VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

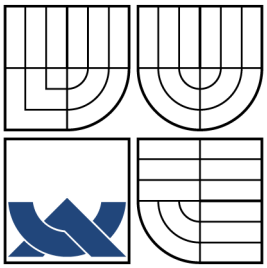
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ŘÍZENÍ VŠESMĚROVÉHO PODVOZKU

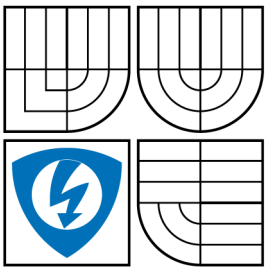
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ADAM HRAZDIRA
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ŘÍZENÍ VŠESMĚROVÉHO PODVOZKU
OMNIDIRECTIONAL CHASSIS CONTROL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ADAM HRAZDIRA
AUTHOR

VEDOUCÍ PRÁCE — ING. JAROSLAV ŠEMBERA
SUPERVISOR

BRNO 2009

ABSTRAKT

Cílem práce byl návrh a realizace softwaru pro řízení modelu synchronního všesměrového podvozku. Nejdříve byl vytvořen program pro samotný podvozek. Program byl napsán v jazyce C++ a má za úkol komunikovat s klientským programem pomocí protokolu TCP/IP a vykonávat přijaté příkazy. Dalším krokem bylo naprogramování grafické klientské aplikace v jazyce C#. Ta slouží k ovládání podvozku typu bicykl a diferenciálního typu. Při tomto ovládání je navíc počítána odometrie. Závěrečným bodem práce bylo využití prostředí Matlab Simulink k simulování chování všesměrového podvozku a jeho řízení.

KLÍČOVÁ SLOVA

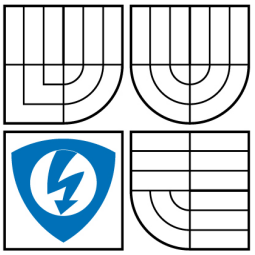
Všesměrový synchronní podvozek, mobilní robotika, Fox Board

ABSTRACT

The aim of this work is to design and implement software for omnidirectional chassis control. Firstly, the chassis control program was created. It is written in C++ programming language and its objective is communicating with the client program via TCP/IP protocol and accomplishing the received commands. The next step was to create a graphic user application in C# programming language. The application can operate the chassis of differential and bicycle type and can compute the oddometry of the chassis. In the end an interactive simulation in Matlab Simulink environment, which can simulate the omnidirectional chassis behaviour and control it as well, was made.

KEYWORDS

Omnidirectional chassis, mobile robotics, Fox Board



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student:	Adam Hrazdira	***ID:***	73024
Ročník:	3	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

Řízení všesměrového podvozku

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte a realizujte software pro model synchronního všesměrového podvozku. Tento model bude sloužit k testování a ověření teoretických znalostí týkajících se tohoto podvozku s velkými manévrovacími schopnostmi. Vytvořte uživatelské rozhraní pro PC a najděte vhodný způsob komunikace s mobilním podvozkem. Vytvořte propojení pro snadné ovládání mobilního podvozku z prostředí Matlab.

DOPORUČENÁ LITERATURA:

Příručka Matlab
Příručka C++
Datasheet motoru AI-1001

Termín zadání:	9.2.2009	***Termín odevzdání:***	1.6.2009

Vedoucí práce: Ing. Jaroslav Šembera

prof. Ing. Pavel Jura, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práve třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

BIBLIOGRAFICKÁ CITACE

HRAZDIRA, A. *Řízení všesměrového podvozku.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 48 s. Vedoucí bakalářské práce Ing. Jaroslav Šembera.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Řízení všesměrového podvozku“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

OBSAH

Úvod **10**

1 Reálný model podvozku **11**
1.1 Konstrukce . . . 11
1.2 Hnací soustava . . . 12
1.3 Řídící počítač . . . 12
1.4 Napájení . . . 13
1.5 Ultrazvukové senzory . . . 14

2 Komunikace **15**
2.1 Komunikace mezi řídícím počítačem podvozku a motory podvozku . . . 15
2.2 Komunikace mezi uživatelským počítačem a řídícím počítačem podvozku 16

3 Program pro podvozek **20**
3.1 Vlákna . . . 20
3.1.1 Využití knihovny ZThread . . . 22
3.2 Struktura programu . . . 23
3.2.1 Správa a sdílení dat . . . 23
3.2.2 Přijímání dat od klienta . . . 24
3.2.3 Odesílání dat klientovi . . . 26
3.2.4 Komunikace s motory a jejich řízení . . . 26
3.2.5 Zjišťování dat z ultrazvukových senzorů . . . 27
3.2.6 Zjišťování hodnoty napájení . . . 27

4 Uživatelský program **28**
4.1 Připojení k podvozku a komunikace s ním . . . 28
4.2 Monitorování stavu motorů . . . 30
4.3 Ovládání a nastavování jednotlivých motorů . . . 30
4.4 Vizualizace natočení hnacích soustav . . . 30
4.5 Ovládání podvozku typu bicykl a diferenciální podvozek . . . 31
4.6 Odometrie . . . 32

5 Využití prostředí Matlab **36**
5.1 Výsledky simulací . . . 37

6 Závěr **39**

Literatura **41**

Seznam příloh **42**

A Programové diagramy vláken **43**

A.1 Diagram hlavního vlákna 43
A.2 Diagram vlákna pro přijímaní dat 44
A.3 Diagram vlákna pro odesílání dat 45
A.4 Diagram vlákna pro ovládání motorů 46
A.5 Diagram vlákna pro zjišťování údajů z ultrazvukových senzorů 47
A.6 Diagram vlákna pro zjišťování hodnoty napájení 48

SEZNAM OBRÁZKŮ

1.1 Reálný model podvozku 11
2.1 Ukázka a) odchozího a b) příchozího paketu 17
3.1 Struktura programu 23
4.1 Ukázka uživatelského rozhraní 29
4.2 Schéma konfigurace hnacích soustav podvozku typu bicykl 31
4.3 Schéma konfigurace hnacích soustav podvozku diferenciálního typu . . . 32
4.4 Graf vývoje pozice podvozku 34
4.5 Graf závislosti uražené vzdálenosti podvozku na čase (typ bicykl) 34
4.6 Schéma k výpočtu odometrie podvozku typu bicykl 35
4.7 Schéma k výpočtu odometrie podvozku diferenciálního typu 35
5.1 Simulinkový blok pro komunikaci s podvozkem 37
5.2 Přechod translačního pohybu do rotačního 38
5.3 Pohyb po kružnici bez natočení těla robota 38
A.1 Diagram hlavního vlákna 43
A.2 Diagram vlákna pro přijímání dat 44
A.3 Diagram vlákna pro odesílání dat 45
A.4 Diagram vlákna pro ovládání motorů 46
A.5 Diagram vlákna pro zjišťování údajů z ultrazvukových senzorů 47
A.6 Diagram vlákna pro zjišťování hodnoty napájení 48

SEZNAM TABULEK

2.1 Typy komunikačních segmentů 18
2.2 Typy chybových hlášení 19

ÚVOD

Cílem této práce je vytvoření reálného funkčního modelu speciálního všesměrového podvozku. V dnešní době je využití robotické techniky velmi častou záležitostí. Roboti usnadňují práci člověku a často ho nahrazují v situacích, kdy člověk sám nemůže zvládat velké množství úkonů najednou, nedosahuje přesnosti robota, situace je pro člověka příliš nebezpečná atd.

Všesměrový podvozek se vyznačuje vysokými manévrovacími schopnostmi. Člověk sám by nebyl schopen řídit všechny součásti podvozku, proto je nutné vytvořit řídící systém, který umožní podvozku být zcela autonomním, či být pouze z části řízen člověkem.

Využití tohoto podvozku je možné například jako tzv. mobilní manipulátor, kde se dá velmi dobře využít možnosti pohybu libovolným směrem, aniž by bylo nutné předem natočit tělo podvozku. Uplatnění může být např. v divadle pro realizaci pohyblivých kulis či v nejrůznějších skladech pro transport materiálu.

Oproti klasickým všesměrovým podvozkům má naše řešení nespornou výhodu v tom, že při změně konfigurace těla robotu se v ideálním případě nevyskytuje tření a prokluz kol. To nám umožňuje využít při navigaci a určování polohy robotu odometrii. Další znatelnou výhodou řešení našich hnacích soustav je fakt, že robot je schopen překonávat větší překážky a nerovnosti terénu, než robot s klasickými všesměrovými koly.

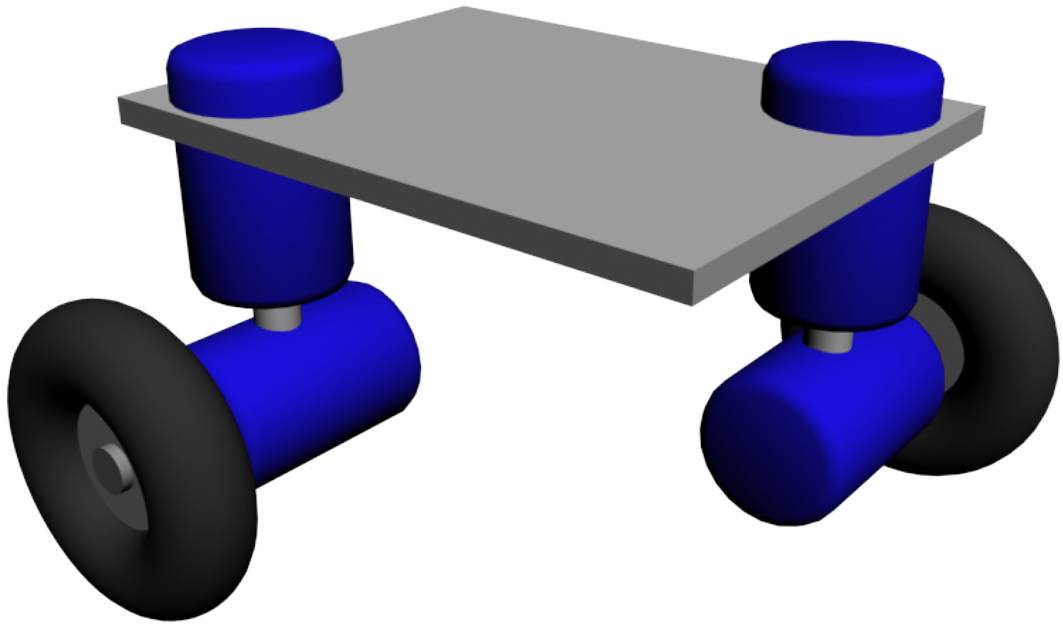
V této práci se budeme zabývat nejdříve konstrukcí samotného reálného modelu podvozku a jeho součástmi (řídicím počítačem a hnacími soustavami). Dále popíšeme způsoby a vlastnosti komunikace s podvozkem. Poté určíme funkčnost a strukturu programu zprostředkovávajícího komunikační rozhraní podvozku a vysvětlíme vše potřebné pro pochopení jeho funkce. V následujícím případě se zaměříme na vytvořené uživatelské prostředí, kde opět objasníme, jak bylo navrženo a realizováno. Na závěr představíme možnost využití prostředí Matlab při simulaci chování všesměrového podvozku a komunikaci s ním.

1 REÁLNÝ MODEL PODVOZKU

1.1 Konstrukce

Náš reálný model podvozku je zobrazen na obrázku 1.1. Skládá se z desky, na které jsou připevněny dva motory s koly, dále dva motory pro natáčení hnacích soustav a pomocné stabilizační kluzáky. Motory jsou umístěny na diagonále v protějších rozích desky. Stejně tak stabilizační kluzáky ve vedlejších rozích (na obrázku nejsou znázorněny). Kola jsou hnaná a jsou připevněna na otáčivé soustavě vůči tělu podvozku.

Osa rotace hnací soustavy neprochází přímo kolem, je tedy možné při natáčení kol dosáhnout menšího tření a smýkání, protože při natáčení kolo opisuje kružnici a jede po ní. To je jedna z výhod oproti jiným všesměrovým podvozkům. Kombinací natočení kol a rychlostí jejich pojezdu ovlivňujeme dráhu pojezdu podvozku.



Obr. 1.1: Reálný model podvozku

Pro lepší pohyblivost a dodatečné snížení tření a smýkání kluzáků by bylo vhodné při dalším vývoji tyto kluzáky nahradit řízenými natáčecími koly.

Náš podvozek se ale od ostatních všesměrových podvozků liší tím, že není schopen v libovolném okamžiku změnit rychlost libovolným směrem. Při této činnosti se pouze změní natočení pohybových částí a samotný podvozek zůstává v původní poloze, narozdíl od jiných podvozků, které využívají natočení těla. Je možno prohlásit, že náš speciální všesměrový podvozek patří do jakési kvazi kategorie všesměrových podvozků.

1.2 Hnací soustava

Jako hnací soustavu jsme zvolili modul zvaný AI MOTOR-701. Jedná se o modul využívaný k pohybu široké škály robotů. Tento modul shrnuje motor, kontrolní obvody a všechny ostatní potřebné součásti do jedné součástky. Je tedy snadné tyto moduly spojovat mezi sebou a vytvářet tak složitější soustavy.

