České vysoké učení technické v Praze
Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE

Robot HERO

Praha, 2007

Autor: Ondřej Šantin

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne _____ _____

podpis

Poděkování

Děkuji především vedoucímu bakalářské práce Doc.Ing. Jiřímu Bayerovi, CSc. za věcné připomínky a vedení při tvorbě této práce. Rád bych také vyjádřil svou vděčnost Ing. Pavlu Píšovi, který mi udělil mnoho cenných rad a věnoval svůj čas, především pak při tvorbě ovladače pro použitou CAN kartu. Můj dík patří též kolegovi Jiřímu Zemánkovi, za jeho podnětné rady a postřehy.

Abstrakt

Cílem této bakalářské práce je přepracovat starou koncepci robota HERO tak, aby ho bylo možno používat jako učební pomůcku při výuce robotiky a umělé inteligence.

Jako řídící jednotka robota je použit počítač PC/104, který má za úkol zpracovávat požadavky od připojených klientů a prostřednictvím protokolu CANopen komunikovat s řídícími jednotkami motorů pohybových os a řídit tak pohyb robota.

Komunikace mezi robotem a osobním počítačem je řešena bezdrátovou technologií Wi-fi.

Abstract

The purporse of this Bachelor's thesis is to remake old conception of robot HERO to be able to use as learning tool for robotics and artificial life.

As unit control of robot is used PC/104 system based, to process demands from connected clients and through protocol CANopen communicate with motor's control units of axes to control a movement of the robot.

Communication between robot and PC is solved by wireless technology Wi-fi.

České vysoké učení technické v Praze, fakulta elektrotechnická

Katedra řídicí techniky

Školní rok: 2006/2007

Zadání bakalářské práce

Student: Ondřej Šantin

Obor: Kybernetika a měření

Název tématu: Robot HERO – dálkové řízení

Zásady pro vypracování:

1. Seznamte se s mechanickou koncepcí mobilního robota HERO a s požadavky na základní řízení jeho pohybových os (pojezd, směr pojezdu, pohyb ruky, stisk chapadla).
2. V návaznosti na paralelní BP J. Zemánka (základní řízení robota) realizujte dálkové řízení pohybových os robota z nadřazeného PC.
3. Pro dálkové řízení využijte jeden ze standardních způsobů přenosu dat (BlueTooth, WiFi, ZigBee).
4. Dálkové řízení koncipujte tak, aby mohlo komunikovat s několika „vloženými" systémy robota zajišťujícími kromě základního řízení pohybových os také další funkce sofistikovaného řízení (orientace v prostředí).
5. Navrhněte koncepční řešení sofistikovaného řízení mobilního robota pro jeho pohyb v neznámém prostředí s překážkami.

Seznam odborné literatury:
dokumentace robota Hero, diplomové práce Vaněk, Pácha

Vedoucí bakalářské práce: Doc. Ing. Jiří Bayer, CSc.

Datum zadání bakalářské práce: zimní semestr 2006/07

Termín odevzdání bakalářské práce: 15. 8. 2007

V1

L.S.

Prof. Ing. Michael Sebek, DrSc.
vedoucí katedry

Prof. Ing. Zbyněk Škvor, CSc.
děkan

V Praze, dne 6. 3. 2007

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

1.1 Účel projektu

Katedrou řídící techniky byl zadán projekt přepracování stávající koncepce mobilního robotu HERO tak, aby bylo na něm možno testovat algoritmy řízení mobilních robotů.

Robot by měl být vybaven řídícím počítačem s operačním systémem GNU/Linux, využívat průmyslovou sběrnici CAN a dále být vybaven senzorovým systémem umožňujícím orientaci v prostoru.

Výsledný model by pak měl sloužit jako učební pomůcka při studiu distribuovaných systémů, robotiky a některých oborů umělé inteligence, především pak těch zabývajících se orientací v prostředí.

Celé zařízení by mělo být bezdrátově připojeno k univerzitní síti. Studenti tak budou moci naprogramovat robot a odeslat výsledný kód přes webové rozhraní. Výsledky své práce pak budou moci sledovat přes webovou kameru.

1.2 Historie projektu

Začátky projektu HERO se datují rokem 1979, kdy americká společnost Heathkit začala vyvýjet roboty pro podporu výuky robotiky a umělé inteligence.

Inženýrům z této firmy se nakonec podařilo vyvinout robota, kterého nazvali HERO 1. Ten byl vybaven řídící jednotkou založenou na 8-bitovém procesoru 6808 s 8KB ROM a 4KB RAM.

Programovat robot bylo možno pomocí připojené klávesnice, případně počítačem

připojeným přes rozhraní RS232.

Robot byl vybaven dále mechanickou rukou s 5-ti stupni volnosti, která umožňovala běžnou manipulaci s lehkými předměty.

Jako pohonné jednotky všech os byly použity krokové motory nevyžadující zpětnou vazbu pro informaci o poloze.

V roce 1985 na projektu pracoval na katedře řídící techniky České vysoké učení technické v PrazeIng. Franěk spolu s Ing. Páchou. Ti předělali řídící systém a použili výkonou procesorovou desku založenou na 80C192. Byly vyměněny výkonové budiče motorů a doplněn senzorový systém o optické závory.

V listopadu 2006 projekt znovu odstartovala tato práce společně s prací Jiřího Zemánka.

1.3 Cíl bakalářské práce

Cílem práce bylo projekt předělat tak aby byl založen na architektuře x86 a jednotivé osy bylo možno řídit distribovaně pomocí sběrnice CAN.

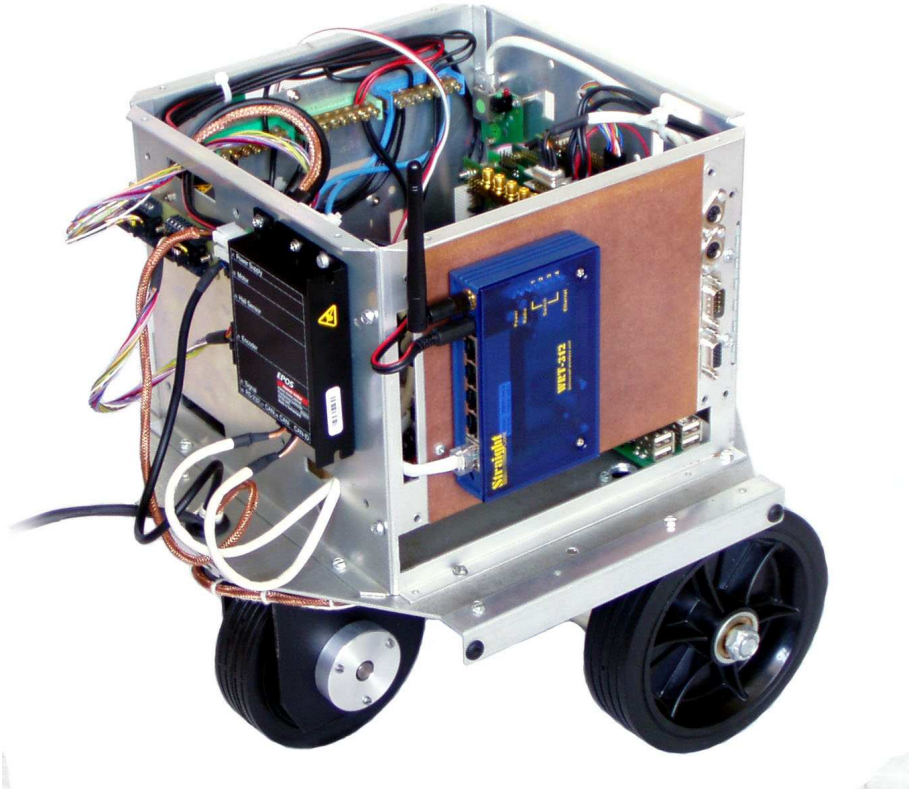
Dále pak vytvořit základní knihovní funkce pro řízení všech os, umožnit robota řídit vzdáleně pomocí některé z běžných bezdrátových technologií a vytvořit grafické uživatelské rozhraní pro snadnější řízení.

Úpravou hardwaru se zabývá bakalářská práce Jiřího Zemánka, viz. [1].

Kapitola 2

Hardware robotu

2.1 Přehled



Obrázek 2.1: Mechanická konstrukce robota HERO

Konstrukce robotu HERO na obr. 2.1, vychází z koncepce původního robotu z osmdesátých let a byla přepracována Jiřím Zemánkem.

Výčet použitého hardwaru, včetně jejich technických parametrů je možné nalézt v jeho bakalářské práci, viz. [1], případně v katalogových listech na přiloženém CD k této práci.

Zde je uveden jen výčet nejdůležitějších komponent týkající se přímo této práce.

2.1.1 Řídící jednotka robota

Pro řízení robota byla vybrány moduly standartu PC104+ od firmy Digital Logic:

- **MSM800XSEV** procerorová deska, 500MHz AMD Geode, 256MB RAM, VGA, USB, LAN
- **MSMPS104** napájecí zdroj, +5V/15A, +12V/1.7A
- **MSMCA104+** 2 kanály CAN
- **MSMG104+** 4x CVBS framegrabber

2.1.2 Řídící jednotky motorů

Pro řízení motorů pohybových os robota, byly vybrány řídící jednotky od firmy MAXON, komunikující pomocí protokolu CANopen. Motoru pohonu ovládá jednotka MAXON EPOS 24/5, zatáčení pak MAXON EPOS 24/1.

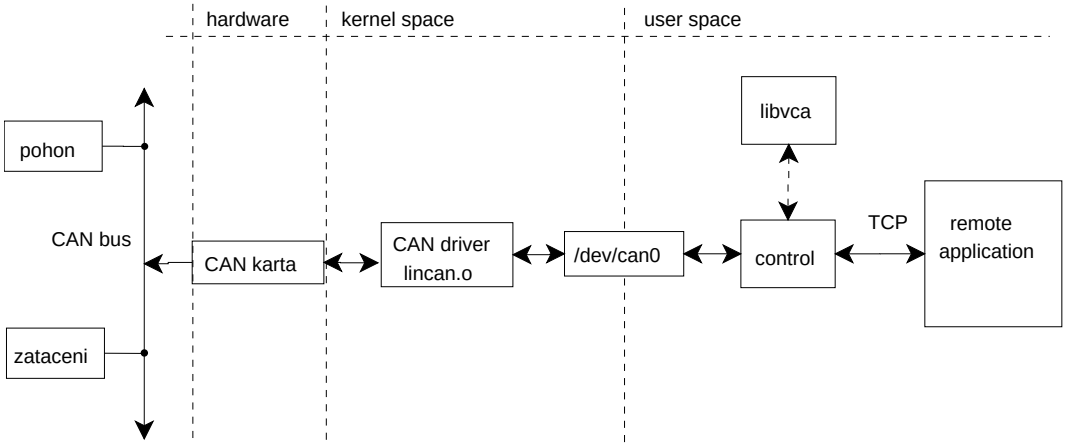
2.1.3 Bezdrátové připojení

Pro bezdrátové připojení robota do fakultní sítě byl využit Wi-Fi (Wireless Fidelity) multifunkční přístupový bod (AP - access point) WRT-312, v režimu bridge(most). Jednotka je nastavena v režimu DHCP klienta a od byla jí přiděléna veřejná adresa 147.32.87.218. LAN port řídící jednotky robota je pak s tímto modulem propojena UTP kabelem, proto jednotka nevyžaduje žádné ovladače.

