



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

DEPARTMENT OF INTELLIGENT SYSTEMS

MODULÁRNÍ ROBOTICKÉ VOZIDLO PRO VÝUKOVÉ ÚČELY

MODULAR ROBOTIC VEHICLE FOR EDUCATIONAL PURPOSES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

RADEK JANEČKA

VEDOUCÍ PRÁCE

SUPERVISOR

doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.

BRNO 2025

Zadání bakalářské práce



164494

Ústav: Ústav inteligentních systémů (UITS)
Student: **Janečka Radek**
Program: Informační technologie
Název: **Modulární robotické vozidlo pro výukové účely**
Kategorie: Vestavěné systémy
Akademický rok: 2024/25

Zadání:

1. Prostudujte problematiku robotických systémů. Zaměřte se na malá robotická vozidla. Seznamte se s prostředím ROS2, MicroROS a YARP. Seznamte se s odpovídajícím komerčními robotickými produkty i DIY projekty s dostupným opensource softwarem.
2. V souladu s pravidly Open Robotic Platform navrhnete jednoduše replikovatelné malé robotické vozidlo pro výukové účely. Preferujte levné, standardně dostupné komponenty, snadno realizovatelnou mechanickou konstrukci.
3. Navržený systém realizujte. Vhodně kombinujte dostupné elektronické moduly, mechanické komponenty, 3D tisk, Raspberry Pi, vhodný robotický middleware.
4. Proved'te testování výsledného řešení a demonstруйте ukázkovou realizaci vybraných základních robotických úloh (např. teleoperation, obstacle avoiding). Proved'te celkové vyhodnocení v kontextu podobných existujících řešení.

Literatura:
Dle pokynů vedoucího.

Podrobné závazné pokyny pro vypracování práce viz <https://www.fit.vut.cz/study/theses/>

Vedoucí práce: **Janoušek Vladimír, doc. Ing., Ph.D.**
Vedoucí ústavu: Kočí Radek, Ing., Ph.D.
Datum zadání: 1.11.2024
Termín pro odevzdání: 14.5.2025
Datum schválení: 4.11.2024

Abstrakt

Cílem této práce je vytvořit modulární robotické vozidlo sloužící pro výukové účely s pomocí prostředí ROS 2. Konstrukce robota je navržena pro snadnou změnu typu podvozku a možnost přidání nových senzorů. Robotické vozidlo podporuje vzdálené ovládání. K implementaci robotického systému je použit robotický middleware ROS 2 a ke zpřístupnění hardwaru podvozku a kontrolérů je použit framework `ros2_control`. Modulární návrh konstrukce a softwaru umožňuje snadnou modifikaci robotického vozidla a jeho řídicího softwaru. V řešení byla provedena rešerše existujících výukových robotických vozidel. V práci je navržen model robota MoRVE, který svým přístupem umožňuje provádět změny v konstrukci a výběru senzorů.

Abstract

The aim of this work is to create a modular robotic vehicle for educational purposes using the ROS 2 environment. The robot structure is designed to easily change the type of chassis and the possibility of adding new sensors. The robotic vehicle supports remote control. The ROS 2 robotic middleware is used to implement the robotic system and the `ros2_control` framework is used to access the chassis hardware and controllers. The modular design of the structure and software allows easy modification of the robotic vehicle and its control software. In the solution, a search of existing educational robotic vehicles was performed. In this thesis, a robot model MoRVE is proposed, whose approach allows for changes in the structure and sensor selection.

Klíčová slova

vyhýbání se překážkám, teleoperace, 3D tisk, robotické vozidlo, výuka, Raspberry Pi, ROS 2, `ros2_control`, modulární

Keywords

obstacle avoiding, teleoperation, 3D printing, robotic vehicle, education, Raspberry Pi, ROS 2, `ros2_control`, modular

Citace

JANEČKA, Radek. *Modulární robotické vozidlo pro výukové účely*. Bakalářská práce. Vedoucí práce doc. Ing. Vladimír Janoušek, Ph.D.. Brno: Vysoké učení technické v Brně, Fakulta informačních technologií, 2025.

Modulární robotické vozidlo pro výukové účely

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Vladimíra Janouška Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Radek Janečka
15. května 2025

Poděkování

Rád bych poděkoval svému vedoucímu panu doc. Ing. Vladimíru Janouškovi, Ph.D. za odborné vedení a přínosné rady, které mi pomohly s vypracováním této bakalářské práce.

Obsah

1	Úvod	3
2	Dosavadní znalosti v oblasti robotických výukových vozidel	4
2.1	Výukové robotické vozidlové kity	4
2.2	Typy podvozků	5
2.3	Možnosti řízení kol robota	7
2.4	Senzory využívané v robotech	10
3	Robotický middleware	14
3.1	Robotic Operating System 2	14
3.2	Ros2_control	16
3.3	YARP	18
4	Pochopení zadání	20
5	Způsob konstrukce a hardwarové vybavení robotického vozidla	21
5.1	Konstrukce robotického vozidla	21
5.2	Napájení robotického vozidla	22
5.3	Monitorování stavu baterie	22
5.4	Vypínací tlačítko	23
5.5	Raspberry Pi 4B	24
5.6	Výběr motorů a kol	24
5.7	Rozšiřující motor HAT pro Raspberry Pi	24
5.8	Sonar	25
6	Implementace řídicího softwaru	27
6.1	Uspořádání robotického softwaru	27
6.2	Implementace ovladačů hardwaru	27
6.3	Implementace robotického podvozku	29
6.4	Samostatné uzly zveřejňující data ze senzorů	34
6.5	Spouštěcí skripty	35
6.6	Analýza spuštěného robotického systému	36
7	Závěr	39
	Literatura	40

Seznam obrázků

2.1	Adept AWR robot kit [12]	5
2.2	Všesměrový podvozek	6
2.3	H-můstek, horní polovina je tvořena PMOS tranzistory a spodní polovina NMOS tranzistory	7
2.4	Výstup signálů A a B z enkodéru. Levý graf znázorňuje rotaci proti směru hodinových ručiček. Pravý graf znázorňuje rotaci po směru hodinových ručiček.	9
2.5	Blokové schéma regulační smyčky se zpětnou vazbou a PID regulátorem [22]	10
2.6	Princip Time of Flight senzoru, výsledná vzdálenost překážky je odvozena z naměřeného fázového posunu ϕ přijatého modulovaného světla [5]	11
2.7	Princip mechanického rotačního LIDARu	12
3.1	Konečný automat životního cyklu hardwarové komponenty nebo kontroléru	18
5.1	Vzhled robota MoRVE	21
5.2	Nalevo step-down měnič s XL4005 a napravo voděodolné pouzdro pro 4 lithiové články 18650 s BMS	22
5.3	INA219 proudový senzor od laskakit [8]	23
5.4	Upravený Adafruit DC & Stepper Motor HAT pro Raspberry Pi	25
6.1	Graf zobrazující řízení snahy PID regulátorem ovlivněných různou velikostí průměrovacího okna úhlové rychlosti	29
6.2	Model robota MoRVE, popsany URDF soubory, zobrazený v programu Rviz	36
6.3	Grafy zobrazující reakci PID regulátoru na změnu cílové rychlosti, v prvním grafu je nastavena nastavena nesprávná horní hranice mrtvé zóny na 270 a v druhém případě je mrtvá zóna nastavena na hodnotu 500	38

Kapitola 1

Úvod

V posledních letech s rozmachem umělé inteligence zažívá růst i poptávka po lidech orientujících se v problematice programování robotů společně se základními znalostmi o jejich fungování. K údržbě a vývoji nových robotických systémů si firmy vychovávají odborníky v robotice, protože neexistuje mnoho odborníků v dané oblasti. Roboty v dnešní době lze najít ve skladištích, v automatizovaných linkách, vývoj robotických protéz pro lidi. Roboty jsou nasazovány i na těžko dostupných místech či životu ohrožujících místech. V lékařství pro snížení rozsahu zákroku a k možným operacím na dálku.

Začlenění výuky s roboty nemusí být jednoduché, ať už jde o nedostatek financí na drahé stavebnice nebo nedostatek zkušených pedagogů. Využití robotů ve výuce seznámí žáky s možností se dále rozvíjet tímto směrem i ve svém budoucím studiu nebo zaměstnání.

K vybrání této práce mě vedla touha vytvořit hmotné dílo, co přesahuje hranici virtuálního světa.

Bakalářská práce si klade za cíl vytvořit modulární robotické vozidlo. Modulární je myšleno ve smyslu možnosti změny použitých senzorů tak i možností rekonfigurace robotického softwaru. Robot má zvládnout základní robotickou úlohu teleoperace.

Druhá kapitola obsahuje rešerši existujících řešení, po ní následují potřebné znalosti z oblasti robotiky k řízení hardwaru a se senzorů. Třetí kapitola popisuje robotické middlewary. Čtvrtá kapitola popisuje pochopení zadání. Pátá a šestá kapitola se zabývá postupně konstrukcí, použitými senzory a popisem robotického systému. Závěr shrnuje dosažené výsledky.

Kapitola 2

Dosavadní znalosti v oblasti robotických výukových vozidel

Obsahem této kapitoly jsou příklady dvou komerčních a dvou otevřených robotických kitů. Následující podkapitoly se zabývají typy podvozků, možnostmi řízení s typem kol a základním regulátorem, dále běžně používané senzory v robotech.

2.1 Výukové robotické vozidlové kity

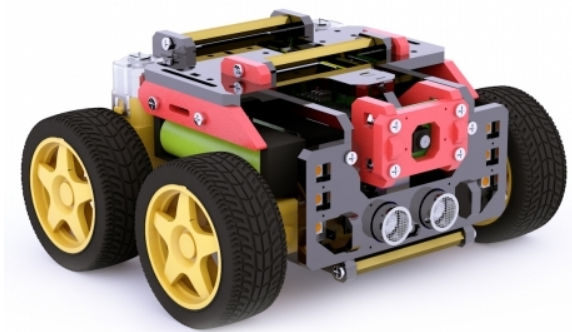
Tato podkapitola uvádí příklady robotických vozidel, které existují a do jisté míry splňují téma výukového robota. Velice zajímavým konceptem v oblasti návrhu je pokus o udělení otevřeného standartu pro konstrukční návrh robotů a jejich komponent s názvem Open Robotic Platform.

Adept AWR 4WD

AWR (Adept Wheeled Robot) je robotické vozidlo používající Raspberry Pi jako řídicí jednotku. Robotické vozidlo disponuje čtyřmi stejnosměrnými motory bez enkodérů osazených koly s pneumatikami. Robotický podvozek může fungovat pouze v konfiguraci diferenciálního řízení podvozků. Podvozek nepodporuje nezávislé řízení kol, protože kola na pravé (respektive levé) straně jsou řízena pomocí totožného H-můstku. Nelze přímo měřit úhlovou rychlost motorů, protože nemají žádnou zpětnou vazbu z enkodérů. Pro řízení robota, aby jel rovně je třeba senzoru IMU (Intertial Measurement Unit) – akcelerometru. Na robotovy je osazen jedním ultrazvukovým sonarem, kamerou a senzorem pro sledování čáry. ukázka tohoto robota je na obrázku 2.1.

Tortoisebot

Jedná se o robotický kit využívající Raspberry Pi pro řízení diferenciálního robota. Dostupný software je psaný pro ROS 2 verzi Humble. Pro fungování robotického softwaru je tedy nutné na Raspberry Pi nainstalovat operační systém Ubuntu server 22.04 LTS. K pohonu využívá dvou motorů bez enkodérů v diferenciálním postavením. Využívá senzorů jako IMU, Lidar a kamery. Dokáže mapovat a navigovat se prostorem. Z výukového hlediska přináší možnost přidávání, obměňování senzorů za jiné, avšak nemá standardizované otvory pro uchycení a je nutné vyrobit vlastní uchycení pro senzor. Robot má dokumentaci, dostupnou na odkaze [16], pro sestavení robota a potřebných komponent.



Obrázek 2.1: Adept AWR robot kit [12]

Robot dle Open Robotic Platform pravidel

Nikodem Bartnik vytvořil soubor pravidel, dle kterých lze navrhovat robotický základový plát se standardizovanými otvory o průměru 3,5 mm vhodné pro šrouby M3, otvory tvoří mřížku o rozměrech 20×20 mm. Otvory lze rozšířit pro větší šrouby. Informace o pravidlech ORP lze najít na stránce [1]. K této mřížce lze přichytit jakýkoli úchyt pro senzor navržený s ohledem na ORP (Open Robotic Platform). Základový plát je možné vyrobit pomocí CNC stroje, laserové řezačky nebo 3D tisku. Zmínka o vytvoření robota dle těchto pravidel je na uvedeném zdroji [3]. Robotická konstrukce je vytvořena z plátů kulatého tvaru dle ORP návrhových pravidel. Je navržena tak, aby byla vhodná pro diferenciální řízení robota. Na kostru robota je možné připevnit jakýkoli senzor i mikrokontrolér nebo mikropočítač jako například Raspberry Pi.

Rosbot 3

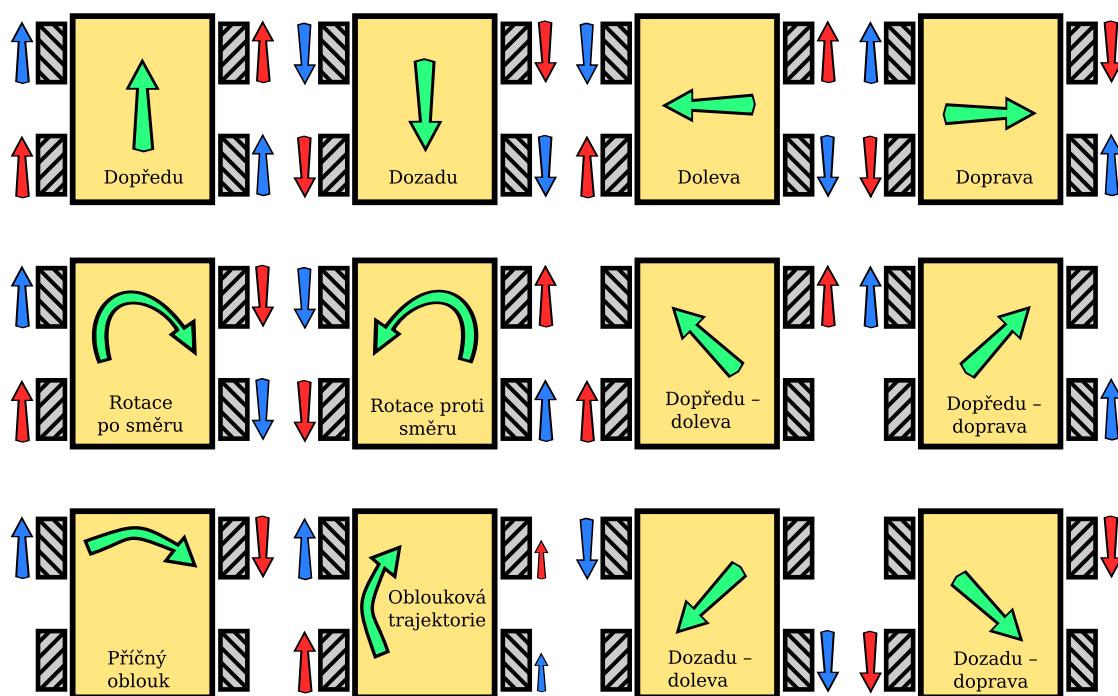
Je jedním z komerčních robotů pro vnitřní použití, který je primárně stavěn pro použití s ROS 2 middlewarem. Jedná se o robot vyráběný firmou Husarion se zaměřením na výzkum a výuku. Nabízí mnoho funkcí, nabíjení tříčlánkové lithiové baterie pomocí externí nabíječky s vyváženým nabíjením baterií přes dedikovaný konektor, podvozek se čtyřmi DC motory společně s enkodéry. Nabízí senzory jako jsou LIDAR, hloubkovou kameru společně s barevnou kamerou, IMU BNO055. Cena se pohybuje od 2 749,00 EUR. Řídící systém lze spustit a psát na Raspberry Pi 5. Dokumentace je dostupná z oficiálních stránek výrobce [14].

