České vysoké učení technické
Fakulta elektrotechnická
Katedra řídicí techniky

Řízení motorů s deskou Raspberry Pi a Linuxem

Bakalářská práce

Autor: Radek Mečiar

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

2014

Vedoucí práce: Ing. Píša Pavel, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Radek Mečiar**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Řízení motorů s deskou Raspberry Pi a Linuxem**

Pokyny pro vypracování:

1. Seznamte se s hardwarem Raspbery Pi (Low cost ARM board) a s rt-preempt rozšířením jádra Linuxu.
2. Navrhněte jak k Raspberry Pi připojit stejnosměrný motor a vyrobte potřebnou redukci.
3. Implementujte software pro řízeni motoru pomocí rt-preempt Linuxu a vytvořte vizualizaci řídicího procesu včetně demo režimu.
4. Vše důkladně otestujte a zdokumentujte.

Seznam odborné literatury:

[1] https://rt.wiki.kernel.org/index.php/Main_Page
[2] https://rtime.felk.cvut.cz/psr/cviceni/semestralka/slides-motor.pdf
[3] https://rtime.felk.cvut.cz/gitweb/shark/motorek-5200.git/blob/HEAD:/motorek.c
[4] https://rtime.felk.cvut.cz/psr/cviceni/semestralka/

Vedoucí: Ing. Pavel Píša, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 8. 1. 2014

PROHLÁŠENÍ

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

V Praze dne 21.5.2014 Podpis: .....

PODĚKOVÁNÍ

Zde bych chtěl poděkovat mému vedoucímu práce Ing. Pavlu Píšovi, PhD za cenné rady z oblasti Linuxu a softwaru, také za věnovaný čas.

ABSTRAKT

Práce ověřuje možnost využítí levné procesorové desky Raspberry Pi pro realizaci real-time řídicího systému – konkrétně řízení servopohonu. V první části práce jsou popsané potřebné úpravy jádra operačního systému Linux, které zajistí spolehlivé časově omezené odezvy systému na události. Dále popisuje návrh a implementace ovladačů a softwaru pro určení polohy motoru a jeho zpětnovazební řízení. K motoru použitému jako objektu řízení byla navržena propojovací periferie s Raspberry Pi. Inkrementální snímač polohy použitého motoru při plné rychlosti otáčení současně generuje signál o maximální frekvenci kolem 28 kHz. Daná platforma byla otestována pro snímání stavu motoru s touto frekvencí. Do polovičních otáček motoru (tj. 2100 ot/min) bylo ověřeno, že jsou započítané všechny impulzy. Od vyšších frekvencí nebo s náročným zatížením systému již docházelo k výpadkům v počítání impulzů. Raspberry Pi je ale určitě použitelné v mnoha případech řízení a může poskytnout daleko větší možnosti než jednoúčelové vestavěné řídicí obvody.

ABSTRACT

This thesis examines the possibility to use cheap processor board Raspberry Pi for realization real-time control system – namely the control of the DC motor. The first part describes the required modifications to the Linux operating system kernel, which ensures reliable time limited system response to external and time events. It also describes the design and implementation of drivers and software to determine the position of the motor and feedback control. The DC motor, which was used as controlled object, is connected to Raspberry Pi with designed interface. Incremental encoder of used motor generates signal of maximum frequency about 28 kHz at full nominal voltage applied to the motor. Raspberry Pi was tested to process the incremental encoder sensor output up to this frequency. The platform is able to capture all signals while the speed is less than half of maximum speed (i.e. 2100 rev/min), but it isn't able to capture correct information when the speed is faster than half of maximum speed or the system is loaded by other tasks. But Raspberry Pi is certainly useful in many cases of control and can provide more options than single-purpose embedded control circuits.

Klíčové slova

Raspberry Pi, motor, řízení, real-time, linux

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

1. ÚVOD

Snaha o počítačovou automatizaci velkého množství i běžných činností, od zařízení v domácnosti, přes hračky až po reportážní quadroptéry a vojenské drony, je charakteristickým znakem a často i hnací silou dnešní technologicky založené společnosti. Důležitou úlohou návrhářů je pak zvolit vhodný řídicí systém. Nejjednodušší aplikace lze realizovat i s využitím samotného analogového řídicího obvodu, ale se stále se snižující cenou procesorových systémů a digitálních komponent jsou postupně i jednoduché úlohy řešeny touto technikou, která nabízí větší uživatelský komfort, snadnější přizpůsobení bez nutnosti měnit zapojení, mnohem jednodušší řízení na dálku a zapojení do větších celků. Pokud je ale na místě finanční otázka, je potřeba vybrat vhodný prostředek pro řízení, který splní požadovanou funkčnost a zároveň je ekonomicky (nej)výhodnější. Takové řešení pak často není univerzální a nabízí méně možností. Například, když zvolíme pro řešení konkrétního problému, podobu jednoduchého obvodu s mikroprocesorem a přesným počtem zpětných vazeb ze senzorů přes jistá rozhraní, tak v budoucnu tato rozhraní jen těžce rozšíříme a především bude složitější itegrace do většího celku. Často je však možné na trhu nalézt výkonnější systém, který je vyráběný ve velkých počtech a tak i přes nadbytečný výkon vychází levněji.

V nabídce je celá řada velmi malých, cenově přijatelných počítačů, realizovaných na jednom plošném spoji, které mají integrované rozhraní poskytující komunikační standardy, jak hardwarové, tak softwarové. Jedním z těchto systémů je i Raspberry Pi (dále RPi) vyvíjené britskou nadací *Raspberry Pi Foundation*, původně především pro výuku a rozvojové země, kde může být platforma využita jako levný domácí počítač, či rozšíření televizoru na multifunkční zařízení. Cena je srovnatelná s cenou zakázkově vyráběné jednoúčelové desky plošných spojů, ale může nabídnout více možností s řádově vyšším výpočetním výkonem a s přítomností obecného operačního systému (nabízí webové rozhraní, komunikace, vzdálenou správu, instalaci SW atd.) a proti profesionálním výkonným obvodům často mnohem nižší cenu, především díky masové výrobě a použití levných komponent.

Široké způsoby využití nabízí RPi především díky operačnímu systému Linux. S pochopením přístupu Linuxu k hardwaru a principu fungování systému, bude poté jednodušší, tím samým řídicím členem, nebo jiným s použitím získaných znalostí, vyřešit i jinou podobnou řídicí úlohu. Vyšší kvalitu řízení zajišťuje úprava jádra operačního systému, která přidává podporu pro real-time plánování a prioritní přednosti pro časově kritické úlohy. Úprava usnadňuje použití operačního systému Linux, pro který jsou takovéto úpravy již k dispozici.

Účelem práce je otestovat, jestli je platforma RPi pro danou úlohu použitelná a je možné chybějící profesionální periferie pro polohové nebo rychlostní řízení nahradit softwarem a využitím relativně velkého výpočetního výkonu centrální procesorové jednotky. Jako každé řešení, i toto má své limitní možnosti funkcionality a ne jen z hlediska periferií, ale také z hlediska rychlosti výpočtů, načítání, ukládání a komu-

nikace. S realizací dané úlohy řízení servopohonu se je pokusím otestovat.

1.1 Motivace

Práce je velice praktického charakteru. Jedná se spíše o vyzkoušení jestli a jak dobře je RPi použitelné ke zpětnovazebnímu řízení motoru. Očekávání nejsou příliš vysoká, protože RPi není výkonný hardware. Naopak je však velmi rozšířený a řešení může být zajímavé pro výukové účely a použití komunitou.

1.2 Způsob řešení

Práci jsem řešil přesně po bodech zadání. První bod je rozdělen na dvě kapitoly 2. *Raspberry Pi* a 3. *Linux a real-time rozšíření*. Druhý bod zadání obsahuje kapitola 4. *Stejnosměrný motorek*. Řídicí software je popsán v kapitole 5. *Řídicí software*, jako třetí bod zadání. A konečné otestování je k nalezení v kapitole 6. *Ověření a možnosti RPi*.

1.3 Zdroje pro práci

Při analýze požadavků na práci se nepodařilo dohledat, že by se někdo jiný využitím RPi v aplikaci zpětnovazebného řízení motoru zabýval. Existuje mnoho zdrojů, jak v knižních, tak elektronických podobách, ale žádný mnou nalezený, neobsahuje řešení obdobné úlohy. Obvykle je řízení motorů s RPi řešeno způsobem otevřené smyčky, tedy zabývají se pouze nastavením PWM, tak aby se motor otáčel pouze na určité procento výkonu.

Nalézt se dá mnoho SW řešení, pro různá rozšíření, jako například *PiBlaster* zmíněný v mé práci (viz. zdroj [25]). A velice rozšířené je *WiringPi*, které umí zprovoznit mnoho funkcí. Uvažoval jsem, že bych toto připravené řešení použil v práci, ale nakonec jsem od toho upustil. Potřeboval jsem zprovoznit pouze pár funkcí a seznamovat se s rozsáhlým řešením, abych pochopil funkcionalitu, bylo zbytečné. Rád bych jej ale doporučil, jako zajímavý zdroj [19] naprogramovaný v jazyce C.

Obdobná práce, ale existuje. Je to práce vyučovaná v předmětu *Programování systému reálného času*, který mám v doporučených zdrojích (viz. [18]). Používá stejný motor pro řízení, ale jiný řídící prvek než RPi.

V mém, poněkud stručném, seznamu zdrojů se nachází čtyři odkazy. Na výše zmíněný předmět a materiály použité k tomuto předmětu a na informace ohledně Linuxu, a *real-time* rozšíření (viz. zdroj [13]).

Potřebné informace jsem pak především hledal v dostupných knihách o programování a systému GNU/Linux (viz. zdroje [1], [2] a [3]). Z nich jsem čerpal hlavně při tvoření soketové komunikace, ale obsahují i mnoho jiných informací. Dále jsem využíval internetové zdroje.

1.4 Předpoklady

Jako operační systém k řešení práce je používán pouze Linux, tedy všechny postupy v této práci jsou určeny právě pro tento systém. I když jsem se snažil vytvořit návody pro zopakování postupů co nejjednodušeji na pochopení, určité základní znalosti práce s tímto systémem jsou v popisech předpokládány. Na přiloženém DVD (příloha A) se nacházejí zdrojové kódy použitého systému i s příslušnou verzí rozšíření. Celý obraz systému se v příloze nenachází. Ke stažení je na internetovém zdroji [21] a v textu dále je popsáno, jak potřebné komponenty doinstalovat.

Práce byla realizována s využitím dvou programovacích jazyků. Základ řízení byl naprogramován v jazyce C a jazyk Java byl použit při implementaci vzdáleného ovládacího panelu, protože nabízí objektový přístup, který se s výhodou použije při tvorbě grafické aplikace a usnadňuje zpracovaní přijímaných dat. Software je navržen pro vzdálené síťové ovládání, ale již neřeší sofistikované navázání komunikace apod. Zdrojové kódy všech implementovaných programů a komponent jsou umístěny na DVD (příloha A), opatřeny komentářem a v práci je vysvětlen účel jednotlivých programů.

Pro návody jsou použity příkazy z terminálového interpreteru příkazů, které odděluje následující rámeček. Konkrétně byl použit interpreter *Bash* (Bourne Again Shell).

```
Bash
```

2. RASPBERRY PI

2.1 Popis

RPi je kompletní miniaturní počítač nenáročný na pořizovací náklady. Není příliš výkonný, ale jeho rozměry přibližně odpovídají kreditní kartě (viz. tabulka 2.1). I přes malou velikost obsahuje veškeré komponenty odpovídající klasickému stolnímu počítači. Úložiště (pevný disk) je však nahrazeno paměťovou kartou SD/MMC.

Verze	**Šířka [mm]**	**Hloubka [mm]**	**Výška [mm]**
Model A	64.3	92.5	18.3
Model B	64.3	92.5	21.3

Tabulka 2.1: Rozměr Raspeberry Pi zahrnující i konektory.

Existují dvě varianty tohoto počítače, modely A a B. Model B prošel ještě vylepšením a je rozdělen na model B revize 1.0 a novější verze model B revize 2.0. Rozdíly jednotlivých verzí ukazuje tabulka 2.2.

Verze	**Model A**	**Model B rev. 1**	**Model a rev. 2**
RAM	256 MB	256 MB	512 MB
USB	1x	2x	2x
Ethernet	Ne	Ano	Ano
Spotřeba	2.5 W	-	3.5 W
Cena[1]	800 Kč	-	1200 Kč

Tabulka 2.2: Přehled rozdílů jednotlivých verzí a modelů.

Všechny modely a revize jsou založené na stejném integrovaném procesorovém obvodu ARM, který je taktovaný na frekvenci 700 MHz s možností přetaktování až na 1 GHz a s grafickým akcelerátorem označeným jako VideoCore IV.

Model B revize 2.0 je ještě vylepšen o montážní díry pro přichycení a má o čtyři piny GPIO více.

Dále pak každá verze RPi obsahuje tyto vstupně výstupní konektory:

- USB a napájecí MicroUSB
- GPIO
- Slot SD/MMC karty
- HDMI
- Kompozitní Video
- Jack 3.5 mm

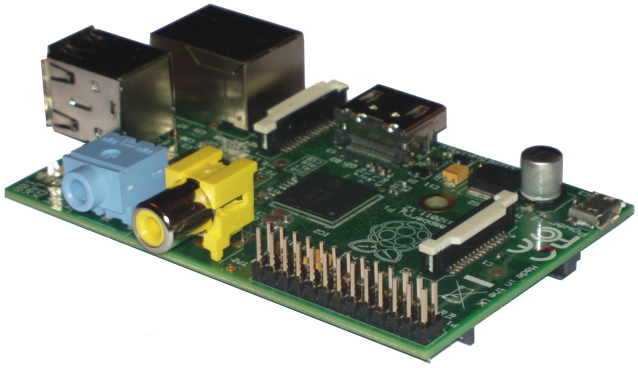
[1] Přibližná cena v lednu 2014 v kamenných obchodech v Praze.

Spotřeba modelu B revize 1.0 nebyla zjištěna a v současné době se již nevyrábí. Zjištěné spotřeby jsou uváděné prodejci RPi. Pokusil jsem se orientačně změřit spotřebu modelu B revize 2.0, který mám k dispozici, za použití běžného levného multimetru, kde byla sledována hodnota proudu. Raspberry Pi je napájené z 5 V stejnosměrného adaptéru, který je schopný dodávat až 1,3 A. Výsledek měření je vidět v tabulce 2.3.

Činnost	**Vypnuto**	**Zapnuto**	**Zatíženo**
Proud [A]	0,15	0,42	0,5
Vypočtený výkon [W]	0,75	2,1	2,5

Tabulka 2.3: Výkon Raspberry Pi model B.

K řešení práce byl použit model B revize 2.0. Fotografie je na obrázku 2.1. Tato verze je energeticky náročnější, ale zase má možnost připojení do ethernetových sítí a 512 MB paměti RAM.



Obrázek 2.1: Raspberry Pi model B rev. 2.0

Více informací o RPi je uvedeno na internetovém zdroji [11] nebo v knihách [4] a [5], ve kterých je obsaženo i množství návodů týkajících se nastavení systému a příkladů jak RPi použít. Obvykle jsou příklady napsané v jazyce *Python*.

2.1.1 Potřebné komponenty

RPi není ihned po koupi připraveno k použití. Jednak se k němu nedodává paměťová karta, ale také není přibalen potřebný adaptér. Minimální velikost paměťové karty je doporučena na 8 GB. Jako zdroj energie se doporučuje adaptér s napětím 5 V, který může dodávat až 700 mA. RPi odebírá přibližně 500 mA, takže pokud je nutné připojit energeticky náročnější periferie, mohou nastat problémy. S adaptérem schopným dodávat větší množství energie, pak mohou nastat problémy při zapojování periferií v zapnutém stavu.

Důvod, proč nemá RPi v balení tyto dvě základní komponenty zahrnuty je primárně cena a ekologické hledisko zbytečné výroby. Využije se tím starší příslušenství, které je dostupné, nebo je možnost objednání nového.

Mimo jiné bude určitě potřeba dalšího počítače, pokud k RPi není objednaná již připravená paměťová karta, bude potřeba vytvořit spustitelný operační systém za pomocí jiného počítače. Návody jsou dostupné na internetu (viz. zdroje [6] a [9]) a pro operační systém Linux je popsán v sekci 2.2.1 Instalace v Linuxu.

Alespoň k prvnímu spuštění je potřeba monitor připojitelný do HDMI konektoru, USB myš a klávesnice. Později může být k RPi přistupováno vzdáleně po síti například pomocí protokolu SSH.