Dále je možné zapojit až 31 modulů pomocí jednoho kabelu sériově a zjednodušit tak zapojení. Směrování příkazů pro jednotlivé moduly se pak provádí pomocí jejich identifikačního čísla. Moduly jsou řízeny pomocí sériového rozhraní RS-232 s možností využití několika rychlostí přenosu. Pro monitorování a kontrolu stavu motorů je možné zjistit proud procházející motorem a aktuální pozici motoru (úhel natočení).

Modul má několik základních operačních módů. Například otáčení o 360 stupňů, přesné nastavení otáčení v rozsahu 0 až 332 stupňů nebo také tzv. „act down“ mód, kdy můžeme měnit pozici osy ručně.

Výraznou nevýhodou při konfiguraci natočení hnacích soustav je omezení otáčení kvůli napájecím drátům motorů. Není tedy možné, aby hnací soustavy nepřetržitě měnily svoje natočení ve všetším rozsahu než 0 až 360 stupňů. Tento fakt značně komplikuje řízení podvozku. Naskýtá se však hned několik řešení. Nejjednodušší variantou je při dosažení jedné z krajních poloh podvozek zastavit, danou hnací soustavu otočit zpět do druhé krajní pozice a pokračovat. Nevýhoda tohoto řešení je především ztráta plynulosti pohybu. Druhou variantou je dynamické přepočítávání dopředné rychlosti kola a zároveň natáčení hnací soustavy tak, aby se omezení natočení neprojevilo. V tomto případě bude pohyb již plynulý. Řešení je však zatíženo vyšší výpočetní náročností. Poslední řešení, které se zdá jako nejideálnější, je zajistit napájení motorů jiným způsobem tak, aby nezáviselo na jejich natočení. Například pomocí kartáčového sběrače. Toto řešení však bude patrně finančně nejnáročnější.

1.3 Řídící počítač

Pro řízení podvozku používáme vestavný modul FOX Board LX832. Jeho rozměry jsou pouze 66 na 72 mm. Modul je osazen 32 bitovým procesorem Axis ETRAX 100LX s

frekvencí 100 MHz s redukovanou instrukční sadou (tzv. RISC z angl. Reduced Instruction Set Code). Dále je modul spravován plnohodnotným operačním systémem Linux s jádrem 2.6.

Pro komunikaci FOX Board poskytuje RJ45 10/100 Mbit Ethernet port. Je tedy možné ho jednoduše připojit do lokálních sítí LAN standardu Ethernet. Pro komunikaci pak využíváme protokoly rodiny TCP/IP.

Dále je možné k FOX Boardu připojit zařízení pomocí dvou sériových rozhraní USB 1.1. Je tedy relativně snadné rozšířit modul například o bezdrátovou komunikaci pomocí standardu Wi-fi.

V neposlední řadě máme k dispozici sériové rozhraní RS-232, které jde mimo jiné využít také jako konzola pro správu modulu.

Modul využívá 32 MB operační paměti RAM a pro vývoj je k dispozici 8 MB FLASH paměti. To je dostatečné množství pro valnou většinu programů. Ty je možné psát v několika jazycích. Pro nás nejdůležitější jsou však C a C++.

Mezi přední výhody modulu FOX Board LX832 patří dobrá cenová dostupnost a minimální náklady na vývoj aplikací. Vývojové nástroje (tzv. SDK z angl. Software Developement Kit) jsou volně dostupné ke stažení přímo na stránkách výrobce. Jelikož modul využívá plnohodnotného operačního systému Linux s jádrem 2.6, je možné bez problémů vyvíjet aplikaci pro modul ve stolním počítači. Modul je navíc možno aktualizovat a doplňovat tak potřebné chybějící komponenty přes sériové rozhraní RS-232 nebo Ethernet.

FOX Board má velmi širokou škálu využití. Díky jeho malým rozměrům se dá snadno zakomponovat do již existujících systémů. Může však samostatně zastávat celou řadu řídících či monitorovacích funkcí, či může být použit ke sběru dat. Jistě by se našlo mnoho aplikačních situací.

1.4 Napájení

K napájení mikropočítače podvozku a jeho hnacích soustav byl zvolen akumulátor typu Li-Pol s napětím 7,4 V a kapacitou 2250 mAh. Spotřeba proudu je závislá na celé řadě faktorů, jako počet aktivních motorů, nerovnosti překonávané podvozkem atd. Pozorování jsme však zjistili, že se spotřeba pohybuje okolo 800 mA. Výdrž akumulátoru na jedno nabití je tedy okolo tří hodin. Pro správnou funkci byl realizován stabilizátor na napětí +5 V typu LM2576T. Ten funguje zároveň i jako podpěťová ochrana, která odpojí napájení při napětí akumulátoru menším jak 6,4 V a tím zabraňuje poškození akumulátoru. Aktuální napětí akumulátoru se zobrazuje na připojeném LCD.

1.5 Ultrazvukové senzory

K mikropočítači jsou též připojeny ultrazvukové senzory typu SRF08. Tyto detektory komunikují přes sběrnici I^2C a je tedy možné z řídícího programu vyčítat vzdálenost případných překážek.

2 KOMUNIKACE

Při řízení podvozku potřebujeme komunikovat dvěmi cestami. Jednak mezi uživatelským počítačem, který slouží k monitorování podvozku a zadávání příkazů, a samotným podvozkem. A dále mezi řídícím počítačem podvozku a motory podvozku. Tyto komunikace však využívají jiných přenosových prostředků.

2.1 Komunikace mezi řídícím počítačem podvozku a motory podvozku

Za předpokladu, že pro podvozek využíváme motory AI 701, je nutné s nimi komunikovat přes sériové rozhraní RS-232 pomocí již předem definovaného komunikačního protokolu, popsaným v uživatelské příručce motoru. Tento protokol zajišťuje určení začátku posílaných dat pomocí hlavičky a kontrolu správnosti přijatých dat pomocí kontrolního součtu. Při realizaci komunikace musíme tento protokol brát v úvahu a implementovat ho.

Výchozí nastavení přenosové rychlosti motoru je 57600 baudů za sekundu. Dále musíme uvažovat správné nastavení sériového rozhraní RS-232 a sice počet bitů 8, 1 stop bit a ignorování parity. Motor je však schopen komunikovat v rychlostech od 2400 do 460800 baudů za sekundu. Výchozí hodnota rychlosti se však jeví jako optimální, protože při nižších rychlostech se velmi zvyšuje doba zpoždění při komunikaci a při vysokých rychlostech naopak může docházet k chybám při přenosu.

Při přijmu dat z motoru po sériové lince musíme počítat s jistým zpožděním příchodu dat. Toto zpoždění je závislé na přenosové rychlosti. V našem případě při přenosové rychlosti 57600 baudů za sekundu je maximální zpoždění operačního příkazu 1701 mikrosekund a maximální zpoždění nastavovacího příkazu je 121492 mikrosekund. Tato zpoždění je nutné respektovat především při softwarové implementaci přijmu dat tak, abychom měli jistotu, že jsme přijali veškerá data. Kompletní tabulku maximálních zpoždění při různých rychlostech je možné najít v příručce motoru.

Při realizaci komunikace vznikaly komplikace při přijmu dat, kdy data dorážela s nepřípustným zpožděním v řádech stovek milisekund. Toto chování bylo způsobeno tím, že procesor ETRAX 100LX neobsahuje hardwarové řešení automatického vyprazdňování FIFO zásobníku čipu UART po určitém uplynulém čase, pokud zásobník není plný. Existují však softwarové metody, kterými se tato vlastnost dá zastoupit. Řešením problému tedy bylo překompilování jádra operačního systému mikropočítače podvozku s experimentálním parametrem CONFIG_ETRAX_SERIAL_FLUSH_DMA_FAST.

2.2 Komunikace mezi uživatelským počítačem a řídícím počítačem podvozku

Pro komunikaci využíváme internetového protokolu TCP/IP. Jedná se o tzv. spojovou službu se spolehlivým doručením dat. To znamená, že před tím, než se odešlou jakákoliv data, bude navázáno a ověřeno spojení. Pokud se tak nestane, data se neodešlou. Navíc protokol zajišťuje doručení všech dat ve správném pořadí. Díky tomu se nemusíme bát, že by data přišla neúplná, nebo poškozená a při návrhu komunikace s tím můžeme počítat.

FOX Board LX832 obsahuje integrovaný Ethernet port (10/100 Mb/s) RJ45 a využívá operačního systému Linux, který nativně podporuje protokol TCP/IP. Díky tomu se můžeme spojit pomocí kabelu s uživatelským počítačem (pro uživatelské počítače je dnes protokol TCP/IP a Ethernet port celosvětovým standardem).

Další možnou variantou je spojení pomocí standardu Wi-fi. Jedná se o bezdrátové spojení, které opět využívá protokol TCP/IP. Součástí desky FOX Board LX832 však není integrovaný Wi-fi modul. Je proto nutné tento modul přidat jako externí zařízení připojením do portu USB 1.1, který se zde již vyskytuje. Komunikace bude probíhat na síťové a transportní vrstvě naprosto stejně, jako v předchozím případě, protože se stále jedná o protokoly TCP/IP. Rozdíl bude pouze na fyzické vrstvě. To nás však opět nemusí z hlediska realizace komunikace zajímat, protože se o vše postará operační systém. My musíme navrhnout pouze nejnižší aplikační vrstvu.

Pro testování časové odezvy při komunikaci byl vytvořen v klientském programu speciální paket, který je odeslán podvozku. V tomto okamžiku si zapamatuje klientský program aktuální čas. Podvozek rozbalí a rozkóduje paket, zjistí, že se jedná o typ PING a neprodleně odešle stejný paket klientovi, jako odpověď. Jakmile klient přijme tento paket s odpovědí, porovná čas odeslání paketu s časem jeho přijmu a z rozdílu zjistí časovou odezvu. Výhoda tohoto řešení spočívá v tom, že zjišťujeme časovou prodlevu v komunikaci mezi nejnižšími aplikačními vrstvami OSI/ISO modelu. Toto zpoždění pak zároveň vypovídá i o zatíženosti programu podvozku.

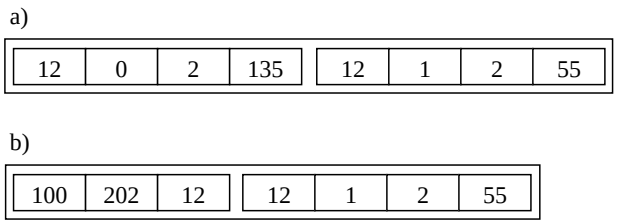
Při testování časové odezvy při připojení přes kabel i bezdrátově pomocí Wi-fi jsme dospěli k podobným závěrům. Odezva byla naměřena v intervalu od 0 milisekund do 15,625 milisekund.

Námi vytvořený komunikační protokol spočívá v posílání paketů, které obsahují segmenty z bajtů. Každý segment představuje jeden příkaz s jeho potřebnými daty. Segmenty se rozlišují podle příznakového bajtu, který je vždy na začátku segmentu. Pomocí těchto příznaků přesně určíme typ, obsah a tím i délku daného segmentu. Paket jako takový může mít teoreticky neomezenou velikost. Jinými slovy se může skládat z libovolného počtu segmentů. V praxi však toto závisí na mnoha faktorech, jako je např. propustnost linky, požadovaná odezva komunikace, omezení paměti atd. Jednotlivé segmenty

pak mají proměnnou délku v intervalu od dvou do sedmi bajtů v závislosti na typu. Seznam typů segmentů se kterými pracuje klientský program pro ovládání podvozku (řídí se jimi pochopitelně i program v mikropočítači podvozku) je možné nalézt v tabulce 2.1.

Při zpracování příkazů od klienta v mikropočítači podvozku mohou nastat nejrůznější chyby a podvozek by měl o těchto chybách klienta informovat. Ve většině případů se jedná o chyby způsobené při komunikaci s motory. Vzniknou např. při nefunkční sériové lince nebo při špatných hodnotách přijatých z motoru. Segment reprezentující chybové hlášení se skládá z příznakového bajtu, dále z čísla chyby a kontextu chyby. Kontextem chyby se míní situace, při které daná chyba vznikla. Kompletní seznam chyb je shrnut v tabulce 2.2.

Na obrázku 2.1 je ukázka odchozího paketu s příkazy od klienta a příchozího paketu s odpovědí z podvozku. V prvním případě se jedná o dva příkazy. Nejdříve příkaz k natočení motoru 0, rychlostí 2 na pozici 135 a dále příkaz k natočení motoru 1, rychlostí 2 na pozici 55. Co se týče příchozích dat s odpovědí, jedná se opět o dva segmenty. První hlásí chybu číslo 202 a tedy podle tabulky 2.2 jde o chybu při přijmu dat z motoru pomocí sériové linky. Kontext chyby je příznakový bajt s hodnotou 12. Podle tabulky 2.1 se tedy jedná o chybu vzniklou při nastavování pozice. Následující segment je odpověď po úspěšně provedeném natočení motoru 1, rychlostí 2 na pozici 55.



