České vysoké učení technické v Praze
Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE

Rozšíření modelu podaktuovaného kráčejícího robota

Praha, 2012 **Autor: Tomáš Bukovský**

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 21. 5. 2012

podpis

i

Poděkování

Na těchto řádcích chci v první řadě poděkovat vedoucímu své práce Milanu Anderlemu za odborné vedení, mnoho cenných rad a ochotu obětovat cenný čas během konzultací. Dále děkuji svým rodičům, nejbližším a přátelům za podporu, jakou mi během celého studia poskytli.

Abstrakt

Bakalářská práce souvisí s experimentálním vývojem modelu podaktuovaného kráčejícího robota. Jejím obsahem je rozšíření stávající elektroniky na mechanickém modelu o dvě elektronické součástky, které sice byly na deskách osazené, ale nebyly dosud využívány. První z nich je externí paměť EEPROM pro ukládání a ladění konstant regulátorů nebo číslicových filtrů. Druhou elektronickou součástkou je hradlové pole, které zpracovává signál z inkrementálního senzoru, který je připojen na hřídel motoru v kloubu mechanického modelu. Pomocí inkrementálního senzoru je možné měřit relativní úhlovou polohu kloubu modelu. Práce obsahuje popis implementace těchto dvou dosud nevyužitých elektronických součástek. Poslední část práce se věnuje vytvoření grafického 3D modelu robota v prostředí Simulink 3D Animation. Toto prostředí umožňuje názorně zobrazit simulaci zpětnovazebního řízení chůze podaktuovaného kráčejícího robota.

Abstract

The bachelor thesis deals with an experimental development of an underactuated walking robot. The main aim of the thesis is the extension of electronic circuits attached to the mechanical model. The extension relates with an external memory and with a CPLD. The external memory will be used for storage of controller constant. The CPLD will be used for processing signals from incremental sensor. The incremental sensor is connected to the actuator and using this sensor is possible to measure relative angle between two links of the mechanical model. The last part of the bachelor thesis deals with 3D animation model in the Simulink 3D Animation environment. Using this environment is possible to show the simulation of feedback control of the underactuated mechanical system.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Tomáš Bukovský**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Rozšíření modelu podaktuovaného kráčejícího robota**

Pokyny pro vypracování:

1. Seznamte se se současným stavem modelu podaktuovaného kráčejícího robota.
2. Vyřešte zápis/čtení prostřednictvím dodaného vizuálního prostředí do/z EEPROM paměti 24LC64T. Zároveň upravte kód v mikroprocesoru, aby uměl data z externí paměti zpracovat.
3. Vyřešte počítání pulsů z inkrementálních čidel pomocí hradlového pole XC95144XL. Zároveň upravte kód v mikroprocesoru, aby uměl komunikovat s hradlovým polem.
4. Vytvořte simulační model ve VR toolboxu, kterým bude možno vytvářet animace z uložených dat z experimentů s reálným modelem.

Seznam odborné literatury:

[1] Westervelt E., Grizzle J., Chevallereau Ch., Choi J. & Morris, B. Feedback Control of Dynamic Bipedal Robot Locomotion. CRC Press, 2007.
[2] Chevallereau Ch., Bessonnet G., Abba G., & Aoustin Y. Bipedal Robots Modeling, Design and Walking Synthesis. WILEY, 2009
[3] Rott, O. Platforma pro řízení podaktuovaných mechanických systémů. Diplomová práce, Katedra řídicí techniky FEL ČVUT, 2012.

Vedoucí: Ing. Milan Anderle

Platnost zadání: do konce zimního semestru 2012/2013

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 26. 1. 2012

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Podaktuované mechanické systémy jsou mechanické systémy s méně pohony než stupni volnosti. Nejjednodušším mechanickým systémem je např. tzv. akrobot, který má dva stupně volnosti a jen jeden pohon umístěný ve spojení dvou článků. To znamená, že úhel v bodě, kde se akrobot dotýká země není možné ovládat. Model akrobota je možné ovládat tak, že jeho výsledný pohyb připomíná lidskou chůzi. Přidáním kloubů a pohonů do kolen vznikne z akrobota tzv. 4-link, jehož chůze je lidské chůzi velmi podobná.

I přesto, že se jedná o na první pohled velmi jednoduché mechanické systémy, jejich řízení není jednoduché a byla mu věnována velká pozornost v minulých desetiletích. Např. celá kniha *Feedback Control of Dynamic Bipedal Robot Locomotion* [2] pojednává o vývoji podaktuovaného kráčejícího robota jménem *Rabbit*. Historický přehled kráčejících robotů, nejnovější teorie v oblasti řízení plně aktuovaných i podaktuovaných kráčejících robotů, techniky modelování kráčejících robotů nebo jednotlivých částí chůze lze najít v [3]. Na katedře řídicí techniky se řízením podaktuovaných kráčejících robotů pomocí nelineárních metod aktivně zabývá skupina kolem profesora Čelikovského, viz např. [4], [5], [6], [7], [8].

Tato práce souvisí s experimentálním vývojem jednoduchého, ale dostačujícího podaktuovaného mechanického modelu kráčejícího robota. Jejím cílem je dokončit nedořešené části hardwarových úprav na reálném modelu kráčejícího robota a vytvořit pokročilé simulační prostředí pro simulace řízení kráčejícího robota pro výukové nebo prezentační účely.

Hardwarové úpravy začaly řešením diplomové práce Ondřeje Rotta, kdy na mechanický model byly připevněny desky s elektronikou umožňující model ovládat nebo měřit a zpracovávat potřebné veličiny. Hardware byl pro potřeby ověření teoretických výsledků dokončen a výsledky byly úspěšně ověřeny, ale z časových důvodů nebyly dvě elektronické součástky využívány. Jednou z nich je externí paměť pro ukládání konstant a jejich

snadnou změnu bez nutnosti přeprogramovávání mikroprocesoru, druhou součástkou je programovatelný logický obvod (CPLD) pro zpracovávání impulsů z inkrementálního čidla. Obě tyto součástky jsou nyní plně funkční a rozšiřují možnosti využití použitého hardwaru.

Nově vytvořený simulační model v prostředí Simulink 3D Animation bude používán při výuce pro názornější simulace zpětnovazebního řízení kráčejícího robota pomocí nelineárních metod. Dále může být model používán při prezentacích pro demonstraci přesnosti zpětnovazebního sledování nově vyvinutých metod.

1.1 Popis jednotlivých kapitol

Stav, ve kterém se mechanický model, elektronika a programové vybavení nacházely před započetím úprav je popsán v kapitole 2. Zároveň se zde nachází základní rozpis součástek osazených na deskách.

Kapitola 3 se zabývá zprovozněním EEPROM paměti. Paměťový čip byl na desky umístěn již při jejich osazování, ale nebyl nijak využit. V kapitole je popsán princip komunikace po sběrnici I^2C, následuje popis změn v programu desek a vysvětlení principu, jak se s konstantami pracuje. V poslední části je uveden popis grafické aplikaci (GUI), která je využívána k nastavování těchto konstant.

Kapitola 4 je věnována zpracování signálu z inkrementálního senzoru. Tento senzor je v současnosti součástí motorů pohybujících kyčlemi robota. Pro zpracování dat ze senzoru byly desky osazeny obvodem CPLD. Kapitola obsahuje popis zpracování dat a realizaci přenosu hodnot mezi CPLD a mikrokotrolérem.

Kapitola 5 se zabývá 3D grafickým modelem robota. Je zde představeno prostředí Simulink 3D Animation, základní popis virtuálních scén a popsán postup, jak byla scéna vytvořena.

V poslední kapitole 6 je uvedeno shrnutí výsledků mé práce.

Kapitola 2

Původní stav

Obsahem této kapitoly je základní popis modelu acrobota a jeho podoby před započetím úprav řešených v rámci této bakalářské práce. Ta svým obsahem navazuje na diplomovou práci Onřeje Rotta [1] a dále rozšiřuje možnosti modelu robota. Kapitola se věnuje popisu mechnické části robota, řídicích desek umístěných v článcích robota a softwarové části, která zahrnuje program pro řídicí desky a grafický program pro PC, který umožňuje zobrazovat hodnoty ze snímačů.

2.1 Mechanický popis robota

Podaktuovaný[1] mechanický model je tvořen celkem čtyřmi klouby a čtyřmi články, kde dva klouby představují kyčle a dva klouby představují kolena robota. Klouby v kyčlích, tedy horní klouby, jsou spojeny v jedné ose. Koncepce mechanického modelu umožňuje jeho řízení jako tzv. Acrobota, neboli dvounohého kráčejícího robota nebo je možné řídit model jako tzv. 4-link, neboli dvounohého kráčejícího robota s ohybem v kolenou. Pokud je model řízen jako tzv. Acrobot, kolena jsou celou dobu napnutá. Pouze v okamžiku míjení nohou dojde k mírnému pokrčení kolena pohybující se nohy, aby nedošlo k nárazu jejího konce do země.

V kloubech kyčlí je umístěn stejnosměrný motor *Maxon A-max 22* sloužící k ovládání modelu robota. Jeho součástí je zároveň i inkrementální senzor, který umožňuje měřit relativní polohu. Dále je v kloubech umístěn rotační potenciometr, který měří absolutní

[1] Podaktuovanost v našem případě znamená, že model nemá chodidla a není tedy možné žádným přímým způsobem řídit úhel mezi zemí a stojnou nohou.

polohu. Klouby v kolenou jsou vybaveny motory *Maxon Re-max 21*. Tyto motory se odlišují absencí inkrementálních senzorů.[2] Spodní články jsou vybaveny mechanickými spínači, jež umožňují detekovat kontakt spodního článku s horním článkem chodidla. Na každém článku robota je umístěna jedna řídicí deska s procesorem. Tato deska je popsána v následující podkapitole. Mechanický model je na obr. 2.1.

Obrázek 2.1: Mechanický model acrobota

2.2 Popis řídicí desky

V této podkapitole je uveden popis elektroniky modelu robota. Vyjma rotačních potenciometrů, které měří polohu kloubů, je elektronika umístěna na čtyřech řídicích deskách, přičemž všechny tyto desky jsou naprosto shodné. Každá tato deska je umístěna na jednom ze čtyř článků robota. Úkolem desek je měření potřebných veličin a ovládání motorů. Hlavní řídicí program běží v PC. Desky na robotovi měří potřebné veličiny a následně je vysílají po sběrnici CAN do PC. Poté PC posílá zpět hodnoty pro řízení motorů v kloubech. Nyní zde uvedu základní popis součástí umístěných na deskách, přičemž se zaměřím především na části, jichž se přímo týká tato bakalářská práce.

