Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

Plzeň, 2019 Josef Švec

Místo této strany bude zadání práce

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 26. dubna 2019

Josef Švec

Abstract

This thesis deals with design of wheeled robot control system using ROS framework. The robot is able to reach the specified location on the map, which the robot creates while moving. In the thesis there is a complete description of the used robotic platform, the control of DC motors and mathematical description of used robot. The next part deals with the ROS framework and its use for wheeled robot control. At the end of the thesis, created experiments verifying the functionality of the proposed control system are described.

Keywords

ROS, differential driven robot, SLAM, DC motors

Abstrakt

Tato práce se zabývá návrhem řídicího systému kolového robota s využitím softwarového rámce ROS. Úkolem robota je dojet na zadané místo v mapě, kterou si sám během jízdy vytváří. V práci je uveden kompletní popis použité robotické platformy, dále je podrobně vysvětleno řízení stejnosměrných motorů, následuje matematický popis použitého robota. Další část pojednává o sofwarovém rámci ROS a jeho využití pro řízení kolového robota. V závěru práce jsou popsány reálné experimenty ověřující funkčnost navrženého řídicího systému.

Klíčová slova

ROS, diferenciálně řízený robot, SLAM, stejnosměrné motory

Obsah

1 Úvod

Kybernetika a robotika jsou vědní disciplíny, které si kladou za cíl především navrhovat řídicí systémy. Na počátku vzniku těchto oborů se k řízení využívaly mechanické systémy (Wattův odstředivý regulátor), později s vývojem dalších vědních oborů se přistoupilo k využití elektronických součástek (analogové řízení) a dnes je řízení spjato především s výkonnou výpočetní technikou a programovacími jazyky. Řízené mohou být živé soustavy, procesy ve společnosti nebo neživé systémy. Správně navržený řídicí algoritmus fungující na reálných systémech je přínosný především tím, že sám vykonává nějakou činnost bez přímé účasti člověka. Lidé se tak nemusí zabývat příliš jednotvárnými činnostmi nebo se nemusí pohybovat v nebezpečných prostředích.

Robot je neživá soustava, která je schopná vykonávat určitý úkol v reálném prostředí. K poznání reálného světa roboti většinou využívají řadu senzorů (kamera, lidar, GPS). Zpracováním dat ze senzorů získají roboti model okolního prostředí a navržením vhodného řídicího systému vznikne neživý stroj, který reaguje tak, aby splnil zadaný úkol.

Existuje nepřeberné množství robotů a jejich dělení. Základní rozdělení můžeme stanovit z hlediska možnosti přemisťovat se. Stacionární roboti se nemohou pohybovat z místa na místo (například průmysloví roboti v továrnách). Mobilní roboti se naopak mohou přemisťovat (například drony nebo autonomní vozidla).

Další důležité rozdělení je z hlediska míry autonomie. Nejjednodušší roboti jsou přímo řízeni člověkem (stroje). Složitější roboti využívají senzory a jsou naprogramováni k plnění úkolu (robotické vysavače). Nejsložitější roboti využívají poznatky umělé inteligence a rozhodují se samostatně (humanoidní roboti). Pokrok směřuje k stále složitějším a inteligentnějším robotům, kteří by ke své činnosti člověka vůbec nepotřebovali. V poslední době se vynořují etické otázky spojené s příliš inteligentními roboty. Zodpovědnost leží jednoznačně na lidech navrhujících řídicí systémy, robot vykonává pouze činnost, na kterou byl naprogramován.

Mobilní roboty můžeme dále rozdělit na základě prostředí, ve kterém se pohybují. Dnes existují roboti, kteří zkoumají oceány a moře. Musí být vodotěsní a dostanou se na místa, kam by se jinak člověk nepodíval. Někteří roboti dokonce prozkoumávají cizí planety (Mars). U těchto robotů jsou kladeny velké nároky na spolehlivost a funkčnost. Dále existují například létající roboti (drony). V poslední době se právě tato kategorie robotů

velmi dynamicky rozvíjí. V této práci bude popsáno řízení kolového robota jezdícího po zemském povrchu.

Cílem této práce je navrhnout řídicí algoritmus tak, aby robot dojel na místo určení v mapě, kterou si sám vytvoří, a sledoval vytyčenou trajektorii co nejlépe. Ke splnění tohoto cíle je nutné vyřešit několik dílčích úkolů. Zpočátku bude nutné řešit především řízení motorů tak, aby se otáčely požadovanou rychlostí. V dalším kroku bude potřeba vytvořit model řízeného robota a následně s využitím volně dostupných nástrojů sestavit celý systém tak, aby plnil zadaný úkol.

V následující kapitole bude popsána robotická platforma, další kapitola bude obsahovat popis návrhu řízení stejnosměrných motorů, čtvrtá kapitola se zaměří na model diferenciálně řízeného robota, pátá kapitola bude zaměřena na softwarový rámec ROS (Robot Operating System) [9], šestá kapitola bude popisovat komponenty softwarového rámce ROS vhodné pro řízení kolového robota, sedmá kapitola bude zaměřena na popis konkrétního použití softwarového rámce ROS a v poslední kapitole budou výsledky experimentů.

2 Robotická platforma

V této kapitole budou popsány všechny důležité komponenty, ze kterých je kolový robot sestaven. Návrh řídicího systému bude následně proveden pro tuto robotickou platformu.

2.1 Mechanická konstrukce

Tato práce byla zpracována s využitím podvozku Thumper. Jedná se o šestikolového robota se stejnosměrnými motory, kde tři kola na straně se pohybují vždy společně (diferenciálně řízený robot). Zatáčení vozítka je realizováno protichůdným otáčením náprav (smykem), jízda dopředu a dozadu pak společným pohybem všech šesti kol najednou. Výhodou této konstrukce je, že se robot dokáže otočit na místě.

váha	2.7 kg
užitečné zatížení	5 kg
světlá výška	60 mm
šířka	245 mm
délka	420 mm
poloměr kola	62 mm
převodový poměr	75:1

Tabulka 2.1: Specifikace mechanické konstrukce vozítka Thumper [2]

Šířka byla měřena mezi místy dotyku prostředních kol s podložkou. Poloměr kola byl změřen mezi středem kola a místem dotyku kola s podložkou. Tyto parametry budou v práci dále využity.

Obrázek 2.1: Thumper [1]

2.2 Senzory

2.2.1 Lidar

Pro snímání překážek a navigaci v prostoru je na vozítku umístěn laserový lidar HOKUYO URG-04LX-UG01. Lidar slouží k měření vzdáleností od překážek. Základní princip spočívá v měření času vyslaného a přijatého laserového paprsku, který se odrazil od překážky v prostoru. Výsledkem měření je mračno bodů, které může sloužit například k tvorbě mapy prostoru nebo 3D modelů budov a jiných objektů. Lidar nachází své uplatnění v geodézii, archeologii, geomatice, seismologii, meteorologii a robotice.

Poprvé se tento vynález objevil v šedesátých letech, krátce po objevení laseru. Využíval se nejprve v meteorologii k měření oblačnosti. Do širšího povědomí se dostal díky vesmírnému výzkumu, kde byl použit k mapování povrchu Měsíce. Dnes běžně dostupné lidary využívají lasery bezpečné pro oči, což znamená, že mohou být používány bez jakýchkoli ochranných pomůcek. Systémy s vysokým výkonem se používají především ve výzkumu atmosféry.

Hlavní výhodou lidaru je, že se laserový paprsek šíří prostorem velmi rychle (oproti sonaru, který využívá ultrazvuk) a dokáže měřit vzdálenosti s vysokou přesností [16].

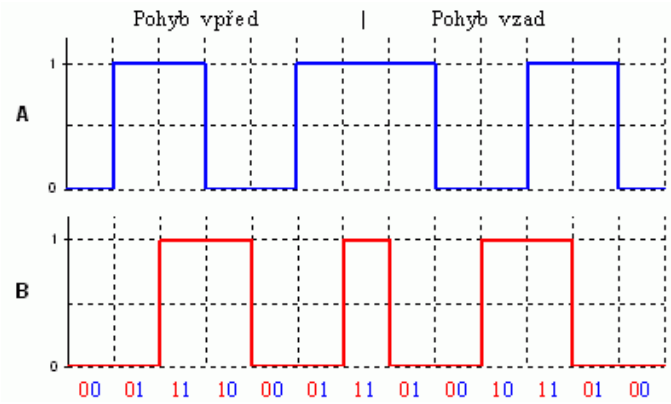
Nevýhodou lidarů je, že jejich schopnosti výrazně snižují nepříznivé podmínky, jako je déšť, mlha, sníh či tma. Ve venkovním prostředí se tedy nelze spolehnout pouze na měření z lidaru.

Obrázek 2.2: Lidar HOKUYO

2.2.2 Enkodéry

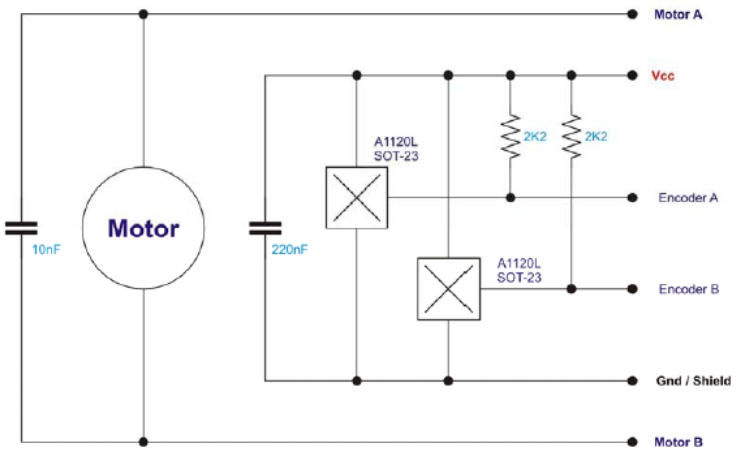
Prostřední motory jsou osazeny kvadraturními enkodéry, které slouží k měření změny polohy kola přímo na robotovi. Z této informace je možné dopočítat aktuální úhlovou rychlost kola, která je pak využita v diskrétním PI regulátoru (kapitola 3).

Princip kvadraturního enkodéru je založený na dvou fázově posunutých signálech A a B. Každý signál je schopný vyvolat přerušení a podle aktuálního stavu se situace vyhodnotí buď jako jízda vpřed nebo zpět.



Obrázek 2.3: Průběh výstupního signálu z enkodéru [5]

Enkodéry na vozítku Thumper se skládají ze speciálních kulatých osmipólových magnetů a dvou Hallových senzorů zapojených dle obrázku (2.4).

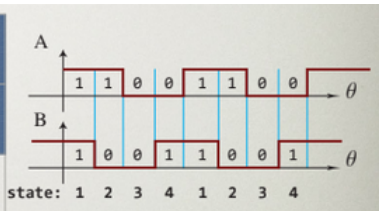


Obrázek 2.4: Schéma zapojení [4]

Přesnost je dána počtem elektrických pulzů na celou otáčku kola robota. Záleží přitom na počtu pólů magnetu enkodéru a také na převodovém poměru. Enkodéry změní svůj stav 16x za otáčku [4]. Převodový poměr je 75:1. Počet pulzů na celou otáčku kola robota je tedy 1200.

Vyhodnocení lze jednoduše provést podle následující tabulky (dec je snížení, inc zvýšení počtu čítaných pulzů).

	prior state	1	2	3	4
present state	AB	11	10	00	01
1	11	X	dec	X	inc
2	10	inc	X	dec	X
3	00	X	inc	x	dec
4	01	dec	X	inc	x



X = do nothing.

Obrázek 2.5: Tabulka pro vyhodnocování

Snímání pulzů je realizováno s periodou 0.1 s (přerušením od časovače). Získáme tedy počet pulzů za jednotku času. Abychom mohli vypočítat aktuální úhlovou rychlost otáčení kola $\omega(k)$, musíme využít následující vzorec:

$$\omega(k) = \frac{n(k) \cdot 2\pi}{1200 \cdot 0.1} \tag{2.1}$$

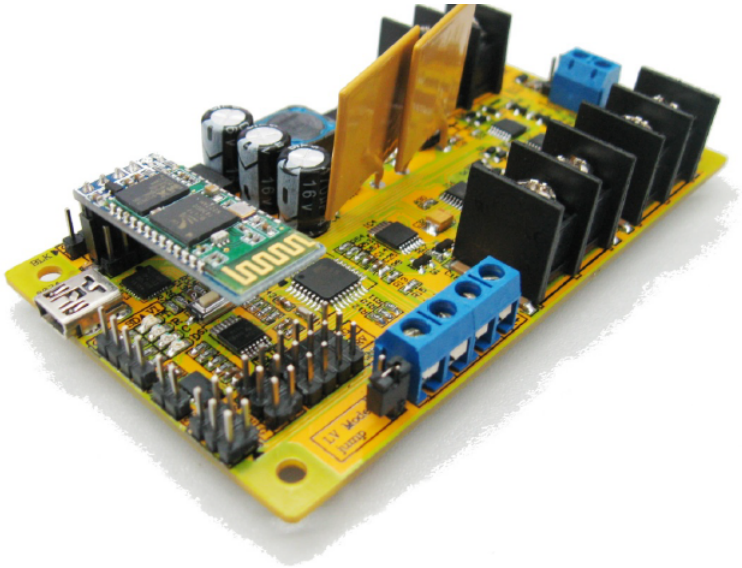
kde $n(k)$ je právě naměřený počet pulzů.

