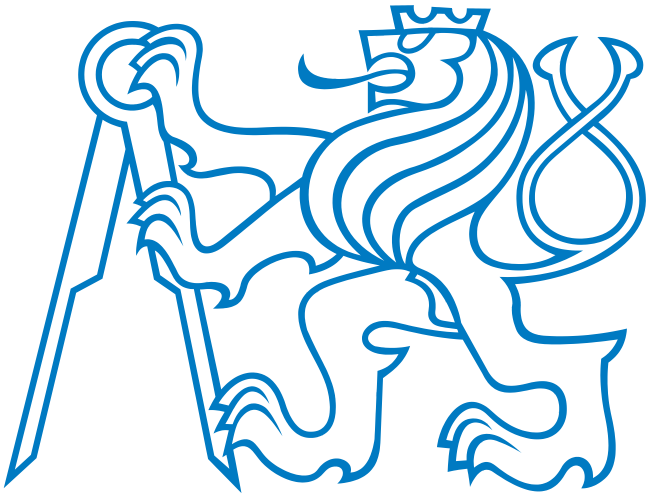
České Vysoké Učení Technické v Praze
Fakulta Elektrotechnická

BAKALÁŘSKÁ PRÁCE



Viktor Kajml

Řízení krokových motorů pro pohon mobilního robotu

Katedra kybernetiky
Vedoucí bakalářské práce: **Ing. Jan Chudoba**

Praha, 2011

Prohlášení

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

V Praze dne 27.6.2017

podpis

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Viktor K a j m l

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný

Obor: Kybernetika a měření

Název tématu: Řídicí jednotka krokových motorů

Pokyny pro vypracování:

1. Nastudujte problematiku řízení krokových motorů.
2. Navrhněte elektronické zapojení a kód mikroprocesoru pro novou jednotku podle pokynů vedoucího práce. Při návrhu berte ohled na zachování funkcí původních řídicích jednotek používaných na platformách G2BOT.
3. Vyrobte prototyp jednotky a proveďte testy její správné funkce. Výsledkem práce budou podklady pro sériovou výrobu jednotky.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Jan Chudoba

Platnost zadání: do konce zimního semestru 2011/2012

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 6. 1. 2011

Abstrakt

Tato bakalářská práce se zabývá problémem návrhu jednotky pro řízení krokových motorů pro použití v mobilní robotice. Cílem je dosažení funkčnosti srovnatelné s komerčně dostupnými jednotkami typu Pilot MC3410. V rámci toho jsou popsána řešení dílčích problémů, jako návrh hardware, software komunikačního protokolu a jejich implementace. Výsledkem práce je funkční prototyp jednotky a podklady pro jeho výrobu. Uvedeno je také krátké srovnání s jednotkami typu Pilot MC3410 na základě provedených experimentů.

Abstract

The thesis deals with the problem of designing a stepper motor control unit for use in the area of mobile robotics. The objective is to achieve a level of functionality comparable to commercially available units like Pilot MC3410. The whole design process is decomposed into several partial problems, such as designing and implementing hardware, software and a suitable communication protocol.The result of this thesis is a design of a functional prototype and documents necessary for it's manufacture. A short comparison with the Pilot MC3410 unit is also included.

Děkuji vedoucímu práce panu Ing. Janu Chudobovi a panu Ing. Ondřeji Fišerovi za pomoc při vytváření této práce.

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů

Kapitola 1

Úvod

Cílem této práce je vytvořit jednotku pro řízení dvou krokových motorů. Jednotka by měla být schopna komunikace s nadřazeným zařízením a disponovat jistou mírou autonomie nutnou k zajištění požadované funkčnosti. Návrh by měl zároveň zachovat následující vhodné vlastnosti původně používaných jednotek Pilot MC3410:

- měření a vyčítání teploty výkonových prvků
- měření a vyčítání hodnot proudu procházejícího vinutími motorů
- měření a vyčítání odometrických dat
- vysoká spolehlivost
- možnost budoucího rozšíření funkčnosti jednotky o ovládání stejnosměrných motorů

Podrobný popis jednotky Pilot MC3410 a jejího použití je možno nalézt v [8] a [9].

1.1 Teoretický úvod

Krokový motor je typ elektrického motoru, který je ovládán elektrickými impulzy a otáčí se nespojitě, po krocích. Princip fungování krokového motoru lze nejlépe demonstrovat na jednoduchém příkladu krokového motoru s permanentním magnetem a krokem o velikosti 90 stupňů provozovaném v unipolárním jednofázovém režimu. Rotor je tvořen permanentním magnetem a stator čtyřmi elektromagnety tak jak je zobrazeno na obrázku 1.1. Pokud aktivujeme jeden z elektromagnetů tím, že na něj přivedeme kladné napětí, dojde k tomu, že se k němu, působením vzniklého elektromagnetického pole, natočí rotor svým severním pólem. Otáčení motoru je tedy možné dosáhnout postupným aktivováním elektromagnetů ve správném pořadí. Plynulost chodu motoru bude záviset na schopnosti dodržet intervaly mezi přepnutím elektromagnetů. Rychlost chodu bude záviset na frekvenci přepínání. Pro správné řízení krokového motoru je tedy nutné generovat přesně definovanou sekvenci řídicích signálů. Uvedený příklad představuje pouze nejjednodušší způsob řízení krokového

motoru, existuje ještě mnoho dalších. V praxi nejvíce využívané způsoby řízení a jim odpovídající řídicí sekvence jsou uvedeny dále. Podrobný výklad fyzikálního principu práce krokového motoru je možné najít v [1].

Fyzickým provedením se od sebe motory liší hlavně provedením rotoru a velikostí kroku. Podle provedení rotoru se motory dělí do tří skupin:

1. s permanentním magnetem
2. s proměnnou reluktancí
3. hybridní

U první skupiny způsob provedení rotoru vyplývá už z názvu. Tvoří jej jednoduchý permanentní magnet. U motorů s proměnnou reluktancí je rotor vyroben z feromagnetického materiálu a má tvar připomínající ozubené kolo, kde počet zubů odpovídá velikosti kroku. Hybridní motory představují kombinaci dvou výše popsaných řešení. Rotor je tvořen axiálně uloženým permanentním magnetem a dvěma součástkami z feromagnetického materiálu, které jsou provedené jako rotor motoru s proměnnou reluktancí a přivařené na oba póly magnetu.

1.1.1 Pracovní režimy krokových motorů

Krokové motory lze provozovat v různých pracovních režimech, daných jejich zapojením a řídicí sekvencí.

Možnosti zapojení krokových motorů

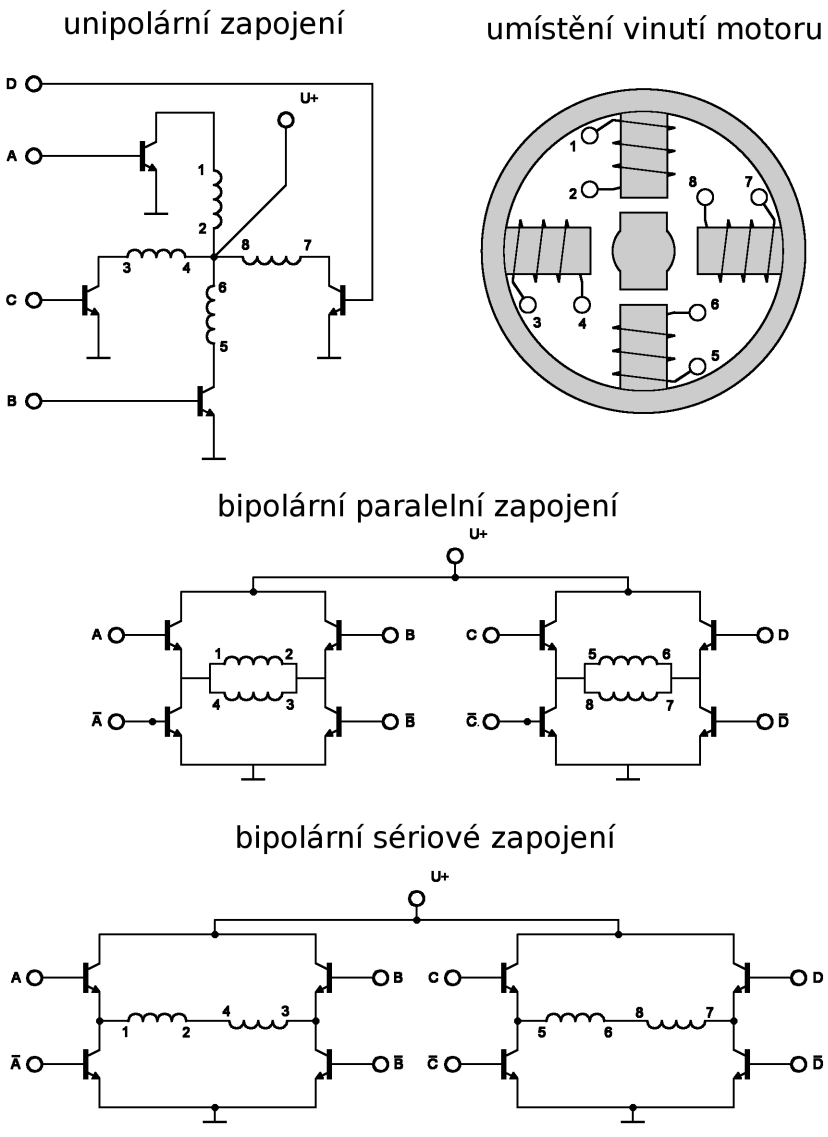
Většina krokových motorů má čtyři vinutí. Podle způsobu jejich zapojení je možné motor řídit buď v unipolárním, nebo bipolárním režimu tak, jak je naznačeno v obr. 1.1. Když v unipolárním zapojení prochází proud některou cívkou, tak k ní protilehlá cívka je bez proudu. V bipolárním zapojení dvěma protilehlými cívkami prochází vždy opačný proud. Bipolární zapojení tak oproti unipolárnímu nabízí vyšší výkon, neboť je najednou pod proudem vždy dvojnásobný počet cívek. Výhodami zapojení unipolárního je zase menší spotřeba a jednodušší realizace řídicího obvodu.

Řídicí sekvence pro ovládání krokových motorů

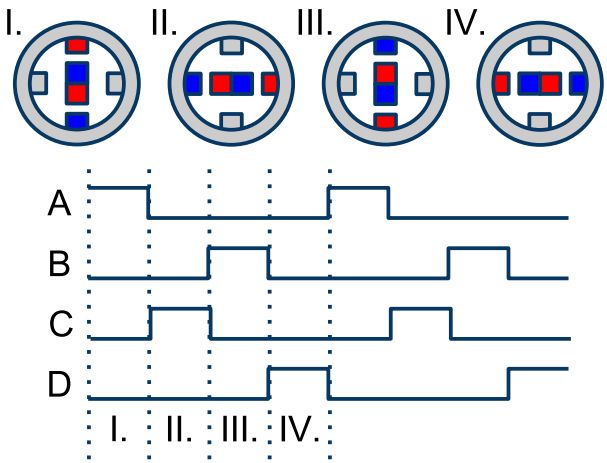
Následující režimy práce motoru je možné provozovat jak při unipolárním, tak i bipolárním zapojení. V ilustrativních obrázcích jsou červeně označena vinutí, která jsou protékána proudem v obou režimech a modře ty, která jsou protékána proudem pouze v bipolárním režimu.

Jednofázový režim (wave drive)

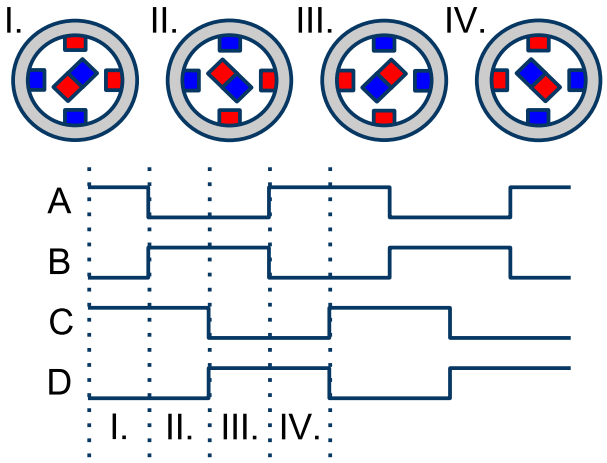
V tomto režimu je protékána proudem vždy pouze jedna cívka v unipolárním, nebo dvě v bipolárním zapojení. Neumožňuje tak dosáhnout plného jmenovitého výkonu motoru a proto se, s výjimkou aplikací, kde je důležité šetřit energií, moc nepoužívá. Řídící sekvence pro tento režim jsou znázorněny na obr. 1.2.



Obrázek 1.1: Možnosti zapojení a řízení krokového motoru



Obrázek 1.2: Řídící sekvence pro jednofázový režim



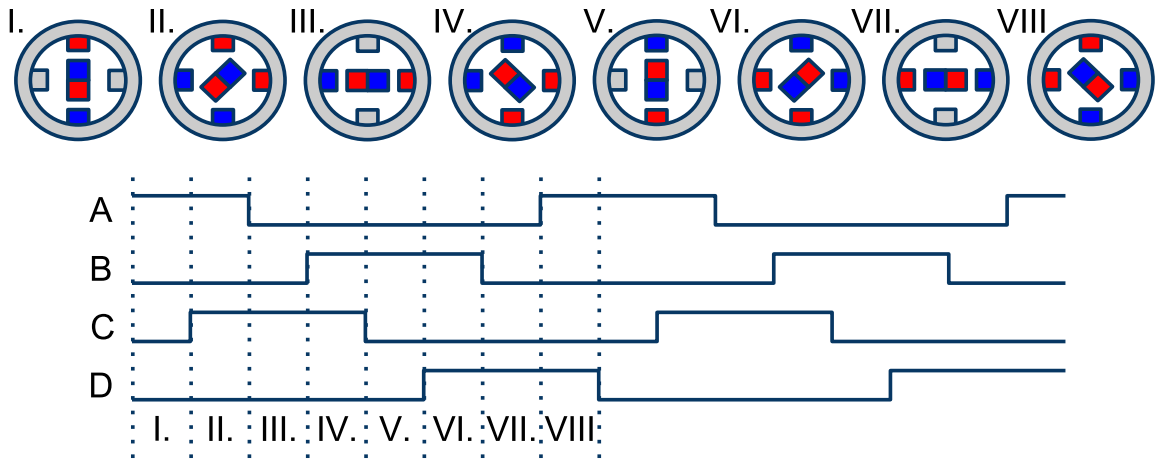
Obrázek 1.3: Řídící sekvence pro dvoufázový režim

Dvoufázový režim (full-step drive)

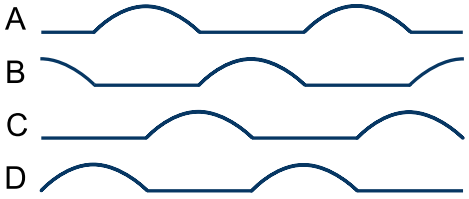
V tomto režimu je, jak je patrné z obr. 1.3, v jeden okamžik protékán proudem vždy dvojnásobný počet cívek než v jednofázovém režimu, tj. dvě pro unipolární zapojení a čtyři pro bipolární. Tak je zajištěno dosažení plného výkonu motoru. Jedná se asi o nejběžnější režim provozu krokových motorů.

