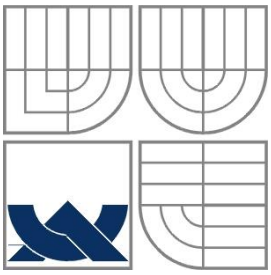
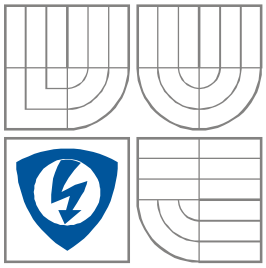


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ŘÍZENÍ 6-TI OSÉHO MANIPULÁTORU

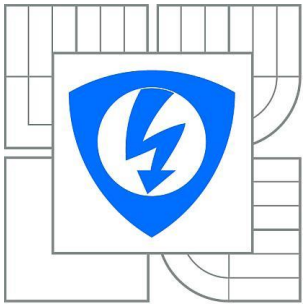
6 AXIS MANIPULATOR CONTROL

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE — Bc. MICHAL SEMRÁD
AUTHOR

VEDOUCÍ PRÁCE — Ing. ONDŘEJ HYNČICA
SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor

Kybernetika, automatizace a měření

Student: Bc. Michal Semrád ***ID:*** 125625
Ročník: 2 ***Akademický rok:*** 2013/2014

NÁZEV TÉMATU:

Řízení 6-ti osého manipulátoru

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s přímou a inverzní kinematickou úlohou.
2. Seznamte se s vlastnostmi a možnostmi jádra ARM Cortex-M3.
3. Seznamte se s HW řešením ovládací jednotky 6-ti osého manipulátoru.
4. Navrhněte a realizujte firmware pro tuto řídicí jednotku s cílem realizovat řízení manipulátoru v reálném case.
5. Propojte jednotku s vámi zvoleným nadřazeným systémem pomocí vhodného komunikačního rozhraní a doplňte firmware jednotky o komunikaci s tímto nadřazeným systémem realizujícím vyšší úroveň řízení.

DOPORUČENÁ LITERATURA:

Yiu, J.: The Definitive Guide to the ARM Cortex-M3, Second Edition. Burlington: Newnes, 2009.

Termín zadání: 10.2.2014 ***Termín odevzdání:*** 19.5. 2014

Vedoucí práce: Ing. Ondřej Hynčica
Konzultanti semestrální práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

Tato diplomová práce se zabývá návrhem a tvorbou řídicího systému pro 6-ti osý manipulátor. Ten je tvořen mikrokontolérem LPC1756 s firmware realizovaným pod operačním systémem reálného času FreeRTOS a GUI aplikací na PC. Robotický manipulátor je řízen PWM signály generovanými mikrokontrolérem, který komunikuje s PC přes sériové rozhraní pomocí SLIP protokolu. Teoretická část práce se zabývá vysvětlením důležitých pojmů, popisem používaného manipulátoru a jeho řídicí jednotky. Praktická část popisuje řešení kinematických úloh pro používaný manipulátor, realizaci firmware a GUI aplikace.

Klíčová slova

ARM Cortex-M3, Servomotor, Robotický manipulátor, LPC1756, CooCox, Inverzní kinematická úloha, Operační systém reálného času, FreeRTOS, Firmware, GUI

Abstract

This master thesis discusses about designing and realization of a control system for a 6-axis robotic arm. The controlling system consist in a microcontroller LPC1756 with its firmware implemented under the Real-time operating system FreeRTOS and GUI application, running on a PC. The Microcontroller communicates with the PC through a serial line via SLIP protocol. Theoretically, it will deal with an explanation of the important terms, and describes the used robotic arm and its controlling unit. The practical part describes kinematics problems solving, firmware's realization and GUI application.

Keywords

ARM Cortex-M3, Servomotor, Robotic arm, LPC1756, CooCox, Inverse kinematics, Real-time operating system, FreeRTOS, Firmware, GUI

Bibliografická citace:

SEMRÁD, M. *Řízení 6-ti osého manipulátoru.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2014. 61 s. Vedoucí diplomové práce byl Ing. Ondřej Hynčica

Prohlášení

„Prohlašuji, že svou diplomovou práci na téma ŘÍZENÍ 6-TI OSÉHO MANIPULÁTORU jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **19. května 2014** ……
podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Ondřeji Hynčicovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **19. května 2014** ……

podpis autora

OBSAH

1 ÚVOD

Cílem této diplomové práce je vytvořit systém pro řízení 6-ti osého manipulátoru. Druhá kapitola zabývající se celkovou koncepcí řídicího systému poskytne čtenáři pomocí koncepčního schématu hned z úvodu potřebnou představu o tom, jak by měl výsledný systém vypadat a jaké by měly být jeho možnosti.

V teoretické části práce bude nejprve obecně seznámeno s robotickými manipulátory a vysvětleny pojmy přímé a inverzní kinematické úlohy. Následovat bude popis řídicí jednotky, zejména jejího nejdůležitějšího prvku, kterým je mikrokontrolér LPC1756 s jádrem ARM Cortex-M3. Součástí teoretického rozboru je také stručný popis dvou základních programových řešení, které jsou využívány při tvorbě řídicích aplikací.

Za teoretickou část je zařazena kapitola o zvoleném operačním systému reálného času FreeRTOS. Budou zde popsány jeho základní vlastnosti a součásti, které jsou využity při vytváření firmware pro řídicí jednotku. Důležitou součástí této kapitoly je popis konfigurace systému pro použití v daném mikrokontroléru.

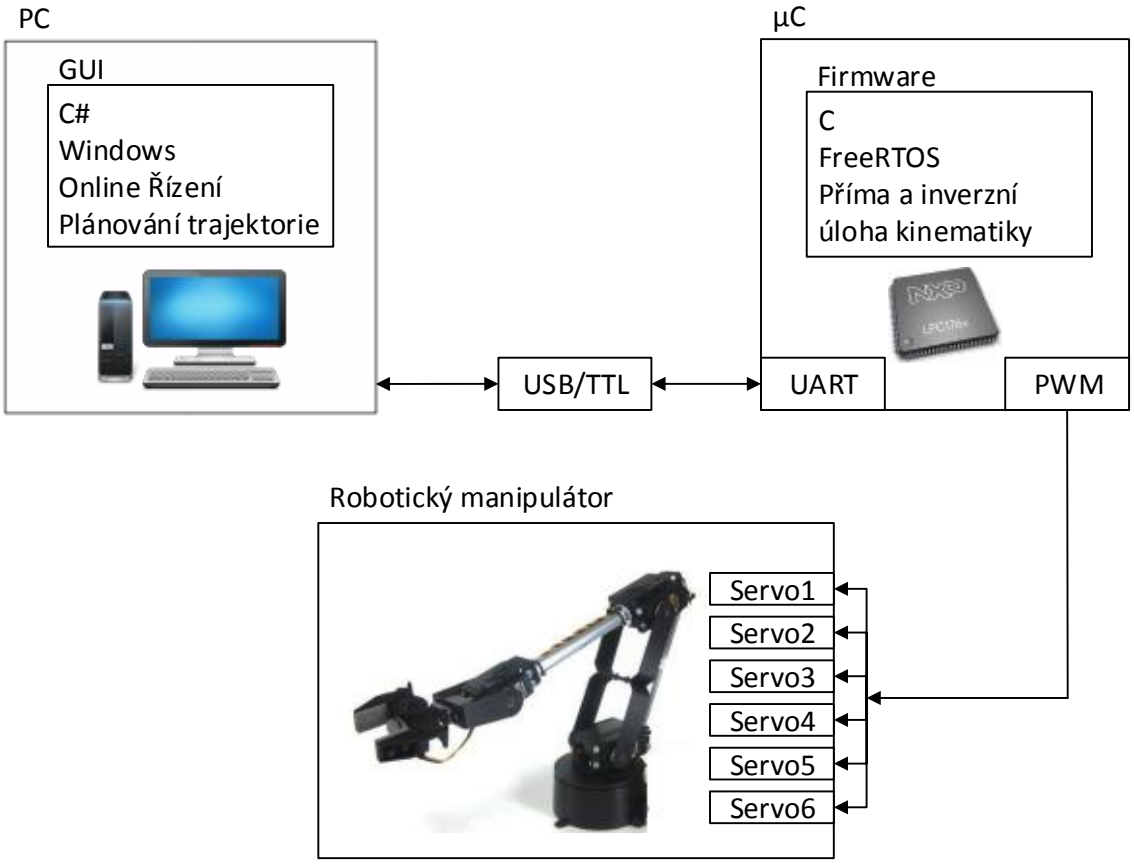
Praktická část práce začne kapitolou, která se bude zabývat kinematikou manipulátoru, konkrétně tedy řešením přímé a inverzní kinematické úlohy. Následovat bude stěžejní kapitola celé práce, a tou je popis vytvořeného firmware pro řídicí jednotku manipulátoru. Nejprve bude popsána jeho struktura a rozčlenění na jednotlivé úlohy. Tyto úlohy budou poté postupně rozebírány a jejich popis doplněn o vývojové diagramy, které pomohou s pochopením jejich činnosti. Jedním z hlavních úkolů firmware je výpočet obou kinematických úloh. Řešením těchto úloh jsou matematické rovnice složené z funkcí jako je sinus, cosinus, arkus tangens a ty musí být nejdříve do mikrokontroléru implementovány.

Práce se bude dále zabývat tvorbou PC aplikace, která slouží jako grafické uživatelské rozhraní pro komunikaci s firmware, a tím i pro ovládání robota. Nejprve bude uvedeno, jak byla tato aplikace vytvořena a poté formou uživatelského manuálu popsáno, jak může být používána. Před samotný závěr práce je zařazena malá kapitola, která má za cíl předvést konkrétní postup při vytváření trajektorie a vyhodnotit funkčnost kompletního řídicího systému i s používaným manipulátorem.

2 KONCEPCE ŘÍDICÍHO SYSTÉMU

Ze zadání tohoto projektu vychází koncepční schéma systému, které je znázorněno na *Obrázek 1*. Stěžejním prvkem systému je mikrokontrolér, především tedy jeho firmware, který je vytvořen pod operačním systémem FreeRTOS v jazyce C. Jeho hlavním úkolem je ovládat 6-ti osý manipulátor pomocí PWM signálů. Firmware je schopen řešit přímou a inverzní kinematickou úlohu v reálném čase. Pro komunikaci s nadřazeným systémem je využito sériové rozhraní mikrokontroléru a převodník z TTL logiky na USB.

Roli nadřazeného systému vykonává PC s aplikací grafického uživatelského rozhraní (dále jen GUI), která je napsaná v jazyce C# a využívá platformu .NET. Cílem tohoto GUI je zasílat povely programu běžícímu ve firmware a řídit tak manipulátor. GUI nabízí možnost online řízení manipulátoru ve dvou variantách a to jak v kartézských, tak v kloubových souřadnicích. Tyto jednotlivé souřadnice tvořící pozici koncového bodu manipulátoru lze postupně ukládat do seznamu příkazů a vytvářet tak trajektorii, kterou má manipulátor vykonat. Příkaz kromě souřadnic koncového bodu obsahuje také údaj o požadované rychlosti a době setrvání v zadané pozici. Po vytvoření trajektorie lze tento seznam příkazů odeslat do mikrokontroléru a spustit tak automatické vykonávání programu. Seznam příkazů je také možné ukládat do textového souboru a zpětně z něho načítat.



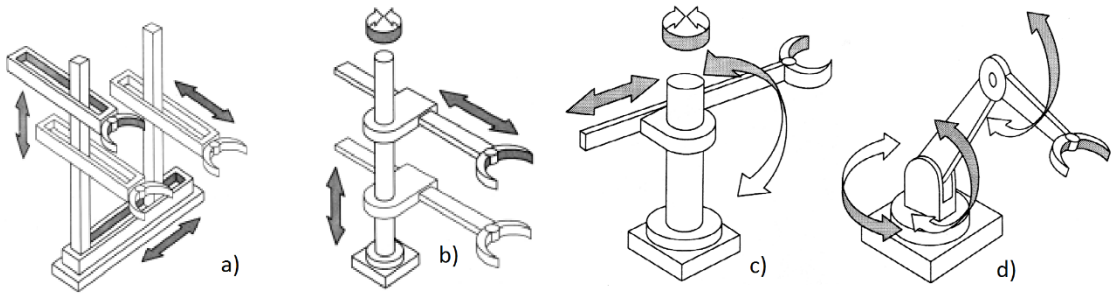
Obrázek 1: Koncepce řídicího systému

3 TEORETICKÝ ROZBOR

3.1 Robotické manipulátory

Robotické manipulátory jsou zařízení určená pro pohybování s různými objekty. Jejich primárním úkolem je nahradit člověkem vykonávanou práci. Používáním manipulátoru namísto lidské práce umožňuje především zvýšit přesnost a opakovatelnost manipulace s objekty. Výhod však existuje celá řada, z těch nejvýznamnějších lze dále vybrat, že nasazení manipulátoru do výroby má za následek zvýšení kvality vyráběného produktu a celkové zefektivnění výrobního procesu.

Manipulátory jsou tvořeny dvěma základními prvky a těmi jsou klouby a ramena. Uspořádáním kloubů a ramen vzniká tzv. kinematický řetězec, kterým je dána koncepce manipulátoru. Úkolem manipulátoru je zajistit polohování uchopeného předmětu v prostoru. Poloha a orientace každého tělesa v prostoru může být definována šesti údaji. 3 hodnoty [x,y,z] udávají polohu objektu v kartézských souřadnicích a další 3 hodnoty [α,β,γ] udávají natočení tělesa v jednotlivých osách. Říkáme tedy, že volné těleso v prostoru má 6 stupňů volnosti. Abychom dosáhli plné manipulovatelnosti objektu pomocí robotického manipulátoru, musí mít tento manipulátor minimálně 6 kloubů, které musí společně s rameny tvořit takovou koncepci, která je schopna zajistit pohyb uchopeného tělesa ve všech šesti stupních volnosti. Robot s méně než šesti klouby má manipulační schopnosti omezené. Údaj o nastavení jednotlivých kloubů nesou tzv. *kloubové proměnné* nazývané také *kloubové souřadnice* a bývají označované symbolem *q*. [18]



Obrázek 2: Základní kinematické koncepce manipulátorů [18]

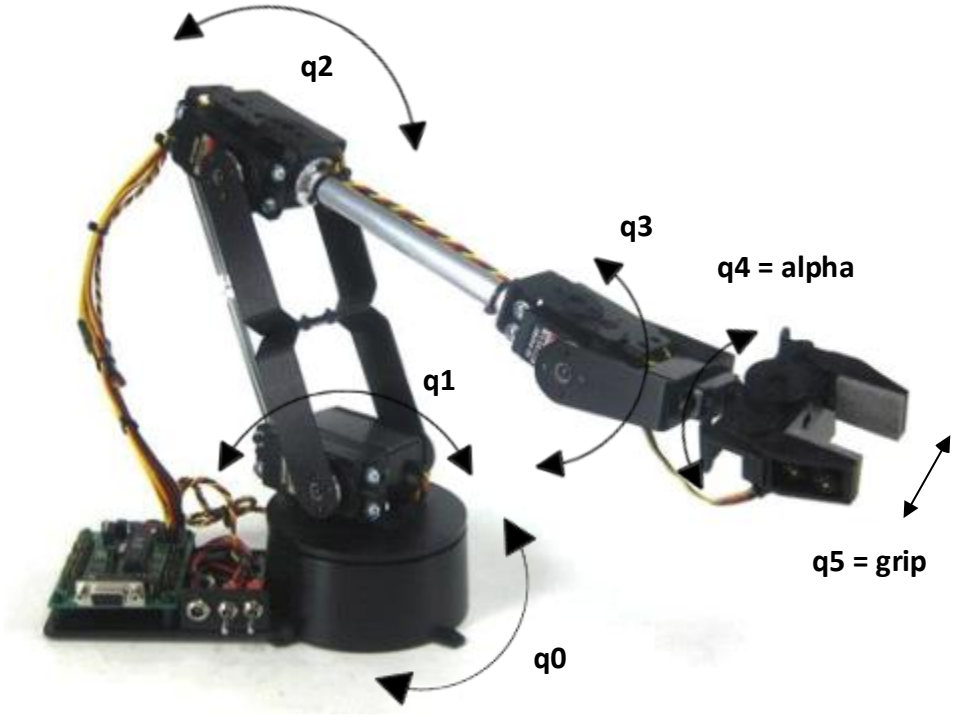
Obrázek 2 ukazuje základní kinematické koncepce manipulátorů – kde a) kartézská, b) cylindrická, c) sférická, d) kloubové rameno

Nyní si představme situaci, že budeme chtít řídit pohyb manipulátoru s koncepcí kloubového ramene s šesti klouby. To lze provádět dvěma způsoby. Jedním z těchto způsobu je přímo zadat vektor kloubových souřadnic $\mathbf{q}=[q0,q1,q2,q3,q4,q5]^T$. Vektor pozice koncového členu robota $\mathbf{P}=[x,y,z,\alpha,\beta,\gamma]^T$ je pak dán jednoznačným řešením sady šesti rovnic, kterou je možné sestavit s běžnými znalostmi geometrie. Matematicky zapsáno $\mathbf{P} = \mathbf{f}(\mathbf{q})$. Tento postup je nazýván jako *přímá úloha kinematiky*. [18]

Pro člověka je však mnohem přirozenější si pohyb robota představovat v kartézských souřadnicích, tzn. zadat vektor **P**. Aby byl robot schopen této pozice dosáhnout, musí znát vektor kloubových souřadnic **q**. Výpočet kloubových souřadnic ze znalosti kartézských souřadnic se nazývá *inverzní úloha kinematiky*, **q** = **f**(**P**). Nalezení tohoto řešení je mnohem složitější než u přímé úlohy kinematiky. Řešení může existovat i více, nebo dokonce i nekonečně mnoho. [18]

3.1.1 Použitý manipulátor

Robotické rameno *(Obrázek 3*) s označením AL5D je zakoupeno jako profesionální výrobek společnosti Lynxmotion. Jedná se o 6-ti osý manipulátor s koncovým bodem polohovatelným v 5 stupních volnosti. Jeden servomotor (pohyb v šesté ose) je využit pro ovládání uchopovacích kleští, z tohoto důvodu má tento manipulátor pouze 5 stupňů volnosti. Jednotlivá ramena jsou vyrobena z hliníkových plechů. Pohon manipulátoru v kloubech zajišťují modelářské servomotory od firmy Hitec. Osazení jednotlivých kloubů uvádí Tabulka 1.



