České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Mobilní robot pro výuku real-time řízení

Vypracoval: Petr Savický
Vedoucí práce: Ing. Michal Sojka

Praha 2005

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

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 27.5.2005 Petr Savický

Děkuji tímto Ing. Michalu Sojkovi za odborné vedení a pomoc při zpracování bakalářské práce.

Také bych chtěl poděkovat svým rodičům za podporu, které se mi od nich dostávalo po celou dobu studia.

České vysoké učení technické v Praze, fakulta elektrotechnická

Katedra řídicí techniky | Školní rok:2004/2005

Zadání bakalářské práce

Student: Petr Savický

Obor: Kybernetika a měření

Název tématu: Mobilní robot pro výuku real-time řízení

Zásady pro vypracování:

1. Seznamte se s aktuálním stavem projektu ReToBot (mobilní robot vyvíjený na katedře řídicí techniky, http://dce.felk.cvut.cz/retobot/).
2. Připravte "hřiště" pro pohyb robota vybavené místem pro automatické nabíjení.
3. Odstraňte nedostatky na mechanice robota (zapojení vypínače, vedení kabeláže atd.)
4. Upravte stávající software a dokumentaci tak, aby byly odstraněny současné nedostatky. Výsledky publikujte na webových stránkách projektu.
5. Zkuste připravit úlohu pro výuku nebo demonstrační aplikaci.

Seznam odborné literatury: Dodá vedoucí práce

Vedoucí bakalářské práce: Ing. Michal Sojka

Datum zadání bakalářské práce: říjen 2004

Termín odevzdání bakalářské práce: 27. 5. 2005

Prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

Prof. Ing. Vladimír Kučera, DrSc.
děkan

V Praze, dne 22. 11. 2004

Abstrakt

Tato práce je koncipována jako manuál k mobilnímu robotu pro výuku real-time řízení vyvíjeného na katedře řídicí techniky. Je zde popsán stávající hardware i software robotu. V rámci práce byly odstraněny zjištěné nedostatky, vybudováno hřiště pro pohyb robotu a napsána uživatelská grafická aplikace pro ruční řízení robotu.

Abstract

This thesis is conceived as a reference manual to mobile robot which allows the students to learn handling real-time operating systems in connection with control engineering. There is described actual hardware as well as software of the robot in the thesis. Within the work there were rectified detected failings, built a playground for the robot and written user graphic remote application for manual control of the robot.

Obsah

1 Úvod

1.1 Účel projektu

Katedrou řídicí techniky byl zadán projekt, jehož cílem je vytvořit model, mobilní robot, pro výuku real-time řízení.

Robot by měl být vybaven řídicím počítačem s operačním systémem Linux, využívat sběrnici CAN a dále být vybaven časově náročnými senzory, aby se využila plná kapacita real-time systému.

Výsledný model by měl umožnit studentům učit se ovládat real-time operační systémy v souvislosti s řízením.

Robot by měl být bezdrátově připojen k univerzitní síti WLAN. Studenti tak budou moci naprogramovat robot a odeslat vytvořený kód přes webové rozhraní. Výsledky své práce budou moci sledovat přes webovou kameru.

1.2 Historie projektu

Na tomto projektu začal pracovat v březnu 2004 zahraniční student Reto Aebersold, který navrhnul robot jak z hlediska mechanického, tak elektrického. Sestavil robot a napsal k němu základní software. Vytvořil také webové stránky projektu (http://dce.felk.cvut.cz/retobot). Svou práci na projektu ukončil v září 2004.

V říjnu 2004 pak navázala tato bakalářská práce.

1.3 Úkol bakalářské práce

Výsledky práce na předchozí části projektu obsahovaly některé nedostatky. Úkolem bakalářské práce bylo odstranit tyto nedostatky, které se týkaly zejména mechaniky robotu, řídicího software a dokumentace na webových stránkách projektu (provedené úpravy a odstraněné nedostatky jsou podrobně uvedeny na konci příslušných kapitol).

Dalším úkolem bylo připravit hřiště pro pohyb robotu vybavené místem pro automatické nabíjení.

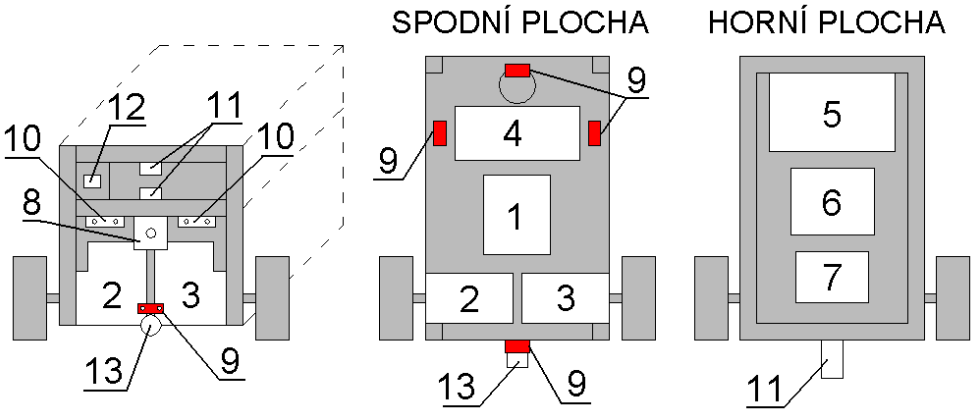
Hlavním úkolem bylo vytvořit uživatelskou grafickou aplikaci pro ruční řízení robotu, která by byla spustitelná jak v systému Linux, tak ve Windows.

2 Hardware robotu

2.1 Přehled

Robot Retobot byl sestaven Retem Aebersoldem. Dispoziční řešení robotu je naznačeno na obrázku 2-2, elektrotechnické řešení na obrázku 2-3. Půdorysné rozměry robotu jsou 22 x 34 cm, výška 23 cm, hmotnost 12 kg.

Obrázek 2-1: Retobot



Obrázek 2-2: Dispoziční řešení robotu

Legenda:

1.....řídicí počítač

2.....pravý motor

3.....levý motor

4.....spodní akumulátor

5.....horní akumulátor

6.....PCB

7.....Wi-Fi

8 .....kamera

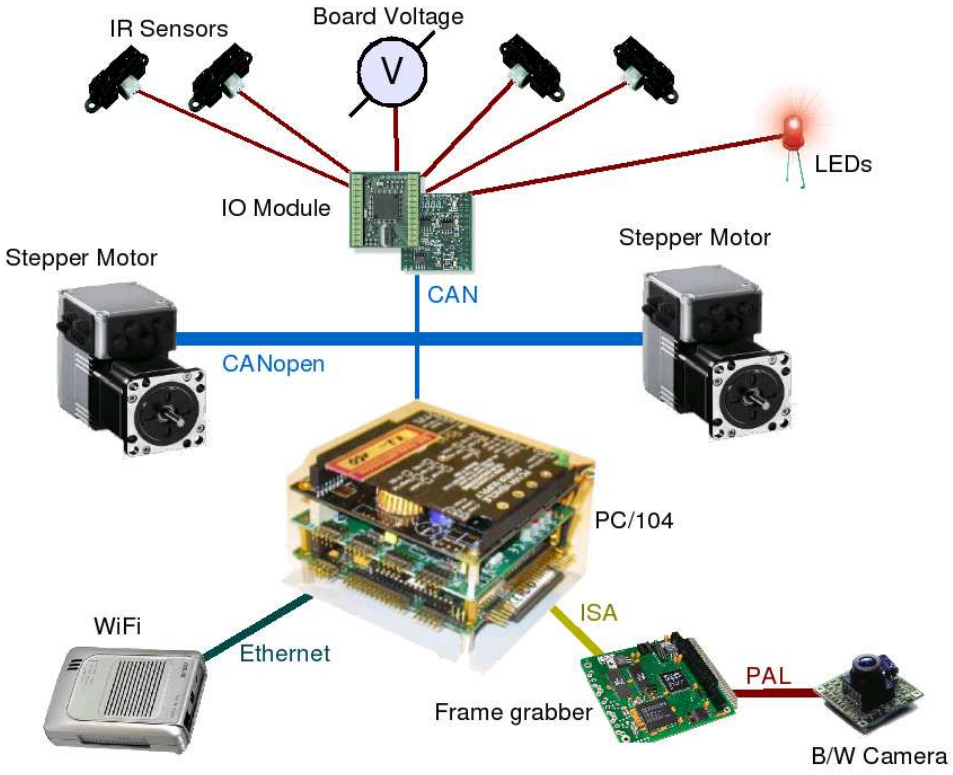
9 .....IR senzory

10 .....LED diody

11 .....nabíjecí kontakty

12 .....vypínač

13 .....nárazník



Obrázek 2-3: Schéma propojení jednotlivých částí robotu

2.2 Moduly standardu PC/104

Obrázek 2-4: Řídicí počítač PC/104

Robot je vybaven řídicím počítačem, který je sestaven ze 4 modulů standardu PC/104:

- hlavní procesorová deska,
- CAN modul,
- framegrabber,
- napájecí modul.

2.2.1 Hlavní procesorová deska

Použitý modul MSM586 obsahuje součásti kompatibilní se standardem PC-AT:

- CPU ELAN520 133 MHz, BIOS ROM, paměť SODIMM, časovače, DMA, hodiny reálného času s CMOS-RAM a další.

Obrázek 2-5: Modul MSM586

Modul obsahuje následující rozhraní:

- paralelní port LPT1
- sériové porty COM1 – COM4
- pro zvukový výstup
- pro klávesnici AT nebo PS/2
- pro myš PS/2
- pro disketovou mechaniku
- AT-IDE (pro pevný disk)
- VGA/LCD
- PC/104 embedded BUS
- pro kartu Compact Flash typ 1
- LAN Ethernet

Rozložení a popis jednotlivých rozhraní na modulu je uvedeno v příloze 1.

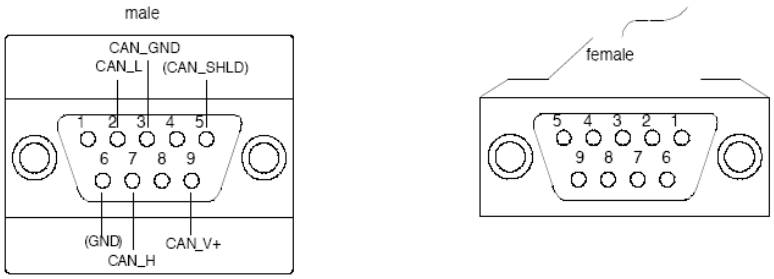
Modul dále obsahuje programovatelný watchdog (1 – 32 s) a sběrnice PCI (vnitřně používaná LAN a VGA) a ISA (společná se sběrnicí PC/104).

2.2.2 CAN modul

CAN modul obsahuje kartu DIMM pro CAN-bus (DIMM/CAN-ETB). Slouží jako CAN rozhraní, ke kterému je připojena sběrnice CAN. K této sběrnici jsou připojeny oba motory robotu a IO-modul.

Obrázek 2-6: CAN modul

CAN modul je opatřen dvěma konektory D-Sub s rozložením vývodů dle obrázku 2-7.



Obrázek 2-7: Rozložení vývodů konektoru D-Sub pro CAN

Pin	Signál	Popis
1	-	Reserved
2	CAN_L	CAN_L bus line (dominant low)
3	CAN_GND	CAN Ground
4	-	Reserved
5	(CAN_SHLD)	Optional CAN Shield
6	(GND)	Optional Ground
7	CAN_H	CAN_H bus line (dominant high)
8	-	Reserved
9	(CAN_V+)	Optional CAN external positive supply (dedicated for supply of transceiver and opto-couplers, if galvanic isolation of the bus node applies)

Tabulka 1: Popis vývodů na konektoru D-Sub

2.2.3 Framegrabber

Řídicí počítač je vybaven modulem PixelSmart512-8 pro ukládání č/b obrazu v reálném čase s rozlišením 768x512 a ve 256 stupních šedi.

Výstupní video signál z č/b kamery vstupuje do framegrabberu přes RCA jack (cinch). Tento video signál je typu composite a je v systému PAL (25 snímků/s, snímky jsou prokládané).

Obrazový buffer má velikost 256kB. Jas a kontrast lze softwarově nastavovat v reálném čase.

Obrázek 2-8: Framegrabber PixelSmart512-8

A/D převodník

Framegrabber je vybaven A/D převodníkem Philips SAA7110. Procesor framegrabberu komunikuje s tímto převodníkem pomocí sběrnice I2C. Ke komunikaci používá registr na adrese 0x2001 (viz mapa paměti v příloze 2).

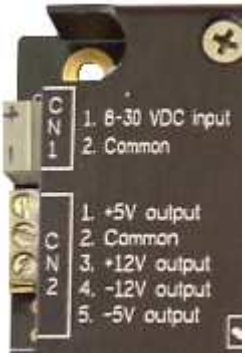
V tabulce 2 jsou uvedeny adresy registrů A/D převodníku pro nastavení jasu a kontrastu obrazu. Názvy uvedené ve sloupci hrt.h jsou jména konstant definované v hlavičkovém souboru hrt.h (viz kap. *2.3.1 Modul hrt.o*).

hrt.h	hodnota	popis
HRT_AD_DEVICE_ID	(128+16+8+4)	The unique I2C bus address of the SAA7110 (A/D) device
HRT_BRIGHTNESS_REG	0x19	
HRT_CONTRAST_REG	0x13	

Tabulka 2: Registry A/D převodníku u framegrabberu

2.2.4 Napájecí modul

Obrázek 2-9: Napájecí modul V104



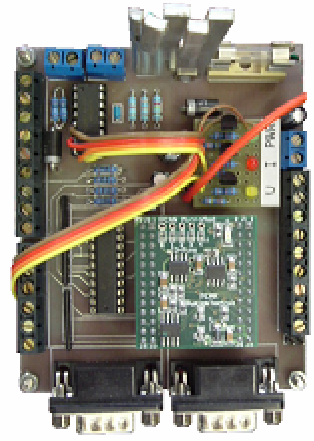
Obrázek 2-10: Detail svorkovnice napájecího modulu

Řídicí počítač je napájen modulem V104. Modul zajišťuje také napájení IO-modulu (5V), IR senzorů (5V) a kamery (12V). Vstupní napětí modulu zajišťují akumulátory (24V) nebo nabíjecí zdroj (30V).