2.2 Operační systém

Samotné RPi se nedodává s jedním předinstalovaným operačním systémem. Ten je však lehce dostupný v sekci *Download* na stránkách `www.raspberrypi.org`, kde je výběr z několika druhů určených přímo pro RPi. *Raspbian*, který je odvozený z distribuce *Debian*. Pokud máte v oblibě distribuci *Fedora*, tak její odvozená verze má název *Pidora*. Dále je také k dispozici *Arch Linux*, či různé další, často specializované, distribuce operačního systému. Například jako *OpenElec* a *Raspbmc*, které udělají z RPi mediální centrum, připojitelné k televizi. V nabídce je i nelinuxový systém *Risc OS*.

Balíček pro začátečníky nazvaný anglicky *NOOBS* obsahuje možnosti nainstalovat jakýkoliv výše zmíněný operační systém i jiné. Pro zavedení na paměťovou kartu je připraven návod, jak pro uživatele Linuxu, tak pro uživatele Windows, či Mac OS. Podle tohoto návodu se dá postupovat i při instalaci pouze jednoho vybraného systému. Linuxová verze návodu je v další sekci 2.2.1 Instalace v Linuxu.

Vyzkoušel jsem všechny operační systémy dostupné z připraveného instalátoru *NOOBS*. Nejvíce jsem považoval za vhodný *Raspbian*, protože je pravděpodobně nejrozšířenější a s distribucemi odvozenými z projektu *Debian* mám zkušenosti. Nejnovější verze *Raspbianu* je ke stažení z internetového zdroje [21], tento zdroj obsahuje již pouze systém *Raspbian*. Bližší informace o tomto systému jsou k dispozici na internetovém zdroji [20].

2.2.1 Instalace v Linuxu

Kroky návodu jsou čerpány ze zdroje [9].

K započetí instalace je potřeba mít stažený image soubor, pokud možno s poslední stabilní verzí systému. Ta je ke stažení na odkazu ve zdroji [21]. Kde se nachází archiv, konkrétně *zip*, který obsahuje soubor typu *image*. Soubor s archivem se za pomocí linuxového terminálu rozbalí příkazem:

```
cd /adresar/obsahujici/soubor/
unzip 2014-01-07-wheezy-raspbian.zip
```

Kde za příkazem `unzip` následuje jméno konkrétního souboru. Nyní se ve složce nachází nově i soubor typu *image*, například s názvem „*2014-01-07-wheezy-raspbian.img*“. Tento soubor je potřeba nakopírovat pomocí programu `dd` na paměťovou kartu. Zjištění názvu paměťové karty, který je potřeba znát pro program `dd`, se

provede následujícím příkazem:

```
sudo fdisk -l | grep /dev/
```

Tento příkaz vytiskne výpis na monitor, ve kterém by se měla nacházet i paměťová karta, pokud je připojena k počítači. Příklad výpisu:

```
...
Disk /dev/mmcblk0: 31.6 GB, 31611420672 bytes
/dev/mmcblk0p1            2048    61741055    30869504   83
   Linux
...
```

Ve výpisu může být více položek, karta se nejlépe pozná podle velikosti, která je zde vidět jako 31.6 GB. Paměť obsahuje jeden oddíl a to „*mmcblk0p1*“. Jméno, které bude potřeba pro kopírování souboru je „*/dev/mmcblk0*“. Pro práci s programem `dd` musí být paměťová karta odpojena a to se provede v tomto případě příkazem:

```
sudo umount /dev/mmcblk0*
```

Příkaz buďto odpojí všechny oddíly paměťové karty, nebo vypíše, že nejsou připojeny. Poté můžeme pokračovat v samotném kopírování souboru typu image, **zkontrolujte si, zda máte správný název zařízení, následujícím příkazem můžete ztratit data**:

```
sudo dd if=2014-01-07-wheezy-raspbian.img of=/dev/mmcblk0
```

Poté co se příkaz úspěšně dokončí, je paměťová karta připravena k použití s RPi. Měla by obsahovat dva oddíly, které se v mém případě jmenují „boot“ a „boot_“. První oddíl „boot“ obsahuje soubory s jádrem a pomocnými soubory načítanými při startu systému. V souborové struktuře se pak napojí do adresáře „/boot“. Druhý oddíl obsahuje celý souborový systém, takzvaný kořen systému (adresář „/“, anglicky *root*).

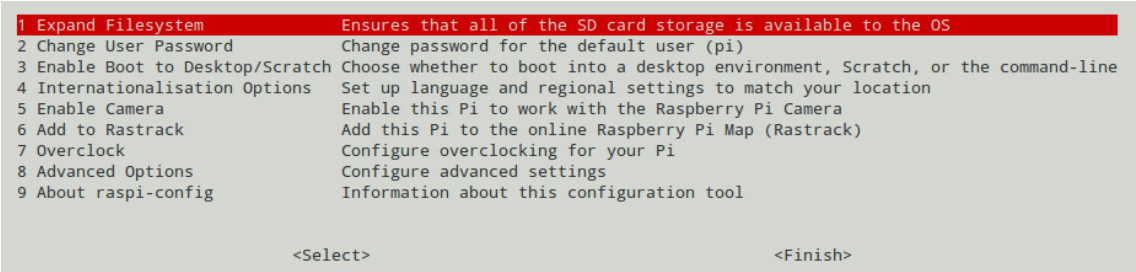
2.2.2 První spuštění

Při prvním spuštění operačního systému se načte textový režim s programem pro základní nastavení systému. Program má jednoduché rozhraní, které je vidět na obrázku 2.2 a jmenuje se *Raspberry Pi Software Configuration Tool*. Příkaz pro opětovné spuštění programu v budoucnu je:

```
sudo raspi-config
```

Z nabídky programu je vhodné použít volbu číslo jedna a to *Expand Filesystem*, která způsobí, že RPi využije celé místo na paměťové kartě.

Pokud je potřeba k RPi přistupovat vzdáleně přes protokol SSH, lze zde tento přístup povolit a to zvolením osmé volby *Advanced Options*, dále pak najít volbu *SSH*, která se v mém případě jmenuje *A4 SSH* a potvrzením *Enable* umožnit komunikaci prostřednictvím SSH protokolu. Volba není zmíněna náhodou. Ve zdroji o *real-time* rozšíření systému [13] je uvedeno, že klasická konzole vykreslovaná na monitor, může způsobovat zpoždění, až v řádech 100 μs, což je pro tuto úlohu nežádoucí



Obrázek 2.2: Rozhraní pro základní konfiguraci Raspberry Pi

a proto doporučuji připojovat se k RPi prostřednictvím tohoto komunikačního protokolu.

V nastavení je i jednoduchá možnost přetaktování procesoru, nebo určení využití paměti RAM pro grafický akcelerátor. Přetaktování nedoporučuji, protože RPi se i při základní frekvenci 700 MHz zahřívá. Možnost ubrat paměť grafickému akcelerátoru je rozumná, protože ji přes protokol SSH stejně RPi nevyužije. Není ovšem tak zásadní v případě této úlohy.

Po potvrzení *Finish* a restartu systému dojde na požadavek k přihlášení. Přednastavené uživatelské jméno je „*pi*“ a k tomuto účtu je heslo „*raspberry*“.

Jako návod, pro další nastavení RPi mohu doporučit knihy z mých zdrojů [4] a [5], kde se nachází například nastavení automatického připojení do Wi-Fi sítě apod.

3. LINUX A REAL-TIME ROZŠÍŘENÍ

Účelem *real-time* rozšíření v operačním systému Linux je změna plánovacích algoritmů, které se starají o přidělování procesorového času jednotlivým, aktuálně běžícím, procesům. V nerozšířeném systému možnost nadřadit proces jako vysoce prioritní také existuje, ale plánovací algoritmy nezaručují stropní hranici latence, kvůli delším úsekům, ve kterých je zablokované přeplánování úlohy nebo i přijetí přerušení. *Real-time* rozšíření vyžaduje složitější algoritmy plánování, takže propustnost a minimální doby odezvy jsou obvykle horší, naopak maximální latence jsou omezené.

Jádra Linuxu s možností vynucené změny pořadí procesů jsou označována jako „PREEMPT“ z anglického *preemption*, ve smyslu přerušení přeplánování. Rozšířená jádra o real-time algoritmy plánování jsou označována jako „PREEMPT_RT“ a odlišují se omezováním úseků s blokovaným přeplánováním a i zpracování vnějších událostí řeší v rámci vláken.

Systémy vrací nejmenší rozlišitelnou jednotku času 1 ns. Reálně je však možnost plánování spuštění procesu s přesností v řádech kolem 10 μs, jak bude ukázáno dále v této kapitole.

3.1 Postup úpravy

Tento postup navazuje na již hotovou instalaci Linuxu *Raspbian* podle návodu v sekci 2.2.1 Instalace v Linuxu a opět je určen pro operační systém Linux. Kroky z návodu jsou čerpány ze zdrojů [7] a [8].

3.1.1 Nahrání rozšíření

Jestliže chceme přidat do jádra systému Linux takto rozsáhlé rozšíření, musíme to provést před kompilací a následně jádro zkompilovat. K tomu, abychom to mohli provést, bude třeba zdrojového kódu jádra. Jádro systému Linux je šířeno pod licencí GPL, která zajišťuje volnou dostupnost zdrojových kódů i následnou možnost dalších úprav. Není tedy problém získat zdrojové kódy jádra. Existuje však celá řada přednastavení pro jádra a mnoho jeho verzí. Nalezením správného zdroje pro RPi si ušetříme čas, který bychom museli vynaložit na vhodnou konfiguraci a optimalizaci pro daný hardware. Jádro určené a správně nakonfigurované pro RPi se nachází například na internetové adrese ve zdroji [22]. Zdroj obsahuje vždy nejnovější verzi z verzovacího systému GIT serveru GitHub, aktuálně[1] s označením 3.10.26. Označení je důležité pro úpravu, která je určena pro konkrétní verze. Aby se dal postup zopakovat, pořídil jsem kopii této verze a umístil ji na DVD (příloha A). Soubor je pojmenovaný jako „linux-rpi-3.10.y.zip“ a obsahuje stabilní větev se zahrnutými aktuálními upravami verze 3.10.y.

[1] Leden 2014

Zdrojový kód ale nezahrnuje potřebné rozšíření, pouze vhodné konfigurace pro RPi. Rozšíření se teprve musí přidat. Zdroj, kde se dá nalézt je označen jako [23] v seznamu souborů na konci práce. Podle verze jádra je potřeba nalézt správnou verzi rozšíření. K verzi 3.10.26, obsažené v příloze na DVD, je to soubor s názvem „patch-3.10.26-rt24.patch.gz“.

Máme tedy dva soubory typu archiv. Nejprve je musíme rozbalit, poté včlenit vylepšení do zdrojového kódu jádra. O úpravy se postará následující sada příkazů terminálu, pokud umístíme soubory do jednoho adresáře. Pro jiné verze, musíte upravit názvy souborů.

```
cd /adresar/obsahujici/soubory
unzip linux-rpi-3.10.y.zip
gunzip patch-3.10.26-rt24.patch.gz
cd linux-rpi-3.10.y
patch p1 < ../patch-3.10.26-rt24.patch
```

3.1.2 Kompilace jádra

Po úspěšném provedení příkazů budeme mít připravený zdrojový kód jádra, konkrétně ve složce „linux-rpi-3.10.y“, obsahující *real-time* vylepšení. Dále je potřeba kód zkompilovat. Máme na výběr více možností, jak kompilaci provést. Můžeme kompilovat přímo na RPi, avšak já bych tuto volbu nedoporučil, protože výkon je nízký a kompilace zabere mnoho hodin. Druhá možnost je využít takzvané *cross* kompilace. Je to druh kompilace, při které se převádí program do strojového kódu na jiném počítači. Pokud mají oba stroje stejnou architekturu procesoru, tak je možno použít přímo kompilátor z jiného počítače. Jestliže nemají, je potřeba sehnat nástroj, který bude schopen připravit strojový kód pro jinou architekturu procesorů. V RPi se nachází procesor s architekturou ARM. *Cross* kompilační nástroj pro tento typ procesoru je na adrese ve zdroji [24]. Opět je dostupný na přiloženém DVD (příloha A) a je pojmenován jako „tools-master.zip“.

Soubor obsahující nástroj pro kompilaci je typu archiv a rozbalíme ho následujícím příkazem. Návod i nadále předpokládá použití stejného pracovního adresáře jako v případě předešlých souborů.

```
cd /adresar/obsahujici/soubory
unzip tools-master.zip
```

Po úspěšném dokončení příkazu je vše připraveno k zahájení samotného procesu kompilace. Složka obsahuje nově složku „tools-master“, v které se nachází kompilační nástroj. Tedy podstatné dva adresáře, jež by se měly nacházet v pracovním adresáři jsou „linux-rpi-3.10.y“ a „tools-master“, případně se můžou lišit podle verze jádra.

Prvním krokem kompilace, pokud provádíme opětovné pokusy, je vyčištění předešlých pokusů. K tomu je určen následující terminálový příkaz, který se musí spustit ve složce, kde je obsažen kód jádra, v tomto případě se jedná o adresář „linux-rpi-3.10.y“.

```
cd /adresar/obsahujici/soubory/linux-rpi-3.10.y
make mrproper
```

Po vyčištění předchozích pokusů o kompilaci, je třeba obnovit konfigurační soubor, v kterém je obsaženo nastavení pro danou kompilaci. Originální soubor s konfigurací pro RPi se dá najít v již běžícím systému ve složce „/proc“ a název souboru je „config.gz“. Pomocí příkazu `zcat` můžeme soubor uložit do textové podoby a pak následně přenést do pracovního adresáře, konkrétně do složky, která obsahuje zdrojový kód Linuxu „linux-rpi-3.10.y“ a soubor pojmenujeme jako „.config“.

Máme-li vhodnou konfiguraci, například z již běžícího systému, kterou chceme zachovat a pouze nastavit dosud nezvolené a nové možnosti, použijeme následující příkaz.

```
make oldconfig
```

Příkaz vyvolá jednoduchého průvodce, v kterém nastavíme pouze nové volby a staré zůstanou nastavené jak byly. Pokud souhlasíme s přednastavenou volbou, která je vyznačena velkým písmenem či šipkou, stačí pokračovat dále klávesou *Enter*. Ukázka výpisu:

```
...
Enable DMI scanning (DMI) [Y/n/?] y
GART IOMMU support (GART_IOMMU) [Y/n/?] y
IBM Calgary IOMMU support (CALGARY_IOMMU) [N/y/?] n
Preemption Model
> 1. No Forced Preemption (Server) (PREEMPT_NONE)
  2. Voluntary Kernel Preemption (Desktop) (PREEMPT_VOLUNTARY)
  3. Preemptible Kernel (Low-Latency Desktop) (PREEMPT__LL) (NEW
     )
  4. Preemptible Kernel (Basic RT) (PREEMPT_RTB) (NEW)
  5. Fully Preemptible Kernel (RT) (PREEMPT_RT_FULL) (NEW)
choice[1-5]: 5
...
```

Výpis obsahuje důležitou část. Po nahrání rozšíření do zdrojového kódu jádra se rozšířily možnosti výběru „Preemption Model“, konkrétně o volbu 4 a 5. Pro tuto úlohu je použita volba 5 „Fully Preemptible Kernel (RT) (PREEMPT_RT_FULL) (NEW)“.

Druhá možnost konfigurace je pomocí následujícího příkazu. Rozdíl oproti předchozímu způsobu konfigurace je takový, že se nezeptá na vše dosud nenastavené, ale musíte si najít konkrétní volbu, kterou chcete nastavit.