[2] Do budoucna se počítá s výměnou motorů v kolenou za jiné, které již budou doplněny inkrementálním senzorem

2.2.1 Mikrokontrolér LPC2368

Jedná se o 32 bitový RISCový mikrokontrolér architektury typu ARM7. Obsahuje řídicí program, který čte naměřené hodnoty ze senzorů. Tyto hodnoty předzpracovává, vysílá do PC a na základě přijatých dat z PC počítá akční zásah pro motory, který pomocí PWM výstupu na motory aplikuje. Podrobnější popis úlohy řídicího programu je uveden v samostatné podkapitole o software. Ke čtení dat z gyroskopu je využíváno rozhraní SPI, proud protékající motory, vzdálenost od laserového senzoru a úhel v kloubech měřený potenciometry je zpracováván pomocí integrovaných A/D převodníků. Dále mikrokontrolér obsahuje 2 CAN kanály, přičemž jeden z nich je využíván ke komunikaci s PC a ostatními deskami.

Základní parametry mikrokontroléru

- Frekvence až 72 MHz, využíván vnitřní oscilátor 4 MHz
- 512 kB Flash ROM paměti
- celkem 58 kB RAM
- celkem 6 10-bitových A/D převodníků
- PWM výstup

2.2.2 CPLD Xilinx XC95144XL

Jak již bylo řečeno v kapitole o mechanické části modelu, instalované motory v kyčlích obsahují inkrementální senzor polohy. Tento senzor umožňuje změřit jednak relativní změnu polohy a jednak úhlovou rychlost motorů, potažmo kloubů. Mohou tak být vhodným doplňkem k rotačnímu potenciometru, jehož úkolem je měřit absolutní polohu kloubů.

Pokud bychom zapojili tento senzor přímo na některý vstup mikrokontroléru a obsluhovali bychom jej například pomocí přerušení, mikrokontrolér by se převážnou část výpočetního času zabýval jeho obsluhou. Aby se tomuto předešlo, byla deska zároveň osazena tímto CPLD obvodem. Jeho úkolem je pomocí čítače předzpracovávat signál ze snímače a hodnotu o poloze poté předávat mikrokontroléru. K němu je připojena pomocí 8-bitové paralelní sběrnice.

V původním stavu již byly desky osazeny tímto CPLD. Nicméně nebylo naprogramováno, ani nebylo k ničemu jinému využíváno. Součástí bakalářské práce je právě vyřešení využití tohoto obvodu k získání relativní polohy z inkrementálního senzoru.

2.2.3 Paměť EEPROM Microchip 24LC02B

Program nahraný v mikrokontroléru využívá konstanty a parametry důležité pro jeho chod (např. konstanty PID regulátorů). Elektronický návrh modelu robota počítá s tím, že tyto konstanty bude možné měnit pomocí grafického uživatelského rozhraní (dále GUI) a navíc tyto konstanty zůstanou zachovány i po vypnutí napájecího napájení.

Proto byla deska osazena touto externí EEPROM pamětí o kapacitě 2 kB. Paměť je organizována jako jeden blok o 256 bytech (tedy 1 x 256 x 8 bitů). K mikrokontroléru je připojena pomocí sběrnice I^2C.

V podobě před započetím úprav však nebyla externí paměť využívána. Konstanty byly napevno kompilovány spolu s celým programem a jediná možnost změny byla přeprogramovávat celý mikrokontrolér novou kompilací programu. Vyřešení zápisu do EEPROM paměti je také součástí mé bakalářské práce.

2.2.4 Senzory a ostatní součástky

Gyroskop ANALOG DEVICES ADIS16260 - využívá se pro měření úhlových rychlostí jednotlivých článků. Gyroskop je jednoosý, tj. měří rychlost pouze v jednom směru, v našem případě kolmém na plochu čipu. Pracuje na principu Coriolisovy síly a je vyroben technologií MEMS. Komunikace s mikrokontrolérem probíhá přes rozhraní SPI.

Rotační potenciometr - slouží pro měření absolutní polohy kloubů. Jsou zapojeny jako děliče napětí. Na potenciometr je přivedeno napětí $3,3\,\mathrm{V}$. Výstup v rozsahu $0 - 3,3\,\mathrm{V}$ je přiveden na A/D převodník mikrokontroléru. Pro omezení šumu je ještě mezi potenciometrem a převodníkem zařazen RC filtr typu dolní propust.

Rozdílový zesilovač Analog Devices AD8210 - řídicí program pro ovládání motorů obsahuje i proudovou zpětnou vazbu. Proto je potřeba měřit proud odebíraný motory. Proud je zároveň i vhodné znát z důvodu ochrany před přetížením motoru.

Proto je v sérii s napájením motorů zařazen i malý rezistor. Úbytek napětí na rezistoru je měřen právě pomocí tohoto integrovaného obvodu. Analogový výstup obvodu je přímo úměrný proudu motorem. Tento výstup je zapojen na jeden z A/D převodníků mikrokontroléru.

Napájecí stabilizátory - návrh desky počítá s extérním napájecím napětím 15 V, které je stabilizováno pomocí obvodu LM2576HV. Protože však deska obsahuje součástky, které vyžadují různá napájecí napětí (3,3 V - mikrokontrolér a CPLD, 5 V - gyroskop a měřič proudu (rozdílový zesilovač), 12 V - H-můstek, 24 V - laserový měřič vzdálenosti), je osazena soustavou dalších DC/DC měničů.

Budič CAN PCA82C250 - komunikace mezi jednotlivými deskami a PC probíhá po sběrnici CAN. Tento budič slouží jako rozhraní mezi mikrokontrolérem a CAN sběrnicí.

H-můstek L6201 - využívá se k výkonovému spínání motorů. Je ovládán řídicími PWM signály z mikrokontroléru.

2.3 Software

Koncepce řízení robota je založena na dvou vzájemně nezávislých a oddělených částech. První část zahrnuje algoritmus pro řízení chůze robota. Jeho součástí je pozorovatel stavů a nelineární regulátor. Vzhledem ke své výpočetní náročnosti je tato část realizována v PC v Matlabu, přičemž s jednotlivými deskami na robotovi komunikuje pomocí sběrnice CAN. Matlab pomocí sběrnice přijímá potřebná data (polohy kloubů a úhlové rychlosti). Tato data poté zpracovává a odesílá zpět točivý moment, který mají vyvinout motory v horních kloubech robota. Princip řízení robota je rozepsán v [1].

Druhá část algoritmu je hierarchicky na nižší úrovni. Je realizována v mikrokontrolérech desek a obsahuje regulační smyčky pro řízení jednotlivých kloubů. Zatímco algoritmus v Matlabu počítá potřebné momenty motorů na základě matematického modelu robota, algoritmus v horních deskách přímo řídí tento moment pomocí proudových PID regulátorů. Program ve spodních deskách je velmi podobný. Jeho úkolem je ovládat dolní klouby. Je v nich implementován regulátor polohy, který pokrčuje nohy při chůzi. Naopak nohu, o kterou se robot momentálně opírá, udržuje propnutou. Podrobnější popis tohoto programu je obsažen v následující podkapitole.

2.3.1 Program v řídicích deskách robota

Jak již bylo řečeno, program je ve všech deskách velmi podobný, proto je popis pro všechny společný. Program se odlišuje v rozdílných implementovaných regulátorech a identifikátorech CAN zpráv. Všechny desky čtou polohy kloubů z rotačních potenciometrů,

proudy tekoucí do motorů a úhlové rychlosti z gyroskopu. Na pravém spodním článku robota je navíc instalován laserový senzor vzdálenosti. Program pravé spodní desky je tedy navíc rozšířen o zpracování dat z tohoto senzoru.

Po spuštění programu se jako první provede inicializace vstupů a výstupů mikrokontroléru pomocí registrů PINSEL. Inicializuje se CAN sběrnice a zaregistruje se metoda pro obsluhu příchozích zpráv. Poté se provede inicializace SPI sběrnice pro komunikaci s gyroskopem. Mikrokontrolér je nastaven jako master, tj. z gyroskopu mu jsou zasílána na jeho vyžádání. Dále se inicializuje PWM výstup pro ovládání motorů. Provede se zaregistrování obsluhy přerušení způsobené sepnutím mechanického spínače ve spodním článku robota.

Po dokončení inicializace začne program počítat hlavní smyčku. Ta se opakuje s frekvencí 200 Hz. V první fázi se čtou data ze senzorů. Jsou vyžádány hodnoty z gyroskopu a přečteny hodnoty z A/D převodníku použitého pro měření polohy pomocí potenciometru. Čtení z A/D převodníku, který převádí měřený proud, je řešeno nikoliv v hlavní smyčce, ale pomocí přerušení každých 10 ms.

Přečtené hodnoty ze senzorů se následně vyšlou na sběrnici CAN. Její vytížení činí, při uvažované frekvenci hlavní smyčky 200 Hz, asi 10%.

V poslední části programu se počítá akční zásah pro příslušný motor pomocí PID regulátoru. Tyto akční zásahy se počítají s frekvencí 100 Hz.

2.3.2 GUI

Součástí programového vybavení robota je také grafické uživatelské prostředí. Jeho úkolem je především grafické zobrazení měřených veličin, možnost změny konstant regulátorů a jejich uložení na EEPROM v deskách robota. Řízení chůze robota není součástí této aplikace. To je realizováno výhradně pomocí Matlabu.

Aplikace byla vyvinuta v programovacím jazyku C#. Byla převzata od týmu pracujícím na projektu Stabilizované kamerové základny a upravena pro účely práce s robotem. Součástí těchto úprav však ještě nebyla provedena implementace kódu umožňující uložení konstant regulátorů do paměti EEPROM a i jejich vyvolání. Vyřešení tohoto úkolu je součástí této bakalářské práce.

