Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

PLZEŇ, 2017 JAN HEVERA

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 15. května 2017

.....

PODĚKOVÁNÍ

Mě poděkování patří vedoucímu bakalářské práce, panu Ing. Liborovi Jelínkovi, Ph.D., za výborné vedení práce, ochotu vždy pomoct a za mnohé cenné rady, které vedly k vypracování této práce. Také bych chtěl poděkovat panu Ing. Miroslavu Flídrovi,PhD., za cenné připomínky k mé práci a mé rodině za podporu během studia.

Anotace

Tato práce se zabývá sledováním trajektorie diferenciálně řízeným kolovým robotem na základě odhadu polohy pomocí inerciálních senzorů. V práci je popsáno použití odometrie, uveden přehled inerciálních senzorů, na něž se navazuje popisem použití enkodéru pro získání rychlosti robotu. Dále je popsán návrh řízení pohybu robotu s následnými experimenty. Na konec práce je uvedeno možné řešení pro zlepšení sledování trajektorie.

Klíčová slova: Arduino, odometrie, inerciální senzory, regulátor, sledování trajektorie

Annotation

This thesis deals with the tracking of the trajectory with a differential wheel driven robot based on position estimation using inertial sensors. The work describes the use of odometry, a list of inertial sensors, followed by a description of the use of a robot speed encoder. In addition, a proposal for robot motion control with subsequent experiments is described. At the end of the work is described a possible solution to improve trajectory tracking.

Key words: Arduino, odometry, inertial sensors, controller, trajectory tracking

Obsah

Seznam obrázků

Kapitola 1

Úvod

Robot je stroj, který plní zadaný úkol s určitou mírou samostatnosti. V dnešní době se vyvíjejí nové robotické systémy, které jsou schopny nahradit člověka při plnění určitého úkolu. V praxi se setkáváme se stacionárními roboty, jejichž pracovní prostor je omezen pouze dosahem ramen, či s mobilními roboty, jejichž možnost pohybu je dána okolím a požadavkem uživatele. Použití mobilních robotů v praxi se stále zvyšuje. Jejich hlavní výhodou je, že mohou do prostorů, kam člověk nemůže, ať z důvodu nepřístupnosti (prostory pod vodou, vesmírná tělesa) či bezpečnosti (prostory s vysokou úrovní radiace, chemicky znečištěné prostory, prostory ohrožené výbušninou).

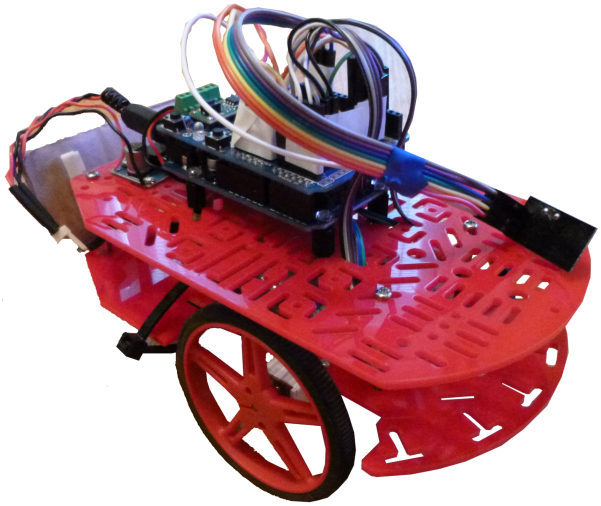
Jelikož se mobilní roboty pohybují v prostoru, je důležitá jejich schopnost určit svoji polohu. Má-li robot informaci o své poloze, je na jejím základě schopen určit, zdali sleduje zadanou trajektorii, aby se dostal k požadovanému cíli, pokud-li se vychyluje, musí upravit pomocí otáček kol své natočení.

1.1 Výchozí stav

První část práce se zabývá geometrickým modelem dvoukolového, diferenciálně řízeného robotu, který byl použit pro úlohu sledování trajektorie. Diferenciálně řízený znamená, že každé kolo má svůj motor, který je řízen nezávisle. Celý robot je ovládán řídící jednotkou Arduino. Arduino je open-source platforma založená na mikrokontrolerech ATMega od firmy Atmel. Deska je uživatelsky programovatelná a aby vykonávala, co je potřeba, je nutno vytvořit program v programovacím jazyku Arduina (založená na jazyku *Wiring*), který je poté na platformu nahrán. V našem případě je nutné upozornit, že jakékoliv periodické úkony, jako je výpočet rychlosti z naměřených impulsů či generování akčního zásahu, je prováděnou se zvolenou periodou T_s 50 ms. Zvolená hodnota je kompromisem mezi zatížením výpočetní jednotky mikrokontroléru a přesností měření a řízení robotu.

Na desku Arduino jsou připojeny výstupy enkodérů (viz Kapitola 2). Z této desky jsou také vyvedeny výstupy pulsně šířkové modulace (PWM - *Pulse Width Modulation*) na vstupy motorů. Pulsně šířková modulace se používá pro přenos analo-

gového signálu pomocí binárního (dvouhodnotového) signálu. Hodnota analogového signálu je v přenosu zakódována jako poměr hodnoty 0/1. Tento poměr nazýváme střída. Pomocí PWM signálu přiváděného na vstup motoru je ovládána jeho rychlost otáčení. Maximální hodnota PWM signálu pro tyto motory je 255 [8]. Na oba motory jsou připevněny magnetické enkodéry (viz Kapitola 3) s rozlišením 300 pulsů na otáčku.



Obrázek 1.1: Použitý robot

1.2 Cíl práce

Cílem této práce je návrh a realizace řídícího algoritmu, pomocí něhož bude robot určovat svoji aktuální polohu v prostoru a sledovat požadovanou trajektorii. Aby byl robot schopen sledovat trasu, je nutné řídit jeho pohyb, což lze zajistit návrhem regulátorů pro řízení rychlosti každého kola. Dále bude nutné provést experimenty, které ověří funkčnost algoritmu a navrženého regulátoru. Na konec práce by bylo vhodné navrhnout možnosti vylepšení řídícího algoritmu.

Kapitola 2

Lokalizace

Tato kapitola se zabývá lokalizací. Ta je stavebním kamenem pro autonomní roboty, jejichž rozsáhlé použití je v průmyslu či například v badatelských odvětvích, jako je průzkum pod mořskou hladinou a průzkum povrchu na cizích planetách. Rozlišují se tři typy autonomie robotů a to ne-autonomní roboty, tj. roboty řízené člověkem, semi-autonomní roboty, které se pohybují samy, či jsou řízené lidmi a nakonec plně-autonomní roboty, které nepotřebují pomoc člověka ke splnění svého úkolu. Minimálně dva poslední typy robotů potřebují mít ke své funkci k dispozici informaci o své aktuální pozici v prostoru. Tato kapitola čerpá z literatury pod číslem [1], [9].

Způsoby, které mohou být použity pro určení pozice robotu v prostoru můžeme rozdělit do dvou tříd:

Prostředky relativní lokalizace

- **Inerciální senzory** (viz Kapitola 3) - Tyto prostředky používají akcelerometry a gyroskopy pro měření zrychlení a míru natočení. Tato měření jsou poté integrována a je získána pozice. Jednou z výhod těchto senzorů je soběstačnost. Na druhou stranu integrace s sebou nese jistá rizika, jelikož data jsou unášena s časem. Tyto senzory se stávají nepřesnými při delším použití.
- **Odometrie** - Tato metoda používá enkodéry k určení míry rotace kol a na jejím základě stanovuje polohu robotu v prostoru (metoda bude vysvětlena v následující podkapitole).

Prostředky absolutní lokalizace

- **Aktivní orientační body** - K navigaci jsou potřeba aktivní vysílače umístěné v prostředí, kde se robot pohybuje. Tyto vysílače, jejichž minimální počet musí být tři, vysílají světelný či rádiový signál, který robot přijímá pomocí přijímače a na základě těchto signálů určí pozici v prostoru.
- **Umělé pasivní orientační prvky** - V této metodě se umisťují umělé navigační body do prostoru pohybu robotu (např. čára na zemi). Výhodou této metody je možnost vhodného umístění pro kvalitní detekci i v nepříznivém prostředí.

- **Přirozené pasivní orientační prvky** - Tato navigace funguje podobně jako předchozí, ale k navigaci používá rozpoznávání určitých navigačních bodů daného prostředí.
- **Porovnávání modelů** - U posledního způsobu se vytváří mapa prostředí na základě měření ze senzorů. Tato mapa je poté porovnána se skutečnou mapou daného prostředí, jestliže mají stejné charakteristické vlastnosti, je možné určit pozici robotu.

Každá z těchto uvedených metod má své klady a zápory. Čím více je použito relevantních informací, tím více je odhad pozice přesnější. Proto je vhodné pro zpřesnění sloučit informace z různých zdrojů (i více než jednu ze skupiny relativních prostředků). My jsme v našem případě použili pouze jeden prostředek a to odometrii.

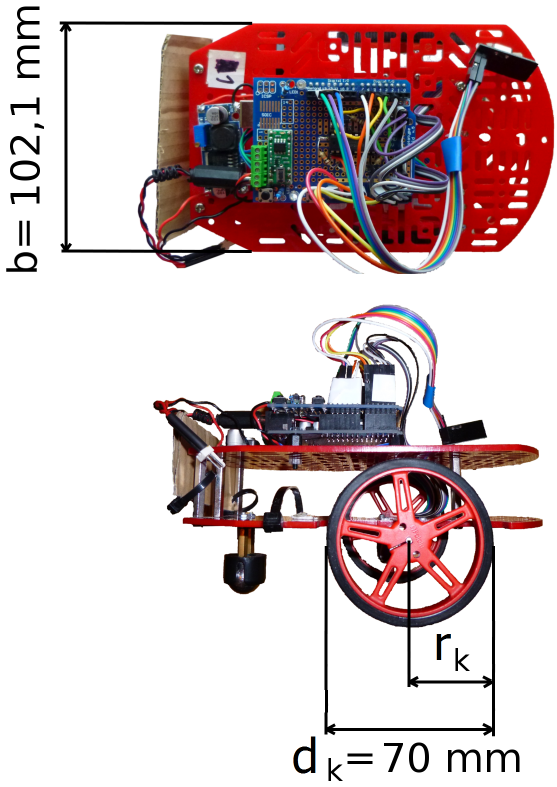
2.1 Odometrie

Jak již bylo řečeno, odometrie patří do prostředků relativní lokalizace. Tyto prostředky vyhodnocují pozici na základě rychlosti, směru jízdy a času, uplynulém od poslední známé pozice. V této metodě jsou používány enkodéry (viz Kapitola 3), které vyvolávají impulsy při otáčení kola. Tyto impulsy jsou poté v robotu zaznamenávány (viz Kapitola 4) a na jejich základě a na základě geometrie daného kola je vypočtena rychlost či ujetá dráha daného kola. Z ujeté dráhy obou kol je pak možné určit ujetou dráhu celého robotu (těžiště robotu) a jeho natočení vůči referenčnímu bodu. Získaná data je možné použít pro převod do vhodného souřadnicového systému.

Jelikož jsou výpočty dány na základě naměřeného počtu otáček kola a jejich skutečný počet není možné samostatně ověřit, je třeba počítat s chybami měření, jako je šum v měření, prokluz kol, nesymetrie v geometrii kol atd. Kvůli těmto negativním vlivům se pak robot může nacházet na jiné pozici, než na pozici, kterou si vypočetl na základě měření z enkodéru.

2.1.1 Geometrický model robotu

Pro výpočet rychlosti, dráhy a v konečném důsledku pozice robotu je velmi důležité popsat jeho geometrický model. Na následujícím obrázku jsou vyobrazeny důležité parametry robotu, se kterými se bude pracovat:



Obrázek 2.1: Parametry robotu

kde
b - rozchod kol
d_k - průměr kol
r_k - poloměr kol

Máme-li naměřené vzdálenosti v jednotlivých časových okamžicích měření, můžeme provést následující výpočty, které jsou založené na geometrickém modelu robotu. Výpočet ujeté vzdálenosti pro těžiště robotu:

$$\Delta s = \frac{\Delta s_L + \Delta s_R}{2} \tag{2.1}$$

pro
$\Delta s_L = s_L(k) - s_L(k-1)$
$\Delta s_R = s_R(k) - s_R(k-1)$
kde
Δs: ujetá dráha středu robotu
Δs_L: ujetá dráha levého kola od posledního časového okamžiku $k-1$
$s_L(k)$: ujetá dráha levého kola v časovém okamžiku k
$s_L(k-1)$: ujetá dráha levého kola v časovém okamžiku $k-1$
Δs_R: ujetá dráha pravého kola od posledního časového okamžiku $k-1$
$s_R(k)$: ujetá dráha pravého kola v časovém okamžiku k
$s_R(k-1)$: ujetá dráha pravého kola v časovém okamžiku $k-1$

Úhel natočení robotu:

$$\Delta\theta = \frac{\Delta s_L - \Delta s_R}{b} \tag{2.2}$$

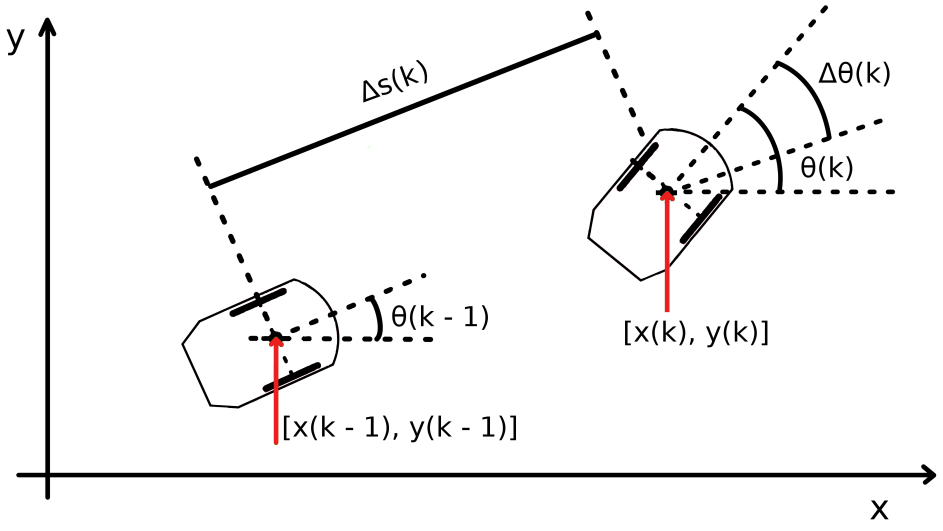
kde
$\Delta\theta$: úhel odbočení robotu od natočení v předchozím kroku
b: rozchod kol robotu

Pohybuje-li se robot po kružnici, tedy rychlost kol je nenulová, na každém kole rozdílná a kola se točí stejným směrem, pak vzdálenost středu robotu od středu této kružnice je dána vztahem:

$$R = \frac{b}{2} \cdot \frac{\Delta s_L + \Delta s_R}{\Delta s_L - \Delta s_R} \tag{2.3}$$

2.1.2 Využití geometrického modelu k zobrazení polohy robotu v kartézské soustavě souřadnic

Chceme-li zobrazit polohu robotu pomocí souřadnic (x,y) v kartézské soustavě, musíme využít geometrický model robotu, definovaný v předchozí podkapitole. Pohyb kol je popsán lineární rychlostí, úhlovou rychlostí a ujetou dráhou. Pohyb těžiště robotu je pak popsán stejnými veličinami. Pomocí těchto veličin se zobrazí poloha robotu v kartézské soustavě souřadnic. Následuje postup přepočtu ujeté dráhy kol na polohu:



Obrázek 2.2: Počáteční a předvídaná poloha robotu

V předchozím obrázku (2.2) je zobrazen pohyb robotu v dvou po sobě jdoucích

diskrétních časových okamžicích. Jeho stav je označen jako $X(k-1)$, přechází do polohy v čase $t(k) = k \cdot T_s$, kde T_s je vzorkovací perioda, do stavu $X(k)$. Tento stav robotu je popsán souřadnicemi v ose $x(k)$, $y(k)$ a natočením robotu $\theta(k)$. Řídící vstup $U(k)$ se skládá z posunu robotu Δs a z úhlu natočení $\Delta\theta(k)$ mezi dvěma časovými okamžiky:

$$X(k) = [x(k), y(k), \theta(k)]^T \tag{2.4}$$

$$U(k) = [\Delta s(k), \Delta\theta(k)]^T \tag{2.5}$$

Aktualizovaný stav $X(k)$ se pak vypočte z předchozího stavu $X(k-1)$, z posunu Δs, z rozdílu natočení $\Delta\theta(k)$ a úhlu natočení robotu $\theta(k-1)$ v posledním časovém okamžiku:

$$x(k) = f_x(X(k-1), U(k-1)) = x(k-1) + \Delta s(k) \cdot cos(\theta(k-1) + \frac{\Delta\theta(k)}{2}) \tag{2.6}$$

$$y(k) = f_y(X(k-1), U(k-1)) = y(k-1) + \Delta s(k) \cdot sin(\theta(k-1) + \frac{\Delta\theta(k)}{2}) \tag{2.7}$$

$$\theta(k) = f_\theta(X(k-1), U(k-1)) = \theta(k-1) + \Delta\theta(k) \tag{2.8}$$

Přepis do maticového tvaru:

$$X(k) = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} = \begin{bmatrix} cos(\overline{\theta}(k)) & 0 \\ sin(\overline{\theta}(k)) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta s(k) \\ \Delta\theta(k) \end{bmatrix} \tag{2.9}$$

kde

$$\overline{\theta}(k) = \theta(k-1) + \frac{\Delta\theta(k)}{2} \tag{2.10}$$

Uvedené rovnice jsou použity v realizaci pro výpočet polohy v kartézské soustavě souřadnic na základě vypočtené ujeté dráhy obou kol, popsané ve čtvrté kapitole.

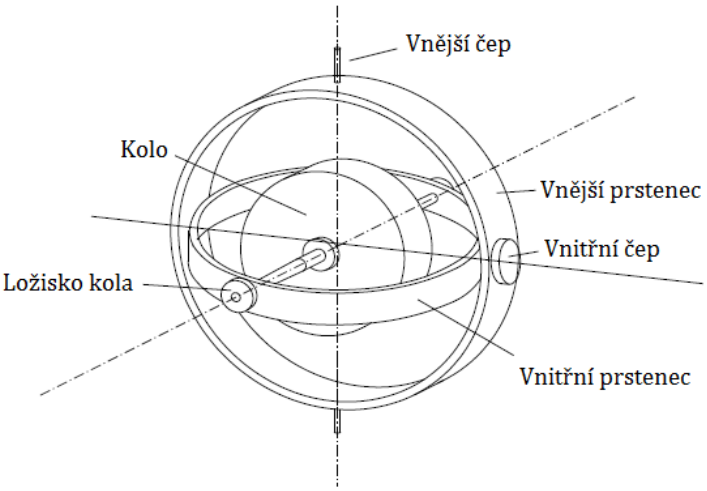
Kapitola 3

Inerciální senzory

Inerciální senzory jsou senzory založené na principu setrvačnosti. Patří do nich gyroskopy, které měří úhlovou rychlost a akcelerometry, které měří zrychlení. Výstupy inerciálních senzorů jsou použity pro stanovení pozice a natočení zařízení, jež těmito senzory disponuje. Pro tuto kapitolu bylo čerpáno z materiálů, uvedených v literatuře pod čísly [2], [1], [10], [11], [5], [12].

3.1 Gyroskopy

Gyroskopy jsou senzory sloužící k měření úhlové rychlosti, k inerciální navigaci a k určení směru. Jak již z názvu vypovídá, pracují na principu gyroskopického jevu, k němuž dochází, když je hmotnost soustředěna po obvodu setrvačníku. Jedná se tedy o setrvačník zavěšený v Cardanově rámu (z důvodu eliminace translačního pohybu), jenž umožňuje rotaci v jakémkoli směru kolem svého těžiště. Gyroskop si tedy snaží zachovat osu rotace díky svému momentu setrvačnosti. Tyto senzory se používají v mnoha odvětvích, ať už je to letectví (gyrokompasy, umělé horizonty, astronomie (stabilizace družic), v elektronice (otáčení obrazu v chytrém telefonu) či ve vojenství (zaměřování, navigace střel). Mechanický princip gyroskopu je vyobrazen na následujícím obrázku:



Obrázek 3.1: Dvouosý mechanický gyroskop (převzato z [1])

Následuje několik běžně používaných typů gyroskopů:

- Mechanické gyroskopy
- Piezoelektrické gyroskopy
- Optické gyroskopy - optické vláknové gyroskopy a laserové prstencové gyroskopy

V našem případě nebyl tento senzor pro určení polohy v prostoru použit. Jeho použití by však přineslo další metodu měření natočení robotu, které by společně s odometrií, umožnilo stanovit více přesnou hodnotou tohoto natočení. Následuje popis mechanického a piezoelektrického gyroskopu.

3.1.1 Mechanické gyroskopy

Tento gyroskop je popsán na obrázku (3.1). Jedná se o setrvačník zavěšený k dvěma prstencům. Díky této konstrukci se může setrvačník pohybovat ve všech směrech. Princip tohoto gyroskopu je založen na zákonu zachování momentu hybnosti, jenž působí na roztočené kolo, které odolává změnám natočení zařízení. Tomu však neodolávají prstence, které po natočení zařízení změní svůj vzájemný úhel. Úhel je odečten úhlovými snímači. Mechanický gyroskop tedy měří orientaci, na rozdíl od ostatních gyroskopů, které měří úhlovou rychlost.

Tyto senzory jsou obvykle velmi přesné. Jejich nevýhodou je větší hmotnost a velmi obtížně se zmenšují. Jejich mechanické části se opotřebovávají, jsou energeticky náročné a rozměrné.

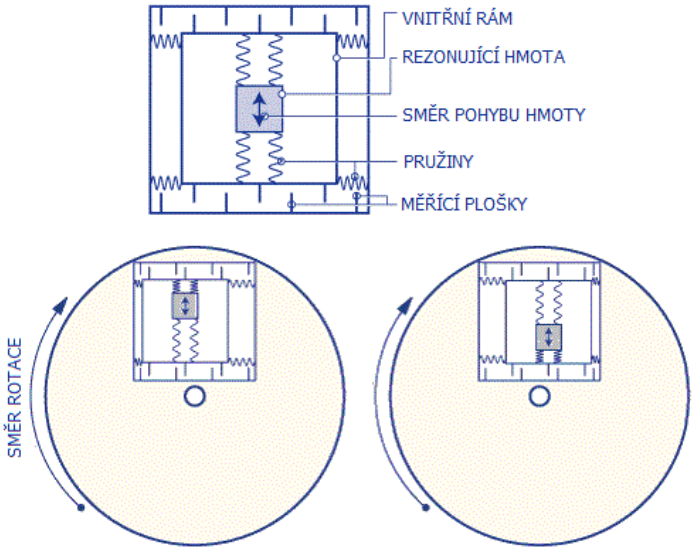
3.1.2 Piezoelektrické gyroskopy

Piezoelektrické rotační gyroskopy využívají Coriolisovy síly pro měření míry natočení. Coriolisova síla působí na těleso pohybující se rychlostí v v soustavě rotující kolem osy rotace úhlovou rychlostí ω:

$$F_c = 2 \cdot m \cdot \vec{v} \times \vec{\omega} \tag{3.1}$$

kde
m: hmotnost
$\vec{v}$: rychlost
$\vec{\omega}$: úhlová rychlost

Zjednodušeně řečeno, na těleso umístěné v rotující soustavě působí síla úměrná úhlové rychlosti. V piezoelektrickém gyroskopu (viz obrázek 3.2) je umístěná periodicky rezonující hmota, která se může pohybovat pouze v kolmém směru ke směru otáčení. Na tuto hmotu působí Coriolisova síla úměrná velikosti úhlové rychlosti otáčení. Síla stlačuje pružiny mezi vnitřním a vnějším rámem, mezi kterými jsou umístěny měřící plošky vzduchových kondenzátorů. Výstupem je pak kapacita úměrná úhlové rychlosti.



