L	LidarBoard_ranging_data_2	Out	8
0x201	0010 0000 0001	L	LidarBoard_tilt_info	Out	6
0x200	0010 0000 0000	L	LidarBoard_Main_Controll	In	6
0x204	0010 0000 0100	L	LidarBoard_magnet_calibration	Out	6
0x205	0010 0000 0101	L	LidarBoard_parity_error	Out	0
0x206	0010 0000 0110	L	LidarBoard_feedback_meas	Out	2
0x207	0010 0000 0111	L	LidarBoard_quaternion_part1	Out	8
0x208	0010 0000 1000	L	LidarBoard_quaternion_part2	Out	8
0x100	0001 0000 0000	M	MotorBoard_Main_Controll	In	8
0x101	0001 0000 0001	M	MotorBoard_set_drive_status	In	6
0x102	0001 0000 0010	M	MotorBoard_get_drive_status	In	0
0x103	0001 0000 0011	M	MotorBoard_drive_status	Out	6

Datové typy:	signed
	unsigned
	float/double

CAN_ID [hex]	Název rámce	Datové byty							
		1	2	3	4	5	6	7	8
0x300	PowerBoard_Main_Controll	enable_v	Bat_Track	rep_int_meas_data[ms]					
0x301	PowerBoard_ADC_Meas_Data	V_Cell_1		V_Cell_2		V_Cell_3			
0x303	PowerBoard_battery_status	percentage		time [s]		mAh drawn			
0x302	PowerBoard_get_status	status_id							
0x304	PowerBoard_main_setup	rep_int_meas_data[ms]		Bat_Track					
0x202	LidarBoard_ranging_data_1	Frame_id		L_Sens_0		L_Sens_1		frame_num	
0x203	LidarBoard_ranging_data_2	Frame_id		L_Sens_2		L_Sens_3		frame_num	
0x201	LidarBoard_tilt_info	Yaw		Pitch		Roll			
0x200	LidarBoard_Main_Controll	Lidar_mode	Meas_EN	Frame_num		rep_int_tilt			
0x204	LidarBoard_magnet_calibration	x		y		z			
0x205	LidarBoard_parity_error								
0x206	LidarBoard_feedback_meas	ADC_meas							
0x207	LidarBoard_quaternion_part1	q1				q2			
0x208	LidarBoard_quaternion_part2	q3				q4			
0x100	MotorBoard_Main_Controll	EN_servo	EN_adap_m						
0x101	MotorBoard_set_drive_status	mot_mode	steer_mode	mot_power		steer_angle			
0x102	MotorBoard_get_drive_status								
0x103	MotorBoard_drive_status	mot_mode	steer_mode	mot_power		steer_angle			