2.3 Řídicí elektronika

K řízení motorů byla použita řídicí deska T-REX, která je kompatibilní s řídicí deskou Arduino. Obsahuje také mikrokontrolér ATmega328P. Součástí řídicí desky T-REX je navíc i duální H můstek, který realizuje pohyb motorů. Dále je možné využít integrovaný akcelerometr a ovládat servomotory.

frekvence	16 MHz
FLASH	32 K
SRAM	2 K
EEPROM	1 K
napájení	6 – 30 V
akcelerometr	MMA7361L
bootloader	Arduino Nano w/ ATmega 328
výstupy servo	6x

Tabulka 2.2: Specifikace řídicí desky T-REX [13]



Obrázek 2.6: T-REX [13]

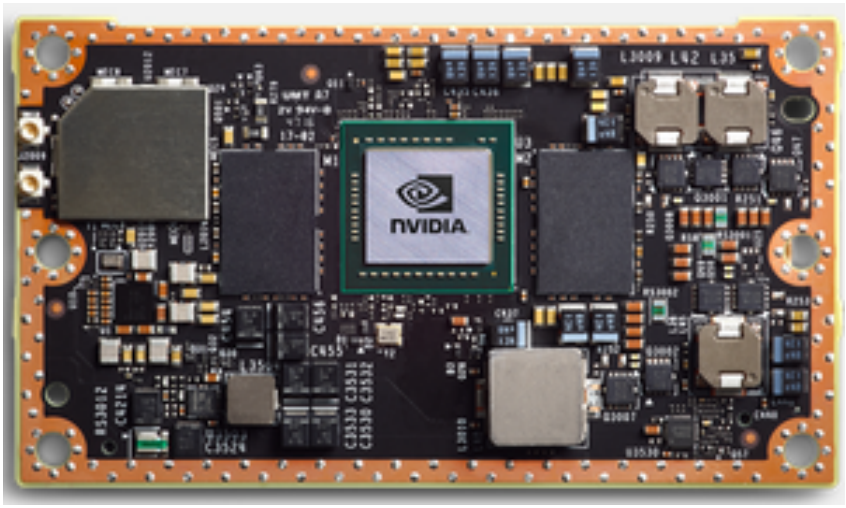
Originálně je vozítko Thumper dodáváno se softwarem pro ovládání přes vysílačku, akcelerometr je využit pro signalizování nárazu.

V této práci byl Thumper řízen přes mikropočítač sestavený z modulu Jetson TX2 a nosné desky Orbitty Carrier. Na tomto mikropočítači běží operační systém Linux. Tato varianta byla zvolena proto, aby bylo možné využít softwarový rámec ROS, který usnadní v budoucnu přidání dalších senzorů a umožní použití volně dostupných nástrojů vhodných pro lokalizaci a navigaci robota.

GPU	NVIDIA Pascal™, 256 CUDA jader
CPU	HMP Dual Denver 2/2 MB L2 + Quad ARM® A57/2 MB L2
video	4K x 2K 60 Hz Encode, 4K x 2K 60 Hz Decode
rozměry	50 x 87 mm
paměť	8 GB 128 bit LPDDR4 59.7 GB/s
displej	2x DSI, 2x DP 1.2 / HDMI 2.0 / eDP 1.4
připojení	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth
úložiště	32 GB eMMC, SDIO, SATA

Tabulka 2.3: Specifikace modulu Jetson TX2 [7]

Modul Jetson TX2 má dostatek výpočetního výkonu i pro zpracování obrazu z kamery. V budoucnu by tedy bylo možné například rozpoznávat předměty okolo robota.

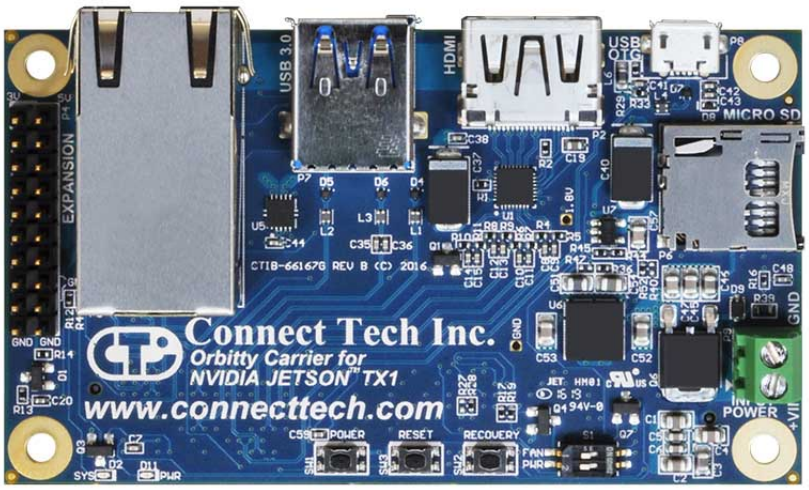


Obrázek 2.7: Modul Jetson TX2

Modul Jetson TX2 je možné zapojit pouze přes speciální konektor. Originálně je modul dodáván ve vývojové sadě s velikou nosnou deskou, která mu zprostředkovává klasické vstupně-výstupní rozhraní. Z důvodu ušetření místa na robotické platformě byla využita náhrada v podobě menší nosné desky Orbitty Carrier. Tato nosná deska zprostředkovává modulu Jetson TX2 napájení a klasické vstupně-výstupní rozhraní.

Napájení	9 – 14 V
USB	1x USB 3.0
připojení	1x Gigabit Ethernet (10/100/1000)
rozměry	50 x 87 mm
SD karta	1x microSD
displej	1x HDMI
sériová komunikace	2x 3.3V UART

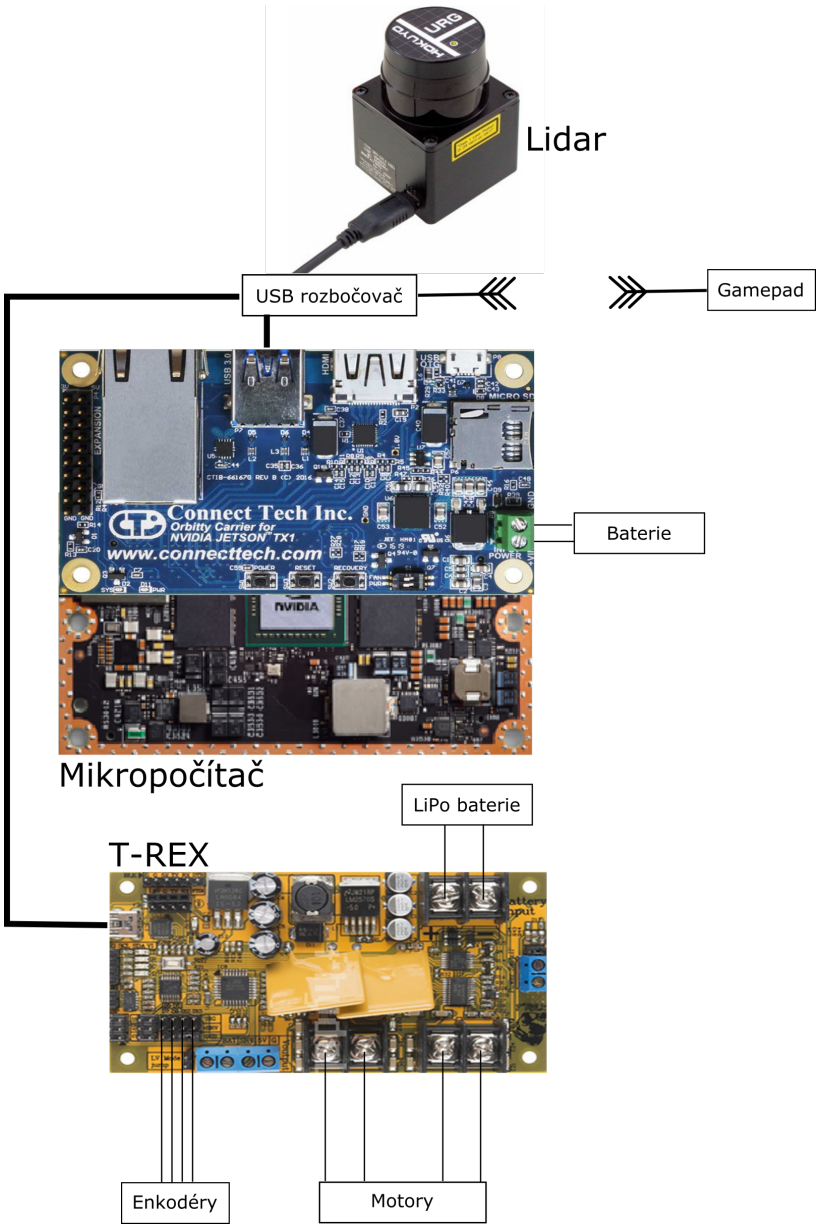
Tabulka 2.4: Specifikace nosné desky Orbitty Carrier [8]



Obrázek 2.8: Nosná deska Orbitty Carrier

2.4 Propojení komponent

Zjednodušené zapojení prvků na vozítku Thumper je na obrázku (2.9). Napájení řídicí desky T-REX a motorů je realizováno dvoučlánkovou LiPo baterií o kapacitě 4000 mAh. Mikropočítač je napájen 9V powerbankou určenou pro notebooky. Gamepad je bezdrátový a do USB rozbočovače je zapojen pouze bluetooth modul do USB.



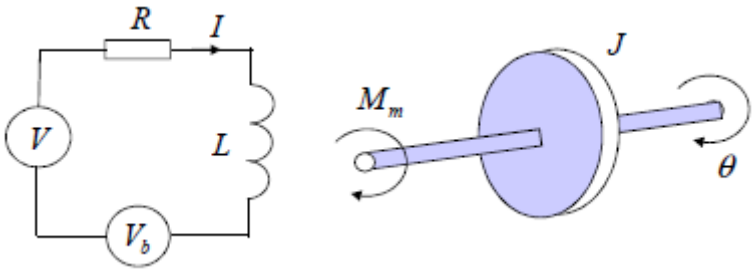
Obrázek 2.9: Schéma zapojení komponent na robotické platformě

3 Řízení stejnosměrných motorů

Stejnosměrný (DC) motor je nejstarším typem elektrického motoru. Je využíván v mnoha robotických aplikacích kvůli jednoduchosti řízení. Skládá se z permanentních magnetů (stator) a cívek (rotor), které indukují vždy stejně orientované magnetické pole jako je v jejich okolí. Tím dojde k otočení hřídele, následně se změní směr protékání proudu cívkami díky komutátoru a magnetické pole je zase orientováno stejně, čímž se motor opět pootočí [21].

3.1 Model

Při návrhu modelu předpokládáme tuhý rotor a hřídel. Na hřídel působí viskózní tření, které se zvyšuje s rostoucí rychlostí otáčení. Obecně je točivý moment vygenerovaný motorem přímo úměrný procházejícímu proudu a velikosti magnetického pole. V tomto případě předpokládáme, že magnetické pole je konstantní. Při návrhu modelu rozdělíme motor na elektrickou a mechanickou část.



Obrázek 3.1: ilustrace motoru [20]

Indukčnost L a odpor R reprezentují indukčnost a činný odpor cívek DC motoru. Napětí V_b odpovídá velikosti elektromotorické síly generované při otáčení motoru. Elektromotorická síla je úměrná úhlové rychlosti otáčení motoru $\frac{d\theta}{dt}$. Moment M_m generovaný motorem je přímo úměrný proudu kotvy i. J označuje celkový moment setrvačnosti motoru a zátěže. B je koeficient viskózního tření. Použitím 2. Newtonova zákona a Kirchhoffových

zákonů získáme následující rovnice.

$$Ri + L\frac{di}{dt} = V - K_f\frac{d\theta}{dt} \tag{3.1}$$

$$K_t i = J\frac{d^2\theta}{dt^2} + B\frac{d\theta}{dt} \tag{3.2}$$

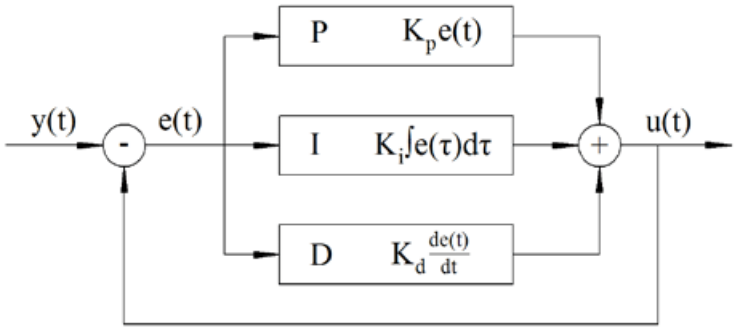
kde V je napětí, K_f je elektrická konstanta a K_t je konstanta motoru. Použitím Laplaceovy transformace získáme přenos V->$\frac{d\theta}{dt}$:

$$P(s) = \frac{K_t}{(Js + B)(Ls + R) + K_t K_f} \tag{3.3}$$

Stejnosměrný motor je tedy systém druhého řádu. Při modelování byly využity zjednodušující předpoklady, reálně se ale jedná o nelineární systém, protože závislost napětí na otáčkách není lineární (s většími otáčkami roste tření a ztráty) a především má každý motor tzv. mrtvé pásmo, které se musí při roztáčení překonat.

3.2 Řízení stejnosměrných motorů

Pro řízení stejnosměrných motorů se nejčastěji využívají PID regulátory [18]. Jedná se o zpětnovazební regulátory, které využívají měření aktuální úhlové rychlosti přímo na systému a podle toho upravují akční zásah. Tento přístup zásadně ovlivňuje přesnost a kvalitu regulace. Hlavní výhoda PID regulátorů je v jejich jednoduchosti, přesnosti a možnosti parametry doladit ručně.

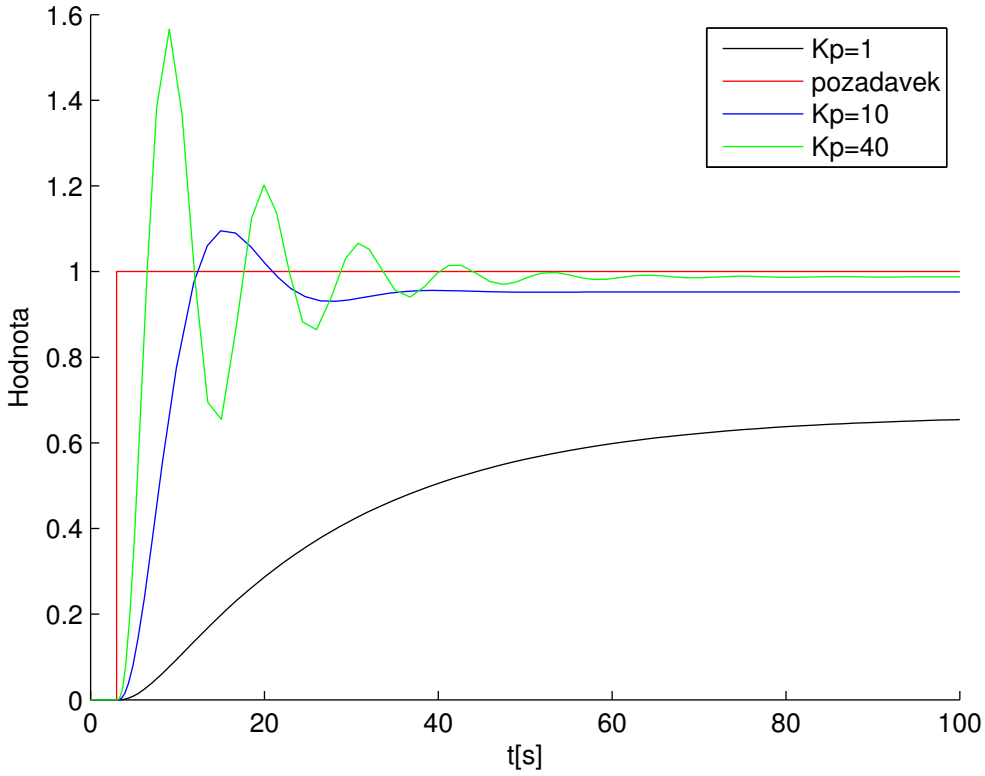


Obrázek 3.2: Schéma spojitého PID regulátoru

Na obrázku (3.2) je blokové schéma spojitého PID regulátoru. Regulační odchylka $e(t)$ je rozdílem požadované hodnoty $y(t)$ a aktuálně naměřené

veličiny $u(t)$. K_p je koeficient pro výpočet proporcionální složky, K_i je koeficient pro výpočet integrační složky a K_d je koeficient pro výpočet derivační složky.