Obrázek 3.2: Princip piezoelektrického gyroskopu (převzato z [2])

Tyto gyroskopy jsou oproti ostatním méně přesné, avšak vynikají menší cenou a je snadné je miniaturizovat, proto se tyto gyroskopy nazývají také MEMS (*Micro-ElectroMechanical Systems*) gyroskopy. MEMS systémy obecně, je spojení senzorů, integrovaných obvodů, mechanických částí, akčních členů na jeden křemíkový substrát, mají tedy miniaturní rozměry.

3.2 Akcelerometry

Akcelerometry patří po boku gyroskopů k hojně používaným inerciálním snímačům. Tyto senzory měří akceleraci, jinak řečeno zrychlení, tedy míru nárůstu či poklesu rychlosti. Jsou také vhodné k měření natočení a pozice tělesa či k měření vibrací. Akcelerometry mohou měřit zrychlení jak dynamické, což je působení zrychlení na akcelerometr, tak statické, neboli gravitační působení na akcelerometr. Dále lze akcelerometry rozdělit podle počtu měřených os na jednoosé, dvouosé a tříosé.

S akcelerometry se opět setkáváme v každodenním životě. V průmyslu se používají pro měření zrychlení automobilů a v bezpečnostních prvcích automobilů. Dále se používají k měření vibrací automobilů, budov, strojů a k měření seismické aktivity. V domácnosti se s nimi můžeme setkat v chytrých telefonech, v noteboocích a ve fotoaparátech. Odvětví, ve kterých se akcelerometry používají, je opravdu mnoho, mimo jiné můžeme ještě uvést zbrojní průmysl, letecký průmysl, zdravotnictví, navigace a v neposlední řadě elektronika.

Následuje několik běžně používaných typů akcelerometrů:

- Piezoelektrické akcelerometry
- Piezorezistivní akcelerometry
- Kapacitní akcelerometry
- Akcelerometry s Hallovým efektem
- Magnetoresistivní akcelerometry
- Tepelné akcelerometry MEMSIC

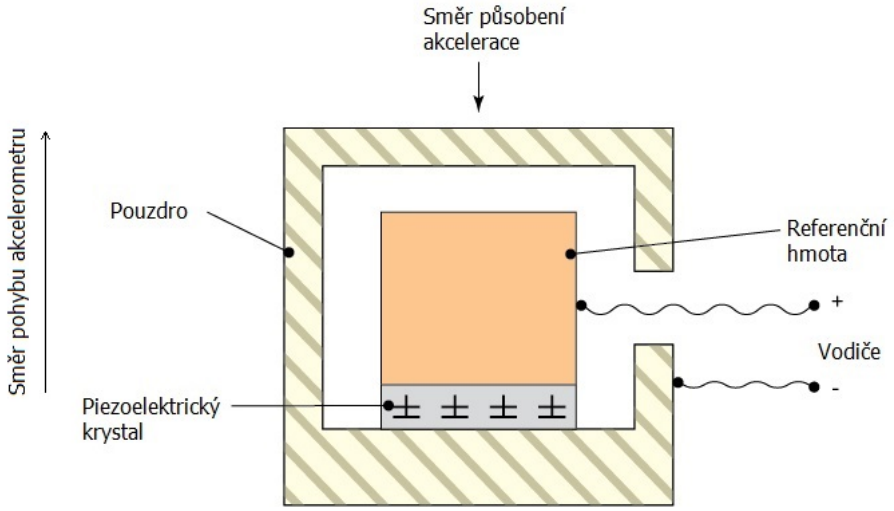
V naší práci budou popsány pouze piezoelektrické akcelerometry.

3.2.1 Piezoelektrické akcelerometry

Piezoelektrické akcelerometry využívají ke své funkci piezoelektrický materiál, jehož charakteristickou vlastností je schopnost generovat náboj při jeho deformování. V akcelerometru (viz obrázek 3.3) je umístěn piezoelektrický krystal, připevněný k pouzdru akcelerometru. Na krystal je připevněná referenční hmota. Ta při pohybu akcelerometru působí svojí hmotností na krystal, který svou deformací generuje výstupní náboj. Ten je úměrný síle působící na referenční hmotu podle druhého Newtonova zákona:

$$F = m \cdot a \tag{3.2}$$

kde
m: hmotnost referenční hmoty
a: zrychlení referenční hmoty



Obrázek 3.3: Princip piezoelektrického akcelerometru (převzato z [3])

Výhodou těchto senzorů je dlouhá životnost, snadná instalace, velký měřící rozsah a přesnost. Nevýhodou jsou pak relativně velké rozměry.

3.3 Enkodéry

Enkodéry jsou senzory, které generují digitální signál v reakci na mechanický pohyb. Existují dva typy enkodérů a to lineární a rotační. Lineární enkodér slouží k měření pohybu po určité přímé trase. Příkladem použití lineárního enkodéru by mohlo být umístění v kolejnici, po které se pohybuje robotické rameno. Výstupem by pak byla pozice ramene v určitém místě kolejnice. Rotační enkodér slouží k měření rotačního pohybu, z toho vyplývá nutnost umístění na rotující části. Rotační enkodéry lze rozdělit podle typu výstupu na inkrementální a absolutní. Inkrementální enkodéry generují řádu impulsů, které je možné použít k určení rychlosti a polohy. Absolutní enkodéry generují jedinečné konfigurace bitů, které přímo vyjadřují polohu otočení.

Enkodéry mají v praxi široké uplatnění. Například v průmyslu se nacházejí v obráběcích a CNC strojích. V domácnosti se používají ve skenerech a tiskárnách. V medicíně se používají pro řízení pohybu mikroskopů a v diagnostických přístrojích. Armáda používá enkodéry při natáčení antén, stejně tak astronomové při natáčení teleskopů.

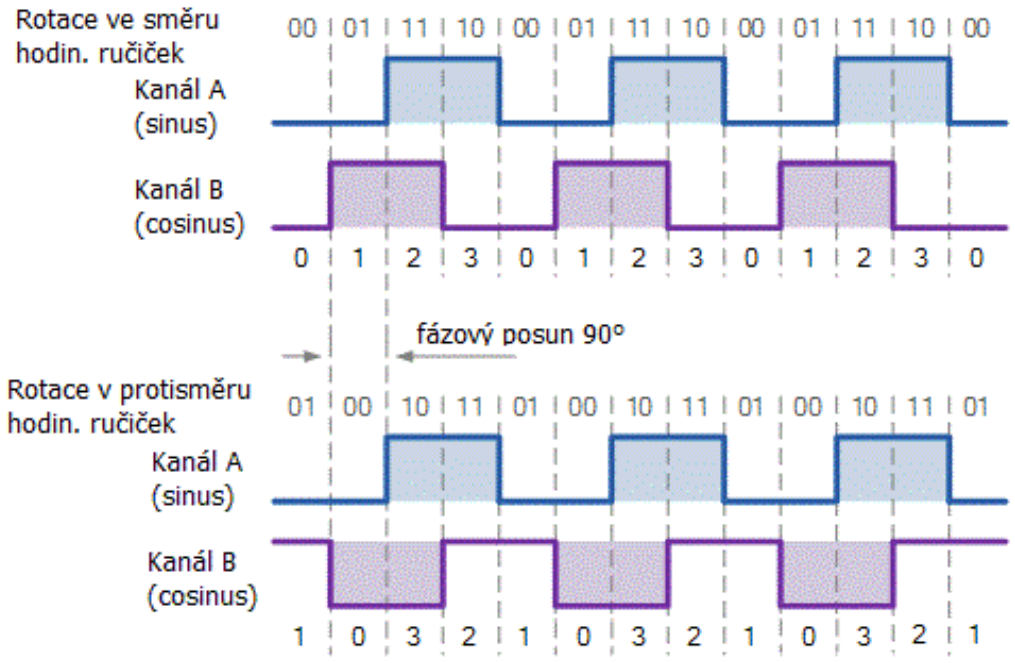
Rozdělení enkodérů:

- Rotační enkodéry
  - Inkrementální enkodéry
    - Optické
    - Magnetické
    - Mechanické

  - Absolutní enkodéry
- Lineární enkodéry

3.3.1 Inkrementální enkodéry

Inkrementální enkodéry, jejichž výstupem jsou dva fázově posunuté signály impulsů, se také nazývají kvadraturní, jelikož enkodér generuje čtyři různé stavy 0, 1, 2, 3, jak je vyobrazeno na následujícím obrázku:



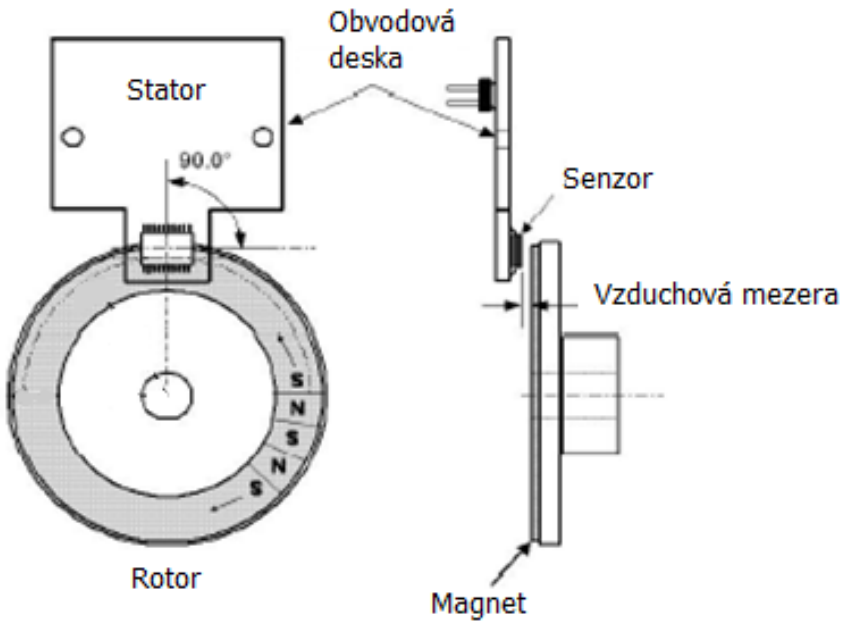
Obrázek 3.4: Výstup kvadraturního enkodéru (převzato z [4])

Tímto enkodérem je nejen možné určit rychlost, ale i směr otáčení, což lze vidět na změně posloupnosti stavů. Způsob jakým je možné generovat impulsy může být optický, magnetický, čí mechanický. Jelikož námi používaný robot má implementované magnetické enkodéry, bude popsána pouze funkce magnetického enkodéru.

Magnetické rotační enkodéry

Magnetické rotační enkodéry se skládají ze dvou částí a to z rotoru a statoru. Rotor se otáčí s hřídelí a obsahuje střídavě rozložené jižní (S - *South*) a severní (N - *North*) póly permanentního magnetu. Na statoru je pak umístěn prvek určen k detekování změn magnetických pólů při otáčení rotoru. Hlavní dva způsoby jak detekovat tuto změnu jsou Hallův jev a magnetorezistence. Hallův jev je základem tzv. Hallovy sondy, na které, po vložení do magnetického pole, vzniká elektrické napětí úměrné magnetické indukci daného magnetického pole. Magnetorezistenci využívají magnetorezistentní snímače, které jsou vyrobeny z materiálu, jehož odpor závisí na

magnetickém poli, ve kterém se snímač nachází. Princip magnetického enkodéru je vykreslen na následujícím obrázku:



Obrázek 3.5: Princip magnetického enkodéru (převzato z [5])

Senzor umístěný na statoru, například Hallova sonda, generuje napětí v závislosti na magnetickém pólu, který se nachází v těsné blízkosti sondy. Otáčí-li se rotor, dochází ke střídání magnetických pólů, tím pádem se generují rozdílné úrovně napětí a tedy i logické úrovně, jak je vidět na obrázku (3.4).

Pro naše účely byl použit tento typ enkodéru na obou kolech robotu. Enkodéry generují logické úrovně při otáčení kol. Tyto signály jsou přivedeny na vstup mikrokontroléru Arduino. Každá vzestupná hrana (změna z logické úrovně 0 na logickou úroveň 1) vyvolá v mikrokontroléru přerušení, které inkrementuje hodnotu proměnné počtu zaznamenaných impulsů na daném kole. Pro měření úhlové rychlosti stačí použít pouze jeden impulsní signál.

Kapitola 4

Měření rychlosti a ujeté dráhy kol

Tato kapitola se zabývá využitím měření pulsů (viz předchozí kapitola) z magnetických enkodérů umístěných na obou kolech robotu. Nejdříve bude popsáno využití těchto měření pro výpočet rychlosti pro vyšší otáčky a poté pro nižší otáčky kola. Poté bude následovat popis filtrace měřených impulsů a na konec této kapitoly bude uveden způsob pro výpočet ujeté dráhy kol.

4.1 Výpočet rychlosti pro střední a vyšší otáčky

V tomto způsobu je dána konstantní perioda čtení počtu nainkrementovaných impulsů. Tento počet impulsů v čase k a počet impulsů z periody čtení $k-1$ je použit pro výpočet rychlosti podle následujícího vztahu:

$$v(k) = \frac{x(k) - x(k-1)}{T} = \frac{\triangle X}{T} \tag{4.1}$$

kde
$v(k)$: rychlost v časovém okamžiku k
$x(k)$: poloha v časovém okamžiku k
$x(k-1)$: poloha v časovém okamžiku $k-1$
T: konstantní časový interval
$\triangle X$: ujetá vzdálenost za čas T

Z tohoto vztahu vyplývá, že je nutné ukládat počet naměřených impulsů z předchozí periody měření. Převedením tohoto obecného vzorce (4.1) pro náš případ, kde má robot rozlišení 300 impulsů na otáčku kola, poloměr kola je $r_k = 0{,}035$ m a perioda vzorkování je $T = T_s = 0{,}05$ s, získáme výraz (4.2):

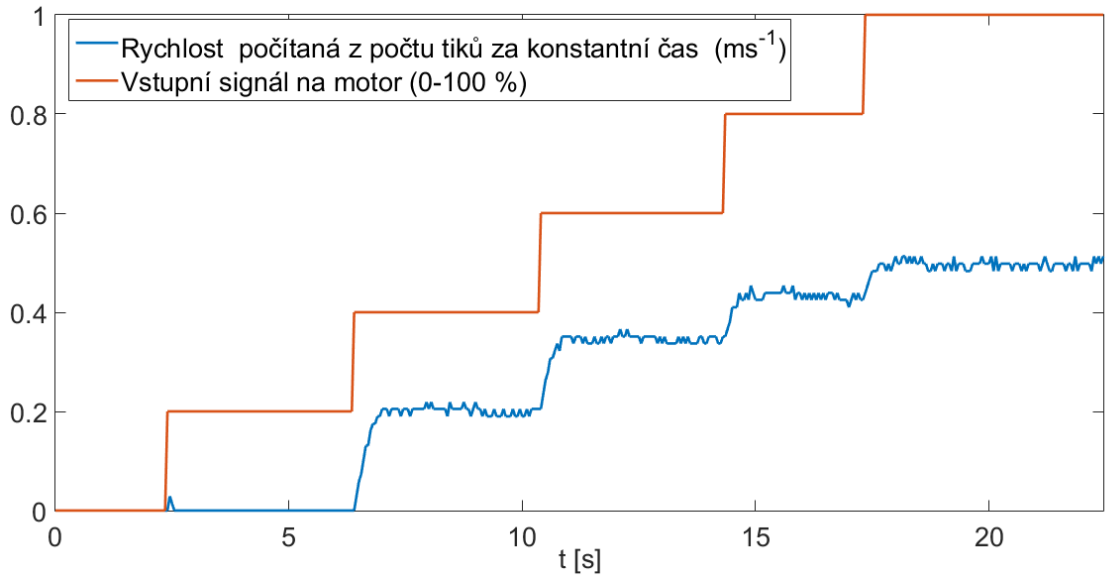
$$v(k) = \frac{(cn(k) - cn(k-1)) \cdot \frac{2 \cdot \Pi \cdot r_k}{rez})}{T} \tag{4.2}$$

kde
$v(k)$: rychlost v časovém okamžiku k

$cn(k)$: počet naměřených impulzů v čase k
$cn(k-1)$: počet naměřených impulzů v čase $k-1$
r_k: je poloměr kola robotu
rez: rozlišení enkodéru, tedy počet impulzů na jednu otáčku kola
T: je perioda vzorkování

Přesnost tohoto výpočtu závisí na počtu impulsů na otáčku a na periodě čtení počtu tiků. Pro experimentálně zjištěnou minimální rychlost robotu $v = 0{,}05$ m/s vychází počet impulsů za vzorkovací periodu 3,41. Je nutno dbát na způsob ukládání této hodnoty v programu, je-li použit typ integer, hodnota se zaokrouhluje dolů (floor), proto chyba vztažená k měřené hodnotě je 1/3,41, tedy 29 %. Pro maximální rychlost robotu $v = 0{,}48$ m/s činí počet impulsů za vzorkovací periodu 32,74, tzn. chyba vztažená k měřené hodnotě je 1/32,74, tedy 3 %. Chyba v celém rozsahu měření nabývá 3-29 %.

Zatížení procesoru u tohoto způsobu měření rychlosti je rovnoměrné a závisí na vzorkovací periodě T. Vyšší perioda znamená vyšší zatížení, ale i přesnější výpočet. Výhodou této metody je filtrování impulsů [13].



Obrázek 4.1: Ukázka vypočítané rychlosti z počtu tiků za konstantní čas pro levý motor

4.2 Výpočet rychlosti pro nižší otáčky

Riziko pro předchozí metodu jsou nižší rychlosti, při kterých se může stát, že za danou periodu (časový interval) měření se nezaznamená ani jeden impuls. Tuto nevýhodu eliminuje následující metoda. Místo měření počtu tiků za konstantní čas se bude měřit čas mezi jednotlivými impulsy:

$$v(k) = \frac{X}{t(k) - t(k-1)} = \frac{X}{\triangle T} \tag{4.3}$$

kde
$v(k)$: rychlost v časovém okamžiku k
$t(k)$: časový okamžik k
$t(k-1)$: časový okamžik $k-1$
X: ujetá vzdálenost příslušející na jeden impuls
$\triangle T$: časový interval mezi dvěma po sobě jdoucími impulsy

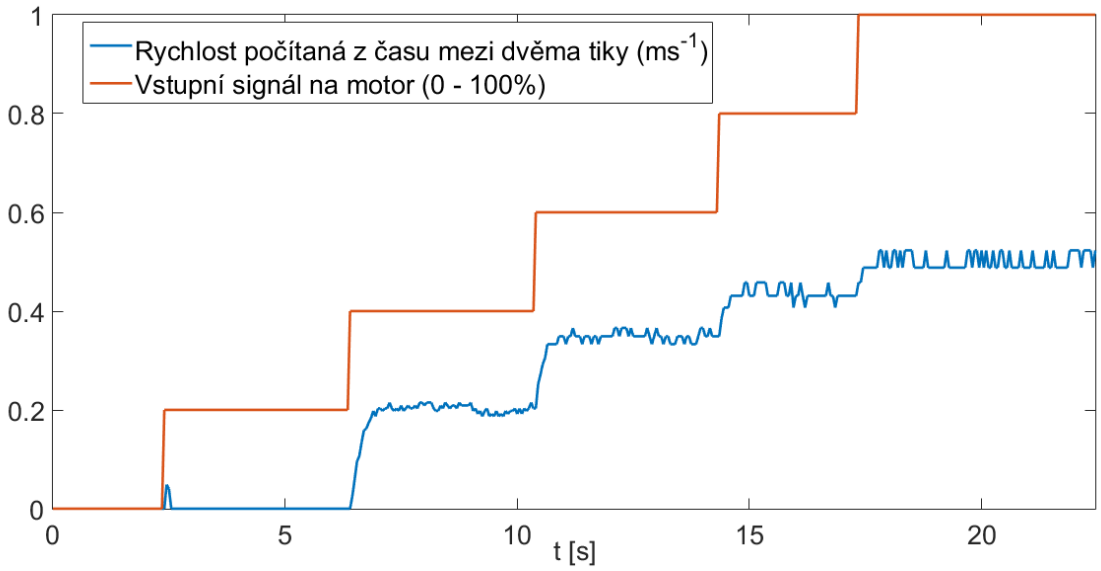
Pro tento způsob je potřeba si uchovávat čas mezi dvěma po sobě následujícími impulsy. Dále, jako v předchozím způsobu výpočtu, je třeba znát počet impulsů daného enkodéru na jednu otáčku a poloměr kola. Na rozdíl od způsobu výpočtu (4.2) není třeba znát periodu vzorkování. Podobným převedením získáme vzorec pro náš případ:

$$v(k) = \frac{2 \cdot \Pi \cdot r_k}{(et(k) - et(k-1)) \cdot rez} \tag{4.4}$$

kde
$v(k)$: rychlost v časovém okamžiku k
$et(k)$: čas přerušení od enkodéru, tedy zaznamenání impulsu
$et(k-1)$: čas zaznamenání impulsu před impulsem $et(k)$
r_k: je poloměr kola robotu
rez: rozlišení enkodéru, tedy počet impulzů na jednu otáčku kola

Přesnost v tomto případě závisí na zvolené časové jednotce. Pro nás byla zvolena jednotka 100 μs. Pro experimentálně zjištěnou minimální rychlost robotu v = 0,05 m/s vychází čas mezi dvěma tiky 14,6 ms. Časový údaj je opět ukládán do proměnné datového typu integer. Chyba vztažená k měřené hodnotě činí 7 %. Tato hodnota je přijatelná, jelikož nepřekračuje hodnotu periody výpočtu 50 ms. Pro nejvyšší rychlost v = 0,48 m/s činí čas mezi dvěma tiky 1,53 ms, chyba vztažená k měřené hodnotě je 65 %. Chyba v celém rozsahu měření nabývá 7-65 %.

Výhodou výpočtu (4.3) je malá náročnost na vytížení procesoru, avšak pouze pro malé otáčky.



Obrázek 4.2: Ukázka vypočítané rychlosti z času mezi dvěma tiky pro levý motor

4.2.1 Časovač

Pro metodu výpočtu podle času mezi dvěma tiky je důležitý časovač. Časovač umožňuje v pravidelných intervalech vyvolávat přerušení bez ohledu na to, co jiného se děje v daném kódu. Každý časovač má čítač, který je inkrementován každý tik hodinami časovače. Přerušení od časovače nastane, jakmile čítač dovrší nějaké specifické hodnoty, uložené v porovnávacím registru. Jestliže čítač dovrší této hodnoty, další tik se vynuluje jeho hodnota a opět se čítá, dokud zase nedovrší hodnoty v porovnávacím registru. Frekvenci přerušení volíme výběrem hodnoty porovnání s čítačem a nastavením rychlosti inkrementace čítače [14].