Kapitola 3

Software robotu

3.1 Přehled



Obrázek 3.1: Blokové schéma softwaru robotu

Na obr. 3.1 je naznačena struktura softwaru robota. Jednotlivé bloky představují:

- **pohon, zataceni** CANopen nodes řídících jednotek motorů MAXON EPOS,
- **CAN karta** PC104+ kompatibilní CAN karta Digital Logic MSMCA104+,
- **CAN driver lincan.o** CAN driver linCAN jako jaderný modul,
- **/dev/can0** zařízení přes které se přistupuje k sběrnici CAN pomocí read, write a ioctl instrukcí,
- **control** řídící aplikace, viz. kapitola 3.3,

- **libvca** knihovna realizující CANopen protokol (součást projektu OCERA, viz. [16])
- **remote application** klientská aplikace, která komunikuje s řídící aplikací *Control* pomocí TCP komunikace

3.2 Operační systém

Na řídícím počítači robotu (PC/104) je nainstalován operační systém GNU/Linux (GNU je rekurzivní zkratka pro GNU's Not Unix) distribuce Gentoo s jádrem 2.6.18-r4 a 2.6.21-r8. Implicitně se zavádí jádro 2.6.21-r8.

3.2.1 Zavádění systému

3.2.1.1 Funkce BIOSu

Po zapnutí napájení robotu dojde ke startu řídící jednotky PC/104. Správu převezme BIOS (Basic Input/Outpus System) a provede test korektnosti připojeného hardware, tzv. *Power Self Test*. Především se provádí test paměti, grafické karty a procesoru.

Proběhnou-li testy v pořádku dojde ke spuštění z ROM paměti BIOSu zaváděcí program zvaný *bootstrap loader*, který podle nastavení hledá nejdříve boot sektor pevného disku(flash karty)nebo jiných zařízení jako CD, diskety.

Boot sektor je prvním sektorem disku, kde by měl být umístěn malý program (o velikost 512B) - zavaděč systému, tzv. *boot loader*, který dokáže spustit operařní systém. Většinou spouští složitější zavaděč (např. GRUB, LILO, XOSL). Boot sektory jsou označeny flagem 0xAA55 na pozici 0x1FE(510), tedy na posledních dvou bajtech boot sektoru. Podle toho zjišťují např. programy fdisk, qtparted zda-li je médium bootovatelné či nikoliv.

3.2.1.2 Zavadače systému

Pro systémy GNU/Linux jsou nejpoužívanějšími zavaděči GRUB a LILO. Vzhledem k tomu, že jde o poměrně rozsáhlé programy s velkými možnostmi konfigurace, tak se nevejdou celé do boot sektoru. Problém byl vyřešen tak, že se skládají z více stupňů(stage).

Program v boot sektoru je stage1. Ten pouze ví, kde je na disku uložen stage2 zavaděče a ten spustí. Stage2 pak vypíše promt a zavede vybraný operační systém, v případě linuxu

se jádro obvykle nachází v souboru /boot/vmlinuz-x.y.z, kde x.y.z je verze jádra.

Činnost zavaděče končí předáním řízení na tzv. *setup* rutinu jádra. Ta připraví přechod na protected mód, dekomprimuje jádro a předá mu řízení. Po inicializaci registrů a kontrole typu procesoru následuje vysokoúrovňová inicializace. Během ní se mj. provádí inicializace poplatné architektuře, inicializace datových struktur, inicializuje se systémová konzole, podpora dynamického zavádění modulů, počítá BogoMips, inicializace VFS (Virtual File System), VM (Virtual Memory manager), vyrovnávací cache, IPC (InterProcess Communication), quota (Subsystém limit a využití disků uživateli), provádí se kontroly na chyby HW a dělají protiopatření (např. f00f chyba P5), připraví se start plánovače "na příští chvíli", odstartuje vlákno (thread) pro start procesu init a přejde do čekací smyčky (idle loop).

3.2.1.3 Linuxové jádro

Vlástní linuxové jádro (též Linux, nebo kernel) je komprimované, a proto se musí před samotným spuštěním dekomprimovat. Jádro má několik hlavních úkolů:

- zprostředkovat hardware programům
- zajištění multitaskingu
- správa paměti
- spuštění programu init

3.2.1.4 Program init

Většina linuxových systémů používá způsob "Systém V" (čti Systém pět) inicializace. Obecně jádro po dokončení svého zavedení a své inicializaci spustí program /sbin/init s PID=1(Process identification). Úkolem *init*-u je, aby vše dále bylo správně spuštěno. Zařizuje připojení ("mountování" - mount) a případnou kontrolu souborových systémů a swapovacích oblastí, nastavení hodin, jména systému, startuje systémové démony, síťové služby a procesy pro přihlášení do systémů (getty/mingetty/xdm).

Init rozlišuje několik úrovní běhu (*runlevel*ů), každá úroveň může mít svojí množinu procesů, které v ní jsou startovány. V Gentoo existuje 7 runlevelů - 3 systémové *(sysinit, shutdown* a *reboot*) které se starají jak názvy napovídají o start, reset a vypínaní systému. Dále existují 4 uživatelské (*boot, default, nonetwork* a *single*).

Konfigurace podle kterého se init spouští jednotlivé inicializační skripty je soubor */etc/initab*. Komunikovat s programem init lze přes zařízení */dev/initctl* např. pomocí programu *telinit*. Další informace je možno dohledat v manuálových stránkách *init* a *inittab*.

3.2.1.4.1 Runlevely Jako první init vstoupí do systémové úrovně 0 a pokusí se připojit souborové systémy. To provede tak, že spustí */sbin/rc sysinit*. Poté vstoupí do úrovně *boot* a provede všechny skripty na které vedou symbolické odkazy z */etc/runlevels/boot*. O vlastní provedení se opět postará program *rc* tentokráte volaný */sbin/rc boot*.

Následuje vstup initu do úrovně 3 - default. V tomto levelu dojde ke spuštění všech skriptů na které vedou symbolické odkazy z */etc/runlevels/default* pomocí */sbin/rc default*.

Init systém Gentoo používá pro rozhodování o pořadí spouštění služeb strom závislostí. Protože jeho udržování je poměrně zdlouhavé a únavné, existují nástroje, které správu runlevelů a init skriptů usnadňují.

Nástrojem *rc-update* lze do runlevelu přidávat a odebírat skripty, a on se sám postará o zavolání skriptu *depscan.sh* pro znovu vytvoření stromu závislostí. Více informací o tomto nástroji lze nalézt v jeho manuálových stránkách.

3.2.1.4.2 Standartní init skripty Inicializační skripty se v distribuci Gentoo nacházejí v adresáři */etc/init.d/*. Konfigurační soubory k těmto skriptům jsou pak v adresáři */etc/conf.d/*. Nejdůležitějšími skripty jsou:

- **clock** nastaví hodiny réálného času
- **consolefont** nastaví font terminálu
- **checkfs** zkontroluje systém souborů
- **localmount** provede připojení lakálních zařízení
- **local** spustí/ukončí lokální aplikace
- **modules** načte jaderné moduly
- **net** aktivuje network adaptéry
- a další.

3.2.1.4.3 Vlastní init skripty V rámci práce na tomto projektu byly vytvořeny init skripty *cfroot* a *k09mount* a modifikována konfigurace skriptu *local.start*.

CF RooT Operační systémy GNU/Linux jsou velmi náchylné na chyby způsobené vypadkem napájení. Taková situace však v případě mobilního robota napájeného z akumulátorů může nastat. Proto bylo třeba napsat init skript, který tento problém pomůže vyřešit.

Principiálně jde o to, že se *root* souborový systém připojí pouze pro čtení (read-only), poté se vytvoří v paměti virtuální dočasný oddíl kam se zkopíruje obsah adresářů */etc/* a */var/*.

Pak se provede přepojení adresářů */etc/* do */etc-cf/* a */var/* do */var-cf/*. Kopie adresářů /etc/ a /var/ v paměti se pak připojí pomocí *mount –bind* namísto původních. Dále už systém nadále pracuje s těmito adresáři jako kdyby byly na fyzickém disku. Při výpadku napájení nedojde k poškození, protože systém v okamžiku výpadku nepracoval s daty uloženými na disku, nýbrž jen s jejich kopií v operační paměti.

Povzbudívým důsledkem je také to, že rychlost celého systému se zvýšila - nejčastěji používané programy jsou přímo v operační paměti a nemusí se tedy natahovat z pomalého disku.

Pokud je potřeba na disk něco zapsat je nutné jej odpojit a znovu připojit s povoleným zápisem. Pak disk opět odpojit, připojit jen pro čtení a znovu spustit tento skript. Konstrukce skriptu vychází ze skriptu cfroot z distribuce Slackware, uložen je v souboru */etc/init.d/cfroot*.

K09 Mount Pro snadnější vývoj řídící aplikace byl napsán skript, zajišťující přes *nfs* (Network File System) připojení adresáře s vyvýjeným projektem na pracovní stanici přímo k robotu. Tím je umožněn vývoj řídících algoritmů mimo řídící jednotku robota. Pokud se navíc do pracovní stanice uloží zdrojové kódy jádra běžícího na řídícím počítači(PC/104), je možné projekt zkompilovat na pracovní stanici a na robotu spustit až hotový binární program. Skript se nachází v adresáři */etc/init.d/k09mount*.

Local.start Init skript *local* je při startu řídící jednotky volán kód uložený v */etc/conf.d/local.st*
Ten provádí:

- vložení ovladače lincan do jádra:
  insmod /home/hero/control/modules/lincan hw=pcan_pci io=0

- spuštění na pozadí řídící aplikaci *Control*, viz. kapitola 3.3:
  /home/hero/control/bin/control

3.2.2 Konfigurace systému

Všechna konfigurace se v systémech GNU/Linux nachází v souborech a to většinou v adresáři */etc/*. Zde je uveden seznam několika nejdůležitějších kroků, které bylo třeba nastavit. Konfigurační soubory jsou obsahem přiloženého CD.

3.2.2.1 Rozdělení disku

Pokud při běhu systému dojde ke stavu, že dochází volná paměť, je nutné buď paměť uvolnit např. násilným ukončením procesu, který o přidělení další paměťi požádá (standartní nastavení jádra). Nebo lze použit metody *swapování*. Tedy kdy se data, která se již do operační paměti nevejdou, začnou ukládat na pomalejší médium např pevný disk.

V operačních systémech GNU/Linux pro toto slouží zvláštní oddíly tzv. *swap*. Doporučená velikost *swap*u je cca. $\frac{1}{2}$ až $\frac{3}{2}$ velikosti fyzické operační paměti.

Proto byla paměťová karta uchovávající operační systém rozdělena programem fdisk na dva oddíly. První byl použit jako swap, druhý naformátován souborovým systémem ext3 kde byl uložen vlastní operační systém spolu s řídícími programy robota.

3.2.2.2 Konfigurace zavaděče

Pro funkci zavadaděče operačního sytému byl použit *GRUB*. Pro úspěšné zavedení systému je třeba sdělit *GRUB*u, z jakého oddílu a jaké jádro má být zavedeno. To je možno nastavit v souboru */boot/grub/grub.conf*.

3.2.2.3 Konfigurace kompilací

V distribuci Gentoo GNU/Linux je většina balíčků včetně samotného jádra poskytována v podobě zdrojových kódů, které je potřeba před použitím zkompilovat. Pro překlad je použit nejrozšířenější překladač gcc, který pokud je správně nastaven dokáže výsledný binární soubor optimalizovat pro použitý procesor.

Pro globální nastavení kompilačních proměnných slouží soubor */etc/make.conf*, pomocí něhož lze též nastavit tzv. USE flagy, tedy informace správci balíčků (emerge) jaké závislosti, chceme instalovaným balíčkům povolit, nebo naopak zakázat. Více informací o souboru *make.conf* lze nalézt v jeho manuálových stránkách.

3.2.2.4 Připojování zařízení

Jako paměťové médium pro uložení operačního systému spolu s řídící aplikací byla použita karta Compact Flash II, pro kterou má použitý řídící počítač (PC/104) slot.

Ten je připojen na primární ovladač IDE jako zařízení typu slave. V operačních systémech GNU/Linux tomu pak odpovídá blokové zařízení */dev/hdb*.