Obrázek 3: Robotické rameno AL5D

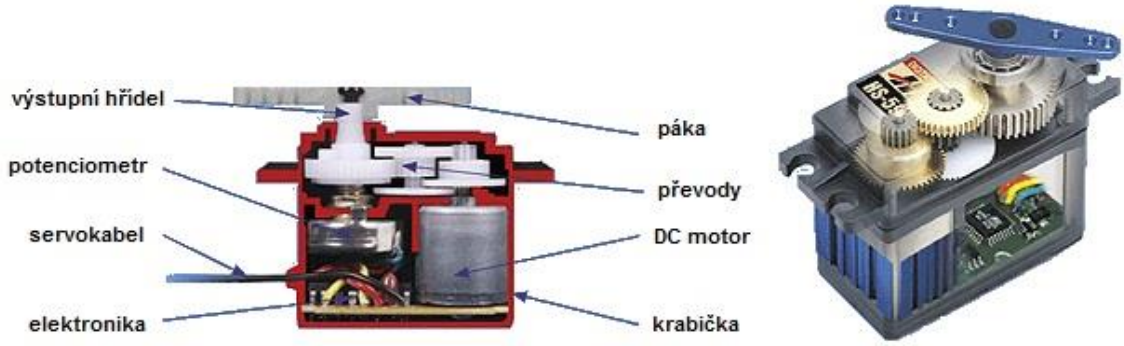
Tabulka 1: Servomotory použité v manipulátoru [16]

Kloub *1)	Typ	Rychlost [sec/60°] *2)	Točivý moment [Nm]*3)	Pracovní rozsah	Proud [mA] *4)
q0	HS-485HB	0,22	0,48	180°	150
q1	HS-805BB	0,19	1,98	180°	700
q2	HS-755HB	0,28	1,10	180°	230
q3	HS-645MG	0,24	0,77	180°	350
q4	HS-485HB	0,22	0,48	180°	150
q5	HS-422	0,21	0,33	180° 37mm	150

*1) Číslováno je směrem od základny k uchopovacím kleštím
*2) Rychlost je doba změny natočení o 60° při napájení 4,8 V nezatíženého servomotoru
*3) Při napájení 4,8 V
*4) Při napájení 4,8 V nezatíženého servomotoru

3.1.2 Modelářské servomotory

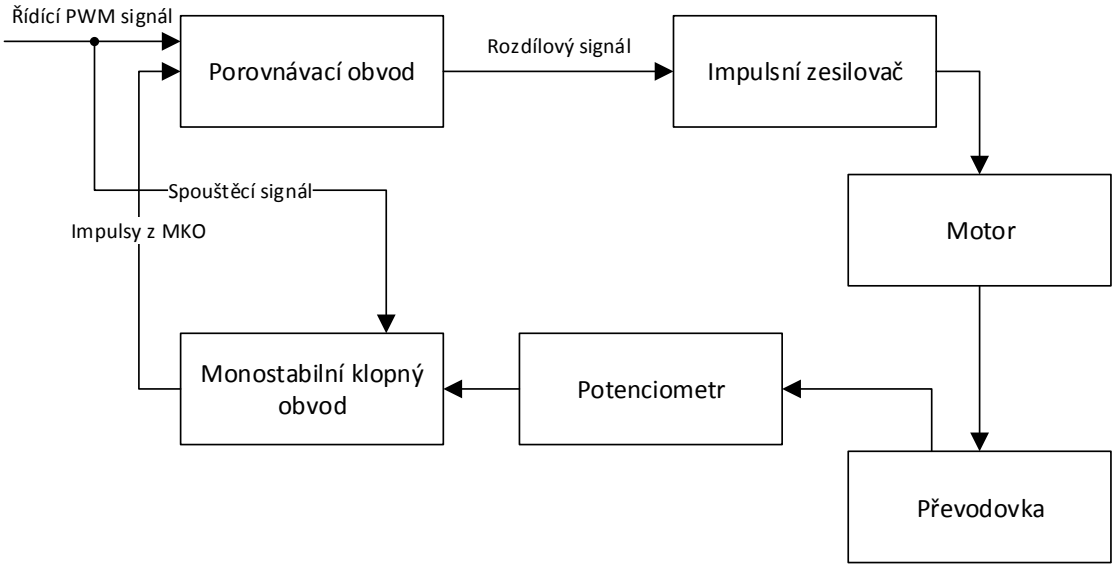
Servomotor je elektrické zařízení, u kterého lze na rozdíl od běžného motoru nastavit přesnou polohu natočení osy. Úhel natočení hřídele je dán řídicím signálem. Modelářské servomotory jsou na svoje malé rozměry velice výkonné a díky své ceně a jednoduchosti jsou velmi rozšířené zejména v robotice. Jsou složeny ze čtyř hlavních prvků: stejnosměrného elektromotoru, převodovky, potenciometru a řídicí elektroniky. Pro připojení slouží 3 vodiče, kde červený je napájení, černý zem a žlutý vodič je řídicí – pulsně šířkově modulovaný (PWM) signál.



Obrázek 4: Složení modelářského servomotoru [12]

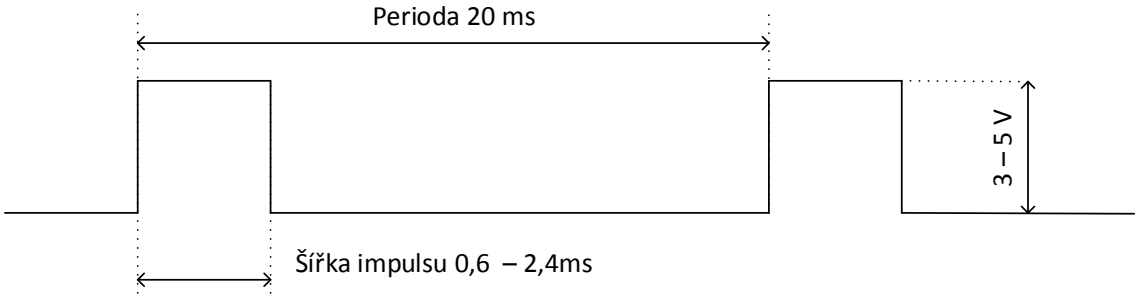
Modelářské servomotory jsou napájeny stejnosměrným napětím 4,8 až 6 V. Typický servomotor je schopen natáčet hřídel od 0 do 180°. Speciální úpravou je však možné motor modifikovat pro kontinuální činnost. Servomotory lze rozdělit na dva druhy; analogové a digitální. Rozdíl mezi nimi je v použití řídicí elektroniky. Digitální serva jsou řízena mikroprocesorem a jsou mnohem přesnější, jejich nevýhodou je však vyšší odběr proudu.

Základem řídicího obvodu je monostabilní klopný obvod (MKO), který je spuštěn vstupním signálem. Potenciometr slouží jako zpětná vazba o natočení hřídele. Poloha potenciometru určuje délku impulsu generovaného z MKO. Tento impuls je opačné polarity než je vstupní řídicí impuls. Porovnáním těchto dvou signálů vzniká rozdílový signál, který se zesílí v impulsním zesilovači a roztočí motor příslušným směrem. K pootočení motoru nedojde, pokud se impulsy rovnají; rozdílový signál je nulový. Pro lepší představu o funkci řídicího obvodu poslouží *Obrázek 5*.



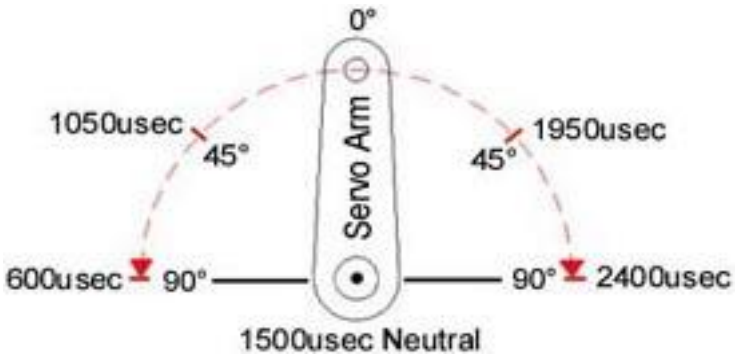
Obrázek 5: Blokové schéma řídicího obvodu analogového servomotoru

Jak již bylo zmíněno výše, modelářská serva jsou řízena PWM signálem. Frekvence tohoto signálu má být 50 Hz, šířka pulsu se musí pohybovat v rozmezí 0,6 – 2,4 ms a amplituda signálu v rozmezí 3 – 5 V.



Obrázek 6: Řídicí PWM signál servomotoru

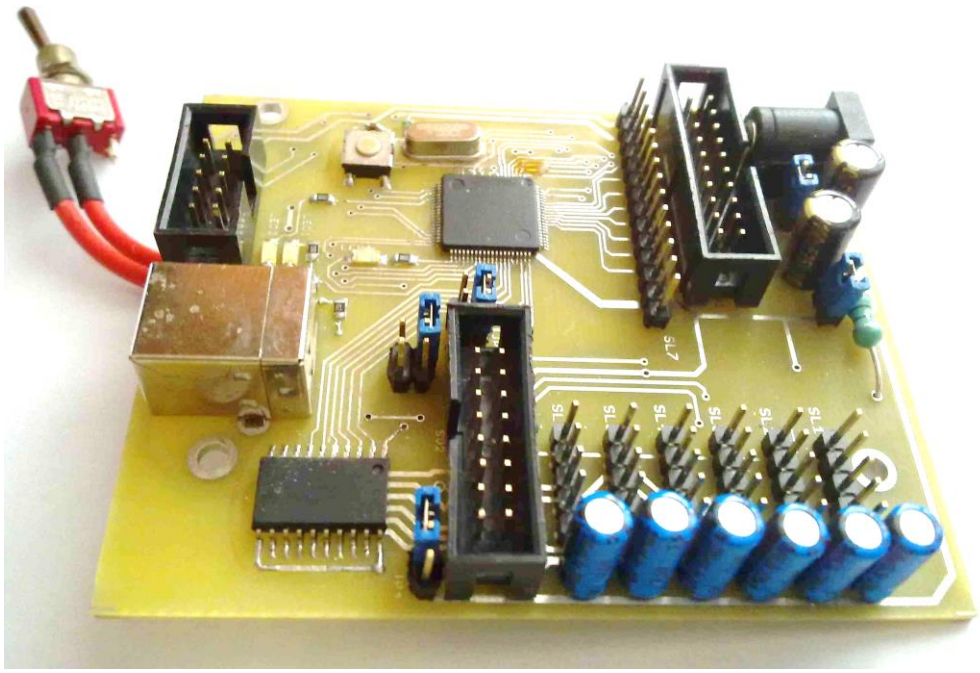
Servomotory se natáčí ve směru hodinových ručiček, tzn. že šířka impulsu 0,6 ms odpovídá levé krajní poloze a 2,4 ms pravé krajní poloze. Tyto hodnoty jsou však pouze orientační a u každého jednotlivého servomotoru je nutno nejprve experimentálně ověřit velikost těchto krajních hodnot, neboť najetí za krajní polohu může způsobit poškození ozubených koleček v převodovce.



Obrázek 7: Souvislost mezi šířkou signálu a úhlem natočení hřídele serva [15]

3.2 Deska řídicí jednotky

Jako řídicí jednotka manipulátoru je využita již vyrobená a plně funkční DPS (deska plošných spojů). Tato DPS byla vyrobena v minulých letech na Ústavu automatizace a měřicí techniky VUT v Brně. K desce byla dodána i dokumentace ve formě schématu a osazovacího plánu (viz příloha).



Obrázek 8: Deska řídicí jednotky

3.2.1 Obvodové řešení desky řídicí jednotky

Deska vyžaduje externí napájení 5 V. Na desce je stabilizátor napětí na 3,3 V. Stabilizovaným napětím je napájen mikrokontrolér a další pomocné obvody. Externí napětí 5 V je přímo přivedeno na konektory pro připojení servomotorů a využito k jejich výkonovému napájení. Pro nahrávání a ladění programu slouží JTAG konektor. Pro širší použitelnost desky lze využít rozšiřující konektor, který je propojen s GPIO porty mikrokontroléru a také je na něj přivedeno napájení 5 V a 3,3 V. DPS dále obsahuje převodníky napěťových úrovní pro RS-232, RS-422 a RS-485.

Dále šest 5pinových konektorů pro připojení šesti servomotorů, kde v každé pětici 2 piny slouží pro napájení, 2 piny pro zpětnou vazbu a 1 pin pro řídicí signál. Výstupní PWM řídicí signál z mikrokontroléru je na tento konektor přiveden přes klopný obvod D (KO-D). Signál zpětné vazby od servomotorů je přes oddělovací zesilovač přiveden na A/D převodník mikrokontroléru. (Zpětná vazba od servomotorů není při řízení využita, protože používané servomotory nemají tento signál vyveden.)

3.2.2 Mikrokontrolér LPC1756

Nejdůležitějším prvkem řídicí desky je Mikrokontrolér LPC1756 od výrobce NXP Semiconductors. Mikrokontrolér je založen na 32bitovém jádře ARM Cortex-M3. Je uložen v 80pinovém pouzdře a napájen stabilizovaným napětím 3,3 V. Jeho základní vlastnosti a vybavení jsou následující:

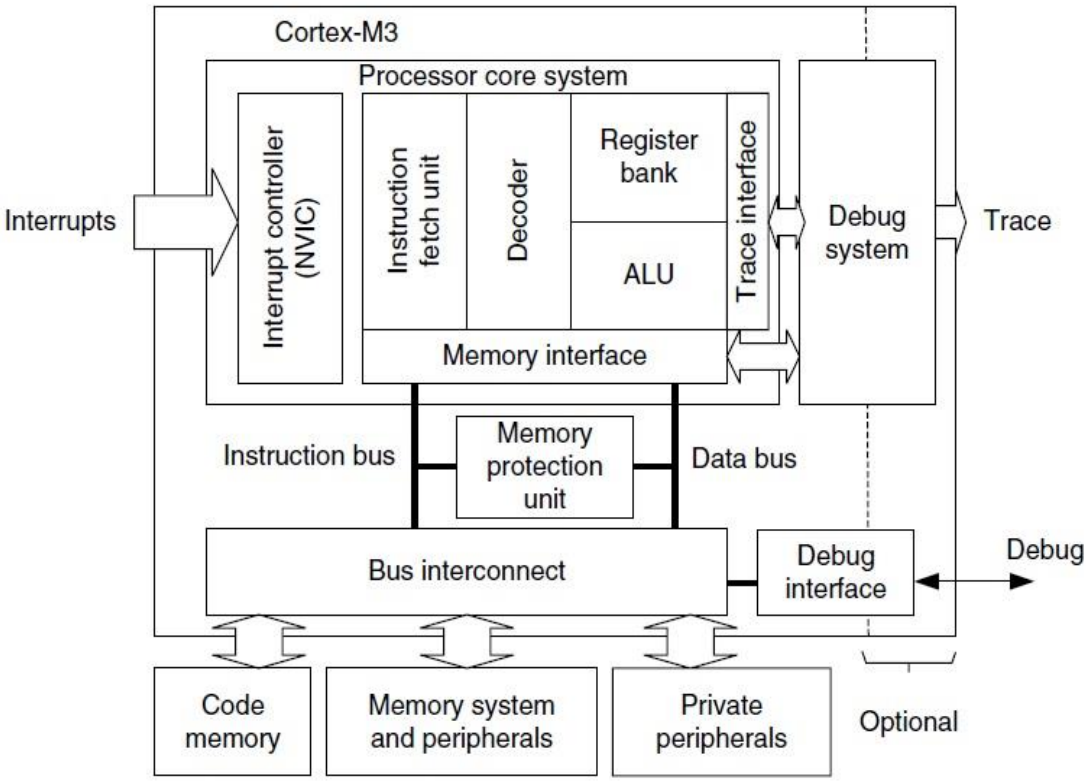
- Frekvence CPU až 100 MHz
- 256 kB FLASH a 32 kB RAM paměť
- 4x 32bitový časovač
- 6x 12bitový A/D převodník
- 6 PWM kanálů
- Komunikační rozhraní USB, CAN, UART, SPI, I^2C
- Teplotní rozsah -40 °C až +85 °C

3.2.3 Jádro ARM Cortex-M3

Společnost ARM (Advanced RISC Machines Ltd.) vznikla v roce 1990 jako společný podnik firem Apple Computer, Acorn Computer Group a VLSI Technology. Jedná se o poměrně rozšířenou 32bitovou architekturu procesorů založenou na RISC filosofii. V současné době je vyráběno více jak 2 miliardy ARM procesorů každý rok. Společnost ARM však tyto procesory přímo nevyrábí, namísto toho ale prodává licence svým obchodním partnerům, kteří využívají architekturu ARM ve svých mikrokontrolérech.[20]

Jádro ARM Cortex-M3 bylo uvedeno na trh v roce 2006 a bylo první z rodiny procesorů ARMv7.[20] Písmeno „M“ v názvu značí, že jde o profil M, který je určen

především pro mikrokontroléry, kde je potřebný dostatečný výkon procesoru, rychlá odezva na přerušení, ale hlavním kritériem je nízká cena a spotřeba energie. Jádro je vystavěno na Harvardské architektuře, což znamená, že má oddělené sběrnice pro instrukce a pro data (viz *Obrázek 9*). [19]Této vlastnosti se využívá při *pipeliningu* (zřetězování instrukcí nebo průtokové zpracování instrukcí). *Pipelining* umožňuje rychlejší vykonávání programu, protože díky odděleným sběrnicím je možné paralelně načítat novou instrukci a zároveň ukládat data výsledku do paměti. *Pipelining* v této architektuře má tři fáze: načtení instrukce, dekódování instrukce a vykonání instrukce.



Obrázek 9: Blokové schéma jádra Cortex-M3 [20]

Jádro má implementovanou novou instrukční sadu Thumb-2, zpětně je však kompatibilní i se sadou Thumb. Instrukční sada Thumb-2 má oproti sadě Thumb výhodu, že značně snižuje velikost kódu, protože obsahuje jak 16bitové, tak 32bitové instrukce. Kombinuje tedy výhody klasické 32bitové ARM instrukční sady a 16bitové instrukční sady Thumb.[14]

Rozhraní jádra pro ladění a trasování kódu podporuje standardní JTAG (Joint Test Action Group – pětivodičové rozhraní) a SWD (Serial Wire Debug – dvouvodičové rozhraní). Dalším prvkem architektury je konfigurovatelná jednotka NVIC (Nested Vectored Interrupt Controller), která slouží pro řízení až 240 přerušení s 256 stupni priorit.

3.3 Možnosti programového řešení

Při programování embedded systémů je většinou požadováno plnit úkoly, provádět výpočty a reagovat na vnější události (např. stisknutí tlačítka, příjem dat na sériové lince) v definovaných časových intervalech. Jestliže je systém schopen plnit vše v těchto časových intervalech, mluvíme o něm jako o Real-time systému. K vytvoření takovéhoto Real-time systému je zapotřebí jednak vhodný hardware, ale především software. V následujících podkapitolách budou stručně probrány dva přístupy, které jsou při programování Real-time systému využívány.