Hodiny Arduina mají frekvenci 16 MHz. Tato hodnota také představuje nejvyšší rychlost čítání, kterou může čítač dosáhnout. Hodnota 16 MHz reprezentuje tik každé 1/16 000 000 sekundy, tedy přibližně 63 ns. Znamená to tedy, že např. 8-bitový časovač dosáhne maximální hodnoty každých 256/16 000 000 sekund ($\sim 16\mu s$), po tomto dojde k tzv. přetečení, čítač se vynuluje a cyklus se opakuje. Toto ovšem není příliš praktické, chceme-li například vyvolat přerušení každou vteřinu. Proto se používá tzv. prescaler, který sníží rychlost časovače. Hodnota prescaleru může být 1, 8, 64, 256, 1024 [14].

Tzn. pro prescaler 1 je rychlost časovače 16 MHz, pro prescaler 2 je rychlost 8 MHz, atd. Výsledný vztah pro výpočet frekvence přerušení je:

$$\text{frekvence přerušení (Hz)} = \frac{\text{frekvence hodin Arduina (16 MHz)}}{\text{prescaler * (hodnota porovnávací registru + 1)}} \quad (4.5)$$

Deska Arduino UNO obsahuje 3 časovače timer0, timer1 a timer2. Časovače timer0 a timer2 jsou 8-bitové, tzn. maximální hodnota porovnávacího registru je 255, timer1 je 16-bitový a maximální hodnota porovnávacího registru je 65 535.

4.3 Filtrace signálu

Měřený signál není vždy ideální pro další zpracování, proto se v mnoha aplikacích používá filtrace signálu. Ta slouží například k výběru určitých frekvencí, ke změně frekvenčního rozsahu nebo k odstranění šumu. Poslední operace se hodí pro náš naměřený signál (diskrétní posloupnost vzorků) rychlosti. Filtry mohou být analogové a číslicové. Jelikož by bylo doplnění analogového filtru do robotu náročné, byl použit číslicový filtr. Jedna z výhod je možnost změny parametrů filtru, bez nutnosti zásahu do hardwaru, další výhoda je časová stálost a lepší tvary charakteristik. Číslicové filtry jsou rekurzivní či nerekurzivní. Následuje příklad dvou číslicových filtrů FIR a IIR [15].

4.3.1 FIR filtr

FIR (Finite Impulze Response) filtr, neboli filtr s konečnou impulsní odezvou je nerekurzivní, spoléhá se tedy na současné a předchozí vstupní vzorky signálu. Výhodnou vlastností tohoto filtru je jeho zaručená stabilita, možnost lineární fáze a jednoduchost návrhu, avšak je výpočetně náročný na čas.

FIR filtr je popsán následující diferenční rovnicí:

$$y(k) = a \cdot u(k) + (1 - a) \cdot u(k - 1) \tag{4.6}$$

kde
$y(k)$: výstupní signál
$u(k)$: vstupní signál v časovém okamžiku k
$u(k-1)$: vstupní signál v časovém okamžiku $k-1$
a: koeficient filtru $\in (0, 1)$

Z výrazu je vidět, že u filtru je nutno nastavit a. Tento parametr se volí s ohledem na to, jak moc chceme potlačit šum signálu, čím víc však jej chceme potlačit, tím více ovlivňujeme dynamiku systému.

4.3.2 IIR filtr

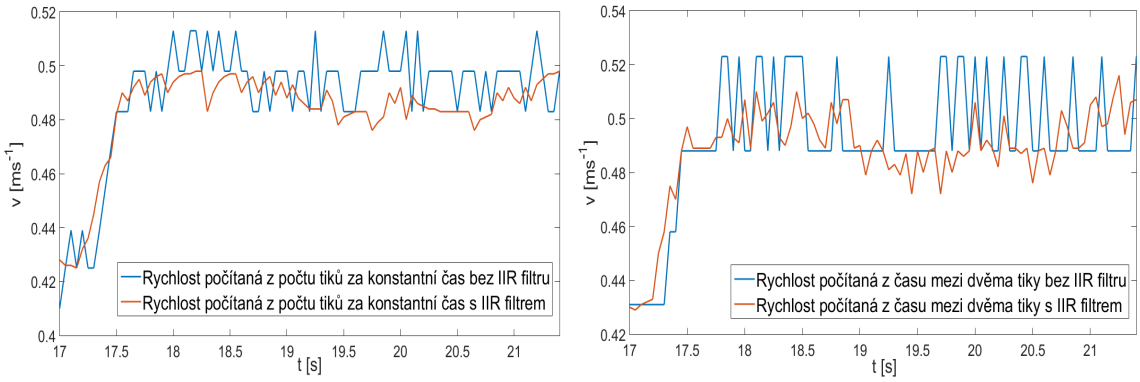
IIR (Infinite Impulze Response) filtr, neboli filtr s nekonečnou impulsní odezvou je rekurzivní, spoléhá se tedy na současné a předchozí výstupní vzorky signálu [15]. Tento filtr je analogií k analogovým filtrům, nemusí být vždy stabilní, má nelineární fázi, oproti FIR filtru se vyznačuje složitějším návrhem a v neposlední řadě je citlivý na numerickou přesnost výpočtu.

IIR filtr je dán následující diferenční rovnicí:

$$y(k) = a \cdot u(k) + (1 - a) \cdot y(k - 1) \tag{4.7}$$

kde
$y(k)$: výstupní signál
$y(k-1)$: výstupní signál v časovém okamžiku $k-1$
$u(k)$: vstupní signál v časovém okamžiku k
a: koeficient filtru $\in (0,1)$

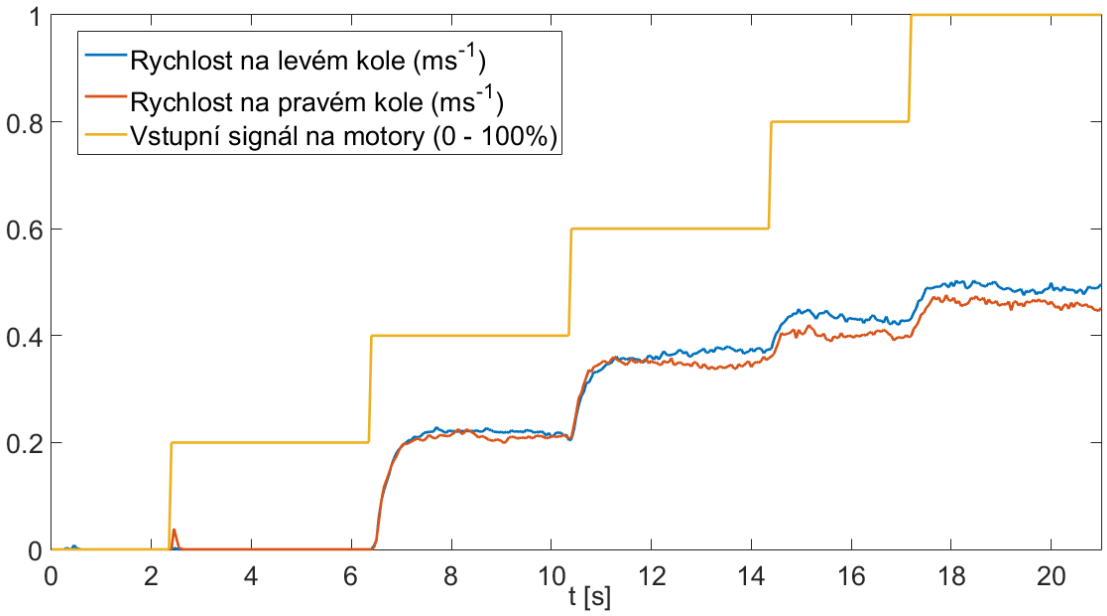
Jelikož je odhad rychlosti nepřesný, byl použit IIR filtr pro snížení rozdílu ve skocích signálu. Filtrace se prováděla pro oba způsoby výpočtu (4.2 a 4.4). Při aplikaci se tedy proměnná u(k) nahradí buďto naměřeným počtem impulsů mezi dvěma periodami výpočtu, anebo časem mezi dvěma tiky. Dosazením y(k), tedy filtrovaného naměřeného počtu impulsů, nebo naměřeného času, se získá filtrovaná rychlost:



(a) Filtrace rychlosti (4.2) (b) Filtrace rychlosti (4.4)

Obrázek 4.3: Ukázka filtrace rychlostí počítané oběma metodami

Na tyto rychlosti je poté použit aritmetický průměr, po kterém je stanovena konečná rychlost kola, která je poté použita pro regulátor:



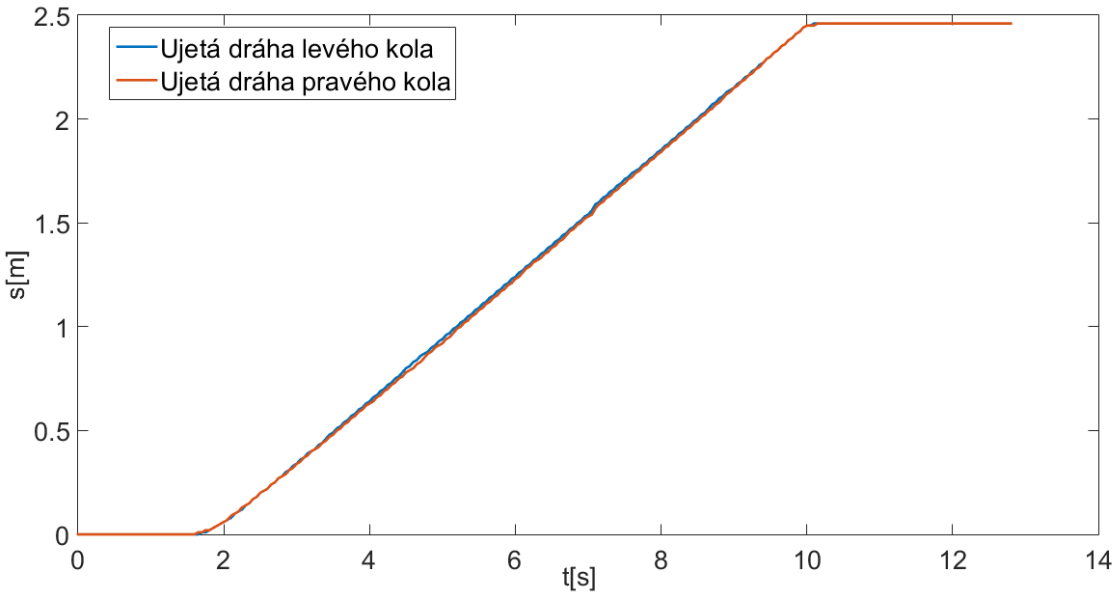
Obrázek 4.4: Ukázka výsledných rychlostí na obou kolech

4.4 Výpočet ujeté dráhy kol

Ujetá dráha se může stanovit z počtu impulsů za konstantní čas i z času mezi dvěma po sobě následujícími impulsy. Při volbě způsobu výpočtu se musí dbát na to, že pro výpočet dráhy za konstantní čas se výpočet provádí každou periodu výpočtu, avšak pro stanovení dráhy podle času mezi jednotlivými impulsy je nutné výpočet provádět při každém přerušení od enkodéru, proto byl v našem kódu použit pouze způsob podle počtu tiků za periodu T:

$$s(k) = s(k-1) + cn(k) \cdot \frac{2 \cdot \Pi \cdot r_k}{rez} \tag{4.8}$$

kde
$s(k)$: dráha v časovém okamžiku k
$s(k-1)$: dráha v časovém okamžiku $k-1$
$cn(k)$: počet naměřených impulzů v čase k
r_k: je poloměr kola robotu
rez: rozlišení enkodéru, tedy počet impulzů na jednu otáčku kola



Obrázek 4.5: Ukázka vypočítané dráhy levého kola a pravého kola

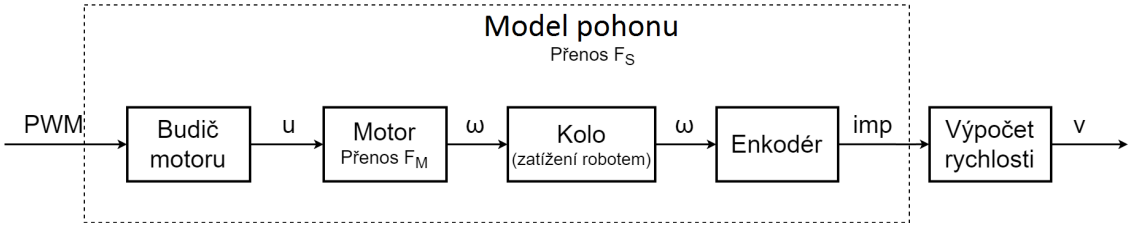
Kapitola 5

Řízení pohybu robotu

Tato kapitola se bude zabývat řízením pohybu robotu, který se sestává z motoru a navrženého regulátoru. První část tedy bude o popisu modelu motoru a poté o identifikaci konkrétních parametrů motorů umístěných na robotu. Stejnosměrný motor má určité statické zesílení, při identifikaci motoru se statické zesílení vyloučí a bude se počítat s jednotkovým statickým zesílením. Druhá část bude o zvolení vhodného regulátoru a návrhu parametrů tohoto regulátoru pro model motoru zatížený robotem, včetně budičů. Třetí část se bude zabývat potlačením nelinearity motoru, v této části se také začlení statické zesílení motoru, které bylo zanedbáno při identifikaci modelu a návrhu regulátoru.

5.1 Model zatíženého pohonu s budičem

Do modelu pohonu nepatří pouze samostatný motor, ale i budič motoru a zatížení motoru (zatížení robotem). Důvod je takový, že regulátor, navrhovaný pro řízení pohybu, nebude navrhován pouze pro výkonovou část motoru. To by bylo pouze v případě, kdyby byl motor spouštěn bez zátěže na hřídeli a buzení tohoto motoru by bylo ideální. Avšak v případě této práce jsou motory zatíženy vlastní vahou robotu. Další důvod je takový, že identifikace přenosu pohonu probíhá z naměřené přechodové charakteristiky, kde vstupem je signál PWM do motoru a výstupem je rychlost, která se počítá na základě výstupních impulsů z enkodéru. Proto bude nutné také začlenit enkodér do modelu pohonu, jelikož i vlastnosti tohoto prvku mohou ovlivňovat výsledný přenos. Jak dále uvidíme, vstupem výkonové části modelu motoru je napětí a výstupem je úhlová rychlost. Nicméně vstupem modelu celého pohonu, zobrazeného na dalším obrázku, je PWM signál a výstupem je rychlost.



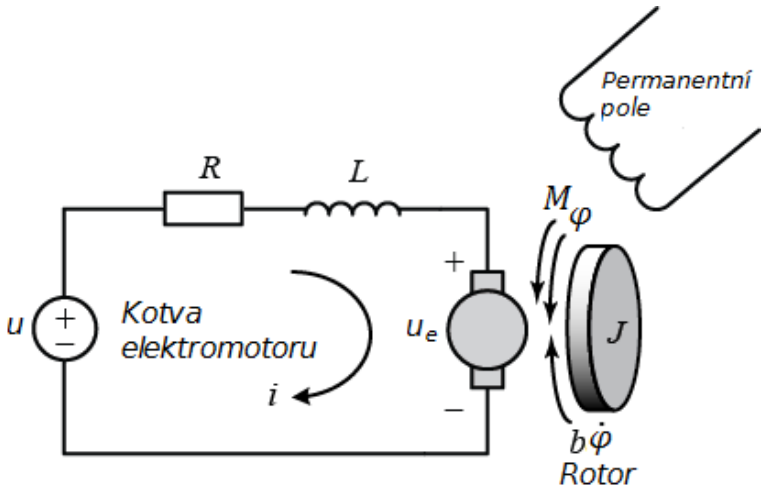
Obrázek 5.1: Schéma pohonu robotu

kde:
PWM - vstupní PWM signál na budič
u - vstupní napětí do motoru
ω - úhlová rychlost motoru
imp - impulsy generované enkodérem
v - výstupní rychlost

Motory, které pohánějí kola našeho robotu, jsou stejnosměrné. Pro popis těchto motorů je třeba určit přenosové funkce. Přenosová funkce je parametrický model vnějšího popisu systému (výstup ku vstupu). Přenosovou funkci je možné určit ze stavového modelu nebo z diferenciálních rovnic [16].

Přenosová funkce stejnosměrného motoru robotu

Stejnosměrné motory se v současné době používají v elektrických regulovaných obvodech (např. automobilový průmysl, elektrická trakce, obráběcí stroje) či v servomechanismech. Přímo zajišťují rotační pohyb a v kombinaci s koly poskytují translační pohyb. Výhodou těchto motorů je snadné řízení změnou budícího napětí na rotoru a lineární charakteristika závislosti otáček na budícím napětí [17]. Matematický model motoru [6] bude odvozen z následujícího schématu stejnosměrného motoru:



Obrázek 5.2: Zjednodušené schéma stejnosměrného elektrického motoru [6]

Označení:

u - napětí přiváděné do kotvy elektromotoru
R - odpor vinutí kotvy
L - indukčnost vinutí kotvy
u_e - napětí vzniklé v důsledku rotace kotvy
i - proud procházející kotvou
M - kroutící moment motoru
φ - úhel natočení hřídele motoru
$b\dot{\varphi}$ - třecí moment, b - konstanta viskózního tření
J - moment setrvačnosti rotoru

Za vstup je považováno napětí přiváděné do kotvy elektromotoru, za výstup pak úhlová rychlost otáčení hřídele $\omega = \frac{d\varphi}{dt}$. Jak bylo řečeno dříve, kroutící moment motoru, neboli točivý moment je úměrný proudu procházejícím kotvou motoru. Předpoklad je konstantní magnetické pole, tudíž kroutící moment je úměrný pouze proudu kotvy (konstantní k_t):

$$M = k_t i \tag{5.1}$$

Napětí vzniklé v důsledku rotace kotvy je úměrné úhlové rychlosti hřídele s konstantní elektromotorickou silou k_e:

$$u_e = k_e \frac{d\varphi}{dt} \tag{5.2}$$

Diferenciální rovnice pro elektrickou část se nechá odvodit z obrázku (5.2) pomocí Kirchhoffova zákona:

$$L\frac{di}{dt} + Ri = u - k_e \frac{d\varphi}{dt} \tag{5.3}$$

Z rovnováhy momentů ($\sum_i M_i = 0$) je pak odvozena diferenciální rovnice pro mechanickou část:

$$J\frac{d^2\varphi}{dt^2} + b\frac{d\varphi}{dt} = k_t i \tag{5.4}$$

Konstanty kroutícího momentu k_t a napětí k_e, vzniklého v důsledku rotace kotvy, budou nahrazeny jednotnou konstantou K, jelikož jsou stejné. Po použití Laplaceovy transformace lze předchozí diferenciální rovnice (5.3 a 5.4) vyjádřit pomocí Laplaceovy proměnné:

$$\begin{aligned} (Ls + R)I(s) &= U(s) - Ks\varphi(s) \\ s(Js + b)\varphi(s) &= KI(s) \end{aligned} \tag{5.5}$$

Vyjádřením I(s) z jedné rovnice v (5.5) a dosazením do druhé, eliminujeme I(s) a dostaneme přenos motoru, kde napětí je považováno za vstup a úhlová rychlost hřídele je považována za výstup:

$$F_M(s) = \frac{\dot{\varphi}(s)}{U(s)} = \frac{K}{(Js+b)(Ls+R)+K^2} = [\frac{rad/s}{V}] \tag{5.6}$$

U tohoto modelu je nutno dbát na to, že do něj není zahrnuta nelinearita skutečného motoru. To tedy znamená, že například reakce na vstupní signál 6 V je stejná jako reakce na vstup 1 V. Ve skutečnosti se však motor při vstupním napětí 1 V nepohne, což je způsobeno vlastním třením v motoru. Je tedy nutno překonat určitý vstupní signál, při kterém se motor uvede do pohybu [6]. Výsledná přenosová funkce je tedy popis pouze nezatíženého motoru bez budičů.

5.1.1 Identifikace parametrů motorů měřením

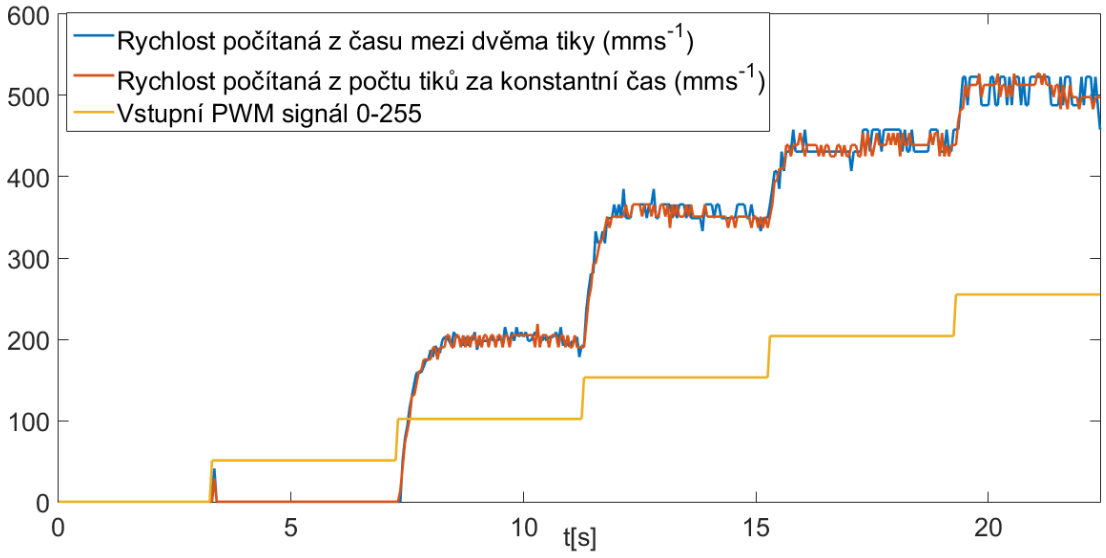
Tato metoda vychází z naměřených fyzikálních veličin motoru. Z přenosu (5.6) je vidět, že veličiny, které je možno dosadit, jsou odpor a indukčnost vinutí kotvy. Tyto parametry se dají zjistit z dokumentace k danému motoru, jelikož ale nebyla k dispozici, tyto veličiny bylo nutné naměřit RLC multimetrem.