Obr. 2.1: Ukázka a) odchozího a b) příchozího paketu

Příznak	Typ	Popis	Odchozí formát	Příchozí formát
1	SET_ID	Nastavení ID motoru	Původní ID, nové ID	Nové ID
2	SET_BAUD	Nastavení přenosové rychlosti motoru	ID, přenosová rychlost	ID, přenosová rychlost
3	SET_GAIN	Nastavení zesílení motoru.	ID, proporcionální zesílení, diferenciální zesílení	ID, proporcionální zesílení, diferenciální zesílení
4	SET_BOUND	Nastavení omezeni otáčení motoru	ID, spodní hranice, dolní hranice	ID, spodní hranice, dolní hranice
5	SET_RESOL	Nastavení rozlišení motoru	ID, rozlišení	ID, rozlišení
6	SET_OVERCT	Nastavení proudové ochrany motoru	ID, hodnota proudu	ID, hodnota proudu
7	READ_POS	Zjištění polohy natočení motoru	ID	ID, poloha
8	READ_GAIN	Zjištění zesílení motoru	ID	ID, proporcionální zesílení, diferenciální zesílení
9	READ_RESOL	Zjištění rozlišení motoru	ID	ID, rozlišení
10	READ_BOUND	Zjištění omezení otáčení motoru	ID	ID, spodní hranice, dolní hranice
11	READ_OVERCT	Zjištění proudové ochrany motoru	ID	ID, hodnota proudu
12	POS_SEND	Nastavení polohy motoru	ID, rychlost, poloha	ID, rychlost, koncová poloha, aktuální poloha
13	SYNC_POS_SEND	Synchronní odeslání polohy více motorům	ID posledního motoru, pole pozic o velikosti posledního ID	poslední ID
14	ROT_360	Rotace motoru	ID, rychlost, směr	ID, počet otáček od zapnutí, aktuální pozice
15	ACT_DOWN	Příkaz pro odpojení motoru tak, že je možné ho natáčet externí silou	ID	ID
16	POWER_DOWN	Příkaz pro odpojení všech motorů	doplňkový bajt	doplňkový bajt
17	DIFF_CHASSIS	Paket pro data určená k odometrii diferenciálního podvozku	doplňkový bajt	počet otáček pravé kolo, pozice pravé kolo, směr otáčení pravé kolo, počet otáček levé kolo, pozice levé kolo, směr otáčení levé kolo
18	BIC_CHASSIS	Paket pro data určená k odometrii podvozku typu bicykl	doplňkový bajt	natočení předního kola, počet otáček kola, pozice kola
98	ULTRASOUND	Paket s daty z ultrazvukových senzorů	—	ID ultrazvuku, dolní bajt vzdálenosti, horní bajt vzdálenosti
99	INTTEST	Experimentální funkce posílání čísla typu INT	—	—
100	ERROR	Hlášení o chybě	—	číslo chyby, kontext chyby
101	SHUT_DOWN	Příkaz k ukončení programu podvozku	doplňkový bajt	—
102	DISCONNECT	Příkaz k odpojení klienta	doplňkový bajt	—
103	INIT	Příkaz k provedení inicializace motorů	doplňkový bajt	ID
104	PING	Paket sloužící k zjištění časové odezvy při komunikaci	doplňkový bajt	doplňkový bajt

Tabulka 2.1: Typy komunikačních segmentů

Číslo chyby	Typ	Popis
200	ERR_SERIAL_OPEN	Sériový port nebyl otevřen pro komunikaci.
201	ERR_SERIAL_SEND	Nepodařilo se odeslat data přes sériový port.
202	ERR_SERIAL_RECEIVE	Nepodařilo se přijmout data přes sériový port.
203	ERR_BAUD_SET	Přenosová rychlost motoru nebyla nastavena.
204	ERR_ID_SET	Nové ID motoru nebylo nastaveno.
205	ERR_UNKNOWN_COMMAND	Program podvozku přijal neznámý příkaz.
206	ERR_GAIN_SET	Zesílení motoru nebylo nastaveno.
207	ERR_RESOL_SET	Rozlišení motoru nebylo nastaveno.
208	ERR_BOUND_SET	Omezení natočení motoru nebylo nastaveno.
209	ERR_OVERCT_SET	Proudová ochrana nebyla nastavena.
210	ERR_GAIN_READ	Zesílení motoru nebylo zjištěno.
211	ERR_RESOL_READ	Rozlišení motoru nebylo zjištěno.
212	ERR_BOUND_READ	Omezení natočení motoru nebylo zjištěno.
213	ERR_OVERCT_READ	Proudová ochrana nebyla zjištěna.
214	ERR_PWD_ERR	Chyba při odpojení motorů.

Tabulka 2.2: Typy chybových hlášení

3 PROGRAM PRO PODVOZEK

Program pro řízení podvozku nyní funguje jako server (angl. sluha - má za úkol podávat informace klientovi). Naslouchá na TCP portu 3000 a čeká na připojení klienta (uživatelského počítače). Jakmile se klient připojí, může začít obousměrná komunikace. V případě, že server dostane nějaký požadavek od klienta, vyhodnotí, co je třeba udělat, provede danou akci a pošle zpět informaci o stavu provedené události. Program všechny události monitoruje a je možné je v reálném čase sledovat.

Pro psaní programu jsme zvolili objektově orientovaný jazyk C++. Výhody oproti neobjektově orientovaným jazykům jsou především v lepším členění kódu. Ten může nabývat vyšší abstrakce a tím pádem lepšího rozdělení do logických funkčních bloků. Dalo by se namítnout, že kód napsaný v C++ může být za některých okolností zpomalen oproti kódu v čistém C (např. obsluhováním vyjímek, voláním virtuálních metod atd.). V našem případě však tyto rozdíly považujeme za minimální a můžeme je zanedbat.

Program je navržen tak, aby bylo možné kdykoliv jednoduše změnit jeho část. Pokud bychom například chtěli na podvozku instalovat jiný typ motorů, bylo by nepraktické přepisovat celý program. Jelikož je náš program koncipován modulárně, stačí pouze změnit těla již implementovaných tříd a metod určených motoru, které slouží jako aplikační rozhraní pro ostatní třídy.

Při realizaci programu pro podvozek jsme narazili na celou řadu problémů. Jeden z výraznějších byl problém s překladem programu pomocí speciálního překladače (tzv. kompilátor) určeného přímo pro procesor Axis ETRAX 100LX, kterým je FOX Board LX832 osazen, a s následným sestavením programu. Náš projekt je totiž tvořen více soubory, navíc operační systém FOX Boardu neobsahuje knihovny potřebné pro programy psané pomocí jazyka C++. Bylo tedy nutné správně sestavit tzv. makefile, tedy soubor, který říká překladači a sestavovacímu programu, jak při překladu programu a následnému sestavování postupovat. K tomu jsme nakonec využili programovacího prostředí pro Linux KDevelop, které nám automaticky vytvoří tento makefile. Pomocí jednoduchého skriptu jsme pak již schopni snadno přeložit a sestavit program i pro FOX Board. Všechny knihovny se pak stanou statickou součástí programu. Po vyjmutí nepotřebných částí knihoven dostáváme výsledný spustitelný soubor, který celkově zabírá méně paměti, než kdyby byly potřebné knihovny součástí operačního systému FOX Boardu.

3.1 Vlákna

Program využívá při svém běhu tzv. vláken. Po svém zapnutí se rozdělí do několika dalších procesů, které mohou běžet paralelně, resp. se po určitém čase střídají ve svém vykonávání. Tento čas je dynamicky udáván operačním systémem, který má na starosti

plánování procesů. Programátor nemůže přímo určit plánování procesů. Je však možné jednotlivá vlákna částečně ovládat (např. je uspat, zastavit, nastavit prioritu atd.).

Modul FOX Board LX832 je osazen mikroprocesorem Axis ETRAX 100LX, což je pouze jednojádrový procesor. Na první pohled by se tedy mohlo zdát, že vlákna nepřinesou zrychlení a jiné výhody pro program, protože nemohou být vykonávána současně. To však není zcela pravda. Vlákna jsme zvolili z několika důvodů. Rozdělí jednotlivé procesy do menších logických celků, takže každý proces může vykonávat pouze specifické, jemu vlastní, funkce (například řídit motor, komunikovat po síti) a pokud má potřebná data, nemusí čekat na dokončení jiných operací. Přestože procesy tedy neběží fyzicky paralelně vedle sebe, mohou se začít vykonávat operace, které by jinak byly zablokovány z důvodu čekání na dokončení předchozích operací. Například pokud v jednom vláknu čekáme na příchozí data od klienta, v jiném vlákně můžeme provádět důležité výpočty pro řízení podvozku.

U méně výkonných systémů mohou vlákna běh programu spíše zpomalit, protože jistá dávka času je zkonzumována operačním systémem při plánování procesů. V naší práci však tento čas zanedbáváme.

Při vývoji programu využívajícího vlákna musíme dávat pozor na celou řadu věcí a důkladně předem promyslet celou jeho koncepci a funkci. Při špatně navrženém programu může docházet k zablokování nebo jinému náhlému chybovému stavu. Tyto stavy se pak velmi špatně odhalují. Navíc mohou nastávat pouze na některých platformách a na jiných se nikdy neprojevit (např. kvůli různému plánování procesů). Časté důvody pádu programu bývají přistupování ke stejnému datovému prostoru ze dvou vláken najednou, nebo k proměnným, které byly zrušeny jiným vláknem. V praxi se tento problém řeší například pomocí tzv. mutexů, které jakoby uzamknou proměnnou pro přístup jiným vláknům, dokud dané vlákno nepřestane s touto proměnnou pracovat. To však může vést k další nežádané situaci a sice k zablokování programu, kdy vlákna navzájem čekají na data z jiného vlákna, které například data nemůže poskytnout nebo už skončilo svoji činnost. Tím pádem čekající vlákno nikdy data nedostane a zablokuje běh programu. Je tedy jasné, že program se musí navrhnout velmi obezřetně.

Na konci vlákna nebo bloku se automaticky zavolají destruktory pro všechny staticky vytvořené proměnné či třídy. Jestliže se jedna instance třídy využívá ve více vláknech, musíme dávat pozor, aby při skončení jednoho vlákna či bloku nebyla instance zrušena i přesto, že jiné vlákno ji stále pro svou funkci potřebuje. Došlo by tak ke čtení z paměti, která je už vrácená operačnímu systému a následně k chybě programu. Pro předejití této situaci využíváme tzv. počítaných ukazatelů. Jedná se o šablonu třídy, jejíž parametrem je ukazatel na sdílenou instanci třídy, která si automaticky hlídá počet vláken pracujích s ukazatelem na danou sdílenou instanci třídy. Na konci programu se tedy nejdřív zkontroluje, zda s instancí ještě pracuje i jiné vlákno. Pokud ano, instance nebude zrušena.

3.1.1 Využití knihovny ZThread

Při implementování vláken do programu jsme se rozhodli využít multiplatformní volně šířitelné knihovny ZThread. Jedná se o objektově orientované aplikační rozhraní v jazyce C++, poskytující vysokou úroveň abstrakce při využívání nativních funkcí pro práci s vlákny. Knihovna poskytuje snadnou a efektivní kontrolu vláken, bezpečné ukončování vláken, správné předávání proměnných mezi vlákny a vše potřebné.

Využitím knihovny ZThread získáváme mnoho výhod především co se týče psaní uživatelského kódu, který není náročný ani obsáhlý. Přesto máme velké možnosti využití této knihovny a záruku lepší spolehlivosti vláken.

Na ukázku jednoduchosti práce s vlákny pomocí knihovny ZThread se můžete podívat na následujícím kódu, který obstará spuštění nového vlákna a provedení metod, které implementuje třída pro příjem dat. Úkoly, které budou provedeny musí být zapsány v přetížené metodě run() třídy.

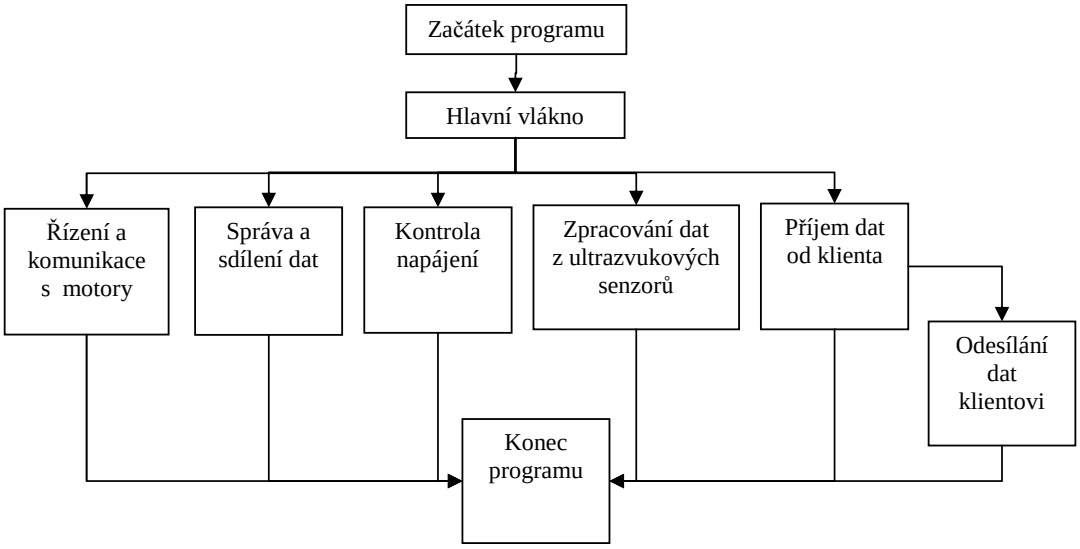
```
#include <iostream>
#include "zthread/Thread.h"
using namespace ZThread;
using namespace std;

class CSocketThread : public Runnable {
  public:
  void run() {
    // metody pro příjem dat
  }
}

int main() {
  try {
    // třída pro spouštění vláken
    ThreadedExecutor executor;

    // vytvoříme instanci třídy pro příjem dat
    CSocketThread* task = new CSocketThread();

    // a spustíme ji v novém vlákně
    executor.execute(task);
  } catch(Synchronization_Exception& e) {
    cerr << e.what() << endl;
```

```
    }
    return 0;
}
```

3.2 Struktura programu

Program se skládá ze šesti vláken, která dohromady spolupracují. Hlavní vlákno slouží ke správě a sběru informací z ostatních vláken. Další vlákno má za úkol řídit motor a komunikovat s ním. Toto vlákno získává informace o požadavcích klienta z dalšího vlákna pro příjem dat pomocí soketů (dat poslaných klientem) a naopak zpět odesílá klientovi informace prostřednictvím dalšího vlákna určeného výhradně k odesílání dat. Zbylá dvě vlákna jsou doplňková. Jedno získává data z ultrazvukových senzorů a druhé monitoruje stav napájení. Struktura programu je znázorněna na obrázku 3.1.