2.2 Typy podvozků

Tato podkapitola popisuje několik druhů podvozků vhodných k realizaci pohybu robota.

Diferenciální podvozek

Diferenciální podvozek je jedním z nejčastěji používaných podvozků u robotických vozidel. I přes svou jednoduchou konstrukci zvládne tento podvozek složitější manévry. Dokáže se otočit na místě, zatáčet doleva, doprava, anebo pohyb vpřed a vzad. Pro konstrukci podvozku mohou být použity dvě kola s dvěma opěrnými body, či dva pásy, nebo čtyři kola.



Obrázek 2.2: Všesměrový podvozek

Všesměrový podvozek

Existují dva typy všesměrových podvozků dle typu kol. Všesměrová kola tvoří sada válečků, umístěných na obvodu kola, kolmých na axiální směr pohybu kola. To umožňuje kolu prokluzovat ve směru kolmém na axiální směr pohybu kola. V konfiguraci tří kol jsou kola umístěna ve středech stran rovnostranného trojúhelníku. Kola konfigurace čtyř kol se nachází na středech stran čtverce.

Mecanum kola jsou speciálním typem všesměrových kol. Skládají se z válečků, které jsou natočeny pod úhlem 45° od axiálního směru pohybu kol. Existují dvě konfigurace těchto kol, kdy jsou válečky natočeny nalevo anebo napravo. Pro správnou funkci podvozku je potřeba obou konfigurací kol. Různou rychlostí a směrem rotace jednotlivých kol se dá docílit všesměrového pohybu. Na obrázku 2.2 je vyobrazen všesměrový podvozek skládající se z mecanum kol.

Ackermanův podvozek

Ackermanův podvozek se skládá ze 4 kol. Typicky k pohybu vpřed a vzad využívá pohonu dvou zadních kol. Přední kola slouží k zatáčení ve směru natočení kol. V běžném provozu nedochází k prokluzu kol s podlahou. Nedokáže se otočit na místě.

Kráčející robot

Kráčející robot ke svému přemísťování využívá kráčejících nohou. Ke svému pohybu využívá kráčející robot minimálně jeden pár nohou a maximální počet párů nohou není omezen. V praxi však počet párů nohou často nepřevyšuje tři páry. Na rozdíl od všech předchozích

podvozků kráčející robot užívá ke svému pohybu obzvláště servomotorů, které disponují velkým točivým momentem a přesností natočení o potřebný úhel.

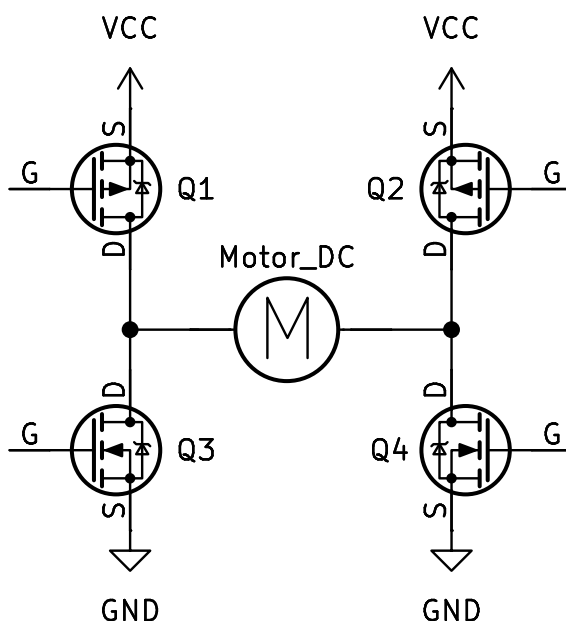
2.3 Možnosti řízení kol robota

V této podkapitole se zmiňují základní druhy motorů a způsob jejich řízení. Dále je zde zmíněn PID (proporční, integrační, derivační) regulátor, který umožňuje řídit motory dle rychlosti nebo pozice namísto ovládání střídavého signálu generovaného pomocí pulzně šířkové modulace (PWM).

Stejnoseměrný motor s enkodérem

V robotice mikrokontrolér nebo mikropočítač ovládá směr rotace stejnosměrného motoru, ale ne přímo, protože mikrokontrolér na svých výstupech nedokáže poskytnout požadované napětí a proud pro provoz motoru. A dalším důvodem je velká indukčnost motorů, která by v určitých případech dokázala mikrokontrolér zničit. Aby se zabránilo poškození a motor měl potřebný výkon, používá se tzv. **H-můstku** vyobrazeného na obrázku 2.3.

Jedná se o relativně jednoduchý obvod, jehož název je odvozen z podoby schématu můstku připomínající písmeno H, který se skládá ze čtyř tranzistorů. H-můstky je možné sestavit z bipolárních tranzistorů, nebo tranzistorů s indukovaným kanálem založených na technologii MOSFET [6]. Na bipolárních tranzistorech dochází k úbytku napětí ze zdroje vyššího napětí, s tím souvisí i jejich menší účinnost a tak zde nebudou rozebrány do detailu. Úbytek napětí na bipolárním H-můstku činí až 1,4 V. Není tedy příliš vhodný pro systémy napájených bateriemi. U tranzistorů technologie MOSFET, stačí přivést napětí nebo zem pro jejich otevření, proud z mikrokontroléru neteče.



Obrázek 2.3: H-můstek, horní polovina je tvořena PMOS tranzistory a spodní polovina NMOS tranzistory

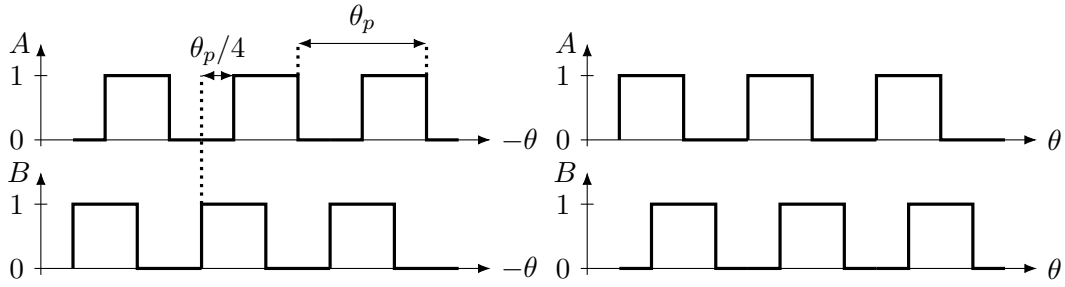
Každý integrovaný obvod H-můstku disponuje minimálně těmito módy:

- **Rotace po směru hodinových ručiček** – V tomto stavu jsou sepnuty tranzistory Q1 a Q4, ostatní jsou otevřeny.
- **Rotace proti směru hodinových ručiček** – Rotace motoru v opačném směru se docílí sepnutím tranzistorů Q2 a Q4. Proud teče motorem v opačném směru než v předchozím stavu.
- **Volnoběh (fast decay, coast)** – Volnoběh nastává v případě, že motorem protékal proud a je mu umožněno chvilkově nabíjet baterii. Tento proud chvilkově protéká přes externí diody anebo otevřením tranzistorů v opačném směru po dobu vybíjení proudu. V tomto stavu dochází k rychlému poklesu proudu na motoru, od toho se odvíjí anglický název fast decay. Nicméně rychlý pokles proudu neznamená brzdění motoru, ale pozvolné zpomalování [4].
- **Brzda (slow decay)** – Brzdění nastane při sepnutí dvou spodních tranzistorů Q3 a Q4. Motor, který byl v pohybu začne generovat zpětnou elektromagnetickou sílu na motor a tím sám sebe efektivně ubrzdí [21].
- **Zkrat (shoot through)** – Ke zkratu dochází v okamžiku, kdy dojde k sepnutí dvou tranzistorů na stejné straně H-můstku, např. Q1 a Q3 tranzistory jsou sepnuty. V tomto stavu nemají tranzistory téměř žádný odpor, dojde k toku velkého proudu ze zdroje napětí, k téměř okamžitému zničení H-můstku a možnému poškození zdroje napětí, pokud se jedná o lithiový zdroj. Této situaci by měl každý integrovaný obvod předcházet v podobě vložení mrtvého času (dead time) mezi komplementární PWM signály [6].

Pulzně šířková modulace nebo zkráceně **PWM** se využívá pro generování různé úrovně napětí ze zdroje s konstantní hodnotou napětí. Jedná se o obloukový signál, pro který známe periodu jeho opakování. Skládá se z části signálu, kdy je v maximální hodnotě a kdy má nulovou hodnotu. Měněním poměru těchto dvou složek signálu, se efektivně mění výsledné napětí tohoto signálu. Vyšší frekvencí signálu se docílí přesnějšího signálu o určitém napětí. Ale při ovládání H-můstku dochází kvůli vyšším frekvencím k většímu zahřívání integrovaného obvodu, kvůli častějšímu spínání a otevírání tranzistorů.

Protože neexistuje lineární závislost středy PWM signálu, kdy je PWM signál ve vysoké úrovni a rychlosti motoru. Tak pro řízení rychlosti motoru nestačí pouze znát střidu PWM signálu, ale je třeba měřit i skutečné otáčky motoru a cíleně měnit střidu PWM signálu pro jejich udržení. Proto se užívá inkrementálního snímače otočení neboli inkrementálního enkodéru.

Inkrementální enkodér poskytuje informace o relativní pozici natočení hřídele motoru. Slouží pro měření otáček motoru, následné odvození výsledné přímočaré rychlosti, které poskytuje kolo připevněné k motoru. Enkodér pro svou funkci využívá mechanické, magnetické či optické senzory [7]. Na výstupu těchto senzorů je periodický obdélníkový signál, jehož perioda T odpovídá době otočení hřídele o úhel jednoho kroku θ_p . Úhel θ_p lze odvodit z počtu pulsů generovaných enkodérem za jednu otočku hřídele enkodéru, pokud tímto počtem podělíme plný úhel dostaneme hodnotu úhlu θ_p . Z doby trvání jedné periody lze odvodit rychlost rotace, u enkodérů s velkým počtem kroků za jednu otočku, se využívá počet kroků za jednotku času pro určení rychlosti. Z jednoho periodického signálu není možné odečíst směr rotace motoru, ale pouze počet uražených kroků. Inkrementální enkodér tedy potřebuje dva výstupní periodické signály A a B, které jsou od sebe fázově posunuty o $\theta_p/4$. A směr rotace určuje zda fáze signálu A předbíhá signál B, nebo naopak.



Obrázek 2.4: Výstup signálů A a B z enkodéru. Levý graf znázorňuje rotaci proti směru hodinových ručiček. Pravý graf znázorňuje rotaci po směru hodinových ručiček.

Některé inkrementální rotační enkodéry mohou mít ještě třetí výstup Z, který vyše signál v momentu, kdy se nachází ve výchozí pozici natočení [7], [23].

Pro určení směru rotace lze použít okamžik sestupné hrany signálu A. Motor se otáčí po směru hodinových ručiček, když v okamžiku sestupné hrany na výstupu A je výstup B v logické 1. Naopak pokud se výstup signálu B nachází v logické 0 při sestupné hraně signálu A, tak se motor otáčí proti směru hodinových ručiček [2]. Takže v okamžiku sestupné, nebo vzestupné hrany jednoho signálu a stavu toho druhého signálu je možné odečítat z enkodéru kroky rotace v obou směrech. Signály A a B jsou znázorněny na obr. 2.4.

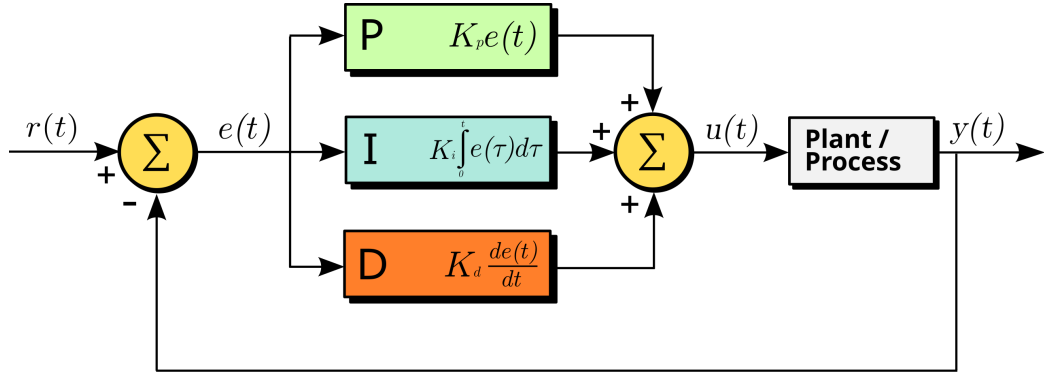
Regulační smyčka se zpětnou vazbou

Pro přesné řízení rychlosti rotace, pozice motoru lze využít regulační smyčku se zpětnou vazbou. Regulační smyčka funguje na principu toho, že se snaží vyrovnat funkční hodnotu funkce $r(t)$, která představuje chtěný stav na funkci výstupu $y(t)$ z netriviálního procesu. Pro jeho řízení se používá řídicí člen na jehož vstupu je difference $e(t)$ mezi vstupní referenční hodnotou $r(t)$ a výstupní hodnotou $y(t)$. Rovnice 2.1 lze využít pro výpočet chyby $e(t)$. Výstupem regulačního členu je řídicí funkce $u(t)$, která řídí netriviální proces jako je např. řízení otáček motoru.

Nejběžněji používaným typem regulátoru je **PID regulátor**. PID je zkratka pro proporční, integrační, derivační regulátor. PID regulátor představuje spojitá rovnice 2.2. Skládá se ze tří částí:

- **proporční** – Je okamžitou reakcí na aktuální diferencí $e(t)$. K_p slouží jako proporční koeficient a ovlivňuje míru vlivu na řídicí signál. Řídicí signál je přímo úměrný s aktuální diferencí.
- **integrační** – Reaguje postupně na akumulovanou chybu pod plochou grafu chyby $e(t)$. Pro ovlivnění působení integrační složky PID regulátoru se využívá integračního koeficientu K_i .
- **derivační** – Derivační část regulátoru předpovídá budoucí vývoj chyby $e(t)$. Hlavní úlohou je zabránit překmitům výstupního signálu $y(t)$ přes stanovenou mez referenčního signálu $r(t)$. K nastavení derivační složky PID regulátoru slouží derivační koeficient K_d .

$$e(t) = r(t) - y(t) \quad (2.1)$$



Obrázek 2.5: Blokové schéma regulační smyčky se zpětnou vazbou a PID regulátorem [22]

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (2.2)$$

Pro snadnější pochopení je zde uvedeno blokové schéma regulační smyčky se zpětnou vazbou na obr. 2.5. V ustáleném stavu, kdy se referenční hodnota funkce $r(t)$ rovná hodnotě funkce výstupu $y(t)$, je hodnota chyby $e(t)$ nulová a řídicí funkce $u(t)$ se ustálila na potřebné hodnotě.