Proporcionální složka je přímým násobkem regulační odchylky, rychle změní akční zásah, ale hrozí kmitání a nestabilita při příliš vysoké hodnotě koeficientu K_p. Proporcionální složka nezaručí nulovou regulační odchylku v ustáleném stavu.

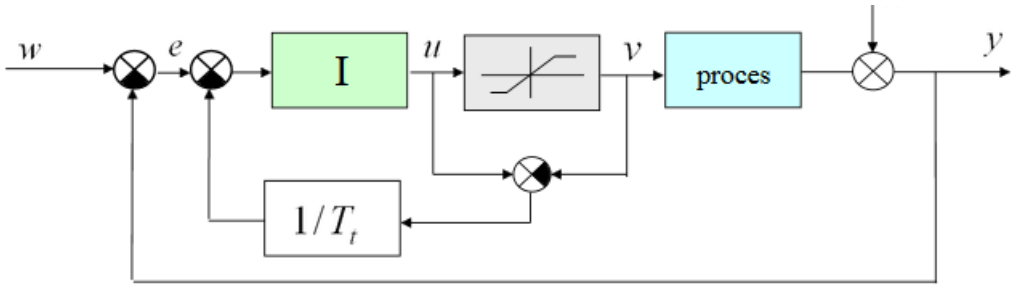


Obrázek 3.3: Odezva proporcionální složky

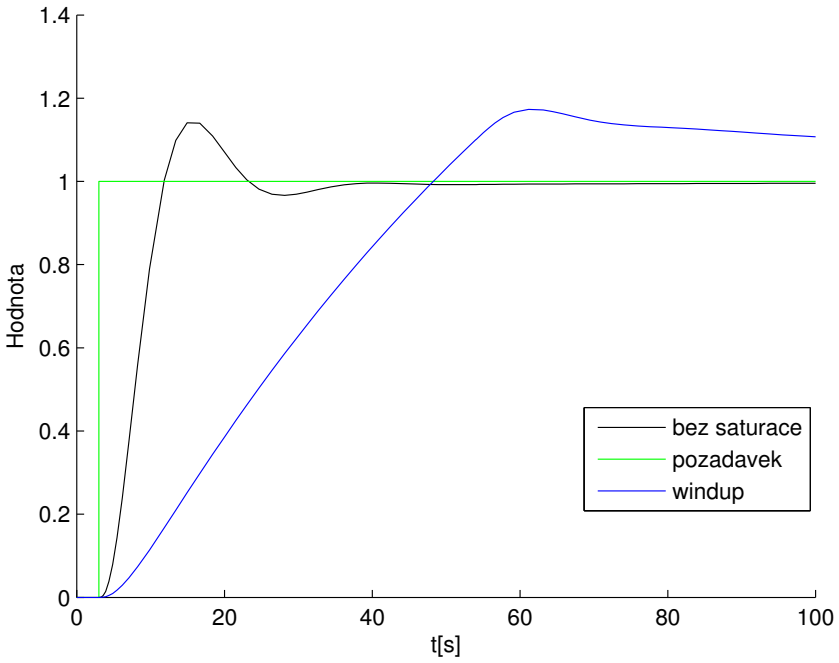
Integrační složka je integrálem (součtem) předchozích regulačních odchylek. Při správném nastavení zlepšuje přesnost regulace a zaručuje nulovou regulační odchylku v ustáleném stavu. V případech, kdy má systém integrační charakter, lze ji vynechat, ale jinak se používá téměř vždy. Při příliš vysoké hodnotě koeficientu K_i může docházet k oscilacím.

Při implementaci PID regulátorů na reálných systémech může docházet k tzv. unášení integrační složky (windup) [14]. Tento jev je způsoben saturací reálných systémů (například na stejnosměrný motor nelze přivést větší napětí než na jaké je konstruován). Regulátor se snaží zvýšit akční zásah, ale regulační odchylka už se dále nezmenšuje. Tím pádem dochází k neustálému

narůstání integrační složky. Naintegrovaná hodnota pak při požadavku na okamžité snížení otáček dlouho klesá a dochází ke zpoždění regulace. Řešení je možné provést například porovnáním akčního zásahu se saturační mezí a zamezením integrování v případě překročení rozsahu.

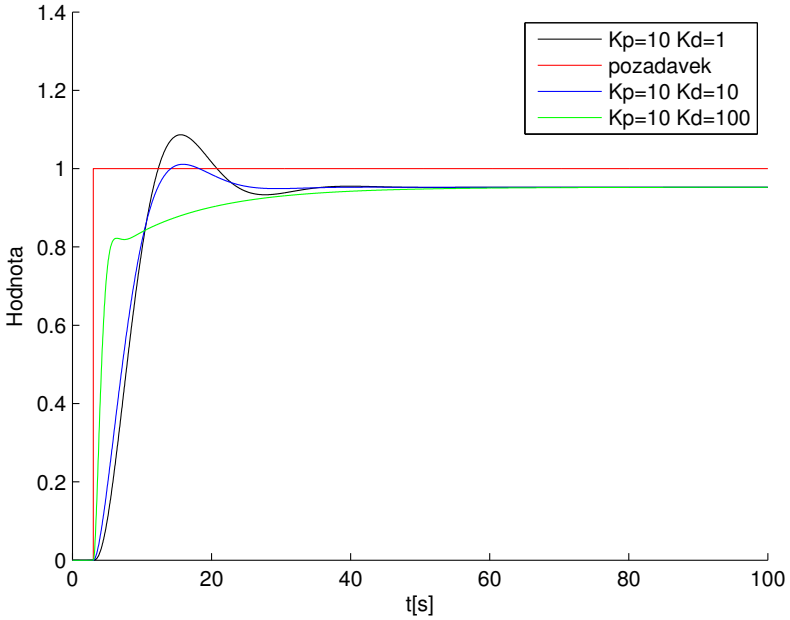


Obrázek 3.4: Blokové schéma s řešením unášení integrační složky

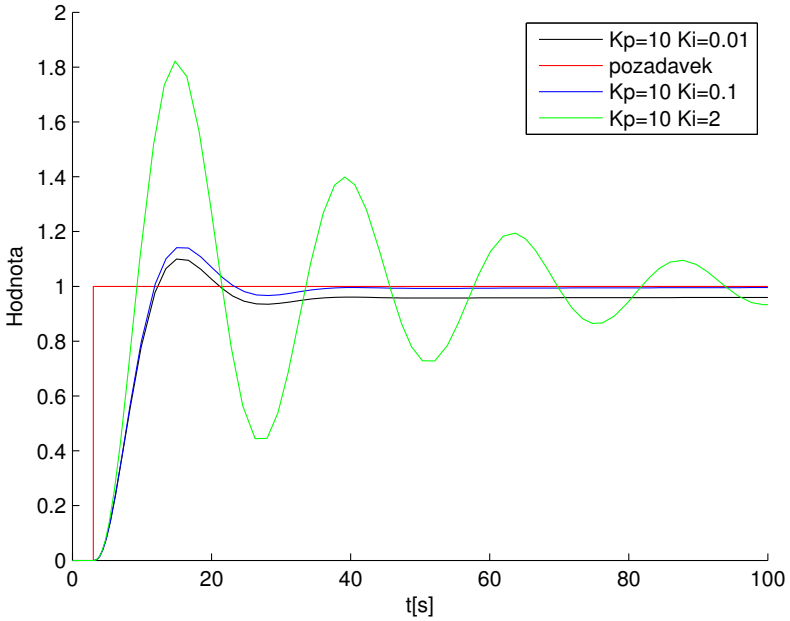


Obrázek 3.5: Zpoždění regulace při unášení integrační složky

Derivační složka je dána derivací regulační odchylky a zrychlí dobu regulace při správném nastavení. Může ale také vést k nestabilitě a zesiluje šum měření [18]. Nestabilita je patrná především při implementaci na reálných systémech, kde měření vždy obsahuje šum získaný nedokonalými senzory.



Obrázek 3.6: Odezva proporcionální a derivační složky



Obrázek 3.7: Odezva proporcionální a integrační složky

Řízení DC motorů kolového robota v této práci bylo prováděno mikrořadičem, takže dále bude popsána implementace diskrétní verze PI regulátoru, jako speciálního případu spojitého PID regulátoru. Derivační složka nebyla využita z důvodu zesílení šumu. Zákon řízení pro diskrétní PI regulátor vypadá takto:

$$u(k) = K_p e(k) + K_i \sum_{i=0}^{k} e(i) \tag{3.4}$$

kde $u(k)$ je výstup regulátoru, $e(k)$ je regulační odchylka, K_p je koeficient pro výpočet proporcionální složky a K_i je koeficient pro výpočet integrační složky. Řídicí smyčka probíhá s periodou 0.1 s. Uvedený vztah je pouze ilustrační, klíčová je implementace regulátoru ve zdrojovém kódu. Na vozítku jsou dva PI regulátory (pro každou stranu jeden). Měření otáček je prováděno na prostředních kolech pomocí kvadraturních enkodérů. Implementace řídicího algoritmu vypadá následovně.

```
e = setpoint - measured_value;
   if ((pid_ < -255 && e < 0) || (pid_ > 255 && e > 0)){
      //do nothing
   }else{
      integral_ = integral_ + ki_*e;
      pid_ = (kp_ * e) + (integral_);
   }

   if(setpoint == 0 && e == 0)
   {
      integral_ = 0;
   }
```

Nejprve se vypočítá regulační odchylka mezi požadovanou a naměřenou hodnotou úhlové rychlosti. První podmínka je zavedena kvůli ošetření saturace. Musí se zabránit tomu, aby se integrovala regulační odchylka, když už je výstup z regulátoru nad hranicí reálné hodnoty (motor už by se při zvýšení akčního zásahu rychleji netočil). Bez saturační podmínky by mohlo docházet k tzv. unášení integrační složky (windup).

Pokud je regulátor v pracovním rozmezí, je nutné vyhodnocovat akční zásah. Proporcionální složka je přímým násobkem regulační odchylky. Integrační složka se počítá z předchozí hodnoty a aktuální regulační odchylky. Při implementaci na reálných systémech je výhodné násobit integračním koeficientem pouze aktuální regulační odchylku. Integrační složka je pak ve stejných jednotkách jako měřená veličina a navíc při změně parametru v

průběhu chodu algoritmu nedochází ke skokové změně akčního zásahu [19]. Poslední podmínka je zavedena, aby při nulové regulační odchylce a nulové požadované úhlové rychlosti regulátor nevytvářel minimální akční zásah, který se už pohybuje v mrtvém pásmu motoru.

Při implementaci je použita aritmetika s plovoucí desetinnou čárkou. Tento přístup má výhodu v jednoduchosti, bohužel není tak přesný a rychlý, protože mikrořadič implementuje tuto aritmetiku softwarově a navíc je použit pouze 32bitový float, což výrazně snižuje přesnost. Pro zvýšení rychlosti a přesnosti výpočtu by se dalo přejít na aritmetiku s pevnou desetinnou čárkou.

4 Model diferenciálně řízeného robota

Pokud si myslíme, že se kola pohybují rychlostí, jakou požadujeme, můžeme přistoupit k dalšímu úkolu. Musíme vytvořit matematický model, který přesně popíše pozici robota v prostoru na základě časových průběhů rychlostí kol. Podle mechanické konstrukce robotů rozlišujeme základní modely podvozku [22]:

Diferenciálně řízený robot je takový, kde se nezávisle pohybuje levá a pravá strana podvozku. Zatáčení se provádí smykem a robot je schopný se otočit na místě.

Ackermannův model řízení je využit například u automobilu. Takový robot se neotočí na místě. Zatáčení se neprovádí smykem, takže spotřebuje méně energie.

Všesměrový robot má speciální kola (obrázek (4.1)), čímž je dosaženo pohybu do všech stran. Takový robot má nejlepší manévrovatelnost.

Obrázek 4.1: Kolo všesměrového robota

4.1 Model diferenciálně řízeného robota

Vozítko Thumper je diferenciálně řízený kolový robot. Pro získání modelu se využívají fyzikální a matematické zákony. Pro zjednodušení odvození se předpokládá, že robot jezdí po dokonale rovném povrchu. Zanedbává se tření a přepokládá se, že jsou všechna kola v kontaktu s podložkou.

Z konstrukčního hlediska je robot schopen provádět pouze pohyb v přímém směru nebo otáčení. Rychlost pohybu v přímém směru vychází ze vzorce pro výpočet obvodové rychlosti kola v_{kola}.

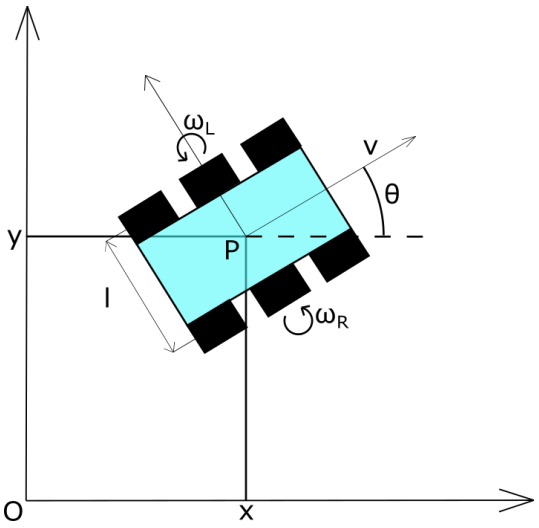
$$v_{kola} = \omega_{kola} r_{kola} \tag{4.1}$$

kde ω_{kola} je úhlová rychlost kola a r_{kola} je poloměr od osy otáčení. Enkodéry měří úhlovou rychlost otáčení prostředních kol na levé (ω_L) a pravé (ω_R) straně robota. Pro rychlost robota v přímém směru (v) platí:

$$v = \frac{\omega_L + \omega_R}{2} \cdot r \tag{4.2}$$

kde r je poloměr kola robota.