Specifikace napájecího modulu:

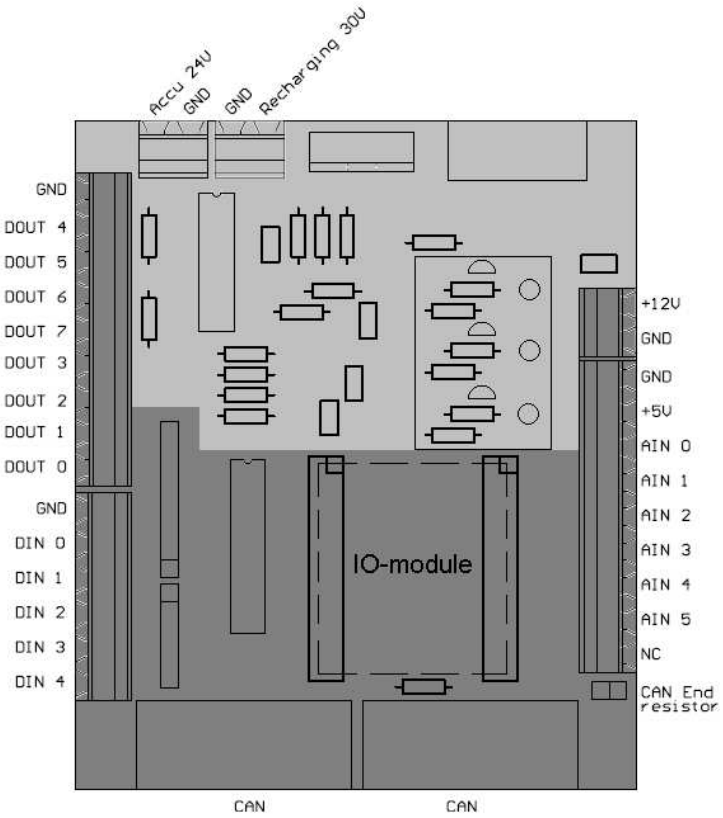
- vstupní napětí: 8 až 30 V
- výstupní napětí / proud: 5V/5A, 12V/1A
- max. výkon: 25W

2.3 Navržená deska plošných spojů



Obrázek 2-11: PCB

Na vrchní ploše robotu je osazena deska plošných spojů (PCB) obsahující 2 nezávislé obvody - obvod s IO-modulem a nabíjecí obvod. Na obrázku 2-12 je zachyceno umístění obou obvodů.



Obrázek 2-12: Dva obvody na PCB – obvod s IO-modulem a nabíjecí obvod

Popis a rozmístění součástek na PCB je uvedeno v příloze 3.

2.3.1 Obvod s IO-modulem

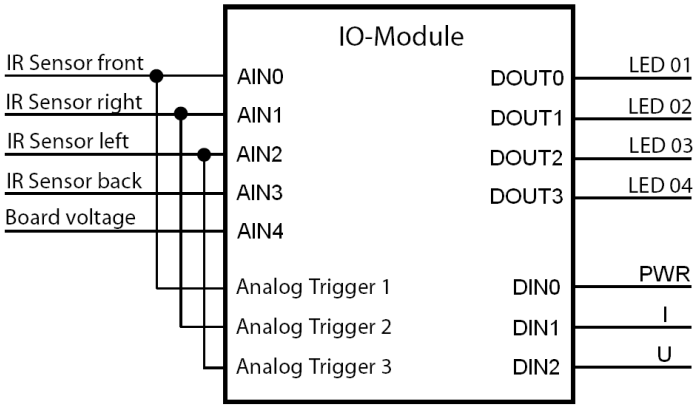
Schéma obvodu s IO-modulem je zobrazeno v příloze 4.

Navržená deska plošných spojů je osazena IO-modulem PCAN MicroMod. Tento modul je připojen ke CAN-modulu PC/104 pomocí sběrnice CAN. Umožňuje snímat digitální a analogové vstupy a zajišťuje jejich přenos po sběrnici CAN. A naopak příchozí zprávy převádí na digitální a frekvenční výstupy.

Specifikace IO-modulu:

- 8 digitálních vstupů TTL
- 8 digitálních výstupů TTL
- 8 analogových vstupů (10 bitů, vstup 0 až Vref, referenční napětí 2,7 až 5 V)
- 4 PWM nebo 2 frekvenční výstupy v rozsahu 1 Hz až 10 kHz
- CAN port

Na IO-modulu jsou využity 4 digitální výstupy k ovládání LED diod, 3 digitální vstupy pro zjišťování nabíjecích stavů a 5 analogových vstupů pro snímání hodnot z IR senzorů a napětí akumulátorů. Využité vstupy a výstupy jsou schematicky znázorněny na obrázku 2-13.



Obrázek 2-13: Využité vstupy a výstupy na IO-modulu

Funkce analogového triggeru je možná pouze na vstupech AIN0, AIN1 a AIN2. Hodnota triggerů je nastavena na 200 mm (více v kap. *3.3.4 IO-Modul*).

Vstupy a výstupy IO-modulu jsou vyvedeny na svorkovnice na PCB. Svorkovnice jsou popsány na obrázku 2-12.

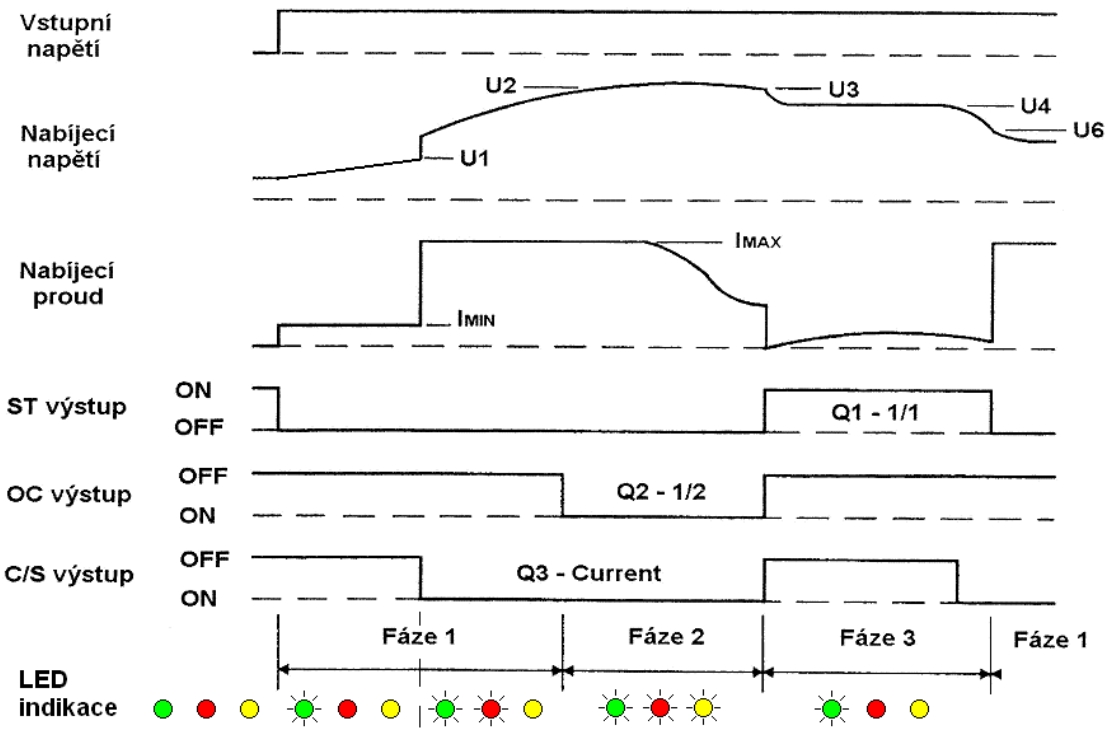
2.3.2 Obvod pro nabíjení olověných akumulátorů

Schéma nabíjecího obvodu je zobrazeno v příloze 5.

Robot je napájen dvěma 12V akumulátory v sériovém zapojení pro zajištění napětí 24V. Toto napětí přímo odebírají motory a napájecí modul PC/104. Potřebných 5V (pro IO-modul a IR senzory) a 12V (pro kameru) zajišťuje napájecí modul PC/104.

Hlavní prvek nabíjecího obvodu tvoří nabíječ olověných akumulátorů UC3906 v pouzdře DIP16. Nabíjecí obvod umožňuje nabíjet baterie podle průběhů uvedených na obrázku 2-14.

V příloze 6 jsou uvedeny výpočty hodnot rezistorů použitých v nabíjecím obvodu.



Obrázek 2-14: Průběhy nabíjení

Indikace nabíjecích stavů:

- zelená LED – indikuje přítomnost vstupního napětí 30V na nabíjecích kontaktech robotu
- červená LED – indikuje výstup C/S, tj. nabíjení poudem I > Imin
- žlutá LED – indikuje výstup OC (fáze 2) , tj. přebíjení, kdy nabíjecí napětí U > U2 a nabíjecí proud klesá

2.4 Krokové motory

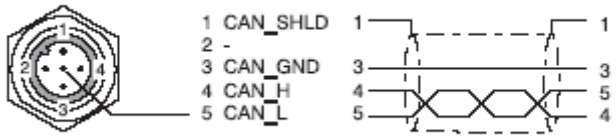
Robot je řízen dvěma krokovými motory IFS61. Motor společně s řídicí a napájecí elektronikou tvoří kompaktní jednotku, která může být konfigurována a řízena přes rozhraní sběrnice CAN pomocí protokolu CANopen. Jednotka může být zapojena jako slave do sítě CANopen v souladu s DS301.

Obrázek 2-15: Krokový motor IFS61

Adresu a přenosovou rychlost sběrnice lze nastavit hardwarově pomocí přepínačů DIP uvnitř jednotky nebo softwarově pomocí parametrů `CAN.canAddr` a `CAN.canBaud` (viz tabulka 9).

Jednotka je napájena napětím 24V (napětí 2 akumulátorů v sérii).

Štítkové parametry motoru: Un = 36VDC, Mn = 0,45Nm, Imax = 3,5A.



Obrázek 2-16: Rozložení vývodů konektoru krokového motoru

Pin	Signál	Význam	Typ signálu
1	CAN_SHLD	Shield connection	
3	CAN_GND	internally positioned at power supply GND	
4	CANH	CAN interface	I/O
5	CANL	CAN interface	I/O

Tabulka 3: Popis vývodů na konektoru krokového motoru

Pro tuto jednotku existují 3 operační módy – rychlostní, point-to-point (PTP) a referenční. Operační módy reprezentují různé možnosti pro polohování. Pro řízení robotu je použitelný mód rychostní a PTP. Parametry těchto módů mohou být nastaveny dle tabulky 8.

Rychlostní mód představuje polohování změnou rychlosti. Motor se zastaví na pokyn uživatele.

Mód PTP představuje polohování s měřením natočení hřídele (v jednotkách *inc*) absolutně nebo relativně. Motor se zastaví na pokyn uživatele nebo po dosažení zadané polohy. Rozlišení inkrementálního snímače je 20000 inc/ot.

Následující možnosti nastavení jsou použitelné pro všechny operační módy:

- zrychlení a zpomalení motoru s možností nastavení akcelerační rampy
- zpomalení motoru "Quick-Stop“ funkcí

Více o nastavení parametrů viz kapitola *3.3.5 Krokové motory*.

2.5 IR senzory

OBRÁZEK

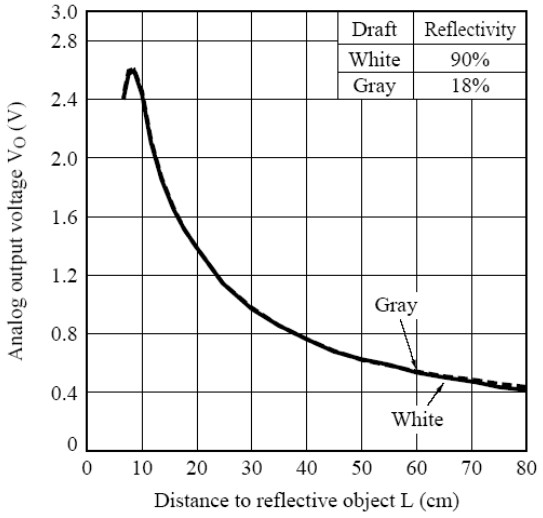
Obrázek 2-17: IR senzor GP2D12

Specifikace senzoru:

- napájecí napětí (Vcc): -0,3 až 7V
- vzdálenostní výstup ve formě analogového napětí: -0,3 až Vcc+0,3V
- detekující vzdálenost: 10 až 80 cm

- doba 1 operace měření vzdálenosti: 29 až 47 ms
- do 5 ms po skončení operace se výsledek promítne do výstupního napětí
- nízká citlivost na barvu odrazivé plochy předmětů, nízká citlivost na reflektivitu

Na obrázku 2-18 je zobrazen graf závislosti výstupního napětí IR senzoru na měřené vzdálenosti. Graf je převzatý z katalogu výrobce.



Obrázek 2-18: Graf závislosti výstupního napětí IR senzoru na měřené vzdálenosti

2.6 Kamera

Robot je vybaven kitem č/b kamery s CCD čipem Sharp. Kamera je přímo připojena k nahrávací kartě přes video signál typu composite v systému PAL. Kamera je napájena 12 VDC.