Pro implementaci proporčně integračně derivačního regulátoru v reálném systému robota je spojitý regulátor nepraktický, a proto se využívá v diskrétní podobě. Spojitý čas t , řídicího signálu a chyby, je vzorkován na diskrétní časové kroky t_k se vzorkovací periodou Δt .

Existují dva přístupy aproximace PID kontroléru. První možností je využít k výpočtu řídicího signálu $u(t_k)$ geometrického významu integrálu a derivace z původně spojitě rovnice 2.2 jejím navzorkováním a odvozením vztahů. Druhou nejpoužívanější možností ve vestavěných systémech je rovnice využívající hodnoty současné chyby $e(t_k)$ a minulých hodnot chyby $e(t_{k-1})$, $e(t_{k-2})$ a minulou hodnotu řídicího signálu $u(t_{k-1})$. Spojitou rovnici PID regulátoru 2.2 lze derivací a za pomoci Newtonovy notace zapsat jako rovnici 2.3. Pro získání výsledné diskrétní podoby rovnice PID regulátoru je za každou derivaci i nově vzniklou dosazeno ze vzorce 2.4. Po všech úpravách vznikne diskrétní rovnice 2.4 PID regulátoru. Informace v tomto odstavci byly převzaty z [22].

$$\dot{u}(t) = K_p \dot{e}(t) + K_i e(t) + K_d \ddot{e}(t) \quad (2.3)$$

$$\dot{f}(t) \approx \frac{f(t_k) - f(t_{k-1})}{\Delta t} \quad (2.4)$$

$$u(t_k) = u(t_{k-1}) + \left(K_p + K_i \Delta t + \frac{K_d}{\Delta t} \right) e(t_k) - \left(K_p + \frac{2K_d}{\Delta t} \right) e(t_{k-1}) + \frac{K_d}{\Delta t} e(t_{k-2}) \quad (2.5)$$

2.4 Sensory využívané v robotech

Tato podkapitola uvádí příklady senzorů užívaných v robotice. Sensory umožňují robotům vnímat a mapovat své okolí. Pro měření hloubky se v robotice především využívá principu

Time of Flight neboli doby letu vlny zvukové či elektromagnetické v podobě světla. Sensory užívající principu ToF jsou popsány níže. Patří mezi ně ultrazvukový sonar, laserové hloubkové senzory se stejnojmenným názvem Time of Flight nebo nazývané jako LIDAR pro svou schopnost mapovat prostor ve 2D nebo 3D. Kamera může být využita k téměř jakémukoliv účelu, je vhodná pro teleoperaci robota na delší vzdálenosti. V neposlední řadě senzor IMU (Inertial Measurement Unit) obsahuje akcelerometr s gyroskopem a v dražších variantách i kompas.

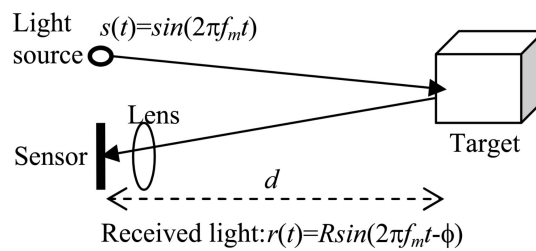
Sonar

Sonar je ultrazvukový hloubkový senzor využívající ultrazvuku pro měření vzdálenosti. Disponuje přijímačem a vysílačem ultrazvukových vln o frekvenci v řádech kHz. Doba letu ultrazvuku k překážce a od překážky zpět k sonaru je měřena a převedena na vzdálenost za pomoci rychlosti šíření zvuku ve vzduchu.

Mezi jedny z nejpoužívanějších sonarů v DIY projektech se řadí **HC-SR04**, který lze najít v mnoha podobách. Senzor využívá čtyř pinů z nichž dva slouží pro napájení a zbylé dva nesou název Trigger a Echo. Pro změření vzdálenosti mikrokontrolér uvede pin Trigger na vysokou úroveň po dobu 10 μ s. Po tomto okamžiku sonar vyšle osm pulzů ultrazvuku a na výstupu Echo je možné změřit délku trvání vysoké úrovně, z které je odvozena změřená vzdálenost [13].

Time of Flight senzor

Typický senzor pro měření vzdálenosti využívající nepřímého principu **time of Flight** se skládá ze zdroje modulovaného světla jako je laser nebo LED a mřížky pixelů, z nichž každý je schopen detekovat fázový posun odraženého světla od překážky. Součástí takového senzoru může být i systém čoček soustřeďujících odražené světlo do senzoru. Příklad time of Flight senzoru je zobrazen na obr. 2.6 [5]. Modulační vysílání světla probíhá rychlým zapínáním a vypínáním zdrojového světla. Naměřená vzdálenost senzorem vychází z naměřeného fázového posunu ϕ modulační přijatého světla od modulační původně vyslaného světla.



Obrázek 2.6: Princip Time of Flight senzoru, výsledná vzdálenost překážky je odvozena z naměřeného fázového posunu ϕ přijatého modulovaného světla [5]

Pro zjednodušení představy světla jako signálu s určitou frekvencí modulační f_m je použit sinusový signál na rozdíl od skutečného obloukového signálu. Rovnice vyslaného světla je dána rovnicí $s(t) = \sin(2\pi f_m t)$.

Světlo odražené od překážky dopadá na pixel senzoru s fázovým posunem ϕ , je popsáno rovnicí 2.6. Kde R je amplituda odraženého světla, d je vzdálenost od senzoru k překážce

a rychlost světla značena jako c s hodnotou 3×10^8 m/s. Vzdálenost d od překážky lze spočítat ze vzorce 2.7. Maximální jednoznačná vzdálenost $d_{max} = \frac{c}{f_m}$, kterou s tímto senzorem lze změřit, závisí na modulační frekvenci f_m . Celý tento úsek o Time of Flight senzoru je převzat z [5].

$$r(t) = R \sin(2\pi f_m t + \phi) = R \sin\left(2\pi f_m \left(t - \frac{2d}{c}\right)\right) \quad (2.6)$$

$$d = \frac{c\phi}{4\pi f_m} \quad (2.7)$$

LIDAR

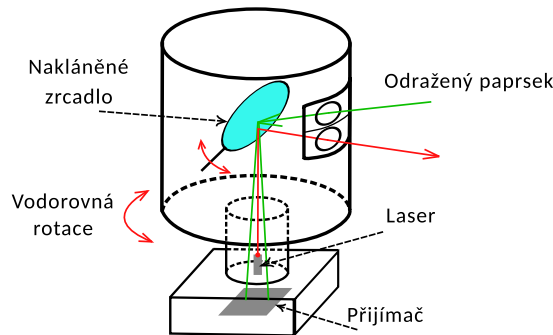
LIDAR neboli Light Detection and Ranging je systém, který lze využít na sbírání prostorových, spektrálních a dočasných informací [11]. V robotice nás zajímají především prostorové a dočasné informace o prostoru neboli pohybující se objekty v prostoru.

Systém LIDARu se skládá z mechanické části, samotného senzoru fungujícího na principu ToF, kontroléru řídící činnost mechanické části a umožňující komunikaci s řídícím počítačem přes sběrnici.

Konstrukce mechanické části LIDARu se může lišit. Existuje provedení, kdy je senzor umístěn na disku poháněném motorem a za provozu se otáčí za účelem skenování prostoru. Druhá možnost obnáší umístění laserového senzoru staticky a za pomoci otáčení zrcadla kolem své osy umožňuje skenovat prostor ve vodorovné rovině a dodatečným nakláněním zrcadla lze docílit skenování prostoru ve 3D. Na obrázku 2.7 je ukázka konstrukce mechanického rotačního LIDARu se zrcadlem, jeho předlohou byl obrázek v tomto článku [9].

Data získaná ze systému LIDAR jsou interpretována jako mračno bodů (point cloud). Každý bod v takovém mračnu bodů charakterizují jeho souřadnice v prostoru.

LIDAR je velice používaným systémem mezi senzory pro svou schopnost skenovat prostor v reálném čase ve větším detailu než obyčejné senzory hloubky. A tím umožňující rychlejší mapování a orientaci v neznámém prostředí.



Obrázek 2.7: Princip mechanického rotačního LIDARu

Kamera

Kamera je jedním z nejvšestrannějších robotických senzorů. Pouhý obraz může umožnit operátorovi robota řídit na velké vzdálenosti bez nutnosti následovat robota fyzicky. Obraz z kamery může být společně s umělou inteligencí použit např. k udržení přímého kurzu

robota, detekci objektů ve scéně a jejich následování, vyhýbání se překážkám, předcházení přepadnutí robota z hrany stolu, sledování čáry.

IMU

Senzor IMU (Inertial Measurement Unit) je obvykle složen z akcelerometru, gyroskopu a kompasu. Akcelerometr lze využít k detekci nárazu, ke zjištění relativního natočení od počátečního natočení, měření natočení je náchylné na akumulované chyby v měření. Gyroskopem lze měřit naklonění robota, zda se pohybuje po rovině nebo nakloněném povrchu. Pokud je k dispozici i kompas (magnetometr), lze absolutní natočení robota určit vzhledem ke světovým stranám.

Kapitola 3

Robotický middleware

V této kapitole jsou uvedeny dva robotické middlewary, které se využívají pro usnadnění řešení komunikace částí robotického systému po síti. Umožňují distribuci výpočetní složitosti mezi více počítači.

Tři hlavní důvody vzniku robotických middlewarů:

1. **Dekompozice** – Umožnit rozdělení robotického systému na podsystémy starající se o jeden úkol, které spolu komunikují. Toto dělení umožňuje vývojářům převzít části podsystému, které se u robotů opakují a nemusejí pokaždé při programování nového robota začínat zcela od nuly. Též toto rozhodnutí dekompozice umožňuje sdílet vývojářům kód mezi sebou.
2. **Abstrakce hardwaru** – Hardware může zastarávat, pro oddělení hardwarových ovladačů od zbytku robotického softwaru se využívá tzv. hardwarových rozhraní. Ty poskytují jednotné metody pro přístup k hardwaru. Při změně hardwaru stačí nahradit hardwarové ovladače a upravením hardwarového rozhraní, aby využívalo nový hardware je výměna dokončena.
3. **Asynchronní komunikace** – Události, které musí robotický systém řešit nastávají v neočekávaný okamžik. Pro jejich zpracování je nutnost existence mechanismů poskytujících možnost reagovat na tyto události asynchronně a nezávisle na ostatních podsystémech.

3.1 Robotic Operating System 2

Tato kapitola čerpá informace primárně z oficiální dokumentace ROS 2 [17] a článku [10]. Jedná se o robotický middleware ROS 2. V této kapitole jsou uvedeny koncepty middlewaru ROS 2. Srdcem ROS 2 systému je síť propojených uzlů (nodes) spolu komunikujících pomocí témat (topics), služeb (services) a akcí (actions), které spolu dohromady tvoří ROS graf.

Uzly

Tvoří základní prvky ROS grafu, zaujímají postavení základních stavebních prvků robotického systému. Každý uzel by měl ideálně řešit jeden problém robotického systému najednou. Uzly spolu komunikují s využitím ROS client library ve stejném procesu, mezi dvěma procesy, nebo i jiným počítačem na lokální síti.

Každý uzel může využívat ke komunikaci s ostatními uzly tři komunikačních protokolů a to témat, služeb a akcí, využívajících rozhraní (interfaces) pro sjednocení podoby zpráv posílaných mezi uzly. Dále uzly mohou poskytovat nastavitelné parametry, jejichž změnou lze docílit změny chování uzlu i v momentu, kdy je spuštěn.

Síťová rozhraní

Síťová rozhraní v ROS 2 jako zprávy, služby a akce jsou definovány zjednodušeným popisným jazykem – **Interface Description Language** (IDL). Z tohoto jednoduchého popisu síťového rozhraní je generován zdrojový kód síťového rozhraní potřebný ke komunikaci uzlů mezi sebou. Dochází ke generování verzí zdrojového kódu ve více cílových programovacích jazycích. Rozhraní zprávy je definováno pro komunikaci přes pojmenované téma, slouží pouze pro přenos informace jedním směrem bez potvrzení. Jednotlivé pole rozhraní tvoří jednoduché i složené datové typy. Jednotlivé části rozhraní oddělují tři spojovníky stojících na samostatném řádku bez mezer. Rozhraní pro služby se skládá z požadavku a odpovědi. Akční rozhraní definuje tři části – cíl, zpětnou vazbu a výsledek.

Témata

Mezi nejzákladnější typ komunikace mezi uzly patří asynchronní komunikace přes pojmenovaná **témata** (topics), který je podobný principu broadcast. Do pojmenovaných témat může publikovat jeden nebo více uzlů (publishers) zprávu s přesně daným rozhraním zprávy typu *msg* (message). Zprávy z pojmenovaných témat může odebírat mnoho uzlů (subscribers). Tento způsob komunikace umožňuje propojit přes téma uzel určen pro inspekci přenášených dat a získaná data analyzovat nebo zaznamenat pro pozdější využití.

Při přenášení zpráv uzly neřeší, od kterých uzlů publikujících data odebírají. Lze tedy snadno změnit uzel publikující data za druhý uzel publikující do téhož tématu.

Služby

Službou v ROS 2 middlewaru je myšleno vzdálené volání procedury. Tato procedura provádí výpočet za klienta a vrací výsledek. Po odeslání požadavku na server poskytující službu klient obvykle aktivně čeká než dorazí výsledek. Jedná se v podstatě o synchronní typ komunikace. U tohoto typu komunikace je tudíž žádoucí, aby uzel poskytující službu odeslal odpověď na žádost v krátkém čase. U požadavku, který vyžaduje dlouhodobou komunikaci mezi klientem a službou se využívá komunikace v podobě akcí. Rozhraní zprávy služby je rozděleno na část žádosti a odpovědi. Pokud službu se stejným jménem poskytuje více serverů najednou, není definováno, který z nich na žádost odpoví. Služebních klientů dotazujících se na stejný server poskytující službu je možné mít nekonečně mnoho.

Akce

Akce v ROS 2 reprezentuje volání vzdálené procedury vykonávající dlouhotrvající úkol, od které je potřeba dostávat průběžný stav z jejího provádění a je možné ji v průběhu provádění přerušit nebo je nutné ji přerušit v důsledku neproveditelnosti úkolu. Jedná se o formu komunikace kombinující synchronní komunikaci s asynchronní komunikací. Rozhraní zprávy akce se skládá z cíle, výsledku a zpětné vazby. Komunikační kanál tvoří dvě entity akční klient a akční server. Stejně jako u témat a služeb není definováno chování v případě existence více akčních serverů se stejně pojmenovanou akcí, které čekají na požadavek od akčního

klienta či klientů. Akční server musí v průběhu posílat zprávy se zpětnou vazbou o průběžném stavu procesu a být schopen reagovat na předčasnou žádost o ukončení akce před jejím dokončením. Akčních klientů naopak může být na stejné pojmenovanou akci napojeno mnoho. Akční klient v průběhu komunikace po odeslání požadavku dostává zpětnou vazbu od akčního serveru a pro ukončení komunikace dostane odpověď s koncovým stavem.