Režim poloviční kroků (half-step drive)

Tento režim představuje kombinaci výše popsaných režimů. Sekvence řídicích signálů, vyobrazená na obr. 1.4, je vytvořena vzájemným proložením sekvencí odpovídajících jednofázovému a dvoufázovému režimu. Ve výsledku tedy na jeden krok připadá sekvence dvojnásobné délky, což znamená dvojnásobné zvýšení úhlového rozlišení motoru.



Obrázek 1.4: Řídící sekvence pro režim polovičních kroků



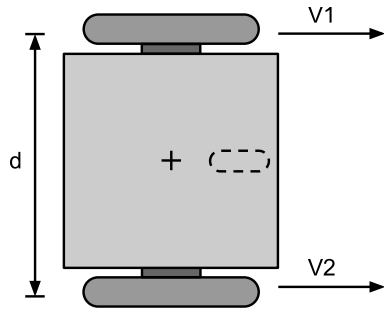
Obrázek 1.5: Řídící sekvence pro režim mikrokroků

Režim mikrokroků (microstepping)

V tomto režimu se pro řízení místo pravoúhlých impulzů používá spojitý signál, který se svým tvarem podobá harmonické funkci. Teoreticky je tak možné docílit zcela spojitého řízení motoru a nekonečného úhlového rozlišení. V praxi toho lze však dosáhnout pouze s obtížemi. Hlavní překážky představují nelinearity způsobené statickým třením a deformacemi elektromagnetického pole okolo zubů rotoru a pólu statoru. Používají se tedy signály, které danou funkci s určitou přesností aproximují. Podle rozlišení použitého signálu se potom mluví o režimu kroků čtvrtinových, osminových, šestnáctinových a tak dále. Hlavní nevýhoda je vyšší složitost generování takových řídících signálů. Problematika přesného ovládání krokových motorů v režimu mikrokroků je dobře popsána v [7]. Průběhy příslušných signálů jsou demonstrovány na obr. 1.5.

Při aplikaci pro pohon mobilních robotů mají krokové motory, v porovnání například s často využívanými stejnosměrnými motory, dvě hlavní přednosti. První je možnost ovládat motor bez nutnosti použití zpětné vazby a z toho vyplývající nižší složitost a cena realizace hardwarového a softwarového řešení. Druhou je schopnost velmi přesně měřit úhlovou polohu hřídele motoru, což vyplývá z diskrétní povahy otáčení krokových motorů. Oproti stejnosměrným motorům mají krokové motory i řadu nevýhod. Mezi ně patří: Větší rozměry a vyšší hmotnost, nutnost generovat složité řídící sekvence a při provozu v bipolárním režimu i nutnost použít dvojnásobný počet spínacích prvků.

Pro dodržení dostatečné přesnosti měření polohy při provozu krokového motoru



Obrázek 1.6: Koncepce robotu s diferenčním pohonem (pohled shora)

při jeho řízení v přímé vazbě, je však nutné zajistit, aby nedocházelo ke „ztrátě kroků“. K tomuto stavu dochází, když řídící jednotka pošle motoru příkaz k provedení kroku, ale k tomu z nějakého důvodu nedojde. Příčinou nejčastěji bývá náhlý nárůst zatížení motoru, například při moc prudkém zrychlení, nebo nárazu robotu do překážky. Při řízení v přímé vazbě není možné nijak ověřit, jestli k provedení kroku opravdu došlo a jestli odhad polohy, který s provedením kroku počítá, odpovídá realitě. Je proto nutné motor řídit tak, aby možnost ztráty kroku byla pokud možno co nejvíce omezena, což klade specifické nároky na návrh řídící jednotky.

1.1.2 Diferenční řízení mobilního robotu

Diferenční řízení, je v mobilní robotice velmi rozšířeným způsobem ovládání polohy robotu. Jeho koncepce je ilustrována na obrázku 1.6.

Pohyb robotu robotu lze ovládat změnou rychlosti otáčení dvou, po stranách umístěných a nezávisle poháněných, kol. Třetí kolo je nepoháněné, volně se otáčející a slouží pouze ke stabilizaci robotu. Hlavní výhodou diferenčního řízení je možnost otáčet se na místě a jednoduchost jak z hlediska technické realizace, tak i řízení. Závislost dopředné a úhlové rychlosti pohybu robotu na rychlosti kol lze popsat následujícími dvěma vzorci:

$$v = \frac{v_1 + v_2}{2} \tag{1.1}$$

$$\omega = \frac{v_1 - v_2}{d} \tag{1.2}$$

v_1 a v_2 jsou rychlosti kol, ω je úhlová rychlost robotu a v je jeho dopředná rychlost.

Způsob výpočtu polohy ze získaných odometrických dat při použití diferenčního řízení popisují následující vzorce:

$$\Delta s = \frac{\Delta s_1 + \Delta s_2}{2} \tag{1.3}$$

$$\Delta \phi = \frac{\Delta s_1 + \Delta s_2}{d} \tag{1.4}$$

$$\Delta x = \Delta s \cdot cos(\Delta \phi) \tag{1.5}$$

$$\Delta y = \Delta s \cdot sin(\Delta \phi) \tag{1.6}$$

$s_{1,2}$ jsou vzdálenosti ujeté jednotlivými koly, d je vzájemná vzdálenost obou kol, x a y jsou souřadnice robotu v rovině a ϕ je úhel jeho natočení.

Kapitola 2

Návrh řídící jednotky

Pro zajištění plné funkčnosti bylo potřeba navrhnou a sestrojit hardware řídící jednotky, navrhnout komunikační protokol mezi řídící a nadřazenou jednotkou, naprogramovat řídící jednotku, vytvořit programovou knihovnu pro ovládání řídicí jednotky nadřazenou jednotkou a napsat jednoduchý program, který ji bude implementovat.

Každý z těchto problémů bude dále popsán zvlášť.

Jednotka je navržena pro řízení motorů řady SX od společnosti Microcon a motorů SST58D společnosti Shinano. Oba typy motorů mají podobné parametry. Použití je ovšem možné se všemi typy krokových motorů se čtyřmi vinutími. Navíc je jednotka navržena tak, že pro použití s výše zmíněnými zamýšlenými typy motorů je mírně předimenzovaná. Tak je možno zajistit maximální spolehlivost a možnost použití i s motory vyšších výkonů. Důležité technické parametry motoru SX23 jsou uvedeny v tabulce č. 2.1.

Bylo rozhodnuto provozovat motory v bipolárním zapojení v režimu polovičních kroků. Tento režim poskytuje vysoký výkon, dostatečné úhlové rozlišení a poměrně malou složitost návrhu řídících obvodů. Dalším důvodem bylo to, že uspořádání výkonových spínacích prvků pro řízení motoru v tomto režimu ponechává možnost případného použití i pro řízení stejnosměrných motorů, což by při použití unipolárního zapojení nebylo možné.

Parametr		**Paralelní zapojení**	**Sériové zapojení**
statický moment	[Nm]	2	1
jmenovitý proud	[A]	2	1
indukčnost	[mH]	5	20
odpor	[Ω]	1,8	7,2
hmotnost	[kg]	0,7	
délka kroku	[°]	$1,8 \pm 0,1$	
počet vinutí		4	

Tabulka 2.1: Technické parametry krokového motoru SX23-1012D

2.1 Návrh hardware

Důležitou částí návrhu jednotky byl výběr vhodného mikroprocesoru, který představuje její centrální prvek. Zde byl s výhodou využit již existující návrh desky plošných spojů s 8-bitovým procesorem Atmel ATmega168, který je pro své vlastnosti a cenu pro dané využití velmi vhodný. Celá jednotka tak sestává z této desky s MCU napojené na dvě další desky, z nichž každá obsahuje obvody pro řízení jednoho krokového motoru. To s sebou přineslo zjednodušení dalšího návrhu a vyšší modularitu celé jednotky. Tyto přídavné desky budou dále označovány jako desky výkonové. Podrobný popis procesoru ATmega168 je uveden v [2].

Hlavní vlastnosti procesoru ATmega168:

- 16 Kbytů programovatelné systémové paměti
- 512 bytů EEPROM paměti
- 1 Kbyte interní SRAM
- 2x 8-bitový čítač
- 1x 16-bitový čítač
- 8mi kanálový 10-bitový ADC převodník
- programovatelné rozhraní USART
- programovatelný watchdog timer
- integrovaný analogový komparátor

2.1.1 Popis desky s MCU

Deska kromě mikroprocesoru obsahuje sériové komunikační rozhraní, stabilizátor napětí 5V, nebo 3,3 V, napěťovou referenci pro AD převodníky procesoru, opakovače pro přívod signálů k dvěma kanálům AD převodníku, obvod s krystalovým oscilátorem a konektory pro propojení s jinými deskami. Velikost napěťové reference lze nastavit změnou hodnot součástek R15 a R16, podle následujícího vzorce:

$$V_{ref} = \frac{R16}{R15 + R16} \cdot V_{cc} \tag{2.1}$$

Na desce osazený krystalový oscilátor má frekvenci 18.432 MHz. Stabilizátor napětí je nastaven na 5V, což je hodnota napětí, vyžadované některými součástkami na výkonových deskách.

Popis použitých portů je uveden v tab. 2.3. Podrobný popis, schéma zapojení, osazovací schéma a doporučená konfigurace jumperů jsou uvedeny v příloze A.

Parametr	Hodnota		Podmínky
R_{DSON}	3.5	Ω	$@V_{GS} = 10V$
V_{DSS}	30	V	
Qg	20	nC	
I_D	19	A	$@T_A = 70°C, V_{GS} = 10V$

Tabulka 2.2: Důležité parametry tranzistoru IRFH793

2.1.2 Návrh výkonové části

Dále bude popsán návrh rozšiřujících obvodů pro řízení krokových motorů, které jsou provedené na vlastní desce plošných spojů. Ta bude dále označována, jako deska výkonová.

Bylo rozhodnuto sestavit výkonovou část obvodu z diskrétních součástek. To s sebou přináší vyšší flexibilitu při návrhu výkonové části, kterou tak bylo možno navrhnout přesně na míru dané aplikaci. Dále se tak jednodušší přístup k součástkám při jejich případné výměně nebo opravách a diagnostice celé desky.

Jako spínací prvky H-můstků byly vybrány tranzistory typu NMOS pro jejich následující vhodné vlastnosti:

- rychlá doba spínání
- kladný teplotní koeficient
- malá závislost parametrů na teplotě
- řízení napětím

Z různých druhů NMOS tranzistorů byly vybrány součástky typu IRFH793, jako kompromis mezi cenou, parametry a dostupností. Jejich nejdůležitější parametry jsou uvedeny v tab. 2.2, podrobný popis v [3].

Fyzické umístění tranzistorů na desce bylo navrženou tak, aby ležely v jedné řadě, a tak na ně bylo možné jednoduše namontovat chladič.

Oba H-můstky jsou připojeny na zem přes snímací odpory o hodnotě 0.27Ω, který slouží k měření hodnoty proudu procházejícího přes celý můstek.

Nevýhodou při použití tranzistorů typu MOSFET je nutnost při spínání zajistit, aby napětí mezi výstupy gate a source bylo vyšší, než napětí mezi drain a source. Pro zajištění rychlého spínání je dále nutné zajistit, aby proud přivedený na gate měl dostatečnou velikost. Z těchto důvodu byly do návrhu zahrnuty budiče L6384. Jedná se o obvod s nábojovou pumpou, který dokáže podle signálů na svém vstupu řídit jednu polovinu H-můstku. Celkem jsou tedy na jedné výkonové desce osazeny čtyři. Výběr této součástky byl opět kompromisem mezi dostupností a cenou. Podrobný popis součástky je možno najít v [4].

2.1.3 Návrh logické části

Dále bylo nutné navrhnout způsob generování řídících signálů. Možnost signály generovat přímo na pinech procesoru byla zavržena pro přílišnou náročnost na výpočetní čas procesoru. Jako lepší řešení se jevilo použít některý z na trhu dostupných integrovaných obvodů určených pro řízení krokových motorů. Vybrán byl obvod L297. Ten na čtyřech výstupech A,B,C a D generuje řídicí sekvenci pro ovládání jednoho krokového motoru. Na každé výkonové desce tedy bude osazena jedna tato součástka. Frekvence výstupních signálů přímo závisí na frekvenci obdélníkového signálu přivedeného na vstup označený CLOCK. Obvod nabízí možnost generovat signály pro řízení motoru v jednofázovém režimu, dvoufázovém režimu a režimu polovičních kroků.