Jestliže se robot otáčí na místě, je jeho rychlost v přímém směru nulová. Úhlové rychlosti jsou v takovém případě opačná čísla. Uvedený vzorec (4.2) tomu logicky odpovídá.



Obrázek 4.2: Obrázek robota v souřadném systému

Rychlost otáčení ω robota kolem osy P vypočteme z rozdílné rychlosti otáčení kol na levé a pravé straně. Opět musíme vycházet ze vztahu pro obvodovou rychlost, využijeme rozdílu úhlových rychlostí a také šířku vozítka.

$$\omega = \frac{\omega_L - \omega_R}{l} \cdot r \tag{4.3}$$

kde r je poloměr kola robota a l šířka robota.

Pro spojitý model diferenciálně řízeného robota pak platí:

$$\dot{\theta} = \omega \tag{4.4}$$

$$\dot{x} = v \cdot cos\theta \tag{4.5}$$

$$\dot{y} = v \cdot sin\theta \tag{4.6}$$

kde ω je rychlost otáčení robota, v je rychlost robota v přímém směru, θ je úhel natočení robota, x je pozice robota ve směru osy x a y je pozice robota ve směru osy y.
Jelikož je robot řízen diskrétně se známou periodou vzorkování (Δt), zavedeme přírůstkové proměnné, které budou vyjadřovat změnu za jednu periodu. Aktuální pozici robota zjistíme sečtením jednotlivých přírůstků. Pro celkové natočení robota (θ) platí:

$$\theta(k+1) = \theta(k) + \omega \cdot \Delta t \tag{4.7}$$

S touto znalostí jsme schopni vypočítat aktuální pozici robota.

$$x(k+1) = x(k) + (v \cdot cos\theta) \cdot \Delta t \tag{4.8}$$

$$y(k+1) = y(k) + (v \cdot sin\theta) \cdot \Delta t \tag{4.9}$$

4.2 Struktura řídicího systému

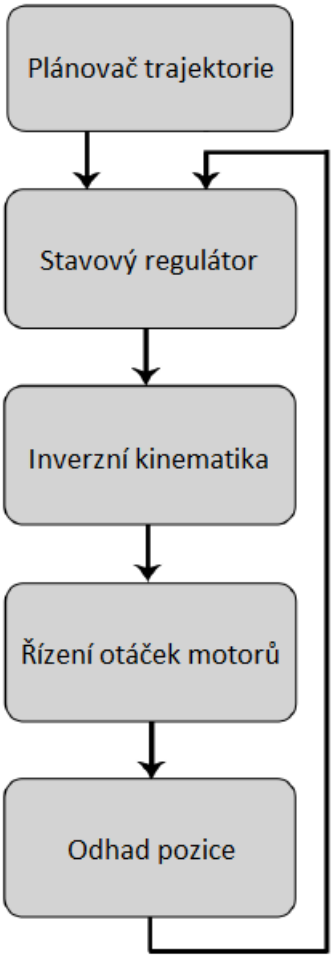
Pro úkol sledování trajektorie je nutné navrhnout obecnou strukturu řídicího systému. Pro lepší představu bylo vytvořeno blokové schéma zobrazené na obrázku (4.3).

Plánovač trajektorie v každém časovém okamžiku vygeneruje požadovanou pozici robota. Porovnáním požadované pozice a aktuální pozice robota je získána regulační odchylka. Tato odchylka je vstupem pro stavový regulátor, který vytvoří požadavek na dopřednou rychlost a rychlost otáčení celého robota. Inverzní kinematikou se tyto požadavky přepočítají na úhlové rychlosti kol podle následujících vzorců:

$$\omega_L = \frac{2v - \omega l}{2r} \tag{4.10}$$

$$\omega_R = \frac{2v + \omega l}{2r} \tag{4.11}$$

kde ω je rychlost otáčení robota, v je rychlost robota v přímém směru, r je poloměr kola robota, l šířka robota, ω_L je rychlost otáčení levých kol robota a ω_R je rychlost otáčení pravých kol robota.
Požadovaných úhlových rychlostí se dosáhne diskrétním PI regulátorem popsaným v kapitole 3. Následně se vypočte odhad aktuální pozice robota podle vztahů (4.8) a (4.9), který se opět porovná s požadovanou pozicí robota poskytnutého plánovačem trajektorie. Jedná se tedy o kaskádní regulaci, kde vnitřní smyčka reguluje otáčky motorů a nadřazená smyčka řídí polohu robota.



Obrázek 4.3: Blokové schéma sledování trajektorie

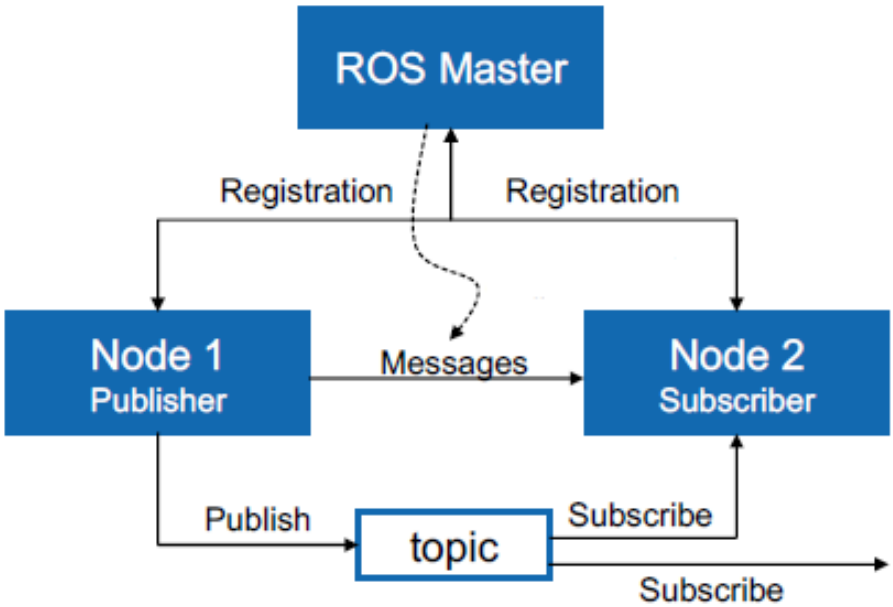
5 ROS

Softwarový rámec ROS (Robot Operating System) je v dnešní době velmi rozšířeným nástrojem pro vývoj robotických systémů. Jeho hlavní výhodou je přenositelnost mezi různými HW platformami. ROS vznikl v roce 2007 ve Stanford Artificial Intelligence Laboratory. Dnes je pod správou Open Source Robotics Foundation [17].

ROS je volně dostupný ke komerčním a vědeckým účelům. Je možné ho volně stahovat, upravovat a sdílet. Existuje několik verzí, v této práci byla použita distribuce ROS Kinetic Kame (dále jen Kinetic). ROS běží na operačních systémech Linux.

5.1 Základy

ROS využívá stromovou strukturu. Kořenový uzel je v systému ROS implementován procesem *roscore*. Na něj se napojují uzly, které vysílají (Publisher) nebo přijímají (Subscriber) zprávy s určitým názvem (topic), a tak komunikují s ostatními uzly. Uzel může například číst data ze senzoru, řídit aktuátor nebo vykonávat nějaký složitější algoritmus (mapování).



Obrázek 5.1: Schéma základního propojení uzlů

Spolu související moduly jsou umisťovány do společného pracovního prostoru, tzv. workspace. Ten obsahuje složku se zdrojovými soubory (src), složku s uloženou mezipamětí a informacemi o konfiguraci (build) a složku s kompilovaným programem (devel). Složky build a devel se vytvářejí automaticky při zavolání příkazu `catkin_make` v terminálu nad složkou workspace. Složka src se dále dělí do tzv. package (balíků). Každý package je vlastně složka, která obsahuje samotný zdrojový kód v souboru s příponou **cpp** nebo **py** (podle programovacího jazyka) a soubory CMakeLists.txt a package.xml, ve kterých jsou informace pro kompilaci a popis package. Uvedený obsah je nezbytně nutný, ale v package můžou být i další soubory. Například s příponou **launch** pro snažší spouštění nebo vlastní definice zpráv.

```
workspace_folder/
  src/
    CMakeLists.txt
    package_1/
      CMakeLists.txt
      package.xml
    ...
    package_n/
      CMakeLists.txt
      package.xml
```

Obrázek 5.2: Struktura workspace

ROS podporuje programovací jazyky C++ a Python. Po vytvoření určitého programu a jeho zkompilování (`catkin_make`) ho lze z terminálu spustit příkazem `rosrun`. Vzájemně propojené ROS moduly mohou běžet i na několika strojích zároveň (například na stolním počítači je zobrazena vizualizace, robot slouží k přijímání dat a zpracování může probíhat na serveru). V takovém případě je spuštěn vždy jen jeden *roscore* a všechny terminály je nutné nastavit tak, aby znaly adresu hlavního uzlu.

5.2 Implementace jednoduchého propojení uzlů

Aby mohla probíhat komunikace mezi uzly, musí být spuštěný centrální proces *roscore*. Vytvoření nejjednodušších komunikujích uzlů bude popsáno v této části.

Ve zdrojovém kódu každého komunikujícího uzlu musí být definovaný nejprve tzv. *NodeHandle*. Tento objekt reprezentuje konkrétní uzel. Pro příjem zpráv se používá objekt *Subscriber*. U objektu *Subscriber* je nutné stanovit název (topic) přijímaných zpráv, jejich typ a musí se definovat `Callback()`, což je metoda, která zprávu přímo zpracuje. Zpracování probíhá okamžitě, jakmile zpráva dorazí. V kódu musí běžet smyčka na příjem zpráv a volání metod typu `Callback()`. Zajistit to lze například příkazem `ros::spin()`. Příklad jednoduchého uzlu pro zachytávání textových řetězců je uveden níže.

```
#include "ros/ros.h"
#include "std_msgs/String.h"

void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}
int main(int argc, char **argv)
{
  ros::init(argc, argv, "listener");
  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("chatter", 1000,
      chatterCallback);
  ros::spin();
  return 0;
}
```

Pro posílání zpráv se používá objekt *Publisher*, u kterého se musí stanovit název (topic) odesílaných zpráv, jejich typ a počet, který se uchová v paměti. Objekt *Publisher* musí zavolat metodu `publish()` s parametrem zprávy, aby došlo k odeslání. Spouštět odesílání lze s nějakou konkrétní frekvencí, neustále nebo pouze jednou. Pro kontrolu lze využít příkaz `ROS_INFO()`, který vypisuje přímo do terminálu zadaný textový řetězec. Další velmi využívaný příkaz je `rostopic list`, který vypíše seznam všech používaných názvů (topiců) zpráv. Aby se vytvořil spustitelný uzel, musí se vhodně upravit konfigurační soubory. Po kompilaci zdrojových kódů (`catkin_make`) je nutné uzly spustit příkazem `rosrun`.

6 Komponenty softwarového rámce ROS pro řízení kolového robota

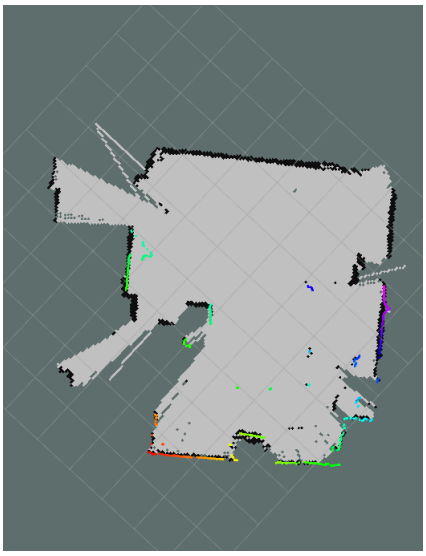
Tato kapitola je věnována konkrétním balíkům systému ROS, které jsou důležité pro úlohu realizace řídicího systému pro autonomního kolového robota.

6.1 Gmapping

ROS nabízí veliké množství již hotových implementací složitých algoritmů. Mezi ty patří i *Gmapping*, který z laserových a odometrických dat vytváří mapu a lokalizuje robota v prostoru. Jedná se tedy o řešení problému SLAM (Simultánní lokalizace a mapování) [15]. Tento problém je velice náročný, neboť pro správnou lokalizaci je potřeba mít bezchybnou mapu a pro získání dobré mapy je zase potřeba mít přesnou pozici robota. *Gmapping* probíhá v reálném čase a využívá tzv. částicový filtr. Každá částice reprezentuje jinou hypotézu trajektorie robota a tím pádem jinou mapu.

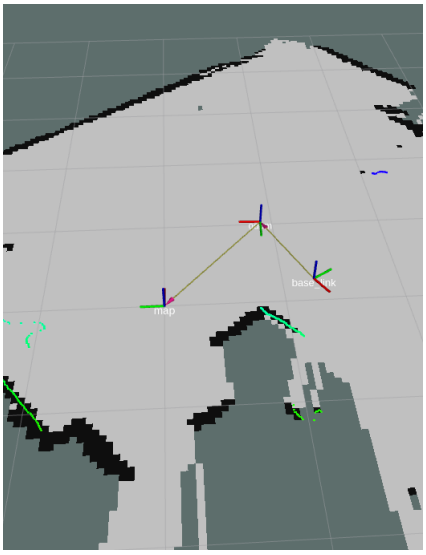
Jakmile se objeví nová informace o změně odometrie a laserových dat, vytvoří se pro každou částici apriorní odhad pozice robota. Vychází se přitom z předchozí pozice robota a pouze z odometrických dat (pohybový model). Potom se provede tzv. scan-matching v okolí prvotního odhadu pozice robota (pozorovací model). V tomto kroku se využijí laserová data a porovnají se s odhadem mapy pro každou částici. Apriorní odhad pozice robota se tím výrazně zpřesní. Následně se aproximuje aktuální pravděpodobnostní rozložení pozice robota Gaussovým rozložením (musí se vzorkovat, spočítat střední hodnota a kovariance). Vybere se nejpravděpodobnější pozice robota (pro každou částici) a přepočítají se váhy všech částic. Aktualizuje se mapa a zkontroluje se, jestli má dojít k převzorkování. Během převzorkování jsou vymazány nejméně pravděpodobné částice [15].

Gmapping má obrovskou výhodu v tom, že nepotřebuje mnoho částic, takže může provádět SLAM v reálném čase.