Parametrizace uzlů

ROS 2 uzly v sobě mají zabudovanou funkcionalitu pro definování parametrů. Ty umožňují měnit chování uzlu i po jeho spuštění podobným způsobem jako volání na služby klientem. Jde o synchronní komunikaci mezi parametrovým serverem a klientem. Klient se dotáže na server a následně dostane odpověď od serveru. Pro zjištění dostupných parametrů je nutné je nejdříve nechat vypsat pomocí ROS 2 konzolového nástroje. S již zjištěným názvem uzlu a parametru, který je dostupný, lze získat aktuálně nastavenou hodnotu parametru, anebo ji také změnit.

Systém s více roboty v jedné síti

Publikování dat do jednoho pojmenovaného tématu může být problematické, toto chování nemusí být vždy žádoucí. V případě, kdy se snažíme na stejné síti zprovoznit robota se stejnými názvy témat i ostatních zpráv dojde k nepředvídatelnému chování systému. Této situaci lze předejít využitím prefixu, který se přidá před všechny dostupné témata, služby a akce pro oddělení od ostatních robotických systémů v tomto případě mezi sebou robotické systémy mohou komunikovat. Pro oddělení komunikace lze využít proměnné prostředí `ROS_DOMAIN_ID`. Uzly, které byly spuštěny v prostředí se stejnou hodnotou proměnné prostředí `ROS_DOMAIN_ID` spolu mohou komunikovat. V případě opaku spolu nemohou navázat spojení ve stejné lokální síti. S výběrem správné hodnoty `ROS_DOMAIN_ID` pomůže dokumentace ROS 2 [18].

3.2 Ros2_control

Jedná se o framework založený na ROS 2 s tím rozdílem, že umožňuje napsat modulární řízení podvozku skládající se z více vrstev, tyto vrstvy mezi sebou komunikují synchronně na rozdíl od ROS 2 uzlů, jejichž hlavní komunikace přes pojmenované témata probíhá asynchronně. Cílem frameworku `ros2_control` je usnadnit integraci hardwaru do existujícího robotického systému. Funkcionalitu řízení implementuje každý robot a tak vznikl tento framework. `Ros2_control` zpřístupňuje hardwarová rozhraní a řídí životní cyklus kontrolérů. Informace v této kapitole jsou čerpány z `ros2_control` dokumentace [19].

Controller Manager

Controller Manager (CM) propojuje kontroléry (controllers) s abstrakcí hardwaru a jejich ovladačů (drivers). Ke spuštění CM se využívá spustitelného souboru `ros2_control_node`, který spustí CM jako jeden uzel z balíčku `controller_manager` nebo je možné vytvořit vlastní konfiguraci, spouštěnou z jiného programu. Kontroléry i hardwarové komponenty jsou implementovány jako pluginy, které jsou načteny dynamicky za běhu CM, tento mechanismus umožňuje snadno měnit podobu řídicí smyčky, jak v použitém hardwaru, tak jeho kontroléry.

CM spravuje životní cyklus kontrolérů a jejich rozhraní, které vyžadují. K hardwarovým komponentám naopak přistupuje CM pomocí Resource Manager, který zpřístupňuje hardwarová rozhraní. CM spojuje poskytnutá hardwarová rozhraní s vyžadovanými rozhraními kontrolérů, kontrolérům uděluje přístup k hardwaru při jejich aktivaci. Uzel CM se zpravidla spouští s konfiguračním souborem `config.yaml`, který upravuje například periodu řídicí smyčky. Dále tento konfigurační soubor obsahuje informace o používaných kontrolérech a jejich používaných rozhraních. CM spravuje provedení řídicí smyčky v metodě `update()`. Při jejímž provádění nejdříve dojde k přečtení dat z hardwarových komponent, aktualizaci výstupů na všech kontrolérech a na závěr zapíše příkazy do hardwarových komponent.

Je hlavním uzlem, který se stará o celý řídicí systém, stará se o načtení hardwaru z pojmenovaného tématu od uzlu `robot_state_publisher` a navázání kontrolérů na zpřístupněná hardwarová rozhraní.

Resource Manager

Resource Manager (RM) spravuje životní cyklus hardwarových komponent. Resource Manager získává informace o potřebných hardwarových komponentách z pojmenovaného tématu `/robot_description`, do kterého publikuje data `robot_state_publisher` ze stejnojmenného balíčku. Konkrétně přes toto pojmenované téma se předává popis z souboru ve formátu URDF (Unified Robot Description Format), který využívá nový tag `<ros2_control>` obsahující název komponenty, její stavové a řídicí názvy rozhraní a parametry komponenty a jejich jednotlivých kloubů (joints). Pro správnou funkci musí soubor definovat klouby se stejným názvem jako v tagu `<ros2_control>`.

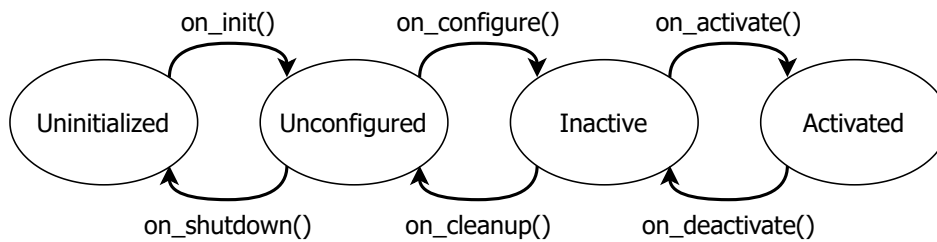
Pokud vše sedí a CM požádá o přístup k hardwarové komponentě, pak RM předá přístup CM. Jméno každého stavového i příkazového rozhraní je vybráno dle dat, které se na rozhraní očekávají, například rychlost, pozice.

Hardwarové komponenty

Hardwarové komponenty komunikují s hardwarem pomocí metod `read()` a `write()`. Hardwarové komponenty jsou psány jako plugin sloužící k abstrakci hardwarových ovladačů, kompatibilních s `ros2_control` frameworkem.

Existují tři typy hardwarových komponent:

1. **Aktuátor** – Hardwarová komponenta implementovaná jako aktuátor se vztahuje pouze k jednomu kloubu (joint). Aktuátor představuje pouze jeden motor, nebo servomotor, který je ovládán. Hardwarová komponenta typu aktuátor poskytuje pouze příkazové rozhraní. Neposkytuje stavové rozhraní.
2. **Senzor** – Senzor poskytuje stavové rozhraní, ze kterého je čten stav hardwaru. Příkladem senzoru je enkodér, senzor teploty na motoru. . . Senzor se také vztahuje pouze jednomu kloubu (joint).
3. **Systém** – Systém tvoří hardwarovou komponentu jak se stavovými, tak i příkazovými rozhraními. Systém je nejpoužívanějším typem, protože je možné jednou hardwarovou komponentou popsat celý systém kloubů. Používá se pro robotický hardware s více stupni svobody. Reprezentuje často systém s komunikací přes jedinou sběrnici.



Obrázek 3.1: Konečný automat životního cyklu hardwarové komponenty nebo kontroléru

Životní cyklus hardwarových komponent a kontrolérů

Životní cyklus hardwarových komponent a kontrolérů představuje konečný automat, skládající se z šesti metod sloužících k přechodu mezi stavy konečného automatu na obrázku 3.1. V aktivovaném stavu se použité metody liší svým názvem pro komunikaci s hardwarem se využívá metod `read()` a `write()`. A u kontrolérů ...

Každá metoda přechodu má svůj význam:

- `on_init()` – Stará se o kontrolu a načtení parametrů z konfiguračního souboru.
- `on_shutdown()` – Stará se o úklid po metodě `on_init()`. Tato metoda je většinou prázdná.
- `on_configure()` – V této metodě se připravuje hardware nebo kontrolér k aktivaci. Dochází zde k alokaci paměti.
- `on_cleanup()` – Tato metoda uvolňuje hardware a uklízí alokovanou paměť.
- `on_activate()` – Dochází k aktivaci hardware nebo kontroléru. Je nutné, aby metoda již neprováděla dlouhotrvající operace ovlivňující chod real-time systému.
- `on_deactivate()` – Deaktivuje hardware anebo kontrolér.

3.3 YARP

YARP neboli A Opět Další Robotická Platforma (Yet Another Robot Platform) je robotický middleware sloužící primárně jako podpůrný software ulehčující síťovou komunikaci mezi procesy robotického systému, komunikujících spolu **peer-to-peer**, procesy jsou rovnocenné, mohou se vydávat jak za klienta, tak i za server. Komunita robotického middleware YARP se primárně soustředí na vývoj humanoidních robotů. YARP oplývá přenositelností mezi operačními systémy díky použití knihovny **ACE** a možností psát kód v téměř jakémkoli jazyce podporovaném nástrojem **SWIG**, jímž lze vygenerovat rozhraní YARP middleware kompatibilní s dalšími programovacími jazyky, jinými než C++, ve kterém je YARP napsán.

Na rozdíl od ROS 2 nevyužívá nových síťových protokolů pro komunikaci, ale používá transportních protokolů TCP, UDP, protokolu ze síťové vrstvy multicastu a dalších složitějších protokolů. Jednotlivé procesy jsou nazývány jako **Porty** (Ports) spojení mezi nimi jako **Spojení** (Connections) a dohromady tvoří orientovaný graf Sítě YARP, kde **Porty** jsou uzly a **Spojení** představují hrany orientovaného grafu. Každý **Port** je reprezentován

unikátním jménem, které je registrováno pomocí **YARP name server**. **YARP name server** registruje Porty s překladem jejich unikátních jmen na IP adresy s čísla portů.

Kapitola 4

Pochopení zadání

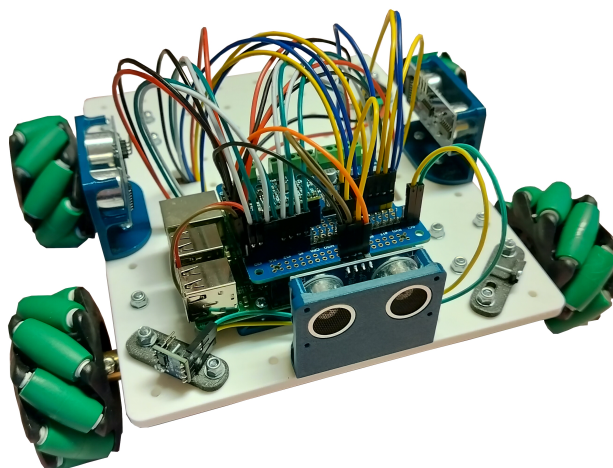
Název bakalářské práce „Modulární robotické vozidlo pro výukové účely“, konkrétně slovo **modulární** vyjadřuje možnou rekonfiguraci hardwaru robota, tak i rekonfiguraci robotického softwaru přidáním nového nebo nahrazením stávajícího. Pro výuku může být užitečný možností přidat nové senzory. Přidáním dalšího patra je možné rozšířit prostor pro další senzory.

Při výběru robotického middlewaru byly uvažovány prostředí ROS 2 a YARP. MicroROS byl vyřazen díky rozhodnutí navrhnout robotické vozidlo řízené pouze pomocí Raspberry Pi a vzdáleného počítače, bez použití mikrokontroléru. Po dalším nastudování robotických middlewarů jsem shledal YARP jako nevhodný pro implementaci robotického vozidla, jelikož jeho komunita se soustředí na vývoj humanoidních robotů. Nakonec byl zvolen ROS 2.

Kapitola 5

Způsob konstrukce a hardwarové vybavení robotického vozidla

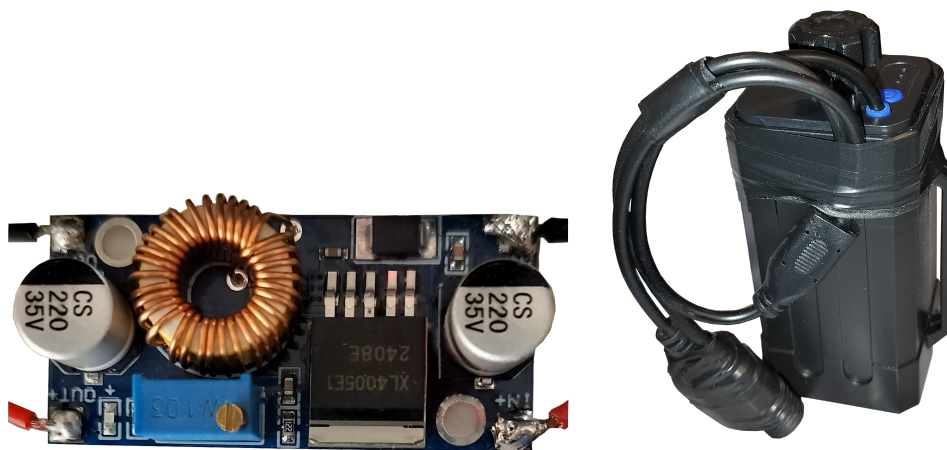
Pro robotické vozidlo byl vytvořen název MoRVE z anglického názvu bakalářské práce Modular Robotic Vehicle for Education. Modulární význam slova je v práci hodně využíván, jak v možnostech stavby robota ve smyslu použitého hardwaru, tak i návrhu robotického softwaru. Software je navržen tak, aby bylo možné nahradit část softwaru, která je nevyhovující, nebo jen změnit počet ovládaných kol, či obměnit typ řízení podvozku. Výsledná podoba robota je na obrázku 5.1.



Obrázek 5.1: Vzhled robota MoRVE

5.1 Konstrukce robotického vozidla

Pro navrhnutí konstrukce robota byl využit 3D modelovací program Fusion 360 od společnosti Autodesk. Konstrukce robota má být snadno konfigurovatelná, a proto při návrhu jednotlivých částí robota jsou základový plát a na něm umístěné různé podpůrné konstrukce pro senzory, baterie a Raspberry Pi navrženy dle návrhových pravidel Open Robotic Platform [1]. Na této webové stránce v záložce knihovna lze nalézt různé komponenty, které jsou spolu kompatibilní. Lze zde dohledat i komponenty využívané k sestavení toho robotického vozidla, které tam byli přidány v rámci této práce.



Obrázek 5.2: Nalevo step-down měnič s XL4005 a napravo voděodolné pouzdro pro 4 lithiové články 18650 s BMS

Všechny komponenty využitě na tomto robotickém vozidle lze vytisknout na 3D tiskárně. K tisku základového plátu se hodí materiál PLA pro svou malou pružnost, a materiál PETG lze využít na ostatní části robotického vozidla. Na propojení jednotlivých komponent je hlavně užito šroubů M3 o délce 12 mm, několik o délce 16 mm a samozjistící matice M3.

Pro upevnění sonarů k podpěře byl navíc použit drát. Pro upevnění Raspberry Pi k držáku a spolehlivému podepření rozšiřujícího motor HATu byly použity distanční sloupky M2,5 matka-šroub o výšce 15 mm a šroubky M2,5 o délce 8 mm.