Kapitola 3

Rozšíření o EEPROM

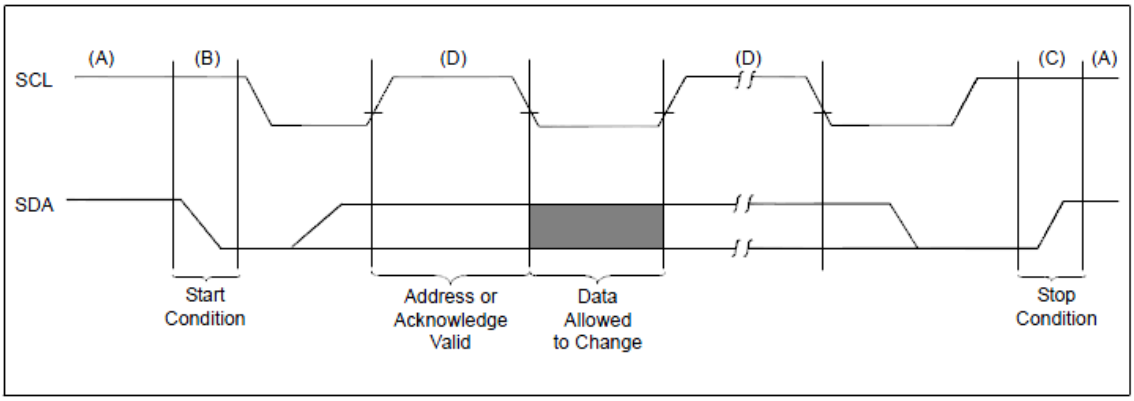
V této kapitole je popsáno rozšíření modelu robota o externí paměť EEPROM. Úlohou této paměti je uchovávat konstanty a parametry řídicího programu. U těchto konstant se předpokládá, že bude čas od času požadována jejich změna, ať už z důvodu ladění nebo z důvodu prováděných pokusů na robotovi v rámci výuky. S tím se počítalo již při návrhu řídicí desky, ta proto byla osazeno obvodem *Microchip 24LC02B*, jež disponuje pamětí o velikosti celkem 2 Kbit. Po vypnutí napájení a opětovném zapnutí je tak možné vyvolat konstanty z EEPROM do operační paměti mikrokontroléru. Pokud by desky nebyly vybaveny touto EEPROM pamětí, musela by být jakákoliv trvalá změna konstant řešena pomocí překompilace programu do flash paměti mikrokontroléru. Aby bylo možné měnit požadované konstanty jak v běžícím programu, tak je zároveň i zálohovat do EEPROM, využívá se grafická aplikace (dále GUI) v PC. Přenos konstant mezi PC a mikrokontrolérem probíhá, podobně jako ostatní komunikace, pomocí sběrnice CAN. V následujících podkapitolách je popsán princip, jakým spolu paměť s mikrokontrolérem komunikuje a s tím související popis mého řešení implementace v programu. Implementace změn do GUI je popsána v podkapitole 3.4.

3.1 Komunikace s EEPROM

Použitá paměť využívá sběrnici I^2C. Výhodou sběrnice je, že pro obousměrnou komunikaci potřebuje pouze dva vodiče, jeden hodinový a jeden datový, označované jako *SCL* a *SDA*. Použitý mikrokontrolér obsahuje 3 samostatné řadiče sběrnice. Paměť je připojená na řadič označený pořadovým číslem nula. Zařízení připojená ke sběrnici mohou být

typu master nebo slave. Přičemž úlohou zařízení typu master je generovat hodinový signál, vysílat start a stop bity a určovat režim činnosti zařízení typu slave. V tomto případě je zařízením typu master mikrokontrolér. Obě zařízení však mohou komunikovat jak v režimu čtení, tak v režimu zápisu. Frekvence hodinového signálu může být až 400 KHz.

Pokud po sběrnici neprobíhá žádná komunikace, jsou oba vodiče implicitně v log. 1. To je zajištěno pomocí pull-up rezistorů připojených na napájecích 3,3 V. Začátek komunikace je označen start bitem na datovém vodiči, konec komunikace stop bitem. Poté začne zařízení typu master generovat hodinový signál. Situace je znázorněna na obr. 3.1. Následuje přenos dat, který je vždy složen z bloku 8 bitů, tedy jednoho bytu. Devátý bit je vždy potvrzení správného přijmutí dat (acknowledge), které odesílá přijímající strana. Byla-li zpráva přijata v pořádku, je odeslána log. 0. Poté buď následuje přenos dalšího bytu, nebo zařízení typu master vyšle stop bit a komunikace je ukončena.



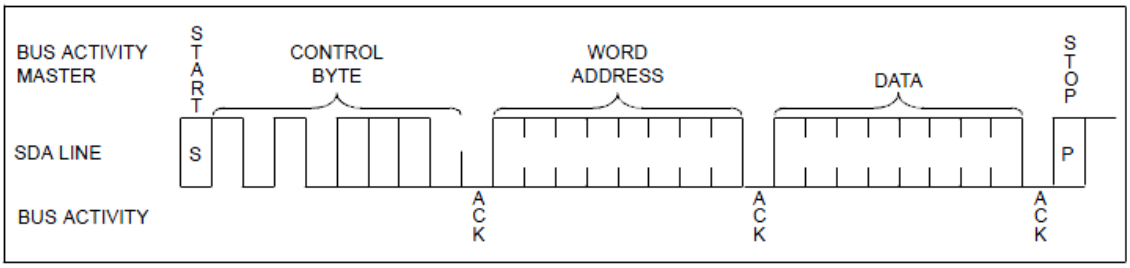
Obrázek 3.1: Komunikace po sběrnici I^2C (zdroj: [10])

U námi využitého EEPROM čipu *24LC02B* po start bitu vždy následuje příjem řídicího bytu. Jeho první 4 bity jsou vždy „1010“, na dalších třech bitech nezáleží. Poslední bit určuje, zda paměť bude pracovat v režimu čtení (1), nebo zápisu (0). Zvolíme-li režim zápisu, další odeslaný byte určuje adresu, kam bude zapisováno (popřípadě čteno - viz níže). Jelikož je naše paměť organizována jako 256 x 8 bitů, lze takto adresovat jednotlivé byty.

Chceme-li data zapisovat, tak mikrokontrolér již může v dalším bytu zasílat data (obr. 3.2). Data je možné posílat buď po jednom bytu, nebo jako tzv. stránky[1]. Jelikož

[1]Jako stránka je zde uvažováno 8 po sobě následujících bytů. Zápis probíhá sekvenčně od adresy, která byla nastavena před začátkem přenosu

pro všechny naše ukládané parametry platí, že jsou typu float a velikost je tedy 4 byty, opakuji přenos 4 krát po 1 bytu.



Obrázek 3.2: Zápis dat do EEPROM (zdroj: [10])

Chceme-li data číst, tak je po odeslání adresy ještě znovu potřeba odeslat start bit a řídicí byte, ve kterým tentokrát nastavíme režim čtení. Poté již paměť odešle data z požadované adresy. Poté je možné přenos ukončit tak, že příjemce odešle negativní signál acknowlege a poté stop bitem ukončí komunikaci. Nebo odešle pozitivní acknowledge, přičemž následně mu jsou odeslána data z následující adresy. Tento postup lze opakovat, dokud příjemce neodešle negativní acknowlegde a stop bit. Tímto způsobem získávám 4 po sobě následující byty, které představují jeden parametr.

3.2 Implementace v programu

Při softwarové implementaci EEPROM paměti jsem vycházel z návrhu, který byl vytvořen týmem vyvíjející *stabilizovanou kamerovou základnu*. Tento návrh je již ověřený, proto jsem jej, s ohledem na potřebné změny, využil i v řídicích deskách robota. V této podkapitole popíšu, jak program s parametry pracuje a jakým způsobem je realizována komunikace s EEPROM čipem a s grafickou aplikací v PC.

Je vhodné uvést, že použitý mikrokontrolér se programuje v jazyku C. Původní část programu byla vyvinuta pomocí nekomerčního nástroje YAGARTO (Yet another GNU ARM toolchain), který jako vývojové prostředí využívá Eclipse. Postup, jak nainstalovat YAGARTO, nastavit parametry, zkompilovat program a naprogramovat jej do mikrokontroléru je uveden v přiloženém souboru *winarm.txt* umístěném ve složce yagarto na přiloženém CD. Postup kompilace je také popsán v [1].

3.2.1 Teoretický rozbor

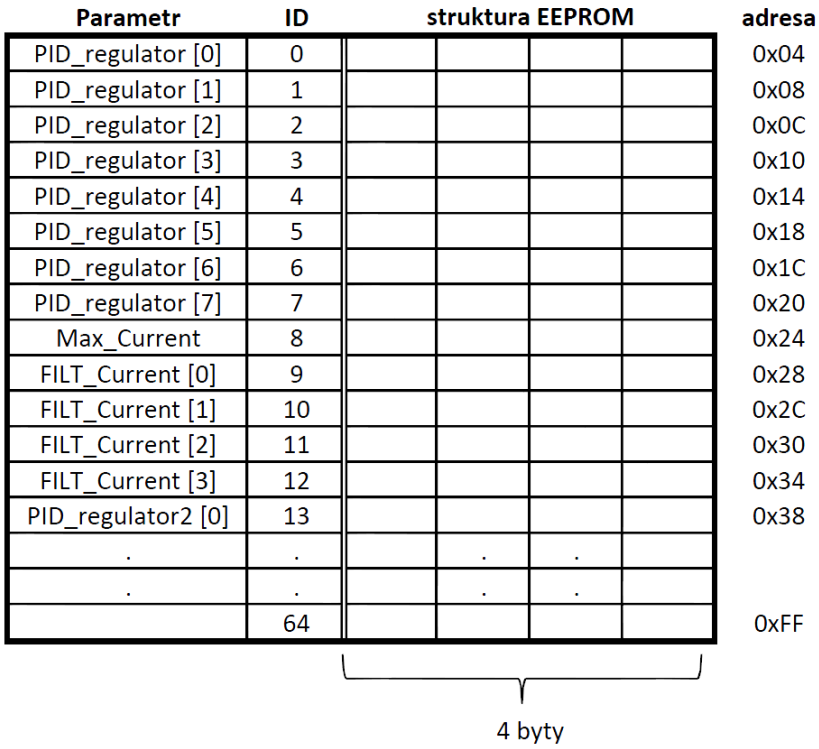
Jak již bylo zmíněno, základní úlohou EEPROM je uchování parametrů po dobu, kdy je vypnuto napájení. Typickým příkladem takových parametrů jsou konstanty PID regulátorů. Tyto konstanty jsou v běžícím programu uloženy v datovém typu *float*[2]. Aby byla práce s EEPROM co nejjednodušší, koncepce předpokládá, že všechny parametry obsadí prostor o shodné velikosti. Nejlepší varianta se zde zdá být využít buňky o velikosti 4 byty, protože kromě typu *float* je využívá i standartní *integer*. Rozdělíme-li tedy paměť o celkové velikosti 256 bytů na buňky o velikosti 4 byty, získáme celkem 64 buněk. Každá buňka může přitom pojmout jeden parametr. EEPROM nám tak umožní uchovat až 64 parametrů.

Je důležité zajistit vhodný způsob adresování, aby nedocházelo k vzájemnému přemazávání parametry mezi sebou. Proto je každému parametru přidělen unikátní číselný **index** v rozsahu 0 až 63. Indexy parametrů musí být definovány v souboru *parameters_nazev_desky.h*. Bližší informace o tomto souboru je uvedena v podkapitole 3.3. Program je s těmito indexy seznámen a při práci s parametry je využívá k adresování. Vynásobíme-li si totiž index číslem 4, získáme přesnou adresu, která ukazuje na odpovídající buňku v paměti. Uspořádání paměti je schématicky znázorněno na obr. 3.3. Jeden řádek v tabulce představuje jednu buňku paměti a potažmo i jeden parametr. Parametr je definován názvem a svým indexem (ID). Každý parametr zaujímá v paměti místo o velikosti 4 byty.