3.3.1 Řízení v supersmyčce

Základním způsobem vytvoření řídicího programu je použít princip tzv. supersmyčky neboli cyklického vykonávání programu. Nejjednodušší variantou, která se obejde i bez přerušovacích rutin, je čistě sekvenční vykonávání programu. Ukázkou je následující kód v jazyce C:

```
void main(void)
{
  Init();
  while(1)
  {
       Task_A();
       Task_B();
       Task_C();
  }
}
```

Po startu programu je zavolána funkce `Init()`, která provede potřebné inicializace, jakými jsou např. používané periferie, proměnné, frekvence CPU atd., poté program vstoupí do nekonečné smyčky, kde jsou postupně volány jednotlivé funkce. Výhodou tohoto řešení je jeho velice snadná implementace, a protože zde není potřeba žádná režie, tak také rychlost vykonávání programu. Doba vykonání jednoho cyklu je dána součtem časů vykonání jednotlivých funkcí. Z této vlastnosti o součtu časů však vyplývá velká nevýhoda tohoto řešení, a tím je velice obtížné přesné časování úloh, a to zejména, pokud mají volané funkce nekonstantní dobu vykonávání. Další nevýhodou je velmi špatná schopnost reagovat na externí události, které mohou do systému vstupovat asynchronně. Vzhledem k těmto vlastnostem je tento způsob vytváření řídicího programu vhodný pouze pro velice jednoduché aplikace, které navíc nemusí reagovat na externí události.

Pokročilejším řešením je kombinace supersmyčky a obslužných rutin přerušení. Představme si systém jako v předchozím případě, kde se provádí cyklické vykonávání

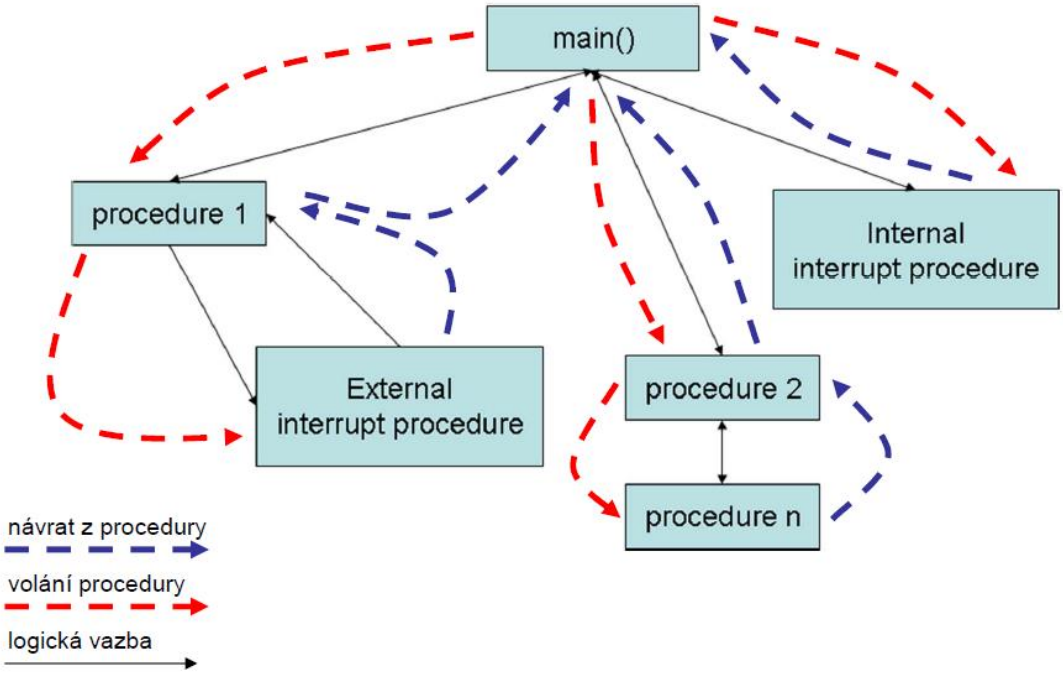
úloh `Task_A-Task_C` a navíc blikání LED diodou s periodou 100 ms, které lze zapínat a vypínat stiskem tlačítka. Kód pro tento systém by mohl vypadat následovně:

```
void main(void)
{
  Init();
  while(1)
  {
       Task_A();
       Task_B();
       Task_C();
  }
}

void SysTick_Handler(void) /*100 ms*/
{
  if(blinking==ON){
       ToogleLED();
  }
}

void GPIO_IRQHandler(void) /*button press*/
{
  if(blinking==ON){
       blinking=OFF;
  }
  else{
       blinking=ON;
  }
}
```

V programu se využívá dvou zdrojů přerušení. Prvním je systémový časovač, který vyvolává obslužnou rutinu `SysTick_Handler` periodicky každých 100 ms, kde se provádí rozsvěcování a zhasínání LED diody. Druhým zdrojem přerušení je stisk tlačítka připojeného k GPIO portu vyvolávající rutinu `GPIO_IRQHandler`. V obsluze tohoto přerušení se provede nastavení globální proměnné `blinking` na opačnou hodnotu, než kterou proměnná měla před zavoláním rutiny. Řídicí program implementovaný tímto způsobem by tedy měl zaručit real-time chování tohoto systému. Je nutné si uvědomit, že ukázaná implementace byla řešením pro stále velice jednoduchý systém. Řídicí program pro obecnější systém je schematicky naznačen na *Obrázek 10.*

Obecně lze o řízení v supersmyčce říci, že pokud je použito na jednoduché systémy, jedná se o velice snadné, rychlé a intuitivní řešení, kde není potřeba žádný operační

systém. Na druhé straně není tento způsob vhodný pro složitější úlohy, kde lze real-time chování systému zaručit obtížně, a to jen velice pečlivým návrhem. Další nevýhodou je ten fakt, že složitější úpravy v programu vedou k časovému rozložení celého systému.[6]

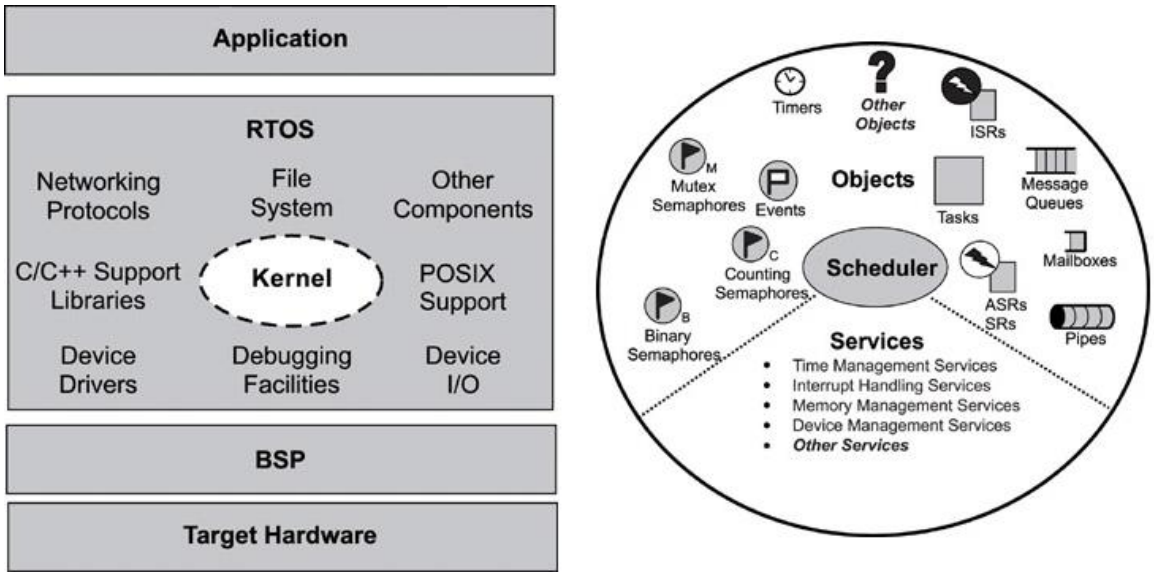


Obrázek 10: Řízení v supersmyčce[6]

3.3.2 Operační systém reálného času

Mnohem větší komfort při tvorbě řídicího programu nám nabízejí operační systémy. Operační systém je soubor programů, které vykonávají řízení a dohled nad běžícími programy. Jejich úkolem je správa systémových prostředků, jimiž jsou procesor, paměti a periferie. Je-li operační systém schopen pracovat v definovaných časových intervalech, jedná se o operační systém reálného času (RTOS).

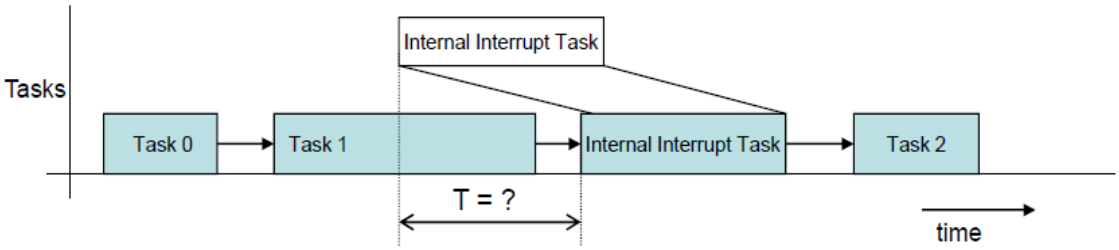
Jak již bylo uvedeno výše, řídicí program musí většinou zvládat vykonávat více úloh najednou. Implementovat všechny tyto úlohy do jedné supersmyčky bývá v mnoha případech značně komplikované. Operační systém nám však dává možnost rozdělit tyto úlohy na jednotlivé bloky, které budou vykonávány paralelně. Stále jsme ale ovlivněni faktem, že k dispozici je pouze jeden procesor, to znamená, že v jeden okamžik může běžet pouze jedna jediná úloha. Paralelismu je však dosahováno rychlým přepínáním mezi úlohami, což vytváří zdání současného běhu neboli multitaskingu.[7]



Obrázek 11: Blokové schéma RTOS (vlevo) a jeho kernelu (vpravo) [7]

Základem operačního systému je jeho systémové jádro neboli kernel. Většina RTOS určená pro malé embedded zařízení je pouze kernel samotný a neobsahuje již žádné další moduly, které jsou vyznačeny na *Obrázek 11* vlevo. Klíčovým prvkem každého kernelu je jeho plánovač úloh (angl. Scheduler). Ten určuje, která z úloh se bude v daný moment vykonávat. Plánovač lze rozdělit na dvě základní kategorie podle toho, který typ multitasking je použit.

- Nepreemptivní plánovač - Jednotlivé úlohy jsou většinou podle své priority řazeny do zásobníku a postupně vykonávány. Každá úloha dostane procesor a je sama zodpovědná za jeho odevzdání. Plánovač má minimální nebo žádnou možnost právě vykonávanou úlohu přerušit, a dát tak příležitost jiné úloze.

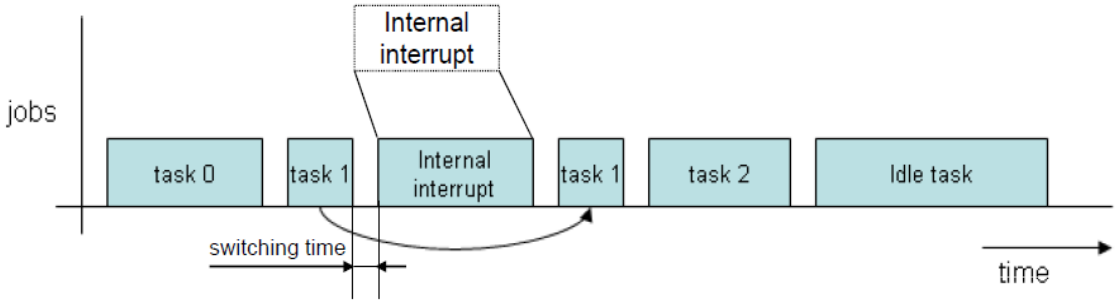


Obrázek 12: Nepreemptivní vykonávání úloh[6]

Z *Obrázek 12* vyplývá, že jestliže je vykonávána úloha 1 a přijde požadavek na přerušení, jeho obslužná rutina musí čekat na procesor neznámý čas T daný úlohou 1. Chování celé úlohy může být tedy nedeterministické a tím pádem real-time chování těžko dosažitelné. Na druhou stranu se jedná o velice

jednoduchý plánovač na implementaci a nedochází k plýtvání času procesoru častým přepínáním mezi úlohami. Nepreemptivní plánovač je tedy velmi podobný způsobu řízení ve smyčce s tím rozdílem, že zde je k dispozici více úlohového prostředí s možností nastavení priorit.

- Preemptivní plánovač - O přidělávání procesoru rozhoduje plánovač. Právě běžící úloha může být v kterémkoliv okamžiku přerušena a procesor poskytnut vyskytnuvší se úloze s logicky vyšší prioritou tak, jak je znázorněno na *Obrázek 13*.



Obrázek 13: Preemptivní vykonávání úloh[6]

S preemptivním plánovačem lze docílit real-time chování celé aplikace. Záporem tohoto řešení je však jeho složitost a také čas, který je ztracen při přepínání úloh - obecněji lze říci při režii.

Je nutné si uvědomit, že prosté použití RTOS pro vytváření řídicího programu nám nezaručí real-time chování celého systému. Vývojář musí vždy zvážit možnosti daného RTOS na konkrétní HW platformě a tyto dostupné prostředky poté vhodným způsobem využít k vytvoření real-time systému. [6]

Vzhledem k nárokům na firmware pro řídicí jednotku manipulátoru jasně vyplívá, že správnou volbou programového řešení je použití operačního systému reálného času. Konkrétně byl vybrán FreeRTOS, a to z důvodu jeho oblíbenosti, velice dobře vedené dokumentace a také na základě práce [11], která se mimo jiné zabývá i testováním třech RTOS pro embedded aplikace, právě FreeRTOS vychází z těchto testů nejlépe.

4 FREERTOS

Obrázek 14: Logo FreeRTOS[3]

FreeRTOS je malý operační systém reálného času určený pro embedded aplikace. Vyvíjený je společností Real Time Engineers Ltd. a je volně šiřitelný jako open-source. Lze ho využít i pro tvorbu komerčních aplikací. Velkou výhodou sytému je jeho portovatelnost. Momentálně totiž podporuje 34 mikroprocesorových architektur různých výrobců, těmi nejvýznamnějšími jsou například Atmel, Freescale, Microchip, NXP, ST Microelectronic a Texas Instruments. Zdrojové kódy jsou napsány v jazyce C. [3]

Jelikož se jedná o malý a jednoduchý RTOS, je tvořen pouze jádrem samotným. Ani řízení běhu programu zde není rozděleno na klasické procesy sestávající se z vláken, jako jsme zvyklí u běžných OS pro PC, ale je rozděleno pouze na jednotlivé úlohy.

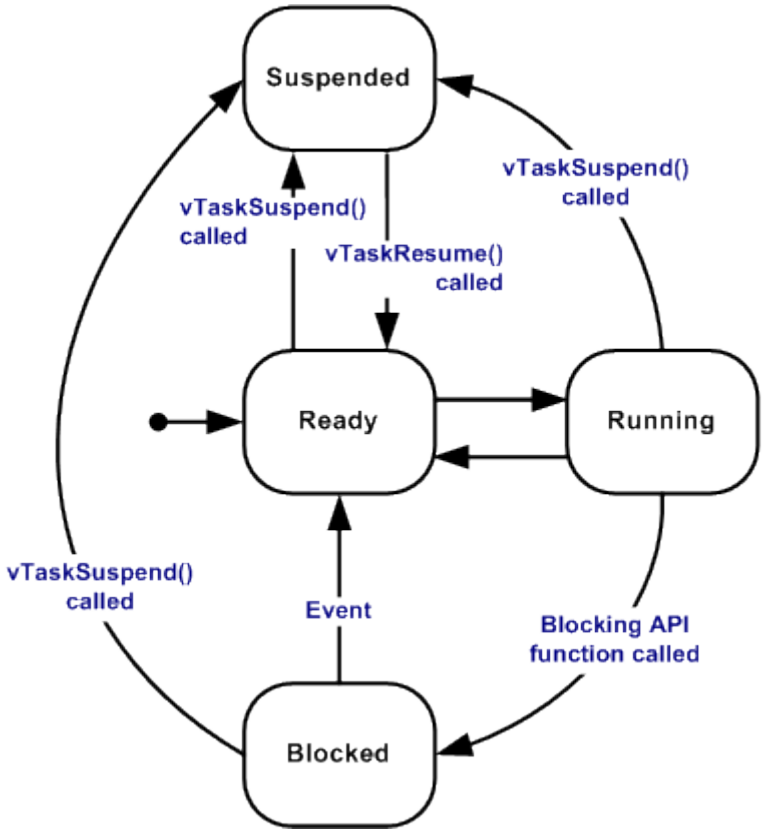
V následujících podkapitolách je provedeno základní seznámení s prvky jádra systému, které jsou využívány při tvorbě firmware pro řídicí jednotku manipulátoru.

4.1 Úlohy a jejich plánování

Základem jádra systému je plánovač úloh. Ten je zodpovědný za rozhodování, která z úloh by měla být vykonávána v daném čase. FreeRTOS plánovač má tu vlastnost, že ho lze nakonfigurovat na nepreemptivní, preemptivní a nebo hybridní.

Úlohy se mohou nacházet v různých stavech a mezi těmito stavy přecházet tak, jak znázorňuje stavový diagram úlohy *Obrázek 15*. Tyto stavy jsou:

- **Running** – Úloha je právě vykonávaná a využívá procesor. V tomto stavu tedy může být v jeden okamžik pouze jedna úloha.
- **Ready** – Úlohy v tomto stavu jsou připraveny k vykonávání. Musejí ale čekat, protože ve stavu *running* je momentálně úloha s vyšší prioritou.
- **Blocked** – Tyto úlohy nejsou připraveny pro plánování, protože čekají buď na nějakou externí, nebo časovou událost. Pokud úloha zavolá funkci `vTaskDelay`, bude blokována a čeká, dokud nevyprší čas zpoždění. Toto je příklad časové události. Externí událostí je zpravidla čekání na semafory, fronty a mutexy.
- **Suspended** – Úlohy v tomto stavu také nejsou připraveny pro plánování. Úlohy do tohoto stavu vstoupí a vystoupí pouze voláním funkcí `vTaskSuspended` a `vTaskResume`.



Obrázek 15: Stavový diagram úlohy ve FreeRTOS [3]

Vytvoření nové úlohy se provádí voláním funkce `xTaskCreate` a nastavují se tyto parametry:

- Název – string hodnota, která se zobrazuje jako název úlohy po zavolání funkce pro výpis úloh `TaskList`.
- Exekuční kód – funkce, kde je napsán vlastní kód úlohy.
- Velikost zásobníku – vyhrazená paměť z FreeRTOS haldy pro zásobník úlohy. Velikost se udává ve slovech používaného mikroprocesoru. (U ARM Cortex-M3 je velikost slova 4B.)
- Priorita – na základě priority plánovač rozhoduje, která z úloh ve stavu *Ready* se dostane do stavu *Running*. Vyšší hodnota znamená vyšší prioritu.
- Vstupní parametry – vstupní parametry pro funkci s exekučním kódem.
- Reference (handle) – ukazatel pro případné manipulování s úlohou.