Obrázek 2-19: Kamera MTV-251CM

Popis vodičů na konektoru kamery:

- červený – napájecí napětí +12VDC (POWER INPUT)
- černý – signálová a napájecí zem (GND)
- bílý – výstupní video signál (VIDEO OUTPUT)
- žlutý – výstupní signál pro ovládání čočky kamery (AUTO IRIS SIGNAL)

2.7 Wi-Fi

WL-330b je bezdrátový přístupový bod standardu Wi-Fi 802.11b s funkcemi bridge. Je vybaven ethernetovým rozhraním, má dvě integrované IFA antény a pro vyšší bezpečnost dat umožňuje použití WEP šifrování. Nevyžaduje instalaci ovladačů, má automatické vyhledávání kompatibilních bezdrátových zařízení a umožňuje webovou správu a konfiguraci.

Obrázek 2-20: Wi-Fi Access Point WL-330b

Specifikace WL-330b:

- rozhraní: Ethernet RJ-45
- přenosová rychlost: až 11 Mbit/s (802.11b)
- RF výstupní výkon: 15 - 18 dBm
- přijímací citlivost: -82 až -85 dBm
- operační dosah v otevřeném terénu: až 457 metrů (při přímé viditelnosti)
- frekvence: 2.412 - 2.472 GHz

2.8 Olověné akumulátory

Robot je napájen dvěma 12V akumulátory v sériovém zapojení pro zajištění napětí 24V potřebné pro krokové motory.

Obrázek 2-21: Akumulátor WP8-12

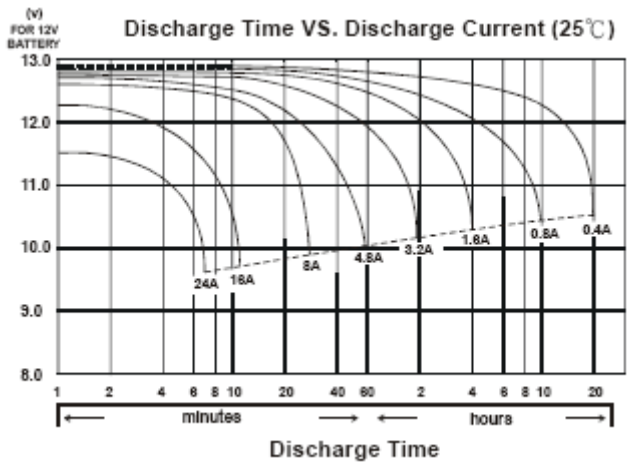
Specifikace akumulátoru WP8-12:

- nominální napětí: 12 V
- nominální kapacita: 8 Ah

Příkon jednotlivých součástí robotu je zaznamenán v tabulce 4.

komponenta	**energie**
kamera	0,8W/12V
Wi-Fi	1,5W/5V
stojící motor se zapnutým zesilovačem	13W/24V
běžící motor	40W/24V
PC104 + IO-modul	14W/24V
nabíjení akumulátorů	0,2 – 28W/28V

tabulka 4: Spotřeba enegie



Obrázek 2-22: Graf vybíjení akumulátoru WP8-12

2.9 Dodatek k hardware

V rámci bakalářské práce byly odstraněny zjištěné nedostatky jak na mechanice, tak elektronice robotu. Níže jsou popsány provedené úpravy.

Vypínač řídicího počítače (PC/104) byl uchycen v přední části robotu. Bylo změněno zapojení vypínače v obvodu, aby se mohl robot nabíjet i při vypnuté PC/104.

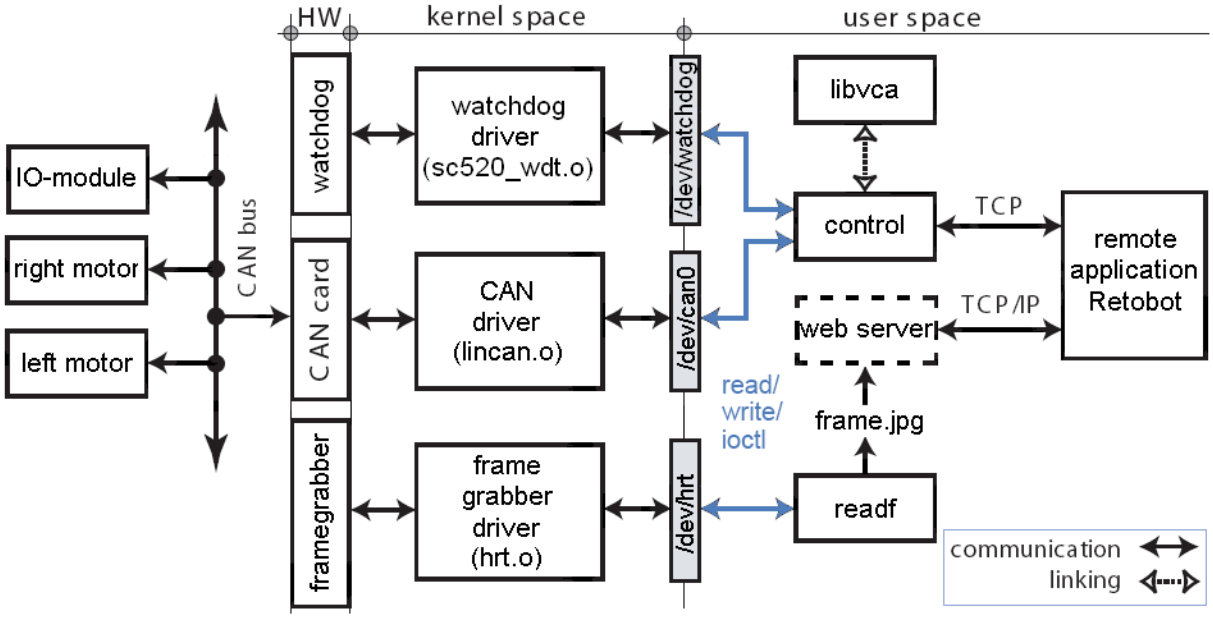
Vedení kabeláže bylo upraveno pro snazší manipulaci a přístup k PC/104. IR senzory robotu byly přemístěny do nižší polohy pro lepší detekci překážek. Robot byl doplněn o zadní IR senzor.

Původní nabíjecí kontakty robotu byly uhlíkové s vlastním odporem 3Ω. Po nedokonalém spojení s nabíjecím místem vznikal velký přechodový odpor (cca 25Ω), a kvůli velkému úbytku napětí se baterie nenabíjely na plnou kapacitu a s menším nabíjecím proudem. Nabíjení tedy trvalo déle. Navíc pružiny, které byly součástí nabíjecích kontaktů, odtlačely robot od nabíjecího místa a docházelo i k úplnému odpojení. Nové nabíjecí elektrody byly vyrobeny z měďeného plechu s odporem menším než 0,1Ω. Konstrukce kontaktů byla vyřešena tak, aby nedocházelo k odtlačení.

Nabíjecí obvod byl doplněn o diodu, která znemožňovala přenos napětí baterií do jiných částí nabíjecího obvodu. Dále byl doplněn o indikaci nabíjecích stavů (viz kapitola *2.3.2 Obvod pro nabíjení olověných akumulátorů*).

3 Software robotu

3.1 Přehled

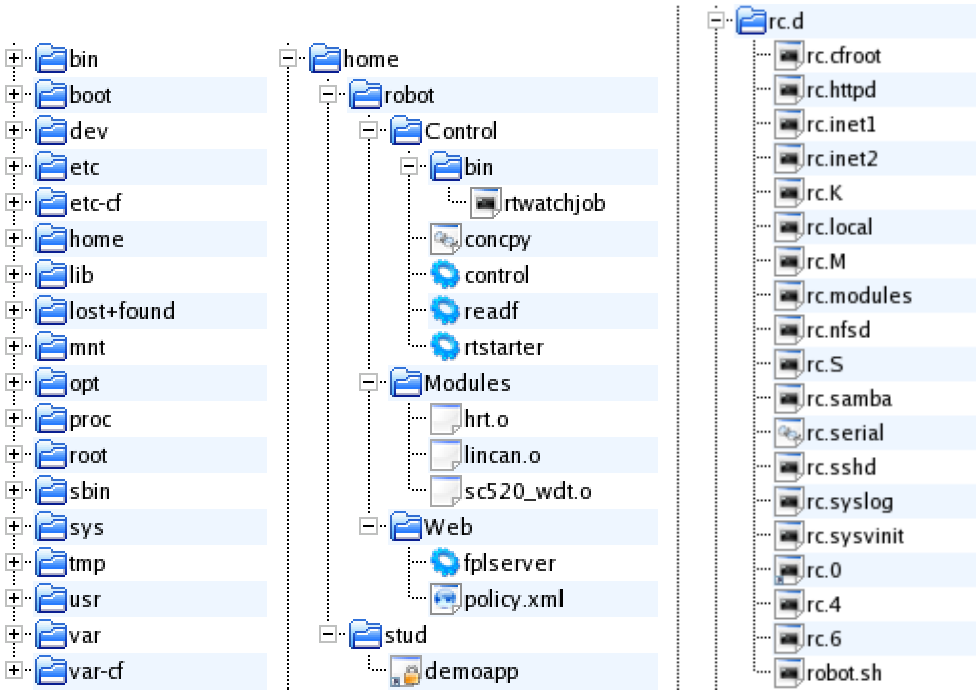


Obrázek 3-1: Software robotu

3.2 Operační systém Linux

Na řídicím počítači robotu (PC/104) je nainstalován operační systém Linux s jádrem 2.6.5.

3.2.1 Struktura adresářů



Obrázek 3-2: Struktura důležitých adresářů na PC/104

Webové rozhraní na robotu je umístěno v adresáři `/var/www/htdocs/` a umožňuje upload vlastní aplikace na server robotu (viz kapitola *3.7.2 Upload řídicí aplikace na server robotu*). Po úspěšném uploadu je aplikace umístěna do `/var/tmp/sapp` se symbolickým odkazem `/home/stud/app`.

Adresář `/home/stud/` je přístupný uživatelům přihlášeným přes webové rozhraní. Obsahuje symbolický odkaz `demoapp` na demonstrační řídicí aplikaci `/home/robot/Control/control`.

3.2.2 Inicializační skripty

Inicializační soubory se na PC/104 nacházejí v adresáři `/etc/rc.d/`

Zavedení operačního systému

Zavedení OS představuje načtení jádra OS do paměti a zahájení jeho činnosti (jádro je na PC/104 umístěno v souboru `/boot/vmlinuz_2_6.5`). Zavádění OS provádí lilo (linux loader), který na závěr spustí program `/sbin/init` jako proces č.1. Činnost programu `init` je řízena obsahem souboru `/etc/inittab`. Tento soubor spouští inicializační skripty podle systémové úrovně (runlevelu), ve které se OS právě nachází. Jako defaultní je nastavený runlevel 3, tj. víceuživatelský mód.

Skripty rc.S a rc.M

Skript `/etc/rc.d/`**`rc.S`** inicializuje systém a spustí se, když OS bootuje. Skript provede následující operace:

- spustí devfs démon: `/sbin/`**`devfsd`** `/dev`
- nahradí některé oblasti root filesystemu pomocí tmpfs: `/etc/rc.d/`**`rc.cfroot`** `start`
- umožní swapping: `/sbin/`**`swapon`** `–a`
- zkontroluje souborový systém: `/sbin/`**`fsck`** `–f –C –a`
- naloaduje kernel moduly z /lib/modules/: `/etc/rc.d/`**`rc.modules`**
- a další

Skript `/etc/rc.d/`**`rc.M`** se spouští ve víceuživatelském módu a provede následující operace:

- inicializuje síťový hardware: `/etc/rc.d/`**`rc.inet1`**
- spustí síťové démony: `/etc/rc.d/`**`rc.inet2`**
  - SSH démon: `/etc/rc.d/`**`rc.sshd`** `start`
  - NFS server: `/etc/rc.d/`**`rc.nfsd`** `start`
- spustí system logger: `/etc/rc.d/`**`rc.syslog`** `start`
- spustí web server: `/etc/rc.d/`**`rc.httpd`** `start`
- spustí Samba server: `/etc/rc.d/`**`rc.samba`** `start`

- inicializuje SystemV: `/etc/rc.d/rc.sysvinit`
- spustí proceduru lokálního nastavení: `/etc/rc.d/rc.local`
- a další

Skript rc.local

Skript `/etc/rc.d/rc.local` se spouští ve skriptu `/etc/rc.d/rc.M` a provede následující operace:

- načte rtl moduly (rtl, rtl_malloc, rtl_sched):

```
modprobe rtl
modprobe rtl_malloc
modprobe rtl_sched
```

- vloží lincan driver do jádra Linuxu (kap. *3.3.2 LinCAN driver*):

```
insmod /home/robot/Modules/lincan.o hw=bfadcan irq=12 \
io=0x200  clock_freq=20000000 \
extended=1 pelican=1 processlocal=1 baudrate=500
```

- vloží framegrabber driver do jádra Linuxu (kap. *3.6 Framegrabber*):

```
insmod -f /home/robot/Modules/hrt.o > /dev/null
mknod /dev/hrt c 254 0
chown robot /dev/hrt
chmod 666 /dev/hrt
```

- vloží watchdog driver do jádra Linuxu (kap. *3.5 Watchdog*):

```
insmod -f /home/robot/Modules/sc520_wdt.o > /dev/null
```

- spustí FPLServer:

```
/home/robot/Web/fplserver -f /home/robot/Web/policy.xml &
```

- spustí aplikaci pro čtení snímků z framegrabberu (kap. *3.6.2 Aplikace readf*):

```
/home/robot/Control/readf &
```

- spustí demonstrační aplikaci přes startovací skript (kap. *3.4 Demonstrační řídicí aplikace control*):

```
/home/robot/Control/bin/rtwatchjob start \
/home/robot/Control/control &
```

Skript rtwatchjob

Skriptem `/home/robot/Control/bin/rtwatchjob` se spouští nebo ukončuje řídicí aplikace (viz kap. *3.7.3 Spuštění řídicí aplikace pomocí skriptu rtwatchjob*). Skript zaprvé nastavuje scheduler jako "realtime" FIFO scheduler a přiřadí aplikaci vysokou prioritu a zadruhé spouští watchdog (viz kap. *3.5 Watchdog*).