Protože úkolem EEPROM je pouze uchovávat konstanty po dobu, kdy je vypnuto napájení, je potřeba zároveň udržovat parametry ve vhodném formátu i v paměti RAM. I zde se využije již zavedených indexů. Parametry se ukládají do datového pole typu *char*. Velikost pole je 4 x *„počet parametrů“*. Každý parametr je tak nezávisle na vlastním datovém typu uložen v tomto poli typu *char*. S parametry v tomto poli lze snadno pracovat po jednotlivých bytech, což je výhodné jak při přenosu do EEPROM, tak při přenosu po sběrnici CAN. Ve zdrojovém kódu je pole s konstantami pojmenováno jako *constants_table[]*.

Pro zlepšení jednoduchosti a přehlednosti kódu využívá řídicí program uživatelsky definované struktury pro PID regulátory, respektive filtry. Tyto struktury jsou složeny z 8, respektive 4 samostatných parametrů. Dojde-li ke změně parametrů, je třeba zajistit, aby se hodnota kromě pole aktualizovala i v těchto strukturách.

[2]Velikost datového typu *float* je 4 byty



Obrázek 3.3: Struktura EEPROM paměti

3.2.2 Program

V této podkapitole blíže přiblížím realizaci práce s parametry a EEPROM v samotném kódu. Veškerý kód pro obsluhu EEPROM je uložen v knihovně *lib_i2c.c*

Po spuštění programu je nejprve potřeba inicializovat sběrnici I^2C. Pomocí registru *PINSEL1* se porty 0.27 a 0.28, ke kterým je EEPROM čip fyzicky připojen, uvedou do režimu činnosti práce s řadičem sběrnice I^2C. Poté se spustí funkce *I2C_init*. V té se nejprve vynulují všechny bity řídícího registru řadiče, čímž se i prozatím deaktivuje jeho funkce. Poté se nastaví frekvence komunikace na 350 KHz. Maximum EEPROM čipu je 400 KHz. Nakonec se zaregistruje funkce pro obsluhu přerušení vyvolaných řadičem. Tím je základní inicializace dokončena.

Poté se v hlavním programu provede funkce *load_all_params()*. Jejím úkolem je vyvolat všechny parametry z EEPROM a uložit je do již zmíněného pole *constants_table[]*. Parametry jsou vyvolávány postupně jeden po druhém. K přečtení jednoho parametru slouží funkce *read_flash_seq()*. Této funkci se v argumentu předává adresa, ze které budou data z EEPROM čteny. Tato adresa se počítá jako index parametru vynásobený čtyřmi.

Následně se zápisem do řídícího registru nejprve aktivuje funkce řadiče a následně nastaví požadavek o zaslání start bitu. Ten nemusí být odeslán okamžitě. Pokud by bylo ke sběrnici připojeno více zařízení, mohla by být obsazena. Proto je přenos ovládán pomocí přerušení, které vyvolává řadič.

Jakmile je start bit odeslán, spustí se přerušení, ve kterém je již možné po sběrnici odeslat první byte. Data, která se mají odeslat, se uloží do registru *I20DAT*. V tomto případě se jedná o řídící byte, který byl popsán v kapitole 3.1. Jakmile je přijat *acknowledge* bit, potvrzující úspěšný příjem bytu, je opět vyvoláno přerušení. V dalším bytu dat se do EEPROM odešle adresa, ze které budou požadovaná data čtena. V následujícím přerušení se odešle opět start bit a řídící byte, kterým tentokrát určíme, že data budou z EEPROM čtena. V dalším přerušení je již přijat první byte hodnot z požadované adresy. Hodnoty si lze vyzvednout v registru *I20DAT*. Úpravou registru *I20CONSET* potvrdíme příjem odesláním bitu *acknowledge*. Na to EEPROM zareaguje odesláním dalšího bytu dat. Přečtením postupně 4 bytů získáme celý parametr. Jakmile přijmeme poslední z nich, je odeslán negativní *acknowledge* následovaný stop bitem. Celá funkce končí opětovnou deaktivací řadiče sběrnice I^2C.

Tímto jsme z EEPROM přečetli jeden parametr a uložili ho do pole *constants_table[]*. Opakovaným voláním funkce *read_flash_seq()* dojde postupně k přečtení všech parametrů. Pokud bychom místo čtení chtěli zapisovat, princip komunikace je podobný. Jen zde není možný sekvenční zápis 4 následujících bytů. Každý byte se proto musí nejprve adresovat a poté teprve odeslat.

Nakonec je ještě potřeba inicializovat datové struktury, které program používá pro PID regulátory a filtry. To provedeme spuštěním funkce *parameters_init*.

3.3 Postup přidání dalšího parametru

V této podkapitole popíšu jednoduchý postup, jak do programu přidat další parametr, který bude využívat možnosti ukládání do EEPROM.

V první řadě je potřeba upravit konfigurační soubor *parameters_nazev_desky.h* ve složce *lib*. V tomto souboru jsou nadefinovány veškeré parametry a jejich indexy. Struktura souboru je přibližně následující:

```
/*! \brief Board ID definition */
#define BOARD_NAME "Right up"
#define BOARD_IDENT 1

/*! \brief Count of parameters avaible for storage, all of these parameters
 are loaded using load\textunderscore all_params(); */
#define PARAM_COUNT 63

/*! \brief PID controller 1*/
#define PIDregulator 0 //PID8

/*! \brief current limit*/
#define Max_current constant_table[8].f

/*! \brief Overcurrent protection filter */
#define FILTcurrent 9 //FILT4

/*! \brief PID controller 2*/
#define PIDregulator 13 //PID8
```

Nyní blíže popíšu význam jednotlivých řádků:

```
#define BOARD_NAME "Right up"
#define BOARD_IDENT 1
```

Zde se nadefinuje název a libovolný unikátní identifikátor konkrétní desky. Jelikož je desek více, mají tyto parametry smysl především pro grafickou aplikaci. Název usnadňuje uživateli určení desky, ve které si přeje parametry měnit. Identifikátor umožňuje rozlišit jednotlivé desky během komunikace po CAN.

```
#define PARAM_COUNT 63
```

Tento řádek určuje maximální počet parametrů. Hodnotu je možné nastavit v rozmezí od 0 do 63. Pokud bude hodnota nižší než 63, mikrokontrolér bude část paměti EEPROM ignorovat a nevyužije ji.

```
#define PIDregulator 0
```

Zde již definujeme první parametr. Nejdříve je název, následuje index. V tomto případě se jedná o PID regulátor. To je důležité rozlišit uvedením klíčové fráze *PID* v jeho názvu. Grafická aplikace totiž předpokládá, že PID konstanty vždy obsadí celkem 8 parametrů. Proto tomuto PID regulátoru přísluší nejen index 0, ale i 7 následujících indexů. Situace je znázorněna na obr. 3.3.

```
#define Max_current constant_table[8].f
```

Tento řádek je příkladem definice obecného parametru. Číslo v hranaté závorce představuje index parametru. Jelikož předcházející PID regulátor obsadil indexy 0 až 7, musí mít tento parametr index 8 (nebo více). Písmena za tečkou určuje datový typ parametru. Ty mohou být následující: f – float, i8 – integer 8 bit, i16 – integer 16 bit, i32 – integer 32 bit a dále varianty u8,u16,u32 představující neznaménkový integer.

```
#define FILTcurrent 9
```

Podobně jako s PID regulátory i s filtry pracuje grafická aplikace specificky. Zde se předpokládá, že se jedná o filtr prvního řádu, který je charakterizován celkem 4 konstantami. V paměti tedy využije celkem 4 parametry s indexy 9 až 12. Filtry rozlišíme uvedením klíčové fráze *FILT* v jejich názvu.

Poslední řádek v ukázkovém kódu je již jen definice dalšího PID regulátoru.

Pokud se mezi našimi parametry nachází PID regulátory nebo filtry, je ještě potřeba přidat do zdrojového kódu inicializaci jejich struktur. Inicializace se nacházejí ve funkci *parameters_init()*.

```
void parameters_init(void){
PID_init_EEPROM(PIDcurr_control,&curr_control);
PID_init_EEPROM(PIDregulator,&regulator);
PID_init_EEPROM(PIDsinus_control,&sinus_control);
Filter_init_EEPROM(FILTcurrent,& current_filter);
Filter_init_EEPROM(FILTangle,&angle_filter);
}
```

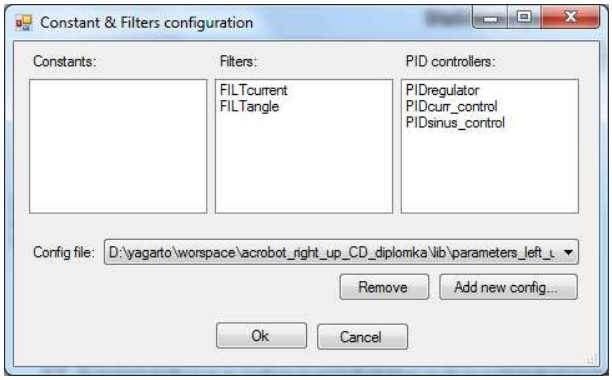
Uvnitř funkce se volají 2 typy jiných funkcí. Jedna (*PID_init_EEPROM()*) inicializuje PID regulátor, druhá (*Filter_init_EEPROM*) inicializuje filtry. Obě funkce mají 2 atributy. Prvním je index parametru, který byl nadefinován v konfiguračním souboru. Zde lze využít definice preprocesoru, kterou jsme použili u konfiguračního souboru. V takovém případě do atributu stačí vložit název parametru. Druhým atributem je reference na inicializovanou strukturu. Struktura musí být nadeklarována v hlavičce zdrojového kódu.

3.4 Implementace v GUI

Grafická aplikace, kterou jsem dostal k dispozici, již umožňovala změnu parametrů v paměti desek a následně i jejich uložení do EEPROM. Aplikaci je možné použít po přizpůsobení konfiguračním souborem, který jsme vytvořili v kapitole 3.1.

3.4.1 Uživatelské prostředí

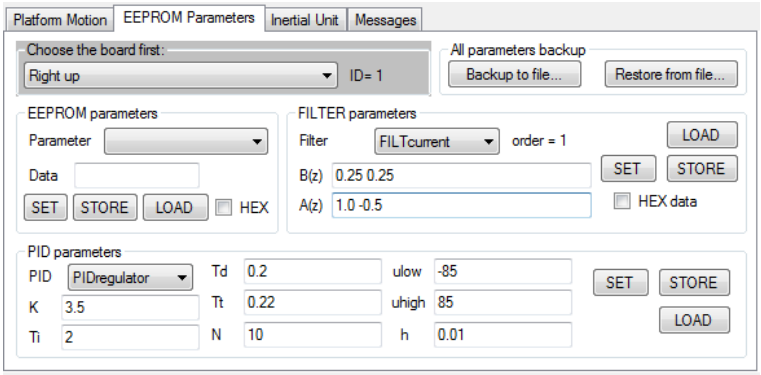
Po spuštění aplikace je potřeba zvolit jeden z pracovních režimů. Mezi režimy lze přepínat výběrovým polem nahoře vlevo *CAN/FILE*. Režim *FILE* slouží k offline zobrazení naměřených dat. Pro práci s EEPROM tedy zvolíme režim *CAN*. Ještě před stistknutím tlačítka *Initialize* je vhodné se v menu *Configuration* → *Can Bus* přesvědčit, že je rychlost komunikace nastavena na nejvyšší možnou, tedy 1 Mb/s. Poté je možné v menu *Configuration* → *Constants & Filters...* nadefinovat cestu k jednotlivým konfiguračním souborům obr. 3.4. Soubory by měli být celkem 4, pro každou desku jeden. Pokud je soubor načten v pořádku, v okně se zobrazí seznam všech parametrů. Parametry jsou roztříděny do skupin podle toho, zda se jedná o obecné parametry, filtry, nebo PID regulátory. Máme-li načteny všechny konfigurační soubory, výběr potvrdíme tlačítkem *OK*. Pokud by jsme nyní aplikaci uzavřeli, cesty ke konfiguračním souborům by zůstali zachovány. Ještě je nutné zmínit, že je bezpodmínečně nutné, aby konfigurační soubory byly totožné s těmi, které byly použity při kompilaci programu desek.