Automaticky vytvářenou úlohou je *IDLE task*, který má nejmenší prioritu 0 a je volán ve zbývajícím čase, kdy není potřeba vykonávat žádnou jinou úlohu.

4.2 Fronty

Pro bezpečné předávání dat mezi úlohami nebo mezi obsluhou přerušení a úlohou slouží ve FreeRTOS fronty zpráv. Jedná se v podstatě o jednoduchý FIFO zásobník, kterému je definován datový typ (např. char, integer, definovaná struktura) a jeho délka. Přenos dat pomocí fronty je realizovaný kopírováním jednotlivých položek (ne předáváním ukazatele).

Při pokusu čtení z fronty může být úloha blokována, a to z těchto dvou důvodů:

- Ve frontě nejsou žádná data a čeká se na její naplnění
- Z fronty se snaží číst více úloh najednou, a přístup k ní tedy dostává pouze úloha s nejvyšší prioritou, ostatní jsou blokovány

Obdobná situace nastává při pokusu čtení dat z fronty, kde k blokaci dochází v těchto případech:

- Fronta je plná, čeká se na uvolnění místa
- do fronty se snaží zapisovat více úloh najednou a přístup získává opět pouze úloha s nejvyšší prioritou.

Pro vytvoření nové fronty slouží funkce `xQueueCreate`, pro zapisování do fronty `xQueueSend` a pro čtení dat z fronty `xQueueRecieve`. U funkcí pro zápis a čtení z fronty je jedním parametrem čas, který udává maximální dobu, po kterou může být úloha blokována čekáním na frontu. Tento parametr se stejným významem se používá i u funkcí pro čekání na semafor a na mutex.

4.3 Binární semafory

Binární semafor je velice podobný mutexu s tím rozdílem, že mutex má mechanismus pro dědění priority, zatímco binární semafor tento mechanismus nemá. Tato skutečnost dělá binární semafor vhodným prvkem pro vzájemnou synchronizaci úloh.

Binární semafor je v podstatě fronta, která má pouze jeden prvek, u kterého nás navíc nezajímá jeho obsah, důležité je pouze, jestli je a nebo není (od toho vzniká název binární).

Co vlastně synchronizace úloh znamená, je možné si ukázat na situaci, která vznikla při vytváření firmware pro řídicí jednotku manipulátoru: jedna úloha má za úkol zpracovávat pakety přijaté po sériové lince a druhá úloha musí poté, co byl paket zpracován, vykonat pohyb manipulátorem a poté opět čekat na další příchozí paket. Synchronizace pomocí binárních semaforů je tedy vyřešena tak, že první úloha po zpracování paketu dává semafor voláním funkce `xSemaphoreGive` a druhá úloha čeká (je blokována) na přijetí semaforu funkcí `xSemaphoreTake`.

Nejprve je však nutné binární semafor vytvořit pomocí funkce `xSemaphoreCreateBinary`

4.4 Mutexy

Jak bylo uvedeno v předchozí podkapitole, mutex se od binárního semaforu liší pouze mechanismem dědění priorit. Jak dědění priorit funguje, lze popsat na jednoduchém příkladu: úloha A s nízkou prioritou získá mutex, poté o něj požádá úloha B, která má vysokou prioritu. Aby nedošlo k problému inverze priorit, zdědí úloha A prioritu úlohy B.

Z tohoto důvodu je mutex vhodný pro ochranu prvků, ke kterým může přistupovat více úloh najednou. Ve firmware se mutex používá jako ochrana všech globálních proměnných.

Vytvoření mutexu se provádí voláním funkce `xSemaphoreCreateMutex`. Chceme-li číst nebo zapisovat do globální proměnné, provedeme v programu tuto sekvenci:

- Získání mutexu voláním funkce `xSemaphoreTake`
- Operace s proměnnou
- Uvolnění mutexu voláním funkce `xSemaphoreGive`

4.5 Časovače

Softwarové časovače ve FreeRTOS slouží pro vykonání určité funkce za daný časový okamžik v budoucnosti a nebo pro periodické vykonávání funkce (tato funkce se nazývá *callback funkce*). Časovači lze stejně jako úloze nastavit prioritu a velikost zásobníku. Tyto dva parametry musí být však pro všechny použité časovače stejné, protože se nezadávají jako parametr funkce pro vytvoření časovače `xTaskCreate`, ale jsou nastaveny na pevnou hodnotu v konfiguračním souboru `FreeRTOSConfig.h`. Dalšími parametry, tentokrát již nastavované při vytváření časovače, jsou:

- Název
- Perioda
- zapnutí/vypnutí *AutoReload* – Jestliže je *AutoReload* zapnutý, *callback funkce* časovače je volána periodicky.
- ID – číslo, kterým je časovač v systému jednoznačně označen
- *Callback funkce* – exekuční kód časovače

Kód *callback funkce* by měl být vykonatelný v co nejkratším čase. Neměly by v něm být tedy volány funkce typu `vTaskDelay` a funkce pro přístup k frontám a semaforům, které specifikují nenulový blokovací čas.

4.6 Konfigurace systému

Konfigurace systému se provádí v souboru `FreeRTOSConfig.h`. Nyní budou uvedena důležitá nastavení v tomto souboru a jejich konkrétní parametry tak, jak je FreeRTOS nakonfigurován ve firmware pro řídicí jednotku manipulátoru.

V prvé řadě se nastaví plánovač úloh jako preemptivní.

```
#define configUSE_PREEMPTION       1
```

Frekvence, na které pracuje procesor

```
#define configCPU_CLOCK_HZ         100000000
```

Základní perioda časování systému – Zde se musí zvolit kompromis mezi přesností a zbytečným zatěžováním procesoru frekventovanou režií systému. Byla tedy nastavena 1 ms, která je pro tuto aplikaci plně dostačující.

```
#define configTICK_RATE_HZ         1000
```

Počet FreeRTOS priorit

```
#define configMAX_PRIORITIES       10
```

Minimální velikost zásobníku pro úlohu – hodnota se udává ve slovech

```
#define configMINIMAL_STACK_SIZE   64
```

Velikost paměti vyčleněné pro FreeRTOS – hodnota se udává v bytech

```
#define configTOTAL_HEAP_SIZE      8192
```

Priorita softwarového časovače

```
#define configTIMER_TASK_PRIORITY  5
```

Velikost zásobníku pro *callback* funkci softwarového časovače – hodnota se udává ve slovech

```
#define configTIMER_TASK_STACK_DEPTH        128
```

NVIC priorita pro jádro FreeRTOS – NVIC priorita má převrácený význam než FreeRTOS priorita. Vyšší hodnota tedy znamená nižší priorita. Hodnota 255 je nejmenší možná priorita systému. Kterékoliv přerušení v mikrokontroléru bude mít tedy vyšší prioritu než jádro FreeRTOS.

```
#define configKERNEL_INTERRUPT_PRIORITY     255
```

Určení hranice pro nastavení NVIC priority, do které je možno bezpečně volat FreeRTOS API funkce s přívlastkem *FromISR* - Obslužné rutiny přerušení (ISR) s malými NVIC priority (nízké hodnoty 1,2,3…) nesmějí být zatěžovány voláním FreeRTOS API funkcí.

Pouze ISR malé NVIC priority (vysoké hodnoty 32,33,34…) mohou bezpečně volat FreeRTOS API funkce s přívlastkem *FromISR*.

```
#define configMAX_SYSCALL_INTERRUPT_PRIORITY 32
```

Jelikož je používán mikrokontrolér s jádrem ARM Cortex-M3, je nutné konfigurační soubor doplnit o tyto tři makra definující obslužné rutiny systémových přerušení

```
#define vPortSVCHandler SVC_Handler
#define xPortPendSVHandler PendSV_Handler
#define xPortSysTickHandler SysTick_Handler
```

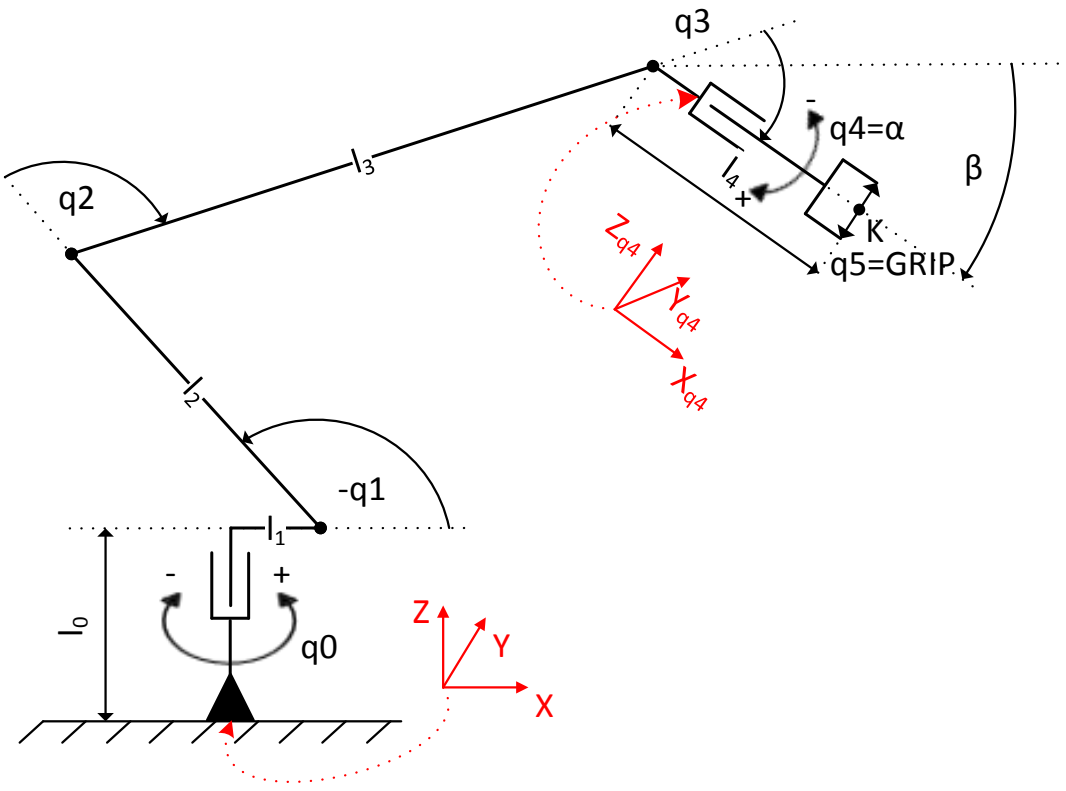
Ostatní nastavení v tomto souboru slouží pro začlenění a vyčlenění používaných a nepoužívaných funkcí FreeRTOS. Vyčleněním nepoužívaných funkcí by se mělo docílit toho, aby FreeRTOS zabíral v paměti co nejméně prostoru. Většina linkerů je však schopna nepoužívané funkce vyčlenit i bez této konfigurace. Konfigurace tedy může sloužit pro programátora aplikace, který tak má větší přehled nad využívanými funkcionalitami FreeRTOS.

5 KINEMATICKÉ ÚLOHY MANIPULÁTORU

V následujícím textu platí tato pravidla a zápisy:

- Pokud není uvedeno jinak, je vše vztaženo k počátku souřadného systému (bázi)
- Pozice koncového bodu je značena x_k,y_k,z_k
- Pozice všech kloubů je značena s dolním indexem jeho pořadí. Např. pozice kloubu q3: x_3,y_3,z_3

Počátek souřadného systému (báze) je zvolen na úrovni podložky pod kloubem q0. (viz *Obrázek 16*) Vektor pozice koncového bodu robota K je označen $\mathbf{P} = [x,y,z,\alpha,\beta,g]^T$. Kde složky *x,y,z* představují souřadnice koncového bodu v souřadném systému (bázi). α představuje rotaci zápěstí v ose X_{q4}. β je úhel, který svírá zápěstí s rovinou tvořenou osami *X* a *Y*. Poslední složka *g* představuje sevření uchopovacích kleští.



Obrázek 16: Kinematický model manipulátoru

Jak uvádí Tabulka 1, všechny klouby mají rozsah pohybu 180°. Pohyb posledního kloubu q5 je pomocí mechanismu převeden na pohyb posuvný, rozsah tohoto pohybu je 7 - 37mm. Délky jednotlivých ramen jsou následující: l_0=70 mm; l_1=18 mm; l_2=145 mm; l_3=186 mm; l_4=100 mm.

5.1 Přímá kinematická úloha

Pro vyřešení přímé kinematické úlohy využijeme matice homogenní transformace (MHT). Tato matice nám vyjadřuje pózu systému 1 vzhledem k systému 0. Postupně tedy budou vypsány jednotlivé MHT určující vztah sousedních systémů. Řešení přímé kinematické úlohy poté dostáváme vynásobením jednotlivých MHT.

$$H_{01} = rot_z(q0) \cdot trans_z(l_0) = \begin{bmatrix} \cos q0 & -\sin q0 & 0 & 0 \\ \sin q0 & \cos q0 & 0 & 0 \\ 0 & 0 & 1 & l_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$H_{12} = rot_x(q1) \cdot trans_x(l_1) = \begin{bmatrix} \cos q1 & 0 & \sin q1 & l_1 \\ 0 & 1 & 0 & 0 \\ -\sin q1 & 0 & \cos q1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$H_{23} = rot_x(q2) \cdot trans_x(l_2) = \begin{bmatrix} \cos q2 & 0 & \sin q2 & l_2 \\ 0 & 1 & 0 & 0 \\ -\sin q2 & 0 & \cos q2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$H_{34} = rot_x(q3) \cdot trans_x(l_3) = \begin{bmatrix} \cos q3 & 0 & \sin q3 & l_3 \\ 0 & 1 & 0 & 0 \\ -\sin q3 & 0 & \cos q3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$H_{4K} = trans_x(l_4) = \begin{bmatrix} 1 & 0 & 0 & l_4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

$$H_{0K} = H_{01} \cdot H_{12} \cdot H_{23} \cdot H_{34} \cdot H_{4K} \tag{6}$$

$$H_{0K} = \begin{bmatrix} c_{123}c_0 & -s_0 & s_{123}c_0 & c_0(l_1 + l_3c_{12} + l_2c_1 + l_4c_{123}) \\ c_{123}s_0 & c_0 & s_{123}s_0 & s_0(l_1 + l_3c_{12} + l_2c_1 + l_4c_{123}) \\ -s_{123} & 0 & c_{123} & l_0 - l_3s_{12} - l_2s_1 - l_4s_{123} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

V rovnici (7) je použito zjednodušeného zápisu funkcí. Např. zkratka $c_{123}c_0$ znamená: $\cos(q1 + q2 + q3) \cdot \cos q0$

Souřadnice koncového bodu manipulátoru v kartézském systému v závislosti na kloubových souřadnicích jsou dány posledním sloupcem výsledné matice H_{0K}.

$$\begin{aligned} x_K &= c_0(l_1 + l_3c_{12} + l_2c_1 + l_4c_{123})\ [mm] \\ y_K &= s_0(l_1 + l_3c_{12} + l_2c_1 + l_4c_{123})\ [mm] \\ z_K &= l_0 - l_3s_{12} - l_2s_1 - l_4s_{123}\ [mm] \end{aligned} \tag{8}$$

Úhel β dostaneme sečtením kloubových souřadnic *q1*,*q2*,*q3*. Prvky α a g se přímo rovnají kloubovým souřadnicím.

$$\begin{aligned} \alpha &= q4\ [°] \\ \beta &= q1 + q2 + q3\ [°] \\ g &= q5\ [mm] \end{aligned} \tag{9}$$

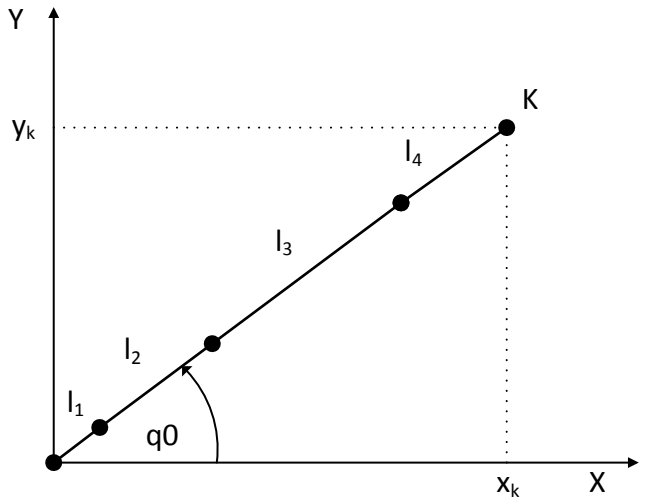
5.2 Inverzní kinematická úloha

Řídicí systém musí umožňovat po zadání polohy a orientace koncového bodu K vektorem **P** = [*x,y,z,α,β,g*]^T vypočet vektoru kloubových souřadnic **q**=[*q0,q1,q2,q3,q4,q5*]^T. Inverzní kinematická úloha bude mít vždy pouze jedno jediné realizovatelné řešení, a to z toho důvodu, že zároveň platí následující podmínky:

- vektor zadaných hodnot **P** má stejnou délku jako vektor výsledných kloubových souřadnic **q**
- rozsah kloubu q0 je pouze od -90° do 90°, kloubu *q1* od 0° do -180° a kloubu *q2* od 0° do 180°
- natočení a sevření uchopovacích kleští α a g se přímo rovná kloubovým souřadnicím *q4* a *q5*.

Výjimku ovšem tvoří všechny body ležící na ose z. Zde bude každý bod dosažitelný nekonečně mnoho způsoby (rotace v kloubu *q0* nijak neovlivní vektor **P**). Při implementaci driveru je tato situace vyřešena tak, že natočení v kloubu *q0* se ponechá stejné, jako bylo vypočtené u předchozího bodu. Tímto se zamezí zbytečnému pohybu v kloubu *q0*.

Nyní již přejdeme k odvození výpočtů jednotlivých kloubových souřadnic. Začneme prvním kloubem *q0*. Situace je názorněji představena na následujícím obrázku.