Skript lze použít následovně:

```
./rtwatchjob start /ABSOLUTE/PATH/TO/CONTROLAPP
```

```
./rtwatchjob stop
```

3.3 CAN

Ke sběrnici CAN jsou připojeny 3 zařízení jako slave: levý motor, pravý motor a IO-modul. CAN modul na PC/104 komunikuje s těmito zařízeními jako master. S IO-modulem komunikuje pomocí protokolu CAN, s motory pomocí protokolu CANopen.

Parametry použité sběrnice CAN:

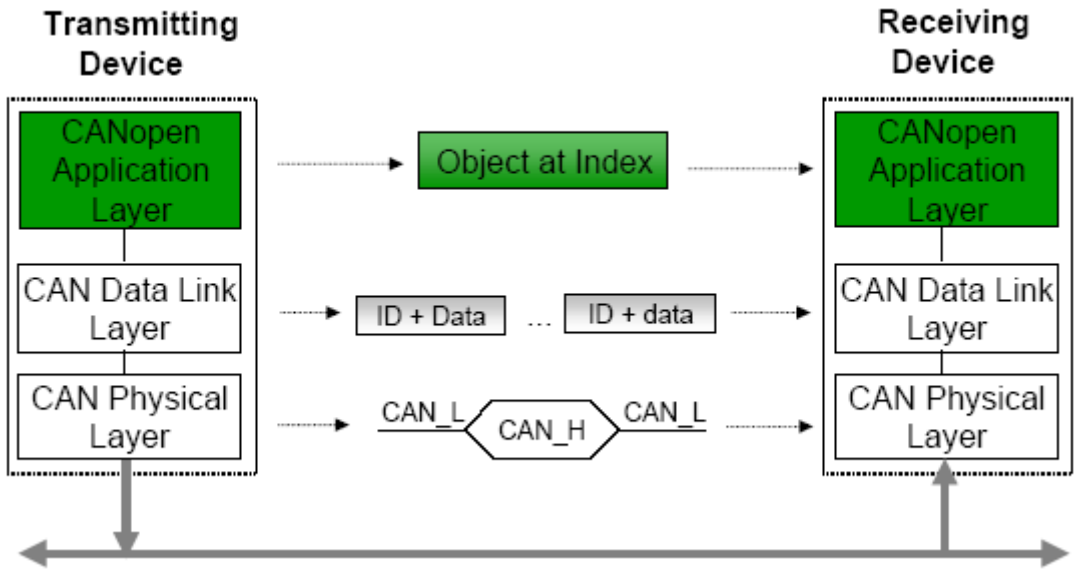
- přenosová rychlost: 500 Mb/s
- zakončovací odpor: 128 ohm
- používá 3 signály: CANH, CANL, GND

Každé zařízení v síti je identifikováno unikátní adresou uzlu:

Zařízení	Adresa uzlu
IO-modul	0x01
levý motor	0x02
pravý motor	0x03

tabulka 5: Adresy uzlů

3.3.1 Protokol CAN/CANopen



Obrázek 3-3: Vrstvy protokolu a jejich interakce

Protokol CAN obsahuje popis pouze první a druhé vrstvy podle modelu ISO OSI. Protokol CANopen definuje i vrstvu aplikační.

V souvislosti s protokolem CANopen se používají pojmy jako objekt, adresář objektů, PDO, SDO, index, subindex, COB ID apod. Některé z těchto pojmů jsou vysvětleny níže.

Adresář objektů (Object Dictionary OD) obsahuje část s obecnou specifikací zařízení, jako jeho název, výrobce, komunikační parametry atd., a potom část, která obsahuje určitou funkčnost zařízení, parametry a data. Každá položka v adresáři se nazývá **objekt** a je určena 16-bitovým indexem a 8-bitovým subindexem.

Rozlišují se dva základní mechanismy přenášení zpráv na CANopen. Procesní data určená pro časově kritickou výměnu, nazývaná Process Data Objects **PDO**, a služební zprávy, jejichž doručení není tolik omezeno časem, nazývané Service Data Objects **SDO**. Služební zprávy se používají především pro přenášení parametrů při konfiguraci zařízení nebo pro přenášení delších zpráv. Zpráva na CANu totiž může obsahovat maximálně 8 bytu dat.

Procesní data se přenášejí buď cyklicky, při změně nebo na vyžádání jako broadcastové zprávy bez dodatečné režie. Rozmístění aplikačních objektů (položek z adresáře objektů) do přenášeného objektu PDO je určeno pomocí tzv. **mapování PDO** – tato informace je uložena v adresáři objektů a může být změněna podle požadavku aplikace (uživatele).

Přenos zpráv SDO odpovídá potvrzovanému přenosu podle modelu client/server a je prováděn jako komunikace mezi dvěma účastníky. Odpovídající položka je v adresáři objektů popsána příslušným indexem a subindexem, delší zprávy jsou fragmentovány, takže maximální délka vyměňovaných dat není omezena. Tento typ přenosu vyžaduje dodatečnou režii protokolu.

Počáteční nastavení a spuštění celého distribuovaného systému provádí správa sítě – Network Management NMT. Zprávy CAN obsahují 11-bitový identifikátor, který mimo jiné určuje také prioritu zprávy (přístup ke sběrnici CAN je založen na rozpoznávání priorit zpráv při případné kolizi a na zrušení zprávy s nižší prioritou). Na CANopen je tento identifikátor označován **COB ID** – Communication Object ID a příslušný COB ID je přidělován jednotlivým zprávám PDO nebo SDO v adresáři objektů. Přitom platí, že čím nižší je COB ID, tím vyšší je priorita zprávy.

3.3.2 LinCAN driver

Umístění důležitých souborů:

- driver-modul (na PC/104): `/home/robot/Modules/`**`lincan.o`**
- device file (na PC/104): `/dev/can0`
- zdrojové soubory (v CVS): `ocera/components/comm/can/`**`lincan/src/`**
- hlavičkové soubory (v CVS): `ocera/components/comm/can/`**`lincan/include/`**
- utility (v CVS): `ocera/components/comm/can/`**`lincan/utils/`**

Modul lincan.o

LinCAN driver implementuje CAN driver pro OS Linux. Rozhraní CAN (CAN modul PC/104) je zpřístupněno CAN driverem jako znakové zařízení. Soubor reprezentující toto zařízení (device file) je umístěn v `/dev/can0` na PC/104.

Aplikační program komunikuje s tímto driverem pomocí standardních nízkoúrovňových systémových vstupně/výstupních operací (open, close, read, write, select a ioctl).

Driver-modul `lincan.o` se načte během startu OS ve skriptu `rc.local` (viz kap. *3.2.3 Inicializační skripty*).

Struktura canmsg_t

Struktura `canmsg_t` je definována v souboru …`/lincan/include/`**`canmsg.h`**. Tato struktura reprezentuje CAN zprávu.

```
struct canmsg_t {
      int             flags;
      int             cob;
      canmsg_id_t     id;
      canmsg_tstamp_t timestamp;
      unsigned short  length;
      unsigned char   data[CAN_MSG_LENGTH];
};
```

Struktura canfilt_t

Struktura `canfilt_t` je definována v souboru …`/lincan/include/`**`canmsg.h`**. Používá se pro nastavení akceptačního filtru.

```
struct canfilt_t {
      int flags;
      int queid;
      int cob;
      unsigned long id;
      unsigned long mask;
};
```

Ioctl příkaz CANQUE_FILTER

Ioctl příkaz (makro) CANQUE_FILTER je definován v souboru …`/lincan/include/`**`can.h`**. Nastavuje akceptační filtr pro frontu CAN zpráv.

Příkaz lze volat pomocí funkce ioctl:

int ioctl(int fd, int command = CANQUE_FILTER, struct canfilt_t * filt);

kde *fd* je file descriptor otevřeného objektu zpráv (can message communication object), *command* značí příkaz pro filtr CAN fronty, CANQUE_FILTER, *filt* je ukazatel na strukturu canfilt_t.

3.3.3 VCA

Umístění důležitých souborů:

- zdrojové a hlavič. soubory (v CVS):

`ocera/components/comm/can/`**`canvca/libvca/`**

Přímá komunikace s driverem pomocí systémových volání se nedoporučuje, protože toto rozhraní je částečně systémově závislé a nepřenositelné do všech prostředí. Doporučenou alternativou je použít knihovnu VCA (Virtual CAN API), kterou poskytuje OCERA. Knihovna VCA definuje přenositelné a čisté rozhraní k implementaci CAN driveru.

Hlavičky základních funkcí pro práci s VCA jsou definovány v
…/canvca/libvca/**can_vca.h**

function	description
vca_open_handle	opens new VCA handle from CAN driver
vca_close_handle	closes previously acquired VCA handle
vca_send_msg_seq	sends sequentially block of CAN messages – pracuje se strukturou canmsg_t
vca_rec_msg_seq	receive sequential block of CAN messages – pracuje se strukturou canmsg_t
vca_wait	blocking wait for the new message(s)

Tabulka 6: Základní funkce pro práci s VCA

Struktura vcasdo_fsm_t

Struktura reprezentující SDO FSM (Finite State Machine) je definována v
…/canvca/libvca/**vcasdo_fsm.h**

```
struct vcasdo_fsm_t {
      unsigned srvcli_cob_id;
      unsigned clisrv_cob_id;
      unsigned node;
      unsigned index, subindex;

      struct timeval last_activity;

      int bytes_to_load;
      int toggle_bit:1;

      int is_server:1; /* client/server */
      int is_uploader:1; /* upload/download */
      int state;
      vcasdo_fsm_state_fnc_t *statefnc;

      int err_no;

      ul_dbuff_t data;
      canmsg_t out_msg;
} vcasdo_fsm_t;
```

function	description
vcasdo_init_fsm	init SDO FSM
vcasdo_destroy_fsm	
vcasdo_fsm_upload	starts upload SDO communication protocol for this FSM
vcasdo_fsm_download	starts download SDO communication protocol for this FSM
vcasdo_fsm_taste_msg	try to process msg in FSM

Tabulka 7: Funkce pro práci se strukturou vcasdo_fsm_t

Struktura ul_dbuff

Struktura a funkce pro práci s obecně použitelným bufferem pro dynamická data jsou definovány v **ul_dbuff.h**

```
typedef struct ul_dbuff {
      unsigned long len;
      unsigned long capacity;
```

```
        int flags;
        unsigned char *data;
        unsigned char sbuff[UL_DBUFF_SLEN];
} ul_dbuff_t;
```

3.3.4 IO-Modul

IO-modul je z hlediska hardwaru popsán v kapitole *2.3.1 Obvod s IO-modulem*.

IO-modul (slave) komunikuje s CAN modulem PC/104 (master) pomocí protokolu CAN.

Obvod s IO-modulem poskytuje analogové vstupy a digitální vstupy/výstupy. Na digitální vstupy lze připojit až 4 rotační enkodéry. Jejich popis je uveden v tabulce 7. Frekvenční vstupy a výstupy nejsou vyvedeny na svorkovnici (ID 151 resp. 161).

Analogová hystereze funguje pro analogové vstupy AIN0, AIN1 a AIN2, na kterých jsou připojeny senzory přední, levý a pravý. Meze hystereze jsou nastaveny pro všechny 3 vstupy stejně a odpovídají vzdálenosti 200 mm od senzoru. Analogová hystereze zde funguje jako analogový trigger, tj. bity AH0, AH1 a AH2 jsou nastaveny na log.1 pokud je překážka blíže než 200 mm.

Hodnoty analogových vstupů jsou posílány IO-modulem na CAN automaticky s periodou 20 ms. Není tedy potřeba posílat žádost o tyto hodnoty (narozdíl od digitálních vstupů).

Typ I/O	**ID**	**byte**								**Rozsah**
		0.	**1.**	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	
Analogové vstupy	101	AIN0		AIN1		AIN2		AIN3		0..1023
	111	AIN4		AIN5		AIN6		AIN7		
Digitální vstupy	121	DIN0	DIN1	DIN2	DIN3	DIN4	DIN5	DIN6	DIN7	0..1
Digitální výstupy	131	DOUT0 až DOUT7								0..1
Analogová hystereze	141	AH0 AH1 AH2								0..1
Rotační enkodéry	171	RE0		RE1		RE2		RE3		0..65535

Tabulka 8: Popis vstupů a výstupů IO-modulu z hlediska protokolu CAN

V následujících příkladech nejsou pro přehlednost ošetřeny návratové hodnoty funkcí. Příklady 2 a 3 využívají proměnné z příkladu 1.

Příklad 1: Inicializace IO-modulu:

```
vca_handle_t canhandle;
vca_open_handle(&canhandle, "/dev/can0", NULL, 0);

/* nastavi ID a masku filtru */
struct canfilt_t canfilt = {0,0,0,0x101,0x101};
int fd = vca_h2fd(canhandle);
ioctl(fd, CANQUE_FILTER, &canfilt);
```

Příklad 2: Načte stav digitálního vstupu DIN3 IO-modulu:

```
struct canmsg_t sendmsg={MSG_RTR,0,0x121,{0,0},8,{1,2,3,4,5,6,7,8}};
```

```
struct canmsg_t readmsg;
readmsg.length = 8;     //delka dat 8 bytu
      /* odesle zadost */
vca_send_msg_seq(canhandle, &sendmsg, 1);
do {
            /* ceka na prichozi zpravu */
      vca_wait(canhandle, 0, 0);
            /* prijme zpravu */
      vca_rec_msg_seq(canhandle, &readmsg, 1);
            /* pokud se prijata zprava tyka DIN3, tak skonci */
} while (readmsg.id != 0x121);
      /* nacte 3.byte, tj. DIN3 */
char din3 = readmsg.data[3];
```

Příklad 3: Nastaví digitální výstupy IO modulu – DOUT5 na log.1 ostatní na log.0:

```
      //32...b00100000
struct canmsg_t sendmsg = {0,0,0x131,{0,0},8,{32,2,3,4,5,6,7,8}};
vca_send_msg_seq(canhandle, &sendmsg, 1);
```

3.3.5 Krokové motory

Krokové motory jsou z hlediska hardwaru popsány v kapitole *2.4 Krokové motory*.