Obrázek 6.1: Výsledná mapa s lidarovými daty

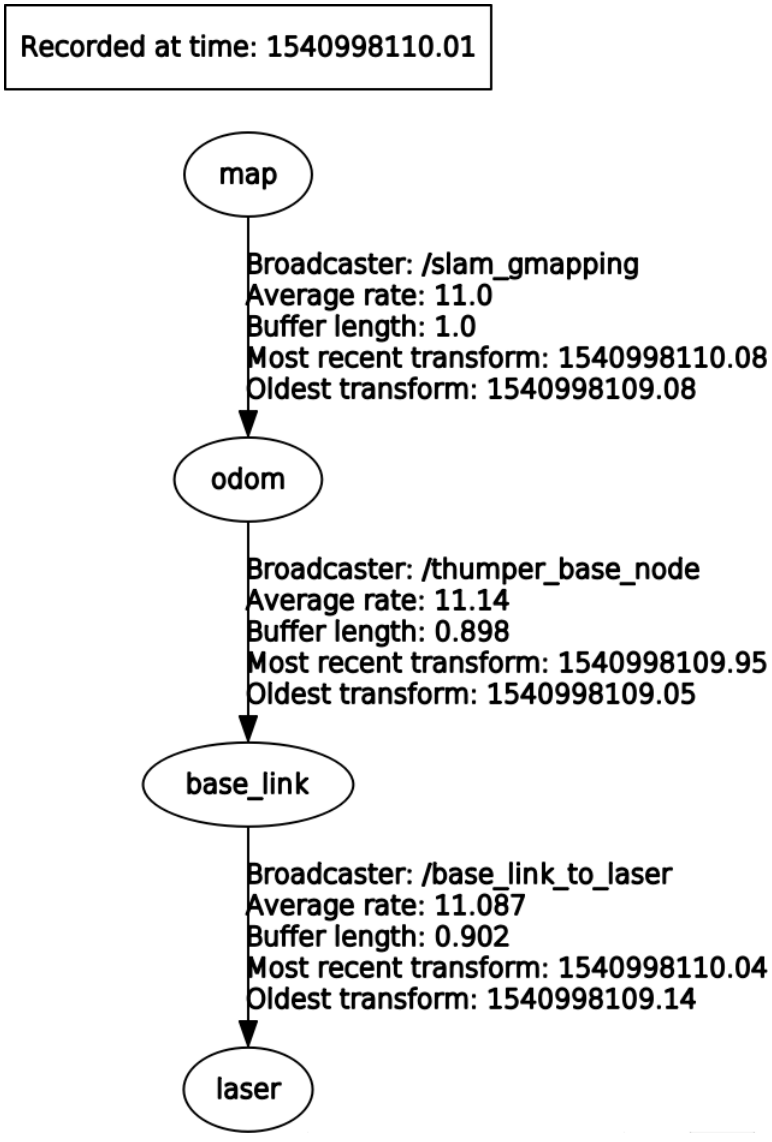
Gmapping pro svoji funkci potřebuje správně nakonfigurovat transformace souřadnic (pomocí balíku *tf*), které slouží k převodu dat mezi souřadnicovými rámci. Na obrázku (6.2) je příklad souřadných rámců robota Thumper.



Obrázek 6.2: Ilustrace souřadných rámců

Gmapping potřebuje zavést transformaci ze souřadnicového systému lidaru (LASER) na souřadnicový systém podvozku robota (BASE_LINK). Tato transformace je statická, protože lidar je pevně připevněn k robotovi. Statickou transformaci lze definovat v souboru s příponou **launch**. Další po-

žadovaná transformace je z odometrického rámce (ODOM) na podvozek robota (BASE_LINK). Tato transformace vyjadřuje pohyb robota po podložce, takže není statická. Je definována pomocí objektu *TransformBroadcaster*, který poskytuje metodu `sendTransform()` naplněnou vypočtenou polohou robota. Příkazem `tf_monitor` lze ověřit správnou funkčnost konkrétní transformace. *Gmapping* vytváří mapu jako zprávu a přidává další tranformaci mezi mapou (MAP) a odometrií (ODOM). V systému ROS je možné zobrazit všechny transformace souřadnic v jednom schématu (6.3).



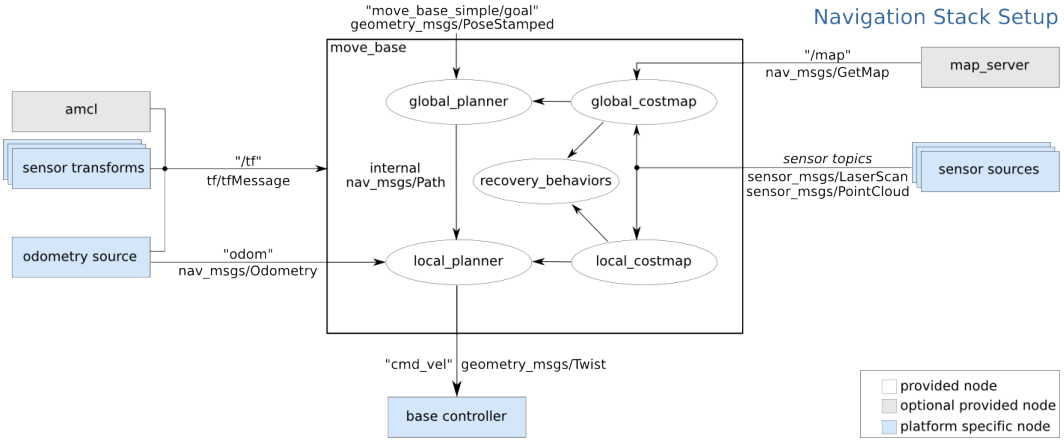
Obrázek 6.3: Strom všech transformací

6.2 Navigation stack

Jak již bylo řečeno výše, řízení kolových robotů je většinou řešeno kaskádně. Tento přístup zvyšuje celkovou přesnost regulace. Nižší stupeň řídí požadované otáčky kol a vyšší stupeň řízení zajišťuje regulaci na požadovanou polohu a orientaci. O řízení na požadovanou polohu se stará právě *Navigation stack* [11]. Zjednodušeně se dá říci, že jako vstup do *Navigation stack* poskytneme odometrická a lidarová data a získáme požadavek na rychlost otáčení kol.

Pro zprovoznění tohoto výkonného nástroje je potřeba znát přesný kinematický model robota. Dále je nutné používat lidar, aby se robot mohl vyhnout případným nepředvídatelným překážkám. *Navigation stack* je totiž schopný přeplánovat trajektorii, pokud se v cestě náhle vyskytne překážka. Nezbytnou součástí pro správnou funkci je také publikace transformací mezi souřadnicovými systémy. *Navigation stack* vyžaduje odometrická data, a to jak ve formě klasické zprávy (`nav_msgs`), tak ve formě transformace souřadnic.

Na obrázku (6.4) je schéma balíku *Navigation stack*. Vnitřní uzly balíku jsou bílé. Některé vnější uzly jsou pro fungování celého systému nezbytné (modré), jiné jsou pouze doplňkové (šedé).

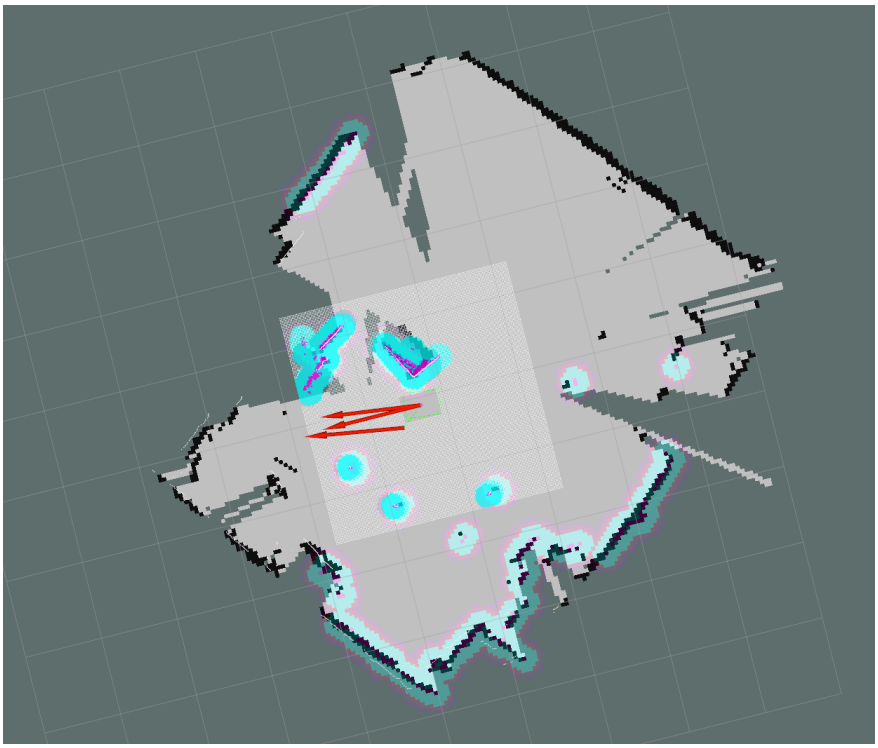


Obrázek 6.4: Schéma *Navigation stack* [11]

Mezi uzly jsou zavedeny informační vazby. Většinou se informace vyměňují pomocí zpráv určitého typu. V další části bude podrobně popsána funkce základních komponent *Navigation stack*.

6.2.1 Costmap

Costmap je 2D mapa (6.5), ve které jsou jednotlivým bodům přiřazeny váhy. *Navigation stack* využívá globální a lokální *costmap*. Pro vytváření *costmap* se využívají data z lidaru, protože poskytují přesnou informaci o vzdálenosti k překážkám. Okolo každé překážky se vytváří ochranný obal (inflation). Jeho velikost lze nastavit při konfiguraci. Body v mapě lze podle ceny rozdělit do pěti kategorií. Smrtící (lethal) jsou body na mapě, kde je skutečná překážka. Tyto body mají největší cenu. Pokud se robot ocitne v tomto prostoru, zcela jistě narazil. Další úroveň jsou inscribed body. Jsou dál od skutečné překážky, ale pořád se jedná o kritickou oblast. Pokud střed robota stojí na takovém bodu, robot narazil. Následuje oblast hraniční, kde robot už nemusí narazit, ale záleží na jeho orientaci. Volný prostor má nejmenší cenu. Robot se v něm může bez problémů pohybovat. Poslední kategorií jsou neznámé body [6]. Může to být například prostor, který je mimo dosah lidaru.

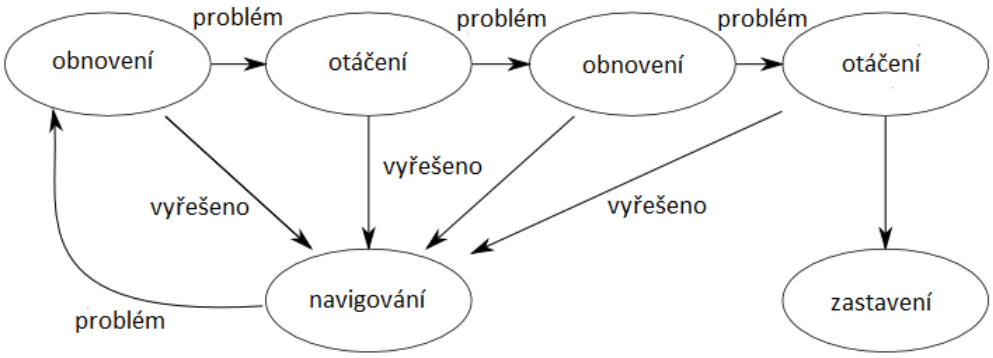


Obrázek 6.5: Zobrazení *costmap* v testovaném prostředí

Při nastavování parametrů *costmap* je nutné definovat tzv. footprint, který vyjadřuje velikost podvozku robota. Postupně se zadávají souřadnice bodů na obvodu robota, přičemž počátek soustavy souřadnic je v těžišti robota. Pro roboty s kruhovou podstavou se nastaví jejich poloměr.

6.2.2 Move base

Tento balík je klíčový pro fungování celého nástroje *Navigation stack*. Při jeho správném nastavení dosáhneme toho, že robot dojede na místo určení se zadanou přesností (vzdálenost od bodu a úhel natočení). Ke splnění tohoto obtížného úkolu jsou využívány dva plánovače. Globální plánovač se stará o kompletní trajektorii, lokální plánovač má na starost nejbližší úsek před robotem. Balík *Move base* propojuje oba dva plánovače dohromady a stará se i o udržování *costmap*, které oba dva plánovače používají. Tento balík publikuje zprávy na požadovanou rychlost jízdy vpřed a rychlost otáčení robota a přijímá zprávy na požadovanou pozici robota. Může se stát, že se robot v nějakém místě zasekne (vlivem pohybu překážek by bylo nebezpečné pokračovat dál). V takovém případě *Move base* řeší tuto situaci podle obrázku (6.6).



Obrázek 6.6: Stavový automat pro řešení nestandardních situací [12]

Snaží se nejprve obnovit lokální *costmap*. Pokud tento postup nepomůže, začne se robot otáčet a zkoumat prostor okolo sebe znovu. Ani to ale nemusí vždy pomoci. Proto se robot snaží stále agresivněji vyčistit prostor okolo sebe. V poslední fázi může být navigace dokonce zastavena, aby nedošlo k nárazu s překážkou.

6.2.3 Global planner

Tento uzel slouží k plánování kompletní trajektorie od aktuální pozice až na požadovanou cílovou pozici. K dispozici jsou tři různé globální plánovače: carrot planner, navfn a global planner.

Nejjednodušší je carrot planner. V případě, že je cílová pozice umístěna do prostoru nějaké překážky, tento plánovač automaticky přesune cílový bod na nejbližší dosažitelné místo, což může být v některých aplikacích výhodné

(protože neznáme prostředí). V komplikovaných vnitřních prostorech tento plánovač není příliš vhodný [23].

Plánovač navfn využívá Dijkstrův algoritmus (konkrétně rychlou interpolační navigační funkci), takže vybere trajektorii s nejmenší cenou (nejmenší vzdáleností mezi počátečním a cílovým bodem).

Poslední zmiňovaný plánovač je nejvíce komplexní. Využívá vylepšený Dijkstrův algoritmus. Zlepšení spočívá ve využití heuristiky. Tento algoritmus se nazývá A* [3]. Dále je možnost využít i jiné algoritmy pro návrh trajektorie. V základním nastavení *Navigation stack* používá plánovač navfn.