Na začátku programu se spustí pouze hlavní vlákno, které následně inicializuje a spustí ostatní vlákna, která se od hlavního odpojí. Pokud se k vláknu pro přijímání dat nepřipojí žádný klient, vlákno pro odesílání soketů se vůbec nespouští.



Obr. 3.1: Struktura programu

3.2.1 Správa a sdílení dat

Toto vlákno slouží k řízení ostatních vláken a k shromažďování informací o veškerých událostech, které vlákna provedla.

Inicializujeme zde instanci třídy pro ukončování vláken. K této instanci přistupují postupně všechna vlákna a ověřují, zda mají pokračovat ve své činnosti či nikoliv. Můžeme tak řídit běh vláken a při ukončení programu je jednoduše vypnout.

Dalším úkolem tohoto vlákna je sdělovat ostatním vláknům, zda mají vypisovat informace o provedených akcích pro ladění programu. Součástí programu je výpis jednoduchého textového rozhraní na konzolu mikropočítače. Pomocí tohoto rozhraní si může uživatel vybrat z textového menu, zda se mají zobrazovat informace o komunikaci pomocí soketů nebo o komunikaci s motory přes sériovou linku. Díky těmto informacím je možné jednoduše zjišťovat a ověřovat posílaná data při komunikaci.

Aby program poznal, zda má nějaké informace vypisovat, je potřebný vstup od uživatele. Vstupní operace je však záležitost, která při absenci vstupních dat zablokuje program. Proto jsme při čtení uživatelského vstupu z klávesnice využili tzv. množin, s jejichž pomocí monitorujeme soubor, reprezentující vstup z konzoly a data čteme pouze pokud se množina s tímto souborem změnila. Pokud do maximální doby 2,5 sekundy nepřijdou vstupní data, program pokračuje ve své činnosti. Toto monitorování probíhá ve stálé smyčce, dokud není program ukončen. Výhoda tohoto přístupu je též v tom, že monitorování nezatěžuje procesor tak, jako kdyby se stále četly data v neblokovacím módu.

Důležitou funkcí tohoto vlákna je však především sdílení dat mezi ostatními vlákny. Programově řečeno, každé další vlákno má na toto vlákno mezi svými členskými proměnnými ukazatel, přes který se může dostat k veškerým sdíleným datům. Ty jsou samozřejmě chráněna proti současnému přístupu z více vláken aby nedocházelo ke kolizím.

Sdílená data jsou převážně data charakterizující jednotlivé motory, jako např. natočení motoru, zesílení, omezení natočení, rozlišení atd. Sdílí se však i některé třídy pro správu. Nejdůležitější z nich jsou dvě třídy, zajišťující fronty příchozích dat od klienta a odchozích dat pro klienta. Dalšími sdílenými daty je třída pro výpis textových informací, která zajišťuje, aby byl text z jednoho vlákna vypsán jednorázově a nesmíchal se s vypsaným textem jiného vlákna.

Vývojový diagram tohoto vlákna je možné nalézt v příloze A.1.

3.2.2 Přijímání dat od klienta

Při spuštění vlákna naslouchámе na uživatelském TCP portu 3000 a čekáme na připojení klienta. Pokud se klient úspěšně připojil, vytvoří se automaticky vlákno sloužící k odesílání dat zpět klientovi. Pokud jsou na vstupu příchozí data od klienta, přijmeme je a nejedná-li se o požadavek k ukončení komunikace, zařadíme data do fronty pro zpracování vláknem řídící motor. Vývojový diagram tohoto vlákna je možné nalézt v příloze A.2.

Po připojení jednoho klienta je nastaven server tak, že už nedovolí připojení dalšího klienta a kontroluje pouze příchozí data od již připojedného počítače. Důvod je zřejmý. Situace, kdy by jeden podvozek ovládalo více klientů by vedla pouze ke zmatku. Po

přerušení klienta je však možné připojit se znova z libovolného počítače bez ohledu na předchozí připojení.

Pro komunikaci prostřednictvím soketů používáme tzv. blokovací režim. Pokud nepřichází data, program je zablokovaný a čeká na jejich příchod. To by však bylo velmi nevýhodné, kdyby došlo na straně klienta k chybě. Nemohl by se tím pádem připojit k serveru a program by zůstal zablokován a nebylo by možné ho vypnout. Proto sledujeme změny soketů pouze po dobu časového limitu (v našem případě po dobu 5 sekund) opět pomocí tzv. množin. Pokud zjistíme, že jsou na vstupu data k vyzvednutí, můžeme použít blokovací režim přijímání dat, protože máme jistotu, že existují data k vyzvednutí. Tento postup opakujeme, dokud jedna ze stran neukončí spojení. Výhoda tohoto přístupu je hlavně v minimálním zatěžování procesoru při čekání na data.

Pro přijímání dat pomocí soketů existuje i tzv. neblokovací režim, který nezpůsobí zablokování programu. Pokud data nejsou na vstupu, nic se nestane a pokračuje se dále v programu. Tato varianta je pro nás ale nevýhodná z důvodu náročnosti pro procesor. Museli bychom totiž stále v cyklu ověřovat, zda nepřišla data. Někdy se této metodě říká ”busy waiting”. To však zbytečně ubírá procesoru výkon.

Pro práci se sokety byla vytvořena speciální třída sloužící jako aplikační rozhraní zjednodušující práci. Na následujícím příkladě kódu je ukázka serveru, který naslouchá na portu 3000, po připojení klienta přijme data a následně je odešle nazpátek. Pro jednoduchost se jedná pouze o blokovací režim.

```
#include "Exceptions.h"
#include "ServerSocket.h"
#include "Packet.h"
using namespace std;

int main() {
  try {
    // vytvoření soketů pro naslouchání a příjem dat
    CServerSocket server(3000), data;
    // paket k zapsání příkazů od klienta
    CCommandPacket command;

    // naslouchame na portu 3000, dokud se nepřipojí klient
    server.SocketAccept(data);

    // uložení příchozích dat
    data >> command;
    // odeslání dat nazpět
```

```
      data << command;
    } catch(SocketException& e) {
      cerr << e.what() << endl;
    }
    return 0;
}
```

3.2.3 Odesílání dat klientovi

Jak již bylo řečeno výše, toto vlákno je spuštěno, jestliže dojde k úspěšnému připojení klienta k vláknu pro přijímání socketů. Čekáme na data od motoru, které přijdou do fronty. Z této fronty data vyzvedáváme a odesíláme je jako odpověď klientovi. Vývojový diagram tohoto vlákna je možné nalézt v příloze A.3.

Rozhodli jsme se pro odesílání dat využít samostatný proces z toho důvodu, že můžeme odeslat data klientovi bezprostředně po jejich zpracování a nemusíme čekat, než přijdou nějaká data od klienta nebo vyprší časový limit blokovacího režimu přijímání soketů.

Odesílání dat pracuje opět v blokovacím režimu. Ten zajistí, že se data odešlou pouze tehdy, je-li dostatek místa na zásobníku soketu a je tedy možné odeslat všechna data najednou. Vzhledem k tomu, že odesíláme pouze velmi malé množství dat, nemusíme se touto skutečností příliš zabývat.

3.2.4 Komunikace s motory a jejich řízení

Abychom mohli komunikovat s motory, musíme zajistit správné nastavení sériové linky RS-232 (viz sekce 2.1). Jakmile je sériový port úspěšně otevřen pro komunikaci, vlákno vyzvedává data z fronty paketů, které přišly od klienta a postupně je zpracovává.

Zpracování spočívá v rozkódování paketů pomocí komunikačního protokolu a přesného určení požadavku od klienta. Po identifikaci dat se provedou příslušné operace a pomocí komunikačního protokolu se opět zakódují data do paketu sloužícího jako odpověď klientovi. Tento paket se zařadí do fronty k odeslání a jakmile bude sám, nebo první v této frontě, bude odeslán vláknem pro odesílání dat klientovi. Vývojový diagram tohoto vlákna je možné nalézt v příloze A.4.

Komunikaci na síťové i transportní vrstvě máme zajištěnou pomocí protokolů TCP/IP. Jak již bylo řečeno v sekci 2.2, jedná se o tzv. spojovou službu se spolehlivým doručením dat. Nemusíme se tedy obávat, že data dorazí poškozená, či neúplná. Musíme však zajistit komunikaci na aplikační vrstvě. Tedy komunikaci programu klienta a programu pro řízení podvozku. Toto je zajištěno při zpracování paketu. Jedná se o jednoduchý algoritmus fungující na principu stavového automatu. Algoritmus přečte první příznakový bajt segmentu

příchozích dat a podle tohoto bajtu určí počet a význam následujících dat. Pokud není druh dat rozpoznán, paket je celý zahozen a klientovi je odeslána zpráva o chybě v komunikaci.

Díky hlavnímu vláknu pro správu a sdílení dat můžeme velmi jednoduše z každého vlákna programu odeslat klientovi data. Následující kód je ukázkou, co musí programátor vykonat pro odeslání dat. Vše ostatní se již provede automaticky. V tomto případě se jedná o odeslání paketu s pozicí motoru.

```
// vytvoření paketu s odpovědí
CResponsePacket response;
// naplnění paketu daty
response.Encode(READ_POS, motorId, position);
// zařazení paketu do odchozí fronty
management->outPacketQueue.put(response);
```

3.2.5 Zjišťování dat z ultrazvukových senzorů

Pokud je k podvozku připojen klient, čteme každou sekundu údaje z ultrazvukových senzorů podvozku a tyto údaje následně odesíláme klientovi. Vývojový diagram tohoto vlákna je možné nalézt v příloze A.5.

3.2.6 Zjišťování hodnoty napájení

Toto vlákno má za úkol pouze každých 20 sekund zjistit aktuální stav napájení a vypsat ho na LCD. Vývojový diagram tohoto vlákna je možné nalézt v příloze A.6.

4 UŽIVATELSKÝ PROGRAM

Pro tvorbu grafického uživatelského rozhraní jsme zvolili jeden z nejrozšířenějších objektově orientovaných formulářových jazyků a sice jazyk C#. Jedná se o jazyk odvozený z C++ (a Javy), ze kterého čerpá syntaxi. Oproti C++ však přináší některé programátorské výhody, jako např. to, že se programátor již téměř nemusí starat o správu paměti. Jazyk C# dělá vše automaticky, což vede k značné eliminaci častých chyb při zápisu a čtení dat. Další výhodou je také skutečnost, že C# využívá tříd .NET. Celá řada algoritmů je tedy již připravena v podobě tříd, které slouží jako aplikační rozhraní. Nemusíme tedy např. znovu psát třídy pro komunikaci pomocí soketů či třídy pro práci s vlákny. Vše je již naprogramováno a my můžeme použít tyto třídy, aniž bychom přesně znali jejich obsah a funkčnost.

Program využívá podobně jako program v mikropočítači podvozku ke své práci vlákna. U formulářových aplikací je to nutné především pro zajištění flexibilního grafického rozhraní. Časově náročné procesy se řeší paralelně v jiném vlákně a tím pádem se zajistí stálá odezva grafického rozhraní. Programátor si však v tomto případě musí uvědomit, že nemůže přistupovat k prvkům formuláře z jiných vláken. K tomu se používají speciální metody a sice tzv. delegáti. Při nastavování některé grafické komponenty z jiného než grafického vlákna se zavolá příslušný delegát, který provede danou akci až v momentě, kdy bude program běžet v tomto grafickém vlákně.

Vzhledem k tomu, že se jedná o událostmi řízené programování, nelze strukturu vykonávání programu jednoznačně popsat vývojovým diagramem.