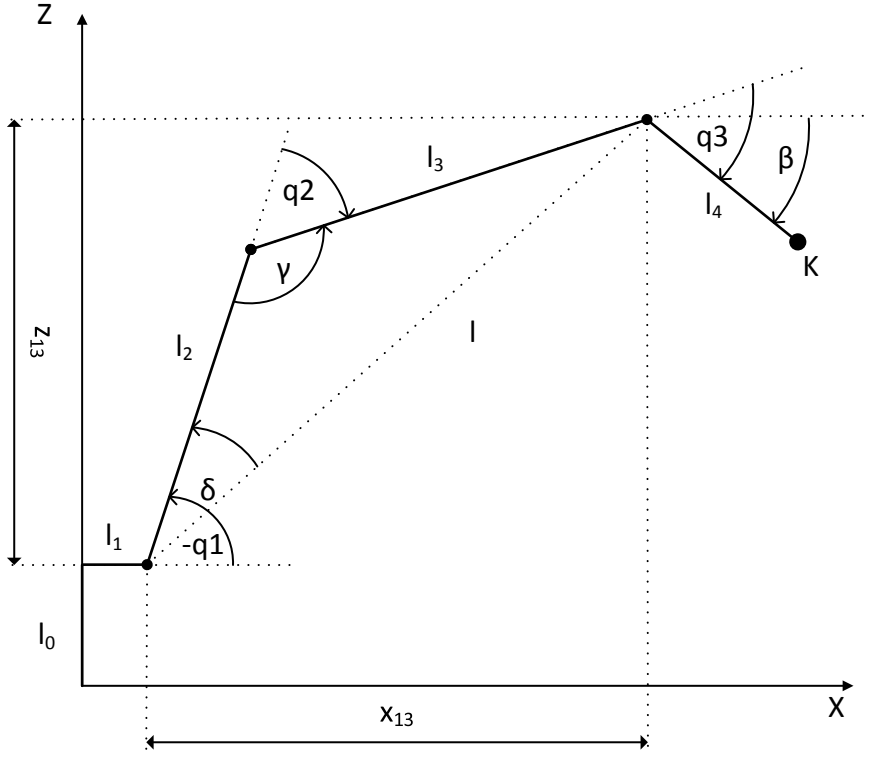


Obrázek 17: Pohled na model manipulátoru v rovině XY

Nyní lze snadno určit

$$q0 = Atan\left(\frac{y_k}{x_k}\right) \ [°] \tag{10}$$

Pro další výpočty pomůže *Obrázek 18*.



Obrázek 18: Pohled na manipulátor v rovině XZ

Aby bylo možné vypočítat kloubové souřadnice *q1* a *q2*, je nutno znát souřadnice bodu, ve kterém je kloub *q3*. Tento bod je možné vypočítat ze zadané pozice koncového bodu *K* a zadaného úhlu β, který značí úhel svírající s podložkou.

$$\begin{aligned} x_{13} &= x_k - l_4 \cos\beta \cos q0 - l_1 \cos q0 \ [mm] \\ y_{13} &= y_k - l_4 \cos\beta \sin q0 - l_1 \sin q0 \ [mm] \\ z_{13} &= z_k + l_4 \sin\beta - l_0 \ [mm] \end{aligned} \tag{11}$$

Tímto vzniká trojúhelník s vrcholy v kloubech *q1*,*q2*,*q3*. V tomto trojúhelníku jsou dány prostorové orientace dvou vrcholů a délky dvou stran. Z těchto údajů lze trigonometrickými vzorci dopočítat jednotlivé úhly.

Strana *l*:

$$l = \sqrt{x_{13}^2 + y_{13}^2 + z_{13}^2} \ [mm] \tag{12}$$

Použitím kosinové věty dostáváme

$$\cos\gamma = \frac{l_2^2 + l_3^2 - l^2}{2 l_2 l_3} = a \ [-] \tag{13}$$

Po vyjádření funkce cosinus jako arcus tangens dostáváme uhel γ

$$\gamma = Atan\left(\frac{\pm\sqrt{1-a^2}}{a}\right) \ [°] \tag{14}$$

Poté lze již velice snadno vyjádřit úhel *q2*

$$q2 = \pi - \gamma \ [°] \tag{15}$$

Použitím sinové věty dostáváme

$$\begin{aligned} \frac{l_3}{\sin\delta} &= \frac{l}{\sin\gamma} \ [mm] \\ \sin\delta &= \frac{l_3}{l}\sin\gamma = b \ [-] \\ \delta &= Atan\left(\frac{b}{\pm\sqrt{1-b^2}}\right) \ [°] \end{aligned} \tag{16}$$

Ze znalosti úhlu δ a souřadnic x_{13}, y_{13}, z_{13} pak dostáváme *q1*

$$q1 = Atan\left(\frac{z_{13}}{\sqrt{x_{13}^2 + y_{13}^2}}\right) + \delta\ [°] \tag{17}$$

Úhel *q3* je dán úhly *q1, q2, β*

$$q3 = \beta - q1 - q2\ [°] \tag{18}$$

Úhel *q4* je dán přímo úhlem natočení zápěstí v ose X_{q4}

$$q4 = \alpha\ [°] \tag{19}$$

A poslední kloubová souřadnice *q5*

$$q5 = g\ [mm] \tag{20}$$

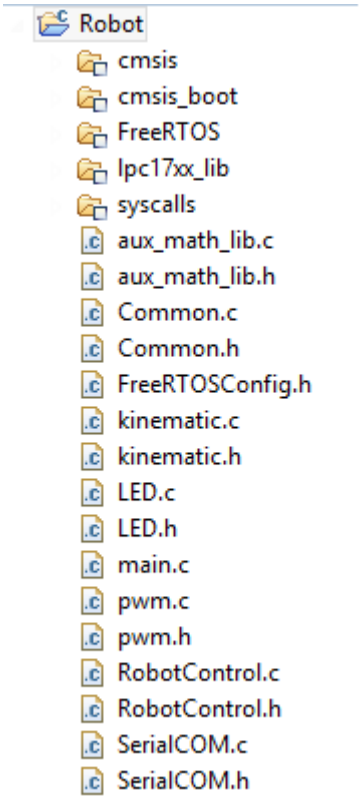
Výsledkem této kapitoly je tedy sada rovnic pro kloubové souřadnice, které vzešly řešením inverzní kinematické úlohy (rovnice (10)(15)(17)(18)(19)(20)) a sada rovnic pro kartézské souřadnice získané přímou kinematickou úlohou (rovnice (8)(9)). Následně budou tyto rovnice implementovány do firmware řídicí jednotky.

6 FIRMWARE PRO ŘÍDICÍ JEDNOTKU MANIPULÁTORU

Robotický manipulátor je řízen mikrokontrolérem LPC1756 tak, že mikrokontrolér generuje řídicí PWM signály pro jednotlivé servomotory a po sériové lince komunikuje s nadřazeným PC systémem. Jádro mikrokontroléru pracuje na maximální možné frekvenci 100 MHz. Pro tvorbu firmware bylo zvoleno vývojové prostředí CooCox CoIDE a operační systém FreeRTOS ver. 7.6.0. Program je psán v jazyce C a pro překlad do strojového kódu je volán GCC překladač. Strojový kód je do mikrokonroléru nahráván ColinkEx adaptérem s rozhraním USB/JTAG.

6.1 Struktura a úloha firmware

Jak již bývá při tvorbě řídicího programu zvykem, je i v tomto případě kód rozdělen do několika souborů. Takovéto rozčlenění do jednotlivých adresářů a souborů napomáhá nejenom snazší orientaci, ale také při překladu kódu, kdy je možné překládat jednotlivé části separátně nebo vytvářet knihovny. *Obrázek 19* nám znázorňuje složení firmware. V adresářích se zpravidla nacházejí zdrojové kódy převzaté. Jedná se o knihovní funkce pro jádro Cortex-M3 (cmsis), pro obsluhu periferií mikrokontroléru LPC1756 (lpc17xx_lib) a zdrojové kódy operačního systému FreeRTOS. Zbylé soubory umístěné mimo adresáře jsou již vytvořené v rámci této diplomové práce.



Obrázek 19: Struktura zdrojových souborů pro firmware

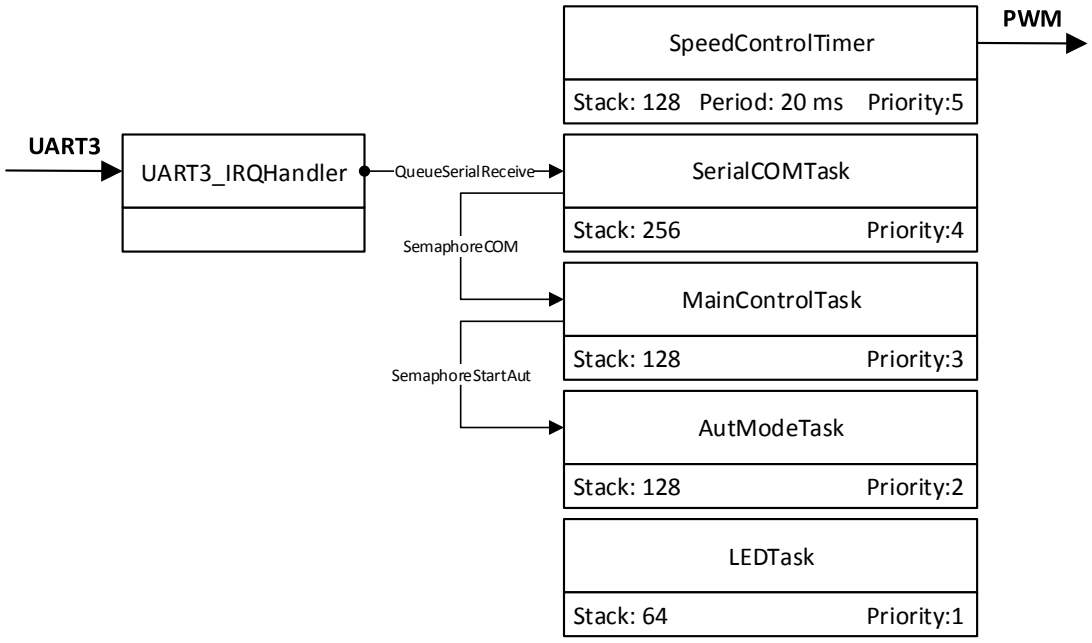
Použití operačního systému FreeRTOS nám umožňuje program jednoduše rozdělit na jednotlivé logické celky – úlohy. Těmto úlohám poté přiřadit priority, velikosti zásobníku, a vykonávat je pseudoparalelně. Firmware se tedy sestává ze 4 úloh, jednoho periodického časovače a z jedné obslužné rutiny přerušení. Schematické zobrazení těchto úloh a komunikaci mezi nimi nám ukazuje *Obrázek 20*.

Obslužná rutina přerušení je spouštěna v okamžiku, kdy sériová linka přijme byte. Jelikož se jedná o čistě hardwarové přerušení, FreeRTOS nemůže vykonání této rutiny nijak pozastavit - rutina má tedy ve firmware nejvyšší prioritu. Činnost této rutiny spočívá pouze v příjmu bytu a zavolaní FreeRTOS API funkce `xQueueSendFromISR`, která tento byte přidá do fronty příchozích bytů.

Časovač `SpeedControlTimer` je spouštěn periodicky každých 20 ms. Ostatní úlohy běží v nekonečné smyčce, která je však ve všech úlohách nějakým způsobem pozastavena (čekaní na data ve frontě, čekaní na semafor, zpoždění úlohy). Úlohám jsou

tedy přiděleny různé priority, a i přesto se mohou stát běžícími. Popis činnosti jednotlivých úloh bude popsán v následujících podkapitolách.

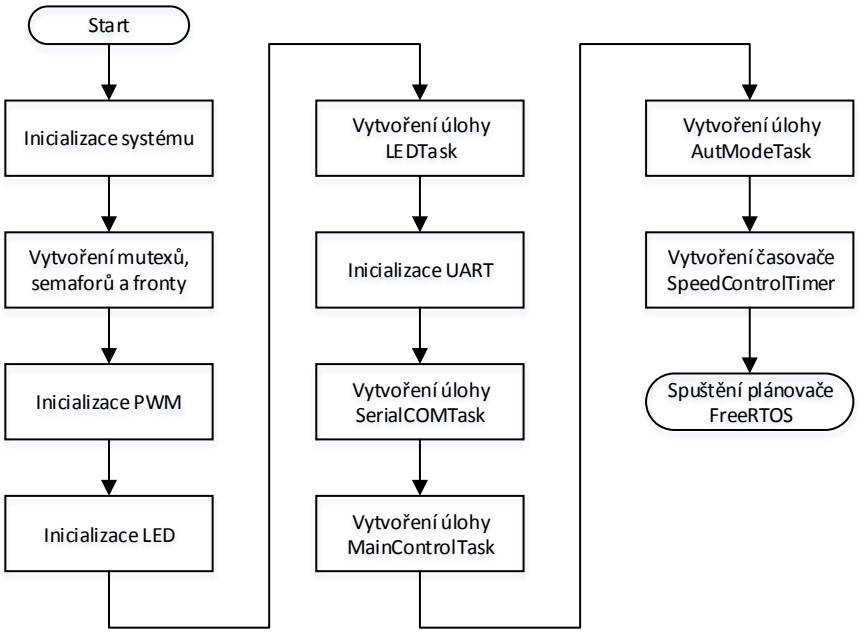
Dále uvedené vývojové diagramy jsou pro přehlednější znázornění principu zjednodušené. Neodpovídají tedy přesnému chování aplikace ve všech detailech.



Obrázek 20: Jednotlivé úlohy a komunikace mezi nimi

6.2 Start systému

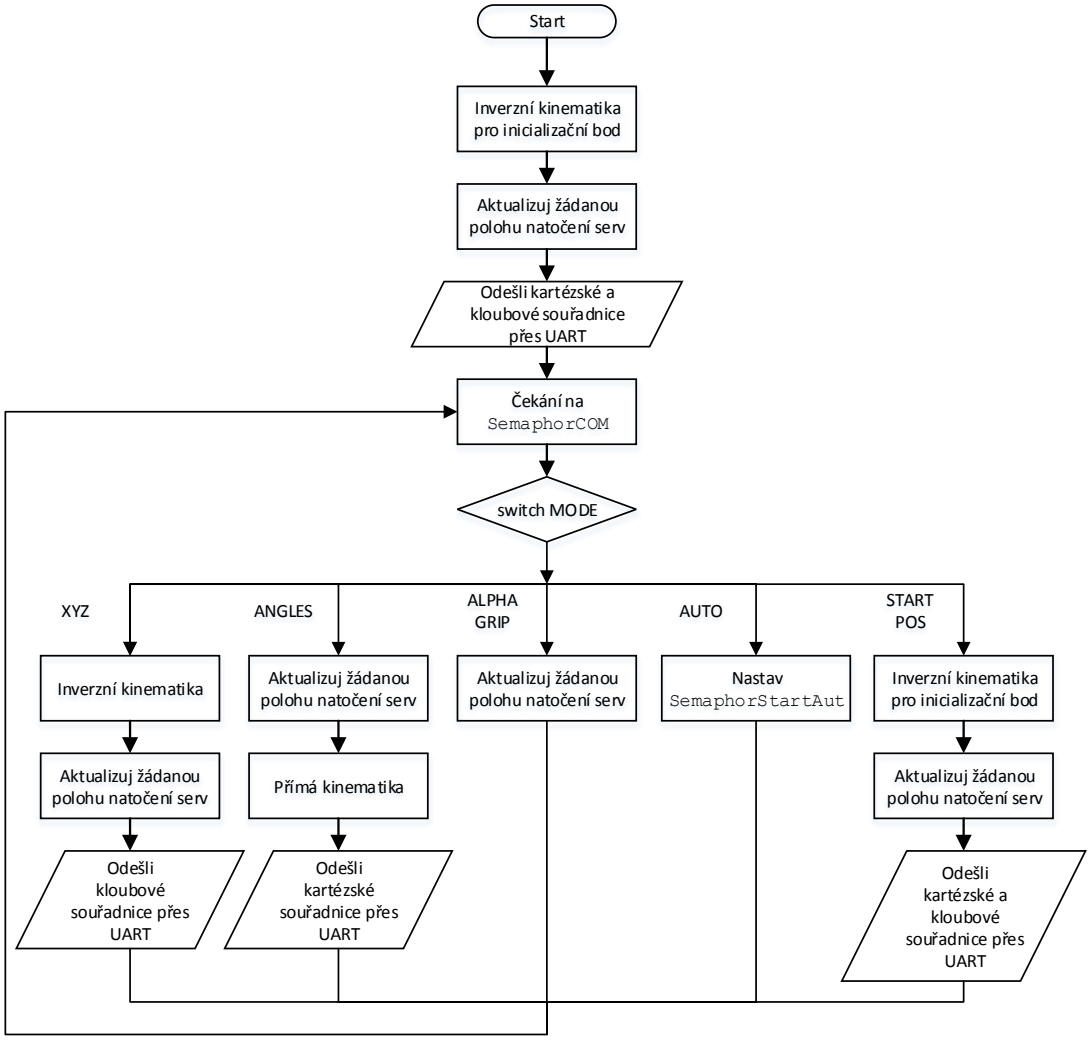
Po restartu mikrokontroléru je provedena kompletní inicializace, která probíhá podle vývojového diagramu na *Obrázek 21*.



Obrázek 21: Vývojový digram pro start systému

6.3 Hlavní řídicí úloha

Při prvním zavolání této úlohy plánovačem se provede výpočet inverzní kinematické úlohy pro inicializační bod zadaný v kartézských souřadnicích a aktualizují se proměnné, ve kterých je uložena žádaná pozice natočení jednotlivých servomotorů na manipulátoru. Kloubové i kartézské souřadnice se odešlou přes UART, aby GUI na PC mělo aktuální informace o těchto žádaných souřadnicích. Úloha poté vstupuje do nekonečné smyčky, kde je pozastavována čekáním na `SemaphoreCOM`. Tento semafor může být aktivován úlohou `SerialCOMTask` po přijmutí a zpracování paketu. Následně je vykonána větev programu, která přísluší aktuálnímu módu činnosti (viz *Obrázek 22*).