Obrázek 3.4: Nastavení konfiguračních souborů

Poté lze již na kartě EEPROM Parameters obr. 3.5 s nadefinovanými parametry pracovat. Ve výběrovém menu *Choose the board first* se vybere příslušná deska. Výběr desek odpovídá počtu načtených konfiguračních souborů. Název, který jsme zvolili v atributu *BOARD_NAME* v konfiguračním souboru se zobrazí právě na tomto místě. Jakmile vybereme konkrétní desku, zpřístupní se volby pro výběr parametrů, filtrů a PID regulátorů. Jakmile si některý zvolíme, je již možné nastavovat hodnoty v jednotlivých polích. Pro komunikaci s mikrokontrolérem tu jsou k dispozici 3 tlačítka. Tlačítko *SET* odešle zadané hodnoty do jeho operační paměti. Tlačítko *LOAD* naopak odešle požadavek o zaslání hodnot z operační paměti. Mikrokontrolér mu je poté v odpovědi pošle. Tlačítko *STORE*

umožňuje uložit hodnoty do EEPROM.



Obrázek 3.5: Karta s EEPROM parametry

3.4.2 Komunikace po CAN

Přenos parametrů dat mezi deskami a PC je realizován pomocí sběrnice CAN. Kromě parametrů se sběrnice využívá ještě k přenosu hodnot ze senzorů a akčních zásahů. Mikrokontrolér má v sobě již integrovaný řadič sběrnice CAN. Na straně PC je situace komplikovanější. Je nutné využít některý převodník CAN - USB. Já jsem k tomuto účelu dostal od vedoucího práce k zapůjčení převodník *Kvaser Leaf Light HS*.

V každé CAN zprávě můžeme přenést až 8 bytů dat. Kromě datových bytů je součástí každé zprávy také její identifikátor. Identifikátor určuje odesílající strana. Příjemci umožňuje identifikovat typ zprávy a vyvolat jí příslušející obsluhu. Seznam identifikátorů souvisejících s přenosem parametrů je uveden v tab. 3.1

Tabulka 3.1: Seznam zpráv sběrnice CAN

ID (hex)	ID (dec)	zdroj zprávy	popis
0x230	560	PC	nastavení nových parametrů pro konkrétní desku
0x235	565	PC	požzadavek na čtení parametrů konkrétní desky
0x236	566	všechny desky	vyslání parametru z desky do PC

Struktura CAN zpráv jednotlivých příkazů *SET*, *STORE* a *LOAD* je vždy velmi podobná. V každé CAN zprávě můžeme přenést až 8 bytů dat. Jeden byte je využit pro identifikaci cílové/zdrojové desky. Další 2 byty jsou potřeba pro identifikaci parametru (jeho index). Pro data zbývá už maximálně 5 bytů. Proto platí, že každá zpráva přenáší

jeden parametr o velikosti 4 byty. Pro přenos PID konstant se tak např. posílá celkem 8 zpráv.

Pokud uživatel zažádá o zaslání parametru z RAM (příkaz *LOAD*), je odeslán pouze požadavek s indexem parametru. Jakmile mikrokontrolér zprávu příjme, odešle zpět zprávu s požadovým parametrem. Příjem zprávy je řešen pomocí přerušení ve funkci *parse_not_graph_data()*. V jeho obsluze jsou hodnoty z CAN zprávy umístěny do odpovídajících textových polí ve formuláři.

Kapitola 4

Zpracování signálu z inkrementálního senzoru

Tato kapitola je věnována zprovoznění inkrementálního snímače, který umožňuje měřit relativní úhlovou polohu kloubu a jeho rychlost. K vyhodnocení hodnot ze snímače se používá CPLD obvod. Ten data předzpracovává a následně dává k dispozici pro mikrokontrolér, který si je v případě potřeby může vyčíst.

4.1 Použité součástky

V horních kloubech robota, tedy jeho pomyslných kyčlích, jsou umístěny motory typu *Maxon A-max 22*. Součástí tohoto motoru je **inkrementální senzor polohy HEDL 5540**. Jelikož je tento senzor digitální, je přesnost měření úhlu omezena pouze jeho vlastní rozlišovací schopností. Přes všechny jeho výhody však z principu nemůže nahradit absolutní snímač polohy [1]. Výstupem snímače jsou dva signály, které jsou nositelem informace o pohybu snímače. Signály jsou vůči sobě posunuty o půl pulzu. Díky této vlastnosti lze rozpoznat směr pohybu. Otočíme-li snímačem o 360°, na výstupu těchto signálů postupně napočítáme 512 pulzů. Námi používaný motor je zpřevodován v poměru 1:270. To znamená, že otočíme-li hřídelí motoru o 360°, napočítáme celkem 184 320 pulzů. To odpovídá

[1] Součástí všech kloubů jsou také odporové snímače polohy. Jejich hlavní výhodou oproti výše zmíněným inkrementálním je fakt, že polohu měří absolutně. Pro potřeby řízení pohybu robota je právě znalost absolutní polohy kloubu nezbytná. Nevýhodou však je, že tyto snímače jsou na výstupu zatíženy šumem. Z tohoto důvodu byly desky osazeny RC filtrem typu dolní propust, aby alespoň částečně vliv šumu omezily.

rozlišovací schopnosti přibližně $5 \cdot 10^{-}6$ stupňů. Ke zvýšení citlivosti lze navíc s výhodou využít právě přítomnost dvou signálů. Jejich vhodným vyhodnocováním (viz 4.2) lze dosáhnout rozlišovací schopnosti ještě 4 krát větší.

Aby bylo možné snímač smysluplně využít k měření polohy, je nutné jeho výstupní signály vhodně zpracovávat. Přesněji řečeno je potřeba zajistit počítání pulzů, z kterých lze následně již vypočítat požadovaný úhel. Prvním řešením, které se nabízí, je vyhodnocování pomocí přerušení mikrokontroléru. Snímač se tak připojí na některý z pinů, na kterých lze aktivovat funkci přerušení. Pokud snímač vyšle pulz, jeho náběžná hrana vyvolá přerušení a mikrokontrolér signál zpracuje pomocí softwarového čítače. Toto řešení je jednoduché, ale neefektivní. Vzhledem k frekvenci, v jaké by snímač při běžném pohybu motorů vysílal pulzy, by byl mikrokontrolér významně vytížen obsluhou takového přerušení. Dle zkušeností jiných pracovníků na katedře by docházelo k úplnému zahlcení mikrokontroléru.

Druhým možným řešením je využít elektronického obvodu, který bude signály ze snímače průběžně zpracovávat. Naměřené hodnoty pak mohou být již ve formě binárního kódu posílany mikrokontroléru. Právě s tímto řešením se počítalo při návrhu desek robota. Ty jsou osazeny tzv. **komplexně programovatelným logickým obvodem (CPLD) Xilinx XC95144XL**. Tato součástka, podobně jako programovatelná hradlová pole (FPGA), umožňuje naprogramovat vlastní logický obvod. Tyto součástky obsahují základní logické jednotky, tzv. makrobuňky. Každá makrobuňka je jeden paměťový člen doplněný o několik dalších hradel. Obvod umožnuje naprogramovat vzájemné propojení makrobuněk a vytvořit tak širokou škálu logických obvodů.

4.2 Programování CPLD

V této podkapitole nejprve popíšu základní činnost s nástrojem pro vývoj obvodů v CPLD. Poté následuje popis mé implementace čítače pulzů inkrementálního senzoru.

4.2.1 ISE Design Suite

CPDL jsem programoval v jazyce VHDL. Od výrobce použitého CPLD, společnosti *Xilinx*, je pro vývoj obvodů k dispozici nástroj **ISE Design Suite**. V aplikaci je potřeba nejprve vytvořit projekt. V parametrech projektu se při této příležitosti nastaví odpovídající

CPLD obvod a jako výchozí jazyk se zvolí VHDL. Poté se projekt otevře v aplikaci. Vlevo na kartě *Design* je přehled souborů projektu. Nově vytvořený projekt zatím žádné neobsahuje. Klikneme tedy pravým tlačítkem na ikonku procesoru a zvolíme *New Source...*. Vybereme *VHDL Module* a soubor vhodně nazveme. V následujícím okně se definují vstupní a výstupní porty. Portu je vhodné přiřadit jméno odpovídající jeho významu. Ke každému portu se přiřadí, zda je vstupní, výstupní, nebo vstupně-výstupní. Po potvrzení dojde k vygenerování zdrojového souboru VHDL. Poté je ještě nutné asociovat porty s konkrétními piny CPLD. To je možné udělat otevřením nástroje *Floorplan IO – Pre-Synthesis* v nabídce *User Constraints*. V okně *Design Object List – I/O Pins* se v parametru *Loc* přiřadí jednotlivým portům hodnota konkrétního pinu. Rozložení pinů je zároveň schématicky zobrazeno v okně po pravé straně. Piny, které jsou zobrazeny ve formě čtverečku, jsou vyhrazeny obvodem a uživatel jim již nemůže přiřadit vlastní port.

Po dokončení konfigurace je možné přistoupit k samotnému programování. Kompilace programu se spustí dvojklikem na *Implement Design*. Pokud je kompilace v pořádku zobrazí se zpráva o jejím výsledku. Jejím obsahem je mimo jiné informace o použitých makrobuňkách, registrech, pinech apod. Pokud chceme obvod nahrát do CPLD stiskneme *Manage Configuration Project*. Otevře se okno se schématickou značkou znázorňující CPLD. Klikneme na něj pravým tlačítkem a stiskneme *Program*. Obvod se následně nahraje do CPLD. K programování se používá převodník pro paralelní port. Popřípadě lze USB převodník, který je ale nepoměrně dražší

4.2.2 Čítač pulzů

Signály z inkrementálního čidla jsou zpracovávány formou čítače. Počet napočítaných pulzů pak přímo odpovídá relativní poloze kloubu. Jinak řečeno počet pulzů odpovídá velikosti vychýlení od pozice, při které bylo CPLD zapnuto. Při návrhu jsem se zabýval otázkou, jak velký čítač zvolit. Nejjednodušším řešením by byl osmibitový. Důvodem je, že CPLD je k mikrokontroléru připojen pomocí 8 vodičů. Pokud by tedy čítač měl právě 8 bitů, bylo by možné všechny výstupy z CPLD trvale obsadit hodnotami čítače. Žádná další komunikace by mezi CPLD a mikrokontrolérem nebyla potřeba. Mikrokotrolér by jen vždy, když by to vyžadoval, přečetl hodnoty ze svých 8 vstupů.