L297 dále obsahuje integrovaný chopper obvod, který slouží k udržení konstantního proudu tekoucího motorem a výkonovými prvky v jejich sepnutém stavu. Princip funkce chopper obvodu spočívá v porovnávání hodnot napětí na vstupech SENS1 a SENS2, s referenční hodnotou na vstupu VREF. Pokud je na jednom ze vstupů SENS naměřena hodnota vyšší než referenční, obvod generuje kladnou úroveň napětí na odpovídajícím výstupu INH1, nebo INH2, což způsobí uzavření celého výkonového můstku a tím pokles proudu, který jim protéká. Frekvence činnosti chopper obvodu je daná hodnotami součástek připojených na pin OSC, podle vzorce: $f_c = 1,45 \cdot R14 \cdot C10$, kde $R14 > 10k\Omega$. Pin SYNC slouží pro synchronizaci chopper obvodů u dvou a více různých součástek L297. Pokud u součástky osazen RC obvod připojený na pin OSC, funguje SYNC jako výstup se synchronizačním signálem. V opačném případě se SYNC chová jako vstup pro synchronizační signál přivedený z jiného čipu L297. Velikost reference je určena napěťovým děličem na pinu VREF.

Činnost celého čipu je možné vypnout přivedením záporné logické úrovně na vstup ENABLE.

Směr generované výstupní sekvence je určen logickou úrovní na vstupu $CW/\overline{CCW}$.

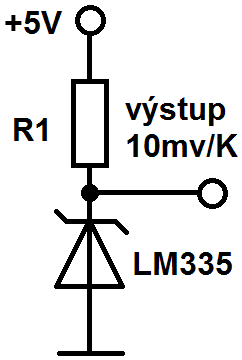
Podrobný popis součástky je možno najít v [5] a [6].

Na desce jsou dále umístěny dva operační zesilovače v neinvertujícím zapojeni. Na jejich vstupy je přivedeno napětí ze snímacích rezistorů. Výstupy jsou připojeny na ADC převodník v MCU přes konektory na portu C. Funkce těchto součástek je zesílit vstupní signál ze snímacích rezistorů tak, aby bylo co nejlépe využito rozlišeni AD převodníku. Pro maximální předpokládanou hodnotu proudu motorem se má signál na výstupu zesilovače zespodu blížit hodnotě napěťové reference MCU, což lze vyjádřit jako : $U_{ADC-max} = U_{ref}$, přičemž pro hodnotu $U_{ADC-MAX}$ platí:

$$U_{ADC-MAX} = I_{MAX} \cdot R2 \cdot \left(1 + \frac{R18}{R17}\right) \tag{2.2}$$

$$U_{ADC-MAX} = I_{MAX} \cdot R4 \cdot \left(1 + \frac{R20}{R19}\right) \tag{2.3}$$

Pro měření teploty jsou použity dva senzory LM335 s převodní charakteristikou 10mv /K. Schéma zapojení jednoho senzoru je na obrázku č. 2.1. Hodnota rezistoru je zvolena, jako $R1 = 500\Omega$, tak aby maximální proud procházející senzorem byl $I_{max} = 10mA$. Vstupy MCU na které jsou senzory připojeny jsou uvedeny v tabulce č. 2.3.



Obrázek 2.1: Schéma zapojení teplotního senzoru LM335

Port	Číslo pinu	Směr	Název	Popis
C1	23	IN	SENS-A	signály od snímacích odporů
C2	24	IN	SENS-B	
C3	25	IN	SENS-C	
C4	26	IN	SENS-D	
ADC6	19	IN	TEMP-A	signály od teplotních senzorů
ADC7	22	IN	TEMP-B	
D3	32	OUT	DIR-A	směr otáčení motoru
D4	1	OUT	CLK-A	frekvenční vstup pro L297
D5	2	OUT	ENABLE	enable signál pro výkonovou část
D6	9	OUT	CLK-B	frekvenční vstup pro L297
D7	10	OUT	DIR-B	směr otáčení motoru

Tabulka 2.3: Přehled použitých portů (směry signálů uváděné vzhledem k MCU)

Příslušná schémata výkonové desky jsou uvedena v příloze B.

2.2 Program MCU

Pro generovaní vstupních frekvenčních signálů pro obvody L297 jsou použity čítače 0 a 2 s rozlišením 8-bit. 16-bit čítač 1 je použit k měření času s nastavenou periodou 1ms. Čítače pracují v režimu Clear Timer on Compare Match, při kterém dochází při každém zvýšení hodnoty čítače k jejímu porovnání s nastavenou hodnotou stropu. Pokud se hodnoty rovnají, dojde k vynulování čítače a změně logické úrovně na příslušném výstupním pinu. Změnou nastavení před-děličky čítače a hodnoty jeho stropu je tak možné měnit frekvenci výstupního signálu. Proces vynulování čítače a změny logické úrovně na jeho výstupu bude dále v textu označován pro jednoduchost, jako jeden tik čítače. Popis všech vstupů a výstupů jednotky je uveden v tabulce 2.3.

Jednotka je navržena pro práci ve dvou režimech: Řízení na rychlost a řízení na polohu.

Při řízení na rychlost je cílem zajistit co nejpřesnější dodržení požadované rychlosti. K tomu je třeba přepočítat požadovanou rychlost na odpovídající hodnotu nastavení

čítačů generujících vstupní signály pro výkonovou část jednotky. Závislost rychlosti na nastavení čítače popisuje následující vzorec:

$$v = \frac{\pi d \frac{k}{360} f_k}{16C}[m/s] \tag{2.4}$$

d je průměr kol v m, k je velikost kroku motoru ve stupních, f_k je frekvence krystalového oscilátoru a C je počet strojových cyklů MCU, po kterém dojde k jednomu tiku čítače.

Pro nastavení správné rychlosti tedy musí MCU provést výpočet odpovídající následujícímu vzorci:

$$C = \frac{\pi d \frac{k}{360} f_k}{16v} \tag{2.5}$$

Je zřejmé, že pro správný výpočet je nutné přesně znát hodnoty konstant f_k, k a d. Přesné určení průměru použitých kol, ale není úplně jednoduché. Jednotku je tedy před použitím nutné zkalibrovat, což je podrobněji popsáno v kapitole 3.1.

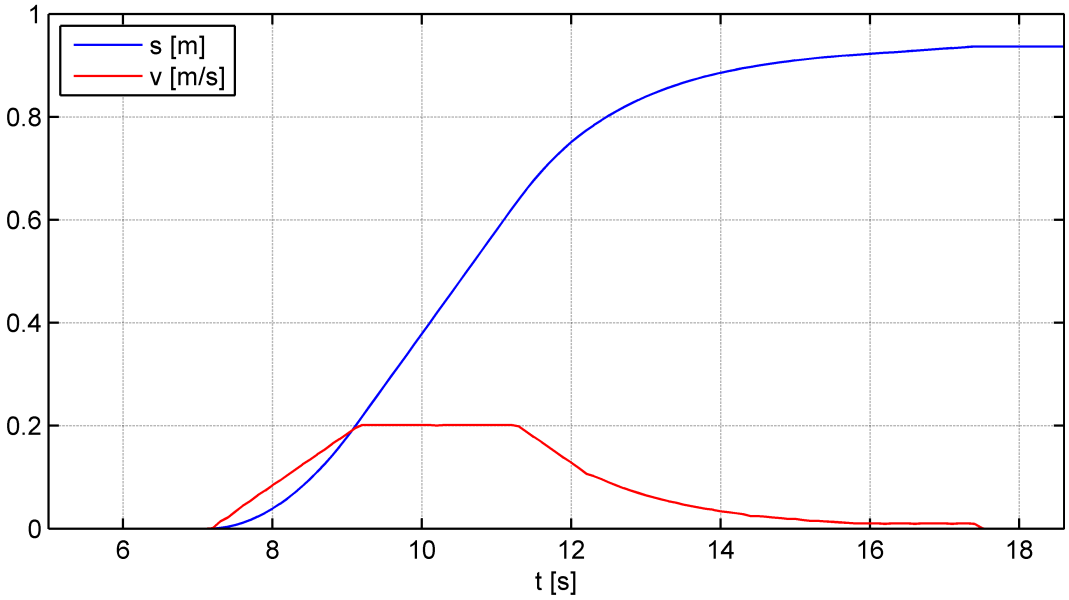
Pro zajištění plynulých rozjezdů a dojezdů tak, aby nedocházelo ke ztrácení kroků, je implementován jednoduchý regulátor, který zajišťuje dodržení nastavené hodnoty zrychlení. Uživatel nastavuje požadovanou rychlost a regulátor poté v definovaných časových intervalech příslušným způsobem mění hodnotu aktuální rychlosti.

Při řízení na polohu, je cílem zajistit, aby ovládaný robot co nejpřesněji ujel požadovanou vzdálenost. To je realizováno pomoci jednoduchého P-regulátoru, jehož vstupem je zadaná vzdálenost a výstupem je požadovaná rychlost pro výše popsaný regulátor rychlosti. Jako zpětná vazba jsou použita naměřená odometrická data. Takový systém má sám o sobě integrační charakter, takže k zajištění nulové ustálené odchylky není třeba implementovat I složku regulátoru. Regulátor pracuje s dráhou zadanou v ticích čítače. Přepočet z reálných jednotek je ponechán na nadřazeném zařízení. Dráhu v m/s připadající na jeden tik čítače lze vyjádřit:

$$s = \frac{\pi d \frac{k}{360}}{16C}[m] \tag{2.6}$$

s je ujetá dráha v m, d je průměr použitých kol v m, k je velikost kroku motoru ve stupních a C je počet tiků čítače generujícího řídící signál pro příslušný motor. Opět je zřejmé, že pro správný výpočet je nutné co nejpřesněji znát průměr použitých kol. Průběh rychlosti a dráhy naměřený při řízení na polohu pro požadovanou ujetou vzdálenost 10000 tiků je uveden na obr. č. 2.2.

Z hlediska přesnosti by bylo pro výpočty nejlepší použít datové typy s plovoucí desetinou čárkou. Procesor ATMega168 je však pouze 8-bitový a čísla s plovoucí desetinou čárkou typicky zabírají 32 bitů. Jejich použití by tak bylo velmi náročné na výpočetní čas. Bylo tedy rozhodnuto použít fixed-point aritmetiku využívající celočíselné datové typy, což s sebou přineslo i nutnost používat a definovat vhodné strojové jednotky. Jejich popis spolu se vztahy pro jejich převod na běžné jednotky, je uveden v tabulce č .2.4. Velikost jednotek byla volena tak, aby bylo možno většinu výpočtů provádět s 16-bitovými čísly a přitom byl zachován dostatečný rozsah použitelných hodnot, jakož i dostatečné rozlišení.



Obrázek 2.2: Průběh rychlosti a dráhy při řízení na polohu

Veličina	**běžná jednotka**	$\frac{\textbf{s.jednotka}}{\textbf{b.jednotka}}$	$\frac{\textbf{b.jednotka}}{\textbf{s.jednotka}}$
rychlost	m/s	$2 \cdot 10^{-}5$	50000
čas	s	$1 \cdot 10^{3}$	1000
zrychlení	m/s^2	$2 \cdot 10^{-}2$	50

Tabulka 2.4: Přehled strojových jednotek

2.2.1 Popis hlavní smyčky programu

Program pro MCU má podobu nekonečné smyčky, kde jsou cyklicky spouštěné jednotlivé funkce. Při vykonávání jedné funkce nesmí dojít k tom, že program čeká na dokončení nějaké akce nebo na nějakou vnější událost, protože tak by se zamezilo vykonávání ostatních, následujících funkcí. Při použití přerušení je také důležité zajistit, aby procesor zůstával ve stavu přerušení co možná nejkratší možnou dobu, aby nenastal stav, kdy nestíhá přerušení zpracovávat.

Struktura hlavní smyčky je naznačena ve výpisu č. 2.1. Podrobnější popis jednotlivých funkcí je uveden dále.

Po každém restartu MCU, před prvním spuštěním celé smyčky dojde k počátečnímu nastavení výstupů, čítačů a sériového rozhraní prostřednictvím funkce `init()`.

Jako první je v hlavní smyčce obslouženo vyčítání hodnot AD převodníkem. Použito je celkem 6 kanálů AD převodníku. Dva jsou určeny pro zpracování výstupů teplotních čidel a zbylé 4 slouží pro měření hodnoty napětí na snímacích rezistorech H-můstků. Funkce `ConversionFinished()` kontroluje, jestli právě probíhá AD konverze. Pokud je konverze už dokončena, je zavolána funkce `StartNewConversion()`, která zpracuje získaná data a poté spustí novou konverzi. Za účelem omezení rušivých vlivů se vždy na jednom kanále provádí 10 odměrů za sebou a dále se pracuje s je-

```
int main()
        init();
        while(1){
                if(ConversionFinished())
                        StartNewConversion();
                if(mainTimer > 0){
                        if(Enabled()){
                                PositionControl();
                                AdjustSpeed():
                                SetSpeed();
                        }
                        UpdateCounters();
                        mainTimer = 0;
                }
                ServiceCommunication();
        }
}
```

Výpis 2.1: Hlavní smyčka programu MCU

jich průměrem. Po výpočtu průměru je výsledek také porovnáván s nastavenými mezními hodnotami měřených veličin. Pokud je zjištěno jejich překročení, MCU vypne výkonovou část a odešle příslušnou chybovou zprávu. Po deseti odběrech je převodník přepnut na další kanál v pořadí.

Dále je proveden blok funkcí zajišťujících úkony, které je potřeba vykonávat v pravidelných časových intervalech. Načasování je zajištěno pomoci proměnné mainTimer, která je inkrementována každých 10 ms, kdy je čítačem číslo 1 generováno přerušení. Na konci tohoto bloku funkcí je proměnná zase vynulována, takže kontrolou její hodnoty je možné zjistit, jestli je už na čase blok funkcí znovu spustit.

Pokud je zapnuta výkonová část jsou spuštěny funkce `PositionControl()`, `AdjustSpeed()` a `SetSpeed()`, které vykonávají vlastní řízení motorů.

Funkce `PositionControl()` implementuje P regulátor zajišťující řízení na polohu. Pokud je nastaven režim řízení na rychlost funkce není spuštěna.

Účelem funkce `AdjustSpeed()` je úprava hodnoty aktuální rychlosti podle nastavených hodnot požadované rychlosti a zrychlení.