Seznam oddílů, které mají být automaticky při startu připojovány init skriptem *localmount* je uveden v souboru */etc/fstab*.

Po startu PC/104 je třeba připojit root systém z oddílu /dev/hdb2 a zajistit inicializaci swap oddílu /dev/hdb1.

3.2.2.5 Konfigurace jádra

Distribuce Gentoo GNU/Linux je založena na tom, že si uživatel musí zkomilovat i jádro operačního systému ze zdrojových kódů. Má to nespornou výhodu v tom, že lze jádro nastavit tak, že podporuje jen toho, co systém opravdu vyžaduje a výsledkem je pak malé a rychlé jádro.

Problém nastává, pokud není znám např. hardware cílového zařízení. V takových případech je možno použít nástrojů jako *genkernel* pro automatické vytvoření jádra, nebo programů z balíčku *pciutils* pro zjištění základních informací o hardware.

Zdrojové kódy jádra jsou obvykle umístěny v adresáři */usr/src/linux-x.y.z-distribuce-revize*, kde x.y.z reprezentuje verzi jádra. Na tento adresář ukazuje symbolický odkaz */usr/src/linux*.

Vlastní konfiguraci jádra je možné zahájit spuštěním jako uživatel *root make menuconfig* v adresáři */usr/src/linux*. Tím vzikne jednoduchá grafická aplikace pomocí níž, je možno nastavit všechny parametry jádra.

Výsledkem konfigurace je pak soubor */usr/src/linux/.config*. Tento v textové podobě reprezentuje veškerou konfiguraci jádra. Při přechodu na novou verzi, stačí tento soubor zkopírovat k novým zdrojovým kódum a tak vytvořit základ nastavení nového jádra. Vlastní kompilaci jádra zajistíme spuštěním *make*. Jaderné moduly pak lze naistalovat pomocí *make modules_install*.

Výsledek kompilace - jádro, je uloženo v */usr/src/linux/arch/X/bzImage*, kde X odpovídá cílové architektuře(v případě PC/104 jde o architekturu i386).

Tento soubor je třeba nakopírovat do adresáře */boot/*, a pokud je potřeba přepsat konfiguraci v *grub.conf*. Nebo lepší způsob je po kompilaci jádra zavolat *make install* a ten se o toto postará. Ten totiž obraz jádra zkopíruje do */boot/* a uloží ho jako *vmlinuz-*

x.y.z, kde x.y.z odpovídá verzi jádra. Zároveň do */boot/* zkopíruje konfigurační soubor *.config* jako *config-x.y.z*. Dále vytvoří symbolický link */boot/vmlinuz* odkazující na nové jádro */boot/vmlinuz-x.y.z*.

V souboru */boot/grub.conf* je možné se odkazovat na tento symbolický link a není tedy nutné po každé kompilaci jádra tento soubor měnit.

Navíc *make install*, udržuje jednoduchou dvou úrovňovou správu verzí jader. Pokud se provede nová kompilace jádra a zavolá se opět make install, dojde k přejmenování staré verze na xxx.old (xxx odpovídá vmlinuz-x.y.z, config-x.y.z) a místo ní je do */boot/* nakopírovaná nová verze podle výše uvedeného schématu.

Celý proces kompilace a instalace jádra lze pak zapsat jako:
make && make modules_install && make install.

To jestli je jádro nastaveno správně lze zjitit jedině tak, že se ho loader pokusí zavést. Pokud se při pokusu o zavedení nového jádra objeví chyby (pravděpodobně kernel panic), je možné nabootovat do starého jádra a pokusit se chyby opravit.

3.3 Řídící aplikace Control

Úkolem této práce bylo vytvořit knihovní funkce a ukázkovou řídící aplikaci pro ruční řízení robota HERO. Aplikace by měla být vzhledem k malému výkonu řídící jednotky nenáročná jak na výpočetní výkon, tak i na paměť. Po spuštění PC/104 je vytořený program *Control* spouštěn inicializačním skriptem *local.start* viz. kapitola 3.2.1.4.3.

3.3.1 Princip

Aplikace *control* pracuje jako server vykonávající příkazy klientů přicházející po síti Internet pomocí TCP soketů. Příkazy a požadavky klientů jsou zpracovávány a posílány pomocí protokolu CANopen řídícím jednotkám motorů.

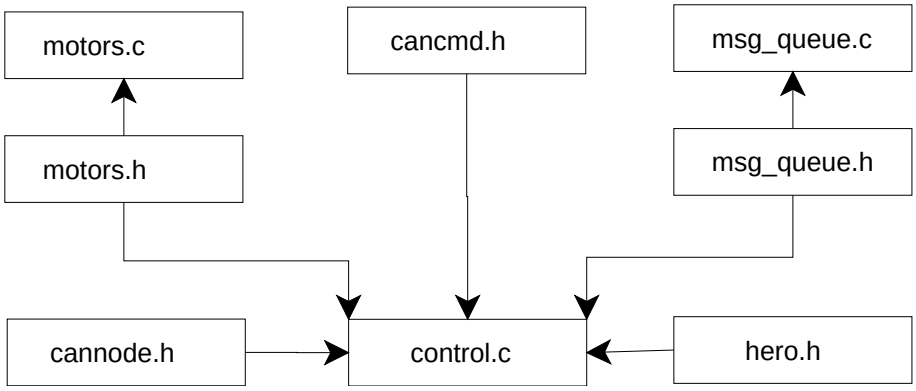
Serveru byla přiřazena IP adresa 147.32.87.218 a poslouchá na portu 6789.

Pro řízení motorů byly použity jadnotky MAXON EPOS 24/1 a 24/5. Jejich technické parametry spolu se začleněním do hardwaru robota je možné nalézt v [1]. Komunikaci mezi serverem a klientem popisuje kapitola 4.4. Více o protokolu CANopen lze nalézt např. v [14].

3.3.2 Struktura

Řídící aplikace byla vytvářena s ohledem na to, aby její jednotlivé části bylo možné použít v dalším vývoji projektu. Zdrojové kódy byly psány v jazyce C, který dovoluje psaní rychlých a efektivních algoritmů. Strukturu zdrojových kódů naznačuje obr. 3.2.

Bližší popis jednotlivých souborů a funkcí lze nalézt v dokumentaci, která je spolu se zdrojovými kódy obsažena na přiloženém CD.



Obrázek 3.2: Struktura zdrojových souborů aplikace *Control*

3.3.2.1 Zdrojové soubory

3.3.2.1.1 cancmd.h Definuje strukturu cancmd uchovávající informace o CAN zprávě.

3.3.2.1.2 cannode.h Definuje struktury canNode pro komunikaci po CAN a canSDONode pro komunikaci po CANopen.

3.3.2.1.3 msq_queue.c Obsahuje funkce pro manipulaci s FIFO frontou CAN zpráv. Funkce jsou uvedeny v tabulka 3.1.

Parametr	**Popis**
init_queue	inicializuje frontu zpráv
enqueue	zařadí novou zprávu do fronty
dequeue	vyjme zprávu z fronty
empty	testuje, jetli je fronta prázdná

Tabulka 3.1: Funkce obsažené v msgqueue.c

3.3.2.1.4 hero.h Definuje strukturu robot. Ta obsahuje vnitřní stavové proměnné robota, a jejich mutexy. Význam nejdůležitějších proměnných uvádí tabulka 3.2.

Proměnná	**Popis**
speed	rychlost hřídele motoru pohonu
direction	poloha hřídele motoru zatáčení
voltage	napětí akumulátorů
light	stav světel
motorSpeed	canSDONode reprezentující pohon
motorDirection	canSDONode reprezentující zatáčení

Tabulka 3.2: Důležité vnitřní proměnné obsažené v hero.h

3.3.2.1.5 motors.h Obsahuje základní definice a hodnoty parametrů týkající se řídících jednotek MAXON EPOS. Přehled nejdůležitějších parametrů uvádí tabulka 3.3[1].

Parametr	**Popis**
CAN_BITRATE_OUR	přenosová rychlost sběrnice CAN (default 1 Mbit)
MOTOR_TYPE_*	typ motoru
CURRENT_LIMIT_*	maximální proud motoru
POLE_PAIRS_*	počet pólových dvojic motoru
THERMAL_TIME_*	maximální doba toku maximálního proudu
HOME_POSITION_ZERO	poloha nuly pro homeing preces
HOME_METHOD_*	metoda homeing procesu
MAXIMAL_VELOCITY_*	maximální rychlost
PULSE_PER_TURN	počet pulzů za otáčku od senzoru polohy
RATIO_*	převodový poměr použitých převodovek

Tabulka 3.3: Nejdůležitější parametry v motor.h

3.3.2.1.6 motors.c Obsahuje základní funkce pro ovládání řídících jednotek motorů MAXON EPOS pomocí protokolu CANopen. Pomocí hlavičkového souboru motors.h

[1]Pro realizaci zatáčení a pohonu robota byly použity rozdílné motory, proto se parametry jednotlivých jednotek liší. V tabulce je pak uveden jen jeden parametr s postfixem "_*" znamenajícím SPEED nebo DIRECTION, tedy nastavení týkající se jednotky pohonu nebo zatáčení.

poskytuje jakékoliv aplikaci funkce pro ovládání robota pomocí protokolu CANopen s využitím MAXON EPOS jednotek. Nejdůležitější funkce ukazuje tabulka 3.4.

Funkce	**Popis**
m_InitSpeedEPOS	inicializuje jednotku pohonu
m_InitDirectionEPOS	inicializuje jednotku zatáčení
m_Quick_Stop	provede rychlé zastavení chodu jednotky
m_Fault_Reset	provede nulování chybových hlášení
m_StopHoming	provede zrušení homeing procesu
m_StopVelocity	zastaví otáčení motoru
m_StopPosition	zastaví proces polohování motoru
m_setAmplifierState	zapne nebo vypne jednotku dle parametru
m_StartHomingDirection	zahájí homeing proces motoru zatáčení
m_StartVelocitySpeed	zahájí otáčení motoru danou rychlostí
m_setPositionSpeed	zahájí polohování motoru pohonu dle parametru
m_setPositionDirection	zahájí polohování motoru zatáčení dle parametru
m_TargetReached	poskytuje informaci, zda byla poloha/rychlost dosažena
m_ReadPosition	zjistí aktuální polohu hřídele motoru
m_ReadVelocity	zjistí aktuální rychlost hřídele motoru
m_ReadCurrent	zjistí aktuální proud tekoucí motorem
m_setDigitalOutput	nastaví hodnoty digitálních výstupů jednotky
m_getDigitalInput	zjistí stav digitálních vstupů jednotky
m_getAnalogInput	zjistí stav analogových vstupů jednotky
send_to_can	odešle zprávu na sběrnici CAN
sendSDO	čeká dokud není přenos SDO objektů ukončen
m_initCAN	inicializuje komunikaci motorů s CAN rozhraním
m_initMotor	nastaví filtr a inicializuje strukturu SDO FSM pro daný motor
m_releaseMotor	odstraní strukturu SDO FSM pro daný motor

Tabulka 3.4: Nejdůležitější funkce v motors.c

3.3.2.1.7 control.c Osahuje vlastní algorimus řízení robota. Definuje globální parametry programu. Nejdůležitější funkce obsažené v tomto souboru uvádí tabulka 3.5, parametry a jejich význam pak tabulka 3.6.