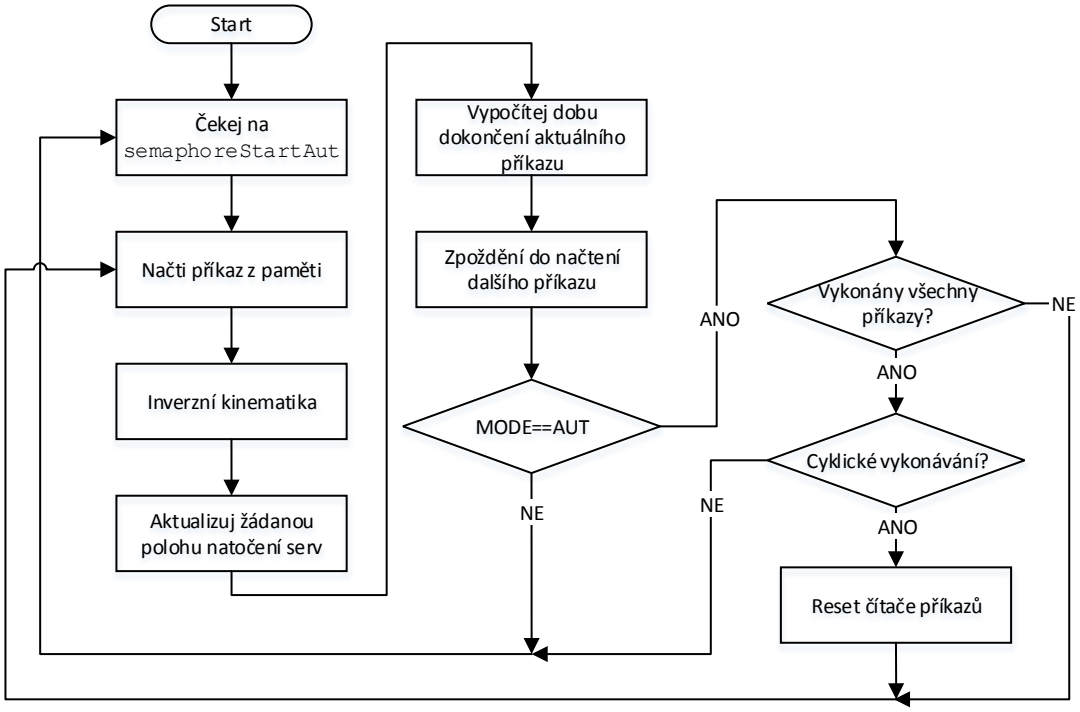
Obrázek 22: Vývojový diagram pro hlavní řídicí úlohu

6.4 Úloha automatického režimu

Ihned po startu vstupuje tato úloha do nekonečné smyčky, kde čeká na `SemaphoreStartAut`, který je aktivován z hlavní řídicí úlohy. Poté je načten první

příkaz, který obsahuje souřadnice koncového bodu manipulátoru rychlost, se kterou se má pohyb provést, a dobu setrvání v pozici. Dále je vypočítána inverzní kinematická úloha a aktualizovány proměnné pro požadované natočení servomotorů. Následuje vypočet času, který bude manipulátor potřebovat na pohyb z aktuální pozice do nové pozice při dané rychlosti. K tomuto času se ještě přičte doba setrvání v pozici, která je obsažena v příkazu, a součet těchto dvou časů je předán jako parametr funkci `vTaskDelay`, která vytvoří zpoždění této úlohy. Po vypršení doby zpoždění je testováno, zdali se stále pracuje v automatickém režimu, jestli je k dispozici další příkaz a nebo zda je zapnutá funkce cyklického vykonávání a sada předchozích příkazů se má vykonat znovu.

Grafické znázornění práce úlohy opět poskytne vývojový diagram *Obrázek 23*.



Obrázek 23: Vývojový diagram pro úlohu automatického režimu

6.5 Sériová komunikace

Důležitou součástí firmware je komunikace s PC aplikací po sériové lince. Mikrokontrolér LPC1756 je vybaven periferií UART s 16bytovým přijímacím a vysílacím FIFO zásobníkem.

Komunikace je nakonfigurována na tyto parametry:

Rychlost	57600 baud
datové bity	8
parita	žádná
stop bit	1

HW řízení toku ne

O fyzickou vrstvu přenosu je tedy postaráno UART periferií. Jak již bylo řečeno v úvodu této kapitoly, samotný příjem jednotlivých bytů je řešen v obslužné rutině přerušení `UART3_IRQHandler`, kde jsou byty předávány FreeRTOS frontě `xSerialReceive`. Zpracování této fronty je již v kompetenci úlohy `SerialCOMTask`.

Při komunikaci je využíváno protokolu SLIP, který je definován v [13]. Jedná se o jednoduchý protokol který byl určen pro přenos IP paketů po sériové lince. Protokol definuje tyto 4 znaky:

Tabulka 2: Znaky definované protokolem SLIP

hex hodnota	**dec hodnota**	**zkratka**	**význam**
0xC0	192	END	Konec paketu
0xDB	219	ESC	Přerušení paketu
0xDC	220	ESC_END	Nahrazení znaku END
0xDD	221	ESC_ESC	Nahrazení znaku ESC

Paket má tento jednoduchý formát:

END	data (n bytů)	END

Jestliže se v posílaných datech vyskytuje byte END, musí být nahrazen posloupností ESC, ESC_END. Podobně je tomu, když se v datech vyskytuje byte ESC. Ten je nahrazován posloupností ESC, ESC_ESC.

Při posílání různých typů zpráv je nutné je od sebe nějakým způsobem rozlišit. Prvním bytem v datech je tedy posílán identifikátor, který určí, o jaký typ zprávy se jedná. K ověření, jestli při přenosu nedošlo k chybě, je na konec přidán kontrolní součet. Používá se metody CRC-16 (16bitový cyklický redundantní součet). Kompletní paket má tedy tento výsledný formát:

END	ID	data(n bytů)	CRC1	CRC2	END

Firmware komunikuje s PC aplikací pomocí těchto zpráv:

Tabulka 3: Identifikátory zpráv

ID	zkratka	délka dat[B]	Význam	Směr komunikace
1	X	2	kartézská souřadnice x	PC→µC
2	Y	2	kartézská souřadnice y	PC→µC
3	Z	2	kartézská souřadnice z	PC→µC
4	BETA	2	uhel zapěstí svírající s podložkou	PC→µC
5	ALPHA	2	rotace zápěstí v ose x	PC→µC
6	GRIP	2	sevření uchopovacích kleští	PC→µC
7	Q0	2	kloubová souřadnice q0	PC→µC
8	Q1	2	kloubová souřadnice q1	PC→µC
9	Q2	2	kloubová souřadnice q2	PC→µC
10	Q3	2	kloubová souřadnice q3	PC→µC
11	STARTPOS	0	manipulátor najede do výchozí pozice	PC→µC
12	INSTR	16 x n	sada instrukcí pro automatický režim	PC→µC
13	SPEED	4	rychlost pohybu manipulátoru	PC→µC
21	XYZABG	12	pozice v kartézských souřadnicích	µC→PC
22	ANGLES	12	kloubové souřadnice	µC→PC
30	FREEHEAP	4	volné místo na haldě FreeRTOS	µC→PC
31	TASKLIST	100	výpis úloh FreeRTOS	µC→PC
32	QFREEHEAP	0	žádost o zaslání volného místa na haldě	PC→µC
33	QTASKLIST	0	žádost o výpis úloh	PC→µC
40	LOGGER	1-50	posílaní textových zpráv o chybách	µC→PC
45	RESET	0	provede kompletní reset mikrokontroléru	PC→µC
46	PAUSE	0	pozastaví automatické vykonávání příkazů	PC→µC
47	CONTINUE	0	pokračuje v aut. vykonávání příkazů	PC→µC
51	CYCLIC	0	cyklické vykonávání příkazů v aut. režimu	PC→µC
52	ONCE	0	jednorázové vykonání příkazů v aut. režimu	PC→µC
53	SENDAGAIN	0	žádost o znovuzaslání paketu (neshoda CRC)	oba směry
54	PWMON	0	zapne servomotory a najede do start pozice	PC→µC
55	PWMOFF	0	najede do start pozice a vypne servomotory	PC→µC

*) n značí počet příchozích příkazů. Počet je omezen na 100 příkazů z důvodu nedostatku volné paměti.

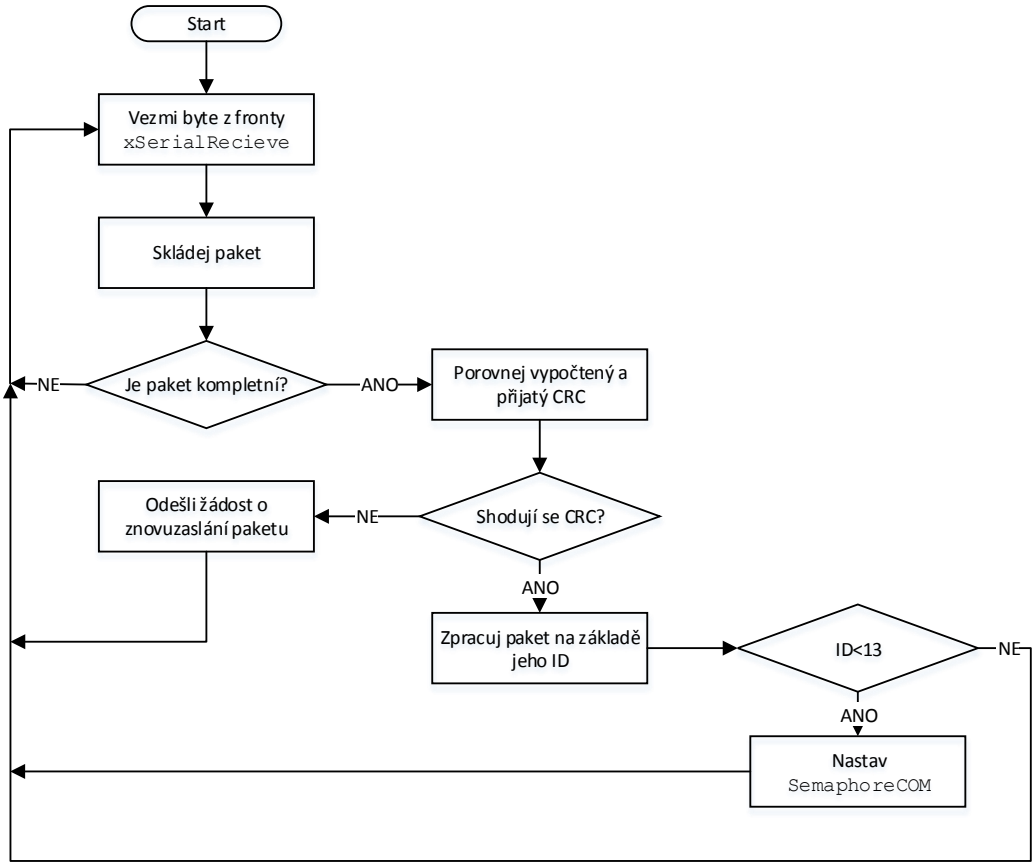
Zprávy lze dělit do jednotlivých skupin. Ty barevně vyznačené mají za následek přepínání režimu v hlavní řídicí úloze *Obrázek 22*

Režim XYZ
Režim ALPHAGRIP
Režim ANGLES
Režim STARTPOS
Režim AUTO

Ostatní zprávy nemají vliv na přepnutí režimu
U zpráv, kde se neposílají žádná data (délka dat = 0) je podstatný pouze identifikátor, který slouží jako příkaz pro firmware. U těchto typů zpráv není využito kontrolního součtu, ten se počítá pouze pro obsah zprávy (pro data).

6.5.1 Úloha sériové komunikace

Po startu úlohy se ihned vstupuje do nekonečné smyčky. Prvním úkolem je vzít byte z fronty příchozích bytů `xSerialRecieve`. Pokud je fronta prázdná, úloha na tomto místě stojí a čeká na její naplnění. Následuje funkce pro skládání paketů, kde se musí rozpoznat začátek, konec a především správně nahradit posloupnosti ESC, ESC_END a ESC, ESC_ESC. Jestliže je přijatý paket kompletní, zavolá se funkce pro výpočet kontrolního součtu a její výsledek se porovná s hodnotou kontrolního součtu obsaženou v paketu. V případě, že se kontrolní součty nerovnají, odešle se žádost o znovuzaslání paketu. V opačném případě se pokračuje zpracováním paketu na základě jeho identifikátoru. Jedná-li se o paket ze skupiny, která bezprostředně ovlivňuje pohyb manipulátoru (ID<13), je nastaven `SemaphoreCOM`, který spouští vykonávání hlavní řídicí úlohy.



Obrázek 24: Vývojový diagram pro úlohu sériové komunikace

6.6 Výpočty kinematických úloh v mikrokontroléru

Přímá a inverzní kinematická úloha byla pro daný typ manipulátoru analyticky řešena v kapitolách 5.1 a 5.2. Výsledkem jsou sady především goniometrických rovnic. Aby bylo možné tyto rovnice vypočítat v mikrokontroléru, je nutno nejprve implementovat funkce odmocnina, sinus, kosinus a arcus tangens.

Používaný kontrolér LPC1756 nemá integrovanou jednotku pro výpočty v plovoucí řádové čárce (dále jen FPU), každá operace s reálným číselným typem (dále jen float) se tedy musí softwarově emulovat, což ovšem zabere procesoru mnohem více času (v průměru asi 15x více) než počítání s celými čísly (dále jen integer). Z tohoto důvodu je většina výpočtů prováděna s integer čísly, jen v několika málo případech, kde by převod na integer byl značně komplikovaný, je počítáno s float.

6.6.1 Implementace matematických funkcí

Ze standardní knihovny pro jádro Cortex-M3 (cmsis) jsou využity funkce pro výpočet odmocniny z 32bitových float čísel a funkce sinus pro 16bitový integer. Používání funkce odmocniny je velice jednoduché, protože vstupem i výstupem funkce je standardní float32.

U funkce sinus je situace o něco komplikovanější. Vstupem je argument v rozmezí 0..2π, který musí být rozšířen do intervalu 0..0x7FFF. Výstup funkce je v rozmezí -1..1 rozšířen do intervalu int16 (-32768..32767). Vnitřní výpočet funkce sinus využívá tzv. LookUp tabulku s 256 16bitovými hodnotami na intervalu 0..2π a proložení polynomem 3. stupně.

Při výpočtu funkce cosinus se využívá té skutečnosti, že platí tato jednoduchá rovnice:

$$\cos(x) = \sin(x + {}^{\pi}/_{2}) \tag{21}$$

Funkce arcus tangens není ve standardní knihovně v žádné formě obsažena, její implementace však byla částečně převzatá z [1]. Hlavička funkce je `uint16_t atan2Lookup(int x, int y)`. Vstupem jsou dva integer parametry udávající poměr dvou stran. Na začátku je na základě těchto dvou parametrů určeno, ve kterém oktantu (část rozděleného kruhu třemi rovinami) se bude výsledný úhel nacházet. Poté je pomocí LookUp tabulky (128 16bitových hodnot na intervalu 0..1) a lineárním proložením nalezen úhel, který je poté ještě posunut do správného oktantu. Funkce vrací úhel $Atan\left(\frac{y}{x}\right)$ jako integer hodnotu z intervalu 0..2π rozšířenou do intervalu 0..0x7FFF.

6.6.2 Časová náročnost výpočtů

Pro představu o době trvání matematických výpočtů bylo provedeno orientační měření pomocí *SysTick* časovače. Jednotlivá měření lze provádět s přesností systémových hodin, tedy s 0,01 μs. Čas vykonání jednotlivých operací a funkcí je však závislý na

operandech, respektive na parametrech. Účelem tedy nebylo provést přesná měření pro všechny možné kombinace těchto parametrů, ale měření, které bude mít pouze informativní charakter. Časy uvedené v následující tabulce jsou pouze odhadnutými průměry při opakovaní jednotlivých měření s náhodnými vstupními parametry funkcí. Rozptyl hodnot byl také pouze odhadnut a pohyboval se okolo 10 % z uvedené průměrné hodnoty. Připomeneme, že jádro procesoru pracuje na frekvenci 100 MHz.

Tabulka 4: Časová náročnost matematických operací

Operace	**Orientační čas vykonání [µs]**
násobení int32	0,05
dělení int32	0,07
násobení float32	0,5
dělení float32	1,9
sinus int16	2,9
cosinus int16	3,0
arkus tangens int32	2,9
odmocnina float32	5,9
přepočet z úhlů na PWM předvolby	28
přímá úloha kinematiky	36
inverzní úloha kinematiky	105

6.7 Časovač řízení rychlosti pohybu a generování PWM

6.7.1 Generování PWM

Mikrokontrolér LPC1756 je vybaven hardwarovou periferií pro generování PWM signálu až pro 6 kanálů. Této skutečnosti je s výhodou využito pro generování řídicích signálů pro servomotory. Toto PWM rozhraní je založeno na klasickém časovači. Generování probíhá tím způsobem, že na začátku nového cyklu je vygenerována nástupná hrana na všech šesti kanálech. Poté je časovačem inkrementován čítač (s frekvencí 25 MHz) a aktuální hodnota tohoto čítače je porovnávána s hodnotou registrů pro jednotlivé kanály (MatchRegister1-6). V případě shody těchto dvou hodnot se generuje sestupná hrana na příslušném kanále. Periodu opakování tohoto cyklu (frekvenci PWM) udává hodnota sedmého registru (MatchRegister0). Po dovršení čítače na hodnotu tohoto registru se čítač vynuluje a začíná nový cyklus.

Frekvence PWM signálu je 50 Hz, čítač pracuje na frekvenci 25 MHz, z tohoto vypočítáme hodnotu registru, který udává periodu opakovaní. Předvolba tedy vychází:

$$MatchRegister0 = \frac{25000000}{50} = 500000 \qquad (22)$$

6.7.2 Kalibrace servomotorů

Jak již bylo uvedeno v teoretickém úvodu, každý servomotor je z důvodu přesnosti a zamezení možnému poškození nutno předem nakalibrovat. Následující tabulka udává tyto naměřené hodnoty. Pro měření byl využit klasický úhloměr. Dosažená přesnost v těchto třech měřených pozicích je asi ±2°. Sloupec *úhel* udává reálnou pozici tohoto úhlu, jeho počátek a orientaci lze vyčíst z *Obrázek 16* a *Obrázek 17*. Sloupec *čas* udává šířku řídicího PWM impulsu a Sloupec *PWM* konkrétní předvolbu pro registry (MatchRegister1-6) v mikrokontroléru.

Tabulka 5: Závislost úhlu natočení na šířce impulsu pro jednotlivé klouby.

	Levé maximum			Neutrál			Pravé maximum		
kloub	úhel [°]	čas [µs]	PWM[-]	úhel [°]	čas [µs]	PWM[-]	úhel [°]	čas [µs]	PWM[-]
q0	+90	560	14000	0	1460	36500	-90	2360	59000
q1	0	660	16500	-90	1480	37000	-180	2300	57500
q2	0	660	16500	+90	1500	37500	+180	2340	58500
q3	+90	680	17000	0	1560	39000	-90	2440	61000
q4	-90	620	15500	0	1480	37000	+90	2340	58500
q5	37 mm	720	18000	22 mm	1580	39500	7 mm	2440	61000

6.7.3 Časovač pro řízení rychlosti pohybu servomotorů

Skoková změna šířky generovaného PWM impulsu zapříčiní pohyb servomotoru maximální možnou rychlostí, jakou uvádí Tabulka 1. Měnit rychlost je však možné softwarově, a to postupným přičítáním malého přírůstku k hodnotě v registru, který udává šířku PWM impulsu, tedy úhel natočení servomotoru. Pro tento účel je využita úloha FreeRTOS časovače, který je nastaven na periodu spouštění 20 ms (doba shodná s frekvencí generování PWM). Šířka impulsu je tedy každých 20 ms inkrementována/dekrementována a změna již není skoková, ale schodovitá. Vlivem časových konstant servomotoru a mechanické stavby manipulátoru je však tento schodovitý průběh vyhlazen a výsledný pohyb manipulátoru je plynulý.