5.2 Napájení robotického vozidla

Pro napájení robotického vozidla byly zvoleny Lithiové články 18650 s nominálním napětím 3,7 V. Jejich spojením do série může být vytvořena baterie 2S s nominálním napětím 7,4 V anebo 3S s nominálním napětím 11,1 V. Pro zvýšení možného průtoku proudu se články zapojují paralelně.

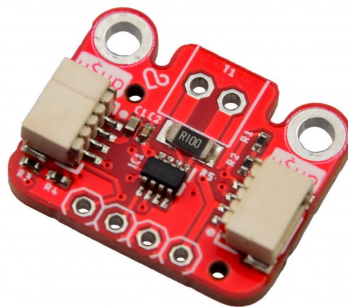
Konkrétně bylo vybráno pouzdro na obrázku 5.2 pro čtyři lithiové články, dva zapojeny sériově a tyto dvě série článků jsou spolu zapojeny paralelně. Takováto baterie má tedy nominální napětí 7,4 V a maximální proud roven dvojnásobku maximálního proudu specifikovaného na jednom článku.

Motory odebírají napětí přímo z baterií a pro snížení napětí pro Raspberry Pi 4B na 5 V je využito step-down měniče na obrázku 5.2 s nastavitelným výstupním napětím.

Pro nabíjení lithiových článků 18650 lze využít samostatnou nabíječku a nabíjet články samostatně. Nebo využít adaptéru s 8,4 V s DC jackem jako výstupním konektorem, tato varianta však postrádá vyvážené nabíjení.

5.3 Monitorování stavu baterie

V robotickém systému je třeba sledovat stav baterie pro včasné vypnutí robotického systému. Či robot, který je zautomatizován sám dojede na místo, kde se dobije a může pokračovat v předešlé činnosti bez vnějšího zásahu.



Obrázek 5.3: INA219 proudový senzor od laskakit [8]

```
[ALL]
# shutdown button
dtoverlay=gpio-shutdown,gpio_pin=21
```

Výpis 5.1: Přenastavení vypínacího pinu pro Raspberry Pi pomocí úpravy v konfiguračním souboru `config.txt`

Pro měření stavu nabití baterie je využito senzoru osazeným čipem INA219 vyobrazeným na obrázku 5.3, který je schopen měřit proud, napětí a spotřebovaný výkon na zátěži. Nicméně pro měření spotřebovaného výkonu na robotickém vozidlu se čtyřmi motory nemá dostatečné specifikace maximálního měřitelného proudu. Proto je měřeno pouze napětí baterie pro včasné vypnutí robotického systému.

Výpadek napájení při činnosti operačního systému na Raspberry Pi může poškodit integritu dat operačního systému a znemožnit opětovný boot operačního systému. Tomuto problému lze zabránit nastavením OS do read-only režimu, kdy nedochází k žádným zápisům na disk, ale všechny změny jsou ukládány do paměti RAM (Random Access Memory), po vypnutí OS jsou tyto změny ztraceny a systém se po opětovném spuštění nachází v původním stavu.

5.4 Vypínací tlačítko

Operační systémy určené pro Raspberry Pi mají možnost v souboru `config.txt` nastavit tzv. `dtoverlay`, který slouží pro nastavení předpřipravených funkcí Raspberry Pi. Pokud na konec souboru jsou přidány následující řádky nastavení z výpisu 5.1, tak lze propojením GPIO21 pinu k zemi zahájit vypínání operačního systému. Bez tohoto nastavení je použit GPIO3 jako výchozí vypínací a zapínací pin. Po aplikaci tohoto nastavení je možné vypnout systém propojením země a pinu GPIO21, zapnout lze Raspberry Pi propojením pinu GPIO3 a země, nebo odpojením napájecího napětí a jeho opětovným zapojením.

Lze tomu zabránit nastavením OS do režimu read-only souborového systému, kdy nedochází k žádným zápisům na disk, ale veškeré změny v systému jsou zapisovány do paměti RAM (Random Access Memory) Toho to lze dosáhnout využitím nástroje **raspi-config**.

Pro automatické odpojení od napájení po vypnutí robota je možné baterii předřadit relé. Relé by umožnilo přivést napětí k Raspberry Pi a k motorům a po vypnutí by Raspberry Pi vyslalo signál k relé, aby se přerušil přívod napájení od baterie.

5.5 Raspberry Pi 4B

Slouží jako hlavní řídicí jednotka robota, pro snadný vývoj softwaru za pomoci ROS 2 verze Jazzy byl jako operační systém zvolen Ubuntu 24.04 LTS. Raspberry Pi poskytuje 26 GPIO výstupů, sběrnici I²C, UART, SPI, dva PWM kanály.

5.6 Výběr motorů a kol

Při rozhodování, které motory budou vhodné pro robotické vozidlo hrála roli jejich cena, kvalita provedení a přínos naučení základů fungování řízení kol robotického vozidla.

Uvažovány byli kvalitní servomotory komunikující po sériové sběrnici. Každý servomotor by měl unikátní adresu, na kterou by se poslal příkaz s úhlovou rychlostí a mnoho dalších hodnot jako zrychlení, aktuální rychlost, teplota by šla přečíst. Tento přístup postrádá nutnost studenta pochopit fungování enkodérů měřících úhlovou rychlost motorů. Proto byli vybrány stejnosměrné motory s enkodéry, u kterých je nutné chápat podstatu fungování pro implementaci jejich hardwarových ovladačů.

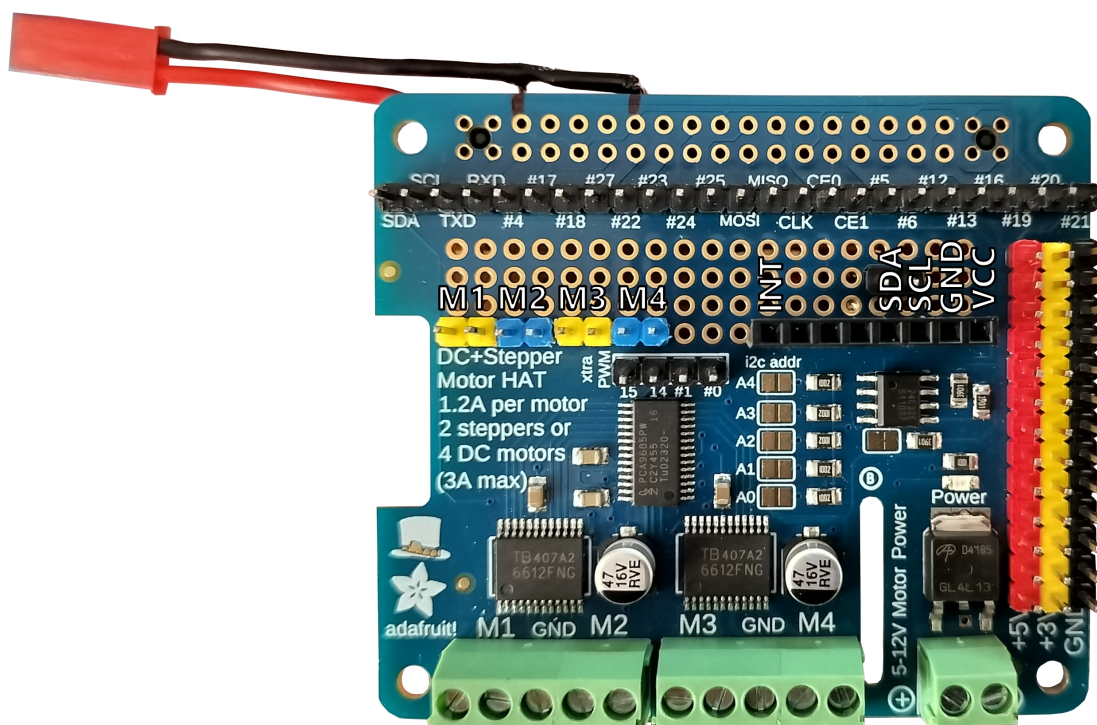
Při výběru stejnosměrných motorů s enkodéry je důležité si uvědomit, že s vyšším rozlišením enkodéru lze otáčky motoru řídit přesněji v nižších otáčkách za sekundu. Při výběru jsem zvolil DC motory 310 od prodejce Yahboom, které potřebují pro jejich funkci relativně nízké nominální napětí 7,4 V. Při transformaci tohoto napětí pro napájení Raspberry Pi na 5,1 V nedochází k tak velkým tepelným ztrátám jako při transformaci z 12 V zdroje.

Pro sledování stavu inkrementálních enkodérů je využito GPIO pinů na Raspberry Pi a pro napájení je použito 3,3 V. Experimentálně bylo zjištěno, že za jedno otočení hřídele motoru je na každém z výstupů A a B enkodéru vygenerováno 265 pulsů. Tato hodnota je dále využívána k nastavení správné funkce ovladače pro enkodér, který umí vyčíst úhlovou rychlost.

Jako kola byla vybrána mecanum kola, aby byla možnost vyměnit softwarově způsob řízení podvozku z diferenciálního řízení na všesměrový způsob řízení. Toto rozhodnutí dodává další možnosti k experimentování s robotickým vozidlem. Pro možnost předělání typu řízení z diferenciálního na všesměrový typ podvozku, bylo třeba najít způsob, jak řídit čtyři motory pomocí Raspberry Pi bez použití dalšího mikrokontroléru.

5.7 Rozšiřující motor HAT pro Raspberry Pi

Pro zapojení čtyř DC motorů byl vybrán Adafruit DC & Stepper Motor HAT pro Raspberry Pi, neboť Raspberry Pi disponuje pouze dvěma hardwarovými PWM výstupy a nebylo by možné ovládat více než dva motory současně. Toto řešení také uvolňuje několik GPIO pinů, které by jinak byly využity pro řízení H-můstků. Pro komunikaci s Raspberry Pi využívá Motor HAT sběrnice I²C. Lze s ním komunikovat na výchozí adrese 0x60, tuto adresu je možné změnit. Konkrétně Raspberry Pi komunikuje s čipem PCA9685, který slouží pro generování PWM signálů na 16 nezávislých kanálech. Frekvence lze nastavit od 24 Hz až na 1526 Hz [15]. Výstupy tohoto čipu jsou napojeny přímo na ovládací piny čtyř H-můstků,



Obrázek 5.4: Upravený Adafruit DC & Stepper Motor HAT pro Raspberry Pi

konkrétně dvou čipů TB6612FNG. Motor HAT je schopný řídit motory s napájecím napětím až 12 V.

Adafruit Motor HAT mění dvojřadé typické rozložení GPIO pinů na jedinou řadu, kvůli nedostatku místa na Motor HATu není využit pin GPIO26. Na obrázku 5.4 je ukázka upraveného Motor HATu.

Pro možnost zapojení senzorů k napájení a GPIO pinům musely být připájeny řady pinů s odstupem 2,54 mm. Pro připojení DC motorů k výstupům Motor HATu, který poskytuje možnost připojení pouze pomocí terminálových bloků, byly připájeny piny na pájivé pole umístěné na Motor HATu a ty propojeny s výstupy terminálových bloků. Pin GPIO26 je propojen s skrze konektor k pinu přerušení od IMU senzoru. K tomuto konektoru je možné připojit IMU senzor a skrze sběrnici I²C s ním komunikuje Raspberry Pi. Motor HAT neposkytuje regulované napájení pro Raspberry Pi z baterií a tak byl přidán konektor pro připojení napájení 5 V ze step-down měniče. Napětí pro spolehlivé napájení Raspberry Pi musí být o desetinu větší než 5 V.

Raspberry Pi je možné napájet buď z baterie, nebo z adaptéru. Nesmí nastat situace, kdy je propojeno napájení z baterie, tak i adaptér pro Raspberry Pi najednou.

5.8 Sonar

Jako sonar byl vybrán senzor HC-SR04 s možností změnit způsob komunikace připájením rezistorů o odporu 1 k Ω z výchozí komunikace s pomocí pinů trigger a echo na komunikaci skrz sběrnici I²C, komunikaci přes UART či jedním drátem sloužícím jako trigger a echo pin zároveň. Je využito výchozí komunikace pro zmenšení vytížení sběrnice I²C. Na robotu

MoRVE jsou k dispozici tři sonary jeden ve předu, druhý vlevo a poslední vpravo. Pro komunikaci každý potřebuje 2 GPIO piny. Aby nemusela být převáděna logická úroveň pomocí děliče napětí, je využito sonarů, které dokážou fungovat s napájecím napětím 3,3 V, a tedy i výstup na pinu echo má logickou úroveň 1 v 3,3 V.

Kapitola 6

Implementace řídicího softwaru

K implementaci řídicího softwaru jsem se rozhodl použít ROS 2 robotický middleware pro jeho rozsáhlou adopci komunitou používaného k implementaci robotických vozidel.

6.1 Uspořádání robotického softwaru

Pro logické oddělení robotického softwaru je využito ROS 2 balíčků. K překladu ROS 2 softwaru slouží nástroj `colcon`. Ten je schopný překládat balíčky využívající k implementaci uzly python, či C++. Každý projekt využívající ROS 2, využívá strukturu s hlavní pracovní složkou s obvyklým názvem `ros2_ws`, ve které se nachází složka `src` obsahující ROS 2 balíčky. K vytvoření šablony nového balíčku se využívá nástroje `ros2` následovaným argumentem `pkg` a příkazem `create`, po nich následují parametry.

Distribuvanost robotického systému

Robotický systém běží na robotovy (Raspberry Pi) a stacionárním počítači, ze kterého jsou posílány příkazy, jak se má robot chovat. Na robotovy běží řídicí smyčka a uzly, které potřebují ke svému běhu hardware. Na stacionárním počítači, který má vyšší výkon než Raspberry Pi, lze spustit Rviz pro zobrazení modelu robota, uzel pro teleoperaci uživatelem a analyzační nástroj pro zkoumání hardware a kontrolérů `ros2_control` frameworku. K analýze chování hardware a kontrolérů byl použit program `plot_juggler` ze stejnojmenného balíčku.

Aby uzly ROS 2 systému mohli mezi sebou komunikovat, musí se robot i stacionární počítač být připojen ke stejné lokální síti. Pro zajištění stejných síťových podmínek v jakémkoli prostředí je využito lokálního WiFi hotspotu ze stacionárního počítače, ke kterému se připojí robot.

6.2 Implementace ovladačů hardware

Pro usnadnění implementace byla využita knihovna `WiringPi`, která poskytuje funkcionalitu pro interakci s GPIO (General Purpose Input Output) piny, komunikaci po sběrnici I²C, vytvoření systémových přerušení (interrupts).

Veškeré knihovny ovladačů jsou přístupné z balíčku `morve_hardware_drivers` a k jejich implementaci byl použit programovací jazyk C++.

Následující podkapitoly vystihují zajímavé detaily z implementace.