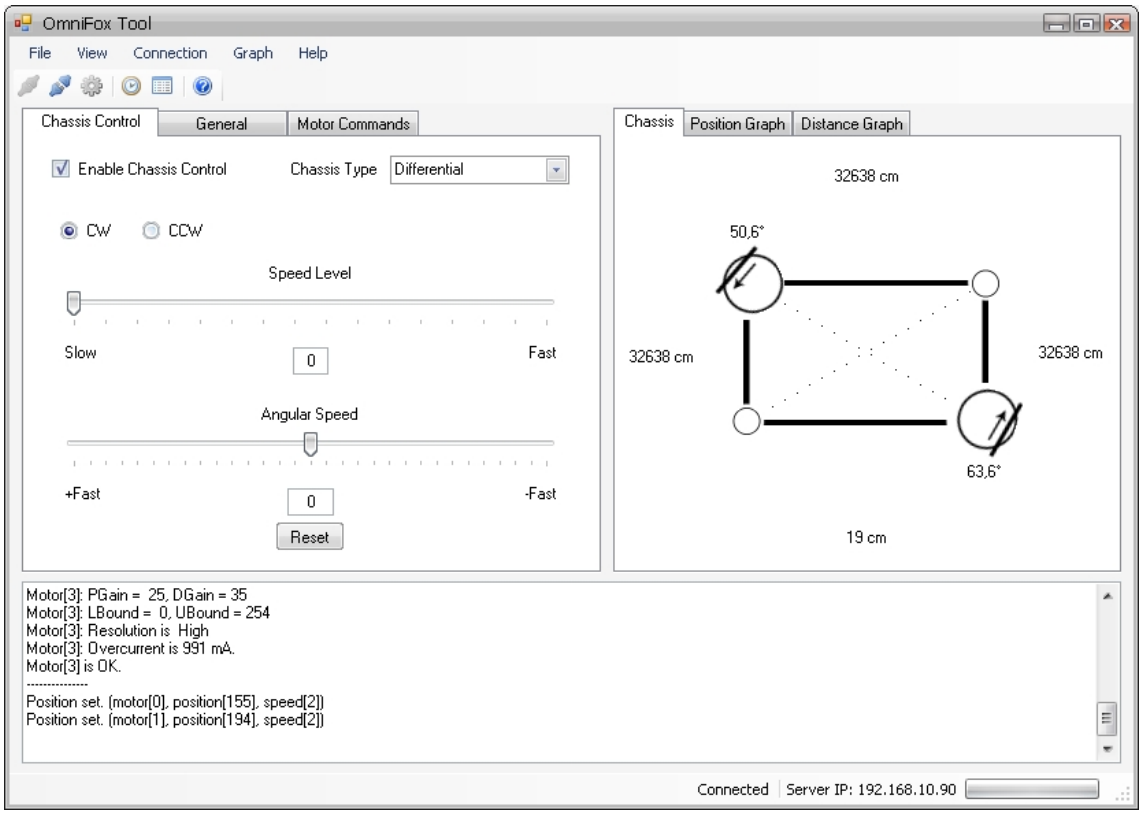
Základní funkce aplikace jsou následující:

- Připojení k podvozku a komunikace s ním
- Monitorování stavu motorů
- Kompletní ovládání a nastavování jednotlivých motorů
- Vizualizace natočení hnacích soustav
- Ovládání podvozku typu bicykl a diferenciální podvozek
- Vykreslování polohy a uražené vzdálenosti podvozku do grafu (odometrie)

Na obrázku 4.1 můžete vidět ukázku grafického uživatelského rozhraní při řízení diferenciálního podvozku.

4.1 Připojení k podvozku a komunikace s ním

Při navazování spojení a příjmu dat používáme tzv. asynchronní metody. Znamená to, že při zavolání příslušné metody nedochází k zablokování programu do jejího ukončení,



Obr. 4.1: Ukázka uživatelského rozhraní

protože metoda probíhá v odděleném vlákně. To je nutné především pro zajištění odpovídajícího grafického rozhraní. Program je nastaven tak, že se snaží připojovat ke klientovi po maximální dobu 30 sekund. Pokud se do této doby nepodaří spojení navázat, je „vyhozena“ vyjímka.

Klient se připojuje k portu 3000, tedy ke stejnému portu, na kterém naslouchá program podvozku. Uživatel má možnost nastavit pouze IP adresu serveru (tedy podvozku). Číslo portu nelze z grafického rozhraní změnit.

Do programu byla zabudována speciální třída, která slouží k ukládání nastavených parametrů a jejich obnovení při opětovném spuštění programu. V tomto případě se jedná o ukládání předchozí IP adresy serveru.

Neprodleně po spuštění programu se spustí nové oddělené vlákno, které slouží pro příjem dat a zpracování dat. Vývojový diagram tohoto vlákna je ve své podstatě stejný, jako u vlákna programu na podvozku pro komunikaci s motory. Tento diagram lze nalézt v příloze A.4. Rozdíl je de facto pouze v tom, že po zpracování dat klient již neodesílá žádnou odpověď. Vlákno tedy pouze čeká na příchozí data do fronty, která vyzvedává, následné dekóduje a provádí příslušné akce a výpočty. Toto se stále opakuje, dokud není ukončen celý program.

4.2 Monitorování stavu motorů

Při spuštění programu jsou údaje o jednotlivých motorech neinicializované. Bezprostředně po připojení klienta však server (podvozek) odešle klientovi všechna potřebná data k inicializaci statistik. Uživatel pak může vidět všechna potřebná nastavení jednotlivých motorů. Jedná se konkrétně o pozici natočení motoru, rozlišení, proudovou ochranu, spodní a horní omezení rozsahu pozice a konečně proporcionální a diferenciální zesílení motoru. Tyto statistiky jsou automaticky aktualizovány při každé provedené změně, jelikož podvozek o každé akci klienta informuje příslušným paketem. Uživatel má z grafického prostředí možnost vynutit si novou inicializaci a tím obnovit aktuální statistiku dat motorů.

Po odpojení klienta všechny statistiky zůstávají nedotčené a je tedy možné je prohlížet i offline. Při každém nové připojení jsou data obnovena.

4.3 Ovládání a nastavování jednotlivých motorů

Program slouží také jako prostředí pro nastavování parametrů a ovládání jednotlivých motorů. Kompletní seznam parametrů a příkazů lze nalézt v příručce motoru. Jmenujme alespoň nejdůležitější z nich a sice zjišťování pozice, zadávání nové pozice, zapínání rotace o 360 stupňů, změnění rozlišení motoru, nastavení ID motoru, či odpojení motoru a tím snížení spotřeby.

Pro zadávání přesné pozice má uživatel k dispozici speciální sekci, kde může synchronně zadávat pozici všem čtyřem motorům. Zmenšuje se tím časová prodleva mezi zadáváním pozic jednotlivým motorům.

Po vykonání daného příkazu podvozek automaticky odpoví potvrzujícím či chybovým hlášením a klientský program zaktualizuje statistiky motorů.

4.4 Vizualizace natočení hnacích soustav

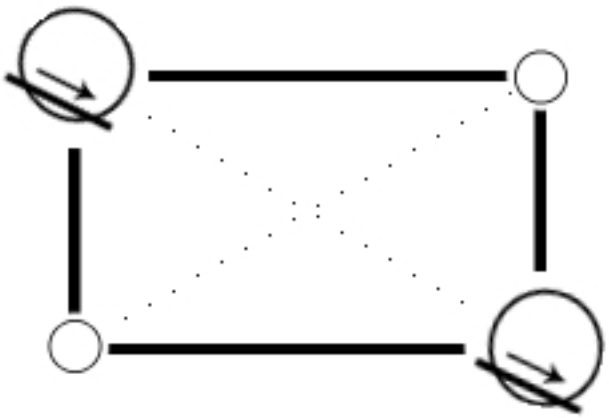
Na obrázku 4.1 je vidět způsob provedení vizualizace podvozku. Jedná se o orientační schéma podvozku při pohledu shora. Uživatel má pak možnost sledovat natočení hnacích soustav. Součástí vizualizace je také zobrazování úhlu natočení motorů. Podvozek je vybaven ultrazvukovými senzory, které snímají okolní předměty. Údaje ze senzorů jsou opět zobrazeny ve schématu.

Vizualizace je aktualizována v „reálném“ čase. Tedy jakmile se úspěšně provede natočení kola, podvozek pošle ztvrzující zprávu klientovi a ten pak na základě této zprávy natočí kolo odpovídajícím způsobem.

4.5 Ovládání podvozku typu bicykl a diferenciální podvozek

Hlavní funkcí uživatelského programu je ovládání podvozku typu bicykl a diferenciálního typu. Při zapnutí ovládání daného typu, se kola vždy nakonfigurují do správné pozice.

Při zapnutí ovládání typu bicykl se kola natočí tak, že jsou jejich osy rovnoběžné a zároveň kolmé na úhlopříčku těla podvozku. Schéma natočení hnacích soustav je možné vidět na obrázku 4.2.



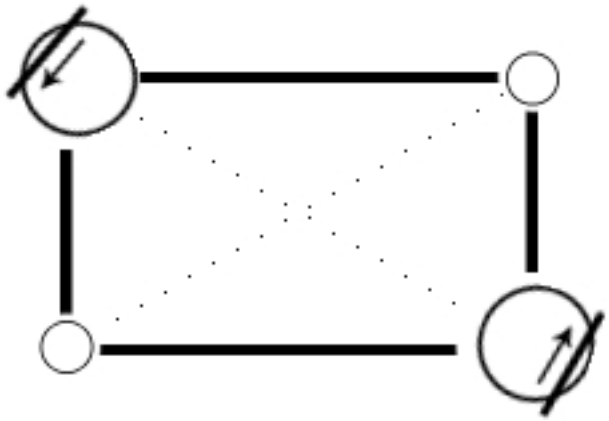
Obr. 4.2: Schéma konfigurace hnacích soustav podvozku typu bicykl

Řízení pohybu v tomto případě pak spočívá pouze v zadávání dopředné rychlosti podvozku a v natáčení předního kola. Rychlost předního a zadního kola je stejná. Oproti jiným typům podvozků je schopnost manévrovat poněkud menší. V některých situacích se může stát, že podvozek není schopen z místa vyjet (např. v rozích místností atd.).

Pokud přepneme v programu řízení na typ diferenciální podvozek, kola se opět samy natočí. A sice tak, že jejich osy leží na stejné přímce, která je zároveň úhlopříčkou těla podvozku. Schéma natočení hnacích soustav je možné vidět na obrázku 4.3.

U tohoto typu řízení se již nemění natočení žádného z kol. Pohyb je řízen pouze rychlostí a směrem otáčení motorů. Pokud se kola otáčejí stejným směrem a stejnou rychlostí, podvozek bude vykonávat přímočarý pohyb. Jestliže se jedno kolo otáčí menší rychlostí, či dokonce jiným směrem, robot bude na tuto stranu zatáčet. Speciální případ nastává tehdy, jestliže se kola točí stejnou rychlostí, avšak opačným směrem. V této situaci se bude podvozek otáčet na místě s osou rotace přímo ve středu svého těla. Osa rotace podvozku tedy leží kdekoliv na přímce os obou kol.

Dopředná rychlost robota a úhlová rychlost jeho otáčení jsou na sobě závislé. Pokud je dopředná rychlost maximální, nemůže podvozek již zatáčet, aniž by se jeho dopředná rychlost nesnížila. Je to způsobeno maximální realizovatelnou rychlostí otáčení kol, resp. motorů. Na tuto skutečnost jsme mysleli při realizaci programu. Při nastavování dopředné



Obr. 4.3: Schéma konfigurace hnacích soustav podvozku diferenciálního typu

se automaticky mění rozsah maximální úhlové rychlosti a naopak. Tímto způsobem můžeme docílit maximálního využití rychlostí motoru.

4.6 Odometrie

Jedna z výhod tohoto podvozku je, že je možné využít odometrii, tedy určování pozice na základě rychlosti a orientace kol. Odometrii je možné spočítat jak pro podvozek typu bicykl či diferenciální typ, tak pro všesměrový podvozek. Je to možné z toho důvodu, že se při správné konfiguraci kol v ideálním případě nevyskytuje tření a prokluz, které se nedají do výpočtu zahrnout z důvodu neurčitosti.

Jakmile se spustí řízení daného typu, automaticky se spustí nové vlákno na pozadí, které v daných časových okamžicích (v našem případě jednou za sekundu) odesílají dotazy podvozku. Podvozek tyto dotazy zpracuje a odpoví potřebnými daty pro daný typ ovládání podvozku, aby z nich bylo možné zjistit aktuální pozici. Odometrie se počítá v klientském programu, aby se program v mikropočítači podvozku zbytečně nezatěžoval. Vypočítaná data se pak zakreslují do grafů v uživatelském prostředí. Konkrétně se jedná o graf pozice podvozku (viz. obrázek 4.4) a o závislost uražené vzdálenosti v čase (viz. obrázek 4.5).