Frekvence zkušebního signálu	Odpor	Indukčnost
0 Hz	3.1Ω	-
100 Hz	3.1Ω	0.58 mH
1 kHz	3.35Ω	0.57 mH
10 kHz	10.1Ω	0.48 mH
100 kHz	191Ω	0.11 mH

Tabulka 5.1: Ukázka naměřených fyzikálních parametrů levého motoru

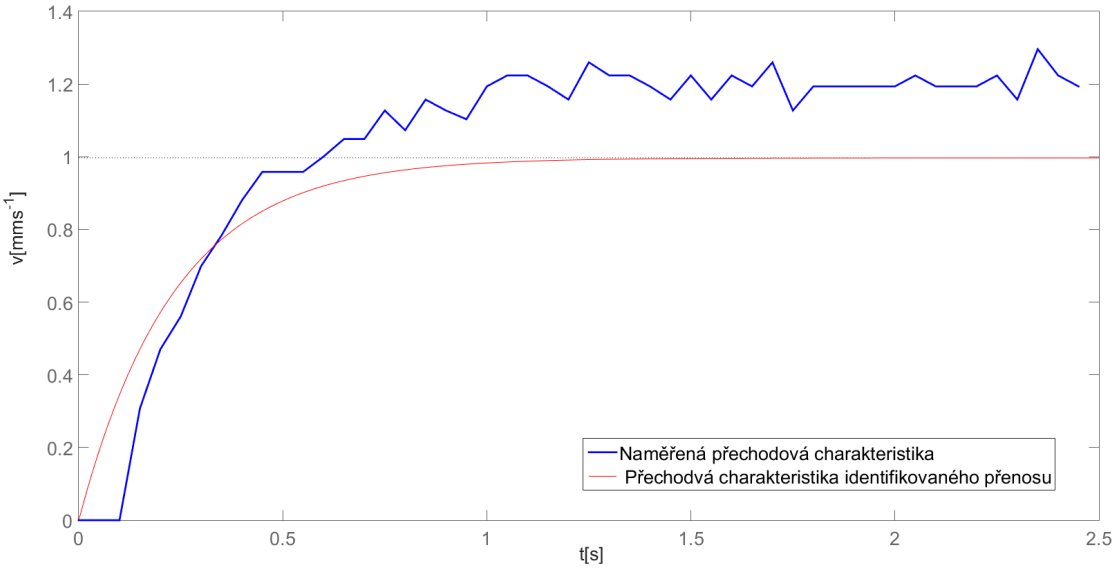
Do přenosu (5.6) pak dosadíme hodnoty naměřené na frekvenci zkušebního signálu 100 Hz. Jelikož jsme dosadili jenom hodnoty odporu a indukčnosti cívky, bylo nutné odhadnout hodnoty momentu setrvačnosti rotoru J a konstantu viskózního tření. Pro tento získaný přenos je pak vygenerována přechodová charakteristika, která je porovnána se skutečnou charakteristikou naměřenou na reálném robotu.

Na následujícím obrázku je vyobrazena naměřená charakteristika, která bude používána v následujících metodách identifikace. Byla naměřena na zatíženém robotu, robot byl tedy puštěn na podlaze. Každé 4 vteřiny se zvyšoval vstupní PWM signál o hodnotu 51 až do maximálních 255.



Obrázek 5.3: Ukázka naměřené přechodové charakteristiky

Již z této charakteristiky je patrné, že výstupní rychlost není lineární vzhledem ke vstupnímu buzení. Důvodem je měření přechodové charakteristiky nejenom modelu motoru, ale i modelu budičů a modelu působící zátěže na motor. Jedna část skoku této charakteristiky je porovnána v následujícím obrázku s přechodovou charakteristikou nezatíženého motoru, popsaného přenosem (5.6):



Obrázek 5.4: Ukázka jednoho úseku přechodové charakteristiky

Z porovnání vidíme, že přechodová charakteristika modelu nezatíženého motoru (přenos 5.6) se podobá přechodové charakteristice modelu zatíženého motoru s budiči. Přenosové funkce obou modelů budou tedy také podobné. Jelikož uvedený přenos (5.6) není standardní formou zápisu přechodové funkce, zvolíme si obecný zápis přechodové funkce systému druhého řádu, vyjádřený zesílením K a časovými

konstantami T_1 a T_2, který je obdobný. Z toho tedy vyplývá, že se nebude pracovat s modelem samotného nezatíženého motoru, ale s modelem zatíženého motoru robotem včetně budičů. Přenos tedy bude vyjadřovat závislost mezi výstupní rychlostí a vstupním signálem PWM.

Obecný zápis přenosové funkce

Je dán systém s obrazem vstupu $U(p)$ a výstupu $Y(p)$. Tyto obrazy byly získány Laplaceovou transformací na diferenciální rovnice daného systému. Pro tyto obrazy platí:

$$F(p) = \frac{Y(p)}{U(p)} = \frac{b_m p^m + b_{m-1} p^{m-1} + b_1 p + b_0}{p^n + a_{n-1} p^{n-1} + a_1 p + a_0} = \frac{b(p)}{a(p)}, \quad m \leq n \tag{5.7}$$

Je to tedy obraz výstupu ku vstupu. Kde kořeny polynomu v čitateli b(p) se označují jako nuly přenosu n_j, j = 1,...m a kořeny polynomu a(p) ve jmenovateli jsou značeny jako póly přenosu p_i, i = 1,...n. [16]

Přenosová funkce ve tvaru součinu kořenových činitelů:

$$F(p) = \frac{Y(p)}{U(p)} = \frac{b_m(p - n_1)(p - n_2)...(p - n_m)}{(p - p_1)(p - p_2)...(p - p_n)} = \frac{b(p)}{a(p)}, \quad m \leq n \tag{5.8}$$

Jestliže jsou reálné části nul či pólů záporné, jedná se o stabilní póly, či nuly. Záporné převrácené hodnoty těchto nul a pólů jsou označovány jako časové konstanty $T_i = -\frac{1}{p_i}$, i = 1,...n; $\tau_j = -\frac{1}{n_j}$, j = 1,...m.

Poslední forma zápisu je pro nás stěžejní, bude používána ve zbytku práce. Jedná se o vyjádření pomocí časových konstant:

$$F(p) = \frac{Y(p)}{U(p)} = \frac{b_0(\tau_1 p + 1)(\tau_2 p + 1)...(\tau_m p + 1)}{a_0(T_1 p + 1)(T_2 p + 1)...(T_n p + 1)} = \frac{b(p)}{a(p)}, \quad m \leq n \tag{5.9}$$

kde b_0/a_0 je statické zesílení systému.

Jelikož přenos (5.6) není standardní forma přenosu, přejdeme podle předchozí definice na podobnou, ale standardní formu přenosu druhého řádu, kde místo parametrů J, L, R, b, vyjádříme přenos pomocí časových konstant T_1 a T_2:

$$F_S(s) = \frac{K}{(1 + T_1 \cdot s)(1 + T_2 \cdot s)} \tag{5.10}$$

kde
K: je statické zesílení systému
T_1 a T_2: jsou časové konstanty systému
s: je Laplaceova proměnná

5.1.2 Identifikace přenosové funkce modelu pohonu - pomocí integrované funkce programu Matlab

Touto metodou budeme identifikovat přenos zatíženého motoru robotem včetně budičů na základě naměřené přechodové charakteristiky zatíženého robotu, tj. bude se určovat model pohonu zatíženého robotu včetně budičů.

Pro identifikaci modelu použijeme integrovanou funkci programu Matlab zvanou *procest*. Tato funkce stanoví přenos systému na základě naměřených časových či frekvenčních dat. V našem případě je to naměřená přechodová charakteristika (obrázek 5.3). Tuto charakteristiku rozdělíme na 4 části, jedna z části bude např. skok PWM signálu ze 102 na 153 a k tomu příslušející rychlost, takto rozdělená data pak budou vkládána jako jeden ze vstupních parametrů této funkce.

Druhý parametr této funkce je nastavení počátečních podmínek, které je v Matlabu značeno jako *init_sys*. Jedna z podmínek je například, jaký řád systému požadujeme, aby funkce *procest* navrhla. V našem případě byl tedy zvolen druhý řád (viz přenos 5.10).

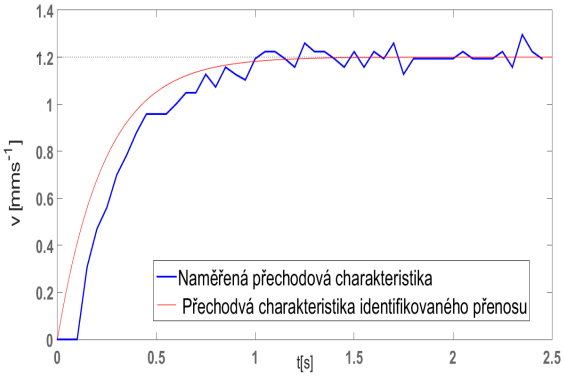
Dále co lze v *init_sys* nastavit, je dolní a horní mez, jakých může daný parametr nabývat, například hodnota statického zesílení systému se může pohybovat od 0 do 20. Máme-li pak systém s dopravním zpožděním, je opět nutné provést úpravu *init_sys*. Funkce *procest* pak navrhuje hodnoty těchto požadovaných proměnných, jež jsme si definovali.

Třetí a poslední nastavitelný parametr funkce *procest* je parametr *opt*. U tohoto parametru je možno nastavit, jakou metodou se bude provádět hledání hodnot přenosu. V našem případě byla zvolena Levenberg-Marquardtova metoda.

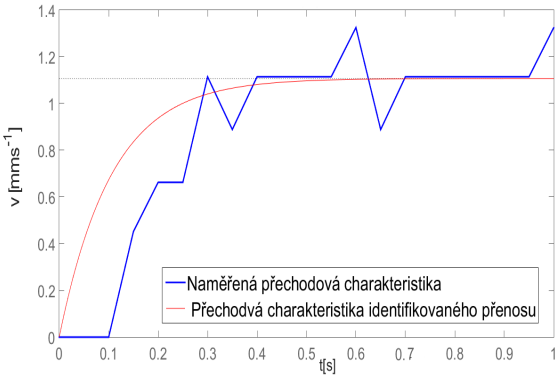
Postupným zkoušením této funkce pro nastavený požadovaný přenos druhého řádu (5.10) bylo zjištěno, že výsledná hodnota $Tp2$, kterou stanovila tato funkce, je velice malá až zanedbatelná. Proto se dále v naší práci bude s motorem pracovat jako se systémem prvního řádu, nikoliv druhého řádu:

$$F_S(s) = \frac{K}{1 + T_1 \cdot s} \tag{5.11}$$

Jelikož se do funkce *procest* vkládaly separované 4 části z přechodové charakteristiky (5.3), byly určeny 4 přenosy pro každý motor. Z těchto 4 přenosů pro každý motor byl vyčíslen aritmetickým průměrem jeden přenos, který byl považován za stěžejní. Průměrná shoda navrženého přenosu s reálným systémem byla 75 %. Pro první dvě části charakteristiky, činila shoda nad 85 %, pro poslední dvě části byla shoda kolem 65 %.



(a) Druhá část (skok PWM z 51 na 102)



(b) Poslední část (skok PWM z 204 na 255)

Obrázek 5.5: Ukázka druhé a poslední části přechodové char. (5.3), a k nim vygenerovaných přech. char. identifikovaných přenosů funkcí *procest*

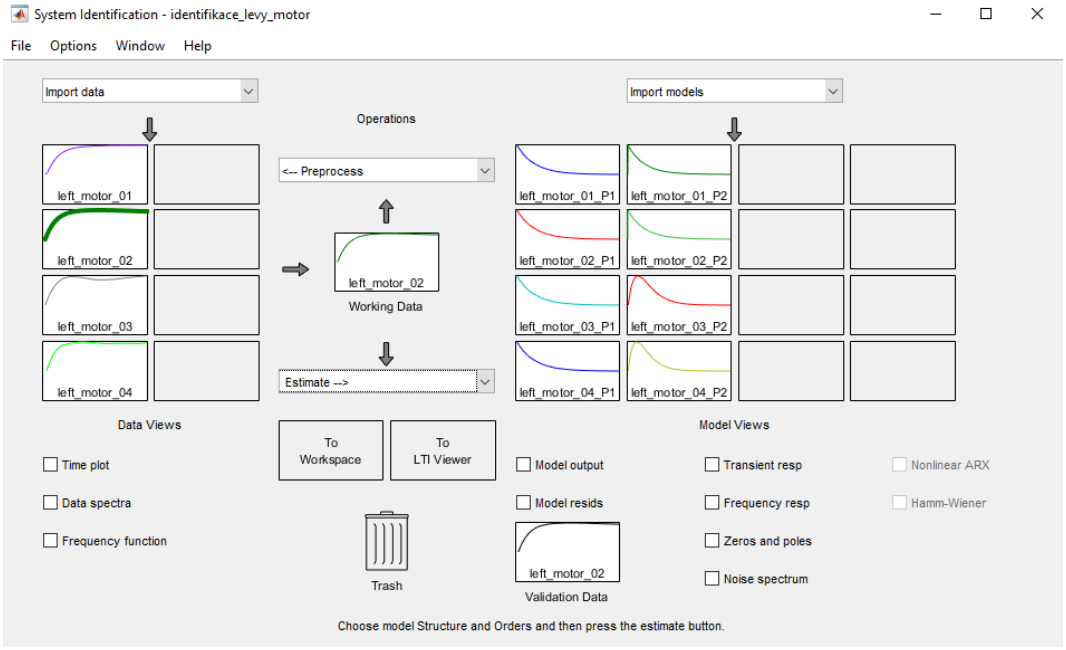
Z obrázku (5.5b) je možné pozorovat, že vygenerovaný signál z identifikovaného přenosu je zpožděn oproti skutečnému signálu. Zpoždění je v tomto případě nejpravděpodobněji zapříčiněné diskrétním měřením signálu. Systém daný přenosem prvního řádu (5.11) musí být doplněn o dopravní zpoždění, které bylo odhadnuto na jeden až dva cykly periody měření T_s:

$$F_S(s) = \frac{K}{1 + T_1 \cdot s} \cdot e^{-s\tau_d} \tag{5.12}$$

kde K: je statické zesílení systému
T_1 : je časová konstanta systému
τ_d: je dopravní zpoždění systému
s: je Laplaceova proměnná

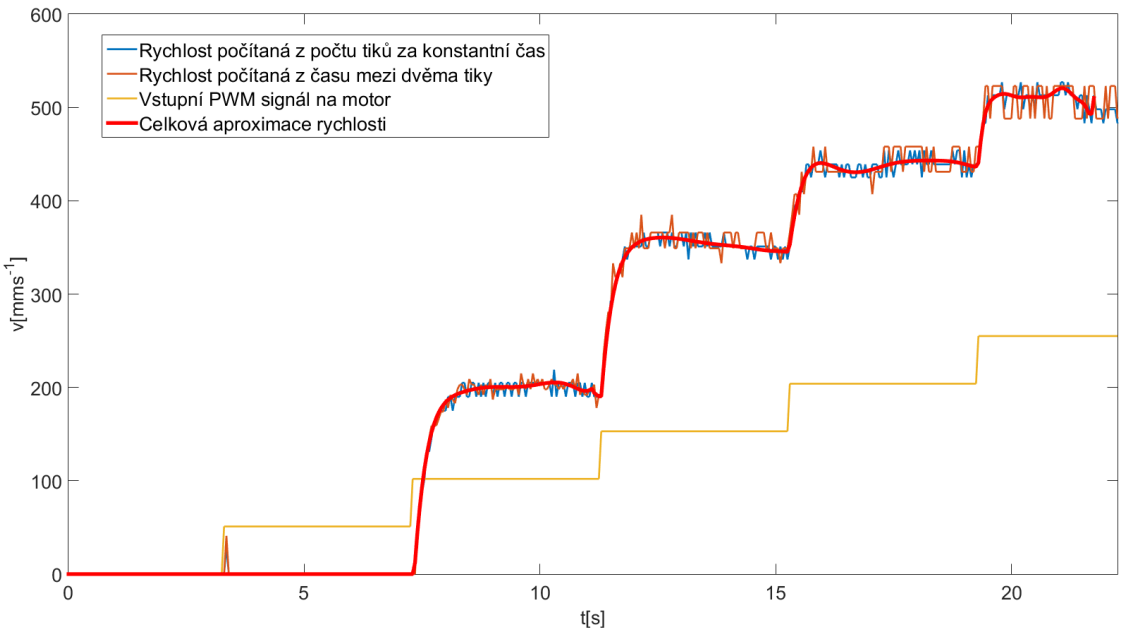
5.1.3 Identifikace přenosové funkce modelu pohonu - pomocí grafického toolboxu programu Matlab

Výstupem této metody bude opět model motoru zatíženého robotem včetně budičů. Grafická metoda identifikace parametrů přenosu je oproti předchozímu způsobu přehlednější. Pro vyvolání grafického uživatelského rozhraní, které je součástí System Identification Toolbox programu Matlab, je nutné zadat příkaz *Ident* v příkazové řádce Matlabu. Identifikace systému probíhá z naměřených vstupně výstupních dat. Možnosti nastavení identifikace tohoto toolboxu jsou rozsáhlé, proto se budeme zabývat pouze funkcemi, které byly využity v naší práci.



Obrázek 5.6: Grafické návrhové prostředí

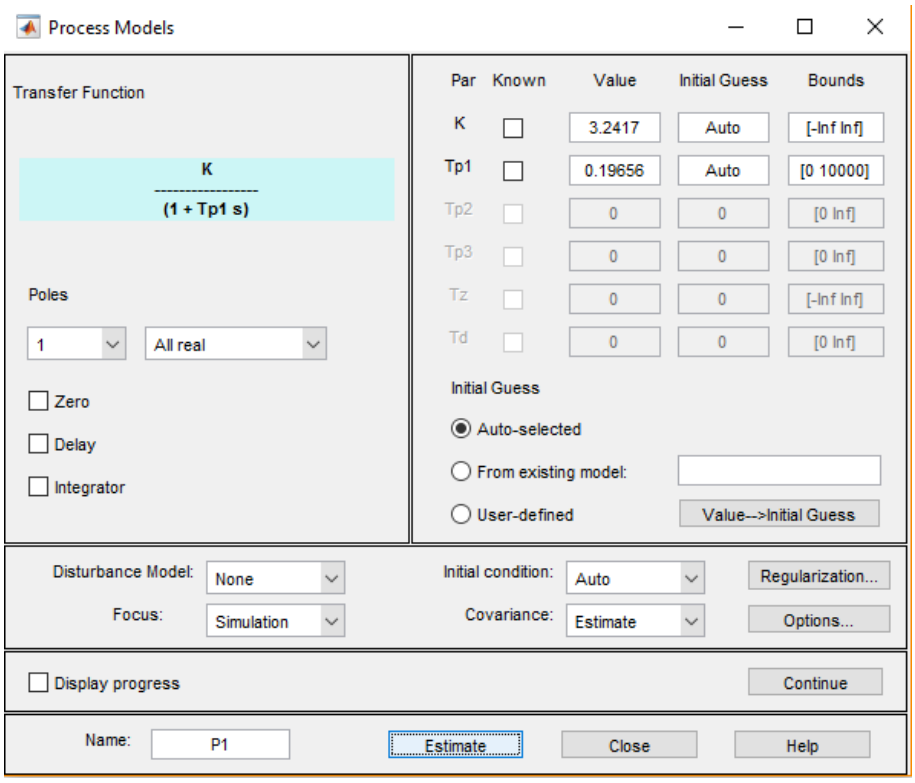
Jako vstupní data pro grafické prostředí posloužily naměřené přechodové charakteristiky pro každý motor(ukázka jedné z charakteristik motoru na obr. 5.3). Před použitím pro identifikaci byla na tyto charakteristiky použita aproximace polynomem pro lepší výsledky samotné identifikace.



Obrázek 5.7: Aproximovaná rychlost použitá jako vstup pro identifikaci parametrů

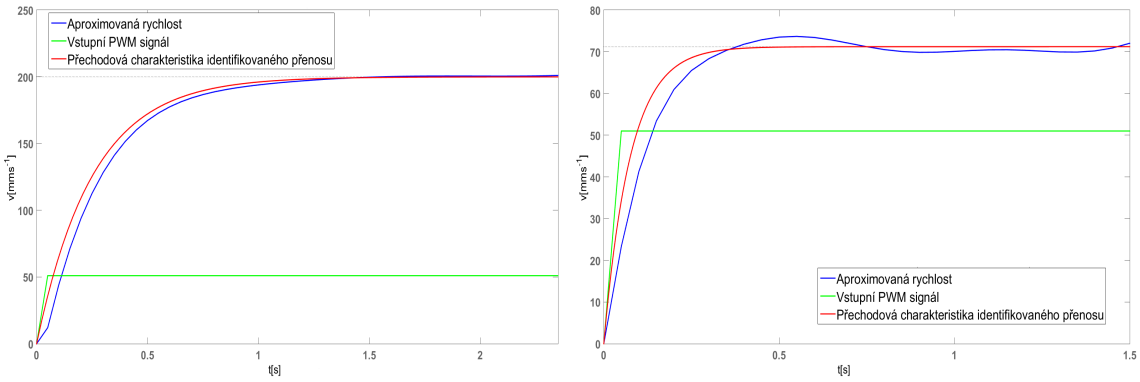
Charakteristika (5.7) byla vložena po částech do identifikačního toolboxu. Po vložení dané části do okna zpracování nazvané Working data se zvolil model Process

Models, kde byla opět možnost vybrat si řád systému, který bude toolbox identifikovat. Na rozdíl od funkce *procest* nebyla zvolena konkrétní metoda identifikace, ale automatický výběr metody identifikace.