6.2.4 Local planner

Lokální plánovač vytváří z lokální *costmap* a globálního plánu trajektorie příkazy na požadované rychlosti (v přímém směru a otáčení). Dostupné jsou dvě implementace algoritmů - Trajectory Rollout a Dynamic Window Approach (DWA). Obě dvě implementace fungují podobně. V blízkém okolí robota jsou simulovány a zaznamenávány trajektorie pro různé volby rychlostí a směrů. Délka simulace a krok vzorkování rychlostí jsou nastavitelné. K výpočtu se používají ještě zadané limity rychlosti (minimum a maximum) a také omezení na zrychlení robota. Každá trajektorie je ohodnocena podle toho, jak je daleko od překážek. Čím blíže jsou překážky, tím je větší riziko nárazu a tím horší bude ohodnocení trajektorie. Trajektorie, které narazí do překážky, jsou odebrány ze seznamu možných trajektorií hned. Dále záleží na blízkosti ke globální trajektorii. V lokální costmapě se vytvoří mřížka, do které se překreslí úsek globální požadované trajektorie. Do této mřížky jsou následně umístěny simulované trajektorie. Při překrytí obou trajektorií je ztráta nulová, jinak se započítá penalizace pro danou simulovanou trajektorii. V posledním kroku algoritmu je vybrána simulovaná trajektorie s nejlepším ohodnocením a rychlosti pro její přesné kopírování jsou poslány na výstup (vstup nižšího řízení). Rozdíl implementací je v tom, že se různě přistupuje ke vzorkování dopředné simulace. DWA by měl být efektivnější z hlediska výpočetní složitosti.

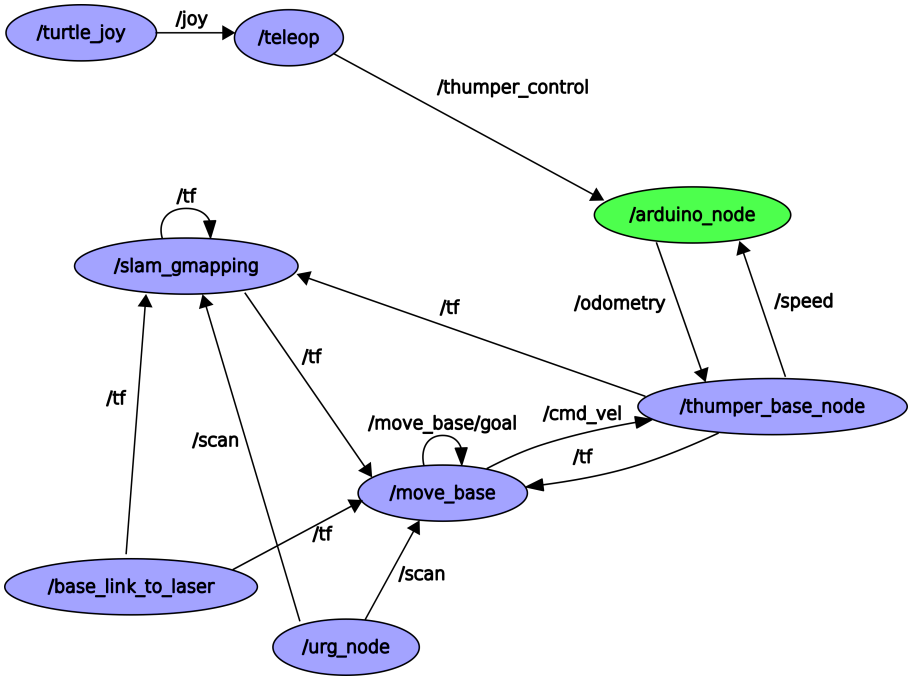
V lokálním plánovači může docházet k oscilacím (robot se pohne a pak se snaží jet zase zpátky, například při průjezdu dveří). Proto je zakázaný návrh trajektorií v opačném směru k aktuálně prováděnému pohybu. Tento směr může být odblokovaný až po ujetí nastavené vzdálenosti, kdy už je jasné, že nejde o oscilaci, ale o požadovaný směr pohybu v rámci dosažení cíle.

7 Použití softwarového rámce ROS pro řízení kolového robota

V této kapitole bude popsáno konkrétní využití softwarového rámce ROS pro realizaci řídicího systému kolového robota. ROS byl využit z toho důvodu, že obsahuje veliké množství balíků vhodných pro řešení této úlohy.

7.1 Propojení uzlů

Softwarový rámec ROS nabízí možnost zobrazit aktuálně spuštěné uzly i se vzájemnými vazbami pomocí příkazu `rosrun rqt_graph rqt_graph`. Na obrázku (7.1) je upravené schéma získané tímto způsobem. Oválné struktury symbolizují spuštěné uzly, šipky naznačují směr posílání dat, jejich popis označuje název (topic) posílaných zpráv. Zelené uzly jsou spuštěné na řídicí desce T-REX, modré uzly jsou spuštěné na mikropočítači.



Obrázek 7.1: Schéma uzlů na robotické platformě

7.2 Řídicí deska T-REX využívající ROS

Řídicí deska T-REX je kompatibilní s řídicí deskou Arduino a je popsána v kapitole 2. ROS nabízí dokumentaci k práci s řídicí deskou Arduino, ze které se vycházelo i v této práci [10].

Motory jsou řízeny přímo přes řídicí desku T-REX, proto je nutné systém ROS implementovat do firmwaru pro tento mikrokontrolér. Především je nutné vygenerovat složku, ve které budou umístěné hlavičkové soubory pro všechny typy používaných zpráv, jádro softwarového rámce ROS a sériové komunikace.

Příkazem `rosrun rosserial_arduino make_libraries.py` v terminálu zajistíme vytvoření potřebných hlavičkových souborů. Ve zdrojovém kódu mikrokontroléru pak musíme importovat vygenerované hlavičkové soubory příkazem `#include <ros.h>`.

Zdrojový kód mikrokontroléru je rozdělen do dvou hlavních částí (`setup()` a `loop()`). Před částí `setup()` jsou inicializovány proměnné pro uchování stavu enkodérů, objekty pro řízení motorů a jsou založeny objekty systému ROS (dojde k vytvoření uzlu, inicializaci objektů *Subscriber/Publisher* a definování metod typu `Callback()`). Ukázka práce s objekty systému ROS je uvedena níže.

```
ros::NodeHandle_<ArduinoHardware,3,2,128,256> nodeHandle;
ros::Subscriber<thumper_msgs::PID> pid_sub("/pid", pidCallback);
ros::Subscriber<thumper_msgs::Speeds> speed_sub("/speed",
    speedCallback);
ros::Subscriber<geometry_msgs::Twist> cmd_sub("/thumper_control",
    commandCallback);
thumper_msgs::odom odometry;
ros::Publisher odom_pub("/odometry", &odometry);
thumper_msgs::pidControl pidc;
ros::Publisher pidc_pub("/pidc", &pidc);
```

Zajímavé je použití výrazu `ArduinoHardware` při vytváření uzlu. Je tak dosaženo omezení velikosti vstupních a výstupních bufferů pro uchovávání zpráv. Klasické zprávy systému ROS jsou velmi obsáhlé a paměti na řídicí desce T-REX není neomezené množství.

V části `setup()` dochází k inicializaci enkodérů a nastavení přerušovacího systému. Především se jedná o správné nastavení časovače, který v pravidelných intervalech vyvolává přerušení. Časovač obsahuje čítač, který inkrementuje svoji hodnotu s určitou frekvencí. Přerušení nastane, jestliže čítač dosáhne specifické hodnoty nastavené v porovnávacím registru. Potom

se čítač nastaví opět na nulu a znovu začne inkrementovat svoji hodnotu.

```
OCR1A = 6249;
TCCR1B |= (1 << WGM12);
TCCR1B |= (1 << CS12);
TIMSK1 |= (1 << OCIE1A);
```

Na první řádce výše uvedeného zdrojového kódu je nastavena specifická hodnota v porovnávacím registru, na druhé řádce je nastaven tzv. CTC mode (dojde k vynulování čítače při dosažení hodnoty v porovnávacím registru), dále je nastaven tzv. prescaler, který sníží frekvenci čítače, a nakonec se povolí přerušení od časovače.

Smyčka `loop()` probíhá opakovaně a slouží k řízení otáček motorů. Uzel je oživen příkazem `nodeHandle.spinOnce()`. Kontroluje se napětí baterie. Při příliš nízké hodnotě napětí dojde k zastavení motorů, zvukové signalizaci a výpisu do terminálu (zdrojový kód uvedený níže). Nastavení otáček motorů je provedeno objektem `Controller`.

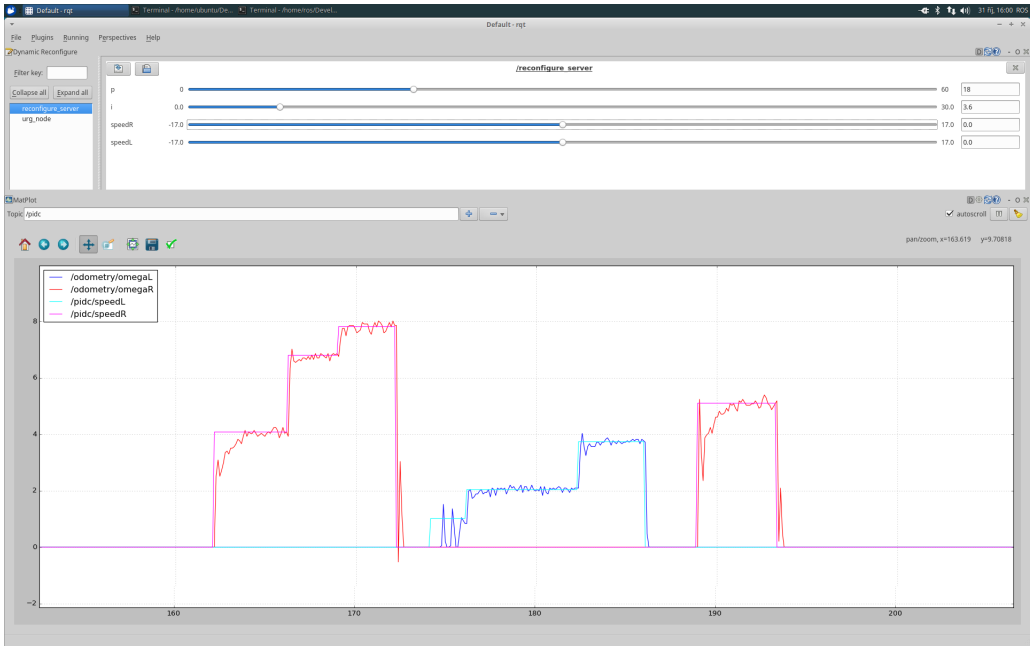
```
void Shutdown()
{
 motorL_controller.spin(0);
 motorR_controller.spin(0);
 motorL_controller.beep(5);
 nodeHandle.loginfo("//====Mam vybite baterky====");
 while(1) { }
}
```

Řídicí deska T-REX přijímá zprávy z gamepadu (ruční režim přeruší regulaci DC motorů a funguje jako bezpečnostní pojistka). Dále jsou přijímány zprávy na požadovanou úhlovou rychlost kol robota a změnu parametrů diskrétního PI regulátoru (využito pouze s modulem *rqt* a parametrickým serverem). Odesílají se informace o aktuálních úhlových rychlostech a vnitřních proměnných hodnotách regulátoru.

7.3 Parametrický server a rqt

Pro nalezení optimálních parametrů PI regulátoru (popsaného v kapitole 3) bylo potřeba najít jiný způsob než ruční změnu kódu, kompilaci a nahrání do řídicí desky, jelikož je tento postup příliš zdlouhavý. ROS poskytuje nástroj pro dynamickou změnu prarametrů. Pro vykreslení chování regulátoru byl využit zobrazovací nástroj *rqt*. Tato součást systému ROS je ideální pro vytváření jednoduchých grafických rozhraní. Jednoduše lze přidat již hotové pluginy (např. grafy, parametrický server) nebo vytvořit vlastní grafické prvky (pomocí frameworku Qt) a uložit vše do jedné konfigurace. Při dalším spuštění stačí načíst uloženou konfiguraci a otevře se stejné grafické rozhraní.

Vytvořené grafické rozhraní pro ladění parametrů PI regulátorů je na obrázku (7.2) a spustí se příkazem:
`roslaunch thumper_params pid_configure.launch`



Obrázek 7.2: Grafické rozhraní pro ladění parametrů PI regulátorů

V horní části rozhraní lze měnit parametry PI regulátoru a požadované rychlosti levého a pravého kola pomocí posuvníků. V dolní části je graf zobrazující hodnoty měřené na robotovi v reálném čase.

Parametrický server zajišťuje čtení a odesílání hodnot z posuvníků do řídicí desky T-REX. Nejdříve je nutné definovat proměnné parametry (p, i, speedL, speedR), jejich typ, počáteční hodnoty a jejich přípustné rozmezí.

Toto nastavení musí být obsaženo v souboru s příponou **cfg**. Ukázka zdrojového kódu je níže.

```
from dynamic_reconfigure.parameter_generator_catkin import *
gen = ParameterGenerator()

gen.add("p",  int_t,   0, "An Integer parameter", 15, 0, 60)
gen.add("i", double_t, 0, "A double parameter", .5, 0, 30)
gen.add("speedR",  double_t,   0, "A parameter", 0, -17, 17)
gen.add("speedL",  double_t,   0, "A parameter", 0, -17, 17)
```

Dále je potřeba vytvořit kód pro obsluhu zpráv (zdrojový kód uveden níže). Je to velmi podobné jako při vytváření komunikace mezi jinými uzly. Opět se musí založit objekt *Publisher* a definovat `Callback()`, který nasbírané informace odešle formou zprávy do řídicí desky T-REX. Zajímavé je použití entity `dynamic_reconfigure`, která sleduje hodnoty definované v souboru s příponou **cfg** a při jejich změně vyšle požadavek na poslání dat. `Callback()` získává při volání dva odlišné objekty *Publisher* a reaguje na oba zároveň. Pokud se hodnota v jedné ze zpráv nezměnila, tak se odešle sice nová zpráva, ale se stejnými hodnotami jako předtím.