Funkce `SetSpeed()` nakonec podle hodnoty aktuální rychlosti provede nastavení čítačů generujících vstupní frekvenční signály pro výkonové desky, podle vzorce č. 2.5.

`UpdateCounters()` zajišťuje funkce pravidelného vyčítání odometrie, teplotních dat a hodnot proudů protékajících výkonovou částí. Dále se také stará o vypnutí výkonové části v případě, že od obdržení poslední přijaté zprávy od nadřazené jednotky uběhl delší než nastavený časový interval. To je žádoucí z hlediska bezpečnosti, pokud by došlo k poruše komunikace.

Nakonec je spuštěna funkce `ServiceCommunication()` která zajišťuje, jak je zřejmé z jejího názvu, obsluhu komunikace. Při jednom spuštění funkce vždy zpracuje

Název	Decimální hodnota	Hexadecimální hodnota
END	192	D0
ESC_END	219, 220	EB, EC
ESC_ESC	219, 221	EB, ED

Tabulka 2.5: Definované sekvence znaků protokolu SLIP

jeden byte z fronty příchozí komunikace, provede jeho dekódování z protokolu SLIP a uloží jej do svého zásobníku. Pokud funkce při zpracovávání dat narazí na na byte s hodnotou SLIP_ESC, tak provede kontrolu dat uložených na svém zásobníku. Pokud je zjištěno, že nejedná o platnou zprávu, je zásobník vynulován. Pokud jsou data platná, je spuštěna příslušná obslužná funkce, podle typu zprávy, odeslána odpověď nadřazené jednotce a nakonec je opět vynulován zásobník, aby bylo možno pokračovat v obsluze další komunikace.

2.3 Návrh komunikačního protokolu

Pro komunikaci s nadřazenou jednotkou bylo třeba navrhnout komunikační protokol. Počítá se pouze s použitím komunikace typu bod-bod přes sériové rozhraní, takže protokol je ve výsledku poměrně jednoduchý.

Při návrhu bylo rozhodnuto využít prvky protokolu SLIP, jehož použití je v oblasti komunikace s MCU velmi rozšířené.

2.3.1 Popis protokolu SLIP

Protokol SLIP definuje speciální znak END, ten se odesílá na konci rámce a slouží k jeho označení. Pokud je potřeba odeslat datový byte jehož hodnota odpovídá znaku END, je nahrazen pomoci dvoubytové sekvence ESC_END. Pokud hodnota odesílaného datového bitu odpovídá prvnímu bitu ze sekvence ESC_END, je nahrazen sekvencí ESC_ESC. Ve finálním návrhu je využita varianta protokolu, kde se znak END zařazuje i na začátek rámce kvůli větší odolnosti proti přijmu bytů způsobených šumem na vedení.

2.3.2 Struktura rámce

Každá zpráva začíná jednobytovou hlavičkou, která určuje o jaký typ zprávy se jedná. Dále mohou následovat příslušná data. Zpráva je vždy ukončena jednobytovým kontrolním součtem, který je realizován jednoduše, jako exklusivní bitová disjunkce přes všechny byty zprávy. Formát rámce je ilustrován na obr. 2.3. MCU se, až na výjimky, chová z hlediska komunikace pasivně, jako zařízení typu slave a pouze odpovídá na příkazy nadřazené jednotky, která se chová jako zařízení typu master.

Definované zprávy lze rozdělit do následujících čtyř skupin:

- zprávy GET

8-bit	proměnná délka	8-bit
HEADER	DATA	CHECKSUM

Obrázek 2.3: Struktura rámce komunikačního protokolu

Název	**Hodnota**	**Význam**
OVERWORKED	33	MCU nestíhá včas dokončit všechny výpočty
DISABLE_TEMP	34	Výkonová část vypnuta kvůli překročení povolené hodnoty teploty
DISABLE_CURRENT	35	Výkonová část vypnuta kvůli překročení povolené hodnoty proudu
DISABLE_TIME	36	Výkonová část vypnuta kvůli dlouhodobé nečinnosti
ROUTE_FINISHED	37	Byla ujeta nastavená vzdálenost

Tabulka 2.6: Definované typy událostí

- zprávy SET
- události
- příkazy

U zpráv SET dochází k přenosu dat od nadřazené jednotky k jednotce MCU, která pouze potvrzuje přijetí zprávy. Jedná se například o příkazy sloužící ke změně rychlosti nebo nastavení konstant MCU.

Zprávy GET slouží k přenosu dat opačným směrem. Nadřazená jednotka posílá žádost o zaslání dat a MCU odpovídá příslušnými daty. Příkladem zprávy GET může být žádost o vyčtení naměřených dat nebo kontrola nastavení MCU. Ke každé zprávě SET existuje i odpovídající zpráva GET. Naopak to ale platit nemusí. Příkladem může být žádost o vyčtení aktuální hodnoty proudu motorem, kde by zpráva SET ani neměla smysl.

Třetím typem zpráv jsou zprávy typu událost, kterými MCU signalizuje nadřazené jednotce vznik nějaké nečekané události. V tomto případě je tedy porušen koncept master-slave, protože podřízená jednotka vysílá bez vyzvání. Nadřazená jednotka musí být připravena tyto neočekávané zprávy zpracovat. Jedná se například o zprávy informující o vypnutí výkonové části jednotky z důvodů překročení povolených hodnot teploty, nebo proudu. Definované události jsou uvedeny v tab. 2.6.

Jako příkazy jsou označovány zprávy vysílané nadřazenou jednotkou, u kterých nedochází k přenosu dat ani v jednom z obou směrů. Všechna potřebná informace je obsažena v hlavičce zprávy. Představiteli tohoto typu zpráv jsou příkazy nařizující MCU uložení programových konstant do paměti EEPROM, nebo jejich zpětné načtení.

2.4 Návrh komunikační knihovny

Součástí zadání bylo i vytvoření knihovny funkcí pro zjednodušení komunikace s MCU. Bylo rozhodnuto knihovnu napsat pro použití v operačním systému Linux, pro jednoduchost s jakou je v něm možné pracovat s ovladači pro sériovou komunikaci. Knihovna byla napsána v jazyce C.

2.4.1 Popis použití knihovny

Před tím, než je možné začít funkce z knihovny používat, je nutné vytvořit instanci struktury `mc_data`. Ta slouží k uchování globálních proměnných, potřebných pro ostatní funkce. Odkaz tuto instanci je potom potřeba předávat všem funkcím z knihovny jako první parametr.

Dále se musí zavolat funkce `mc_init()` a jako parametr ji předat adresu souboru, který odpovídá komunikačnímu portu, na kterém si uživatel přeje zahájit komunikaci. Funkce `mc_init()` provede otevření portu a jeho konfiguraci. Dále vytvoří vlákno, určené ke čtení a zpracovávání příchozích dat. Nakonec se provede test komunikace s MCU, tím že se z něj vyčte jeho aktuální nastavení, což slouží i k inicializaci proměnných v datové struktuře `mc_data()`.

Po skončení práce s knihovnou se musí zavolat funkce `mc_quit()`, která uvolní alokovanou paměť a ukončí běžící vlákna.

Knihovna nabízí funkce pro odeslání všech typů zprávy definovaných protokolem popsaným v kapitole 2.3. Funkce se liší počtem parametrů. U funkcí odpovídajících zprávám GET jsou parametry ukazatele na návratové proměnné. U zpráv SET odpovídají parametry nastavovaným veličinám a jsou zpravidla typu float. Při odesílání funkce typicky provede konverzi dat ze zadaných reálných jednotek na strojové jednotky MCU, sestavení zprávy, přidání kontrolního součtu, zakódování protokolem SLIP a nakonec její odeslání. Potom je vlákno, ze kterého byla funkce zavolána zablokováno a po nastavenou dobu čeká na odpověď nebo potvrzení přijetí zprávy od MCU. Pokud vyprší čas nebo dojde v průběhu funkce k jiné chybě, je vrácena chybová hodnota. V případě, že je vše v pořádku, je vrácena 0.

Pro obsluhu neočekávaných zpráv je uživateli poskytnuta možnost nastavit v datové struktuře ukazatel na jím implementovanou obslužnou funkci. Ta se spustí při výskytu neočekávané události a bude ji předán parametr identifikující druh nastalé události.

Obsluha zpráv pravidelného vyčítání teploty, proudu a odometrie je řešena analogicky. Pro každou vyčítanou veličinu je možné nastavit jednu uživatelskou funkci, které budou předána příslušné hodnoty.

Podrobný popis všech funkcí knihovny je uveden v příloze C.

2.4.2 Příklad použití knihovny

Výpis č. 2.2 obsahuje kus kódu představující jednoduchý program napsaný v jazyce C, který využívá komunikační knihovnu. Cílem programu je řídit mobilní robot tak, aby čtyřikrát za sebou projel čtvercovou dráhu.

Napřed jsou deklarovány a inicializovány všechny potřebné konstanty a proměnné. Dále je vytvořena datová struktura komunikační knihovny, které jsou poté předány příslušné proměnné. V hlavní smyčce programu je jednotce poslán příkaz k ujetí vzdálenosti nebo otočení na místě. Po jeho odeslání program čeká, dokud neobdrží od jednotky zprávu o dokončení požadovaného manévru. Celý proces se opakuje, dokud není proveden nastavený počet opakování. Před ukončením programu je odeslán příkaz k vypnutí výkonové části jednotky a pomoci funkce `mc_quit` jsou uvolněny programové prostředky využívané komunikační knihovnou.

```
#include "mclib.h"

int finished = 0;
void EventHandler(int val){
        if(val == ROUTE_FINISHED)
                finished = 1;
}

int main(int argc, char* argv[])
{
        char* filepath = "/dev/ttyUSB0";
        float accel            =     0.1;
        float maxSpd           =     0.3;
        float pConst           =     9.0;
        float ticksPerMeter    = 10677.33704;
        float ticksPerRotation = 12508.23;
        int reps     = 4;
        float scale  = 1.5;

        struct mc_data data;
        mc_init(filepath);
        mc_set_accel(&data,accel);
        mc_set_max_speed(&data,maxSpd);
        mc_set_pconst(&data,pConst);
        mc_set_ticks_per_meter(&data,ticksPerMeter);
        mc_set_ticks_per_rotation(&data,ticksPerRotation)
           ;
        mc_set_event_callback(&data,&EventHandler);
        mc_set_enable(&data,0);
        int i;
        for(i=0;i<reps;i++){
                finished = 0;
                mc_set_distance(data,scale,scale);
                while(!finished)
                        ;
                finished = 0;
                mc_turn(data,90.0);
                while(!finished)
                        ;
        }
        mc_set_enable(data,0);
        mc_quit(&data);
        return 0;
}
```

Výpis 2.2: Ukázka použití komunikační knihovny

Kapitola 3

Experimenty

3.1 Kalibrace jednotky

Jak již bylo zmíněno, přesnost jednotky při řízení na polohu závisí velmi citlivě na fyzické konfiguraci použitého mobilního robotu. Konkrétně se jedná o průměr použitých kol a jejich vzájemnou vzdálenost. Samotná řídící jednotka z důvodů zjednodušení výpočtů místo těchto parametrů pracuje pracuje s parametry, které jsou od nich odvozené a které byly nazvány *konstanta vzdálenosti* a *konstanta otáčení*. Konstanta vzdálenosti je počet tiků čítače připadajících na metr ujeté vzdálenosti. Konstanta otáčení je počet tiků připadajících na otočení robotu o 360 stupňů. Za účelem zjištění správných hodnot těchto konstant, byla pro každou z nich provedena jedna série měření. Všechny naměřené hodnoty jsou uvedeny v příloze D.

3.1.1 Měření konstanty vzdálenosti

Pro zjištění konstanty vzdálenosti bylo provedeno celkem 90 měření, pro různé hodnoty maximální rychlosti a zrychlení. Měřené hodnosti rychlosti byly $v = 0.1$, 0.2, 0.3 m/s a zrychlení $a = 0.1$, 0.2, 0.3 m/s^2. To umožnilo, kromě nalezení správné hodnoty konstanty vzdálenosti, také ověřit případný vliv velikosti rychlosti a zrychlení na její velikost. Pro každou z devíti možných kombinací těchto hodnot bylo provedeno deset měření.