Obrázek 5.8: Okno identu Procces Models

Výsledkem bylo opět 8 přenosů, 4 přenosy pro každý motor. Procentuální shoda činila 75 až 95 %. Opět následuje ukázka porovnání vygenerované charakteristiky identifikovaného přenosu s naměřenou charakteristikou pro druhý a poslední skok charakteristiky (5.3):

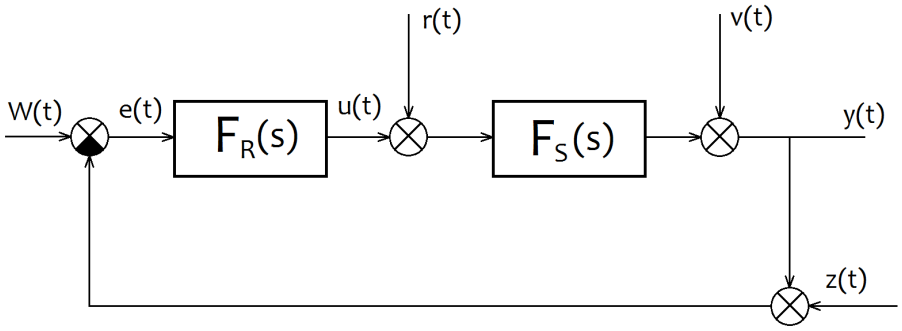


(a) Druhá část (skok PWM z 51 na 102) (b) Poslední část (skok PWM z 204 na 255)

Obrázek 5.9: Ukázka druhé a poslední části přechodové char. (5.3) a k nim vygenerovaných přech. char. identifikovaných přenosů toolboxem ident

5.2 Návrh regulátoru

Funkcí regulátoru je ovlivňovat výstup řízeného systému pomocí jeho vstupu. Pro zadaný referenční signál tedy musí regulátor vytvořit takový akční zásah do systému, po kterém se výstup řízeného systému bude minimálně odlišovat od daného referenčního signálu. Regulátor s řízeným systémem tvoří regulační obvod. Řízení systému bez zpětné vazby, tj. výstup řízeného systému se neporovnává s referenčním signálem, se nazývá přímovazební. Řízení se zpětnou vazbou, obvykle zápornou, tzn. od referenčního signálu se odečítá výstupní signál řízeného systému, se nazývá zpětnovazební [16]. Naším úkolem bude navrhnout regulátory, které pro zadanou rychlost vygenerují takový akční zásah na oba motory(PWM signál), po kterém se robot bude pohybovat zadanou rychlostí. Požadavku na regulaci na zadanou hodnotu není možné dosáhnout přímovazebním řízením, z tohoto důvodu použijeme zpětnovazební řízení popsané na obrázku níže:



Obrázek 5.10: Regulační obvod se zpětnou vazbou

kde
$F_R(s)$ - navrhovaný regulátor
$F_S(s)$ - modelu motoru zatíženého robotem včetně buzení
$w(t)$ - referenční signál
$e(t)$ - regulační odchylka $e(t) = w(t) - y(t)$
$u(t)$ - akční zásah regulátoru
$r(t)$ - aditivní porucha na vstupu systému
$v(t)$ - porucha na výstupu systému
$y(t)$ - regulovaný výstup
$z(t)$ - porucha ve zpětné vazbě (porucha senzoru)

5.2.1 Rozdělení regulátorů

Proporcionální regulátor P

Představitelem proporcionálního regulátoru může být zesilovač. Výstup tohoto regulátoru je úměrný regulační odchylce, tzn. čím větší je odchylka, tím větší zásah regulátor vyvine, avšak čím blíže se regulovaný výstup blíží referenčnímu signálu,

tím menší zásah vyvíjí, až nakonec zásah není dost silný na to, aby změnil regulovaný výstup. Pomocí proporcionálního regulátoru tedy není možné dosáhnou nulové regulační odchylky. Výstup regulátoru lze vyjádřit vztahem:

$$u(t) = K \cdot e(t) \tag{5.13}$$

kde
$u(t)$: výstup regulátoru (akční zásah)
$e(t)$: regulační odchylka
K: proporcionální zesílení

Integrační regulátor I

Výstup regulátoru I je úměrný integrálu regulační odchylky. Tento regulátor tedy vyvíjí zásah když není nulová regulační odchylka, to znamená, že na rozdíl od proporcionálního regulátoru je schopný dosáhnout nulové regulační odchylky. Snaží se vyrovnávat rozdíl mezi časem stráveným pod a nad referenčním signálem. Zmenšováním konstanty T_I se zvyšuje rychlost regulace, ale i přeregulovaní. Vztah pro integrační regulátor:

$$u(t) = K \cdot \frac{1}{T_I} \int_0^t e(\tau) d\tau \tag{5.14}$$

kde
$u(t)$: výstup regulátoru (akční zásah)
$e(t)$: regulační odchylka
T_I: integrační konstanta
K: zesílení

Takový akční zásah je vítaný u rychlých systému kde nevadí překročení referenčního signálu regulovaným výstupem. Avšak je-li třeba řídit pomalý systém, kde je překročení požadované veličiny nežádoucí efekt, např. hladina v nádrži či teplota v peci, je třeba zavést derivační regulátor.

Derivační regulátor D

Výstup regulátoru D je úměrný derivaci regulační odchylky. Samostatně není realizovatelný. Nese v sobě informaci o budoucím vývoji regulační odchylky. Čím větší je regulační odchylka, tím více působí proti tomuto zásahu:

$$u(t) = K \cdot T_D \cdot \frac{de(t)}{dt} \tag{5.15}$$

kde
$u(t)$: výstup regulátoru (akční zásah)
$e(t)$: regulační odchylka
T_D: derivační konstanta
K: zesílení

Složený regulátor PI

PI regulátor je složený z proporcionální a integrační složky. Takový regulátor je schopný dosáhnout nulové regulační odchylky. Vztah pro PI regulátor:

$$u(t) = K \cdot [e(t) + \frac{1}{T_I} \int_0^t e(\tau) d\tau] \tag{5.16}$$

Složený regulátor PD

PD regulátor je složený z proporcionální a derivační složky. Takový regulátor má rychlejší odezvu než regulátor PI. Vztah pro PD regulátor:

$$u(t) = K \cdot [e(t) + T_D \cdot \frac{de(t)}{dt}] \tag{5.17}$$

Složený regulátor PID

PID regulátor je složený z proporcionální, integrační a derivační složky. Shrnutím uvedených faktů o jednotlivých složkách se dozvídáme, že proporcionální složka se stará o zesílení regulační odchylky, integrační složka zajišťuje přesnost regulace, avšak zavádí fázové zpoždění a snižuje robustnost ve stabilitě. Derivační složka pak brzdí zásah proporcionální a integrační složky, zvyšuje robustnost, zvětšuje fázový předstih a správným nastavením konstanty T_D zvyšuje odezvu [18]. Vztah pro PID regulátor:

$$u(t) = K \cdot [e(t) + \frac{1}{T_I} \int_0^t e(\tau) d\tau + T_D \cdot \frac{de(t)}{dt}] \tag{5.18}$$

5.2.2 Volba regulátoru

Z uvedených regulátorů bylo třeba vybrat vhodný regulátor pro náš model motoru zatíženého robotem včetně budičů. Naším požadavkem je, aby regulátor dobře reguloval při skokových změnách vstupního signálu, při lineárním nárůstu vstupního signálu a pro konstantní vstup.

Integrační, proporcionální a derivační složky se nepoužívají ve většině případech samostatně. PD regulátor byl vyloučen z výběru, jelikož bez integrační složky nedosáhneme nulové regulační odchylky. Tím se zúžil výběr na PI a PID regulátor.

U PID regulátoru je třeba dávat pozor na derivační složku. Jeden z hlavních důvodů proč nepoužít regulátor s derivační složkou je, že vypočítaná rychlost je zašuměná (velké skoky). Derivace takového signálu je velice problematická a ve výsledném akčním zásahu by měla derivační složka negativní účinek. Další důvod je rychlost systému. Stejnosměrné motory jsou rychlé a mírné přeregulování není nežádoucí efekt. Nakonec nastavení tří složek PID regulátoru je daleko složitější, než nastavení pouze dvou složek PI regulátoru.

Pro naše motory byl tedy zvolen PI regulátor, jehož parametry se navrhovaly pro každý motor zvlášť. Diskretizováním vztahu pro PI regulátor (5.16) zpětnou obdélníkovou aproximací získáme vztah absolutního algoritmu řízení:

$$u(k) = K \cdot [e(k) + \frac{T_S}{T_I} \sum_{n=1}^{k} e(n)] \tag{5.19}$$

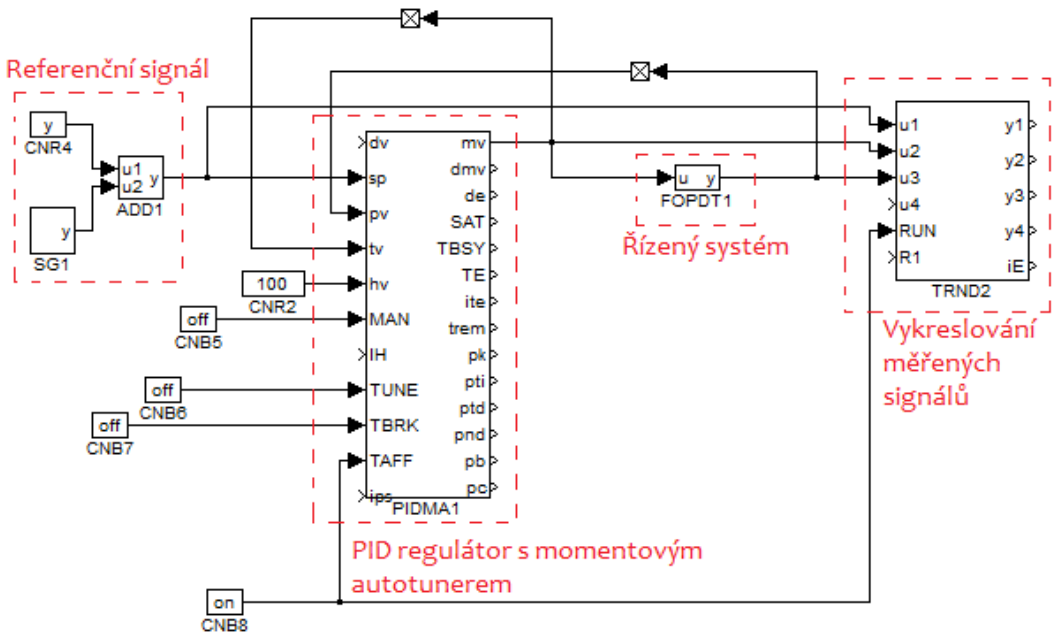
kde
$u(k)$: výstup regulátoru v časovém okamžiku k
$e(k)$: regulační odchylka v časovém okamžiku k
K: zesílení
T_I: integrační konstanta
T_S: perioda vzorkování

Ze vtahu je vidět nutnost ukládání v čase k všech předchozích regulačních odchylek. Pro získání přenosu pak na vztah PI regulátoru (5.16) aplikujeme Laplaceovu transformaci:

$$F_R(s) = \frac{U(s)}{E(s)} = K \cdot (1 + \frac{1}{T_I \cdot s}) = K + \frac{K_I}{s} \quad kde \quad K_I = \frac{K}{T_I} \tag{5.20}$$

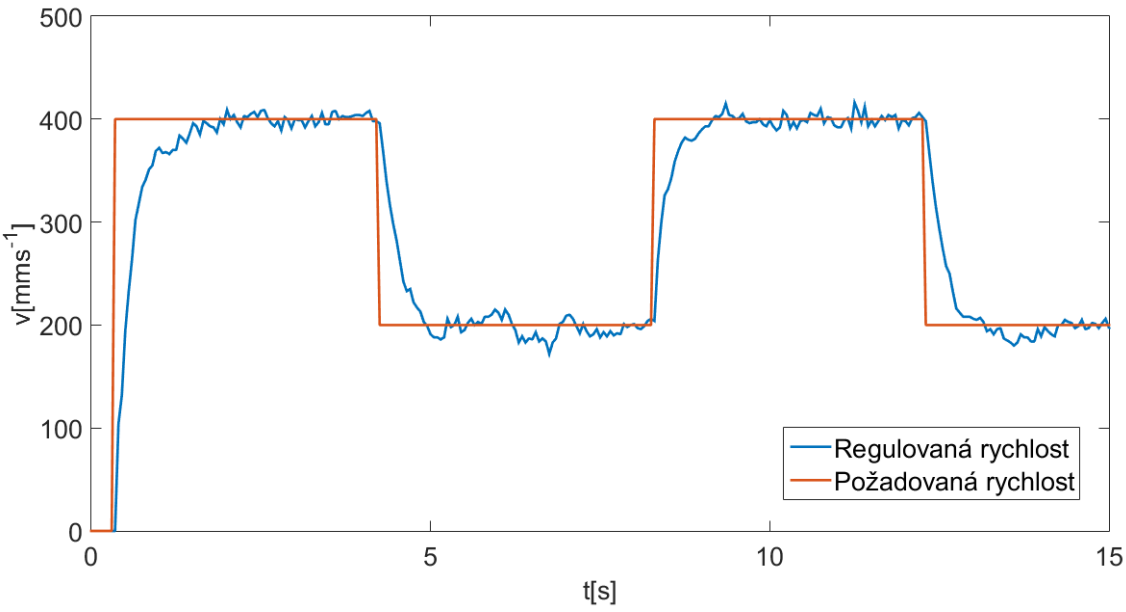
5.2.3 Ladění parametrů regulátoru použitím systému REX

Pro vhodné řízení našeho modelu pohonu je nutné stanovit konstanty PI regulátoru, vyhovující našemu modelu motorů. Pro ladění parametrů PI regulátoru byl použit systém REX. Je to řídící systém pro řízení strojů, technologií a procesů. Jeden z nástrojů REX souboru softwarových nástrojů je RexDraw, který umožňuje grafický návrh regulačních obvodů. Další nástroj je pak RexView, který umožňuje vykreslit měřená data v RexDraw. V programu RexDraw jsme si tedy navrhli náš regulační obvod pomocí funkčních bloků. Blok, který realizuje náš systém prvního řádu s dopravním zpožděním (přenos 5.12), se nazývá *FOPDT*. V tomto bloku se nastaví všechny identifikované konstanty motoru, které byly zjištěny v předchozích metodách v programu Matlab. Další blok našeho obvodu se pak nazývá *PIDMA*. Tento blok reprezentuje PID regulátor s momentovým autotunerem.



Obrázek 5.11: Regulační obvod s sestavený v programu RexDraw

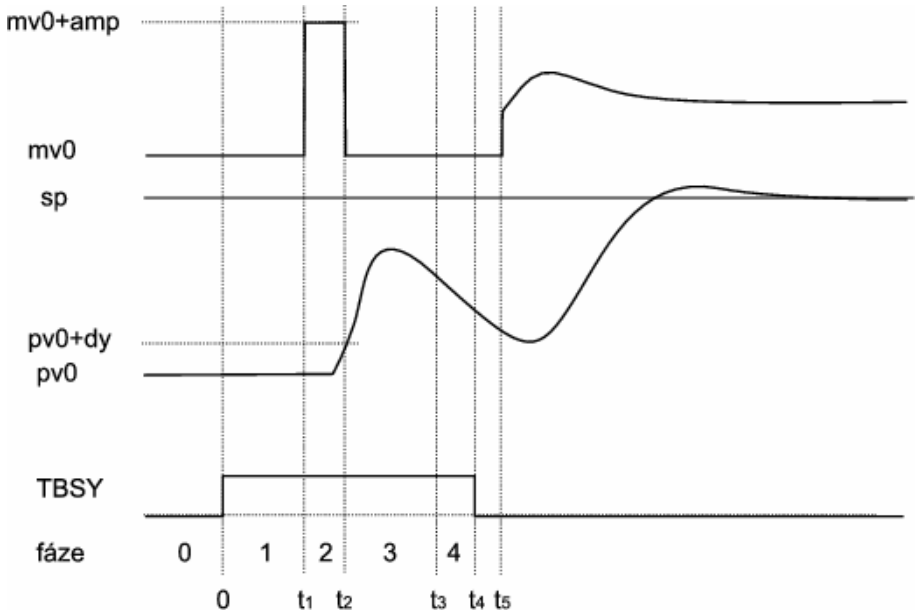
Blok PIDMA, v našem případě nastavený ve funkci pouze jako PI regulátor, měl na začátku nastavené libovolné hodnoty konstant regulátoru K a T_I, poté byl spuštěn momentový autotuner, který pro daný model systému s přenosem (5.12) navrhl parametry K a T_I PI regulátoru s přenosem (5.20). Takto navržené parametry byly následně použity v algoritmu řízení (5.19), jenž byl implementován v našem robotu. Regulace s těmito parametry na požadovanou rychlost je vyobrazena na následujícím obrázku:



Obrázek 5.12: Ukázka funkce regulátoru navrženého blokem PIDMA

PID regulátor s momentovým autotunerem

PIDMA blok použitý ve schématu (5.11), je možné použít jako P, I, D, PI, PD, nebo PID regulátor, záleží na požadovaném nastavení uživatele. Běžná funkce regulátoru je zajištěna logickou hodnotou false na vstupu označeném jako *MAN*. Je-li však na tento vstup přivedena logická hodnota true, je spuštěn momentový autotuner. Na začátku experimentu je odhadnut drift a šum regulované veličiny. Poté je systém vybuzen nastaveným obdélníkovým pulsem, po němž jsou získány první tři momenty přenosové funkce řízeného systému. Tyto momenty pak tvoří společně s okruhem systémů s monotonní přechodovou charakteristikou množinu přípustných systémů. Parametry regulátoru se pak navrhnou pro tuto množinu systémů tak, aby bylo minimalizováno určité optimalizační kritérium, a aby byl navržený regulátor robustní [7], [19].



Obrázek 5.13: Funkce momentového autotuneru [7]

Při spuštěném experimentu (hodnota *MAN* je true) je hodnota na vstupu *hv* kopírovaná na výstup *mv*. Aktivní signál *TBSY* značí aktivní experiment autotuneru. Fáze 1, je fáze odhadu driftu a šumu, fáze 2 pak vyvolá obdélníkový puls, který skončí poté, co hodnota regulované veličiny překročí hodnotu *dy*. Po skončení tohoto experimentu (*TBSY* je false), se na výstupu *pk*, *pti* zobrazí navržené hodnoty zvoleného PI regulátoru K a T_I.

5.2.4 Ladění parametrů regulátoru použitím metody GMK

Další způsob ladění konstant PI regulátoru je metoda GMK, neboli geometrické místo kořenů. Metoda pro zobrazení závislosti kořenů nějakého polynomu na jeho parametrech v komplexní rovině. Máme-li tedy přenos nějakého systému, vykreslíme nuly a póly tohoto přenosu v komplexní rovině. Návrh regulátoru pomocí GMK pak

závisí na požadovaných vlastnostech regulace, tedy na době regulace a hodnotě přeregulovaní. Z těchto parametrů se vypočítají požadované póly, v jejichž blízkosti se volí nuly regulátoru. Přenos regulátoru s těmito nulami a přenos řízeného systému pak tvoří otevřenou regulační smyčku, pro kterou se vykreslí GMK. Z GMK se pak odečítá zesílení v blízkosti umístění požadovaných pólu. Z tohoto zesílení je pak možné dopočítat zbylé parametry regulátoru.

Pro lepší představu budeme do následujících vztahů dosazovat konkrétní hodnoty. Jako první si stanovíme požadavky na uzavřený regulační obvod. Po regulátoru budeme požadovat, aby vygeneroval takový akční zásah, že doba regulace bude 0,5 sekund a dovolíme i 10% přeregulování.

Doba regulace: $T_{reg} \leq 0.05s$
Maximální relativní přeregulování: $\sigma_{max} = 0.1$

Nyní použijeme tyto návrhové požadavky do vztahů pro souvislost doby regulace T_{reg} a maximálního přeregulování σ_{max} s činitelem relativního tlumení ξ a netlumenou frekvencí ω_n:

$$\begin{aligned} \xi^* &= \frac{\frac{ln(\sigma_{max})}{\pi}}{\sqrt{1+(\frac{ln(\sigma_{max})}{\pi})^2}} = 0,591 \cong 0,6 \\ \omega_n^* &= \frac{4,6}{\xi^* \cdot T_{reg}} = 15,33 rad/s \end{aligned} \tag{5.21}$$

Požadovaný přenos druhého řádu:

$$F_S^*(s) = \frac{\omega_n^{*2}}{s^2 + 2\xi^*\omega_n^* s + \omega_n^{*2}} \tag{5.22}$$

Dosazením hodnot (5.21) do přenosu (5.22) získáme přenos:

$$F_S^*(s) = \frac{235}{s^2 + 18,396s + 235} \tag{5.23}$$

s póly $p_{1,2} = -\xi^*\omega_n^* \pm j\omega_n^*\sqrt{1-\xi^{*2}} = -9,198 \pm j12,264$

Máme-li navržený systém druhého řádu, o kterém víme, že splňuje dané požadavky, musíme vypočítat parametry regulátoru tak, aby společně s přenosem řízného systému dával totožný přenos. Určíme si tedy, jak bude vypadat přenos uzavřené regulační smyčky s PI regulátorem a se systémem prvního řádu bez dopravního zpoždění, které je zanedbáno z důvodu nutnosti použití Padého aproximace.

Zvolme jeden z identifikovaných přenosů prvního řádu:

$$F_S(s) = \frac{1,126}{0.187s + 1} \tag{5.24}$$

Nyní pro tento přenos a přenos PI regulátoru (5.20) vypočítáme přenos uzavřené regulační smyčky (viz obrázek 5.10):

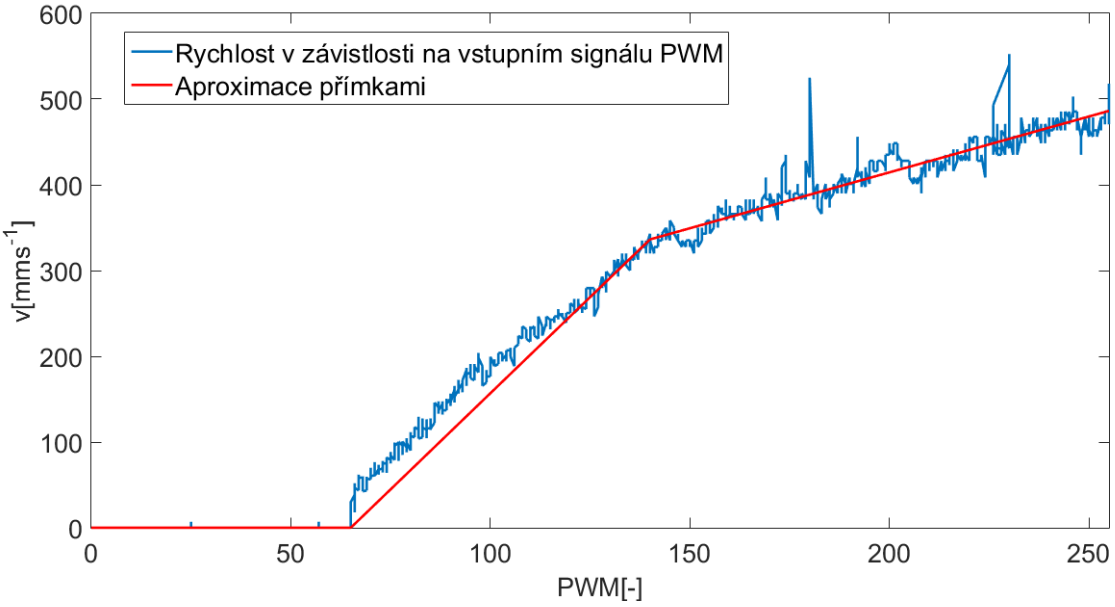
$$F_{y,w}(s) = \frac{F_S(s) \cdot F_R(s)}{1 + F_S(s) \cdot F_R(s)} = \frac{1,126(Ks + K_I)}{s^2 + (\frac{1+1,126 \cdot K}{0,187})s + (\frac{1,126 \cdot K_I}{0,187})} \tag{5.25}$$

Výsledkem je přenos druhého řádu. Porovnáním tohoto přenosu s přenosem (5.23)získáme parametry regulátoru K a K_I:

$$\begin{aligned} \frac{1+1,126\cdot K}{0,187} &= 18,396 \Rightarrow K = 2,167 \\ \frac{1,126\cdot K_I}{0,187} &= 235 \Rightarrow K_I = 39,029 \end{aligned} \tag{5.26}$$

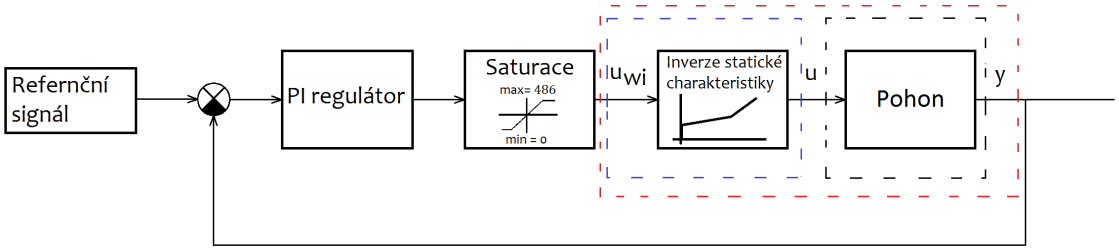
5.3 Kompenzace nelinearity zatíženého pohonu

Cílem této úlohy bude linearizovat závislost mezi vstupní veličinou (PWM kódem) a výstupní veličinou (rychlostí). Jak už bylo řečeno, stejnosměrné motory umístěné na robotu vykazují nelinearitu. Ta se skládá z necitlivosti způsobené zatížením motoru robotem. Další složka nelinearity je pak nelinearita v samotné dynamice pohonu. Tyto dvě složky jsou vidět na následující statické charakteristice motoru:



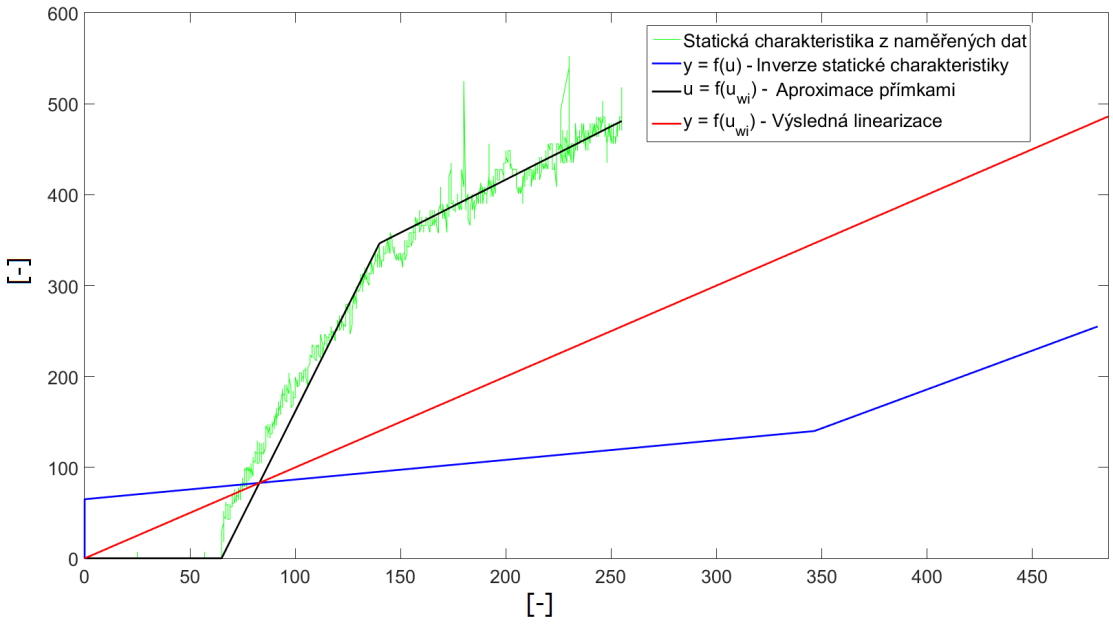
Obrázek 5.14: Statická charakteristika levého motoru

Z obrázku (5.14) je patrné, že necitlivost motoru se pohybuje do 65-té hodnoty PWM signálu. Nelinearitu dynamiky lze pak aproximovat dvěma přímkami. Nezapomínejme, že motor na levé a pravé straně se vyznačuje každý svoji vlastní dynamikou a svojí statickou charakteristikou. Určené přímky se hodí pro potlačení nelinearity motoru. Stanovíme-li inverzní přímky těchto přímek, jejich použití na výstup regulátoru bude mít za následek linearizaci celkového výstupu regulátoru a potlačení obou uvedených nelinearit.