Funkce	**Popis**
main	inicializuje komunikaci, spuuští vlákna
thr_comm_server	vlákno TCP serveru
thr_refresh_job	vlákno provádějící periodický refresh vnitřních proměnných robota
thr_can_command	vlákno posílající zprávy na CAN sběrnici
execute_client_cmd	parsování příchozích zpráv od klientů a jejich vykonávání
start_control	zapne řídící jednotky
stop_control	vypne řídící jednotky
set_speed	nastaví rychlost robota dle parametru [ot/min]
set_direction	nastaví směr robota dle parametru [°]
set_light	nastaví stav světel podle parametru
get_voltage	zjistí napětí akumutátorů [V]
get_speed	zjistí rychlost robota [ot/min]
get_direction	zjistí směr robota [°]
join_thr	čeká na ukončení všech vláken

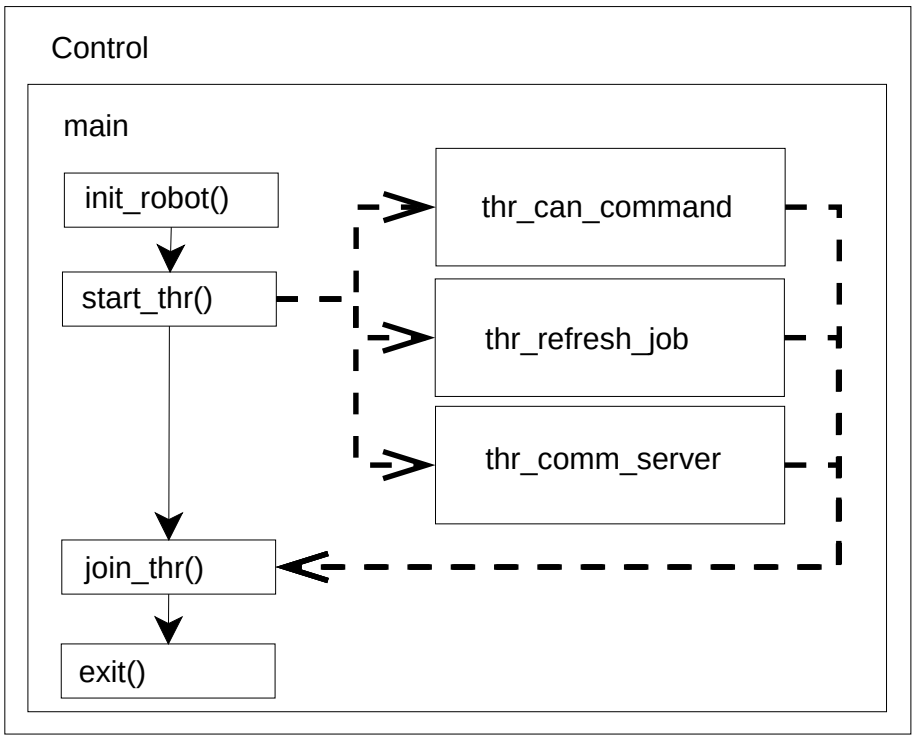
Tabulka 3.5: Nejdůležitější funkce obsažené v souboru control.c

Parametr	**Popis**
DELIMITER	znak konce zprávy ('\n')
CAN_IFACE	CAN zařízení (např. */dev/can0*)
PORT	číslo portu na kterém server poslouchá
REFRESH_TIME	perioda refresh vlákna [s]
MAX_CLIENTS	maximální počet připojených klientů

Tabulka 3.6: Nejdůležitější parametry obsažené v souboru control.c

3.3.3 Běh programu

Aplikace *control* byla napsána jako multivláknová. Bylo tedy nutné zabezpečit všechny sdílené proměnné mutexy(zámky) a funkce psát jako tzv. *thread-safe*, tedy před přístupem k proměnné se pokusit zamknout příslušný mutex, poté vykonat požadovanou operaci nad proměnnou a následně odemknout příslušný zámek a tím umožnit dalším vláknům přístup k proměnné.



Obrázek 3.3: Blokové schéma řídící aplikace

Program *control* se skládá z jediného procesu (obr. 3.3), který v sobě integruje 4 vlákna, která sdílí společný paměťový prostor a filedeskriptory.

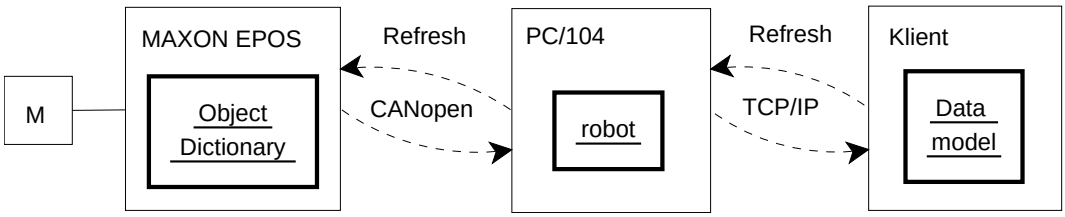
Hlavní je vlákno *main*, které nejprve provede otevření a inicilizaci CAN zařízení, dále pak inicializaci všech zámků. Pokud je předešlý krok úspěšně splněn, jsou postupně spouštěna vlákna:

- **thr_can_command** - odesílání CAN zpráv z FIFO fronty,
- **thr_refresh_job** - periodická aktualizace vnitřních proměnných robota,
- **thr_comm_server** - obsluha požadavků přicházejících od klientů po TCP.

Následně přechází hlavní vlákno do stavu *wait*, kdy čeká na skončení ostatních vláken. Teprve pak program končí.
Pokud program v nějakém bodě selže je vypsáno příslušné chybové hlášení a program je ukončen.

3.3.4 Refresh dat

Vnitřní proměnné robota jsou uloženy v paměti PC104 ve struktuře *robot*. Ta je periodicky aktualizována vláknem *thr_refresh_job*, které načítá aktuální hodnoty objektů z Object Dictionary (slovník objektů CANopen) řídících jednotek motorů. Periodu obnovování dat lze nastavit pomocí parametru *REFRESH_TIME* v souboru *control.c*.



Obrázek 3.4: Znázornění obnovování dat

Klientská aplikace pak pro obnovování svého lokálního datového modelu robota používá soketové komunikace popsané v kapitola 4.4. Na dotaz o stavu konkrétní vnitřní proměnné robota, server odesílá právě aktuální hodnotu proměnné uložené ve struktuře *robot*.

Perioda obnovování datového modelu klienta by měla být volena s ohledem na hodnotu parametru *REFRESH_TIME*. Volbou periody menší než tato hodnota klient dostává duplicitní informace a zbytečně zatěžuje server svými požadavky. Příliš velká perioda zase přináší do systému problémy podobné dopravnímu zpoždění z regulačních obvodů. Proto je nutné volit obnovovací periodu v rozumném intervalu odpovídajícím požadované činnosti.
Např. pro ruční řízení robota připojeného do bezdrátové sítě wifi je optimální perioda 0,5s. Perioda vlákna *thr_refresh_job* by pak měla být stejná nebo lépe menší.

3.3.5 Kompilace

Pro kompilaci programu byl použit kompilační systém OMK dovolující snadnou konfiguraci parametrů. Ty se nacházejí v souboru Makefile.omk v adresáři se zdrojovými kódy. Pro úspěšnou kompilaci jsou vyžadovány všechny zdrojové kódy programu a externí knihovny *libulut* a *libvca*, které jsou součástí CAN driveru linCAN a nacházejí se v řídící jednotce PC/104 na */home/hero/control/lib*.

Pokud je třeba kompilovat na jinou verzi jádra než je právě běžící, je třeba zadat

pomocí parametru LINUX_DIR cestu ke zdrojovým kódům verze jádra na kterém aplikace poběží.

Pak již stačí spustit kompilační proces pomocí *make*. Tím se vytvoří dva nové adresáře: *_build* a *_compiled*. První obsahuje objektové soubory *.o jednotlivých zdrojových kódů, druhý pak výsledek kompilace po slinkování všech objektových souborů a knihoven.

V adresáři *_compiled/bin/* se pak nachází výsledná aplikace. Adresář *_compiled/include/* obsahuje použité hlavičkové soubory.

Kapitola 4

Grafická aplikace pro ruční řízení robotu

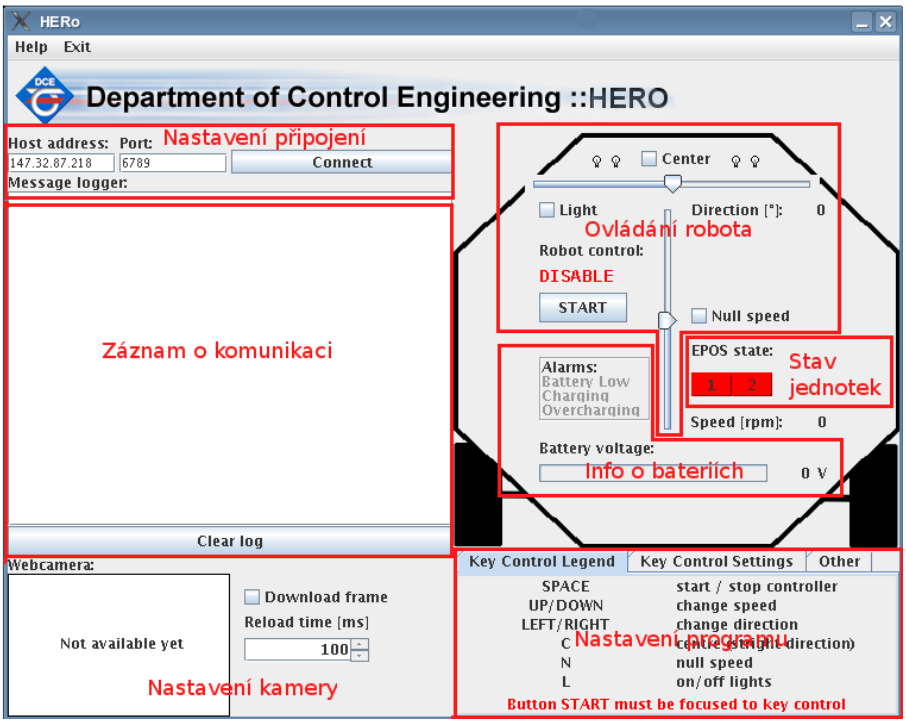
Jedním z hlavních bodů bakalářské práce bylo vytvořit aplikaci pro ruční ovládání robota, která by běžela jak v systému Linux, tak ve Windows.

Aplikace byla napsána v jazyce JAVA za pomoci vývojového nástroje NetBeans 5.5. Využívá grafických prvků poskytovaných knihovnou Swing a pro komunikaci se serverem třídy Socket.

4.1 Přehled

4.1.1 Vzhled

Na obr. 4.1 je znázorněn pohled na vytvořenou aplikaci po spuštění. V její levé části se nachází prvky týkající se připojení aplikace k serveru. V pravé pak jsou zobrazovány aktuální hodnoty měřených veličin jako rychlost robotu nebo napětí baterií. Dále se tam pak nachází prvky pro ovládání funkcí robota. V pravé spodní části je pak umístěn panel s nastavením programu.



Obrázek 4.1: Grafická aplikace pro ruční řízení s popisem důležitých oblastí

4.1.1.1 Nastavení připojení

Zde je třeba zadat IP adresu serveru a číslo portu, na kterém server poslouchá. Kliknutím na tlačítko Connect se provede pokus o připojení k požadovaném serveru na zadaném portu. Po kliknutí na tlačítko Disconnect dojde k odpojení klienta od serveru.

4.1.1.2 Záznam o komunikaci

Zde se zobrazuje průběh celé komunikace. Vypisují se zde tedy veškeré příchozí i odchozí zprávy a veškerá chybová hlášení programu. Stisknutím tlačítka Clear Log dojde k vymazání záznamu o komunikaci.

4.1.1.3 Nastavení kamery

Zde je možno zapnout / vypnout snímání scény kamerou. Také je možné nastavit periodu snímání jednotlivých obrázků. Tato funkce není prozatím podprována řídící aplikací v robotu.

4.1.1.4 Ovládání robota

Zde se nachází soubor několika grafických prvků použitých pro změnu vnitřních proměnných robota. Také jsou zde zobrazovány aktuální hodnoty rychlosti a směru robota. Nápis ENABLE/DISABLE indikuje stav motorových jednotek (zapnuty/vypnuty).