Při každém z nich byl robot manuálně řízen tak, aby ujel předem vyměřenou vzdálenost 6m. Na konci dráhy byla vyčtena odometrická data udávající, kolik bylo za celou dráhu naměřeno tiků. Přesnost měření vzdálenosti byla odhadem $\pm 1mm$. Měření bylo prováděno na linoleovém povrchu. V tabulce č. 3.1 jsou uvedeny hodnoty konstanty vzdálenosti pro různé kombinace hodnot rychlostí a zrychlení, vypočítané jako průměr ze všech deseti měření provedených prodané hodnoty. Z naměřených výsledků není patrné, že by konstanta vzdálenosti nějakým výrazným způsobem závisela na hodnotě maximální rychlosti, nebo zrychlení. Pro další měření byla použita konstanta vzdálenosti , vypočítaná jako aritmetický průměr všech naměřených výsledků, o hodnotě:

$$K_v = 10677 \quad tik/metr \tag{3.1}$$

	$\mathbf{v_{max} = 0.1m/s}$	$\mathbf{v_{max} = 0.2m/s}$	$\mathbf{v_{max} = 0.3m}/s$
$\mathbf{a = 0.1m/s^2}$	10677	10682	10680
$\mathbf{a = 0.2m/s^2}$	10675	10676	10677
$\mathbf{a = 0.3m/s^2}$	10677	10676	10677

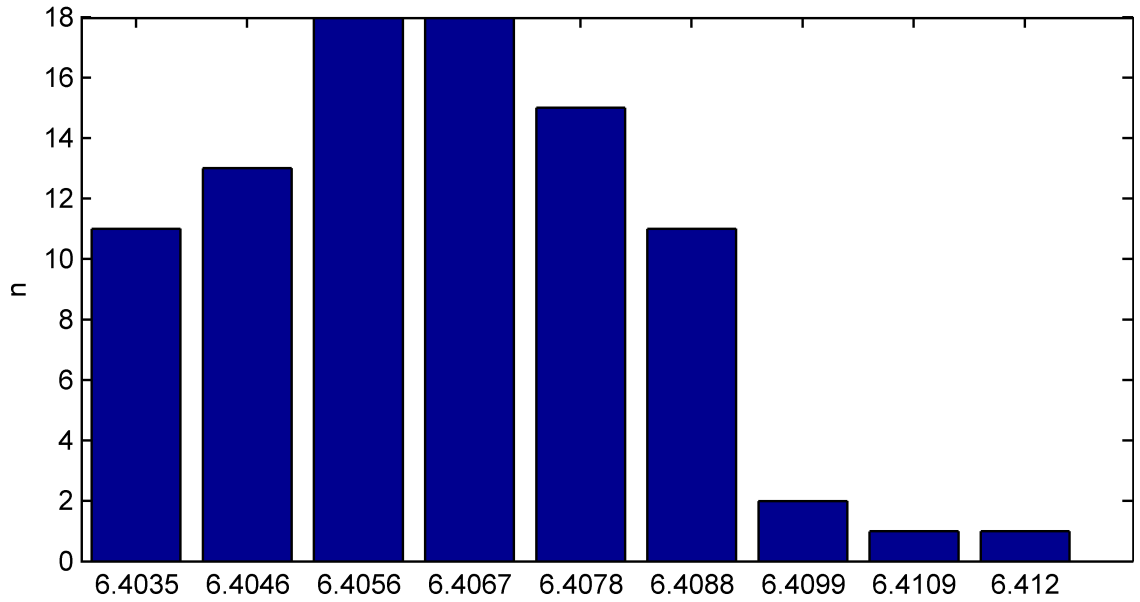
Tabulka 3.1: Naměřené hodnoty konstanty vzdálenosti pro různé rychlosti a zrychlení (v ticích na metr)

Směrodatná odchylka a rozptyl byly určeny jako:

$$\sigma^2 = 9.5 \quad tik/metr^2 \tag{3.2}$$

$$\sigma = 3.1 \quad tik/metr = 0.03\% \tag{3.3}$$

Rozložení naměřených hodnot je zobrazeno v histogramu v obr. 3.1.



Obrázek 3.1: Histogram naměřených hodnot konstanty vzdálenosti

3.1.2 Měření konstanty otáčení

Měření konstanty otáčení probíhalo obdobným způsobem. Robot byl manuálně ovládán tak, aby se otočil desetkrát kolem dokola. Měření bylo provedeno pouze pro jednu hodnotu maximální rychlosti a zrychlení $a = 0.2m/s^2, v = 0.2m/s$ a bylo opakováno desetkrát. Hodnota konstanty otáčení, vypočítaná jako aritmetický průměr všech naměřených hodnot a byla určena jako:

$$K_o = 12508.23 \quad tik/2\pi. \tag{3.4}$$

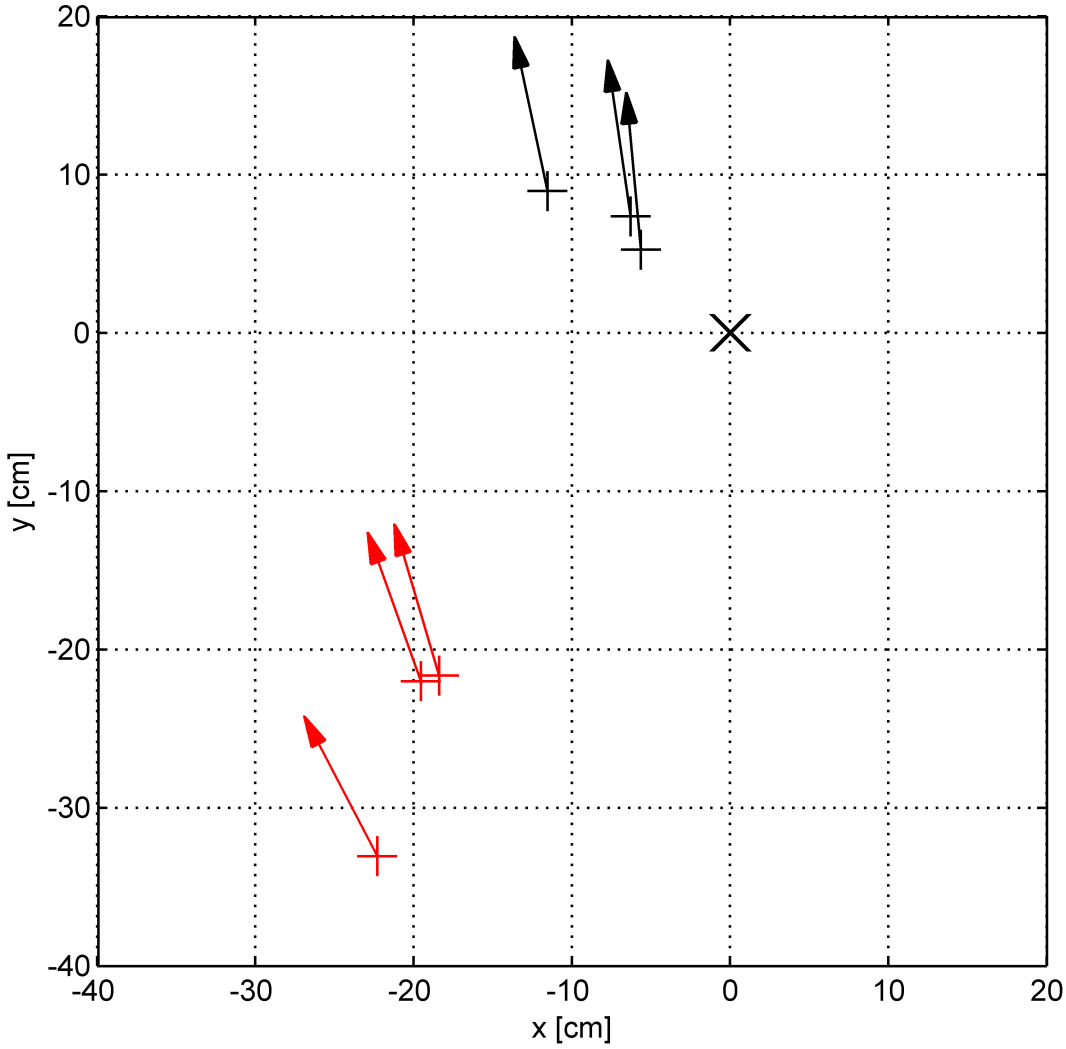
Směrodatná odchylka a rozptyl byly určeny jako:

$$\sigma^2 = 22.3 \quad (tik/2\pi)^2 \tag{3.5}$$

$$\sigma = 4.7 \quad tik/2\pi = 0.04\% \tag{3.6}$$

3.2 Měření přesnosti řízení na polohu

Za účelem ověření přesnosti řízení na polohu byla provedena série měření, při kterých robot opakovaně projížděl dráhu ve tvaru čtverce o straně 1m. Po jejím ujetí byla změřena odchylka robotu od startovní polohy a jeho natočení. Bylo provedeno celkem šest měření. U prvních třech robot projížděl dráhu desetkrát za sebou po směru hodinových ručiček, u ostatních tří proti směru. Robot tak tedy při každém měření ujel dráhu o délce 40m a otočil se celkem o 3600°. Výsledky jsou zpracované v tab. 3.2. a obr. 3.2. Hodnoty naměřené při jízdě robotu po směru hodinových ručiček jsou v obrázku označeny červeně.



Obrázek 3.2: Výsledky měření přesnosti řízení na polohu

Číslo měření	**Směr otáčení**	**Δx[cm]**	**Δy[cm]**	**Δϕ[rad]**	**Δϕ[°]**
1.	+	-6.3	7.4	0.145	8.3
2.	+	-5.7	5.3	0.093	5.3
3.	+	-11.6	9.0	0.209	12.0
4.	-	-18.4	-21.7	0.290	16.6
5.	-	-19.6	-22.0	0.339	19.4
6.	-	-22.3	-33.1	0.482	27.6

Tabulka 3.2: Výsledky měření přesnosti řízení na polohu

Směrodatná odchylka a rozptyl polohy byly určeny jako:

$$\sigma_{XY}{}^2 = 13.73 \cdot 10^{-3} m^2 \tag{3.7}$$

$$\sigma_{XY} = 0.117m = 0.29\% \tag{3.8}$$

Směrodatná odchylka a rozptyl úhlu byly určeny jako:

$$\sigma_{\phi}{}^2 = 16.7 \cdot 10^{-3} rad^2 \tag{3.9}$$

$$\sigma_{\phi} =).129rad = 0.21\% \tag{3.10}$$

U všech provedených měření byly zjištěny záporné hodnoty odchylek souřadnice x a kladné hodnoty odchylek úhlu. Velikosti směrodatných odchylek úhlu a polohy jsou také přibližně o řád větší, než velikosti směrodatných odchylek získaných při předchozích dvou kalibračních měřeních. Z toho lze usuzovat, že se jedná o projev nějaké, dříve nezaznamenané chyby v řídící jednotce, která způsobuje to, že robot zatáčí doleva více než by měl. Příčinu této chyby se bohužel, z časových důvodů, před termínem odevzdání této práce nepodařilo odhalit.

Kapitola 4

Závěr

Cílem této práce bylo vytvořit jednotku pro řízení krokových motorů mobilního robotu splňující požadavky popsané v kapitole 1. V rámci toho byl navržen hardware jednotky a sestaven její funkční prototyp. Dále bylo navrženo a implementováno její softwarové vybavení a navržen protokol pro komunikaci s nadřazeným zařízením. Pro usnadnění komunikace na straně nadřazené jednotky byla vytvořena knihovna funkcí v jazyce C. Jednotka je schopná nezávisle ovládat dva krokové motory v režimech řízení na rychlost a na polohu, měřit a vyčítat odometrická data, teplotní data a hodnoty proudů protékajících jejími výkonovými prvky a disponuje jistou mírou autonomie pro zajištění bezpečného provozu.

Z časových důvodů bohužel nebylo možné všechny funkce jednotky důkladně otestovat.

V průběhu práce s prototypem bylo zjištěno, že použité budiče L6384 vyžadují pro svůj provoz napětí okolo 13V, což je pro dlouhodobý provoz s napájením z baterií (použity jsou hermeticky uzavřené olověné akumulátory) nevhodné. Při dalším vývoji jednotky tedy bude nutné se na tento problém zaměřit. Vhodným řešením by mohlo být zahrnutí nábojové pumpy do výkonové části jednotky. Možné je také použití jiných typů budičů, například součástek IR2104, které jsou velmi podobné nyní používaným součástkám, ale mají jinou konfiguraci výstupů, takže je nebylo možné osadit na stávající desku plošných spojů.

Při návrhu se podařilo implementovat všechny důležité vlastnosti dříve používané jednotky Pilot MC3410, které byly vybrané na začátku návrhu. V porovnání s ní je přesnost řízení na polohu nové jednotky nižší. Cílem pro další návrh by tedy mělo být její zvýšení a identifikace a odstranění chyb, které nepřesnost způsobují. Dále pak bude implementovat doplňkové vlastnosti, jako možnost řízení po křivkových drahách, nebo výběr z více řídících regulátorů, aby bylo dosaženo funkčnosti plně srovnatelné s jednotami typu Pilot.

Literatura

[1] VOŽENÍLEK, Petr; JANOUŠEK, Josef; PETRÁSEK, František. *Základy silnoproudé elektrotechniky*. 1. vydání. Praha: Vydavatelství ČVUT, 2005. ISBN 80-01-03135-7.

[2] *ATmega168 Datasheet* [online]. verze 36.19 Rev. 2545B-01/0. [citováno 22.5.2011] URL: <http://www.atmel.com/dyn/resources/prod_documents/doc2545.pdf>

[3] *IRFH7934 Datasheet* [online]. poslední úprava - 11.2.2009. [citováno 22.5.2011] URL: <http://www.irf.com/product-info/datasheets/data/irfh7934pbf.pdf>

[4] *L6384 Datasheet* [online]. poslední úprava - květen 2000. [citováno 22.5.2011] URL: <http://www.st.com/stonline/books/pdf/docs/5651.pdf>

[5] *L297 Stepper Motor Controllers* [online]. poslední úprava - prosinec 2001. [citováno 22.5.2011] URL: <http://www.st.com/stonline/books/pdf/docs/1334.pdf>

[6] *AN470 Application Note: The L297 Stepper Motor Controller* [online]. URL: <http://www.st.com/stonline/books/pdf/docs/1734.pdf> poslední úprava - listopad 2003. [citováno 22.5.2011]

[7] JONES, Douglas W. *Control of Stepping Motors: A Tutorial* [online]. poslední úprava - 22.12.2010. [citováno 22.5.2011] URL: <http://www.cs.uiowa.edu/~jones/step/>

[8] *Pilot^{TM} Motion Processor: MC3410 Single Chip Technical Specifications for Microstepping Motion Control* [online]. poslední úprava - 1.6.2004. [citováno 20.5.2011] URL: <http://pmd.bentech-taiwan.com/PMD/MC3410_Technical_Specifications.pdf>

[9] *Pilot^{TM} Motion Processor: User's Guide* [online]. poslední úprava - 1.7.2004. [citováno 20.5.2011] URL: <http://pmd.bentech-taiwan.com/PMD/Pilot_Motion_Processor_Users_Guide.pdf>