Obrázek 5.15: Schéma postupu při kompenzaci nelinearit

Můžeme si všimnout, že výstup regulátoru není 0 až 255, ale 0 až maximální možná rychlost motoru. Tento akční zásah, který jsme označili u_{wi}, je poté přepočítán rovnicí přímky pro inverzní statickou charakteristiku a tento výstup je označen jako u. Uvedeme-li příklad s konkrétními hodnotami, tak je-li například výstup regulátoru u_{wi} 25, pak výstup po přepočtu inverzní funkcí u je přibližně 70. Tímto postupem byla tedy potlačena necitlivost motoru a zároveň „vráceno“ statické zesílení, zanedbané při identifikaci parametrů motoru a ladění parametrů regulátoru. V následující charakteristice je vykreslený princip této kompenzace:



Obrázek 5.16: Statická charakteristika a její inverze

Kapitola 6

Realizace navržených algoritmů a ověření funkčnosti

V této kapitole se budeme zabývat realizací navržených algoritmů a experimenty, které jsme provedli pro ověření námi navržených parametrů regulátoru. Provedené experimenty jsme vyobrazili v programovém prostředí Matlab. Robot byl při těchto experimentech postaven na co nejhladší podlaze s co nejméně nerovnostmi, na které byla páskou vytvořena referenční trajektorie pro lepší pozorování odchýlení robotu od této trajektorie. Podle zvolené trajektorie, tedy její délky, byl vypočítán čas, po který se robot musel pohybovat při zvolené rychlosti, aby dosáhl právě dané trajektorie. Nesmíme zapomenout, že v našem případě se jedná pouze o programové řízení robotu, tedy k řízení dochází pouze na základě naměřených dat z enkodéru.

6.1 Implementace navržených algoritmů

Celý algoritmus řízení byl poskládán z jednotlivých úkonů popsaných v této práci. V kódu Arduina byl jako první realizovány moduly přerušení od enkodérů (včetně modulu časovače) a moduly filtrace měření impulsů. V rámci optimalizace kódu se převáděly hodnoty v plovoucí řádové čárce (datový typ float) na hodnoty celočíselné (datový typ integer), a to násobením třícifernou celočíselnou konstantou, kterou se poté muselo dělit pro dosažení požadované jednotky rychlosti či dráhy. Tím vznikaly nevyhnutelně chyby v zaokrouhlování (přetypování používá zaokrouhlování dolů), které měly vliv na výslednou rychlost.

Následoval modul výpočtu rychlostí jednotlivých kol, a protože rychlosti kol by v jednotkách SI (ms^{-1}) nabývaly hodnot menších než jedna, což by znamenalo nutnost použít datový typ s plovoucí řádovou čárkou, byly rychlosti převedeny na jednotku v $mms{-}1$ a ukládány do celočíselného typu integer. Opět sice došlo k chybě zaokrouhlováním, ale jelikož byly poté rychlosti používány v regulátoru, stačilo použít celočíselné aritmetiky, což šetřilo výpočetní výkon mikrokontroléru.

Poté byly v kódu obsaženy moduly regulátorů a moduly pro kompenzaci nelinearity pohonů, ve kterých se prováděl přepočet výstupu regulátorů pomocí rovnic

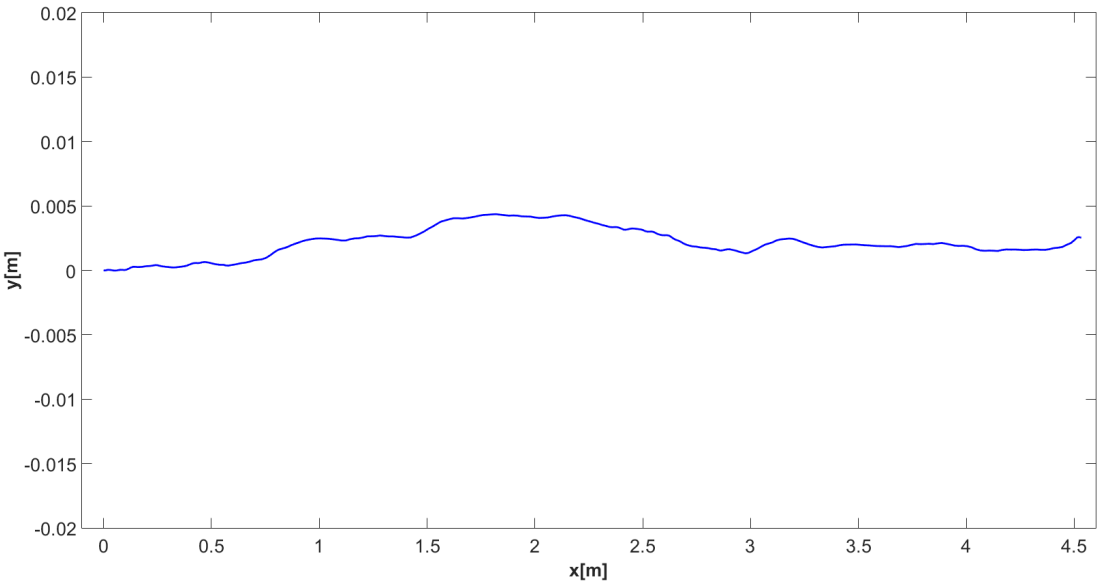
přímky. Tady ovšem musíme upozornit, že takový výpočet opět bere mikrokontroléru výpočetní výkon a lepší způsob by byl, kdyby hodnoty byly tabelované a uložené v paměti. Pro posílaní dat z robotu do stolního počítač byl napsán modul pro komunikaci, využívající bezdrátový modul nRF24L01. V tomto modulu se ukládaly všechny potřebné hodnoty do datové struktury, která se poté posílala na druhou desku Arduina, připojenou k počítači, kde byla data poté použita pro offline vizualizaci (viz Vizualizační nástroj).

Aby se všechny moduly nevykonávali příliš často, byl použit takzvaný plánovač. Ten umožnil volat všechny moduly se zvolenou periodou, což byla v našem případě již zmíněná perioda T_S s 50 ms. V plánovači nebyly zahrnuty moduly enkodéru, které se vyvolávaly přerušením od enkodérů, stejně tak modul časovače.

Celý kód poskládaný z jednotlivých modulů je uveden v příloze A.

6.2 Vizualizační nástroj

Pro zobrazení výsledků naměřených experimentů byl použit program Matlab. Tento program, původně určený pro práci z maticemi, je používán jako nástroj pro matematické výpočty, maticové počty a algoritmy. Mimo jiné má tento program spoustu nadstaveb, jednu z nich jsme již použili při návrhu regulátoru. Nám pro vizualizaci bude stačit, že na základě vstupních dat, v našem případě tedy naměřených dat, je tento program schopen vykreslit graf. Než jsme však mohli nějaká data do Matlabu vkládat, bylo nutné zajistit bezdrátový přenos dat z robotu do počítače. Přenos byl zajištěn bezdrátovými moduly nRF24L01, jeden byl připojen na desku Arduino robotu ve funkci vysílače a druhý taktéž připojen na desku Arduino, ale k počítači ve funkci přijímače. Data byla posílána z vysílače do přijímače s periodou T_s 50 ms. Takto získaná data byla uložena do počítače a poté načtena do programu Matlab pro zpracování. Vizualizace tedy probíhala offline na základě naměřených dat, hlavně tedy ujeté vzdálenosti každého kola. Tyto vzdálenosti a informace o geometrii vozidla, byly použity ve výpočtech uvedených v druhé kapitole. Výsledkem těchto výpočtu bude poloha robotu v kartézské soustavě souřadnic.

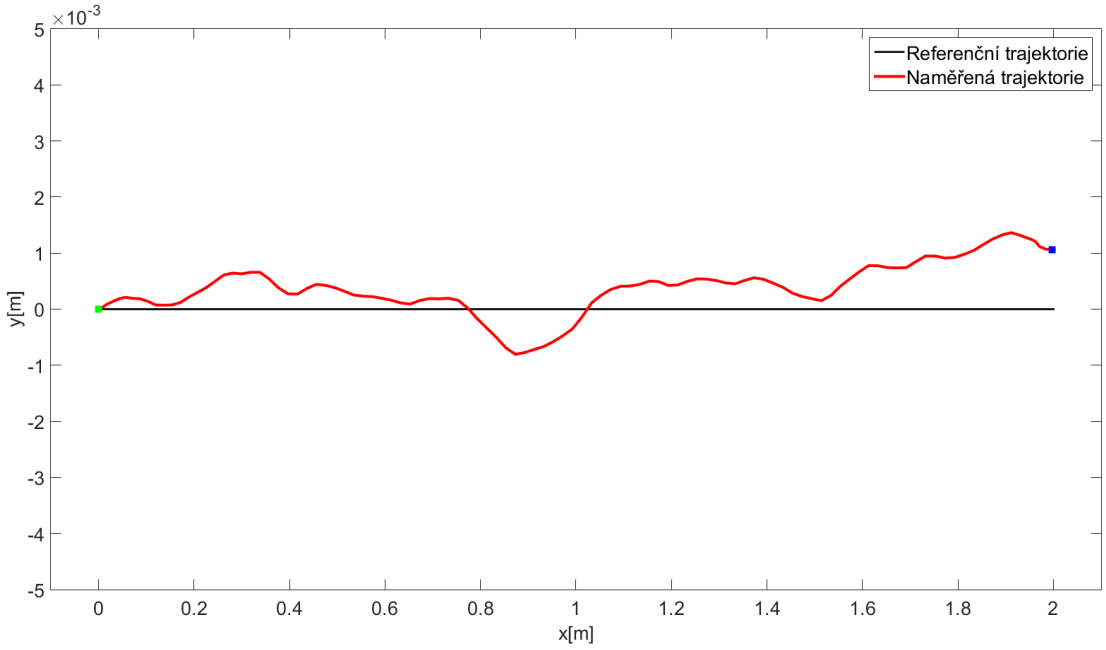


Obrázek 6.1: Ukázka trajektorie robotu v kartézské soustavě souřadnic

6.3 Experimenty

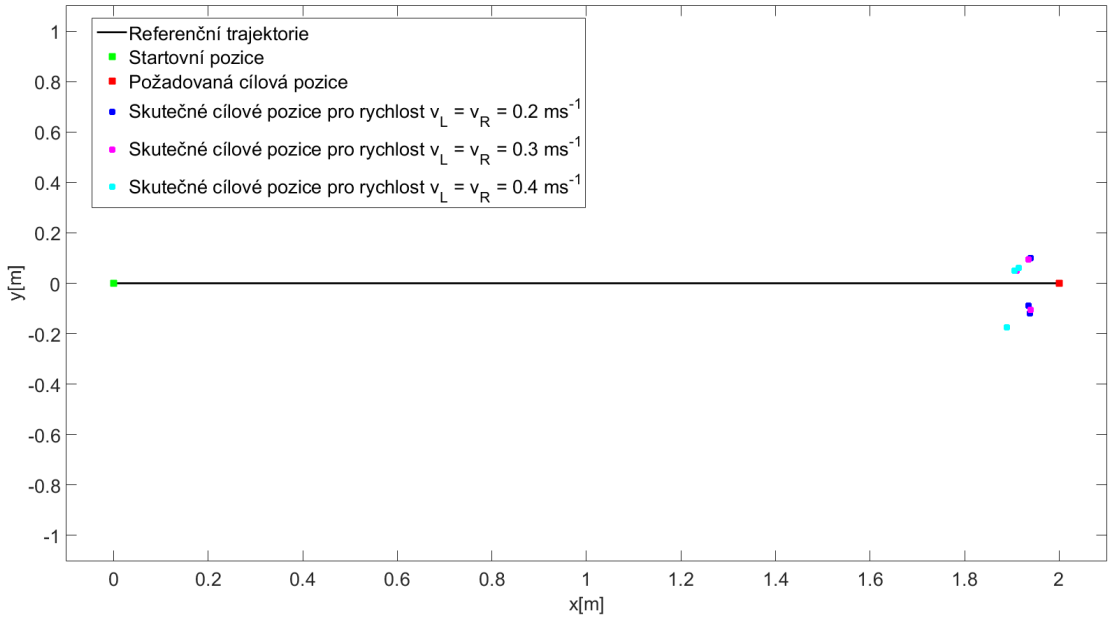
6.3.1 Přímá dráha

Rychlosti robotu pro přímou dráhu se budou rovnat $v_L = v_R$. V tomto případě byly zadány rychlosti kol $v_L = v_R = 0.4ms^{-1}$. Délka dráhy, kterou má robot ujet je 2 m. Čas, po který budou zadané rychlosti přiváděny do regulačních obvodů obou motorů, bude činit 5 sekund, poté bude na vstupy přivedena nulová rychlost. Na následujícím grafu je vykreslena zadaná referenční trajektorie a trajektorie robotu vypočtená z naměřených dat z obou enkodérů:



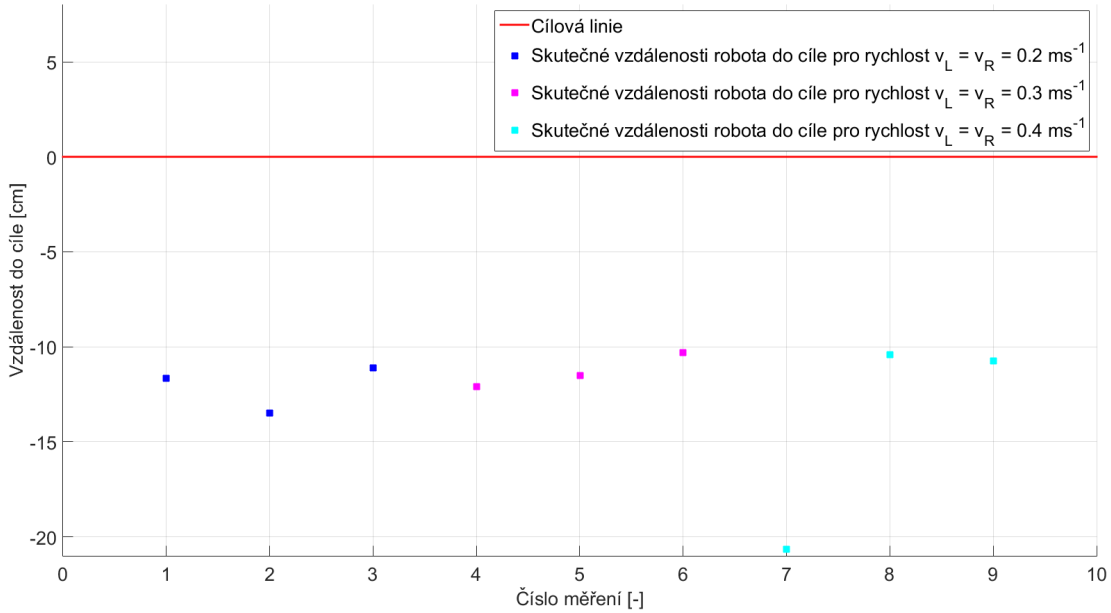
Obrázek 6.2: Naměřená trajektorie robotu pro zadanou přímou dráhu

Z předchozího grafu vyčteme, že robot se dostane od referenční trajektorie nejdále na hodnotu přes jeden milimetr. Musíme si ovšem uvědomit, že tato trajektorie je vykreslena na základě dat naměřených z enkodérů. Jak již bylo naznačeno, toto měření není příliš přesné, jelikož s větší ujetou drahou se zvětšuje i chyba měření, což může být zapříčiněno například zaokrouhlováním při ukládání proměnné. Na následujícím obrázku je pak možné vidět, že ve skutečnosti, se robot nacházel od cílového bodu ve větší vzdálenosti.



Obrázek 6.3: Konečné skutečné polohy robotu pro zadanou přímou dráhu

V tomto obrázku můžeme vidět několik provedených experimentů pro různé rychlosti. Vykreslené pozice robotu jsou vždy před cílovým bodem, ihned tedy můžeme říci, že robot nikdy nedojel úplně do cíle a nejedná se tedy o náhodnou chybu, což je lépe ukázáno na dalším obrázku:

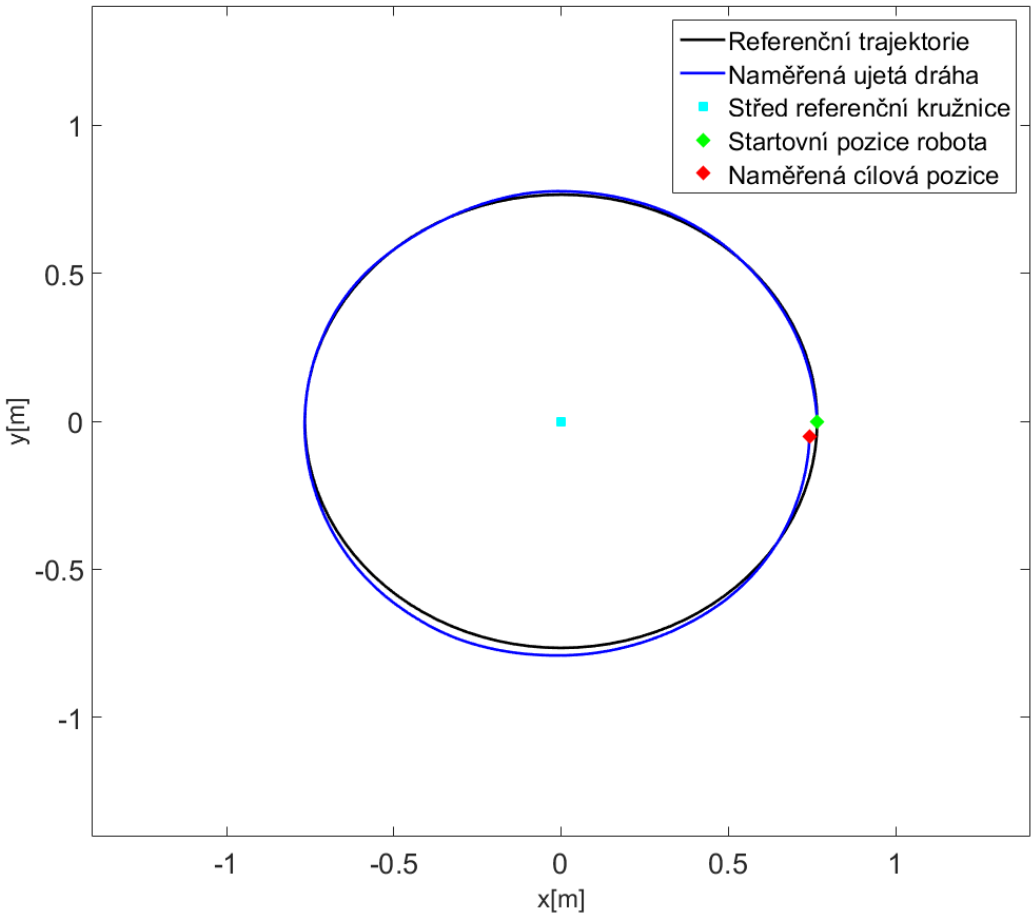


Obrázek 6.4: Zobrazení vzdáleností mezi robotem a cílem pro několik měření

Z tohoto lze usoudit, že se jedná o systematické chyby. Možné důvody, proč k takové chybě dochází, probereme ve Zhodnocení výsledků.