Krokové motory (slave) komunikují s CAN modulem PC/104 (master) pomocí protokolu CANopen.

Kompaktní jednotka (řídicí systém + motor) získává data a příkazy od mastera. Řídicí systém jednotky posílá stavové informace (stav zařízení a stav procesu) zpátky masteru jako potvrzení. Rychlostní mód má potvrzovací parametr `VEL.stateVel`. PTP mód má potvrzovací parametr `PTP.statePTP`.

V tabulce 5 jsou popsány skupiny parametrů motoru, skupiny použitelné pro řízení robotu jsou zvýrazněny tučně.

Skupina	**Popis**
CAN	CAN bus settings
Capture	"Fast position capture" function
Commands	Save parameter status change in EEPROM Initialise default parameters
Config	Drive configuration
ErrMem0	Error memory
Homing	"Referencing“ operating mode
I/O	Status and definition of inputs and outputs
Motion	Operating function "Definition of direction of rotation“ Operating function "Quick-Stop“ Default setpoint speed Acceleration and deceleration
Profibus	Profibus settings
ProgIO0..3	Operating function "Programmable inputs/outputs“
PTP	"Point-to-point" operating mode
RS485	RS485 bus settings
Settings	User device names Phase currents Monitoring inputs
Status	Status information
VEL	"Speed mode" operating mode

Tabulka 9: Skupiny parametrů motoru pro CANopen

V tabulce 8 jsou popsány vybrané parametry použitelné pro řízení robotu. K indexu je nutno přičíst **0x3000** (např. u objektu `Commands.driveCtrl` se použije místo indexu 0x1C index 0x301C).

Skupina.Jméno objektu	Index:Subindex dec. (hex.)	Popis parametru (objektu)	Rozsah
CAN.canAddr	23:2 (17:02h)	Address CAN Bus	1..127
CAN.canBaud	23:3 (17:03h)	Baud rate CAN Bus v kBaud	20..1000
Commands.driveCtrl	28:1 (1C:01h)	Control word for status change Bit0: Disable power amplifier Bit1: Enable power amplifier Bit2: Quick-Stop Bit3: FaultReset Bit4: Quick-Stop release Bit5..15: reserved	0..31
Motion.invertDir	28:6 (1C:06h)	Definition of the direction of rotation Value 0: no inversion of direction Value 1: direction reversal active	0..1
Motion.dec_Stop	28:21 (1C:15h)	Delay for Quick-Stop Deceleration used for every Quick-Stop: - Quick-Stop via control word - Quick-Stop via ext. Monitoring signal - Quick-Stop by error of classes 1,2 (Deceleration for "Quick-Stop")	12..765000 [rpm*s]
Motion.v_target0	29:23 (1D:17h)	Default setpoint speed Remanent default value for v_tarPTP. Speed for PTP mode if no value has been written to v_tarPTP. Note: This remanent value is used exclusively when switching on the drive as a default assignment for v_tarPTP.	1..3000 [rpm]
Motion.acc	29:26 (1D:1Ah)	Acceleration Value determines acceleration and deceleration. New values are only imported after drive standstill (Acceleration and deceleration in general.)	12..765000 [rpm*s]
PTP.p_absPTP	35:1 (23:01h)	Target position and absolute positioning start action object: Write access triggers absolute positioning in increments (Absolute position + start positioning)	INT32 [inc]
PTP.StatePTP	35:2 (23:02h)	Acknowledgment: PTP positioning Bit15: ptp_err Bit14: ptp_end	UINT16 [-]

		Bit13: Set position reached Bit7: SW_STOP	
PTP.p_relPTP	35:3 (23:03h)	Path and relative positioning start action object: Write access triggers relative positioning in increments (Relative position + start positioning)	INT32 [inc]
PTP.v_tarPTP	35:5 (23:05h)	Setpoint speed for PTP positioning Default is *Motion.v_target0*	1..3000 [rpm]
Status.v_act	31:2 (1F:02h)	Actual speed Speed captured by sensor with closed loop drives. With open-loop drives the commutation speed.	INT32 [inc/s]
Status.n_act	31:9 (1F:09h)	Actual speed Corresponds to v_act converted to rpm.	INT16 [rpm]
Status.UDC_act	31:20 (1F:14h)	Voltage of power supply	UINT16 [0,1V]
Status.TPA_act	31:25 (1F:19h)	Temperature of power amplifier	20..110 [st. C]
VEL.velocity	36:1 (24:01h)	Start with setpoint speed in speed-profile mode action object: Write access triggers movement (Constant speed mode.)	-3000.. 3000 [rpm]
VEL.stateVEL	36:2 (24:02h)	Acknowledgement: Speed profile mode Bit15: vel_err Bit14: vel_end Bit13: setpoint speed reached Bit7: SW_STOP	UINT16 [-]

Tabulka 10: Vybrané parametry použitelné pro řízení robotu

Jednotky použité v tabulce: inc…increments, rpm…rotates per minute

Zesilovač výkonu motoru:

K tomu, aby mohla být jednotka polohována, musí být zesilovač zapnutý (parametr `Commands.driveCtrl`, bit1=log.1). Pak motorem prochází proud a kompaktní jednotka je připravena k polohování.

Funkce Quick-Stop:

Nastavením bitu 2 na log.1 parametru `Commands.driveCtrl` se motor okamžitě zastaví specifikovaným zpomalením Quick-Stop (parametr `Motion.dec_Stop`). Po zastavení proud stále prochází motorem (funguje jako brzda), ale nemůže být dále polohován. K tomu je nutné uvolnit Quick-Stop nastavením bitu 4 na log.1 parametru `Commands.driveCtrl`**.**

Příklad 1: Relativní polohování PTP

1. Nastavit požadovanou hodnotu rychlosti (např. 600 rpm) pomocí parametru `PTP.v_tarPTP`

2. Nastavit relativní vzdálenost (např. 1000 inc) a odstartovat polohování pomocí parametru `PTP.v_relPTP`

3. Dosažená poloha je indikovaná bitem 13 v parametru `PTP.StatePTP`

Příklad 2: Inicializace levého motoru:

```
vca_handle_t canhandle;
vca_open_handle(&canhandle, "/dev/can0", NULL, 0);

/* nastavi ID a masku filtru */
struct canfilt_t canfilt = {0,0,0,0x400,0x400};
int fd = vca_h2fd(canhandle);
ioctl(fd, CANQUE_FILTER, &canfilt);

/* inicializuje strukturu FSM */
vcasdo_fsm_t fsm;
vcasdo_init_fsm(&fsm, 0, 0, 0x02); //node leveho motoru = 0x02
```

Příklad 3: Zjistí aktuální rychlost levého motoru:

```
      //node leveho motoru = 0x02
      //index:subindex = 0x301F:0x09 (parametr Status.n_act)
vcasdo_fsm_upload(fsm, 0x02, 0x301F, 0x09, 0, 0);
vca_send_msg_seq(canhandle, &fsm->out_msg, 1);

/* V cyklu cte prichozi zpravy (po 8 bytech)
 * dokud není fsm->state == sdofsmDone
 * Vysledek je ulozen ve fsm->data->data
 */

//Vice viz sendSDO() v can_motor_ifs.c
```

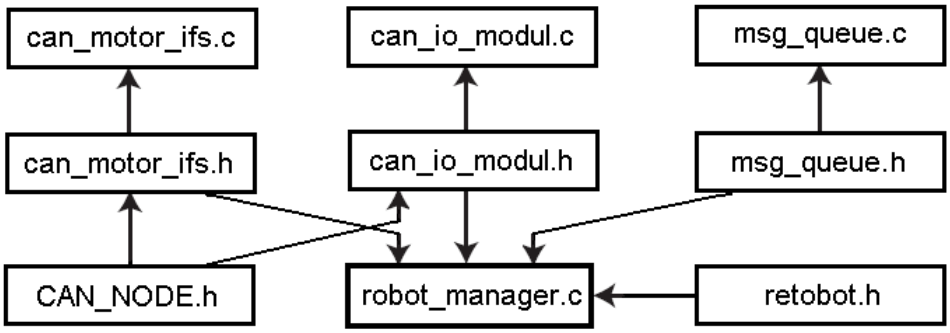
Příklad 4: Nastaví rychlost levého motoru na 5 rpm:

```
      //node leveho motoru = 0x02
      //index:subindex = 0x3024:0x01 (parametr VEL.velocity)
char speed[2] = {0x05, 0x00};
ul_dbuff_cpy(&fsm->data, speed, 2);
vcasdo_fsm_download(fsm, &fsm->data, 0x02, 0x3024, 0x01, 0, 0);
vca_send_msg_seq(canhandle, &fsm->out_msg, 1);
```

3.4 Demonstrační řídicí aplikace control

Umístění důležitých souborů:

- aplikace `control` (na PC/104): `/home/robot/Control/`
- zdrojové a hlavičkové soubory (v CVS): `[cvs]/sw/qt-reto/`



Obrázek 3-4: Struktura zdrojových souborů aplikace control

Aplikace `control` umožňuje řídit robot vzdálenou aplikací pomocí TCP socketů. S touto aplikací komunikuje komunikačním protokolem popsaným v kapitole 6.2.

Aplikace se spouští po startu OS ve skriptu `rc.local` (viz kapitola *3.2.3 Inicializační skripty*).

3.4.1 can_io_module.c

function	description
cleanIODevice	Close the CAN IO-Module.
initIODevice	Init the IO device.
io_set_fout	Set freqenucy output
io_get_fin	Get specified frequency input
io_get_din	Get specified digital input
io_get_ain	Get specified analogue input
io_set_dout	Set digital outputs
thr_ir_catch	
catch_IR_trigger	Wait for trigger events form the anloge hysterese.

Tabulka 11: Funkce definované v can_io_modul.h

3.4.2 can_motor_ifs.c

function	description
subtimeval	
send_to_can	
m_initCAN	Init the CAN interface.
m_init	Init the Motor.
m_cleanup	Cleanup the motor structure.
m_amplifier_on_SDO	Switch on motor amplifier using SDO communication.
m_amplifier_on	Switch on motor amplifier.
m_getSpeed	Print out the actual Motor speed.
m_getDriveStat	Print out the actual Motor speed.
m_startMotor_SDO	Start the motor using SDO communication.
m_stopMotor_SDO	Stop the motor using SDO communication.
m_stopMotor	Stop the motor.

m_changeSpeed_SDO	Change the speed using SDO communication.
m_changeSpeed	Change the speed.
m_setAcc	Set the acc/deccelleartion.
sendSDO	

Tabulka 12: Funkce definované v can_motor_ifs.h

3.4.3 msg_queue.c

function	description
init_queue	
enqueue	
dequeue	
empty	

tabulka 13: Funkce definované v msg_queue.h

3.4.4 robot_manager.c

function	description
send_can_msg	
thr_can_command	
start_amps	
stop_all	
setSpeedRobot	
thr_loadstop	
getVoltage	
getChargeStates	
thr_charger	
execute_client_cmd	
read_from_client	
write_acception	
make_socket	
thr_comm_server	
thr_update_clt	
thr_periodic	
thr_wtd_trigger	
callb_func	
init_robot	
start_thr	
join_thr	
termination_handler	

tabulka 14: Funkce implementované v robot_manager.c

3.5 Watchdog

Umístění důležitých souborů:

- driver-modul (na PC/104): `/home/robot/Modules/`**`sc520_wdt.o`**
- device file (na PC/104): `/dev/`**`watchdog`**

Driver-modul `sc520_wdt.o` se načte během startu OS ve skriptu `rc.local` (viz kapitola *3.2.3 Inicializační skripty*).

Aby se robot rebootoval automaticky v případě výpadku systému, je na robotu implementován časovač watchdog. Po nahrání a spuštění řídicí aplikace na robotu pomocí skriptu `rwatchjob` (viz kapitola *3.7.4 Spuštění startovacího skriptu rtwatchjob*) se watchdog spustí a čeká cca 2s na další signál. Pokud se v této době neobjeví žádný signál, systém bude okamžitě rebootován. Řídicí aplikace by tedy měla posílat signál do watchdogu alespoň každé 2s.

Příklad 1: Posílá watchdogu signál (znak ´\0´) 1x za vterinu.

```
struct timespec timeout = {.tv_sec = 1, .tv_nsec = 0};
struct timespec rem;
int fd_wdt = open ("/dev/watchdog", O_WRONLY);
while(1){
       write (fd_wdt, "\0", 1);
       nanosleep (&timeout, &rem);
}
```

3.6 Framegrabber

3.6.1 Modul hrt.o

Umístění důležitých souborů:

- driver-modul (na PC/104): `/home/robot/Modules/`**`hrt.o`**
- zdrojové a hlavičkové soubory: `hrt.c, hrt.h, sysdep.h`

  `[cvs]/sw/hrt/`

Framegrabber je inicializován při startu systému driver-modulem `hrt.o`. Soubor reprezentující toto zařízení (device file) je umístěn v `/dev/hrt` na PC/104.

Příklad 1: Načte do bufferu jeden řádek obrazu (768 pixelů) na dané pozici (200) z framegrabberu:

```
#include <stdio.h>

#define WINSIZEX 768
#define LINE_NR 200

int hfd;
unsigned char line_buffer[WINSIZEX];    /* line buffer */

hfd = open ("/dev/hrt", O_RDWR);        /* open framegrabber device */
lseek (hfd, LINE_NR*WINSIZEX, SEEK_SET);/* set file position */
read (hfd, line_buffer, WINSIZEX);     /* read the pixels into buffer*/
close (hfd);                            /* close device file */
```

Kromě standardních souborových operací (open, lseek, read) umožňuje modul `hrt.o` použít operaci ioctl pro získání nebo nastavení parametrů framegrabberu. Ve svém hlavičkovém souboru `hrt.h` definuje ioctl příkazy, které lze volat pomocí funkce

```
int ioctl(int fd, int cmd, arg);
```

kde `fd` je identifikátor otevřeného souboru (file descriptor), `cmd` je ioctl příkaz (makro), `arg` je ukazatel na parametr, který chceme číst nebo nastavit.