Schémata MCU

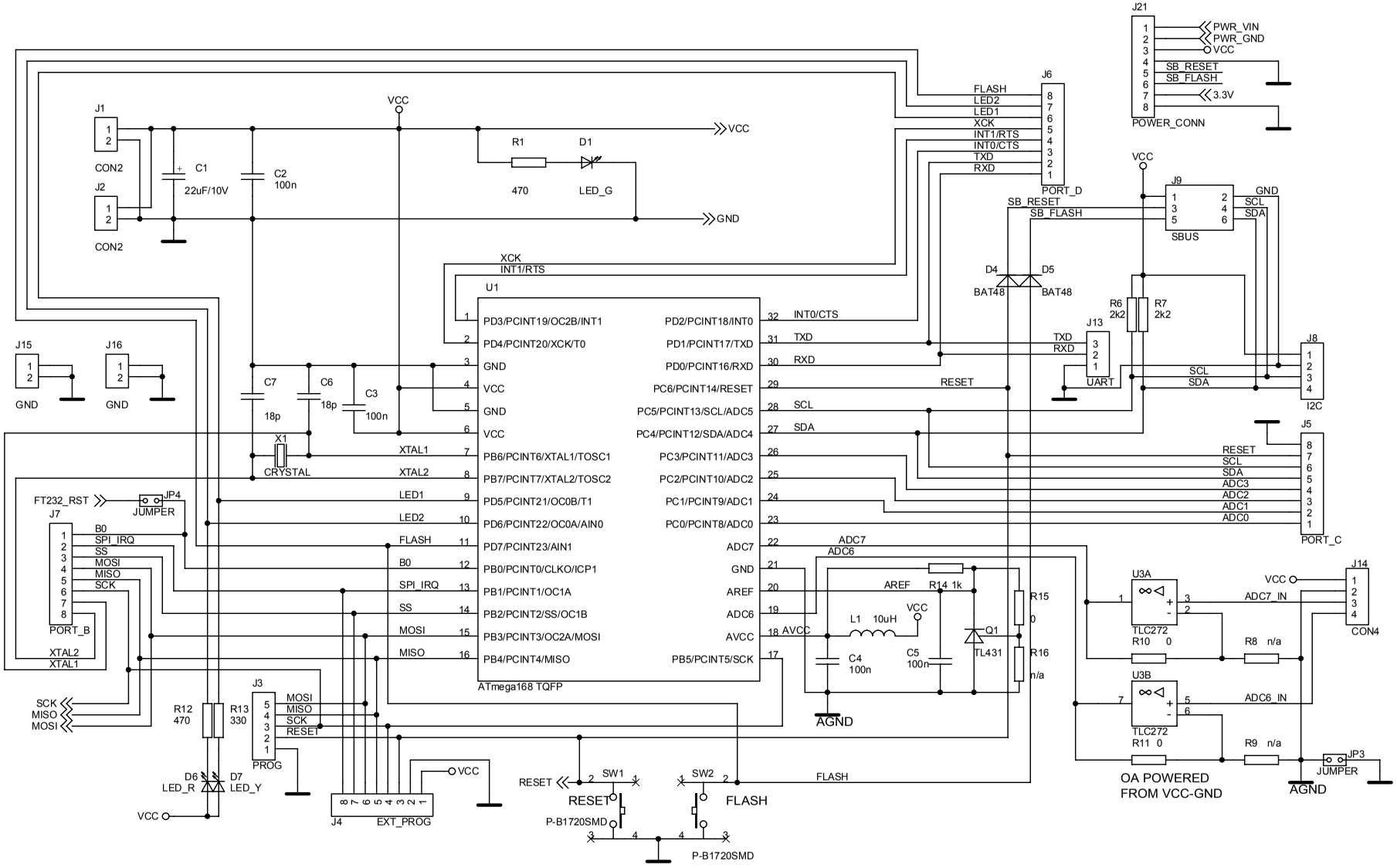
Konektor	Funkce
J1	V_{CC} (pomocný)
J3	ISP programovací
J5	ATmega PORT C
J6	ATmega PORT D
J7	ATmega PORT B
J10	mini USB
J11	USB UART
J13	ATmega UART (nutno propojit jumpery na USB UART)
J14	vstupy A/D převodníku
J17	napájecí (pomocný)
J21	napájení: 1 baterie + 2 baterie - 3 V_{CC} 5V 4 GND

Tabulka a: Popis důležitých konektorů na desce s MCU

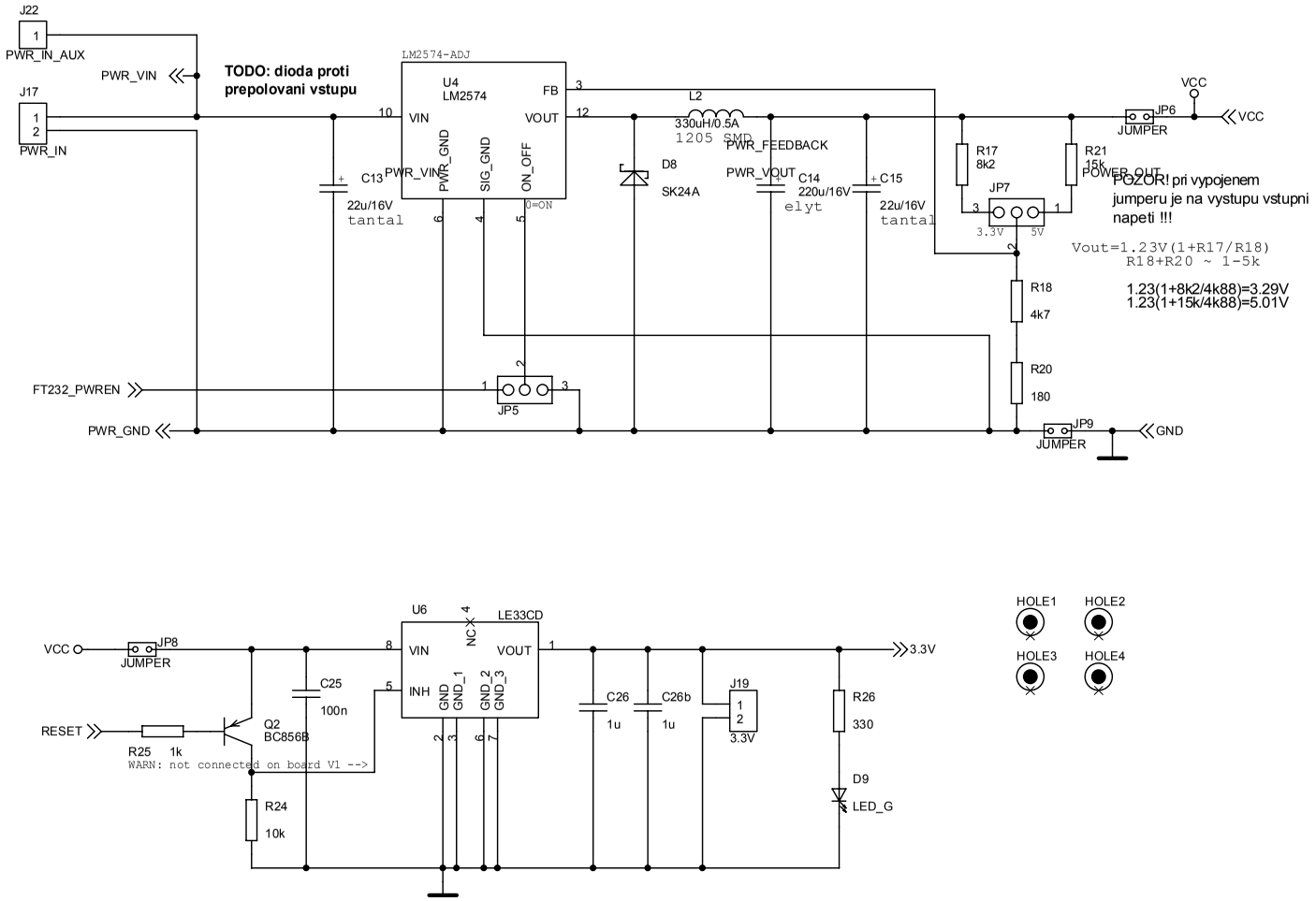
Následující součástky není potřeba na desce osazovat: C1, C17-C-26, D5, D9, J2, J8, J9, J12, J18-J20, JP4, JP5, L3, L4, Q2, R6, R7, R19, R22-26, X2.

Jumper	Funkce	Propojení
JP1	I/O úroveň USB	propojit 1-2 pro 5V
JP2	napájení desky z USB	nepropojovat
JP3	propojení analogové a napájecí země	propájet
JP4		nepropojovat
JP5		nepropojovat
JP6	propojeni stabilizátoru a V_{CC}	propojit
JP7	napětí stabilizátoru	propojit 1-2 pro 5V
JP8		nepropojovat
JP9	propojeni země stabilizátoru a GND	propojit

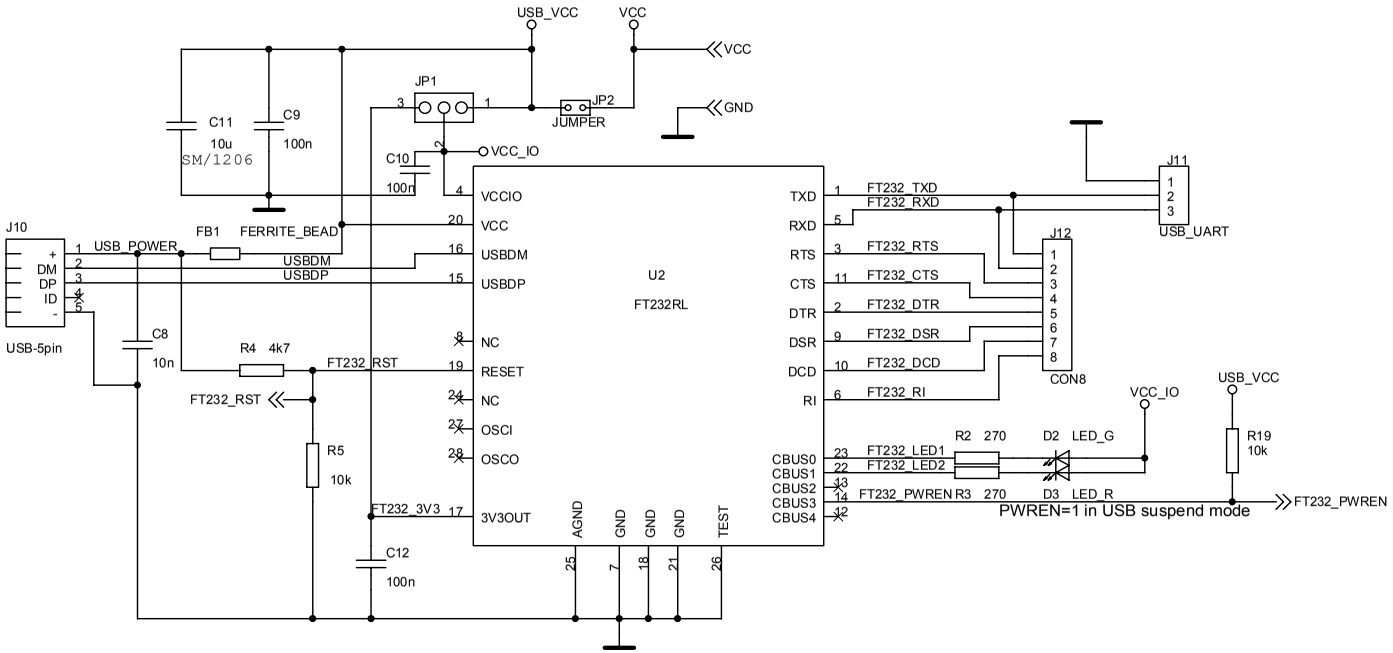
Tabulka b: Doporučené zapojení jumperů na desce s MCU



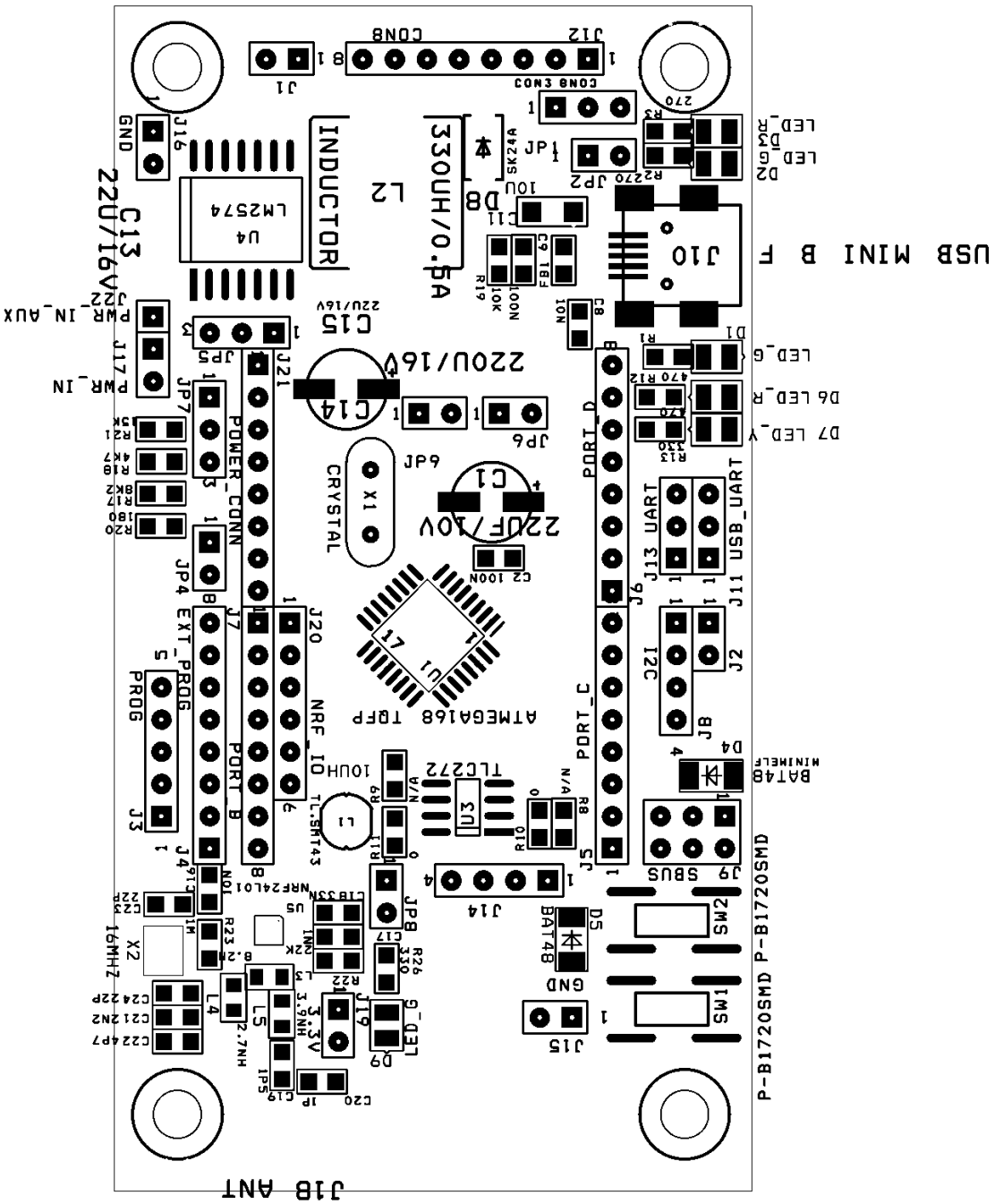
Obrázek a: Schéma zapojení procesouru ATmega168