Ovladač enkodéru

Využívá systémového přerušení k počítání pulzů. K výpočtu úhlové pozice ω_p natočení kola se využívá rovnice 6.2, kde N_c reprezentuje pozici natočení kola v jednotkách pulzů od začátku spuštění ovladače (může být i záporný). Pro výpočet úhlové rychlosti ω_v kola je užito rovnice 6.1, kde je třeba znát rozdíl pulzů $N_{\Delta t}$ před a po uběhnutí periody Δt , se kterou je měřena rychlost. A jako poslední údaj je nutné znát počet pulzů za jednu celou otočku kola N_f .

$$\omega_v = \frac{2\pi N_{\Delta t}}{N_f \Delta t} \quad (6.1)$$

$$\omega_p = \frac{2\pi N_c}{N_f} \quad (6.2)$$

Při implementaci ovladače pro enkodéry motoru jsem si uvědomil, že díky relativně malému rozlišení enkodérů, které činí 265 pulsů za jedno otočení hřídele motoru, měření úhlové rychlosti v nízkých otáčkách bude relativně nepřesné, a tudíž bude třeba zvolit takovou frekvenci odečítání rychlosti, aby za danou periodu vygeneroval více pulzů pro větší přesnost. Frekvenci čtení úhlové rychlosti nelze snižovat do absurdních hodnot, protože se zvyšuje odezva systému. Při rychlosti jednu otočku za sekundu již tento problém není tak markantní. Dostal jsem i nápad na implementaci enkodéru, který by nevyužíval počet pulsů za sekundu pro měření úhlové rychlosti, ale došlo by přímo k měření času mezi jednotlivými pulzy. Jenže toto řešení má zásadní nedostatek a to v podobě falešných pulzů s opačným směrem rotace než skutečný směr rotace motoru a to by způsobilo detekci rychlosti v jednom směru v aktuální rychlosti a následně ve stejné rychlosti, ale v opačném směru, a tedy způsobilo mnohem větší nestabilitu než původní řešení.

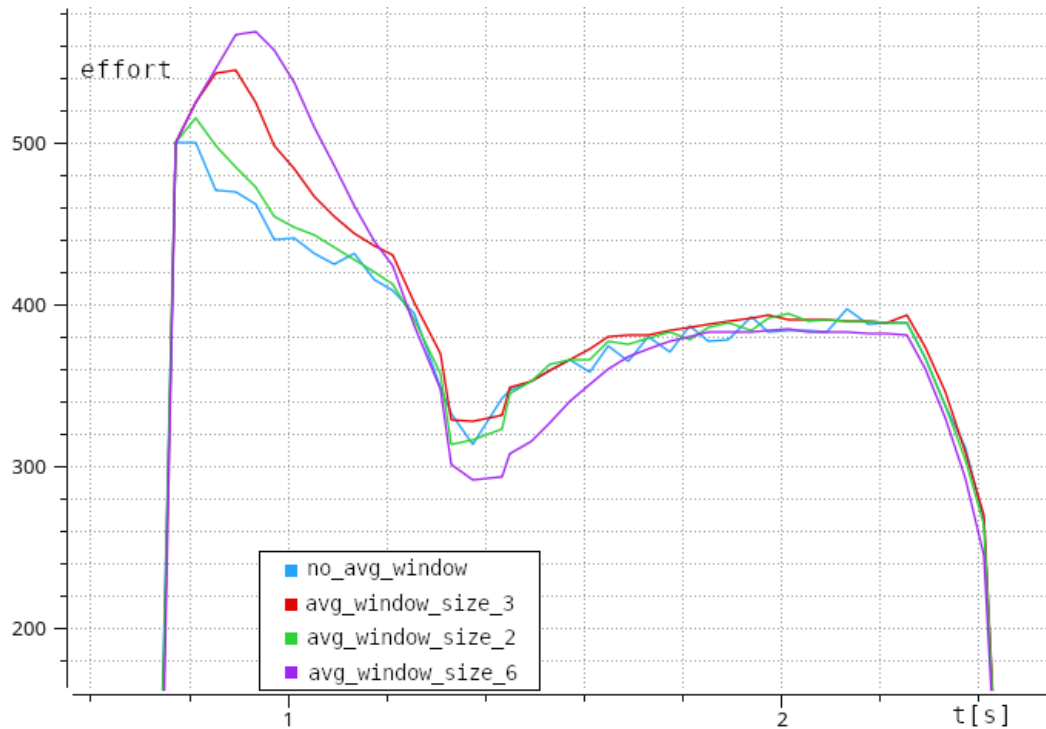
Protože rychlost čtená z enkodérů má relativně velký rozptyl hodnoty způsobený malým rozlišením enkodérů, tak byl navrženo využití průměrovacího okna, které si pamatuje určitý počet minulých hodnot rychlosti naměřených enkodérem výstupní hodnota tohoto filtru se vypočítává z předešlých hodnot a ze současné hodnoty rychlosti vydělených velikostí okna. Experimentálně bylo zjištěno, že nejlepší velikost průměrovacího okna naměřených rychlostí je 3. S delším oknem dochází k delší odezvě mezi skutečnou rychlostí motoru a měřenou rychlostí enkodérem, která je průměrována. A také dochází k výraznější odchylce od požadované rychlosti. Díky stabilnější měřené rychlosti je i PID regulátorem řízená snaha (PWM signál) k motorům stabilnější. Na obrázku 6.1 je ukázáno rozdílné chování řízení snahy pro různé velikosti průměrovacího okna i případ bez průměrovacího okna.

Ovladač pro motor HAT

Ovladač pro motor HAT je rozdělen do tří tříd, které postupně využívají funkcionality předchozí třídy ke správnému fungování. První třída `pca9685` implementuje komunikaci s čipem PCA9685 po sériové sběrnici I²C. Implementace umožňuje nastavit na výstupech PCA9685 čipu jakoukoli střidu PWM signálu nebo nechat výstup trvale v log. 0 nebo log. 1.

Třída `TB6612FNG` představující H-můstek, využívá předchozí implementace komunikace s čipem PCA9685, implementuje módy H-můstku pro snazší ovládání a možnost definovat, který výstup z čipu PCA9685 je napojen k řídicím pinům H-můstku.

Třída `motor_hat` implementující motor HAT využívá třídy `TB6612FNG` a definuje propojení výstupů z čipu PCA9685 k řídicím pinům H-můstků. Při inicializaci třídy lze zvolit adresu motoru HATu.



Obrázek 6.1: Graf zobrazující řízení snahy PID regulátorem ovlivněných různou velikostí průměrovacího okna úhlové rychlosti

Ovladač senzoru INA219

K implementaci ovladače bylo využito dokumentace [20]. Knihovna ovladače se stará pouze o čtení napětí, proud ani spotřebovaný výkon neřeší z důvodu nemožnosti měřit tyto údaje při měření napětí na baterii a relativně velkými skoky ve spotřebě proudu na motorech. Pro čtení napětí senzor využívá módu (triggered mode), ve kterém aktivně neměří hodnotu napětí, ale je dotázán, aby změřil hodnotu a tu poté vrátí. Dle specifikované přesnosti čeká ovladač určitý čas na naměřenou hodnotu dle dokumentace. Knihovna se senzorem komunikuje přes sběrnici I²C na adrese **0x40**.

Ovladač také implementuje přechod do úsporného režimu.

6.3 Implementace robotického podvozku

Pro implementaci řídicího systému podvozku je použit `ros2_control` framework. Cílem je vytvořit robotický podvozek, kterému budou posílány zprávy typu **TwistStamped** přes pojmenované téma `/cmd_vel`. Tento typ zprávy obsahuje informaci o translaci v m/s a rotaci v rad/s. Využití `ros2_control` frameworku umožňuje rozdělení implementace do logických vrstev starajících se o přístup k hardwaru (hardwarová komponenta), kontroléry postupně zjednodušující ovládání nejdříve samotných kol (PID regulátor) a poté podvozku jako celku – diferenciální či všesměrový podvozek.

Hardwarová komponenta

K zpřístupnění hardwaru pro ovládání motorů pomocí motor HATu a čtení stavu pozice a úhlové rychlosti hřídele motoru musela být napsána hardwarová komponenta název hardwarové komponenty. Skrze příkazová a stavová rozhraní umožňuje přístup k hardwaru řídicím kontrolérům, které přidávají nový způsob řízení samotných motorů, tak i celého podvozku.

Hardwarová komponenta je psána jako plugin, který je načten po spuštění controller manageru z pojmenovaného tématu `/robot_description`. Komponenta `motor_interface` z balíčku `morve_hardware_interfaces` je typu systém, takže využívá, jak stavových, tak i řídicích rozhraní. Hardwarový plugin slouží pro zpřístupnění motorů s enkodéry ovládaných pomocí motor HATu a enkodéry jsou zapojeny pomocí GPIO pinů. Hardwarová komponenta pro fungování vyžaduje minimálně 2 motory s enkodéry a maximálně lze definovat až 4 motory. Lze tedy pomocí tohoto hardwarového pluginu popsat robotický systém se dvěma až čtyřmi stupni volnosti. Pro předávání parametrů hardwarové komponentě je využit soubor typu URDF (Unified Robot Description Format), který je rozšířen o tag `<ros2_control>`. Uvnitř tohoto tagu se nachází tag `<hardware>`, jehož atribut `name` říká, ve kterém balíčku se nachází hardwarový plugin a samotný název pluginu. Tag `<hardware>` dále může obsahovat tag pro parametr `<parameter>`. Kromě tagu pro hardware obsahuje `<ros2_control>` tag pro klouby `<joint>`, jehož atribut `name` definuje jméno kloubu. Kloub zde představuje motor s enkodérem, kloub potřebuje znát své stavové rozhraní `<state_interface>` v jehož těle je uveden typ stavového rozhraní, v případě této hardwarové komponenty je třeba úhlová rychlost (`velocity`) a úhlová pozice (`position`) natočení kola. Pro příkazové rozhraní je použit tag `<command_interface>` v jehož těle je uveden typ rozhraní – snaha (`effort`) představující číslo, které ovlivňuje střihu PWM signálu přivedenému k motoru. Pro maximální střihu PWM signálu musí být hodnota rovna rozlišení $2^{12} = 4096$ PWM generátoru. Pro vyjádření střihu v procentech je třeba podělit snahu 4096, pokud je snaha kladná, kolo se otáčí po směru hodinových ručiček. Změna střihu PWM signálu ovlivňuje přivedené napětí k motoru a protože motor potřebuje k rozjezdu minimální nenulové napětí, tak nedochází k rozjezdu motoru okamžitě při použití obyčejného PID regulátoru, který postupně přidává k řízení veličině. Zóně, ve které motor zůstává stát, i přes to že je aplikováno napětí, se přezdívá mrtvá zóna. Protože střih PWM signálu dokáže přivést k motoru zlomek nebo aktuální hodnotu napětí z baterie, tak se velikost mrtvé zóny s klesajícím napětím rozšiřuje. Tento jev mění chování řízení při plně nabitě baterii a při téměř vybité baterii.

Hardware enkodéru potřebuje znát GPIO piny, s kterými má komunikovat, aby měřil rychlost na správném řízeném kole. V hardwaru se vybírá, který výstup motor HATu odpovídá, kterému kolu (kloubu).

Model robota MoRVE je popsán v balíčku `morve_description`. Popis robota je oddělen do několika samostatných souborů URDF používajících rozšíření z programu `xacro`. Soubor `ros2_control.urdf.xacro` popisuje použitý hardwarový plugin `motor_interface` s jeho parametry a propojenými klouby. Ukázka tohoto souboru je ve výpisu 6.1. Popis robota je rozdělen do samostatných souborů pro lepší udržitelnost. Pro jejich kompletaci se využívá program `xacro`, nabízí funkcionalitu maker pro snížení duplicit v popisu robota. Před každým spuštěním je nutné sestavit popis robota pro možnost přenosu jednotného kompletního URDF popisu pomocí uzlu `robot_state_publisher`.

Hardwarová komponenta přistupuje k hardwaru skrz metody `read()` a `write()`. Tyto metody jsou volány periodicky dle nastavené frekvence řídicí smyčky pro controller

```

<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  <xacro:macro name="morve_ros2_control" params="name prefix">
    <ros2_control name="${name}" type="system">
      <hardware>
        <plugin>morve_hardware_interfaces/motor_interface</plugin>
        <param name="motor_hat_i2c_addr">0x60</param>
      </hardware>

      <joint name="${prefix}front_left_wheel_joint">
        <param name="motor_hat_output_mapping">motor0</param>
        <param name="encoder_line_A_gpio_pin">gpio7</param>
        <param name="encoder_line_B_gpio_pin">gpio5</param>
        <param name="encoder_pulses_per_rotation">265</param>
        <param name="wheel_diameter_metres">0.065</param>
        <param name="window_size">1</param>
        <command_interface name="effort"/>
        <state_interface name="velocity"/>
        <state_interface name="position"/>
      </joint>
    </ros2_control>
  </xacro:macro>
</robot>

```

Výpis 6.1: Ukázka URDF souboru s konfigurací pro hardwarový plugin `motor_interface`, není kompletní chybí definice dalších kloubů pro funkčnost

`manager`. Metoda `write()` zapisuje příkaz z příkazového rozhraní do hardwaru a metoda `read()` čte stav z enkodéru a provádí zápis aktuálního stavu do stavových rozhraní pozice a rychlosti.

PID kontrolér

`Ros2_control` framework nabízí zřetěžitelný PID kontrolér implementující PID regulátor pro regulaci veličiny pomocí jiného řídicího rozhraní dle aktuální odchylky od požadovaného stavu. Umožňuje na jakýkoli počet kloubů a jejich stavová a řídicí rozhraní aplikovat PID regulátor s vlastním nastavením koeficientů pro každý PID regulátor. Parametry a název použitého kontroléru jsou uvedeny v konfiguračním souboru `config.yaml` umístěného v balíčku `morve_launch`. PID kontrolér je závislý na hardwarových rozhraních z `motor_interface`. PID kontrolér má za úkol regulovat úhlovou rychlost změřenou enkodérem na motoru a nahradit řídicí rozhraní typu snaha (effort) představující střidu PWM až do maximální hodnoty 4096. PID kontroléru je zadána referenční hodnota, které má motor dosáhnout. V případě opomenutí připojení napájení k motorům bude integrační složka bez nastavení omezení nekontrolovatelně růst, protože nedostane žádnou změnu rychlosti ze stavových rozhraní rychlosti na motorech.

`Ros2_control` framework nabízí implementaci PID kontroléru `pid_controller` využívající `control_toolbox::Pid`, který implementuje PID regulátor. Tento kontrolér neuvažuje možnost ovládání motoru na úrovni střidy PWM, kde by mohla existovat mrtvá zóna, ve které se regulátor ustálí na nechtěné hodnotě. Příkladem je rozjezd robota po prvním spuštění hardwaru a PID kontroléru. Kdy referenční hodnota rychlosti je nulová a řídicí výstup pro motor HAT je také nulový. Po rozjezdu robota a následném zastavení je referenční hodnota rychlosti opět nulová, ale řídicí výstup PID regulátoru napojený na příkazové rozhraní snahy je nenulový. Stane se tak, protože v první fázi rozjezdu PID regulátor překoná horní hranici mrtvé zóny motoru, napětí, při kterém dochází k rozjezdu motoru, ve fázi zastavování již nemá PID regulátor potřebu dále snižovat hodnotu řízeného výstupu PWM k nule, neboť motor je již ve stavu, kdy stojí. Tato vlastnost může pomoci v případě, kdy by robot měl jet pouze vpřed, po zastavení zůstává integrální složka nenulová a rozjezd robotického vozidla je rychlejší. V opačném případě pokud robot musí couvat, tak musí překonat celou velikost mrtvé zóny od horní kladné hranice až po zápornou dolní hranici pro opačný směr rotace motorů. V těchto případech nedošlo k velkým chybám při řízení jen k prodlevám. Stav, kdy robot stojí a je mu dodáváno nenulové napětí, vytváří při stání robota nepříjemný vysoký pískající zvuk. Sekundárním problémem, který je poznat až při řízení robota, který způsobuje také mrtvá zóna, je že nenulová integrační složka PID regulátoru se nemusí ustálit na každém motoru na stejné hodnotě. Například robot se otáčí na místě, a proto potřebuje, aby motory na jedné straně se otáčeli vzad a na druhé vpřed. Po zastavení zůstane výstup PID regulátorů na levé straně v záporných nenulových hodnotách a v PID regulátorech pravých motorů zůstává hodnota v kladných nenulových hodnotách. Pokud po tomto stavu je robotovy vyslán příkaz jízdy vpřed dochází k dřívějšímu rozjezdu motorů na pravé straně a motorům na pravé straně trvá chvíli než překonají celou velikost mrtvé zóny, která je v tomto momentu dvojnásobná. Tudíž robot místo jízdy vpřed ještě chvíli zatáčí než se vyrovnají rychlosti na obou stranách robota.