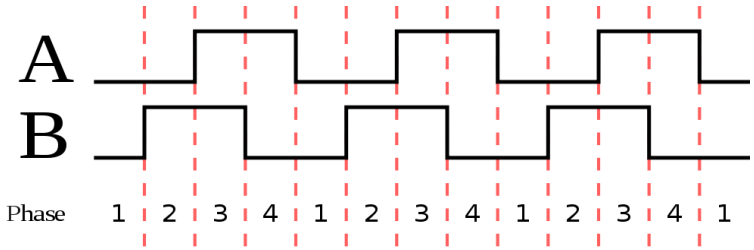
Toto řešení jsem však zavrhl. Osmibitový čítač umožňuje uložit jen 256 hodnot. To by v případě našeho senzoru znamenalo, že by se již při mirném pootočení hřídele došlo k přetečení čítače. Pak by ještě mohl nastat druhý problém. Pokud by nebyl přenos mezi CPLD a mikrokontrolérem nijak synchronizován, mohlo by dojít k hazardům. Tedy, že

by mikrokontrolér četl v okamžiku, kdy na výstupu z CPLD dochází k překlopení.

Abych předešel prvnímu problému, rozhodl jsem se použít čítač o velikosti 20 bitů. Ten umožňuje zaznamenat 1 048 576 různých hodnot. Tato hodnota již umožňuje otočit hřídelí motoru již více než jednou okolo své osy. Ani k tomu však nedojde, jelikož to mechanické dorazy robota neumožňují. Při inicializaci CPLD se hodnota čítače přednastaví na výchozích 524 288, aby riziko přetečení a podtečení bylo totožné. Problém s přenosem dat do mikrokontroléru je řešen v 4.2.3

Je vhodné ještě zmínit, že snímač mimo dvou signálů pro počítání pulzů obsahuje navíc třetí signál, tvz. indexový nebo nulový. K jeho generování dochází pouze v jedné tzv. nulové poloze z celého otočení snímačem. Detekce této polohy se někdy využívá k nulování čítače. V tu chvíli lze senzor využít nejen k detekování relativní změny polohy, ale i k určení absolutní polohy. V našem případě toho však příliš využít nelze. Díky zpřevodování by k nulování čítače docházelo 270 krát během jedné otáčky. Proto jsem se rozhodl indexový signál nevyužít.

Nyní popíšu samotné zpracování signálů s využitím CPLD. Časový průběh signálů snímače je zobrazen na obr. 4.1. Obrázek platí pro kladný směr otáčení. V obrázku však lze vypozorovat i průběh signálů pro záporný směr otáčení. V takovém případě si stačí představit, že časová osa roste nikoliv zleva doprava, ale zprava doleva. Což je i analogické ke skutečné konstrukci snímače. Dojde-li ke změně směru otáčení, signály se začnou generovat pozpátku.



Obrázek 4.1: Průběh signálu inkrementálního snímače (zdroj: [11]

Jak již bylo řečeno v kapitole 4.1, použité senzory generují dva signály s celkem 512 pulzy na jednu otáčku. Běžně se tyto signály vyhodnocují tak, že pokud se objeví na kanálu A vzestupná hrana a kanál B je v tu chvíli v log. 1, dojde k přičtení čítače. Pokud se naopak objeví na kanálu B vzestupná hrana a kanál je v tu chvíli v log. 1, dojde k odečtení čítače. Tento princip umožňuje napočítat všech 512 pulzů na otáčku a zároveň rozlišit směr otáčení.

Máme-li však vyhodnocovací zařízení, které je schopné detekce i sestupných hran a zároveň má dostatečnou vzorkovací frekveci, je možné napočítat až 2048 pulzů na otáčku. Lze totiž s výhodou využít toho, že snímač generuje dva vzájemně fázově posunuté signály. Získáváme tak vlastně dvoubitový binární kód, který nám dovoluje rozlišit 4 různé stavy v rámci 1 pulzu. Situace je na obr. 4.1 a v tab. 4.1, 4.2 znázorněna pomocí fází. S postupným otáčením snímače platí, že fáze 1 přechází ve fazí 2, fáze 2 ve fázi 3 atd.

Tabulka 4.1: Kladný směr otáčení

Fáze	A	B
1	0	0
2	0	1
3	1	1
4	1	0

Tabulka 4.2: Záporný směr otáčení

Fáze	A	B
1	1	0
2	1	1
3	0	1
4	0	0

Tento způsob zpracování signálů ze snímače tedy umožňuje navýšit počet čítaných pulzů 4x. Nakonec je potřeba zmínit, jakým způsobem se rozlišuje otáčení vpřed a otáčení zpět. K tomu se využívá paměti předchozích hodnot signálů. Je-li současný kód například 11 a předchozí kód byl 01, je z tab. 4.1 zřejmé, že se snímač pohybuje směrem vpřed. Naopak pokud by kód přecházel z 10, jednalo by se o otáčení v záporném směru (viz tab. 4.2).

Na následujícím fragmentu kódu popíšu realizaci tohoto řešení ve VHDL.

```
process (clock)
begin
 if rising_edge(clock) then
        input_last <= input;
        input(0) <= IRC_AM_A;
        input(1) <= IRC_AM_B;


        if (input(0) = '1' and input_last(0) = '0') then -- A rising edge
                if (input(1)='1') then -- forward
                        counter <= counter + 1;
                elsif (input(0)='0') then -- reverse
                        counter <= counter - 1;
      end if;

   elsif (input(1) = '1' and input_last(1) = '0') then -- B rising edge
      if (input(0)='0') then -- forward
                        counter <= counter + 1;
      elsif (input(0)='1') then -- reverse
                        counter <= counter + 1;
      end if;
```

Vyhodnocení signálů je součástí procesu. Tento proces se spouští s každou vzestupnou hranou hodin. Frekvence těchto hodin odpovídá osazenému krystalu - 100 MHz. Nejprve pro pozdější porovnání uchovají původní hodnoty signálů, nové hodnoty se naopak načtou. Poté následuje podmínka, která zjišťuje, zda nedošlo na signálu A k vzestupné hraně. Pokud ano, prověřuje se signál B. Pokud má hodnotu log. 1, došlo k pootočení v kladném směru. Pokud má hodnotu log. 0, došlo k pootočení v záporném směru. Podobné je to u další podmínky, která vyhodnocuje vzestupnou hranu signálu B. Dále v kódu následuje (zde již není uvedeno) vyhodnocení sestupných hran obou signálů.

4.2.3 Komunikace mezi CPLD a mikrokontrolérem

Při návrhu způsobu přenosu dat mezi CPLD a mikrokontrolérem bylo rozhodující, že velikost čítače s hodnotou je 20 bitů, zatímco šířka univerzální sběrnice jen 8 bitů. Rozhodl jsem se rozdělit sběrnici na 4 řídící a 4 datové signály. Datové vodiče jsou z pohledu CPLD výstupní, zatímco všechny řídící vodiče jsou vstupní. Jeden signál nechávám nevyužitý. Rozložení je znázorněno v tab. 4.3

Tabulka 4.3: Sběrnice mezi CPLD a mikrokontrolérem

bit	význam
0	adresa[0]
1	adresa[1]
2	adresa[2]
3	požadavek na data
4	data[0]
5	data[1]
6	data[2]
7	data[3]

Jelikož jsou datové vodiče pouze 4, provádí se přenos sekvenčně, kdy jsou 5 krát po sobě posílány 4 bity. Celý přenos je řízen mikrokotrolérem, který ovládá řídící bity. 3 adresové bity slouží k výběru požadované čtveřice bitů. Je-li např. na adresních bitech nastavena hodnota log. 000, CPLD umístí na výstup 4 nejméně významné bity čítače. Naopak nastaví-li mikrokontrolér adresní vodiče na 101, jsou na datové vodiče umístěny 4 nejvýznamnější bity.

Průběh přenosu nyní ještě popíšu na části z kódu, která ho zajišťuje:

```
unsigned long tmp;
unsigned long read_value = 0x00000000;
int16 value;

//set GPIO 1.14, 1.15, 1.16, 1.17 to input (0)
IODIR1 &= ~(1 << 14 | 1 << 15  | 1 << 16 | 1 << 17);
//set GPIO 1.4, 1.8, 1.9 to output (1)
IODIR1 |=  (1 << 4) | (1 << 8) | (1 << 9) | (1 << 10);


//set request for data
IOSET1 |= (1 << 10);

//set address first half-byte
IOSET1 &= ~(1 << 4 | 1 << 8 | 1 << 9);
delay_us(1); //wait for 0.1 us
tmp = FIO1PIN;
tmp >> 14;
read_value |= (tmp & 0x0000000F);
```

Po deklaraci proměnných se pomocí registru *IODIR1* definují, které z *GPIO* (General purpose input output) portů budou vstupní a které výstupní. Poté je pomocí signálu č. 2 zaslána informace o tom, že bude probíhat čtení dat z čítače. To se v CPLD projeví tak, že aktuální hodnota čítače se nakopíruje do nezávislého registru. Zajistí se tak, že se hodnota nebude v průběhu přenosu měnit. Poté se provede již výše uvedené nastavení adresních vodičů. Následně mikrokontrolér čeká po dobu 1 μs, aby mohlo dojít k ustálení všech stavů. Data ze vstupu se uloží do proměnné *tmp*, kde jsou ještě pomocí rotace a vymaskování přesunuty na jim odpovídající místo. V další části kódu, který zde již není zobrazen, se přenos opakuje s adresními signály zvětšenými o jedna.

Po dokončení přenosu se od hodnoty čítače odečte hodnota 524 288, o kterou byl čítač uměle navýšen, aby nedocházelo k jeho podtečení.

Výše uvedený kód jsem umístil do souboru *lib_cpld.c*, který se v projektu nachází ve složce *lib*. Ve zdrojovém kódu se taktéž nachází funkce pro převod hodnoty z čítače na hodnotu úhlu.