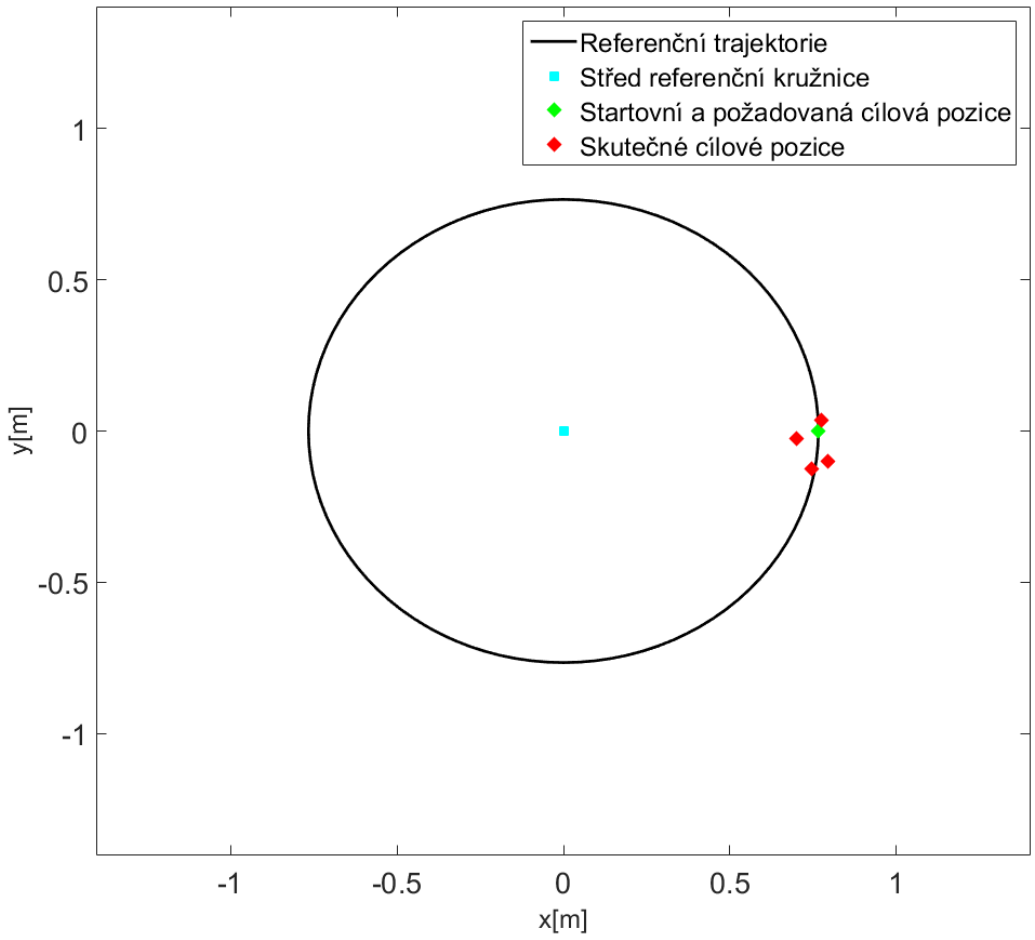
6.3.2 Kruhová dráha

Pro kruhovou dráhu musí být rychlosti robotu rozdílné, konstantní a ve stejném směru $v_L \neq v_R$. Pro levé kolo byla zvolena rychlost $v_L = 0.35ms^{-1}$ a rychlost pravého kola $v_R = 0.4ms^{-1}$. Pro zobrazení referenční trajektorie je tedy třeba znát poloměr kruhu, který by robot měl sledovat. Zjištění poloměru kružnice docílíme dosazením zadaných rychlostí a rozchodu kol do výrazu (2.3). Pro uvedené rychlosti bude tedy poloměr kružnice činit 0,766 m:



Obrázek 6.5: Naměřená trajektorie robotu pro zadanou kruhovou dráhu

U kruhu podobně jako u přímé dráhy dochází opět k rozdílu mezi naměřenou a skutečnou trajektorií, nehledě na rozdíl mezi skutečnou a referenční trajektorií. Lze si i povšimnout, že v horním půlkruhu je trajektorie robotu blízká té referenční. V nějakém okamžiku dále pak mohlo jedno z kol robotu proklouznout, či mohl najet na určitou nerovnost, která jej vychýlila pouze na okamžik, ale protože zde není implementován žádný jiný algoritmus než odometrie na základě dat z enkodérů, nedokáže se robot vrátit zpět na referenční trajektorii.



Obrázek 6.6: Konečné skutečné polohy robotu pro zadanou kruhovou dráhu

6.4 Zhodnocení výsledků

Celkově pozorujeme, že robot pokaždé nedojel do cíle a vzdálenost, která do cíle zbývala, byla v každém měření podobná. Maximální chyba nepřesáhla 11 % z plánované trasy. Jsou to tedy chyby, které jsou částečně systematické. Příčin těchto chyb může být několik. Začneme podmínkami při kterých byly dané experimenty provedeny. Jeden z hlavních problémů může být podlaha, na které byl robot postaven. Ta nemusí být vždy úplně vyhlazená a na jejím povrchu se mohou vyskytovat nepatrná porušení či nějaké objekty jako malé kamínky apod. Dále pak může docházet k prokluzování kol, kvůli kterému může dojít k otočení kola, avšak robot zůstává na místě. Chyby pak mohou vznikat v geometrii robotu, jedna z chyb může být rozdílný průměr kola na každé straně či špatně naměřený poloměr, který je základem výpočtu rychlostí. Další chyba pak může být špatně změřený rozchod kol a nesouosost kol. Ve zpracování mohl být špatně identifikovaný model motoru, na který se navrhoval regulátor.

Jednou z možností jak snížit náhodné chyby v měření je použití jednoho z možných filtrů, jako je Bayesův filtr, histogramový filtr, částicový filtr či Kalmanův filtr,

který používá nejen naměřená data ale i dynamický model systému. Dále by pak bylo vhodné doplnit odometrii o jiný prostředek lokalizace, např. akcelerometr či gyroskop.

Jelikož jsme použili jen programové řízení pohybu po zadané trajektorii s využitím inverzní kinematiky bez zpětné vazby, mělo by zavedení zpětné vazby pozitivní vliv na účinnost sledování referenční trajektorie robotem. Následkem toho by bylo možné sledovat trajektorie složitějších tvarů a porovnávat současné souřadnice robotu s požadovanými, což by se dalo považovat za nadřazený řídící systém. Tato možnost je teoreticky rozebrána v následující kapitole.

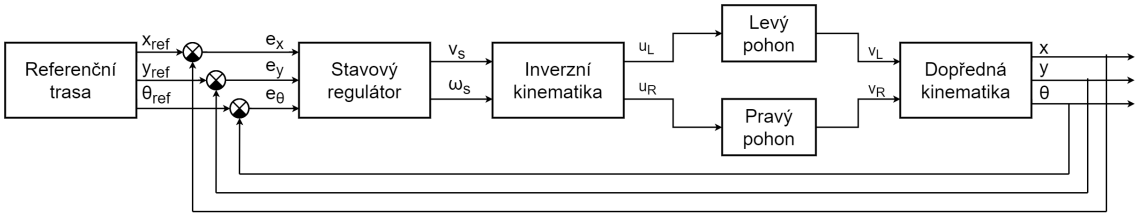
Kapitola 7

Zpřesnění řízení pro sledování zadané trajektorie

V této kapitole se budeme zabývat algoritmem, který bude mít za úkol řídit rychlost kol tak, aby sledoval námi zadanou trajektorii. Rozebereme jednotlivé kroky potřebné pro splnění tohoto požadavku. Je nutno poznamenat, že následující algoritmus je teoretickým návrhem pro řešení sledování zadané trajektorie a nebyl implementován v robotu. Rovnice v této kapitole byly odvozeny z použitého materiálu uvedeného v literatuře pod číslem [20].

7.1 Implementace sledování zadané trajektorie se zpětnou vazbou

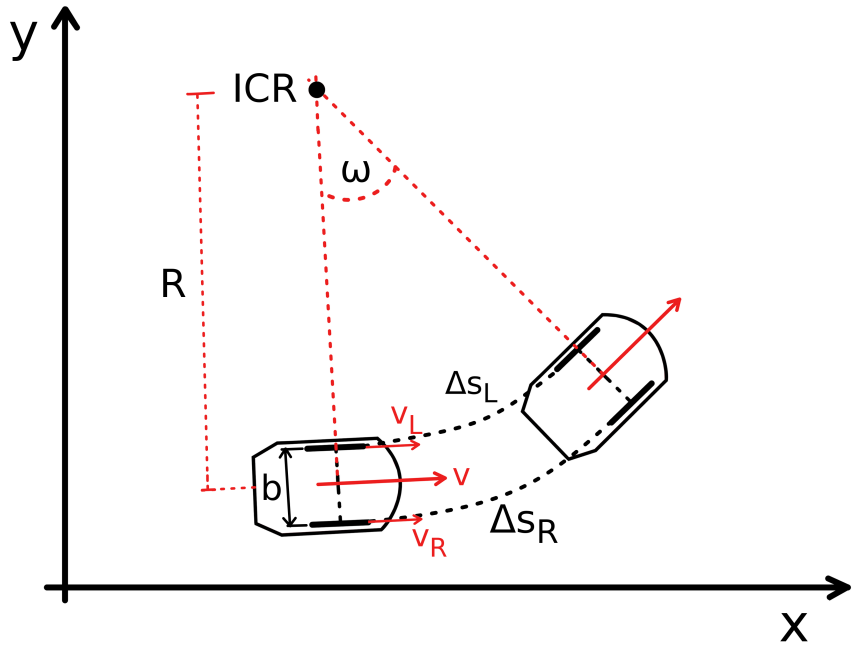
Aby bylo možné sledovat zadanou trajektorii, je ji nejprve nutné vyjádřit pomocí souřadnic v kartézské soustavě, o tuto činnost se stará generátor referenční trasy. Dále pak odečtením souřadnic polohy robotu (získané dopřednou kinematikou) od referenční trasy je definována regulační odchylka. Přivedením této odchylky na vstup námi zvoleného stavového regulátoru je vypočteno potřebné řízení, které je pomocí inverzní kinematiky převedeno na požadované lineární rychlosti jednotlivých kol. Tyto rychlosti jsou pak vstupem jednotlivých regulačních obvodů, které zajišťují požadovanou rychlost kol. Robot by měl po těchto krocích sledovat zadanou trajektorii.



Obrázek 7.1: Schéma jednotlivých kroků sledovaní zadané trajektorie

7.1.1 Dopředná kinematika diferenčně řízeného vozidla

Diferenčně řízeným vozidlem rozumíme robotu, jehož kola jsou řízena nezávisle na sobě, takový robot se tedy může otočit na místě. V dopředné kinematice jsou dány hnací souřadnice (polohy pohonů, tedy motorů) a určují se hnané souřadnice (polohy hnaných mechanických částí, tedy poloha těžiště celého robotu). U robotu je tedy dán časový průběh pohonů a určuje se pohyb těžiště robotu.



Obrázek 7.2: Popis kinematiky robotu

Za hnací souřadnice tedy uvažujeme rychlosti robotu ve směru osy x a y, dále pak úhlovou rychlost otáčení kolem bodu ICR (Instantaneous Center of Rotation). Jelikož se robot může pohybovat pouze dopředu a dozadu, nikoliv do stran, rychlost ve směru osy y robotu bude nulová. Pro rychlosti jednotlivých kol platí:

$$\begin{aligned} v &= R \cdot \omega \\ v_L &= (R - \tfrac{b}{2}) \cdot \omega \\ v_R &= (R + \tfrac{b}{2}) \cdot \omega \end{aligned} \tag{7.1}$$

Lineární rychlost ve směru osy x robotu pak bude dána rychlostmi obou kol dle vztahu:

$$v = \frac{v_L + v_R}{2} \tag{7.2}$$

Úhlová rychlost otáčení robotu kolem ICR je pak dána vztahem:

$$\omega = \frac{\omega_L - \omega_R}{b} \tag{7.3}$$

Tyto dvě veličiny tvoří vektor souřadného systému robotu, označeného jako R:

$$\dot{\xi}_R = \begin{bmatrix} v \\ 0 \\ \omega \end{bmatrix} \tag{7.4}$$

Hnané souřadnice jsou dány polohou robotu na ose x, y a úhlem natočení robotu θ. Tento vektor pak bude popisovat polohu robotu v inerciální soustavě, značené jako I:

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{7.5}$$

Avšak bavíme-li se o kinematice, je potřeba zjistit rychlosti v inerciální soustavě, víme-li tedy že platí:

$$v_x = \frac{dx}{dt} \tag{7.6}$$

Můžeme vyjádřit vektor poloh (7.5) jako vektor rychlostí:

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{7.7}$$

Jak již bylo řečeno, jedná-li se o dopřednou kinematiku, je třeba uskutečnit převod daného vektoru (7.4) ze soustavy robotu R do inerciální soustavy I. Toho se docílí pomocí rotační matice značené jako R_{RI}:

$$\begin{aligned} \dot{\xi}_I &= R_{RI} \cdot \dot{\xi}_R \\ \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} &= \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ 0 \\ \omega \end{bmatrix} = \begin{bmatrix} v \cdot cod(\theta) \\ v \cdot sin(\theta) \\ \omega \end{bmatrix} \end{aligned} \tag{7.8}$$

7.1.2 Generování referenční trasy

Aby bylo možné sledovat referenční trasu, neboli požadovanou trasu, kterou bude robot sledovat, je nutné ji nejdříve vygenerovat. Jelikož se poloha robotu zobrazuje v inerciální soustavě I, neboli poloha daná souřadnicemi x,y a úhlem natočení robotu θ, bude referenční cesta dána souřadnicemi v téže soustavě. Například požadavek na rovnou dráhu znamená konstantní hodnotu souřadnice y (popřípadě x) a uzavřený

interval hodnoty x (respektive y). Požadavek je, aby poloha robotu, daná souřadnicemi (x(t),y(t)), byla co nejvíce podobná referenční poloze ($x_{ref}(t)$, $y_{ref}(t)$) pro časový okamžik t $\in$ [0, T]. Derivováním těchto souřadnic dostaneme referenční rychlosti $\dot{x}_{ref}$, $\dot{y}_{ref}$ v osách x a y. Referenční úhel natočení θ_{ref} se pak získá z rychlostí $\dot{x}_{ref}$, $\dot{y}_{ref}$ pomocí funkce ATAN2:

$$\theta_{ref}(t) = ATAN2(\dot{y}_{ref}(t), \dot{x}_{ref}(t)) + k\pi, \quad k = 0, 1 \tag{7.9}$$

Funkce ATAN2 je čtyřkvadrantová inverzní tangens funkce (tan^{-1}), neboli arkus tangens. Vrací úhel mezi osou y a přímkou, na níž leží počátek (0; 0) a bod se souřadnicemi ($\dot{y}_{ref}$, $\dot{x}_{ref}$).

Následuje přepočet rychlostí $\dot{x}_{ref}$, $\dot{y}_{ref}$ v osách x,y na celkovou rychlost referenčního robotu v_{ref} a úhlovou rychlost $\omega_{ref}(t)$:

$$v_{ref}(t) = \pm\sqrt{\dot{x}^2_{ref}(t) + \dot{y}^2_{ref}(t)} \tag{7.10}$$

$$\omega_{ref}(t) = \frac{\ddot{y}_{ref}(t) \cdot \dot{x}_{ref}(t) - \ddot{x}_{ref}(t) \cdot \dot{y}_{ref}(t)}{\dot{x}^2_{ref}(t) + \dot{y}^2_{ref}(t)} \tag{7.11}$$

7.1.3 Stavový regulátor

Pro potřeby sledování referenční trajektorie je nutné použít prvek, který bude porovnávat polohu skutečného robotu s referenční polohou v daném čase. Jedna z možností je použít stavový regulátor. Ten plní standardní funkci regulátoru, na základě regulační odchylky stavu na vstupu generuje takové řízení na výstupu regulátoru, které zapříčiní sledování referenční trasy robotem. Regulační odchylka je definována následovně:

$$\begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} x_{ref} - x \\ y_{ref} - y \\ \theta_{ref} - \theta \end{bmatrix} \tag{7.12}$$

Jelikož je předchozí vztah v inerciální soustavě I, je nutno ji převést do souřadného systému robotu R. Toho docílíme rotační maticí R_{IR}:

$$e = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} \tag{7.13}$$

Jako další je nutné určit řídící lineární rychlost v_s a řídící úhlovou rychlost ω_s těžiště robotu v soustavě robotu R, na základě vypočtených referenčních veličin robotu rychlosti v_{ref} a úhlové rychlosti ω_{ref} (viz 7.10 a 7.11) v inerciální soustavě I aplikací nelineární transformace:

$$\begin{aligned} v_s &= v_{ref} \cdot cos(e_3) - u_1 \\ \omega_s &= \omega_{ref} - u_2 \end{aligned} \tag{7.14}$$

Dynamický model odchylek je pak popsán referenční lineární a úhlovou rychlostí a řízením dynamického modelu:

$$\dot{e} = \begin{bmatrix} 0 & \omega_{ref} & 0 \\ -\omega_{ref} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot e + \begin{bmatrix} 0 \\ sin(e_3) \\ 0 \end{bmatrix} \cdot v_{ref} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{7.15}$$

kde u_1 a u_2 jsou řídící vstupy dynamického modelu. Linearizací předchozího vztahu okolo bodu referenční trajektorie jsou získány lineární časové variantní rovnice, nyní se stavem odchylek e a se vstupem (u_1, u_2) souřadného systému robotu R. Regulační odchylky e ovlivňují řídící vstupy stavového regulátoru u_1 a u_2:

$$\begin{aligned} u_1 &= -k_1 \cdot e_1 \\ u_2 &= -k_2 \cdot sign(v_{ref}(t)) \cdot e_2 - k_3 \cdot e_3 \end{aligned} \tag{7.16}$$

Konstanty k_1, k_2, k_3 jsou parametry zesílení regulátoru. Tyto parametry se určí z charakteristické rovnice požadované uzavřené smyčky:

$$(p + 2 \cdot \xi \cdot a) \cdot (p^2 + 2 \cdot \xi \cdot a \cdot p + a^2), \quad \xi, a > 0 \tag{7.17}$$

kde
ξ: tlumení z intervalu (0,1)
a: přirozená úhlová frekvence a > 0

Z předchozího vztahu lze pozorovat, že uzavřená smyčka bude mít tři póly:

$$\begin{aligned} p_1 &= -2 \cdot \xi \cdot a \\ p_{23} &= -\xi \cdot a \pm j \cdot a \cdot \sqrt{1 - \xi^2} \end{aligned} \tag{7.18}$$

Konstanty k_1, k_2, k_3 se tedy volí dle následujících vztahů:

$$k_1 = k_3 = 2 \cdot \xi \cdot a, \quad k_2 = \frac{a^2 - \omega_s^2(t)}{|v_{ref}(t)|} \tag{7.19}$$

Pro případ, kdy v_{ref} se bude blížit nule, se bude zesílení k_2 blížit nekonečnu a tedy i generované řízení. Tento nechtěný efekt lze potlačit vyjádřením frekvence a jako $a = a(t) = \sqrt{\omega_{ref}^2(t) + b \cdot v_{ref}^2(t)}$, kde b je přídavný stupeň volnosti a dosazením do vtahu (7.19):

$$k_1 = k_3 = 2 \cdot \xi \cdot \sqrt{\omega_{ref}^2(t) + b \cdot v_{ref}^2(t)}, \quad k_2 = b \cdot |v_{ref}(t)| \tag{7.20}$$

Takto získané parametry zesílení lze pak použít pro výsledný vztah řízení, jenž byl získán dosazením rovnic (7.13) a (7.16) do rovnice (7.14):

$$\begin{aligned} v_s &= v_{ref} \cdot cos(\theta_{ref} - \theta) + k_1 \cdot [cos(\theta) \cdot (x_{ref} - x) + sin(\theta) \cdot (y_{ref} - y)] \\ \omega_s &= \omega_{ref} + k_2 \cdot sign(v_{ref}) \cdot [cos(\theta) \cdot (x_{ref} - x) - sin(\theta) \cdot (y_{ref} - y)] + k_3 \cdot (\theta_{ref} - \theta) \end{aligned} \tag{7.21}$$

7.1.4 Inverzní kinematika diferenčně řízeného vozidla

Jak již z názvu vypovídá, inverzní kinematika bude provádět opačný proces dopředné kinematiky. V inverzní kinematice jsou tedy dány hnané souřadnice (polohy hnaných mechanických částí, tedy poloha těžiště celého robotu) a určují se hnací souřadnice (polohy pohonů, tedy motorů). Z předchozí části byl získán předpis pohybu těžiště robotu daný lineární rychlostí v_s a úhlovou rychlosti ω_s (viz (7.21). Dosazením těchto proměnných do vztahu (7.1) získáme předpis pro požadované rychlosti levého a pravého kola:

$$\begin{aligned} v_L &= (R - \tfrac{b}{2}) \cdot \omega_s \\ v_R &= (R + \tfrac{b}{2}) \cdot \omega_s \end{aligned} \tag{7.22}$$

Vyjádření pomocí obou rychlostí je dáno vztahy:

$$\begin{aligned} v_L &= (v_s - \tfrac{b}{2} \cdot \omega_s) \\ v_R &= (v_s + \tfrac{b}{2} \cdot \omega_s) \end{aligned} \tag{7.23}$$

Tyto rychlosti jsou dále předány jako vstupy regulátorů jednotlivých kol.

Kapitola 8

Závěr

Hlavním cílem práce bylo navrhnout a realizovat řídící algoritmus, díky němuž bude robot sledovat požadovanou trajektorii.

V úvodní fázi práce proběhla implementace výpočtu rychlosti (ujeté dráhy) kol robotu na základě naměřených dat z enkodéru. V kódu byl navržen výpočet rychlosti pro vyšší počet otáček i pro nižší počet otáček. Testováním bylo zjištěno, že takto vypočtené rychlosti byly příliš zkreslené zaokrouhlovací chybou (celočíselným výpočtem), proto byl pro jednotlivé metody měření navržen IIR filtr, který filtroval naměřené impulsy z enkodéru. Obě metody výpočtu poté používaly filtrované impulsy pro stanovení rychlosti. Aby byla snížena celková chyba měření, byly rychlosti z obou metod pomocí aritmetického průměru, sloučeny do jedné rychlosti daného kola.

Další část se zabývala identifikací modelu motoru. Nejdříve byl stanoven přenos nezatíženého motoru, který byl popsán pomocí fyzikálních veličin. Do tohoto přenosu byly dosazeny pouze dvě změřené fyzikální veličiny a to indukčnost a odpor. Po zobrazení charakteristik nezatíženého modelu motoru bylo zjištěno, že tato forma přenosu bude pravděpodobně podobná přenosu modelu zatíženého motoru včetně budičů. Pro snadnější identifikaci byla zvolena obecná forma přenosu nekmitavého systému druhého řádu. Parametry tohoto přenosu byly identifikovány dvěma různými metodami, avšak obě metody vycházely z naměřené přechodové charakteristiky.

Na identifikovávané modely zatíženého motoru včetně budičů byly poté navrženy PI regulátory, které měly za úkol dobře regulovat rychlosti kol při skokových změnách požadované veličiny, při lineárním nárůstu požadované veličiny a pro konstantní požadovanou veličinu. Jako další byla řešena kompenzace nelinearity modelu pohonu, složená z necitlivosti zatíženého robotu, při malých vstupech na motor a z nelinearity dynamiky motoru. Potlačení nelinearity bylo řešeno pomocí inverzní statické charakteristiky.

V poslední fázi proběhla implementace navrženého řídícího algoritmu, na něž byly následně provedeny experimenty. Jelikož se jednalo o programové řízení pohybu po zadané trajektorii s využitím inverzní kinematiky bez zpětné vazby a výsledky experimentů nebyly přesvědčivé, byl na závěr teoreticky popsán algoritmus programového řízení se zavedenou zpětnou vazbou, který by znamenal zlepšení přesnosti sledování zadané trajektorie. Další možnost jak vylepšit sledování zadané trajektorie by byla implementace dalšího inerciálního senzoru. Ještě přesnější sledování by pak zajistila implementace jednoho z absolutních prostředků lokalizace.