Příklad 2: Nastaví jas pomocí ioctl a načte celý obraz do bufferu:

```
#include <stdio.h>
#include <sys/ioctl.h>
#include "hrt.h"

int hfd;
unsigned char buffer[HRT_FRAMESIZE];
struct i2c_regval brightness = {HRT_BRIGHTNESS_REG, 128};

hfd = open ("/dev/hrt", O_RDWR);
if (ioctl(hfd, IOC_HRT_SET_I2CREG, &brightness) < 0) {
    perror("ioctl IOC_HRT_SET_I2CREG");
}
read (hfd, buffer, HRT_FRAMESIZE);
close (hfd);
```

Hlavičkový soubor `hrt.h` definuje následující struktury, konstanty a ioctl příkazy:

```
struct subwindow {
    int width, height;
    int startx, starty;
};

struct i2c_regval {
    int reg;            /* Register number */
    unsigned char val;  /* Value to set it to */
};

/**
 * Card specific constants
 */
#define HRT_CONTROL_REG       0x2000
#define HRT_Y_LOW_REG         0x2002
#define HRT_Y_HIGH_REG        0x2003


/* PAL 8-bit greyscale */
#define HRT_WIDTH             768
#define HRT_HEIGHT            512
#define HRT_BYTES_PER_PIXEL   1
#define HRT_BYTES_PER_LINE    (HRT_WIDTH * HRT_BYTES_PER_PIXEL)
#define HRT_FRAMESIZE         (HRT_WIDTH * HRT_HEIGHT *
HRT_BYTES_PER_PIXEL)

/*
 * The commands for freeze, live, etc.
 */
#define HRT_FIELD_MASK        0x01
#define HRT_LIVE_CMD          0x91
#define HRT_FREEZE_IMM_CMD    0x5B
```

```
#define HRT_FREEZE_NEXT_CMD   0x99

/* A/D registers */
#define HRT_BRIGHTNESS_REG    0x19
#define HRT_CONTRAST_REG      0x13
```

ioctl příkaz	typ parametru	význam příkazu
IOC_HRT_SET_I2CREG	struct i2c_regval	nastaví hodnotu daného registru
IOC_HRT_GET_I2CREG	struct i2c_regval	načte hodnotu daného registru
IOC_HRT_SET_WIDTH	int	nastaví šířku ROI
IOC_HRT_SET_HEIGHT	int	nastaví výšku ROI
IOC_HRT_SET_STARTX	int	nastaví souřadnici levého horního rohu ROI
IOC_HRT_SET_STARTY	int	
IOC_HRT_SET_ROI	struct subwindow	nastaví ROI
IOC_HRT_GET_ROI	struct subwindow	načte parametry ROI
IOC_HRT_SET_I2CREGS	char *	

Tabulka 15: Ioctl příkazy v hrt.h

Pomocí ioctl příkazů v tabulce 15 lze číst nebo nastavovat hodnoty registrů a požadované oblasti snímku - ROI (Region of Interest).

3.6.2 Aplikace readf

Umístění důležitých souborů:

- aplikace `readf` (na PC/104): `/home/robot/Control/`
- zdrojové a hlavičkové soubory: `[cvs]/sw/FrameReader/readframe.c`

Jedná se o vzorovou aplikaci pro čtení snímku z framegrabberu.

Aplikace po spuštění otevře devicefile `/dev/hrt` a spustí funkci `readFrame`. Tato funkce provádí nekonečnou smyčku `while`, ve které načte celý snímek (768x512) do bufferu a provede jpeg kompresi. Během komprese průběžně ukládá výsledek do souboru `/var/tmp/tmpf.jpeg`. Po ukončení komprese přejmenuje soubor `tmpf.jpeg` na `frame.jpeg`. Na konci smyčky `while` čeká 1 sekundu a pak načítá snímek nový.

Snímek z kamery je tedy k dispozici ve `/var/tmp/frame.jpeg` a je obnovován 1x za vteřinu.

3.7 Doporučení pro napsání a spuštění řídicí aplikace

3.7.1 Použití knihoven projektu OCERA

Pro komunikaci s CAN budete potřebovat některé knihovny z následujících zdrojů projektu OCERA:

- `[cvs]/ocera/ocera/components/comm/can/`**`candev`**`/`
- `[cvs]/ocera/ocera/components/comm/can/`**`canvca/`**
- `[cvs]/ocera/ocera/components/comm/can/`**`utils/`**

CVS zdroje týkající se pouze CAN komponent získáte následovně:

```
cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/ocera login
cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/ocera co
          ocera/components/comm/can
```

Příkazy pro sestavení CAN komponent mimo stromovou strukturu OCERA:

```
cd ocera/components/comm/can
./switch2standalone
make
```

3.7.2 Upload řídicí aplikace na server robotu

Vlastní řídicí aplikaci můžete přenést na server robotu přes http://147.32.86.98/.

Na vstupní webové stránce můžete specifikovat vlastní soubor a provést upload na server robotu. Po úspěšném uploadu je aplikace přístupná přes symbolický odkaz `/home/stud/app`.

3.7.3 Spuštění řídicí aplikace pomocí skriptu rtwatchjob

Vlastní řídicí aplikaci lze spustit (po úspěšném uploadu) pomocí SSH Java appletu, který emuluje terminál. Po přihlášení aplikaci spustíte příkazem

```
rtwatchjob start ./app
```

Běžící aplikaci lze zastavit příkazem

```
rtwatchjob stop
```

Po přihlášení se dříve běžící demonstrační aplikace `control` přeruší. Aplikaci `control` lze znovu spustit příkazem

```
rtwatchjob start ./demoapp
```

Aplikace `control` se automaticky spustí po odhlášení uživatele nebo ztrátě připojení.

V případě, že spustíte svou aplikaci bez `rtwatchjob`, nemůže být zaručeno, že robot bude restartován po spadnutí systému.

3.8 Dodatek k software

V rámci bakalářské práce byly odstraněny zjištěné nedostatky na software robotu. Níže jsou popsány provedené úpravy.

V souboru `robot_manager.c` prováděla funkce `read_from_client` chybné parsování příchozích zpráv od klienta (původní flash aplikace). Tato funkce byla doplněna kódem provádějící správné parsování.

V souboru `robot_manager.c` byl upraven formát odchozích zpráv se standardním zakončením znakem ´/n´. Komunikační protokol mezi klientem a serverem byl zjednodušen a doplněn o nové příkazy. Původní flash aplikace po této úpravě je již nefunkční. Místo ní byla napsána nová grafická aplikace (viz. kapitola *5 Uživatelská grafická aplikace pro ruční řízení*).

Vlastní algoritmus řízení robotu byl zcela přepracován a doplněn o funkci couvání. Tento algoritmus byl přesunut z řídicí aplikace do uživatelské grafické aplikace.

4 Hřiště a nabíjecí místo

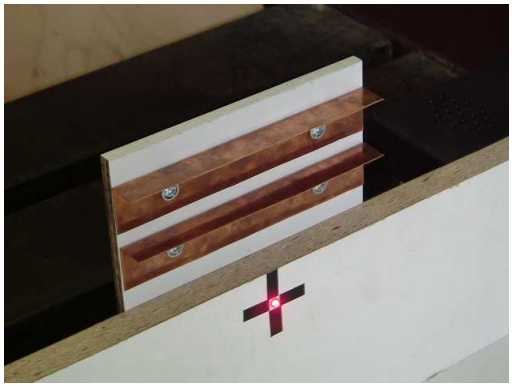
4.1 Hřiště pro robot

Rozměry hřiště: 104 x 183 cm

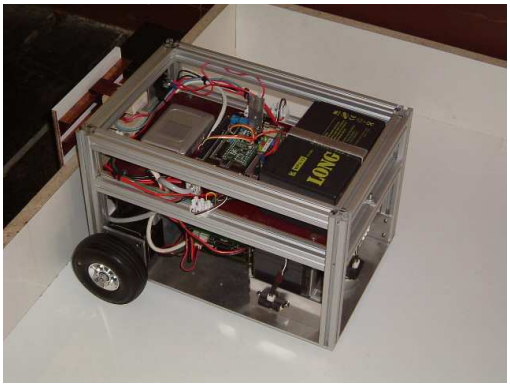
Výška mantinelu: 15 cm

Obrázek 4-1: Hřiště pro robot

4.2 Nabíjecí místo



Obrázek 4-2: Nabíjecí místo - pohled zepředu



Obrázek 4-3: Robot se nabíjí

Obrázek 4-4: Nabíjecí místo - pohled zezadu

Obrázek 4-5: Napájecí zdroj

Kontaktní plochy nabíjecího místa jsou vyrobeny z měděného plechu a jsou připojeny k napájecímu zdroji JS-75-300/DIN.

Napájecí zdroj je spínaný, odolný vůči zkratu na výstupu. Zatížení zdroje je indikováno pomocí 5 LED diod. Pokud dojde k vypnutí zdroje interními ochranami, opětovně se spustí odpojením od sítě na cca 5 minut.

Specifikace napájecího zdroje JS-75-300/DIN:

- výkon: 75 W
- vstupní napětí: 180-260 VAC
- výstupní napětí: 30 VDC
- vstupní frekvence: 47-63 Hz
- vstupní proud: 1 A / 230 VAC
- min. výstupní proud: 0,1 A

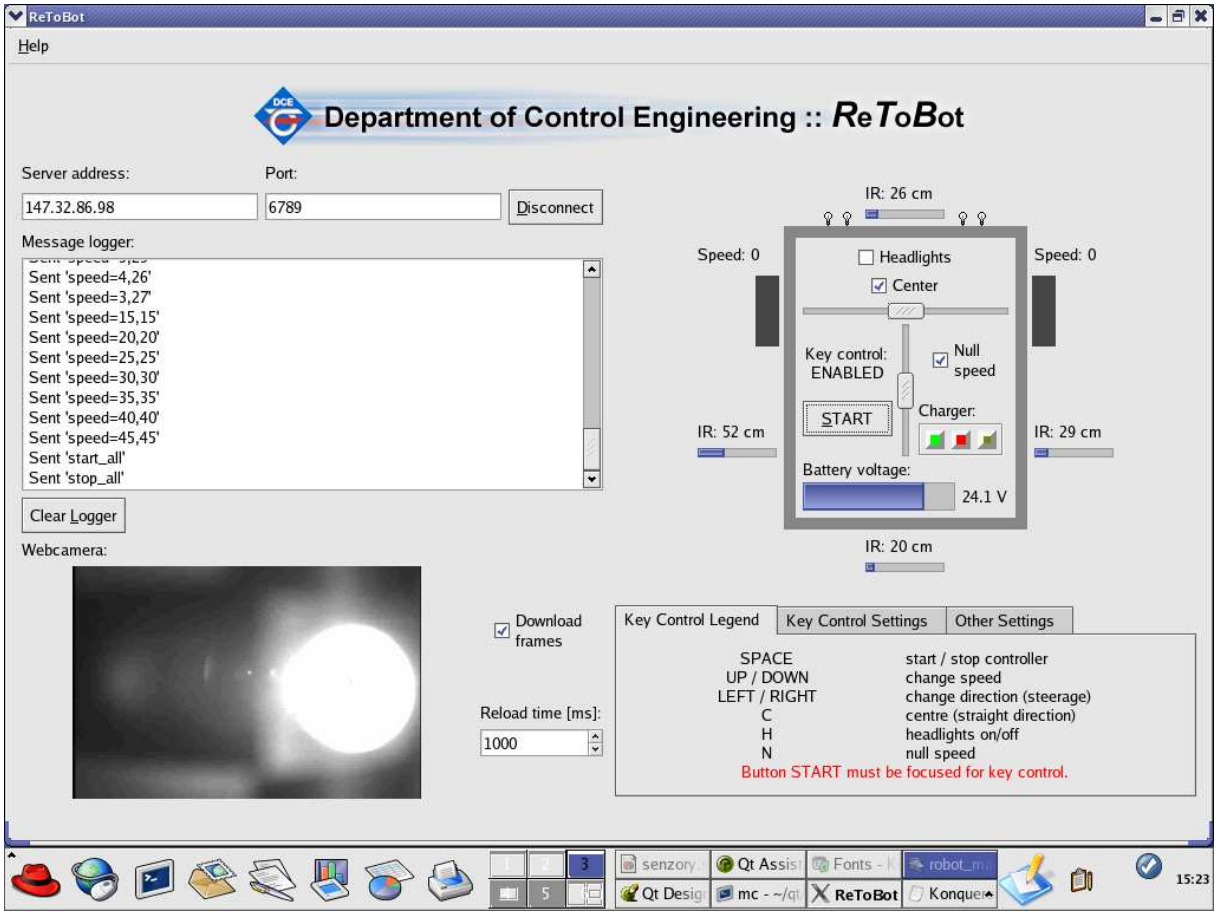
Nabíjecí místo je vybaveno LED diodou, která indikuje přítomnost napětí na nabíjecích kontaktech. Dioda by měla také sloužit pro snazší vyhledání nabíjecího místa robotem.

5 Uživatelská grafická aplikace pro ruční řízení

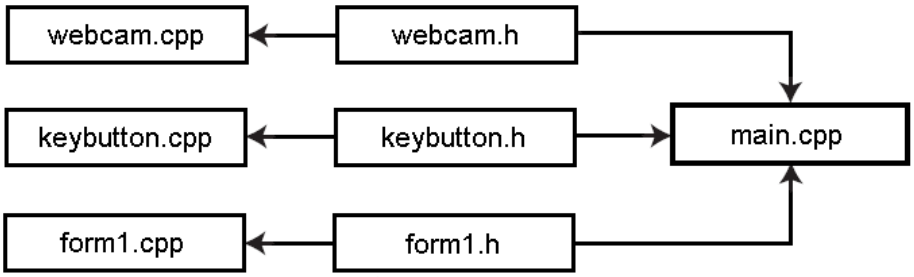
5.1 Přehled

Hlavním úkolem BP bylo vytvořit grafickou aplikaci pro ruční řízení robotu, která by běžela jak v systému Linux, tak ve Windows.