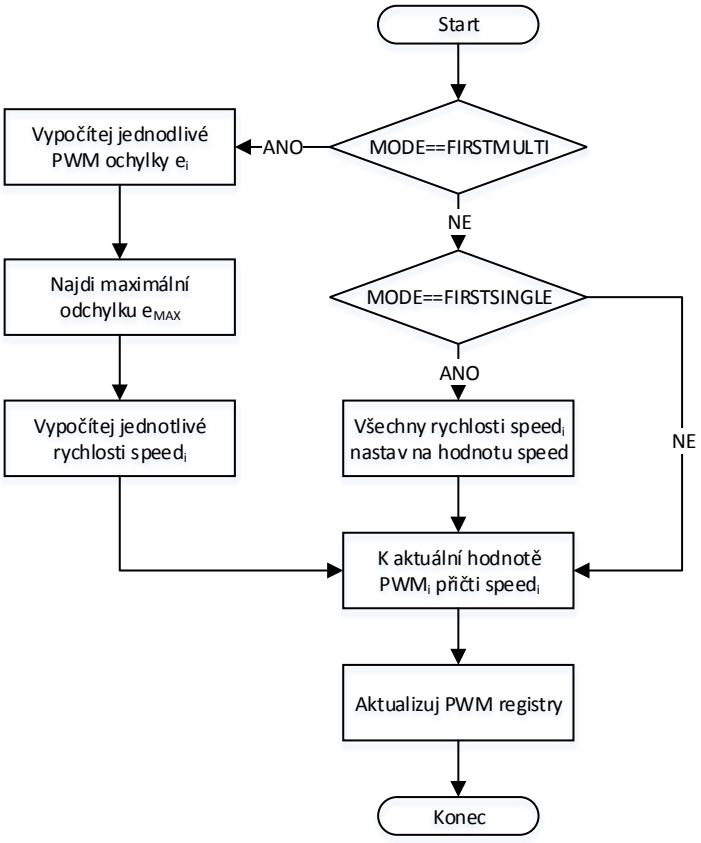
Jestliže se má koncový bod manipulátoru přemístit z jednoho bodu do druhého, bude k tomu ve většině případů servomotory nutné pohnout více najednou. Žádoucí ovšem bude, aby pohyb ve všech kloubech začal i skončil ve stejný okamžik. Úhly pootočení jednotlivých kloubů nebudou však téměř nikdy shodné a tím pádem by kloub s menším žádaným úhlem pootočení svůj pohyb při stejné rychlosti dokončil dříve, nežli kloub, kterým je vyžadováno provést větší pootočení. Z tohoto důvodu se musejí jednotlivé přírůstky pro PWM registry přepočítat podle následující rovnice.

$$speed_i = \frac{e_i \cdot speed}{e_{MAX}} \; [-] \tag{23}$$

kde $speed_i$ jsou jednotlivé přírůstky,
e_i jsou jednotlivé odchylky vypočítané jako $PWM_{aktual} - PWM_{\check{z}\acute{a}dan\acute{a}}$,
e_{MAX} je maximální odchylka z jednotlivých e_i,
$speed$ je požadovaná rychlost.

Požadovanou rychlostí se tedy bude pohybovat pouze kloub, který se musí pootočit o největší úhel. Ostatní pohyby kloubů budou zpomalené tak, aby dosáhly žádané pozice ve stejný okamžik.

Tento výpočet rychlostí se bude provádět pouze v tom případě, kdy je úloha časovače vykonána poprvé po změně žádaných hodnot PWM. V programu a vývojovém diagramu značeno makrem FIRSTMULTI. V případě, kdy je požadován pohyb pouze jedním servem, musejí se jednotlivé rychlosti resetovat nastavením na hodnotu požadované rychlosti $speed$. Tento mód je značen makrem FIRSTSINGLE. V ostatních případech se v časovači pouze navýší PWM_{aktual} o $speed_i$ a výsledné hodnoty se zapíší do PWM registrů. Viz vývojový diagram na *Obrázek 25*



Obrázek 25: Vývojový diagram časovače pro řízení rychlosti

6.8 Indikační LED dioda

FreeRTOS úloha, která ovládá LED diodu běží v jádře s nejmenší prioritou ze všech úloh. Úkolem LED diody je poskytovat pouze základní informaci o stavu firmware. Může nabývat tří různých stavů uvedených v následující tabulce.

Tabulka 6: Význam LED diody

Stav	**Význam**
Nesvítí	Negeneruje PWM signál, manipulátor je vypnutý
Svítí	Generuje PWM signál, manipulátor je v žádané pozici
Bliká	Manipulátoru byla zadána nedosažitelná poloha, nachází se tedy v poslední dosažitelné pozici.

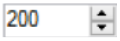
7 PC APLIKACE

Pro komunikaci s firmware v mikrokontroléru byla vytvořena PC aplikace, která slouží jako GUI. Pro tvorbu této aplikace byl zvolen programovací jazyk C# a jako vývojové prostředí je používán produkt firmy Microsoft – Visual Studio 2012.

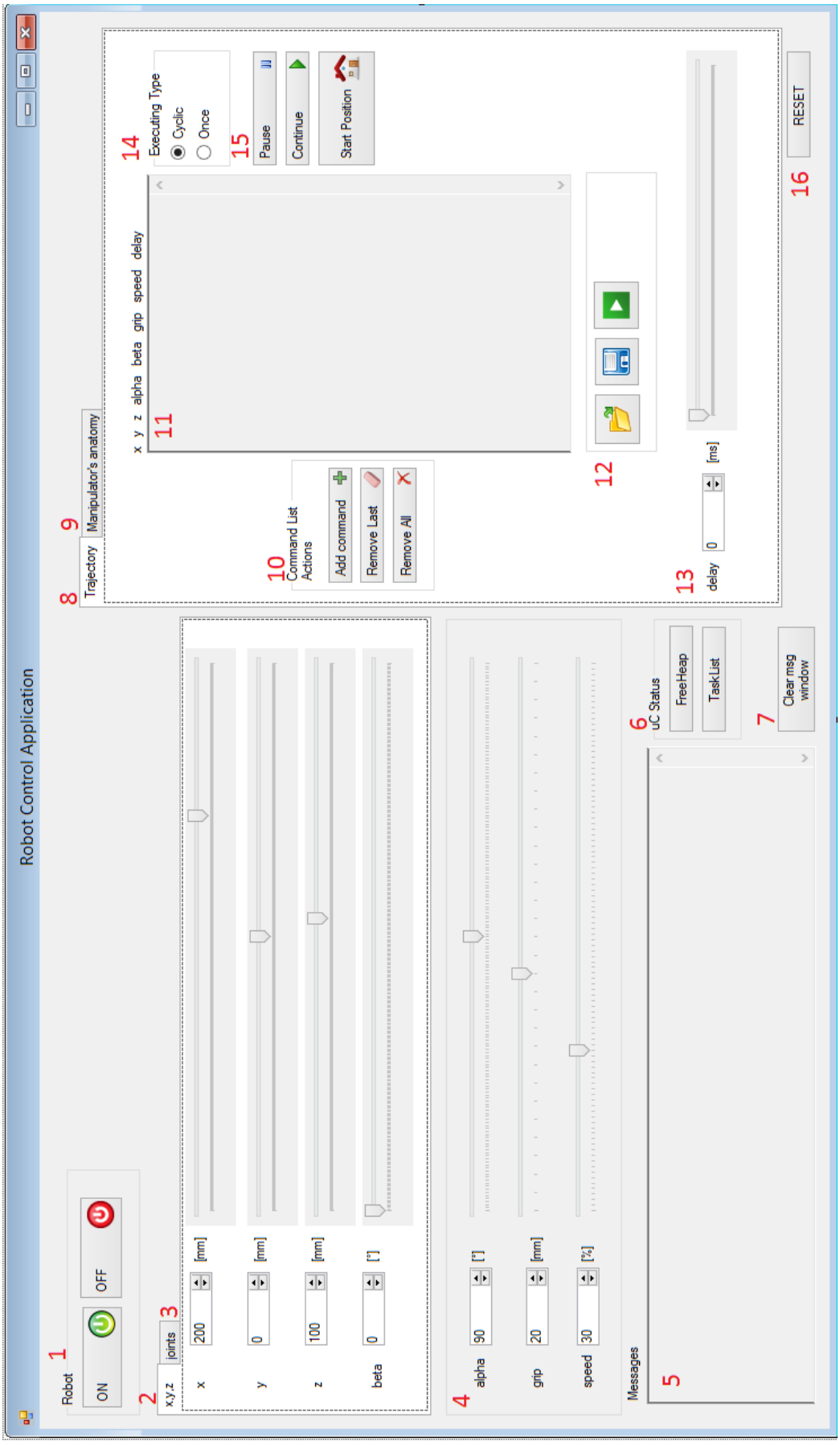
Vytvořený kód je rozdělen do dvou tříd – `Form1.cs` a `SerialCOM.cs`. Většina kódu náleží třídě hlavního okna, tedy `Form1.cs`. Úlohou třídy `SerialCOM.cs` je komunikace po sériové lince, ke které patří i kódování a dekódování paketů a výpočet kontrolního součtu.

GUI aplikace umožňuje především online ovládání manipulátoru jak v kartézských, tak v kloubových souřadnicích a vytváření trajektorie postupným ukládáním jednotlivých bodů. Podrobnější možnosti této aplikace jsou uvedeny v následující podkapitole, která slouží jako uživatelský manuál pro ovládání GUI.

7.1 Návod pro ovládání GUI rozhraní

Náhled obrazovky uživatelského rozhraní je i s očíslováním jednotlivých skupin na *Obrázek 26*. V celém GUI je každý *NumericUpDown* 

spárovaný se sousedním *TrackBar*  , takže změna v kterémkoliv z těchto prvků vyvolá i aktualizaci prvku párového. Změny na prvcích ze skupin 2,3 a 4 jsou skenovány po 100 ms intervalech, aby se zabránilo přehlcení sériové linky, které by mohlo vzniknout rychlými pohyby *TrackBaru*.



Obrázek 26: Obrazovka GUI rozhraní

Popis jednotlivých skupin:

1) Zapínání a vypínaní manipulátoru - Po stisknutí tlačítka 

proběhne pokus o inicializaci sériové komunikace. Jestliže připojení proběhne v pořádku, manipulátor se zapne a najede do výchozí pozice. V opačném případě vyskočí chybové hlášení

Obrázek 27: Chybové hlášení při otevírání sériového portu

Tlačítkem 

se manipulátor vypne tak, že najede do výchozí polohy. Poté mikrokontrolér přestane generovat PWM signály a jednotlivé servomotory se uvolní.

2) Online ovládání koncového bodu manipulátoru v kartézských souřadnicích – Změna kterékoliv ze souřadnic je odeslána do mikrokontroléru, kde je vypočítána inverzní úloha kinematiky, proveden pohyb manipulátorem a kloubové souřadnice jsou odeslány zpět GUI. Jestliže je nastaven bod, který je nedosažitelný, obrazovka GUI zčervená a v okně zpráv (5) se objeví, z jakého důvodu je bod nedosažitelný. Manipulátor přitom setrvává v poslední dosažitelné pozici.

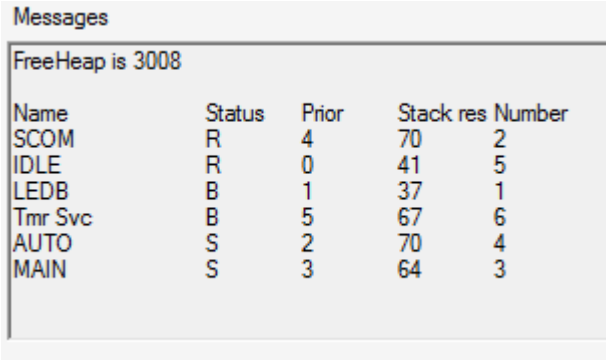
3) Online ovládání koncového bodu manipulátoru v kloubových souřadnicích – Zde je postup přesně opačný než v předchozím případě, tzn. kloubové souřadnice jsou posílány do mikrokontroléru, kde je proveden pohyb manipulátorem, vypočítána přímá úloha kinematiky a kartézské souřadnice jsou odeslány zpět GUI.

4) Natočení a sevření uchopovacích kleští (*alpha*, *grip*) a nastavení rychlosti pohybu servomotorů (*speed*) – *alpha* a *grip* není zahrnuto do kinematických úloh, protože jejich změna neovlivní žádné jiné souřadnice. Při změně v těchto prvcích je jejich nová hodnota odeslána do mikrokontroléru a proveden pohyb manipulátorem, respektive aktualizována rychlost.

5) Okno pro vypisování příchozích zpráv od mikrokonroléru – Zprávy jsou typu:

   - důvod, proč je bod nedosažitelný
   - vyskytnuvší se chyby ve firmware
   - výpis úloh a informace o volné paměti na haldě FreeRTOS

6) Žádost o výpis úloh a informaci o volné paměti na haldě FreeRTOS – Po stisknutí obou těchto tlačítek se zobrazí následující výpis.



Obrázek 28: Výpis úloh a volné haldy FreeRTOS

7) Vymazání všech výpisů v okně *Messages*.

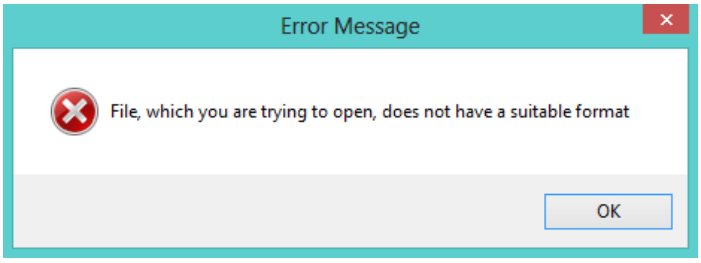
8) Záložka pro plánování trajektorie.

9) Schéma manipulátoru – V této záložce se nachází náhledy na anatomii manipulátoru stejné jako *Obrázek 3* a *Obrázek 16* .

10) Práce se seznamem příkazů – Po stisknutí tlačítka Add command se vezmou hodnoty ovládacích prvků ve skupinách 2, 4 a 13, oddělí se čárkou a vzniká příkaz, který je uložen do seznamu . Tlačítko je stisknutelné pouze v případě, že nastavená pozice manipulátoru je dosažitelná. Tlačítko Remove Last odstraní posledně přidaný příkaz ze seznamu a tlačítko Remove All vymaže seznam kompletně.

11) Seznam příkazů – Toto okno slouží pouze jako náhled na seznam uložených příkazů (nelze editovat). Příkazy je možné přidávat pouze tlačítkem *Add command*. Jeden řádek znamená jeden příkaz, který má striktně daný formát *x,y,z,alpha,beta,grip,speed,delay*.

12) Otevření existujícího, uložení a spuštění vykonávání seznamu příkazů – Stiskem se otevře klasické dialogové okno pro otevírání souborů. Po otevření se zkontroluje jeho obsah a pokud nesplňuje formát uvedený v předchozím bodě, vyskočí okno s chybovým hlášením.



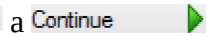
Obrázek 29: Chybové hlášení při otevírání seznamu instrukcí

Jestliže kontrola proběhne v pořádku, seznam příkazů se zobrazí v okně (11). Stiskem  se opět otevírá klasické dialogové okno, tentokrát však pro uložení seznamu příkazů do textového souboru. Tlačítkem  se příkazy odešlou do mikrokontroléru, kde se ihned začne s jejich vykonáváním.

13) Nastavení doby setrvání v poloze – Jedná se o poslední parametr, kterým je tvořen příkaz.

14) Přepínač cyklického a jednorázového vykonávání programu – Je-li změněn, je jeho nový stav odeslán do mikrokontroléru.

15) Pozastavení, pokračování a přerušení vykonávání sady příkazů s najetím do výchozí pozice – Tlačítka 

a 

jsou použitelná pouze při vykonávání sady příkazů (automatický režim ve firmware) a slouží pro pozastavování a znovu rozběhnutí programu. Stisk 

má za následek najetí manipulátoru do výchozí pozice. Jestliže je vykonáván automatický režim, je jeho běh okamžitě přerušen.

16) Reset – Provede kompletní restart mikrokontroléru i GUI aplikace.

8 VYHODNOCENÍ FUNKČNOSTI CELÉHO PROJEKTU

Výsledkem tohoto projektu je funkční systém pro řízení 6-ti osého manipulátoru. Byl vytvořen firmware pro řídicí desku s mikrokontrolérem LPC1756, který je schopen řešit přímou a inverzní kinematickou úlohu v reálném čase a komunikovat s GUI aplikací po sériové lince. GUI aplikace běžící na PC nabízí rozhraní pro online řízení manipulátoru a vytváření trajektorie pro automatický režim.

Vyhodnocení funkčnosti bude ukázáno na úloze, kde bude mít manipulátor za úkol postupně sesbírat kostky lega, volně rozmístěné v jeho manipulačním prostoru, do krabice. Nejprve bude v jednotlivých krocích uveden postup, jak lze tuto úlohu vytvořit a spustit:

- Základním předpokladem je správné propojení desky řídicí jednotky s PC pomocí kabelu s převodníkem TTL/USB a přivedené napájecí napětí 5 V do řídicí desky.
- Na PC spustíme GUI aplikaci a tlačítkem *ON* zapneme manipulátor, který tak najede do své výchozí polohy.
- Manipulátor je nyní v online režimu. Pohybem posuvníků nebo změnou číselné hodnoty na jednotlivých souřadnicích zapříčiníme pohyb manipulátoru požadovanou rychlostí, která je po startu nastavena na 30 %. Je-li zadána pozice, která je pro manipulátor nedosažitelná, obrazovka GUI zčervená, na řídicí desce se rozbliká dioda a v okně příchozích zpráv se zobrazí důvod, proč je bod nedosažitelný. Manipulátor přitom zůstává v poslední dosažitelné poloze.
- Můžeme začít s postupným vytvářením trajektorie. Najíždět manipulátorem do bodů lze jak v kartézských, tak kloubových souřadnicích. Oba dva způsoby je možné kombinovat, protože hodnoty jsou z jedněch souřadnic do druhých průběžně přepočítávány a v okně GUI aktualizovány.
- Některým z těchto způsobů tedy dostaneme uchopovací kleště nad kostku lega, kterou budeme chtít uchopit, a kleště dostatečně rozevřeme. Pomocí tlačítka *Add Command* tento bod uložíme do seznamu příkazů a ten v něm bude zobrazen.
- S manipulátorem najedeme v ose *z* níže, tak aby kostka lega byla mezi uchopovacími kleštěmi, které poté sevřeme a příkaz opět uložíme.
- Tímto způsobem pokračujeme, dokud nevytvoříme program pro přesun všech kostek do krabice.
- Nakonec lze trajektorii uložit do textového souboru a stiskem tlačítka  odeslat seznam příkazů do mikrokontroléru, který hned po jeho přijetí začne s vykonáváním v automatickém režimu.

Kompletní seznam příkazů vytvořený pro úlohu sesbírání kostek lega je uveden na konci této práce v příloze C a video, na kterém je vykonání této úlohy zaznamenáno, je součástí přiloženého DVD.

Trajektorii lze vytvářet i přímým zapisováním příkazů do textového souboru mimo GUI. Tento postup však není doporučován, protože při načítání tohoto souboru do GUI je kontrolován pouze formát instrukcí, nikoliv konkrétní hodnoty. Mohlo by se tedy stát, že takto vytvořený seznam příkazů bude obsahovat nedosažitelné body. Ve výsledné trajektorii by tyto nedosažitelné body scházely a pohyb manipulátoru by byl jiný, než jaký byl zamýšlen.