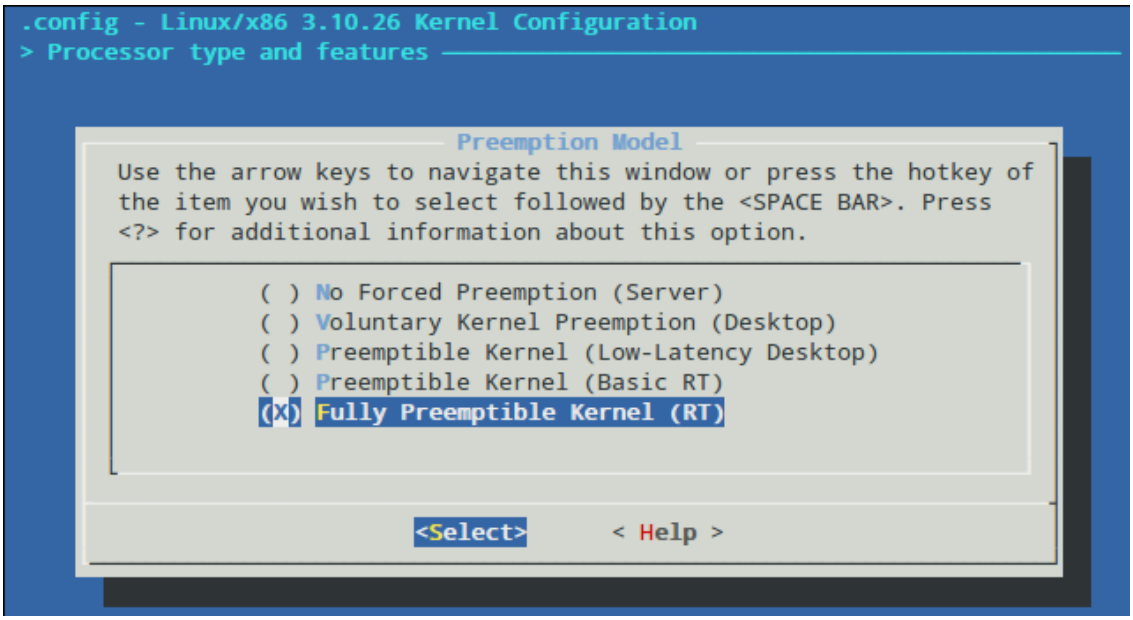
```
make menuconfig
```

Vyvolá jednoduché grafické prostředí jako je na obrázku 3.1, který ukazuje opět důležitou volbu s nastavením „Preemption Model“. Volba je umístěna pod volbou „Processor type and features —>“ a v ní je možnost „Preemption Model (Fully Preemptible Kernel (RT)) —>“.

Poslední konfigurace před spuštěním kompilace, se týká proměnných terminálového prostředí. Musíme zvolit vhodnou architekturu a uložit ji do proměnné s ná-

zvem „ARCH“. Další volba se týká kompilačního nástroje, který se nachází ve složce „tools-master“. Do proměnné „CROSS_COMPILER“ musíme uložit cestu použitého kompilačního nástroje. A poslední proměnná s názvem „INSTALL_MOD_PATH“ určuje umístnění pro nově zkompilované moduly. Pokud bychom vynechali tuto volbu, zavedli by jsme nově vytvořené moduly do aktuálně běžícího systému. K nastavení proměnného prostředí terminálu jsou určeny následující příkazy.

```
export ARCH=arm
export CROSS_COMPILE="/adresar/obsahujici/soubory/tools-master/
   arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin/arm-
   linux-gnueabihf-"
mkdir "/adresar/obsahujici/soubory/moduly"
export INSTALL_MOD_PATH="/adresar/obsahujici/soubory/moduly"
```



Obrázek 3.1: GUI pro konfiguraci jádra před kompilací.

Kompilace se opět musí spustit z adresáře obsahujícího zdrojový kód. První příkaz z další sady příkazů, se postará o přesun do správného umístění. Druhý příkaz spustí kompilaci *kernelu*, neboli jádra. Zbývající příkazy pustí kompilaci modulů a následné instalování, v našem případě nakopírování do umístění nastaveného v proměnné „INSTALL_MOD_PATH“.

```
cd /adresar/obsahujici/soubory/linux-rpi-3.10.y
make bzImage
make modules
make modules_install
```

Příkaz `make` je vhodné použít s parametrem `-j N`, kde N je počet běžících procesů, které provádí kompilaci najednou. Pokud použijeme pouze `-j`, tak počet procesů není omezen. V případě, že provádíme kompilaci na jedno jádrovém procesoru, tak tento parametr nemá až takový význam. Více informací obsahují manuálové stránky Linuxu (příkaz `man make`).

Kompilace se na běžném notebooku provede do jedné hodiny. RPi kompiluje mnohem déle. Kompilace běžela přes jeden den a poté mi systém zamrzl, takže nemám ani přibližný údaj, jak dlouho může trvat. Raději doporučuji *cross* kompilaci, docílíte velké úspory času.

3.1.3 Zavedení nového jádra

Pokud kompilace proběhne v pořádku až do konce, objeví se v pracovním adresáři na cestě „linux-rpi-3.10.y/arch/arm/boot“ soubor s názvem „zImage“. Dále také složka „moduly“ bude obsahovat adresář „lib“, kde jsou obsaženy moduly zkompilované pro nové jádro.

Nyní musíme zavést do již nainstalovaného systému jádro i s moduly. Překopírujeme soubor „zImage“ na paměťovou kartu na oddíl „boot“ přímo do kořenového adresáře oddílu. Doporučuji soubor přejmenovat na aktuální verzi jádra, například jako „zImage-3.10.26-rt24“. Moduly se skládají ze dvou adresářů „lib/firmware“ a „lib/-modules“. Obsah složky „modules“, která obsahuje podsložky pojmenované podle verze jádra, bude třeba překopírovat na druhý oddíl paměťové karty „boot_“ do umístnění „/lib/modules“. Názvy složek s moduly, by se neměly shodovat, protože za aktuálně zkompilovanou verzí bude přidáno „-rt“, pokud se však shodovat budou, doporučuji pro jistotu, jednoduché přejmenování stávajícího adresáře na jiné jméno. Složku „firmware“ není potřeba kopírovat.

Nakonec je potřeba změnit konfigurační soubory, které se nacházejí v oddílu „boot“. Jedná se o soubor „config.txt“, kde se musí změnit parametr „kernel=“ na název aktuálního souboru s jádrem. Můžete mít v systému více jader a pomocí změny tohoto parametru určovat, které jádro se načte. Doporučuji tedy všechny řádky začínající „kernel=“ v souboru zakomentovat pomocí „#“ a přidat řádek „kernel=zImage-3.10.26-rt24“. Poslední změnu provedeme v konfiguračním souboru „cmdline.txt“ a to přidáním parametru „sdhci_bcm2708.enable_llm=0“ na konec souboru, pokud používáme kartu typu SD. Bez toho parametru by se zaseklo načítání systému. V případě karty typu MMC přiřadíme parametru hodnotu 1. Systém je nyní připraven k použití s RPi.

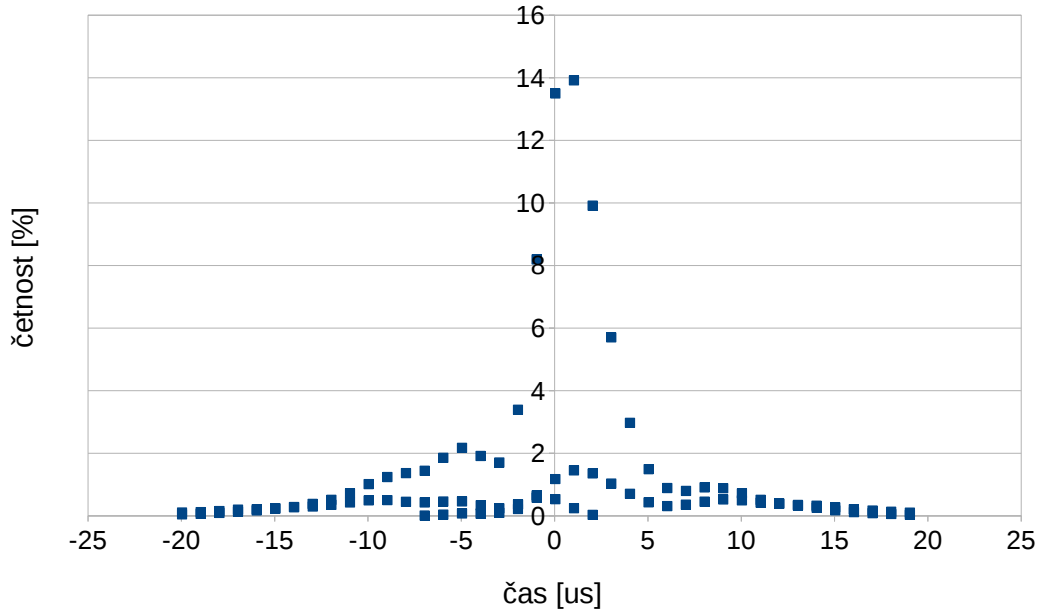
3.2 Test systému

3.2.1 Jednoduché vlákno

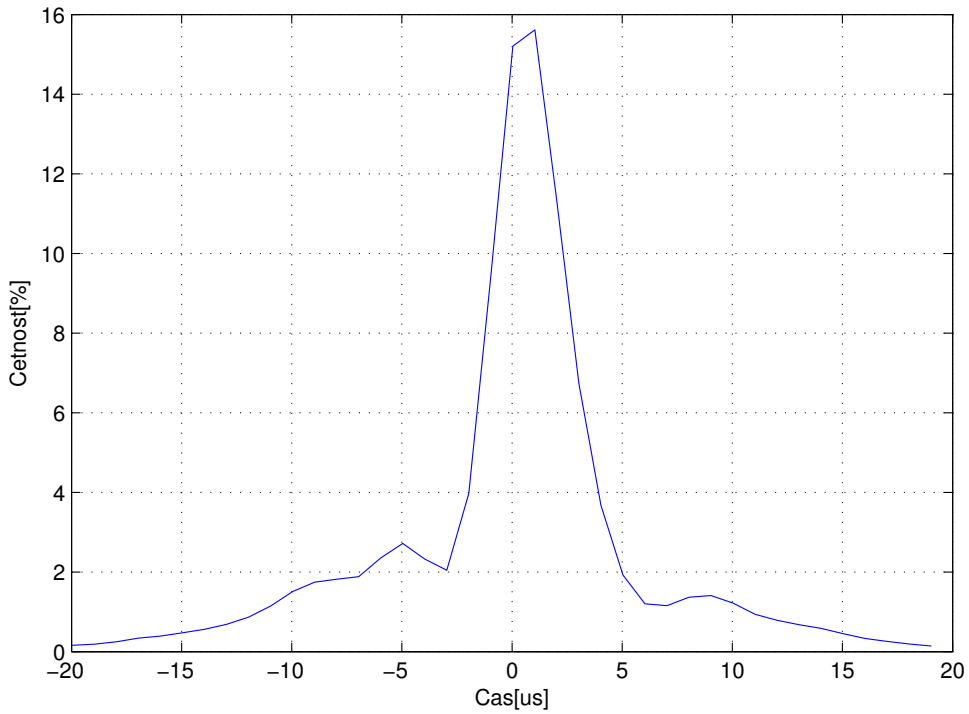
V systému jsem vytvořil vlákno, které běželo s vysokou prioritou (98) a politikou plánování FIFO. Bylo cyklicky volané s frekvencí 1 kHz. Vykonávalo pouze ukládání času do pole, pro následné vypsání a hlídání počtu opakování na 600 000, což je 10 minut. Po doběhnutí test vytvoří soubor „vystup.csv“, ve kterém je četnost jednotlivých nepřesností plánování. Výsledek je vidět v podobě grafu na obrázku 3.2, kde hodnota nuly odpovídá času 1 ms od předchozího spuštění vlákna. Z testu je vidět, že převážná většina vzorků vykazuje chybu kolem -2.5 μs až 5 μs. Zdrojový kód testu je v příloze A s názvem souboru „vlakno_test_planovani_1kHz.c“ ve složce „test“ a kompilace se provede následujícím příkazem.

```
gcc -lrt -lpthread vlakno_test_planovani_1kHz.c
```

Z grafu je patrné, že se jednotky času v systému pohybují kolem 1 μs a to i když systém vrací nejmenší jednotuku 1 ns. Body, které jsou na grafu vykresleny pod sebou, jsou zaznamenány s rozdílem 1 ns. Graf z obrázku 3.3 je vytvořen součtem hodnot četností v bodech s okolím do ± 1 ns a je vykreslen jako spojitá funkce.



Obrázek 3.2: Četnost chyb plánování procesů.



Obrázek 3.3: Seskupené četnosti chyb plánování procesů.

3.2.2 Cyclictest

Systém jsem otestoval s testem, který se jmenuje *Cyclictest*. Informace o testu jsou dostupné ve zdroji [14], kde je k dispozici zdrojový kód i s návodem, jak tento test použít. Výsledky testu jsou takové, že při zatížení systému různými způsoby najednou, se pohybovala přesnost plánování real-time procesu s vysokou prioritou v řádech kolem desítek μs.

K RPi jsem se připojil přes čtyři SSH terminály. Každý z nich zaměstnával systém jinou činností:

find / ..... výpis stromu souborů, pro zaměstnání vstupu a výstupu

while true; do true; done ..... nekonečná smyčka, pro zaměstnání procesoru

htop ..... sledování sytému

cyclictest ..... test prodlevy plánování

Test jsem nechal běžet po celou dobu výpisu stromu souboru systému. Konkrétně s těmito parametry se pustilo jedno vlákno s prioritou 99 a frekvencí 1 kHz. Výsledek byl následující výpis:

```
sudo cyclictest -t1 -p 99 -n -i 1000
 /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 2.75 1.66 0.74 2/89 2796

T: 0 ( 2789) P:99 I:1000 C:  90502 Min:     16 Act:   52 Avg:
   51 Max:     111
```

Výsledkem je, že maximální latence čekání byla 111 μs, průměrná prodleva 51 μs a minimální 16 μs, při stoprocentní zátěži procesoru a také zátěži vstupně výstupních periferií.

Vyzkoušel jsem spustit test i bez zatížení systému. Parametry nastavují test tak, aby spustil pět vláken s prioritou 80.

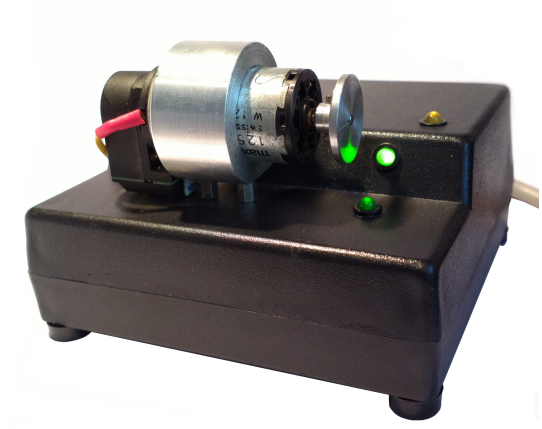
```
sudo cyclictest -t5 -p 80 -n -i 1000
policy: fifo: loadavg: 0.27 0.22 0.19 1/79 2351
T: 0 ( 2347) P:80 I:1000 C: 51489 Min: 13 Act: 44 Avg: 25 Max:
   85
T: 1 ( 2348) P:80 I:1500 C: 34327 Min: 16 Act: 21 Avg: 26 Max:
   93
T: 2 ( 2349) P:80 I:2000 C: 25748 Min: 15 Act: 38 Avg: 24 Max:
   62
T: 3 ( 2350) P:80 I:2500 C: 20598 Min: 16 Act: 19 Avg: 26 Max:
   79
T: 4 ( 2351) P:80 I:3000 C: 17165 Min: 16 Act: 21 Avg: 25 Max:
   100
```

Vlákna mají různé periody a to $T = 1000 + k \cdot 500$, kde $k \in \{0, 1, 2, 3, 4\}$. Z testu je vidět ze průměrná chyba načasování je přibližně poloviční a to 25 μs.

4. STEJNOSMĚRNÝ MOTOREK

4.1 Popis

Jako objekt pro řízení jsem použil stejnosměrný motor, se kterým se vyučuje podobná úloha řízení (více informací ve zdroji [18]). Motor je připraven k řízení se svojí vnitřní elektronikou a vše je zahrnuto v jedné krabičce. Řešení je vidět na obrázku 4.1a. Originální dokumentace a informace k tomuto řešení jsou dostupné ve zdrojích [15], [16] a [17]. Schéma motoru je obsaženo v příloze C na konci práce.

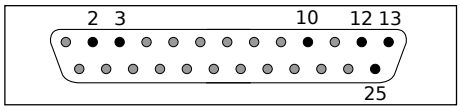


(a) Motor

(b) Rozhraní *CANON 25*

Obrázek 4.1: Použitý motor.

K motoru vedou dva kabely. Jeden je napájecí a druhý slouží jako vstup pro ovládání a výstup zpětné vazby. Energii dodává 24 V zdroj napětí a napájecí konektor je rozdělen na dva klasické tzv. banánkové konektory. Kabel pro přivedení řídicích signálů a výstup zpětné vazby je zakončený konektorem typu *CANON 25*, který je vidět na obrázku 4.1b. Toto rozhraní obsahuje 25 pinů, funkčních je však jen 6. Schématické rozhraní je vidět na obrázku 4.2 a popis funkcí jednotlivých pinů je uveden v tabulce 4.1.