Více o tom jak ovládat robota popisuje kapitola 4.3.

4.1.1.5 Informace o bateriích

Zde se zobrazují důležité zprávy o bateriích. Jsou to alarmy jako vybitá baterie, přebíjení baterií a nabíjení baterií. Také je zde zobrazován aktualní stav napětí baterií.

4.1.1.6 Stav jednotek

Jednotky motorů se mohou nacházet ve dvou stavech indikovanými LED diodami na jejich pouzdře jak uvádí tabulka 4.1. Jednotkám byly pro jednoduchost grafické aplikace přiřazena čísla, jak ukazuje tabulka 4.2.

Stav	**Význam**	**Barva LED**
normální	běžný provoz	zelená
chybový	nastala chyba v jednotce	červená

Tabulka 4.1: Indikace stavů jednotek motorů

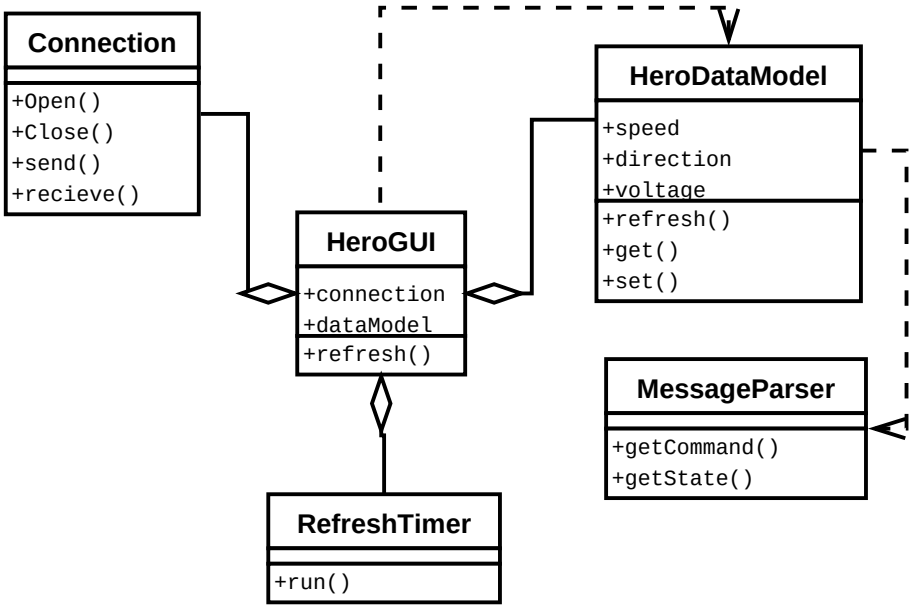
Jednotka	**Identifikační číslo**
jednotka pohonu	1
jednotka směru	2

Tabulka 4.2: Přiřazení čísel jednotkám v GUI

4.1.1.7 Nastavení programu

V této sekci je možno nastavit některé parametry programu. Podrobně se o tomto zmiňuje kapitola 4.2.

4.1.2 Struktura programu



Obrázek 4.2: UML diagram apliakce

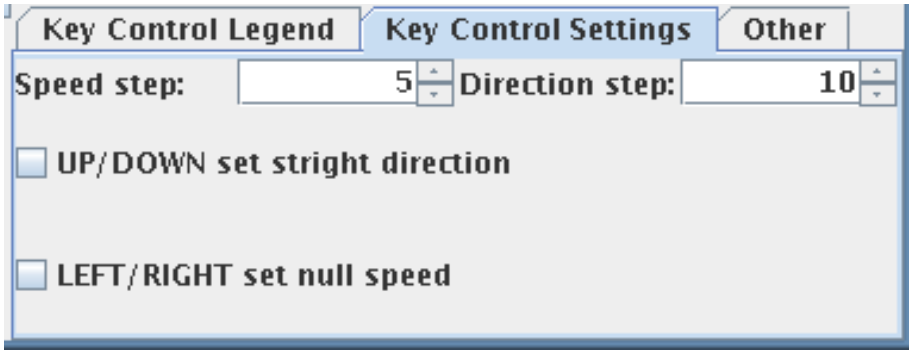
Na obr. 4.2 je zobrazen Class diagram aplikace. Ta byla vytvořena podle návrhového vzoru *MVC* (Model View Controller), kdy jsou v aplikaci data, řízení toku dat a jejich zobrazování navzájem oddělené. Tento styl umožňuje vytvářet jednoduché a pro čtení kódu průhledné aplikace. Více o návrhových vzorech je možné nalézt v [4].

Třída HeroGUI zde vystupuje současně jako řadič a zobrazovač třídy HeroDataModel, která je souborem dat reprezentující lokální model vnitřních proměnných robota. HeroGUI tedy zachytává a obsluhuje všechny události přicházející jak od klávesnice, myši tak i od připojeného socketu.

Třída RefreshTimer reprezentuje vlákno běžící na pozadí, které provádí periodickou aktualizaci vnitřního stavu robota, tedy načítá aktuální rychlost, napětí baterií atd. Periodu vzorkování tohoto vlákna je možno nastavit viz. kapitola 4.2.

MessageParser je, jak název napovídá, třída obsahující funkce pro zpracování příchozích zpráv od serveru. Connection je třída reprezentující vlastní socketové spojení a je navržená návrhovým vzorem *Single Tone*, tj. při běhu aplikace tedy může býti pouze jediná instance.

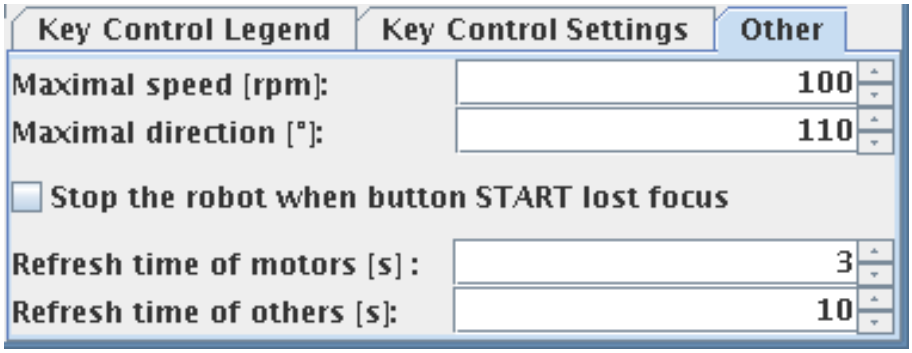
4.2 Nastavení



Obrázek 4.3: Záložka Key Control Setiings

Záložka Key Control Settings (obr. 4.3) umožňuje:

- nastavit citlivost klávesového řízení (krok rychlosti a směru)
- zvolit, zda se při stisku kurzorových kláves UP / DOWN nastaví přímý směr
- zvolit, zda se při stisku kurzorových kláves LEFT / RIGHT nastaví nulovou rychlost



Obrázek 4.4: Záložka Other

Záložka Other (obr. 4.4) umožňuje:

- nastavit maximální rychlost robotu [ot/min]
- nastavit maximální úhel směru [°]
- zvolit, zda se má robot zastavit jestliže tlačítko START přestane být aktivní

- nastavit periodu aktualizace paramerů motorů (rychlost, směr)
- nastavit periodu aktualizace ostatních parametrů (napětí beterií,...)

4.3 Ovládání robotu

Po úspěšném připojení klienta k serveru může být robot řízen buď klávesami nebo grafickými ovládacími prvky.

Pro ovládání klávesami musí být tlačítko START aktivní (focusable), tj. přijímat zprávy od klávesnice. Po stisku některé z kláves je vygenerována událost *KeyPressed*, která je obsloužena funkcí *StartKeyPressed*, zde se vyhodnotí o jakou klávesu šlo a podle toho se vykoná operace jak ukazuje tabulka 4.3. Ostatní klávesy jsou ignorovány.

Aby robot vykonával příkazy týkající se změny rychlosti a směru, je nutné aby byly zapnuté jednotky motorů. Je tedy nutné nejprve poslat příkaz zapnutí jednotek.

Ovládací klávesa	**Ovládací prvek**	**Funkce**
mezerník	tlačítko START	zapíná / vypíná jednotky motorů
kurzorové klávesy UP / DOWN	svislý slider	mění rychlost robotu
kurzorové klávesy LEFT / RIGHT	vodorovný slider	mění směr jízdy
c	zaškrtávací tlačítko center	nastaví přímý směr jízdy
n	zaškrtávací tlačítko null speed	zastaví robota
l	zaškrtávací tlačítko light	zapíná / vypíná světla

Tabulka 4.3: Ovládání robotu

4.4 Komunikace mezi klientem a serverem

Komunikace mezi klientem (grafickou aplikací) a serverem (aplikace běžící na robotu) probíhá pomocí socketů protokolem TCP (Transmision Control Protocol). Jde tedy o potrvrzovaný přenos zpráv pomocí paketů.

Vlastní řízení pak probíhá na základě jednoduchých textových zpráv v znakové sadě *ASCII*. To umožňuje robota řídit např. pomocí služby *telnet* a testovat tak základní funkce robotu lokálně přímo na řídícím počítači PC/104. Všechny zprávy musí být zakončeny standartním ukončovacím znakem nového řádku ('\n'), případně znaky '\r\n' pro platformu Windows.

Koncepce komunikace mezi klientem a serverem je založena na dotazech a příkazech. Komunikaci vždy zahajuje klient a to buď zasláním příkazu o změně vnitřní proměnné robota nebo vyžádání stavu vnitřní proměnné robota pomocí dotazu, např. o stavu baterií.

Na příkazy klienta server pouze změní své vnitřní proměnné, na dotazy naopak posílá klientovy požádované údaje. Seznam příkazů akceptovaných serverem a jejich význam je uvádí tabulka 4.4. Výpis dotazů klienta na server spolu s jejich významem naznačuje tabulka 4.5. Odpovědi na dotazy s popisem jejich významu ukazuje tabulka 4.6.

Příkaz	**Význam příkazu**
start_con	zapnout jednotky motorů
stop_con	vypnout jednotky motorů
speed=N	nastavit rychlost robotu na N [ot/min]
direction=N	nastavit směr [°]
light_on	rozsvítir LED diody
light_off	zhasnout LED diody
downcam_on	snímej scénu kamerou
downcam_off	nesnímej scénu kamerou

Tabulka 4.4: Příkazy odesílané klientem a jejich význam

Dotaz	**Význam dotazu**
state_control	zjisti jestli jsou motorové jednotky zapnuté
state_epos_speed	zjisti stav jednotky pohonu
state_epos_direction	zjisti stav jednotky otáčení
state_speed	zjisti rychlost robota
state_direction	zjisti směr robota
state_voltage	zjisti napětí beterií
state_bat	zjisti stav nabíjecích obvodů
state_light	zjisti stav světel
state_downcam	zjisti stav kamery

Tabulka 4.5: Dotazy odesílané klientem a jejich význam

Dotaz	**Odpověď od serveru**	**Význam odpovědi**
state_control	control_on	jednotky jsou zapnuté
	control_off	jednotky jsou vypnuté
state_epos_speed	epos_speed_on	jednotka pohonu je v normálním režimu
	epos_speed$_err$	jednotka pohonu je v chybovém režimu
state_epos_direction	epos_direction_on	jednotka směru je v normálním režimu
	epos_direction_err	jednotka směru je v chybovém režimu
state_speed	speed=N	aktuální rychlost je N [ot/min]
state_direction	directon=N	aktuální rychlost je N [°]
state_voltage	voltage=x.yy	napětí baterií je x.yy
state_bat	bat_charging	baterie se nabíjejí
	bat_low	baterie jsou vybité
	bat_overcharging	baterie se přebíjí
state_light	light_on	světla jsou zapnutá
	light_off	světla jsou vypnutá
state_downcam	downcam_on	kamera snímá scénu
	downcam_off	kamera nesmímá scénu

Tabulka 4.6: Dotazy odesílané klientem význam odpovědí od serveru

Kapitola 5

ZigBee ovladač robotu

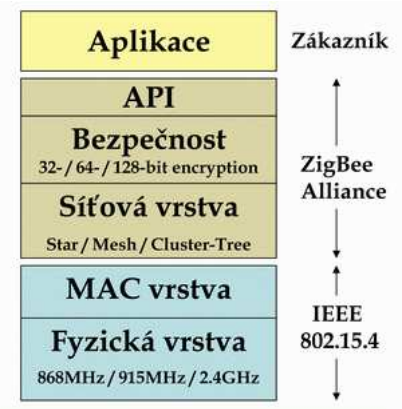
V rámci této práce byla navžena struktura systému umožňující ruční řízení robota HERO pomocí technologie ZigBee. Předpokladem je nejméně dvojice ZigBee modulů PAN802154 s nahraným firmware Wireless Modem (Z-Stack 1.0.0) popsaným v [8].
Z požadavků na řízené zařízení je nutné zdůraznit nutnost přítomnosti sériového rozhraní RS232, běžící řídící serverové aplikace očekávající příkazy z nadřazeného terminálu pomocí soketů a běžící aplikace *ZigClient* jehož struktura je popsána ukazuje kapitola 5.3.2.2. Na straně řídící stanice je nutná přítomnost sériové linky a běžicí aplikace *ZigServer* popsaná v kapitola 5.3.2.1.