Literatura

[1] Johann Borenstein, HR Everett, Liqiang Feng, et al. Where am i? sensors and methods for mobile robot positioning. *University of Michigan*, 119(120):27, 1996.

[2] Antonín Vojáček. Integrované mems gyroskopy. `http://automatizace.hw.cz/integrovane-mems-gyroskopy`, 2009.

[3] Engineering360 News Desk. Specifying an accelerometer: Function and applications. `http://insights.globalspec.com/article/1263/specifying-an-accelerometer-function-and-applications#`, 2015.

[4] ElectronicsTutorials. Position sensors. `http://www.electronics-tutorials.ws/io/io_2.html`, 2015.

[5] Anaheim Automation. Encoder guide. `http://www.anaheimautomation.com/manuals/forms/encoder-guide.php#sthash.9I8W2AbJ.dpbs`, 2017.

[6] Bill Messner and Dawn Tilbury. Dc motor position: System modeling. `http://ctms.engin.umich.edu/CTMS/index.php?example=MotorPosition&section=SystemModeling`, 2011.

[7] REX Controls s.r.o. Funkční bloky systému rex. `https://www.rexcontrols.cz/media/2.50.1/doc/CZECH/MANUALS/BRef/BRef_CZ.html`, 2016.

[8] Michael Barr. Pulse width modulation. *Embedded Systems Programming*, 14(10):103–104, 2001.

[9] Nabil Zhafri Mohd Nasir, Muhammad Aizzat Zakaria, Saifudin Razali, and Mohd Yazid bin Abu. Autonomous mobile robot localization using kalman filter. In *MATEC Web of Conferences*, volume 90, page 01069. EDP Sciences, 2017.

[10] Neubrex Co. Fiber optic gyroscope development. `http://www.neubrex.com/htm/applications/gyro-principle.htm`, 2007.

[11] Engineers Edge. Accelerometers review and application. `http://www.engineersedge.com/instrumentation/accelerometers.htm`, 2000-2017.

[12] PC Control Ltd. Accelerometers. `https://www.pc-control.co.uk/accelerometers.htm`, 2008.

[13] Maritn Locker. Inkrementální enkodér. http://robotika.vosrk.cz/guide/sensors/decode/cs. Navštíveno: 30-04-2017, 2009.

[14] amandaghassaei. Arduino timer interrupts. http://www.instructables.com/id/Arduino-Timer-Interrupts/, 2012.

[15] Josef Štětina. Filtrace signálu. http://ottp.fme.vutbr.cz/skripta/vlab/daq/Ka05-04.htm, 2003.

[16] Jiří Melichar. Lineární systémy 1. http://www.kky.zcu.cz/uploads/courses/ls1/LS1_Ucebni_texty_2011.pdf, 2011.

[17] Václav Vrána. Stejnosměrné stroje. http://fei1.vsb.cz/kat420/vyuka/Bakalarske_FS/prednasky/sylab_stejnosmerne%20stroje_bc%20FS.pdf, 2004.

[18] Jiří Melichar. Lineární systémy 2. http://www.kky.zcu.cz/uploads/courses/ls2/LS2_Ucebni_texty_2011.pdf, 2011.

[19] M. Schlegel, P. Balda, and M. Štětina. Robustní pid autotuner – momentová metoda. *Automatizace*, 4:242–246, 2003.

[20] Alessandro De Luca, Giuseppe Oriolo, and Marilena Vendittelli. Control of wheeled mobile robots: An experimental overview. In *Ramsete*, pages 181–226. Springer, 2001.

Příloha A

```
/* Řízení robotu pomocí informací získaných z enkodéru
 *
 */

#include <PinChangeInt.h>
#include <RF24.h>
#include <nRF24L01.h>
#include <SPI.h>
#include <Queue.h>
#include <math.h>

// Motor
#define DIR_L    4    // levy motor vpred(HIGH),vzad(LOW)
#define DIR_R    7    // pravy motor vpred(HIGH),vzad(LOW)
#define PWM_L    5    // levy motor rychlost
#define PWM_R    6    // pravy motor rychlost
#define Pin_10   10

// Enkoder
#define encoderL1    A4     // levy enkoder
#define encoderL2    A2     // levy enkoder (faz.posunuty)
#define encoderP1    A5     // pravy enkoder
#define encoderP2    A3     // pravy enkoder (faz.posunuty)

// Ryhlost smycky
#define LOOPTIME  50

// Zde se zadavaji pozadovane veliciny
//-------------------------------------------------------------

// Zde zadejte pozadovanou rychlost od 0 do 0.48 m/s
float required_Speed_L = 0.0;
float required_Speed_R = 0.0;

// PWM nastavene na motory (0-255)
#define leftWheelSpeed     80
#define rightWheelSpeed    80

// Nastaveni konstant regulatoru
//-------------------------------------------------------------

// Vzorkovaci konstanta
float Ts = 0.05;

// Nastaveni konstant leveho regulatoru
float K_L = 0.77161;
float Ti_L = 0.20427;  //0.1870
float Td_L = 0;
const int umax_L = 486; // Maxilni rychlost, tj. saturace regulatoru

// Vypocet jednotlivych zesileni leveho PID regulatoru
float Kp_L = K_L;
float Ki_L = K_L * (Ts / Ti_L);
float Kd_L = K_L * (Td_L / Ts);
float Kt_L = Ts / Ti_L;

// Konstanty inverznich primek pro vystup regulatoru leveho motoru
const float const_a_01_L = 0.2230;
const float const_b_01_L = 65;

const float const_a_02_L = 0.7671;
const float const_b_02_L = -118.0238;

// Nastaveni konstant praveho regulatoru
float K_R = 0.76961;
float Ti_R = 0.206;  //0.2317
float Td_R = 0;
const int umax_R = 481; // Maxilni rychlost, tj. saturace regulatoru

// Vypocet jednotlivych zesileni praveho PID regulatoru
float Kp_R = K_R;
float Ki_R = K_R * (Ts / Ti_R);
float Kd_R = K_R * (Td_R / Ts);
float Kt_R = Ts / Ti_R;

// Konstanty inverznich primek pro vystup regulatoru praveho motoru
```

```
77 const float const_a_01_R = 0.2165;
   const float const_b_01_R = 65;
79
   const float const_a_02_R = 0.8550;
81 const float const_b_02_R = -156.2357;

83 // Vypocteny zasah regulatoru pro levy a pravy motor
   volatile int u2_L;
85 volatile int u2_R;

87 //------------------------------------------------------------

89 // Prevedeni pozadovane rychlosti z m/s na mm/s a pretypovani z floatu na int
   volatile int req_speed_L = (int)(required_Speed_L * 1000);
91 volatile int req_speed_R = (int)(required_Speed_R * 1000);

93 // Promenna do ktere se inkrementuji tiky z enkoderu (pocitani rychlosti pri konstantnim case)
   volatile long count_L = 0;
95 volatile long count_R = 0;

97 volatile long last_count_L = 0;
   volatile long last_count_R = 0;
99
   // Promenna casu z casovace
101 volatile uint32_t time = 0;

103 // Promenna ktera obsahuje uplynuly cas (ms) mezi dvěma tikama (pocitani rychlosti pro konstantni
        vzdalenost)
    volatile uint32_t elapsed_time_L = 0;
105 volatile uint32_t elapsed_time_R = 0;

107 // Promenna rychlosti vypocitana podle poctu tiku za konstatni cas
    int speed_L_count = 0;
109 int speed_R_count = 0;

111 // Promenna rychlosti vypocitana podle casu uplynulym mezi dvema tiky
    int speed_L_time = 0;
113 int speed_R_time = 0;

115 // Vybrane rychlosti
    volatile int speed_L = 0;
117 volatile int speed_R = 0;

119 // Promenna ujete vzdalenosti
    float dist_L = 0;
121 float dist_R = 0;

123 // Vypocitane PWM z regulatoru
    int PWM_calc_L = 0;
125 int PWM_calc_R = 0;

127 // Konstanta potrebna pro vypocet rychlosti (* 1000 prevod z m/s na mm/s)
    const float const_count = (((2 * PI * 35) / 300) / LOOPTIME) * 1000;
129 const float const_time = (2 * PI * 35);

131 // Konstanta potrebna pro vypocet vzdalenosti (/ 1000 prevod z mm na m)
    const float const_dist = ((2 * PI * 35) / 300) / 1000;
133
    // Konstanta a vystupy IIR filtru
135 const float IIR_a = 0.5;

137 volatile uint32_t IIR_time_L = 0;
    volatile uint32_t IIR_time_R = 0;
139
    long IIR_count_L = 0;
141 long IIR_count_R = 0;

143 int PWM_speed = 0;

145 // Radio nRF24L01(+)
    //-----------------------------------------------------
147
    // Adresace radia
149 byte addresses[][6] = {"1Node","2Node"};

151 // Nastaveni radia jako radio 0 nebo 1 (nezalezi cislo druheho radia se musi lisit)
    bool radioNumber = 0;
153
    // Konstanta kterou se ovlada prijmani dat nebo vysilani (1 vysilani, 0 prijmani)
155 volatile int role = 1;

157 // Datova struktura ktera se posila radiem payload (max. 32 bytu)
    struct dataStruct{
159   //unsigned long _millis;
      //float parametres[6];
161   int velocity_L_time;
      int velocity_L_count;
163   int velocity_R_time;
      int velocity_R_count;
165   int speed_L;
      int speed_R;
167   int required_speed_L;
      int required_speed_R;
169
      int distance_L;
171   int distance_R;
```

```
  int PWM_speed;
}myData;

// nRF24L01(+)
RF24 radio(14,15);

//-------------------------------------------------

// Instance planovace
Queue robotQueue;

void setup() {
  Serial.begin(115200);

  // Inicializace pinu motoru
  pinMode(DIR_L, OUTPUT);
  pinMode(DIR_R, OUTPUT);
  pinMode(PWM_L, OUTPUT);
  pinMode(PWM_R, OUTPUT);
  pinMode(10, OUTPUT);

  // Inicializace pinu enkoderu
  pinMode(encoderL1, INPUT);
  pinMode(encoderL2, INPUT);
  pinMode(encoderP1, INPUT);
  pinMode(encoderP2, INPUT);

  // Pouziti vnitrnich pullup rezistoru
  digitalWrite(encoderL1, HIGH);
  digitalWrite(encoderL2, HIGH);
  digitalWrite(encoderP1, HIGH);
  digitalWrite(encoderP2, HIGH);

  // Prirazecni prerueni k pinum enkoderu
  PCintPort::attachInterrupt(encoderL2, leftEncoder, FALLING);
  PCintPort::attachInterrupt(encoderP1, rightEncoder, FALLING);

  // Iniciazlizace 16-bitového časovače timer1 (frekvence 10kHz)
  noInterrupts();
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1 = 0;

  //OCR1A = 15625;
  OCR1A = 200;
  TCCR1B |= (1 << WGM12);
  //TCCR1B |= (1 << CS10);
  TCCR1B |= (1 << CS11);  //prescaler 8
  //TCCR1B |= (1 << CS12);
  TIMSK1 |= (1 << OCIE1A);  // povoluje CTC mod
  interrupts();

  // Sestaveni planovace
  robotQueue.scheduleFunction(calculateVelocityLeft, "test1", 0, LOOPTIME);
  robotQueue.scheduleFunction(calculateVelocityRight, "test2", 0, LOOPTIME);

  //robotQueue.scheduleFunction(PWM_rise, "test3", 0, 4000);
  robotQueue.scheduleFunction(goLine, "test10", 0, LOOPTIME);
  //robotQueue.scheduleFunction(goSquare, "test12", 0, LOOPTIME);
  //robotQueue.scheduleFunction(speedUpDown, "test13", 0, LOOPTIME);

  robotQueue.scheduleFunction(pidRegulationLeft, "test4", 0, LOOPTIME);
  robotQueue.scheduleFunction(pidRegulationRight, "test5", 0, LOOPTIME);

  robotQueue.scheduleFunction(communication, "test6", 0, LOOPTIME);


  SPI.begin();

  radio.begin();

  radio.setPALevel(RF24_PA_LOW);

  if(radioNumber){
    radio.openWritingPipe(addresses[1]);
    radio.openReadingPipe(1,addresses[0]);
  }else{
    radio.openWritingPipe(addresses[0]);
    radio.openReadingPipe(1,addresses[1]);
  }

}

// Funkce pro pricitani tiku z leveho enkoderu a pro pocitani casu mezi jednotlivymi tiky
void leftEncoder() {
  static uint32_t last_time_L = time;
  static uint32_t last_IIR_time_L = 0;

  if (digitalRead(encoderL2) == LOW)
  {
    count_L++;
  }
  else
  {
    count_L--;
  }
```

```
269   elapsed_time_L = (time - last_time_L);
      last_time_L = time;
271
      IIR_time_L = (uint32_t)((IIR_a * elapsed_time_L * 100) + ((1 - IIR_a) * last_IIR_time_L));
273   last_IIR_time_L = IIR_time_L;

275   //Serial.print ("count_L:  "); Serial.println(count_L);
      //Serial.print ("IIR_time_L:  "); Serial.println(IIR_time_L);
277
}
279
// Funkce pro pricitani tiku z praveho enkoderu a pro pocitani casu mezi jednotlivymi tiky
281 void rightEncoder() {
      static uint32_t last_time_R = time;
283   static uint32_t last_IIR_time_R = 0;

285   if (digitalRead(encoderP1) == LOW)
      {
287     count_R++;
      }
289   else
      {
291     count_R--;
      }
293
      elapsed_time_R = (time - last_time_R);
295   last_time_R = time;

297   IIR_time_R = (uint32_t)((IIR_a * elapsed_time_R * 100) + ((1 - IIR_a) * last_IIR_time_R));
      last_IIR_time_R = IIR_time_R;
299
}
301
// Funkce pro vypocet rychlosti na levem kole v mm/s
303 int calculateVelocityLeft(unsigned long now) {
      static long count_prev_L;
305   static long last_IIR_count_L;
      int count_diff_L;
307
      count_diff_L = count_L - count_prev_L;
309
      // Filtrace poctu tiku
311   IIR_count_L = (IIR_a * count_diff_L * 100) + ((1 - IIR_a) * last_IIR_count_L);
      last_IIR_count_L = IIR_count_L;
313
      // Vypocet ujete vzdalenosti
315   dist_L = dist_L + (IIR_count_L * const_dist / 100);   // Vzdalenost v m

317   // Vypocet rychlosti podle poctu tiku za konstatni cas
      speed_L_count = (int)((IIR_count_L * const_count) / 100);  // Rychlost v m/s
319   count_prev_L = count_L;

321   if (IIR_time_L == 0)
      {
323     speed_L_time = 0;
      }
325   else
      {
327     // Vypocet rychlosti podle castu mezi dvema tiky ( / ve IIR filtru optimalizace *100, proto
        zde misto *10000 je *1000000)
        speed_L_time = (int)((const_time * 1000000) / (IIR_time_L * 300));
329   }
      IIR_time_L = 0;
331
      speed_L = (int)(speed_L_time + speed_L_count) / 2;
333 }

335 // Funkce pro vypocet rychlosti na pravem kole v mm/s
    int calculateVelocityRight(unsigned long now) {
337   static long count_prev_R;
      static long last_IIR_count_R;
339   int count_diff_R;

341   count_diff_R = count_R - count_prev_R;

343   // Filtrace poctu tiku
      IIR_count_R = (IIR_a * count_diff_R * 100) + ((1 - IIR_a) * last_IIR_count_R);
345   last_IIR_count_R = IIR_count_R;

347   // Vypocet ujete vzdalenosti
      dist_R = dist_R + (IIR_count_R * const_dist / 100);   // Vzdalenost v m
349
      // Vypocet rychlosti podle poctu tiku za konstatni cas
351   speed_R_count = (int)((IIR_count_R * const_count) / 100);  // Rychlost v m/s
      count_prev_R = count_R;
353
      if (IIR_time_R == 0)
355   {
        speed_R_time = 0;
357   }
      else
359   {
        // Vypocet rychlosti podle castu mezi dvema tiky
361     speed_R_time = (int)((const_time * 1000000) / (IIR_time_R * 300));
      }
```

```
363   IIR_time_R = 0;

365   speed_R = (int)(speed_R_time + speed_R_count) / 2;
}
367
// Funkce regulátoru PID pro levy motor
369 int pidRegulationLeft(unsigned long now) {

371   int s_L = 0;
   int error_L, u_L, u1_L = 0;
373   float up_L, ui_L, ud_L = 0;

375   static int last_error_L, correction_L;
   static float last_ui_L;
377
  s_L = sgn(req_speed_L);
379
   error_L = abs(req_speed_L) - (speed_L);
381
   up_L = Kp_L * error_L;
383   ui_L = Ki_L * error_L - Kt_L * correction_L + last_ui_L;
   ud_L = Kd_L * (error_L - last_error_L);
385   last_ui_L = ui_L;

387   u_L = (int)(up_L + ui_L + ud_L);

389   // Vypocet korekce integracni slozky a saturace
   if (u_L < 0)
391   {
     u1_L = 0;
393   }
   else if (u_L > umax_L)
395   {
     correction_L = u_L - umax_L;
397     u1_L = umax_L;
   }
399   else
   {
401     correction_L = 0;
     u1_L = u_L;
403   }
   last_error_L = error_L;
405
   // Inverzni funkce pro vystup regulatoru
407   if (u1_L == 0)
   {
409     u2_L = 0;
   }
411   else if (u1_L > 0 && u1_L < 336)
   {
413     u2_L = (int)(u1_L * const_a_01_L + const_b_01_L);
   }
415   else if (u1_L >= 336 && u1_L <= umax_L)
   {
417     u2_L = (int)(u1_L * const_a_02_L + const_b_02_L);
   }
419   else if (u1_L > umax_L)
   {
421     u2_L = 255;
   }
423 }

425 // Funkce regulátoru PID pro pravy motor
int pidRegulationRight(unsigned long now) {
427
   int s_R = 0;
429   int error_R, u_R, u1_R = 0;
   float up_R, ui_R, ud_R = 0;
431   static int last_error_R, correction_R;
   static float last_ui_R;
433
  s_R = sgn(req_speed_R);
435
   error_R = abs(req_speed_R) - (speed_R);
437
   up_R = Kp_R * error_R;
439   ui_R = Ki_R * error_R - Kt_R * correction_R + last_ui_R;
   ud_R = Kd_R * (error_R - last_error_R);
441   last_ui_R = ui_R;

443   u_R = (int)(up_R + ui_R + ud_R);

445   // Vypocet korekce integracni slozky a saturace
   if (u_R < 0)
447   {
     u1_R = 0;
449   }
   else if (u_R > umax_R)
451   {
     correction_R = u_R - umax_R;
453     u1_R = umax_R;
   }
455   else
   {
457     correction_R = 0;
     u1_R = u_R;
```

```
459   }
      last_error_R = error_R;
461
      // Inverzni funkce pro vystup regulatoru
463   if (u1_R == 0)
      {
465     u2_R = 0;
      }
467   else if (u1_R > 0 && u1_R < 336)
      {
469     u2_R = (int)(u1_R * const_a_01_R + const_b_01_R);
      }
471   else if (u1_R >= 336 && u1_R <= umax_R)
      {
473     u2_R = (int)(u1_R * const_a_02_R + const_b_02_R);
      }
475   else if (u1_R > umax_R)
      {
477     u2_R = 255;
      }
479
  }
481
  // Radio nRF24L01(+)
483 int communication(unsigned long now) {

485 if (role == 1)  {

487     radio.stopListening();

489     //Serial.println(F("Now sending"));

491     myData.velocity_L_time = speed_L_time;
        myData.velocity_L_count = speed_L_count;
493     myData.velocity_R_time = speed_R_time;
        myData.velocity_R_count = speed_R_count;
495     myData.speed_L = speed_L;
        myData.speed_R = speed_R;
497     myData.distance_L = (int)(dist_L * 1000);
        myData.distance_R = (int)(dist_R * 1000);
499     myData.required_speed_L = req_speed_L;
        myData.required_speed_R = req_speed_R;
501     myData.PWM_speed = PWM_speed;

503     if (!radio.write( &myData, sizeof(myData) )){
          //Serial.println(F("failed"));
505     }

507     radio.startListening();

509     unsigned long started_waiting_at = millis();

511     boolean timeout = false;
        if ( timeout ){
513         Serial.println(F("Failed, response timed out."));
        }else{
515         radio.read( &myData, sizeof(myData) );
        }
517   }
  }
519
  void movementControl() {
521   digitalWrite(DIR_L, HIGH);
      digitalWrite(DIR_R, HIGH);
523
      analogWrite(PWM_L, u2_L);
525   analogWrite(PWM_R, u2_R);
  }
527
  void forward() {
529   digitalWrite(DIR_L, HIGH);
      digitalWrite(DIR_R, HIGH);
531
      analogWrite(PWM_L, leftWheelSpeed);
533   analogWrite(PWM_R, rightWheelSpeed);
  }
535
  void forward_rising() {
537   digitalWrite(DIR_L, HIGH);
      digitalWrite(DIR_R, HIGH);
539
      analogWrite(PWM_L, PWM_speed);
541   analogWrite(PWM_R, PWM_speed);
  }
543
  int PWM_rise(unsigned long now) {
545   if (millis() < 1000)
      {
547     PWM_speed = 0;
      }
549   else
      {
551     PWM_speed = constrain(PWM_speed + 51, 0, 255);
      }
  }
553
```

```
555 ISR(TIMER1_COMPA_vect)          // timer compare interrupt service routine
{
557   time++;
}
559
int goLine(unsigned long now) {
561
  float dist_mean;
563   static float start_dist;
  static long start_time = 0;
565
  if (millis() >= 4000 && millis() <= 16830 )
567   {
    req_speed_L = 350;
569     req_speed_R = 400;
  }
571   else
  {
573     req_speed_L = 0;
    req_speed_R = 0;
575   }
}
577
int goSquare(unsigned long now) {
579
  float dist_mean, dist_diff;
581   static float start_dist = 0;
  static long start_time = 0;
583
  dist_mean = (dist_L + dist_R) / 2;
585
  dist_diff = dist_mean - start_dist;
587
  if (dist_diff  > 0.95)
589   {
    if ((millis() - start_time) <= 2827)
591     {
      req_speed_L = 0;
593       req_speed_R = 300;
    }
595     else
    {
597       dist_L = 0;
      dist_R = 0;
599       start_dist = 0;
    }
601   }
  else
603   {
    req_speed_L = 300;
605     req_speed_L = 300;
    start_time = millis();
607   }
}
609
int speedUpDown(unsigned long now) {
611
  static long start_time = 0;
613
  if ((millis() - start_time) > 4000)
615   {
    if(req_speed_L == 400)
617     {
      req_speed_L = 200;
619       req_speed_R = 200;
      start_time = millis();
621     }
    else if(req_speed_L == 200)
623     {
      req_speed_L = 400;
625       req_speed_R = 400;
      start_time = millis();
627     }
    else
629     {
      req_speed_L = 200;
631       req_speed_R = 200;
    }
633   }
}
635
void loop() {
637   robotQueue.Run(millis());
  movementControl();
639 }
```