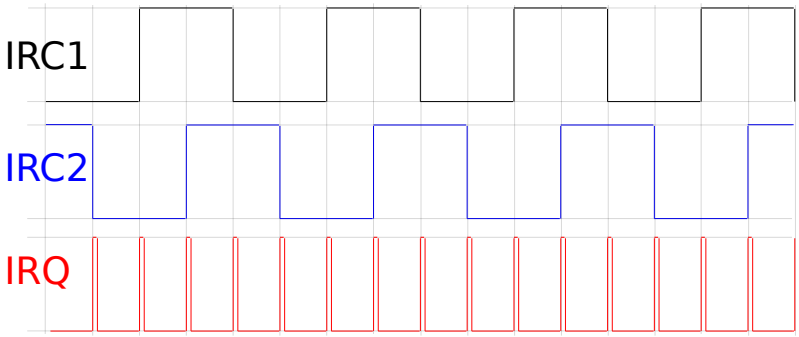


Obrázek 4.2: Schéma rozhraní.

Pin	Funkce	Označení
2	Otáčení doprava	PWM-R
3	Otáčení doleva	PWM-L
10	Přerušení	IRQ
12	IRC první kanál	IRC1
13	IRC druhý kanál	IRC2
25	Uzemnění	GND

Tabulka 4.1: Funkce pinů konektoru motoru.

Elektronika motoru obsahuje výkonový H můstek, který umožňuje řídit směr otáčení, tj. přivést na motor napětí o obou polaritách. Pro zpětnou vazbu je na hřídel motoru připojený dvoukanálový rotační inkrementální senzor (IRC). Druhý kanál je posunut o 1/4 délky periody oproti prvnímu. Na jedno otočení hřídele připadá 100 pulzů od každého kanálu. Elektronika motoru obsahuje obvod generující impulz přerušení (IRQ) při každé změně kteréhokoliv z obou kanálů, tedy 400 impulzů za celou otáčku hřídele. Obrázek 4.3 ukazuje průběh signálů IRC1, IRC2 a IRQ pro konstantní rychlost otáčení. Výstupní logika je 5 V. Maximální otáčky motoru se zdrojem 24 V jsou přibližně 4200 ot/min, tedy maximální frekvence impulzů IRQ kanálu se pohybuje kolem 28 kHz.



Obrázek 4.3: Graf průběhu signálu IRC a IRQ z motoru.

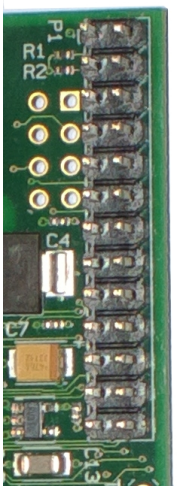
4.2 Propojení s Raspberry Pi

Logika RPi je 3.3 V bez tolerance pro 5 V logiku a RPi má pouze jeden výstup schopný generovat signál s pulzně šířkovou modulací (PWM). Z těchto důvodu je nutné navrhnout propojovací periferii mezi motorkem a RPi.

4.2.1 PWM a GPIO

Pro účel řízení servomotoru obsahuje RPi již hardwarovou podporu PWM, která nezatěžuje procesor samotným vytvářením signálu. Je třeba ji jen správně nastavit a použít. Dostupný je pouze jeden kanál schopný generovat PWM signál a to konkrétně na rozhraní GPIO. Vlastní čip obsahuje i další PWM periferie, ale ty nejsou přímo přístupné na vnějších konektrorech. Pro více motorů se dá pomocí řadiče DMA vytvořit další PWM, které opět nebudou zatěžovat procesor samotnou generací signálu, ale pouze jeho změnou. Na toto rozšíření je již připraveno několik hotových řešení. Jedno z nich se jmenuje *PiBlaster* (viz. zdroj [25]).

Rozhraní GPIO je na RPi realizováno několika konektory. Hlavní konektor označený jako P1 je již připraven k použití. U ostatních se musí připájet piny. Konektor P1 má vše, co je potřeba pro řízení servomotoru. Kompletní přehled rozhraní typu GPIO a alternativních funkcí RPi se nachází na internetové adrese ve zdroji [10].



Obrázek 4.4: Rozhraní GPIO konektor P1 na Raspberry Pi.

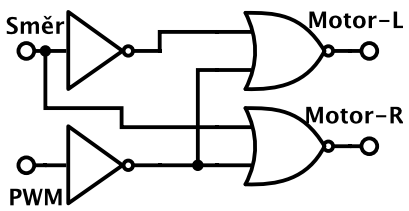
Funkce	HW označení		Funkce
3.3 V	P1-01	P1-02	5 V
GPIO 2	P1-03	P1-04	5 V
GPIO 3	P1-05	P1-06	GND
GPIO 4	P1-07	P1-08	GPIO 14
GND	P1-09	P1-10	GPIO 15
GPIO 17	P1-11	P1-12	GPIO 18
GPIO 27	P1-13	P1-14	GND
GPIO 22	P1-15	P1-16	GPIO 23
3.3 V	P1-17	P1-18	GPIO 24
GPIO 10	P1-19	P1-20	GND
GPIO 9	P1-21	P1-22	GPIO 25
GPIO 11	P1-23	P1-24	GPIO 8
GND	P1-25	P1-26	GPIO 7

Tabulka 4.2: Rozhraní GPIO, konektor P1 na Raspberry Pi.

Konektor GPIO P1 je plochý dvouřadý konektor (2x13 pinů) s roztečí 2,54 mm mezi jednotlivými piny a tloušťkou pinu 0,6 mm. Zobrazený je na obrázku 4.4. Tabulka 4.2 obsahuje jména pinů shodná s dokumentací [12].

Propojovací periferie by měla mít na jedné straně tento typ konektoru, který je na obrázku 4.7a, pro připojení k RPi. Na straně druhé konektor *CANON 25* zobrazený na obrázku 4.7c pro připojení k motoru. Dále musí zajistit ovládání s jednou PWM na oba směry a převod napětí z 5 V na 3.3 V.

Řízení v obou směrech jsem realizoval využitím dalšího GPIO pinu, který slouží k volbě směru otáčení. Pro převod logických úrovní jsem použil sériový odporový dělič. Výběr vhodného logického členu byl konzultován s vedoucím práce a schéma řešení ukazuje obrázek 4.6.



Obrázek 4.5: Zjednodušené schéma ovládání.

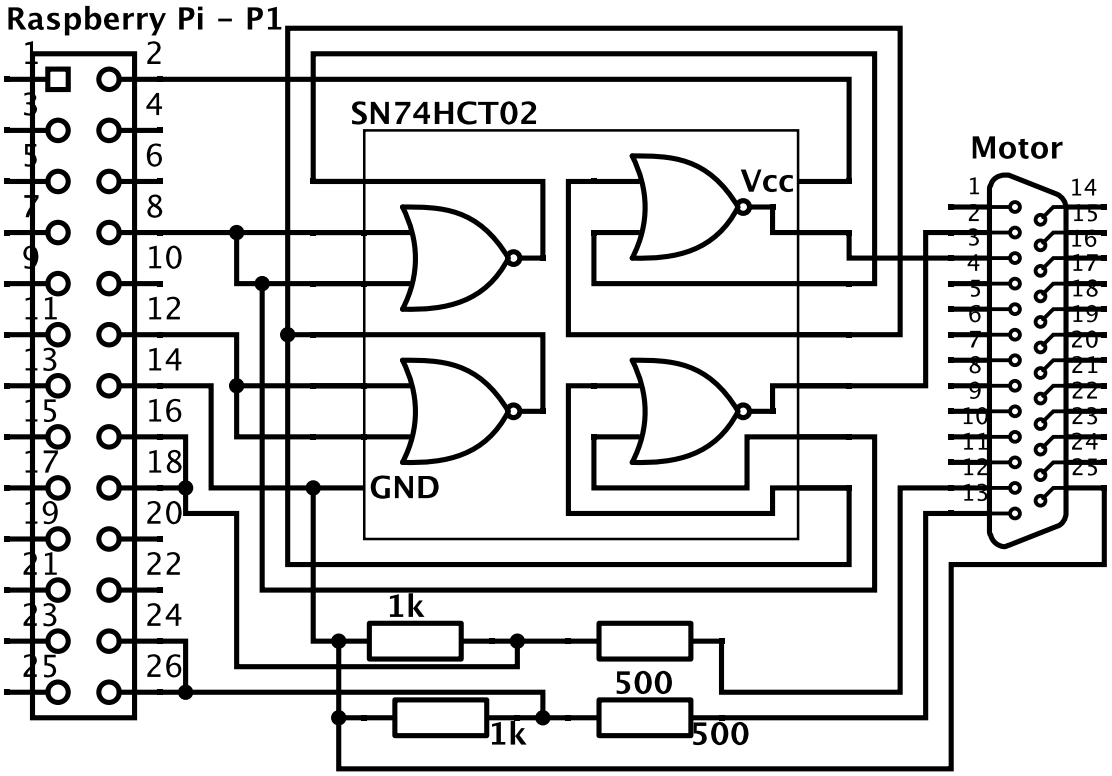
Směr	PWM	L	R
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

Tabulka 4.3: Pravdivostní tabulka.

Výsledné řešení je složeno z hradel typu NOR, z toho dvě jsou použitá jako invertor spojením vstupů. Zjednodušený logický obvod ukazuje obrázek 4.5 s pravdivostní tabulkou 4.3. Slovně se dá interpretovat jako: „Pokud je hodnota směru

v logické jedničce a PWM generuje signál, motor se otáčí vlevo. Pokud je hodnota směru v logické nule a PWM generuje signál, motor se otáčí vpravo. A v případě, že signál PWM není generován, neotáčí se motor na žádnou stranu.“

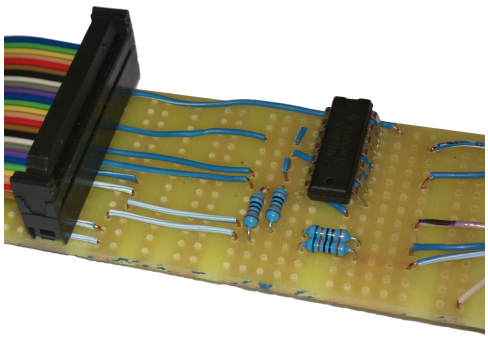
Zdvojení signálu IRC1 a IRC2 na dva různé piny GPIO bylo provedeno při snaze o vylepšení získávání informace ze zpětné vazby motoru. Podrobnosti o tomto kroku jsou zmíněny dále v sekci 5.2.



Obrázek 4.6: Schéma rozhraní mezi Raspberry Pi a motorem.

Výsledné řešení v podobě ručně napájeného obvodu, zobrazuje obrázek 4.7.

(a) Konektor GPIO.



(b) Obvod.

(c) Konektory *CANON 25*.

Obrázek 4.7: Propojovací obvod.

5. ŘÍDICÍ SOFTWARE

5.1 Základní popis

Řídicí software je navržen tak, jak ukazuje diagram na obrázku 5.1. Jméno programu je *servoPi* a v základu je rozdělen na šest komponent. První komponentou je *servoPi_modul*, který zpracovává zpětnou vazbu z motoru. S vysokou prioritou běží vlákno *threadRT_control*, které zajišťuje periodické řízení. Dále je navrženo grafické rozhraní *servoPi_gui* pro ovládání a to využívá programů *servoPi_send* a *servoPi_server*. Program *servoPi_send* běží jako periodické vlákno s nižší prioritou a je určeno k odesílání aktuální pozice natočení hřídele motoru v čase. K účelu zobrazování zpětné vazby také slouží program *servoPi_webserver*, který umožňuje jednoduché sledování přes internetový prohlížeč. Poslední program *servoPi_server* přijme požadovanou hodnotu, na kterou má regulátor regulovat otáčky a předá ji programu *servoPi_control*, konkrétně vláknu *thread_readValue*, které běží s normální prioritou.

Všechny zdrojové kódy programu se nacházejí v příloze A ve složce *servoPi* a na odkazu ve zdroji [26].

Každá funkce je opatřena stručným komentářem, který vysvětluje účel funkce. Pokud jsou některé řádky kódu podstatné, či neobvyklé, jako nastavení PWM, je komentář umístěn i u daného řádku.

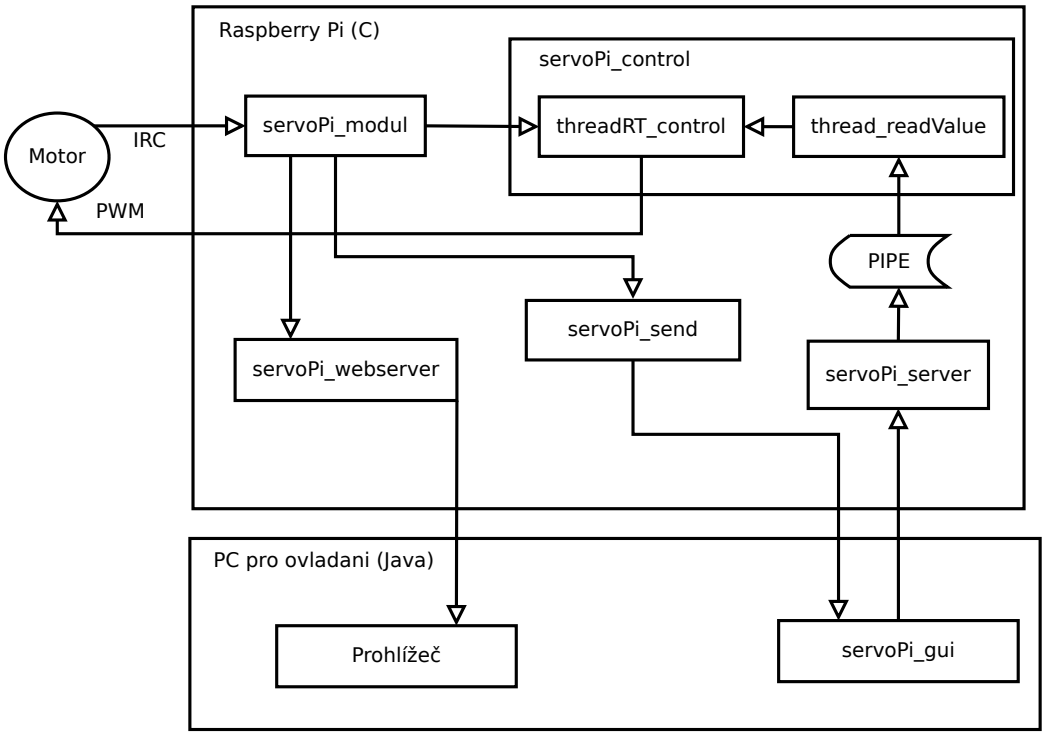
Zdroje pro psaní programu jsou [1], [2], [3], [10], [13], [12] a [18]. Ze zdrojů [1] a [2] jsem čerpal při tvorbě soketové komunikace programů. Zdroj [3] jsem prostudoval a inspiroval se pro vytvoření modulu do jádra a také obsahuje informace ohledně Linuxu a systémového času. Namapování adres registrů vlastního čipu RPi a potřebné makra pro práci s GPIO periferií obsahuje zdroj [10]. Příklad s jednoduchým *real-time* vláknem je obsažen ve zdroji [13]. Adresy registrů s popisem funkcí se nacházejí v dokumentaci [12]. A zdroj [18] odkazuje na obdobnou práci, ze které byl čerpán příklad s webovým rozhraním.

5.2 Ovladač

Tato část je implementovaná jako modul, který se dá zavést do jádra systému. Pojmenovaný je *servoPi_modul* a slouží jako ovladač, který umí přečíst polohu natočení hřídele v aktuálním čase. Ovladač běží v systémovém režimu (kernel space) a využívá ke své činnosti podpůrných funkcí a mechanizmů jádra Linuxu. Zapne potřebné GPIO piny pro vstup signálu. Provede registraci přerušení v procesoru na signál a převádí signál z motoru na polohu.

Modul využívá GPIO piny 7, 8, 23 a 24. Piny nakonfiguruje do funkce logického vstupu a zaregistruje pro ně i přerušení na jednotlivé náběžné a sestupné hrany.

Signál IRC1, který je přiveden na GPIO pin 7 a 8, je zdvojen, aby bylo možné rozlišit mezi vyvoláním přerušení sestupnou, či náběžnou hranou. Stejně tak je to se signálem IRC2 a GPIO piny 23 a 24. Provede se registrace čtyřech funkcí na



Obrázek 5.1: Diagram řídicího softwaru.

přerušení vyvolaném dvěma signály IRC. Každá z funkcí ví, jakou událostí je vyvolána a neztrácí zjištěním zdroje a stavu pinů čas díky této informaci. Zdvojení je provedeno, protože nelze zaregistrovat dvojí přerušení (na sestupnou a na náběžnou hranu) zvlášť pro jeden GPIO pin.