5.1 Bezdrátová technologie ZigBee

Komunikačná technologie ZigBee je nový nízkorychlostní standard bezdrátové komunikace, zaměřený především na oblasti automatizace a řídící techniky. Jedná se o bezdrátovou komunikační technologii schválenou jako mezinárodní standard nadnárodní organizací ZigBee Alliance a standardizační organizací IEEE.
Tato perspektivní bezdrátová komunikační technologie najde uplatnění zejména v takových oborech, jako jsou řízení budov, dálkové ovládání, monitorování a diagnostika zařízení, vzdálené čtení měřených hodnot, počítačové periferie nebo spotřební elektronika.

5.1.1 Struktura komunikačního standardu

Stejně jako každý jiný komunikační standard i ZigBee lze popsat *OSI* modelem. Ten lze rozdělit do třech základních bloků podle toho kým jsou definovány (viz. obr. 5.1).



Obrázek 5.1: OSI model komunikačního protokolu ZigBee

5.1.1.1 IEEE 802.15.4

Standard IEEE 802.15.4 definuje fyzickou a linkovou vrstvu (MAC vrstvu) standardu ZigBee. Fyzická vrstva určuje způsob konkrétní fyzické bezdrátové komunikace, jíž bylo přiděleno několik radiových pásem :

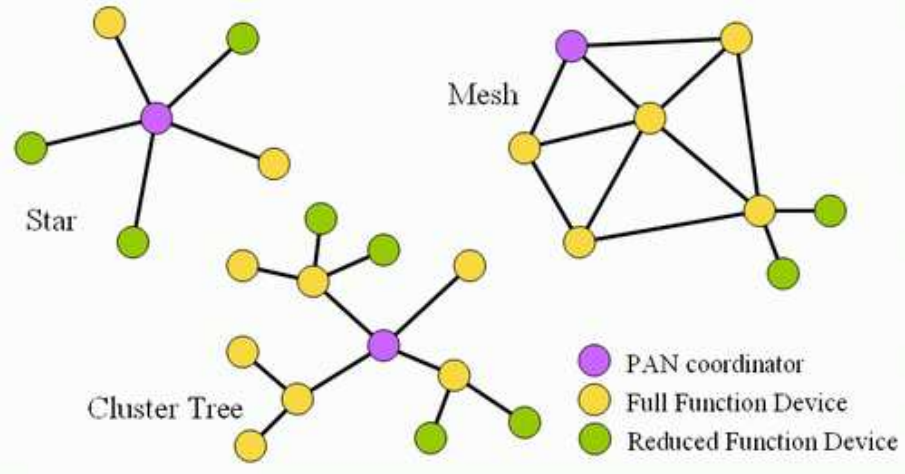
- pásmo ISM 2.4 GHz, 16 kanálů, přenosová rychlost 250kb/s, definováno celosvětově
- pásmo 915 MHz, 10 kanálů, přenosová rychlost 40kb/s, definováno pro americký kontinent
- pásmo 868 MHz, 1 kanál, přenosová rychlost 20kb/s, definováno pro Evropu

Pro přenos se datový signál moduluje metodou O-QPSK a vzduchem přenáší metodou DSSS (Direct Sequence Spread Spectrum). Pro přístup na kanál se využívá metody CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance and optional time slotting).

MAC vrstva (linková vrstva) definuje již samotnou komunikaci mezi jednotlivými zařízeními(uzly sítě)prostřednictvím rámců. Konkrétně jsou definovány čtyři typy komunikačních rámců využívané buď pro přenos užitečných datových informací, nebo k režijním účelům souvisejícím se sestavením, správou a řízením sítě:

- **Data Frame** rámec pro přenos užitečné informace pro všechny datové přenosy
- **Acknowledgement Frame** rámec pro přenos potvrzovací informace, využitelný pouze na úrovni MAC pro potvrzovanou komunikaci
- **MAC Command Frame** rámec k centralizovanému konfigurování, nastavení a řízení klientských zařízení v síti
- **Beacon Frame** rámec k synchronizaci zařízení v síti, je využíván hlavně při konfiguraci sítě v módu v němž umožňuje uvádění klientských zařízení do spánkových režimů s extrémně sníženou spotřebou.

Na základě časové synchronizace mezi centrální stanicí a koncovou stanicí dochází u uspané koncové stanice k probouzení ve vymezeném časovém intervalu, a poté jsou přeneseny veškeré potřebné informace. Interval synchronizačních sekvencí může být nastaven v rozmezí 15 ms až přibližně 15 minut.



Obrázek 5.2: Příklady topologié realizované standartem ZigBee

Standard IEEE 802.15.4 využívá pro adresaci jednotlivých zařízení binární adresovací kódy, které mohou být buď dlouhé (64 bitů), či zkrácené (16 bitů). Lokální adresa zkráceného adresovacího kódu umožňuje v jedné síti adresovat maximálně 65 535 zařízení. Každá sestavená síť je dále identifikována 16bitovým identifikátorem PAN ID (Personal Area Network ID), který slouží pro rozlišení překrývajících se sítí. Z hlediska topologie jsou definovány tři typy sítí (viz. obr. 5.2). Základní je topologie typu hvězda (star topology), kde řízením je pověřen PAN koordinátor (jedno zařízení) a ostatní pracují jako

koncová zařízení. Úpravou lze získat typ strom (tree topology). Třetí typ (Mesh topology) je kombinace obou předchozích.

5.1.1.2 Vyšší vrstvy - ZigBee Alliance

Struktura protokolů standardu ZigBee je navržena maximálně úsporně kvůli předpokládané implementaci do málo výkonných jednočipových 8bitových mikrokontrolérů s velmi omezenými paměťovými dispozicemi. Proto struktura protokolů nezabere více než asi 30 kB v systémové paměti, a je tedy mnohonásobně úspornější než standard Bluetooth, který vyžaduje více něž 100 kB operační paměti.

Nad vrstvami standardu IEEE 802.15.4 je definována síťová vrstva (NWK) a struktura pro aplikační vrstvu (APL). Síťová vrstva provádí připojování k síti a odpojování od ní, zabezpečení a směrování paketů. Jako základní zabezpečení mechanismus se používá 64bitový nebo 128bitový kryptografický algoritmus AES (Advanced Encryption Standard). Dále zajišťuje nalezení zařízení v rámci jednoho přeskoku. V případě koordinátora sítě je odpovědná za start sítě a přiřazování adres nově začleněným zařízením.

Aplikační vrstva protokolu ZigBee se skládá z pomocné aplikační podvrstvy (APS), objektů ZigBee (ZDO) a uživatelských aplikačních objektů. Aplikační pomocná podvrstva umožňuje párování zařízení podle poskytovaných služeb a požadavků. Objekt ZigBee definuje roli zařízení v rámci sítě (koordinátor, směrovač nebo koncové zařízení) a spravuje poskytované služby.

5.1.2 Spotřeba zařízení

Komunikační standard ZigBee je navržen pro aplikace, v nichž zařízení potřebují vysílat a přijímat pouze malé objemy dat a kde je vyžadována extrémně nízká spotřeba. Protokoly jsou proto navrženy s ohledem na co nejmenší spotřebu energie koncových zařízení, u kterých se předpokládá napájení z baterií. Koordinátor a směrovače by však neměly být napájeny bateriově, protože funkčnost sítě je na nich závislá.

Při nejjednodušší topologii hvězdě a využití technologie *beacon* se koncové zařízení aktivuje po přejmutí sekvence *beacon* a vyšle svá data. Koordinátor data přijme a uloží do paměti. Při přijetí další sekvence beacon indikuje koordinátor cílovému zařízení, že pro něj má data. Koordinátor data předá ve chvíli, kdy si je koncové zařízení vyžádá. Tento způsob zaručuje nejnižší spotřebu energie pro koncová zařízení, která jsou většinu doby přepnuta v úsporném režimu. Největší nároky jsou kladeny na koordinátora, který

musí být schopen uložit všechna data pro jednotlivá zařízení.

Text kapitola 5.1 byl převzat z [5].

5.2 Použitý hardware

Pro bezdrátovou komunikaci bylo použito modulů PAN802154HAR od firmy Panasonic. To jsou nízkopříkonové komunikační zařízení založené na Freescale ZigBee Application Refence Design (SARD) vývojové platformě. Pracují v pásmu ISM 2.4GHz a jsou plně kompatibilní se standartem IEEE 802.15.4. Moduly PAN802154 používají Freescale transievery MC13193, mikrokontroler GT60 a jsou licencovány k používání Freescale ZigBee Protocol stack. Jsou též vybaveny interface RS232 a dvěmi anténami.

Obrázek 5.3: Modul PAN802154

Základní vlastnosti modulů PAN802154:

- plná podpora ZigBee 802.15.4
- 2.4 GHz ISM, ZigBee
- přenosová rychlost až 250kbps
- RS232 port, 2x10-bitový A/D převodník, 8x I/O port
- výstupní výkon 1mW
- rozsah napájecího napětí 2.2-3.4V DC nebo 3.0-3.4V DC při použití RS232

- spotřeba typicky 35mA při přijímání / odesílání, v režimu sleep typicky 5

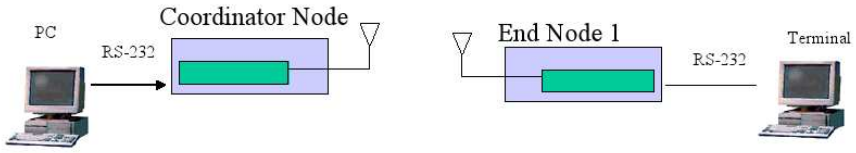
Technické parametry modulu PAN802154HAR převzaty z jeho katalogových listů [7].

5.3 Struktura řešení

5.3.1 Modemová komunikace

Základním stavebním prvkem systému jsou moduly PAN802154. Alespoň jeden musí být v režimu koordinátor - master a ostatní v režimu koncového zařízení. Strukturu řešení naznačuje obr. 5.4. Celý systém je pak postaven na aplikačních poznámkách Wireless Modem (Z-Stack 1.0.0) viz. [8].