Kapitola 5

Virtuální model v Simulink 3D Animation

Tato kapitola se zabývá implementací 3D modelu robota do prostředí Simulink 3D Animation. Tento toolbox umožňuje využít výsledků simulace v programech Matlab a Simulink k vizualizaci pohybu 3D modelu. Cílem je vytvořit grafický model podaktuovaného robota a vizualizovat pohyb, kterého je schopna jeho reálná předloha. Součástí kapitoly je základní popis prostředí virtuální reality v programu Matlab. Poté následuje postup, jak vytvořit novou 3D scénu a napojit ji na blokový model v Simulinku.

5.1 Simulink 3D Animation

Pro práci s 3D virtuální realitou je v Matlabu k dispozici Simulink 3D Animation toolbox. Tento toolbox je součástí distribucí 2009a a novější. Starší verze obsahovaly Virtual Reality Toolbox. Oba toolboxy jsou však totožné, došlo jen k přejmenování. Proto i možnosti a schopnosti zůstaly zachovány. Kromě samotné vizualizace scén umožňují i obousměrné spojení dynamického modelu a 3D modelu, tvorbu vlastních uživatelských rozhraní nebo připojení vstupních zařízení jako je např. joystick.

5.1.1 VRLM

Veškeré virtuální scény jsou uloženy ve formátu VRLM (Virtual Reality Modeling Language), konkrétně ve verzi 2.0, někdy také nazývána VRLM97. Výhodou formátu je jeho

univerzálnost a široké spektrum využití při zachování relativní jednoduchosti. Díky jeho rozšířenosti je možné vytvořit virtuální scénu i v jiném editoru a následně ji využívat v Simulink 3D Animation. Virtuální scény v tomto jazyce se často nazývají jako virtuální světy (popřípadě jen světy). Tomu odpovídá i přípona souborů *.wrl.

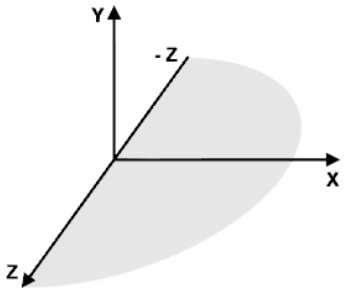
Ačkoliv se virtuální světy vytvářejí především v grafických editorech, je pro potřebu tvorby 3D simulací nutné znát alespoň základní strukturu VRLM souborů. Každý virtuální svět je složen z jednotlivých objektů, kterým se říká uzly. Každý uzel obsahuje vlastnosti a parametry, které je možné měnit. Přičemž jedním z parametrů může být i jiný uzel. To umožňuje tvořit hierarchické skupiny složené z uzlů. Změna vlastnosti uzlu vždy ovlivní i všechny jeho potomky. Uzlem může být 3D objekt, jehož parametrem bude tvar, textura, poloha, rotace aj. Existují však i uzly s naprosto odlišnými schopnostmi, jako je např. nastavení pohledů nebo průletů kamery. Celkem takto ve VRLM97 existuje 54 druhů různých uzlů. Pro účely simulací jsou nejvýznamnější tyto:

- **Transform** – seskupovací uzel. Obsahuje vlastnosti jako pozici, rotaci, velikost všech svých podřízených uzlů.
- **Sphere, Box, Cone, Cylinder** - základní tvary, z nichž lze tvořit 3D objekty
- **IndexedFaceSet, Extrusion aj.** - skupina uzlů, z kterých jsou vytvořeny komplikovanější 3D tvary.
- **Material, ImageTexture aj.** - skupina uzlů, které definují barvu nebo texturu na povrchu objektů
- **DirectionalLight** - zdroj přímého světla
- **Viewpoint** - nastavení pohledu, tj. poloha a směr kam se uživatel dívá

Při tvorbě virtuálních scén se tak nejprve vytvoří příslušné tvary s potřebnými texturami. Tyto tvary se vhodně seskupí. Dynamika scény se poté vytvoří vhodnou změnou parametrů polohy. Platí totiž, že libovolný parametr uzlů lze průběžně měnit i v průběhu simulace.

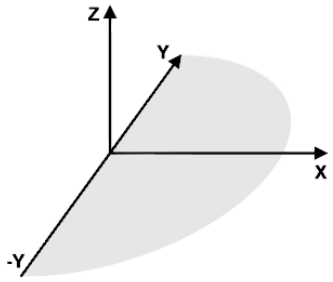
Souřadný systém VRLM a použité fyzikální jednotky

VRLM využívá pravotočivý Kartézský souřadný systém s osami, jak je zobrazeno na obr. 5.1



Obrázek 5.1: Souřadnicový systém VRLM

Souřadnicový systém se mírně liší od toho, jaký standardně používá Matlab (obr. 5.2), u kterého osa +Y směřuje od pozorovatele a osa +Z míří nahoru. Využívají-li se tedy data z Matlabu, je nutné prohodit 2. a 3. složku vektoru a u jedné z nich zaměnit znaménka.



Obrázek 5.2: Souřadnicový systém Matlabu

Dále platí, že všechny rozměry a vzdálenosti jsou definovány v metrech, úhly v radiánech a čas v sekundách. Veškeré parametry je tedy nutné transformovat do těchto jednotek.

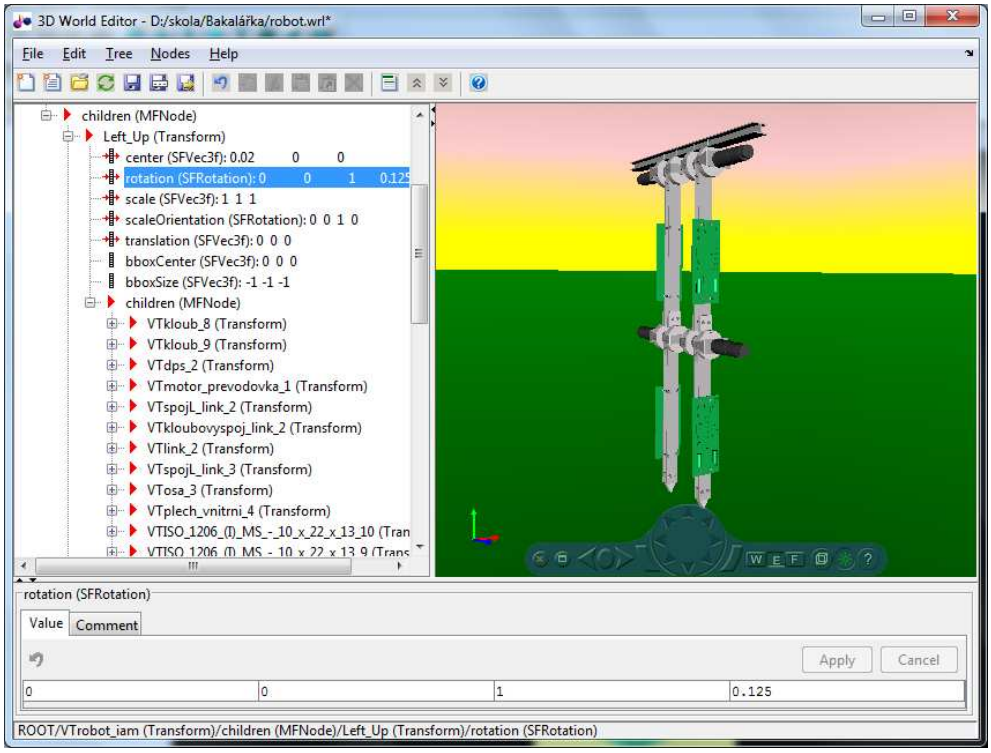
5.1.2 3D editory

VRLM je deklarativní programovací jazyk a při dostatečné znalosti jeho syntaxe je z principu možné tvořit libovolné modely v jakémkoliv textovém editoru. Především při tvorbě složitějších 3D scén je však určitě výhodnější využít některý z 3D grafických editorů. Dva takové editory jsou přímo součástí Simulink 3D Animation. Novějším a dnes již výchozím je 3D World Editor. Druhým je V-Realm Builder. Jeho poslední verze 2.0 je sice již více než 10 let stará, ve výsledku to však ničemu nevadí. Vzhledem k tomu, že oba dva editory pracují se standardem VRLM97, který je neměnný, nabízejí i stejné možnosti. Záleží tak spíše na uživateli, který z grafických editorů mu je přívětivější. Práci v nich je také možné

kombinovat. Nevýhodou V-Realm je jeho dostupnost pouze pro platformu Microsoft Windows. Při práci jsem využil spíše 3D World Editor. Pro úplnost tak alespoň dodám, že nejjednodušší způsob, jak spustit V-Realm, je otevřít si ho přímo z jeho úložiště. V praxi bývá umístěn zde:

C:\Program Files\MATLAB \R20XX\Toolbox\sl3d\vreaml\program\vrbuild2.exe.

Mimo těchto dvou lze využít i jiné nativní VRLM editory. Ty také využívají VRLM jako svůj hlavní formát a umožňují tak využít všech vlastností jazyka. Mezi ně patří např. Flux Studio nebo SwirlX3D. Poslední možností je využít nějaký univerzální 3D modelovací editor, který umožňuje export do VRLM. Takovým je např. 3ds Max, Autodesk Maya, Google SketchUp.



Obrázek 5.3: 3D World Editor

Vedoucí bakalářské práce mi dal k dispozici již vytvořený 3D model v programu Autodesk Inventor. Tento program také patří mezi univerzálních 3D editory. Na rozdíl od již zmíněných však ve výchozí konfiguraci nenabízí export do VRLM. Nalezl jsem si teda přídavný plug-in VRML Translator for Inventor, který Inventor o tuto schopnost rozšíří. Plug-in je komerční nástroj. Lze však využít jeho trial verzi, která dovoluje neomezené použití po dobu 15 dní, což pro jednorázový export bohatě dostačuje.

5.1.3 3D World Editor

Model ve formátu VRLM si je již možné otevřít v prostředí 3D World Editor (obr. 5.3), ve kterém jsem prováděl dodatečné úpravy. Editor spustíme nejjednodušeji v Matlabu pomocí jeho ikonky start v levém dolním rohu. Zde zvolíme *Start* → *Simulink* → *Simulink 3D Animation* → *3D World Editor*.

V okně editoru je po levé straně umístěn seznam všech uzlů. Každý samostatný objekt, který byl v Inventoru vytvořen, je zde zastoupen jedním uzlem. Jelikož je model navržen až do úrovně jednotlivých šroubů, je složen z více než 100 uzlů. Uzly je potřeba vhodně roztřídit tak, aby uzlu, které se vůči sobě vzájemně nepohybují, byly součástí jednoho nadřazeného uzlu.

Robot má celkem 5 částí, které se vůči sobě mohou navzájem pohybovat. Čtyři z nich jsou jednotlivé články nohou. Pátá část je osa, která spojuje levou a pravou nohu. Vytvořil jsem tedy 5 uzlů typu *Transform*[1], do kterých jsem seskupil části, které se vůči sobě nepohybují. Horní část, spojující pravou a levou nohu jsem nazval *Body*. Články nohou jsem nazval *Left_up*, *Left_down*, *Right_up* a *Right_down*. Hierarchicky jsou spodní články potomky horních článků a horní články jsou potomky spojující vrchní osy.