Procesor nedokáže v jednom čase vyvolat více funkcí pro zpracování přerušení současně a přijetí přerušení a i vlastní funkce určitou dobu trvá. Pokud přichází události o změnách v rychlejším sledu, než odpovídá době nutné na jejich zpracování, začne docházet ke ztrátám informace o poloze. Výše zmíněnou optimalizací funkcí a jejich volání došlo k výraznému zvýšení frekvence, do které nedochází ke ztrátám polohových inkrementů oproti řešení ve zdroji [18], které pokaždé počítá směr ze znalosti předešlého stavu a aktuálního stavu. V mém řešení je potřeba zjisti pouze aktuální stav u druhého IRC signálu, jež nevyvolal přerušení. Tabulka 5.1 ukazuje jak lze určit směr pohybu.

K zlepšení přispívá i ukládání informace o předešlé hraně a směru aktuálního otáčení. Pokud souhlasí pořadí v jakém jsou hrany v daném směru za sebou, nemusí se ani volat funkce pro zjištění hodnoty druhého signálu a rovnou se určí směr. Funkce pro zjištění hodnoty druhého signálu, se volá vždy na začátku a pouze v případě vynechání informace předešlého inkrementu, protože zpomaluje rychlost modulu a tím způsobuje nepřesnosti.

Jak je zmíněno v sekci 4.1, daný motor má 200 sestupných a 200 náběžných hran, při jedné otáčce hřídele. Každé zjištění směru (doprava/doleva) znamená posun o $^1/_{400}$ otáčky. Informace se uchová v proměnné datového typu `uint32_t`. Tento

Signál	Hrana	Druhé IRC (hodnota)	Směr
Funkce 1			
IRC1	Sestupná	1	LEVO
IRC1	Sestupná	0	PRAVO
Funkce 2			
IRC1	Náběžná	1	PRAVO
IRC1	Náběžná	0	LEVO
Funkce 3			
IRC2	Sestupná	1	PRAVO
IRC2	Sestupná	0	LEVO
Funkce 4			
IRC2	Náběžná	1	LEVO
IRC2	Náběžná	0	PRAVO

Tabulka 5.1: Tabulka pro určení směru.

datový typ je bez znaménkový, celočíselný a může nabývat hodnot 0 až $(2^{32} - 1)$. Není problém s přetečením, takže je to vhodný datový typ pro získávání informace o rychlosti. Informaci o poloze, může také uchovávat, pouze vždy při přetečení dojde ke ztrátě předešlé informace o absolutní poloze.

Při zavolání čtení ze souboru „/dev/irc0“, modul vrátí informaci o poloze. Jindy se poloha nevypisuje.

Zdrojový kód modulu je obsažen v příloze A (na DVD). Název souboru je „servoPi_modul.c“ a je umístěn v adresáři „servoPi/kernel_modul“. Společně s ním je připraven skript „build.sh“, v kterém je nutné změnit správně cesty k umístnění daných souborů a následným spuštěním se provede kompilace.

Po úspěšném provedení kompilace se v adresáři objeví soubor typu *kernel object* s koncovkou „.ko“. Pokud ponecháte názvy, tak celý název vytvořeného souboru bude *servoPi_modul.ko*. Ten se dá do jádra zavést následujícím příkazem.

```
sudo insmod servoPi_modul.ko
```

Kontrola úspěšného zavedení se provede příkazem, který vytiskne na obrazovku výpis z jádra:

```
dmesg
```

A pokud chceme modul odebrat, stačí zadat tento příkaz.

```
sudo rmmod servoPi_modul
```

5.3 Řídicí program

Tento program se skládá ze dvou vláken. První je určeno pro řízení a druhé přijímá hodnotu, kterou požadujeme pro řízení. Název programu je *servoPi_control* a může

běžet i samostatně s modulem. Bude pak regulovat otáčky na přednastavenou hodnotu 75 ot/min. Pokud chceme regulovat otáčky na jinou hodnotu, musíme použít ostatní části.

Řídicí program obsahuje funkce, které použijí GPIO piny 14 a 18. Pin 14 pro určení směru otáčení a pin 18 pro generování signálu PWM. Piny jsou nakonfigurovány pro funkci logického výstupu.

Signál PWM je generován automaticky bez zatížení procesoru. V dokumentaci [12] je popsáno, do kterých registrů je třeba zapsat konkrétní hodnoty, aby byl PWM signál specifikován. Nabízí několik různě rychlých zdrojů hodinového signálu, jejichž frekvence se dá zmenšit pomocí dvou děliček. Zvolil jsem frekvenci 25 kHz a podařilo se mi nastavit rozlišení střídy signálu na 0,025 %, neboli mohu zvolit hodnotu od 0 do 4000, kde 4000 se rovná 100 % střídě signálu.

5.3.1 Hlavní vlákno

Vlákno, které obstarává řízení, běží s vysokou real-time prioritou 95 a plánovací politikou FIFO. Aktivované je s frekvencí 1 kHz.

Vždy zjistí novou hodnotu polohy natočení hřídele ze souboru „/dev/irc0“, kterou zpřístupňuje ovladač *servoPi_modul*. Motor dosahuje maximálně 4200 ot/min a každá otáčka má 400 rozlišitelných stupňů polohy. S frekvenci 1 kHz může nabývat rychlost hodnot od -28 až 28 v celých číslech. Takto rozdělený rozsah požadované rychlosti je příliš hrubý. Proto je počítán umělý vektor polohy. S každou periodou se spočítá hodnota umělého vektoru a následně se porovná s reálnou polohou. Je tím dosažen jemnější rozsah pro nastavení rychlosti otáčení.

Princip tohoto rozšíření je takový, že hodnota umělého vektoru je uchována v proměnné datového typu `uint64_t`. Prvních 32 bitů vyššího významu obsahuje skutečně měřitelnou a zároveň požadovanou hodnotu polohy. Zbylých 32 bitů obsahuje desetinnou část. Dále je zavedena proměnná typu `int64_t`, která obsahuje změnu polohy za čas s periodou 1 ms. Na začátku každé periody se přičte požadovaná změna k umělé poloze. Poté se chyba použitá pro regulaci spočte porovnáním prvních nejvyšší 32 bitů z umělého vektoru polohy a skutečnou polohou vrácenou modulem.

Tato chyba je dále poslána do funkce, která obstarává výpočet akčního zásahu. Regulátor je typu PID a protože jsem neměl konkrétní požadavky na chování, je zvolen metodou pozorování a manuálního ladění tak, aby byl systém stabilní s co nejrychlejším ustálením. Je použit jeden regulátor na rychlost i na polohu. Poloha se reguluje při nastavení nulové rychlosti. Grafy ukazující průběh regulace otáček jsou k vidění na obrázcích 6.3a a 6.4a.

5.3.2 Nastavování hodnoty

Druhé vlákno, běžící s nižší prioritou, je určené k přijetí hodnoty z takzvané „roury“, neboli *PIPE*. Hodnotu očekává ve formátu `int64_t` s obsahem popsaným výše. Je to proto, aby nemusela být počítána v programu, který obstarává řízení. Zamezí se tím používání datového typu `double`, který vyžaduje matematicky koprocesor.

Hodnotu lze posílat i jiným programem a soubor obsahující rouru se nachází v umístnění „/media/ramdisk/otackyFIFO“ [1].

5.4 Server pro nastavení hodnoty

Aplikace jako celek byla navržena pro řízení přes síť. Přijmutí požadované hodnoty, na kterou má být rychlost motoru regulována, je na straně RPi realizováno pomocí této části, která je pojmenována *servoPi_server*. Účel této části je přijmutí hodnoty typu `double` v ot/min, následné převedení do datového typu `int64_t` podle výše zmíněných požadavků a odeslání do části *servoPi_control* za pomocí roury. Soubor se zdrojovým kódem obsahuje příloha A, pojmenovaný jako „servoPi_server.c“.

5.5 Klient pro posílání hodnoty

Tato část je určená k odesílání hodnoty polohy a času, kdy byla hodnota zaznamenána ze strany RPi. Je posílána na jiný počítač v síti a ten se následně stará o další zpracování. Perioda volání je zvolena tak, aby se stíhalo bez problémů vykreslovat v další části programu na 50 Hz.

Vlákno se tedy nemusí starat o převod z polohy na rychlost, či vykreslování, pouze přečte a odešle hodnotu, aby zbytečně nezatěžovalo RPi. Běží s normální prioritou a název této části je *servoPi_send*. Zdrojový kód je k nalezení v příloze A, pojmenovaný jako „servoPi_send.c“.

5.6 Webové rozhraní

Podle vzoru předmětu *Programování systému reálného času*, jsem přidal k práci webové rozhraní. Postupoval jsem podle návodu, který je určen studentům tohoto předmětu (viz. [18]).

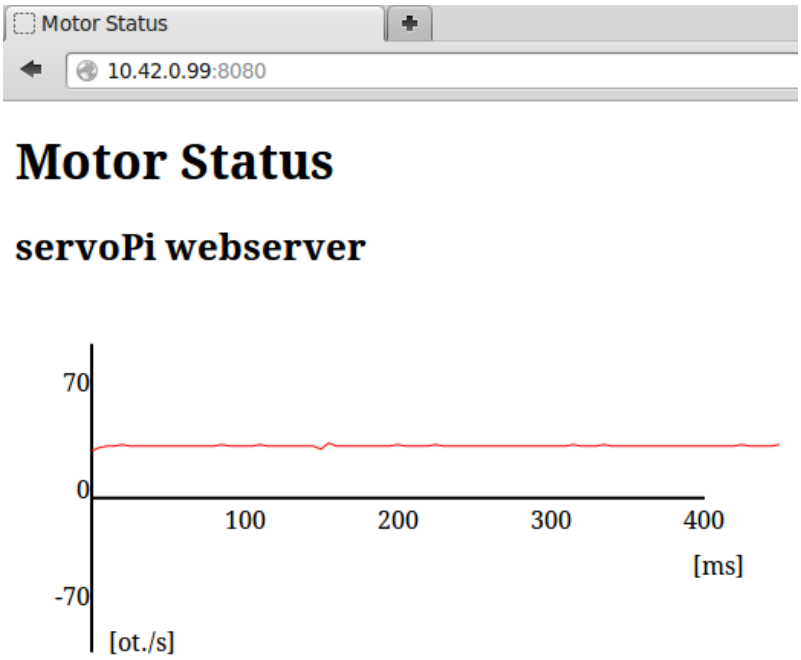
Na webovou stránku, která obsahuje informace v podobě grafu rychlosti otáčení motoru, se dostanete zadáním IP adresy RPi do adresního řádku prohlížeče a pro komunikaci využívá portu 8080. Stránka se automaticky obnovuje po 0,2 s a jednotky grafu jsou na horizontální ose v ms a na vertikální ose v ot/s.

Program jsem pojmenoval *servoPi_webserver*. Soubor se zdrojovým kódem je k nalezení v příloze A, konkrétně na cestě „servoPi/servoPi_webserver.c“.

5.7 Grafický program

Poslední částí celku je aplikace *servoPi_gui*. Jako jediná ze zmíněných částí byla naprogramována v jazyce Java. Volba zmíněného jazyka byla použita z důvodu

[1] K složce ”/media/ramdisk”je připojen svazek, který je realizovaný souborovým systémem `tmpfs`, který data ukládá pouze do paměti RAM (nastavený limit 10 MB). Není však nutné mít toto rozšíření, program bude fungovat i bez něj.



Obrázek 5.2: Webové rozhraní.

jednoduché objektové implementace grafického panelu i s plynulým vykreslováním. Program je určen k použití na jiném počítači na síti, tedy nezatěžuje vykreslováním a přepočítáváním RPi.

Realizována je struktura objektů, které uchovávají informaci o poloze v čase a také umí vrátit průměrné rychlosti v časech. Tyto informace jsou použity pro vykreslení na běžném formuláři v podobě grafu.

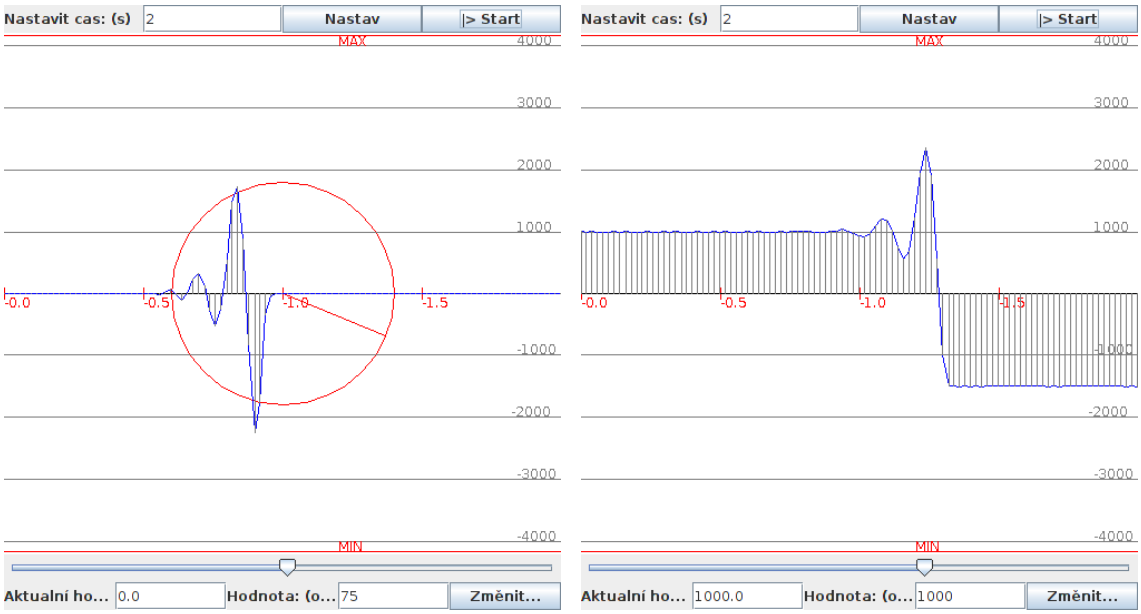
Graf obsahuje možnost nahlížet do historie 1 s až 20 s. Aktuální data se vykreslují v levé části a jsou posouvány vpravo. Při zpomalení otáček na rychlost -199 ot/min až 199 ot/min se začne ještě vykreslovat kolečko znázorňující natočení hřídele (viz. obrázek 5.3a). Celý graf se dá pozastavit a následně opět pustit.

Formulář dále obsahuje posuvník, kterým se nastavuje rychlost otáčení. Pro přesnější nastavení je určeno textové pole s tlačítkem k odeslání. V levé dolní části se nachází číselně vyjádřená hodnota aktuální rychlosti v jednotkách ot/min průměrována za posledních 5 vzorků. Příklad vykresleného grafu je vidět na obrázku 5.3b a jednotky horizontálně oddělených úrovní jsou v ot/min.

Zdrojový kód části obsahuje příloha A, konkrétně v archivu *zip* se jménem souboru „servoPi_gui.zip“.

5.8 Porovnání zobrazovacích programů

Pravděpodobně by se dal s některou z dnešních webových technologií vytvořit plynule se vykreslující graf, ale pořád je tu problém, že se o vytváření grafu musí postarat server. Což je v tomto případě RPi, které je tímto zatěžováno další úlohou přepočítávání polohy na rychlost atd. Zbytečné přenosy celé webové stránky a komunikace s prohlížečem, jsou také další přítěží.



(a) Vykreslení polohy.

(b) Vykeslení grafu rychlosti.

Obrázek 5.3: Program servoPi_gui.