Aplikace byla psaná v C++ v multiplatformním QT Designeru v systému Linux. Aplikace využívá mechanizmu signálů a slotů pro komunikaci mezi objekty (widgety). Pro automatickou změnu velikosti a umístění grafických objektů ve formuláři je využito layoutů a spacerů.



Obrázek 5-1: Uživatelská grafická aplikace



Obrázek 5-2: Struktura zdrojových souborů grafické aplikace

5.2 Komunikace mezi klientem a serverem

Komunikace mezi klientem (grafickou aplikací) a serverem (aplikací běžící na robotu) probíhá pomocí socketů s protokolem TCP. Funkce pro tuto komunikaci byly napsány s využitím QT knihovny `<qsocket.h>`.

Po stisku tlačítka `Connect` se zavolá slot (signal handler) `btnConnect_clicked()`, který se pokusí navázat spojení se serverem na IP adrese 147.32.86.98 a portu 6789.

Sloty `socketConnected()`, `socketError(int e)`, `socketConnectionClosed()` a `socketClosed()` informují uživatele o navázání nebo přerušení spojení výpisem v okně Message Logger.

Po úspěšném připojení mohou klient a server komunikovat pomocí zpráv vypsaných v tabulce 16. Každá zpráva je zakončena znakem '\n'.

Zprávy odesílané serverem	**Význam zprávy**	**Zprávy odesílané klientem**	**Význam zprávy**
voltage=n		speed=l,p	
light_on		light_on	
light_off		light_off	
low_bat		start_all	
speed=l,p		stop_all	
ir_front=n		start_amp	
ir_left=n		stop_amp	
ir_right=n			
ir_back=n			
charge_power_on			
charge_power_off			
charging_on			
charging_off			
overcharging_on			
overcharging_off			
amplifier_off			

Tabulka 16: Komunikační protokol

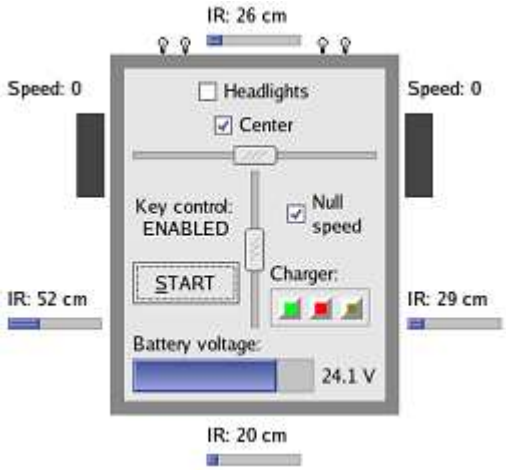
Klient posílá zprávy pomocí funkce `sendCommand(QString cmd)`, která využívá funkci `sendToServer(QString cmd)`. Zprávy přijímá po zavolání slotu `socketReadyRead()`, který používá funkci `parseMessage(QString msg)`.

Po stisku tlačítka `Disconnect` se uzavře spojení mezi klientem a serverem zavoláním funkce `closeConnection()`.

5.3 Zobrazované hodnoty

Na formuláři aplikace se zobrazují tyto hodnoty:

- rychlost levého a pravého kola v ot./min,
- vzdálenosti měřené 4 IR senzory buď v cm nebo v původních hodnotách (volitelné v nastavení),
- napětí akumulátorů buď ve voltech nebo v původních hodnotách,
- nabíjecí stavy (viz kap. *2.3.2 Obvod pro nabíjení olověných akumulátorů*),
- zhasnuté nebo rozsvícené světla na robotu (bílé LED diody).



Obrázek 5-3: Zobrazované hodnoty v aplikaci

Pozn.: původní hodnoty jsou údaje získané IO-modulem.

5.3.1 Kalibrace hodnot IR senzorů

OBRÁZEK

Obrázek 5-4: Graf závislosti výstupu IR senzoru na vzdálenosti

Byla napsána funkce `ir2cm()`, která převádí výstup A/D převodníku na cm podle vzorce

```
cm = -c * ln((ir - b)/a)
```

kde a, b, c jsou konstanty uvedené v tabulce 17. Funkce aproximuje graf na obrázku 5-4 dvěma exponenciálami.

	260 < ir <= 800	**800 < ir**
a	2000	4500
b	260	600
c	24	10

Tabulka 17: Konstanty pro funkci ir2cm

5.3.2 Kalibrace hodnot napětí na bateriích

```
volt = (ival – 1247) * (26.4 – 22.9) / (1435 – 1247) + 22.9
```

5.4 Ovládání robotu

Robot lze ovládat pomocí kláves nebo ovládacích prvků – tlačítka START, sliderů a zaškrtávacích políček. Ovládací klávesy lze použít jen pokud má tlačítko START zaměření.

Byl napsán zdrojový soubor `keybutton.cpp`, který obsahuje funkce pro obsluhu tlačítka START vytvořeného po spuštění aplikace. Umožňuje po zaměření tlačítka START snímat

stisknuté klávesy pomocí funkce `keyPressEvent()`. Tato funkce vyšle signál po stisku klávesy, který je následně zachycen slotem v hlavním formuláři. Podle druhu stisknuté klávesy vykoná tento slot odpovídající funkci. Dále obsahuje funkce `focusOutEvent()` a `focusInEvent()`, které podobný způsobem vysílají signál hlavnímu formuláři o ztrátě, resp. získání zaměření tlačítka.

Sloty `btnStart_focusOut(), btnStart_focusIn() a btnStart_keyPressed()` obsluhují signály vyslané tlačítkem START.

```
void Form1::startController()
```

```
void Form1::stopController()
```

ovládací klávesa	ovládací prvek	funkce
mezerník	tlačítko START	zapíná / vypíná zesilovače motorů
kurzorové klávesy UP / DOWN	svislý slider	mění průměrnou rychlost obou motorů
kurzorové klávesy LEFT / RIGHT	vodorovný slider	umožňuje zatáčet vlevo / vpravo
c	zaškrtávací políčko Center	nastaví přímý směr jízdy
h	zaškrtávací políčko Headlights	zapíná / vypíná světla
n	zaškrtávací políčko Null speed	zastaví robot

Tabulka 18:

Funkce `setMotorSpeed(int averageSpeed, int angle)` nastaví rychlosti obou motorů podle vstupních parametrů – průměrné rychlosti a úhlu.

5.5 Nastavení

OBRÁZEK

Obrázek 5-5: Záložka Key Control Settings

Záložka Key Control Settings umožňuje:

- nastavit citlivost klávesového řízení (krok rychlosti a směru),
- zvolit, zda při zpomalování přejde rychlost ihned na 0 nebo se bude snižovat zvoleným krokem
- zvolit, zda se při stisku kurzorových kláves UP a DOWN nastaví přímý směr
- zvolit, zda se při stisku kurzorových kláves LEFT a RIGHT nastaví přímý směr

OBRÁZEK

Obrázek 5-6: Záložka Other Settings

Záložka Other Settings umožňuje:

- nastavit maximální rychlost motorů v rozsahu 10 až 90 ot./min.,
- zvolit, zda se budou zobrazovat původní hodnoty napětí a IR senzorů,
- zvolit, zda se má robot zastavit po ztrátě zaměření tlačítka START
- zvolit, zda se má robot zastavit po detekci překážky

5.6 Obraz snímaný kamerou

Jednotlivé snímky z kamery je možné stahovat z webserveru běžícím na robotu s využitím QT knihoven `<qurloperator.h>` a `<qnetworkprotocol.h>`. Aktuální snímek z kamery je umístěn na http://147.32.86.98/webcam.jpeg.

Pro stažení souboru webcam.jpeg se volá `dataNet_slot(const QByteArray &data, QNetworkOperation *)`.

Po ukončení operace se volá `finishedNet_slot(QNetworkOperation *op)`. V případě úspěchu zobrazí stažený snímek, v opačném případě zobrazí chybovou hlášku.

Na formuláři aplikace lze povolit nebo zakázat stahování snímků z webserveru a nastavit periodu stahování.

Pro zobrazování snímků a chybových hlášek na formuláři aplikace byl napsán zdrojový soubor `webcam.cpp`.

6 Závěr

7 Seznam použité literatury

[1] Sobell, M. G.: *Linux – Praktický průvodce*, Computer Press, Praha 1999

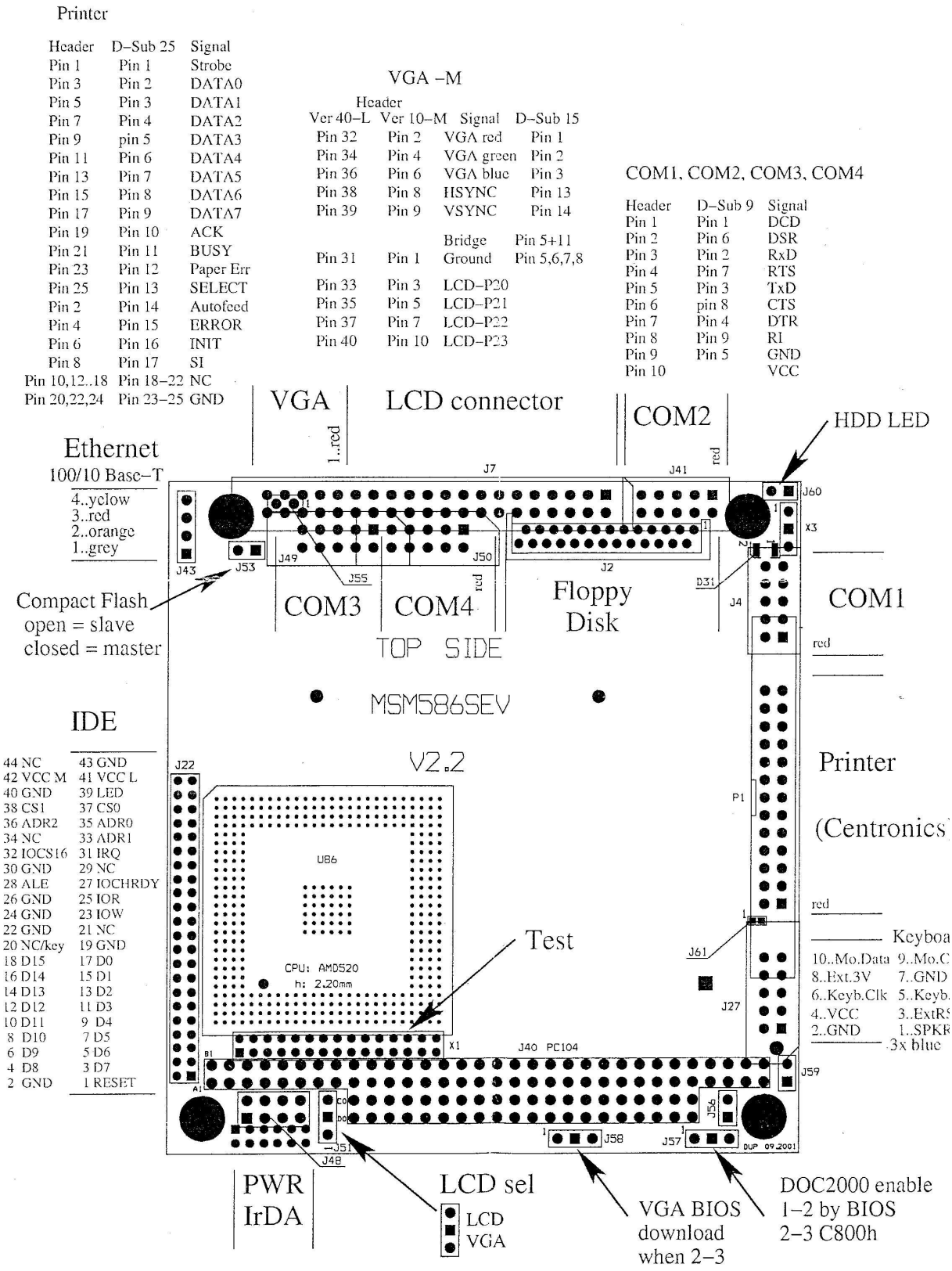
[2] Linux Device Drivers

[3] Technical Documentation IclA IFS, Field Bus Stepper Motor

8 Seznam příloh

Příloha 1 – Rozhraní modulu MSM586

MSM586SEN/SEV V2.1/V2.2



Příloha 2 – Mapa paměti framegrabberu

Horizontální řádky ve snímku jsou číslovány od 0 do 511 odshora dolů. Snímek je složen z lichého a sudého pole reprezentující liché, resp. sudé řádky. Ve výstupním signálu jsou pak všechny liché řádky následované všemi sudými (tj. prokládané video).

Organizace video paměti:

- Video paměť je přístupná přes paměťový adresový prostor procesoru (CMAS = CPU's memory address space). Jeden rastrový řádek je v CMAS najednou.
- Adresa video bufferu začíná na základní adrese desky plus 0.
- Adresa pixelu nejvíce vlevo v CMAS je rovna základní adrese desky + 0
- Adresa pixelu nejvíce vpravo v procesorovém paměťovém prostoru je rovna základní adrese desky + (horizontální rozlišení - 1)

Rastrové řádky jsou vybírány bankou v CMAS. Rastrový řádek, který je v CMAS, je vybrán 9-ti bitovým registrem (y-registr). 8 nejméně významných bitů (LSB) Y-registru jsou na paměťové adrese 2002h a nejvíce významný bit (MSB) je na paměťové adrese 2003h (bit 0).