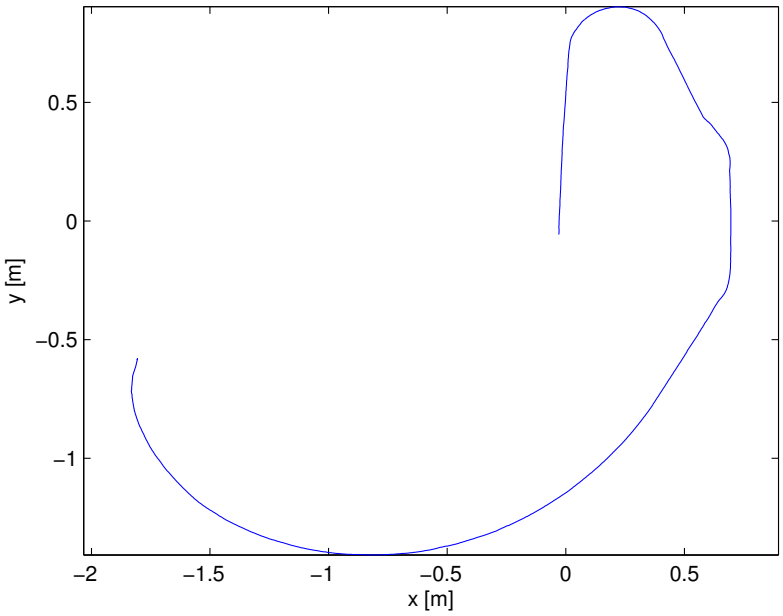
```
dynamic_reconfigure::Server<thumper::PIDConfig> server;
dynamic_reconfigure::Server<thumper::PIDConfig>::CallbackType f;
f =
    boost::bind(&callback,boost::ref(pub_pid),boost::ref(pub_speed),
    _1, _2);
server.setCallback(f);

void callback( ros::Publisher &pub_pid,ros::Publisher
    &pub_speed,thumper::PIDConfig &config, uint32_t level)
{

  thumper_msgs::PID pid;
  thumper_msgs::Speeds speed;
  pid.p=config.p;
  pid.i=config.i;
  speed.speedL=config.speedL;
  speed.speedR=config.speedR;
  pub_pid.publish(pid);
  pub_speed.publish(speed);

}
```

8 Experimenty

Tato kapitola je věnována experimentům, které slouží k ověření funkčnosti robota. První podkapitola popisuje experiment zaměřený na zjištění kvality odometrie získané pomocí enkodérů. Druhá podkapitola popisuje experiment, který slouží k ověření funkčnosti diskrétního PI regulátoru. Třetí podkapitola popisuje experiment sloužící k porovnání naplánované a vykonané trajektorie a ověření dosažení zadaného cíle.

8.1 Ověření kvality odometrie

Během tohoto experimentu byl robot řízen ručně dálkovým ovládáním a zaznamenávaly se úhlové rychlosti kol měřené pomocí enkodérů na robotovi s periodou 0.1 s. Pro porovnání byla stejná trajektorie měřená současně sys-



Obrázek 8.1: Vykonaná trajektorie během experimentu

témem Vicon (8.1), který slouží k přesnému zachycení předmětu v prostoru. Je složen z osmi vysokorychlostních kamer, které pracují v infračerveném spektru. Systém Vicon měří s periodou 0.01 s. Pro sběr dat lze použít ROS, jelikož existuje balík *Vicon bridge*, který posílá zprávy o aktuální pozici (souřadnice x, y, z), natočení (kvaternion udávající směr) a čas, který je společný

jak pro vozítko tak pro Vicon. Veškerou komunikaci v rámci systému ROS lze nahrávat pomocí příkazu `rosbag`.

Přepočtem dat ze systému Vicon lze zpětně získat úhlové rychlosti kol robota. Nejprve je nutné převést kvaternion na Eulerovy úhly (funkce v Matlabu). Úhel natočení robota θ je pak využit v následujících vzorcích:

$$x_L = x + \frac{l}{2} cos\theta \tag{8.1}$$

$$x_R = x - \frac{l}{2} cos\theta \tag{8.2}$$

$$y_L = y + \frac{l}{2} sin\theta \tag{8.3}$$

$$y_R = y - \frac{l}{2} sin\theta \tag{8.4}$$

kde l je šířka robota, x a y jsou pozice změřené systémem Vicon, x_L a y_L jsou souřadnice levého prostředního kola a x_R a y_R jsou souřadnice pravého prostředního kola.

Úhlová rychlost pravého kola ω_R se pak spočítá takto:

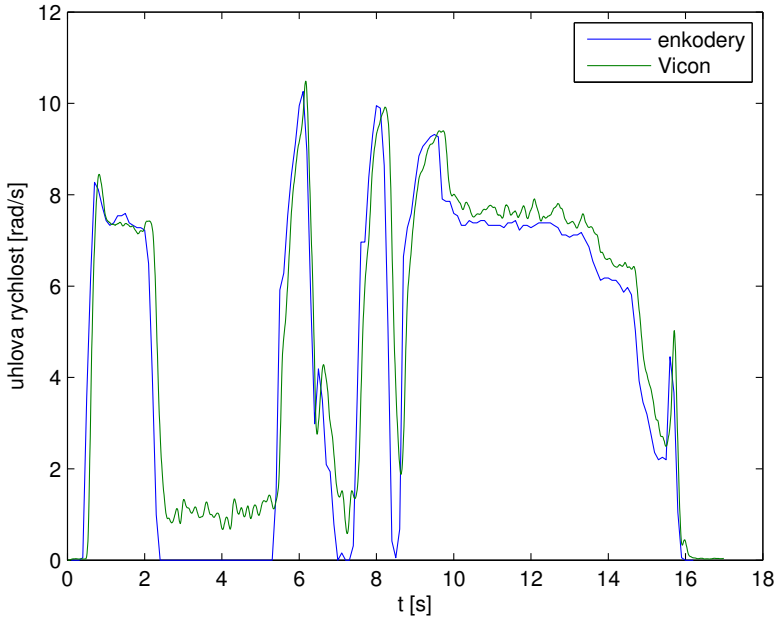
$$\omega_R = \frac{\sqrt{(x_{R2} - x_{R1})^2 + (y_{R2} - y_{R1})^2}}{\Delta t r} \tag{8.5}$$

kde x_{R2} a y_{R2} jsou souřadnice pravého prostředního kola na konci periody, x_{R1} a y_{R1} jsou souřadnice pravého prostředního kola na začátku určité periody, Δt je perioda měření a r je poloměr pravého kola. Obdobný vztah platí i pro levé kolo.

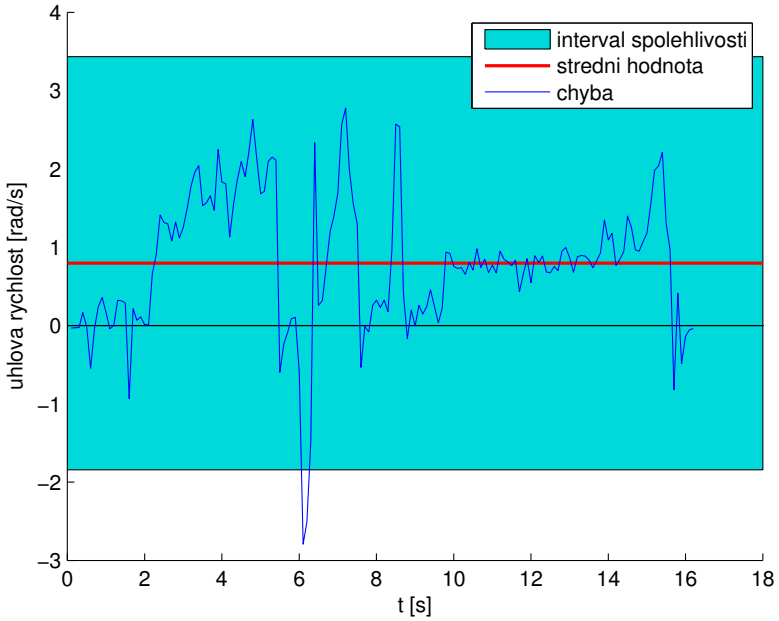
Vyhodnocení bylo provedeno výpočtem rozdílu úhlových rychlostí měřených pomocí enkodérů a úhlových rychlostí získaných ze systému Vicon (obrázky (8.3) a (8.5)). Z tohoto rozdílu byla vypočtena střední hodnota a směrodatná odchylka.

	levé kolo	pravé kolo
střední hodnota úhlové rychlosti	0.796	-0.427
směrodatná odchylka úhlové rychlosti	0.879	0.645

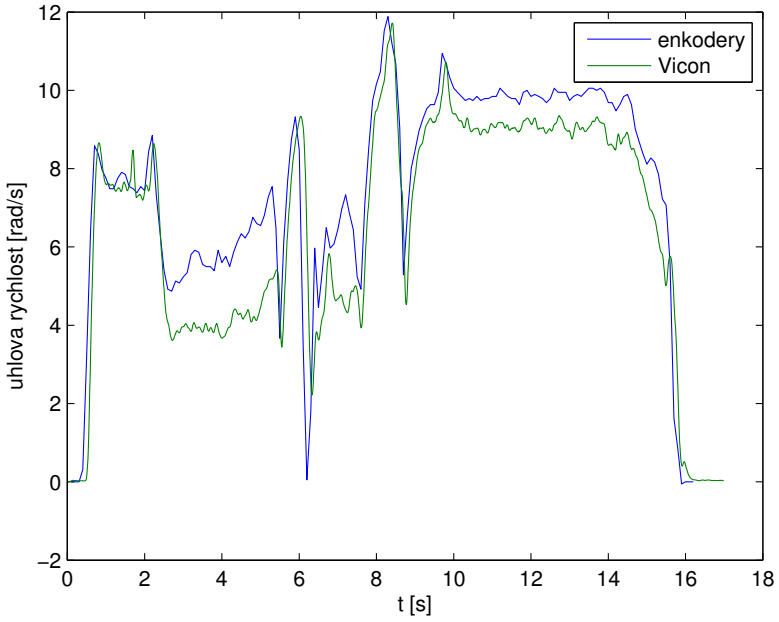
Tabulka 8.1: Výsledky prvního experimentu



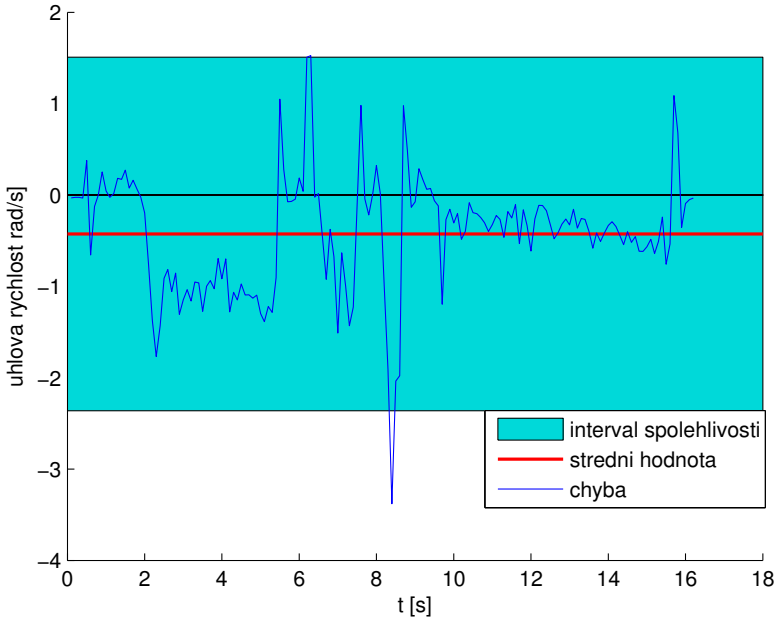
Obrázek 8.2: Úhlová rychlost levého kola



Obrázek 8.3: Vyhodnocení chyby odometrie levého kola



Obrázek 8.4: Úhlová rychlost pravého kola

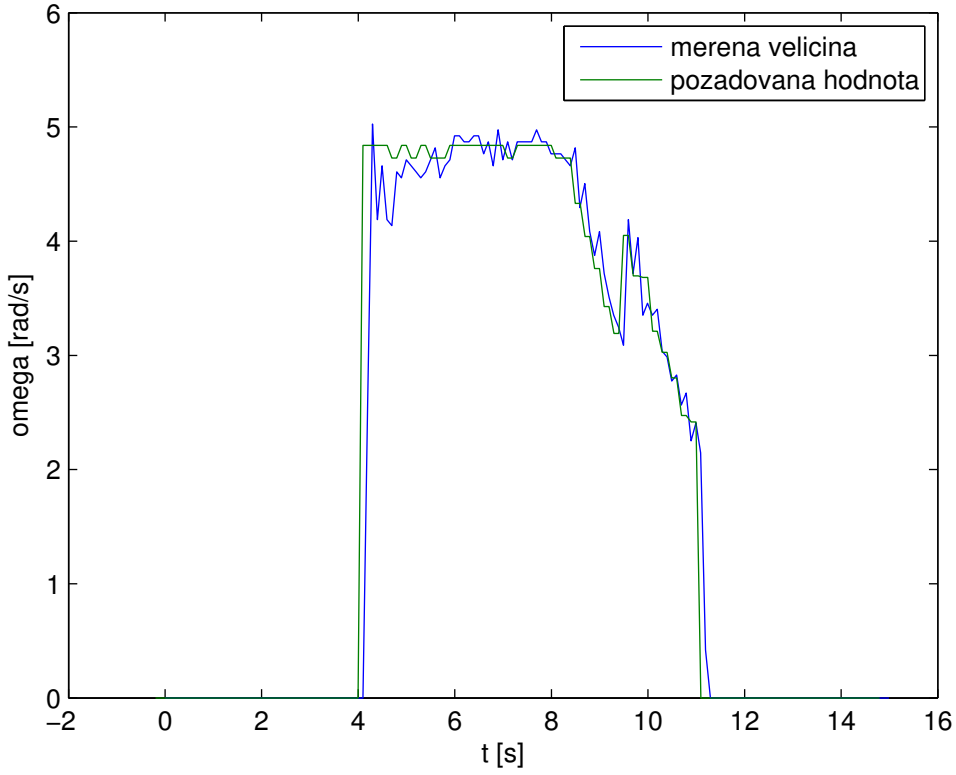


Obrázek 8.5: Vyhodnocení chyby odometrie pravého kola

Z obrázků (8.2) a (8.4) je patrné, že data ze systému Vicon obsahují šum. Ten je způsobený měřením s kratší periodou. K jeho odstranění byl využit frekvenční filtr typu dolní propust. Je patrné, že měření odometrie z enkodérů se dokonale neshoduje s měřením ze systému Vicon. Odchylky mohou být způsobené prokluzováním kol, nedokonalou geometrií robota a omezenou přesností enkodérů.