Na (obr. 5.3) je vlevo vidět několik parametrů, které k uzlu příslušejí. Pro nás mají největší význam parametry *center*, *rotation*, *translation*.

Parametr *center* je vektor o třech prvcích a určuje polohu středu uzlu (objektu). Pokud budeme objektem otáčet, bude se točit právě okolo tohoto bodu. Proto jsem střed všech čtyřech článků nohou robota umístil do středu jejich kloubů.

Parametr *rotation* je vektor o 4 prvcích. První 3 prvky určují osy, okolo kterých se objekt bude otáčet. Články nohou se mohou otáčet pouze v jedné ose, v tomto případě ose *Z*. My chceme otáčet pouze okolo osy *Z*. Proto pouze třetí prvek má hodnotu 1. Ostatní dva jsou nulové. Hodnoty vektoru lze měnit ve spodní části okna programu. Poslední prvek vektoru je samotný úhel otočení. To budu měnit až v průběhu samotné simulace.

Parametr *translation* má tři prvky a definuje polohu prvku v prostoru.

Pokud by jsme neměli model již připravený z jiného 3D editoru, je možné ho vytvořit přímo v 3D World Editoru. V hlavním okně v menu *Nodes* je funkce *Add*, pomocí které je možné přidat uzel libovolného typu. Další funkce *Insert From...* pak umožňuje uzlům přiřadit vlastní materiál nebo texturu. Popřípadě lze vložit již nějaký hotový objekt ve formátu VRLM. Součástí 3D World Editoru je knihovna již vytvořených objektů.

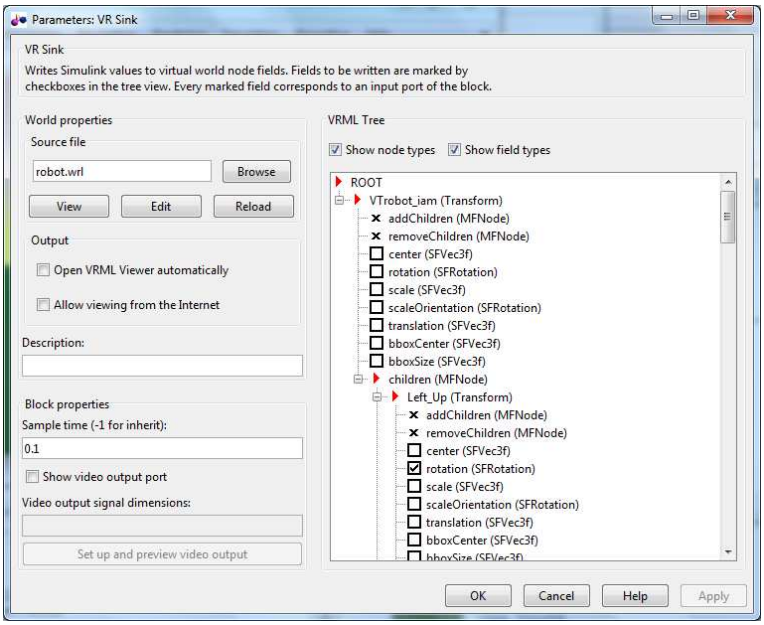
[1] *Transform* je základní seskupovací uzel, jehož parametry umožňují určit polohu jeho potomků v prostoru.

Rozsáhlejší úprava v tomto editoru je poměrně zdlouhavá a uživatelsky nepřívětivá. Proto při tvorbě složitějších objektů je určitě výhodné využít jiných editorů.

5.1.4 Propojení modelu se Simulinkem

V této podkapitole popíšu, jak se již připravený model ve VRLM propojí se schématem v Simulinku.

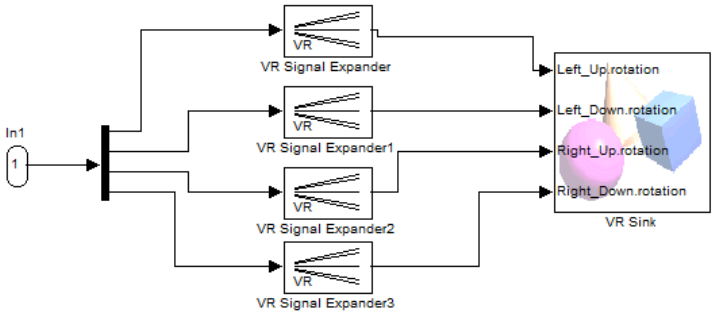
Nejprve si spustíme Simulink a vytvoříme si nové blokové schéma. Otevřeme si knihovnu příkazem *View* → *Library Browser*. Vlevo se nachází rozpis celé knihovny funkčních bloků. Rozklikneme *Simulink 3D Animation*. Vpravo se objeví výpis blokových schémat. My budeme potřebovat *VR Sink* a *VR Signal Expander*. Přesuneme si je tedy do našeho schématu. *VR Sink* Umožňuje měnit v reálném čase parametry uzlů modelu. Vstupem bloku jsou signály, které jsou přiřazeny ke konkrétnímu parametru. Hodnota signálu se během simulace přičítá k parametru. Úkolem bloku *VR Signal Expander* je vytvořit ze skalárního signálu vektorový. Signál vstupující do bloku *VR Sink* musí mít stejný rozměr, jako je rozměr parametru. Parametry mají obvykle tři až čtyři složky.



Obrázek 5.4: Nastavení bloku VR Sink

Nyní dvojklikem otevřeme blok *VR Sink*(obr. 5.4). V sekci *Source File* klikneme na *Browse* a vybereme připravený VRLM model. V sekci *VRLM Tree* se zobrazí hierarchická struktura uzlů načteného modelu. U parametrů, které je možné ovládat, je k dispozici

zaškrtávací čtvereček. U modelu robota budeme měnit rotační polohu jednotlivých článků. Zaškrtneme tedy parametr *rotation* pro uzly *Left_up*, *Left_down*, *Right_up* a *Right_down*. Okno uzavřeme potvrzovacím tlačítkem *OK*.



Obrázek 5.5: Simulinkové schéma s virtuální realitou

Ve simulinkovém schématu (obr. 5.5) bude mít nyní blok *VR Sink* čtyři vstupy. Máme-li vhodný zdroj referenčních signálů, můžeme je již k bloku *VR Sink* připojit. Pokud by signál byl v jiných jednotkách než v radiánech, bylo by nutné hodnoty nejprve převést. Poté je již možné spustit simulaci.

Kapitola 6

Závěr

Cílem této práce bylo pokračovat na vývoji modelu podaktuovaného kráčejícího robota. Svým zaměřením práce navazovala na diplomovou práci Ondřeje Rotta [1]. Ten na základě požadavků svého zadání navrhl vysoce komplexní platformu skládající se jak z hardwarového, tak softwarového řešení. Dále se také zabýval teorií řízení chůze a stability robota a navrhl algoritmy pro řízení robota. Činnost těchto algoritmů poté ověřoval pomocí pokusů. Součástí návrhu řídicích desek byly i dvě součástky, které byly osazeny, ale z časových důvodů již nebyla vyřešena jejich softwarová implementace.

První součástkou je externí paměť. Jejím úkolem je uchovávat číselné konstanty a zároveň umožnit jejich snadnou změnu. V původním dočasném řešení byly hodnoty konstant součástí kompilace programu. K jejich změně se tak musela provádět poměrně komplikovaná překompilace mikroprocesoru. Po provedených úpravách se po spustění programu konstanty již nahrávají z externí paměti. Zároveň lze tyto konstanty snadno měnit pomocí uživatelsky přívětivé grafické aplikace.

Druhá součástka, která nebyla využívána, je programovatelný logický obvod (CPLD). Jejím úkolem je počítat pulzy z inkrementálního čidla umístěného v motorech robota. Tato čidla umožnují velice přesně měřit relativní změnu úhlové polohy kloubů. V původním stavu však čidla nebyla využita a měření polohy se provádělo pouze pomocí rotačních odporových snímačů. Navrhl jsem tedy logický obvod, který pulzy z čidla počítá a hodnotu umožňuje odeslat do mikrokontroléru. V současnosti již je možné využít absolutních odporových čidel polohy i těchto relativních inkrementálních čidel.

Poslední část práce je věnována vytvoření simulačního modelu v prostředí Simulink 3D Animation. Tato aplikace umožňuje na 3D modelu kráčejícího robota vizualizovat výsledky simulací zpětnovazebního řízení. Výsledkem práce je 3D model, který již lze vložit do libovolné simulace. Model byl otestován na vzorové simulaci a je plně funkční.

Literatura

[1] O. Rott, *Platforma pro řízení podaktuovaných mechanických systémů*, Diplomová práce, ČVUT Praha, 2012.

[2] E. Westervelt, J. Grizzle, Ch. Chevallereau, J. Choi, B. Morris, *Feedback Control of Dynamic Bipedal Robot Locomotion*, CRC Press, 2007

[3] Ch. Chevallereau, G. Bessonet, G. Abba, Y. Aoustin, *Bipeda Robots Modeling, Design and Walking Synthesis*, WILEY, 2009

[4] M. Anderle, S. Čelikovský, D. Henrion, J. Zikmund, *Advanced LMI based analysis and design for Acrobot walking*, International Journal of Control, vol. 83, no. 8, 2010

[5] M. Anderle, S. Čelikovský, *Stability analysis of the Acrobot walking with observed geometry*, Proceedings of the 18th IFAC World Congress, 2011

[6] M. Anderle, S. Čelikovský, *Feedback design for the Acrobot walking-like trajectory tracking and computational test of its exponential stability*, IEEE Multi-Conference on Systems and Control, 2011

[7] M. Anderle, S. Čelikovský, *Sustainable Acrobot Walking Based on the Swing Phase Exponentially Stable Tracking*, Proceedings of the ASME 2010 Dynamic Systems and Control Conference, 2010

[8] M. Anderle, S. Čelikovský, *Preprints of the 8th IFAC Symposium on Nonlinear Control Systems*,Preprints of the 8th IFAC Symposium on Nonlinear Control Systems, 2010

[9] J. Pinker, M. Poupa, *Číslicové systémy a jazyk VHDL*, BEN, 2006

[10] *Microchip 24LC02B*, Datasheet

[11] *Rotary encoder*, URL: `http://en.wikipedia.org/wiki/Rotary_encoder`

[12] *Simulink 3D Animation User's Guide*, URL: http://www.mathworks.com/help/toolbox/sl3d/ug_i

Příloha A

Obsah přiloženého CD

K této práci je přiloženo CD, které je uspořádáno do následujích složek:

- Thesis: tento dokument ve formátu PDF.
- Simulink: simulační model pro Simulink 3D Animation
- Eclipse: zdrojové kódy pro řídicí desky robota
- GUI: grafická aplikace pro komuniakci mezi PC a robotem
- Datasheets: technická dokumentace k použitým součástkám