Při výpočtu odometrie podvozku typu bicykl uvažujeme lineární aproximaci pohybu po kružnici. Přesnost je tedy závislá na intervalech vzorkování. Vycházíme z předpokladu, že známe uraženou vzdálenost podvozku a úhel natočení předního kola. Z těchto údajů pak spočítáme aktuální orientaci podvozku a poté uraženou vzdálenost ve směru osy x a osy y. Schéma k výpočtu odometrie je znázorněno na obrázku 4.6. Ze známé délky uhlopříčky podvozku l a z úhlu natočení předního kola alfa můžeme spočítat přibližný

poloměr otáčení:

$$r = \frac{l}{\sin \alpha} \tag{4.1}$$

Dále již jednoduše spočítáme úhel natočení podvozku beta a celkovou orientaci:

$$\beta = \arcsin \frac{d}{r} \tag{4.2}$$

$$O(t+1) = O(t) + \beta \tag{4.3}$$

Z aktuální orientace spočítáme souřadnice polohy:

$$x(t+1) = x(t) + d \cos O(t+1) \tag{4.4}$$

$$y(t+1) = y(t) + d \sin O(t+1) \tag{4.5}$$

Při výpočtu odometrie podvozku diferenciálního typu uvažujeme opět lineární aproximaci pohybu po kružnici. Schéma k výpočtu odometrie je znázorněno na obrázku 4.7. Jestliže známe rozpětí kol W a uraženou vzdálesnost každého kola D_R a D_L, můžeme vypočítat aktuální orientaci pomocí vztahu:

$$O(t+1) = O(t) + \frac{(D_R - D_L)}{W} \tag{4.6}$$

Dále uraženou vzdálenost:

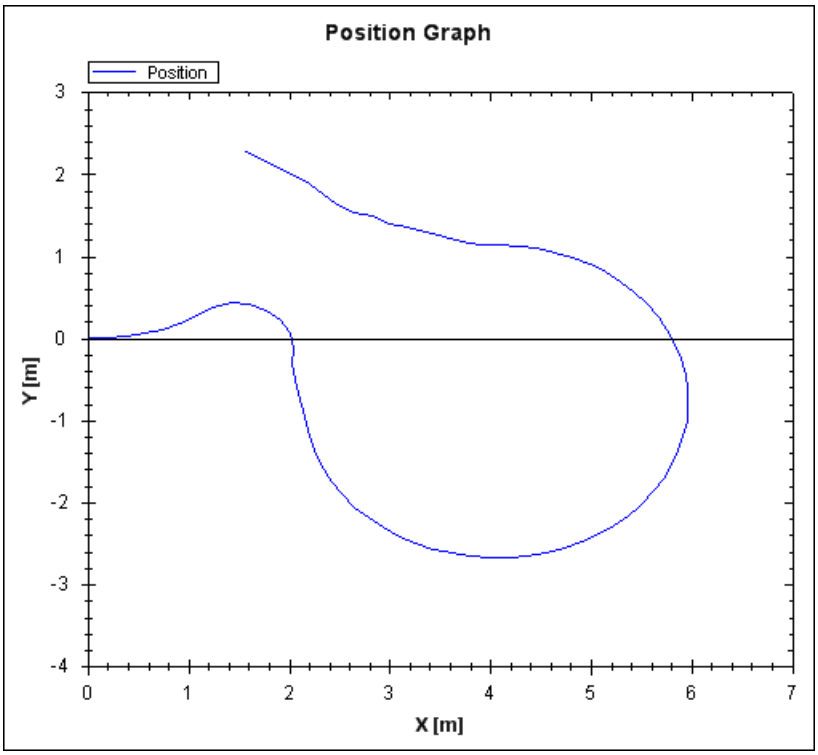
$$D(t,t+1) = \frac{(D_R - D_L)}{2} \tag{4.7}$$

Z aktuální orientace a uražené vzdálenosti spočítáme souřadnice polohy:

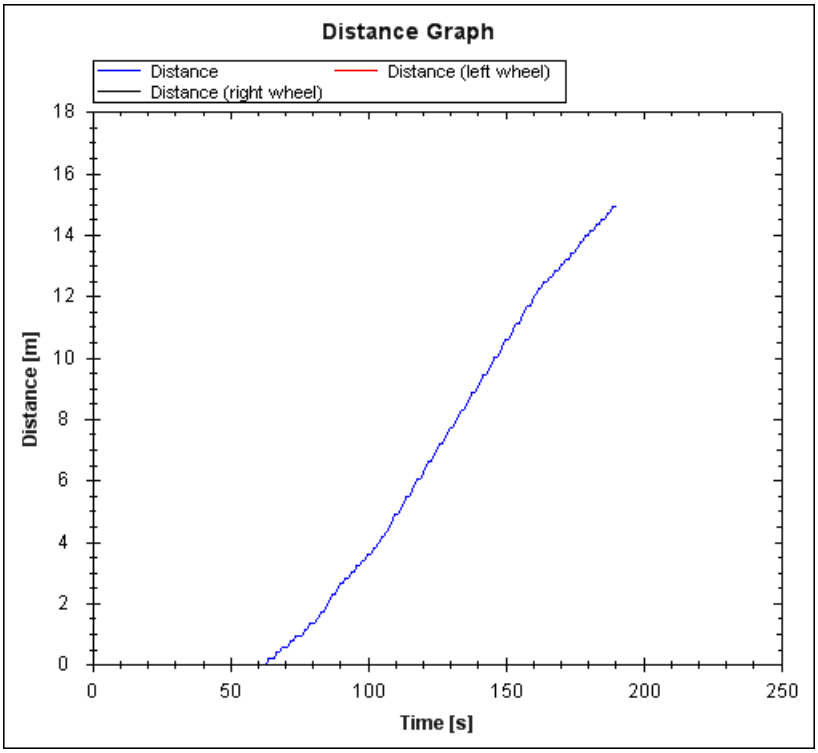
$$x(t+1) = x(t) + D(t,t+1) \cos O(t+1) \tag{4.8}$$

$$y(t+1) = y(t) + D(t,t+1) \sin O(t+1) \tag{4.9}$$

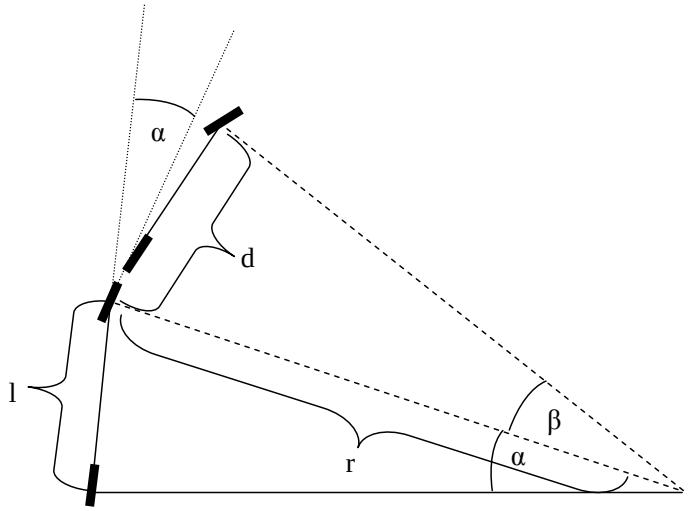
Při výpočtech narážíme na problém se zjišťováním uražené vzdálenosti jednotlivých kol. Z motorů se nedá jednoznačně určit přesné natočení mezi otáčkami. K odometrii je tedy možné použít pouze zjišťování uražené vzdálenosti na základě provedených otáček, což je v případě diferenciálního podvozku příliš hrubý údaj. U typu bicykl se tento problém neprojeví, protože se kola točí stejně. U diferenciálního typu však při výpočtu pozice vznikají příliš velké skoky, protože algoritmus má za to, že se jedno kolo např. vůbec neotočilo a druhé naopak udělalo celou otáčku, což nemusí být pravda. Přestože je algoritmus výpočtu správný, pozice není zcela jednoznačně určitelná. Řešením by bylo vyměnění hnacích soustav za motory, které dokáží přesněji určovat svoje natočení.



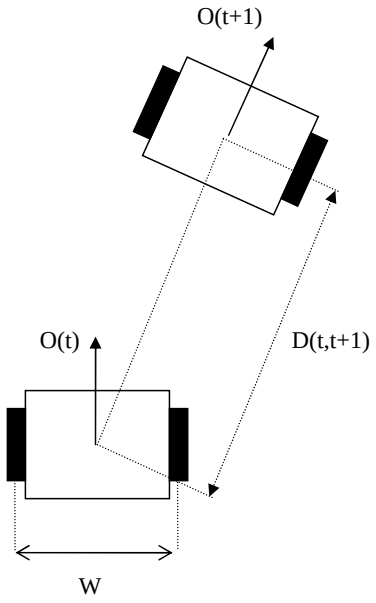
Obr. 4.4: Graf vývoje pozice podvozku



Obr. 4.5: Graf závislosti uražené vzdálenosti podvozku na čase (typ bicykl)



Obr. 4.6: Schéma k výpočtu odometrie podvozku typu bicykl



Obr. 4.7: Schéma k výpočtu odometrie podvozku diferenciálního typu

5 VYUŽITÍ PROSTŘEDÍ MATLAB

Pro simulaci chování všesměrového podvozku využíváme prostředí Matlab Simulink. Pomocí matematického modelu vypočítáváme požadované parametry pro řízení podvozku. Pro výpočet zadáváme dopřednou rychlost jednoho kola a úhlové rychlosti otáčení obou kol. Z těchto hodnot následně dopočítáváme dopřednou rychlost druhého kola tak, aby při pohybu podvozku nedocházelo ke smyku či prokluzu kol.

Simulinkový model je přizpůsoben k tomu, aby bylo možné zadané a vypočítané hodnoty posílat pomocí komunikačního protokolu přímo reálnému modelu podvozku a v „reálném“ čase sledovat, jak se podvozek chová. Momentální nastavená perioda výpočtů je 0,2 sekundy. Přibližně po těchto intervalech by se měla data posílat podvozku.

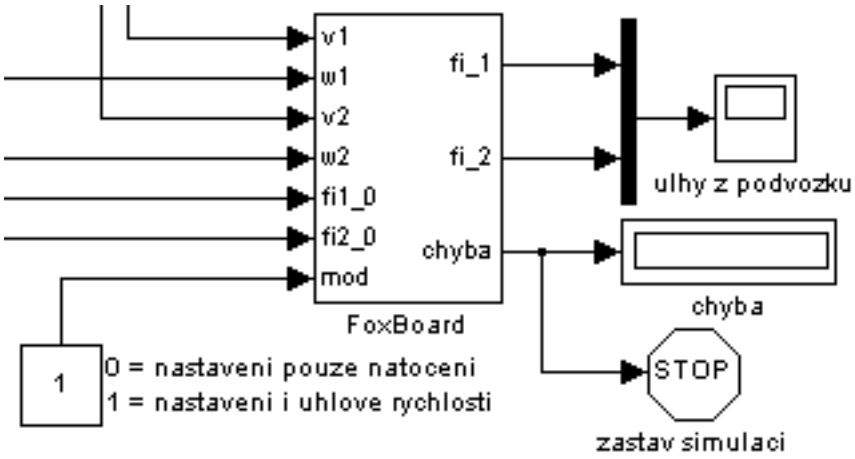
Před spuštěním matematického modelu je nutné provést inicializaci proměnných potřebných pro výpočty a pro připojení k podvozku. Jakmile se nastaví správná IP adresa pro připojení a byl spuštěn inicializační soubor, je možné spustit matematický model. Při spuštění se Simulink automaticky připojí k podvozku. Před začátkem simulace jsou kola natočena na výchozí úhel, který je zároveň jako počáteční podmínka integrátorů v modelu. V průběhu simulace program odesílá v daných časových intervalech parametry nastavení motorů. Jakmile simulace skončí, program odešle podvozku paket s příkazem k zastavení a odpojení motorů. Ukončením programu Simulink se automaticky program odpojí a podvozek čeká na další připojení klienta. Pro připojení k podvozku program využívá tzv. Instrument Control Toolbox™ a jeho funkcí pro práci s protokolem TCP/IP, který je součástí prostředí Matlab.

Největším současným omezením našeho řešení jsou však ne zcela vyhovující motory podvozku. Hlavní problém je, jak již bylo zmíněno v kapitole 1.2, že se kvůli připojení motorů na napájení nemohou hnací soustavy nepřetržitě otáčet. Momentálně se tedy motor při zadané úhlové rychlosti otáčení přesune pouze do jedné z krajních poloh a tam se zastaví. Další nevýhodou je fakt, že pro natáčení hnacích soustav musel být zvolen mód pro nastavování přesné pozice motoru, který umožňuje měnit rychlost pouze v rozsahu pětistupňové stupnice. Jinými slovy vzniká poměrně vysoká kvantovací chyba úhlové rychlosti natáčení kol. Při vývoji byla snaha využít tzv. mód rotace o 360 stupňů, který umožňuje nastavit rychlost rotace v rozsahu šestnáctistupňové stupnice. Při tomto módu však nelze jednoznačně určit natočení motoru, tudíž nebylo možné ošetřit zastavení natáčení v případě dosažení některé z krajních pozic. Možným řešením je buď nahradit stávající motory za jiné s lepšími parametry, či např. přidat senzory krajních poloh natočení motorů.

Byly realizovány dva typy simulace. U prvního typu je možné zadat podvozku pouze úhel natočení kol a dopřednou rychlost prvního kola. Dopředná rychlost druhého kola je pak automaticky dopočítána a úhlová rychlost otáčení hnacích soustav je nulová. Tento model funguje bez problémů. U druhého typu je již možné zadat všechny parametry,

čili počáteční úhly natočení hnacích soustav, úhlové rychlosti otáčení hnacích soustav a dopřednou rychlost prvního kola. Dopředná rychlost druhého kola je opět dopočítávána. Tento model výrazně trpí problémy popsanými v předchozím odstavci.

Na obrázku 5.1 je zobrazen simulinkový blok sloužící ke komunikaci s podvozkem. Jak je vidět, blok má šest vstupů pro parametry motorů a jeden vstup, kterým se dá přepínat mód. Pokud se vstup „mod“ rovná nule, nebudou se kola natáčet a budou pouze nakonfigurována na výchozí úhel před začátkem simulace. Pokud bude hodnota tohoto vstupu jedna, budou podvozku posílány i údaje o úhlové rychlosti natáčení kol.