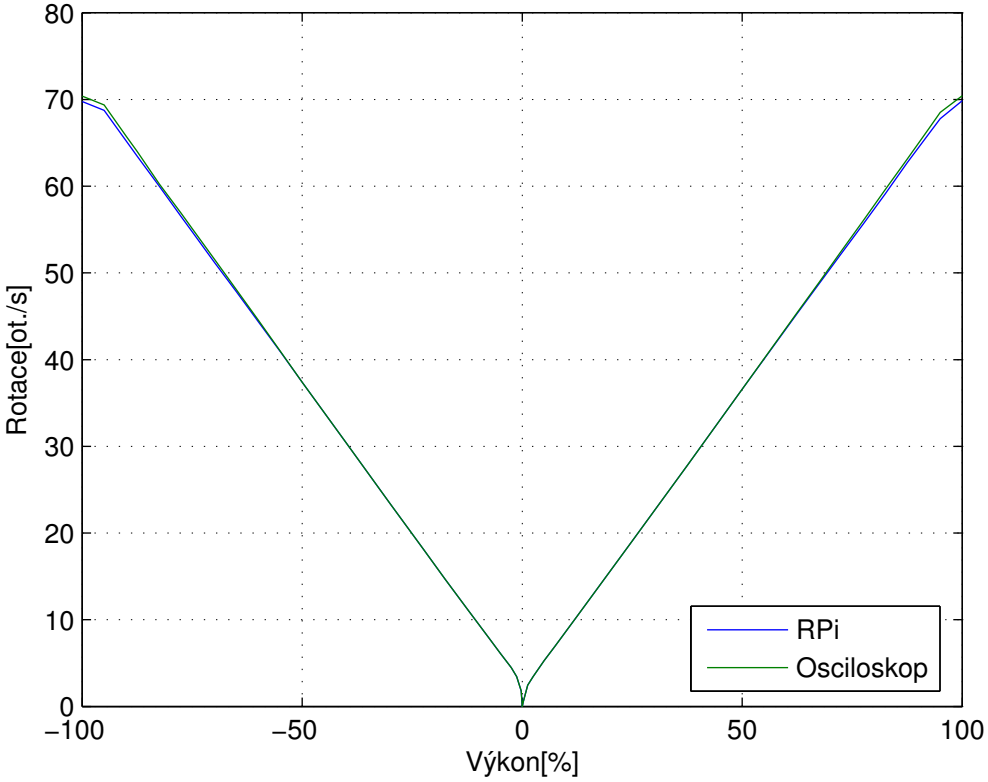
Oproti tomu mé řešení s grafickým rozhraním, které přijímá pouze statická data (čas a polohu v čase), nezatěžuje RPi ničím jiným než odesláním pár bajtů informací. Navíc je graf vykreslován průběžně a je možné jej pozastavit, či zvolit časový rozsah. V neposlední řadě přepočty z polohy na rychlost, provádí počítač, který se nestará o řízení motoru.

6. OVĚŘENÍ A MOŽNOSTI RPI

6.1 Frekvenční možnosti snímání zpětné vazby

Funkce modulu, neboli určení aktuální rychlosti, byla ověřena pomocí osciloskopu. Výsledek je zobrazen v grafu (viz. obrázek 6.1).

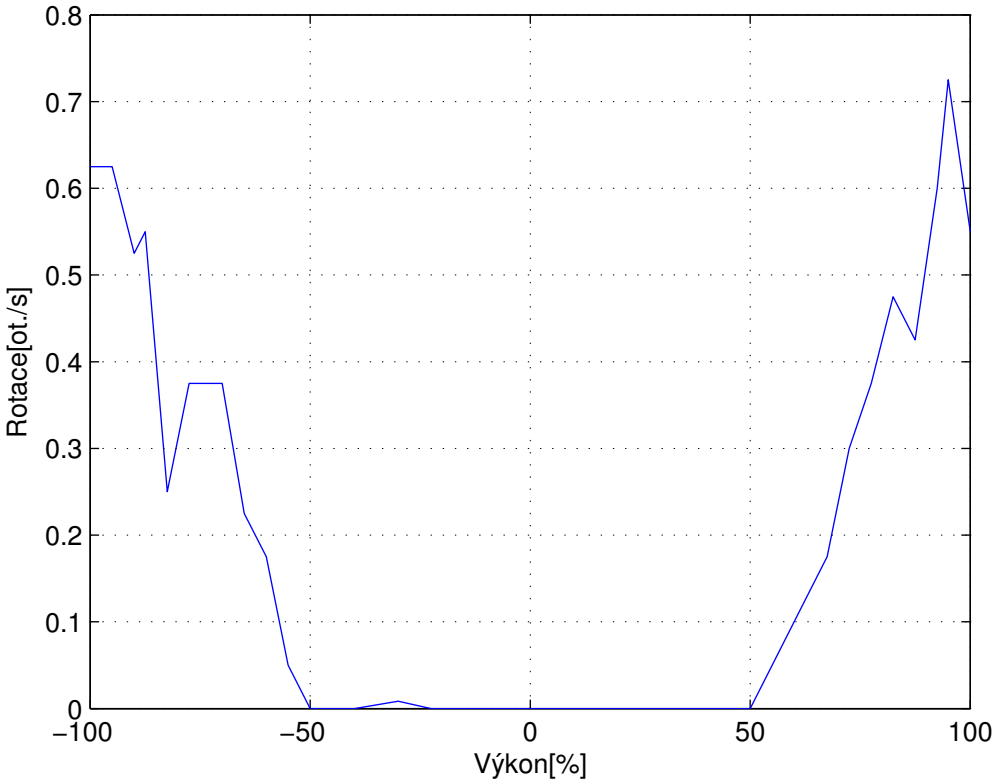
Na osciloskopu byl měřen signál IRQ a RPi měřilo signál z dvoukanálového IRC senzoru. Hodnoty frekvencí se i při nastavení určité pevné střídy nepatrně měnily a protože bylo vše odečteno ručně, měření podléhá této chybě. Frekvence byla přepočítána na otáčky za vteřinu a větší patrnost rozdílu je vidět v grafu na obrázku 6.2, který zobrazuje rozdíl otáček za vteřinu.



Obrázek 6.1: Graf závislosti střídy a naměřených otáček.

Střída v rozsahu od -50 % do 50 % (to odpovídá asi ±35 ot./s) je měřena přesněji. Od vyšších otáček vzniká chyba, která ještě více vzroste při zatížení RPi. Nemožnost měřit správně otáčky od vyšších frekvencí zapříčiní, že ani regulátor nemůže regulovat otáčky na správnou hodnotu. Pro tento konkrétní motor je tedy relativní chyba kolem jednoho procenta, ale bez zatížení systému.

Při zatížení systému, hlavě vstupních a výstupních periferií, je chyba náhodná a větší. Zatížení procesoru měření nečiní nepřesným.



Obrázek 6.2: Graf rozdílu naměřených otáček s RPi oproti osciloskopu.

6.2 Zhodnocení řízení motoru

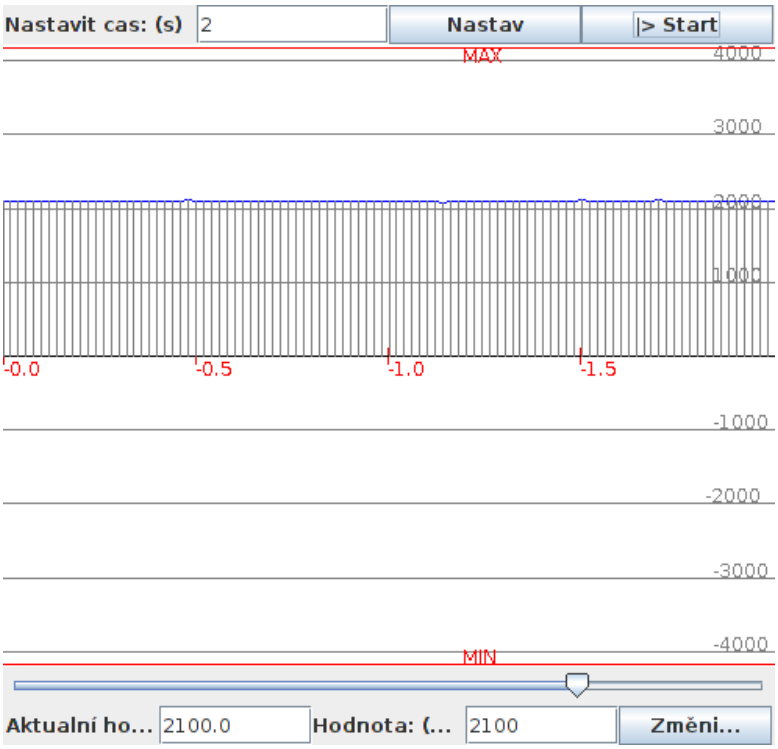
Vzhledem k předchozímu měření, je dané, že RPi nemůže přesně regulovat otáčky, protože je nestihne bezchybně měřit pro vyšší frekvence. Tedy regulování otáček velkých rychlostí není pro samotné RPi vhodná úloha, při potřebě vyšší rychlosti a přesnosti. Pokud není potřeba řídit rychlost na vyšší otáčky nebo při řízení polohy bude zajištěno, že referenční poloha se bude měnit maximálně 14 IRC/ms, tak bude systém plně použitelný.

S řádově polovičními otáčkami oproti maximálním daného motoru, je již rychlost regulovatelná s relativně velkou přesností. Vylepšení, které umožňuje zvolit mezi mnoha úrovněmi rychlostí, zároveň zapříčiní, že je regulovaná průměrná rychlost a nikoliv aktuální. Má to za následek, že pokud na hřídel působí delší dobu síla, kterou akční zásah nezvládne vyregulovat, potom při uvolnění této síly se regulátor bude snažit prodlevu dohnat a to 100% silou, až dokud nevyrovná průměrnou rychlost a poté se opět vrátí ke správnému regulování rychlosti.

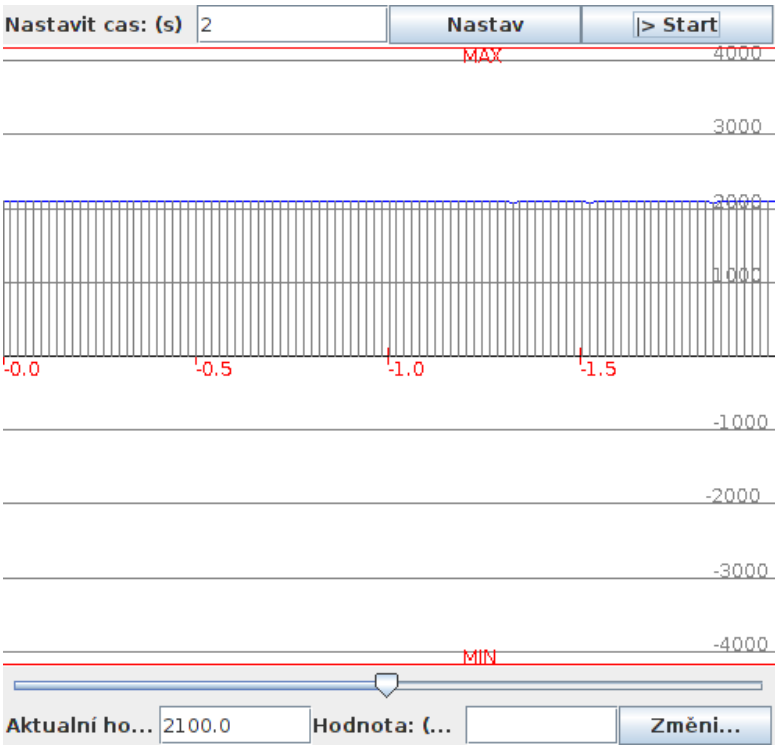
Regulování polohy je možné při nastavení nulové rychlosti. Systém je vcelku schopný vrátit polohu při vychýlení do správné pozice. Ale opět zde hraje roli měření z předchozí sekce. Tedy pokud se při vychýlení dosáhne akčního zásahu, který způsobí, že rychlost motoru nebude kompletně měřitelná, potom se ztratí neurčitý počet inkrementů a poloha bude o tuto ztrátu posunuta.

Obrázek 6.3a ukazuje regulaci otáček na hranici bez ztrátové oblasti měření pro

2100 ot/min a s nastavením pevné střídy PWM, v přibližně stejné oblasti rychlosti otáčení, zobrazuje průběh obrázek 6.3b. Podobně pak obrázeky 6.4a a 6.4b ukazují měření s regulátorem a měření pevné střídy PWM v oblasti 4000 ot/min.

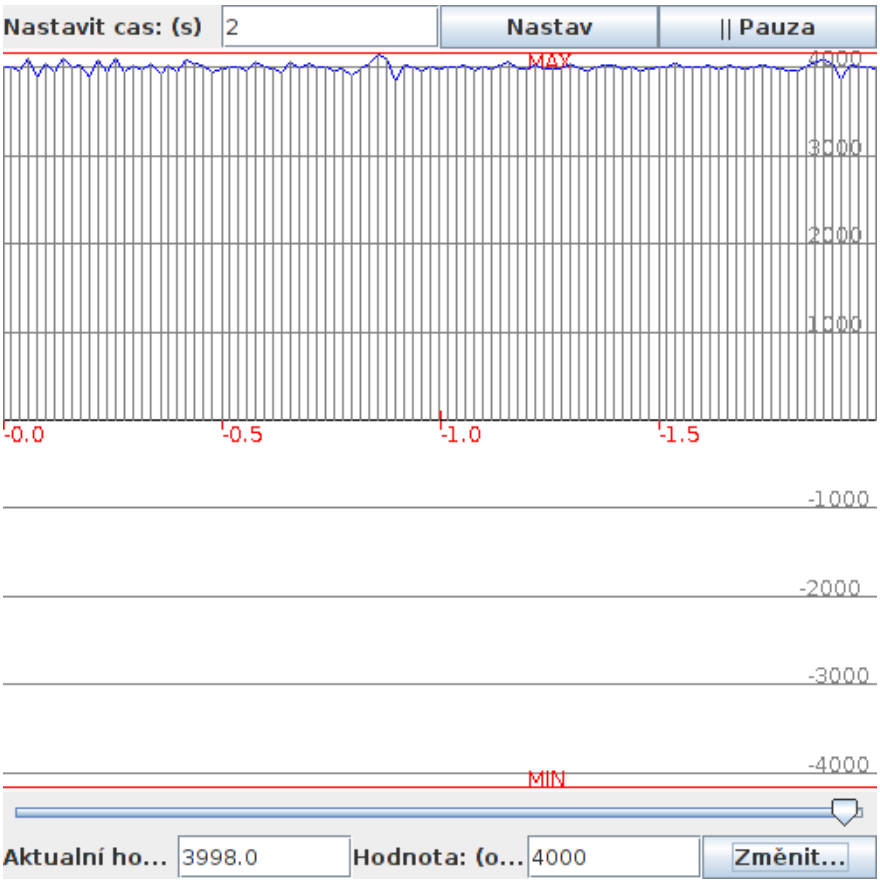


(a) Řízení rychlosti pro 2100 ot/min.

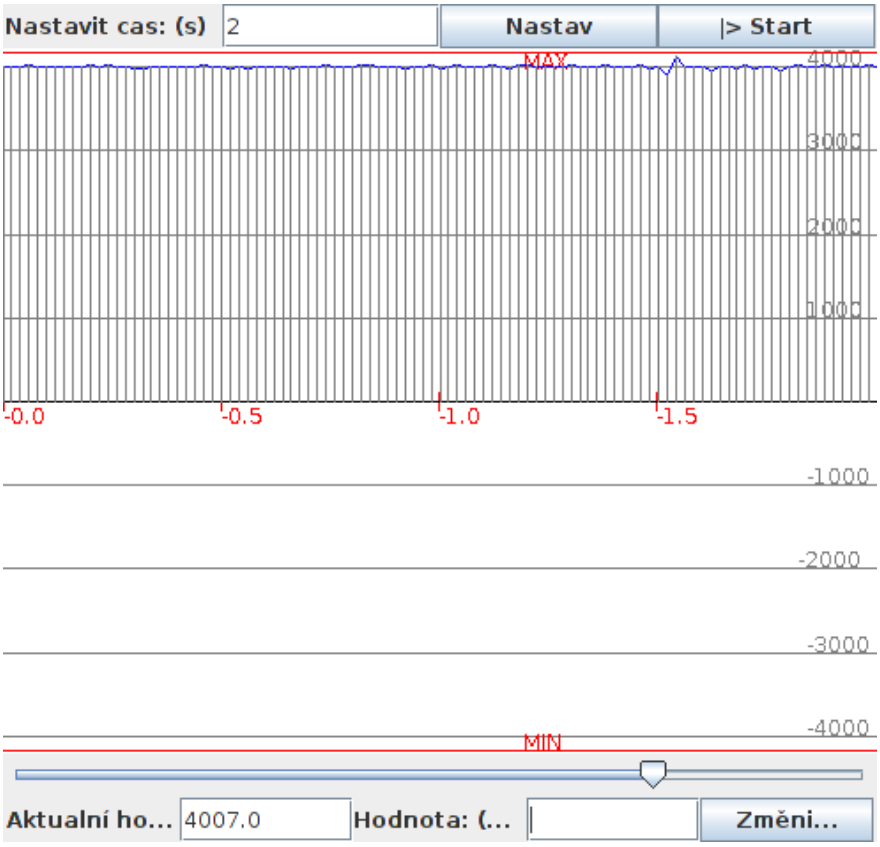


(b) Měření rychlosti s pevně nastaveným výkonem motoru 48 %.