Aby tyto problémy mohli být vyřešeny, musel být implementován kontrolér `PwmPidController::pwm_pid_controller` kontrolér, který bere mrtvou zónu v potaz. Jedná se o reimplementaci oficiálního kontroléru `PidController::pid_controller`. Pro reimplementaci chování musel být nahrazen `control_toolbox::Pid` novou implementací PID regu-

látoru. V původní implementaci je využíváno vzorců pro výpočet řídicího signálu z grafické reprezentace významu chyby. Pro novou implementaci jsem využil vzorec 2.5 využívající aktuální hodnoty chyby a dvou minulých hodnot. Dále je tento PID regulátor doplněn o podmínky zajišťující případy:

1. **Zastavení robota** – po obdržení příkazu (cíle) k zastavení je integrační složka PID regulátoru vynulována na nulovou hodnotu, PID regulátor může stále měnit hodnotu integrační složky, pokud nedochází k postupnému zastavení, záleží i na nastavených koeficientech PID regulátoru
2. **Rozjezd robota** – při rozjezdu je k integrační složce PID regulátoru přičtena nastavená hodnota, která přibližně odpovídá mrtvé zóně motoru
3. **Změna směru rotace kola** – pokud je nutné změnit směr rotace motoru, je v tomto okamžiku nastavena taková hodnota integrační složky PID regulátoru, aby došlo k okamžitému rozjezdu motoru na opačnou stranu

Po vynulování integrační složky nedochází již k pískání motorů. `PwmPidController` počítá pouze s propojením stavového rozhraní rychlosti. A řídicí rozhraní může být pouze snaha, přesněji pwm střída.

Stejně jako jeho původní verze je možné chování PID kontroléru měnit za pomoci parametrů. Některé parametry chybí, protože byly nepotřebné pro regulaci rychlosti, ale byly důležité pro řízení pozice. Parametry je možné měnit i za běhu po spuštění kontroléru v rámci `controller manageru` pomocí `ros2` konzolového nástroje.

Napojení jiných uzlů na parametry je možné docílit měnící se hranice mrtvé zóny a zlepšit celkovou odezvu systému v závislosti na aktuálním napětí baterie, které je dodáváno k motorům. Se stejnou střídou PWM signálu je s nižším napětím dodáváno motorům méně napětí, a tak se zákonitě musí i hranice mrtvé zóny zvětšit. Tímto způsobem se dá za běhu systému měnit vlastnosti `pwm_pid_controller::PwmPidController`. Běžně by to nemělo být postřehnutelný rozdíl mezi činnostmi vylepšeného systému se systémem se statickou hodnotou mrtvé zóny. Napětí baterie je přenášeno společně s informací zda je baterie téměř vybitá. Hodnoty napětí může využít jiný uzel v systému, který zná rozlišení PWM signálu pro přesné odečtení hodnoty, ve které by měla začínat mrtvá zóna s měnícím se napětím a pro správné nastavení hranic mrtvé zóny v `pwm_pid_controller::PwmPidController`.

Nastavení hodnot PID regulátoru

Pro nastavení konstant PID regulátoru byl zvolen postup pokus omyl. Pro vyladění konstant jsou všechny koeficienty nastaveny na nulové. Začne se postupným zvyšováním proporčního koeficientu a pozorováním řízeného výstupu, dokud nenastanou oscilace výstupního signálu regulátoru, poté se sníží koeficient na hodnotu při které neosciloval. Proporční koeficient se ponechá již nastaven. Integrační koeficient byl zvětšován postupně a byly pozorovány změny reakce regulátoru, dokud nezačne trvat déle než se ustálí na cílové hodnotě. Při nastavování derivačního koeficientu, který by měl zmírnit překmity řízené veličiny, jsem zjistil, že to úplně neplatí. Místo utlumení překmitů došlo k nekontrolovatelné oscilaci při nastavení derivačního koeficientu na hodnotu, kdy začal působit. Proto tento koeficient je nastaven jako téměř nulový. K vizualizaci chování PID regulátoru byl použit vizualizační program `plotjuggler`, ve kterém jdou analyzovat data přenášena přes stavová a příkazová rozhraní hardwarové komponenty či kontroléru v reálném čase. Umožňuje zobrazovat graf zvolené veličiny v reálném čase.

Kontrolér řídící podvozek, Kontrolér představující typ podvozku, Diferenciální podvozek

Na nově vzniklá příkazová rozhraní typu `velocity` očekávající úhlovou rychlost vytvořených PID kontrolérem, lze napojit kontrolér představující podvozek. Konstrukce robotického vozidla umožňuje využití diferenciálního podvozku nebo všesměrového. Rozhodl jsem se využít kontroléru implementujícího diferenciální podvozek, lze ovšem použít i všesměrový typ podvozku.

Podvozky implementované v `ros2_controllers` odebírají data z pojmenovaného tématu `/cmd_vel`, přenášená data jsou typu zprávy **TwistStamped**. Dříve pro přenos byla používána pouze zpráva typu **Twisted**, která se skládá z části představující translaci a druhé části představující rotaci. Při použití této zprávy mohlo dojít ke zpoždění zprávy, která došla k podvozku robota a ta by se zpracovala jako by došla včas. Zpráva **TwistStamped** navíc obsahuje časový údaj, který má zabránit v přijímání opožděných zpráv a nezpracovávat staré zprávy. Pro řízení diferenciálního podvozku stačí v údaji o posunu využívat pouze osu `x` představující pohyb vpřed či vzad. Při použití všesměrového podvozku je využita i osa `y` společně s osou `x` pro popsání všesměrového pohybu bez rotace.

Stejně jako všechny ostatní kontroléry jeho nastavené parametry se nachází v konfiguračním souboru `config.yaml`. K využití diferenciálního podvozku stačí, pokud jsou dostupné dvě ovládatelné kola (klouby).

Podvozek může publikovat data o odometrii, která jsou důležitá pro propojení s informacemi z ostatních senzorů a hlavně mapování. K části mapování jsem se bohužel nedostal. To však může být plán pro pokračování.

Ovládání robota

K řízení podvozku robota je použit uzel `teleop_twist_keyboard` ze stejnojmenného balíčku. Implementuje ovládání robota skrze klávesnici a konzolovou aplikaci. Příkazy k pohybu robota posílá přes pojmenované téma `/cmd_vel`, při použití výchozího nastavení posílá zprávy bez časové známky, kdy byly vytvořeny. Aby komunikoval zprávou **TwistStamped** je nutné přiřadit parametru `stamped` hodnotu `true`. Řídící uzel je uzpůsoben pro řízení diferenciálního robota. Zprávou je zasílán požadavek rychlosti v jednotkách `m/s` a pro otáčení používá úhlovou rychlost v `rad/s`.

6.4 Samostatné uzly zveřejňující data ze senzorů

Pro implementaci přenosu dat z ostatních senzorů byl využit asynchronní přístup přes pojmenované témata. Do těchto pojmenovaných témat zveřejňují data uzly mající přístup k danému hardwaru. Data ze senzorů je pak možné odebírat z pojmenovaných témat pro další zpracování.

Uzel měřící napětí baterie

Tento uzel `battery_voltage` je implementován v balíčku `battery_voltage`. Lze nastavit při jaké hranici napětí je baterie považována za téměř vybitou. Měření napětí baterie je prováděno senzorem `INA219` a je publikováno uzlem `battery_voltage` na pojmenované téma `/battery`. Typ zprávy `BatteryStatus` posílané na pojmenované téma `/battery` je definován v balíčku `morve_interfaces`. Zpráva obsahuje informaci o aktuálním napětí baterie a příznaku vybité baterie.

Uzly měřící vzdálenosti od překážek

Uzel `sonar_node` z balíčku `sonar_distance` měří vzdálenost pomocí sonaru a publikuje data na pojmenované téma dle názvu uzlu. Pro přenos dat ze sonaru je použit již existující typ zprávy `Range` z balíčku `sensor_msgs`. Periodicita měření je ovlivněna nastaveným časem pro měření.

6.5 Spouštěcí skripty

Spouštěcí (Launch) skripty slouží pro spuštění uzlů robotického systému s jejich konfigurací aniž by musely být manuálně v konzoli přiřazovány hodnoty parametrům. Dalším důvodem jejich vzniku je zdlouhavé spuštění celého systému uzlů. Lze pomocí spouštěcích skriptů spustit i systém s více uzly, nebo pro spuštění robotického podvozku v `ros2_control` frameworku, kdy dochází k postupnému načítání hardwaru a poté kontrolérů, které je řízeno vnějšími skripty. Ke zpřístupnění všech lokálních balíčků a jejich spustitelných uzlů a spouštěcích skriptů je třeba zpřístupnit cesty k těmto balíčků z pracovního prostředí `ros2_ws` pomocí příkazu:

```
source ./install/setup.bash
```

Manuální spuštění uzlu

Pro manuální spuštění uzlů je třeba znát název balíčku a název uzlu, který je třeba spustit. Pro zjištění dostupných parametrů je nutné uzel nejdříve spustit a za jeho běhu zjistit pomocí konzolového nástroje `ros2` – příkazem `ros2 param list` se vypíše všechny dostupné parametry uzlů, které mohou být změněny i za běhu anebo při spuštění uzlů. Pro samotné spuštění uzlů v příkazové řádce je použito syntaxe příkazu:

```
ros2 run package node --ros-args -p parameter:=value -r /topic:=/new_t
```

Kde za přepínačem `--ros-args` jsou umístěny přepínače pro přiřazení hodnot parametrům – dlouhá varianta `--param` nebo krátká varianta již uvedená v příkladu `-p`. Pro přejmenování názvů pojmenovaných témat, služeb a akcí, skrze která uzel komunikuje, se používá dlouhého přepínače `--remap` anebo krátkého `-r`.

Manuální spuštění uzlu lze provést pro uzel `battery_voltage` publikující data o aktuálním napětí baterie. Příkladem jeho spuštění může být následující příkaz bez změny výchozích hodnot parametrů a přejmenování pojmenovaného tématu `/battery`:

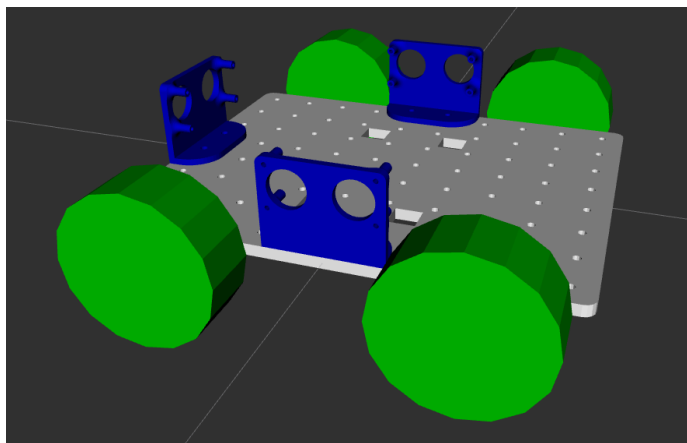
```
ros2 run battery_voltage battery_voltage
```

Spouštění uzlů pomocí spouštěcích skriptů

Spouštěcí (Launch) skripty mohou být napsány v jazyce XML, YAML, nebo Pythonu. Spouštěcí skripty v jazyce Python jsou novinkou v ROS 2 oproti ROS 1, umožňují psát komplexnější spouštěcí skripty. K napsání spouštěcích skriptů byl použit Python. Spouštěcí skripty jsou umístěné v balíčku `morve_launch` a pro spuštění jakéhokoliv skriptu se používá daná syntaxe:

```
ros2 launch package script.launch.py parameter:=value
```

Pro změnu hodnoty parametru stačí napsat název parametru a přiřadit mu novou hodnotu. Na rozdíl od obyčejného ROS 2 uzlu je možné vypsát parametry daného spouštěcího skriptu



Obrázek 6.2: Model robota MoRVE, popsaný URDF soubory, zobrazený v programu Rviz

přidáním dlouhého přepínače `--show-args`, vypíše parametry i s jejich popisem a výchozí hodnotou, parametry spouštěných uzlů jsou nadále dostupné přes nástroj **ros2** příkazové řádky.

Konkrétní spouštěcí skripty

Všechny skripty jsou umístěny v balíčku `morve_launch`. Skript `morve_view.launch.py` slouží k zobrazení virtuálního modelu robota popsáno v URDF formátu a zároveň ověření správnosti URDF souborů. Model robota MoRVE je zobrazen v programu Rviz a je možné otáčet simulovanými koly pomocí posuvníků. Na obrázku 6.2 je zobrazen model robota MoRVE v programu Rviz.

Pro spuštění robotického podvozku byl vytvořen skript `morve.launch.py`. Ten spustí **controller manager** a postupně načítá hardware a kontroléry jako plugíny. Aby bylo umožněna změna PID kontroléru řešícího regulaci střídavé PWM s mrtvou zónou za kontrolér implementující obyčejný PID regulátor. Tak konfigurační soubor `morve_controllers.yaml` s parametry kontrolérů obsahuje všechny kontroléry a spouštěcí skript načte všechny tyto kontroléry, ale aktivuje pouze kontrolér implementující PID regulátor s mrtvou zónou a jeho navazující diferenciální kontrolér. Ostatní kontroléry zůstanou v nenakonfigurovaném (`unconfigured`) stavu. Pro manipulaci s kontroléry se používá nástroje příkazové řádky `ros2 control`. K vypsání stavu a aktuálně načtených kontrolérů lze použít příkaz `ros2 control list_controllers`.

Pro docílení automatizovaného spuštění hardwarových uzlů na robotovy je možné spouštěcí skripty spouštět v rámci služeb podsystému **systemd** Operačního Systému. Uzly by tedy byly spuštěny při náběhu OS. Bez tohoto mechanismu je nutné se k robotu připojit pomocí protokolu ssh (secure shell).

6.6 Analýza spuštěného robotického systému

- Ros2_control framework nabízí možnost analýzy doby provádění jednotlivých kontrolérů spuštěných v uzlu controller manager. Statistická data z controller manageru jsou publikována na pojmenované téma `/diagnostics`. Zpráva obsahuje data o životním stavu kontrolérů a hardwarových komponent. Ros2_control framework nenabízí možnost změny

životního stavu aktivovaného hardwaru na například neaktivovaný. Je tomu tak aby se zabránilo možným únikům paměti a zdrojů. Kontrolér je možné načíst v nenakonfigurovaném stavu, ale nelze jej do tohoto stavu dostat z již neaktivního (nakonfigurovaného) stavu. Pro dostání kontroléru do nenakonfigurovaného (unconfigured) stavu je nutné kontrolér uvolnit a opět ho spustit.