9 bitů = 0..512

Tato organizace obrazového bufferu umožňuje (počítá s) velmi rychlým přístupem daného X,Y pixelu. Můžete jednoduše zapsat souřadnici y do Y-registru a číst nebo zapisovat byte na pozici X v obrazovém bufferu. Další operace (možnosti) vyžadují najít adresu pixelu v paměťovém bufferu porovnáním vzorce s argumenty X a Y.

Doporučený algoritmus jak číst z frame bufferu: číst liché řádky během ukládání, snímání (capturing) sudých řádků.

Mapa paměti:

Hodnoty uvedené ve sloupci adresa jsou hexadecimální a jsou ofsetem k bázové adrese zařízení. Hodnoty uvedené ve sloupci hrt.h jsou konstanty definované v hlavičkovém souboru hrt.h (viz kap. 2.3.1 Modul hrt.o).

hrt.h	adresa	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
	0000 ... 01ff	Video Image Buffer (VIB) 512B							
HRT_CONTROL_REG	2000	Grabbing Control Registr (GCR) (when writing)							
		DONE	GRAB	VS	V_DE	Y8	Y7	ETI	F_ID
	2001	not used					GO	DATA	CLOCK
HRT_Y_LOW_REG	2002	y7	y6	y5	y4	y3	y2	y1	y0
HRT_Y_HIGH_REG	2003	not used							y8
	2004								REP
	2005 ... 3fff								

Tabulka 19:

Registr GCR

Zápisem příslušné hodnoty (uvedené v tabulce…) do registru GCR se nastaví jeden z následujících módů framegrabberu:

1) Mód LIVE:

A/D převodník zapisuje nové pixely do Video Memory Buffer

2) Mód FREEZE_NEXT_VSP:

A/D převodník zastaví zapisování nových pixelů do Video Memory Buffer (příkaz se nevykoná, dokud neskončilo právě prováděné pole videa). Zastaví se na nejbližším pulsu vertikální synchronizace (VSP).

3) Mód FREEZE_IMMEDIATELY:

A/D převodník zastaví okamžitě zapisování nových pixelů do Video Memory Buffer (příkaz nebude čekat na skončení právě prováděného pole videa)

hrt.h	hodnota GCR	význam příkazu
HRT_LIVE_CMD	0x91	framegrabber přejde do módu LIVE
HRT_FREEZE_IMM_CMD	0x5B	framegrabber přejde do módu FREEZE_IMMEDIATELY
HRT_FREEZE_NEXT_CMD	0x99	framegrabber přejde do módu FREEZE_NEXT_VSP

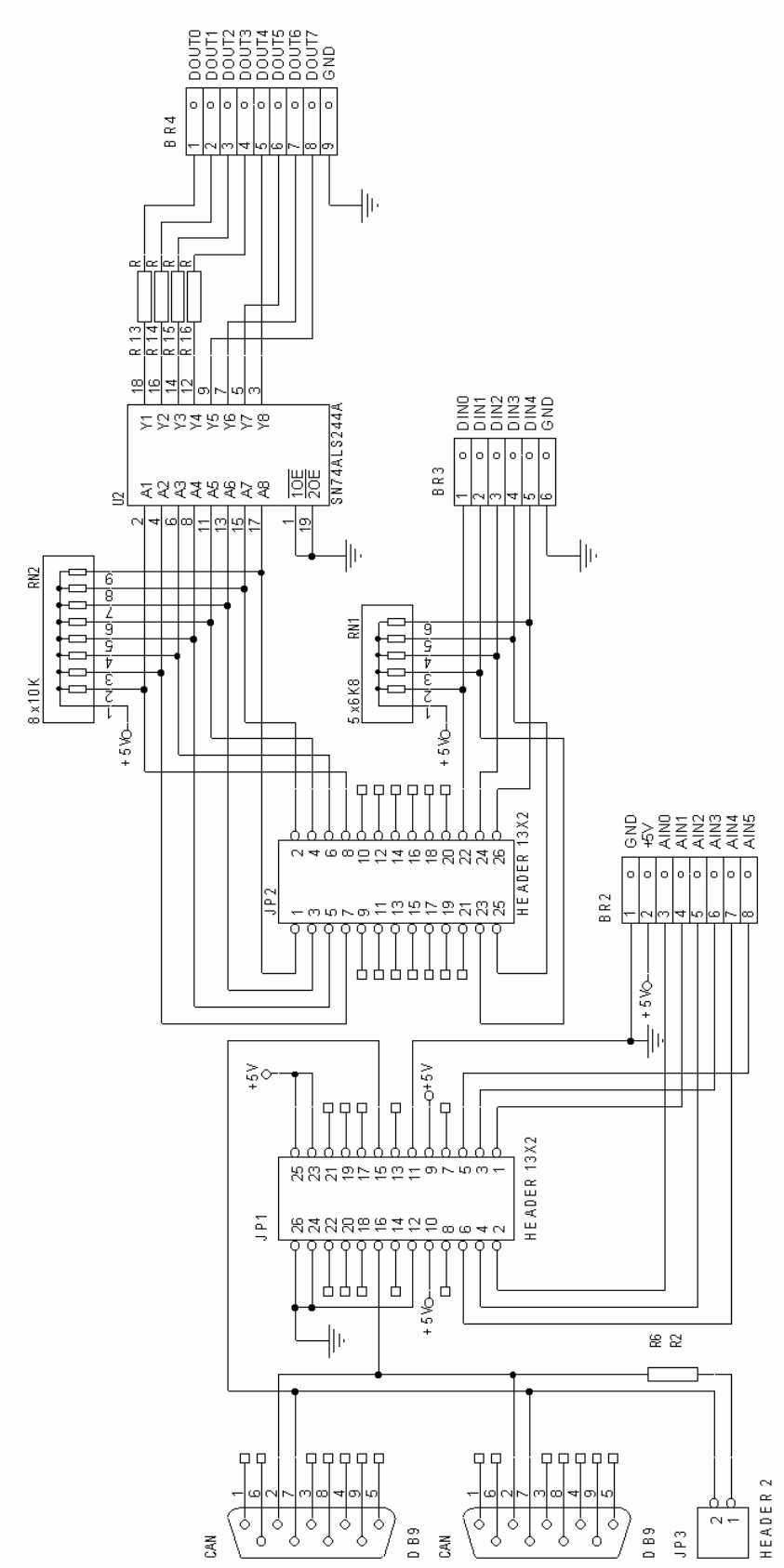
Status bit F_ID (field ID)

Čtením bitu F_ID (nultý bit na adrese 2000h) se zjistí jaké pole je právě snímáno. Tento bit změní stav po přečtení celého pole (sudého nebo lichého). Lze tak zastavit A/D převodník v okamžiku, kdy je ve video paměti celý snímek.

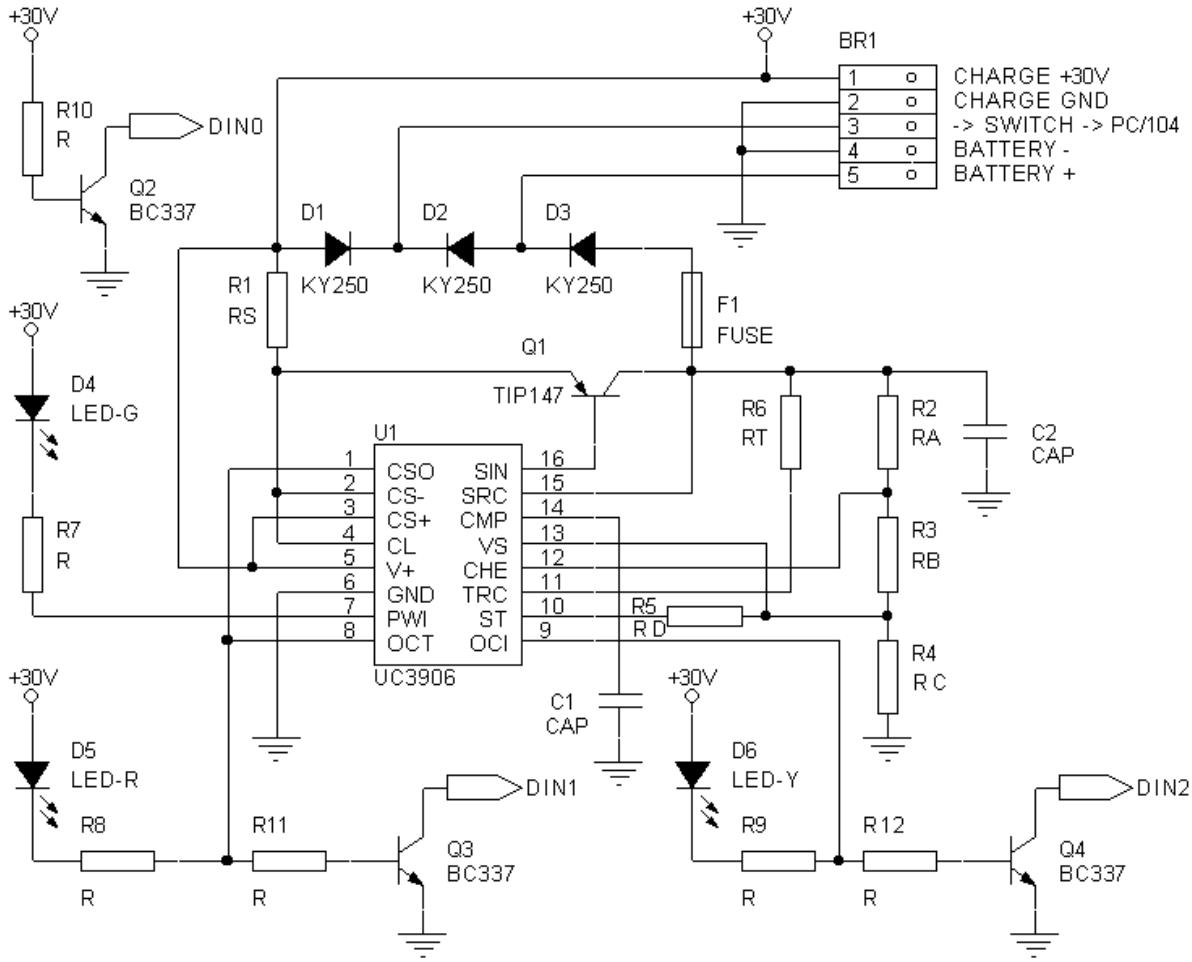
Pozn.: Existuje jiná metoda pro rychlejší čtení Video Buffer Memory. Paměť je duálně portovaná. To znamená, že CPU může číst paměť ve stejnou dobu, kdy A/D převodník zapisuje pixely do paměti. Programový kód by mohl číst jedno pole, zatímco je nové pole ukládáno do paměti A/D převodníku. Při použití této metody se musí monitorovat bit F_ID tak, že se bude přistupovat na právě opačné pole, než A/D převodník. Musí se číst od nejvrchnějšího rastrového řádku (0 nebo 1) a inkrementovat rastrové číslo o 2 po každém přečtení řádku.

Příloha 3 – Rozmístění součástek na PCB

Příloha 4 – Schéma obvodu s IO-modulem



Příloha 5 – Schéma nabíjecího obvodu



Příloha 6 – Výpočty hodnot rezistorů použitých v nabíjecím obvodu

$$I_D = 66.5mA$$
$$V_F = 13.8V \cdot 2$$
$$V_T = 12V \cdot 2$$
$$V_{OC} = 14.5V$$

$$I_{MAX} = 0.8A$$
$$R_C = \frac{2.3V}{I_D} = 34.6k\Omega$$
$$R_{SUM} = \frac{(V_F - 2.3V}{I_D} = 38k\Omega$$
$$R_D = \frac{2.3V \cdot R_{SUM}}{(V_{OC} - V_F)} = 625k\Omega$$
$$Rx = \frac{R_C \cdot R_D}{(R_C + R_D)} = 32.8k\Omega$$
$$R_A = (R_{SUM} + R_X) \cdot (1 - \frac{2.3V}{V_T}) = 373.6k\Omega$$
$$R_B = R_A - R_{SUM} = 6.8k\Omega$$
$$R_S = \frac{0.25V}{I_{MAX}} = 0.313\Omega$$

Příloha 7 - Obsah přiloženého CD

```
/**
 * struct vcasdo_fsm_t - structure representing SDO FSM
 * @srvcli_cob_id:  SDO server-client COB_ID (default is 0x580 + node),
port on which master listen
 * @clisrv_cob_id:  SDO client-server COB_ID (default is 0x600 + node),
port on which slave listen
 * @node:           CANopen node number
 * @index:          index of communicated object
 * @subindex:       subindex of communicated object
 * @is_server:      type of FSM client or server (Master or Slave)
(internal use)
 * @is_uploader:    processing upload/download in state sdofsmRun,
sdofsmDone
 * @state:          state of SDO (%sdofsmIdle = 0, %sdofsmRun, %sdofsmDone,
%sdofsmError, %sdofsmAbort)
 * @err_no:         error number in state %sdofsmError.
 * @data:           uploaded/downloaded bytes (see %ul_dbuff.h)
 * @out_msg:        if vcasdo_taste_msg() generates answer, it is stored in
the %out_msg
 * @statefnc:       pointer to the state function (internal use)
 * @last_activity:  time of last FSM activity (internal use)
 * @bytes_to_load:  number of stil not uploaded SDO data bytes (internal
use)
 * @toggle_bit:     (internal use)
 * @rx_cob_id:      COBID on which fsm listen
 * @tx_cob_id:      COBID on which fsm talk to CAN
 * Header: vcasdo_fsm.h
 */
typedef struct vcasdo_fsm_t {
    unsigned srvcli_cob_id;
    unsigned clisrv_cob_id;
    unsigned node;
    unsigned index, subindex;

    struct timeval last_activity;

    int bytes_to_load;
    unsigned char toggle_bit;

    char is_server; /* client/server */
    char is_uploader; /* upload/download */
    int state;
    vcasdo_fsm_state_fnc_t *statefnc;

    int err_no;

    ul_dbuff_t data;
    canmsg_t out_msg;
} vcasdo_fsm_t;
```