Obrázek 6.3: Grafy rychlosti v oblasti s přesným měřením.



(a) Řízení rychlosti pro 4000 ot/min.



(b) Měření rychlosti s pevně nastaveným výkonem motoru 93 %.

Obrázek 6.4: Grafy rychlosti v oblasti s nepřesným měřením.

7. NÁVRH ÚPRAV A ZLEPŠENÍ

7.1 Priority

Při ověřování systému a hledání možností, jak zpřesnit měření, bylo zjištěno, že v jádře Linuxu, rozšířeného o real-time plánovací algoritmy, je možné určovat prioritu i pro vlákna obsluhující přerušení. Pomocí externího nástroje `schedtool` byla změněna priorita obslužných vláken IRC signálu ze přednastavené priority 50 na prioritu 90. Dále pak byly použity jednotlivé priority podle tabulky 7.1.

Komponenta	**Priorita**
servoPi_modul	90
servoPi_control	89
servoPi_send	55
servoPi_server	51

Tabulka 7.1: Zvolené priority.

Zaznamenáno bylo výrazné zlepšení, jak měření, tak řízení. Měření již pravděpodobně nepodléhalo procentuální chybě a nejvyšší zaznamenané otáčky přesahovaly hranici 4200 ot/min.

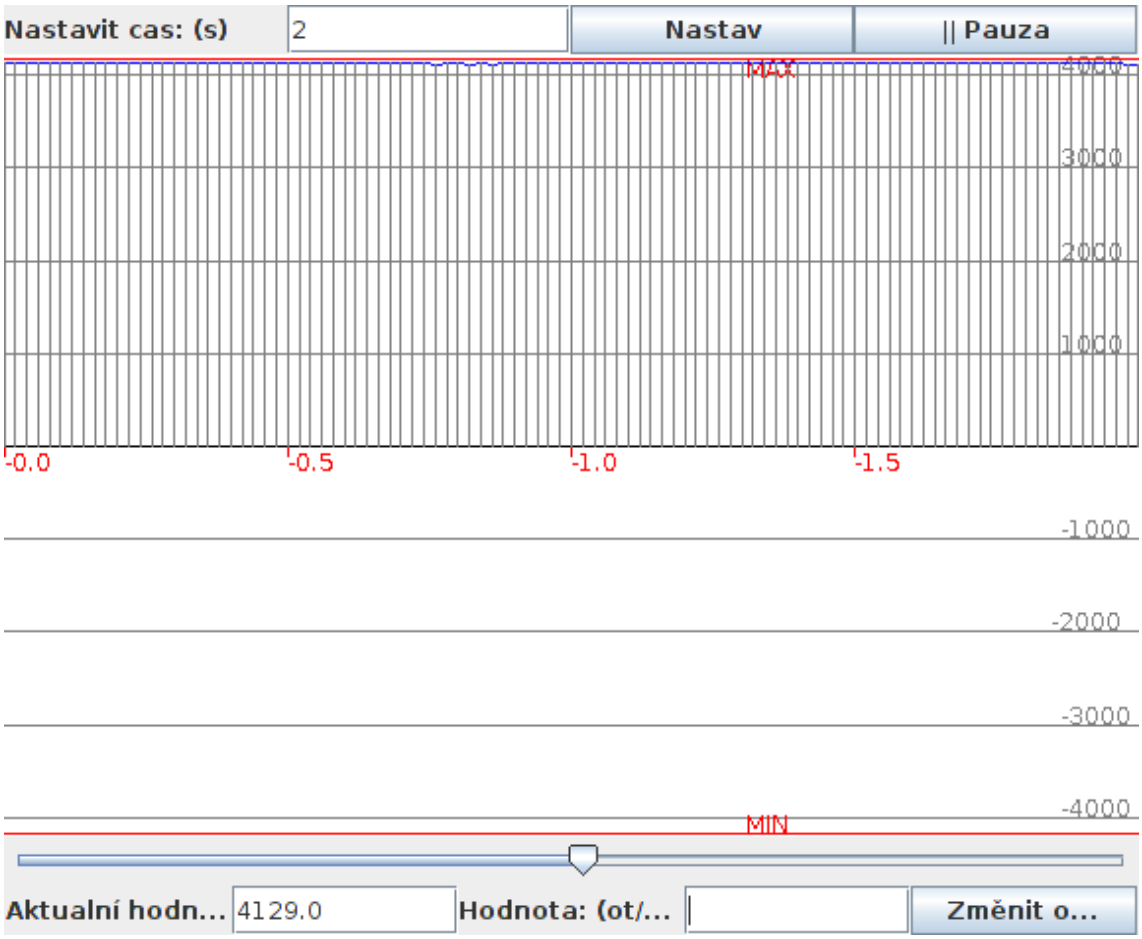
Měření nebylo otestováno za pomocí externího ověření, ale pouze za pomocí programu *servoPi*. První test ukazuje obrázek 7.1, kde byla nastavena pevná střída signálu PWM na hodnotu přibližně 98 %. Měření se jeví jako stabilní, až na drobné výkyvy, které se projevovaly i při měření osciloskopem v předchozí sekci 6.1. Kolísání může způsobovat nerovnoměrné rozložením tření hřídele motoru, či nerovnoměrnost vynutí, nebo také nestabilita zdroje energie při zatížení.

Druhý test byl proveden se zapnutým regulátorem ve ztrátové oblasti zjištěné v předchozí kapitole 6. Průběh testu ukazuje obrázek 7.2. O regulaci se dá konstatovat, že se jeví jako stabilnější a bez kolísání, ale nebyla ověřena externími měřícími přístroji.

7.2 Webové rozhraní

Jako lepší implementaci webového rozhraní by mohl být použit applet. Z programu *servoPi_gui* by již nebylo obtížné provést dědění z třídy *JApplet*.

Řešení by zatěžovalo RPi pouze prvotním nahráním grafické aplikace do prohlížeče na stranu klienta, ale jinak by zachovalo všechny výhody návrhu programu *servoPi*. Nevýhodou oproti jednoduchému webovému rozhraní ze sekce 5.6 je nutnost mít na klientské stanici nainstalovánu podporu pro jazyk Java.

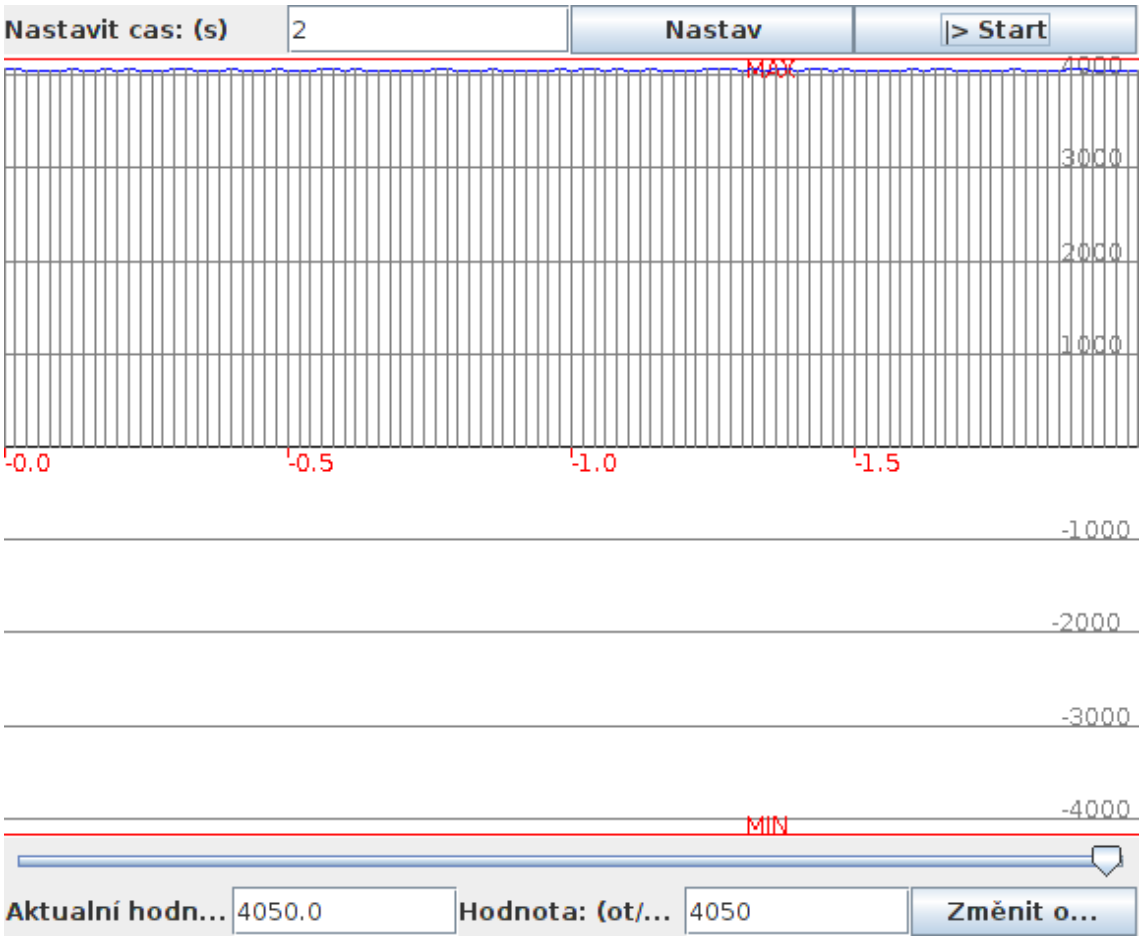


Obrázek 7.1: Test měření pevně nastavené střídy PWM na 98 %.

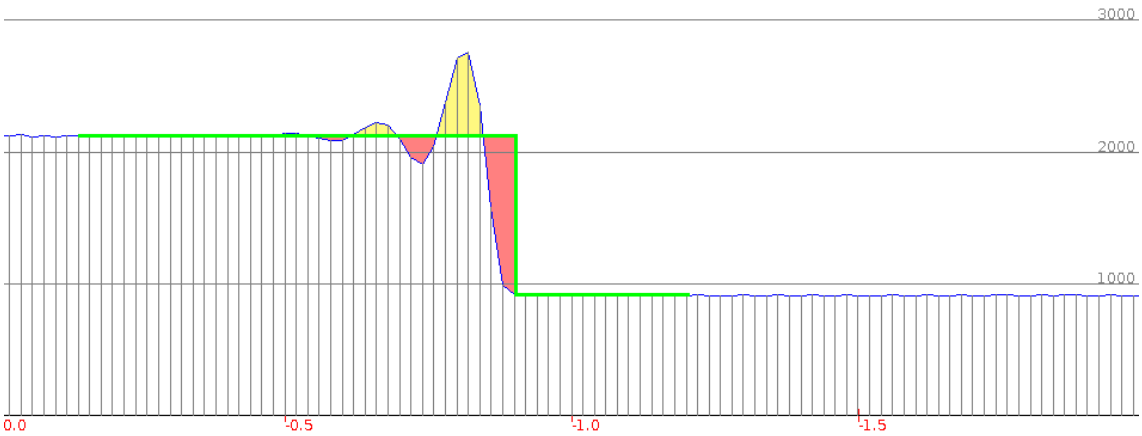
7.3 Regulátory

Rozšíření na mnoho úrovní nastavitelné rychlosti (viz. sekce 5.3.1) zároveň způsobuje přesah při změně požadované hodnoty. Obrázek 7.3 ukazuje zeleně vyznačenou barvou požadovanou hodnotu, červené plochy ukazují hodnotu, jež způsobuje akumulování stále horší průměrné rychlosti a žluté úseky, které se naopak snaží kompenzovat ztrátu rychlosti. Proto má také změna přibližně 50% překmitnutí oproti požadované hodnotě.

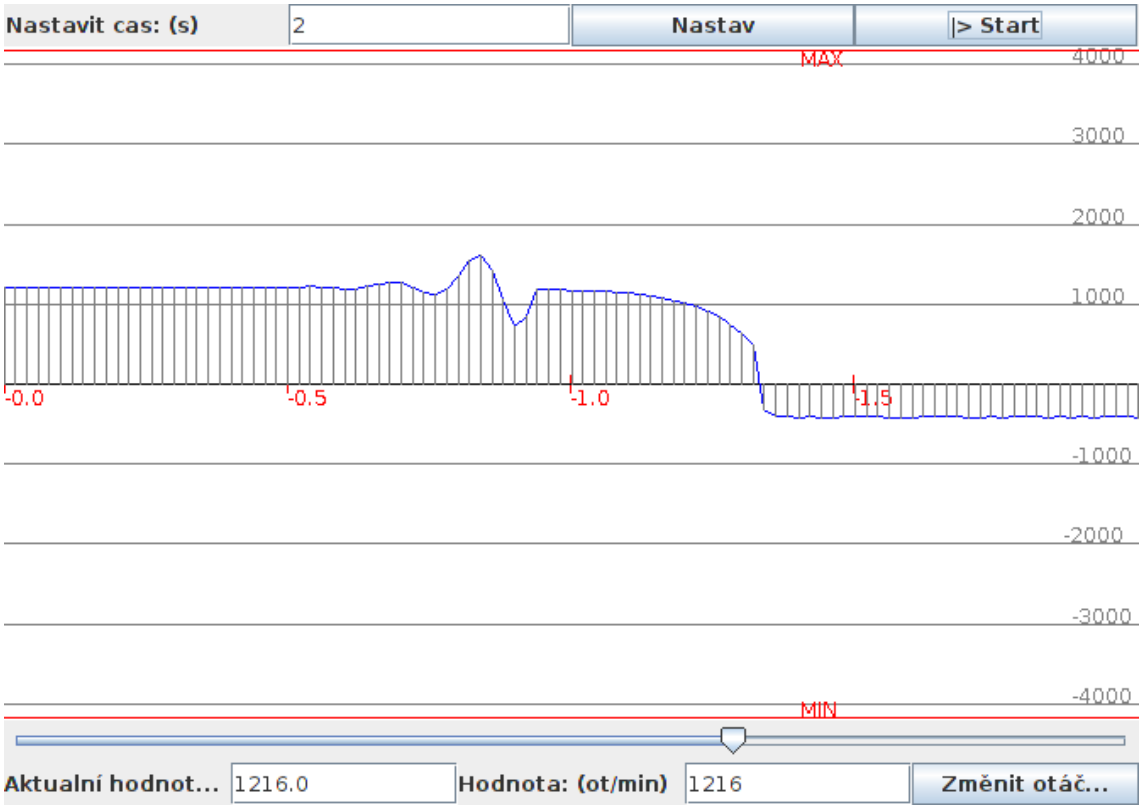
Rozšíření je ale potřebné, jinak by možnosti volby rychlostí byly příliš hrubé. Zachování této jemnosti volby a zároveň dosažení lepší odezvy na změnu je možné s přepínáním regulátorů. První regulátor reguluje aktuální rychlost do přibližné požadované rychlosti. Následně se přepne na regulátor, který využívá umělý vektor polohy. Řešení nebylo odladěno do finální podoby. Problém je v správném nastavení již naintegrované složky, která způsobuje, že po přepnutí regulátorů se objeví kolísání. Pokus zobrazuje obrázek 7.4, kde je vidět nejprve lepší průběh ustálení, ale po přepnutí následuje zmíněné kolísání.



Obrázek 7.2: Test regulace na 4050 ot/min.



Obrázek 7.3: Výřez programu servoPi.



Obrázek 7.4: Pokus s přepínáním regulátorů.

8. ZÁVĚR

Práce ověřila, že je možné použít levnou platformu RPi pro zpětnovazební řízení rychlosti s inkrementálním senzorem čistě softwarově bez nutnosti návrhu periferie pro hardwarové zpracování a čítání inkrementů polohy. Použitelnost je ovšem limitována maximální frekvencí inkrementů, které je RPi a jádro systému Linux schopné zpracovat. Pro použitý motor (400 inkrementů na otáčku, maximální rychlost 4200 ot/min) je řízení při maximálních otáčkách na hranici propustnosti systému a pro více než 2100 ot/min dochází již k občasné ztrátě inkrementu.

Do frekvence 14 kHz byl kontrolním pozorováním na osciloskopu ověřen souhlas s rychlostí otáčení měřenou RPi. Od vyšších frekvencí než 14 kHz byl již rozdíl pozorovatelný tím více, čím vyšší byla frekvence (a rychlost otáčení). Relativní chybovost pro tento motor, která dosahuje nejvyšších hodnot při největší rychlosti otáčení motoru, je s nezatíženým systémem kolem 1 % a při zatížení systému je úměrná míře zatížení především vstupně výstupních periferií.