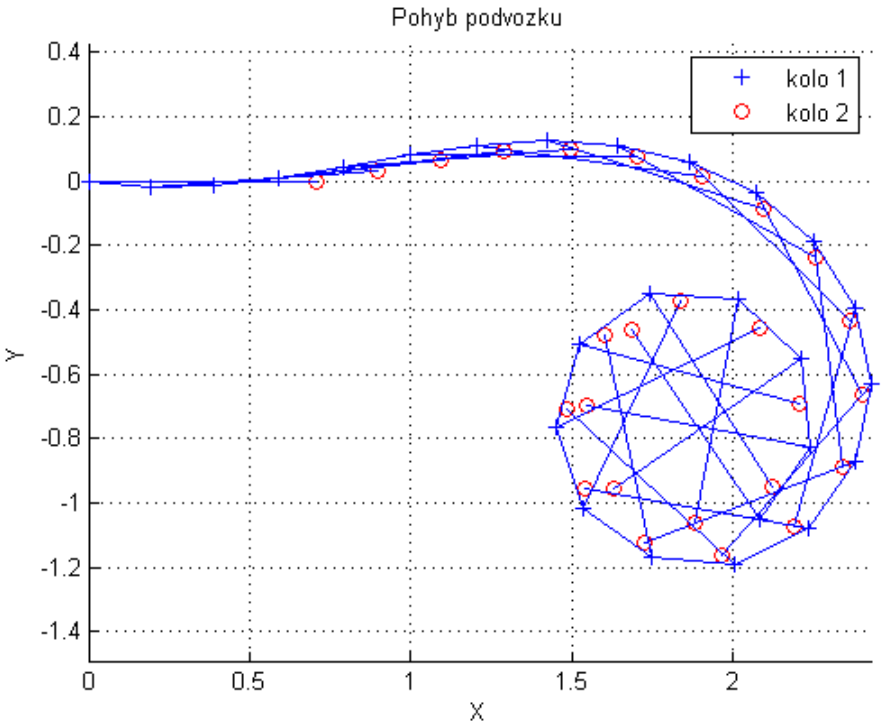
Obr. 5.1: Simulinkový blok pro komunikaci s podvozkem

5.1 Výsledky simulací

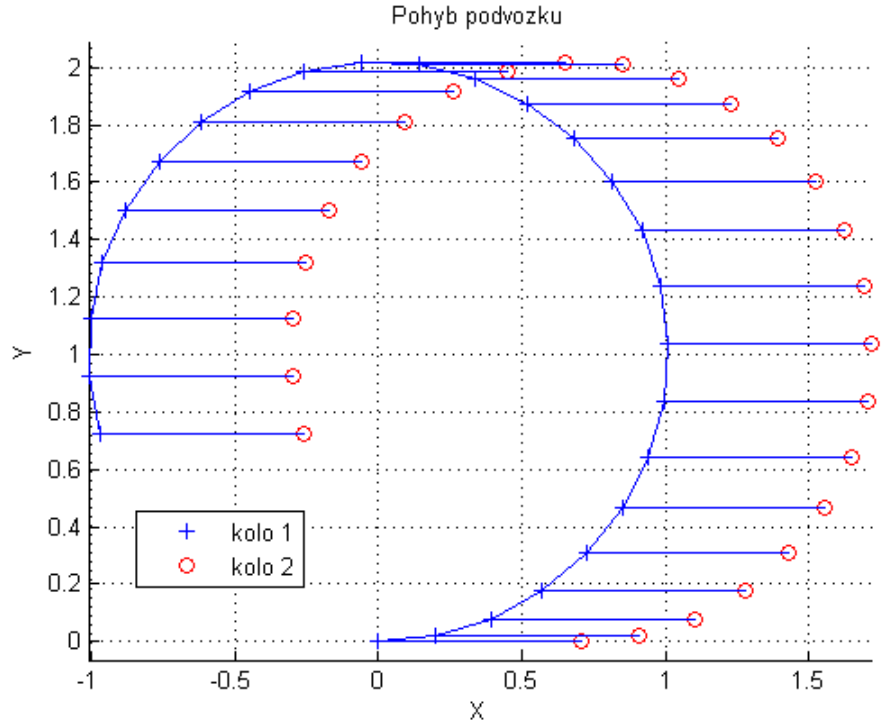
Na obrázku 5.2 je možné vidět typ pohybu, kdy podvozek začne vykonávat translační pohyb a následně se jeho pohyb změní v rotační. To je docíleno tím, že jsou osy kol nejdříve téměř rovnoběžné a tedy se ideálně protínají v nekonečnu. Jedná se nejdříve o podvozek typu bicykl. Následně jsou kola konfigurována tak, že se stále natáčí a bod protnutí jejich os se přibližuje k tělu robota, dokud nejsou osy zcela totožné. V tomto okamžiku se z podvozku stává diferenciální podvozek a točí se na místě.

Následující simulace z obrázku 5.3 zobrazuje případ, kdy se podvozek pohybuje po kružnici. Zvláštnost je v tom, že při tomto pohybu nedochází k změně natočení těla podvozku. Kola jsou při pohybu natáčena tak, že jejich osy směřují do středu opisované kružnice. Je nutné si uvědomit, že střed kružnice je pro každé kolo jinde.

Následným zkombinováním předešlých dvou pohybů je pak možno docílit toho, že se bude robot pohybovat po dané trajektorii a při tomto pohybuje bude navíc vykonávat rotační pohyb.



Obr. 5.2: Přechod translačního pohybu do rotačního



Obr. 5.3: Pohyb po kružnici bez natočení těla robota

6 ZÁVĚR

Práce zahrnovala navržení a realizaci komunikace podvozku s počítačem. Návrh programu pro komunikaci je založen na využití více vláken programu a následně i realizován. Pomocí počítače jsme schopni komunikovat s řídícím počítačem podvozku přes Ethernet kabel či bezdrátově pomocí Wi-fi, zadávat mu příkazy k provedení a poté dostávat odpovědi z podvozku o aktuálním stavu provedení našich příkazů. To vše pomocí jednoduchého, námi navrženého, komunikačního protokolu. Dodatečně je možné sledovat všechny prováděné akce přímo z programu podvozku. Program navíc zpracovává data z ultrazvukových senzorů a odesílá je připojenému klientovi. V neposlední řadě je zobrazován stav napájení na LCD.

Dalším úkolem bylo vytvoření uživatelského prostředí. K realizaci jsme využili jednoho z nejrozšířenějších formulářových jazyků a sice jazyk C#. Uživatelský program je schopen připojit se k mikropočítači podvozku a komunikovat s ním. Hlavní funkce programu jsou především ovládání podvozku typu bicykl a podvozku diferenciálního typu. Uživatel je při tom informován o přibližné aktuální poloze podvozku a jeho uražené trajektorii pomocí odometrie. Taktéž je možné zobrazit jednoduchou vizualizaci podvozku, která znázorňuje natočení hnacích soustav. Program slouží též jako rozhraní pro nastavování a zobrazení jednotlivých parametrů motorů.

Posledním bodem práce bylo propojení a komunikace podvozku s prostředím Matlab. Toto propojení slouží k simulaci chování podvozku všesměrového typu. Pomocí matematického modelu sestaveného v Simulinku jsou dopočítávány parametry nastavení motorů a následně odesílány v „reálném“ čase podvozku. Je tedy možné srovnávat výsledky simulace a chování reálného modelu podvozku.

Při realizaci programu pro podvozek jsme narazili na celou řadu problémů. Jeden z výraznějších byl problém se sériovou linkou při komunikaci s motory, kdy jejich odezva byla místo několika milisekund udávaných výrobcem v řádech stovek milisekund, což bylo nepřípustné. Řešením bylo přenastavení parametrů jádra operačního systému mikropočítače podvozku tak, aby bylo možné rychleji číst data ze sériové linky. Další nemalou překážkou byl problém s překladem programu pomocí speciálního překladače určeného přímo pro procesor Axis ETRAX 100LX, kterým je FOX Board LX832 osazen, a s následným sestavením programu. Náš projekt je totiž tvořen více soubory, navíc operační systém FOX Boardu neobsahuje knihovny potřebné pro programy psané pomocí jazyka C++. Bylo tedy nutné správně sestavit tzv. makefile, tedy soubor, který říká překladači a sestavovacímu programu, jak při překladu programu a následnému sestavování postupovat. K tomu jsme nakonec využili programovacího prostředí pro Linux KDevelop, které nám tento makefile automaticky vytvoří.

V dalším vývoji podvozku by bylo vhodné přidat např. joystick, který by umožnil pohodlnější řízení podvozku, či se dokonce pokusit o jeho autonomní chování. Velkým

krokem vpřed by též bylo nahrazení současných hnacích soustav za takové, které by umožňovaly kontinuální natáčení směru kol, aniž by byly omezovány přívodem napájení. To by bylo možné např. pomocí kartáčového sběrače.

LITERATURA

[1] GREPL, R. *Modelování mechatronických systémů v Matlab SimMechanics.* [s.l.] : [s.n.], 2007. 151 s. ISBN 978-80-7300-226-8.

[2] ŠEMBERA, J., ŠOLC, F. Modelování a řízení mobilního robotu s diferenciálním podvozkem. *AT&P journal* [online]. 2007 [cit. 2009-05-26]. Dostupný z WWW: <http://www.atpjournal.sk/atpplus/archiv/2007_1/PDF/plus203-_207.pdf>.

[3] ECKEL, B. *Thinking in C++: Volume 1*. 2nd edition. [s.l.] : [s.n.], 2003. 878 s. ISBN 0-13-979809-9.

[4] ECKEL, B., ALLISON, C. *Thinking In C++ : Volume 2: Practical Programming.*. [s.l.] : [s.n.], 2004. 791 s. ISBN 0-13-035313-2.

[5] Acme Systems. *FOX Board LX documentation index*. [online] 2009 [cit. 2009-05-26]. Dostupný z WWW: <http://foxlx.acmesystems.it/?id=14>.

[6] CRAHEN, E. *ZThreads. A platform-independent, multi-threading and synchronization library for C++*. [online] 2000-2005, 2005-03-13 [cit. 2009-05-26]. Dostupný z WWW: <http://zthread.sourceforge.net>.

[7] The MathWorks, Inc. *Controlling Instruments Using TCP/IP and UDP* [online]. 1984-2009 [cit. 2009-05-26]. Dostupný z WWW: <http://www.mathworks.com/access/helpdesk/help/toolbox/instrument/index.html-?/access/helpdesk/help/toolbox/instrument/index.html>.

[8] Megarobotics, Ltd. *AI Motor-701 Manual. Ver 1.02* [online]. [2008] [cit. 2009-05-26]. Dostupný z WWW: <http://www.tribotix.info/Downloads/-Megarobotics/AI%20MOTOR-701%20manual%20v1.02.pdf>.

SEZNAM PŘÍLOH

A Programové diagramy vláken **43**

A.1 Diagram hlavního vlákna 43

A.2 Diagram vlákna pro přijímaní dat 44

A.3 Diagram vlákna pro odesílání dat 45

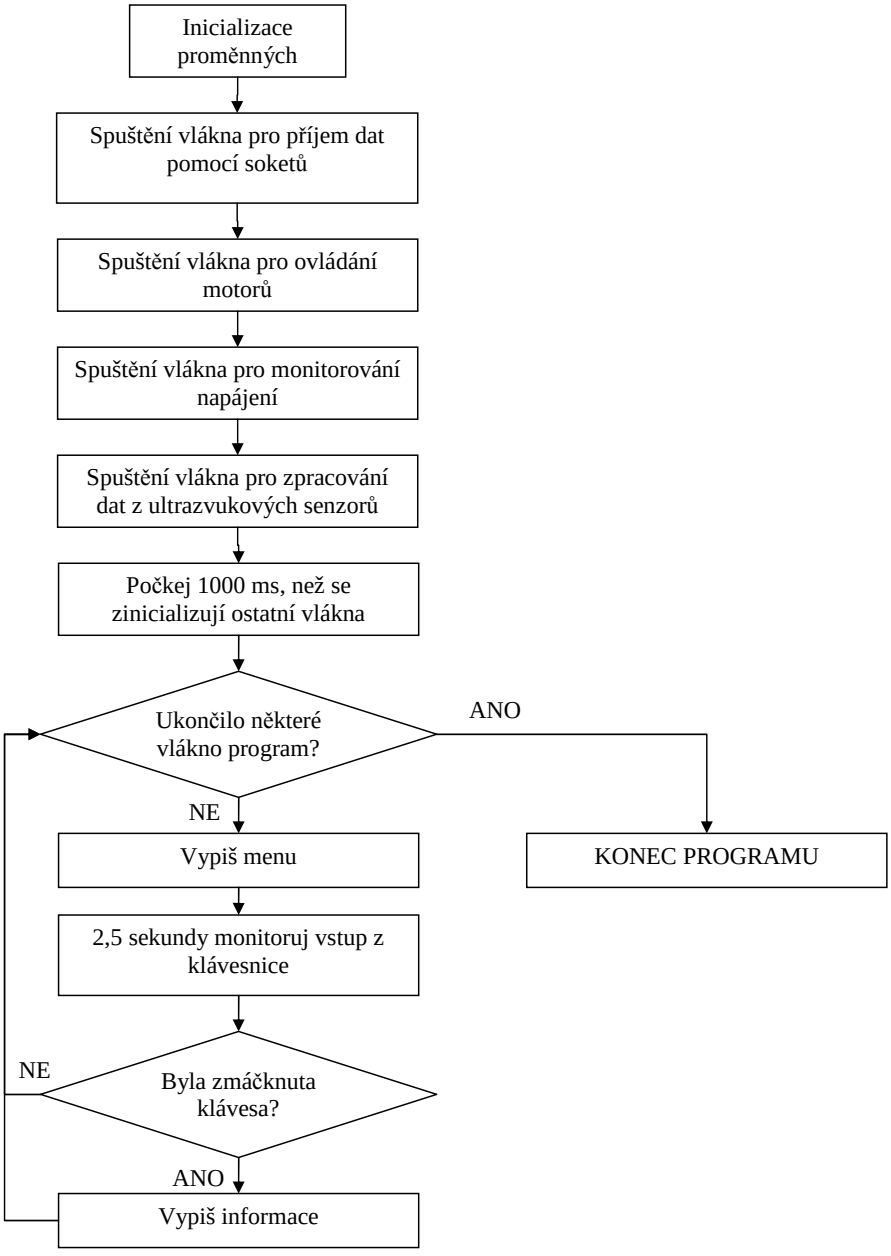
A.4 Diagram vlákna pro ovládání motorů 46