Druhý důvod, proč tento způsob vytváření trajektorie offline není vhodný, je používaný manipulátor. Ten je velkou slabinou celého systému. Jedná se totiž spíše o výukovou pomůcku, nežli o zařízení, kterým by se dalo dosahovat uspokojivé přesnosti a opakovatelnosti. Servomotory v kloubech totiž nemají dostatečnou sílu na to, aby koncový bod manipulátoru udržely v požadované výšce. Vlivem gravitační síly se požadovaná poloha v ose z značným způsobem liší od polohy skutečné. Tento rozdíl je navíc závislý na hmotnosti uchopeného tělesa. Především servomotor v kloubu q1, který je zatížen největším momentem, je poddimenzovaný. Při větších rozpětích ramene nezvládá unést ani vlastní hmotnost manipulátoru. Řídicí systém také postrádá jakoukoliv informaci o zpětné vazbě. Spoléhá se pouze na hodnoty PWM signálů, kterými manipulátor ovládá. Vytvářením trajektorie v online režimu přes GUI rozhraní, kdy můžeme zároveň vidět skutečnou polohu manipulátoru, nám tedy dává možnost tyto nedostatky manipulátoru alespoň částečně eliminovat.

Jelikož byl pro implementaci firmware zvolen operační systém reálného času FreeRTOS, stává se tak do značné míry modulární, a tím i otevřený pro případné úpravy nebo rozšíření. Omezujícím faktorem je však RAM paměť mikrokontroléru, kde zbývají necelé 4 KB volného místa z celkových 32 KB. Doplněn by mohl být například interpolační modul pro přesun mezi dvěma body v prostoru po přímkové trajektorii.

Téměř neomezený prostor pro úpravy však nabízí nadřazený systém ve formě GUI aplikace běžící na PC. Zde by se dalo pokračovat především ve vývoji modulu pro plánování trajektorie, který v současném stavu nabízí pouze možnost postupného přidávání jednotlivých bodů. Naskýtá se tedy příležitost pro vytvoření kompletního skriptovacího jazyka s několika příkazy, který by tak mohl umožnit pohodlnější a rychlejší tvorbu trajektorie.

9 ZÁVĚR

Úkolem prvních tří bodů zadání této práce bylo se seznámit s přímou a inverzní kinematickou úlohou, s HW řešením poskytnuté řídicí jednotky a s vlastnostmi jádra ARM Cortex-M3. Všemi těmito body zadání se zabývá teoretická část této práce.

Dalším bodem zadání bylo vytvořit firmware pro řídicí jednotku s cílem realizovat řízení manipulátoru v reálném čase. Jelikož bylo požadováno ovládání jak v kloubových, tak v kartézských souřadnicích, musely být pro použitý manipulátor vyřešeny úlohy přímé a inverzní kinematiky a výsledné rovnice poté implementovány do firmware. Bylo provedeno i orientační měření pro zjištění doby výpočtu jednotlivých matematických funkcí i pro celou přímou i inverzní kinematickou úlohu. Inverzní úloha kinematiky zabere procesoru zhruba 105 μs a následné přepočítání úhlu na PWM předvolby dalších 28 μs. Přímá úloha kinematiky je zvládnuta za zhruba 36 μs. Tyto časy platí pro maximální možnou frekvenci jádra 100 MHz. Z hlediska požadavků kladených na firmware byl pro jeho vytvoření zvolen operační systém reálného času FreeRTOS. To umožnilo rozdělit činnost firmware na 4 samostatné úlohy a jeden softwarový časovač, kterým je řízena rychlost pohybu manipulátoru. Pro samotné generování PWM signálu je využita hardwarová periferie mikrokontroléru.

Posledním bodem zadání bylo firmware doplnit o komunikační modul a řídicí jednotku propojit s nadřazeným systémem, který bude umožňovat vyšší úroveň řízení. Pro komunikaci tedy bylo zvoleno sériové rozhraní a protokol SLIP. Jednotlivé zprávy jsou posílány ve formě paketů. Každý paket je na začátku rozlišován identifikátorem a navíc doplněn o 16bitový kontrolní součet. Roli nadřazeného systému zastává PC s GUI aplikací, která je napsaná v jazyce C# a využívá platformu .NET. Skrze tuto aplikaci je možné provádět online řízení manipulátoru a vytvářet trajektorii pro automatický režim.

V předchozí kapitole byla předvedena funkčnost systému na jednoduché úloze sesbíraní kostek lega do krabice a zároveň poukázáno na velkou slabinu, kterou je bezesporu používaný manipulátor. Z hlediska dalšího vývoje tohoto projektu by bylo vhodné manipulátor opatřit snímači, kterými by bylo možné získávat informace o skutečné poloze koncového bodu manipulátoru. S touto zpětnou vazbou by poté pracoval firmware, který by mohl být doplněn o regulátor, čímž by se zajistilo dorovnávání koncového bodu manipulátoru.

Na úplný závěr lze říci, že všechny body zadání byly splněny a výsledkem je funkční systém pro řízení 6-ti osého manipulátoru, který je však stále otevřený pro další možná vylepšení.

LITERATURA

[1] COOCOX. CoLinkEx User Manual. 2012.

[2] *Coranac: Off on a tangent: a look at arctangent implementations*. [online]. [cit. 2014-02-06]. Dostupné z: http://www.coranac.com/documents/arctangent/

[3] *FreeRTOS* [online]. ©2004-2010 [cit. 2014-05-17]. Dostupné z: http://www.freertos.org/

[4] GÁBRLÍK, Petr. *Univerzální řídicí systém pro quadrocopter*. Brno, 2012. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Vedoucí práce Luděk Žalud.

[5] KARGER, Adolf. Základy robotiky a prostorové kinematiky. 1. vyd. Praha: Vydavatelství ČVUT, 2000. 265 s. ISBN 80-010-2183-1.

[6] KUČERA, Pavel. *MRTS*. 2012. Přednášky. VUT Brno.

[7] LI, Qing. *Real-time concepts for embedded systems*. San Francisco: CMP Books, 2003, xii, 294 s. ISBN 15-782-0124-1

[8] MICROSOFT. MSDN - Microsoft Developer Network [online]. 2014 [cit. 2014-05-12]. Dostupné z: http://msdn.microsoft.com

[9] NXP Semiconductors [online]. ©2006-2014 [cit. 2014-05-17]. Dostupné z: http://www.nxp.com/

[10] NXP. *UM10360: LPC176x/5x User manual*. 2013. Dostupné z: http://www.nxp.com/documents/user_manual/UM10360.pdf

[11] OTAVA, Lukáš. *Firmware pro robotické vozítko*. Brno, 2013. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Vedoucí práce Pavel Kučera.

[12] *PELIKAN DANIEL: Serva*. [online]. [cit. 2014-02-04]. Dostupné z: http://www.pelikandaniel.com/?sec=page&id=22

[13] ROMKEY, J. RFC 1055 - NONSTANDARD FOR TRANSMISSION OF IP DATAGRAMS OVER SERIAL LINES: SLIP. [online]. 1988 [cit. 2014-05-06]. Dostupné z: http://tools.ietf.org/html/rfc1055

[14] *Root.cz: Instrukční sada Thumb-2 u mikroprocesorů ARM* [online]. [cit. 2014-02-04]. Dostupné z: http://www.root.cz/clanky/instrukcni-sada-thumb-2-u-mikroprocesoru-arm/

[15] *Serva a jejich ovládání: Řízení serva - teorie* [online]. [cit. 2014-02-04]. Dostupné z: http://serva.webnode.cz/rizeni-serva-teorie/

[16] *Servocity* [online]. © 1999-2014 [cit. 2014-05-14]. Dostupné z: http://www.servocity.com/

[17] SKAŘUPA, Jiří. Roboty a manipulátory. Ostrava, 2012. Učební text. Vysoká škola báňská – Technická univerzita Ostrava.

[18] ŠOLC, František a Luděk ŽALUD. *Robotika*. Brno, 2006. Skripta. VUT Brno.

[19] *uCSimply: Cortex-M3 z pohledu programátora*. [online]. [cit. 2014-02-04]. Dostupné z: http://www.ucsimply.cz/cm3/popis-procesoru/

[20] YIU, Joseph. *The Definitive Guide to the ARM Cortex-M3*. Second Edition. Burlington, USA: Newnes, 2009.

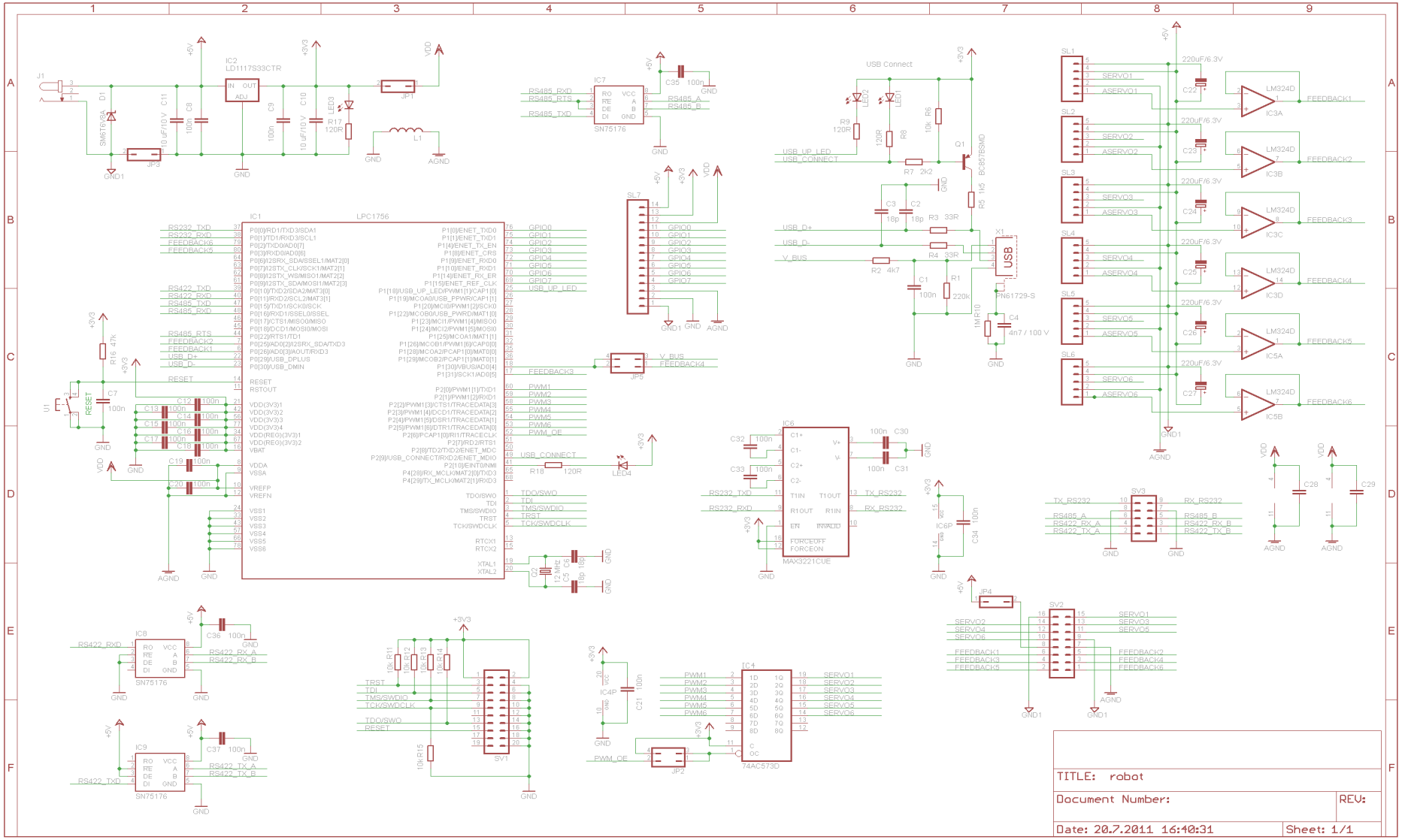
Seznam obrázků

Seznam tabulek

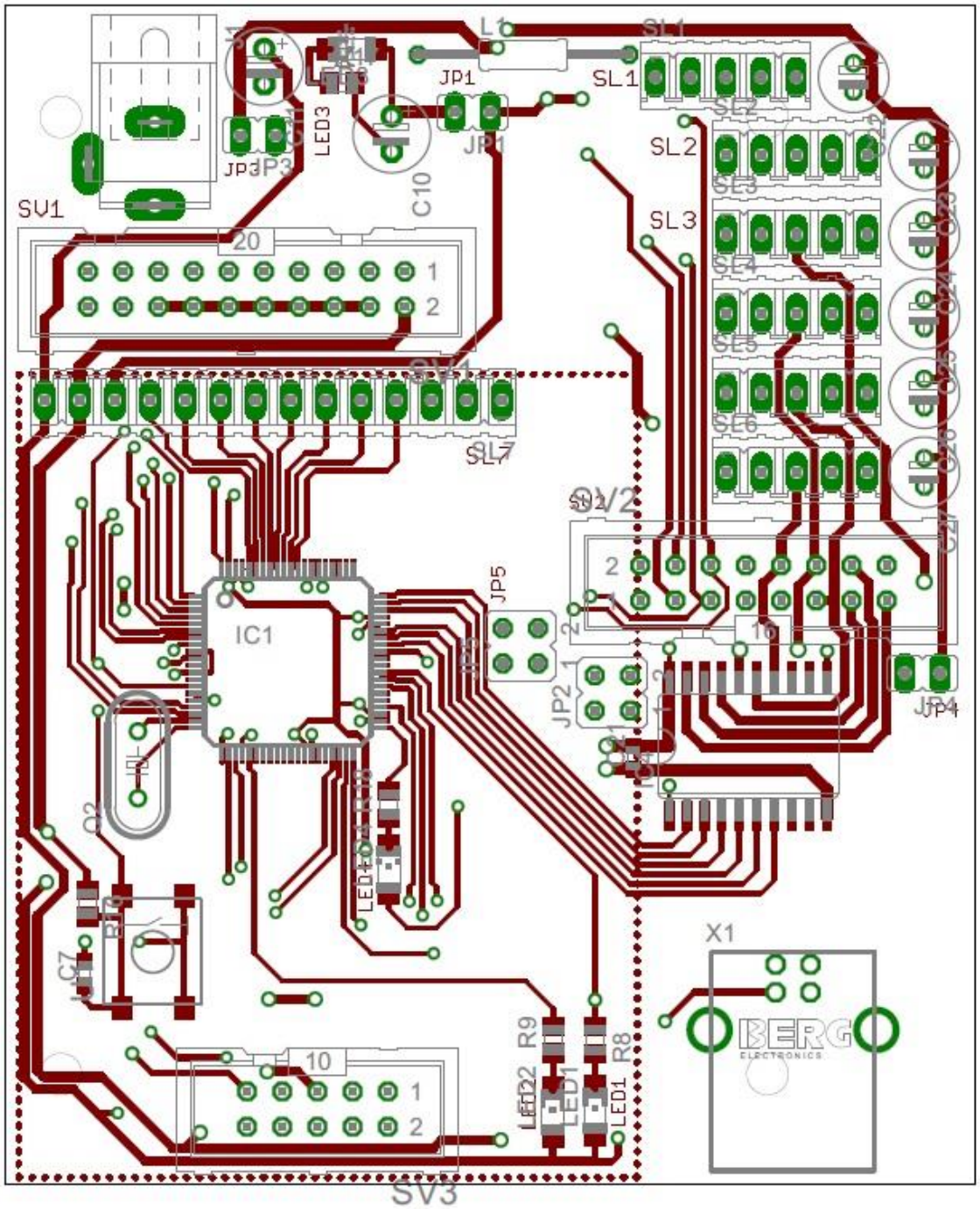
Seznam použitých zkratek

API	Application Programming Interface – Rozhraní pro programování aplikací
CMSIS	Cortex Microcontroller Software Interface Standard – Standardní softwarové rozhraní pro mikrokontroléry ARM Cortex-M
CRC	Cyclic Redundancy Check – Cyklický redundantní součet
DPS	Deska Plošných Spojů
FIFO	First In First Out – První dovnitř, první ven
FPU	Floating Point Unit – Jednotka pro výpočty v plovoucí řádové čárce
GPIO	General Purpose Input/Output – Vstupy/výstupy pro všeobecné použití
GUI	Graphic User Interface – Grafické uživatelské rozhraní
I^2C	Inter-Interred Circuit bus – vnitřní sběrnice mezi integrovanými obvody
ISR	Interrupt Service Routine – Obslužná rutina přerušení
JTAG	Joint Test Action Group – rozhraní pro testování integrovaných obvodů
MHT	Matice homogenní transformace
MKO	Monostabilní klopný obvod
NVIC	Nested Vectored Interrupt Controller – Jednotka vnořeného přerušení
OS	Operating system – Operační systém
PWM	Pulse Width Modulation – Pulsně šířková modulace
RTOS	Real-time Operating System – Operační systém reálného času
SLIP	Serial Line Internet Protocol – Internetový protokol pro sériovou linku
SWD	Serial Wire Debug – sériové ladící rozhraní s minimem potřebných vodičů
TTL	Transistor-Transistor Logic – Tranzistorově-tranzistorová logika
UART	Universal Asynchronous Receiver/Transmitter – Univerzální asynchronní přijímač/vysílač
USB	Universal Serial Bus – Univerzální sériová sběrnic

A SCHÉMA ZAPOJENÍ ŘÍDICÍ DESKY



B PLOŠNÝ SPOJ ŘÍDICÍ DESKY



C UKÁZKA SEZNAMU PŘÍKAZŮ

```
200,0,100,90,0,20,80,0
143,87,34,39,81,37,80,0
143,87,-11,39,81,37,60,0
143,87,-11,39,81,7,100,0
143,87,64,39,81,7,60,0
86,210,190,51,42,7,60,0
86,210,103,51,42,7,50,0
86,210,103,51,42,37,100,0
92,192,256,90,0,37,80,0
50,-150,79,24,84,37,80,0
50,-159,-2,24,84,37,80,0
50,-159,-2,24,84,7,100,0
50,-159,61,24,84,7,60,0
86,-159,238,24,36,7,60,0
86,200,238,24,36,7,80,0
86,200,124,24,69,7,50,0
86,200,124,24,69,37,100,0
86,200,211,24,42,37,40,0
296,-61,154,174,66,37,80,0
302,-61,62,174,66,37,40,0
302,-61,62,174,66,7,100,0
302,-61,139,174,66,7,100,0
101,200,268,174,15,7,70,0
101,200,166,174,66,7,40,0
101,200,166,174,66,37,100,0
245,161,166,123,66,20,80,0
245,161,139,123,66,20,80,0
245,161,121,123,66,20,40,0
116,161,121,123,66,20,40,0
200,0,100,90,0,20,100,0
```