Obrázek b: Schéma zapojení stabilizátorů napětí na desce s MCU



Obrázek c: Schéma zapojení USB rozhraní na desce s MCU

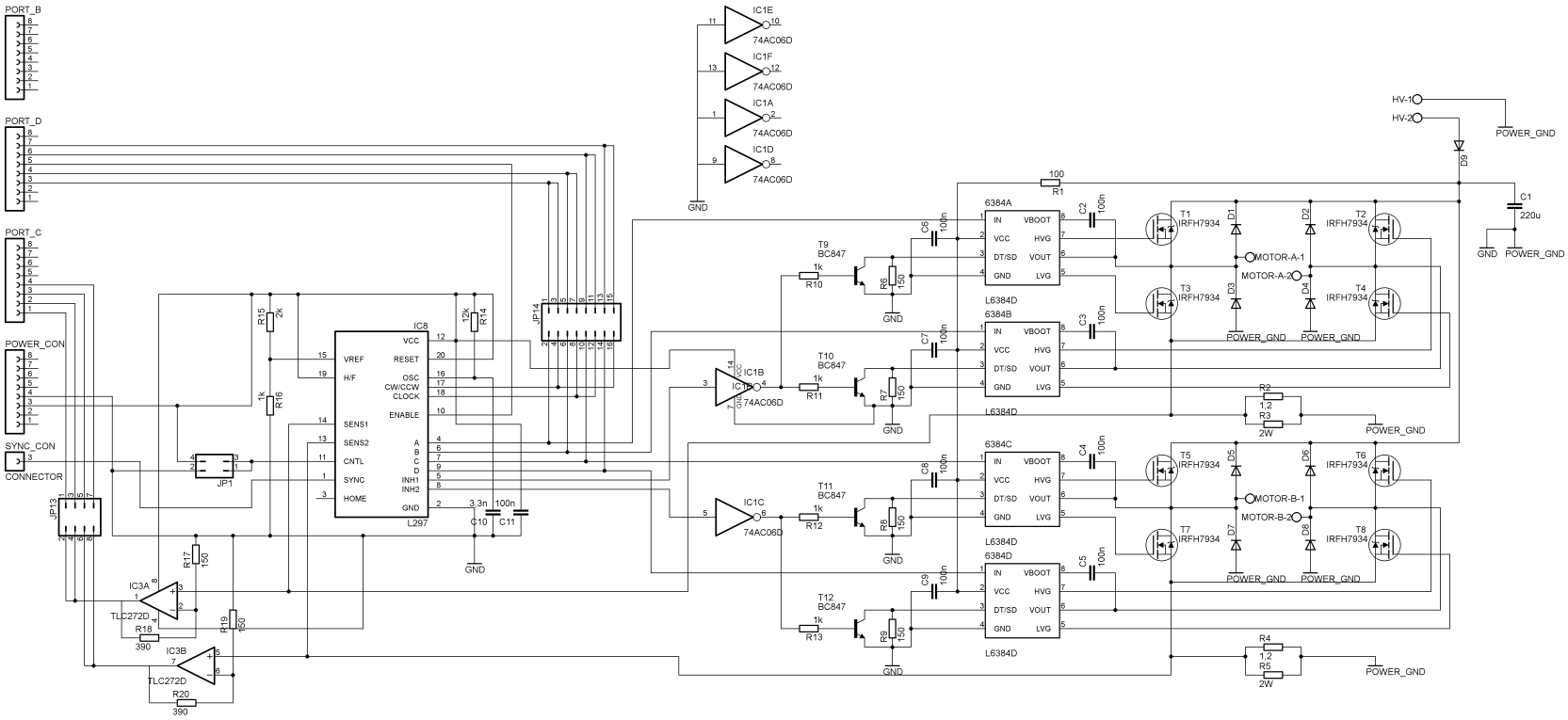


Obrázek d: Osazovací schéma desky s MCU

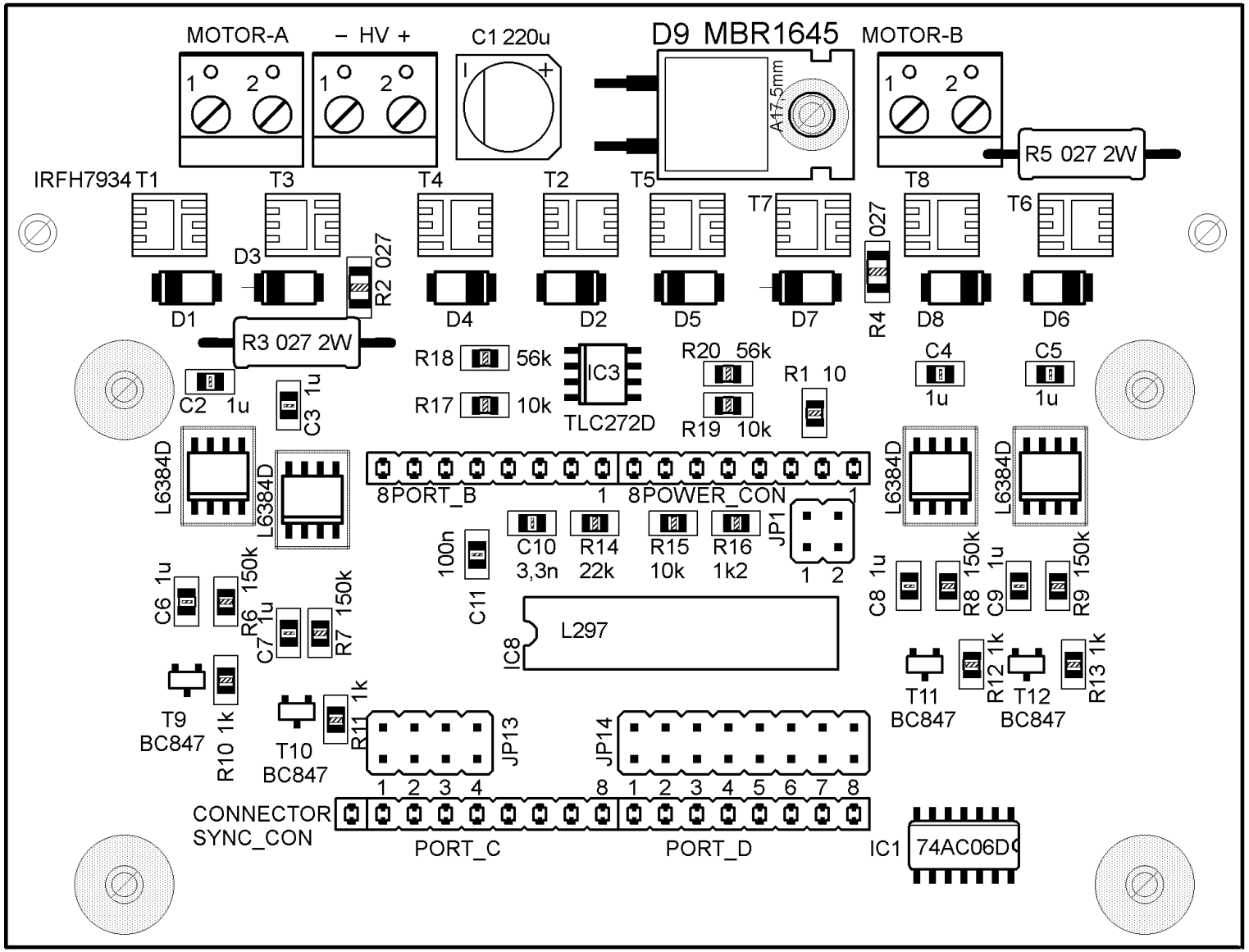
Schémata výkonové desky

Jumper	**Číslo**	**Vstup**	**Výstup**	**Propojení**
JP1:	1-2	L297 CNTL	GND	Propojit na obou deskách
	3-4		V_{CC}	nepropojovat
JP13:	1-2	PORT C - 1	výstup OZ-A	propojit na desce A
	3-4	PORT C - 2		propojit na desce B
	5-6	PORT C - 3	výstup OZ-B	propojit na desce A
	7-8	PORT C - 4		propojit na desce B
JP14:	1-2	PORT D - 3	vstup budiče A	nepropojovat
	3-4	PORT D - 3	L297 $CW/\overline{CCW}$	propojit na desce A
	5-6	PORT D - 4	vstup budiče B	nepropojovat
	7-8	PORT D - 4	L297 CLOCK	propojit na desce A
	9-10	PORT D - 6	vstup budiče C	nepropojovat
	11-12	PORT D - 6	L297 CLOCK	propojit na desce B
	13-14	PORT D - 7	vstup budiče D	nepropojovat
	15-16	PORT D - 7	L297 $CW/\overline{CCW}$	propojit na desce B

Tabulka c: Doporučené zapojení jumperů na výkonové desce



Obrázek e: Schéma zapojení výkonové desky



Obrázek f: Osazovací schéma výkonové desky

Seznam funkci

Všem funkcím se jako první parametr vždy předává struktura `mc_data`, která je knihovnou používána k uchovávání globálních proměnných. Protože význam tohoto parametru je u všech funkcí stejný, není dále zvlášť popisován. Všechny funkce mají návratový typ `int`, který také není dále u každé funkce zvlášť vypisován. Návratová hodnota je podle POSIX standartu nula, pokud funkce proběhla úspěšně. V opačném případě je rovna příslušnému chybovému kódu.

Pokud při zavolání funkce dochází ke komunikaci s MCU jsou pod názvem funkce vypsány také parametry příslušné zprávy. Napřed je uvedeno o jaký druh zprávy se jedná, následuje jméno její hlavičky, tak jak je definováno v komunikační knihovně a následně je uveden formát dat, která zpráva obsahuje. Kontrolní součet, který je na konci každé zprávy už uváděn není. V případě zpráv typu SET se rozumí, že jsou uvedená data odesílána od nadřazené jednotky směrem k MCU, které pouze potvrzuje přijetí zprávy zopakováním hlavičky zprávy. U zpráv GET je k MCU vyslána pouze příslušná hlavička a data jsou obsažena v odpovědi. U zpráv typu příkaz nedochází k přenosu dat ani v jednom směru.

`mc_init(char* device_name)`
: Musí být zavolána před použitím ostatních funkcí z knihovny. Provede inicializaci komunikace. Parametr `device_name` je C-string s adresou komunikačního portu, přes který má komunikace probíhat.

`mc_set_max_speed(float spd)`
: typ: SET, hlavička: `SET_MAX_SPD`, data: `uint16_t`
Nastaví maximální povolenou rychlost jednotky. Parametr `spd` je požadovaná hodnota rychlosti v m/s.

`mc_set_accel(float accel)`
: typ: SET, hlavička: `SET_ACCEL`, data: `uint16_t`
Nastaví hodnotu zrychlení, se kterou jednotka pracuje. Parametr `accel` je požadovaná hodnota zrychlení v m/s^2.

`mc_set_actual_speed(float spd1, float spd2)`
: typ: SET, hlavička: `SET_ACTUAL_SPD`, data: 2x `int16_t`
Nastaví hodnotu aktuální rychlosti jednotky. To znamená, že jednotka okamžitě změní rychlost na zadanou hodnotu, bez ohledu na to, jak je nastavené zrychlení. Parametry `spd1` a `spd2` jsou požadované hodnoty rychlosti v m/s. Pokud je zadaná rychlost vyšší než nastavená maximální rychlost, je jednotkou nastavena maximální rychlost.

`mc_set_goal_speed(float spd1, float spd2)`
typ: SET, hlavička: `SET_GOAL_SPD`, data: 2x `int16_t`
Nastaví hodnotu požadované rychlosti. Po obdržení tohoto příkazu provede jednotka plynulou změnu rychlosti na zadanou hodnotu. Parametry `spd1` a `spd2` jsou požadované hodnoty rychlosti v m/s. Pokud je zadaná rychlost vyšší než nastavená maximální rychlost, je jednotkou nastavena maximální rychlost.

`mc_set_distance(float dist1, float dist2)`
typ: SET, hlavička: `SET_DISTANCE`, data: 2x `int32_t`
Pošle jednotce příkaz, aby každým svým kolem ujela zadanou vzdálenost. Parametry `dist1` a `dist2` jsou hodnoty vzdálenosti, která má být ujeta jednotlivými koly v m. Jednotce je údaj o vzdálenosti posílán v ticích. přepočet je prováděn podle aktuálně nastavené hodnoty konstanty vzdálenosti. Při plnění tohoto příkazu se bude jednotka chovat dle nastavených hodnot maximální rychlosti, zrychlení a konstanty P-regulátoru.

`mc_set_distance_raw(float dist1, float dist2)`
typ: SET, hlavička: `SET_DISTANCE`, data: 2x `int32_t`
Pošle jednotce příkaz, aby každým svým kolem ujela zadanou vzdálenost. Parametry `dist1` a `dist2` jsou hodnoty vzdálenosti, která má být ujeta jednotlivými přímo v ticích. Při plnění tohoto příkazu se bude jednotka chovat dle nastavených hodnot maximální rychlosti, zrychlení a konstanty P-regulátoru.

`mc_set_step_angle(float sa)`
typ: SET, hlavička: `SET_STEP_ANGLE`, data: `uint16_t`
Nastaví velikost jednoho kroku použitých motorů. Parametr `sa` je velikost kroku ve stupních.