A.5 Diagram vlákna pro zjišťování údajů z ultrazvukových senzorů 47

A.6 Diagram vlákna pro zjišťování hodnoty napájení 48

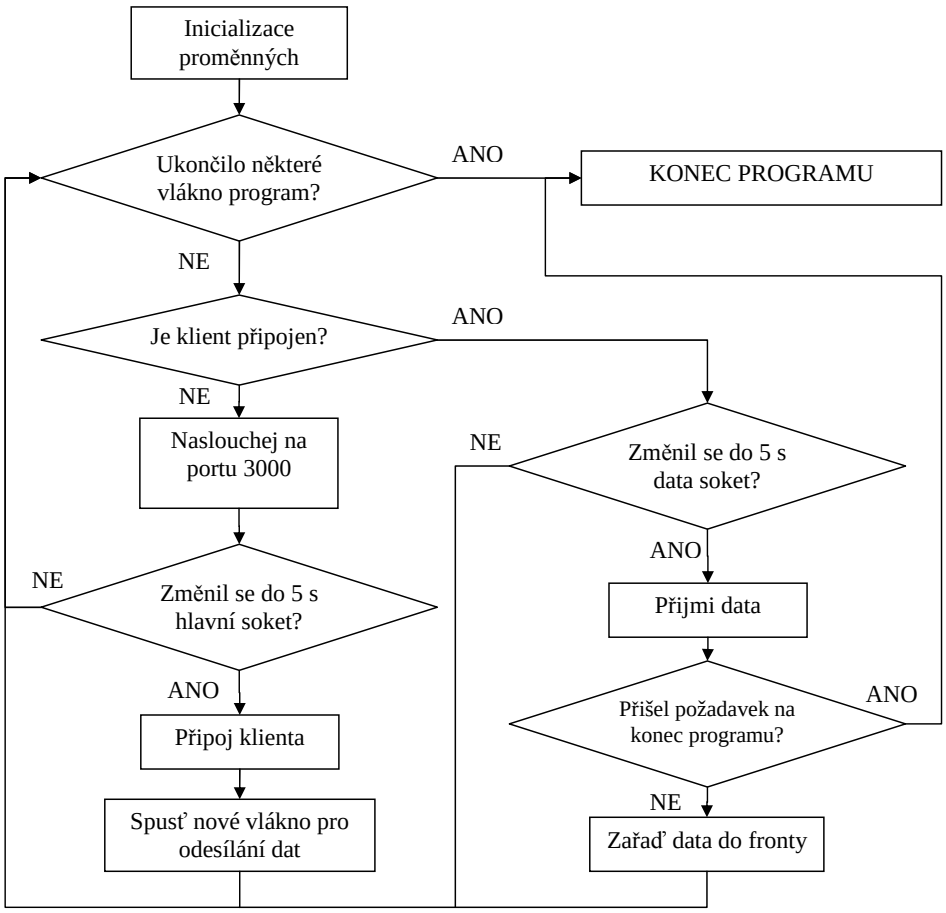
A PROGRAMOVÉ DIAGRAMY VLÁKEN

A.1 Diagram hlavního vlákna



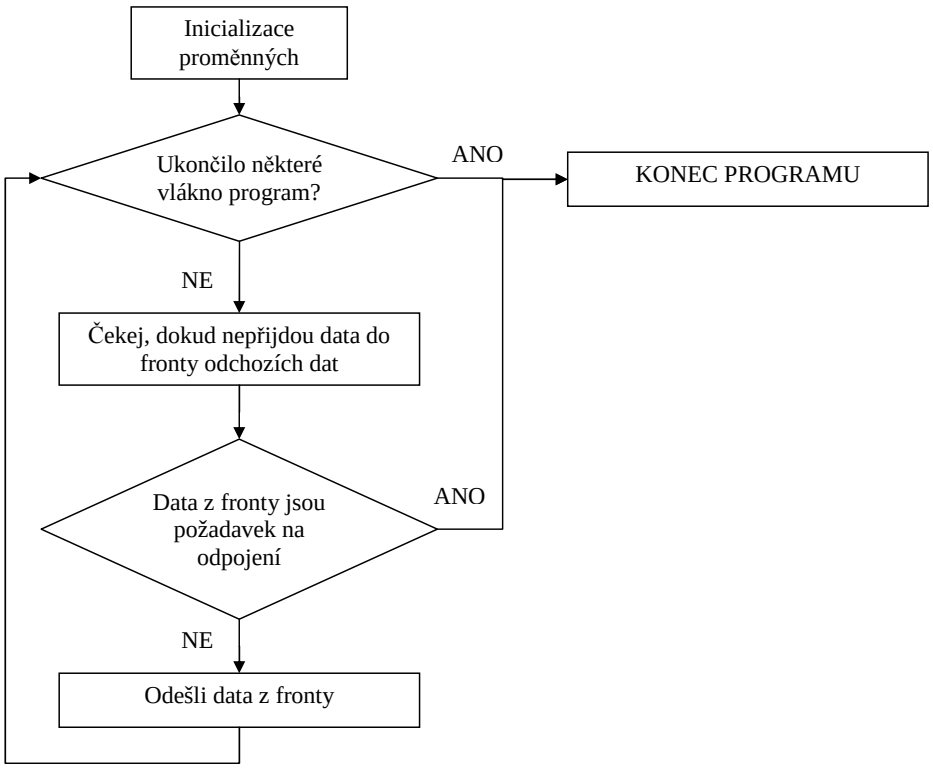
Obr. A.1: Diagram hlavního vlákna

A.2 Diagram vlákna pro přijímaní dat



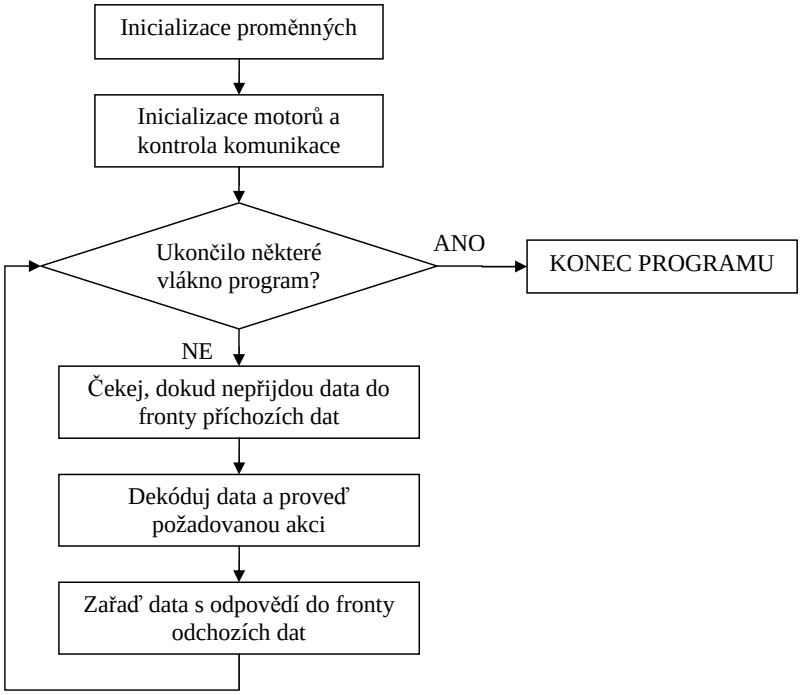
Obr. A.2: Diagram vlákna pro přijímání dat

A.3 Diagram vlákna pro odesílání dat



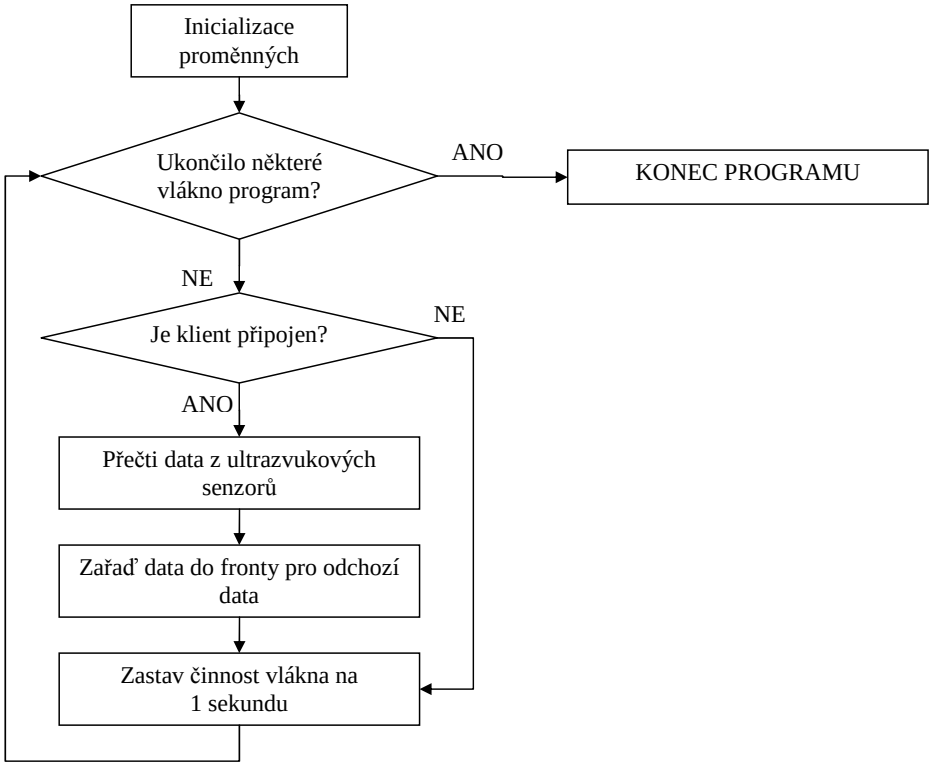
Obr. A.3: Diagram vlákna pro odesílání dat

A.4 Diagram vlákna pro ovládání motorů



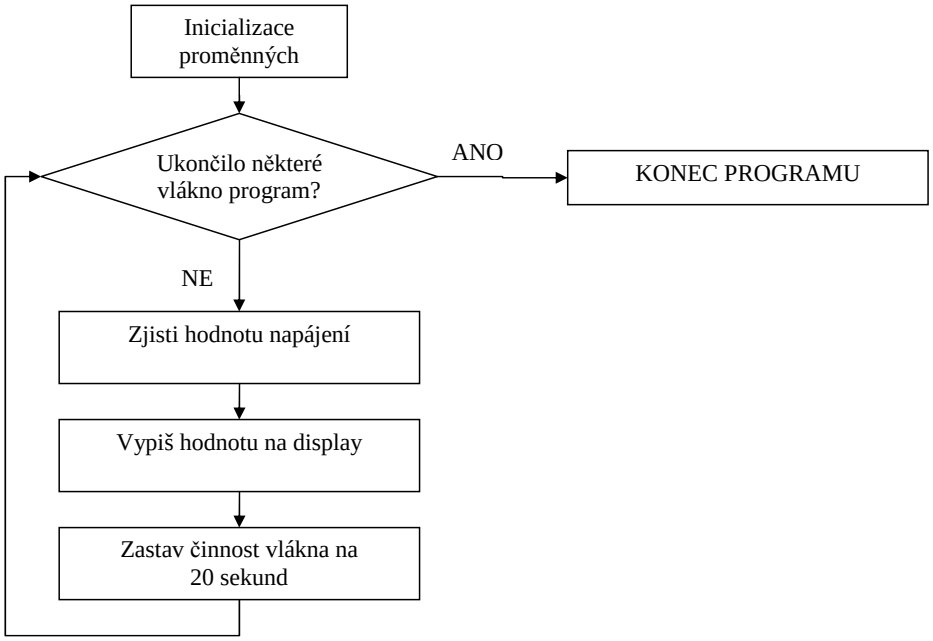
Obr. A.4: Diagram vlákna pro ovládání motorů

A.5 Diagram vlákna pro zjišťování údajů z ultrazvukových senzorů



Obr. A.5: Diagram vlákna pro zjišťování údajů z ultrazvukových senzorů

A.6 Diagram vlákna pro zjišťování hodnoty napájení



Obr. A.6: Diagram vlákna pro zjišťování hodnoty napájení