Možnost regulovat otáčky je závislá od možnosti otáčky určit. Pro první polovinu rozsahu otáček (tj. do 2100 ot/min) s použitým motorem je regulace přesná a bez výkyvů (viz. 6.3a). Zbylý rozsah vyšších otáček je do určité míry regulovatelný (viz. 6.4a), ale náhodné chyby měření způsobují, že regulátor se snaží chybu kompenzovat i pokud je reálná rychlost správná a to vede k dalším chybám.

Během poslední fáze ověřování systému byly i znova hledány možnosti snížení počtu nezapočítaných inkrementů. Jako nejvýhodnější se ukázalo řešení, kdy byly zvýšeny priority příslušných vláken pro obsluhu přerušení. Toto řešení bylo nalezeno relativně pozdě a potřebovalo by další kolo důkladného testování. Ale i na základě zběžného testování lze předpokládat odstranění procentuální chyby měření při vysoké rychlosti otáčení.

Za předpokladu, že výše zmíněná úprava opravdu vylepšila měření i řízení, se dá konstatovat, že Raspberry Pi je vhodné k řízení celého rozsahu použitého motoru. Pravděpodobně umožňuje i zpětnovazební řízení pro jiné úlohy o ještě vyšších frekvencích snímaní zpětné vazby. Ale tyto skutečnosti je nutné ověřit.

Seznam zdrojů

Knihy

[1] MITCHELL, Mark; OLDHAM Jeffry; SAMUEL Alex. *Pokročilé programování v operačním systému Linux*. Praha : SoftPress s. r. o. 2002. 320 s. ISBN 80-86497-29-1.

[2] STONES Richard; MATTHEW Neil. *Linux : Začínáme programovat*. Praha : Computer Press. 2000. 897 s. ISBN 80-7226-307-2.

[3] JELÍNEK Lukáš. *Jádro systému Linux : kompletní průvodce programátora*. Praha : Computer Press. 2008. 686 s. ISBN 978-80-251-2084-2.

[4] MEMBRAY Peter; HOWS David. *Learn Raspberry Pi with Linux*. New York : Apress. 2013. 249 s. xix. ISBN 978-1-4302-4821-7.

[5] UPTON Eben; HALFACREE Gareth. *Raspberry Pi : User Guide*. Chichester : Wiley. 2012. xiv. 248 s. ISBN 978-1-118-46446-5.

Internetové zdroje

[6] *NOOBS setup | Raspberry Pi* [online]. [cit. 2014-05-05]. Dostupné z: `http://www.raspberrypi.org/help/noobs-setup/`

[7] *Raspberry Pi · View topic - CONFIG_PREEMPT_RT on Raspberry Pi* [online]. [cit. 2014-05-05]. Dostupné z: `http://www.raspberrypi.org/forums/viewtopic.php?t=39951`

[8] *RPi Kernel Compilation - eLinux.org* [online]. [cit. 2014-05-05]. Dostupné z: `http://elinux.org/RPi_Kernel_Compilation`

[9] *INSTALLING OPERATING SYSTEM IMAGES ON LINUX* [online]. [cit. 2014-05-05]. Dostupné z: `http://www.raspberrypi.org/documentation/installation/installing-images/linux.md`

[10] *RPi Low-level peripherals* [online]. [cit. 2014-05-05]. Dostupné z: `http://elinux.org/RPi_Low-level_peripherals`

[11] *Raspberry Pi - Wikipedia* [online]. [cit. 2014-05-05]. Dostupné z: `http://cs.wikipedia.org/wiki/Raspberry_Pi`

[12] *Broadcom - BCM2835 ARM Peripherals* [online]. [cit. 2014-05-05]. Dostupné z: `http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf`

[13] *Real-Time Linux Wiki* [online]. [cit. 2014-05-05]. Dostupné z: https://rt.wiki.kernel.org/index.php/Main_Page

[14] *Cyclictest - RTwiki* [online]. [cit. 2014-05-05]. Dostupné z: https://rt.wiki.kernel.org/index.php/Cyclictest

[15] *motor controller scheme* [online]. 2004. [cit. 2014-05-05]. Dostupné z: https://rtime.felk.cvut.cz/psr/cviceni/semestralka/sch-controller.pdf

[16] *Zapojeni konektoru* [online]. 2004. [cit. 2014-05-05]. Dostupné z: https://rtime.felk.cvut.cz/psr/cviceni/semestralka/sch-connectors.pdf

[17] SOJKA Michal; ŠPINKA Ondřej. *RTLinux Motor Controller* [online]. [cit. 2014-05-05]. Dostupné z: https://rtime.felk.cvut.cz/psr/cviceni/semestralka/slides-motor.pdf

[18] *Semestral Work - Motor Control* [online]. [cit. 2014-05-05]. Dostupné z: https://rtime.felk.cvut.cz/psr/cviceni/semestralka/

[19] *WiringPi* [online]. [cit. 2014-05-05]. Dostupné z: http://wiringpi.com/

[20] *FrontPage - Raspbian* [online]. [cit. 2014-05-05]. Dostupné z: www.raspbian.org

Seznam zdrojů souborů a zdrojových kódů

[21] *2014-01-07-wheezy-raspbian* [online]. [cit. 2014-05-05]. Dostupné z: http://downloads.raspberrypi.org/raspbian_latest

[22] *raspberrypi/linux* [online]. [cit. 2014-05-05]. Dostupné z: https://github.com/raspberrypi/linux

[23] *Index of /pub/linux/kernel/projects/rt* [online]. [cit. 2014-05-05]. Dostupné z: https://www.kernel.org/pub/linux/kernel/projects/rt/

[24] *raspberrypi/tools* [online]. [cit. 2014-05-05]. Dostupné z: https://github.com/raspberrypi/tools

[25] *sarfata/pi-blaster* [online]. [cit. 2014-05-05]. Dostupné z: https://github.com/sarfata/pi-blaster/

[26] *Ramese/servoPi* [online]. [cit. 2014-05-05]. Dostupné z: https://github.com/Ramese/servoPi/

A. DVD

(Pro zdrojové kódy je použita zkratka ZK.)
Přiložené DVD obsahuje následující soubory:

servoPi ..... Adresář obsahující program servoPi

- **kernel_modul** ..... Adresář obsahující modul
  - build.sh ..... Pomocný skript
  - Makefile ..... Pomocný skript
  - servoPi_modul.c ..... ZK modulu
- **spustitelné** ..... Zkompilované soubory
  - servoPi_control.run ..... Regulační program
  - servoPi_modul.ko ..... Modul do jádra
  - servoPi_send.run ..... Program posílající hodnoty
  - servoPi_server.run ..... Program pro přijetí hodnoty
  - servoPi_webserver.run ..... Program webserveru
- servoPi_send.c ..... ZK programu pro posílání hodnoty
- servoPi_server.c ..... ZK programu pro přijetí hodnoty
- servoPi_control.c ..... ZK programu regulátoru
- servoPi_gui(Java).zip ..... NetBeans projekt zobrazovacího programu
- servoPi_webserver.c ..... ZK webserveru

Linux ..... Adresář s OS Linux

- linux-rpi-3.10.y.zip ..... ZK Linuxu pro RPi
- patch-3.10.26-rt24.patch.gz ..... Real-time rozšíření
- tools-master.zip ..... Cross kompilační nástroj

test ..... Adresář s testem

- vlakno_test_planovani_1kHz.c ..... ZK testu
- vlakno_test_planovani_1kHz.run ..... Spustitelný test

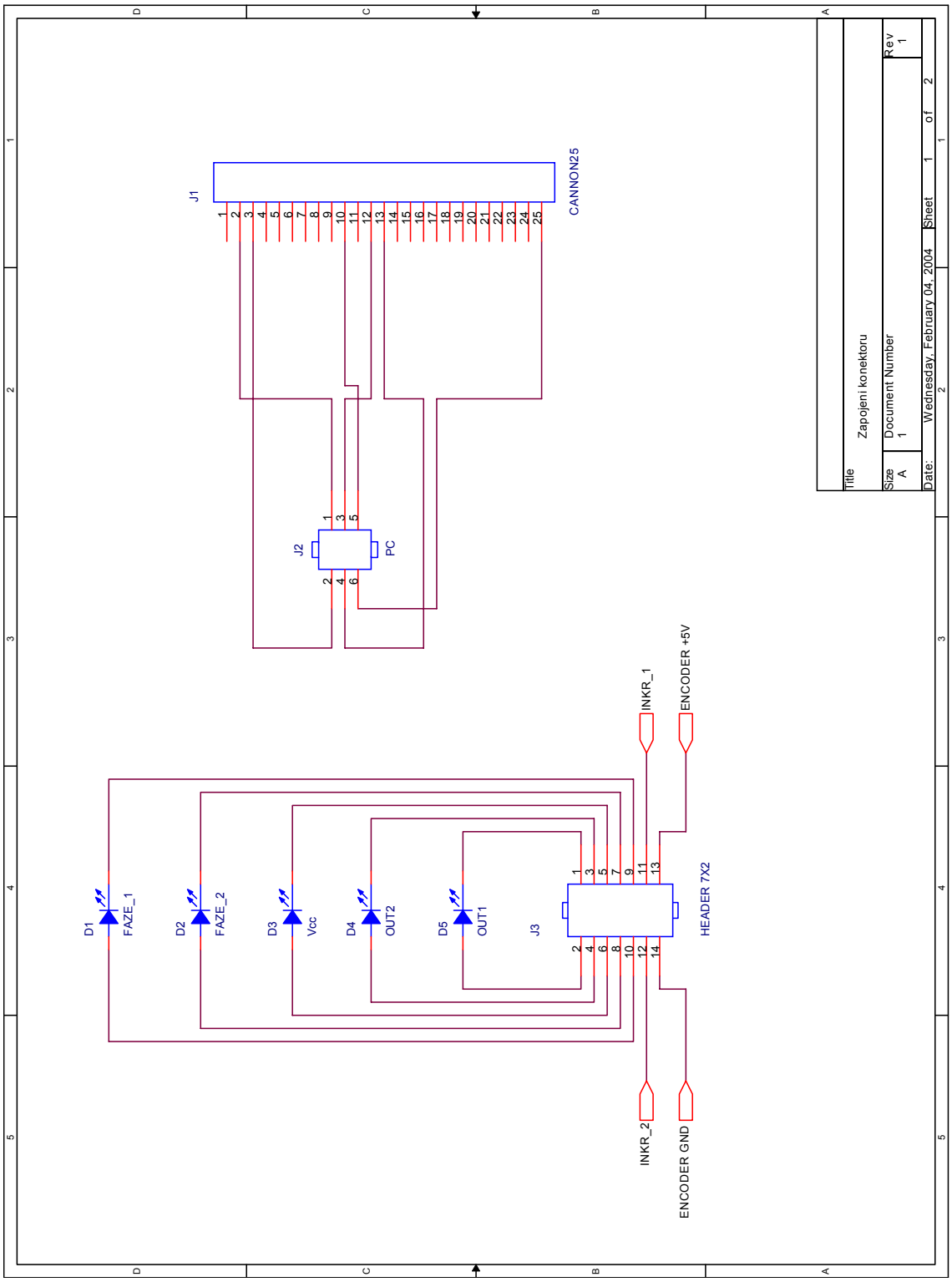
Text práce ..... Adresář obsahující text práce

- radek_meciar_BP_tisk.pdf ..... Soubor vhodný pro tisk
- radek_meciar_BP_eview.pdf ..... Soubor vhodný pro prohlížení

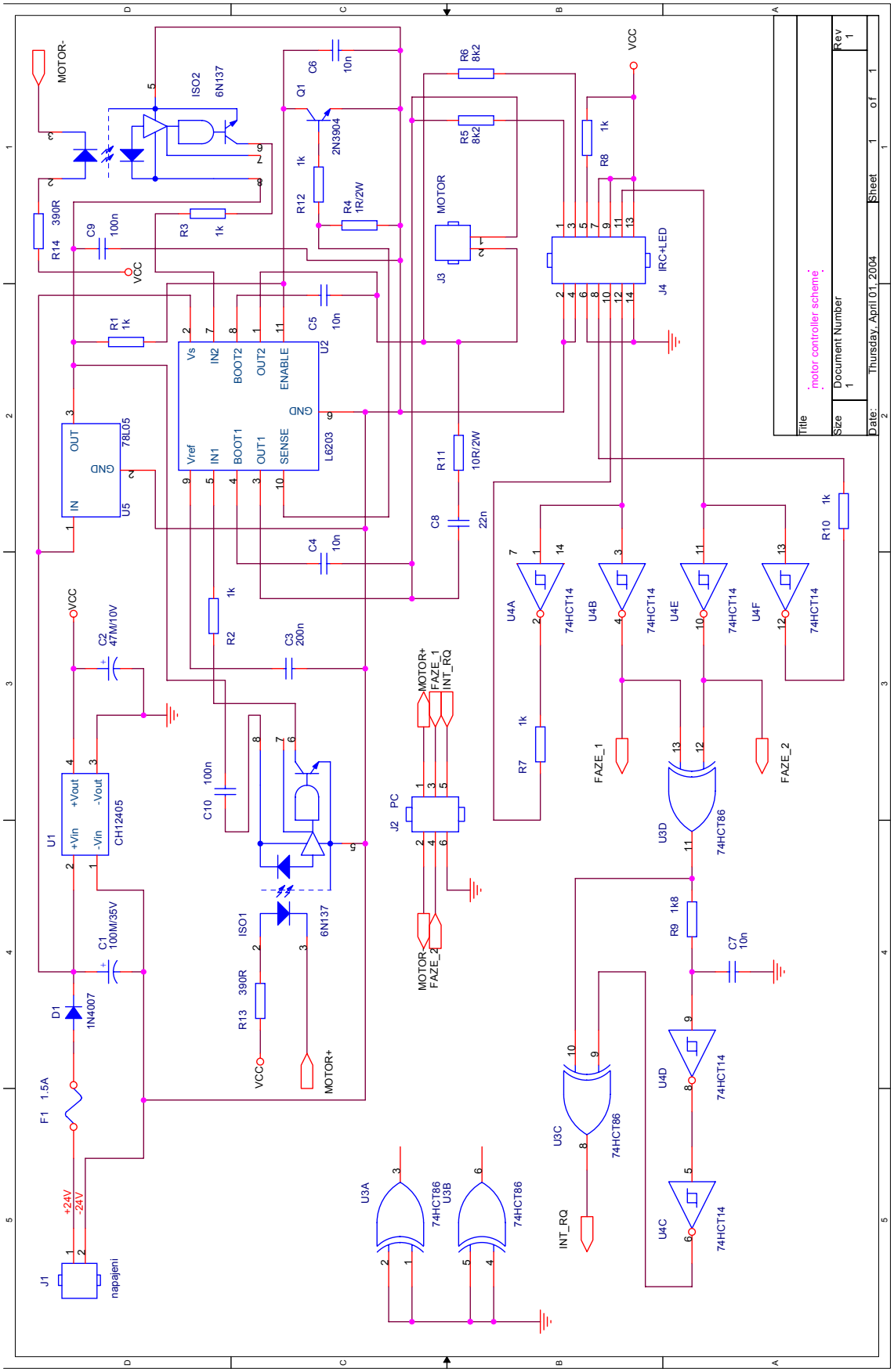
B. SEZNAM ZKRATEK

Zkratka	Anglický význam	Český význam
DMA	Direct Memory Access	Přímý přístup do paměti
GPIO	General-purpose input/output	Základní vstupně výstupní rozhraní
GPL	General Public License	Všeobecná veřejná licence
GUI	Graphical user interface	Grafické uživatelské rozhraní
HDMI	High-Definition Multimedia Interface	Vysoko kvalitní multimediální rozhraní
HW	Hardware	Technické vybavení
IRC	Incremental Rotary Encoder	Inkrementální rotační snímač
IRQ	Interrupt Request	Žádost o přerušení
OS	Operating System	Operační systém
PWM	Pulse Width Modulation	Pulzně šířková modulace
RAM	Random Access Memory	Paměť s přímým přístupem
RPi	Raspberry Pi	Raspberry Pi
SSH	Secure Shell	Zabezpečený komunikační protokol
SW	Software	Programové vybavení
USB	Universal Serial Bus	Univerzální sériová sběrnice

C. SCHÉMA MOTORU



Obrázek C.1: Schéma motoru ze zdroje [16]



Obrázek C.2: Schéma motoru ze zdroje [15]