8.2 Zpětnovazební řízení otáček motorů

V tomto experimentu bylo ověřeno, že diskrétní PI regulátor skutečně nastavuje otáčky motorů na požadovanou hodnotu. Během experimentu jel robot pouze rovně, takže je profil požadované rychlosti jednoduchý (8.6).

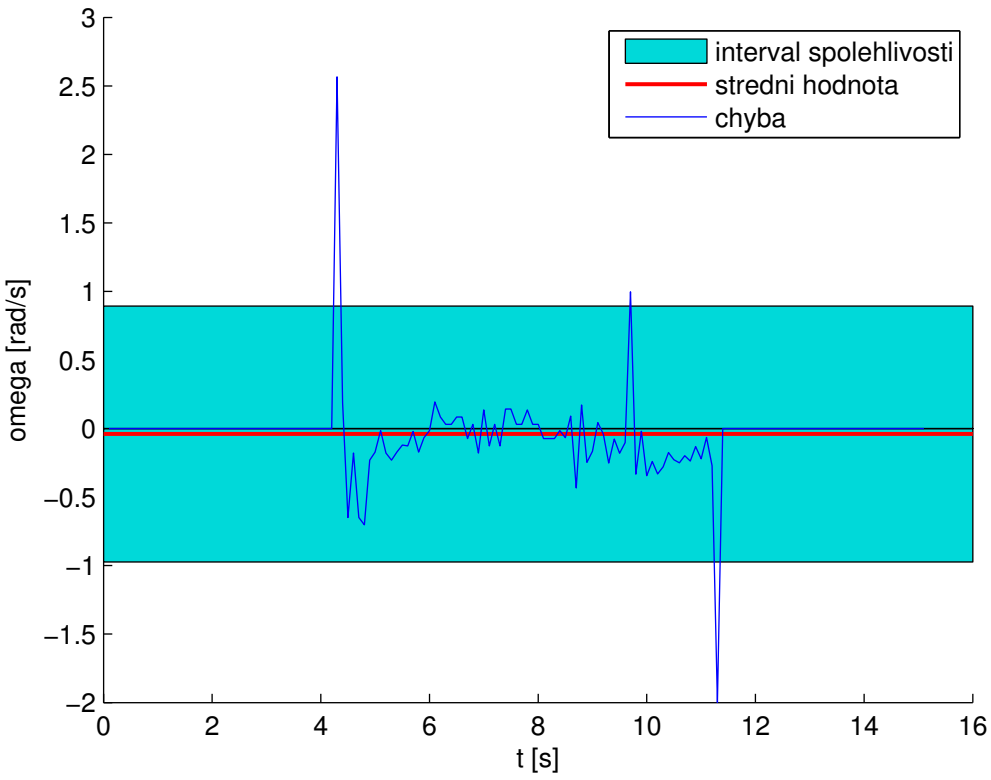


Obrázek 8.6: Úhlové rychlosti při jízdě rovně

Při řízení nedochází k překmitu, doba regulace je přijatelná. Ustálená hodnota mírně osciluje v rámci přijatelných mezí. Při působení chyby (zvýšené tření během otáčení robota namístě) regulátor dočasně zvýší výkon, aby úhlové rychlosti odpovídaly požadavkům. Při agresivnějším nastavení

parametrů regulátoru by se zrychlila doba regulace, ale také by se zvýšilo kmitání okolo požadované hodnoty. Je potřeba zvolit určitý kompromis. Řízený systém navíc není lineární, takže při malém vybuzení se chová jinak než při větší regulační výchylce. Parametry regulátoru byly zvoleny dostatečně robustně, aby nedocházelo k oscilacím.

Vyhodnocení bylo provedeno výpočtem rozdílu požadované a naměřené úhlové rychlosti kola (pomocí enkodérů). Z tohoto rozdílu byla vypočtena střední hodnota a směrodatná odchylka (8.7).



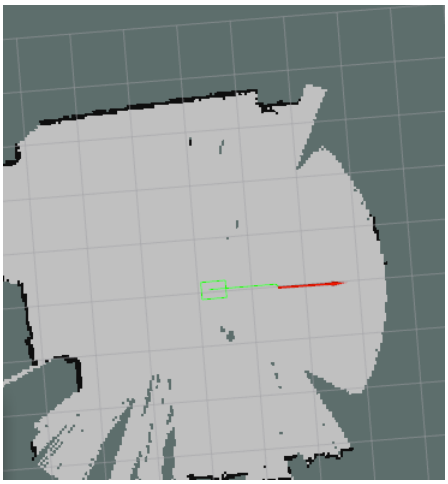
Obrázek 8.7: Vyhodnocení chyby regulace

K největší odchylce od požadované hodnoty dochází při náhlých změnách (začátek a konec pohybu). Střední hodnota se pohybuje blízko nulové hladiny (-0.04). Směrodatná odchylka je také nízká (0.31).

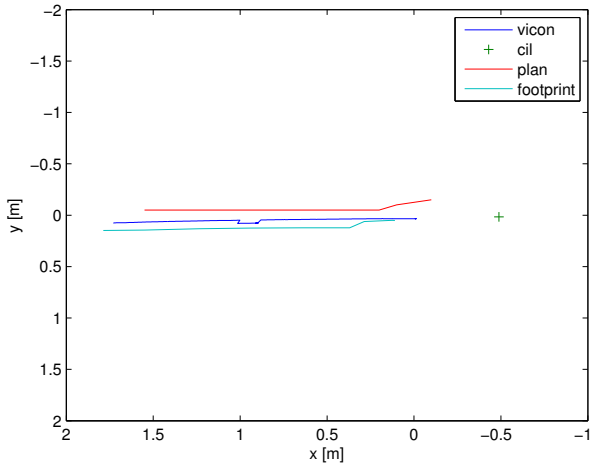
8.3 Řízení s využitím Navigation stack

Tento experiment slouží k porovnání naplánované a realizované trajektorie a dále slouží k ověření dosažení zadaného cíle. Pro naměření skutečně provedené trajektorie byl použit systém Vicon. Naplánovaná trajektorie lze zobrazit ve vizualizačním nástroji *Rviz* (součást ROS).

V prvním případě byl robot umístěn na místo o souřadnicích [1.73; 0.08] a měl za úkol dojet na místo o souřadnicích [-0.49; 0.02].



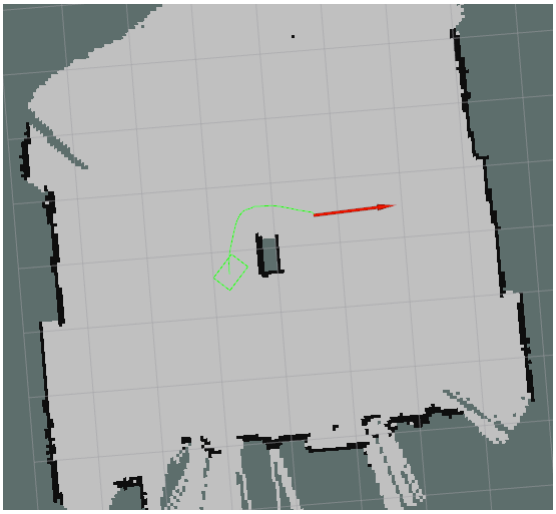
Obrázek 8.8: Naplánovaná rovná trajektorie, cílový bod (červená šipka) a aktuální pozice robota (zelený obdélník) v mapě vytvořené balíkem Gmapping



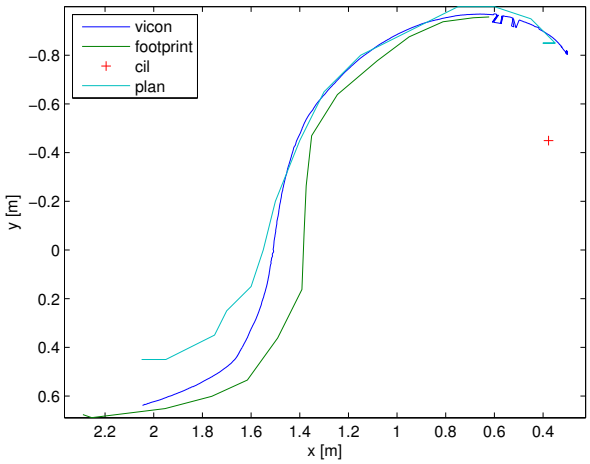
Obrázek 8.9: Výsledek měření prvního případu (footprint vyjadřuje pohyb těžiště robota v mapě vytvořené balíkem Gmapping)

Robot dojel se zadanou přesností (0.4 m) do cílového bodu. Trajektorie se liší z důvodu nepřesnosti ve vytváření mapy a nepřesné odometrie.

V druhém případě byl robot postaven na pozici [2.05; 0.64] před překážku (papírová krabice) a cíl byl umístěn až za ní na souřadnice [0.38; -0.45]. Robot nalezl trajektorii s nejmenší cenou a následně trajektorii realizoval.



Obrázek 8.10: Naplánovaná trajektorie okolo překážky, cílový bod (červená šipka) a aktuální pozice robota (zelený obdélník) v mapě



Obrázek 8.11: Uskutečněná trajektorie okolo překážky

Cílový bod je opět dosažen se zadanou přesností. Překážka nebyla porušena.

Pro vytvoření grafů (8.9) a (8.11) bylo potřeba transformovat data získaná systémem Vicon tak, aby odpovídala souřadnému rámci robota.

9 Závěr

Výsledkem této práce je robot, který dojede na místo určení v mapě, kterou si sám vytváří pomocí algoritmu Gmapping z lidarových a odometrických dat. Pokud se v naplánované trajektorii objeví překážka, najde si robot alternativní trasu nebo se zastaví.

Zpočátku bylo nutné vyřešit řízení stejnosměrných motorů tak, aby se otáčely požadovanou úhlovou rychlostí. K tomuto účelu byl vytvořen diskrétní PI regulátor. V dalším kroku bylo nutné se seznámit s modelem diferenciálně řízeného robota, aby bylo možné předvídat jeho pohyb. V závěrečné fázi byl nastaven systém ROS tak, aby všechny komponenty vykonávaly správně svoji funkci a robot plnil požadovaný cíl.

V další práci by bylo dobré využít systém GPS pro počáteční odhad prostředí okolo robota. V návaznosti by se dalo vytvořit software pro snadnější ovládání a zobrazování dat. Odometrická data měřená pouze pomocí enkodéru lze zpřesnit použitím inerciální měřicí jednotky a systému GPS. S využitím kamery by bylo možné rozpoznávat předměty v okolí robota.

Literatura

[1] *Wild Thumper 6WD Chassis* [online]. [navštíveno 19. 11. 2018]. Dostupné z: https://www.sparkfun.com/products/11056.

[2] *Dagu Wild Thumper 6WD All-Terrain Chassis, Silver, 75:1* [online]. [navštíveno 19. 11. 2018]. Dostupné z: https://www.pololu.com/product/1561.

[3] *Algoritmus A-Star* [online]. [navštíveno 4. 1. 2019]. Dostupné z: http://voho.eu/wiki/algoritmus-a-star/.

[4] *Quadrature encoders* [online]. [navštíveno 31. 1. 2019]. Dostupné z: http://www.communica.co.za/Content/Catalog/Documents/D2749631912.pdf.

[5] *Inkrementální enkodér* [online]. [navštíveno 31. 1. 2019]. Dostupné z: http://robotika.vosrk.cz/guide/sensors/decode/cs.

[6] *costmap 2d* [online]. [navštíveno 19.11. 2018]. Dostupné z: http://wiki.ros.org/costmap_2d?distro=kinetic.

[7] *Nvidia Jetson Systems* [online]. [navštíveno 24. 1. 2019]. Dostupné z: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/.

[8] *Orbitty Carrier* [online]. [navštíveno 16.3. 2019]. Dostupné z: http://connecttech.com/product/orbitty-carrier-for-nvidia-jetson-tx2-tx1/.

[9] *Documentation* [online]. [navštíveno 24. 1. 2019]. Dostupné z: http://wiki.ros.org/Documentation.

[10] *Arduino IDE Setup* [online]. [navštíveno 1. 2. 2019]. Dostupné z: http://wiki.ros.org/rosserial_arduino/Tutorials/ArduinoIDESetup.

[11] *Setup and Configuration of the Navigation Stack on a Robot* [online]. [navštíveno 19.11. 2018]. Dostupné z: http://wiki.ros.org/navigation/Tutorials/RobotSetup.

[12] *move base* [online]. [navštíveno 19.11. 2018]. Dostupné z: http://wiki.ros.org/move_base?distro=kinetic.

[13] *T'Rex Robot/Motor Controller* [online]. [navštíveno 24. 1. 2019]. Dostupné z: https://www.sparkfun.com/products/retired/12075.

[14] *Anti-Windup Control Using a PID Controller* [online]. [navštíveno 31. 1. 2019]. Dostupné z: https://www.mathworks.com/help/simulink/examples/anti-windup-control-using-a-pid-controller.html.

[15] Grisetti. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. Dostupné z: http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti07tro.pdf.

[16] Kilián, K. *Čím se LIDAR liší od radaru a jaká je jeho role v autonomních vozidlech* [online]. [navštíveno 19. 11. 2018]. Dostupné z: https://vtm.zive.cz/clanky/cim-se-lidar-lisi-od-radaru-a-jaka-je-jeho-role-v-autonomnich-vozidlech/sc-870-a-195431/default.aspx.

[17] Mahtani, F. M. J. S. *ROS Programming: Building Powerful Robots*. Packt Publishing Ltd., 2018. ISBN 978-1-78862-743-6.

[18] Murray, A. *Feedback Systems*. Princeton university press. Version v3.0h, 8 Nov 2016.

[19] Sachs, J. *How to Build a Fixed-Point PI Controller That Just Works: Part I* [online]. [navštíveno 19. 11. 2018]. Dostupné z: https://www.embeddedrelated.com/showarticle/121.php.

[20] Schlegel, M. *Elektromagnetické systémy - Stejnosměrný motor* [online]. prezentace z předmětu KKY/SM [15.11.2012]. Dostupné z: https://portal.zcu.cz/portal/studium/courseware/kky/sm/prednasky.html.

[21] Wikipedie. *Stejnosměrný motor — Wikipedie: Otevřená encyklopedie* [online]. 2018. [navštíveno 21. 12. 2018]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Stejnosm%C4%9Brn%C3%BD_motor&oldid=16614219.

[22] Winkler, Z. *Odometrie-modely kolových vozidel* [online]. [navštíveno 19. 11. 2018]. Dostupné z: https://robotika.cz/guide/odometry/cs.

[23] Zheng. ROS Navigation Tuning Guide. 2016. Dostupné z: http://kaiyuzheng.me/documents/papers/ros_navguide.pdf.