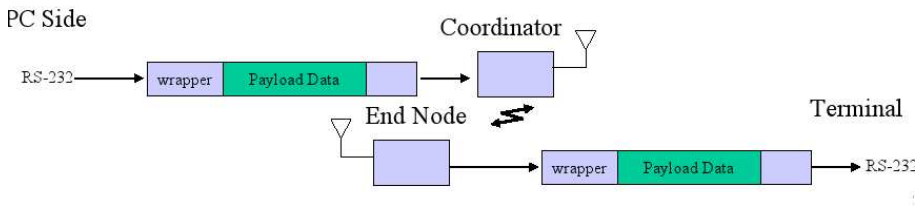
Řešení je vhodné pro řízení, nebo sběr dat ze zařízení na které jsou kladeny požadavky na bezdrátové spojení s nízkou spotřebu, velký dosahem a nízkým datovým tokem. Najde tedy uplatnění hlavně v oblasti sběru dat ze senzorů, v domácnostech jako systém bezdrátových ovladačů světel, alarmů, a v neposlední řadě jako komunikační prostředek pro časově nekritické aplikace jako jsou například řízení jednoduchých modelů především pak letadel a vznášedel.



Obrázek 5.4: Struktura modemové komunikace pomocí ZigBee

Většina mikroprocesorů je vybavena jednotkou sériového portu RS232, který svou přenosovou rychlostí pro nenáročné aplikace naprosto postačuje. Proto byl systém postaven právě na tomto rozhraní.

Příkazy a data jsou tedy posílána po standartní lince RS232 do bezdrátových modulů, kde jsou zpracovány, převedeny do ZigBee paketů a odeslány, jak zobrazuje obr. 5.5.



Obrázek 5.5: Tok dat v ZigBee modemem

Síť tvořená uzly (koncová zařízení) a koordinátorem tvoří hvězdicovou topologii. Koordinátor může komunikovat s jakoukoliv koncovou stanicí, ale koncové stanice mohou komunikovat pouze s koordinátorm.

5.3.1.1 Identifikace zařízení

Zařízení jsou v síti identifikována 64-bitovým MAC číslem jednoznačně určující každé zařízení. MAC adresa je číslo pevně dané v programové paměti každého zařízení. Výrobcem modulů jsou dodány výsledné produkty po kompilaci pro 4 zařízení. Přičemž v síti musí být vždy jedno zařízení koordinátor. MAC adresy zařízení ukazuje tabulka 5.1

Zařízení	**MAC adresa**
koordinátor	0x1716151413121110
koncová stanice 1	0x2726252423222120
koncová stanice 2	0x2726252423222121
koncová stanice 3	0x2726252423222122

Tabulka 5.1: MAC adresy zařízení

5.3.1.2 Formát paketu RS232

Data která je potřeba přenést přes ZigBee rozhraní je třeba začlenit do RS232 paketu akceptovaného ZigBee moduly. Formát takovéhoto paketu ukazuje obr. 5.6.

SOP	**CMD**	**Dest/Src MAC Addr**	**Len**	**Payload**	**EOP**
1 byte	1 byte	8 bytes (MSB first)	1 byte	'Len' bytes	1 byte

Obrázek 5.6: Formát paketu RS232

Kde význam jednotlivých parametrů popisuje tabulka 5.2.

Parametr	**Význam**
SOP	začátek paketu - vždy 0x02
CMD	nepoužito - vždy 0x07
Dest/Src MAC Addr	cílová/zdrojová MAC adresa
Len	délka přenášených dat
Payload	přenášená data
EOP	konec paketu - vždy 0xAA

Tabulka 5.2: Význam parametrů paketu

V paketu je především nutno zadat MAC adresu cílové stanice a délku přenášených dat. V případě příchozího paketu je formát stejný ale na místě cílové MAC adresy je adresa stanice, která zprávu vyslala.

Maximální délka přenášených dat obsažených v jednom paketu je 90B. Každý modul implementuje FIFO frontu pro celý jeden MAC paket o délce 128 B.

5.3.1.3 Parametry přenosu

ZigBee zařízení mají nastaveny přenosové rychlosti svých sériových portů na 38400kb/s. Komunikace probíhá vždy po 8-mi bitech s jedním start a jedním stop bitem. Parita ani řízení toku dat není použito. Pro bezdrátovou komunikaci je použit 11. kanál.

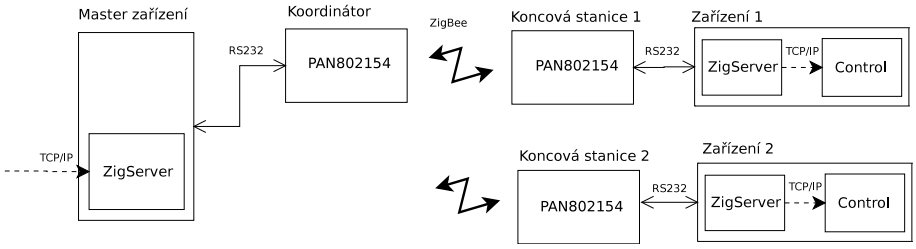
5.3.2 Paketová komunikace

Nevýhodou modemové komunikace je její dosah omezený vysílacím výkonem ZigBee modulů. Řešením tohoto problému, je zakomponovat do datového řetězce TCP pakety a pomocí nich nabídnout vzdálenou správu zařízení komunikujících pomocí ZigBee přes Internet.

Strukturu tohoto řešení ukazuje obr. 5.7. Master zařízení je pracovní stanice připojená k síti Internet. Na jednom ze svých sériových portů má připojen ZigBee modul v režimu koordinátor. Na pozadí na něm běží aplikace *ZigServer* překládající IP pakety na pakety formátu RS232 a opačně.

Zařízení jsou procesy, které je třeba řídit. Na sériovém portu mají připojen ZigBee

modul v režimu koncová stanice a běží na nich aplikace *ZigClient* spolu s řídící aplikací *Control* procesu.

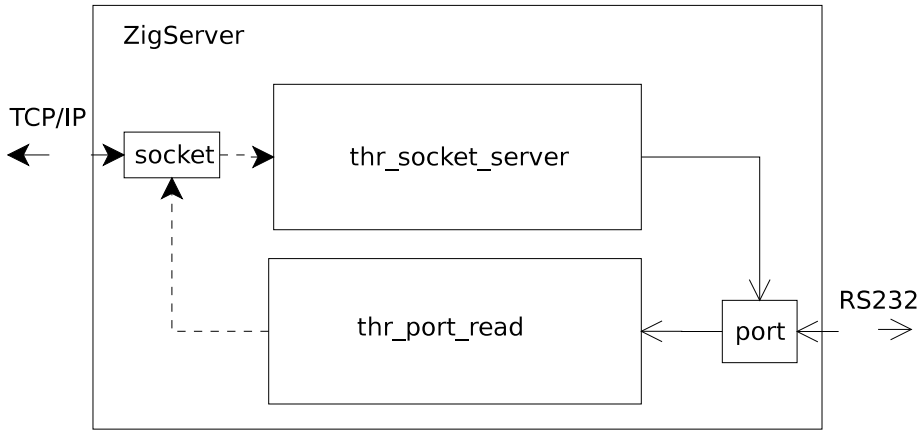


Obrázek 5.7: Struktura systému doplněná o paketovou komunikaci

5.3.2.1 Aplikace ZigServer

ZigServer je se multi-vláknová serverová aplikace, která transformuje příchozí TCP pakety ze sítě Internet do paketů RS232 pro ZigBee moduly. Je spuštěna na zařízení s připojeným ZigBee modulem v režimu koordinátor sítě. Jak ukazuje obr. 5.8 skládá se ze dvou hlavních vláken. Vlákno *thr_socket_server* po startu aplikace otevře TCP port pro komunikaci a čeká na připojení klienta ze sítě Internet. Po připojení klienta provádí transformaci přijatých zpráv do formátu paketu vhodného pro poslání do modulu ZigBee.

Druhé vlákno *thr_port_read* čte RS232 pakety přicházející ze sériového portu a převádí je do formátu TCP/IP paketu vhodného k poslání do sítě Internet.



Obrázek 5.8: Aplikace ZigServer

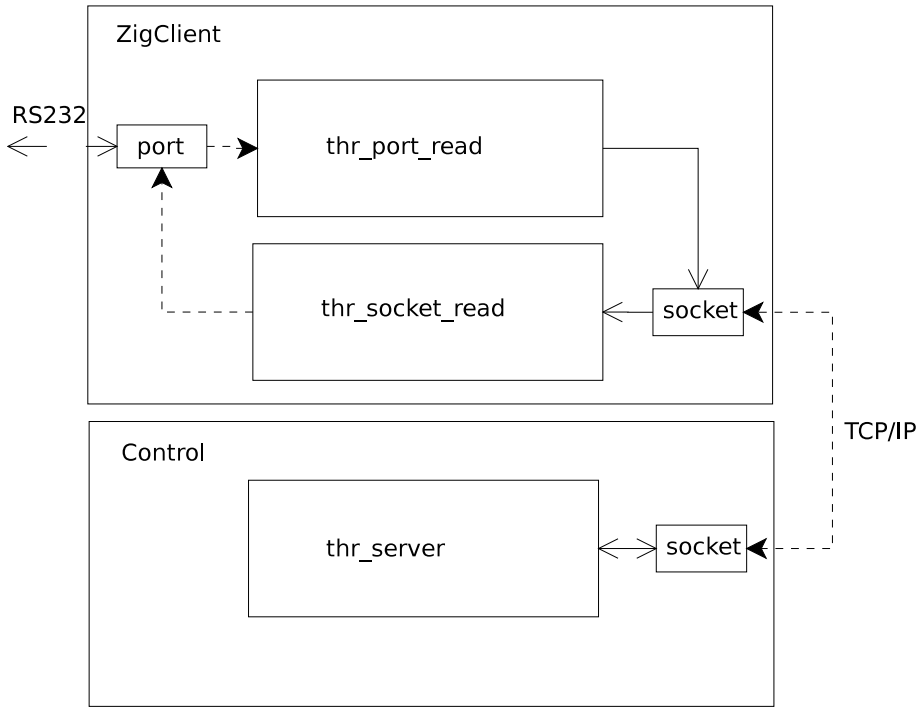
5.3.2.2 Aplikace ZigClient

ZigClient je multivláknová aplikace, která transformuje příchozí RS232 pakety ze sériového portu do podoby TCP/IP paketů, pomocí nichž se připojuje k řídící serverové aplikaci Control.

Po spuštění se aplikace neprve pokusí navázat komunikaci s řídící aplikací *Control* viz. kapitola 3.3. Namísto adresy serveru, ke kterému se má aplikace připojt, se použije symbolické jméno *localhost* určující zařízení na kterém je aplikace právě spuštěna. Číslo portu na kterém server aplikace *Control* poslouchá je nutné zjistit z její dokumentace, nicméně mělo by být voleno číslo v rozsahu 1023 - 65535. Porty nižší než dolní hranice jsou totiž v Unixových systémech vyhrazeny výhradně uživatelům *root*. Pokud se podařilo připojit k aplikaci *Control* je otevřen sériový port na kterém je připojen ZigBee modul a dojde ke spuštěn9 vláken obsluhující komunikaci.

Na obr. 5.9 je zobrazena struktura aplikace. Skládá se opět ze dvojice vláken. Vlákno thr_port_read čte RS232 pakety ze sériového portu a transformuje je na TCP pakety posílané pomocí soketu aplikaci *Control* kde se vykoná příkaz obsažený v paketu.

Vlákno thr_socket_read čte data přicházející od aplikace *Control* a převádí je na tvar RS232 paketu, který odesílá řídící stanici.