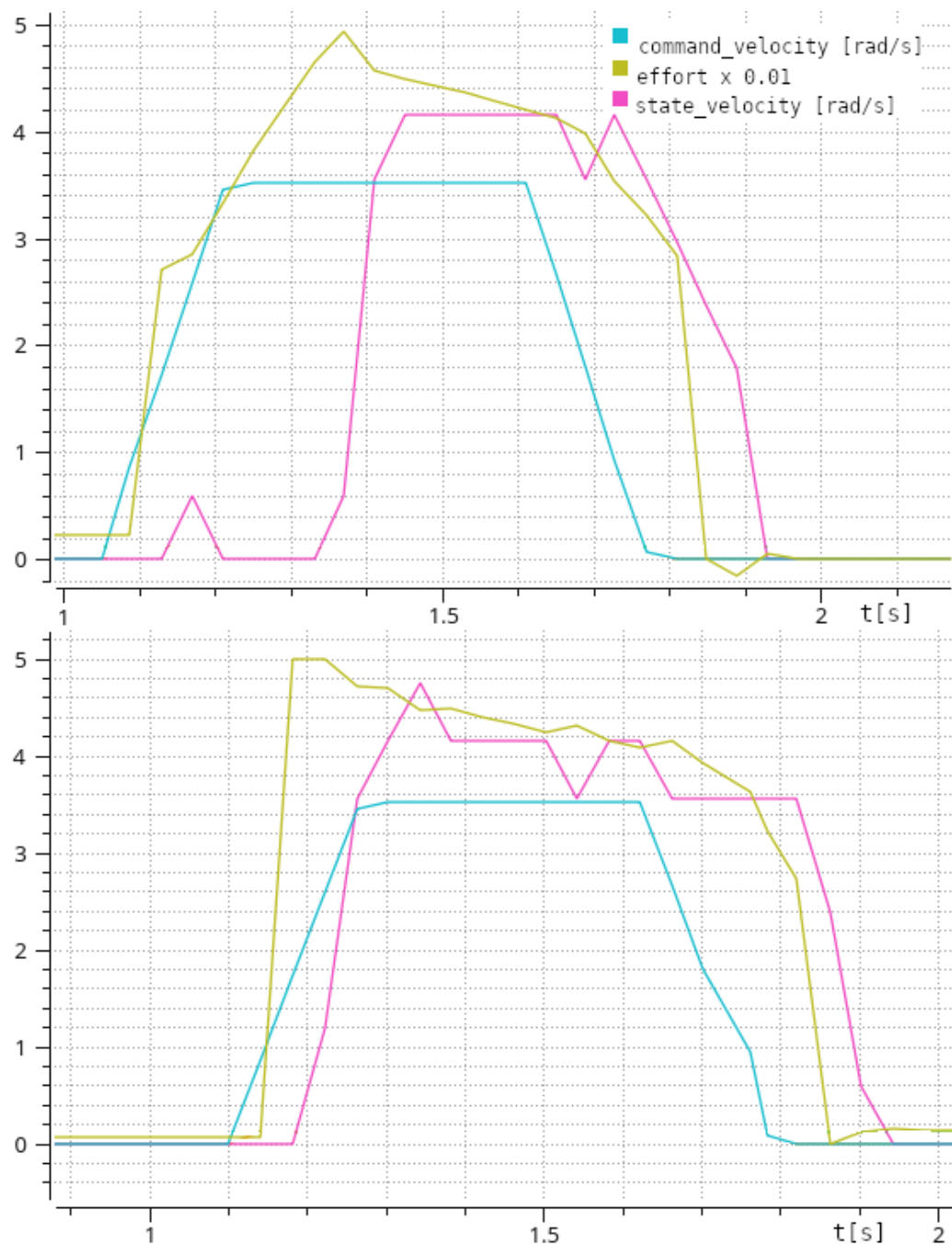
Analýza odezvy podvozku

K analýze odezvy podvozku na příkaz `z /cmd_vel` je použito programu `plotjuggler`. Pro analýzu kontrolérů `ros2_control` frameworku je nutné odebírat data z pojmenovaných témat `~/introspection_data/values` a `~/introspection_data/names`. Odezvu controller manageru nebylo možné pouze pomocí `plotjuggleru` změřit. Ale místo toho jsem zjistil, že hlavním důvodem proč se zdá, že systém podvozku vykazuje známky odezvy nesouvisí s rychlostí přenosu zprávy po síti. Nesprávné nastavení hranice mrtvé zóny ovlivňoval rychlý start motorů, a tak se zdálo, že je v systému delší odezva.

Pomocí programu `plotjuggler` je zaznamenán průběh řídicího signálu snahy, aktuální rychlosti, a referenční rychlosti. Tento průběh je zobrazen na obrázku 6.3. První graf ukazuje případ, kdy je nesprávně nastavená horní hranice mrtvé zóny na hodnotu snahy 270. To způsobuje, že po skokovém nárůstu řízené snahy (střídý PWM signálu) je napětí pro rozjetí motoru nedostatečné a integrační část PID regulátoru musí zareagovat a zvýšit svou hodnotu, aby došlo k rozpohybování kola. Druhý případ ukazuje průběh s nastavenou hranicí mrtvé zóny na hodnotu snahy 500. Dochází k téměř okamžitému rozpohybování kola v reakci na příkaz cílové rychlosti kola.

Hodnota snahy v grafu je sto krát zmenšena, aby mohla být zobrazena společně s ostatními průběhy grafů.

Protože musí být překonána mrtvá zóna a cílová rychlost je nastavena na relativně nízkou, tak dojde k přestřelení úhlové rychlosti rotace kola a dochází k pozvolnému zpomalování a snižování střídý PWM signálu.



Obrázek 6.3: Grafy zobrazující reakci PID regulátoru na změnu cílové rychlosti, v prvním grafu je nastavena nastavena nesprávná horní hranice mrtvé zóny na 270 a v druhém případě je mrtvá zóna nastavena na hodnotu 500

Kapitola 7

Závěr

Cílem práce bylo navrhnout a postavit robotické výukové vozidlo a napsat pro něj robotický software s pomocí jednoho zvoleného robotického middlewaru, konkrétně ROS 2. Demonstrovat základní robotické úlohu teleoperaci. Cíle práce bylo dosaženo. Konstrukce robotického vozidla byla navržena s pravidly Open Robotic Platform a k výrobě různých držáků a plátů byl využit 3D tisk. K propojení vytisknutých částí robota je použito standardně dostupných šroubů. Robotický systém je rozdělen na uzly zpřístupňující hardware běžící na Raspberry Pi a stacionárního počítače sloužícího k analýze robotického systému a spouštění hlavních řídicích uzlů teleoperace. Robot využívá 4 DC motorů s enkodéry ke svému pohybu. Pro jejich řízení je vytvořena hardwarová komponenta v `ros2_control` frameworku a reimplementaci PID kontroléru řešícího mrtvou zónu při řízení DC motoru. Robotický systém implementuje ROS 2 uzly s měřením napětí baterie a pro publikaci vzdáleností sonary.

K navázání na práci je možné využít uzel zveřejňující napětí baterie k včasnému vypnutí robota před vybitím baterie, dále pro dynamickou změnu hranic mrtvé zóny PID kontroléru pro rychlejší reakce podvozku na příkazy k pohybu. Další rozšíření pomocí přidání senzorů jako je LIDAR, kamera. Implementace algoritmu pro navigaci a mapování. Umožnění spouštění robotického systému bez nutnosti se připojit k robotovy a spouštět uzly a možnost měnit životní cyklus kontrolérů `ros2_control` frameworku robotického systému pouze ze stacionárního počítače. Repozitář¹ se softwarem je dostupný online.

¹https://github.com/janeckar/Modular_robotic_vehicle_software

Literatura

- [1] BARTNIK, N. *Design rules* online. Nikodem Bartnik, 2023. Dostupné z: <https://openroboticplatform.com/designrules>. [cit. 2025-02-15].
- [2] CABRÉ, T. P.; VELA, A. S.; RIBES, M. T.; BLANC, J. M.; PABLO, J. R. et al. Didactic platform for DC motor speed and position control in Z-plane. *ISA Transactions* online, 2021, sv. 118, s. 116–132. revidováno: 12. ledna 2021. ISSN 0019-0578. Dostupné z: <https://doi.org/10.1016/j.isatra.2021.02.020>.
- [3] CONRADIE, D. *Building Robots With A 20×20 Grid* online. Hackaday, 5. února 2024. Dostupné z: <https://hackaday.com/2024/02/05/building-robots-with-a-20x20-grid/>. [cit. 2025-04-26].
- [4] DAVIS, N. *The Difference Between Slow Decay Mode and Fast Decay Mode in H-Bridge DC Motor Applications* online. EETech, 10. srpna 2017. Dostupné z: <https://www.allaboutcircuits.com/technical-articles/difference-slow-decay-mode-fast-decay-mode-h-bridge-dc-motor-applications/>. [cit. 2024-11-29].
- [5] GOKTURK, S.; YALCIN, H. a BAMJI, C. A Time-Of-Flight Depth Sensor – System Description, Issues and Solutions. In: *2004 Conference on Computer Vision and Pattern Recognition Workshop*. IEEE, 2004, s. 35–35.
- [6] HERTZ, J. *H-bridge DC Motor Control Using Complementary PWM, Shoot-through, and Dead-time* online. EETech, 29. srpna 2022. Dostupné z: <https://www.allaboutcircuits.com/technical-articles/h-bridge-dc-motor-control-complementary-pulse-width-modulation-pwm-shoot-through-dead-time-pwm/>. [cit. 2024-11-13].
- [7] INCZE, I.; SZABÓ, C. a IMECS, M. Incremental Encoder in Electrical Drives: Modeling and Simulation. In: RUDAS, I. J.; FODOR, J. a KACPRZYK, J., ed. *Computational Intelligence in Engineering*. Springer Berlin Heidelberg, říjen 2010, sv. 313, s. 287–300. ISBN 978-3-642-15219-1.
- [8] LASKAKIT. *LaskaKit INA219 proudový senzor* online. Laskakit, 2025. Dostupné z: <https://www.laskakit.cz/laskakit-ina219-proudovy-senzor/>. [cit. 2025-04-15].
- [9] LI, Y. a IBANEZ GUZMAN, J. Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems. *IEEE Signal Processing Magazine*. Institute of Electrical and Electronics Engineers (IEEE), červenec 2020, sv. 37, č. 4, s. 50–61. ISSN 1558-0792.

- [10] MACENSKI, S.; FOOTE, T.; GERKEY, B.; LALANCETTE, C. a WOODALL, W. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, Květen 2022, sv. 7, č. 66.
- [11] RAJ, T.; HASHIM, F. H.; HUDDIN, A. B.; IBRAHIM, M. F. a HUSSAIN, A. A Survey on LiDAR Scanning Mechanisms. *Electronics*, 2020, sv. 9, č. 5, článek 741. ISSN 2079-9292.
- [12] ADEEPT. *Adept AWR 4WD WiFi Smart Robot Car Kit for Raspberry Pi 4/3 Model B+/B/2B, DIY Robot Kit for Kids and Adults, OpenCV Target Tracking* online. Adept, 2025. Dostupné z: https://www.adept.com/adept-awr-4wd-wifi-smart-robot-car-kit-for-raspberry-pi-3-model-b-b-2b-diy-robot-kit-for-kids-and-adults-opencv-target-tracking_p0122_s0033.html. [cit. 2025-04-26].
- [13] HANDSON TECHNOLOGY. *HC-SR04 Ultrasonic Sensor Module User Guide* online. Handson Technology. Dostupné z: <https://www.handsontec.com/dataspecs/HC-SR04-Ultrasonic.pdf>. [cit. 2025-1-18].
- [14] HUSARION. *ROSbot 3 (and previous 2x versions)* online. Husarion, 2025. Dostupné z: <https://husarion.com/manuals/rosbot/>. [cit. 2025-05-12].
- [15] NXP SEMICONDUCTORS. *PCA9685: 16-channel, 12-bit PWM Fm+ I2C-bus LED controller* online. NXP Semiconductors, duben 2015, revidováno: 16. 4. 2015. Dostupné z: <https://www.nxp.com/docs/en/data-sheet/PCA9685.pdf>. [cit. 2025-04-18].
- [16] RIGBETEL LABS LLP. *Tortoisebot ROS2 Humble Release* online. Github, 11. prosince 2024. Dostupné z: <https://github.com/rigbetellabs/tortoisebot>. [cit. 2025-04-26].
- [17] ROS COMUNITY. *ROS 2 Documentation* online. Open Robotics, 2025. Dostupné z: <https://docs.ros.org/en/jazzy/index.html>. [cit. 2025-03-26].
- [18] ROS COMUNITY. *ROS 2 Documentation* online. Open Robotics, 2025. Dostupné z: <https://docs.ros.org/en/jazzy/Concepts/Intermediate/About-Domain-ID.html>. [cit. 2025-04-10].
- [19] ROS2__CONTROL DEVELOPMENT TEAM. *Welcome to the ros2_control documentation – Jazzy!* online. ros2_control Development Team, 2025. Dostupné z: <https://control.ros.org/jazzy/index.html>. [cit. 2025-04-29].
- [20] TEXAS INSTRUMENTS. *INA219 Zero-Drift, Bidirectional Current/Power Monitor With I²C Interface* online. Texas Instruments, srpen 2008, revidováno: prosinec 2015. Dostupné z: https://www.ti.com/lit/ds/symlink/ina219.pdf?ts=1746203207020&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FINA219%253Futm_source%253Dgoogle%2526utm_medium%253Dcpc%2526utm_campaign%253Dasc-amps-null-44700045336317113_prodfolderdynamic-cpc-pf-google-ww_en_int%2526utm_content%253Dprodfolddynamic%2526ds_k%253DDYNAMIC+SEARCH+ADS%2526DCM%253Dyes%2526gad_source%253D1%2526gad_campaignid%253D6465287935%2526gbraid%253D0AAAAAC068F29A-VFUFq14RGIMSgS5JF2z%2526gclid%253DCj0KCQjw2tHABhCiARIsANZzDWoEXFkzyL_LeVAGcFnL_-XGNWpMiArUVX6090TsJwf9046oUFW0bgaAqp2EALw_wcB%2526gclidsrc%253Daw.ds. [cit. 2025-04-30].

- [21] TEXAS INSTRUMENTS. *DRV8840 DC Motor Driver IC* online. Dokumentace. Texas Instruments, březen 2010, revidováno: březen 2023. Dostupné z: <https://www.ti.com/lit/ds/slvsab7e/slvsab7e.pdf?ts=1732892247625>. [cit. 2024-11-29].
- [22] WIKIPEDIA. *Proportional–integral–derivative controller* online. Wikipedia, 25. listopadu 2024. Dostupné z: https://en.wikipedia.org/wiki/Proportional%E2%80%93integral%E2%80%93derivative_controller. [cit. 2024-12-11].
- [23] WIKIPEDIA. *Rotary encoder* online. Wikipedia, 28. listopadu 2024. Dostupné z: https://en.wikipedia.org/wiki/Rotary_encoder#Magnetic_absolute_encoders. [cit. 2024-12-2].