`mc_set_current_threshold(float thr)`
typ: SET, hlavička: `SET_CURRENT_THRESHOLD`, data: `uint16_t`
Nastaví mezní hodnotu proudu protékajícího jedním z motorů, při které má dojít k samočinnému vypnutí výkonové části jednotky. Parametr `thr` je požadovaná hodnota proudu v A. Pokud je nastavená 0, kontrola hodnot proudu je vypnuta.

`mc_set_temp_threshold(float thr)`
typ: SET, hlavička: `SET_TEMP_THRESHOLD`, data: `uint16_t`
Nastaví mezní hodnotu teploty výkonové části, při které má dojít k jejímu samočinnému vypnutí. Parametr `thr` je požadovaná teplota v $°C$. Pokud je nastavená 0, kontrola teploty je vypnuta.

`mc_set_idle_time(float it)`
typ: SET, hlavička: `SET_IDLE_TIME`, data: `uint16_t`
Nastavuje časový interval, který musí uběhnout od posledního přijetí zprávy MCU, aby došlo k samočinnému vypnutí výkonové části. Parametr `it` je délka požadovaného intervalu v s. Pokud je interval nastaven na 0, je vypínání výkonové části z časových důvodů vypnuto.

`mc_set_temp_interval(float it)`
typ: SET, hlavička: `SET_TEMP_INTERVAL`, data: `uint16_t`
Nastavuje časovou periodu pro pravidelné automatické vyčítání teploty. Parametr

`it` je délka periody v s. Pokud je perioda nastavená na 0, automatické vyčítání je vypnuto.

`mc_set_odometry_interval(float it)`
typ: SET, hlavička: `SET_ODOMETRY_INTERVAL`, data: `uint16_t`
Nastavuje časovou periodu pro pravidelné automatické vyčítání odometrie. Parametr `it` je délka periody v s. Pokud je perioda nastavená na 0, automatické vyčítání je vypnuto.

`mc_set_current_interval(float it)`
typ: SET, hlavička: `SET_CURRENT_INTERVAL`, data: `uint16_t`
Nastavuje časovou periodu pro pravidelné automatické vyčítání proudu. Parametr `it` je délka periody v s. Pokud je perioda nastavená na 0, automatické vyčítání je vypnuto.

`mc_set_enable(int val)`
typ: SET, hlavička: `SET_ENABLE`, data: `uint8_t`
Spouští nebo vypíná výkonovou část jednotky. Pokud je parametr `val` 0 je odeslán příkaz k vypnutí, pro jakoukoliv jinou hodnotu je vyslán příkaz k sepnutí.

`mc_set_pconst(float pconst)`
typ: SET, hlavička: `SET_PCONST`, data: `uint16_t`
Nastaví hodnotu konstanty P-regulátoru. Parametr `thr` je požadovaná bezrozměrná hodnota.

`mc_set_ticks_per_meter(double ticks)`
typ: SET, hlavička: `SET_WHEEL_DIAMETER`, data: `uint16_t`
Nastaví konstantu vzdálenosti. Parametr `ticks` udává požadovanou hodnotu v ticích na metr.

`mc_set_ticks_per_rotation(double ticks)`
Nastaví konstantu otáčení. Parametr `ticks` udává požadovanou hodnotu tiků potřebných pro otočení o 360°.

`mc_get_max_speed(float* rval)`
typ: GET, hlavička: `GET_MAX_SPEED`, data: `uint16_t`
Vyčte aktuálně nastavenou hodnotu maximální rychlosti v m/s. Parametr `rval` je ukazatel na proměnnou do které se má hodnota zapsat.

`mc_get_accel(float* rval)`
typ: GET, hlavička: `GET_ACCEL`, data: `uint16_t`
Vyčte aktuálně nastavenou hodnotu zrychlení v m/s^2. Parametr `rval` je ukazatel na proměnnou do které se má hodnota zapsat.

`mc_get_actual_speed(float* rval1, float* rval2)`
typ: GET, hlavička: `GET_ACTUAL_SPEED`, data: 2x `int16_t`
Vyčte hodnoty aktuálních rychlostí obou kol. Parametry `rval1` a `rval2` jsou ukazatelé na proměnné do kterých se mají hodnoty zapsat.

`mc_get_goal_speed(float* rval1, float* rval2)`
typ: GET, hlavička: `GET_GOAL_SPEED`, data: 2x `int16_t`
Vyčte hodnoty aktuálních rychlostí obou kol. Parametry `rval1` a `rval2` jsou

ukazatelé na proměnné do kterých se mají hodnoty zapsat.

`mc_get_step_angle(float* rval)`
typ: GET, hlavička: `GET_STEP_ANGLE`, data: `uint16_t`
Vyčte aktuálně nastavenou hodnotu velikosti kroku motoru ve stupních. Parametr `rval` je ukazatel na proměnnou do které se má hodnota zapsat.

`mc_get_temp(float* rval1, float* rval2)`
typ: GET, hlavička: `GET_TEMP`, data: 2x `uint16_t`
Vyčte aktuální hodnoty teplot obou výkonových desek. Parametry `rval1` a `rval2` jsou ukazatelé na proměnné do kterých se mají hodnoty zapsat.

`mc_get_current(float* rval1,float* rval2,float* rval3,float* rval4)`
typ: GET, hlavička: `GET_CURRENT`, data: 4x `uint16_t`
Vyčte aktuální hodnoty proudů protékajících všemi čtyřmi H-můstky. Parametry `rval1` - `rval4` jsou ukazatelé na proměnné do kterých se mají hodnoty zapsat.

`mc_get_odometry(float* rval1, float* rval2)`
typ: GET, hlavička: `GET_ODOMETRY`, data: 2x `int32_t`
Vyčte odometrická data a provede jejich přepočet na metry podle aktuální hodnoty konstanty vzdálenosti. Parametry `rval1` a `rval2` jsou ukazatelé na proměnné do kterých se mají zapsat výsledné hodnoty.

`mc_get_current_threshold(float* rval)`
typ: GET, hlavička: `GET_CURRENT_THRESHOLD`, data: `uint16_t`
Vyčte aktuálně nastavenou mezní hodnotu velikosti proudu protékajícího motorem, při jejímž překročení dochází k automatickému vypnutí výkonové části. Parametr `rval` je ukazatel na proměnnou do které se zapíše hodnota v A.

`mc_get_temp_threshold(float* rval)`
typ: GET, hlavička: `GET_TEMP_THRESHOLD`, data: `uint16_t`
Vyčte aktuálně nastavenou mezní hodnotu teploty výkonové části, při jejímž překročení dochází k jejímu automatickému vypnutí. Parametr `rval` je ukazatel na proměnnou do které se zapíše hodnota ve $°C$.

`mc_get_idle_time(float* rval)`
typ: GET, hlavička: `GET_IDLE_TIME`, data: `uint16_t`
Vyčte aktuální hodnotu časového intervalu, který musí uběhnout od posledního přijetí zprávy MCU, aby došlo k samočinnému vypnutí výkonové části. Parametr `rval` je ukazatel na proměnnou do které se zapíše jeho hodnota v s.

`mc_get_temp_interval(float* rval)`
typ: GET, hlavička: `GET_TEMP_INTERVAL`, data: `uint16_t`
Vyčte aktuální délku periody pro automatické vyčítání teplotních dat. Parametr `rval` je ukazatel na proměnnou do které se zapíše její hodnota v s.

`mc_get_odometry_interval(float* rval1`
typ: GET, hlavička: `GET_ODOMETRY_INTERVAL`, data: `uint16_t`
Vyčte aktuální délku periody pro automatické vyčítání odometrických dat.
Parametr `rval` je ukazatel na proměnnou do které se zapíše její hodnota v s.

`mc_get_current_interval(float* rval1)`
typ: GET, hlavička: `GET_CURRENT_INTERVAL`, data: `uint16_t`
Vyčte aktuální délku periody pro automatické vyčítání hodnot proudu. Parametr `rval` je ukazatel na proměnnou do které se zapíše její hodnota v s.

`mc_get_enable(int* rval)`
typ: GET, hlavička: `GET_ENABLE`, data: `uint8_t`
Slouží k zjištění stavu výkonové části. V případě, že je sepnutá je do proměnné, na kterou odkazuje ukazatel `rval` zapsána 1, v opačném případě 0.

`mc_get_pconst(float* rval)`
typ: GET, hlavička: `GET_PCONST`, data: `uint16_t`
Vyčte hodnotu konstanty P-regulátoru. Parametr `rval` je ukazatel na proměnnou, kam bude hodnota zapsána.

`mc_memory_save()`
typ: příkaz, hlavička: `MEMORY_SAVE`
Dá MCU příkaz k uložení následujících konstant do paměti EEPROM: maximální rychlost, zrychlení, konstanta vzdálenosti, konstanta otáčení, mezní hodnoty proudu a teploty, čas automatického vypnutí při nečinnosti, velikost kroku motoru a konstanta P-regulátoru.

`mc_memory_load()`
typ: příkaz, hlavička: `MEMORY_LOAD`
Dá MCU příkaz nahrát z paměti EEPROM výše uvedené konstanty.

`mc_reset_odometry()`
typ: příkaz, hlavička: `RESET_ODOMETRY`
Pošle MCU příkaz k vynulování svého vnitřního čítače odometrie.

`mc_get_ticks_per_meter(double* rval)`
typ: GET, hlavička: `GET_WHEEL_DIAMETER`, data: `uint16_t`
Vyčte konstantu vzdálenosti. Parametr `rval` je ukazatel na proměnnou, kam bude zapsána hodnota konstanty v ticích na metr.

`mc_get_ticks_per_rotation(double* rval)`
Vyčte konstantu otáčení. Parametr `rval` je ukazatel na proměnnou, kam bude zapsána hodnota konstanty v ticích potřebných pro otočení o 360°.

`mc_quit()`
Tuto funkci je nutné zavolat před ukončením práce s knihovnou. Provede uzavření komunikačního portu, zruší vytvořená vlákna a uvolní alokovanou paměť.

`mc_set_temp_callback(void (*callback_pointer)(float, float))`
Předá datové struktuře ukazatel na funkci, která bude spuštěna při obdržení nevyžádaných dat o teplotě od MCU. Jako nevyžádaná data se počítají i data, která jsou z MCU vyčítána pomoci funkce `mc_set_temp_interval`. Fukce musí mít předepsaný formát, aby ji mohla být předána příslušná data přes její dva parametry.

`mc_set_odometry_callback(void (*callback_pointer)(float, float))`
Předá datové struktuře ukazatel na funkci, která bude spuštěna při obdržení

nevyžádaných odometrických dat od MCU. Jako nevyžádaná data se počítají i data, která jsou vyčítána pomoci funkce `mc_set_odometry_interval`. Fukce musí mít předepsaný formát, aby ji mohla být předána příslušná data přes její dva parametry.

`mc_set_current_callback(void (*callback_pointer)...`
`...(float, float, float, float))`
Předá datové struktuře ukazatel na funkci, která bude spuštěna při obdržení nevyžádaných dat o hodnotách proudu od MCU. Jako nevyžádaná data se počítají i data, která jsou vyčítána pomoci funkce `mc_set_current_interval`. Fukce musí mít předepsaný formát, aby ji přes její parametry mohla být předána příslušná data.

`mc_set_event_callback(void (*callback_pointer)(int))`
Předá datové struktuře ukazatel na funkci, která bude spuštěna při obdržení zpráv o událostech. Použitá funkce musí mít jeden parametr typu `int`, přes který ji bude předávána hodnota hlavičky obdržené zprávy, která identifikuje typ nastalé události.

Naměřené hodnoty

$\mathbf{v_{max} = 0.2 m/s, a = 0.1 m/s^2}$		
Číslo měření	**Kladný směr otáčení**	**Záporný směr otáčení**
1.	125190	125067
2.	125112	125080
3.	125106	125040
4.	125116	125038
5.	125102	125034

Tabulka d: Výsledky měření konstanty otáčení

$\mathbf{v_{max} = 0.1 m/s}$			
Číslo měření	$\mathbf{a = 0.1 m/s^2}$	$\mathbf{a = 0.2 m/s^2}$	$\mathbf{a = 0.3 m/s^2}$
1.	64043	64048	64040
2.	64067	64040	64035
3.	64054	64045	64032
4.	64086	64040	64061
5.	64064	64060	64086
6.	64068	64040	64094
7.	64044	64044	64066
8.	64082	64050	64080
9.	64030	64064	64076
10.	64056	64054	64064
$\mathbf{v_{max} = 0.2 m/s}$			
Číslo měření	$\mathbf{a = 0.1 m/s^2}$	$\mathbf{a = 0.2 m/s^2}$	$\mathbf{a = 0.3 m/s^2}$
1.	64125	64061	64086
2.	64101	64053	64050
3.	64068	64066	64034
4.	64110	64062	64063
5.	64084	64056	64052
6.	64088	64040	64048
7.	64080	64064	64033
8.	64088	64035	64053
9.	64090	64068	64050
10.	64076	64055	64063
$\mathbf{v_{max} = 0.3 m/s}$			
Číslo měření	$\mathbf{a = 0.1 m/s^2}$	$\mathbf{a = 0.2 m/s^2}$	$\mathbf{a = 0.3 m/s^2}$
1.	64084	64082	64079
2.	64085	64044	64056
3.	64081	64058	64043
4.	64061	64068	64057
5.	64076	64044	64054
6.	64074	64069	64065
7.	64086	64068	64047
8.	64071	64080	64077
9.	64084	64076	64060
10.	64081	64060	64077

Tabulka e: Výsledky měření konstanty vzdálenosti

Obsah CD

Následující tabulka shrnuje obsah přiloženého CD:

Adresář	**Popis**
`mclib`	zdrojové kódy knihovny + jednoduchý komunikační program
`MCU`	zdrojové kódy programu MCU + .hex soubor + bootloader
`doc`	zdrojové kódy textu bakalářské práce
`datasheets`	technická dokumentace použitých součástek
`thesis.pdf`	text bakalářské práce

Tabulka f: Adresářová struktura na CD