Obrázek 5.9: Aplikace ZigClient

5.3.2.3 Transformace paketů

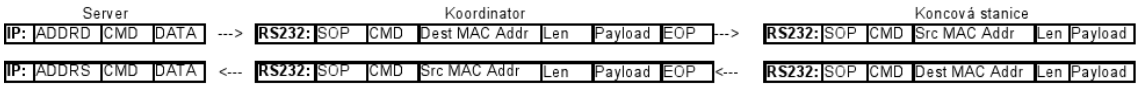
Rozsah počtu zařízení je v réálných bezdrátových sítích mnohem menší než dovoluje maximální množství dané normou dané bezdrátové technologie. Protože se nepředpokládá, že se síti používající výše uvedenou strukturu bude používat více než 255 zařízení, byl parametr určující adresu cílové a zdrojové stanice redukován z 8B na 1B. Navíc vzhledem k tomu, že se také předpokládá posílání krátkých zpráv jen o několika bytech tvořila by jen adresa převážnou část obsahu celého posílaného paketu.

Zařízení	**MAC adresa**	**Adresa v TCP paketech**
koordinátor	0x1716151413121110	0x00
koncová stanice 1	0x2726252423222120	0x01
koncová stanice 2	0x2726252423222121	0x02
koncová stanice 3	0x2726252423222122	0x03

Tabulka 5.3: Transformace MAC adres

Při transformaci formátu zpráv je nutný překlad MAC adres ZigBee zařízení na zkrácené adresy použité pro přenos v TCP paketech. Transformace může být implementována v podobě interní tabulky v paměti, tabulkou v externím souboru nebo být uložena v databázi. Příklad transformační tabulky ukazuje tabulka 5.3.

Formát TCP paketu spolu s jeho zařazením do komunikačního procesu je zobrazen na obr. 5.10. Význam jednotlivých položek pak vysvětluje tabulka 5.4.



Obrázek 5.10: Tok paketů a jejich transformace

Parametr	**Význam**	**Délka** [B]
ADDRD	adresa cílového zařízení	1
ADDRS	adresa zdrojového zařízení	1
CMD	nepoužito (možnost rozšíření počtu zařízení až na 65535)	1
DATA	přenášená data zakončená znakem nového řádku ('\n')	až 90

Tabulka 5.4: Význam parametrů TCP paketu

Kapitola 6

Závěr

6.1 Řešení práce

Tato práce se zabývá návrhem a realizací dálkového řízení mobilného zařízení s použitím bezdrátové technologie Wi-Fi. Mobilní zařízení v tomto případě představoval reálný model robota HERO, který je umístěn v laboratoři katedry řídící techniky Českého vysokého učení technického v Praze. Úkolem tedy bylo navrhnout a realizovat hardwarové a softwarové vybavení, které zajistí komunikaci robota s okolím. Realizace celé práce měla několik částí: V první fázi bylo třeba vybrat vhodnou řídící jednotku robota. Vzhledem k tomu, že v budoucím vývoji projektu je počítáno s prací s kamerami a infračervenými 3D scanery byly zavrženy jako nedostatečně výkonné všechny platformy postavené na mikropočítačích a nakonec byl vybrán průmyslový standart PC/104. Jako ovladače motorů pohyblivých os byly použity jednotky MAXON EPOS komunikující protokolem CANopen. Proto byla koupena i CAN karta do řídící jednotky PC/104. Konstrukcí robota se zabývala paralelní práce Jiřího Zemánka [1]. Na řídící jednotku byl nainstalován operační systém GNU/Linux distribuce Gentoo s jádrem 2.6.21-gentoo-r4. Pro podporu protokolu CANopen byl použit ovladač linCAN, který je součástí projektu OCERA. Použitá CAN karta však nebyla plně podporována a bylo tedy nutné napsat nový ovladač.

V druhé fázi byla napsány základní funkce pro ovládání pohybových os a obecně praci s řídícími jednotkami MAXON EPOS prostřednictvím protokolu CANopen. K jejich otestování byl napsána také demonstrační řídící aplikace *Control*, poskytující vzdálené řízení robota pomocí protokolu TCP. Dále byla v jazyce Java vytvořena i grafická aplikace pomocí níž je možno robota řídit vzdáleně s využitím protokolu TCP. Kladla si za cíl otestovat funkčnost bezdrátové komunikace a řídící aplikace *Control* v robotu. Dále by

měla sloužit jako demostrační aplikace pro první seznámení s robotem, a vzhledem k přenositelnosti Java programů ji je možno s výhodou spouštět např. i na PDA vybavenými Wi-Fi.

Během práce vznikl také koncept řízení robota pomocí bezdrátové technologie ZigBee jak popisuje kapitola 5. Zde byly jako základ použity ZigBee moduly PAN802154 s nahraným firmwarem Z-Modem. Moduly zde vystupují ve funkce modemu, kdy překládají data přicházející po sériové lince RS232 na ZigBee pakety. Z hlediska zařízení přípojených k modulům je pak celá ZigBee komunikace naprosto transparentní.

6.2 Testování

Proces testování byl zahájen nejprve testy komunikace CAN. Byly vzájemně propojeny konektory na CAN kartě křiženým kabelem a spuštěno několik instancí programů *readburst* a *sendburst*, dodávaným společně s ovladačem linCAN. Tyto programy posílají a čtou ze sběrnice CAN rámce. Během těchto testů docházelo k zatuhnutí operačního systému. Příčinou zatuhnutí bylo to, že CAN karta začala "bombardovat" jádro požadavky o přerušení (např. *cat /proc/interrupts*) a to z bezpečnostních důvodů zastavilo chod systému.

Po nutném tvdém restartu nebo i vypnutí systému nebylo možné ani vložit ovladač linCAN do jádra protože se karta CAN vůbec systému nehlásila. Pro opětovnou částečnou provozu schopnost bylo nutné řídící jednotku nechat vypnoutou alespoň 10 min.

Zdroj chyby může být buď problém na desce CAN, na desce CPU (problém s časováním PCI sběrnice) nebo problém ovladače linCAN.

Pro zjištění příčiny by bylo možné se pokusit sestavu chladit výkoným ventilátorem a poté zkusit ohřát např. féném. Pokud by byla prokázána korelace s problémy jde pravděpodobně o chybu v hardware. Také by bylo možno otestovat sestavu s jinou CAN PCI kartou a pokusit se tak vyvrátit možnost chyb způsobené vadnými ovladači nebo vzájemnou nekompatibilitou hardwaru.

Jiným možným postupem je pokusit se kontaktovat výrobce karty a požádat ho o zapůjčení totožné karty k otestování.

Dálším bodem testování bylo ověření funkčnosti bezdrátové komunikace a grafické aplikace zasíláním příkazů a požadavků na běžící aplikaci *Control* na řídící jednotce ro-

bota. Všechny přijímané zprávy byly korektně zpracovány, odpovědi na dotazy klientů též probíhaly podle komunikačního zadaného protokolu.

Do okamžiku psaní tohoto dokumnetu se bohužel nepodařilo vzhledem k problémům s CAN kartou otestovat komunikaci řídícího programu *Control* s řídícími jednotkami motorů a tak vlastně nebylo splněno zadání.

6.3 Možnosti rozšíření

Na místě je naznačit možnost dalšího pokračování projektu. V prvé řadě je potřeba odstranit výše uvedený problém s CAN kartou jedním z uvedených možných řešení a následně provést testování a možné úpravy řídící aplikace *Control*.

Pro další vývoj by bylo dobré robota rozšířit o senzorový systém. Vhodné by byly např. infračervené senzory polohy GP2D12 nebo jejich digitální varianty, rozmístěné vždy na bocích robotu. Konstrukce robota též dovoluje použití infračervených 3D scanerů, díky nimž by bylo možno vytvářet dynamicky mapu prostředí.

Další možné rozšíření je zapojení již zakoupené webové kamery pro snímání scény a vytvoření podpory pro odesílání obrazu přes TCP/IP připojeným klientům.

Výhodou by byla přímo na robotu přítomnost webového serveru, který by umožňoval řízení přes webové rozhraní např. pomocí Java apletu. Doporučeným webovým serverem je Apache 2. Pro snadnější nahrávání řídících aplikací přes webové rozhraní by bylo potřeba vybudovat jednoduchý ukládací systém který by dovoloval, kromě nahrání testovaného řídícího algoritmu i jeho spuštění a zastavení. Takový systém lze napsat např. pomocí jazyka PHP, .NET, nebo opět využitím Java apletu.

Literatura

[1] J. Zemánek: *Bakalářská práce: Robot HERO.*
Katedra řídící techniky, České vysoké učení technické v Praze , 2007

[2] F. Vaněk: *Diplomová práce: Robot HERO.*
Katedra řídící techniky, České vysoké učení technické v Praze , 1985

[3] P. Savický: *Bakalářská práce: Mobilní robot pro výuku real-time řízení.*
Katedra řídící techniky, České vysoké učení technické v Praze , 2005

[4] M. Shalloway: *Design Patterns: From Analysis to Implementation.*
Net Objectives, 2006

[5] A. Vojáček: *ZigBee - novinka na poli bezdrátové komunikace [online].*
Poslední aktualizace 2005-04-04 [cit. 2007-07-04],
http://hw.cz/Rozhrani/ART1299-ZigBee—novinka-na-poli-bezdratove-komunikace.html

[6] Z. Bradáč: *Bezdrátový komunikační standart ZigBee [online].*
Poslední aktualizace 2005-08-10 [cit. 2007-07-04],
http://www.automatizace.cz/download.php?d=QXRtX0FydGljbGUscGRmX2FydCw2Mzg=

[7] Panasonic: *PAN802154 Product Specificaton [online].*
Poslední aktualizace 2006-08-03 [cit. 2007-07-04],
http://www.panasonic.com/industrial/components/pdf/zigbee_spec.pdf

[8] Panasonic: *Wireless Modem Demo (Z-stack) [online].*
Poslední aktualizace 2006-07-04 [cit. 2007-07-04],
http://www.panasonic.com/industrial/components/zigbee_agreement.asp

[9] Gentoo Linux: *Gentoo Linux x86 Handbook [online].*
Poslední aktualizace 2006-11-03 [cit. 2006-11-09],
http://www.gentoo.org/doc/cs/handbook/handbook-x86.xml?full=1

[10] Gentoo Linux: *Init Skripty [online].*
Poslední aktualizace 2006-09-12 [cit. 2007-06-16],
http://www.gentoo.org/doc/cs/handbook/handbook-x86.xml?part=2chap=4

[11] Gentoo Linux Wiki: *HOWTO Share Directories via NFS [online].*
Poslední aktualizace 2007-06-19 [cit. 2007-06-20],
http://gentoo-wiki.com/HOWTO_Share_Directories_via_NFS

[12] Wiki Books: *Start Linuxu [online].*
Poslední aktualizace 2007-03-10 [cit. 2007-08-07],
http://cs.wikibooks.org/wiki/Start_Linuxu

[13] D. Kačmář: *Jazyk C*
Computer Press, Praha, 2001
ISBN: 80-7226-295-5

[14] CiA-CANopen protocol *CiA CANopen protocol [online]*
Poslední aktualizace 2007-05-16 [cit. 2007-08-13],
http://www.can-cia.org/canopen/

[15] OCERA *OMK OCERA Make system [online]*
Poslední aktualizace 2007-06-10 [cit. 2007-08-13],
http://rtime.felk.cvut.cz/hw/index.php/OMK

[16] OCERA *OCERA - Open Components for Embedded Real-Time Applications [online]*
Poslední aktualizace 2006-02-21 [cit. 2007-08-04],
http://www.ocera.org/

[17] MAXON: *MAXON EPOS Position Controller - Firmware Specification*
MAXON Motor Control , 2006

Příloha A

Obsah přiloženého CD

Na přiloženém CD je následující struktura adresářů:

/appnote/	obsahuje aplikační poznámky k řídícím jednotkám MAXON a modulu ZigBee
/bp/	obsahuje tuto práci ve fromátu pdf
/config/	obsahuje konfigurační soubory operačního systému GNU/Linux
/control/	obsahuje zdrojové kódy a dokumentaci k řídící aplikaci *control*
/HeroGUI/	obsahuje zdrojové kódy a projekt pro NetBeans 5.5 aplikace pro ruční řízení
/manuals/	obsahuje použité manuály a datasheety